Pembuatan Bot Discord untuk Pengiriman Pesan Rahasia Menggunakan Kriptografi Kunci Simetris dan Tanda Tangan Digital

Frederic Ronaldi - 13519134
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519134@std.stei.itb.ac.id

***Abstract*—Discord, platform komunikasi yang populer di kalangan komunitas daring, tidak secara langsung menyediakan opsi enkripsi *end-to-end* untuk pesan, yang meninggalkan ruang bagi potensi keamanan. Penulis menciptakan *bot* yang dapat mengenkripsi pesan, mengirimnya secara anonim, serta menanda tangani pesan secara digital untuk memverifikasi autentisitas pesan. Dengan bot ini, maka pihak manapun yang tidak memiliki kunci untuk dekripsi tidak dapat membaca pesan yang sebenarnya, ditambah dengan keaslian pengirim yang terjamin dengan adanya tanda tangan digital.**

***Keywords*—Discord, *bot*, *end-to-end encryption*, enkripsi, dekripsi, tanda tangan digital.**

I. PENDAHULUAN

Dalam era digital yang semakin berkembang, komunikasi daring menjadi elemen kunci dalam berbagai aspek kehidupan kita, mulai dari interaksi sosial hingga kerja dan belajar. Discord, sebagai platform komunikasi yang sedang berkembang dan populer, memfasilitasi jutaan pengguna setiap harinya, terutama dalam komunitas permainan video, teknologi, dan pendidikan. Meskipun Discord menawarkan berbagai fitur yang menarik untuk mempermudah komunikasi dan kolaborasi, salah satu aspek penting yang kurang adalah enkripsi *end-to-*end untuk pesan, yang dapat membatasi privasi dan keamanan pengguna.

Kriptografi adalah salah satu alat paling andal untuk melindungi informasi dalam komunikasi digital. Kriptografi menawarkan 4 prinsip utama, yaitu:

1. Kerahasiaan pesan (*Condifentiality*)
2. Keaslian pesan (*Data integrity*)
3. Keaslian pengirim dan penerima (*Authentication*)
4. Anti penyangkalan (*Non-repudiation*)

Melihat kebutuhan tersebut, terdapat berbagai metode kriptografi yang dapat digunakan. Untuk menjaga kerahasiaan pesan, metode kriptografi kunci simetris dapat digunakan untuk mengenkripsi dan mengdekripsi pesan menggunakan kunci yang sama. Kriptografi kunci simetris efisien dan cepat, menjadikannya ideal untuk aplikasi di mana waktu dan sumber daya terbatas, termasuk *bot* Discord.

Untuk memastikan keaslian pesan, otentikasi, serta anti penyangkalan, kita dapat menggunakan tanda tangan digital. Tanda tangan digital berfungsi untuk memverifikasi asal-usul dan integritas data. Melalui tanda tangan digital, pengirim tidak dapat menyangkal bahwa mereka mengirim pesan tersebut, dan penerima dapat yakin bahwa pesan yang mereka terima berasal dari pengirim yang sebenarnya dan tidak diubah selama transmisi.

Oleh karena itu, *bot* Discord ini akan dirancang untuk mengenkripsi dan mendekripsi pesan, memastikan kerahasiaan pesan, serta membuat dan memverifikasi tanda tangan digital, untuk memastikan keaslian pesan dan pengirim. *Bot* Discord ini diharapkan dapat membantu meningkatkan privasi dan keamanan dalam komunikasi daring, khususnya pada platform Discord yang tidak memiliki enkripsi *end-to-end*.

II. TEORI DASAR

A. Kriptografi Kunci Simetris

Kriptografi kunci simetris adalah metode kriptografi yang menggunakan satu kunci untuk proses enkripsi dan dekripsi. Hal ini disebut dengan simetris karena operasi enkripsi dan dekripsi simetris satu sama lain, atau fungsi satu sama lain, menggunakan kunci yang sama. Kriptografi ini salah satu bentuk tertua dan paling cepat dari kriptografi, dan masih digunakan secara luas dalam berbagai aplikasi di mana kecepatan dan efisiensi penting.

Operasi enkripsi adalah proses mengkonversi data asli (*plaintext*) menjadi teks terenkripsi (*ciphertext*) menggunakan sebuah kunci. Algoritma kriptografi kunci simetris, seperti *Data Encryption Standard* (DES) atau *Advanced Encryption Standard* (AES) digunakan untuk ini. Proses enkripsi memastikan kerahasiaan pesan, mencegah siapa pun yang tidak memiliki kunci membaca teks asli. Sedangkan, operasi dekripsi adalah proses mengkonversi teks terenkripsi (*ciphertext*) kembali menjadi data asli (*plaintext*) menggunakan kunci yang sama. Dekripsi memungkinkan penerima pesan yang sah untuk membaca pesan asli.

B. *Advanced Encryption Standard (AES)*

Salah satu algoritma kriptografi kunci simetris adalah *Advanced Encryption Standard* atau yang sering disebut dengan AES. Algoritma ini menggantikan *Data Encryption Standard* (DES) yang sebelumnya populer, menawarkan peningkatan keamanan yang signifikan.

