Efficient Off-Chain Transaction to Avoid Inaccessible Coins in Cryptocurrencies

Hossein Rezaeighaleh
Department of Computer Science
University of Central Florida
Orlando, USA
rezaei@knights.ucf.edu

Cliff C. Zou
Department of Computer Science
University of Central Florida
Orlando, USA
czou@cs.ucf.edu

***Abstract*— Bitcoin and other altcoin cryptocurrencies use the Elliptic-Curve cryptography to control the ownership of coins. A user has one or more private keys to sign a transaction and send coins to others. The user locks her private keys with a password and stores them on a piece of software or a hardware wallet to protect them. A challenge in cryptocurrencies is losing access to private keys by its user, resulting in inaccessible coins. These coins are assigned to addresses which access to their private keys is impossible. Today, about 20 percent of all possible bitcoins are inaccessible and lost forever. A promising solution is the off-chain recovery transaction that aggregates all available coins to send them to an address when the private key is not accessible. Unfortunately, this recovery transaction must be regenerated after all sends and receives, and it is time-consuming to generate on hardware wallets. In this paper, we propose a new mechanism called *lean* recovery transaction to tackle this problem. We make a change in wallet key management to generate the recovery transaction as less frequently as possible. In our design, the wallet generates a lean recovery transaction only when needed and provides better performance, especially for micropayment. We evaluate the regular recovery transaction on two real hardware wallets and implement our proposed mechanism on a hardware wallet. We achieve a %40 percentage of less processing time for generating payment transactions with few numbers of inputs. The performance difference becomes even more significant, with a larger number of inputs.**

Keywords—blockchain, cryptocurrency, off-chain, Bitcoin.

I. INTRODUCTION

Today, a user can perform various electronic commerce transactions like paying a bill, booking a hotel or flight, purchasing online products, and paying taxes with cryptocurrency. While cryptocurrencies become more usable for average users, the inaccessible coins issue arises as a challenging problem in cryptocurrencies. Since, as a design paradigm, only the user's private key can send the coins from its associated address, if the user cannot access her private key, she loses her coins. The user may forget her password, or in a worst-case, she may die, and her coins will be lost forever. It happens in the cryptocurrency ecosystem many times, and as several reports like [1] shows, about 21 percent of all possible bitcoins are out of circulation and maybe are lost forever.

Furthermore, in several cases where the owner of the key dies or pretends to die to steal the coins from others. These persons control other users' coins, like investors in the position of an online cryptocurrency exchange president [2]. Since there is no clear technical solution to recover the lost coins, the investors lost their money.

There are limited choices for users to avoid inaccessible coins, such as creating a backup for another person or using a multi-signature wallet. These solutions not only are inconvenient but also put the user at risk. Recently, authors of [3] suggested generating an off-chain recovery transaction and publishing such a transaction when the coins are inaccessible. The wallet must frequently regenerate this transaction because any change in inputs by a sending transaction invalidates the previously generated recovery transaction. Also, any receiving transaction conveys new coins that should be added to the recovery transaction.

This paper investigates this off-chain recovery transaction and evaluates its performance in real conditions with actual hardware wallets as a secure option for cryptocurrency users. We demonstrate that generating such a recovery transaction consumes a significant amount of time on hardware wallets or other resource-constraint wallets. Hence, it is not a practical solution in real life. We propose a new key management schema to separate frequent micropayments from other transactions and keep the recovery transaction updated with regenerating as less frequently as possible to resolve this performance challenge. Our proposed schema prevents inaccessible coins in most cases and provides better performance compared to the previous method. This paper offers the following research contributions:

- Evaluating off-chain recovery transaction in real hardware wallets
- Proposing a new key management schema to minimize the frequency of regenerating *lean* recovery transactions
- Implementation of proposed lean recovery transaction on a hardware wallet as a proof-of-concept

II. Related Works

There are a few solutions to protect the wallets from being inaccessible but do not thoroughly meet the requirements. We review them here before starting.

A. Wallet Backup

Existing cryptocurrency wallets usually use the paper backup. The wallet generates a mnemonic word list to convert the master seed from digital form to physical form as a backup [4]. The user may either save these words in a computer file or writes them down on a piece of paper. In our previous paper [5] we suggest a new mechanism to back up a wallet on another wallet directly with the elliptic-curve Diffie-Hellman key agreement.

