Penggunaan Algoritma Elliptic Curve Diffie Hellman dan AES 256 pada Implementasi *End-to-End Encryption* WhatsApp

Graciella Valeska Liander (18219075)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
gracedmi.gvl@gmail.com

***Abstract*—Meningkatnya intensitas penggunaan komunikasi digital menyebabkan banyaknya data yang tersebar secara bebas di jaringan. Untuk tetap menjaga keamanan data penggunanya, WhatsApp, salah satu platform messaging terbesar di dunia menerapkan sebuah metode bernama *end-to-end encryption*. Metode ini bertujuan untuk mengenkripsi pesan di perangkat penggunanya sehingga data hanya dapat dibaca oleh pengirim pesan dan penerima pesan saja. Bahkan, WhatsApp sendiri tidak dapat mengetahui pesan tersebut. Untuk melakukan *end-to-end encryption*, WhatsApp menggunakan algoritma pertukaran kunci, yaitu algoritma *Elliptic Curve Diffie Hellman* serta algoritma kunci simetri AES 256 untuk mengenkripsi pesannya [5]. Pada makalah kali ini, penulis akan menjelaskan tentang cara kerja algoritma Elliptic Curve Diffie Hellman serta implementasinya pada sistem *end-to-end encryption* yang digunakan oleh WhatsApp.**

Keywords—diffie hellman algorithm, elliptic curve diffie-hellman, end-to-end encryption

I. Pendahuluan

Saat ini, komunikasi secara digital sudah sangat umum dilakukan. Komunikasi tersebut dapat dilakukan melalui platform seperti Facebook Messenger, LINE, WhatsApp, Telegram, dan masih banyak platform *messaging* lainnya. Terkadang, pesan yang dikirim melalui platform tersebut bersifat rahasia sehingga kita tidak ingin ada pihak lain yang membaca pesan tersebut. Meskipun pesan yang disimpan kedalam database server dapat berupa pesan yang sudah di enkripsi, tetap ada peluang bagi penyerang untuk mencuri informasi selama perjalanan pesan dari *client* ke *server*. Alasan ini lah yang membuat salah satu platform messaging terbesar saat ini, WhatsApp melakukan *end-to-end encryption* pada aplikasinya. End-to-end encryption merupakan salah satu metode untuk mengamankan komunikasi digital dengan melakukan enkripsi pesan di client. Dengan menggunakan metode ini, pesan hanya dapat dibaca di *device client* yang mengirim pesan dan *device client* yang menerima pesan.

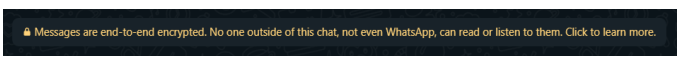

Fig. 1. Ilustrasi Penggunaan *End-to-End Encryption* pada WhatsApp (sumber: dokumentasi pribadi)

II. Metode

A. Literature Review

Langkah pertama yang dilakukan dalam pembuatan makalah ini adalah melakukan literature review, yaitu mencari informasi dari sumber digital berupa jurnal, paper, maupun materi perkuliahan mengenai *diffie hellman, elliptic curve diffie hellman*, AES 256, *end-to-end encryption* serta penerapannya pada aplikasi WhatsApp.

B. Eksperimen

Selanjutnya, penulis melakukan eksperimen dengan membuat program dengan menggunakan library crypto pada sebuah *simple messaging website*. Program ini ditulis menggunakan JavaScript dengan bantuan framework React.js.

Terdapat 2 mode yang dimiliki oleh halaman web, yaitu sebelum melakukan pertukaran *key* dan setelah melakukan pertukaran *key*. Sebelum melakukan pertukaran key, seharusnya tidak terdapat perubahan pesan yang dikirim. Sedangkan, setelah melakukan pertukaran key, pesan akan berubah menjadi kumpulan string yang tidak memiliki makna yang berarti.

III. Landasan Teori

A. Algoritma Diffie Hellman

Algoritma Diffie Hellman merupakan algoritma yang digunakan untuk melakukan pertukaran kunci. Algoritma ini bukan merupakan algoritma utama untuk melakukan enkripsi melainkan algoritma pendukung yang dapat membantu proses enkripsi menjadi lebih cepat. Algoritma ini biasanya dipasangkan dengan algoritma kriptografi kunci simetri untuk menemukan kunci simetri bersama.

