Sistem Pembagian File Sederhana Berbasis Proxy Re-encryption

Haziq Abiyyu Mahdy - 13521170
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: haziq.a.mahdy@gmail.com, 13521170@std.stei.itb.ac.id

***Abstrak*—Dalam era *cloud computing*, kebutuhan untuk berbagi data secara aman dan efisien menjadi semakin penting. Salah satu tantangan utama dalam sistem pembagian file adalah bagaimana menjaga kerahasiaan data saat melibatkan pihak ketiga sebagai perantara penyimpanan dan distribusi. Makalah ini mengusulkan dan mengimplementasikan sistem pembagian file sederhana berbasis *proxy re-encryption* (PRE), suatu skema kriptografi yang memungkinkan pihak ketiga melakukan transformasi *ciphertext* dari satu penerima ke penerima lain tanpa dapat membaca isi data. Sistem yang diusulkan memanfaatkan skema Umbral sebagai implementasi PRE. *File* yang dibagikan tidak perlu dienkripsi ulang oleh pengirim saat hendak dibagikan ke pengguna lain, cukup dengan menghasilkan *re-encryption key*. Sistem ini dapat menjaga keamanan data meskipun disimpan di server, dan secara signifikan mengurangi beban penyimpanan serta komputasi dari sisi pengunggah file. Sistem ini cocok digunakan pada lingkungan *cloud* yang melibatkan banyak pengguna dan memerlukan fleksibilitas dalam pengaturan akses file terenkripsi.**

Kata kunci—pembagian file, kriptografi, keamanan data, proxy re-encryption

I. Pendahuluan

Berbagi file secara aman menjadi kebutuhan penting dalam berbagai aplikasi, mulai dari sistem *email* hingga layanan penyimpanan berbasis *cloud*. Dalam praktiknya, proses berbagi data seringkali melibatkan pihak ketiga seperti *server* atau penyedia layanan, yang berperan sebagai perantara penyimpanan dan distribusi. Hal ini menimbulkan tantangan baru dalam menjaga kerahasiaan dan integritas data, terutama ketika pihak ketiga tersebut tidak sepenuhnya dipercaya. Pendekatan tradisional seperti enkripsi menggunakan *public key cryptography* atau *hybrid cryptography* memiliki keterbatasan dari sisi efisiensi dan pengelolaan kunci, khususnya ketika data perlu dibagikan ke banyak penerima.

Proxy Re-encryption (PRE) merupakan salah satu solusi yang memungkinkan delegasi hak dekripsi tanpa perlu memberikan akses langsung terhadap *plaintext* atau *secret key*. Dengan PRE, pengunggah file cukup mengenkripsi file sekali, lalu memberikan *re-encryption key* kepada server (*proxy*) agar dapat mentransformasikan *ciphertext* agar dapat didekripsi oleh penerima lain. Salah satu implementasi PRE yang populer adalah skema Umbral, yang mendukung pengaturan ambang dan verifikasi re-enkripsi secara kriptografis.

Makalah ini membahas rancangan dan implementasi sistem pembagian file sederhana berbasis PRE, yang memungkinkan pengguna untuk berbagi file dengan aman melalui *server* yang semi-terpercaya. Sistem ini dibangun menggunakan library Umbral, dan menunjukkan bagaimana PRE dapat digunakan untuk meningkatkan efisiensi dan keamanan dalam proses distribusi file, tanpa mengorbankan fleksibilitas dalam pengaturan akses pengguna.

II. Landasan Teori

A. *Algoritma Kriptografi Kunci Simetris* [1]

Algoritma kriptografi kunci simetris adalah algoritma kriptografi yang menggunakan kunci yang sama untuk proses enkripsi dan dekripsi. Algoritma kriptografi simetris terbagi menjadi dua, yaitu :

1) *Stream cipher*
   Algoritma ini beroperasi pada bit-bit pesan secara individual. Enkripsi dan dekripsi pesan dilakukan secara *bitwise* dengan operasi XOR.

2) *Block cipher*
   Algoritma ini beroperasi pada blok-blok (sekumpulan bit) pesan, misalnya 128 bit per blok. Enkripsi dan dekripsi dilakukan secara blok-per-blok.

B. *Algoritma Kriptografi Kunci Asimetris (Public Key Cryptography)* [2]

Algoritma kriptografi kunci asimetris, atau sering disebut sebagai kriptografi kunci publik adalah algoritma yang menggunakan dua kunci yang berbeda untuk proses enkripsi dan dekripsi. Pengirim dan penerima pesan masing-masing mempunyai sepasang kunci, yaitu *public key pk* yang tidak bersifat rahasia dan *secret key sk* yang bersifat rahasia. Sebelum pesan dikirimkan kepada penerima, pesan terlebih dahulu dienkripsi menggunakan *public key* penerima. Kemudian, *ciphertext* dikirim dan oleh penerima akan didekripsi menggunakan *secret key* penerima. Dengan begitu, *secret key* tidak perlu dikirimkan melalui saluran komunikasi, dan hanya pemilik *secret key* yang mengetahui kuncinya sendiri.

