Implementasi Algoritma RSA dalam Mengenkripsi Data pada *Video-based Storage*

Gagas Praharsa Bahar - 13520016
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520016@std.stei.itb.ac.id

***Abstract*—Kriptografi kunci publik adalah salah satu bentuk kriptografi yang umum digunakan untuk mengenkripsi data. Data dapat ditulis dalam berbagai macam format, bahkan dapat dibentuk dalam gambar. Video adalah salah satu format yang dapat digunakan untuk menulis data, walaupun tidak banyak digunakan. Hal ini memungkinkan pengguna untuk mengirimkan pesan ataupun data dalam format video yang dapat diputar, yang secara kasat mata tidak akan dicurigai oleh orang lain. Kriptografi kunci publik akan membantu mengamankan proses ini untuk menjamin yang dapat membuka data tersebut hanyalah yang memiliki kunci.**

Keywords—cryptography; storage; video; public key cryptography; RSA;

I. PENDAHULUAN

Dewasa ini, perkembangan teknologi membawa banyak perubahan dalam kehidupan manusia. Kehadiran teknologi baru juga mengakibatkan adanya perubahan dalam cara hidup manusia secara fundamental. Perkembangan teknologi informasi dan internet telah menghasilkan pertumbuhan atas kebutuhan data yang sangat pesat, sehingga dibutuhkan kanal penyimpanan data yang lebih besar. Selain itu, perkembangan teknologi informasi dan internet juga menghasilkan banyak hal lainnya, seperti situs-situs berbagi video yang semakin meluas di dunia maya, seperti YouTube. Konsep yang menarik adalah kemampuan situs-situs berbagi video ini untuk menyimpan data dengan menggunakan teknik pengkodean dalam video.

Namun, perlu diingat bahwa keamanan data menjadi hal yang krusial dalam konteks ini. Untuk menjaga kerahasiaan dan integritas data yang disimpan, diperlukan penerapan kriptografi kunci publik. Teknik ini memanfaatkan pasangan kunci, yaitu kunci publik dan kunci privat, untuk mengamankan data yang dikirim melalui jaringan.

Dengan menggunakan kriptografi kunci publik, informasi sensitif yang dikodekan dalam video dapat dijaga kerahasiaannya. Kunci publik digunakan untuk mengenkripsi data sebelum disimpan dalam video, sementara kunci privat yang hanya diketahui oleh pemiliknya digunakan untuk mendekripsi data tersebut. Dengan demikian, meskipun data tersimpan secara terbuka di situs berbagi video, hanya pemilik kunci privat yang memiliki akses penuh terhadap data yang tersembunyi di dalamnya.

Penerapan kriptografi kunci publik dalam penyimpanan data menggunakan situs berbagi video merupakan langkah penting untuk memastikan keamanan dan privasi informasi. Dengan teknik ini, pengguna dapat merasa lebih tenang saat menyimpan data sensitif, karena hanya pihak yang memiliki akses ke kunci privat yang dapat mengakses dan membaca data yang tersembunyi di dalam video.

II. TEORI DASAR

A. *Penyimpanan Data (Data Storage)*

Penyimpanan data atau *data storage* adalah perekaman data dalam sebuah *storage medium*. Bentuk medium ini bermacam-macam, mulai dari tulisan tangan, *magnetic tape, optical discs, solid state drives*, dan masih banyak lagi yang lainnya. Molekul biologis seperti RNA dan DNA juga dianggap oleh beberapa pihak sebagai sebuah *data storage*.

Apabila data disimpan secara digital dengan medium yang dapat dibaca oleh mesin, data tersebut dapat dikatakan sebagai data digital. Penyimpanan data dalam komputer adalah salah satu fungsi inti dari sebuah komputer modern. Penyimpanan data secara elektronik juga biasanya jauh lebih irit tempat dibandingkan menyimpan data dalam bentuk fisik (seperti dalam kertas).

Secara global, pertumbuhan data yang dibentuk oleh manusia sangatlah pesat seiring dengan berkembangnya teknologi informasi. Pada sebuah studi yang diadakan oleh International Data Corporation pada 2007, diestimasikan terdapat 281 exabytes (1 exabyte = 1.000.000 terabytes) digital data yang ada di dunia.

B. *Format File*

Format konten adalah sebuah format yang telah terkodekan untuk mengkonversi sebuah tipe data untuk menjadi informasi yang dapat dilihat atau diproses.

