Securing Bitcoin wallets via a new DSA/ECDSA threshold signature scheme

Steven Goldfeder
Princeton University
stevenag@cs.princeton.edu

Rosario Gennaro
City University of New York
rosario@cs.ccny.cuny.edu

Harry Kalodner
Princeton University
kalodner@cs.princeton.edu

Joseph Bonneau
Stanford University
jbonneau@cs.stanford.edu

Joshua A. Kroll
Princeton University
kroll@cs.princeton.edu

Edward W. Felten
Princeton University
felten@cs.princeton.edu

Arvind Narayanan
Princeton University
arvindn@cs.princeton.edu

Abstract

The Bitcoin ecosystem has suffered frequent thefts and losses affecting both businesses and individuals. 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.

To address this problem, we present the first threshold signature scheme compatible with Bitcoin's ECDSA signatures and show how distributed Bitcoin wallets can be built using this primitive. For businesses, we show how our distributed wallets can be used to systematically eliminate single points of failure at every stage of the flow of bitcoins through the system. For individuals, we design, implement, and evaluate a two-factor secure Bitcoin wallet.

1 Introduction

1.1 Bitcoin's security conundrum

The statistics on Bitcoin[1] hacks, thefts, and losses are extraordinary — there have been ten thefts of over 10,000 BTC each since mid-2011, and another thirty of over 1,000 BTC[2] [1]. Kaspersky labs report detecting about a million infections per month of malware designed to search for and steal bitcoins from machines they infect [2].

The pervasiveness and regularity of these vulnerabilities highlights how Bitcoin is inherently theft-prone. Although similar in functionality, Bitcoin transactions differ from traditional payments in several crucial 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 transactions 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.

In traditional business and banking, there are a variety of internal controls to deter fraud, encompassing prevention, detection, and recovery. In the Bitcoin context, preventive internal controls become both more important (due to irreversibility and pseudonymity) and harder to implement (due to automation). These problems are particularly acute when bitcoins are in *hot storage*, i.e., the private keys are stored on online devices. Businesses that actively transact in Bitcoin must necessarily keep some of their balance in hot storage.

For Bitcoin and cryptocurrencies to gain mainstream adoption, a breakthrough in security is needed — the current situation where a single rogue employee or a piece of malware can empty an organization's funds in hot storage instantly, irreversibly, and anonymously is simply untenable.

One possible improvement is *joint control of bitcoins*, i.e., requiring multiple designated participants to sign a transaction before it will be considered valid. Joint control mitigates the risk of internal fraud as no participant (employee, device, department, etc., as the case may be) has the ability to single-handedly misappropriate funds. Completing a fraudulent transaction would require multiple insiders to collude, or multiple devices to be stolen or compromised.

[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.

[2]As of this writing, a bitcoin trades for around USD 230

Bitcoin natively supports "multi-signature" addresses. These are special addresses associated with *n* specified public keys and a threshold $t \leq n$. Spending bitcoins from such addresses requires signatures from at least *t* of the *n* keys. This can be used to achieve joint control, but multi-signature transactions inherently suffer significant drawbacks in Bitcoin: the number of keys *n* has a hard-coded limit, mult-sig transactions require a business's access control policies to be made public and can severely harm the anonymity of individual users, and these transactions incur increased transaction fees (Section 4.3).

Instead, we observe that joint control can be accomplished using *threshold signatures*. In a threshold signature scheme, the ability to construct a signature is distributed among *n* participants, or 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 to multiple participants. Threshold signatures look no different from single-key signatures and thus avoid the shortcomings of multi-signature transactions. Threshold signatures applied to Bitcoin wallets can be considered "stealth multi-signatures."

1.2 Our contributions

Bitcoin uses ECDSA signatures [3], the elliptic curve variant of the Digital Signature Algorithm (DSA) [4], to validate transactions. Unfortunately, ECDSA (and DSA in general) is not a particularly friendly signature scheme for threshold cryptography[3] The best known scheme is a 2-party threshold DSA signature protocol due Mackenzie and Reiter [6, 7].

Our primary contribution is to introduce the first practical *t*-of-*n* threshold signature scheme compatible with ECDSA signatures (and hence compatible with Bitcoin). More specifically, we construct a *t*-out-of-*t* threshold signature protocol, and derive a *t*-out-of-*n* protocol via a combinatorial construction that involves $\binom{n}{t}$ sets of shares. This is acceptable in practice since *n* and *t* are both very small in virtually every practical application. We present concrete running time measurements of the protocol, summarized below.

Next, we show how to secure *Bitcoin wallets* using threshold cryptography. A Bitcoin wallet is a software abstraction that seamlessly manages multiple addresses on behalf of the user. We begin by identifying necessary and desirable security properties of Bitcoin wallets and construct a *DNF wallet*, a wallet that utilizes our signature scheme and realizes any access structure (access control policy) expressed as a disjunctive normal formula. As a special case of a DNF wallet we derive a *threshold Bitcoin wallet*. A key aspect of the construction is to realize threshold *deterministic* wallets, a property that allows a single set of shares to control an arbitrary number of addresses.

Third, we show how a merchant or another organization transacting in Bitcoin can prevent single points of failure throughout the flow of bitcoins through the system. This breaks down into three sub-problems: a threshold version of the BIP-70 payment protocol for receiving payments, securing bitcoins at rest using threshold wallets, and finally, cryptographic enforcement of "functional separation of duties" wherein an operation succeeds if and only if multiple parties act at different points on its path according to a specified policy.

Fourth, we address several important design considerations in implementing our protocols, including synchronous vs. asynchronous design and back-up and recovery of shares.

Finally, we design, implement and evaluate a two-factor secure wallet by modifying the Multibit wallet program and building an Android app to go with it. Our implementation uses a QR code displayed on one device and captured via the camera on the other in order to securely pair and share key material. The signature protocol completes in under 15 seconds.

1.3 Implications.

Threshold wallets promise a significant leap in Bitcoin security. In terms of the prevent-detect-recover paradigm, Bitcoin doesn't allow recovery due to irreversible transactions, with or without threshold signatures, but when it comes to prevention and detection, our methods match and in some cases even improve on traditional controls in finance, as we will show.

How much can such improvements affect the Bitcoin ecosystem? The benefits of using Bitcoin over government currencies either for individuals or businesses are currently questionable. For individuals, anonymity (i.e., pseudonymity) and financial privacy are the main potential benefits, but this potential has not been realized for a simple reason: maintaining one's own wallet was quickly found to be far too insecure, and so most Bitcoin users instead use online services such as Coinbase, losing any anonymity. Multi-signature wallets have been proposed as a way to take control of one's own bitcoins, but as we show in Section 4.3, multi-signature addresses are far more identifiable than vanilla Bitcoin addresses.

For merchants and payment services, the main benefit of Bitcoin is fast, low-cost transactions, but so far the

[3]Plausible alternative schemes with equivalent security, such as Schnorr signatures have well-known and relatively straightforward threshold versions [5]. While proposals for introducing alternative signatures schemes for Bitcoin do exist, there's no evidence that a soft fork (which is necessary) to incorporate such a scheme into Bitcoin is planned.

lower transaction fees have been more than nullified by the premium arising from fraud in the system. If theft and fraud can be controlled, the economic benefits can finally be realized. Again, multi-signature wallets are just as problematic: we explain in Section 4.3 why they break confidentiality of transactions. Most businesses would rather keep their books private!

Thus, the availability of threshold wallets may finally make good on Bitcoin's promise to bring various benefits over traditional payments to consumers and businesses. Additionally, better security will indirectly lead to improved privacy and lower service fees and arguably pave the way for mainstream adoption of cryptocurrencies.

2 Background

2.1 Internal controls

Internal financial controls are conventionally classified as *preventive*, *detective*, and *corrective* [8, 9], or *prevent-detect-recover* [10]. 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. Ross Anderson provides a classic example: "A manager takes a purchase decision and tells the purchasing department; a clerk there raises a purchase order; the store clerk records the goods' arrival; an invoice arrives at accounts; the accounts clerk correlates it with the purchase order and the stores receipt and raises a check; and the accounts manager signs the check." Anderson also notes that functional separation continues after the completion of the transaction with many levels of logging and audit. [10].

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.

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.

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 all employees who acted on the transaction's path. These logs can then be audited to detect fraudulent transactions.

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 [8].

2.2 Bitcoin

Bitcoin is a decentralized digital currency [11]. Bitcoins are owned by *addresses*; an address is 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 [11, 12].

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 possible for the sender to include their input address in the output, the best and recommended practice is to send the change 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.

A Bitcoin *wallet* is a software abstraction which seamlessly manages 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 common practice is to provide a fresh address every time someone wishes to send bitcoins. This serves two purposes: it allows the business to easily disambiguate between multiple payers (e.g. if Alice and Bob are each paying 1 BTC, by giving a different address to each payer, the business can now track whom it received payment from) and it also increases unlinkability between the business's various transactions.

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 an append-only global log 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 secp256k1 curve [12, 13, 14].

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, 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 [15].

For the (t,t) case, a simple secret sharing scheme is realized by giving each of the t players a value x_i such that the secret is equal to the product of the t shares.

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. In a threshold signature scheme, 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. Indeed, threshold cryptography is a specific case which led to the more general development of secure multiparty computation [16].

3 A NEW THRESHOLD DSA/ECDSA SIGNATURE SCHEME

3.1 Overview

In this section, we present a threshold signature scheme for DSA and ECDSA. We introduce a generic notation that applies to both DSA and ECDSA, and we present our scheme using this notation.

We define a generic G-DSA signature algorithm as follows. The public parameters, pp, include a cyclic group $\mathcal{G}$ of prime order q generated by an element G, and another hash function H' defined from $\mathcal{G}$ to Z_q.

- $(x,y) \leftarrow \texttt{Gen}(pp)$

  - The secret key x is chosen uniformly at random in Z_q.
  - The public key $y = G^x$ computed in $\mathcal{G}$.

- $(r,s) \leftarrow \texttt{Sign}(M,x)$
  - On input an of an arbitrary message M, compute $m = H(M) \in Z_q$. Then the signer chooses k uniformly at random in Z_q and computes $R = G^k$ in $\mathcal{G}$ and $r = H'(R) \in Z_q$. Then she computes $s = k^{-1}(m+xr) \bmod q$. The signature on M is the pair (r,s).

- Yes/No $\leftarrow \texttt{Verify}(M,(r,s),y)$
  - The receiver checks that $r,s \in Z_q$ and computes

$$R' = G^{ms^{-1} \bmod q} y^{rs^{-1} \bmod q} \text{ in } \mathcal{G}$$

and accepts if $H'(R') = r$.

The traditional DSA algorithm is obtained by choosing large primes p,q such that $q|(p-1)$ and setting $\mathcal{G}$ to be the subgroup of Z_p^* of order q. In this case the multiplication operation in $\mathcal{G}$ is multiplication modulo p. The function H' is defined as $H'(R) = R \bmod q$.

The ECDSA scheme is obtained by choosing $\mathcal{G}$ as a group of points on an elliptic curve of cardinality q. In this case the multiplication operation in $\mathcal{G}$ is the group operation over the curve. The function H' is defined as $H'(R) = R_x \bmod q$ where R_x is the x-coordinate of the point R.

3.2 Threshold DSA: what's known

Definition 1 (Threshold signature scheme (informal)) *In a t-out-of-n threshold signature scheme the secret key is shared among n participants in such a way that any t of them can execute a protocol to compute the signature of a given message. Moreover the signature scheme is unforgeable against any $t-1$ colluding participants.*

Note that the definition rules out the trivial solution of reconstructing the key and having one of the participants sign the message, as this would violate the unforgeability property — if a malicious participant reconstructs the key, he can later sign messages by himself. For a full, formal definition of threshold signature schemes, see Boldyreva [17].

