Emergency Button: Evacuation of Crypto Asset When Key Loss

Natsume Matsuzaki
Faculty of Information Systems
University of Nagasaki
Nagasaki, Japan
matsuzaki@sun.ac.jp

Masayuki Fukumitsu
Faculty of Information Systems
University of Nagasaki
Nagasaki, Japan
fukumitsu@sun.ac.jp

Yoshihiro Kita
Faculty of Information Systems
University of Nagasaki
Nagasaki, Japan
kita@sun.ac.jp

***Abstract*—We discuss countermeasures against the loss of private keys in crypto assets. Since the private key is necessary to sign transactions when transferring crypto assets, the loss of the private key causes the loss of the corresponding crypto assets. In this paper, we propose a concept and the implementation of "Emergency Button" as a countermeasure against key loss. In our proposed method, a smart contract as "Emergency Button" is generated in advance to evacuate crypto assets corresponding to a private key to another account. When the key is lost, the owner of the crypto asset activates the emergency button and the crypto assets are moved to another account. Owner can use the private key of the other account instead of the lost key. One of the advantages of our proposed method is that the owner can immediately take the action, namely push the emergency button when he or she notices the loss of a key.**

***Index Terms*—crypto asset, key loss, smart contract, Ethereum**

I. Introduction

A. Background

In this paper, we discuss the key management of crypto assets, especially countermeasures when the loss of private keys. As is well known, blockchain, which is the foundational technology for crypto assets, uses public key cryptography which has a pair of private and public keys. Since the private key is necessary to sign transactions when transferring crypto assets. The loss of the private key means the loss of not only it but also the corresponding crypto assets. In other words, the blockchain is a P2P decentralized system, and each user is responsible for managing his/her private keys.

There are two ways to manage private keys and crypto assets: (1) we outsource the management of private keys to crypto exchanges or custodians, or (2) it can be managed by individual users. In case (1), private keys are safely managed by the vendor against leakage or loss of keys. However, these ways incur management costs and also the risk of cyber-attacks targeting the vendor. In fact, the major vendor Coincheck experienced a cyber-attack that resulted in a leak of NEM digital currency in the 2018 [9] [14].

In this paper, we focus on case (2), namely the way to manage own private keys individually. Since each user is responsible for managing the key, it is essential to have a user-friendly key management system that balances ease of use with robust security measures. There are mainly two kinds of key management methods called a wallet below [7] [10].

- Hot wallet : A hot wallet is a type of blockchain wallet that stores keys while connected to the Internet or on a desktop or mobile device. Although it is convenient because the private key can be used immediately when needed, there is a risk of attacks from the Internet.
- Cold wallet : A cold wallet resides in a medium such as a USB memory or a paper that is not connected to the Internet, thereby protecting the private keys from unauthorized access. However, if the private key is stored in a safe, for example, it takes time for the key to become available to use, take it out from the safe, connect it to the Internet, etc. In addition, the more frequently they are taken in and out of the safe, the more likely they are to be lost.

B. Contribution

To resolve the above problems, we propose a secure and user-friendly countermeasure against the key lost. Our method introduces a smart contract as "Emergency button" that evacuates crypto assets corresponding to a private key to another account. When the key is lost, the owner of the crypto asset activates the Emergency Button and the crypto assets are moved to another account. The owner can use the private key of another account instead of the lost key.

C. Related work

The following three methods have been proposed as previous countermeasures against the loss of keys. The first method is to deposit divided keys in the surrounding nodes and when a key is lost, it is recovered by collecting keys from the nodes around it [2] [6] [12]. The private key is divided by using Shamir's secret sharing [15] as one of methods. This method requires the users to prove that they are the original owner of the key when collecting it from its surroundings. Therefore, it takes a lot of time and effort for the user to recover the original keys. In addition, since the transaction of crypto assets is generally desired to be anonymous and not linked to the owner, it is difficult to prove the owner's identity.

The second method is an anonymous tracing method to deter unauthorized use of crypto assets using leaked keys [13]. This

2832-1324/23/$31.00 ©2023 IEEE
DOI 10.1109/CANDARW60564.2023.00048

method is useful when the user manages the private key in a USB memory and loses it. However, with this method, the crypto assets do not return to the original user.

The third method is not to record the private key, but to generate the private key each time using biometrics information and public information [8]. This method is effective not only to prevent key loss, but also to prevent key leakage since there is no recorded key to attack. However, applications using this method are limited, because this method requires on-site biometrics information each time.

