IEEE ICBC'24 1570981888

Gasless On-Chain Password Manager: A Comparative Analysis Across EVM-Based Platforms

***Abstract*—In the 21st century, with the exponential advancement of the technology industry, most people manage multiple digital identities for diverse online interactions. Safeguarding this plethora of digital IDs necessitates the use of passwords, but the task of remembering them is a hassle of its own. While there exist different types of password managers as solutions, each comes with its own set of advantages and drawbacks. Centralized password managers have a single point of failure, which poses a substantial threat of cyberattacks and security breaches. Additionally, a centralized system requires a form of centralized trust. Conversely, blockchain-based password managers have gas costs and scalability issues. In this paper, we have discussed an architecture to develop a trustless novel gasless on-chain password manager with a robust recovery mechanism. The fundamental goal is to overcome the prevalent challenges related to gas fees in current on-chain password managers, which often impede accessibility and usability. To validate its efficacy, a comparative analysis was conducted across various EVM-compatible platforms, demonstrating the solution's performance across these platforms.**

***Index Terms*—Gasless, Password Manager, Blockchain, EVM platforms, Security, AES 256, Falcon, Symmetric and asymmetric cryptographic algorithms.**

I. Introduction

With data breaches costing them millions, cybersecurity has become an important factor for tech companies worldwide. The same goes for individuals entrusting their personal information to online platforms. Nearly all user-centric websites, including social media and bank services, now mandate robust security measures to verify user identity. One of the most widely used approaches is user ID and password verification. As cybersecurity threats continue to escalate, individuals are compelled to use lengthy and complex passwords to enhance protection. However, It's almost impossible to memorize such intricate passwords for a user. On the other hand, relying on a single password across multiple digital platforms poses an even greater security risk. In the event of a compromise, the attacker can gain access to a multitude of accounts from different sites.

This is where password managers step in. They efficiently handle all passwords for an individual across different websites, eliminating the need to memorize multiple complex passwords. When logging in to a specific site, the password manager automatically fills in the credentials, streamlining the user experience. The password manager has already affiliated a user-decided password with that specific site during the time of registration.

Most widely adopted password managers operate on a centralized architecture, where all user credentials are stored and managed by a single trusted authority. This centralized approach, however, introduces a major vulnerability - a single point of failure. Storing all user credentials in a central database makes it a prime target for cyberattacks and security breaches. Despite implementing encryption and security measures, these systems remain susceptible to vulnerabilities such as software bugs and insider threats. Moreover, the users are heavily reliant on the service providers for maintenance and security, introducing increased risks of service disruptions that could impede access to critical credentials. Compatibility issues, limited integration, and potential subscription-based models further limit accessibility and user preference in these systems, emphasizing the need for more secure and decentralized alternatives.

A decentralized on-chain password manager offers a range of benefits that enhance security and accessibility. The primary advantage of a blockchain-based password manager is secure password storage that leverages immutability and decentralization. As a result, passwords are no longer stored in a central database, ensuring protection from centralized vulnerabilities and potential breaches. Additionally, it enables users to access their passwords across multiple platforms and devices, leading to seamless authentication without relying on a single service provider.

While decentralized password managers offer increased security, they have their own disadvantages. Transaction costs (gas fees) on blockchain networks can be prohibitive for frequent password updates or interactions, impacting usability.

Usually, when an individual saves a password for a site by calling the smart contract with his account, he needs to pay crypto as a transaction fee in the form of gas. There are also scalability issues, specifically on busy networks, that may hinder efficient management and retrieval of passwords.

In this paper, we have designed a blockchain-based, gasless password manager using the EIP-712 protocol [1]. Along with literature survey, we have also discussed how the solution is implemented and how it performed on different EVM based platforms.

II. Literature Review

To overcome these prevalent challenges of password managers, researchers have already explored innovative solutions that are based on blockchain. In this section, we are going to review the existing work to provide a comprehensive understanding of the current state of password managers.

C. Luevanos et al. did a security analysis of three open-source, centralized password managers - Passbolt, Padlock, and Encryptr. In this paper, the authors examined their features, encryption methods, and vulnerabilities. They also compared the pros and cons for both open-sourced and closed-sourced password managers in terms of code transparency, security audits, user trust, and attack resistance. They also developed a set of guidelines to create a robust, more secure password manager [2]. However, in this paper, the authors do not touch on the scope of a decentralized password manager.