The backup mechanism is not a suitable solution for the inaccessible coins challenge. If the user uses a password for the backup on the paper or another wallet, the coins are inaccessible when she forgets the password or dies. On the other hand, if she gives the backup and password to another person, this person has equal access and can use the coins anytime without the primary user acknowledge.

B. Multi-Signature Wallet

Multi-Signature Wallet is another solution [6] where multiple private keys with a threshold (e.g., two of three) are used to sign a transaction; if one key (or more key) is lost, other keys can recover the coins. Some literature like [7] and [8] advise the users to use multi-signature; however, it has significant drawbacks. Multi-signature requires multiple wallets to sign a transaction, which would cause complexity in the signing procedure. It may be good enough for corporate use-cases. However, it is more than challenging for average individuals.

C. Recovery Transaction

Authors of [3] explain a mechanism to recover inaccessible wallets using an off-chain transaction. Each time that the wallet sends or receives a coin, the wallet creates a recovery transaction to gather all available coins in Unspent Transaction Outputs (UTXO) and saves it on a file. When the user forgets her wallet password or the password became inaccessible because of any reason like death, the wallet *retirement* mechanism activates with a policy like no login for more than six months. The wallet publishes the last recovery transaction and transfers all coins to a reserved address. Since the recovery transaction is signed in-advance, there is no need for private keys.

On the other hand, all received coins in the last six months, aka the retirement period, are lost because these new coins are not included in the recovery transaction. The user can set the retirement period. It is not a timer on the wallet; it is a value embedded into the recovery transaction itself. If the wallet or other entity publishes the recovery transaction on the blockchain, it will not be effective until the pre-defined time. This mechanism works for UTXO-based cryptocurrencies like bitcoin, and the lock time in bitcoin transactions supports this feature.

The recovery transaction that authors of [3] explain is designed for old-fashion software wallets like Satoshi Client [9] that runs on a powerful enough personal computer. However, regenerating a recovery transaction has a significant performance problem in modern wallets like mobile wallets that use Trusted Execution Environment [10] [11] and hardware wallets running on a microcontroller secure element with limited resources. This paper proposes a practical off-chain recovery transaction that avoids inaccessible coins in hardware wallets with a minimum performance penalty. We call it the *lean* recovery transaction.

III. Technical Background

A. Bitcoin Transaction

There are two transaction models in blockchains. Bitcoin uses the Unspent Transaction Output (UTXO) model, and Ethereum uses the Account model. In the UTXO model, a transaction has inputs and outputs. Each input indicates one previous transaction output and its amount. The new transaction spends all amounts of the previous transactions' outputs and moves them into its outputs. The next transaction does the same.

If an output of a transaction is used as an input on another transaction, it is called "spent output". If no transaction uses an output of a transaction, it is called "unspent output". The blockchain nodes only accept a transaction that all of its inputs are unspent; otherwise, it is a double-spending that is not permitted in the blockchain. Hence, this model is called the Unspent Transaction Output model. It is similar to cash circulation in the real world, where the coins move from one person to another. On the other hand, there is another model called Account-based. In this model used in Ethereum, each account has a balance, and each transaction indicates fund transfer from one account to another, similar to the real world's bank accounts.

Bitcoin transactions may have several inputs and outputs; however, they usually follow regular formats. For example, a payment transaction usually has one input and two outputs. The input is a UTXO that the payer has enough coins on it. One of the outputs includes the payee's address, and another output conveys the payer change address to receive the remaining coins of the input after deduction of payee output. On average, 60 percent of all bitcoin transactions are in this format, with one input and two outputs [12].

B. Hierarchical Deterministic Wallet

In the blockchain, a sender signs a transaction with her private key and inserts the receiver's public key into the transaction output. It decreases privacy since everyone who has access to the blockchain network can track a particular user's activities. To avoid tracking, the user can generate a random key for each transaction. Therefore, a potential hacker or investigator cannot link different transactions to gather information about a user. It is already a best practice in bitcoin and many cryptocurrencies [7]. However, it causes another

challenge since the user should manage many keys. The solution is a deterministic pseudorandom algorithm for key generation. The cryptocurrency community developed a multi-level deterministic algorithm called Hierarchical Deterministic Wallet [13]. HD wallet has a key tree, and each node is derived from its parent. The tree's root is called 'master private key' and derived from a random value called 'master seed'. Therefore, with a given master seed, the wallet builds the entire key tree. Consequently, the user only needs to keep the master seed safe and enjoys a brand-new key for each transaction.

