We've found an error in our paper. In the threshold signature scheme that we used, there are restrictions on the threshold value. In particular if the key is shared over a degree t polynomial, then $2t+1$ players (not $t+1$) are required to to construct a signature. We've laid out how to fix it in this blog post and will be updating the paper accordingly.

Securing Bitcoin wallets via threshold signatures

Steven Goldfeder
stevenag@cs.princeton.edu

Joseph Bonneau
jbonneau@princeton.edu

Edward W. Felten
felten@cs.princeton.edu

Joshua A. Kroll
kroll@cs.princeton.edu

Arvind Narayanan
arvindn@cs.princeton.edu

ABSTRACT
The Bitcoin ecosystem has suffered frequent thefts and losses affecting both businesses and individuals. The insider threat faced by a business is particularly serious. Due to the irreversibility, automation, and pseudonymity of transactions, Bitcoin currently lacks support for the sophisticated internal control systems deployed by modern businesses to deter fraud.

We seek to bridge this gap. We show that a threshold-signature scheme compatible with Bitcoin's ECDSA signatures can be used to enforce complex yet useful security policies including: (1) shared control of a wallet, (2) secure bookkeeping, a Bitcoin-specific form of accountability, (3) secure delegation of authority, and (4) two-factor security for personal wallets.

1. INTRODUCTION
Businesses that need to protect their assets from insider threats must implement appropriate security measures. For those businesses holding significant assets in Bitcoin and other cryptocurrencies, it is important to translate existing internal control paradigms to the Bitcoin world. Although similar in functionality, Bitcoin transactions differ from traditional banking transactions in fundamental ways:

- *Irreversibility.* Once a Bitcoin transaction has been broadcast and confirmed in the block chain, it is generally irreversible even if later shown to be fraudulent (e.g., a stolen private key was used).

- *Automation.* Banking transactions beyond a certain size typically require human action. Bitcoin transaction of any size can be fully automated, authorized only with a cryptographic signature.

- *Pseudonymity.* If traditional corporate assets are fraudulently transferred, banks may be able to (or be legally obligated to) assist in identifying the receiving account owner. Bitcoin addresses are not required to be linked to an offline identity.

These problems pose a daunting challenge for organizations considering doing business in Bitcoin, making internal controls both more important (due to irreversibility and pseudonymity) and harder to implement (due to automation). These problems are particularly acute for *hot wallets*, wherein the private keys are stored on online devices. Businesses that actively transact in Bitcoin must necessarily keep some of their balance in hot wallets.

However, Bitcoin also presents a compelling opportunity as controls which have traditionally been slow and required a human in the loop can now be specified and enforced cryptographically. In this work, we explore such possibilities. Our starting point is to investigate ways to achieve *joint control of bitcoins*, i.e., require multiple designated participants to sign a transaction before it will be considered valid.[1] Joint control mitigates the risk of internal fraud as no employee has the ability to single-handedly misappropriate funds. Completing a fraudulent transaction would require the collusion of multiple insiders.

We show that this can be accomplished using *threshold signatures*. In a threshold signature scheme, the ability to construct a signature is distributed among n players, each of whom receives a share of the private signing key. The participation of t or more of them is required to sign (for some fixed $t \leq n$). Thus a business can implement joint control of a Bitcoin address by distributing shares of the private key among multiple employees' devices.

Bitcoin supports "multi-signature" transactions where at least t of n keys must sign in order to spend bitcoins. This provides an alternate way to realize joint control, but this approach has undesirable impacts on anonymity and confidentiality, as we explain in Section 3.2, as well as incurring increased transaction sizes and therefore fees. Threshold signatures look no different from single-key signatures and thus avoid the shortcomings as of multi-signature transactions. Threshold signatures applied to Bitcoin wallets can be considered "stealth multi-signatures."

Though little-known in either the cryptographic literature or the Bitcoin community, it is possible to build an efficient threshold scheme on top of Bitcoin's default ECDSA signature algorithm[11, 21]. *Our primary contribution is to demonstrate that this construction can be deployed immediately and efficiently to protect Bitcoin transactions.* We have implemented threshold signatures for ECDSA and proven compatibility with Bitcoin by successfully relaying a transaction authorized by a 9-out-of-12 threshold signature onto

[1]Here and throughout the paper, our discussion refers to Bitcoin, but our results are equally applicable to the other signature-based cryptocurrencies such as Litecoin.

Bitcoin's block chain. To the external world, the transaction is indistinguishable from non-threshold transactions.

Second, we show how a single set of shares can enable joint control of arbitrarily many Bitcoin addresses. These addresses can be generated on demand by the participants, with no limit on the total number. We achieve this via a novel cryptographic construction. This is a crucial capability that allows us to apply security policies to wallets rather than individual addresses in a practical manner.

Third, we define the properties of *secure bookkeeping* and *secure delegation* and motivate their importance to corporate security. We show how threshold signatures can be used in sometimes non-obvious ways to realize these properties.

Fourth, we discuss various system design issues that arise for any business that wants to implement a threshold-signature-based access control system. We provide strategies and recommended practices to implement the system in a way that is easily understood and maintained by non-expert users.

Finally, we highlight an application of our techniques to the security of a personal Bitcoin wallet, rather than that of a business. We show how threshold signatures can be used to achieve *two-factor security* for wallets. Whereas in the corporate application, we use threshold signatures to split control among two or more people people, in the two-factor security application, we use it to split control among two devices. Rather than storing a private key on a single machine, shares of the key are stored on two devices, say a desktop computer and a smartphone. As neither device contains the key, both devices need to cooperate to sign a transaction. The user initiates a transaction on the desktop and then approves it using the smartphone. Once approved, the two devices engage in the threshold signature protocol and sign the transaction. To steal bitcoins from a two-factor secure wallet, an attacker would have to simultaneously compromise both devices. We also include recovery options that enable the user to regain control of her bitcoins if one of her devices is stolen or compromised, or both devices are lost (but not both controlled by the adversary).

