Timed Release Cryptography Tanpa Agen Terpercaya Menggunakan Blockchain

Joshua Christo Randiny 13517063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517063@std.stei.itb.ac.id

***Abstrak*—*Timed Release Cryptography* adalah permasalahan yang sudah ada sejak lama. Solusi yang ada sekarang memanfaatkan adanya pihak ketiga terpercaya. Hal tersebut menjadi sulit terutama di era modern ini yang penuh dengan serangan. Salah satu teknologi modern yang terlihat mampu untuk menyelesaikan masalah ini adalah *blockchain* terutama Ethereum. Dengan kemampuan untuk menjalankan kode program melalui *smart contract* maka sangat mungkin untuk dibuat program *timed release cryptography* pada *blockchain* Ethereum. Pada makalah ini akan dibuat suatu *proof of concept* untuk sistem tersebut dan akan dilihat kelayakannya.**

Keywords—Timed release cryptography, Shamir secret sharing, blockchain, smart contract

I. PENDAHULUAN

Di dunia ini ada banyak persoalan yang membutuhkan sistem penguncian berbasis waktu atau *timed release cryptography*. Yang dimaksud di sini adalah kemampuan suatu sistem penguncian atau enkripsi untuk hanya dapat dibuka setelah sekian waktu dilewati. Kegunaan dari sistem tersebut sangatlah beragam seperti misalkan pada sebuah sistem ujian elektronik. Soal ujian dapat disebarkan diunduh dahulu oleh peserta ujian tapi tidak dapat dibuka sebelum waktu ujian dimulai.

Sebelumnya sudah ada banyak usaha untuk mengimplementasi sistem *timelock* tersebut. Salah satunya adalah oleh Ronald Rivest, Adi Shamir, dan David Wagner pada tahun 1996 [1]. Mereka berusaha mengimplementasikan sistem tersebut dengan menggunakan *timelock puzzle* atau agen terpecaya. Implementasi menggunakan *timelock puzzle* punya beberapa kekurangan yaitu pada sulitnya memprediksi kecepatan komputer di masa depan. Selain itu, *timelock puzzle* mengharuskan siapa pun yang ingin memecahkannya untuk membuang-buang sumber daya komputasi secara kontinu selama waktu yang ditentukan.

Pendekatan lain selain menggunakan *timelock puzzle* adalah dengan menggunakan agen terpercaya. Agen tersebut yang kemudian akan memegang kunci dekripsi dari sebuah data. Akan tetapi pendekatan ini punya kelemahan fatal yaitu harusnya ada kepercayaan yang diberikan kepada agen terpercaya tersebut. Namun, dengan menggunakan agen terpercaya, pihak pembuka tidak butuh menghabiskan banyak sumber daya untuk memecahkan persoalan algoritma secara terus menerus. Hal tersebut menjadikan pendekatan ini jauh lebih praktis untuk diimplementasikan.

Maju beberapa tahun ke depan, konsep *blockchain* mulai diperkenalkan. Salah satu kelebihan dari *blockchain* adalah kemampuan untuk mempunyai lingkungan tanpa kepercayaan. Konsep tersebut terlihat sangat cocok untuk digabungkan dengan *timed release cryptography* ini. Selain itu, salah satu *blockchain* yang juga mempunyai kemampuan unik untuk menjalankan program adalah Ehereum. Ethereum dapat digambarkan sebagai sebuah komputer bersama dan dapat menjalankan program yang dinamai *smart contract*. Dengan teknologi tersebut maka implementasi praktis dari *timed release cryptography* tanpa agen terpercaya bisa direalisasikan.

II. DASAR TEORI

A. Timed Release Cryptography

Seperti yang sudah dijelaskan singkat sebelumnya, *timed release cryptography* adalah suatu cara untuk mengenkripsi pesan yang hanya dapat didekripsi setelah waktu yang telah ditentukan di awal berlalu. Dua pendekatan yang dapat dilakukan untuk mencapainya adalah pendekatan *timelock puzzle* dan agen terpercaya [1].

Timelock puzzle bekerja dengan membuat suatu persoalan matematis yang sangat sulit sehingga jika sebuah komputer ditugaskan untuk menyelesaikannya, butuh waktu minimal sesuai yang diinginkan. Persoalan yang digunakan harus tidak dapat diparalelisasi sehingga banyak komputer tidak dapat menyelesaikan persoalan dengan lebih cepat. Hal tersebut dicapai dengan persoalan yang bersifat sekuensial sehingga tidak dapat diparalelisasi.