AES bekerja dengan struktur data blok dan kunci. AES mengenkripsi data dalam blok berukuran 128-bit, dan menggunakan kunci enkripsi berukuran 128, 192, atau 256-bit. Proses enkripsi ini melibatkan serangkaian transformasi yang diterapkan berulang-ulang, atau '*rounds*' pada blok data. Jumlah *rounds* yang digunakan berbeda berdasarkan ukuran kunci: 10 *rounds* untuk 128-bit, 12 *rounds* untuk 192-bit, dan 14 *rounds* untuk 256-bit.

Tidak seperti DES yang beroperasi dalam bit, AES bekerja dalam *byte*. Secara garis besar, AES bekerja dengan proses sebagai berikut:

1. *AddRoundKey*: melakukan XOR dengan *state* awal dengan *cipher key*
2. Putaran sebanyak N-1 kali, masing-masing putaran akan dilakukan proses berikut:
    a. *SubBytes*: menggunakan tabel substitusi (S-box) untuk mengubah setiap *byte* dalam blok
    b. *ShiftRows*: pergeseran baris dalam blok
    c. *MixColumns*: penggabungan setiap kolom dalam blok
    d. *AddRoundKey*: kombinasi blok dengan kunci *round*
3. Untuk putaran terakhir, akan dilakukan *SubBytes*, *ShiftRows*, dan *AddRoundKeys*

Setelah semua *rounds* selesai dilakukan, hasilnya adalah blok data yang telah selesai dienkripsi. Untuk mendekripsi blok data, proses ini dijalankan dalam urutan terbalik dengan kunci yang sama. Setiap transformasi memiliki kebalikkannya sendiri untuk proses dekripsi.

AES diklaim resisten terhadap berbagai jenis serangan kriptografis, dan kecepatannya dan efisiensinya membuatnya menjadi pilihan yang baik untuk berbagai aplikasi, mulai dari komunikasi nirkabel bahkan hingga transaksi keuangan.

C. *Tanda Tangan Digital*

Tanda tangan digital merupakan teknik kriptografi yang memberikan prinsip otentisitas, integritas, dan non-repudiasi dokumen atau pesan digital. Teknik ini memastikan bahwa entitas pengirim adalah pemilik sebenarnya dan dokumen atau pesan yang dikirim tidak mengalami modifikasi ataupun gangguan selama proses transmisi, mau itu dari pihak manapun.

Pembuatan tanda tangan digital memanfaatkan pasangan kunci publik dan privat. Kunci privat digunakan untuk proses pembuatan tanda tangan oleh pengirim, sedangkan kunci publik berperan dalam verifikasi tanda tangan yang telah dibuat oleh penerima.

Dalam proses pembuatan tanda tangan digital, dokumen atau pesan yang dikirim akan di-*hash* menggunakan algoritma kriptografi *hashing* seperti SHA-2 atau SHA-3. Hasil *hash* ini lalu akan dienkripsi dengan kunci privat menggunakan algoritma kunci publik seperti RSA, DSA, maupun ECDSA yang memanfaatkan ECC. Hasil enkripsi ini disebut dengan tanda tangan digital. Pesan akan di-*hash* untuk membuat ukuran tanda tangan digital tetap meskipun ukuran dokumen atau pesan digital bervariasi. Penggunaan hash juga akan menjamin integritas data dari dokumen atau pesan digital tersebut.

Dalam proses verifikasi tanda tangan digital, dokumen atau pesan yang telah diterima akan di-*hash* kembali menggunakan algoritma hash yang sama, sehingga didapatkan *hash digest* yang sama seperti hash yang dibuat oleh pengirim. Kemudian, tanda tangan digital akan didekripsi menggunakan kunci publik pengirim yang sebanding dengan kunci privat yang dimiliki oleh pengirim dalam proses pembuatan. Hasil dekripsi seharusnya sama dengan *hash* dokumen atau pesan yang diperiksa. Apabila keduanya sama, maka tanda tangan digital tersebut dianggap sah, dan dokumen atau pesan tersebut disahkan sebagai asli dan tidak tersentuh selama proses pengiriman.

D. *SHA3 (Keccak)*

Secure Hash Algorithm-3 (SHA-3) adalah algoritma fungsi *hash* yang didesain untuk mengkonversi data masukan menjadi nilai *hash* dengan tingkat keamanan yang tinggi. Algoritma ini berasal dari kompetisi *hash* yang diselenggarakan oleh *National Institute of Standards and Technology* (NIST) pada tahun 2007, sebagai respons terhadap keraguan keamanan SHA-1 dan SHA-2. SHA-3 didesain untuk menggantikan SHA-1 dan SHA-2 yang keamanannya sudah mulai diragukan. SHA-3 memakai pendekatan desain baru yang berbeda.

