Implementasi *Hashed Time Locked Contract* (HTLC) pada *blockchain* Berbasis EVM sebagai Alternatif *Third Party Middleman*

Fakhri Putra Mahardika (18221080)
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): fakhriputramahardika@gmail.com

***Abstract*—Seiring Dalam era digital saat ini, permintaan akan perantara yang dapat dipercaya dalam transaksi antar pengguna semakin meningkat. Namun, kehadiran perantara sering kali membawa biaya tambahan dan risiko potensial, seperti keterlambatan transaksi dan kerentanan terhadap serangan. Untuk mengatasi tantangan ini, teknologi blockchain menawarkan solusi alternatif, salah satunya adalah Hash Time Locked Contract(HTLC). HTLC adalah Smart Contract yang memungkinkan transaksi aman dan terdesentralisasi antara dua pihak tanpa memerlukan perantara. Kontrak ini beroperasi dengan mengunci aset menggunakan hash dari nilai rahasia dan membatasi waktu penyelesaian transaksi. Implementasi HTLC pada blockchain berbasis Mesin Virtual Ethereum (EVM) memberikan solusi yang efisien dan terintegrasi dengan jaringan yang ada, memungkinkan transaksi lintas jaringan dengan lebih aman dan efisien. Namun, tantangan seperti skalabilitas, efisiensi, serta sinkronisasi waktu dan keamanan blockchain tetap perlu diatasi. Penelitian ini mengusulkan implementasi HTLC pada blockchain berbasis EVM sebagai alternatif untuk mengeliminasi kebutuhan akan perantara pihak ketiga, dengan mengeksplorasi keuntungan dan tantangan dari pendekatan ini.**

Kata kunci: Blockchain, Smart Contract, Ethereum, Ethereum Virtual Machine (EVM), Hashed Time Locked Contract (HTLC), Keamanan Data, Transaksi Peer-to-Peer.

I. PENDAHULUAN

Dalam ekosistem internet tradisional, transaksi antar pengguna sering kali membutuhkan perantara pihak ketiga (*third party middleman)* untuk menjamin keamanan dan kepercayaan antar pihak yang terlibat. Meskipun penggunaan perantara dapat memberikan jaminan keamanan, hal ini juga memperkenalkan biaya tambahan dan potensi risiko lainnya, seperti keterlambatan transaksi dan kerentanan terhadap serangan.

Seiring dengan berkembangnya teknologi, solusi alternatif untuk menghilangkan kebutuhan akan pihak ketiga terus dieksplorasi. Salah satu teknologi yang muncul sebagai solusi yang berpotensi adalah Hashed Time Locked Contract (HTLC). HTLC adalah sebuah kontrak pintar (*smart contract)* yang memungkinkan dua pihak untuk melakukan transaksi dengan aman dan terdesentralisasi, tanpa perlu mempercayai pihak ketiga. Kontrak ini bekerja dengan mengunci aset menggunakan hash dari suatu nilai rahasia dan membatasi waktu penyelesaian transaksi. Apabila penerima dapat menyediakan nilai rahasia yang tepat dalam jangka waktu yang ditentukan, maka aset akan dikirimkan; sebaliknya, aset akan dikembalikan kepada pengirim.

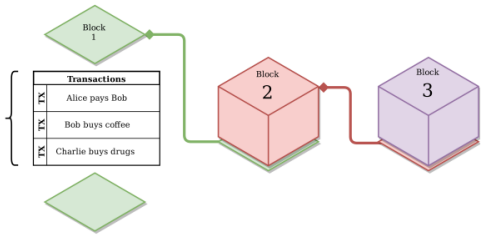
Implementasi HTLC dapat dilakukan pada blockchain berbasis *Ethereum Virtual Machine* (EVM). adalah salah satu platform paling populer dan banyak digunakan dalam pengembangan aplikasi desentralisasi (dApps). Dengan menggunakan EVM, HTLC dapat diimplementasikan dengan mudah dan terintegrasi dengan jaringan yang sudah ada, seperti Ethereum, Binance Smart Chain, dan lain-lain. Hal ini memungkinkan pengguna untuk melakukan transaksi lintas jaringan (*cross-chain transactions*) dengan lebih aman dan efisien

Meskipun HTLC menawarkan banyak keuntungan, masih ada tantangan yang harus dihadapi dalam implementasinya. Salah satunya adalah masalah skalabilitas dan efisiensi, karena eksekusi kontrak pintar pada blockchain dapat memerlukan biaya gas yang signifikan. Selain itu, keberhasilan HTLC juga bergantung pada sinkronisasi waktu dan keamanan *blockchain* yang digunakan.