Kekurangan fatal dari sistem ini adalah kebutuhan untuk sebuah komputer yang ingin memecahkannya untuk melakukan komputasi selama waktu yang diinginkan. Jika persoalan diatur untuk dapat dibuka setelah lima hari tapi komputer pemecah tidak melakukan apa pun maka setelah lima hari persoalan tidak akan selesai secara sendirinya. Kekurangan lain adalah sulitnya menyesuaikan waktu CPU dan waktu dunia nyata. Dengan sulitnya memprediksi kecepatan komputasi di masa yang akan datang maka akan sulit untuk membuat *timelock puzzle* yang dapat tetap akurat untuk jangka waktu yang lama.

Untuk metode *timelock puzzle* salah satu algoritma yang dapat

digunakan adalah dengan melakukan perpangkatan angka bebas (a) sebanyak dua pangkat t kali dalam modulus n. Dengan n adalah nilai modulus yang dihitung sama dengan perhitungan n pada algoritma RSA yaitu perkalian dua prima besar.

$$\text{solusi} = a^{2^t} \pmod{n}$$

Sampai sekarang belum ada cara untuk mencari angka hasil dengan lebih cepat selain melakukan perhitungan perpangkatan tersebut. Percepatan hanya bisa dilakukan jika faktor dari n diketahui. Akan tetapi faktor dari n hampir tidak mungkin untuk dicari sama seperti n pada algoritma RSA.

Selain *timelock puzzle*, pendekatan lain yang mungkin adalah dengan penggunaan agen terpercaya. Setiap agen akan memegang kunci masing-masing dan dapat dimintai pada suatu waktu untuk memberikan kuncinya. Kunci tersebut dapat berupa kunci kriptografi utuh atau bagian dari kunci menggunakan teknik *secret sharing* agar lebih aman. Jika belum mencapai waktu yang ditentukan maka agen harus tidak memberikan kuncinya. Kondisi inilah yang menyebabkan kepercayaan pada agen menjadi penting, jika agen tidak bisa dipercaya maka bisa saja agen tersebut memberikan kunci untuk masa depan.

Walaupun begitu, pendekatan ini sangatlah baik dalam hal komputasi dan akurasi karena tidak terpengaruh terhadap kekuatan komputasi penerima dan juga waktu *release* dari kunci dapat diatur dengan sangat tepat tanpa bergantung pada kekuatan komputasi.

B. Shamir Secret Sharing

Skema pembagian data rahasia atau *secret sharing* adalah metode untuk membagikan data rahasia ke beberapa pihak sedemikian rupa sehingga rahasia asli tidak dapat direkonstruksi tanpa ada minimal x pihak yang memberikan bagiannya. Salah satu metode yang paling terkenal untuk *secret sharing* adalah yang ditemukan oleh Shamir pada tahun 1979 [3].

Skema yang digunakan Shamir untuk implementasi *secret sharing* adalah persoalan interpolasi. Untuk memecahkan suatu polinom dengan derajat n maka dibutuhkan minimal (n+1) titik data. Jika jumlah titik kurang maka penyelesaian persoalan akan menjadi sangat sulit mendekati mustahil untuk angka yang besar.

Dalam skema Shamir terdapat dua tahap penting yaitu tahap pembagian *share* dan rekonstruksi *secret*. *Share* yang dimaksud adalah pecahan dari *secret* awal yang akan dibagikan ke tiap pihak. Polinom yang akan dipakai untuk skema ini adalah polinom berderajat *t*-1. Berikut adalah langkah yang dilakukan pada skema Shamir.

1. Pembagian *share*

   Untuk memulai pertama akan dipilih suatu bilangan prima *p* yang lebih besar dari semua kemungkinan nilai pesan *M*. Kemudian akan dipilih *t*-1 (dengan *t* adalah nilai *threshold* atau ambang) bilangan bulat acak s_1, s_2, ... s_{t-1}. Kemudian akan dibentuk polinom seperti berikut

   $$s(x) \equiv M + s_1 x + s_2 x^2 + \ldots + s_{t-1} x^{t-1} \pmod{p}$$

   Dari polinom tersebut dapat dibuat titik sebanyak-banyaknya sesuai yang diinginkan dengan pertama membangkitkan *w* (jumlah pihak) buah bilangan bulat dalam modulus *p* yang akan digunakan sebagai nilai X dari titik (x_1, x_2, dan seterusnya). Pastikan bahwa jumlah pihak *w* harus lebih besar sama dengan dari nilai ambang *t* agar *secret* dapat direkonstruksi di akhir. Untuk nilai Y dari titik (y_1, y_2, dan seterusnya) dapat dihitung dengan memasukkan nilai X ke dalam persamaan polinom.