Threshold signatures alone are not a complete security solution. They can and should be combined with system security and network security measures, business processes and accounting rules encoded in software, use of cold storage when appropriate, and so on. Threshold signatures do seem though to be an essential tool that has so far been missing from the Bitcoin ecosystem.

2. BACKGROUND

2.1 Internal controls

Internal controls can be classified as *preventive*, *detective*, and *corrective* [20, 23], or *prevent-detect-recover* [7]. Not all controls are technical, and many can be applied to Bitcoin assets without modification. Let us discuss the three classes of controls and highlight those for which a Bitcoin-specific solution is necessary.

Prevention: dual control. Two of the most common techniques for fraud prevention are functional separation and dual control. Both are ways of requiring two (or more) employees to co-operate to complete a financial transaction or some other action.

1. *Functional separation (series).* Different employees are involved at different points in the path of a transaction. For instance, a manager may have the ability to decide to purchase something, but the ability to transfer funds resides with the purchasing department, and there may be many such steps needed to complete the purchase and receive goods [7].

2. *Dual control (parallel).* Here two or more employees must simultaneously sign off on a given transaction. Our manager in the above example would not be able to single-handedly approve a transaction, but may need a second manager to sign off. Policies for parallel control can take various and complex forms. In a simple system, all employees are treated as equals and some fixed number of employees are needed to approve a transaction. The number of employees needed may depend on various factors such as the number of bitcoins involved.

   A more complex system may assign different ranks to different employees. Complex policies can specify different combinations of differently ranked employees that are necessary to approve a transaction. To illustrate, consider a business in which there are two levels of employees: managers and non-managers. An example policy in this setting would allow the following groups to independently sign off on transactions: (1) any two managers (2) a single manager together with two non-managers, or (3) 5 non-managers.

Functional separation is more of a business process problem than a technological one. Implementing functional separation over processes where money transfer happens via Bitcoin is not much different than if they happened via traditional banking, and thus we will not have much to say about these controls. We will focus on parallel controls as the problem has a technological solution.

Detection: bookkeeping. A business may decide that it is not cost efficient or simply impossible to prevent all fraudulent transactions. This necessitates ways to detect fraud after the fact with the aim of potentially recovering some of the theft. Detective measures also serve an additional purpose of evaluating the effectiveness of preventive controls.

A key component for detecting fraud is *bookkeeping*. Businesses need to keep accurate records of all of their assets. Every transaction must be logged along with the identity of the employee who enacted that transaction. These logs can then be audited to detect fraudulent transactions.

Of course fraud will only be detected if all transactions are logged accurately. If employees have the ability to alter the logs to cover up unauthorized transactions, then they can evade detection. Keeping with the principle of separation of duties, *double-entry bookkeeping* is a ubiquitous technique. In this system, every transaction is posted to two different books, in one as a credit and in one as a debit. The two books are kept by different people. Books are regularly checked to ensure that they add to zero, or *balance*. A single employee will not be able to alter both books, and thus a fraudulent transaction requires multiple employees to cooperate. To detect collusion between employees, both regular and and spontaneous audits are carried out [7].

Bitcoin offers possibilities for implementing the same separation-of-duties bookkeeping principle, but in novel ways not found in traditional systems. Since Bitcoin provides a secure transaction log (the block chain), could we perhaps cryptographically link the public block chain with internal, private double-entry bookkeeping records so that the latter cannot be tampered with without affecting the former?

Recovery. Once fraud has been detected, the business must take corrective action to recover. In the recovery phase, Bitcoin transactions deviate greatly from normal bank transactions. In the traditional system, if fraud is detected quickly, the victim can contact their bank and have the transaction reversed. Even if some time has passed, legal measures could be taken to coerce the bank to reverse the transaction. In the Bitcoin protocol as it currently stands, this is not possible. Transactions are irreversible; as soon as a valid transaction is included in the block chain it cannot be undone even if it is known to have been fraudulent.

For this reason, the *prevent-detect-recover* model is different with Bitcoin than it is with traditional banking. More resources must be spent on prevention, as the recovery options are far more limited. Of course, fraud detection is still extremely valuable — employees can be held accountable for theft, and even if recovery is not possible, detection helps evaluate the preventive controls [20].

Delegation of authority. Internal controls are further complicated by the practical need to for one employee to delegate authority for transactions to another employee (or a non-employee). For example, a manager may wish to give her administrative assistant spending power over her bank account or Bitcoin wallet while enforcing some policy. The policy may specify acceptable purposes of expenditure, spending limits, validity periods, and so forth. Not all delegation policies may be algorithmically enforceable.

2.2 Bitcoin

Bitcoin is a decentralized digital currency [15]. Bitcoins are assigned to *addresses*,[2] which are simply the hash of a public key. To transfer bitcoins from one address to another, a *transaction* is constructed that specifies one or more input addresses from which the funds are to be debited, and one or more output addresses to which the funds are to be credited. For each input address, the transaction contains a reference to a previous transaction which contained this address as an output address. In order for the transaction to be valid, it must be signed by the private key associated with each input address, and the funds in the referenced transactions must not have already been spent[15, 5].

Each output of a transaction may only be referenced as the input to a single subsequent transaction. It is thus necessary to spend the entire output at once. It is often the case that one only wishes to spend a part of the output that was received in a previous transaction. This is accomplished by means of a *change address* where one lists their own address as an output of the transaction. So, for example, if Alice received 5 bitcoins in a transaction and wants to transfer 3 of them to Bob, she constructs a transaction in which she transfers 3 to Bob's address and the remaining 2 to her own change address.

While it is technically possible for the sender to include their input address in the output, in practice, the change is generally sent to a newly generated addresses. The motivation for generating new addresses is increased anonymity since it makes it harder to track which addresses are owned by which individuals.

Bitcoin *wallets* are a software abstraction which seamlessly manage multiple addresses on behalf of a user. Users do not deal with the low level details of their addresses. They just see their total balance, and when they want to transfer bitcoins to another address, they specify the amount to be transferred. The wallet software chooses the input addresses and change addresses and constructs the transaction. New addresses can be generated at any point, and individual Bitcoin users typically have many addresses. The standard Bitcoin wallet implementation generates a new change address for every transaction.