II. Preliminaries

a) Blockchain, Smart Contract: A blockchain is a distributed ledger for securely sharing data. Blockchain ensures data integrity through a single trusted source, eliminates data duplication, and enhances security. It also prevents fraud and data tampering because data cannot be changed without the permission of a certain number of participants. Transactions are valuable data in blockchain protocol. The transaction contains "from" account address, "to" account address and "value" of the crypto asset. It also contains the signatures with these data by the private key. A smart contract is a contractual act that is automated using information technology [18]. They are executed according to predefined contracts (rules), triggered by transactions or information outside the blockchain. In the context of blockchain, it can also refer to a program implemented specifically on Ethereum [4].

b) IPFS: One of the problems on blockchain is scalability. Blockchains cannot accommodate a large number of transactions due to the fixed size limit of a single block. IPFS [11] which stands for "Inter Planetary File System" is an efficient solution to this problem, which mitigates the problem by utilizing the distributed storage provided by IPFS. IPFS was developed by the American company Protocol Labs. With the growing attention to Web3, IPFS, a distributed protocol, has been attracting attention recently.

c) ChainLink: Chainlink is a distributed oracle network built on Ethereum, which can be combined with external adapters to realize various functions. We implement the decryption oracle by Chainlink [5] with an external [3].

III. Proposed Scheme

A. Conceptual Model

We outline a concept of our proposed method called "Emergency Button". The purpose of this method is not to recover lost private keys, but rather to recover crypto assets by transferring the associated crypto assets to another account of the same user in the event that the private key is lost. Figure 1 shows a conceptual diagram of our proposed method.

In Figure 1, following symbols are used hereafter:

- A: an original account of a user
- X: crypto asset linked with the account A
- sk: a private key corresponding to the account A
- B: another account of the user
- sk': a private key corresponding to the account B
- SC: a smart contract compared to "Emergency Button"

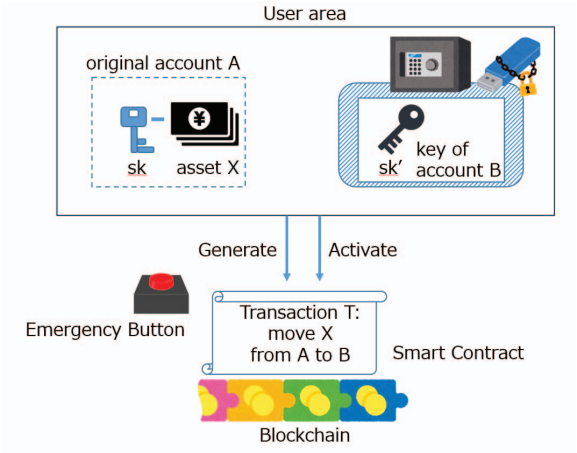

Fig. 1. Concept of out proposal

- T: a transaction that moves crypto asset X from A to B. T is including in SC.

The original account A of a user including private key sk links the crypto asset X. We assume that sk is a key that is managed by a hot wallet such as metamask, and can be used immediately upon access. Therefore, for example, if we forget the password to use sk, you may not be able to access the metamask. We call it "key loss" in this paper.

Before the loss of the private key sk, the user generates another account of which the private key is sk'. sk' is not used on a daily basis, but managed by disconnecting from the Internet and locking a physical device, such as a USB device in a safe. Moreover, the user prepares a smart contract SC as "Emergency Button". That SC includes a pre-generated transaction that moves crypto asset X from A to B. As soon as the user notices the loss of private key sk, the user activates the SC.

B. Basic Construction

Our proposed method has mainly the following four modules.

- Set up account A: $A = \texttt{gen\_account}(sk, X)$, where crypto asset X is linked to the private key sk.
- Create another account B: $B = \texttt{gen\_account}(sk', \text{null})$, where no crypto asset is linked to sk'.
- Generate smart contract SC: $SC = \texttt{gen\_SC}(sk, T : X \text{ from A to B})$, where SC includes transaction T that moves crypto asset X from account A to B and a signature generated using private key sk.
- Activate the smart contract: $\texttt{act\_SC}(SC)$

We describe our proposal in detail. Let us consider a scenario where the private key sk is linked to crypto asset X, namely, $A = \texttt{gen\_account}(sk, X)$, is lost.

