Project 3: Smart Tokens

ERC20 for Equity Certificates through Smart Contracts

Project Team 2
Lynette Cary, Marius Nsamou, Nigil Jeyashekar, Rajiv Shrestha, and Rawad Habib

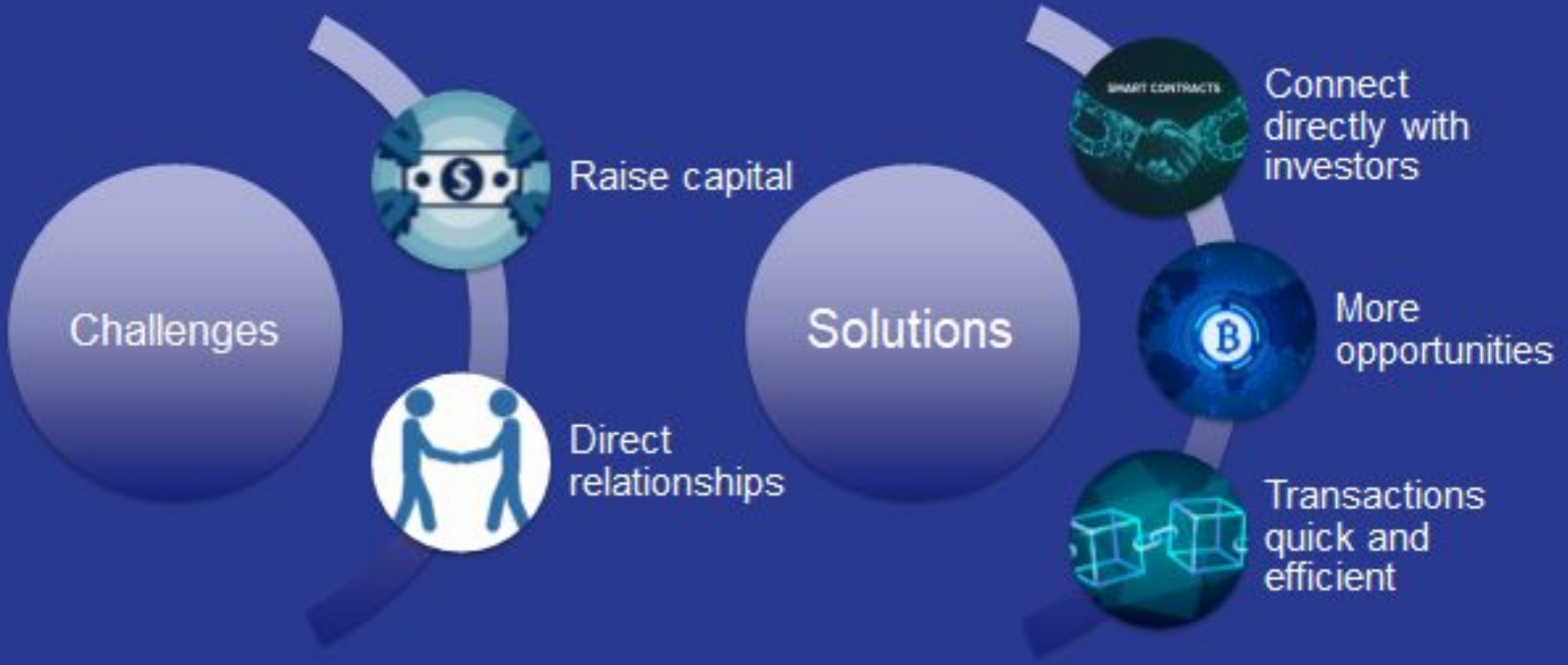
Concept Creation Basis: Regulation of Digital Assets

According to the Commodity Futures Trading Commission (CFTC), who oversees the digital assets market:

“Digital assets offer benefits of increased transaction speed, efficiency, certainty, democratization of markets and financial inclusion, automation through smart contracts, greater liquidity for assets, and enhanced security.”