BIP-32 is a Bitcoin Improvement Proposal that defines the Hierarchical Deterministic (HD) Wallet [13]. It explains different algorithms to derive a node from its parent in the key tree. This document's core is the master key generation, and two Child Key Derivation (CKD) functions. The master key generation function generates the master key using the HMAC-SHA512 on a 128-bit to a 512-bit random value called the master seed. On the other hand, BIP-44 defines a comprehensive path for all cryptocurrencies' key trees on a wallet using only one master seed [14]. A path in BIP-44 format has the following levels in BIP-32:

path = m/purpose′/coin′/account′/change/address_index

In this path, *m* is the master seed, and the *purpose* is 44 for BIP-44, *coin* is a predefined value for registered coins, for example, 0 for bitcoin. An *account* is a group of funds that helps the user manage her money, such as creating a separate key set for a spending account and a savings account. *Change* element is 0 for external address and 1 for internal address. The external address is a regular address published to others to receive funds. In contrast, an internal address is "change address" for receiving remaining funds from the spending transaction and never published to others. *address_index* is a sequential number that starts from 0 to generate multiple unique addresses.

C. Hardware Wallet

The hardware wallet is a dedicated cryptographic device to generate and store the secret keys and sign the transactions. Since a hardware wallet is not a general-purpose computer, a hacker cannot easily install a malware program. Furthermore, some secure hardware wallets have a *secure element*. It is a tamper-resistant module to protect the secrets from electrical and physical attacks such as side-channel attacks and power-analysis.

Hardware wallets usually have a screen and a few buttons to interact with the user directly; otherwise, they are vulnerable to Man-In-The-Middle attack [15]. Figure 1 depicts the general components of hardware wallets. They usually have a main control unit (MCU) that connects all components and communicates with the host application via USB, Bluetooth, or NFC.

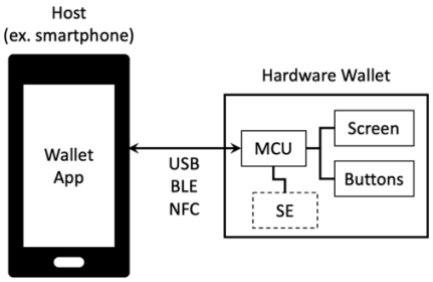

Figure 1. General hardware wallet components

Since a hardware wallet does not have internet access, it uses an app on the host like a personal computer or a smartphone to connect to the blockchain network. However, critical tasks like storing the keys and signing a transaction will be done on the hardware wallet. The overall procedure of signing a transaction on a hardware wallet is as follows.

Transaction Signing Process on a Hardware Wallet:

1. Host App: Gather information from blockchain nodes and prepare inputs and outputs.
2. Hardware Wallet: Receive data and display the receiving addresses, amount, and fee of the transaction on the embedded screen and get the user confirmation by pressing an embedded button.
3. Hardware Wallet: Derive required keys, sign the transaction for each input, and return the result to the host app.
4. Host App: Publish the signed transaction to the blockchain nodes.

While the network connection is good, and the host has enough resources, the time-consuming steps are step 2 and step 3 that run on the hardware wallet. A transaction with more input UTXOs takes more time on the hardware wallet for key derivation and digital signature.

IV. Experiment

This section conducts some experiments to evaluate the recovery transaction suggested in [3] with real hardware wallets. We illustrate that the recovery transaction is a heavy-loaded transaction to generate. We show that creating a brand-new recovery transaction for all sendings and receivings has a significant performance penalty, making it impractical in resource-constraint cryptocurrency wallets like hardware wallets.