1) First, we assume that crypto asset X is linked to account A, which contains the private key sk. The user generates

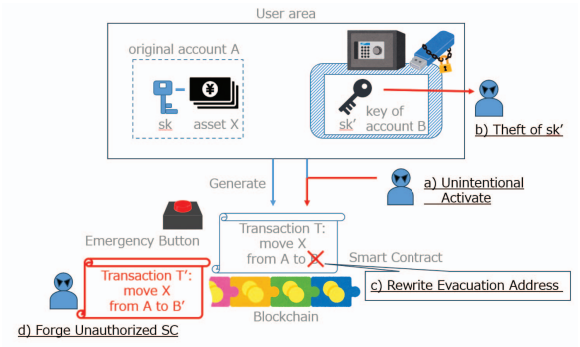

Fig. 2. Thread scenarios

another account B including a private key sk': B = $\texttt{gen\_account}(\text{sk}', \text{null})$. The user should keep sk' in a cold wallet such as USB memory in a safe. Since no crypto assets are linked to the private key sk', storing sk' in the cold wallet does not matter for user convenience.

2) Before the user loses the private key sk, the user prepares a smart contract SC = $\texttt{gen\_SC}(\text{sk}, \text{T} : \text{X from A to B})$ including transaction T that moves crypto assets X linking the original account A to the account B. The transaction T has its signature generated by the private key sk.
3) After recognizing the loss of the key, the user immediately executes the smart contract SC: $\texttt{act\_SC}(\text{SC})$. This is why SC is compared to an "Emergency Button".
4) By executing the smart contract SC, crypto assets that the original private key sk is linking with a move to another account B.
5) After moving the crypto asset X to the account B, the user takes out sk' from the safe. Since the crypto asset X is linked to the new private key sk', the user can use the crypto asset X using sk'.

C. Thread Model

We consider the following four thread scenarios. Figure 2, shows the each thread added to the figure 1.

a) Unintentional Activation: The first scenario is that the attacker activates the SC before the user himself. SC is published in the blockchain and does not require authentication at the time of activation. Therefore, there is the possibility that an attacker activates the SC before the original user needs to do so. The main purpose of the attack is mischief. Since SC is launched over the blockchain, invoking the SC is traced. We believe that tracing the user who invokes the SC is useful against this attack. Although it cannot prevent the SC activation itself, it can serve as a deterrent. The introduction of tracing will require the owner of the crypto asset to be identified, and anonymity will be broken. The owner of the crypto asset will be sought and anonymity will be broken. However, since this is an emergency situation, we will accept it.

b) Theft of sk': The private key sk' can be stored, for example, in a physically offline cold wallet. Therefore, it is assumed that it is hard for the attackers to steal sk'. Even so, we have to consider what to do if an attacker somehow gets sk'. The attacker should authenticate himself/herself when retrieving the private key. This prevents an attacker from illegally obtaining the private key of the safe. However, if an attacker still manages to obtain the private key of the vault, the corresponding SC must be disabled. The deactivating function of SCs must be publicly accessible. One of the possible ways is to use the list of public addresses corresponding to the SCs to be deactivated on the blockchain. The SC can be deactivated by posting the list of public addresses corresponding to the SC to be deactivated on the blockchain. In the case of asset transfers, it is necessary to confirm whether or not the destination address is not included in the list of deactivated SCs.

c) Rewriting Evacuation Address: The third scenario is that an attacker rewrites down the evacuation address to the attacker's account in the SC. It is basically considered to be difficult to rewrite the SC by restricting that the original user can only initialize the SC itself in step 2) of III-B.

d) Forging Unauthorized SC: The last scenario is that an attacker forges an unauthorized SC. We can restrict that the original private key sk is necessary for SC generation. Therefore, it is difficult for an attacker to forge it.