Separate from change addresses, businesses may wish to maintain multiple addresses in their wallet for other reasons. A frequent practice is to provide a fresh address every time someone wishes to send bitcoins. Again, this increases unlinkability between addresses.

Signed transactions are broadcast to the Bitcoin peer-to-peer network. They are validated by *miners* who group transactions together into *blocks*. Miners participate in a distributed consensus protocol that collects these blocks into a public ledger called the *block chain*.

Our treatment of transactions thus far has described what a *typical* Bitcoin transaction looks like. However, Bitcoin allows for far more complex transactions. Every transaction contains a *script* that specifies how the transferred funds may be redeemed. For a typical transaction, the script specifies that one who wants to spend the bitcoins must present a public key that when hashed yields the output address, and they must sign the new transaction with the corresponding private key. A transaction can include a script that specifies complex series of rules that need to be enforced in order for the bitcoins to be spent.

While the original Bitcoin paper does not specify the signature algorithm to be used, the current implementation uses the Elliptic Curve Digital Signature Algorithm (ECDSA) over the NIST P-256 curve [5, 2].

2.3 Secret sharing and threshold cryptography

Threshold secret sharing is a way to split a secret value into shares that can be given to different participants, or players,

[2]To be precise, bitcoins are assigned to (and redeemed from) transactions and not addresses, but conceptually they can be thought of as belonging to the addresses named in those transactions.

with two properties: (1) any subset of shares can reconstruct the secret, as long as the size of the subset equals or exceeds a specified threshold (2) any subset of shares smaller than this threshold together yields *no information* about the secret. In the most popular scheme, due to Shamir, the secret can be encoded as a degree $t-1$ polynomial and a random point on the polynomial given to each of n players, any t of which can be used to precisely reconstruct the polynomial using Lagrange interpolation [16].

Secret sharing schemes are fundamentally one-time use in that once the secret is reconstructed, it is known to those who participated in reconstructing it. A more general approach is *threshold cryptography*, whereby a sufficient quorum of participants can agree to use a secret to execute a cryptographic computation without necessarily reconstructing the secret in the process. A (t, n)-*threshold signature* scheme distributes signing power to n players. Any group of at least t players can generate a signature, whereas a group of less than t cannot.

A key property of threshold signatures is that the private key need not ever be reconstructed. Even after repeated signing, nobody learns any information about the private key that would allow them to produce signatures without a threshold sized group.

ECDSA threshold signatures

Below, we first present the usual ECDSA signature generation scheme [13], and then the threshold scheme due to [11].

Standard ECDSA signature generation

The inputs are the base point G of order n, the private d, and the message to be signed, m. For the values of the domain parameters used in Bitcoin, see [3].

1. Compute e =SHA-1(m). Convert e to an integer using the method in ANSI X9.62.
2. Select an integer k such that $1 \leq k \leq n-1$
3. Compute $(x_1, y_1) = kG$.
4. Convert x_1 to an integer using the method in ANSI X9.62. Compute $r = x_1 \mod n$. If $r = 0$, return to step 2.
5. Compute $s = k^{-1}(e + dr) \mod n$. If $s = 0$, return to step 2.
6. The signature for m using the key d is the pair (r, s).

Threshold ECDSA signature generation

The steps in the threshold scheme parallel the regular scheme. The key difference is that the players compute on shares, and they only interpolate values that do not leak information about the private key. The only value that the players recover that depends on the shares of d is s, which is part of the signature (and this is publicized even in the non-threshold scheme). In particular, k and k^{-1} are never interpolated . This is important — given k and the signature (r, s), it is trivial to recover d. Even though $k \cdot G$ is known to players, the presumed hardness of the discrete logarithm problem for elliptic curves maintains that recovering k is intractable.

In the setup phase, a dealer D distributes shares of private key d on a random polynomial f of degree $t-1$ such that $d = f(0)$. There are n players, each having a unique index i, such that $1 \leq i \leq n$. Player i is given the share $f(i)$. Once this is complete, players can sign message m as follows:

1. Players compute e = SHA-1(m). Convert e to an integer using the method in ANSI X9.62. As m is public, this is no different from before.
2. Players run a joint random secret sharing (JRSS) protocol (see [11] for details) to share a random value k mod n on a polynomial g of degree t. Player i ends up with share $g(i)$.
3. Player i computes the value of the Lagrange basis polynomial at 0,
$$b_i(0) = \prod_{j \neq i, j \in B} \frac{j}{j-i}$$
B is the set of indices of the $t+1$ participants.
4. Player i computes $y_i = b_i k_i$ and broadcasts $V_i = y_i G$.
5. Players can now compute
$$(x_1, y_1) = kG = \sum_{i \in B} V_i$$
.
6. Convert x_1 to an integer using the method in ANSI X9.62. Compute $r = x_1 \mod n$. If $r = 0$, return to step 2.
7. Players run the secure reciprocal protocol given in [11] (together with the secure degree reduction protocol in [9]) using their shares of k to compute shares of k^{-1} on a degree t polynomial.
8. Players now compute shares of $w = dk^{-1}$ over a degree t polynomial by multiplying their shares of d and k^{-1} and running the secure degree reduction protocol given in [9].
9. Each player now computes the share
$$s_i = k_i^{-1} \cdot e + r \cdot w_i = k_i^{-1} \cdot e + r(d_i \cdot k_i^{-1}) = k_i^{-1}(e + d_i r)$$
They run secure degree reduction to reduce the degree of the polynomial sharing s back to t.
10. Players can now interpolate their shares of s to recover $s = k^{-1}(e + dr)$. If $s = 0$, return to step 2.
11. The signature for m using key d is the pair (r, s).

3. SECURE PROTOCOLS

Threshold signatures can be used as a primitive to build various secure protocols. In this section, we will describe four such protools. But first we discuss the threat model.