N. Jain proposed a password manager powered by blockchain, which consists of a back-end blockchain network and a front-end web application. He further explained that the password manager works using AES-256-CBC encryption to store and retrieve credentials on the blockchain and uses proof of work as the consensus algorithm to validate the transactions and blocks [3]. However, in this paper, he did not talk about any transaction fees regarding storing the passwords.

C. -H. Liao et al. introduced a banking ecosystem-specific decentralized identity management and access control framework called BIMAC. This framework uses smart contracts, stateless authentication, and self-sovereign identity to provide functionalities such as third-party login, data authorization, etc. The authors have also compared BIMAC with other related works and claimed that it can provide a more reliable, secure, and convenient platform for banking services [4]. Since this decentralized framework is bank services-centric, it would be difficult to re-purpose it to a more generalized use case.

A. Debbie et al. patented a system that can monitor and validate passwords across different domains using blockchain to store and compare password hash values. They implemented the system with three components - a password monitor agent, a password hash generator, and a password history chain. The password monitor agent can identify password input events and send them to the password hash value generator, which then creates a password hash using a pre-existing hash function and compares the hash against the password history chain. This prevents users from reusing passwords that have been used in other accounts [5]. Although this system enhances security, it does not solve the problem of having to memorize passwords for users.

I. A. Mohammed discussed the application of identity and access management in blockchain technology, with a focus on Ethereum. He used concepts such as hashing, proof-of-work, smart contracts, and self-sovereign identity to design the system. He described how users can register their identities on the blockchain by creating a public-private key pair and a smart contract that contains their personal information. The users can then authenticate themselves to other parties by signing a challenge message with their private key and sending it along with their smart contract address [6]. This paper also does not discuss anything about the transaction costs on the blockchain.

D. Tse talked about a password-keeping system using blockchain technology, employing AES-128 symmetric encryption and asymmetric key encryption to enhance security and trustworthiness. The system encrypts a user's password with a symmetric AES-128 key, which is then encrypted with an asymmetric Master Public Key. Both encrypted keys are stored on the blockchain along with the user's website domain name and username [7]. In case of password retrieval, the Master Private Key, owned by the user, is used to decrypt the AES-128 key, which in turn decrypts the password. This approach ensures high efficiency, convenience, and security, leveraging the immutable and transparent nature of the blockchain and the robustness of AES-128 encryption. The approach of this paper is quite similar to our work since we are also taking advantage of the immutable and transparent nature of the blockchain for storing encrypted passwords. Here, the paper implements AES-128 for symmetric encryption, while we have explored both AES-256 and Falcon algorithms to ensure additional quantum resistance and gasless transactions.

Similarly, Y. Lou et al. also implemented a password management system with a novel memory-hard password hashing scheme (MH-PBKDF2) for blockchain-based cyber-physical systems (CPS). This scheme is an extension of the classic PBKDF2, integrating a memory-hard feature to enhance security against attacks facilitated by parallel computing devices. Their target was to make the system more efficient and lightweight, catering to the resource-constrained nature of CPS. [8] This paper has both drawbacks and benefits. While we are considering a quantum-resistant encryption method, their scheme was to defend against parallel computing attacks. However, this could be a potential limitation because they mostly focused on memory-hardness, potentially overlooking other security aspects.

A. Catalfamo et al. [9] proposed a complex implementation of the MBB-OTP protocol, which is a decentralized solution for authentication, combining microservices architecture and blockchain technology. It involves three main components: Authentication Microservice (AMS), OTP Generation Microservice (OTP-GMS), and Sender Microservice (SMS). These components work together to generate and deliver OTPs, ensuring integrity and security. The protocol uses Blockchain to store a shared secret vital for OTP gen-

eration, enhancing security against threats like man-in-the-middle (MITM) and denial of service (DoS) attacks. However, the implementation is quite complex compared to centralized solutions. There will also be potential scalability issues with increasing user bases and transaction volumes that are not considered in their work.

C Zhuang et al. [10] presented a key recovery system utilizing password-protected secret sharing (PPSS) within a permissioned blockchain. This method enhances security by encrypting secret fragments with a user's password, ensuring that the key recovery server cannot access the user's secret. Their target was to ensure password-protected secret sharing (PPSS) for key recovery in permissioned blockchains, whereas our work concentrated on quantum-resistant cryptography. The PPSS system is tailored for permissioned blockchain environments, ensuring confidentiality and security against specific types of attacks prevalent in these systems. However, similar to the previous paper, this paper has the following issue: Complexity and potential overhead in implementing the password-protected secret sharing system. Moreover, there is potential scalability issues with the increasing number of users and secret fragments.