D. Discussion

We discuss the following points in our proposal.

a) Loss of sk' and Start Address of SC: In our proposed method, we need to be careful not to lose the 'sk' and the start address of SC to prevent the loss of sk. Therefore, it remains a risk of these losses. We consider this issue as follows. The private key sk needs to be "used" in such a way that it is "not leak" and "not lost". It is especially difficult to manage the private keys by satisfying the above three conditions because the keys have to be connected to the Internet when using them. On the other hand, sk' is not usually used, it can be stored separately from the Internet. sk' is stored in a safe to keep the secret and prevent it from being lost it. Moreover, countermeasures for activating SC by prank are necessary, but the starting address of SC does not need to be kept secret. SC can only transfer the crypto assets from the original account to another one. Therefore, even if they are leaked, it will only result in the transfer of crypto assets between user-managed accounts. In addition, since the start address of SC is used in case of emergency, it is possible to manage it so that it will "not be lost". It can be considered that the proposed method distributes the risk of loss of sk to the risk of loss of sk and one of loss of the starting address of SC.

b) A signature of transaction: SC contains pre-calculating results of transactions with account address A, B and value X of crypto assets and the signatures using sk. Therefore, each time the user pays a part of crypto asset X, the SC must be re-created, which is time-consuming. To resolve this problem, it is necessary to consider dividing X

into multiple values in advance and applying a signature to each part.

c) Detection of key loss: We are thinking of a scenario in which a user wants to use a crypto asset and then realizes that the key has been lost. Our proposed method is a countermeasure after user's detection of key loss. However, it is also necessary to detect the loss of the key as soon as possible. One way to do this, for example, is to run an automatic program that periodically generates a signature with the private key sk and makes an alert when it fails to do so. These consideration is a future work.

IV. Implementation

A. Points of Implementation

The main features that differ from the method considered for implementation are the following three points.

1) Encrypting and recording the transaction
2) Transaction is recorded as a value-divided LIST
3) Encrypted transaction is stored in IPFS

Each of these features is described below.

a) Encrypting and Recording Transaction: We need to consider how to realize the SC prepared in step 2) on Section III-B. In order to move crypto assets that are linked to the original account A to the evacuated one B, the transactions signed by the account A are required. On the other hand, such transactions cannot be issued when step 4) is executed, since the secret key of A has already been lost. Therefore, the signed transactions should be issued at step 2). Reflecting on this problem, we explore a way that the signed transactions can be retrieved with the help of the smart contract SC. If the transactions are recorded in plaintext, then there is a risk that the account could be identified and exploited by an attacker. The first feature of this implementation is that the signed transactions are encrypted with a key K.

It should be noted that another problem of managing K arises instead of the ones for the original secret key explained in Section I-A. Therefore, our idea to resolve this problem is that K is also encrypted, and then it can be decrypted only when the SC is executed. To realize this idea, we involve Chainlink [5] with an external adapter. Chainlink is a distributed oracle network built on Ethereum, which can be combined with external adapters to realize various functions. Namely, we implement the decryption oracle by Chainlink [5] with an external adapter. Eventually, when the egress SC corresponding to the emergency button is executed, the SC sends the encrypted K to the decryption oracle. Since the invocation is implemented by SC, the execution log is kept, and hence it is possible to deter attacks. Therefore, this feature enables to migrate the thread a) discussed in Section III-C.

b) Transaction is Recorded as Value-divided LIST: To issue a signed transaction, the value should be included in the transaction. Recall that the transaction is prepared at step 2), but it is sent after the key is lost (step 4)). There is the possibility that the balance of the original account A is varied at the time where step 4 is performed. If the balance is less than the value written in the signed transaction, then step 4) is no longer performed.

The second feature of the proposed implementation is that the multiple transactions whose value is divided from the current balance are generated at step 2), and then are listed. A signature is issued for each of the divided values. Here, we can allow that the divided values are different such as 100, 50, and 25. At step 4), the crypto assets are moved multiple times starting with the largest crypto assets. The move process is terminated when the amount of crypto assets to be transferred is less than the Gas fee. This idea also helps to set the nonce to the transactions. For specifications of Ethereum, the total number of transactions generated by the user should be written as the nonce. Recall that the list of transactions is prepared at step 2), but these will be sent to the Ethereum network at step 4). This means that the user should predict the nonce. To overcome the difficulty, sufficiently many transactions whose nonce is different are prepared, and the user can use the transactions whose nonce is appropriate at step 4).

c) Encrypted transaction is stored in IPFS: As described in the second feature, the multiple transactions are recorded as a list. The size of the list is increased by increasing the number of transactions. This means that many Gas fees are required if the list is stored at the SC directly. The third feature is to save gas fees by using distributed storage IPFS [11] to record the transactions list. The recorded address in IPFS is also encrypted to prevent unauthorized deletion of encrypted transactions on IPFS, and then the encrypted address is stored in SC instead of the signed transactions list directly.