Salah satu bentuk format konten adalah *file format*, sebuah cara terstandarisasi untuk mengkodekan sebuah informasi dalam bentuk *file* komputer. Dengan sebuah *file format*, bit-bit data yang ada pada sistem dapat merepresentasikan sesuatu yang lainnya.

File format dapat didesain untuk penggunaan spesifik (seperti *file* PNG yang menyimpan gambar bitmap dengan

lossless data compression), tetapi ada juga yang didesain untuk pengaplikasian yang lebih umum, seperti file text yang dapat menyimpan rangkaian karakter dalam bentuk apapun (termasuk *control characters*) dan dengan teknik *encode* apapun. Atau seperti *source code* program komputer yang format filenya hanyalah sebagai penanda sintaks bahasa pemrograman yang dipakainya.

C. *Penyimpanan Data berbasis Video (Video-based Data Storage)*

Data dapat dikodekan dalam banyak bentuk, namun salah satu konsep menarik (walaupun tidak praktis) adalah dengan mengkodekan data dalam bentuk rangkaian gambar yang nantinya akan disambungkan menjadi sebuah video. Hal ini didasari dengan banyaknya situs berbagi video yang menggratiskan penyimpanan video, sehingga secara teoretis, dapat menyimpan sebanyak mungkin data (namun tentu dengan peraturan yang berlaku dari situs penyimpanan video yang terkait). Penggunaan dalam bentuk ini tidak direkomendasikan dan hanya akan digunakan sebagai bahan pembelajaran saja. Selain itu, hal ini juga dapat digunakan secara teoretis untuk mengembalikan sebuah pesan rahasia, karena bentuk gambar yang dikeluarkan akan mirip seperti *static* yang biasa terdapat pada siaran televisi zaman dahulu.

Dalam melakukan penyimpanan data berbasis video dan membagikannya secara daring, terdapat sebuah tantangan menarik yang harus diselesaikan, yaitu algoritma kompresi situs berbagi video yang seringkali cukup tinggi tingkat kompresinya, sehingga sulit untuk melakukan *decode* terhadap video yang diunduh nantinya.

Terdapat dua mode algoritma *encode* yang digunakan, yaitu sebagai berikut:

- Mode RGB

  Dalam mode RGB, tiap *byte* di-*encode* sebagai warna pada RGB pixel. Dalam satu buah pixel, terdapat tiga warna (red, green, blue), sehingga setiap pixel dapat menyimpan 3 byte data.

- Mode Binary

  Dalam mode binary, tiap pixel di-*encode* sebagai warna hitam atau putih, dengan hitam melambangkan 0 dan putih melambangkan 1. Mode ini kurang efisien dibandingkan mode RGB, namun lebih terjamin karena akan lebih tahan terhadap algoritma kompresi yang digunakan.

Dalam kedua mode, dikarenakan apabila hanya menggunakan pixel individual akan berkemungkinan besar dapat dikompres oleh algoritma kompresi, maka yang digunakan adalah blok pixel 2x2 untuk melambangkan satu *byte*.

Setelah melakukan *encoding* terhadap data sebagai kumpulan pixel, pixel-pixel tersebut kemudian dibariskan sebagai kumpulan gambar-gambar yang nantinya akan membuat sebuah video. Setting yang digunakan dalam proses *encoding* juga akan disematkan pada *frame* pertama video agar program dapat dengan mudah mengetahui mode encoding yang digunakan oleh program sebelumnya.

D. *Kriptografi Kunci Publik (Public Key Cryptography)*

Kriptografi kunci publik atau kriptografi asimetri adalah sebuah jenis kriptografi yang menggunakan sepasang kunci, yaitu kunci publik yang bisa disebarkan serta diakses oleh semua pihak dan juga kunci privat yang harus dijaga kerahasiaannya dan hanya diketahui oleh pemiliknya. Pembuatan kunci ini didasarkan pada algoritma atau protokol kriptografi yang digunakan, dan umumnya didasari pada sifat matematis atau masalah matematis yang sulit untuk dipecahkan. Keamanan dari kriptografi jenis ini pada dasarnya tergantung dengan penyimpanan kunci pribadi pada tempat masing-masing.

Dalam sistem kriptografi kunci publik, setiap orang dapat mengenkripsi pesan dengan kunci publik (yang biasanya tersedia secara publik) namun yang dapat mendekripsi hanyalah pemegang kunci privat. Contoh pengaplikasiannya adalah seperti pengiriman pesan kepada server dengan mengenkripsinya dengan kunci publiknya. Pesan tersebut dapat berisi kunci baru yang akan dipakai untuk kriptografi simetris, sehingga kemudian dapat berkirim pesan satu sama lain dengan kunci simetris yang telah didapatkan.