Dalam makalah ini, akan dibahas konsep dan mekanisme HTLC, serta mengusulkan implementasi HTLC pada blockchain berbasis EVM sebagai alternatif untuk mengeliminasi kebutuhan akan pihak ketiga. Kami akan mengeksplorasi keuntungan dan tantangan dari pendekatan ini, serta memberikan studi kasus untuk menggambarkan aplikasi praktis HTLC dalam ekosistem blockchain. Diharapkan, penelitian ini dapat memberikan kontribusi signifikan dalam pemanfaatan teknologi *blockchain*.

II. DASAR TEORI

A. *Blockchain*



Gambar 1 Visualisasi Blockchain

Blockchain adalah sebuah buku besar publik, atau lebih mudahnya sebuah *database*, dari semua transaksi yang tersimpan dalam sebuah jaringan *node* yang terdesentralisasi. *Database* ini disebut *blockchain* karena transaksi yang termasuk dalam *database* disimpan dalam blok-blok yang berurutan secara linier yang mana setiap blok memiliki referensi ke blok sebelumnya seperti yang dapat dilihat pada gambar 2.X. Untuk membuat blok baru, *node* mengumpulkan sekumpulan transaksi yang tidak termasuk dalam blok sebelumnya dan menghitung nilai *hash* untuk memecahkan teka-teki kriptografi yang memakan sumberdaya yang besar. Proses ini disebut dengan mining. Ketika sebuah blok baru telah dibuat, *node* yang pertama kali dapat menambang dan mendistribusikan blok baru tersebut ke dalam jaringan akan mendapatkan hadiah berupa koin baru. Biaya harus dibayarkan untuk setiap transaksi dan ini ditransfer ke penambang blok ketika transaksi tersebut dimasukkan ke dalam blok baru. Biaya ini dan imbalan yang didapatkan dari menambang blok baru memberikan insentif tambahan untuk penambang mata uang digital untuk melanjutkan aktivitas mereka.

B. *Smart Contract*

Smart Contract, seperti yang diuraikan oleh Szabo, adalah sebuah kontrak yang dieksekusi secara elektronik dengan aturan, persyaratan, dan eksekusi yang dijelaskan dalam kode. Szabo membuat perbandingan dengan mesin penjual otomatis. Seorang klien memasukkan koin dan menerima barang sesuai dengan logika yang diprogram oleh mesin penjual otomatis. Smart contract sering kali menjadi bagian integral dari logika *blockchain* dan memungkinkan pemrograman yang melibatkan mata uang, tanpa keterlibatan pihak ketiga yang terpusat. Fungsionalitas *smart contract* sangat berbeda tergantung pada *blockchain* yang digunakan. ETH(EVM) menawarkan pemrograman smart contract dalam bahasa *turing complete* yang disebut Solidity, sedangkan BTC menawarkan fungsionalitas yang jauh lebih rendah dalam bahasa *non-turing complete*, yaitu Bitcoin Script.

C. *Ethereum Virtual Machine (EVM)*

```
contract Token {
 mapping(address=>uint) balances;
 function deposit() payable {
  // (msg is a global representing
  //  the current call)
  balances[msg.sender] += msg.value;
 }

 function transfer(address recipient,
  uint amount) returns(bool success) {
   if (balances[msg.sender] >= amount) {
     balances[msg.sender] -= amount;
     balances[recipient] += amount;
     return true;
   }
   return false;
 }
}
```

```
JUMPDEST
PUSH F*40
CALLER
AND
PUSH 0
SWAP1
DUP2
MSTORE
...
PUSH 40
SWAP1
SHA3
DUP1
SLOAD
CALLVALUE
ADD
SWAP1
SSTORE
```