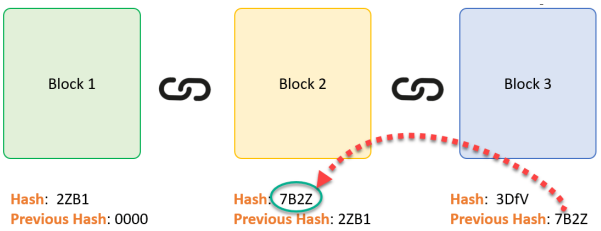
   $$y_i \equiv s(x_i) \pmod{p}$$

2. Rekonstruksi *secret*

   Untuk melakukan rekonstruksi *secret* dari titik-titik maka untuk setiap titik dapat dijadikan suatu persamaan dengan melakukan substitusi titik ke persamaan polinom yang ada. Maka akan ada persamaan sebanyak titik yang ada. Seperti yang diketahui, untuk melakukan interpolasi suatu polinom dengan derajat *t*-1 maka dibutuhkan minimal t persamaan. Setelah didapat *t* persamaan maka persamaan dapat diselesaikan dengan metode apa pun. Salah satu metode yang dapat dilakukan adalah metode eliminasi Gauss-Jordan.

C. Blockchain

Seperti pada namanya *blockchain* adalah serangkaian *block* yang disusun sebagai suatu rangkaian. Rangkaian tersebut saling sambung menyambung dengan mengandalkan referensi *hash* dari *block* sebelumnya seperti yang sudah dijelaskan di atas. Jika ada suatu blok yang ingin diganti datanya maka seluruh rantai akan otomatis terputus karena nilai *hash* akan berubah. Fakta tersebut membantu menjamin bahwa data yang sudah ada pada *blockchain* hampir mustahil untuk diubah oleh sembarang pihak. Satu-satunya cara jika sembarang pihak ingin mengubah data yang sudah disimpan pada *blockchain* adalah dengan juga ikut mengubah semua *block* setelahnya. Hal tersebut sangatlah sulit karena dalam pembuatan *block* baru tidak dapat secara sembarang dilakukan. Ada kondisi yang harus terpenuhi untuk pembuatan sebuah *block* baru, kondisi tersebut akan dibahas pada bagian lain.



Gambar 1. Ilustrasi *blockchain*
Sumber: https://www.guru99.com/blockchain-tutorial.html

Blockchain sendiri dapat dibagi menjadi 3 jenis berdasarkan aksesnya yaitu *public*, *private*, dan *consortium* [4]. Sesuai namanya, *public blockchain* dapat diakses oleh publik dan anggotanya boleh siapa saja tanpa batasan. Pada sisi lainnya ada *private blockchain* yang aksesnya dibatasi ke satu organisasi

terpusat. Di tengah-tengahnya ada *consortium blockchain* yang aksesnya dipegang oleh lebih dari satu organisasi terpisah. Keuntungan dari bentuk tersebut adalah kecepatan yang tinggi dan biaya rendah. Jika jumlah organisasi cukup banyak maka keamanan juga masih dapat terjamin. Dari ketiga jenis tersebut *private blockchain* merupakan yang paling tidak berguna karena dapat dengan mudah digantikan dengan sistem terdistribusi atau terpusat tradisional yang tentu akan lebih simpel. Untuk kedua jenis lainnya berguna tergantung pada jenis transaksi yang dilakukan. Jika transaksi ditujukan untuk masyarakat umum seperti pembayaran maka cocok untuk menggunakan *public blockchain*. Untuk kasus yang penggunaanya bukan ditujukan untuk semua orang, maka lebih cocok untuk menggunakan sistem *consortium blockchain*.

D. Ethereum

Ethereum adalah salah satu sistem *blockchain* yang tujuan utamanya bukanlah untuk melakukan transaksi jual beli saja. Tujuan dari Ethereum adalah bertindak sebagai sebuah platform *turing complete* untuk menjalankan sekumpulan kode yang dinamai *smart contract* [1]. Jika dilihat secara umum maka sebuah jaringan Ethereum dapat dianggap sebuah komputer besar lambat yang terdistribusi dan terpercaya. Setiap transaksi dapat dianggap sebuah fungsi transisi antara *state*. Oleh sebabnya kadang Ethereum juga disebut sebagai Ethereum Virtual Machine (EVM). *State* yang disimpan dapat berupa apa saja termasuk *state* untuk sebuah sistem *timed lock cryptography*.