Adversary	Hot wallet	Cold wallet
Insider	Vulnerable by default; our methods are necessary	Reduces to physical security by default; our methods can help
External (network)	Reduces to network security by default; our methods can help	Safe

Table 1: Taxonomy of threats

3.1 Threat model

To classify the problems, we distinguish between internal and external threats as well as between hot and cold wallets. While the term wallet is generally used loosely to refer to a software abstraction (described in Section 2), we will use the term in the rest of the paper in a more limited, precise sense.

Definition 1 (wallet). *A collection of addresses with the same security policy.*

"Security policy" encompasses the ownership or access-control list, how the key material stored, and the conditions under which bitcoins in the wallet may be spent.

The terms *hot wallet* and *cold wallet* derive from the more general terms *hot storage*, meaning online storage, and *cold storage*, meaning offline storage. A hot wallet is a Bitcoin wallet for which the private keys are stored on a network-connected machine (i.e. in hot storage). By contrast, for a cold wallet the private keys are stored offline.

In the context of regular Bitcoin transactions, these definitions suffice. Yet when with the introduction of threshold signature and multi-signature transactions, we need to be more precise. These definitions assume a single key that is stored in one location, an assumption that we will break. Using our definition for wallet, we can differentiate between different types of wallets based on the differences in their security policies. We propose the following definitions:

Definition 2 (Hot wallet). *A wallet from which bitcoins can be spent without accessing cold storage.*

Definition 3 (Cold wallet). *A wallet from which bitcoins cannot be spent without accessing cold storage.*

Note that these new definitions refer to the desired effect, not the method of achieving it. The desired effect of a business that maintains a hot wallet is the ability to spend Bitcoin online without having to accessing cold storage.

Table 1 shows four types of possible threats. Securing a cold wallet is a physical security problem. While a network adversary is unable to get to a cold wallet, traditional physical security measures can be used to protect it from insiders — for example, private keys printed on paper and stored in a locked safe with video surveillance.

In addition, our methods may be used to supplement physical security measures. Instead of storing the key in a single location, the business can store shares of the key in different locations. The adversary will thus have to compromise security in multiple locations in order to recover the key. This is all we will say about securing cold wallets.

Protecting hot wallets from external attackers is a network security problem; if the network were completely secure, then this would not be an issue. We can use threshold signatures to reduce our reliance on network security. Our secure delegation protocol (Section 3.5) aims to reduce the reliance on network security in the context of delegation.

Protecting hot wallets from internal attackers is the most pressing problem. Our central claim, as stated earlier, is that the level of insecurity of this threat category has no parallels in traditional finance or network security, necessitating Bitcoin-specific solutions. Our protocols for parallel control (Section 3.2) and secure bookkeeping (Section 3.4) both address this problem. While parallel control is *preventive*, secure bookkeeping is *detective* .

3.2 Parallel control

Perhaps the most natural application of threshold signatures to Bitcoin is to build a system of parallel control. Consider a business that wishes to implement the following simple policy: transactions must be approved by t or more employees. For the purpose of this policy, all employees are considered equal; any t employees can approve a transaction, whereas any group of less than t employees cannot. To implement this policy, the business first shares its private key among its n employees using a (t, n) secret sharing scheme. Each employee is given a single share.

In a naive design, creating a transaction would involve first reconstructing the key and then signing in the regular manner. This approach does not work as the transaction is not "bound" to the execution of the protocol. In other words, once a subset of employees reconstruct the key, there is no mechanism that specifies how they must use that key. The participants (or at least one of them) will gain knowledge of the private key, and then have free reign to construct whatever transactions they would like without the approval of any other employees.

Instead of using secret sharing alone, we use the ECDSA threshold signature scheme presented above to achieve parallel control. In this scheme, the agreed-upon transaction is bound to the protocol: the output of the protocol is the signed transaction. Moreover, since the key is never reconstructed during the protocol, participants will not be able to sign subsequent transactions independently, and thus parallel control of the address is maintained. We note that the constructed signature looks exactly like a regular signature, and the resulting transaction is thus indistinguishable from a typical transaction.

Complex Policies. We have thus far only considered a simple policy in which t employees are required to approve a transaction. In practice, however, different employees may be assigned different ranks, and the threshold may depend on the ranks of the employees involved. There has been significant work showing how to extend Shamir's secret sharing scheme to arbitrary access policies [10, 12, 18]. Many of these methods are out-of-the-box compatible with the threshold signature scheme we presented.

While in theory, we can realize arbitrary access structures, we expect that all practical policies can be realized using threshold access structures. This is for two reasons. First, we expect that control of wallets will be shared not among all employees of a business, but rather within small teams. Second, threshold secret sharing can realize access control policies with employees of different ranks by assigning more than one share per employee.

To illustrate, consider an address that is shared among a manager and her five subordinates; transactions require approval of the manager and at least one other employee. In particular, non-managers cannot spend bitcoins without a manager, and a manager cannot spend bitcoins alone. To model this, we create a 6-of-10 threshold scheme. The manager is given 5 shares while each of her subordinates are given a single share. Only the manager together with one or more of the other employees can meet the threshold of 6 shares and spend the bitcoins.

Comparison with multi-signature approach

Bitcoin transactions are not limited to require a single signature to be redeemed, but in fact specify a script written in a stack-based, non-Turing complete programming language which defines the conditions under which a transaction may be redeemed. This scripting language includes support (`OP_CHECKMULTISIG`) for *multi-signature* scripts [8] which require at least t of n specified public keys to provide a signature on the redeeming transaction. By default, multi-signature transactions are currently only relayed with $n \leq 3$ keys, but may specify up to an absolute limit of $n = 20$.

A relatively recent feature of Bitcoin, *pay-to-script-hash*, enables payment to an address that is the hash of a script. This enables senders to specify a script hash, with the exact script provided by the recipient when funds are redeemd. This enables multi-signature transactions without the sender knowing the access control policy at the time of sending. A quirk of pay-to-script hash is that the $n \leq 3$ restriction is removed from t-out-of-n mult-signature transactions. However, due to a hardcoded limit on the overall size of a hashed script, the recipients are still limited to $n \leq 15$.