Gambar 2 Simple Solidity Smart Contract (Kiri) dan Hasil Compile EVM Function (Kanan)

. Gambar 2.2 menunjukkan contoh hasil *smart contract* Solidity yang di-*compile* ke EVM. Secara khusus, kutipan ini membaca saldo pengirim dari kontrak penyimpanan / variable *global*, menambahkan nilai yang dipanggil saat ini ke saldo kontrak ini (membuat token baru dengan imbalan Ether yang dikirim ke kontrak), dan menyimpan jumlah baru ini kembali ke entri yang relevan. Untuk mencegah program dieksekusi tanpa batas waktu, pengirim dari setiap transaksi membayar biaya kepada para penambang. (penambang adalah pengguna yang mengurutkan transaksi-transaksi ini ke dalam blok-blok dalam *blockchain*). Biaya ini dibebankan secara proporsional dengan jumlah gas yang digunakan yang digunakan oleh kontrak. Total biaya untuk eksekusi sepenuhnya sepenuhnya disepakati oleh jaringan, dan setiap transaksi menentukan jumlah maksimum gas(biaya) yang ingin digunakan, serta nilai tukar antara Ether dan gas. Jika saat transaksi sedang berjalan kehabisan gas selama eksekusi, transaksi dibatalkan, pembaruan state-nya dikembalikan, dan penambang tetap mendapat biaya gas transaksi. Ini menempatkan semua transaksi EVM, melakukan penghentian, dan memungkinkan jaringan untuk membebankan biaya transaksi sebanding dengan biaya komputasi yang mereka keluarkan.

D. *Third Party Middleman*

Perantara pihak ketiga tradisional memainkan peran penting dalam memfasilitasi rasa percaya antara pihak-pihak yang terlibat dalam transaksi atau interaksi. Mereka bertindak sebagai perantara, memberikan jaminan dan pengawasan untuk memastikan integritas dan keadilan pertukaran. Dalam konteks blockchain, di mana teknologi ini bertujuan untuk mendesentralisasi dan menghilangkan kebutuhan akan perantara tersebut, peran perantara pihak ketiga mengalami transformasi yang signifikan. Alih-alih bergantung pada otoritas atau institusi terpusat untuk verifikasi kepercayaan, blockchain menggantikannya dengan mekanisme konsensus terdistribusi dan protokol kriptografi. Akan tetapi, dalam skenario tertentu, perantara pihak ketiga mungkin masih ada dalam ekosistem blockchain, terutama dalam kasus dimana keahlian khusus, arbitrase, atau lapisan kepercayaan tambahan diperlukan di luar kemampuan jaringan terdesentralisasi. Perantara ini dapat berfungsi sebagai validator, arbiter, atau penyedia layanan, yang menawarkan layanan bernilai tambah

sambil menavigasi lanskap kepercayaan yang berkembang dalam sistem terdesentralisasi.

III. Implementasi

Pembuatan HTLC dilakukan melalui beberapa langkah. Dimulai dengan penyiapan lingkungan, lalu pembuatan *contract, contract deployment,* dan lalu penggunaan.

A. *Environment Setup*

Pengaturan dimulai dengan melakukan instalasi *Truffle* untuk membantu pengembangan *Smart Contract*.

```
$ npm install -g truffle
$ truffle init
```

Pastikan NVM(node version manager) sudah terpasang karena truffle memerlukan npm versi maksimum 18.X.Y sedangkan npm sekarang berada pada versi 20.X.Y

```
$ nvm install 18
$ nvm use 18
```

Selanjutnya pada file truffle-config.js dibuat menjadi seperti berikut. Dibutuhkan private key yang berisi *address* yang memiliki ETH/ *Balance* koin utama network yang cukup untuk melakukan deployment yaitu kira-kira sekitar 0.1ETH. Jangan lupa mengonfigurasi *deployment* untuk test network Sepolia dengan *API-Key* yang sesuai. Berikut merupakan isi file tersebut.

```
const MetaprivateKey =
['0x9c07d35226bc642575ae96ca41b514cefa1a57cc077d94018614a545523a5ca2'];
const HDWalletProvider = require('@truffle/hdwallet-provider');

const api_key = 'https://eth-sepolia.g.alchemy.com/v2/mlbXSJTgFXSeZvKvVnsaTIyPACpFlxhR';

module.exports = {
  networks: {
    sepolia: {
      provider: () => new HDWalletProvider(MetaprivateKey, api_key),
      networkCheckTimeout: 100000,
      gasLimit : 700000,
      timeoutBlocks: 200,
      network_id: 11155111,
      skipDryRun: true
    },
  },

  compilers: {
    solc: {
      version: "0.8.20",
    }
  },
};
```