In contrast to payment transactions, recovery transaction has several inputs and only one output. It aggregates entire available UTXOs to transfer all coins to the reserved address. Multiple inputs make the recovery transaction larger than a typical payment transaction. A recovery transaction needs several key derivations to calculate required private keys for all UTXOs and several ECC signings to generate outputs. Even though a recovery transaction is not very different from a payment transaction for traditional software wallets like Satoshi Client [9] that runs on a computer, it has a significant

performance penalty on a resource-constraint device Hardware Wallets.

Since bitcoin is the gold standard in UTXO-based cryptocurrencies and many other coins copy the entire or parts of its codebase, we choose bitcoin to do our measurement. We also choose the Segregated Witness protocol, aka SegWit, to perform our tests. It is a new version of the bitcoin protocol [16] with better performance for multiple inputs. To employ SegWit protocol, we use the following path for key derivation:

path = m/49′/1′/0′/change/address_index

The number 49 refers to BIP-49 [17] that defines the derivation scheme for SegWit addresses. Next, number 1 is the defined constant for bitcoin testnet. To compare recovery transactions with typical payment transactions, we use the typical payment transaction format with one input and two outputs. The recovery transaction has one to ten inputs for available UTXOs and one output for the reserved address. It may have more than ten inputs in real life, but we assume this number just for demonstration. We use WireShark to monitor USB packets and measure the timing [18]. Our tests were executed on a MacBook Pro with Intel Core i7 2.2 GHz processor and 16 GB memory, and we use the same USB port for all tests.

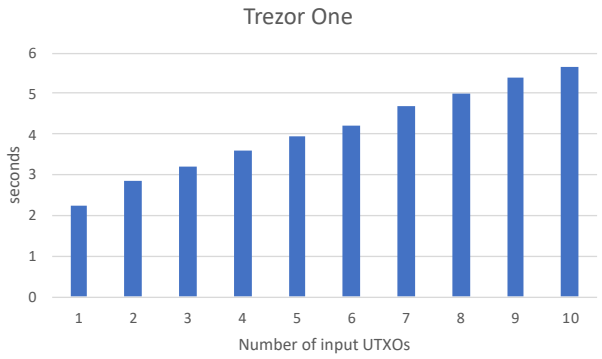

Figure 2. Performance of generating recovery transaction on a Trezor One hardware wallet

To evaluate generating a recovery transaction on hardware wallets, we only measure step 2 and step 3 of the *Transaction Signing Process on the Hardware Wallet* because step 1 and step 4 are executed on the host application and network. Figure 2 and Figure 3 demonstrate the results for two hardware wallets [19] [20]. Increasing the number of input UTXOs takes more time on the wallet to generate a recovery transaction. In comparison, Ledger Nano S has lower performance because it uses a secure element [21]. In the worst-case scenario, generating a recovery transaction on a secure hardware wallet like Ledger Nano S takes around 40 seconds, with only ten input UTXOs.

Authors of [3] discussed that the wallet must create a recovery transaction after all sending transactions because one or more input UTXOs is spent. Spending invalidates the previous recovery transaction because at least one of its input UTXOs is not available. In other words, the wallet has to generate a brand-new recovery transaction after even a micropayment transaction like buying a coffee, purchasing a ticket, or paying a bill.

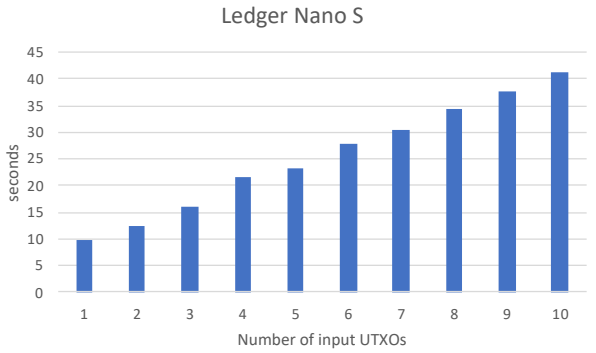

Figure 3. Performance of generating recovery transaction on a Ledger Nano S hardware wallet

V. Proposed Lean Recovery Transaction

As we explained, inaccessible coins are a big challenge in cryptocurrencies. The recovery transaction proposed in [3] to generate two transactions for each payment and save one of them off-chain for disaster recovery has a significant performance penalty in reality.

