A distribution protocol for dealerless secret distribution

Author: Craig Wright

Abstract

As the value of bitcoin increases, more incidents such as those involving Mt Gox and Bitfinex will occur in standard centralised systems. The addition of group based threshold cryptography with the ability to be deployed without a dealer and which supports the non-interactive signing of messages provides for the division of private keys into shares that can be distributed to individuals and groups to provide additional security. This scheme creates a distributed key generation system for bitcoin that removes the necessity for any centralised control list minimising any threat of fraud or attack. In the application of threshold based solutions for DSA to ECDSA, we have created an entirely distributive signature system for bitcoin that mitigates against any single point of failure. When coupled with retrieval schemes involving CLTV and multisig wallets, our solution provides an infinitely extensible and secure means of deploying bitcoin. Using Group and ring based systems we can implement blind signatures against issued transactions.

Keywords: Bitcoin, ECDSA, Elliptic Curve Cryptography, Threshold Cryptography

I. Introduction

In this paper, we present a threshold based dealerless private key distribution system that is fully compatible with bitcoin. The system builds on a group signature scheme that departs from the traditional individual signing systems deployed within bitcoin wallets. As deployed, the system is both extensible and robust tolerating errors and malicious adversaries. The system is supportive of both dealer and dealerless systems and deployment in an infinitely flexible combination of distributions.

Individual parties can act as single participants or in combination as a dealer distributing slices of their protected key slice across machines for security and recoverability or in groups for the vote threshold based deployment of roles and access control lists.

There is no limit to the depth of how far a slice can be divided. This issue of complexity needs to be weighed against the distinct deployment. In this manner, as the record of signing and transactions can be hidden using this methodology from outside participants all with extensions to be presented in a subsequent paper even from those within the groups, we have introduced a level of anonymity and plausible deniability into bitcoin transactions increasing the standard of pseudonymous protection to the users.

Ibrahim et al. [2003] developed an initial robust threshold ECDSA scheme. The following protocol is a further extension of what forms the elliptic curve form of the threshold DSS introduced by Gennaro et al. [1996].

Table 1 Definitions

m	The message incl. Bitcoin transaction
$\mathrm{e} = H(m)$	The hash of the message
CURVE	The elliptic curve and field deployed (summarised as E)
G	The elliptic curve base point. This point is a generator of the elliptic curve with large prime order n
n	Integer order of G such that n x $G = \varnothing$ This is defined as the number of rational points that satisfy the elliptic function and represents the order of the curve E.
k	The threshold value for the key splitting algorithm. This value represents the number of keys needed to recover the key. The secret is safe for $(k-1)$ shares or less and hence can be retrieved with k shares.
d_A	A private key integer randomly selected in the interval $[1,(n-1)]$
Q_A	The public key derived from the curve point $Q_A = d_A$ x G [1]
×	Represents elliptic curve point multiplication by a scalar
j	The number of participants in the scheme.

The use of group mathematics allows is to create a verifiable secret sharing scheme (VSS) that extends the work of Shamir [1979] in secret hiding and that of Feldman [1987] and Pedersen [1992] from RSA and DSA schemes so that we can use it within ECC and ECDSA-based Signature systems such as Bitcoin [Koblitz, 1998]. Our system is tolerant against malicious adversaries, halting and is robust against eavesdropping.

In this paper, we start by presenting a method to allow for the cooperative signing of ECDSA signatures where no one-party ever knows the private key. Moreover, we allow for the private key pairs to be updated and refreshed without the necessity of changing the private key. In a subsequent paper following this one, we will detail the mathematics needed to add and remove members to a group while maintaining the secrecy and privacy of the private key.