Advantages of multi-signatures. Multi-signature transactions have one clear benefit over using threshold signatures in that they can be signed independently by each participant in a non-interactive manner, whereas the ECDSA threshold signature protocol requires multiple rounds of interaction. Another potential benefit is that the redeeming transaction provides a public record of exactly which t of n keys were used to redeem the transaction, meaning secure bookkeeping is provided by default (though is also leaked publicly).

Advantages of threshold signatures. We argue that threshold signatures offer fundamental advantages stemming from the fact that in the multi-signature approach, the access-control policy is encoded in the transaction and eventually public revealed:

Flexibility. If a business using multi-signature transactions wants to make any modification to its access control policy, such as adding or removing an employee from those with transaction approval power, this requires a new script and thus a new address. This prevents businesses wishing to transact in Bitcoin from using a long-term static address and requires moving funds with each policy update. With threshold signatures, the policy is encoded not in the address but in the shares. To change the policy, the business would just need to re-deal key shares according to the new policy. Businesses can still use a static address for a receivable account and no Bitcoin transactions are required to change access control policy.

Confidentiality. When a business presents its script to spend a transaction, its internal access control policy is exposed to the world. Many companies will want confidentiality as to the internal controls that they enforce. Threshold-signed transactions are completely indistinguishable from regular transactions. Not only do they not leak the details of the access-control policy, they do not even reveal that access control is being used at all.

Anonymity. As we mentioned in Section 2, for purposes of increasing anonymity, the general practice is to use newly generated change addresses which cannot easily be linked to the input addresses. With multi-signature transactions, unlinkable change addresses are much harder to achieve, as even if fresh keys are used the destination address (even if a script hash) must have an identical t-of-n access control structure to prevent easily linking the change address with the sending address. With threshold signatures, change addresses will be unlinkable when sending funds to any regular (single-key) address or other threshold address, which should be the vast majority of cases (though not when interacting with multi-signature addresses or other script hash addresses). In Section 3.3 we show how to generate change addresses with equivalent threshold access control without distributing new shares.

Scalability. With multi-sig transactions, the size of transactions grows linearly with the access policy as all of the valid signing keys are included in the redeeming script (as well as the sending script, for non-script hash transactions). In addition to hard limits which Bitcoin enforces ($n \leq 15$ for script hash transactions and $n \leq 20$ in general), this means that more complex access control policies are subject to increased transaction fees and lead to bloat on the block chain. As threshold signature transactions are indistinguishable from ordinary transactions no matter how complex the underlying access policy is they will not require increased fees or generate additional data which must be globally broadcast.

3.3 Extending security policies from addresses to wallets

Our discussion of parallel control until this point has focussed on addresses. We now describe how to extend our threshold-signature based system to wallets. The solutions we present will not be limited to parallel control, but will apply generally to threshold-signature-based systems. As secure bookkeeping (Section 3.4), secure delegation (Section 3.5), and 2-factor security (Section 3.6) are all built on top of threshold signatures, this lets us extend all of our security policies to wallets.

The key technical challenge in extending threshold signatures to wallets (i.e., collections of addresses) is that new addresses need to be generated on demand and we do not know in advance how many addresses will be needed. It is not feasible to execute the dealing step of secret sharing each time we need to generate a new address, since this step affects all the participants in the group and requires extra security precautions.

Two approaches can be used to generate fresh addresses in the threshold context. As we'll see the latter approach is clearly superior.

Generating in advance. Some Bitcoin clients generate a buffer of addresses, typically 100 at a time [1]. We can implement this in our threshold scheme as well. During the initial share distribution, we will also share the keys for multiple newly generated addresses. The extra addresses can be used as change addresses or given out as fresh addresses. Once all the addresses are used up, participants will need to one again be dealt shares of new addresses. By generating a large number of change addresses each time, we can make the need to share new addresses very infrequent. This approach has two drawbacks. Firstly, as the addresses in this wallet are not linked to each other, each address has to be backed up separately. As there is no concise backup string that can recover the entire wallet, this approach is not compatible with paper backups. Second, to minimize the hassle of re-dealing, we will need to be very generous with the number of addresses that we generate, wasting storage on keys that we do not yet need.

Deterministic wallets

Deterministic Wallets are sophisticated wallets in which fresh keys can be derived from previous keys, with the additional property that the fresh public key can't be linked to the previous public key without knowledge of the private key(s), preserving unlinkability.[3] We now present a threshold variant of deterministic wallets. Our construction introduces the notion of "extended keys" which are regular Elliptic Curve keys appended with a private 32-byte salt. If a key pair contains a public key K and a private key k, the extended key pair is the pair $(K||c, k||c)$, where $||$ represents concatenation. From the initial extended key, or the *master* extended key, new *child* key pairs can be generated in a deterministic fashion.

Using secret sharing, we can create shared deterministic wallets. We modify the key derivation procedure given in [6] so that child keys can be derived in a threshold manner. This modified scheme allows shared child addresses to be derived in a deterministic manner from a shared master address. In particular, if an address is shared in an (t, n) fashion, our scheme allows each shareholder to independently compute their share of the private key for a new child address. The new address is shared with the same (t, n) structure as the master address. Each shareholder can also independently derive the new public key and address.

[3]Deterministic wallets typically have another property, hierarchical key derivation, which is not relevant to us.

Threshold deterministic address derivation

We refer to a curve of order n with base point G. We refer to χP, the compressed form of curve point P. The master private key is k_{mas}, and corresponding master public key is $K_{mas} = k_{mas} \cdot G$. We refer to c, the 32-byte salt. We will refer to a master extended key pair $(K_{mas}||c, k_{mas}||c)$ and use it to derive a child key pair (K_i, k_i) in which i is the sequence number of the child. Our protocol begins with a shared address, which we call the master address. In particular, the jth participant has a share $k_{mas}^{(j)}$ of k_{mas}.