III. Methodology

The methodology encompasses a comprehensive approach, initiating research and requirement analysis to define user needs and system essentials. It progresses through design, development, testing and comparative analysis. Each phase aims to craft, refine, and evaluate the gasless on-chain password manager, ensuring robust security, usability, and platform adaptability.

A. Design and Development

This phase focuses on crafting the architecture and framework of the gasless password manager. It incorporates encryption mechanisms, cryptographic key storage protocols, and recovery mechanisms into the system design. Leveraging smart contracts and blockchain technology, we developed a prototype ensuring gasless transactions and effective recovery procedures. The tools used for the experiment were Node.js for the backend, hardhat and solidity for smart contract development, JS libraries like chai, and mocha for testing purposes.

- A Node.js project was set up along with the hardhat project for creating a smart contract development environment.
- The smart contract was developed with solidity.
- The compilation and deployment of the contract was done using hardhat.
- Test script was written in JavaScript and Typescript to verify that the contracts were behaving in the desired manner.
- Typescript was also used for writing custom deployment scripts.
- The contracts were deployed on 11 different Ethereum Virtual Machine(EVM) compatible distributed ledgers.
- In the backend, two different cryptographic algorithms were implemented(AES 256, Falcon)
- Scripts were written to compare the performance of the two.
- The key exchange, encryption, and decryption flow simulation were also written in Typescript.
- Performance and gas cost of the five smart contract functions(registerFunction, registerPlatform, storePassword, getPassword, freezeAccount) were tested for all the 11 EVM-based distributed ledgers.
- Ethers library was used to simulate off-chain signatures.

For the adopted algorithm, we have explored one symmetrical cryptographic algorithm called AES 256 and an asymmetric algorithm called Falcon. The comparative analysis of these two algorithms is done on three factors: with the increased size of strong passwords plotted out the encryption time, decryption time, and the encrypted password size.

B. Testing and Security Audit

We iterated the original code a couple of times to find bugs in the solidity contract. We have also implemented some automated testing scripts to make the implementation more robust. This rigorous testing helped to validate the functionality, security, and resilience of the developed password manager.

C. Comparative Analysis across EVM-based Platforms

The deployment of the password manager across eleven EVM-based platforms forms the crux of this phase. We have evaluated the performance in terms of gas cost and timing of calling different functions for these EVM-based platforms - *Ethereum, Polygon, Celo, Avalanche, Hedera Hashgraph, Binance smart chain, Arbitrum, Optimism, Fantom, Tron, Conflux*. For comparative analysis, testnet is used. We set up a wallet for each of the networks and filled each wallet with a balance from the faucet network. We deployed the contract 5 times in a row and called the five smart contract functions with the exact same input for all the networks. We measure the performance in terms of time and cost and calculate the mathematical average. We repeat the same process three times on three different random days and do an average of three. This is done because the cost and time fluctuate a little depending on which time of the day the functions were called, based on network traffic.

IV. Solution

The proposed gasless password manager securely stores encrypted passwords on the blockchain without incurring gas fees for users. It utilizes off-chain signature mechanisms for authentication, allowing encrypted passwords to be stored within smart contracts with user control. Additionally, the system incorporates a robust account freezing mechanism, enabling users to safeguard their accounts in case of security threats or unauthorized access

A. Components

The proposed system solution has a couple of web2 components working together with the web3 components. Here are the components of the system:

- Client
- Web2 Back-end Server
- Blockchain
- Smart Contract
- Decentralized Oracle

These components are discussed in details in this section of the paper.