C. *Hybrid Cryptography* [3]

Hybrid cryptography merupakan yang menggabungkan kriptografi kunci simetris dengan kriptografi kunci

asimetris. Misalkan Alice ingin mengirimkan pesan kepada Bob. Alice mengenkripsi pesan tersebut dengan algoritma kriptografi kunci simetris (misalnya AES). Kemudian, kunci simetris tersebut akan dienkripsi dengan algoritma kriptografi kunci asimetris, menggunakan *public key* milik Bob. Kemudian, pesan dan kunci yang sudah terenkripsi akan dikirimkan kepada Bob. Bob terlebih dahulu mendekripsi kunci simetri dengan *secret key* miliknya, kemudian mendekripsi pesan menggunakan kunci simetri yang telah didekripsi sebelumnya. Konsep ini menggabungkan kelebihan antara algoritma kriptografi kunci simetris yang cepat dalam mengenkripsi data yang berukuran besar, dengan algoritma kriptografi kunci asimetris yang memiliki *public key* dan *secret key* terpisah.

Selain untuk enkripsi, kriptografi kunci publik juga digunakan dalam proses tanda tangan digital (*digital signature*). Dalam hal ini, pengirim dapat menandatangani suatu pesan menggunakan secret key miliknya. Penerima atau pihak lain kemudian dapat memverifikasi keaslian dan integritas pesan tersebut menggunakan public key pengirim. Dengan demikian, tanda tangan digital menjamin bahwa pesan benar-benar berasal dari pengirim yang sah dan tidak mengalami perubahan selama transmisi.

D. *Skema Ambang dalam Skema Pembagian Rahasia* [4]

Skema ambang (*threshold scheme*) merupakan pendekatan dalam kriptografi yang memungkinkan pembagian suatu rahasia ke dalam *n* bagian atau *share*, sedemikian sehingga hanya sebagian tertentu (minimal *t* dari *n*) yang dibutuhkan untuk merekonstruksi kembali rahasia tersebut. Skema ambang umumnya dinotasikan sebagai *skema(t, n)*.

Salah satu skema ambang yang terkenal adalah Shamir's Secret Sharing, yang diperkenalkan oleh Adi Shamir pada tahun 1979. Skema ini didasarkan pada prinsip interpolasi polinomial. Untuk membagi rahasia, diperlukan polinomial acak berderajat *t-1* seperti pada persamaan (1)

$$f(x) = a_0+a_1x+a_2x^2+\cdots+a_{t-1}x^{t-1} \quad (1)$$

dengan $s=a_0=f(0)$ sebagai rahasia yang ingin disembunyikan, dan koefisien lainnya $a_i \in Zp$ dipilih secara acak. Selanjutnya, pembuat rahasia menghasilkan *n* buah share dalam bentuk pasangan $(x_i, f(x_i))$ dengan $x_i \in Zp \setminus \{0\}$ dapat dipilih secara acak dan tiap-tiap x_i bernilai unik. Setiap peserta menerima satu share. Untuk merekonstruksi rahasia, dibutuhkan minimal *t* share. Dengan menggunakan *interpolasi Lagrange* atau Vandermonde, nilai rahasia $s=f(0)$ dapat diperoleh kembali.

E. *Proxy Re-encryption (PRE)*

Proxy re-encryption (PRE) adalah salah satu konsep pada kriptografi yang memungkinkan suatu pesan yang pada awalnya ditujukan untuk suatu pihak (misalnya dienkripsi dengan *public key* milik Alice) dapat ditransformasi oleh pihak ketiga (*proxy*) sedemikian sehingga dapat didekripsi oleh pihak lain (misalnya menggunakan *secret key* milik Bob). Transformasi ini dilakukan tanpa harus mendekripsi pesan menjadi *plaintext* semula, sehingga *proxy* tidak dapat mengetahui pesan aslinya. Dengan begitu, Alice dapat mengenkripsi pesan dengan *secret key* miliknya, lalu pesan yang terenkripsi tersebut dikirimkan kepada *proxy*. Jika Alice ingin membagikan pesan tersebut kepada Bob, Alice cukup memberikan *re-encryption key* kepada *proxy* yang akan digunakan untuk mentransformasi pesan tersebut sedemikian sehingga dapat didekripsi oleh Bob [5]. Skema PRE dijelaskan melalui gambar II.1 berikut.

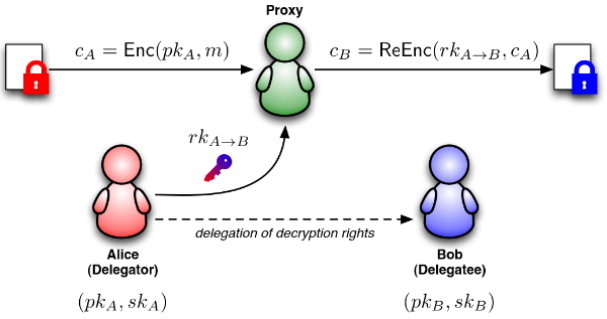

Fig II.1. PRE, diambil dari Nunez, D. A. V. I. D. "Umbral: a threshold proxy re-encryption scheme." NuCypher Inc and NICS Lab, University of Malaga, Spain (2018).

