Knox Proposal

Andres Montoya, Nathan Ostrowski[1]
[1]*CPS 590, Duke University*

I. INTRODUCTION

On current blockchain systems, a user's private key is their single point of failure. However, this private key is also necessary for every transaction a user wishes to make. For users prioritizing security, we believe this paradox of a private key providing the root of function and the root of security is a fundamental design flaw. Because the user must keep their private key readily available for every transaction, an attacker may more easily steal it by socially engineering the user or hacking the user's machine. In practice, this accounts for the majority of cryptocurrency stolen each year [Su19]. We propose a system called Knox to protect against this threat model.

II. RELATED WORK

Dodis et al. consider a system that defends against private key compromise by continually reissuing new "secret keys" where each is only valid for some interval i. At initialization, a user registers some Public Key (PK) and some master Secret Key (SK_*) and stores the latter on a trusted, secure device. The PK used to encrypt messages does not change, but encodes the interval during which it was encrypted. When a user wishes to decrypt a message, the user must request an SK_i from the device containing SK_*. Therefore, if an attacker gains SK_i but not SK_*, the attacker cannot read messages from the next interval forward. The authors also consider a system where there is no such trusted, secure device—and the user must manually generate PK and SK, publish PK, and derive SK_* and SK_0. The user may send SK_* to an untrusted device, store SK_0 personally, then when reading, request a partial key SK_i' from the untrusted device and use SK_{i-1} and SK_i' to compute the actual key SK_i [Dod+02].

Xu et al. propose and formally evaluate a system formed around Key Compromise-Resilient Signatures (KCRS). Similar to the system proposed by Dodis et al., there exists a master public/private key pair (pk_M, sk_M) which, when forming a block to place on the chain, one supplies to *SPKGen* and *SSKGen* functions with a random number to receive a randomly selected signing key pair (pk_S, sk_S). The user signs their message using the signing key pair and a function *SigGen*, which a miner can verify matches the signing public key and message using a function *SigVerify*. Importantly, the miner may also verify using the *SKDetect* function that the signing public key pk_S was generated using the master key pair (pk_M, sk_M). Because the signing key pair is only used for a single message, and the random number supplied to *SSKGen* is kept secret by the user, even if an attacker finds sk_S, further messages signed using this key will be rejected by miners on the chain. The authors of this proposal do not see how this method reduces the use of the master private key, as the user must supply their master private key sk_M every time they wish to generate a new key pair. Rather, this system seems to defend against only the situation where the attacker compromises a signing private key sk_S, which the user uses only once and then discards [Xu+20].

Pal et al. proposed an innovative key management solution to protect against private key compromise. Specifically, they describe the security vulnerabilities relating to using block chain within IoT systems and proposed a Group Key Management (GKM) system to alleviate the burden. This ensures the security of payloads containing crypto keys within network communications for blockchain. GKM architecture divides a blockchain network into several groups and levels. Each group has its own group key (GK) and only nodes within that group have permission to access data within it. In code the framework is represented by (i, j, k) where i is the level, j is the position of parent group in the upper layer, and k is the position of the group in the current layer under the parent group.

III. PROPOSED SOLUTION

A. Overview

The goal of Knox is to provide a coin that maintains security for a user's funds even when the user's private key is stolen. Knox implements three high-level, fundamental changes to conventional blockchain systems to achieve this goal:

- Users specify security codes to associate with their account upon initialization
- Transactions are delayed
- A user may exchange their public/private key pair using their security codes

At present, we believe properly implementing this goal will require creating a new altcoin. We propose to build this coin off of Ethereum using Solidity (see Scope).

B. Security Codes and Private Keys

At the initialization of the account, the user commits to a set of security codes by specifying an array of public keys. These keys correlate to the initial private key and the security codes (which would serve as private keys for the user in the future). Thus, whenever the user transfers ownership to a new private key (in the event of private key compromise), a miner may prove the validity of the following private key by referring to the initialization block. The security codes themselves will be stored by the user. We intend to explore optimal methods for securely doing so.

C. Threat Model

Knox's primary goal is to protect users from the most common blockchain attack vector: private key compromise. We consider an adversary A_S whose goal is to compromise the user's private key and extract the funds associated with the user. We also consider an adversary A_E whose goal is to extort the user, seeking complete control over a user's account (even without the ability to extract funds) and demanding some ransom to relinquish control. We consider Routing, Sybil, and 51 percent attacks which an adversary may also use to extract the funds associated with a user to be out of scope. We assume that the system is secure at initialization; cases where A_S or A_E has compromised a user's computer prior to setup are out of scope. Our threat model assumes that the user will monitor their account at regular intervals, noticing (and cancelling) transactions within a set period of time. Our proposed system acknowledges the following attack vectors:

The worst-case scenario occurs when A_S successfully obtains access to the private key and *all* security codes before the user issues a call to rehash their key. In this case, A_S obtains the same privileges as the user. Unless A_S is able to hide these codes from the user, both A_S and the user will have the same

The worst case scenario would be when they rehash the private key and remove access to the user's security codes. The threat actor would then have higher privileges than the user and have full control of the account funds. If a threat actor has access to the private key, they can "hold funds hostage" by blocking all outgoing transfer requests to secure accounts

- As long as the threat actor does not have the same privileges as the user the private key can be rehashed and the threat actor would lose transfer privileges

If threat actor has access to the private key, they can "fee siphon" by continuously creating transaction requests to move the users funds to the threat actor's accounts

- It is important to note that the inbuilt transaction rate-limiter within Knox will prevent "fee siphoning" from occurring indefinitely
- The user would need to manually cancel all fraudulent transactions. This research may explore methods by which the user may "lock" their wallet to automatically cancel any transactions sent to the blockchain.
- "Fee siphoning" would only occur until private key is rehashed

The private key may be obtained through phishing or hacking the user's computer.