1) Client: The client device, often a user's personal computer or mobile device, acts as the gateway to the gasless password manager system. In this research setting, we have used Next.js, HTML, and CSS to implement and host the front-end application. Using this stack, a secure interface was created to manage passwords and interact with blockchain functionalities. This device serves as the platform for secure encryption and decryption processes, ensuring confidentiality and protection of sensitive data. This device also facilitated user interactions with back-end services and enabled seamless communication with the server-side API and the blockchain network, ensuring convenient and secure access to the password manager from various operating systems and devices. This front-end part is responsible for three types of key management:

- Symmetric and Asymmetric Key Pair Storage for Encrypting Password. Here, we have explored two cryptographic algorithms for encryption:
  - Symmetric Key Pair Storage:
    The client application manages the symmetric keys used for encrypting passwords securely. These keys are crucial for encryption and decryption operations, ensuring password confidentiality. They're securely stored and managed within the client-side application.
  - Asymmetric Key Pairs for encryption:
    Alternatively, the client app can adopt an asymmetric cryptographic scheme with a set of asymmetric key pairs used specifically for password encryption. These keys enable the secure transmission of encrypted passwords to the server, safeguarding them from unauthorized access
- Asymmetric Key Pairs for Ring Signature:
  The client side manages another set of asymmetric key pairs dedicated to ring signatures. These keys are vital for generating and verifying ring signatures, providing an extra layer of anonymity and authenticity for interacting with the backend rest APIs.
- Blockchain Address Key Pairs:
  The client application is responsible for managing the key pairs associated with the user's blockchain address. These keys facilitate interactions with the blockchain, such as signing transactions, verifying authenticity, and ensuring the user's identity during interactions with smart contracts or other blockchain-based operations.

2) Web2 Backend Server: The gasless password manager system's back-end component manages essential backend operations, such as key pair management. It also provides RESTful APIs that facilitate secure communication between the front end and the blockchain.

- Key Pair Storage for Admin Blockchain Address:
  The backend system securely stores the key pairs associated with the admin's blockchain address. These keys are essential for system administration, allowing interactions with the blockchain on behalf of the system.
- RESTful APIs for Smart Contract Interactions:
  - APIs Calling Smart Contract on User's Behalf:
    Backend offers APIs that initiate interactions with smart contracts on behalf of users. These APIs enable actions such as storing encrypted passwords on the blockchain or retrieving encrypted password data securely.
  - APIs with User Off-chain Signatures:
    Additionally, the backend provides APIs that require user-off-chain signatures. These APIs facilitate actions such as initiating transactions with the blockchain or verifying the user's authenticity for specific operations.

3) Blockchain: The gasless password manager system employs blockchain technology to establish a decentralized foundation for secure password management. The blockchain component of this system utilizes a distributed network to guarantee data integrity, security, and transparency. By utilizing a consensus mechanism, it protects encrypted password data stored on the blockchain, providing users with unchangeable and tamper-proof records. The decentralized nature of the blockchain component eliminates the possibility of single points of failure, building trust in the system and allowing users to interact with their password data securely without relying on centralized authorities.

4) Smart Contract: The gasless password manager system has a smart contract segment that controls interactions with the blockchain. These self-governing contracts are created using Solidity to perform predefined functions like storing encrypted passwords and verifying off-chain signatures. Smart contracts use cryptography to enable gasless transactions for users, ensuring efficient password-related operations while maintaining data confidentiality. Furthermore, these contracts create guidelines for user authentication and access control, providing a secure environment for password management in the blockchain ecosystem. The smart contract was initially developed and tested locally, and then deployed on blockchain networks.

5) Decentralized Oracle: The gasless password manager system relies on the decentralized Oracle component to generate secure and transparent random numbers. This component connects the blockchain with the off-chain world by utilizing decentralized oracle networks like Chainlink oracles. It collects random number data from various off-chain sources and validates and aggregates it before feeding it into the blockchain. To ensure the randomness and reliability of gen-

erated numbers, the decentralized oracle leverages multiple independent data sources and consensus mechanisms, which mitigates the risks associated with single-point manipulation or biases. Acting as a trusted bridge, this component provides the gasless password manager system with decentralized and verifiable random number generation, which is essential for cryptographic operations and secure password management.

B. Core Functionalities

1) Decentralized Password Manager Architecture: Here, we will explain in detail how the store-encrypted password mechanism works.