Untuk algoritma konsensus, Ethereum menggunakan sistem *proof of work*. Secara singkat algoritma *proof of work* bekerja dengan memanfaatkan properti dari fungsi *hash* yang tidak dapat dicari *inverse*-nya. Oleh karena itu untuk mendapatkan suatu nilai *hash* tertentu maka harus dicari secara *bruteforce*. Untuk memvalidasi suatu *block* maka harus dicari sebuah angka yang jika ditambahkan ke *block* akan menghasilkan nilai hash tertentu (misalkan yang dimulai dengan angka 0 sejumlah X). Ethereum sendiri menggunakan fungsi *hash* yang didasarkan dari algoritma Keccak dengan modifikasi.

Ethereum juga memilih untuk membuat algoritma yang berat pada *bandwith* IO dan bukan pada tenaga CPU. Dengan begitu maka penggunaan perangkat keras ASIC khusus dapat dihindari [5]. Hal ini penting agar tidak ada pihak yang dapat mendominasi sistem dan menghilangkan jaminan kepercayaan sistem. Pihak yang ingin menyerang sistem butuh menguasai minimal 51 persen dari semua sumber daya komputasi Ethereum, hal yang hampir mustahil untuk dilakukan.

III. Rancangan Solusi Dan Implementasi

Untuk menyelesaikan masalah *timed release cryptography* ini akan digunakan sistem dengan pendekatan agen eksternal. Yang unik dan baru dari pendekatan yang akan dibuat adalah fakta bahwa agen tersebut tidaklah harus dapat dipercaya dalam arti pengirim dan penerima data mengetahui siapa agennya. Hal tersebut dapat terjadi dengan memanfaatkan sistem *blockchain*. Sebelumnya, berikut adalah masalah yang muncul dari penggunaan agen tidak terpercaya.

1. Agen dapat membocorkan kunci sebelum waktu yang telah ditentukan
2. Agen dapat tidak memberikan kunci walaupun waktu sudah lewat
3. Memastikan keamanan agen untuk tidak diretas oleh pihak lain
4. Komunikasi aman antara agen, pengirim, dan penerima data

Dengan menggunakan *smart contract* pada sistem Ethereum keempat masalah ini dapat diselesaikan. *Smart contract* yang akan dibuat akan bertindak sebagai fasilitator komunikasi untuk menyelesaikan masalah empat. Berikut adalah data yang disimpan pada *smart contract* agar dapat melakukan tugasnya.

```
uint256 lastId;
uint256 amount;

struct LockReservation {
    uint256 openTime;
    uint256 payment;
    address requestor;
    uint256 processorRequired;
    address payable[] processor;
    bool valid;
    bool paid;
}

mapping(uint256 => LockReservation) reservationMap;
mapping(address => string) processorAddress;
```

Kode 1. *State* yang disimpan oleh *smart contract*

Seperti yang dapat dilihat pada potongan kode di atas (ditulis dalam bahasa Solodity), *smart contract* menyimpan daftar semua reservasi kunci yang tiap entrinya menyimpan waktu dibuka, imbalan bagi agen atau *processor*, alamat peminta, jumlah agen yang dibutuhkan dan daftar alamat untuk tiap agen. Selain itu juga ada *boolean* untuk menyimpan status validitas agen dan status pembayaran agen. Selain dari itu, *smart contract* juga akan menyimpan daftar alamat *endpoint* API dari setiap agen yang berpartisipasi.

Selain itu akan ada beberapa *event* yang dapat dipancarkan oleh *smart contract* yaitu sebagai berikut.

```
event ReservationCreated(
    uint256 reservationId,
    uint256 payment
);
event ReservationTaken(
    uint256 reservationId,
    string processorAddress,
    bool valid
);
event OpenRequest(
    uint256 reservationId,
    string destination
);
```

Kode 2. *Event* yang dapat dipancarkan oleh *smart contract*

Dari potongan kode di atas dapat dilihat bahwa ada tiga *event* utama yaitu pertama ReservationCreated yang akan dipancarkan saat *client* pertama membuat reservasi kunci. Kemudian ReservationTaken adalah *event* yang akan dipancarkan saat ada agen yang mendaftarkan diri sebagai agen untuk suatu reservasi kunci. Terakhir ada *event* untuk permintaan pembukaan kunci.

Untuk proses pembuatan reservasi kunci sistem akan melewati lima tahap utama sebagai berikut.