If threat actor had compromised the computer the private key was contained in, they would be able to remove the user's access to their private key granting the threat actor the highest level of privilege and control of the account

- This could be mitigated through distributed key storage across devices through wallets
- Furthermore, the user could use one of their security codes to exchange public/private keys.

If a threat actor puts all of the user's funds in a transaction, the user would not be able to cover the gas fees to cancel the transaction

- This issue is handled by ensuring that an address contains the funds necessary to cancel a transaction before a transaction is requested.

It is important to note that the attack vectors which lead to fund compromise require security codes as well as private keys to be compromised. As each key exchange requires a security code, the individual in control of the most security codes including the current private key will be able to do the most key exchanges. This means that the threat actor must control more security codes than the user to supersede the user's privileges. However, the expected extent of security code compromises are vastly lower.

A threat actor could read security codes (r) or wipe and read security codes (wr); let the total number of security codes be n and let h be the total number of security codes the user has. In order for the user to maintain control of the account they would need to have access to more security codes than the attacker. This is because it takes one security code per key exchange. Thus as long as $h > r$ or $n - wr/r <$ (n - wr the account will not be compromised.

Knox expects the range of events where security codes are compromised to be between [0-1]; security codes are stored in separate systems and are never exposed within transactions. The range of compromises expected to occur are between [1-2]; threat actors are likely to attempt a transaction which would immediately notify the user of compromise (or other aspects of their system would become compromised alerting them of security breaches). Thus the range of security codes which a threat actor can be reasonably expected to compromise is [0-4].

D. Delaying Transactions

In order to delay transactions and give a user time to see an unauthorized transaction, we propose indirecting transactions and wrapping them in a cancellable smart contract. To break down the process, let's take a user, Jane, who wants to execute a transaction. Jane makes an API request to Knox software with the details of her intended transaction, then the software formats the intended transaction into a basic smart contract (e.g. a piece of code that will execute this transaction), then wraps this basic contract in a more robust Knox contract. The wrapping Knox contract contains five additional pieces of code: an id, a delay value, a function to check for cancellation, and two balance safety functions (outlined below). A Knox miner executing this wrapping contract will only execute after the delay has passed. When it has, the miner executes the cancellation check function, which verifies that no cancellation block that identifies this transaction (signed with the initiating user's private key) has been added to the blockchain. The format of this Knox contract does not change across users or transactions. If an attacker steals Jane's private key and attempts to add a transaction which does not attach the Knox contract, the attacker's transaction will not be verified by Knox miners, and will fail.

Wrapping transaction code in Knox contracts—which may be cancelled by adding a cancellation block to the chain—introduces two new attack vectors against which the Knox contract must protect. First, because adding a block to the chain necessarily incurs some gas fees, if an attacker wished to transact using all of Jane's coin, Jane would not have the funds left to cancel the attacker's transaction. Thus, the Knox contract's first balance safety function verifies that X coin still remains in Jane's wallet, where X is some margin of likely gas fee. The second balance safety function verifies that Jane also has the necessary gas fees to attach a rehash contract to her cancellation block and render the attacker's compromised private key useless.

IV. LIMITATIONS

By necessarily delaying transactions, Knox increases security by sacrificing immediacy. A user cannot immediately send funds, nor can a user immediately receive funds. Additionally, the model of smart contracts wrapped around every transaction and every change of ownership (where those contracts are included on the blockchain) necessarily incurs gas fees. For smaller transactions, this would perhaps be untenable. For larger transactions, it may not be an issue.

V. IMMEDIATE RESEARCH QUESTIONS

In further developing this system, we intend to explore the following questions:

- Is it possible for a third party to validate a private key hashed from a security code only knowing the hashed version of the security code?
- Is it possible to reduce transaction times from the current proposed system while preventing fraudulent transactions?

VI. SCOPE

Completely creating and implementing an altcoin on its own blockchain would be a burdensome undertaking, and thus is out of scope for this class. Instead, we wish to:

- Develop a whitepaper that more formally outlines our system and its implementation. And, in doing so, answer the above immediate research questions.
- Create an initial version of a Fort ledger on top of Ethereum that operates correctly in a local environment.
- Correct operation entails implementing standard functionality of Ethereum along with the three changes that Knox requires
- Operation within a local environment means the ledger works within a self contained program without any network calls. This would essentially serve as a proof of concept.

REFERENCES

[Dod+02] Yevgeniy Dodis et al. "Key-Insulated Public Key Cryptosystems". In: *Advances in Cryptology — EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 65–82. ISBN: 978-3-540-46035-0.

[Su19] Jeb Su. "Hackers Stole over 4 Billion from Crypto Crimes in 2019 so Far, up from 1.7 Billion in all of 2018." In: *Forbes Magazine* (2019).

[Xu+20] Lei Xu et al. "KCRS: A Blockchain-Based Key Compromise Resilient Signature System". In: *Blockchain and Trustworthy Systems*. Ed. by Zibin Zheng et al. Singapore: Springer Singapore, 2020, pp. 226–239. ISBN: 978-981-15-2777-7.