Selain untuk enkripsi, kriptografi kunci publik juga dapat digunakan untuk autentikasi, seperti dengan pembubuhan tanda tangan digital pada akhir pesan. Semua orang dapat memastikan keaslian pesan dari pengirim dengan menggunakan kunci publik, namun yang dapat memberikan tanda tangan hanyalah pemegang kunci privat.

Berikut adalah ilustrasi cara kerja kriptografi kunci publik:

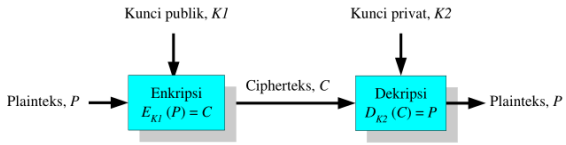


Fig. 1. Ilustrasi kriptografi kunci publik

Kelebihan utama kriptografi kunci publik adalah kemampuannya untuk mendukung keamanan dan kepercayaan dalam lingkungan yang tidak aman atau terbuka, seperti internet. Kriptografi kunci publik juga dapat didukung dengan protokol seperti pertukaran kunci Diffie-Hellman untuk melakukan generasi atas kunci yang digunakan.

Saat diaplikasikan dalam konteks penyimpanan data menggunakan situs berbagi video seperti bahasan saat ini, kriptografi kunci publik memainkan peran penting dalam menjaga kerahasiaan informasi. Data sensitif yang diencode dalam video dapat dienkripsi menggunakan kunci publik sebelum diunggah ke situs berbagi video. Sebagai hasilnya, hanya pemilik kunci privat yang dapat mendekripsi dan mengakses data yang tersembunyi di dalam video tersebut. Dengan kata lain, meskipun data tersimpan secara terbuka di situs berbagi video, hanya pihak yang memiliki kunci privat yang dapat membaca dan memahami isi data tersebut.

E. *Algoritma Kunci Publik RSA*

RSA (Rivest-Shamir-Adleman) merupakan kriptosistem kunci-publik yang cukup banyak digunakan dan banyak

aplikasinya dalam enkripsi data sehari-hari. Nama algoritma RSA diambil dari nama pembuatnya, yaitu Ronald Rivest, Adi Shamir, dan Len Adleman pada tahun 1976. Keamanan algoritma RSA terletak pada sulitnya memfaktorkan bilangan bulat yang besar menjadi faktor-faktor primanya (yang besar juga bilangannya).

Berikut ini adalah beberapa properti penting di dalam algoritma RSA:

1. p dan q bilangan prima (rahasia)
2. $n = p \cdot q$ (tidak rahasia)
3. $\Phi(n) = (p - 1)(q - 1)$ (rahasia)
4. e (kunci enkripsi) (tidak rahasia)

Syarat: PBB(e, Φ(n)) = 1, PBB = pembagi bersama terbesar = gcd (greatest common divisor)

5. d (kunci dekripsi) (rahasia)

d dihitung dari $d \equiv e^{-1} \pmod{\Phi(n)}$

6. m (plainteks) (rahasia)
7. c (cipherteks) (tidak rahasia)

Dari properti-properti diatas, terdapat beberapa properti yang berlaku sebagai kunci:

- Kunci publik = (e, n)
- Kunci privat = (d, n)

Berikut ini adalah langkah-langkah untuk melakukan enkripsi pesan dengan menggunakan algoritma RSA:

1. Nyatakan pesan menjadi blok-blok plainteks: m1, m2, m3, m4, … (syarat: $0 \leq m_i < n - 1$).
2. Untuk setiap blok plainteks, hitunglah blok cipherteks ci untuk blok plainteks mi menggunakan kunci publik e dengan persamaan berikut ini.

$$c_i = m_i^e \bmod n$$

3. Gabungkan blok-blok cipherteks tersebut secara terurut sehingga menjadi satu cipherteks utuh.

Berikut ini adalah langkah-langkah untuk melakukan dekripsi pesan dengan menggunakan algoritma RSA:

1. Nyatakan cipherteks ke dalam blok-blok cipherteks: c1, c2, c3, c4, … (syarat: $0 \leq c_i < n - 1$).
2. Untuk setiap blok cipherteks, hitunglah blok plainteks mi untuk blok cipherteks ci menggunakan kunci privat d dengan persamaan berikut ini.

$$m_i = c_i^d \bmod n$$

3. Gabungkan setiap blok plainteks tersebut secara terurut sehingga menjadi satu plainteks utuh.

III. Penerapan Algoritma

Dalam penerapan algoritma pengkodean data pada video serta pengamanan dengan algoritma RSA, akan digunakan repositori *open source* sebagai dasar dari penerapan algoritma, yakni https://github.com/DvorakDwarf/Infinite-Storage-Glitch . Kode dimodifikasi oleh penulis untuk mengimplementasikan kepentingan pengamanan data serta sedikit perubahan untuk kompatibilitas.