Berikut adalah ilustrasi implementasi algoritma diffie hellman.

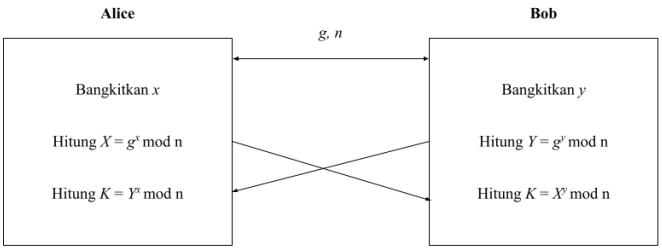

Fig. 2. Ilustrasi Implementasi Algoritma Diffie Hellman (sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-danKoding/Tanda-tangan-digital-2021.pdf)

Berikut adalah langkah penggunaan algoritma DH:

1. Pilih sembarang pasangan prima (g, n) dengan $g < n$.
2. A memilih sebuah bilangan bulat acak x yang akan menjadi private key A dan mengirimkan hasil perkalian berikut ke B
$$X = g^x \bmod n$$
3. B memilih sebuah bilangan bulat acak y yang akan menjadi private key B dan mengirimkan hasil perkalian berikut ke A
$$Y = g^y \bmod n$$
4. Masing-masing A dan B membuat secret key dengan mengalikan kunci privatenya dengan kunci publik yang diterima.
$$K = Y^x \bmod n = X^y \bmod n$$

B. *Algoritma Elliptic Curve Diffie Hellman*

Elliptic Curve Cryptography merupakan algoritma yang umum digunakan untuk menggantikan kriptografi kunci publik seperti RSA/ DSA. Hal ini dikarenakan ECC memiliki tingkat keamanan yang sama dengan RSA/DSA dengan panjang kunci yang lebih pendek. Berikut adalah perbandingan panjang kunci yang diperlukan dihasilkan oleh kedua algoritma tersebut.

TABLE I. PERBANDINGAN PANJANG KUNCI DENGAN TINGKAT KEAMANAN SETARA

Panjang Kunci RSA/ DSA	***Panjang Kunci ECC***
1024	160
2048	224
3072	256
7680	384
15360	512

(Sumber: Analysis of Diffie-Hellman Key Exchange Algorithm with Proposed Key Exchange Algorithm [7] page 168)

Salah satu pengembangan ECC adalah Elliptic Curve Diffie Hellman (ECDH). Metode ini menawarkan tingkat keamanan yang lebih tinggi jika dibandingkan dengan algoritma Diffie Hellman dengan panjang kunci yang lebih pendek [7]. Selain itu, EHDC juga memiliki tingkat *processing overhead* yang lebih rendah dan tingkat komputasi yang lebih cepat.

Berikut adalah langkah penggunaan algoritma ECC DH:

5. Pilih sebuah angka prima $p \approx 2^{180}$ dan sebuah kurva parameter a dan b untuk persamaan kurva eliptik
$$y^3 = x^3 + ax + b \,(mod\, p)$$
6. Pilih sebuah titik $G = (x_1, y_1) \in E_p(a, b)$ yang nilainya lebih besar dari n
7. A memilih sebuah integer $n_A < n$ yang akan menjadi private key A. Selanjutnya, A dapat membuat public key dengan menggunakan
$$P_A = n_A \times G \in E_p(a, b)$$
8. B memilih sebuah integer $n_B < n$ yang akan menjadi private key B. Selanjutnya, B dapat membuat public key dengan menggunakan
$$P_B = n_B \times G \in E_p(a, b)$$
9. Selanjutnya, A memberikan P_A kepada B dan B memberikan P_B kepada A.
10. Masing-masing A dan B membuat secret key dengan mengalikan kunci privatenya dengan kunci publik yang diterima.
$$K = n_A \times P_B = n_B \times P_A$$

C. *AES 256*

Algoritma AES atau dikenal sebagai algoritma Rijndael merupakan algoritma kriptografi kunci simetri yang menggunakan kunci sepanjang 128, 192, atau 256 bit untuk mengubah plaintext sepanjang 128 bit menjadi ciphertext sepanjang 128 bit. Algoritma ini memanfaatkan fungsi substitusi dan permutasi. Pada setiap jenis AES, baik AES 128, AES 192, maupun AES 256, proses yang dilakukan untuk melakukan enkripsi pada dasarnya sama. Perbedaannya hanya terdapat pada jumlah permutasi yang dilakukan. Pada AES 256, terdapat 14 round permutasi yang dilakukan dalam proses enkripsinya.