B. *Pembuatan Contract*

Selanjutnya yaitu pembuatan *smart contract.* Terdapat 3 file yang dibutuhkan dalam tahap ini. Pertama HTLC.sol yang berisi *smart contract* dalam Bahasa Solidity.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract HTLC {
    address payable public sender;
    address payable public receiver;
    address public reviewer;
    uint256 public amount;
    uint256 public endTime;
    bool public isApproved = false;

    constructor(address payable _receiver, address _reviewer, uint256 _endTime) payable {
        require(_endTime > block.timestamp, "End time should be in the future");
        sender = payable(msg.sender);
        receiver = _receiver;
        reviewer = _reviewer;
        amount = msg.value;
        endTime = _endTime;
    }

    function approve() public {
        require(msg.sender == reviewer, "Only reviewer can approve");
        require(block.timestamp < endTime, "Cannot approve after end time");
        isApproved = true;
        receiver.transfer(amount);
    }

    function refund() public {
        require(block.timestamp >= endTime, "Too early to refund");
        require(!isApproved, "Already approved");
        sender.transfer(amount);
    }
}
```

Lalu, kedua HTLC_test.js untuk melakukan *testing terhadap smart contract.*

```
const HTLC = artifacts.require("HTLC");
```

```
contract("HTLC", (accounts) => {
  const [sender, receiver, reviewer] = accounts;

  it("should deploy and initialize correctly", async
() => {
    const endTime = Math.floor(Date.now() / 1000) +
60; // 1 minute from now
    const htlc = await HTLC.new(receiver, reviewer,
endTime, { from: sender, value:
web3.utils.toWei('1', 'ether') });

    assert.equal(await htlc.sender(), sender);
    assert.equal(await htlc.receiver(), receiver);
    assert.equal(await htlc.reviewer(), reviewer);
    assert.equal((await htlc.amount()).toString(),
web3.utils.toWei('1', 'ether'));
    assert.equal((await htlc.endTime()).toNumber(),
endTime);
  });

  it("should allow reviewer to approve the
transaction", async () => {
    const endTime = Math.floor(Date.now() / 1000) +
10800; // 3 hour from now
    const htlc = await HTLC.new(receiver, reviewer,
endTime, { from: sender, value:
web3.utils.toWei('1', 'ether') });

    await htlc.approve({ from: reviewer });

    const isApproved = await htlc.isApproved();
    assert.equal(isApproved, true, "The transaction
should be approved");

    const receiverBalance = await
web3.eth.getBalance(receiver);
    assert(receiverBalance > 0, "Receiver should
have received the funds");
  });

  it("should allow sender to refund after end time
if not approved", async () => {
    const endTime = Math.floor(Date.now() / 1000) +
5; // 5 seconds from now
    const htlc = await HTLC.new(receiver, reviewer,
endTime, { from: sender, value:
web3.utils.toWei('1', 'ether') });

    await new Promise(resolve => setTimeout(resolve,
6000)); // wait for 6 seconds

    await htlc.refund({ from: sender });

    const senderBalance = await
web3.eth.getBalance(sender);
    assert(senderBalance > 0, "Sender should have
received the refund");
  });
});
```

Ketiga yaitu 1_deploy_contract.js yang digunakan sebagai *entry file* yang menjadi *script* deployment.