1. Let $T = \text{HMAC-SHA512}(c_{mas}||\chi(K_{mas})||i)$
2. Split $T = T_L||T_R$ into two 32-byte sequences, T_L and T_R
3. $k_i^{(j)} = T_L + k_{mas}^{(j)}$

Each participant now holds a share of the private key $k_i = T_L + k_{mas}$. The public key corresponding to the derived private key can be reconstructed from just the master extended public key. As this is known to all participants, they can each compute it independently as follows:

$$T_L \cdot G + K_{mas} = (T_L + k_{mas}) \cdot G = k_i \cdot G = K_i$$

Participants can use their shares of the child private key to construct a signature from the new address in a threshold manner. At no point in the derivation of the child key or the subsequent signature generations is either key constructed. It is also worth noting that this derivation protocol is non-interactive. Participants can derive shares of the new key completely on their own.

In order to derive the child private key share, each participant will have to use their share of the master private key. However, to derive the child public key, it suffices to know the master public key and T_L. To derive T_L, one only needs the salt, c, and the master public key (i.e. the extended master public key).

It is important that the salt be secret. If we publicize c, anybody that knows the master public key can derive the public keys and addresses for the i^{th} child. If we are using the child addresses as change addresses, this would defeat the entire purposes of change addresses since they can be derived from, and hence linked to, the initial address (assuming that i is chosen in a predictable manner).

We have thus shown how we can generalize our protocols from addresses to wallets. The scheme presented here enables generating new addresses on demand that have the same access-control policy as the original address. The protocol is non-interactive, and eliminates the need for a dealer to generate and distribute shares of new addresses. This doesn't completely eliminate the need for a dealer since a change in the access-control policy requires re-sharing. See Section 4.5 and Section 4.6 for techniques to further reduce the frequency of re-sharing.

3.4 Secure bookkeeping

We will now show how threshold signatures can be used to build a secure *detection* mechanism. We will informally define secure bookkeeping and show how it can be realized using threshold signatures.

Definition 4 (Secure bookkeeping). *A bookkeeper is an entity that logs transactions along with the identity of the participant(s) responsible for it. A bookkeeping system is* **secure** *if for every valid transaction, the bookkeeper can cryptographically prove which participants were involved in it.*

For wallets controlled by a single individual, the block chain itself provides secure bookkeeping. Every transaction is logged, and the signed transaction itself serves as a cryptographic proof of the initiating (pseudo-)identity.

Achieving secure bookkeeping for a shared wallet is more difficult. When multiple users share a wallet, the transaction itself does not identify which users participated in signing the transaction. This holds true for all access policies. Certainly in a 1-out-of-n access policy, in which each employee is given a copy of the key, the transaction does not look unique for each employee. But even with a more complex t-out-of-n access structures in which $t \neq 1$, it is impossible to identify the participants from the signed transaction.

To achieve bookkeeping for shared wallets, we require that every employee be issued an asymmetric key pair (unrelated to Bitcoin key pairs). The company maintains a PKI that maps employees to public keys. All participants in the threshold signature protocol sign each message using their private key.

A designated bookkeeper will log all signed messages, and these logs can later be used to provably identify the participants of a given transaction. Secure bookkeeping is compatible with any parallel control policy. If a wallet is governed by a parallel control policy based on t-out-of-n secret sharing, to achieve a bookkeeping-enabled version of the same policy:

- use $n + t$-out-of-$2n$ secret sharing
- distribute n of the shares to the bookkeeper
- distribute the remaining t shares as in the original policy.

This ensures that no transaction can succeed without the approval of the bookkeeper. The bookkeeper is designed as an always-online entity that always approves every transaction request; its sole function is logging.

An alternative to a dedicated bookkeeping entity is to parallelize bookkeeping as well. Here every participant stores logs of the transactions they participated in. As long as *any one participant* produces their logs, they can prove the involvement of every participant involved in a given transaction.

3.5 Secure delegation

Say Alice wants to give Bob control of her Bitcoin wallet while enforcing some policy. The policy may be temporal, limiting the period of time for which Bob has control. The policy may contain a blacklist of addresses Bob cannot send to, or it may contain a whitelist of the addresses that Bob is allowed to send bitcoins to. Alice may wish to enforce a spending limit for a given transaction or an overall spending limit for Bob. Policies can be arbitrarily complex, but we will restrict our focus to those policies that are algorithmically enforceable.

Note that Alice cannot give Bob her private key since once Bob has the key, Alice can no longer restrict how he uses it. To avoid this issue, we will consider delegation protocols in which Alice gives Bob some credential other than the key. Bob will be able to access a server that Alice set up and use this credential to create a transaction that is allowed by the policy.

Alice can run a server that authenticates Bob and signs transactions on her behalf provided that they are permitted by the policy. While this would work, it relies heavily on network security. An adversary who compromises the server will learn the key. This motivates:

Definition 5 (Secure delegation). *A delegation system is* **secure** *if: (1) Bob can produce signed transactions if and only if they are allowed by the policy, and (2) Alice does not use a hot wallet.*

We can achieve secure delegation with a 2-out-of-2 threshold signature scheme. Alice creates two shares of her private key.[4] She gives one to Bob and stores the other on her server. Alice configures her server to only participate in generating a threshold signature for transactions that are allowed by the policy. In this system, an attacker who compromises Alice's server will gain nothing as the key share on the server is useless without Bob's share. Of course if an attacker learns Alice's key he can steal her bitcoins, but Alice's key is stored offline, making this attack much more difficult.

Recall that because we are using threshold signatures, the key is never reconstructed. Thus even after Bob successfully creates a signed transaction from Alice's address, he is still unable to sign further transactions without the participation of Alice's server. Furthermore, Alice can revoke the delegation by simply destroying her share. Bob's share is now useless, and Alice's wallet remains secure.

3.6 Two-factor security

[4]For simplicity of exposition we can assume that there is a single address that Alice is delegating, but as described in Section 3.3, to delegate a wallet Alice would share her master private key.

Thus far, we focused on a corporate environment with multiple actors. But we can extend the principles of dual control to a situation in which there is only a single person — here we split control between different *machines*, not people. The private key is not stored on any machine nor is it ever reconstructed during signature generation.

