Detecting and Preventing Credential Misuse in OTP-Based Two and Half Factor Authentication Toward Centralized Services Utilizing Blockchain-Based Identity Management

Jozef Drga  
Brno University of Technology  
Faculty of Information Technology  
idrga@fit.vutbr.cz

Ivan Homoliak  
Brno University of Technology  
Faculty of Information Technology  
ihomoliak@fit.vutbr.cz

Juraj Vanco  
University of Tomas Bata  
Faculty of Applied Informatics  
vanco@utb.cz

Martin Perešíni  
Brno University of Technology  
Faculty of Information Technology  
iperesini@fit.vutbr.cz

Petr Hanacek  
Brno University of Technology  
Faculty of Information Technology  
hanacek@fit.vutbr.cz

Athanasios Vasilakos  
University of Agder  
thanos.vasilakos@uia.no

Abstract—This paper focuses on the problem of detection and prevention of stolen and misused secrets (such as private keys) for authentication toward centralized services. We propose a solution for this problem, based on SmartOTPs, the two-factor authentication scheme against the blockchain, which is intended for smart contract wallets and utilizes one-time passwords (OTPs). We modify SmartOTPs for our purposes and utilize them in the setting of two-and-a-half-factor authentication against a centralized service provider. Out of two and a half factors of our solution, the first factor stands for the private key, and the second and a half factor stands for OTPs and their precursors (a.k.a., pre-images), where OTPs are obtained from the precursors by cryptographically secure hashing.

We describe the protocol for bootstrapping our approach as well as the authentication procedure. In the case of stolen credentials from the client, we show that our solution enables the user to immediately detect it and proceed to re-initialization with fresh credentials. We utilize blockchain-based identity management and decentralized identities of users to simplify the overhead of the registration process and reinitialization.

I. INTRODUCTION

A potential solution to the password problems (e.g., forgetting, weak passwords) might be resolved by third-party authentication schemes that enable delegation of authentication to service providers such as Facebook, Google, etc. The well-known solution for third-party authentication is Open Authorization (OAuth) [1]. However, even OAuth protocol v2.0 is vulnerable to phishing attacks [2], click-jacking [3], and redirect URL attacks [4]. An alternative to password-based authentication is public key cryptography. However, the malware was developed to steal private keys as well [5]. Misusing any single-factor credentials or the authentication approach by the attacker might have serious consequences. Therefore, as a general rule of thumb, the users utilize two-factor authentication (2FA) solutions.

Our approach is based on SmartOTPs [6], which serves as 2FA for non-custodial smart contract wallets. SmartOTPs enable to enrich public key-based authentication by the second factor with interesting usability properties stemming from short OTPs (i.e., 12 mnemonic words of BIP-39 [7]) and elimination of a shared secret (that is presented in HOTP [8] and TOTP [9]). The 2FA of SmartOTPs is performed in two stages of interaction with the blockchain, requiring two transactions—one for each factor: (1) initialization of the operation with the wallet using the private key and (2) the confirmation of the operation by publishing a corresponding OTP. OTPs are aggregated by a combination of Merkle tree and hash chains (see Fig. 1), and they are iterated in layers from the top to the bottom of hash chains. The root of the Merkle tree can be made public (and thus stored at e.g.,
We consider four entities: the authenticator \( A \), the user (at the client), the service provider, and the smart contract (at the blockchain). We assume that each service provider has its smart contract deployed for our solution and its code is publicly listed [11]. The first factor stands for traditional public-key authentication. The other 1.5 factors are computed from a secret seed \( k \) that is stored at \( A \). A list of \( N \) OTPs is derived together with their precursors from \( k \), denoted as \( O T P s \). OTPs from the penultimate iteration layer of SmartOTPs (i.e., \( O T P s \)) represent the second factor of authentication while revealing the precursors of OTPs stands for another “half” factor of this process. The situation is depicted in Fig. 2. In detail, precursors \( O T P s \) are stored at \( A \), while OTPs (and all nodes of Merkle tree) are stored in the client (e.g., browser). For each user, the service provider initially stores only the public key and the root hash of the Merkle tree aggregating OTPs.