CFTC Digital Assets Primer: https://www.ctfc.gov/media/5476/DigitalAssetsPrimer/download

Concept Creation Basis: Invest in Private Equity Firms



Medium Article, [Invest in Private Equity? Here are the Top 4 Reasons to Get Excited for Tokenization]

Project Setup: Framework and Dependencies

- Smart contract created to tokenize all of the firm's equity
  - Ethereum smart contract created using Solidity
  - Created ERC20-compatible token
  - Dependencies: Remix IDE, Ganache, MetaMask, and Pinata

- Standards imported from OpenZeppelin contracts:
  - SafeMath, ERC20, ERC20Detailed, ERC20Mintable, CrowdSale, MintedCrowdSale, and Ownable

Project Setup: Testing, Deployment, and Transaction

- Smart contracts compiled and deployed to a blockchain and tested using the Ropsten Test Network on Metamask

- Variables set and used throughout the contracts:
  - Contract ‘owner’ set to Firm and only the Firm can mint tokens
  - Token ‘symbol’ set to ‘DIGITAL’
  - ‘Exchange_rate’ set to equal 100 shares to be distributed per Ether

Smart Contract Framework: DigitalToken

Functions used in DigitalToken Contract

- **‘balance’** - returns available owner shares.
- **‘transfer’** - accepts recipient address to make transfer from owner to recipient.
- **‘mint’** - allows the company to create new tokes when needed`
- **‘purchase’** - calculates number of shares to distribute and adds value to the shares and transfers the value to the owner address.

```
contract DigitalToken {
    using SafeMath for uint;

address payable owner = msg.sender;
string public symbol = "DIGITAL";
uint public exchange_rate = 100;

mapping(address => uint) balances;
```

Smart Contract Framework: DigitalTokenERC20

Step 1:

```
contract DigitalToken is ERC20, ERC20Detailed {
    address payable owner;
```

Step 2:

```
modifier onlyOwner {
    require(msg.sender == owner, "You do not have permission to mint these tokens!");
    _;
}
```

Step 3:

```
function mint(address recipient, uint amount) public onlyOwner {
    _mint(recipient, amount);
}
```

Step 4:

```
constructor(uint initial_supply) ERC20Detailed("DigitalToken", "DIGITAL", 18) public {
    owner = msg.sender;
    _mint(owner, initial_supply);
}
```

Smart Contract Framework: DigitalTokenMinted

- DigtalTokenMinted contract passes ERC20, ERC20Detailed, ERC20Mintable to DigitalToken to mint an initial supply of tokens and additional tokens as needed.
    - Beneficiary name, token symbol, and initial token supply are constructor parameters and passed to ERC20Detailed contract.

```
contract DigitalToken is ERC20, ERC20Detailed, ERC20Mintable {
    constructor(
        string memory name,
        string memory symbol,
        uint initial_supply
    )
        ERC20Detailed(name, symbol, 18)
        public
    {
```

Smart Contract Framework: DigitalTokenSale

- Contract is created to manage the sale and minting of DigitalToken (shares).

- **Constructor parameters:**
  - rate - conversion between wei and token unit.
  - wallet - address receives the ether from sale of digital asset.
  - token - set to DigitalToken compatible with ERC20 interface required by crowdsale.

- Body of the constructor is empty and will inherit logic from OpenZeppelin: CrowdSale and MintedCrowdSale contracts.