SHA-3 memakai fungsi spons untuk menghasilkan hash digest. Fungsi spons memiliki dua fase, yaitu penyerapan (*absorbing*) dan pemerasan (*squeezing*). Dalam fase penyerapan, fungsi akan secara bertahap menyerap pesan dan menghasilakn state transisi di setiap iterasinya. Lalu, proses berlanjut ke fase pemerasan. Pada fase ini, digest dari pesan yang telah diserap akan dikeluarkan atau diperas. Proses pemerasan ini dulangi hingga ukuran digest yang dihasilkan sesuai dengan yang diharapkan. Karena itu, SHA-3 memiliki flesibilitas yang tinggi dalam menghasilkan *message digest* yang beragam. Keamanan SHA-3 juga didasarkan pada kekuatan dari fungsi *hash* yang tidak dapat diinversi, serta sifat deterministif dari fungsi tersebut, setiap input pesan akan menghasil *digest* yang berbeda. Selain itu, SHA-3 juga memiliki kemampuan untuk menangani serangan *collision*, yaitu ketika dua input pesan yang berbeda menghasilkan output *hash* yang sama.

E. *Digital Signature Algorithm (DSA)*

Digital Signature Algorithm (DSA) adalah algoritma tanda tangan digital yang dapat digunakan untuk memverifikasi otentisitas dokumen digital atau pesan elektronik. Algoritma ini didasarkan pada sulitnya untuk memecahkan permasalahan logaritma diskrit dalam bilangan bulat modulo.

Pasangan kunci kriptografi, yaitu kunci privat dan kunci publik, digunakan dalam DSA. Kunci privat berfungsi dalam pembuatan tanda tangan digital, sedangkan kunci publik digunakan untuk memverifikasi tanda tangan tersebut.

Proses pembuatan tanda tangan digital dalam DSA dimulai dengan mengambil hash dari dokumen atau pesan yang akan ditandatangani. Lalu, proses pembuatan tanda tangan digital akan dilanjutkan. Beberapa proses yang terdapat dalam DSA adalah:

1. Pembangkitan kunci
    a. Pilih bilangan prima p,q sedemikian rupa sehingga q adalah faktor dari (p-1)
    b. Pilih angka acak g yang kurang dari p, dimana g = h^((p-1)/q) mod p, 1 < h < p – 1 dan g > 1
    c. Pilih kunci privat x, yang dalam hal ini x < q
    d. Hitung kunci publik y = g^x mod p
    e. Dihasilkan kunci publik = (p, q, g, y) dan kunci privat = x
2. Pembuatan tanda tangan digital
    a. Dapatkan *message digest* dari algoritma *hash*
    b. Pilih bilangan acak k < q
    c. Tanda tangan merupakan pasangan nilai r dan s dengan:

        r = (g^k mod p) mod q

        s = (k^-1 (digest + x . r)) mod q
3. Verifikasi tanda tangan digital
    a. Dapatkan *message digest* dari algoritma *hash*
    b. Hitung w = s^-1 mod q
    c. Hitung u1 = (digest . w) mod q
    d. Hitung u2 = r . w mod q
    e. Hitung v = ((g^u1 . y^u2) mod p) mod q)
    f. Tanda tangan sah apabila v = r

Melalui metode DSA ini, seseorang dapat memastikan bahwa pesan yang mereka terima tidak berubah selama transmisi dan berasal dari pihak yang sebenarnya, sebab hanya pemegang kunci privat yang dapat menghasilkan tanda tangan valid yang bisa diverifikasi dengan kunci publiknya.

III. IMPLEMENTASI

Pada bagian ini, penulis akan melakukan implementasi untuk *bot* Discord tersebut. Implementasi akan menggunakan bahasa Javascript dengan *library* "discord.js" yang akan mempermudah penulis dalam berinteraksi dengan Discord API untuk membuat *bot*. Kriptografi kunci simetris yang digunakan adalah AES dengan kunci 128-bit, sedangkan tanda tangan digital akan menggunakan fungsi *hash* SHA-3 dan algoritma DSA untuk pembuatan dan verifikasinya.

Pertama, penulis mendaftarkan *bot* tersebut dalam Discord *Developer Portal*, memberikan permission yang dapat dilakukan dalam Discord, menghasilkan token baru untuk otentikasi *bot*, serta menambahkan *bot* dalam sebuah server uji coba. Setelah itu, penulis membuat 1 buah saluran teks khusus yang dapat digunakan untuk menghasilkan pasangan kunci publik dan kunci privat oleh pengguna. *Bot* Discord yang dibuat akan memiliki 3 perintah utama yang didukung oleh fitur "*slash command*" di Discord, yaitu:

1. /*encrypt*: perintah untuk mengenkripsi pesan dengan sebuah kunci, menerima tiga parameter, kunci berukuran 128-bit, pesan yang akan dienkripsi, dan apakah pesan yang dikirim bersifat anonim atau tidak. Apbila pesan anonim, maka pesan tidak akan mengandung nama pengirim, sebaliknya, apabila tidak maka pesan akan mengandung nama pengirim.
2. */generatesignkey*: perintah untuk menghasilkan pasangan kunci publik dan privat, hanya berjalan di satu buah saluran teks khusus. Kunci publik dapat dilihat oleh semua orang di saluran teks khusus tersebut, sedangkan kunci privat hanya dapat dilihat oleh orang yang melakukan perintah tersebut.
3. */encryptandsign*: perintah untuk mengenkripsi pesan dengan sebuah kunci, dan menambahkan tanda tangan digital terhadap pesan tersebut. Perintah ini menerima tiga parameter, kunci yang berukuran 128-bit, pesan yang akan dienkripsi, dan kunci privat yang akan digunakan untuk menandatangani pesan.