- Register User:
  On the user's device, the client application is installed, the three key managers are set up, and all key pairs are created including the blockchain credentials. From the client side, REST API is called with the blockchain address, and on the backend, the user is registered with the smart contract call.
- Register platform:
  Platform refers to the website/application whose password the system is storing. In this step, REST API is called from the client side with the user blockchain address and platform name. In the back end, the platform is registered on the smart contract function call.
- Password storage mechanism:
  - Secure Transmission of public key:
    From the client side, the user's public key is sent via RESTful API.
  - Random number generation:
    The server generates a random number from the decentralized Oracle setup in the Chainlink platform. This random number serves as a crucial element in the encryption and signature process.
  - Encryption of random number:
    Upon receiving the public key, the server encrypts the generated random number with the user's public key using the Elliptic Curve Cryptography (ECC) asymmetric encryption algorithm.
  - Transmission to user and decryption:
    The server sends the encrypted random number securely to the client. The client decrypts the received encrypted random number using their private key.
  - Off-Chain Signature Creation:
    The client, having decrypted the random number, generates a signature off-chain using their private key. This signature corresponds to the user's blockchain address and serves as an authentication mechanism on-chain.
    Three components of the generated signature are called '*v*', '*r*' and '*s*'
    I. 'v' (Recovery ID): It denotes the recovery identifier used to retrieve the public key from the signature. In EIP-712, 'v' stores the recovery ID or chain ID, which helps in determining the chain context for the signature. This value is crucial for correctly recovering the signer's address from the signature.
    II. 'r' (Signature Parameter): 'r' represents the first 32 bytes of the signature's cryptographic representation. It's part of the signature generated during the signing process and helps in uniquely identifying the signature.
    III. 's' (Signature Parameter): Similar to 'r', 's' is the second 32 bytes of the signature's cryptographic representation. It complements 'r' and, when combined, forms the complete cryptographic signature.
    Together, 'v', 'r', and 's' constitute the components of an off-chain signature following the EIP-712 standard.
  - Secure Password Saving with Off-Chain Signature:
    With the encrypted password, the off-chain signature arguments, and the random number, the user securely sends these details to the server using asymmetric cryptography.
  - Smart Contract Validation and Password Saving:
    The server receives the encrypted password and the user's off-chain signature arguments. The server then interacts with the smart contract on the blockchain, passing the encrypted password, the user's off-chain signature, and the random number. At the contract level, the system verifies if the off-chain signature belongs to the user. The encrypted password is securely saved to the smart contract if the validation succeeds.

This whole process is illustrated through a simple diagram as shown below in Fig. 1:

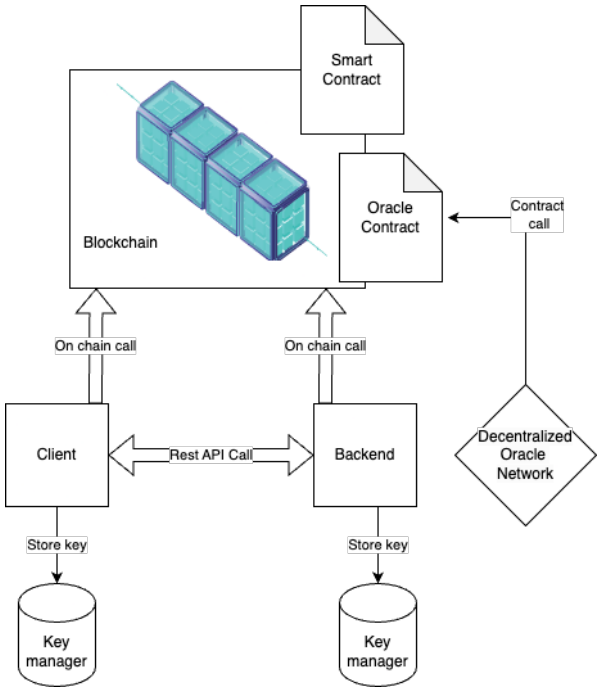

Fig. 1. Store Encrypted Password.

2) Freeze Account: For the recovery mechanism, a different blockchain address is set up on a different device. If the original client device is lost, then with the backup device the

stored password can be frozen. The mechanism works in the same way as other mechanisms in terms of storing passwords with the off-chain signature.

V. Analysis and Results

We have compared how the Falcon as well as the AES 256 cryptography algorithm performed in various cases. Moreover, we have done a comparative analysis of the encryption schemes for the overall gasless password management system.

A. Security Analysis of AES 256 vs Falcon

We have done the security analysis of Falcon vs AES 256 from four different perspectives, as shown in "Table I".

TABLE I
Security Analysis

Metrics	Cryptography algorithms	
	AES 256	***Falcon***
Key length	Fixed Length. 256-bit key.	Varying length with desired security. 1024-bit key when compared to AES 256.
Vulnerability	Not breakable with known attacks.	Not breakable with known attacks.
Performance	Very efficient. Well suited for variety of application	Slower than AES 256. Considered efficient for applications with high level of security. Used in digital signatures required for financial transactions and electronic voting.
Quantum Resistance	Quantum resistant till now. Future risk of vulnerability against powerful quantum computer.	Post-quantum cryptography (PQC), lattice-based algorithm . Designed to resist attacks from quantum computers. So quantum resistant.