Secara garis besar, berikut adalah cara kerja algoritma AES.

1. Initial Round, fase untuk melakukan XOR antara plaintext dengan key.
2. Proses selama permutasi, fase yang dilakukan sebanyak *Nr-1* kali. Setiap putaran, proses yang dilakukan adalah:
    a. SubBytes: substitusi byte dengan menggunakan tabel substitusi (S-box)

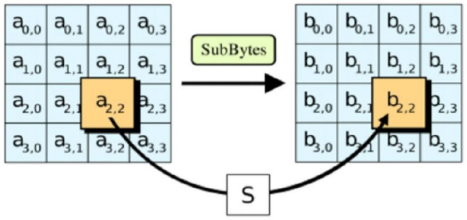

Fig. 3. SubBytes (sumber: AES Encryption: Study & Evaluation)

b. ShiftRows: pergeseran baris-baris array state secara wrapping

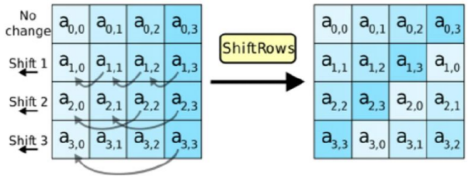

Fig. 4. ShiftRows (sumber: AES Encryption: Study & Evaluation)

c. MixColumns: mengacak data di masing-masing kolom array state.

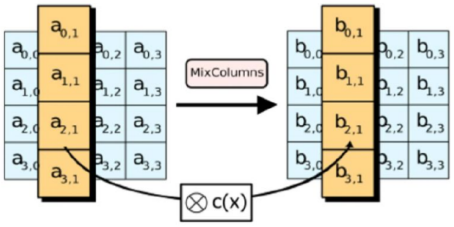

Fig. 5. MixColumns (sumber: AES Encryption: Study & Evaluation)

d. AddRoundKey: melakukan XOR antara state sekarang round key

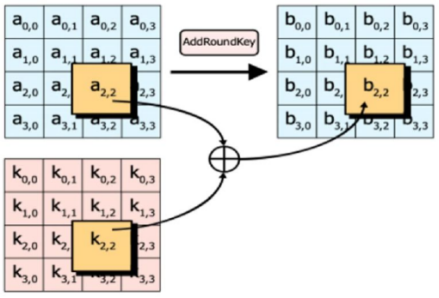

Fig. 6. AddRoundKey (sumber: AES Encryption: Study & Evaluation)

3. Final Round, fase putaran terakhir

a. SubBytes: substitusi byte dengan menggunakan tabel substitusi (S-box)
b. ShiftRows: pergeseran baris-baris array state secara wrapping
c. AddRoundKey: melakukan XOR antara state sekarang round key