1. Pengguna meminta reservasi pembuatan kunci ke *blockchain*

   Pengguna akan memanggil program *client* sederhana pada komputer masing-masing yang akan memanggil fungsi dari *smart contract*. Pengguna dapat membuat reservasi kunci dengan memberikan lima parameter yaitu waktu dibuka, jumlah agen, pesan rahasia, bayaran untuk agen, dan *threshold* agen yang dibutuhkan untuk merekonstruksi pesan rahasia.

   Sistem kemudian akan memanggil fungsi *smart contract* dengan memberikan parameter waktu dibuka dan jumlah agen yang diinginkan. Fungsi tersebut hanya dapat dipanggil dengan menyertakan pembayaran yang kemudian nantinya akan dibagikan kepada agen yang melayani.

2. *Blockchain* memancarkan *event* tentang permintaan kunci baru

   Setelah menerima permintaan untuk reservasi kunci baru maka *smart contract* akan memancarkan *event* ReservationCreated yang dapat ditangkap oleh semua agen yang aktif.

3. Agen menerima *event* dan berebut untuk melayani permintaan tersebut

   Setiap agen yang ingin menerima pekerjaan tersebut dapat membalasnya dengan memanggil fungsi pada *smart contract* untuk mengambil pekerjaan tersebut. Semua agen akan berebut untuk mengambil pekerjaan tersebut dan jika jumlah agen sudah tercukupi maka permintaan akan ditolak. Setiap kali ada agen yang berhasil mendapat *slot* dari suatu reservasi maka *smart contract* akan mengirim *event* ReservationTaken yang dapat ditangkap *client*.

4. Pengguna diberikan *event* oleh *blockchain* jika permintaan terpenuhi

   Setelah *client* mendapat *event* ReservationTaken, maka alamat dari agen yang mendaftar akan dikirimkan ke *client*.

5. Pengguna mengirimkan data *secret* yang telah dipecah ke masing-masing agen

   Pesan rahasia akan dipecah menggunakan skema Shamir untuk memastikan kerahasiaannya dan dikirim potongannya ke masing-masing agen

Untuk proses pembukaan kunci maka tahap yang dilakukan adalah sebagai berikut

1. *Client* meminta ke *blockchain* untuk membuka kunci

   Kapan saja sebuah *client* dapat meminta kepada *blockchain* untuk membuka suatu kunci waktu. Permintaan ini dilakukan dengan memanggil fungsi pada *smart contract* dengan parameter berisi id dari kunci.

2. *Blockchain* memvalidasi permintaan lalu mengirim *event* jika valid

   Jika permintaan valid, artinya permintaan buka kunci dilakukan melewati waktu minimal, maka *smart contract* akan mengirimkan *event* OpenRequest ke semua agen untuk mengirimkan *share secret* masing-masing ke *client*. Pada tahap ini juga agen akan dibayar oleh sistem seusai dengan perjanjian yang telah dibuat.

3. *Client* merekonstruksi pesan rahasia

   Dengan potongan rahasia yang diterima maka *client* dapat merekonstruksi pesan rahasia asli. Dengan demikian maka selesailah rangkaian pembukaan kunci.

Semua implementasi *smart contract* dilakukan dalam bahasa Solidity dan untuk *server* API diimplementasi dengan bahasa TypeScript.

Dengan proses yang telah dirancang maka semua masalah yang telah disebutkan di awal dapat diselesaikan. Untuk masalah satu yaitu untuk pembocoran kunci, hal tersebut dapat dimitigasi dengan skema Shamir *secret sharing* yang digunakan. Dibutuhkan kerja sama antara banyak agen untuk dapat mencuri data rahasianya. Kerja sama antara banyak agen menjadi sulit untuk dilakukan karena sistem pemilihan agen yang tidak mempunyai prioritas sehingga tersebar secara *random*. Selain itu, untuk keamanan lebih, *client* dapat menyebarkan kunci ke sebanyak-banyaknya agen yang diinginkan. Skema Shamir juga menyelesaikan masalah ketiga karena peretasan harus terjadi pada agen dalam jumlah besar.

Untuk masalah kedua, hal tersebut diselesaikan dengan penggunaan *smart contract* dan *blockchain*. Setiap agen menerima sinyal pelepasan melalui *event* dari *smart contract* dan hal tersebut tidak dapat dimanipulasi. Untuk memanipulasi *blockchain* dibutuhkan penguasaan sebanyak 51 persen dari sumber daya komputasi total *blockchain*. Hal tersebut merupakan hal yang mustahil untuk sistem besar seperti Ethereum.