B. Concrete Implementation

The operation of the Emergency Button SC is as follows: (1) Emergency Button SC creation phase, (2) Emergency Button SC activation phase, and (3) Emergency Button SC deactivation phase. The operations of phases (1) and (2) are described below using Figures 3 and 4.

The deactivation phase in (3) is used to invalidate the SC execution itself, for example, when the secret key sk' of the save destination has been compromised. We omit the explanation here.

In Figures 3 and 4, A is the current account (client), SC is a smart contract (saved SC). CL is Chainlink with an external adapter and operates as a decryption oracle.

It is recommended to generate the saved SC at the time when account A is created. As an example of front-end operation, specify the original account A and press the generate button. The above operation is performed, and the emergency button addr_EES is displayed. Note that the account A consumes the gas money (Ether) that is required for step 9) in Figure 3.

(1) Emergency Button SC Creation Phase: In this phase, the Emergency Button is created. Suppose that the user A has an account. The sequence diagram in Figure 3 is used to explain the process.

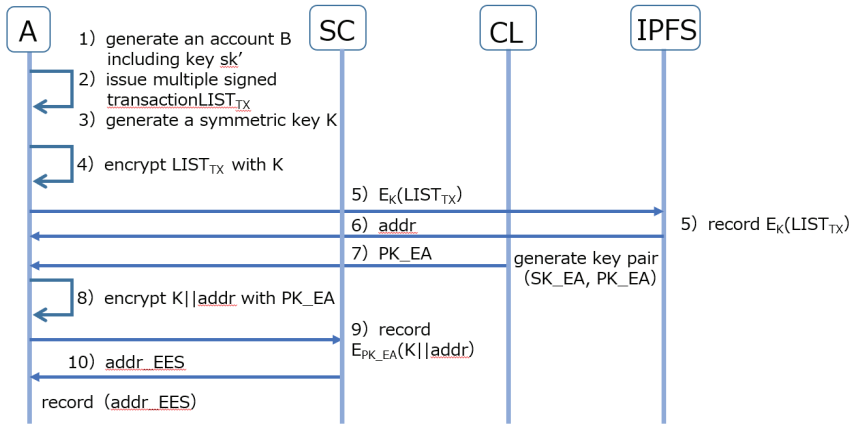

Fig. 3. Operation of SC creation phase

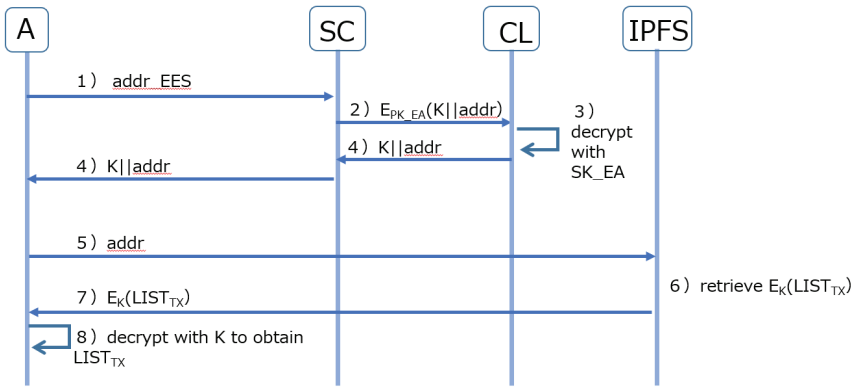

Fig. 4. Operation of SC activation phase

1) A generates an evacuated account B whose secret key is denoted by sk'. Then, sk' is securely stored e.g. in the cold wallet.
2) A issues multiple signed transactions to move the crypto assets to account B, and then these are listed as $\mathrm{LIST_{Tx}}$.
3) A generates a key K to encrypt $\mathrm{LIST_{Tx}}$.
4) A encrypts $\mathrm{LIST_{Tx}}$ with the symmetric encryption scheme such as AES. The ciphertext is denoted by $\mathrm{E_K(LIST_{TX})}$.
5) A sends $\mathrm{E_K(LIST_{TX})}$ to the IPFS, and then IPFS stores $\mathrm{E_K(LIST_{TX})}$ at the address addr.
6) IPFS returns addr to A.
7) A retrieves the public key PK_EA of the external adapter which is for the public key encryption such as RSA-OAEP.
8) A encrypt the concatenation of K and addr with the public key PK_EA. The ciphertext is denoted by $\mathrm{E_{PK\_EA}(K||addr)}$
9) A includes $\mathrm{E_{PK\_EA}(K||addr)}$ to the SC, and then deploys the SC.
10) A receives the address addr_EES of the SC, and then memorise addr_EES.