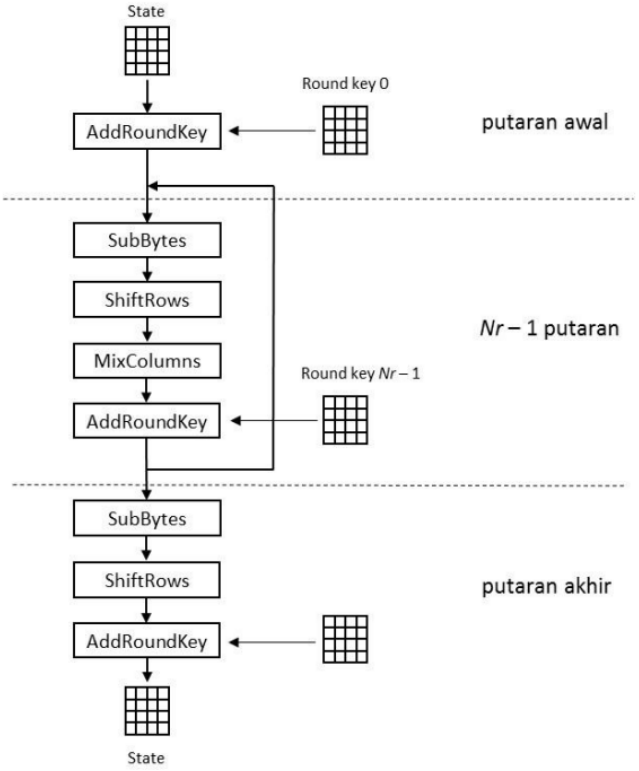

Fig. 7. Ilustrasi Algoritma AES 256(sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-danKoding/Tanda-tangan-digital-2021.pdf)

D. *End-to-End Encryption*

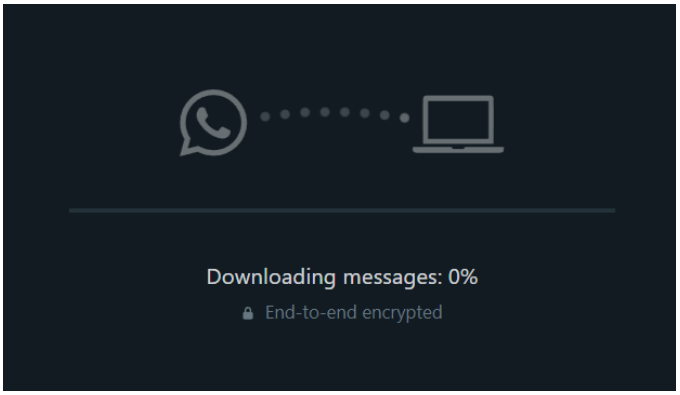

Fig. 8. Tampilan informasi *end-to-end encrypted* oleh WhatsApp Web (sumber: dokumentasi pribadi)

End-to-end encryption merupakan salah satu metode enkripsi yang digunakan dengan melakukan enkripsi pada client, bukan pada server. Dengan menggunakan metode ini, pesan hanya dapat diketahui oleh pengirim pesan dan penerima pesan saja. Berikut adalah ilustrasi implementasi *end-to-end encryption*.

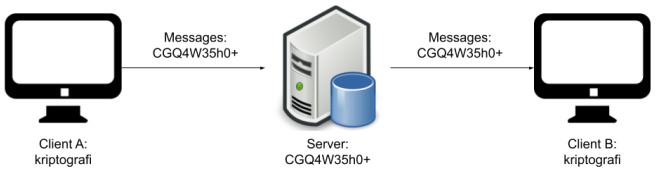

Fig. 9. Ilustrasi penggunaan *end-to-end encrypted* (sumber: dokumentasi pribadi)

Untuk memberikan gambaran yang lebih jelas terkait dengan perbedaan pengiriman pesan menggunakan *end-to-end encryption* dan pengiriman pesan biasa, berikut adalah ilustrasi pengiriman pesan tanpa menggunakan *end-to-end encryption*

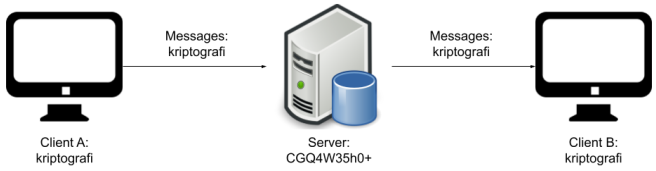

Fig. 10. Ilustrasi Pengiriman Pesan Tanpa End-to-End Encryption

Dapat dilihat perbedaan pada Fig. 4. dan Fig. 5. terdapat pada pesan ketika berpindah dari *client* ke *server*. Pada implementasi *end-to-end encryption*, pesan yang terkirim merupakan pesan yang sudah dienkrip. Server tidak mengetahui pesan asli dan tidak dapat melakukan dekripsi pesan tersebut karena tidak memiliki kunci untuk mendekripsinya. Sedangkan, pada ilustrasi pengiriman pesan tanpa *end-to-end encryption*, pesan yang dikirim ke server merupakan pesan asli. Server dapat mengetahui isi pesannya sebelum menyimpannya dalam versi yang sudah dienkrip. Metode pengiriman pesan ini memiliki kerentanan karena dapat dilakukan penyerangan apabila attacker memiliki akses diantara client dan server. Pesan dapat dengan mudah dimodifikasi maupun dihapus.