Smart Contract Logic: DigitalTokenDeployer

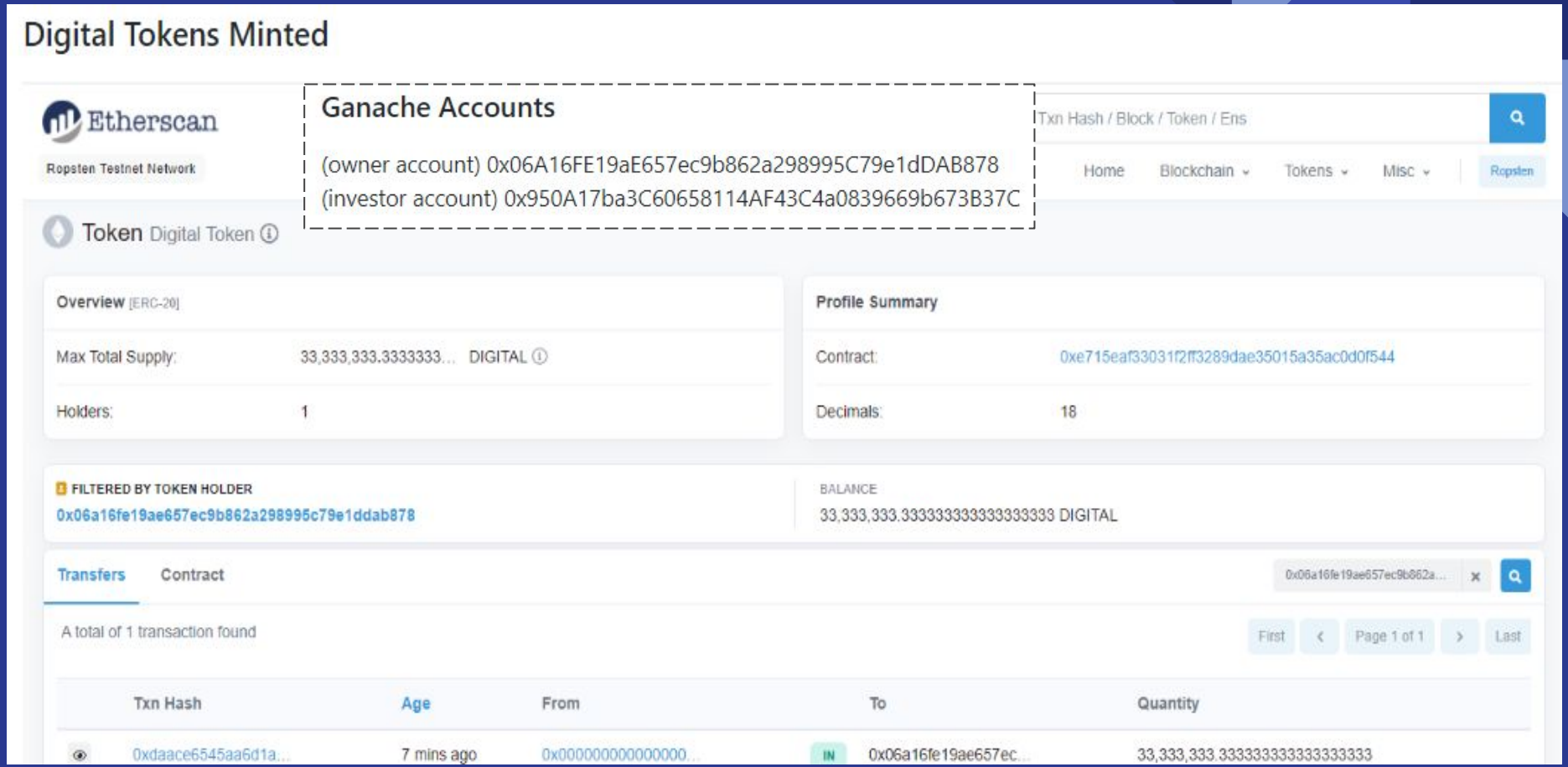
- Contract created stores addresses from DigitalToken and DigitalTokenSale once deployed.

- **Parameters:**
  - ‘rate’ - hardcoded to 1 to maintain the same units as Ether.
  - ‘wallet’ - will be paid all of the Ether raised by DigitalTokenSale.
  - ‘token’ - location where DigitalToken is stored.