Quantum computers are still in their early stages of development, but they have the potential to break many of the encryption algorithms that are currently in use. As a result, it is important to continue to research and develop new PQC algorithms, such as Falcon.

B. Comparative analysis on encryption schemes

We have recorded the encryption and decryption time as well as the encryption password size for the two algorithms as shown below in the graphs:

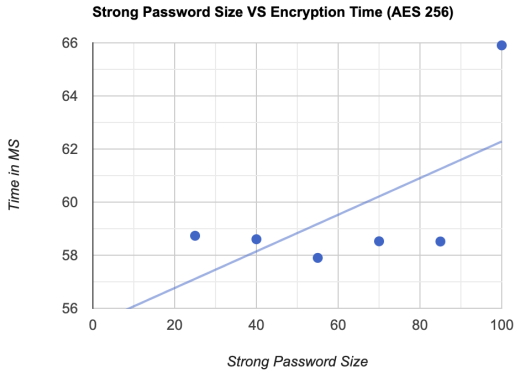

Fig. 2. AES 256 encryption time

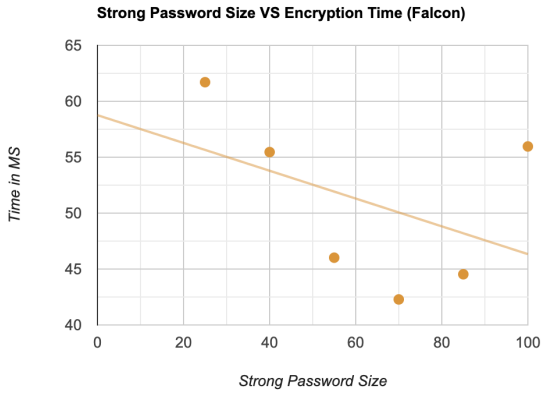

Fig. 3. Falcon encryption time

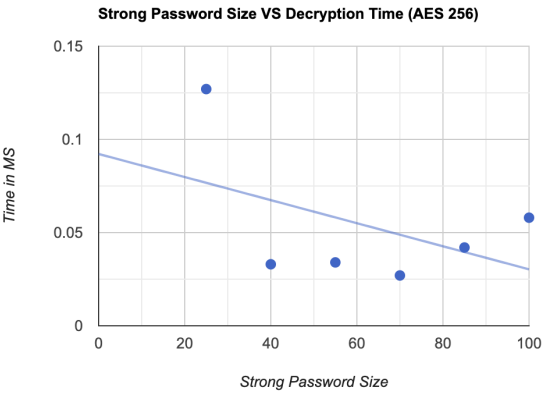

Fig. 4. AES 256 decryption time

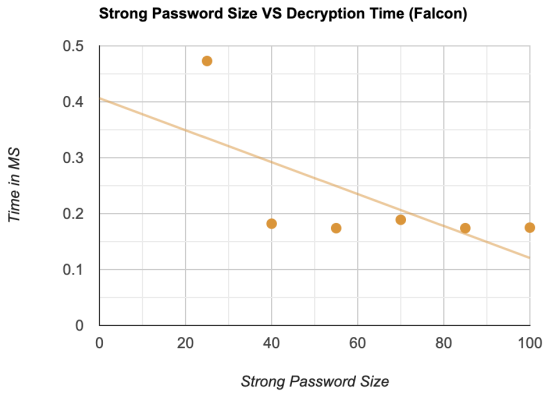

Fig. 5. Falcon decryption time

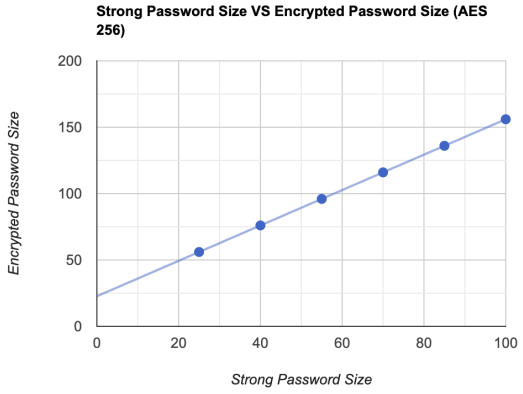

Fig. 6. AES 256 encrypted password size

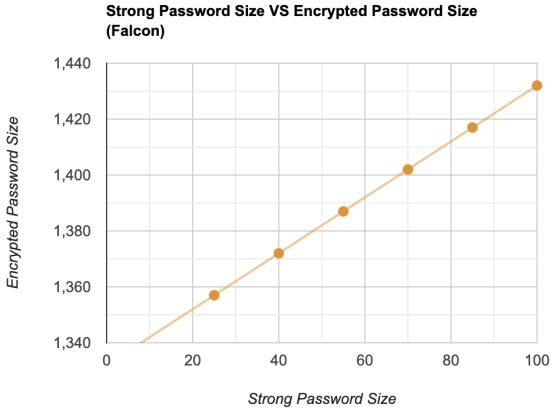

Fig. 7. Falcon encrypted password size

TABLE II
ALGORITHM PERFORMANCE

	Cryptography algorithms	
Mean time	***AES 256***	***Falcon***
Encryption	59.697ms.	50.994ms
Decryption	0.0535ms.	0.228ms.

As shown in "Table II" AES 256 is faster at decryption, but it is worth adopting Falcon as it is more secure. Although the size of the on-chain encrypted password is significantly longer for Falcon, resulting in higher gas costs, a shorter stronger password can be used. For the current context, the encryption time also includes the key generation part since they happened in tandem, one time for a single application context. So shorter decryption time is more significant and AES-256 would be better.

C. Comparative analysis across EVM platforms

In this section, performance across eleven EVM-compatible platforms is discussed in detail.

Functions	registerFunction	registerPlatform	storePassword	getPassword	freezeAccount
Ethereum	8.212628532	8.212628532	14.31291324	0	13.678413
Polygon	0.0006222645437	0.0006222645437	0.001105600272	0	0.001051448126
Celo	0.0002749783878	0.0002749783878	0.0004885756739	0	0.000464645868
Avalanche	0.045192355	0.045192355	0.0802968025	0	0.07636396875
Hedera Hashgraph	0.01357073867	0.01357073867	0.01357073867	0.01357073867	0.01357073867
Binance smart chain	0.217808344	0.217808344	0.386997172	0	0.36804255