To protect against theft, Alice distributes 2-out-of-2 shares of her private key among two devices that she owns, say her computer and smartphone. When Alice initiates a Bitcoin transaction from her computer, a prompt containing the transaction details will appear on her smartphone via her wallet app. If she confirms, the two devices will sign the transaction using the threshold scheme and broadcast it. We stress that at no point was the key reconstructed on either device; on its own, neither device contains enough information to create a signature. An attacker will have to compromise both her computer and her smartphone to steal her bitcoins.

A Bitcoin wallet with two-factor security is arguably more secure than cash, especially with appropriate backup and recovery options (Section 4.7). We can further improve security by generalizing to multi-factor security, but given the usability drawbacks it is not clear if this will be useful in practice.

4. SYSTEM DESIGN

We address the most important design decisions that need to be made to build a system that implements the protocols we've presented. Our primary design goal is to build usable systems that have a clear security model and are easy to administer. Our decisions, therefore, favor simplicity over designs that require expert knowledge to administer securely.

4.1 Identity

Although not required by the threshold signature protocol, all participants in our system are identified and all protocol messages are signed. This improves accountability, and in particular enables the secure bookkeeping protocol of Section 3.4. It also has the side-benefit of preventing denial-of-service attacks on the protocol by outsiders.

We intend to use X.509 certificates [19], which is convenient because many companies already issue certificates to their employees. We can layer our protocol over TLS with all communication going through a server, and each participant will authenticate themselves using a certificate. Alternatively, participants can communicate directly and sign each message.

4.2 Synchronous vs. asynchronous design

The ECDSA threshold signature protocol proceeds in rounds and requires interaction between players. The requirement for interaction favors a synchronous design in which all users are online when the signature is being generated. The need for synchronicity is a limitation, and for this reason non-interactive protocols are preferred. While non-interactive threshold signature schemes do exist for other signing algorithms [17], there is no known non-interactive ECDSA threshold signature algorithm. Synchronicity can be practically achieved in a number of different ways:

1. Signing can be done from employees' workstations that are always left on. In this design, an employee would broadcast a request to sign. This would send a notification to all employees in the group. As soon as the threshold number approve the request, the signature generation would immediately run as all workstations are left on.

2. Instead of their workstations, employees can use always-on devices such as smartphones or dedicated devices.

3. Employees can coordinate a time out-of-band during which they will be online.

4.3 The curse of homogeneity

Our main goal in creating parallel control systems has been to eliminate single points of failure. In a corporate environment, however, all employees often use standard-issue systems, and system administrators sometimes have the ability to access all of these machines. A rogue system administrator or an adversary who gains access to a system administrator's credentials may be able to bypass systems of parallel control. Further, malware may simultaneously compromise many or all machines due to software homogeneity.

Thus parallel control is more effective when the software platform is internally heterogeneous, when system administrators don't directly control employee machines, or when shares are stored on personal/dedicated devices instead of workstations. Another possibility is to combine two-factor security (Section 3.6) with parallel control.

4.4 Security of the dealing step

The threshold signature protocol as presented relies on administrators to distribute shares. Proper checks must be put in place on administrators to ensure that they do not deal extra key shares to themselves. To mitigate this risk, dealing should only be done very infrequently and with human oversight rather than in an automated way. The shared deterministic wallet protocol of 3.3 obviates the need to re-deal shares unless the access-control list changes, and the techniques of Section 4.5 minimize the need to re-share even when the list does change.

Alternately, there exists a protocol to deal shares without a dedicated dealer (we have omitted it, but it is straightforward). Each participant must check the identities of the other participants to avoid executing the protocol with impostors. This is more secure but requires all employees to participate in the dealing step synchronously.

4.5 Changing the access-control list

In a parallel control system, how do we handle changes to the access list? In practice we expect this to be an infrequent occurrence — companies' finances are organized into hierarchical *cost centers*, and control will be shared between the (relatively few) employees in a cost center rather than across the business. Nevertheless, we explore solutions that further reduce the frequency of creating and distributing new shares.

The problem of how to add and revoke shares in a secret sharing scheme is well studied [22, 14, 25, 24], and elegant solutions exist. The protocols presented in [14, 25] enable the creation of new shares without changing the shares of existing employees. The schemes for revocation do require all employees to update their shares. But even when the secret is re-shared, these schemes are secure in that they do not require reconstructing the secret.

There are less complex alternatives. One technique is to generate some extra shares during the dealing step and store them on hardware devices. These devices are physically secured so that no employee can access them. If there are initially n employees and we wish to share with a threshold of t, we create $n + m$ shares with the same threshold t. When new employees arrive, we give them one of these extra shares, and we can do this until all m extra shares have been used up. Variations of this technique are possible when employees have different ranks.

A single employee leaving may not necessitate re-sharing the key. As a concrete example, with 3-out-of-5 parallel control, if one employee leaves, this ex-employee will still have to collude with 2 of the remaining employees to create a fraudulent transaction. This is not likely and we may not have to worry about it. A security policy can specify that the key be re-shared only after a certain number of employees leave.

4.6 Parallel control backup and recovery

An employee losing a share has a similar effect as changes to the access-control list. The most secure option is to re-share the key every time a loss or theft of any share is detected or suspected.

Alternatively, we can apply the same principle as before and generate extra shares. When an employee loses her share, we simply give her one of the extras. We only need re-share once a significant number (as specified in the security policy) of shares has been lost. Of course, employees can pretend to lose their shares in order to gain access to multiple shares. Therefore the security policy should also contain limitations on how many extra shares can be given to a single employee. For example, if the same employee loses their share more than once, then we re-share rather than give that employee a second one of the extras.

4.7 Two-factor security backup and recovery

In the two-factor security system described in Section 3.6, if one of Alice's devices is stolen but the other is secure, the thief will not be able to spend Alice's bitcoins, *but neither will Alice*. We present two options for backup of shares. First, we can use 2-out-of-3 sharing instead of 2-out-of-2 sharing, with the third share used as an offline backup. Once Alice detects a loss, theft or intrusion that affects one share, she can re-secure her wallet by reconstructing the keys, re-sharing them, and deleting the original shares. The lost/stolen shares will thus be rendered useless and the wallet will be two-factor secure.