There have been other schemes in the literature that use a weaker definition of threshold signatures. In particular, these schemes allow for a larger separation between the number of participants that can learn information about the key and the number of participants that can produce a signature. For the case of the DSA signature scheme, in [18, 19] Gennaro *et al.* present such a scheme for which s participants could jointly reconstruct the key, but $t = 2s+1$ participants were required to construct a signature. Ibrahim et al. explicitly convert Gennaro's scheme to ECDSA [20].

Unfortunately, the weaker definition does not allow the most common use cases of threshold signatures. In particular, it does not support the commonly desired 2-out-of-2 case, in which the key is split among 2 participants so that both have to cooperate to sign, while neither participant by himself has any information about the secret key (in [18, 19] if 1 participant has no information about the key, then one would need at least 3 participants to sign). Indeed, the majority of applications that we present in Section 5 are not realizable with schemes that only meet this weaker definition.

Mackenzie and Reiter present a solution that supports only the 2-out-of-2 case [7] which we extend. They present their scheme for the specific case of the DSA scheme, but the G-DSA notation and the method for conversion that we showed in Section 3.1 applies to their work as well, and thus shows how to convert their scheme from DSA to ECDSA. We present all of our results using the G-DSA notation that we introduced, and they therefore immediately apply to both DSA and ECDSA.

Our definition of threshold signatures requires the stronger property that that t players can sign and $t-1$ learn nothing. There has been no threshold DSA/ECDSA signature scheme that meets this requirement except for the case in which $t = 2$.[4]

3.3 Our scheme

The difficulty of building a threshold G-DSA threshold signature scheme results from the fact that it requires adding, multiplying, and inverting shared secrets. With Shamir's secret sharing scheme, shares can easily be added, but multiplication and inversion cause the degree of the shared polynomial to increase. Indeed, this is the basis of Gennaro *et al.*'s protocol in [18, 19], and the reason why it requires $t = 2s+1$ participants to construct a signature.

Mackenzie and Reiter use multiplicative secret sharing which makes multiplication and inversion of secrets easy. Addition of secrets is now more difficult, and to

[4]For the sake of completeness, we mention that Langford presents a scheme for DSA that achieves this with the help of a trusted dealer that precomputes and distributes shares for every signature that will ever be created. Note that this is quite different than a trusted dealer on setup, as the trusted dealer has to distribute shares for every signature that will be generated. The protocol does not solve the difficulties associated with DSA threshold signatures, but instead relies on a trusted dealer to precompute them. Its limitations make it not useful in the Bitcoin context.

get around this, they use an additively homomorphic encryption scheme. Their scheme is specifically for the two party case, and one of the two parties has a decryption key for the additively homomorphic scheme. This party uses its secret share to compute a partial signature, encrypts the partial signature, and sends the resulting encrypted values to the second party. The second party cannot learn the value of the encrypted partial signature, but it uses its share of the key to contribute its portion of the signature to the ciphertext (as the scheme is additively homomorphic), and then sends the resulting ciphertext back to the first party. The first party then decrypts the ciphertext to reveal the signature. The scheme also incorporates zero knowledge proofs to prove that each party is following the protocol and that the encrypted values that they produce are consistent with well-formed shares (i.e. it is secure against malicious parties).

Our scheme generalizes Mackenzie and Reiter's scheme to the t-of-t case. The intuitive idea is that t parties pass around the ciphertext and do computations on it with their share, and also construct zero knowledge proofs that their values are consistent. As in Mackenzie and Reiter's scheme, the end result is a ciphertext which is an encryption of the signature. However, whereas in the two party case, one of the parties held the decryption key and can single-handedly decrypt the final signature, in the t party case, the homomorphic decryption key is itself distributed among the parties such that all of them have to cooperate to decrypt the key.

Our protocol proceeds in $3t-2$ rounds, after which the parties have an encrypted signature which they can then jointly decrypt.

Our protocol works for t-out-of-t case, and we use standard combinatorial structures we show how to use the t-out-of-t scheme to build a t-out-of-n scheme.

Assumptions and Setup. We assume that there are t participants $P_1,\ldots,P_t$ initialized as follows:

- Participant P_i holds a value $x_i \in Z_q$ chosen uniformly at random. The secret key is $x = \Pi_i x_i \bmod q$ and the public key is $y = G^x$ in $\mathscr{G}$. We assume the values $y_i = G^{x_i}$ are public.

- There is a separate public key additively homomorphic encryption scheme E, whose secret key D is shared in a t-out-of-t fashion among the participants. The encryption scheme is homomorphic modulo a large integer N: i.e. given $\alpha = E(a)$ and $\beta = E(b)$, where $a,b \in Z_N$, there is an efficiently computable operation $+_E$ over the ciphertext space such that

$$\alpha +_E \beta = E(a+b \bmod N)$$

  Note that if x is an integer, given $\alpha = E(a)$ we can also compute $E(xa \bmod N)$ efficiently. We refer to this operation as $x \times_E \alpha$.

  We denote the message space of E by $\mathscr{M}_E$ and the ciphertext space by $\mathscr{C}_E$.

  We will choose N large enough so that operations modulo N will not "wrap around" and will be consistent to doing them over the integers (that's because we are interested in really doing the operations modulo q, the order of the group). This requires $N > q^{3t+3}$.

- The participants are associated to signature public keys. We assume that they sign every message. In the following the signature is implicitly contained in the messages and verified by each participant upon receipt of a signed message.[5]

With and without a trusted dealer In our protocol, we have two different shared keys, the main G-DSA key and the key for the threshold homomorphic encryption scheme, and an important question is how these shares are generated and distributed.

The simplest way to do this involves a trusted dealer who begins with the constructed key, generates the shares, and distributes them to each party. Of course this has a weakness in that the trusted dealer is a single point of failure. A more sophisticated approach eliminates the trusted dealer and allows the parties to generate shares of a key in a distributed manner without ever constructing the key in the process.

Neither approach is strictly better than the other. Although having a trusted dealer is a weakness, in some cases it is strictly necessary. A dealerless protocol allows the parties to generate a new key, but it does not allow players to distribute an already existing key. In particular, in the Bitcoin context, if someone already has an address that they want to later add threshold security to, they would use the trusted dealer protocol to generate shares from the existing private key.

When generating a new address, however, the dealerless protocol is generally superior. Mackenzie and Reiter show how to distribute key shares without a trusted dealer in the two party case, and their scheme can be extended to the multiparty case. For the homomorphic encryption scheme, one good candidate is the Paillier encryption scheme, and techniques exist in the literature to distribute Paillier key shares without a trusted dealer (See [21, 22, 23]).

Threshold signature protocol. We now present our threshold G-DSA protocol. The protocol proceeds in

[5] In our protocol participant P_i will forward to P_j something he received from P_ℓ. By verifying P_ℓ's signature on the forwarded message, P_j is guaranteed of its authenticity.

rounds, where each player receives some input, performs some computation, and then passes along the output of this computation. There are n players, $P_1,\dots,P_n$. In our protocol, players $P_2,\dots,P_{n-1}$ have completely symmetric roles. That is, they all receive inputs of identical form from the previous player, run the same algorithm, and pass along the message to the next player. However, the computation done by P_1 and P_n is not identical.

We stress, however, that while from a computational perspective not all players have the same role, from a security perspective, all players are identically secure in the same threat model. No player is privileged or trusted in any manner. Consequently, from a security perspective it makes absolutely no difference how the players are numbered and which players are given the roles of P_1 and P_n.

- Round 1

  On input the message M, participant P_1

  - chooses $k_1 \in_R Z_q$ and computes $z_1 = k_1^{-1} \bmod q$
  - computes $\alpha_1 = E(z_1)$ and $\beta_1 = E(x_1 z_1 \bmod q)$
  - sets $\hat{\alpha}_1 = \hat{\beta}_1 = \perp$
  - sends $M, \alpha_1, \beta_1, \hat{\alpha}_1, \hat{\beta}_1$ to P_2

- Rounds 2 to $t-1$

  At round $i = 2,\dots,t-1$, on input the message $M, \alpha_1,\dots,\alpha_{i-1}, \beta_1,\dots,\beta_{i-1}$, $\hat{\alpha}_1,\dots,\hat{\alpha}_{i-1}$, $\hat{\beta}_1,\dots,\hat{\beta}_{i-1}$, participant P_i

  - abort if $\alpha_1,\dots,\alpha_{i-1},\beta_1,\dots,\beta_{i-1},\hat{\alpha}_1,\dots,\hat{\alpha}_{i-1},\hat{\beta}_1,\dots,\hat{\beta}_{i-1} \notin \mathscr{C}_E$
  - chooses $k_i \in_R Z_q$ and computes $z_i = k_i^{-1} \bmod q$
  - computes $\alpha_i = z_i \times_E \alpha_{i-1}$ and $\beta_i = (x_i z_i \bmod q) \times_E \beta_{i-1}$
  - computes $\hat{\alpha}_i = E(z_i)$ and $\hat{\beta}_i = E(x_i z_i \bmod q)$
  - sends $M, \alpha_1,\dots,\alpha_i, \beta_1,\dots,\beta_i, \hat{\alpha}_1,\dots,\hat{\alpha}_i, \hat{\beta}_1, \dots, \hat{\beta}_i$ to P_{i+1}

- Round t

  On input the message $M, \alpha_1,\dots,\alpha_{t-1}$, $\beta_1,\dots,\beta_{t-1}$, $\hat{\alpha}_1,\dots,\hat{\alpha}_{t-1}$, $\hat{\beta}_1,\dots,\hat{\beta}_{t-1}$, participant P_t

  - abort if $\alpha_1,\dots,\alpha_{t-1},\beta_1,\dots,\beta_{t-1}, \hat{\alpha}_1,\dots,\hat{\alpha}_{t-1},\hat{\beta}_1,\dots,\hat{\beta}_{t-1} \notin \mathscr{C}_E$
  - chooses $k_t \in_R Z_q$ and computes $z_t = k_t^{-1} \bmod q$
  - computes $R_t = G^{k_t}$ in $\mathscr{G}$
  - sends R_t to P_{t-1}

- Rounds $t+1$ to $2t-2$

  At round $t+i$ for $i = 1,\dots,t-2$, on input the message $R_t,\dots,R_{t-i+1}$, participant P_{t-i}

  - computes $R_{t-i} = R_{t-i+1}^{k_{t-i}}$ in $\mathscr{G}$
  - sends $R_t,\dots,R_{t-i}$ to P_{t-i-1}

- Round $2t-1$

  On input the message $R_t,\dots,R_2$, participant P_1

  - computes $R_1 = R_2^{k_1}$ in G.
  - computes the ZK proof Π_1 which states
    - $\exists\, \eta_1, \eta_2 \in [-q^3, q^3]$ such that
    - $R_1^{\eta_1} = R_2$ and $G^{\eta_2/\eta_1} = y_1$
    - $D(\alpha_1) = \eta_1$ and $D(\beta_1) = \eta_2$
  - sends R_1, Π_1 to P_2

- Round $2t+i-2$ for $i = 2,\dots,t-1$

  On input $R_1,\dots,R_{i-1}$, $\Pi_1,\dots,\Pi_{i-1}$, participant P_i

  - computes the ZK proof Π_i which states
    - $\exists\, \eta_1, \eta_2 \in [-q^3, q^3]$ such that
    - $R_i^{\eta_1} = R_{i+1}$ and $G^{\eta_2/\eta_1} = y_i$
    - $D(\alpha_i) = \eta_1 D(\alpha_{i-1})$ and $D(\beta_i) = \eta_2 D(\beta_{i-1})$
    - $D(\hat{\alpha}_i) = \eta_1$ and $D(\hat{\beta}_i) = \eta_2$
  - sends $R_1,\dots,R_i$, $\Pi_1,\dots,\Pi_i$ to P_{i+1}

- Round $3t-2$

  On input $R_1,\dots,R_{t-1}$, $\Pi_1,\dots,\Pi_{t-1}$, participant P_t

  - choose $c \in_R Z_{q^{3t-1}}$
  - computes $m = H(M)$ and $r = H'(R_1) \in Z_q$
  - computes $\hat{\mu} = E(z_t)$
  - computes $\mu = [(m z_3 \bmod q) \times_E \alpha_{t-1}] +_E [(r x_3 z_3 \times_E \beta_{t-1}] +_E E(cq)$
  - computes the ZK proof Π_t which states
    - $\exists\, \eta_1, \eta_2 \in [-q^3, q^3]$ such that
    - $R_t^{\eta_1} = G$ and $G^{\eta_2/\eta_1} = y_t$
    - $D(\mu) = m\eta_1 D(\alpha_{t-1}) + r\eta_2 D(\beta_{t-1})$
    - $D(\hat{\mu}) = \eta_1$
  - sends $\mu, \hat{\mu}, \Pi_i, \dots, \Pi_t$ to all the other participants

- Final Decryption Rounds

  At the end of the protocol, each player should have a proof from every other player. They must verify these proofs and abort if the verification fails[6].