```
const HTLC = artifacts.require("HTLC");

module.exports = function(deployer, network,
accounts) {
  const receiver =
"0x97956E6119Cb444c7D13b59A0048D7073576ed55";
  const reviewer =
"0xD13BF0Dc8c3adC2bFD9885572f13E9775F05E3a1";
  const endTime = Math.floor(Date.now() / 1000) +
10800; // 3 hour from now

  deployer.deploy(HTLC, receiver, reviewer, endTime,
{ from: "0xdA2685Ff4c390551899BB81Ad9f92EBb21488D22"
, value: web3.utils.toWei('0.1', 'ether') });
};
```

C. *Contract Deployment*

Selanjutnya melakukan deployment pada testnet Ethereum berbasis EVM yakni network Sepolia.

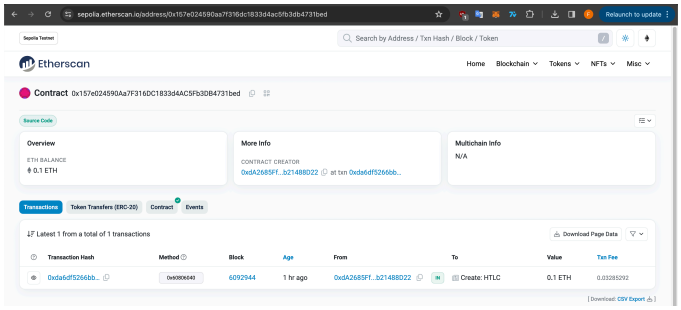
```
$ truffle migrate --network sepolia
```

Dapat dicek pada terminal dipastikan hingga contract berhasil di-deploy seperti pada gambar berikut

```
fakhripm@Fakhris-MacBook-Air truffle % truffle migrate --network sepolia

Compiling your contracts...
===========================
> Everything is up to date, there is nothing to compile.


Starting migrations...
======================
> Network name:    'sepolia'
> Network id:      11155111
> Block gas limit: 30000000 (0x1c9c380)


1_deploy_contract.js
====================

   Deploying 'HTLC'
   ----------------
   > transaction hash:    0xda6df5266bb312612a91aa4bf048e3219c0244b7594a8bc0080ec24c703091ac
   > Blocks: 1            Seconds: 8
   > contract address:    0x157e024590Aa7F316DC1833d4AC5Fb3DB4731bed
   > block number:        6092944
   > block timestamp:     1718202684
   > account:             0xdA2685Ff4c390551899BB81Ad9f92EBb21488D22
   > balance:             0.265926945277781504
   > gas used:            566897 (0x8a671)
   > gas price:           57.952196198 gwei
   > value sent:          0.1 ETH
   > total cost:          0.132852926168057606 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:     0.132852926168057606 ETH