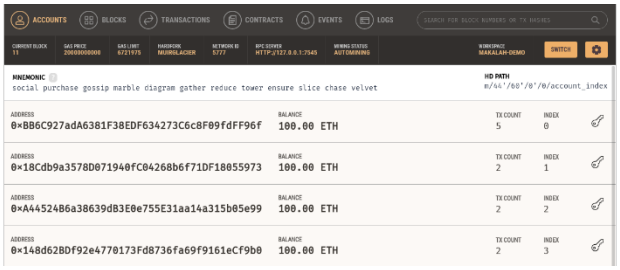
Untuk masalah keempat, penyelesaian juga didapat dari penggunaan *smart contract* yang akan mengatur segala proses komunikasi antara *client* dan agen agar terjadi secara terstandar dan aman. Sistem pembayaran juga dilakukan dengan *smart contract* agar semua pihak yakin diutungungkan (agen dibayar dan *client* memberikan jaminan uang untuk penguncian datanya).

IV. Eksperimen

Untuk eksperimen ini akan dilakukan pada jaringan lokal dengan bantuan kakas Ganache. Konfigurasi yang digunakan adalah 3 agen dengan 1 *client*. Untuk biaya transaksi akan digunakan angka bulat yaitu 1 ETH atau 10^{18} Wei. Untuk waktu pembukaan akan diatur sekitar 10 menit di masa depan. Untuk alamat akun dari tiap partisipan adalah sebagai berikut

Tabel 1. Alamat akun dari tiap partisipan

Partisipan	Alamat Akun
Client	0xBB6C927adA6381F38EDF634273C6c8F09fdFF96f
Agen 1	0x18Cdb9a3578D071940fC04268b6f71DF18055973
Agen 2	0xA44524B6a38639dB3E0e755E31aa14a315b05e99
Agen 3	0x148d62BDf92e4770173Fd8736fa69f9161eCf9b0



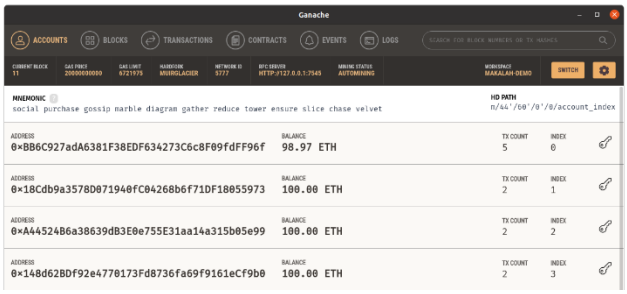
Gambar 2. Status akun pada kondisi awal, semua mempunyai 100 ETH

Setelah itu pada *client* akan dipanggil API untuk membuat reservasi kunci dengan parameter berikut

```
{
    "timestamp": 1608474600,
    "processor": 3,
    "secret": "ini secret super rahasia",
    "threshold": 2,
    "payment": 1000000000000000000
}
```

Kode 3. Permintaan untuk mengunci suatu rahasia untuk sekian waktu di masa depan

Dapat dilihat bahwa *client* meminta *share* disebarkan ke tiga agen tapi dengan nilai ambang dua. Artinya jika ada satu agen yang rusak maka tidak menjadi masalah dalam rekonstruksi data. Setelah proses tersebut, maka saldo dari alamat akun *client* akan berkurang sejumlah kurang lebih 1 ETH. Dapat dilihat pada gambar berikut bahwa saldo sudah berkurang tapi saldo dari agen tidak bertambah karena agen belum menyelesaikan tugasnya. Saldo akan sementara disimpan pada akun *smart contract*.



Gambar 3. Saldo setelah melakukan reservasi kunci

Setelah itu yang akan dilakukan adalah proses pemecahan pesan rahasia dan mengirimkannya ke tiap agen yang berpartisipasi. Berikut adalah nilai (dalam hex) untuk tiap *share* dan aslinya.

Tabel 2. Data Rahasia dan Representasinya di Tiap Agen

Data Asli
696e692073656372657420737570657220726861736961
Agen 1
08035b8404d9106108983633d6a5f1b28da0bd9d7d82e3619c35b9ce52d48de48c79a6ac51864341596b07e28b30fbecf8fa5a0c25720030e531f6fc49d943792dd2
Agen 2
080299f8f365ebb5fb1b24226fc655dcf634d6c956db496fe8f225ac974cf696034ec4e6952c89a56e61f166f2cb596f5b8f6c26cd7d00034d0fa4438542897436bb
Agen 3
0801c27cf7bcfbd4f3831211b963a46e7b956b352b30aa7d74a69c0ac5f97b008f176238c4cfca94377ff6f779dba2f7a3103658e86c0056a84d529fccf2ca631b00

Jika dilihat *file* yang terdapat pada tiap agen tidak sama dengan pesan rahasia asli. Hal ini membuktikan bahwa jika ada peretasan, rahasia tetaplah aman. Selain itu juga dapat terlihat bahwa potongan *share* ukurannya lebih besar dari pesan asli. Hal ini menunjukkan salah satu kelemahan yaitu ukuran *share* yang relatif besar. Untuk itu, lebih baik yang disimpan pada sistem hanyalah kunci simetris untuk suatu *file* terenkripsi dan bukan *file* itu sendiri.