In this section, we propose a more efficient solution called *lean* recovery transaction. In this solution, the wallet generates the recovery transaction less frequently and only when needed. To do that, we make a change in wallet key management and divide the key tree into two sections. One section is assigned to a spending account, and the other section includes other accounts. The path is as follows when the account is 0 for spending account and non-zero for non-spending accounts.

path=m/purpose′/coin′/account(0|n)′/change/addr_index

Figure 4 illustrates a sample key tree. The wallet uses only the spending section for all spendings (micropayments). It creates the off-chain recovery transaction only for the non-spending section, which means all addresses except under the spending account. We call it the *lean* recovery transaction because it does not include the massive part of a recovery transaction, including many but small amounts of UTXOs.

Only sending and receiving for addresses out of the spending account requires regenerating a recovery transaction, like buying new bitcoins or getting paid for salary with cryptocurrency. So, micropayments do not change the existing lean recovery transaction inputs, and only large payments need a new one.

In another scenario, for receiving transactions, a new received UTXO must be added to the recovery transaction to avoid potential inaccessibility of it. To prevent from regenerating a recovery transaction for all receives even small transactions, we define a threshold that can be changed by the user. If the sum of receiving coins reaches the threshold, the wallet generates a new recovery transaction to add the new UTXOs.

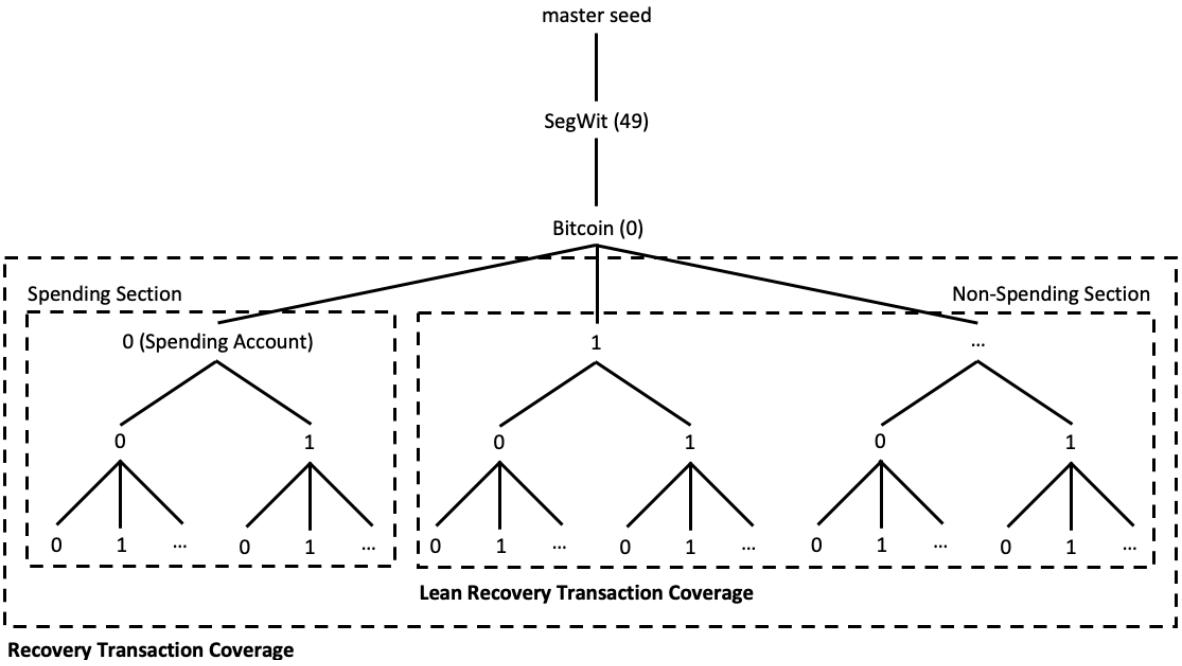

Figure 4. Sample key tree to illustrate the coverages of Recovery Transaction and our proposed Lean Recovery Transaction

The spending account does not receive any coins from outside. So, we define a new *transfer* function, where the user transfers coins from other accounts to the spending account or, in other words, from the non-spending section to the spending section. After creating a transfer transaction, the wallet generates a new recovery transaction because its inputs have been changed.