Arbitrum	0.00000486772	0.00000486772	0.000008649036	0	0.00000822525
Optimism	0.00001256553441	0.00001256553441	0.0000223261707	0	0.00001903158122
Fantom	0.00001887248063	0.00001887248063	0.0000335316710	0	0.00003188948616
Tron	0.9113727043	0.9299274863	2.024121626	0	1.958799652
Conflux	0.000005299455	0.0000052359354	0.0000090415719	0	0.00000868870575

Fig. 8. Gas cost in USD for executing smart contract functions across different EVMs

Functions	registerFunction	registerPlatform	storePassword	getPassword	freezeAccount
Ethereum	13022.69272	12136.16151	12762.51745	820.5156307	12448.86171
Polygon	11163.25069	7064.824853	6836.539128	594.7521028	6477.21927
Celo	6178.699833	6833.937672	6515.576522	528.5284112	7096.561603
Avalanche	7487.621658	7077.504169	7807.787775	379.8512639	7463.262217
Hedera Hashgraph	6832.049089	5481.2968	3719.560586	3618.026856	3748.728625
Binance smart chain	6063.814847	5984.01553	6087.543025	332.9079584	6299.909478
Arbitrum	3880.488636	4222.161178	4063.781914	614.3877582	3822.37267
Optimism	8306.435394	7692.127003	8535.5063	854.0839861	8668.356781
Fantom	3384.954478	2779.500939	2398.680856	479.0308168	2726.77513
Tron	65401.36344	66111.88889	65387.84802	1388.194741	65595.71727
Conflux	8782.807906	9265.415633	8621.609728	766.2970834	8690.024861

Fig. 9. Time in milliseconds for executing smart contract functions across different EVMs

As shown in the tables in "Figure 8" and "Figure 9" we have compared the gas cost and time cost of executing smart contract functions. For the execution of the functions that change

the smart contract state (registerFunction, registerPlatform, storePassword, freeezeAccount) we found that Ethereum is the most expensive one. For the get function (getPassword) most EVM-compatible platforms do not require gas fees except for Hedera Hashgraph which also takes more time to execute. The least amount of gas cost for blockchain is for Arbitrum which is a layer 2 solution, and among layer 1 EVM-supported platforms, Conflux requires the least gas fees. In terms of time cost, the least time was recorded for the get function of Binance Smart Chain, while for other functions, the Fantom platform performed most efficiently.