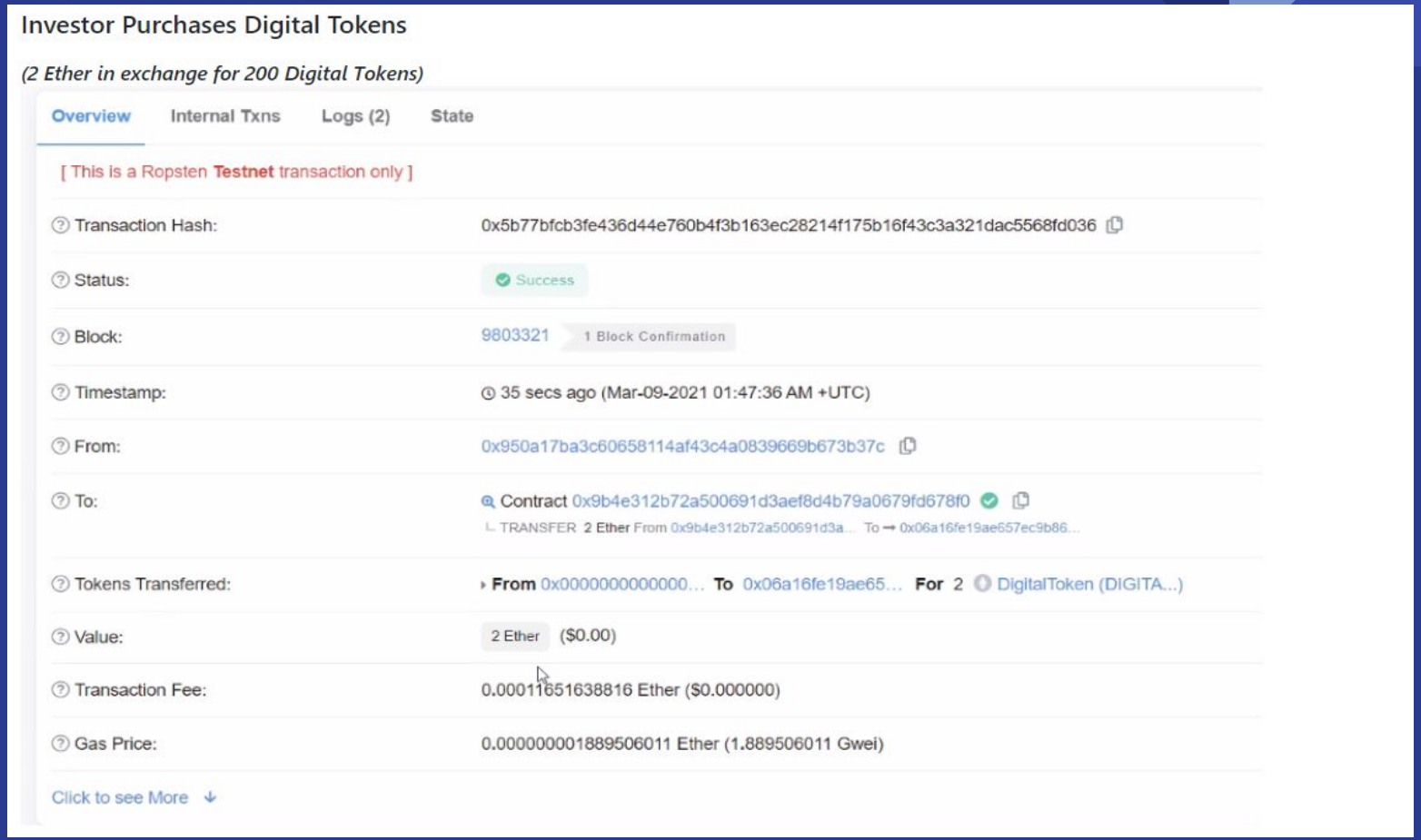
```
DigitalTokenSale token_sale = new DigitalTokenSale(1, wallet, token);
token_sale_address = address(token_sale);

                    token.addMinter(token_sale_address);
                    token.renounceMinter();
```