1) *Operasi yang terdapat pada PRE [6]*

Berikut adalah beberapa operasi yang tersedia pada PRE.

a) ***Setup***$(1^k) \rightarrow PP$
Diberikan parameter input 1^k, operasi ini menghasilkan parameter publik PP. Contoh parameter input adalah ukuran kunci (misalnya 128 bit), sedangkan contoh parameter publik adalah grup bilangan prima G atau parameter eliptik jika menggunakan *elliptic curve cryptography*.

b) ***KeyGen***$(PP) \rightarrow (pk, sk)$
Diberikan parameter publik PP, operasi ini menghasilkan pasangan *public key pk* dan *secret key sk*.

c) ***ReKeyGen***$(PP, sk_i, pk_j) \rightarrow rk_{i\to j}$
Diberikan parameter publik PP, *secret key* pengirim sk_i, dan *public key* penerima pk_j, operasi ini menghasilkan *re-encryption key* $\text{rk}_{\text{i}\to\text{j}}$.

d) ***Enc***$(PP, pk_i, m) \rightarrow C_i$
Diberikan parameter publik PP, *public key* pk_i dan *plaintext* *m*, operasi ini menghasilkan *ciphertext* C_i yang hanya dapat didekripsi dengan sk_i.

e) ***ReEnc***$(PP, rk_{i\to j}, C_i) \rightarrow C_j$
Diberikan parameter publik PP, *re-encryption key* $rk_{i\to j}$, dan *ciphertext* C_i,

operasi ini menghasilkan *ciphertext* C_j yang hanya dapat didekripsi dengan sk_j.

f) ***Dec****(PP, sk_i, C_i)* →m
Diberikan parameter publik PP, *secret key* sk_i, dan *ciphertext* C_i, operasi ini menghasilkan *plaintext m*.

Secara umum, properti enkripsi dan dekripsi sama seperti algoritma kriptografi asimetri pada umumnya, yaitu pesan yang dienkripsi menggunakan suatu *public key* pk_i dan didekripsi dengan *secret key* sk_i yang bersesuaian akan menghasilkan pesan semula, atau dinyatakan dalam persamaan (1).

$$Dec(PP, sk_i, Enc(PP, pk_i, m)) = m \quad (1)$$

Namun, terdapat properti tambahan dengan *re-encryption key*, yaitu pesan yang dienkripsi dengan *public key* pk_i, lalu dire-enkripsi dengan *re-encryption key* $rk_{i \to j}$, dan didekripsi dengan *secret key* sk_j akan menghasilkan pesan semula. Hal ini dinyatakan dalam persamaan (2).

$$Dec(PP, sk_j, ReEnc(PP, rk_{i \to j}, Enc(PP, pk_i, m))) = m \quad (2)$$

2) Beberapa Contoh Penggunaan PRE.
PRE sangat berguna pada kasus yang melibatkan pihak ketiga yang semi-terpercaya (*semi-trusted*) dalam pengiriman pesan rahasia. Berikut adalah beberapa contoh kasus yang dapat menggunakan PRE.

a) Penerusan *Email* Aman (*Secure Email Forwarding*)
Pengiriman *email* dari pengirim ke penerima umumnya akan *mail server* terlebih dahulu. Hal ini dapat berpotensi membahayakan keamanan pesan jika *mail server* dapat melihat pesan dalam bentuk *plaintext*. Salah satu solusi untuk menjaga keamanan *email* adalah dengan menggunakan PRE.

Misalkan Alice mengirim *email* kepada Bob. Alice mengenkripsi email menggunakan sendiri. Kemudian, Alice menghasilkan sebuah *re-encryption key*, yang memungkinkan *mail server* untuk mengubah *ciphertext* dari yang dienkripsi untuk Alice menjadi *ciphertext* yang dapat didekripsi oleh Bob. *Re-encryption key* ini kemudian diberikan kepada server email. Ketika Alice meminta *email* untuk diteruskan ke Bob, *mail server* akan menggunakan *re-encryption key* untuk mentransformasi *ciphertext* tersebut dan meneruskannya ke Bob. Kemudian, Bob dapat mendekripsi pesan tersebut menggunakan *secret key* miliknya.

b) Pengiriman PIN pada Bank [7]
Pada saat pengguna memasukkan PIN-nya ke mesin EDC untuk melakukan transaksi, PIN tersebut akan dienkripsi oleh EDC, kemudian dikirimkan kepada beberapa pihak secara berurutan, yaitu *payment gateway, merchant bank, network processor,* dan *issuer bank*. Setiap pesan sampai pada suatu pihak, pesan tersebut akan didekripsi dan dienkripsi ulang sebelum diteruskan kepada pihak selanjutnya, seperti pada gambar II.2 berikut.