[6]We aimed to simplify the communication channel, but if there is a broadcast channel, each player can directly broadcast its proof to all other players.

The participants invoke the distributed decryption protocol for D over the ciphertext μ. Let $s = D(\mu) \bmod q$. The participants output (r,s) as the signature for M.

Encryption Scheme As in [7] we instantiate E with Paillier's encryption scheme [24]. We recall the scheme here.

- Key Generation: generate two large primes P,Q of equal length. and set $N = PQ$. Let $\lambda(N) = lcm(P-1, Q-1)$ be the Carmichael function of N. Finally choose $g \in Z^*_{N^2}$ such that its order is a multiple of N. The public key is (N,g) and the secret key is $\lambda(N)$.
- Encryption: to encrypt a message $m \in Z_N$, select $x \in_R Z^*_N$ and return $c = g^m x^N \bmod N^2$.
- Decryption: to decrypt a ciphertext $c \in Z_{N^2}$, let L be a function defined over the set $\{u \in Z_{N^2} : u = 1 \bmod N\}$ computed as $L(u) = (u-1)/N$. Then the decryption of c is computed as $L(c^{\lambda(N)})/L(g^{\lambda(N)}) \bmod N$.
- Homomorphic Properties: Given two ciphertexts $c_1, c_2 \in Z_{N^2}$ it is easy to see that $c_1 +_E c_2 = c_1 c_2 \bmod N^2$ (If $c_i = E(m_i)$ then $c_1 +_E c_2 = E(m_1 + m_2 \bmod N)$. Similarly, given a ciphertext $c = E(m) \in Z_{N^2}$ and a number $a \in Z_n$ we have that $a \times_E c = c^a \bmod N^2 = E(am \bmod N)$.

We point out that threshold variations of Paillier's scheme have been presented in the literature [21, 22].

Zero-knowledge proofs. The ZK proof Π_1 is already described in [7] (as ZK proof Π in their paper). Similarly the ZK proof Π_t is described as Π' in [7].

We now describe the ZK proof Π_i used by the intermediate participants. This is always the same proof $\hat{\Pi}$ called on different inputs. As in [7] we make use of an auxiliary RSA modulus $\tilde{N}$ which is the product of two safe primes $\tilde{N} = \tilde{P}\tilde{Q}$ and two elements $h_1, h_2 \in Z^*_{\tilde{N}}$ used to construct range commitments.

For public values $c, d, w_1, w_2, m_1, m_2, m_3, m_4, m_5, m_6$ we construct a ZK proof $\hat{\Pi}$ that proves

- $\exists\, x_1, x_2 \in [-q^3, q^3]$ such that
- $c^{x_1} = w_1$ and $d^{x_2/x_1} = w_2$
- $D(m_1) = x_1 D(m_3)$ and $D(m_2) = x_2 D(m_4)$
- $D(m_5) = x_1$ and $D(m_6) = x_2$

The protocol is as follows. We assume the Prover knows the values $r_5, r_6 \in Z^*_N$ such that $m_5 = g^{x_1} r_5^N \bmod N^2$ and $m_6 = g^{x_2} r_6^N \bmod N^2$. Moreover, the proof that we give is non-interactive. It relies on using a hash function to compute the challenge, e, and it is secure in the Random Oracle Model.

The prover chooses uniformly at random:

$$\alpha, \delta \in Z_{q^3} \qquad \beta_1, \beta_2 \in Z^*_N \qquad \rho_3, \varepsilon \in Z_q$$
$$\rho_1, \rho_2 \in Z_{q\tilde{N}} \qquad \gamma, \nu \in Z_{q^3\tilde{N}}$$

The prover computes

$$\begin{array}{ll}
z_1 = h_1^{x_1} h_2^{\rho_1} \bmod \tilde{N} & v_1 = d^{\delta+\varepsilon} \text{ in } \mathcal{G} \\
u_1 = c^{\alpha} \text{ in } \mathcal{G} & v_2 = w_2^{\alpha} d^{\varepsilon} \text{ in } \mathcal{G} \\
u_2 = g^{\alpha} \beta_1^N \bmod N^2 & v_3 = m_3^{\alpha} \bmod N^2 \\
u_3 = h_1^{\alpha} h_2^{\gamma} \bmod \tilde{N} & v_4 = m_4^{\delta} \bmod N^2 \\
z_2 = h_1^{x_2} h_2^{\rho_2} \bmod \tilde{N} & v_5 = g^{\delta} \beta_2^N \bmod N^2 \\
y = d^{x_2+\rho_3} \text{ in } \mathcal{G} & v_6 = h_1^{\delta} h_2^{\nu} \bmod \tilde{N}
\end{array}$$

$$e = \mathtt{hash}(c, d, w_1, w_2, m_1, m_2, m_3, m_4, m_5, m_6, z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, v_3, v_4, v_5, v_6)^{7}$$

$$\begin{array}{ll}
s_1 = e x_1 + \alpha & t_2 = e\rho_3 + \varepsilon \\
s_2 = (r_5)^e \beta_1 \bmod N & t_3 = (r_6)^e \beta_2 \bmod N \\
s_3 = e\rho_1 + \gamma & t_4 = e\rho_2 + \nu \\
t_1 = e x_2 + \delta &
\end{array}$$

The prover sends all of these values to the Verifier.

The Verifier checks that all the values are in the correct range and moreover that the following equations hold

$$\begin{array}{ll}
u_1 = c^{s_1} w_1^{-e} \text{ in } \mathcal{G} & v_4 = m_4^{t_1} m_2^{-e} \bmod N^2 \\
u_2 = g^{s_1} s_2^N m_5^{-e} \bmod N^2 & v_5 = g^{t_1} t_3^N m_6^{-e} \bmod N^2 \\
u_3 = h_1^{s_1} h_2^{s_3} z_1^{-e} \bmod \tilde{N} & v_6 = h_1^{t_1} h_2^{t_4} z_2^{-e} \bmod \tilde{N} \\
v_1 = d^{t_1+t_2} y^{-e} \text{ in } \mathcal{G} & e = \mathtt{hash}(c, d, w_1, w_2, \\
v_2 = w_2^{s_1} d^{t_2} y^{-e} \text{ in } \mathcal{G} & m_1, m_2, m_3, m_4, m_5, m_6, \\
v_3 = m_3^{s_1} m_1^{-e} \bmod N^2 & z_1, u_1, u_2, u_3, z_2, y, v_1, v_2, \\
 & v_3, v_4, v_5, v_6)
\end{array}$$

See Figure 1 for a diagram of the protocol in the three party case.

[7]This is the step of the proof that relies on the Random Oracle Model. We can construct the proof without random oracles using an interactive proof. In the interactive version of the proof, the Prover sends all of the values computed until this point. The Verifier then issues a challenge e, and the proof proceeds exactly as in the non-interactive version.

***t*-out-of-*n* threshold signature scheme.** A t-out-of-n scheme can be obtained by considering all possible subsets of t participants and instantiating the above protocol for each subset. We stress that the performance of the t-of-n protocol depends on t and not on n. The only performance overhead of t-of-n over t-of-t is identifying the proper t-of-t share to use.

One possible optimization is that the n participants can use a single encryption key E (rather than one for each subset), where the secret key D is shared in a t-out-of-n fashion among the participants. Again we point out that this exists for Paillier [21, 22].

3.4 Size of shares

The combinatorial structure to go from t-out-of-t to t-out-of-n requires $O(n^t)$ storage, making it feasible only for small values of n and t. It is an interesting open question to construct threshold DSA signature scheme that does not require storage that is exponential in t.

Interestingly, every application of threshold security to Bitcoin appears to be easily capable of handling the combinatorial structure, for one of two reasons.

1. Many applications require (t,t) sharing and not (t,n) for $t < n$. The (t,t) case does not use the combinatorial structure and thus only requires a single key share stored by each party. Indeed, ours is the first work to propose a (t,t) threshold DSA signature scheme for $t > 2$.

2. Even for our applications that do require a (t,n) signature, the values of t and n are inherently very small due to the nature of security policies used in practice (Section 5.2).

3.5 Security Analysis

A detailed security analysis will be presented in the final version, however it is not hard to see that the security proof follows the same lines of the proof in [7], and therefore the security of the entire distributed DSA signature scheme can be reduced to (i) the unforgeability of the DSA signature scheme; (ii) the semantic security of the Paillier encryption scheme (which we recall is equivalent to the N-residuosity assumption modulo N^2) and (iii) to the Strong-RSA Assumption (modulo $\tilde{N}$).

More specifically we prove *existential unforgeability against chosen message attack*, the strongest security notion for signatures. In the distributed case consider an adversary $\mathscr{A}$ controlling $t-1$ players. Even after the entire set of t parties signs ℓ messages $M^{(1)},\ldots,M^{(\ell)}$ chosen by $\mathscr{A}$, it should be computationally infeasible for $\mathscr{A}$ to compute a valid signature on a message $M \neq M_i$. We prove that this is the case by a simulation argument which shows that if such an adversary $\mathscr{A}$ exists then there exists a forger $\mathscr{F}$ that can forge a signatures in the underlying "centralized" DSA signature scheme. Since we assume the latter to be unforgeable, then the former cannot happen. We assume a static corruption model, in which $\mathscr{A}$ assumes control of $t-1$ players at the beginning of the protocol.

So let us assume by contradiction that $\mathscr{A}$ exists and show how to construct $\mathscr{F}$. This forger $\mathscr{F}$ also works in the *chosen message attack model*, i.e. on input a DSA public key y, it has access to a "signature oracle" which on input $\hat{M}$ returns the signature $\hat{r},\hat{s}$ under the public key y.

$\mathscr{F}$ runs on input y, the public key of the underlying DSA scheme. It will initiate $\mathscr{A}$ and assume the role of P_i the only honest player not corrupted by $\mathscr{A}$.

Key Generation. Assuming a trusted party initialization of the system, $\mathscr{F}$ will create the public key for the pk for the encryption scheme E and share sk among the players. Note that $\mathscr{F}$ knows sk. Then it will generate random values $x_j \in Z_q$ as the secret share of player P_j; it will compute $\lambda = (\Pi_{j\neq i} x_j)^{-1}$ mod q and will set $y_i = y^{\lambda}$.

Signature Generation. When $\mathscr{A}$ requests the signature of a message M, the forger $\mathscr{F}$ will query its signature oracle and get r,s. Let $R = G^{H(M)s^{-1}} y^{rs^{-1}}$. We now show how to simulate a signature protocol so that it results in this signature being output.