Berikutnya juga akan dilihat respons dari *smart contract* jika permintaan pembukaan diberikan sebelum waktu yang telah ditentukan.

```
{
    "data": {
        "0x1905dde27b6d6e015fb2fe833ef79ee5b9c0abd425b1adecf854ed202701ab97": {
            "error": "revert",
            "program_counter": 1106,
            "return": "0x08c379a0000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000002f556e6c6f636b2074696d657374616d70206d757374206265206c657373206f7220657175616c207468616e206e6f77000000000000000000000000000000000000000000000000",
            "reason": "Unlock timestamp must be less or equal than now"
        },
        "stack": "RuntimeError: VM Exception while processing transaction: revert Unlock timestamp must be less or equal than now\n    at Function.RuntimeError.fromResults (/tmp/.mount_ganach7ufgnC/resources/static/node/node_modules/ganache-core/lib/utils/runtimeerror.js:94:13)\n    at BlockchainDouble.processBlock (/tmp/.mount_ganach7ufgnC/resources/static/node/node_modules/ganache-core/lib/blockchain_double.js:627:24)\n    at processTicksAndRejections (internal/process/task_queues.js:93:5)",
        "name": "RuntimeError"
    }
}
```

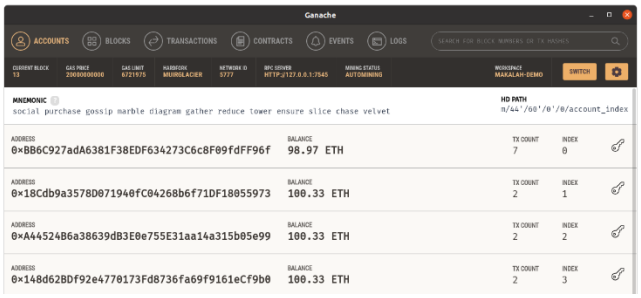
Kode 4. Balasan dari *smart contract* jika dicoba untuk membuka kunci sebelum waktunya

Dapat dilihat bahwa *smart contract* akan mengeluarkan pesan kesalahan “Unlock timestamp must be less or equal than now” yang berarti pesan tidak dapat dibuka sampai waktu yang ditentukan telah dilewati. Jika permintaan dilakukan setelah waktu yang ditentukan maka kembalian dari pemanggilan API nya adalah sebagai berikut

```
{
    "result": "ini secret super rahasia"
}
```

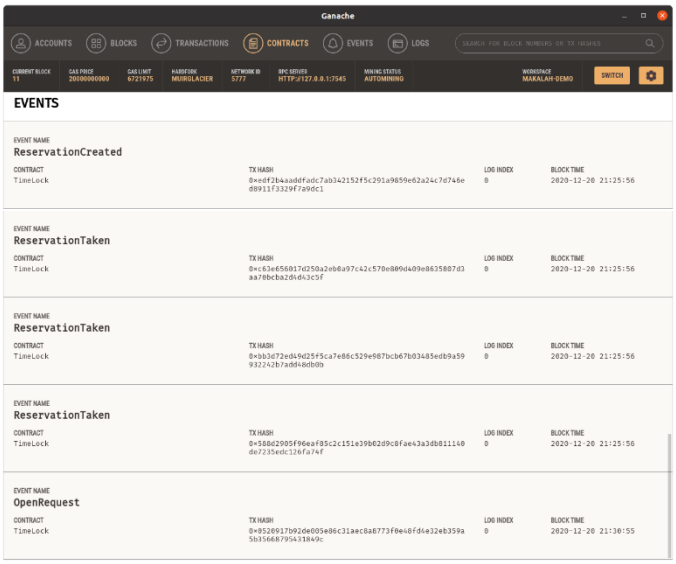
Kode 5. Balasan dari *smart contract* jika pesan dicoba untuk dibuka setelah waktu yang ditentukan

Jika pesan dibuka setelah waktu yang ditentukan maka pekerjaan agen telah terpenuhi dan agen akan dibayar. Tiap agen akan diberikan pembagian rata dari biaya tiap transaksi yang dibayarkan di awal setelah dikurangi biaya layanan.