Our proposed schema has the following advantages in comparison to the recovery transaction proposed in [3]:

- Generating a lean recovery transaction takes considerably less time on the wallet because it has fewer input UTXOs, crucial to hardware wallets.
- The wallet generates the lean recovery transaction less frequently because spending from the spending account does not change the input UTXOs of the existing lean recovery transaction and does not invalidate it.
- Everyday payment transactions for micropayments are faster in our proposed mechanism because they do not need to generate a recovery transaction.
- The wallet adds new receiving UTXOs into a lean recovery transaction only when their total funds reaches a defined threshold, and it makes generating recovery transactions less frequent.

To help a reader understanding our proposed lean recovery transaction mechanism, we use an example to illustrate and compare the recovery transaction in [3] and our proposed method. Assume that a user has a bitcoin wallet with a $7000 value that conveys three UTXOs with $500, $2500, and $4000 equivalent bitcoin.

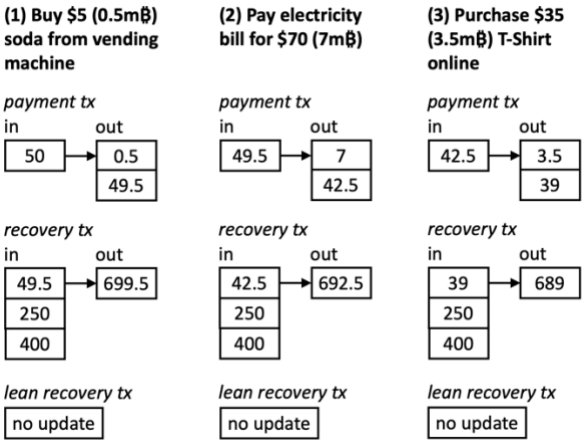

Figure 5. Example of comparing lean recovery transaction with recovery transaction

Suppose the user makes three regular payments today to buy a $5 soda, pay an electricity bill for $70, and purchase a $35 T-Shirt from an online store. She uses her wallet to make these payments by bitcoin. Figure 5 illustrates sample bitcoin transactions that the wallet generates. We ignore purchase taxes and fees, bitcoin exchange fees, and bitcoin network fees to simplify the example. We assume the bitcoin price is $10,000, and we use milli-bitcoin (mBTC) in our sample.

We assume that before beginning, the wallet has generated a valid recovery transaction. In the first scenario, the wallet uses the recovery transaction described in [3], including all three UTXOs with 50, 250, and 400 mBTC. In the second scenario, the wallet uses our proposed lean recovery transaction, including only

two UTXOs with 250 and 400 mBTC, and assigns one UTXO with 50 mBTC to the spending account.

This example demonstrates that each payment in the first scenario includes generating a payment transaction and a recovery transaction. In contrast, it includes generating only a payment transaction without any recovery transaction in the second scenario.

In our test setup described in Section IV, Trezor One [19] hardware wallet takes 2.2 seconds for a payment transaction and 3.2 seconds for a recovery transaction with three UTXOs. Ledger Nano S takes 9.7 seconds and 15.8 seconds, respectively. The payment process takes 5.4 seconds for Trezor One and 25.5 seconds for Ledger Nano S in scenario one. In comparison, it uses 2.2 seconds for Trezor One and 9.7 seconds for Ledger Nano S in scenario two when using the lean recovery transaction. Therefore, the lean recovery transaction has a significant advantage, at least 40 percent of less processing time for generating payment transactions with three input UTXOs. The performance difference becomes even more meaningful with a larger number of UTXOs in the wallet.

VI. Proof-of-Concept

To evaluate the lean recovery transaction model, we implement a hardware wallet from scratch that supports fundamental functionalities of hierarchical deterministic wallets, according to BIP-32 [13] and BIP-44 [14]. We use a secure element for key operations such as key generation and digital signature.

We choose a device that has essential parts of a secure hardware wallet. It has a secure element for cryptography operations and key storage, a screen to display sensitive information to the user, and a button to get confirmation from the user. Figure 6 demonstrates a picture of our test device. This device is in credit card size and has NFC and contact interfaces to communicate.

Since the secure element is a resource-constraint device with limited memory and processing ability, our code must use the minimum memory amount. We use the sharing memory technique and allocate the entire memory to only two arrays. We pass these arrays with the maintained indexes to the functions that require arrays, minimizing the heap consumption.