VI. Limitation

Our proposed gasless on-chain password manager, while innovative, faces several limitations. Its scalability and performance are a concern, particularly under high transaction volumes on blockchain networks, which could affect efficiency. The system's reliance on the stability and security of blockchain platforms poses risks, as any inherent vulnerabilities could impact its functionality. Our research, focused mainly on EVM-compatible platforms, may not fully represent the diversity of blockchain environments, potentially limiting the generalizability of the findings. Diversifying the study to include a wider range of blockchain platforms beyond EVM-compatible ones would provide a more comprehensive understanding of the system's applicability. Additionally, the rapidly evolving cybersecurity landscape demands continuous adaptation of the system to new threats, a need that might not be fully addressed in the current framework. These limitations for the ongoing research might not directly affect the system's robustness and adaptability in the dynamic fields of cybersecurity and blockchain technology, however, these are areas we will be able to work on in the near future to enhance the overall usability of the system.

VII. Conclusion

In summary, we propose a scheme for a trustless decentralized way of managing passwords on web platforms utilizing blockchain technology. Additionally, from a user perspective, we debarred the acquaintance of blockchain-based wallets to keep in mind the usability of the platform. As part of our research, we tried out a couple of promising cryptographic algorithms for on-device secure password storage. We removed the need for refilling the user wallet address by implementing the on-chain password storing mechanism on the smart contract level using EIP 712 off-chain signature. Future works may extend this paper's work on exploring different cryptographic schemes and testing on other blockchain-based platforms. In our result analysis, we found the chosen symmetric platform (AES-256) gave better results in the context of a password manager. In terms of blockchain layer1 solutions, Fantom, and Conflux demonstrated superior performance and among layer2 solutions, Arbitrium stood out in our decentralized password manager. Finally, the comparative analysis code for the research work is maintained using GitHub public respository.

References

[1] Ethereum Gasless Meta-transactions by Using EIP-712, https://hackernoon.com/ethereum-gasless-metatransactions-by-using-eip-712.

[2] C. Luevanos, J. Elizarraras, K. Hirschi and J. -h. Yeh, "Analysis on the Security and Use of Password Managers," 2017 18th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), Taipei, Taiwan, 2017, pp. 17-24.

[3] N. Jain, "Transparent and secure password storing mechanism using blockchain", International Research Journal of Modernization in Engineering Technology and Science, vol. 04, no. 06, 2022.

[4] C. -H. Liao, X. -Q. Guan, J. -H. Cheng, S. -M. Yuan, Blockchain-based identity management and access control framework for open banking ecosystem, Future Generation Computer Systems, vol. 135, pp. 450-466, 2022.

[5] A. Debbie, A. Howard, L. Su, L. Yu, Password management and verification with a blockchain, 2020, https://patents.justia.com/patent/10594487.

[6] I. A. Mohammed, "IDENTITY and ACCESS MANAGEMENT SYSTEM BASED ON BLOCKCHAIN", IDENTITY, vol. 8, no. 6, 2021.

[7] Tse, D., Huang, K., Bin, C., and Liang, K. (2018, December 1). Robust Password-keeping System Using Blockchain Technology. https://doi.org/10.1109/ieem.2018.8607284

[8] Luo, Y., Su, Z., Zheng, W., Chen, Z., Wang, F., Zhang, Z., and Chen, J. (2021, March 8). A Novel Memory-hard Password Hashing Scheme for Blockchain-based Cyber-physical Systems. ACM Transactions on Internet Technology. https://doi.org/10.1145/3408310

[9] Catalfamo, A., Ruggeri, A., Celesti, A., Fazio, M., and Villari, M. (2021, September 5). A Microservice and Blockchain Based One Time Password (MBB-OTP) Protocol for Security-Enhanced Authentication. 2021 IEEE Symposium on Computers and Communications (ISCC). https://doi.org/10.1109/iscc53001.2021.9631479

[10] Zhuang, C., Dai, Q., and Zhang, Y. (2022, January 1). BCPPT: A blockchain-based privacy-preserving and traceability identity management scheme for intellectual property. Peer-to-Peer Networking and Applications https://doi.org/10.1007/s12083-021-01277-1