(2) Emergency Button SC Activation Phase: This is the phase in which assets are saved to the destination account by activating the Emergency Button SC after the user concerned loses his/her private key. The sequence diagram in Figure 4 is used to explain the process.

1) A uses another account to activate the evicted SC by specifying addr_EES, which is an emergency button.
2) The Emergency Button SC sends the encrypted pair $\mathrm{E_{PK\_EA}(K||addr)}$ of the key and the address on IPFS to the external adapter, which is the decryption oracle, via Chainlink.
3) The external adapter decrypts $\mathrm{E_{PK\_EA}(K||addr)}$ with its own secret key, SK_EA.
4) The Emergency Button SC receives the decryption result, the common key K and the IPFS address addr, from Chainlink, and sends them to A.
5) A accesses the IPFS using the address addr.
6) The IPFS retrieves the encrypted list $\mathrm{E_K(LIST_{TX})}$ of transaction.

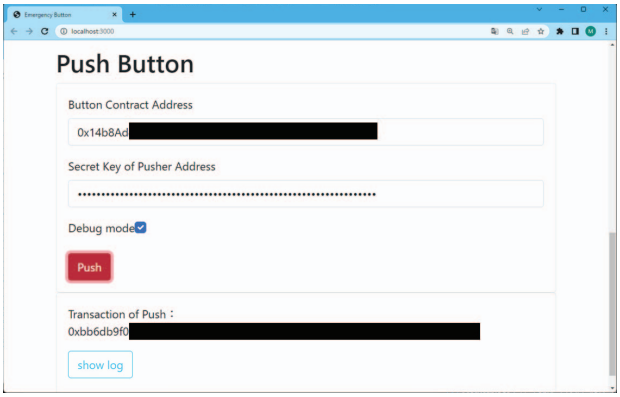

Fig. 5. Screen shot of Front-end

7) A receives the encrypted list $\mathrm{E_K(LIST_{TX})}$ from the IPFS.
8) A decrypts the encrypted LIST using the key K, and then obtains $\mathrm{LIST_{Tx}}$.

An example of a front-end operation is shown in Figure 5. It has the text fields for the private key of A's other account, the one for the emergency button addr_EES, and the start button. When the button is pushed, the above operation is performed, and the specified asset transfer is completed. Note that the Gas fee (Ether) is required in step 1) and, the Ether is required to be transferred to the address for the CL to run the function of the SC in step 4) by the CL. In addition, a Link token is required to use Chainlink.

C. Experimental Evaluation

For the purpose of confirming the operation of the Emergency Button SC and a simple evaluation of its performance, an "Emergency Button" SC was implemented using Solidity. The following PC environment was used for evaluation.

- OS: Windows 11 Pro
- CPU: Intel(R) Core(TM) i7-8565U @ 1.80GHz 1.99GHz
- RAM: 8.0 GB.

Experiments were conducted under the following three environmental conditions.

- Using the Ethereum Goerli testnet
- Connecting to the Goerli network via an API provided by Alchemy.com [1]
- The client (A in the sequence diagram), CL, and EA each run on Docker. The client and EA use the "node" image [17], and CL uses the "smartcontract/chainlink:1.3.0" image [16] [5].

Then, we execute the emergency button SC creation phase and the emergency button SC activation phase explained in the previous section 18 times. Here, 20 transactions whose value is set as 0.001 ETH are created in 2) of the emergency button SC creation phase. Before executing 1) of the emergency button SC activation phase, A pays 0.001 ETH to CL, since the fee of the transaction from CL to SC in 4) is required. Moreover, A pays 0.1 LINK tokens to SC which are required to run CL in 2).

TABLE I
MEASUREMENT RESULTS

step	item	time
steps 1)-5)	average time (msec)	42738.9
	standard deviation (msec)	9880.4
steps 6)-8)	average time (msec)	202.2
	standard deviation (msec)	254.4

step	item	fee
step 1)	mining fee (GoerliEther)	0.0000777
step 4)	mining fee (GoerliEther)	0.0000315
step 1)-5)	Gas (GoerliEther)	0.0002590

Note that each process corresponds to the process number in Figure 4.