Simulating r. The players run the protocol up to Round $2t-1$ with the difference that at Round i the Forger encrypts arbitrary values (e.g. 0) in the $\alpha_i, \hat{\alpha}_i, \beta_i, \hat{\beta}_i$ ciphertexts. Note that at the end of Round t, the Forger knows all the values k_j chosen by $\mathscr{A}$ (since he knows the sk). At round $2t-i$ when $\mathscr{F}$ has to announce R_i it will it will compute $\lambda' = (\Pi_{j\neq i} k_j)^{-1}$ mod q and will set $R_i = R^{\lambda'}$.

Simulating s. The players run the protocol from Round $2t-1$ to the end. The forger $\mathscr{F}$ will simulate the ZK proof Π_i, since it is now proving an incorrect statement. Over the final ciphertext μ, the forger will now simulate the distributed decryption protocol for E so that it results in a value $s' \in Z_{q^{3t}}$ s.t. $s' = s$ mod q

In order to conclude the proof we must argue that the above simulation is indistinguishable from a real execution of the protocol. Indeed only under this condition we can claim that $\mathscr{A}$ will output a forgery, and therefore $\mathscr{F}$ will succeed.

We note that the above simulation differs from the real execution in three main points

- The final decryption protocol is simulated to "hit" a specific value, instead of the correct decryption of the ciphertext μ. But if the threshold encryption scheme used to do distributed decryption is secure, then this step is indistinguishable from the real-life protocol.

- the ZK proof Π_i is simulated. Due to the zero-knowledge properties, a simulated proof is indistinguishable from the real one.

- The simulated ciphertexts $\alpha_i, \hat{\alpha}_i, \beta_i, \hat{\beta}_i$ sent by $\mathscr{F}$ encrypt values with a different distribution that in the real protocol. But if E is semantically secure then these simulated ciphertexts are computationally indistinguishable from the real ones. Note that this requires another reduction, where we use $\mathscr{A}$ to break the encryption scheme E (in this case the simulation knows the secret key x of the DSA scheme, but does not know sk).

4 Threshold wallets

In this section, we describe a set of threshold-signature based protocols that will allow both businesses and individuals to secure their Bitcoin wallets. Before presenting the protocols, we discuss the threat model.

4.1 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 2.1), we will use the term in the rest of the paper in a more precise sense.

Definition 2 (wallet) *A collection of addresses with the same security policy together with a software program or protocol that allows spending from those addresses in accordance with that policy.*

"Security policy" encompasses the ownership or access-control list 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.

Definition 3 (Hot wallet/Cold wallet) *A* hot wallet *is a wallet from which bitcoins can be spent without accessing cold storage. Conversely, a* cold wallet *is a wallet from which bitcoins cannot be spent without accessing cold storage.*

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

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 bitcoins online without having to access 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. Protecting hot wallets from internal attackers is the most pressing problem. Our central claim 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 in Section 5 address the protection of hot wallets from both insider and external attacks.

4.2 DNF wallets and threshold wallets

For a typical Bitcoin wallet, the security policy specifies that a single user logs on to their computer (if they manage their own wallet) or authenticate to some web service (if they use an online provider). But, security policies can be far more complex, and they can involve multiple participants.

Given a set of participants $P_1, \ldots, P_n$, a security policy may specify conditions of which the approval of multiple parties is required to authorize a transaction. Such a security policy can be specified as a boolean formula

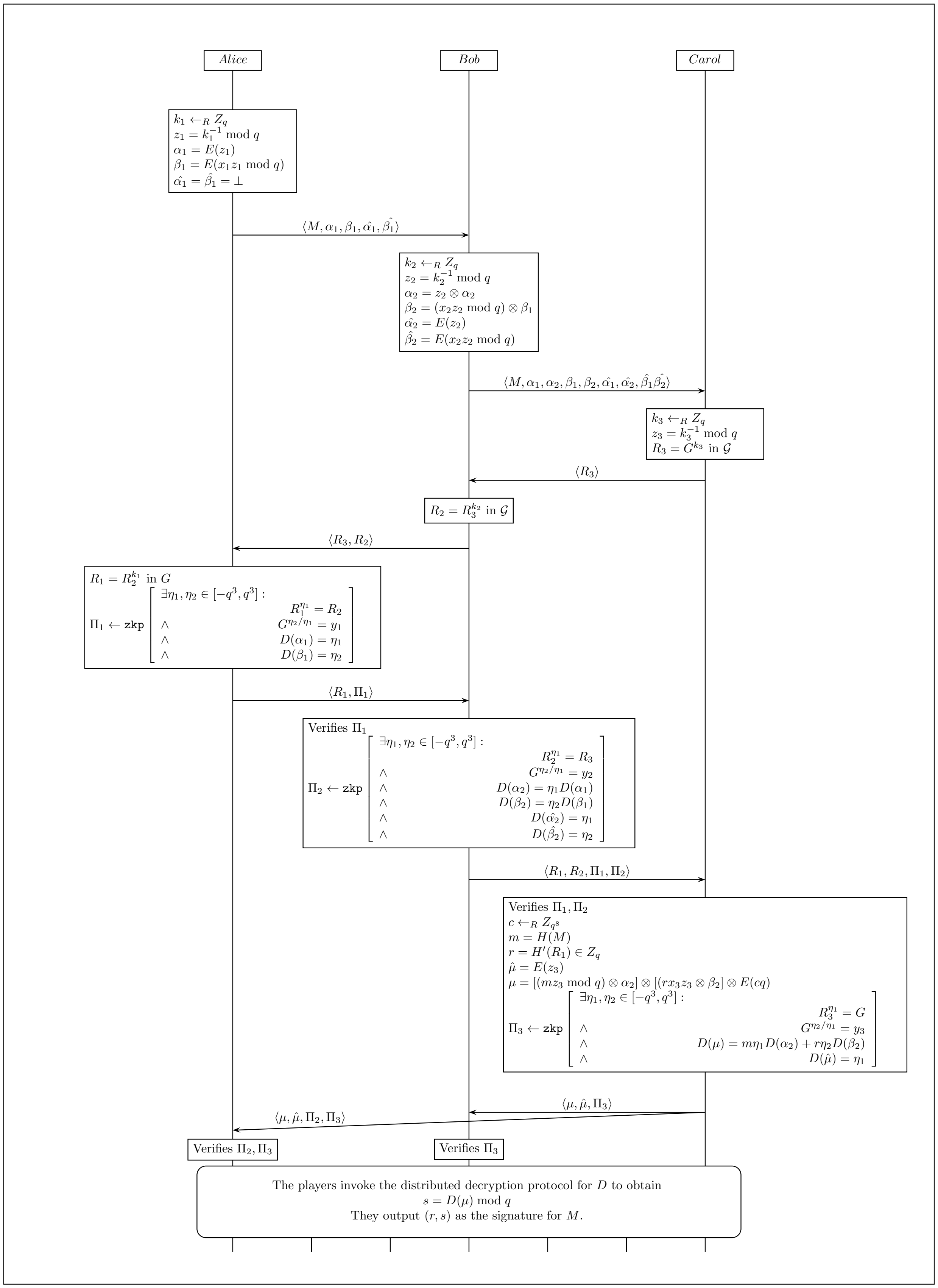

Alice
Bob
Carol
$k_1 \leftarrow_R Z_q$
$z_1 = k_1^{-1} \bmod q$
$\alpha_1 = E(z_1)$
$\beta_1 = E(x_1 z_1 \bmod q)$
$\hat{\alpha_1} = \hat{\beta_1} = \perp$
$\langle M, \alpha_1, \beta_1, \hat{\alpha_1}, \hat{\beta_1} \rangle$
$k_2 \leftarrow_R Z_q$
$z_2 = k_2^{-1} \bmod q$
$\alpha_2 = z_2 \otimes \alpha_2$
$\beta_2 = (x_2 z_2 \bmod q) \otimes \beta_1$
$\hat{\alpha_2} = E(z_2)$
$\hat{\beta_2} = E(x_2 z_2 \bmod q)$
$\langle M, \alpha_1, \alpha_2, \beta_1, \beta_2, \hat{\alpha_1}, \hat{\alpha_2}, \hat{\beta_1} \hat{\beta_2} \rangle$
$k_3 \leftarrow_R Z_q$
$z_3 = k_3^{-1} \bmod q$
$R_3 = G^{k_3}$ in $\mathcal{G}$
$\langle R_3 \rangle$
$R_2 = R_3^{k_2}$ in $\mathcal{G}$
$\langle R_3, R_2 \rangle$
$R_1 = R_2^{k_1}$ in G
$\Pi_1 \leftarrow \texttt{zkp} \begin{bmatrix} \exists \eta_1, \eta_2 \in [-q^3, q^3] : & \\ & R_1^{\eta_1} = R_2 \\ \wedge & G^{\eta_2/\eta_1} = y_1 \\ \wedge & D(\alpha_1) = \eta_1 \\ \wedge & D(\beta_1) = \eta_2 \end{bmatrix}$
$\langle R_1, \Pi_1 \rangle$
Verifies Π_1
$\Pi_2 \leftarrow \texttt{zkp} \begin{bmatrix} \exists \eta_1, \eta_2 \in [-q^3, q^3] : & \\ & R_2^{\eta_1} = R_3 \\ \wedge & G^{\eta_2/\eta_1} = y_2 \\ \wedge & D(\alpha_2) = \eta_1 D(\alpha_1) \\ \wedge & D(\beta_2) = \eta_2 D(\beta_1) \\ \wedge & D(\hat{\alpha_2}) = \eta_1 \\ \wedge & D(\hat{\beta_2}) = \eta_2 \end{bmatrix}$
$\langle R_1, R_2, \Pi_1, \Pi_2 \rangle$
Verifies Π_1, Π_2
$c \leftarrow_R Z_{q^8}$
$m = H(M)$
$r = H'(R_1) \in Z_q$
$\hat{\mu} = E(z_3)$
$\mu = [(m z_3 \bmod q) \otimes \alpha_2] \otimes [(r x_3 z_3 \otimes \beta_2] \otimes E(cq)$
$\Pi_3 \leftarrow \texttt{zkp} \begin{bmatrix} \exists \eta_1, \eta_2 \in [-q^3, q^3] : & \\ & R_3^{\eta_1} = G \\ \wedge & G^{\eta_2/\eta_1} = y_3 \\ \wedge & D(\mu) = m\eta_1 D(\alpha_2) + r\eta_2 D(\beta_2) \\ \wedge & D(\hat{\mu}) = \eta_1 \end{bmatrix}$
$\langle \mu, \hat{\mu}, \Pi_3 \rangle$
$\langle \mu, \hat{\mu}, \Pi_2, \Pi_3 \rangle$
Verifies Π_2, Π_3
Verifies Π_3
The players invoke the distributed decryption protocol for D to obtain
$s = D(\mu) \bmod q$
They output (r, s) as the signature for M.

Figure 1: 3-of-3 G-DSA Threshold Signature protocol

over $p_1, \ldots, p_n$ which are boolean variables that indicate which participants are present. A transaction is authorized if and only if the formula evaluates to `true`.

Policies whose formulas are in Disjunctive Normal From (DNF) are of particular interest to us. A DNF formula is a disjunction of conjunctions of literals, as in:

$$(p_1 \wedge p_3 \wedge p_4) \vee (p_2 \wedge p_3) \vee (p_5)$$

We refer to a wallet whose security policy is a DNF over a set of participants a *DNF wallet*. A special case of a DNF formula is the disjunction of all possible conjunctions of size t. We denote by n the total number of participants, and we call such a policy a t-of-n threshold policy as it specifies that any group of at least size t can authorize a transaction. We refer to a wallet with a t-of-n threshold policy as a (t,n)*-threshold wallet*. Moreover, in the special case in which the DNF contains just a single clause that is a conjunction of all the participants, we call this a (t,t)*-threshold wallet*.