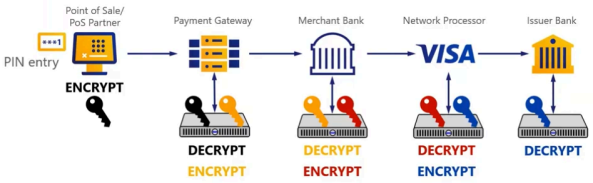

Fig II.2. Sistem Pengiriman PIN Konvensional, diambil dari Gaddam, Sivanarayana, et al. "Reducing {HSM} Reliance in Payments through Proxy {Re-Encryption}." *30th USENIX security symposium (USENIX Security 21)*. 2021.

Sistem ini memiliki kekurangan, yaitu tiap pihak dapat melihat *plaintext* dari PIN karena pihak tersebut mendekripsi PIN sebelum dienkripsi kembali dan diteruskan ke pihak selanjutnya. Solusi yang digunakan saat ini adalah dengan menggunakan *Hardware Security Module* (*HSM*), yaitu perangkat keras yang berfungsi khusus untuk melakukan operasi kriptografis dan dapat memastikan *plaintext* tidak akan keluar dari perangkat keras tersebut. Namun, *HSM* memiliki harga yang cukup mahal dan kurang *scalable*, karena semakin banyak pengguna, maka akan dibutuhkan semakin banyak *HSM* yang dibutuhkan. Sistem pengiriman PIN menggunakan *PRE* dijelaskan pada Gambar III.3 berikut.

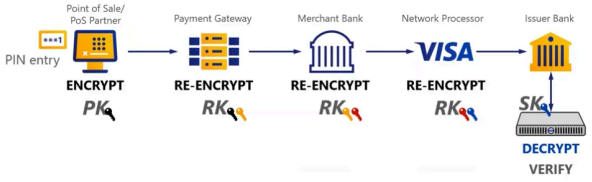

Fig II.3. Sistem Pengiriman PIN dengan PRE, diambil dari Gaddam, Sivanarayana, et al. "Reducing {HSM} Reliance in Payments through Proxy {Re-Encryption}." *30th USENIX security symposium (USENIX Security 21)*. 2021.

3) Skema Umbral: Salah Satu Implementasi PRE [8]

Dalam skema Umbral, proses utama yang terjadi adalah serangkaian langkah yang memungkinkan pihak ketiga (*proxy*) untuk mentransformasikan *capsule* (hasil enkapsulasi kunci) dari pengirim (misalnya Alice) agar dapat digunakan oleh penerima yang didelegasikan (misalnya Bob), tanpa mengungkapkan isi pesan asli maupun kunci rahasia. Proses ini dibagi menjadi tiga tahap utama: *Encapsulation*, *Re-encapsulation*, dan *Decapsulation*. Pertama, pengirim mengenkripsi data menggunakan kunci simetris yang dihasilkan melalui proses *encapsulation*, menghasilkan pasangan kunci dan *capsule*. *Capsule* ini kemudian dapat dire-enkripsi oleh sekumpulan proxy menggunakan fragmen kunci re-enkripsi (*kfrag*) yang sebelumnya dihasilkan oleh Alice untuk Bob. Setiap proxy menghasilkan sebuah *capsule fragment* (*cfrag*). Apabila Bob menerima sedikitnya *t* dari *N cfrag* yang sah, ia dapat melakukan *decapsulation* terhadap potongan tersebut menggunakan kunci privat miliknya. Hasil akhirnya adalah kunci simetris yang sama seperti yang dihasilkan oleh Alice, yang kemudian digunakan untuk mendekripsi pesan. Dengan desain ini, Umbral menjamin bahwa proses delegasi akses berlangsung secara aman, terdistribusi, dan dapat diverifikasi, tanpa mengorbankan kerahasiaan data asli. Skema Umbral dijelaskan melalui gambar II.4 berikut.

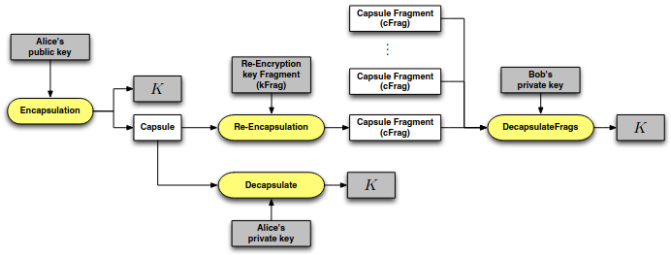

Fig II.4. Skema Umbral, diambil dari Nunez, D. A. V. I. D. "Umbral: a threshold proxy re-encryption scheme." NuCypher Inc and NICS Lab, University of Malaga, Spain (2018).

Dalam implementasinya, skema Umbral memanfaatkan beberapa konsep yang telah dijelaskan sebelumnya, di antaranya algoritma kriptografi kunci simetris (ChaCha20-poly1305 *cipher*), algoritma kriptografi publik (ECC), *hybrid cryptography,* tanda tangan digital, serta skema ambang yang terinspirasi dari Shamir's Secret Sharing (SSS).