### A. Bootstrapping Phase

The bootstrapping phase of our approach is to establish binding among \( A \), the client, and the service provider. The communicating entities in this phase trust each other (i.e., we assume a secure bootstrapping). The bootstrapping phase is depicted in Fig. 3, and it contains several steps: (1) The user generates the public/private key pair \( PK_U/SK_U \) at her client. (2) The user creates her decentralized identifier (DID) [10] at (external) identity-oriented blockchain and then registers it together with her \( PK_U \) at (external) identity provider, obtaining verifiable credentials. (3) \( A \) generates the secret seed \( k \). (4) The user temporarily transfers \( k \) to the client by an air-gapped transfer – e.g., transcription of mnemonic words [7]. (5) \( N \) precursors \( O T P s \) are generated from \( k \) by a pseudo-random function \( F_k(i) \) for \( i = 1, \ldots, N \) and then all \( O T P s \) are computed from the precursors by cryptographically secure hashing (see Fig. 2). The Merkle tree with its root aggregating all \( O T P s \) is computed at the client. (6) All nodes of the Merkle tree are stored at the client (enabling

\[ F_k(i) \text{ can be made from a secure hash function } h(.) \text{ as } F_k(i) = h(k || i). \]
faster computation of Merkle proofs). (7) Client deletes \( k \) and \( \text{OTPs} \) from the blockchain, only at the service provider. (8) The client sends securely her verifiable credentials (with \( PK_U \) and her DID) together with the root hash of the Merkle tree to the service provider within a signed TLS-encrypted message. (9) Using a PK of a publicly-known identity provider, the service provider verifies the identity of the user from her verifiable credentials. (10) Service provider saves \( PK_U \), DID, and the root hash of the Merkle tree.

**B. Operational Phase**

The authentication protocol is depicted in Fig. 4, and it consists of the following: (1) Client sends the TLS-encrypted signed message containing \( \text{OTP}_i \) and its Merkle proof \( \pi_{\text{OTP}_i} \) to the service provider, where \( i \) is the increment-only counter of successful user sessions, and it matches with the OTP ID in the Merkle tree. (2) Service provider verifies the signature of the user against \( PK_U \) that was extracted from her verifiable credentials. (3) Service provider verifies whether the membership of the \( \text{OTP}_i \) in the user’s Merkle tree using \( \pi_{\text{OTP}_i} \). (4) Service provider verifies whether the previous session ID stored by the service provider has been completely authenticated (thus invalidated) and is equal to \( i - 1 \). If so, the service provider increments the session ID to \( i \). Otherwise, the service provider sends an alert about the attempted misuse of the session to the client. (5) Service provider sends the transaction containing the current and the previous user’s OTPs (i.e., \( \text{OTP}_i \) and \( \text{OTP}_{i-1} \)) to the smart contract. (6) Smart contract checks whether \( \text{OTP}_i \) is not presented in the blockchain of last used OTPs, and reverts with attempted credential misuse event if so (which is listened to by the users). Smart contract deletes \( \text{OTP}_{i-1} \) from its registry and inserts \( \text{OTP}_i \). Therefore, these OTPs and their insertion into the registry cannot be linked to the particular user identity. The only information that is visible to the public is that “someone” tries to authenticate at a particular service provider. Finally, the smart contract returns success together with the transaction ID. (7) Service provider updates \( \text{OTP}_{i-1} := \text{OTP}_i \) in its user data record, which is always appended to the next transaction publishing \( \text{OTP}_i \). (8) Service provider returns the \( \text{OTP}_i \) and transaction to the client. (9) The client checks whether a returned transaction contains correct data (related to \( \text{OTP}_i \)). Then, the client fetches the inclusion proof \( \pi_{\text{inc}} \) of the transaction from the blockchain and uses her light client checks whether the transaction was indeed included in the blockchain. (10) The user enters \( i \) to the authenticator that will display the OTP precursor (i.e., \( \text{OTP}'_i \)) encoded as a mnemonic string or QR code. (11) The user transfers \( \text{OTP}'_i \) from the authenticator to the client. (12) The user sends the TLS-encrypted signed message containing the transaction, its inclusion proof, and \( \text{OTP}'_i \) to the service provider. (13) Service provider verifies the signature of the user and checks whether the session ID \( i \) extracted from the corresponding Merkle proof of \( \text{OTP}'_i \) is a valid session ID. If not, it aborts the process and returns failure. (14) Service provider verifies the precursor of \( \text{OTP}'_i \), by checking equality \( \text{OTP}'_i = h(\text{OTP}'_i) \), representing the verification of the last half factor of 2.5 factors. (15) Service provider verifies the transaction and its inclusion proof \( \pi_{\text{inc}} \). (16) The service provider grants access to the user and marks the Session ID as invalidated.