The second option is to stick to 2-out-of-2 secret sharing but separately store backups of the shares of each device. The recovery process (reconstruction, resharing, and deletion of original shares) is identical. The first option has the advantage that if the backup share is stolen it doesn't confer knowledge of the private key, whereas the second option allows recovering from a loss of both devices (as long as the adversary doesn't compromise both).

As our wallets are deterministic, with either approach, the only key that needs to be backed up is the master key. Once the master key has been reshared, participants can derive their shares for all other keys in the wallet.

5. IMPLEMENTATION

We created a prototype implementation of the ECDSA threshold signature scheme in Java. We began by implementing Shamir secret sharing, and we then built the threshold signature on top of it. Our implementation consisted of participants that interacted over the network. Each participant had a static IP that was registered at the time when the key shares were distributed. The prototype implementation did not authenticate participants, but in the production implementation, we intend to layer the protocol over TLS to provide authentication.

Once we constructed the signature, we integrated it into the Bitcoinj library. We replaced Bitcoinj's key generation code with code that generated key shares and distributed them among the participants. We replaced Bitcoinj's signing code with the threshold construction. Using this setup, we created a Bitcoin address and had the key distributed amongst 12 participants with a threshold of 9. We transferred a small amount of Bitcoin to this address, and then forwarded the Bitcoin onwards by threshold-signing a transaction. We broadcast the transaction to the Bitcoin peer-to-peer network, and it was included in the block chain.[5]

6. CONCLUSION

Corporations require internal financial controls to operate effectively. We demonstrated that current Bitcoin technologies such as multi-signature scripts are inadequate for realizing such controls in a practical manner. Instead, we showed why a threshold-signature-based system is the right approach. We presented the cryptographic underpinnings of such a system and addressed various design questions. We are now working on an implementation of both parallel control for businesses and two-factor security for individuals. Our techniques have the potential to dramatically improve Bitcoin security, moving it closer to widespread adoption as a currency.

7. REFERENCES

[1] Bitcoin wiki: Deterministic Wallet. `https://en.bitcoin.it/wiki/Deterministic_Wallet`, accessed: 2014-02-11

[2] Bitcoin wiki: Elliptic Curve Digital Signature Algorithm. `https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm`, accessed: 2014-02-11

[3] Bitcoin wiki: Secp265k1. `https://en.bitcoin.it/wiki/Secp256k1`, accessed: 2014-03-09

[5]Full details of this transaction can be viewed online at `https://blockchain.info/tx/55451090debe729120d4a2e6b49bd3d5cabb881e753ccda0f5ea499a3438de9b`

[4] Bitcoin wiki: Transaction Fees. `https://en.bitcoin.it/wiki/Transaction_fees`, accessed: 2014-02-11

[5] Bitcoin wiki: Transactions. `https://en.bitcoin.it/wiki/Transactions`, accessed: 2014-02-11

[6] Github: BIP: 32. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[7] Anderson, R.: Security engineering. Wiley. com (2008)

[8] Andresen, G.: Github: Shared Wallets Design. `https://gist.github.com/gavinandresen/4039433`, accessed: 2014-03-20

[9] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 1–10. ACM (1988)

[10] Ghodosi, H., Pieprzyk, J., Safavi-Naini, R.: Remarks on the multiple assignment secret sharing scheme. Information and Communications Security pp. 72–80 (1997)

[11] Ibrahim, M.H., Ali, I., Ibrahim, I., El-sawi, A.: A robust threshold elliptic curve digital signature providing a new verifiable secret sharing scheme. In: Circuits and Systems, 2003 IEEE 46th Midwest Symposium on. vol. 1, pp. 276–280. IEEE (2003)

[12] Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (Part III: Fundamental Electronic Science) 72(9), 56–64 (1989)

[13] Johnson, D., Menezes, A., Vanstone, S.: The elliptic curve digital signature algorithm (ecdsa). International Journal of Information Security 1(1), 36–63 (2001)

[14] Li, X., He, M.: A protocol of member-join in a secret sharing scheme. In: Information Security Practice and Experience, pp. 134–141. Springer (2006)

[15] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Consulted 1, 2012 (2008)

[16] Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)

[17] Shoup, V.: Practical threshold signatures. In: Advances in Cryptology—EUROCRYPT 2000. pp. 207–220. Springer (2000)

[18] Simmons, G.J.: How to (really) share a secret. In: Proceedings on Advances in cryptology. pp. 390–448. Springer-Verlag New York, Inc. (1990)

[19] Solo, D., Housley, R., Ford, W.: Internet x. 509 public key infrastructure certificate and certificate revocation list (crl) profile (2002)

[20] Tipton, H.F., Krause, M.: Information security management handbook. CRC Press (2004)

[21] Wang, H., Wu, Z., Tan, X.: A new secure authentication scheme based threshold ecdsa for wireless sensor network. In: Security and Management. pp. 129–133 (2006)

[22] Wong, T.M., Wang, C., Wing, J.M.: Verifiable secret redistribution for archive systems. In: Security in Storage Workshop, 2002. Proceedings. First International IEEE. pp. 94–105. IEEE (2002)

[23] Wood, J., Brown, W., Howe, H.: IT Auditing and Application Controls for Small and Mid-Sized Enterprises: Revenue, Expenditure, Inventory, Payroll, and More, vol. 573. John Wiley & Sons (2014)

[24] Yu, J., Kong, F., Cheng, X., Hao, R.: Two protocols for member revocation in secret sharing schemes. In: Intelligence and Security Informatics, pp. 64–70. Springer (2011)

[25] Yu, J., Kong, F., Hao, R., Li, X.: How to publicly verifiably expand a member without changing old shares in a secret sharing scheme. In: Intelligence and Security Informatics, pp. 138–148. Springer (2008)