Pesan yang dienkripsi akan memiliki tombol "*Decrypt*", serta pesan yang terdapat tanda tangan digital akan memiliki tombol "*Verify*" di bawahnya. Tombol yang diklik akan mengeluarkan modal untuk melakukan operasi tersebut dan menerima input yang sesuai untuk operasinya.

A. Advanced Encryption Standard (AES)

Advanced Encryption Standard akan digunakan untuk mengenkripsi pesan menggunakan kunci 128-bit yang didapatkan dari parameter perintah. Pertama, pesan akan dipecah menjadi blok yang berukuran 128-bit, dan pemberian padding sehingga keseluruhan pesan dapat dipecah menjadi blok-blok 128-bit.

Berikut adalah *pseudocode* yang telah disederhanakan untuk implementasi enkripsi AES:

```
function AES_Encryption(input_plaintext,
key_128bit)

      1. Start

      2. Pastikan plaintext dapat dibagi
menjadi block berukuran 16 bytes

          2.1 Apabila tidak, maka tambahkan
padding

      3. Konversi key 128 bit ke key schedule
(algoritma key expansion)

      4. Plaintext dibagi menjadi block-block
berukuran 16 bytes
```

```
       5. Result = ""
       6. For each block do
           6.1. Konversi block ke 4x4 byte
matrix (disebut State)
           6.2. Round awal:
               6.2.1 AddRoundKey(State,
key_128bit)
           6.3. Fungsi round:
               6.3.1 For round = 1 to 9 do
                   6.3.1.1 SubBytes(State)
                   6.3.1.2 ShiftRows(State)
                   6.3.1.3 MixColumns(State)
                   6.3.1.4 AddRoundKey(State,
round_key) (round_key diturunkan dari key
schedule)
               End for
           6.4. Round terakhir (ke-10):
               6.4.1 SubBytes(State)
               6.4.2 ShiftRows(State)
               6.4.3 AddRoundKey(State,
round_key) (round_key diturunkan dari key
schedule)
           Result += State
       End for
       7. Result adalah ciphertext
    8. End, return Result
```

Berikut adalah *pseudocode* yang telah disederhanakan untuk implementasi dekripsi pada AES:

```
function AES_Decryption(input_ciphertext,
key_128bit)
       1. Start
       2. Pastikan ciphertext dapat dibagi
menjadi block berukuran 16 bytes
       3. Konversi key 128 bit ke key schedule
(algoritma key expansion)
       4. Ciphertext dibagi menjadi block-block
berukuran 16 bytes
       5. Result = ""
       6. For each block do
           6.1. Konversi block ke 4x4 byte
matrix (disebut State)
           6.2. Round awal:
               6.2.1 AddRoundKey(State,
round_key) (round_key adalah kunci terakhir
yang diturunkan dari key schedule)
           6.3. Fungsi round:
               6.3.1 For round = 9 to 1 step -1
do
                   6.3.1.1 InvShiftRows(State)
                   6.3.1.2 InvSubBytes(State)
                   6.3.1.3 AddRoundKey(State,
round_key) (round_key diturunkan dari key
schedule)
                   6.3.1.4 InvMixColumns(State)
               End for
           6.4. Round terakhir (ke-10):
               6.4.1 InvShiftRows(State)
               6.4.2 InvSubBytes(State)
               6.4.3 AddRoundKey(State,
round_key) (round_key adalah kunci pertama yang
diturunkan dari key schedule)
           Result += State
       End for
       7. Hapus padding jika ada pada Result
       8. Result adalah plaintext
        9. End, return Result
```

B. SHA-3 (Keccak)

SHA-3 akan digunakan sebagai algoritma *hash* yang digunakan dalam pembuatan dan memverifikasi tanda tangan digital. SHA-3 akan mengambil input pesan dan mengeluarkan output yang konstan yaitu 512 bit. Algoritma ini dapat mengeluarkan output yang bervariasi, bergantung dengan keinginan pengguna, seperti 256, 384, 1024 bit, atau n-bit.

Pertama, input pesan akan diabsorpsi ke dalam sebuah *state* internal dengan ukuran yang sama dengan output *hash*, kemudian dilakukan permutasi pada *state* tersebut dengan operasi XOR dan rotasi bit. Proses ini dilakukan secara berulang dengan ronde-ronde yang telah ditentukan sebelumnya, dan akhirnya output *hash* dihasilkan dengan mengambil sebagian *state* internal.

Algoritma Keccak, yang menjadi dasar SHA-3, menggunakan fungsi f yang memiliki banyak variasi yang sesuai dengan ukuran permutasinya, yang juga merupaakn ukuran *state* di fungsi spons. *State* diatur dalam array 5x5, sehingga saat diimplementasi pada prosesor 64-bit, maka dipilih ukuran 1600 bit (5x5x64).