**C. Attempt of Credential Misuse & Reinitialization**

The service provider and the user can detect an attempt of credentials misuse during the authentication. However, the authentication cannot be finished by the attacker since she does not possess \( \text{OTPs} \). In such a case, the user aborts the authentication process and generates new credentials (and thus secrets). Next, the user ensures that her client and PC do not contain any malware, and then repeats the bootstrapping protocol, which will introduce a new private key and OTPs including their precursors, while the public data linked to these secrets will be updated at the service provider as well. To optimize the re-initialization, our scheme can be bootstrapped with the fresh key pair, which is signed by the original key pair registered at the service provider and identity provider.

**IV. Conclusion**

To enhance the authentication process of centralized services based on public keys, we added 1.5 OTP factors obtained from a modification of SmartOTPs. Our approach splits OTPs into precursors and OTPs that are derived from their precursors by secure hashing. OTPs are provided as the second factor, while their precursors serve as the final half factor. Our approach enables users to instantly detect stolen and misused credentials from the public registry in the blockchain that stores the last used OTP of each user. To optimize cost and performance, it is recommended to use a public permissioned blockchain operated by multiple service providers. Security analysis, performance evaluation, and discussion will be provided in the full version of our paper.
REFERENCES

[1] M. Jones and D. Hardt, “The OAuth 2.0 Authorization Framework: Bearer Token Usage,” Internet Requests for Comments, RFC Editor, RFC 6750, October 2012. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6750.html

[2] C.-J. Chae, K.-B. Kim, and H.-J. Cho, “A study on secure user authentication and authorization in oauth protocol,” Cluster Computing, vol. 22, no. 1, pp. 1991–1999, Jan 2019. [Online]. Available: https://doi.org/10.1007/s10586-017-1119-6

[3] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson, “Clickjacking: Attacks and defenses,” in 21st USENIX Security Symposium (USENIX Security 12). Bellevue, WA: USENIX Association, Aug. 2012, pp. 413–428. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/huang

[4] X. Wang, W. C. Lau, R. Yang, and S. Shi, “Make redirection evil again: Url parser issues in oauth,” in BlackHat Asia 2019. BlackHat Asia, Mar. 2019. [Online]. Available: https://i.blackhat.com/asia-19/Fri-March-29/bh-asia-Wang-Make-Redirection-Evil-Again-wp.pdf

[5] P. Pal, S. K J L, and R. B. S, “Wip: Criminal smart contract for private key theft in end to end encrypted applications,” in Information Systems Security, D. Garg, N. V. N. Kumar, and R. K. Shyamasundar, Eds. Cham: Springer International Publishing, 2019, pp. 21–32.

[6] I. Homoliak, D. Breitenbacher, O. Hujnak, P. Hartel, A. Binder, and P. Szalachowski, “Smartotps: An air-gapped 2-factor authentication for smart-contract wallets,” in Proceedings of the 2nd ACM Conference on Advances in Financial Technologies, ser. AFT ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 145–162. [Online]. Available: https://doi.org/10.1145/3419614.3423257

[7] A. V. S. B. Marek Palatinus, Pavol Rusnak, “BIP 0039,” 2013. [Online]. Available: https://en.bitcoin.it/wiki/BIP_0039

[8] M. View, J. Rydell, M. Pei, and S. Machani, “TOTP: Time-Based One-Time Password Algorithm,” RFC 6238, May 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6238

[9] M. View, J. Rydell, M. Pei, and S. Machani, “TOTP: Time-Based One-Time Password Algorithm,” RFC 6238, May 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6238

[10] D. C. Manu Spony, Dave Longley, “Verifiable credentials data model v1.1,” 2023. [Online]. Available: https://www.w3.org/TR/vc-data-model/

[11] P. Hartel, I. Homoliak, and D. Reijserbergen, “An empirical study into the success of listed smart contracts in ethereum,” IEEE Access, vol. 7, pp. 177 539–177 555, 2019.