Furthermore, we do not use a very nested function and any recursive call, minimizing stack memory usage. We use the Java Card framework [22] to program the secure element. It is a limited version of Java Virtual Machine with fewer features to run on microcontrollers and secure elements. We compile the code with the Java Development Kit, convert it to a Card Application (CAP), and load it into the secure element.

One of our implementation challenges is the public key derivation. A public key calculates by multiplying the private key and the Generator point (G) [23] in ECC.

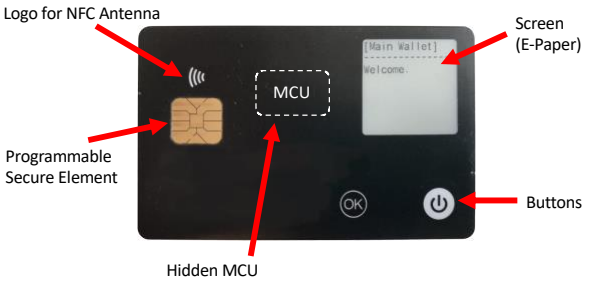

Figure 6. Test device with secure element, screen, and button to create a hardware wallet

Unfortunately, our secure element (and many others) does not support EEC multiplication, and its software implementation has no acceptable performance due to the limited resources of the secure element. However, Java Card API and our secure element support Elliptic-Curve Diffie-Hellman (ECDH) key agreement. In ECDH, each party calculates a secret by multiplying its private key and the other party public key. An ECC public key is an EC point. Therefore, the ECDH function mathematically is multiplying a scaler and an EC point. We use the ECDH function with the private key as the scaler and the Generator point (G) as the EC point. Thus, the result of ECDH will be the public key.

VII. Evaluation

As discussed, our proposed lean recovery transaction has several advantages compared to the recovery transaction explained in [3] because it generates lighter recovery transactions with less input UTXOs. It reduces the number of generating recovery transactions by assigning a section in the key tree to spending and defining a threshold for adding receiving funds to the recovery transaction. In this section, we measure our proposed lean recovery transaction's performance on our implemented proof-of-concept wallet with a secure element.

We test our implementation with two payment scenarios. In the first scenario, the wallet uses the recovery transaction proposed in [3] and generates a recovery transaction just after the payment transaction. In the second scenario, the wallet uses the lean recovery transaction mechanism, and it does not generate a recovery transaction for payment transactions. Figure 7 illustrates our tests' results in both scenarios for various recovery transaction sizes with one to ten input UTXOs.

Since the lean recovery transaction schema does not require regenerating a recovery transaction after each micropayment, the payment transaction performance does not change in Figure 7. On the other hand, the regular recovery transaction makes double the payment transaction time on the wallet, and more input UTXOs increase its generating time.

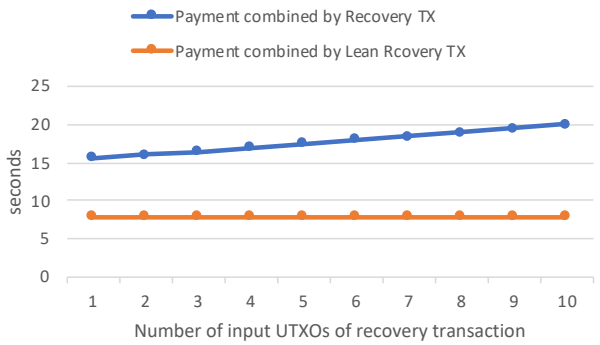

Figure 7. Comparison of micropayment transactions in recovery transaction proposed in [3] and our proposed lean recovery transaction schemas

Our tests have been executed on a MacBook Pro with Intel Core i7 2.2 GHz processor and 16 GB memory, and we use the same USB port for all tests, which is similar to our tests in Section IV.

VIII. Conclusion

This paper proposed a new mechanism called lean recovery transaction to avoid inaccessible coins and achieve optimum performance to generate off-chain recovery transactions. We examined the previous recovery transaction on real hardware wallets to show its significant performance penalty. We proposed separating spending account from other accounts in the wallet key management and define a threshold for adding new received UTXOs into the recovery transaction.