A. *Rancangan Solusi*

Program implementasi dari algoritma ini akan dibuat dalam bahasa pemrograman Rust, seperti repositori dasarnya. Program akan dibuat dengan CLI (Command-Line Interface). Program ditulis dengan spesifikasi kebutuhan fungsional seperti berikut:

- Program dapat menerima masukan berupa *file* komputer.
- Program dapat mengkodekan *file* dalam bentuk video output.
- Program dapat mengembalikan *file* tersembunyi yang tersimpan pada video hasil enkripsi program.
- Program mengimplementasikan algoritma RSA pada tahap *encoding.*
- Program mengimplementasikan algoritma RSA pada tahap *decoding*.

Adapun *flow* program adalah seperti berikut:

- *Build executable* program terlebih dahulu.
- Jalankan *executable* program hasil kompilasi.
- Program akan berjalan dan menunjukkan tiga pilihan menu:
    - Embed untuk *encode* data menjadi video
    - Download untuk mengunduh video dari YouTube
    - Dislodge untuk mengembalikan file yang terdapat pada video
- Bergantung pada pilihan yang diambil, program akan meminta masukan yang sesuai.
    - Apabila melakukan embed, maka program akan meminta masukan *file path* yang ingin di encode
    - Apabila melakukan download, maka program akan meminta tautan YouTube
    - Apabila melakukan dislodge, maka program akan meminta *file path* video yang ingin di-*decode*.
- Program akan mengembalikan keluaran sesuai pilihan yang diambil.

B. Implementasi Algoritma Encoding to Video

Bagian utama kode yang berisi algoritma pengkodean menuju video terdapat pada cuplikan kode dibawah ini (dalam mode enkoding binary):

```
OutputMode::Binary => {
        let length = data.binary.len();
        let frame_size = (settings.width * settings.height) as usize;
        let frame_data_size = frame_size / settings.size.pow(2) as usize;
        let frame_length = length / frame_data_size;
        let chunk_frame_size = (frame_length / settings.threads) + 1;
        let chunk_data_size = chunk_frame_size * frame_data_size;
        let chunks = data.binary.chunks(chunk_data_size);
        for chunk in chunks {
          let chunk_copy = chunk.to_vec();
          let thread =thread::spawn(move || {
            let mut frames = Vec::new();
            let mut index: usize = 0;
            loop {
              let mut source =
                  EmbedSource::new(settings.size, settings.width, settings.height);
              match etch_bw(&mut source, &chunk_copy,&mut index) {
                Ok(_) => frames.push(source),
                Err(_v) => {
                  frames.push(source);
                  println!("Embedding thread complete!");
                  break;
                }
              }
            }
            return frames;
          });
          spool.push(thread);
        }
    }
```

Cuplikan kode diatas berisi algoritma pengkodean utama. Data yang dimasukkan dalam bentuk *bytes* terlebih dahulu akan dibagi ke dalam beberapa *chunk* untuk dijalankan secara paralel dengan *multithreading*. Setiap *chunk* tersebut lalu akan memanggil fungsi etch_bw untuk setiap *frame* yang dibutuhkan. Fungsi etch_bw akan mengambil data satu persatu dan mengkonversinya menjadi sebuah gambar dengan pixel hitam atau putih dalam mode *binary*. Implementasi fungsi etch_bw dapat dilihat dibawah ini:

```
   fn etch_bw(
     source: &mut EmbedSource,
     data: &Vec<bool>,
     global_index: &mut usize,
   ) -> anyhow::Result<()> {
     let _timer = Timer::new("Etching frame");

     let width = source.actual_size.width;
     let height = source.actual_size.height;
     let size = source.size as usize;

     for y in (0..height).step_by(size) {
       for x in (0..width).step_by(size) {
         let local_index = global_index.clone();

         let brightness = if data[local_index] == true {
           255 // 1
         } else {
           0 // 0
         };
         let rgb = vec![brightness, brightness,brightness];
         etch_pixel(source, rgb, x, y).unwrap();

         *global_index += 1;
         if *global_index >= data.len() {
           return Err(Error::msg("Index beyond data"));
         }
       }
     }
     return Ok(());
}
```

