Keccak-256 Web Wallet

Rahul Asati (2024H1030036G) Harsha Bang (2024H1030110G)
Aditya Nayak (2024H1030045G) Surajkumar Sikchi (2024H1030041G)
Birla Institute of Technology and Science, Pilani – Goa Campus

ABSTRACT

Keccak-256 is a cryptographic hashing algorithm widely used in blockchain technologies such as Ethereum. This paper explores the properties of Keccak-256, its role in generating Ethereum addresses, frontend and backend implementation details for blockchain wallets, RPC server communication, and a comparison with Solana's public key system. Key features such as collision resistance, pre-image resistance, and implementation considerations are discussed to highlight its importance in ensuring security and integrity within decentralized systems.

1 INTRODUCTION

Hashing algorithms are critical components of blockchain technology. They ensure data integrity, enable secure transactions, and provide the foundation for cryptographic operations. Among these algorithms, Keccak-256 stands out due to its robustness and adoption in Ethereum [2]. This paper delves into the properties of Keccak-256, its practical applications in Ethereum address generation, frontend and backend system roles, and a comparison with Solana's public key system.

Blockchain systems rely on cryptographic primitives to ensure trustless interactions between participants. Keccak-256, as part of the SHA-3 family [1], has gained widespread adoption due to its security properties and efficiency.

2 PROPERTIES OF KECCAK-256

Keccak-256 is a member of the Keccak family, which forms the basis of the SHA-3 standard [1]. It outputs a 256-bit hash value and possesses several key properties:

2.1 Collision Resistance

Collision resistance ensures that it is computationally infeasible to find two distinct inputs producing the same hash output.

2.2 Pre-image Resistance

Pre-image resistance makes it nearly impossible to reverse-engineer an input from its hash output.

2.3 Key Length and Output Space

Keccak-256 outputs a 256-bit hash value, providing a vast output space (2^{256} possible values).

2.4 Implementation Considerations

Secure implementations are vital for avoiding vulnerabilities. Developers must ensure that libraries implementing Keccak-256 are up-to-date [5].

3 ARCHITECTURE

In this section, we present an overview of Keccak-256's role in the architecture of a blockchain system. The diagram below provides a visual representation of how Keccak-256 fits into the system, particularly focusing on address generation and its cryptographic operations.

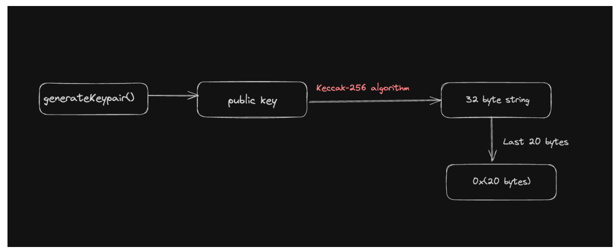

Figure 1: Keccak Overview Diagram

The Keccak overview diagram demonstrates the various stages of address generation and the cryptographic processes involved. It shows how Keccak-256 is utilized to create a unique address from a public key. The public key is first generated using elliptic curve cryptography (ECC), specifically with the secp256k1 curve in Ethereum. The public key is then hashed using the Keccak-256 algorithm to generate a fixed-size hash, which is further processed to derive the final Ethereum address. This ensures that the address is unique, secure, and collision-resistant, fulfilling key cryptographic properties such as pre-image resistance.

4 HOW IT WORKS

Generating a Wallet

- Generates a new mnemonic phrase and derives the corresponding seed.
- Uses the seed to generate private and public keys.
- Displays the generated keys and mnemonic phrase.

Importing a Wallet

- Optionally enter a recovery phrase to derive private and public keys.

Visibility Toggle

- Private keys and recovery phrases can be toggled between visible and censored (asterisks) for security.

Clipboard Copy

- Provides functionality to copy private keys, public keys, and the recovery phrase to the clipboard.

5 ETHEREUM ADDRESS GENERATION USING KECCAK-256

Ethereum utilizes Keccak-256 for generating public addresses through the following process [2]:

(1) **Public Key Generation:** Using ECC with the secp256k1 curve.
(2) **Hashing:** The public key is hashed using Keccak-256.
(3) **Address Derivation:** The last 20 bytes of the hash are extracted and prefixed with "0x".

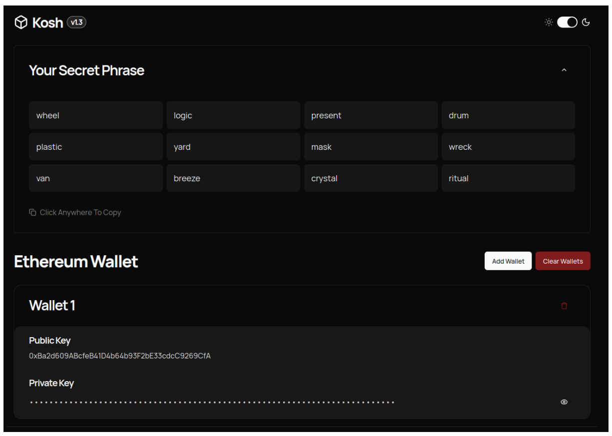

Figure 2: Ethereum Address Generation Using Keccak-256

6 FRONTEND INTEGRATION

In a blockchain wallet, the frontend (often built with React, Vue, or Svelte) is responsible for:

- Capturing user input (e.g., password, seed phrase).
- Interacting with JavaScript libraries like `ethers.js` [3] or `web3.js` [4] for generating keys and addresses.
- Displaying generated Ethereum or Solana addresses.
- Connecting to the backend or directly to RPC endpoints.