Untuk melakukan implementasi *end-to-end encryption*, pertama-tama kedua client akan melakukan *key exchange* (pertukaran kunci). Terdapat beberapa jenis algoritma pertukaran kunci yang dapat digunakan. Namun, algoritma yang paling umum adalah algoritma *diffie hellman* dan *elliptic curve diffie hellman* seperti yang sudah dijelaskan pada bagian sebelumnya.

Selanjutnya, setelah kedua client menerima kunci simetri yang sama, mereka akan melakukan enkripsi sebelum mengirim pesan dan melakukan dekripsi sebelum menerima pesan. Pihak lain tidak dapat membaca pesan yang dikirim karena mereka tidak memiliki kunci simetri yang digunakan untuk mengenkripsi pesan tersebut. Algoritma enkripsi yang biasa digunakan adalah algoritma kriptografi kunci simetri. Algoritma ini dipilih karena kedua client sudah memiliki kunci yang serupa dan kompleksitas komputasi yang rendah.

E. Kelebihan dan Kekurangan End-to-End Encryption

Kelebihan utama dari penggunaan *end-to-end encryption* adalah privasi yang lebih terjaga karena pesan hanya dapat dibaca oleh pengirim pesan dan penerima pesan. Kelebihan lainnya adalah integritas pesan akan sangat terjaga karena tidak ada pihak yang bisa mengganti pesan. Apabila pesan diubah, maka ketika pesan didekripsi, isi pesan akan menjadi kacau.

Meskipun demikian, tetap terdapat kekurangan dari implementasi *end-to-end encryption* tersebut. Kekurangan pertama adalah karena kunci pesan hanya disimpan di perangkat pengirim dan penerima pesan, apabila perangkat tersebut dapat diakses oleh *attacker*, maka *attacker* bisa langsung membaca pesan, serta menulis dan mengirim pesan atas nama pemilik perangkat.

IV. IMPLEMENTASI

Untuk memberikan gambaran yang lebih jelas terkait dengan implementasinya, penulis telah membuat sebuah halaman website yang serupa dengan WhatsApp. Website dapat diakses melalui link https://e2ee-diffie-hellman.vercel.app/. Berikut adalah tampilan halaman website.

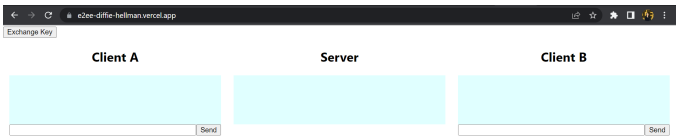

Fig. 11. Tampilan antarmuka halaman web (sumber: dokumentasi pribadi)

Dalam implementasinya, terdapat 3 fungsi utama yang digunakan, yaitu fungsi exchangeKey, encrypt, dan decrypt. Berikut adalah penjelasan untuk masing-masing fungsi

A. Fungsi Pertukaran Kunci

Algoritma fungsi pertukaran kunci yang digunakan adalah algoritma *Elliptic Curve Diffie Hellman*. Algoritma ini sesuai dengan algoritma yang digunakan oleh WhatsApp dan tertulis pada technical white paper.

Pada implementasi yang digunakan oleh makalah ini, program akan dibantu dengan sebuah library bernama crypto. Library ini memiliki fungsi createEHDC yang akan digunakan untuk menginisiasi penggunaan algoritma EHDC. Tipe curve yang digunakan adalah kurva secp256k1. Berikut adalah ilustrasi kurva secp256k1 yang digunakan pada fungsi tersebut:

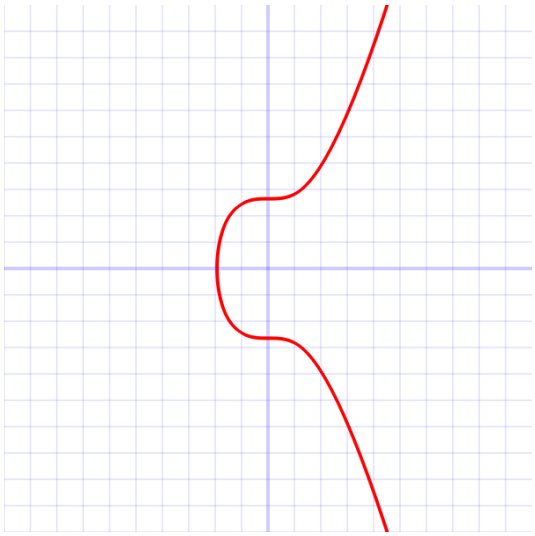

Fig. 12. Tipe Kurva secp256k1 (sumber: Wikipedia)

Setelah berhasil membuat kunci untuk masing-masing user, kita akan membuat sebuah secret yang diketahui oleh kedua user. Fungsi tersebut memerlukan public key dari user lain, tipe encoding, serta format binarynya. Hasil dari *secret key* ini akan sama untuk kedua user. *Key* inilah yang akan disimpan dan digunakan selama percakapan.

```
const exchangeKey = () => {
  setMessage([]);
  const alice = crypto.createECDH("secp256k1");
  alice.generateKeys();

  const bob = crypto.createECDH("secp256k1");
  bob.generateKeys();

  const alicePublicKeyBase64 = alice.getPublicKey().toString("base64");
  const bobPublicKeyBase64 = bob.getPublicKey().toString("base64");

  const aliceSharedKey = alice.computeSecret(
    bobPublicKeyBase64,
    "base64",
    "hex"
  );

  const bobSharedKey = bob.computeSecret(
    alicePublicKeyBase64,
    "base64",
    "hex"
  );

  setSharedKey({ alice: aliceSharedKey, bob: bobSharedKey });
};
```

Fig. 13. Blok Kode Fungsi exchangeKey (sumber: dokumentasi pribadi)

Setelah berhasil membuat kunci untuk masing-masing user, kita akan membuat sebuah secret yang diketahui oleh kedua user. Fungsi tersebut memerlukan public key dari user lain, tipe encoding, serta format binarynya. Hasil dari *secret key* ini akan sama untuk kedua user. *Key* inilah yang akan disimpan dan digunakan selama percakapan.

B. *Fungsi Enkripsi*

Proses enkripsi akan menggunakan algoritma kunci simetri, yaitu AES 256. Pada library crypto, algoritma ini memiliki kode “aes-256-gcm”. Enkripsi ini menggunakan key yang telah dihasilkan pada fungsi exchangeKey.

```
const encrypt = (plain) => {
  if (!sharedKey.alice) return plain;
  const IV = crypto.randomBytes(16);
  const cipher = crypto.createCipheriv(
    "aes-256-gcm",
    Buffer.from(sharedKey.alice, "hex"),
    IV
  );

  let encrypted = cipher.update(plain, "utf8", "hex");
  encrypted += cipher.final("hex");

  const auth_tag = cipher.getAuthTag().toString("hex");

  const payload = IV.toString("hex") + encrypted + auth_tag;
  const payload64 = Buffer.from(payload, "hex").toString("base64");

  return payload64;
};
```

Fig. 14. Blok Kode Fungsi Enkripsi (sumber: dokumentasi pribadi)

C. *Fungsi Dekripsi*

Proses dekripsi yang dilakukan juga menggunakan library crypto dengan membuat sebuah decipheriv. Karena kedua pihak sudah memiliki key, maka proses dekripsi dapat dengan mudah dilakukan.

```
const decrypt = (cipher) => {
  if (!sharedKey.alice) return cipher;

  const decrypted_payload = Buffer.from(cipher, "base64").toString("hex");
  const decrypted_iv = decrypted_payload.substr(0, 32);
  const decrypted_text = decrypted_payload.substr(
    32,
    decrypted_payload.length - 32 - 32
  );
  const decrypted_auth_tag = decrypted_payload.substr(
    decrypted_payload.length - 32,
    32
  );

  let decrypted = "";

  try {
    const decipher = crypto.createDecipheriv(
      "aes-256-gcm",
      Buffer.from(sharedKey.bob, "hex"),
      Buffer.from(decrypted_iv, "hex")
    );

    decipher.setAuthTag(Buffer.from(decrypted_auth_tag, "hex"));

    decrypted = decipher.update(decrypted_text, "hex", "utf8");
    decrypted += decipher.final("utf8");
  } catch (error) {
    console.log(error.message);
  }
  return decrypted;
};
```