Gambar 4. Saldo setelah proses pembukaan sukses

Dapat dilihat juga *log* untuk *event* dari *smart contract* pada antarmuka Ganache. Berikut adalah tangkapan layar untuk *log event* yang terjadi.



Gambar 5. Daftar *event* yang terjadi selama satu proses penguncian pesan rahasia

V. Analisis Keamanan

Keamanan dari sistem yang dibuat bergantung pada dua asumsi keamanan yaitu.

- Jam pada *blockchain* diasumsikan akurat
- Tidak ada pihak yang dapat menguasai lebih dari 51 persen kekuatan komputasi *blockchain*

Untuk asumsi pertama, hal tersebut merupakan asumsi yang valid sesuai dengan desain dari *blockchain* itu sendiri. Pada jaringan Ethereum ada batas maksimal 900 detik untuk toleransi *skew* waktu pada tiap *block*-nya. Angka tersebut terlihat besar namun angka tersebut hanya merupakan angka teoritis maksimal yang diperbolehkan sistem. Pada kenyataannya *skew* dari jam hampir tidak ada. Bahkan pada implementasi nyatanya *skew* waktu lebih dari 15 akan menyebabkan penolakan *block* [6].

Untuk asumsi kedua, pada saat tulisan ini dibuat, nilai *hash rate* pada jaringan Ethereum mencapai angka 286 Terahash/detik [7]. Sebagai referensi, satu AMD R9 hanya mempunyai *hash rate* sebesar 46 Megahash/detik. Dibutuhkan uang yang sangat besar untuk menyerang jaringan sebesar itu dan sampai sekarang jaringan Ethereum dan *blockchain* besar lainnya belum pernah mengalami serangan 51 persen. Ditambah lagi ada rencana di masa depan untuk mengubah algoritma konsensus yang digunakan untuk lebih tahan serangan lagi.

VI. Kesimpulan dan Saran

Timed release cryptography merupakan suatu sistem untuk mengunci data yang hanya dapat dibuka setelah sekian waktu terlewati. Pada tradisionalnya hal ini hanya dapat terjadi dengan menggunakan agen terpercaya. Namun, dengan adanya teknologi *blockchain* maka hal tersebut sudah terbukti dapat dilakukan dengan agen yang tidak dipercaya sepenuhnya. Semua kemungkinan masalah yang sudah dijelaskan sebelumnya dapat diselesaikan dengan penggunaan *blockchain*. Selain itu analisis keamanan menyatakan bahwa sistem cukup aman untuk digunakan.

Untuk pengembangan ke depannya dapat dilakukan protokol delegasi *share* ke agen lain untuk agen yang ingin mengundurkan diri dari sistem. Selain itu juga dapat dikembangkan sistem untuk melakukan verifikasi versi dan modifikasi yang dilakukan pada perangkat lunak agen dan *client*. Tujuannya agar keamanan dapat lebih terjamin.

VII. Ucapan Terima Kasih

Pertama-tama penulis mengucapkan terima kasih kepada Tuhan YME atas kasih dan berkat-Nya sehingga penulis dapat menyelesaikan makalah kriptografi ini tepat waktu. Penulis juga berterima kasih kepada Bapak Rinaldi Munir atas bimbingannya selama ini pada mata kuliah kriptografi. Tidak lupa penulis ucapkan terima kasih kepada teman-teman yang sudah memberikan semangat dalam penulisan makalah ini.

Referensi

[1] Rivest, R. L., Shamir, A., & Wagner, D. A. (1996). Time-lock puzzles and timed-release crypto

[2] Ethereum Whitepaper. (t.t.). Ethereum.Org. Diambil 18 Oktober 2020, dari https://ethereum.org

[3] Shamir, A. (1979). How to share a secret. Communications Of The ACM, 22(11), 612-613. doi: 10.1145/359168.359176

[4] Buterin, V. (2015). On Public and Private Blockchains | Ethereum Foundation Blog. Diambil 18 Oktober 2020, dari https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/
[5] Ethash Design Rationale. (t.t.). Ethereum Wiki. Diambil 18 Oktober 2020, dari https://eth.wiki/concepts/ethash/design-rationale
[6] Kode sumber Go-Ethereum. Diambil 21 Desember 2020, dari https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/consensus.go#L45
[7] Hashrate Ehtereum. Diambil 21 Desember 2020, dari https://etherscan.io/chart/hashrate

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.

Jakarta, 21 Desember 2020

Joshua Christo Randiny
13517063