This process will also be extended in sister papers to demonstrate how cooperative computations can be performed while the output is dependent on the secure input of separate entities in a manner that

[1] Bitcoin uses secp256k1. This defines the parameters of the ECDSA curve used in Bitcoin, and may be referenced from the Standards for Efficient Cryptography (SEC) (Certicom Research, http://www.secg.org/sec2-v2.pdf).

does not require trust. Existing solutions all require a trusted party. Utilising the solution presented in this paper will also allow us to extend the work of Chaum [1983] from a centralised system into a truly distributed manner of issuing electronic notes that can be settled directly on the Bitcoin Blockchain making the requirements for alternate Blockchains or sidechains obsolete.

Matters of trust

All existing systems require some level of trust. Until this time, bitcoin has needed the protection of a private key using a secure system that is isolated from the world in which has proven difficult to achieve. Of note, systems where Bitcoin can be exchanged or stored, require trust in a centralised authority. Our work changes this requirement entirely distributing and decentralising the key creation and message signing processes within bitcoin while not changing any of the core requirements of the protocol. The methodologies noted in this paper may be implemented without modifying the bitcoin protocol and in fact, there is no way to determine whether this process has been deployed through the analysing of a signed message.

In creating a distributed signature scheme for Bitcoin, we allow for a group of people or systems to securely hold a key in a way that leaves no individual capable of generating a signature on their own. When extended, this scheme also allows for the secure recovery of each of the shares as well as the bitcoin private key itself. The group generated signature is indistinguishable from that generated from the existing protocol. As such signature verification remains as if it was enacted through a single person signer using a standard transaction.

This increase in trust is achieved as the secret key is shared by a group of n participants or m groups of participants. A threshold number of participants is required for the signing of a transaction, and any coalition of participants or groups of participants that meet the minimum threshold can perform the signature operation. Importantly, this protocol can be enacted synchronously or as a batched process where individuals or groups can attempt to create a coalition of participants.

II. Prior work

Shamir [1979] first introduced a dealer based secret sharing scheme that allowed for a distributed management of keys. The problems associated with this scheme come from the necessity of trusting a dealer who cannot be verified. This form of the scheme is fully compatible with the current system presented in this paper and can be used for group distribution of individual key slices that are created through the process noted herein.

Joint Random Secret Sharing (JRSS) [Pedersen, 1992]
The stated aim of this procedure is to create a method where a group of participants may collectively share a secret without any participant having knowledge of the secret. Each participant selects a random value as their local secret and distributes a value derived from this using S*SSS* with the group. Each participant then adds all the shares received from the participants, including its own. This sum is the joint random secret share. The randomness offered by a single honest participant is sufficient to maintain the confidentiality of the combined secret value. This state remains true even if all $(n-1)$ other participants intentionally select non-random secret values).

Joint Zero Secret Sharing (JZSS) [Ben-Or, 1988]
JZSS is like *JRSS*, with the difference that each participant shares 0 as an alternative to the random value. The shares produced using this technique aid in removing any potential weak points in the JRSS algorithm.

Desmedt [1987] introduce the concept of group orientated cryptography. This process allowed a participant to send a message to a group of people in a manner that only allowed a selected subset of

the participants to decrypt the message. In the system, the members were said to be known if the sender must know them using a public key and the group is anonymous if there is a single public key for the group that is held independently of the members. Our system integrates both methodologies and allows for known and anonymous senders and signers to exist within a group simultaneously.

III. Methods

For any elliptic curve (CURVE) with a large order prime, and a base point $G \in CURVE\left(\mathbb{Z}_p\right)$ of order n defined over the prime field $\mathbb{Z}_p$; we can create a system that allows for the secure distribution of an ECC private key into key shares and its use without any participant being able to recreate the original private key from less than a threshold of shares.

For an unknown integer d_A where $1 \le d_A \le \left(n-1\right)$ we know that it is extremely difficult to calculate d_A given $Q_A = d_A \times G$ [Kapoor, 2008].

Our fundamental technique is derived using the application of threshold cryptography. In this system, the ECDSA private key only exists as a potential and need never be recreated on any system. Each of these multiple shares are distributed to multiple participants [$p_{(i)}$] in a manner that is extensible and allows for the introduction of both group and individual party signature formats. Thus, the signing process differs from that deployed within bitcoin. In this process, a coordinating participant $p_{(c)}$ creates a transaction and a message signature that is distributed to the group. Each participant can vote on the use of its private key share by either computing a partial signature or passing.