Summary
=======
> Total deployments:   1
> Final cost:          0.132852926168057606 ETH
```

Gambar 3 Deployment Smart Contract HTLC.sol

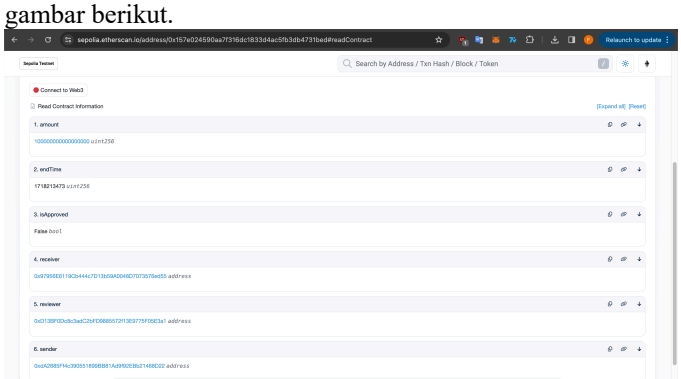
Lalu dapat dilihat dengan menggunakan *website* https://sepolia.etherscan.io/address/0x157e024590aa7f316dc1833d4ac5fb3db4731bed



Gambar 4 Deployment Contract Berhasil, Dilihat melalui etherscan

> Setelah melakukan verifikasi *smart contract* nantinya semua orang dapat melihat secara langsung kontrak yang sudah di-deploy tersebut dan dapat menggantikan *third party middleman*.

Pada Halaman Contract, dapat dilakukan *read and write* pada contract tersebut. Untuk bagian *read* dapat dilihat pada gambar berikut.

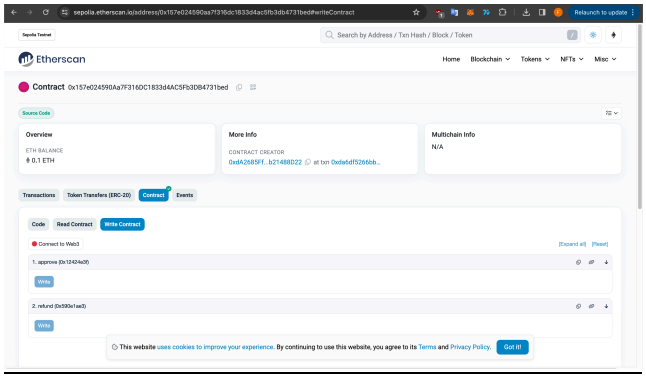


Gambar 5 Read Contract HTLC

Pada gambar tersebut, contract tersebut merupakan kontrak antara *sender* 0xdA2685Ff4c390551899BB81Ad9f92EBb21488D22 dan receiver 0x97956E6119Cb444c7D13b59A0048D7073576ed55. Terdapat juga reviewer yang merupakan opsional jika kedua belah pihak dirasa masih memerlukan pihak ketiga yang lebih transparan dan terakuntabilitas. 0xD13BF0Dc8c3adC2bFD9885572f13E9775F05E3a1.

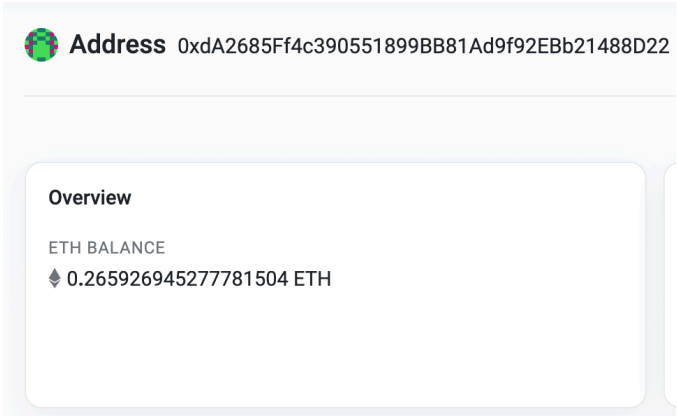
D. Penggunaan

Penggunaan HTLC dapat dilakukan melalui frontend third party seperti etherscan atau siapapun dapat membuat *front-endnya* sendiri. Dalam kasus ini cukup digunakan *tools metamask* yang digunakan untuk menyimpan *private key* dari *wallet* yang digunakan *sender* dan *receiver*. Dan *front-end* dari etherscan pada url



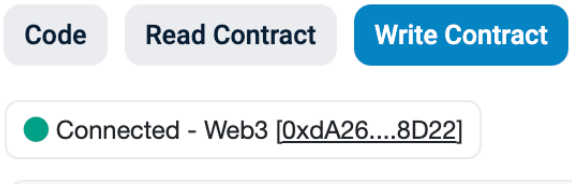
Gambar 6 Smart Contract pada etherscan

Akan dilakukan simulasi apabila pihak *receiver* tidak melakukan kewajiban pada *contract* sehingga *sender* akan request refund. Saat in saldo pada sender yakni 0.26 ETH



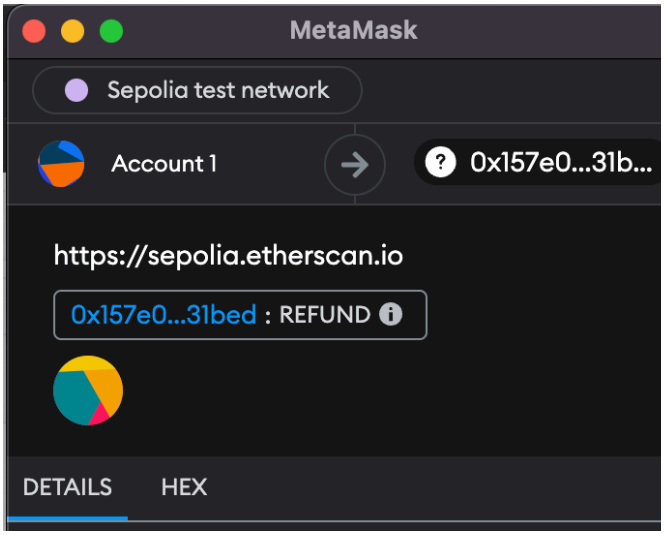
Gambar 7 Balance Sender

Lalu Menggunakan metamask, *sender* akan memanggil *method refund* karena pihak *receiver* tidak memenuhi kewajibannya hingga masa berakhir kontrak.



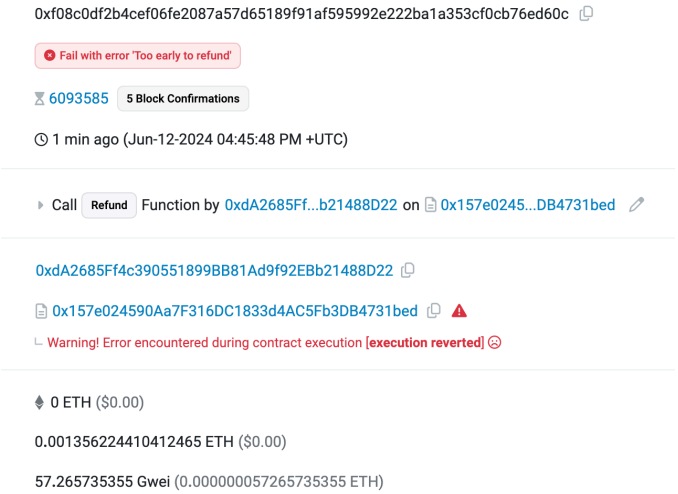
Gambar 8 Sender Terhubung ke Etherscan

Dapat dilihat bahwa bagian awal dan akhir *address* yang terhubung merupakan *sender*.



Gambar 9 Transaksi Sender