III. Implementasi

Pada makalah ini, akan diimplementasikan sistem pembagian sederhana sebagai salah satu *use-case* dari PRE. Kasus ini digunakan karena melibatkan pembagian data (*file*) melalui pihak ketiga (*server*). Operasi-operasi pada PRE menggunakan *library* Umbral dari NuCypher Inc [9]. Terdapat salah satu penyesuaian yang dilakukan dengan skema Umbral, yaitu nilai t yang ditetapkan adalah 1, karena dalam kasus skema pembagian *file* sederhana, Bob (penerima) tidak memerlukan izin dari berbagai pihak terlebih dahulu. Selain itu, karena skema Umbral menggunakan konsep *digital signature* untuk menjaga integritas data, tiap pengguna pada sistem pembagian *file* yang dikembangkan akan memiliki dua pasang kunci.

A. *Encryption Key untuk Proses Enkripsi*, yang Terdiri atas:
   1) *Encryption Secret Key (encryption_sk)*
   2) *Encryption Public Key (encryption_pk)*

B. *Signing untuk Proses Penandatanganan dan Verifikasi, yang Terdiri atas:*
   1) *Signing Secret Key (signing_sk)*
   2) *Signing Public Key (signing_pk)*

Implementasi sistem pembagian *file* akan berupa aplikasi yang terdiri atas *client* dan *server*. Secara umum, *client* dapat mengunggah file, memberikan akses kepada client lain, serta mengakses file yang dibagikan oleh client lain. *Server* hanya bertindak sebagai *proxy* pada PRE, yang hanya dapat menyimpan *ciphertext* serta melakukan re-enkripsi, tetapi tidak bisa mendekripsi *ciphertext* tersebut. Selain itu, *server* juga menyimpan kredensial pengguna seperti *username* dan *password* untuk otentikasi serta *public key* untuk beberapa operasi pada PRE. Rancangan Server

Server diimplementasikan dengan bahasa Python dan kakas FastAPI untuk menerima *HTTP request*. *Server* menyimpan *file* dalam bentuk *ciphertext* pada *file system*, sedangkan *metadata file*, *permission*, dan kredensial pengguna disimpan di basis data SQLite dengan skema sesuai gambar III. 1 berikut.

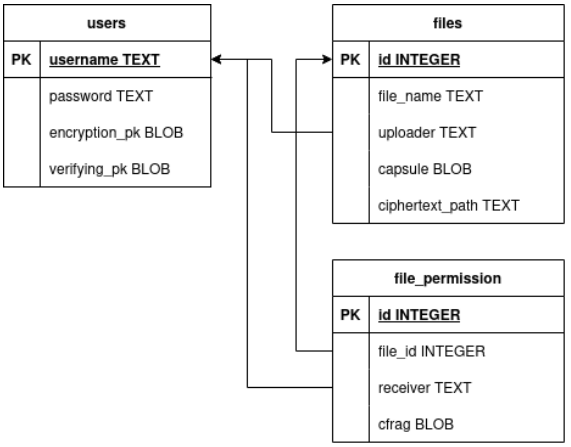

Fig III.1. Alice mengunggah *file* file.txt berisi rahasia dan memberikan akses ke Bob

Berikut adalah fungsionalitas yang disediakan pada *server*.

1) *Login*
   Fungsionalitas ini menerima masukan berupa *username* dan *password* dari *client* dan mencocokkan kredensial tersebut dengan basis data. Apabila sesuai, maka pengguna terotentikasi.

2) *Get public keys*

Fungsionalitas ini menerima masukan berupa *username* dan mengembalikan *encryption public key* dan *signing public key* dari pengguna dengan *username* tersebut. Fungsionalitas ini berguna untuk mendukung operasi PRE yang membutuhkan *public key*.

3) *Upload file*
Fungsionalitas ini menerima masukan berupa *file* dalam bentuk *ciphertext*, *capsule,* nama *file* dan *username* pengunggah file. *File* dalam *ciphertext* akan disimpan di *file system*, sedangkan *metadata* seperti pengunggah *file*, nama *file*, serta *capsule* disimpan di basis data.

4) *Grant access*
Fungsionalitas ini menerima masukan berupa nama *file*, *username* pengunggah *file* (yang sekaligus menjadi pengguna yang memberikan akses), *username* penerima *file*, serta *kfrag*, yang merupakan *re-encryption key.* Fungsionalitas ini akan melakukan proses re-enkripsi serta memasukkan *cfrag* (hasil re-enkripsi) ke dalam basis data beserta *username* penerima *file*.

5) *Access File*
Fungsionalitas ini menerima masukan berupa nama *file*, *username* pengunggah *file*, serta *username* penerima *file.* Fungsionalitas ini akan mencari *cfrag* (yang telah dimasukkan pada fungsionalitas *grant access*) di basis data. Kemudian, fungsionalitas ini akan mengembalikan *ciphertext, capsule*, serta *cfrag* ke *client* yang akan digunakan untuk mendekripsi *file* dengan *private key* penerima *file*.