Fig. 15. Blok Kode Fungsi Dekripsi (sumber: dokumentasi pribadi)

V. PEMBAHASAN

Untuk melakukan pengujian apakah end-to-end encryption telah berhasil diimplementasikan, penulis membuat sebuah tombol *exchange key*. Sebelum tombol ditekan, belum terdapat pertukaran kunci sehingga tidak dilakukan proses enkripsi maupun dekrispi. Oleh karena itu, hasil pesan yang dikirim

client A ke client B tidak mengalami perubahan. Selain itu, data pada server juga tidak berubah. Proses pengujian dapat dilihat pada gambar berikut:

Fig. 16. Tampilan layar pengujian tanpa menggunakan *end-to-end encryption* (sumber: dokumentasi pribadi)

Setelah melakukan pertukaran kunci, client A dan client B akan memiliki kunci simetri. Kunci ini digunakan untuk melakukan enkripsi pesan. Algoritma yang digunakan adalah AES 256.

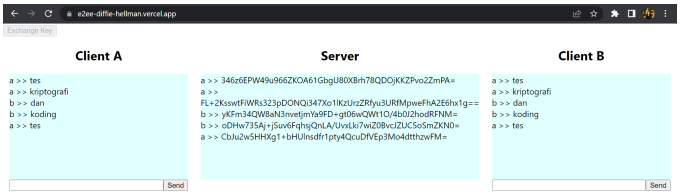

Fig. 17. Tampilan layar pengujian dengan menggunakan *end-to-end encryption* (sumber: dokumentasi pribadi)

Pada proses enkripsi, meskipun menggunakan kunci yang sama dan pesan yang sama. Hasil enkripsi dapat berbeda. Hal ini juga berkontribusi dalam meningkatkan tingkat keamanan pesan yang disimpan.

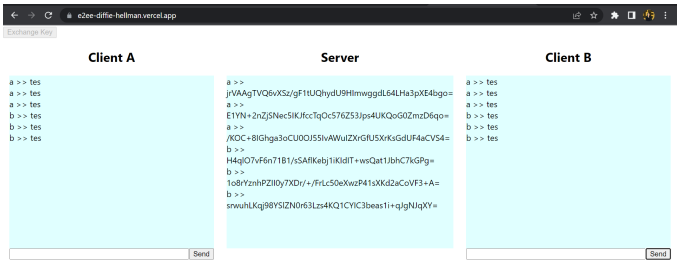

Fig. 18. Tampilan layar pengujian implementasi algoritma AES 256 (sumber: dokumentasi pribadi)

Berdasarkan hasil pengujian tersebut, dapat dibuktikan bahwa penggunaan *end-to-end encryption* sangat berperan dalam menjaga keamanan pesan. Pesan yang dikirim hanya dapat dibaca oleh Client A dan Client B.

Algoritma ECDH hanya perlu dilakukan di awal sebagai inisiasi percakapan. Sebelum algoritma ini digunakan, tidak terjadi proses enkripsi sehingga proses pengiriman data tidak aman. Namun, setelah melakukan pertukaran kunci, data akan dienkripsi sebelum dikirim dan didekripsi sebelum ditampilkan ke penerima. Metode ini lebih aman dalam menjaga pesan ketika sedang dikirim. Selain itu, metode ini terhitung cukup efektif karena hanya perlu dilakukan sekali di awal inisiasi percakapan saja dan dapat digunakan sepanjang percakapan.