Realizing DNF wallets and threshold wallets. First, a security policy consisting of a single conjunction $p_{i_1} \wedge \ldots \wedge p_{i_t}$ can be realized by t-out-of-t secret sharing of the wallet private key among the participants $P_{i_1} \wedge \ldots \wedge P_{i_t}$. To sign a transaction, those participants execute the threshold signature protocol.

Next, a DNF policy which is a disjunction of m conjunctions can be implemented by m separate secret sharings (with possibly different thresholds) of the key. When a group of participants wish to sign, they first determine one of the conjunctions that is satisfied (recall that participants in our threshold signature protocol know each other's identities). Once they find a conjunction, they use the corresponding set of shares.

Finally, a t-out-of-n threshold policy is realized as a DNF policy with $\binom{n}{t}$ different conjunctions, one corresponding to each subset of t participants. This is equivalent to using the t-out-of-n signature scheme that we described.

Note that a wallet has many addresses and many private keys associated with them, but the above description refers to a single wallet private key. In Section 4.4 we show how a single set of key shares can control arbitrarily many addresses in a shared wallet.

4.3 Comparison with multisignature approach

While most Bitcoin transactions are spent with a single signature, Bitcoin in fact specifies a script written in a stack-based programming language which defines the conditions under which a transaction may be redeemed. This scripting language includes support (`OP_CHECKMULTISIG`) for *multisignature* scripts [25] which require at least t of n specified ECDSA public keys to provide a signature on the redeeming transaction. By default, multisignature 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. When this is used, senders specify a script hash, and the exact script is provided by the recipient when funds are redeemed. This enables multisignature 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 multisignature transactions. However, due to a hard-coded limit on the overall size of a hashed script, the recipients are still limited to $n \leq 15$.

4.3.1 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).

4.3.2 Advantages of threshold signatures.

We argue that threshold signatures offer fundamental advantages stemming from the fact that in the multisignature approach, the access-control policy is encoded in the transaction and eventually publicly revealed:

Flexibility. Threshold signatures are more flexible than multisignatures in the access policies that they permit as well as in the ability to modify the access policies.

Whereas threshold signatures can realize any DNF-wallet structure, multisignatures are far less expressive and only allow for t-of-n access structures.

Threshold signatures also allow more flexibility for making changes to the access control policy. If a business using multisignature 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 as it requires moving funds to a new address with each policy update. For some business practices, the ability to have a static address is fundamental. As an example, consider an organization that prints promotional materials with a donation address on it. Multisignatures would not allow them to change the access control policy while keeping that address.

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 can maintain the address even if the access control policy changes.

More generally, it is impossible to add multisignature security to an existing address since the two types of addresses are syntactically distinct. The only way to attain multi-factor security is to create a new multisignature address. Threshold signatures, on the other hand, allow one to split up the key of an existing address.

Anonymity. While Bitcoin allows users to be pseudonymous, it does not provide any anonymity guarantees. Indeed, it has been shown that it is not difficult to link various addresses belonging to a single user [26]. Moreover, because the entire transaction log is public, once an address has been associated with a real world identity, one can immediately view every other transaction associated with that address.

Because of Bitcoin's inherent lack of anonymity, various techniques have been developed to provide additional anonymity for Bitcoin users. Three of the most prominent techniques are Mixcoin [**?**], CoinJoin [**?**], and the use of change addresses. We show now that none of these techniques are compatible with multisignatures, while they all work as intended with threshold signatures.

As we mentioned in Section 2.1, for purposes of increasing anonymity, the general practice is to use newly generated change addresses which cannot easily be linked to the input addresses [26]. With multisignature transactions, unlinkable change addresses are much harder to achieve. Suppose Alice uses multisignature-based security and makes a purchase. Then the spending address(es) and change address will all have the same t-of-n access control structure, whereas the destination address most likely will not. This allows easily linking Alice's input and output addresses. With threshold signatures, on the other hand, change addresses will be unlinkable when sending funds to any regular (single-key) address or other threshold address (though not when interacting with multisignature addresses or other script hash addresses). In particular, change addresses will provide the exact same benefits with threshold signatures as they do with a single-key address. In Section 4.4 we show how to generate change addresses with equivalent threshold access control without distributing new shares.

Mixcoin and CoinJoin are both based on the technique of mixing, or shuffling the inputs amongst multiple users. Both protocols proceed in independent rounds. During a single Mixcoin round, each user sends a fixed amount of coins to a mixing party which sends the same amount of coins back to a fresh address provided by that user. CoinJoin is also based on the mixing idea but instead of having a centralized mixing party, users combine their inputs and outputs into a single joint transaction that they all sign. Once coins have been mixed with either protocol, it becomes nearly impossible to identify the mapping between input and output addresses.

Consider what happens, however, when one tries to use either Mixcoin or CoinJoin with multisignature addresses. Both of these protocols rely on the fact that all of the input and output addresses are structurally identical and that there is an abundance of such addresses. In order to maintain multisignature security, both the input and output addresses will have to be multisignature addresses. Moreover, they will have to have the same access structure (i.e. the same t and n). Multisignature addresses cannot be mixed together with regular addresses as it is trivial to link an input address with an output address by just examining the access structure. Moreover, it is highly unlikely that there will be a sufficient number of addresses with a given access structure that are interested in mixing to facilitate mixing each type of address on its own. [8]

Multisignatures also cause a loss of anonymity since the access structure is published on the block chain. 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 reveal that access control is being used at all.

Advantages of multi-signatures. Multi-signature transactions have a 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. The performance of multisignatures is also an advantage. Whereas our threshold signature scheme requires zero knowledge proofs which are computationally expensive, multisignatures only require t standard ECDSA signatures.

[8]One might be tempted to suggest that funds be temporarily transferred to a single signature address for mixing. This is problematic for two reasons, however. Firstly, transferring to a single signature address introduces a single point of failure as the bitcoins can be stolen during this period if the key is compromised. Moreover, one can do second-order analysis and still link the input and output addresses by examining the access structure of the multisignature addresses that transferred bitcoins to the input address and received bitcoins from the output address.

4.4 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 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.

Deterministic wallets

Deterministic wallets [28] 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.[9]

We present a construction that realizes deterministic wallets for shared addresses. In particular, consider a public key, pk_{mas}, for which the corresponding private key, sk_{mas} is shared in a (t,n) manner amongst n players — the jth participant has key share $sk_{mas}^{(j)}$ We construct a threshold scheme for deriving the ith child key pair (pk_i, sk_i)[10]such that the following properties hold:

- For a given index i, any participant can independently generate pk_i from pk_{mas}
- For a given index i, participant P_j can independently generate its new key share $sk_i^{(j)}$ from its master key share $sk_{mas}^{(j)}$
- The resulting key pair (pk_i, sk_i) is shared with the identical threshold properties of the master pair (pk_{mas}, sk_{mas})

It is clear from the above specification that for each additional child address, participants must only store the index i as the rest can be derived. We now present the details of our construction, and we stress that it is compatible both with the 2-of-2 threshold signature scheme in [7] as well as with our generalized (t,t) and (t,n) schemes.

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

[10]In our scheme, we refer to the public keys rather than addresses, but the address is computed by simply hashing the public key

Threshold deterministic address derivation

We refer to a curve of order n with base point G. We define a function $f(P)$, which outputs a serialized form (that can be input into a hash function) of the curve point P. As before, the master private key is sk_{mas}, and corresponding master public key is $pk_{mas} = sk_{mas} \cdot G$. Our construction also refers to c, a 256-bit nonce that is chosen uniformly at random. c is known to each of the participants but is not shared publicly. We also use a hash function H that maps arbitrary input strings to 256-bit output strings.

We assume that sk_{mas} is multiplicatively shared amongst t participants as in [7] and in our ECDSA protocol. In particular, participant P_j has a share $sk_{mas}^{(j)}$ of sk_{mas}.

When the master key pair (pk_{mas}, sk_{mas}) is initially shared among the t participants, one of the participants is designated as the leader L. The child key pair derivation protocol consists of the following two functions:

Child Key Share Derivation Function for player P_j:

- If $P_j = L$
  - compute $T = H(c||f(pk_{mas})||i)$
  - $sk_i^{(j)} = sk_{mas}^{(j)} \cdot T$
- else
  - $sk_i^{(j)} = sk_{mas}^{(j)}$

Public Key Derivation Function:

- compute $T = H(c||f(pk_{mas})||i)$
- $pk_i = pk_{mas} \cdot T$

It is clear that the t participants now hold multiplicative shares of sk_i, the private key corresponding to the derived public key pk_i. In particular, we have:

$$\begin{aligned} pk_i &= pk_{mas} \cdot T \\ &= sk_{mas} \cdot G \cdot T \\ &= \left[\prod_{j=1}^{t} sk_{mas}^{(j)}\right] \cdot T \cdot G \\ &= \left[\prod_{j=1}^{t} sk_i^{(j)}\right] \cdot G \\ &= sk_i \cdot G \end{aligned}$$

It is worth noting that both of the derivation functions are non-interactive. Participants can derive their shares of the new private key as well as the public key completely on their own.

Security analysis. 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 child key share derivation protocol or the subsequent signature generations is either key constructed.

Note that for all participants other than the leader, the child key share is identical to its master key share. For the leader the child key share is its master share multiplied by T. Since for all players, the child key share is fully dependent on the master key share, an adversary that does not know the master key share cannot derive the child key share (even if the adversary knows T).

It is important that the nonce be kept 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 purpose 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).

Note that if an adversary compromises any one of the participants, the adversary will learn c and all of the i's that are used, and they can therefore link all of the accounts addresses. This is acceptable as even in the non-deterministic wallet version, it is expected that participants are aware of and store the public key associated with their share. Importantly, there is no security risk as the adversary cannot learn the private key unless it has compromised all t participants.

If a breach of any of the participants is detected, the key should be re-shared and all participants should delete their old shares so that the share that the adversary learns is no longer useful. Note that as long as even a single participant successfully deletes its share, the compromised share is no longer useful. Moreover, to recover from a breach, a new c should be chosen so that the adversary cannot link freshly generated addresses to the other addresses.

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. We have shown the scheme for a (t,t) shared address. For the combinatorial (t,n) scheme, the derivation function is applied to each set of (t,t) shares.

5 THRESHOLD SECURITY FOR AN ORGANIZATION

Since security is only as good as the weakest link, in this section we consider the question, *can an organization remove single points of failure in every step of the flow of bitcoins through the system — receiving bitcoins, transferring and managing them internally, and finally spending them?* We show how this can be achieved using DNF and threshold wallets, borrowing many ideas from traditional financial control mechanisms. We also show that throughout this process we can generate secure audit logs.

5.1 Receiving bitcoins securely

The first stage in the flow of bitcoins through an organization is receiving them. Most commonly, it may be a merchant receiving payment from a user, or an online wallet receiving a deposit, and so on.

Here the merchant's web server is a single point of failure. If it is compromised, when the user executes the payment protocol (BIP 70), the attacker may substitute his address for the merchant's.

Threshold cryptography once again provides a solution. Note that the goal here is only to securely communicate the receiving address to the user; how bitcoins received at that address are secured is an orthogonal question. So the threshold cryptography we use is unrelated to Bitcoin's cryptography.

In the BIP 70 protocol, the merchant authenticates himself using standard PKI, and in particular, presents an X.509 certificate that authenticates the merchant's identity. X.509 provides RSA, DSA, and ECDSA signature options. Our description references details of the BIP 70 protocol, and we refer the reader to it [30].

The merchant implements a user-facing web server and t *share servers*.