C. Implementasi Algoritma RSA

Dalam mengimplementasikan algoritma RSA, penulis menggunakan kakas yang sudah dibuat sebelumnya (*library* RSA pada Rust). Namun, diperlukan beberapa *preprocessing* terhadap data agar dapat dienkripsi dengan baik. Data yang telah dibaca dari sebuah *file* lalu dipecah-pecah menjadi block berukuran 245 bit untuk diproses pada algoritma RSA.

Beberapa konfigurasi yang dipakai pada penerapan RSA kali ini adalah:

- Bit size = 2048
- Chunk size = 256
- Padding: PKCS#1 v1.5
- Total size per block = 245 (256 dikurang 11 byte padding)

Saat melakukan enkripsi untuk pertama kali, program akan menyimpan private key dalam format .pem dalam komputer. Private key ini tidak boleh disebarkan, dan hanya akan digunakan pada tahap dekripsi.

Pada tahap dekripsi, sama seperti tahap enkripsi, data akan dipecah menjadi blok-blok yang berukuran 256 bit. Program kemudian akan mendekripsi setiap blok satu per satu dengan *private key* yang telah di generate pada tahap enkripsi.

D. Hasil Pengujian Enkripsi-Dekripsi

Pengujian dilakukan dengan cara melakukan *embed* terhadap *text file* test.txt yang berukuran 2.400 KB.

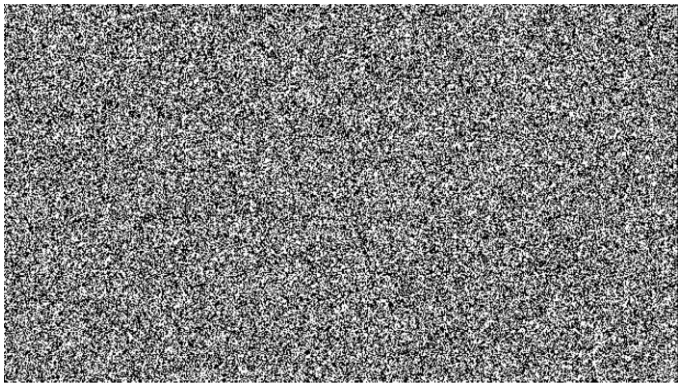
- Proses enkripsi

Proses enkripsi dijalankan terhadap file uji. Setelah menunggu beberapa saat, berikut luaran yang dihasilkan pada terminal:

```
Video embedded successfully at output.avi
Etching video ended in 6797ms
```

Fig. 2. Luaran enkripsi pada terminal

Proses enkripsi juga mengembalikan sebuah video yang telah ter-*embed* data dalam format .avi. Berikut adalah tangkapan layar sebagai contoh:



Selain itu, proses enkripsi juga menghasilkan sebuah luaran private key yang akan tersimpan pada private_key.pem. Berikut adalah cuplikan *private key* (dipotong karena terlalu panjang)

-----BEGIN RSA PRIVATE KEY-----

MIIEpQIBAAKCAQEAxKLt6xGsVqUUBi9FdY/e9P3z7Q6hzj6GJlfeGPPckdXtcgrl

/P4lBkFbGzbqeLiQxGLJ/pKT8IwOg9y51nIqivYDtbOPvYqEgRSY43RtA6NSQABl

+yYJnVwQ+Xal+SMY0j+Veorssp/cNBuNqrR9IeU0rgT4gqckCmhFxGKopxGuwEt

h

EgbAQ7sWW8CJYbqqoYAjSPqRYnOyjpp73Mg0vbj79ZehCMKW/Y9czvSzpY3w

qOeG

zykKx5cRGRhbO/gqWBG8sqvlcsf3XNQTnfZTh8YbMfRyWCvgPO5Q250F4QEEJ

IJ2

3rO1XfavOy3ALoenCtpIM24k+fmp9X7uZARVlwIDAQABAoIBAQC6TZhXqNX

8/cjc

FzubKCXZPyc2Si+55flaKNQwFhS8Q9EFxRekvPI9W/zC6yP6HvxiZSr2vM/YEArM

- Proses dekripsi

Proses dekripsi dijalankan terhadap *file* output.avi yang menyimpan data hasil enkripsi. Setelah menunggu beberapa saat, berikut luaran yang dihasilkan pada terminal:

```
On frame: 20
On frame: 40
On frame: 60
On frame: 80
Video read successfully
Start decrypting data with private_key.pem
decrypted data: 2457252
Success decrypting data
Dislodging frame ended in 22357ms
File written successfully
```