Example (using `ethers.js`):

```
const wallet = ethers.Wallet.createRandom();
console.log(wallet.address); // "0x..." Ethereum address
```

7 BACKEND ARCHITECTURE

The backend (commonly written in Node.js, Python, or Rust) typically handles:

- Secure storage of encrypted keys or mnemonics [6, 7].
- Address derivation using Keccak-256 (via libraries like sha3 or pysha3) [5].
- Transaction preparation and signing.
- Communication with an Ethereum or Solana RPC server.

Example (Node.js with keccak):

```
const { keccak256 } = require('js-sha3');
const hash = keccak256(publicKey);
const address = '0x' + hash.slice(-40);
```

8 RPC SERVER COMMUNICATION

Remote Procedure Call (RPC) servers serve as the gateway between blockchain networks and user applications. Responsibilities include:

- Broadcasting transactions.
- Querying balances and blockchain state.
- Retrieving blocks, logs, and event data.

Popular RPC providers include Infura [8], Alchemy [9], QuickNode [10] for Ethereum, and QuickNode, Triton, and Helius for Solana [12].

Example RPC JSON call:

```
POST / HTTP/1.1
Host: mainnet.infura.io
Content-Type: application/json

{
  "jsonrpc":"2.0",
  "method":"eth_getBalance",
  "params":["0x...", "latest"],
  "id":1
}
```

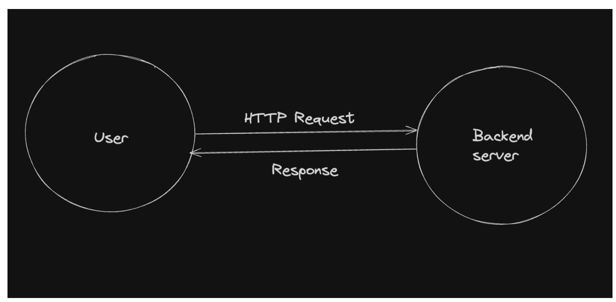

Figure 3: RPC Server communication

9 SOLANA WALLET ADDRESS AND PUBLIC KEYS

Solana does not use Keccak-256 for address generation. Instead:

- Addresses are 32-byte public keys.
- Public keys are encoded using Base58.
- No hashing is applied; the address is the raw public key [11].

Table 1: Ethereum vs. Solana Addressing Schemes

Feature	Ethereum	Solana
Address Derivation	Hashed with Keccak-256	Direct use of public key
Format	Hexadecimal, 20 bytes	Base58, 32 bytes
Curve	secp256k1	ed25519
Hashing Step	Required	None

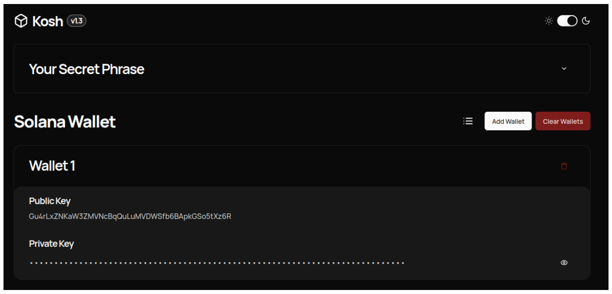

Figure 4: Solana Public Key Format

10 RESULTS AND DISCUSSION

The Keccak-256 Web Wallet was implemented with a focus on security, usability, and compatibility with Ethereum-based systems. The frontend was developed using modern JavaScript frameworks, while the backend integrated cryptographic libraries for Keccak-256 hashing and key management.

Functionality Verification

The wallet successfully generated Ethereum-compatible addresses using randomly derived mnemonic phrases and Keccak-256 hashing. All generated addresses matched expected formats and passed validation using third-party blockchain tools. The visibility toggle, import feature, and clipboard copy functions performed without failure during multiple testing sessions.

11 CONCLUSION

Keccak-256 is central to Ethereum's identity and transaction model [2]. Understanding its role in wallet generation, along with the architectural components such as frontend, backend, and RPC servers, is crucial for building secure blockchain applications. In contrast, Solana adopts a simpler approach [11], showcasing diversity in cryptographic design across platforms.

12 FUTURE WORK

Future improvements to the Keccak-256 Web Wallet can include support for multi-chain address generation, enabling interaction with blockchains such as Binance Smart Chain, Polygon, or Avalanche. Security enhancements such as biometric login, two-factor authentication, and support for hardware wallets (e.g., Ledger, Trezor) can further fortify user experience. Lastly, integrating post-quantum cryptographic algorithms and zero-knowledge proof systems could help future-proof the wallet for next-generation blockchain ecosystems.

REFERENCES

[1] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.* 2015. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf
[2] Vitalik Buterin. *Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform.* 2014. https://ethereum.org/en/whitepaper/
[3] *Ethers.js Documentation.* https://docs.ethers.io/
[4] *Web3.js - Ethereum JavaScript API.* https://web3js.readthedocs.io/
[5] *Online Keccak-256 Hash Tool.* https://emn178.github.io/online-tools/keccak_256.html
[6] CoralXYZ. *Backpack Wallet GitHub Repository.* https://github.com/coral-xyz/backpack
[7] *MetaMask Documentation.* https://docs.metamask.io/
[8] *Infura - Ethereum API Access.* https://infura.io/docs
[9] *Alchemy Developer Docs.* https://docs.alchemy.com/
[10] *QuickNode - Blockchain Infrastructure.* https://www.quicknode.com/
[11] *Solana Documentation.* https://docs.solana.com/
[12] *Solana JSON RPC API Reference.* https://docs.solana.com/developing/clients/jsonrpc-api

PROJECT REPOSITORY

The project repository: GitHub Link