Setup. The merchant sets up shares of the private key for the X.509 certificate on each share server.

Payment protocol.

- Each share server independently generates a fresh address according to the threshold deterministic wallet protocol of Section 4.4.
- The share servers threshold-sign the address using their shares of the X.509 private key.
- The webserver executes the payment protocol as usual, except that when it computes PaymentDetails, it leaves the Output field empty (which represents the address). It then sends PaymentDetails to the share servers which fill in the newly generated address, execute a threshold signature protocol, and return it to the webserver which uses the result as the signature field in its PaymentRequest message to the user.

Not that the threshold signing of the address is just a local protocol executed by the share servers. In particular, it is not constrained by the Bitcoin protocol, and thus

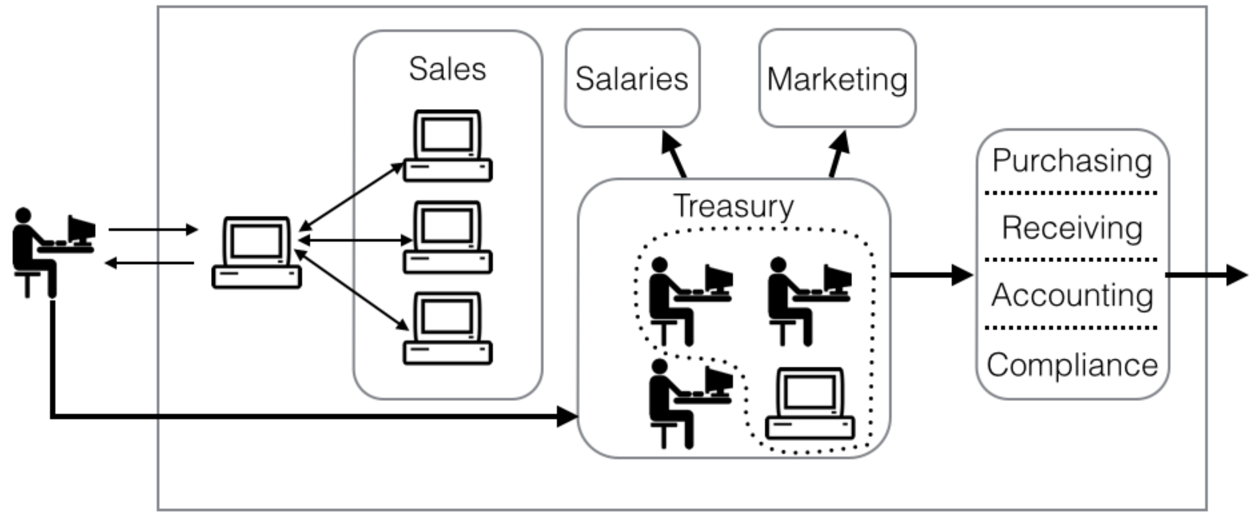


Figure 2: Removing single points of failure: illustrative example. First a customer interacts with a front-end server run by the sales department and executes the BIP 70 protocol. The payment address is generated in a 3-out-of-3 threshold fashion. The treasury department controls the address at which the user makes a payment. The treasury managers authorize disbursements to various departments in a 2-out-of-3 fashion (Section 5.2), but a compliance server must be involved for secure audits (Section 5.4). One of the departments receiving funds is purchasing, who controls their spending based on a functional-separation-of-duties security policy enforced cryptographically using distributed wallets (Section 5.3).

we can support any X.509-compatible signature. Threshold RSA signatures are well-known [31, 32], whereas threshold DSA and ECDSA signatures are known, fittingly enough, from the present work.

5.2 Bitcoins at rest: joint control

Bitcoins will spend most of their lifecycle within an organization at rest. Ideally, they will be in cold storage. Organizations may have a treasury department to centrally manage funds at this stage. The treasury manages funds and disburses them to different departments for spending as needed. These internal flows are less frequent and larger than the external flows that we'll discuss in the next subsection.

Security for bitcions at rest can be implemented using joint control or shared custody. The participants are functionally equivalent and all have access to the same information. This is a straightforward application of threshold wallets. A physical world analog would be a safe that requires two keys to be simultaneously turned.

5.3 Spending

Let us return to our motivating example: an accounting system might be willing to disburse up to \$1000 to a supplier as long as (1) there is a valid purchase order issued by the purchasing department to the supplier, and (2) the receiving department has a record of receiving the goods named in the purchase order (3) an accounts manager initiates a payment that matches the items/amount in the purchase order, and (4) the purchase order and store receipt have not already been used to authorize another disbursement.

Such business logic is traditionally handled in software by business process management software using *role-based access control*. Unfortunately, since a single OS or middleware layer must authenticate users and perform necessary checks, it inevitably introduces a single point of failure. System administrators, in particular, are typically in a position to override controls, defeating the point of functional separation. Anderson notes that such vulnerabilities are in fact very common in practice [10].

To realize functional separation despite fallible software components, we must maintain cryptographic separation between components that implement different roles/duties. This suggests separate servers for each role that together implement a DNF wallet. But first, we need some formalism. We assume that there is evolving *state* S_i associated with each participant (i.e., duty or role). For example, the state associated with the *receiving* role might consist of the set of records of all items received.

Definition 4 *A functional-separation-of-duties security policy is a set of* authorization paths *and an authorization condition corresponding to each path. An authorization path is a subset of participants; an authorization condition is boolean function that takes as input a set of partic-*

ipant states and the transaction's parameters. The policy evaluates to true if at least one of the authorization conditions does.

Under this formalism, our motivating corporate purchasing example would translate to a single authorization path. If the same corporate wallet were also used for disbursing salaries, it would result entirely different authorization path involving different participants and condition for authorization. Perhaps it is more prudent to use entirely different wallets for different purposes, but for full generality we discuss security policies that are composites of multiple separations of duty.

Definition 5 *To say that a wallet implements a policy means that signing a given transaction is possible if and only if: (1) all participants involved in one of the authorization paths participate in the signature protocol and (2) the corresponding authorization condition over those participants' inputs and the transaction's parameters evaluates to true.*

DNF wallets are a good fit for implementing functional-separation-of-duties security policies; we just need the participants to execute a preliminary step to verify the authorization condition before proceeding with the signature step. We call this an *extended DNF wallet.* Concretely:

Setup. For each authorization path A_i, we create a separate $|A_i|$-out-of-$|A_i|$ sharing of its private key and distribute shares to participants involved in A_i.

Spend. One of the participants *initiates* a spend by proposing a transaction and and sending a message to participants on a potential authorization path. The participants on that path first execute a joint protocol to verify the authorization condition based on their inputs and proposed transaction parameters. Each participant aborts if the check fails, otherwise proceeds to the threshold signature step. ■

How the authorization condition is verified is left intentionally unspecified. The participants could conceivably broadcast their state files to each other, since there is no secrecy requirement over these inputs. In practice we would use an efficient protocol that exchanges only the state relevant to the transaction in question, such as the appropriate purchase record.

Observe that functional separation of duties maps precisely to DNF formulas. If we instead had a CNF wallet, for example, it is not clear how to realize functional separation. This observation motivates:

Claim 1 *Extended DNF wallets efficiently implement functional-separation-of-duties security policies. By efficient we mean that the total size of each participant's shares is linear in the size of the policy.* ■

Back to our example: to raise a purchase order, the purchase clerk creates a record (supplier_id, order_id, item_id, amount, payment_address) in the purchase server. When the receiving clerk receives an item, he creates a record (supplier_id, order_id, item_id). When the account manager receives an invoice, she creates a record (supplier_id, order_id, amount, payment_address) and initiates a spend. Each server checks that the various ID fields are consistent between all participants and that the same (supplier_id, order_id) pair — which acts as a unique ID for the transaction — has not been previously used. If the checks pass the servers proceed to the threshold signature protocol.

The compliance server can be used to enforce additional, custom rules that aren't dictated by the basic logic of purchasing. Examples include: (i) no orders above \$1,000 (ii) orders above \$1,000 will trigger email warnings to an accounts supervisor (iii) orders above \$1,000 require additional approval by an accounts supervisor (iv) no more than \$10,000 of orders per supplier per month (v) orders must be within 20% of the cheapest price of a similar item with another supplier.

How do we handle the fact that there may be many purchasing and receiving clerks, and perhaps many account managers? We might assign a separate share to each possible employee in each role, but this is a bad design because adding an employee to a role would require changes to other employees' shares. We could provide each employee in each role (duty) with a copy of that role's share, but this makes revocation of role assignment a nightmare.

Instead, an elegant solution is to combine threshold wallets and regular access control. Specifically, each role server authenticates the employee and participates in the signature protocol on her behalf. The server does not allow employees to access their shares directly. Within each role, there is an arbitrary level of flexibility as long as it can be enforced by software. For example, the purchase server or receiving server might require two different clerks to sign off. The account manager might have the ability to configure the accounts server to automatically approve up to \$50/day per purchasing clerk.

As in any internal control system, technological measures are not enough; we need best practices to minimize human error. For example, the accounts manager should use the Bitcoin address in the invoice rather than copying it from the purchasing clerk's record. Thus, even though each server's state is not cryptographically hidden from other servers, servers should not present them to employees. Instead, the accounting server and the purchasing server should each require that the Bitcoin address is input directly to them. This encourage independent updates to state filess and minimize errors due to laziness.

5.4 Secure bookkeeping

Bookkeeping and auditing are crucial components of internal control. To enable auditing, we must ensure that all transactions are logged accurately.

Each participant could log its own records, of course, but these logs can be tampered with or destroyed. A better solution is to have a dedicated logging server. This allows enforcing extra security properties since the server's functionality is minimal: it can be physically secured and configured to output logs to redundant write-only tamper-proof media.

Definition 6 (Secure bookkeeping) *A bookkeeper is an entity that logs transactions along with all transaction metadata. A bookkeeping system is* ***secure*** *if a valid transaction cannot be generated without creating a correct log entry of it.*

DNF wallets lend themselves well to the addition of a bookkeeping entity. Given a policy $c_1 \vee \ldots \vee c_N$ where each c_i represents a conjunction over a subset of participants, we simply convert it to $(c_1 \wedge b) \vee \ldots \vee (c_N \wedge b)$ where b represents the presence of the bookkeeper. In other words, we add the bookkeeper to every possible authorization path.

Thus, in the purchasing example, we simply use 5-out-of-5 instead of 4-out-of-4 secret sharing and give an extra share to a logging server. The signature cannot be generated without the logging server's participation. This server does not enforce any policies and always completes the protocol, ensuring that it keeps a complete log of all messages signed by the respective participants. An auditor can later verify, for example, that receiving clerk Alice logged a certain (supplier_id, order_id, item_id) tuple. Here we're assuming that the 'receiving' role server adds the identity of the employee to each record created by that employee.

6 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.

6.1 Identity

For $t > 2$ the threshold signature protocol requires that all protocol messages are signed, since it involves participants forwarding other participants' messages. The ideal solution is X.509 certificates [33] because many companies already issue certificates to their employees. This also allows layering the protocol over TLS which is a convenient way to prevent spam or denial-of-service attacks.

6.2 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 participants are online when the signature is being generated. While non-interactive threshold signature schemes do exist for other signing algorithms [31], there is no known non-interactive ECDSA threshold signature algorithm.

Non-interactivity is not a limitation in the corporate context, as all the application scenarios we've described use a synchronous design anyway. In the case of a two-factor wallet for individuals, we can see how a non-interactive signature scheme might be better: the user can generate a share of a signature on her phone, whereupon the phone app would email it to herself. On her desktop, she would paste the share from her email into her wallet app, which would complete the signature. This would avoid the need for a custom protocol for the devices to securely pair and communicate with each other.