Berikut adalah *pseudocode* yang telah disederhanakan untuk implementasi SHA-3:

```
Fungsi SHA3(input_pesan)
       1. Start
       2. Atur spesifikasi:
           2.1 r = 1088
           2.2 c = 512
```

```
        2.3 output_length = 256

        2.4 mbits = [0,1]

    3. Ubah pesan menjadi biner dan simpan
sebagai intermediate

    4. Tambah padding pada intermediate
sampai panjangnya dapat dibagi oleh r (rate)

    5. Inisialisasi array 5x5x64 (w) S
dengan 0

    6. For each r-bit dari pesan yang telah
ditambah padding do

        6.1. For each bit di bagian r-bit do

            6.1.1. XOR bit tersebut dengan
bit yang sesuai di S

        6.2. Terapkan fungsi
keccak_f1600_permutation pada S

    7. Kumpulkan bit output dari S sampai
panjang output yang diinginkan tercapai

    8. Kembalikan bit yang telah dikumpulkan
sebagai hasil hash

    9. End


Fungsi keccak_f1600_permutation(state)

    1. For each round do

        1.1. Hitung parity semua kolom dan
simpan di C

        1.2. Untuk setiap kolom lakukan

            1.2.1. XOR paritas kolom
sebelumnya dan parity kolom berikutnya yang
telah diputar

        1.3. Untuk setiap bit di state
lakukan

            1.3.1. XOR bit tersebut dengan
bit yang sesuai di hasil dari langkah 1.2

        1.4. Putar setiap bit di state ke
lokasi baru

        1.5. Untuk setiap baris di state
lakukan

            1.5.1. Ganti setiap bit dengan
hasil dari (bit_sekarang XOR (NOT
bit_selanjutnya AND bit_setelah_selanjutnya))

        1.6. XOR bit pertama dari state
dengan konstanta ronde

    2. Kembalikan state

    3. End
```

Berikut adalah konstanta ronde yang digunakan:

TABLE I. KONSTANTA RONDE UNTUK SHA-3

RC[0]	0x0000000000000001
RC[1]	0x0000000000008082
RC[2]	0x800000000000808A
RC[3]	0x8000000080008000
RC[4]	0x000000000000808B
RC[5]	0x0000000080000001
RC[6]	0x8000000080008081
RC[7]	0x8000000000008009
RC[8]	0x000000000000008A
RC[9]	0x0000000000000088
RC[10]	0x0000000080008009
RC[11]	0x000000008000000A
RC[12]	0x000000008000808B
RC[13]	0x800000000000008B
RC[14]	0x8000000000008089
RC[15]	0x8000000000008003
RC[16]	0x8000000000008002
RC[17]	0x8000000000000080
RC[18]	0x000000000000800A
RC[19]	0x800000008000000A
RC[20]	0x8000000080008081
RC[21]	0x8000000000008080
RC[22]	0x0000000080000001
RC[23]	0x8000000080008008

C. Digital Signature Algorithm (DSA)

Pada DSA, biasanya digunakan algoritma *hash* SHA-1 untuk *message digest*-nya. Namun, pada implementasi kali ini, digunakan algoritma *hash* SHA-3 (Keccak). Kunci publik dan privat akan diberikan ke pengguna dalam bentuk base64 agar lebih mudah dibaca. Tanda tangan digital juga akan ditampilkan kepada penerima dalam bentuk base64.

Berikut adalah *pseudocode* yang telah disederhanakan untuk pembangkitan kunci publik dan privat:

```
Fungsi DSA_KeyGeneration

    1. Start

    2. Pilih ukuran bit keamanan L dan N
(contoh: L = 2048 dan N = 224)

    3. Cari bilangan prima acak q dengan
panjang 2048 bit

    4. Cari bilangan prima acak p dengan
panjang 224 bit, sedemikian rupa sehingga p-1
dapat dibagi oleh q

    5. Pilih bilangan acak g dengan rentang
```

```
1 < g < p-1, kemudian hitung h = g^((p-1)/q) mod p
          - Jika h = 1, pilih g yang lain dan ulangi langkah ini
       6. Pilih bilangan acak x dengan rentang 1 < x < q (untuk kunci privat)
       7. Hitung y = g^x mod p
       8. Kunci publik adalah (p, q, g, y)
       9. Kunci privat adalah x
       10. End, return kunci publik dan privat
```

Berikut adalah *pseudocode* yang telah disederhanakan untuk pembuatan dan verifikasi tanda tangan digital:

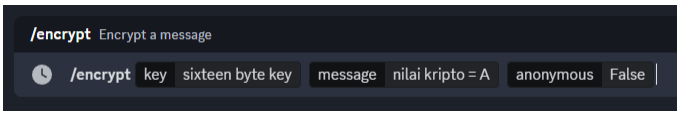
```
Fungsi DSA_SignatureGeneration(message, kunci_privat)
       1. Start
       2. Hitung H(message), dimana H() adalah fungsi hash SHA-3
       3. Pilih bilangan acak k dengan rentang 0 < k < q
       4. Hitung r = (g^k mod p) mod q
           - Jika r = 0, pilih k yang lain dan ulangi langkah ini
       5. Hitung s = (k^(-1) * (H(message) + x*r)) mod q, dimana k^(-1) adalah invers modular k modulo q
           - Jika s = 0, pilih k yang lain dan ulangi langkah ini
       6. Tanda tangan digital adalah (r, s)
       7. End, return tanda tangan digital dalam bentuk base64


Fungsi DSA_SignatureVerification(message, tanda_tangan, kunci_publik)
       1. Start
       2. Ekstrak (r, s) dari tanda tangan (decode base64)
       3. Hitung H(message), dimana H() adalah fungsi hash SHA-3
       4. Periksa apakah 0 < r < q dan 0 < s < q
           - Jika salah satu pemeriksaan gagal, tanda tangan tidak valid
       5. Hitung w = s^(-1) mod q
       6. Hitung u1 = (H(message) * w) mod q
       7. Hitung u2 = (r * w) mod q
       8. Hitung v = ((g^u1 * y^u2) mod p) mod q
       9. Jika v = r,
           9.1. tanda tangan valid
           9.2. Jika tidak, tanda tangan tidak valid
       10. End, kembalikan status validasi tanda tangan
```

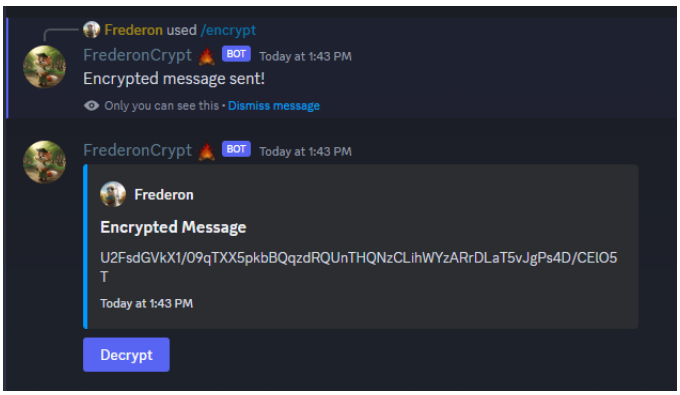
IV. PENGUJIAN

Setelah implementasi selesai, *bot* ditambahkan ke sebuah server Discord khusus untuk pengujian. Saluran teks khusus dibuat untuk menampung pasangan kunci publik dan privat, dalam kasus ini adalah *#key*.

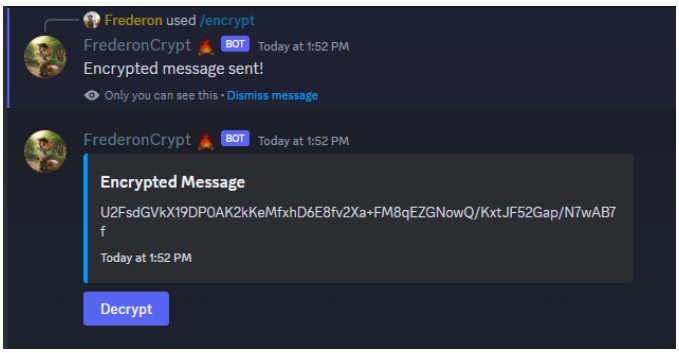
Pertama, akan diuji operasi "*/encrypt*" dengan parameter sebagai berikut:



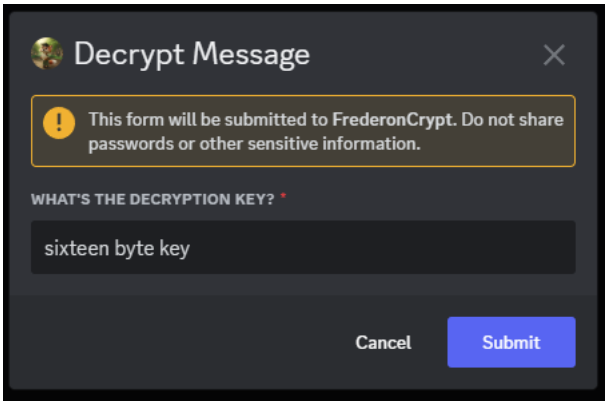
Kunci yang digunakan adalah "*sixteen byte key*", pesan yang dienkripsi adalah "nilai kripto = A", dengan mode tidak anonim. Ketika dijalankan, maka akan menghasilkan pesan seperti berikut:



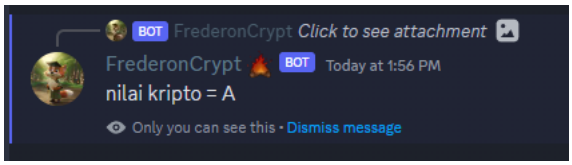
Apabila perintah yang sama dijalankan ulang dengan mode anonim, maka didapatkan hasil sebagai berikut:



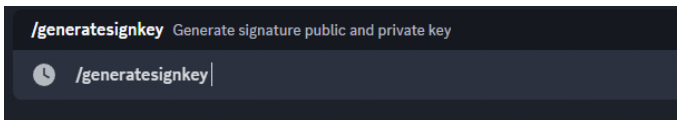
Disini terlihat bahwa nama pengirim tidak ditampilkan dalam pesan, sehingga kerahasiaan pengirim dapat dijaga. Setelah itu, apabila tombol "*Decrypt"* ditekan, maka akan muncul modal yang akan meminta input kunci dari pengguna, seperti berikut ini:

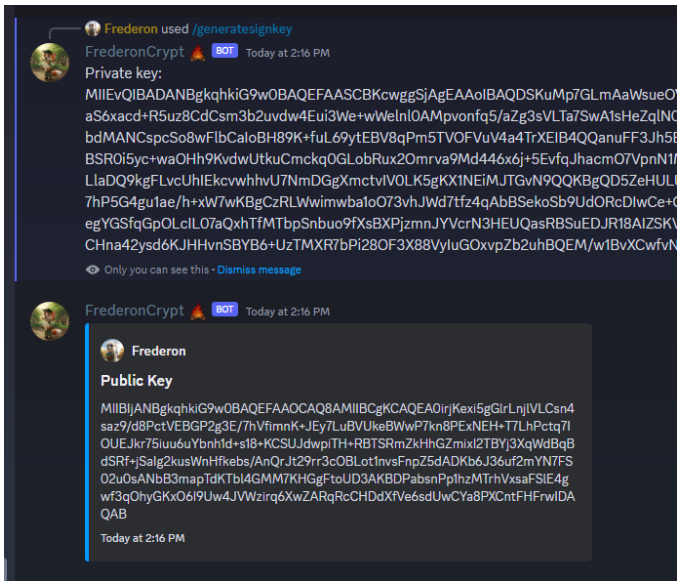


Ketika kunci dekripsi dikirimkan, maka pesan asli akan ditampilkan untuk pengguna tersebut saja, sehingga pesan asli tidak akan terlihat bagi pengguna lain yang tidak mengirimkan kunci dekripsi.

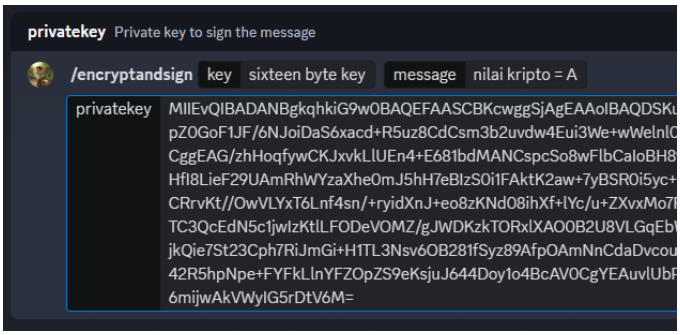


Untuk menguji tanda tangan digital, dibutuhkan pasangan kunci publik dan privat. Untuk mendapatkan pasangan kunci tersebut, kita dapat menggunakan perintah "*/generatesignkey*" tanpa parameter apapun. Perintah ini hanya berjalan di saluran teks khusus yang telah dibuat sebelumnya, yaitu *#key*.

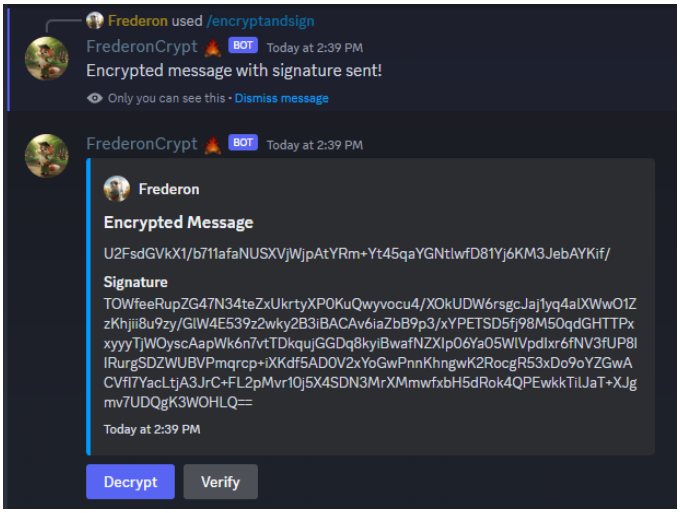




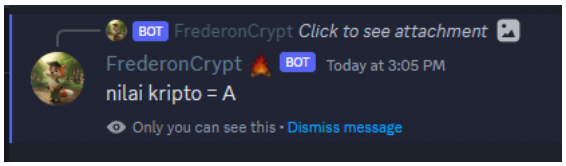
Kunci publik dapat dilihat semua orang di saluran teks khusus, namun kunci privat hanya dapat dilihat oleh orang yang menjalankan perintah tersebut. Setelah didapatkan pasangan kunci publik dan privat, maka kita dapat menggunakan perintah "*/encryptandsign*" untuk mengenkripsi dan menandatangani pesan. Parameter yang digunakan untuk perintah ini adalah sebagai berikut:



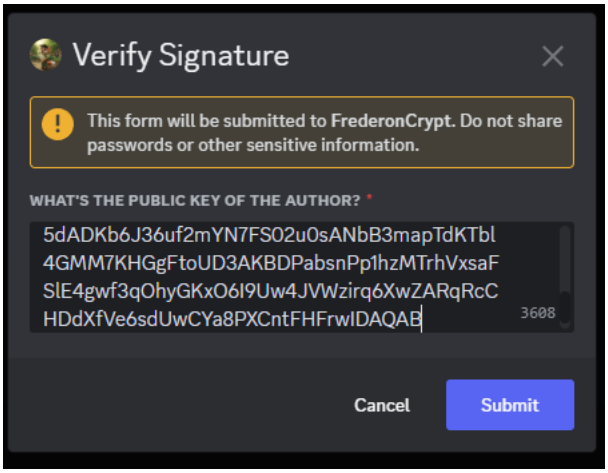
Kunci yang digunakan adalah *"sixteen byte key*", pesan yang dienkripsi adalah "nilai kripto = A", dan dengan kunci publik sebelumnya. Ketika dijalankan, maka akan menghasilkan pesan dan tanda tangan seperti berikut:



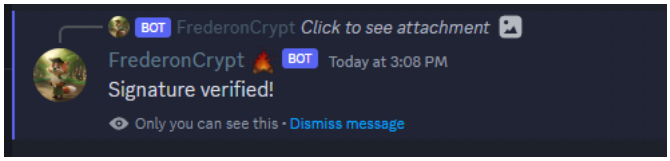
Terlihat bahwa pesan yang telah dienkripsi ditampilkan, beserta tanda tangan digitalnya. Perintah ini tidak memiliki mode anonim karena penerima harus mengetahui siapa pengirimnya dan melihat kunci publiknya untuk memverifikasi tanda tangan digital. Apabila tombol "*Decrypt"* ditekan, maka modal yang sama yang meminta kunci dekripsi akan ditampilkan.



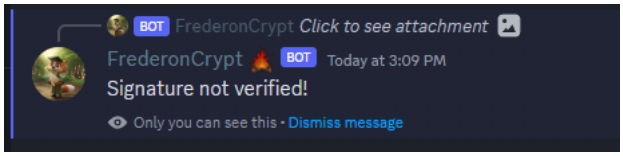
Apabila tombol "*Verify"* ditekan, maka akan modal untuk verifikasi ditampilkan, seperti berikut ini:



Ketika kunci publik berhasil dikirimkan, maka akan ditampilkan apabila pesan yang dikirim merupakan pesan asli atau tidak.



Apabila menggunakan kunci publik yang berbeda, maka akan ditampilkan bahwa pesan yang dikirim tidak terverifikasi.



V. Kesimpulan Dan Saran

Solusi yang diimplementasi berhasil mengintegrasikan teknik kriptografi kunci simetris yaitu *Advanced Encryption Standard* (AES), algoritma *hash* SHA-3 (Keccak), dan *Digital Signature Algorithm* (DSA). Solusi yang dikembangkan dapat meningkatkan keamanan, privasi, dan integritas pesan yang dikirimkan melalui platform daring Discord. Pengguna dapat mengirimkan pesan yang terenkripsi menggunakan kunci simetris dengan ukuran 128-bit, membuat pasangan kunci publik dan privat, membuat tanda tangan digital, serta melakukan verifikasi tanda tangan digital. Kriptografis kunci simetris melalui AES menjamin kerahasiaan pesan, sementara tanda tangan digital melalui DSA dan SHA-3 untuk memastikan keaslian dan integritas pesan.

Kedepannya, solusi dapat dikembangkan lebih lanjut, misalnya menggunakan kriptografi kunci publik untuk enkripsi pesan. Selain itu, integrasi dengan berbagai fitur Discord lainnya juga dapat dilakukan, misalnya mengenkripsi file atau gambar. Ditambah, pencarian kunci publik masih harus dilakukan secara manual dalam saluran teks khusus, hal ini masih dapat dikembangkan untuk pencarian otomatis.

Ucapan Terima Kasih

Puji syukur saya panjatkan kepada Tuhan Yang Maha Esa karena berkat dan rahmat-nya saya dapat menyelesaikan makalah ini dengan baik dan benar. Saya juga mengucapkan terima kasih kepada Bapak Rinaldi Munir selaku pengajar mata kuliah Kriptografi dan pembimbing dalam pembuatan makalah ini.

Referensi

[1] Munir, Rinaldi. 2023. Slide Kuliah IF4020 Kriptografi: *Advanced Encryption Standard*

[2] Munir, Rinaldi. 2023. Slide Kuliah IF4020 Kriptografi: Kriptografi Kunci Publik

[3] Munir, Rinaldi. 2023. Slide Kuliah IF4020 Kriptografi: Fungsi *Hash*

[4] Munir, Rinaldi. 2023. Slide Kuliah IF4020 Kriptografi: *Digital Signature Standard*

[5] Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Assche, G. V., Keer, R. V. 2016. Keccak Specifications Summary. Diakses pada 19 Mei 2023.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang sayqa tulis ini adalah tulisan saya sendiri, bukan saduran, atua terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Mei 2023

Frederic Ronaldi

13519134