A. Rancangan Client

Berikut adalah fungsionalitas yang disediakan pada *client*.

1) *Login dan Get Public Keys*
Kedua fungsionalitas ini menerima masukan dari pengguna dan meneruskannya pada fungsionalitas yang bersesuaian di *server* melalui *HTTP Request*.

2) *Upload File*
Fungsionalitas ini menerima sebuah *file* dan mengenkripsinya menggunakan *secret key* pengguna sehingga menghasilkan *ciphertext* dan *capsule.* Kemudian, fungsionalitas ini akan meneruskan *ciphertext, capsule*, serta *metadata file* (nama *file* dan pengunggah) pada fungsionalitas *upload file* di *server* melalui *HTTP Request*.

3) *Grant Access*
Fungsionalitas ini membangkitkan *kfrag* (*re-encryption key*) menggunakan *encryption secret key* pengunggah *file*, *signing secret key* pengunggah *file,* serta *public key* penerima *file*. Kemudian, *kfrag*, nama *file*, *username* pengunggah serta *username* penerima akan diteruskan ke fungsionalitas *grant access* di *server* melalui *HTTP Request*.

4) *Access File*
Fungsionalitas ini menerima nama *file*, nama pengunggah, serta nama penerima dan meneruskannya pada fungsionalitas *access file* di *server*. Kemudian, *server* akan mengembalikan *ciphertext, capsule,* serta *cfrag* yang akan digunakan untuk mendekripsi *file*. *File* yang telah didekripsi dapat diakses oleh pengguna.

B. Pengujian

Berikut adalah tahapan yang perlu dilakukan untuk berbagi *file* dari Alice ke Bob.

1) Dari sisi Alice
   a) *Membuat file rahasia*.
   b) *Login.*
   c) *Mengunggah file dengan fungsionalitas upload file*.
   d) *Memberikan akses ke bob dengan fungsionalitas grant access*.

2) Dari sisi Bob
   a) *Login.*
   b) *Mengakses file yang sudah diberikan oleh Alice dengan fungsionalitas access file.*
   c) *Membuka file.*

Apabila Alice ingin membagikan *file* yang sama ke Carol, dapat dilakukan tahapan berikut.

1) Dari sisi Alice
   a) *Memberikan akses ke Carol dengan fungsionalitas grant access* (Alice diasumsikan sudah *login* sebelumnya).

2) Dari sisi Carol
   a) *Login.*
   b) *Mengakses file yang sudah diberikan oleh Alice dengan fungsionalitas access file.*
   c) *Membuka file.*

Dengan tahapan tersebut, Bob dan Carol kini dapat mengakses *file* yang dibagikan oleh Alice. Untuk memverifikasi fungsionalitas sistem, dilakukan pengujian dengan skenario yang sama sebagai berikut.

1) Alice membuat *file* bernama file.txt yang berisi rahasia, kemudian mengunggahnya ke *server* dan memberikan akses ke Bob.

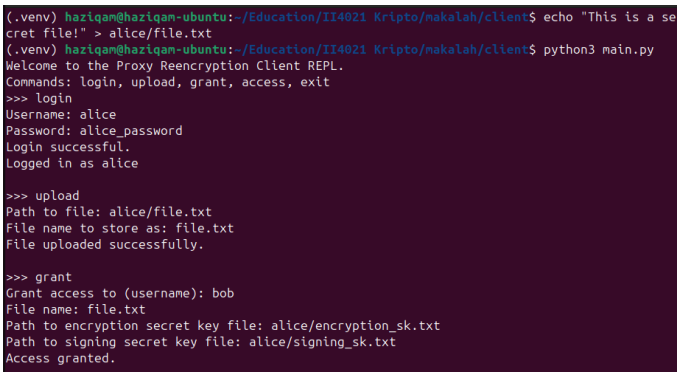

Fig III.2. Alice mengunggah *file* file.txt berisi rahasia dan memberikan akses ke Bob

2) Bob berhasil menyimpan file yang telah dibagikan Alice ke direktori miliknya

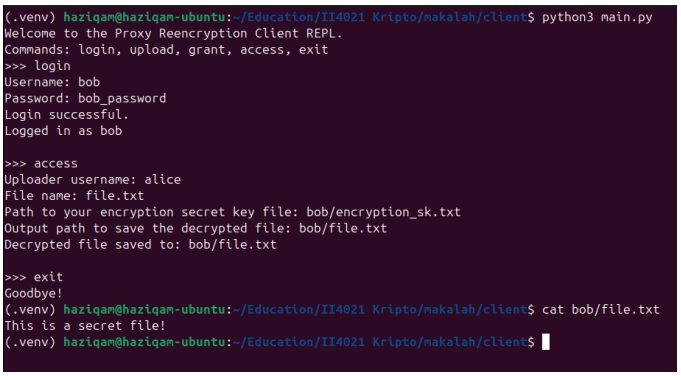

Fig III.3. Bob berhasil menyimpan *file* yang telah dibagikan Alice ke direktori miliknya