We do not believe that interactivity is a significant limitation in practice — as we detail in Section 7, our implementation of a protocol for two nearby devices to discover each other over a shared WiFi connection was very straightforward.

6.3 Size of shares

Unlike functional separation policies, there is the worry that joint control policies might require a large number of shares using DNF wallets due to the combinatorial construction. However, a look practical setups shows that this worry is unfounded. For example, for joint control over transaction approval in banks, the value of the threshold t is overwhelmingly just 2 [10, 29] . When $t = 2$ the number of shares per participant is only linear in n, resulting in trivially small sets of shares even for say $n = 100$.

At any rate, if there are 100 managers sharing custody of an account, the chance that some two of them might collude starts to become high, or at least difficult to estimate and manage. For this reason, security policies in practice are always kept tractable. Another way of looking at it is that the cognitive complexity of reasoning about security becomes a problem long before computational concerns do.

6.4 The curse of homogeneity

Our main goal in designing internal control systems has been to eliminate single points of failure. For this reason we believe our protocols in Section 5 are more secure than alternatives that don't use internal controls or implement them via access control.

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 that implement separation of duties. Further, malware may simultaneously compromise many or all machines due to software homogeneity.

Thus internal technical controls are more effective when the software platform is internally heterogeneous and system administrators don't directly control employee machines. Another possibility is to combine two-factor security (Section 7) with parallel control. Inevitably, these defenses all introduce a security-usability trade-off.

6.5 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 4.4 obviates the need to re-deal shares unless the security policy changes, and the techniques of Section 6.6 minimize the need to re-share even when the list does change.

Alternately, we can utilize protocols to deal shares without a dedicated dealer using secure multi-party computation [7, 21, 22].

6.6 Changing the security policy or access-control list

DNF wallets implementing functional separation (Section 5.3) support changes to the security policy relatively easily. Adding a new authorization path is a matter of generating a new set of shares and does not require changing existing shares. To enable removing authorization path, we can proceed as follows: we add a revocation server that is part of every authorization path, and to remove a path, we simply remove the share corresponding to that path from the revocation server.

In a joint control threshold wallet scenario where shares are given to employees who may join or leave the group, one technique is to generate some extra shares during the dealing step and store them on physically secured media (analogous to cold storage). If there are initially n participants 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. Handling employees leaving can be done as before, using a revocation server that's a necessary for all signatures. Re-sharing the key is arguably a more secure option. We note that when $t=2$, even with a revocation server, each participant's total share size linear in $n+m$. The revocation server's share size is quadratic. See the discussion in Section 5.2 for why these parameters are typically very small in practice.

6.7 Parallel control backup and recovery

Losing a share (due to malware, data loss, accidental deletion, physical loss of a device, etc.) has a similar effect as changes to the access-control list — specifically, an addition and a deletion. The most secure option is to re-share the key every time a loss or theft of any share is detected or suspected. Alternately, we can use a combination of extra shares and a revocation server as discussed above.

Unlike the corporate scenario, for online Bitcoin service providers (merchants, exchanges, wallets, etc.) there is another failure mode: a server going down. Service providers would like to maintain as high an uptime as possible, and the worry is that a threshold wallet multiplies the downtime by a factor of t.[11] Ideally, the threshold wallet must be able to recover from failure of a server with no downtime, i.e., without requiring any manual intervention. We can accomplish this by having one or two backup machines with additional shares. Instead of a t-out-of-t sharing, we would use t-out-of-$t+m$ sharing where m is the number of backup servers.

6.8 Two-factor security backup and recovery

In the two-factor security system described in Section 7, 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*. The ideal solution is to store backups of each share independently to minimize the chances of both being stolen.

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. As our wallets are deterministic, the only keyshares that need to be backed up are shares of the

[11] In our conversations with a prominent service provider who approached us about adopting our technology, it turned out that this was their primary concern.

master key. Once the master key has been reshared, participants can derive their shares for all other keys in the wallet.

7 Implementation and Evaluation

In this section, we describe the design, creation, and evaluation of a two-factor-secure distributed wallet that we described briefly in Section 7.

We can extend the principles of dual control to the security of an individual's wallet — here we split control between different the user's personal devices. 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 6.8). 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.

7.1 Design decisions

Our goal was to create pair of applications, one for a desktop computer and one for a smartphone, which would together form a easy to use wallet. We chose Java because of its cross-platform nature and the availability of many useful libraries. We wanted our code to be easily incorporated into other wallets, so we decided to make our code part of BitcoinJ, the most commonly used Java Bitcoin Library. As a result, we used BouncyCastle, the crypto library used by BitcoinJ, to implement our crypto code.

On the desktop, we created a modified version of the MultiBit wallet software since it is Java based and open source. On the phone (Android) we wrote a simple application from scratch since we required very little user interface.

The options for communication between the two devices are Bluetooth, WiFi, or a centralized server. We ruled out the latter since direct communication is faster, simpler, and more privacy-preserving. Our experience with Bluetooth on Android taught us that it it fairly unreliable, so we settled on WiFi communication. For device discovery (to initiate the communication) we used DNS Service Discovery, (DNS-SD), a system that uses DNS messaging to advertise services on a network. Once the phone and desktop had discovered each other, they used TLS in order to establish secure communication.

To initiate a secure connection we need an out-of-band exchange of key material (since there is no PKI); the method with the best usability-security trade-off seems to be using the phone's camera to capture a 2D-barcode on the desktop. We used the ZXIng barcode library. It can both create and read bar codes so we were able to use it on both devices.

7.2 Security Model

The desktop acts as a trusted dealer when distributing the phone's keyshare. Although there is some risk in using a trusted dealer, it can be alleviated by booting off a live disk image. Since the initialization phase requires no internet connection, this eliminates the danger of malware as long as the disk is trusted. The desktop transfers the keyshare and public key to the phone which completes the initialization. The desktop then deletes all record of the phone's keyshare.

After this point all future communication occurs over TLS (with self-signed client and server certificates) ensuring a completely secure and trusted connection. Although only authenticated messages are required by the threshold signature protocol (since it leaks no confidential information), using TLS prevents any denial of service attacks against the desktop.

7.3 Two-factor application protocol

- Initialization
  - Desktop: Create wallet and display QR code with Public Certificate and one-time-password
  - Phone: Scan QR code and initiate TLS connection using Public Certificate.
  - Phone: Authenticate using one-time-password
  - Phone: Send over public certificate and receive keyshare
- Transaction
  - Desktop: Create transaction
  - Desktop: Create TLS server socket and wait for phone to connect
  - Phone: Connect to desktop using TLS with client side authentication

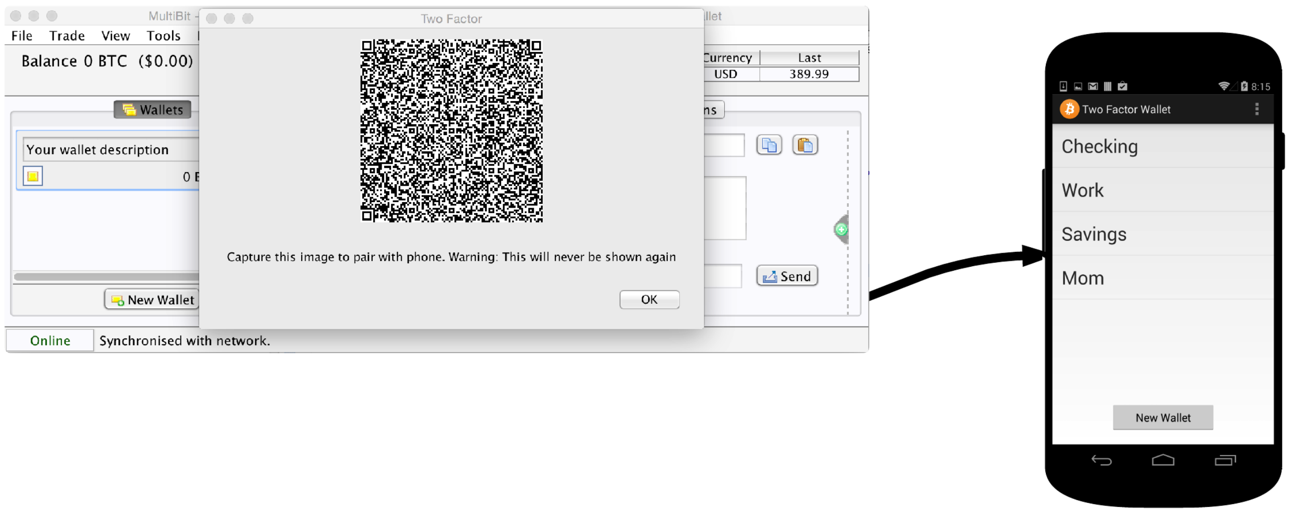


Figure 3: Initialization Protocol

- Phone: Give user choice to approve transaction. Continue if the user approves
- Desktop: Initiate threshold protocol
- Phone: Participate in threshold protocol
- Desktop: Complete transaction with produced signature and add to blockchain

7.4 Usage

When a new wallet is created in MultiBit, a QR code is displayed. The Android application scans the QR code which contains a self-signed certificate for the desktop and a one-time-password. The phone then initiates a TLS connection with the desktop using the certificate. The phone authenticates itself using the one-time-password and then sends its own self-signed certificate so that TLS client authentication can be enabled on future connections. The desktop then sends the phone's keyshare and deletes it from memory.

When MultiBit tries to sign a transaction, a server is started and a DNS-SD service is registered to advertise the server. While the phone application runs, it looks for this service and tries to initiate a TLS connection with the server. If it succeeds, the desktop sends the transaction information along with the wallet public key to the phone. If the phone has a keyshare for the public key, it presents the user with the transaction information along with the ability to allow or cancel the transaction. If the user chooses to allow it, the threshold scheme is run to produce a signature on the desktop. Finally the desktop broadcasts the signed transaction to the Bitcoin peer-to-peer network. The precise protocol is presented in Appendix 7.3.

	Time (Seconds)
Round 1 (Computer)	0.26
Round 2 (Phone)	0.36
Round 3 (Computer)	0.58
Round 4 (Phone)	11.04
Total	13.26

Table 2: This table demonstrates the per round running time of the threshold wallet app as recorded directly on the devices. The majority of time is spent in round 4. During this round 89% is spent creating and verifying zero knowledge proofs. The discrepancy between total time and the sum of the rounds is due to the computer verifying the phones final zero knowledge proof.

We transferred a small amount of bitcoin to our specially created wallet and then spent it by threshold-signing a transaction. Our threshold-signed transaction can be viewed in the block chain.[12]

7.5 Implementation performance

Our implementation exhibits reasonable performance. As seen in table 2, the vast majority of time spent while signing is during round 4 creating and verifying zero knowledge proofs. All other parts of the signing take

[12]Full details of this transaction can be viewed online at `https://blockchain.info/tx/cf5344b625fe87efa351aadf0bd542ec437c327b7c29e52245d3b41cea3e205b`

negligible amounts of time. Thus the total execution time is is small compared to the time required for a Bitcoin transaction to receive a confirmation on the block chain, which is 10 minutes on average. Thus our system is efficient enough to work well in practice. It is important to note that the signature generated by the protocol is a regular Bitcoin signature and thus can be verified in the same amount of time.

7.6 t-out-of-t performance

We evaluated the performance of the t-out-of-t threshold signature scheme. We implemented all participants on a single desktop but measured the running time for each participant; we report the per-participant execution time.

The encryption scheme uses a large modulus N whose size grows linearly with t, the number of participants. The bottleneck step is contained in the zero knowledge proofs which repeatedly invoke exponentiation with a modulus of N^2 whose size is again linear in t. The number of ZK proofs is $O(t)$.

Note that the running time is not dependent on n: even though our t-out-of-n signature scheme involves a combinatorial construction to distribute shares, the signature protocol itself is just a t-out-of-t protocol.