Table I shows the measurement results. As shown in Table I, steps 1-5) took much time. The main reason is that communication between A and CL requires the use of the SC with mining.

V. CONCLUSION

As is widely known, the security of crypto assets depends on their key management. In order for crypto assets to be more widely used, secure, easy-to-use, and cost-effective key management is required. In this paper, we focused on the loss of keys, and proposed a concept that uses smart contracts called "Emergency Button". By activating this smart contract when a user loses a key, the crypto assets corresponding to the lost key move automatically to another account of the user. This increases the possibility that the user can take action to recover the crypto asset before the lost key is picked up by others. We experimented with this idea on Ethereum using Chainlink with an external adapter as a decryption oracle. It has the following three features.

- Transactions are encrypted and recorded in advance to prevent misuse by attackers. When the emergency button is activated, the decryption oracle implemented by chainlink and an external adapter is used.
- Transaction values are recorded in multiple parts, so that re-signing is not required when crypto assets are consumed or added after signing.
- The transaction is stored in IPFS to save Gas cost.

ACKNOWLEDGEMENTS

We sincerely appreciate anonymous reviews for insightful comments. This work was supported by JSPS KAKENHI Grant Number JP 20K11815.

REFERENCES

[1] Alchemy Insights, Inc. Alchemy. https://www.alchemy.com/. (Accessed: September 4, 2023).
[2] C. Allen, A. Brock, V. Buterin, J. Callas, D. Dorje, C. Lundkvist, P. Kvavchenko, J. Nelson, D. Reed, M. Sabadello, G. Slepak, N. Thorp, and H. T. Wood. Decentralized public key infrastructure - a white paper from rebooting the web of trust. https://www.weboftrust.info/downloads/dpki.pdf, December 23, 2015.
[3] A. Beniiche. A study of blockchain oracles, 2020.

[4] C. Dannen. *Introducing Ethereum and Solidity Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Springer-Verlag, New York, first edition, 2017.
[5] S. Ellis, A. Juels, and S. Nazarov. Chainlink a decentralized oracle network, March, 2018.
[6] C. Fromknecht, D. Velicanu, and S. Yakoubov. A decentralized public key infrastructure with identity retention. https://eprint.iacr.org/2014/803.pdf, November 11, 2014.
[7] S. He, Q. Wu, X. Luo, Z. Liang, D. Li, H. Feng, H. Zheng, and Y. Li. A social-network-based cryptocurrency wallet-management scheme. *IEEE Access*, 6:7654–7663, 2018.
[8] Y. Kaga, M. Fujio, K. Naganuma, K. Takahashi, T. Murakami, T. Ohki, and M. Nishigaki. A secure and practical signature scheme for blockchain based on biometrics. In J. K. Liu and P. Samarati, editors, *Information Security Practice and Experience*, pages 877–891, Cham, 2017. Springer International Publishing.
[9] K. Kamei. Hacked japan cryptocurrency platform hit with improvement orders. https://asia.nikkei.com/Spotlight/Cryptocurrencies/Hacked-Japan-cryptocurrency-platform-hit-with-improvement-order. (Accessed: September 4, 2023).
[10] O. Pal, B. Alam, V. Thakur, and S. Singh. Key management for blockchain technology. *ICT Express*, 7(1):76–80, 2021.
[11] Protocol Labs. Ipfs. https://ipfs.tech/. (Accessed: September 4, 2023).
[12] G. Ra, C.-H. Roh, and I.-Y. Lee. A key recovery system based on password-protected secret sharing in a permissioned blockchain. *Computers, Materials Continua*, 65:153–170, 01 2020.
[13] T. Sato, K. Emura, T. Fujitani, and K. Omote. An anonymous trust-marking scheme on blockchain systems. https://arxiv.org/abs/2010.00206, 2021.
[14] T. Sato, M. Imamura, and K. Omote. Survey on tracking of leaked nem by coincheck incident. *IEICE Technical Committee Submission System, ISEC*, March, 2018.
[15] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.
[16] smartcontract. smartcontract/chainlink. https://hub.docker.com/r/smartcontract/chainlink. (Accessed: September 4, 2023).
[17] The Node.js Docker Team. node. https://hub.docker.com/_/node/. (Accessed: September 4, 2023).
[18] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.