3) Alice memberikan akses *file* yang sama ke Carol

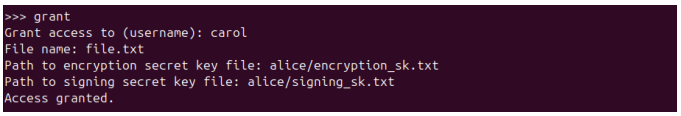

Fig III.4. Alice memberikan akses *file* yang sama ke Carol

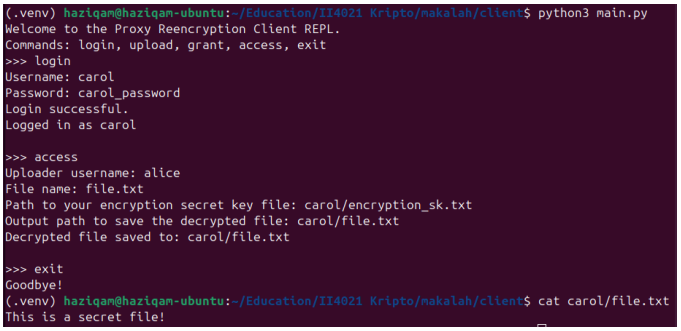

Fig III.5. Carol berhasil menyimpan *file* yang telah dibagikan Alice ke direktori miliknya

Gambar III.4 dan III.5 berurutan menunjukkan *file* yang tersimpan pada *server* berupa *ciphertext,* beserta isi tabel file dan file_permission pada basis data. *Server* tidak dapat mengetahui isi *file* karena masih dalam bentuk *ciphertext* dan hanya dapat didekripsi menggunakan *encryption secret key* milik Bob atau Carol.

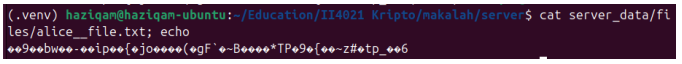

Fig III.6. Tampilan *file* yang tersimpan pada server

Fig III.7. Isi tabel file dan file_permission pada basis data.

IV. Analisis dan Pembahasan

Berdasarkan hasil pengujian, dapat disimpulkan bahwa sistem pembagian *file* dapat bekerja dengan baik untuk mengirimkan *file* dari satu pengguna ke pengguna lain, dan *server* tidak dapat mengetahui isi *file*. Alice (pengunggah *file*) hanya perlu membangkitkan *kfrag* (*re-encryption key*) apabila ia ingin menambahkan pengguna lain untuk mengakses *file* miliknya. Berikut adalah perbandingan pembagian *file* menggunakan PRE dengan beberapa sistem lainnya.

A. *Pembagian file dengan public key cryptography*

Apabila Alice ingin membagikan *file* ke Bob menggunakan *public key cryptography*, Alice perlu mengenkripsi *file* dengan *public key* milik Bob. Kemudian, *ciphertext* dikirimkan ke Bob dan Bob dapat mendekripsinya menggunakan *private key* miliknya. Apabila Alice ingin membagikan *file* yang sama ke Carol (pengguna lain), maka Alice harus melakukan langkah-langkah yang sama seperti sebelumnya.

Alice harus mengenkripsi *file* yang sama berkali-kali menggunakan *public key cryptography*, yang memerlukan komputasi yang mahal. Selain itu, Alice juga harus menyimpan *file* dalam bentuk *plaintext* secara lokal agar dapat dienkripsi lagi jika ia ingin membagikannya kepada pengguna lain. Alternatif lain adalah menyimpan *file* dalam bentuk *ciphertext* di *server*, namun Alice harus mendapatkan kembali *ciphertext* tersebut, mendekripsinya, mengenkripsi ulang, dan mengirimkannya ke pengguna lain.

Jika menggunakan PRE, Alice tidak perlu mengenkripsi *file* berulang kali untuk membagikan *file* ke pengguna lain. Alice cukup membangkitkan *re-encryption key*, kemudian *server* dapat menggunakannya untuk mere-enkripsi *ciphertext* yang sudah tersimpan dalam *server* agar dapat didekripsi menggunakan *secret key* milik penerima. Pendekatan ini juga akan lebih menghemat penyimpanan, karena Alice tidak perlu lagi menyimpan *file* dalam *plaintext* secara lokal. Alice

dapat mempercayakan penyimpanan *file* sepenuhnya ke *server*, karena *server* tidak bisa melihat isi *file*.

B. *Pembagian file dengan hybrid cryptography*

Apabila Alice ingin membagikan *file* ke Bob menggunakan *hybrid cryptography*, Alice perlu mengenkripsi *file* dengan kunci simetri terlebih dahulu. Kemudian, kunci simetri akan dienkripsi kembali dengan *public key* milik Bob dan keduanya akan dikirimkan ke Bob. Setelah itu, Bob akan mendekripsi kunci simetri terlebih dahulu menggunakan *secret key* miliknya. Setelah itu, kunci simetri akan digunakan untuk mendekripsi *file*.