Acknowledgment

The National Science Foundation supported this work under grant DGE-1915780 and DGE-1723587.

References

[1] B. Brown, "21% of Bitcoin Hasn't Moved for Five Years, Stroking Monumental Supply Shock," 24 July 2019. [Online]. Available: https://www.ccn.com/21-of-bitcoin-hasnt-moved-for-five-years-stroking-monumental-supply-shock/.

[2] N. De, "Troubled Canadian crypto exchange QuadrigaCX owes its customers $190 million and cannot access most of the funds, according to a court filing obtained by CoinDesk," 1 Feb 2019. [Online]. Available: https://www.coindesk.com/quadriga-creditor-protection-filing.

[3] P. Rakdej, N. Janpitak, M. Warasart and W. Lilakiatsakun, "Coin Recovery from Inaccessible Cryptocurrency Wallet Using Unspent Transaction Output," in *2019 4th International Conference on Information Technology (InCIT)*, Bangkok, Thailand, 2019.

[4] M. Palatinus, P. Rusnak, A. Voisine and S. Bowe, "Mnemonic code for generating deterministic keys," 2013. [Online]. Available: https://en.bitcoin.it/wiki/BIP_0039.

[5] H. Rezaeighaleh and C. Zou, "New Secure Approach to Backup Cryptocurrency Wallets," in *The IEEE 2019 Global Communications Conference (GLOBECOM-2019)*, Hawaii, US, 2019.

[6] "Multisignature," [Online]. Available: https://en.bitcoin.it/wiki/Multisignature.

[7] A. Narayanan, J. Bonneau, E. Felten, A. Miller and S. Goldfeder, Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction, Princeton, New Jersey, USA: Princeton University Press, 2016.

[8] S. Meiklejohn, "Top Ten Obstacles along Distributed Ledgers Path to Adoption," *IEEE Security & Privacy*, vol. 16, no. 4, pp. 13 - 19, 2018.

[9] "Original Bitcoin client," [Online]. Available: https://en.bitcoin.it/wiki/Original_Bitcoin_client.

[10] M. Gentilal, P. Martins and a. L. Sousa, "TrustZone-backed bitcoin wallet," in *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems*, 2017.

[11] W. Dai, J. Deng, Q. Wang, C. Cui, D. Zou and a. H. Jin, "SBLWT: A secure blockchain lightweight wallet based on Trustzone," *IEEE Access*, vol. 6, pp. 40638-40648, 2018.

[12] transactionfee.info, "One-Input and two-Output Transactions," transactionfee.info, 2019. [Online]. Available: https://transactionfee.info/charts/transactions-1in-2out/?avg=1&start=2019-01-01&end=2019-12-31.

[13] P. Wuille, "Hierarchical Deterministic Wallets," 2012. [Online]. Available: https://en.bitcoin.it/wiki/BIP_0032.

[14] M. Palatinus and P. Rusnak, "Multi-Account Hierarchy for Deterministic Wallets," 2014. [Online]. Available: https://en.bitcoin.it/wiki/BIP_0044.

[15] H. Rezaeighaleh, R. Laurens and C. C. Zou, "Secure Smart Card Signing with Time-based Digital Signature," in *2018 International Conference on Computing, Networking and Communications (ICNC)*, Maui, HI, USA, 2018.

[16] "Segregated Witness," [Online]. Available: https://en.bitcoin.it/wiki/Segregated_Witness.

[17] D. Weigl, "Derivation scheme for P2WPKH-nested-in-P2SH based accounts," 2016. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0049.mediawiki.

[18] G. Harris, "USB capture setup," 2019. [Online]. Available: https://wiki.wireshark.org/CaptureSetup/USB.

[19] "Trezor One," Trezor, [Online]. Available: https://shop.trezor.io/product/trezor-one-white.

[20] "Ledger Nano S," Ledger, [Online]. Available: https://shop.ledger.com/products/ledger-nano-s.

[21] "Frequently asked questions - Ledger Support," [Online]. Available: https://support.ledger.com/hc/en-us/articles/360015216913-Frequently-asked-questions.

[22] Oracle, "Java Card 3 Platform Runtime Environment Specification, Classic Edition Version 3.0.5," 2015.

[23] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters," 2000.