Fig. 3. Luaran dekripsi pada terminal

Setelah selesai, program akan menghasilkan sebuah luaran berbentuk *text file* yang memiliki konten yang sama dengan masukan, pertanda bahwa proses dekripsi berhasil dan data tidak *corrupt*.

```
According to all known laws of aviation, there is no way a bee
should be able to fly. Its wings are too small to get its fat
little body off the ground. The bee, of course, flies anyway
because bees don't care what humans think is impossible. Yellow,
black. Yellow, black. Yellow, black. Yellow, black. Ooh, black and
yellow! Let's shake it up a little. Barry! Breakfast is ready!
Coming! Hang on a second. Hello? - Barry? - Adam? - Can you
believe this is happening? - I can't. I'll pick you up. Looking
sharp. Use the stairs. Your father paid good money for those.
Sorry. I'm excited. Here's the graduate. We're very proud of you,
son. A perfect report card, all B's. Very proud. Ma! I got a thing
going here. - You got lint on your fuzz. - Ow! That's me! - Wave
to us! We'll be in row 118,000. - Bye! Barry, I told you, stop
flying in the house! - Hey, Adam. - Hey, Barry. - Is that fuzz
```

Fig. 4. Cuplikan luaran dekripsi

- Proses dekripsi dengan private key yang salah

Proses dekripsi tidak akan dapat berjalan tanpa *private key*, karena hal tersebut dibutuhkan untuk melakukan dekripsi terhadap data. Berikut adalah yang akan terjadi apabila memakai *private key* yang tidak valid:

```
On frame: 20
On frame: 40
On frame: 60
On frame: 80
Video read successfully
Start decrypting data with private_key.pem
thread 'main' panicked at 'failed to parse priva
te key: Asn1(Error { kind: TagUnknown { byte: 0
}, position: None })', src/etcher.rs:703:71
note: run with `RUST_BACKTRACE=1` environment va
riable to display a backtrace
Dislodging frame ended in 4841ms
```

Fig. 5. Luaran dekripsi pada terminal saat private key tidak valid

IV. Kesimpulan dan Saran Pengembangan

Algoritma RSA adalah satu dari sekian banyak algoritma yang dapat digunakan untuk mengamankan sebuah data dengan cara mengenkripsinya. Penerapan algoritma RSA pada konteks ini berhasil memastikan bahwa pihak yang tidak memiliki kunci privat tidak akan bisa mengetahui isi pesan yang di-*embed* pada video.

Saran pengembangan untuk solusi ini, kedepannya dapat dikembangkan lebih lanjut dalam beberapa bagian, seperti penyimpanan kunci publik yang lebih baik, serta *interface* untuk memasukkan kunci privat maupun publik yang akan dipakai untuk sesi-sesi berikutnya.

Ucapan Terima Kasih

Penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas berkah dan rahmatnya sehingga makalah ini yang berjudul “Implementasi Algoritma RSA dalam Mengenkripsi Data pada Video-based Storage” dapat diselesaikan dengan baik dan lancar. Penulis juga mengucapkan terima kasih kepada bapak Rinaldi Munir sebagai pengampu mata kuliah IF4020 Kriptografi tahun akademik 2022/2023 semester genap yang sudah membagikan ilmunya kepada seluruh mahasiswa Teknik Informatika ITB angkatan 2020 dan angkatan 2019. Ucapan terima kasih tak lupa penulis ucapkan kepada orangtua dan keluarga yang selalu mendukung penulis, dan teman-teman terdekat penulis yang memberikan dukungan baik dari segi mental maupun materil.

Fig. 6. Contoh tangkapan layar data yang ter-*embed* pada video

References

[1] Rotenstreich, Shmuel. "The Difference between Electronic and Paper Documents" (PDF). George Washington University. Diakses 20 Mei 2023.

[2] David Austerberry, The Technology of Video and Audio Streaming, Second Edition, Sep 2004 pp: 328

[3] https://github.com/DvorakDwarf/Infinite-Storage-Glitch. Diakses 20 Mei 2023.

[4] https://hackaday.com/2023/02/21/youtube-as-infinite-file-storage/. Diakses 20 Mei 2023