Meskipun demikian, ketika halaman berganti atau terdapat kunci baru yang dibuat, pesan sebelumnya sudah tidak dapat dibaca lagi karena kunci yang digunakan untuk melakukan enkripsi dan dekripsi sudah berbeda. Konsep ini serupa dengan konsep yang digunakan oleh WhatsApp. Setiap pengguna akan memiliki sebuah *key* di *device* yang digunakan untuk login ke WhatsApp. Ketika pengguna mengganti perangkatnya dan berusaha login menggunakan akun yang sama, pesan dari perangkat lamanya tidak dapat dibuka di perangkat baru. Hal ini dikarenakan perangkat barunya tidak memiliki kunci yang digunakan untuk mengenkripsi pesan.

Berdasarkan dokumentasi yang diberikan oleh WhatsApp pada dokumen Technical White Paper, metode enkripsi yang digunakan adalah AES 256. Algoritma ini digunakan karena AES merupakan protokol keamanan yang paling kuat karena tidak hanya melibatkan software, tetapi juga hardware [9]. Selain itu, untuk memecahkan 128 bits, diperlukan sekitar 2128 percobaan sehingga sangat sulit untuk melakukan proses *hacking*. Sebenarnya, metode enkripsi-dekripsi pesan tidak terlalu berpengaruh pada implementasi *end-to-end encryption* selama kunci dan algoritma telah disepakati menggunakan metode ECDH.

VI. Kesimpulan dan Saran

Berdasarkan hasil pembahasan, dapat disimpulkan bahwa *end-to-end encryption* bermanfaat untuk mengurangi kebocoran pesan yang disalurkan melalui internet. Pada WhatsApp, *end-to-end encryption* digunakan menggunakan algoritma *elliptic curve diffie hellman* dan AES 256. Algoritma *elliptic curve diffie hellman* digunakan untuk melakukan pertukaran kunci sebelum menggunakan algoritma kriptografi kunci simetri AES 256.

Video Link at Youtube

https://youtu.be/oqi8oupGtO0

Source Code

https://github.com/graciellavl/e2ee-diffie-hellman

Ucapan Terima Kasih

Puji dan syukur saya ucapkan kepada Tuhan Yang Maha Esa yang telah membantu saya dalam pembuatan makalah ini. Tak lupa juga kepada kedua orang tua saya yang senantiasa memberikan dukungan kepada saya. Saya juga ingin mengucapkan terima kasih kepada Pak Rinaldi Munir yang sudah memberikan materi tentang Kriptografi dan Koding dan mengemasnya dalam bentuk yang mudah dipahami.

Terdapat lebih banyak pihak yang membantu saya dalam proses pembuatan makalah ini seperti rekan-rekan yang mengambil mata kuliah Kriptografi dan Koding, kakak tingkat yang memberikan referensi pembuatan makalah, serta penulis-penulis yang mempublikasikan hasil penelitiannya di internet.

References

[1] Andreas Steffen, Elliptic Curve Cryptography, Zürcher Hochschule Winterthur.

[2] Debdeep Mukhopadhyay, Elliptic Curve Cryptography , Dept of Computer Sc and Engg IIT Madras.

[3] Anoop MS, Elliptic Curve Cryptography, an Implementation Guide

[4] William Stallings, Cryptography and Network Security Chapter 10, 5th Edition

[5] WhatsApp (2021) WhatsApp Encryption Overview Technical White Paper.

[6] Susantio, D. R., Muchtadi-Alamsyah, I , (2016), "Implementation of Elliptic Curve Cryptography in Binary Field." Journal of Physics: Conference Series, vol.710, no. 1, https://doi.org/10.1088/1742-6596/710/1/012022.

[7] Kumar, R., Ravindranath, C. C., (2015), "Analysis of Diffie-Hellman Key Exchange Algorithm with Proposed Key Exchange Algorithm." International Journal of Emerging Trends Technology in Computer Science (IJETTCS), vol. no. 1, p. 40-43.

[8] Srinivasa naresh, Vankamamidi & Murthy, Nistala V.E.S. & Rao, V.V.. (2011). Elliptic Curve Cryptography-Diffie-Hellman Technique Extended ForMultiple Two Party Keys At A Time. International Journal of Computer Science Issues.. Vol. 8. P167-172.

[9] Sousi, Ahmad-Loay & Yehya, Dalia & Joudi, Mohamad. (2020). AES Encryption: Study & Evaluation.

PERNYATAAN

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

Bandung, 25 Mei 2021

Graciella Valeska Liander (18219075)