REPUBLIC OF CAMEROON
PEACE – WORK – FATHERLAND
****
UNIVERSITY OF BAMENDA
****
FACULTY OF SCIENCE
****
Centre for Cybersecurity and Mathematical Cryptology
****

REPUBLIQUE DU CAMEROUN
PAIX – TRAVAIL – PATRIE
****
UNIVERSITE DE BAMENDA
****
FACULTE DES SCIENCES
****
Centre de cybersécurité et de cryptologie mathématique
****

School: NAHPI

CENTRE FOR CYBERSECURITY AND MATHEMATICAL CRYPTOLOGY

Year 4

CYBE 6223 - Distributed Systems and Blockchains Security Project Report

Group 12 ERC20 Token (G12TK) and UBaEducationCredentialsStore Smart Contract on Sepolia

Submited by:

Names	Matricule	Option	Contribution
Nai Emmanuel	UBA24EP042	PMC	100%
Kuate Demmano Reine Erika	UBA24EP033	PMC	100%
Ngatchou Njiatou Landry Dimitri	UBA24EP023	PMC	100%

Course Lecturer: Dr. Konan Tchinda R

ACADEMIC YEAR: 2024/2025

I. INTRODUCTION

The primary objective of this project was to design, develop, and deploy a complete decentralized application (dApp) ecosystem on the Ethereum blockchain. This involved the creation of a ERC20 utility token, governed by a secure multi-signature protocol, and a secondary service contract that utilizes this token for payments. The project demonstrates a full development lifecycle, from local testing and security analysis to live deployment and interaction on the Sepolia public testnet, adhering to modern blockchain development best practices.

II. SYSTEM ARCHITECTURE AND DESIGN

The system is comprised of two distinct but interconnected smart contracts designed with a separation of concerns:

1. ***MyToken.sol***: This contract is a fully compliant ERC20 token named *"Group 12 Token"* **(G12TK).** Its primary role is to manage the currency of our ecosystem. It includes functions for users to **purchase** tokens with ETH and, most critically, implements a 2-of-3 **multi-signature scheme** for privileged administrative actions like **minting** new tokens or **withdrawing** collected ETH fees

2. ***UBaEducationCredentialsStore.sol*** : This contract provides the core utility of the application. It allows users to pay a fee in *G12TK* to permanently **store** a cryptographic hash (a "digital fingerprint") of their educational credentials on the blockchain. This serves as an immutable, publicly verifiable proof-of-existence for their documents. It is designed to interact with any ERC20 token and is managed by a single owner for fee adjustments and withdrawal of collected service fees.

a. The Mytoken.Sol Erc20 Contract

The *MyToken* contract was built using the OpenZeppelin library to ensure a secure and standard-compliant ERC20 implementation. This provides out-of-the-box functionality for ***transfer***, ***balanceOf***, ***approve***, etc. Custom logic was added to allow users to purchase G12TK directly by sending Ether to the contract address, facilitated by the ***receive()*** and ***buyTokens()*** functions, which calculate the token amount based on a fixed price.

The Multi-Signature Protocol

A key security requirement of the project was to avoid ***centralization of power***. Instead of a single "owner" controlling the token supply, we implemented a 2-of-3 multi-signature protocol. This is fundamentally more secure for two reasons:

- ***No Single Point of Failure***: If one of the three designated signer wallets loses its private key, the other two can still collaborate to manage the contract, preventing the project from being frozen.

- ***Increased Security***: A malicious actor would need to compromise two separate wallets, not just one, to perform unauthorized administrative actions, significantly increasing the difficulty of an attack.

This was achieved through an ***approve/execute*** pattern. For an action like minting, two of the three signers must first call the ***approveMint*** function with identical parameters. Only after ***two unique*** approvals have been registered can the ***executeMint*** function be successfully called to create the new tokens. The same logic applies to ***withdrawing*** ETH from the contract's vault.

b. The UBaEducationCredentialsStore.sol Service Contract

The ***UBaEducationCredentialsStore*** contract provides a decentralized service for notarizing documents. Users can submit a hash of a document, such as a university transcript or degree, to be stored immutably on the blockchain. This creates a permanent record that can be used to prove the existence and integrity of that specific document at a specific point in time.

Proof-of-Existence via Hashing

A critical design decision was to store only a cryptographic hash (specifically, a SHA-256 hash) of the user's credentials, not the credentials themselves. This approach was mandated by two fundamental principles of blockchain design:

- **Privacy**: The Ethereum blockchain is a public ledger, and storing sensitive, personal information like grades or student IDs directly on-chain would be a major privacy violation. A hash acts as a one-way "fingerprint"; it can be used for verification but cannot be reverse-engineered to reveal the original data.
- **Cost**: On-chain storage is extremely expensive. Storing a large file like a PDF or JSON object would cost a prohibitive amount in gas fees. A SHA-256 hash is always a fixed, small size (32 bytes), making it very cheap to store, regardless of the original document's size.

*The **approve/transferFrom** Mechanism*

To handle payments with our ***G12TK token***, we implemented the industry-standard ***approve/transferFrom*** mechanism. This is a secure, two-transaction process that ensures users maintain control of their funds.

- ***Approval***: The user first calls the ***approve()*** function on the ***MyToken*** contract, granting the ***UBaEducationCredentialsStore*** contract permission to withdraw a specific amount (the service fee).
- ***Transfer***: The user then calls the ***storeCredential()*** function on the service contract. Internally, the service contract then executes the ***transferFrom()*** command on the token contract to pull the pre-approved fee from the user's wallet into its own.