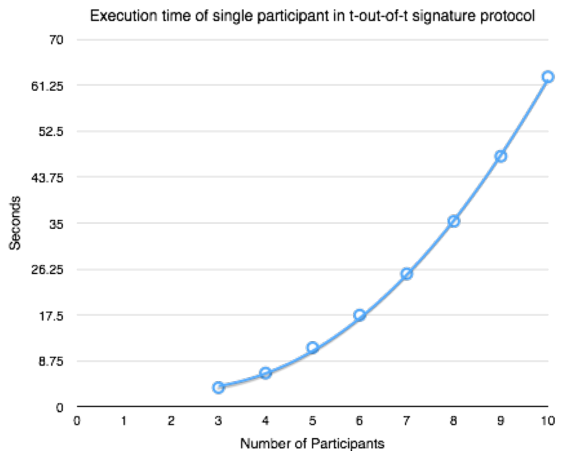


Figure 5: Running time versus number of participants

8 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 first practical DSA/ECDSA threshold-signature scheme, which we believe to be of independent interest. We showed how to use this to realize threshold wallets and various internal control protocols. We are currently in discussions with a prominent Bitcoin wallet software to integrate our implementation of two-factor security. Our techniques have the potential to dramatically improve Bitcoin security, moving it closer to widespread adoption as a currency.

References

[1] Bitcoin Forum member dree12, “List of Bitcoin Heists,” https://bitcointalk.org/index.php?topic=83794.0, 2013.

[2] Kaspersky Labs, “Financial cyber threats in 2013. Part 2: malware,” http://securelist.com/analysis/kaspersky-security-bulletin/59414/financial-cyber-threats-in-2013-part-2-malware/, 2013.

[3] D. Johnson, A. Menezes, and S. Vanstone, “The elliptic curve digital signature algorithm (ecdsa),” *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63, 2001.

[4] D. W. Kravitz, “Digital signature algorithm,” Jul. 27 1993, uS Patent 5,231,668.

[5] D. R. Stinson and R. Strobl, “Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates,” in *Information Security and Privacy*. Springer, 2001, pp. 417–434.

[6] P. MacKenzie and M. K. Reiter, “Two-party generation of dsa signatures,” in *Advances in Cryptology—CRYPTO 2001*. Springer, 2001, pp. 137–154.

[7] ——, “Two-party generation of dsa signatures,” *International Journal of Information Security*, vol. 2, no. 3-4, pp. 218–239, 2004.

[8] H. F. Tipton and M. Krause, *Information security management handbook*. CRC Press, 2004.

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

[10] R. Anderson, *Security engineering*. Wiley. com, 2008.

[11] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Consulted*, vol. 1, p. 2012, 2008.

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

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

[14] "Bitcoin wiki: Elliptic Curve Digital Signature Algorithm," https://en.bitcoin.it/w/index.php?title=Secp256k1&oldid=51490, accessed: 2014-02-11.

[15] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[16] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary version*, 1998.

[17] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public key cryptography—PKC 2003*. Springer, 2002, pp. 31–46.

[18] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold dss signatures," in *Advances in Cryptology—EUROCRYPT'96*. Springer, 1996, pp. 354–371.

[19] ——, "Secure distributed key generation for discrete-log based cryptosystems," in *Advances in Cryptology—EUROCRYPT'99*. Springer, 1999, pp. 295–310.

[20] M. H. Ibrahim, I. Ali, I. Ibrahim, and A. El-sawi, "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. IEEE, 2003, pp. 276–280.

[21] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," in *Public Key Cryptography*. Springer, 2001, pp. 119–136.

[22] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, and G. Poupard, "Practical multi-candidate election system," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*. ACM, 2001, pp. 274–283.

[23] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in *Information Security Applications*. Springer, 2011, pp. 44–60.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptology—EUROCRYPT'99*. Springer, 1999, pp. 223–238.

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

[26] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 127–140.

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

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

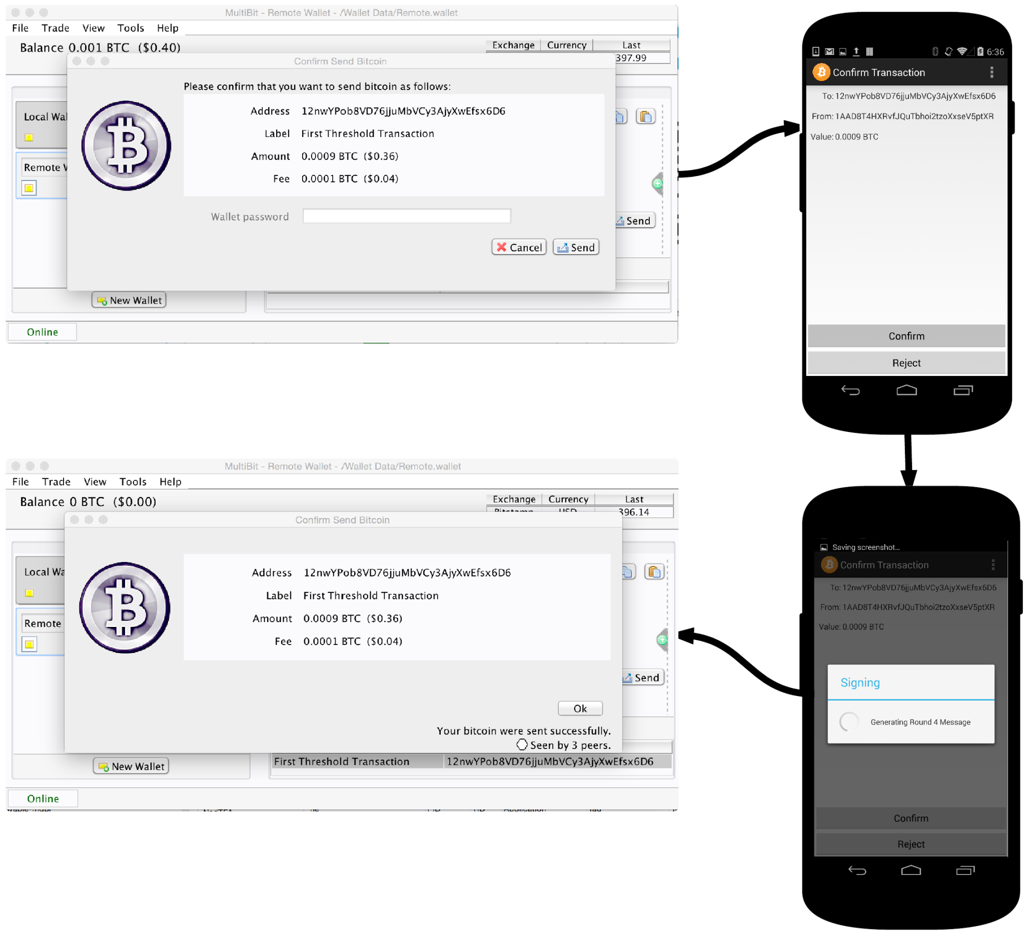
[29] C. of the Currency Administrator of National Banks, "Comptroller's Handbook: Internal Control," http://www.occ.gov/publications/publications-by-type/comptrollers-handbook/intcntrl.pdf.

[30] "BIP 70: Payment Protocol," https://github.com/bitcoin/bips/blob/master/bip-0070.mediawiki.

[31] V. Shoup, "Practical threshold signatures," in *Advances in Cryptology—EUROCRYPT 2000*. Springer, 2000, pp. 207–220.

[32] I. Damgård and M. Koprowski, *Practical threshold RSA signatures without a trusted dealer*. Springer, 2001.

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



MultiBit - Remote Wallet - /Wallet Data/Remote.wallet

Balance 0.001 BTC ($0.40)

Confirm Send Bitcoin

Please confirm that you want to send bitcoin as follows:

Address	12nwYPob8VD76jjuMbVCy3AjyXwEfsx6D6
Label	First Threshold Transaction
Amount	0.0009 BTC ($0.36)
Fee	0.0001 BTC ($0.04)

Wallet password

Confirm Transaction

To: 12nwYPob8VD76jjuMbVCy3AjyXwEfsx6D6

From: 1AAD8T4HXRvfJQuTbhoi2tzoXxseV5ptXR

Value: 0.0009 BTC

MultiBit - Remote Wallet - /Wallet Data/Remote.wallet

Balance 0 BTC ($0.00)

Confirm Send Bitcoin

Address	12nwYPob8VD76jjuMbVCy3AjyXwEfsx6D6
Label	First Threshold Transaction
Amount	0.0009 BTC ($0.36)
Fee	0.0001 BTC ($0.04)

Your bitcoin were sent successfully.

Seen by 3 peers.

First Threshold Transaction 12nwYPob8VD76jjuMbVCy3AjyXwEfsx6D6

Saving screenshot...

Confirm Transaction

To: 12nwYPob8VD76jjuMbVCy3AjyXwEfsx6D6

From: 1AAD8T4HXRvfJQuTbhoi2tzoXxseV5ptXR

Value: 0.0009 BTC

Signing

Generating Round 4 Message

Figure 4: Transaction Protocol

A 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, or it might contain a spending limit. 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 7 (Secure delegation) *A wallet delegation protocol 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.[13] 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.

B Online wallets: secure withdrawal

In our corporate purchasing example, security derives from the fact that every piece of information necessary for completion of the transaction is entered into the system by at least two participants. In particular, the supplier communicates his receiving address to the purchasing clerk, and then again to the accounts department as part of the invoice.

Consider, on the other hand, an online wallet service that holds funds on users' behalf. Users may withdraw bitcoins (to their own wallets, or another service) at any time simply by using the web interface. Avoiding a single point of failure in such a system appears hopeless: anyone who hacks into or controls the web server might be able to withdraw all users' funds.

One solution is a 2-out-of-2 wallet where one share is held by the user and the other by the online wallet provider. However, this results in a severe usability and security cost for the user — he may not want to run a Bitcoin node, and losing his device might make his funds unrecoverable. Perhaps more importantly, it locks up users' deposits and prevents the service provider from maintaining fractional reserves, as virtually all banks do.

Surprisingly, there is a solution to this conundrum using threshold wallets. It requires introducing a delay (say, 24 hours) between a user requesting a withdrawal and actually executing the transaction. This delay is used to notify the user by email and allow him to cancel the transaction if it is unauthorized. All security-critical steps are executed in a threshold fashion.

- There are t share servers in addition to a web server. Each share server maintains the user's current Bitcoin balance and a list of user-uploaded withdrawal addresses a table (user_id, user_email_addr, balance, withdrawal_addresses).
- The user may request a withdrawal at any time, and the web server sends this request to each share server. If the user has sufficient balance and the requested withdrawal address is on the approved list, the share servers execute a threshold signature protocol immediately to transfer the funds.
- If the address is not on the approved list, the share servers add it to a pending address list and each share server emails the user that a new pending address has been added.
- If the user takes no action, the pending address gets added to the approved list after the delay period and the transaction is then executed.

Thus, even if the web server and all but one share server are compromised, the user has a delay period to take action. Since the web server may be compromised, the recovery procedure should not be web-based; it could instead involve calling the company's customer support

[13] For simplicity of exposition we can assume that there is a single address that Alice is delegating, but as described in Section 4.4, to delegate a wallet Alice would share her master private key.

line. As an incidental benefit, the scheme also provides protection against compromise of the user's account.

When the user wishes to initiate a payment to a merchant via the wallet service (rather than withdrawing the bulk of his funds), a 24-hour delay is not acceptable. So there would have to be a daily spending limit for addresses not on the approved list; below this limit, there is no delay period.

From a usability perspective, having each share server independently email the user is undesirable. It is not clear if it is possible to avoid this without compromising security.