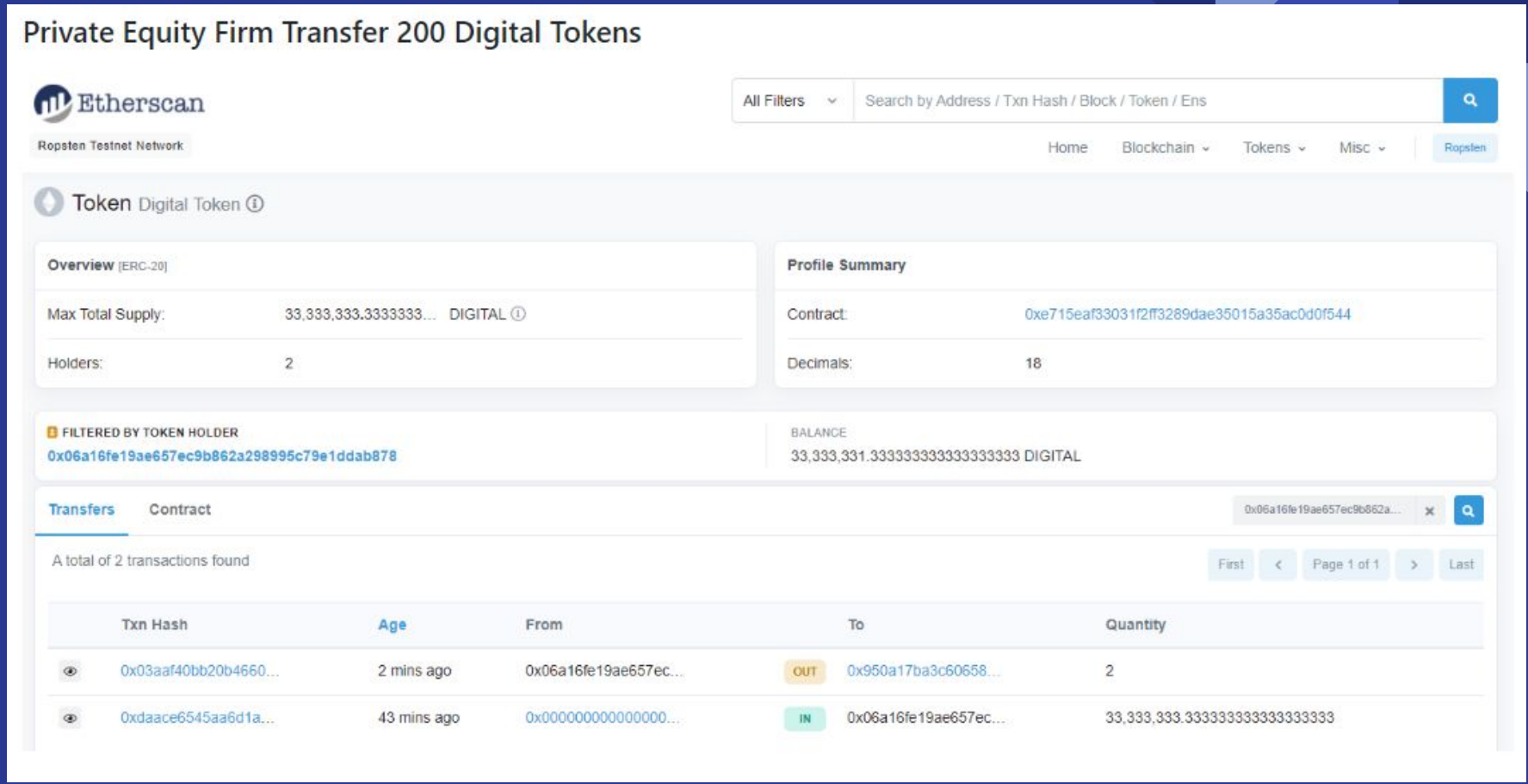
Smart Contract: Deployment and Test Transaction