III. Deployment & On-Chain Evidence

Both contracts were successfully deployed and verified on the Sepolia public testnet. The final addresses are as follows:

- ***MyToken (G12TK)*** *: 0x35d2D00a2e1F426d511D8ffFDAa03596E902CbCe*
- ***UBaEducationCredentialsStore*** *:0x399995ba204589e006063977dACF2b43Df7DC501*

Required Transaction

As per the project requirements, a transfer of 10 G12TK tokens was successfully executed. The transaction can be viewed on Etherscan via the following hash:

0xed2189a3b9b416f6c327bfdb262c215d76d978aded4d0b552528fbcb04adcde8

CONCLUSION

This project provided a comprehensive, hands-on experience in the end-to-end development of a decentralized application. Key learnings included the implementation of critical security patterns like multi-signature administration, the nuances of on-chain data management via hashing, and the standard protocols for token-based payments. The successful deployment and interaction with the contracts on a public testnet confirm the validity and functionality of the designed system.

Recommendations: *Working on this project without prior knowledge or experience on the subject was a bit daunting added to this the electricity issues encountered during the development of the application and other unforeseen situations made the project fill more daunting to tackle. Taking into account these factors in the future may improve better collaboration and enhanced results*

USEFUL LINKS

- [youtube presentation] https://youtu.be/1NqVuWa2Bek
- [github repository] https://github.com/aspi1234/G12TK

Appendix

```
MyToken Contract
  Deployment
    ✓ Should set the correct token name and symbol (87ms)
    ✓ Should assign the correct signers (93ms)
  Token Purchasing
    ✓ Should allow users to buy tokens (812ms)
    ✓ Should emit a TokensPurchased event (318ms)
    ✓ Should revert if a user sends 0 ETH
  Multi-Signature Minting
    ✓ Should allow signers to approve a mint and execute it after 2 approvals (328ms)
    ✓ Should fail to execute mint with only 1 approval (100ms)
    ✓ Should prevent non-signers from approving a mint
    ✓ Should prevent a signer from approving twice
  Multi-Signature Withdrawal
    ✓ Should allow withdrawal after 2 of 3 signers approve (509ms)
    ✓ Should prevent non-signers from approving withdrawal

UBaEducationCredentialsStore Contract
  Deployment
    ✓ Should set the correct payment token address and fee
    ✓ Should set the deployer as the owner (41ms)
  Storing Credentials
    ✓ Should allow a user to store a credential hash after approving tokens (95ms)
    ✓ Should FAIL if the user has not approved enough tokens
    ✓ Should FAIL if the credential hash has already been stored (196ms)
  Administrative Functions
    ✓ Should allow the owner to withdraw collected fees
    ✓ Should prevent a non-owner from withdrawing fees (103ms)
    ✓ Should allow the owner to change the fee (45ms)
    ✓ Should prevent a non-owner from changing the fee


20 passing (13s)
```

Figure 1: All smart contract test unit test validation

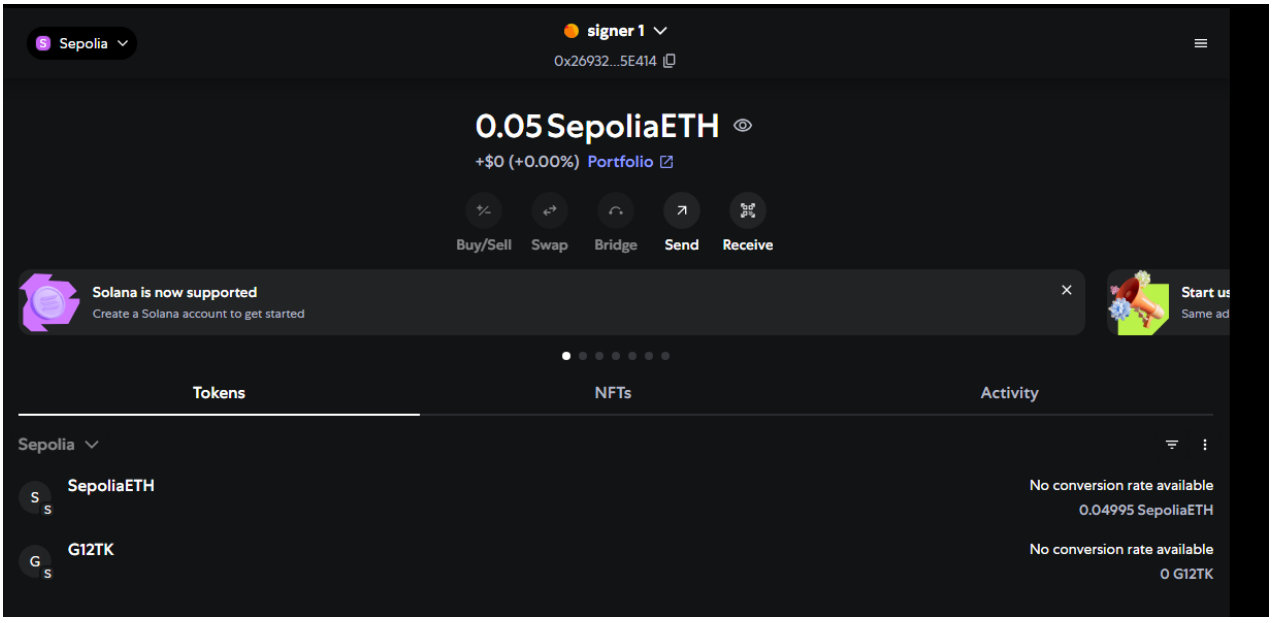

Figure 2 : Importing Token