Transaksi Seharusnya akan gagal karena waktu dari *deployment* hingga sekarang belum mencapai 10800 detik (3jam) yang merupakan parameter *deployment*.



Gambar 10 Transaksi Gagal

Transaksi error dengan pesan "Too early to refund" Hal ini menandakan pihak *receiver* masih punya waktu untuk menjalankan kewajiibannya agar mendapat dana dari pihak *sender*.

IV. PENUTUP

Dalam makalah ini, telah dibahas konsep dan mekanisme HTLC serta implementasi pada blockchain berbasis EVM sebagai solusi alternatif untuk menghilangkan kebutuhan akan middleman pihak ketiga dalam transaksi. Melalui penggunaan kontrak pintar, HTLC memungkinkan transaksi yang aman dan terdesentralisasi antara dua pihak tanpa memerlukan intervensi pihak ketiga yang mempercayai. Implementasi pada blockchain berbasis EVM, seperti Ethereum, memungkinkan transaksi lintas jaringan dengan efisien dan terintegrasi dengan jaringan yang sudah ada. Meskipun demikian, tantangan seperti skalabilitas, efisiensi, dan keamanan tetap menjadi perhatian yang perlu diatasi dalam mengimplementasikan HTLC. Diharapkan penelitian ini dapat memberikan kontribusi signifikan dalam pemanfaatan teknologi blockchain dan meningkatkan keamanan serta efisiensi dalam transaksi digital di masa depan.

Link github
https://github.com/Fakhripm/KripToko

References

[1] Yu, B., Kermanshahi, S. K., Sakzad, A., & Nepal, S. (2019). Chameleon hash time-lock contract for privacy preserving payment channel networks. In *Provable Security: 13th International Conference, ProvSec 2019, Cairns, QLD, Australia, October 1–4, 2019, Proceedings 13* (pp. 303-318). Springer International Publishing.

[2] Monika, Bhatia, R., Jain, A., & Singh, B. (2022). Hash time locked contract based asset exchange solution for probabilistic public blockchains. *Cluster Computing*, *25*(6), 4189-4201.K. Elissa, "Title of paper if known," unpublished.

[3] Andersson, M., Chen Trieu, K., Debesay, P., Persson, J., Torrång, J., & Utbult, S. (2018). Decentralized Cryptocurrency Exchange A Proof-of-Concept based on Hashed Timelock Contracts

[4] Szabo, N. (1996). Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, *18*(2), 28

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, 12 Juni 2024

Fakhri Putra Mahardika 18221080