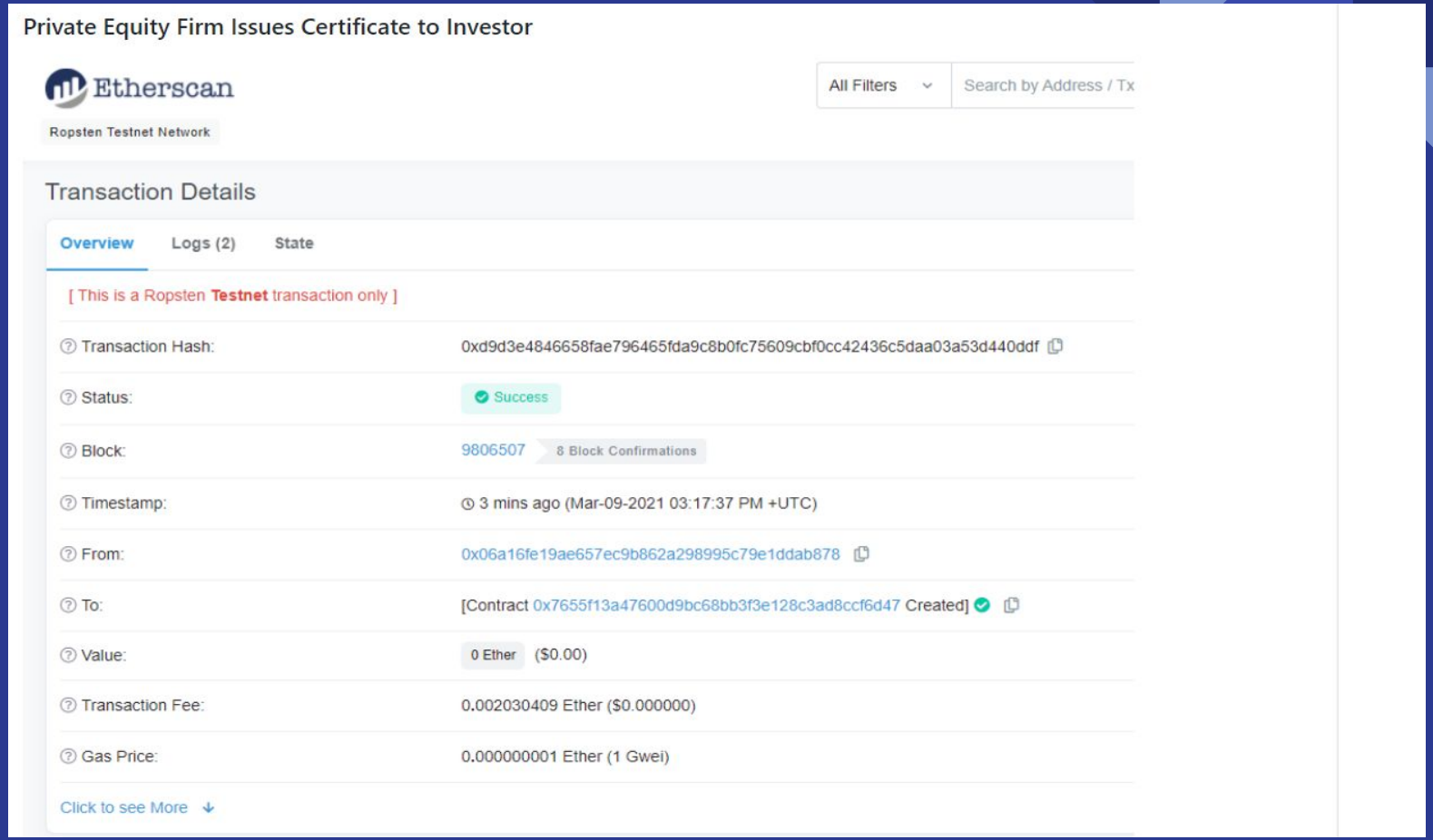
Smart Contract: Deployment and Test Transaction



Smart Contract: Deployment and Test Transaction



Smart Contract: Deployment and Test Transaction



Discussion

- Advantage of using blockchain technology and smart contracts is to increase the private equity firm's access to raise capital and provides investors with direct access to purchase equity in the firm.

- Blockchain is limited to the amount of data that can be stored.
  - Current smart contract framework in this project benefits a small private equity firm, with low number of transactions.
    - For larger firms with high number of transactions, we explored the option of using IPFS to store large amounts of data and certificates including the benefits that IPFS offers to provide verifiability on behalf of the firm.

- Each time tokens and certificates are issued, there is an increase in gas fees to the firm .
  - While there are various methods that can be utilized to help lower gas fees, we specifically identified that reducing the storage and size of variables used in writing smart contracts can help to reduce gas costs.

Post-Mortem

- **Difficulties**
  - Minting digital tokens required several iterations when transacting using metamask as smart contracts were not a single solidity file.

- **Additional questions for future research**
  - If more time was available the group could explore adding machine learning models and algorithms to analyze large amounts of data.
  - Further develop smart contracts to fully utilize IPFS for storing data and certificates.

Questions?