Jika Alice ingin membagikan file yang sama ke Carol, maka Alice harus mengenkripsi ulang kunci simetri tersebut menggunakan *public key* milik Carol. Hal ini berarti Alice harus selalu menyimpan kunci simetri tersebut secara lokal agar dapat dienkripsi lagi jika ia ingin membagikannya kepada pengguna lain. Alternatif lainnya adalah mengenkripsi kunci simetri dengan *secret key* milik Alice dan menyimpannya di *server*. Namun, apabila Alice ingin membagikan *file* ke pengguna lain, maka ia harus mendapatkan kembali kunci simetri yang terenkripsi tersebut, mendekripsinya dengan *secret key* miliknya, dan mengenkripsinya kembali dengan *public key* milik penerima.

Jika menggunakan PRE, pengelolaan kunci simetri sudah ditangani secara aman di *server*, sehingga Alice tidak perlu menyimpannya sendiri atau mengenkripsinya secara manual. Alice hanya bertanggung jawab untuk membangkitkan *kfrag* (*re-encryption key*) dengan fungsionalitas yang sudah disediakan oleh pustaka PRE apabila ia ingin membagikan *file* miliknya ke pengguna lain.

V. KESIMPULAN

Berdasarkan pengujian, analisis dan pembahasan, dapat disimpulkan bahwa, penggunaan PRE dalam sistem pembagian *file* dapat menghemat penyimpanan dan komputasi dari sisi pemilik *file*, karena pemilik *file* dapat mendelegasikan kemampuan untuk menyimpan dan mere-enkripsi kepada *proxy*. Hal inilah yang menyebabkan PRE cocok digunakan pada sistem berbasis *cloud*, karena memungkinkan pemilik file untuk menyimpan data terenkripsi sepenuhnya di server tanpa kehilangan fleksibilitas dalam mengatur akses, serta tanpa mengorbankan kerahasiaan data. *Proxy* tidak perlu mengetahui isi file maupun *secret key* pengguna, namun tetap dapat membantu dalam proses distribusi *file* kepada penerima baru secara efisien. Dengan demikian, PRE memberikan kombinasi optimal antara keamanan, efisiensi, dan skalabilitas untuk kebutuhan berbagi file dalam lingkungan *cloud* yang dinamis dan melibatkan banyak pengguna.

PRANALA *SOURCE CODE*

Source code lengkap dapat diakses melalui pranala berikut. https://github.com/haziqam/makalah-kripto

UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur ke hadirat Allah Swt. karena atas rahmat dan karunia-Nya, makalah ini dapat diselesaikan dengan baik. Penulis juga berterima kasih kepada Dr. Rinaldi Munir, S.T., M.T. selaku dosen mata kuliah II4021 (Kriptografi) yang telah menyediakan materi pembelajaran yang digunakan pada makalah ini. Semoga makalah ini dapat memberi manfaat bagi penulis serta memberi inspirasi di kalangan mahasiswa teknik informatika.

REFERENSI

[1] Rinaldi Munir, Kriptografi Modern. Diakses pada 20 Juni 2025 melalui tautan berikut https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2024-2025/11-Kripto-modern-2025.pdf

[2] Rinaldi Munir, Kriptografi Kunci Publik. Diakses pada 20 Juni 2025 melalui tautan berikut https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2024-2025/18-Kriptografi-Kunci-Publik-2025.pdf

[3] Rinaldi Munir, Algoritma Diffie-Hellman. Diakses pada 20 Juni 2025 melalui tautan berikut https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2024-2025/21-Algoritma-Diffie-Hellman-2025.pdf

[4] Rinaldi Munir, Skema Pembagian Data Rahasia. Diakses pada 20 Juni 2025 melalui tautan berikut https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2024-2025/35-Skema-Pembagian-Data-Rahasia-2025.pdf

[5] Nuñez, David, Isaac Agudo, and Javier Lopez. "Proxy re-encryption: Analysis of constructions and its application to secure access delegation." *Journal of Network and Computer Applications* 87 (2017): 193-209.

[6] Ateniese, Giuseppe, Karyn Benson, and Susan Hohenberger. "Key-private proxy re-encryption." *Topics in Cryptology–CT-RSA 2009: The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*. Springer Berlin Heidelberg, 2009.

[7] Gaddam, Sivanarayana, et al. "Reducing {HSM} Reliance in Payments through Proxy {Re-Encryption}." *30th USENIX security symposium (USENIX Security 21)*. 2021.

[8] Nunez, D. A. V. I. D. "Umbral: a threshold proxy re-encryption scheme." NuCypher Inc and NICS Lab, University of Malaga, Spain (2018).

[9] NuCypher, Umbral: Threshold Proxy Re-encryption Scheme for Python*, versi 0.3.0, [Online]. Diakses melalui tautan berikut. https://pypi.org/project/umbral/

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, 1 Juni 2025

Haziq Abiyyu Mahdy (13521170)