In effect, passing would be equivalent to a no vote. The coordinating participant $p_{(c)}$ will collate the responses and combine these to form a full signature if they have received the minimum threshold number of partial signatures.

The coordinating participant $p_{(c)}$ can either accept the no vote and do the calculation based on a null value from the other party or can seek to lobby the party and convince them to sign the message. The protocol can be implemented with a set coordinator, or any individual or group can form this role and propose a transaction to the threshold group to be signed. Our system extends the work of Ibrahim et. al. [2003] providing a completely distributed ECDSA private key generation algorithm. This paper also presents a distributed key re-sharing algorithm and a distributed ECDSA signing algorithm for use with bitcoin. The key re-sharing algorithm may be used to invalidate all private key shares that currently exist in favour of new ones or for the reallocation of private key shares to new participants. This protocol extends to the sharing of not only the ECDSA private key but to private key shares as well. The consequences of this mean that shares can be constructed and voted on as a group process.

This system removes all requirements for a trusted third party to exist. Consequently, it is possible to create the new overlay and wallet for Bitcoin that is entirely compatible with the existing protocol and yet removes any remaining single points of failure while also allowing greater extensibility. This system can also be extended to allow for the introduction of blind signatures.

As our system does not require a private key ever to be loaded into memory, we not only remove the need for a trusted third party but further remove a broad range of common attacks. The protocol is extensible allowing the required number of shares and the distribution of shares to be decided by the use case, economic scenario and risk requirements.

The system we have implemented mitigates all side channel attacks and thus any cache timing attacks. This system takes the work of Gennaro et. al. [1996] and extends it from DSS such that it can be successfully used in any ECDSA based application.

ECDSA

Bitcoin uses ECDSA based on the secp256k1 curve. ECDSA was first standardised by NIST in 2003 [NIST] varying the requirements for Diffie-Hellman-based key exchanges using elliptic curve cryptography (ECC). The creation of ECC was particularly important due to the reduction in key size and processing power when compared to other public/private key systems. No sub-exponential time algorithm has been discovered for ECDLP. ECDLP is known to be intractable and refers to the elliptic curve discrete logarithm problem [Johnson, 2001].

The parameters used throughout this paper are documented in Table 1.

Security considerations

The system is bounded by the security of ECDSA which is a current limitation within bitcoin. At present, ECDSA remains secure if a private key can be securely deployed. Our system mitigates side channel attacks and memory disclosure attacks up to the threshold value requiring that the threshold number of participants have been compromised before the rekeying event. Additionally, any uncompromised threshold majority will be able to identify compromised participants at less than the threshold value.

Halting problems

Service disruption is a form of attack that can be engaged in by a malicious adversary attempting to create a denial of service attack against the participants. This attack would require the participants to either receive invalid signatures that they would expend processing time analysing or through flooding network messages that would be subsequently dropped.

The requirement to encrypt messages to the participants using either ECC or signcryption based ECC mitigates this attack vector. Before an attacker can send invalid partially signed messages, they would need to have already compromised a participant, making this form of attack no longer necessary.

Randomness

Algorithm 2 provides a scenario where sufficient randomness is introduced even if (n-1) participants fail to choose random values. A possible addition to this protocol is the introduction of group oracles designed solely for the introduction of random values to the signing and rekeying process. In this optional scenario, each of the key slices can be generated using the same protocol. For instance, if we have an m of n primary slice requirement, each of the underlying key slices can also be generated and managed using an m' of n' threshold condition.

A participant using this system would be able to have the addition of an external Oracle that does nothing other than injecting randomness into the protocol. A user with m' key slices (where m'<n-1) could choose to recreate and process their signature solution based on the key slices they hold or may introduce an external Oracle that is unnecessary other than for the introduction of randomness.

Each slice could be likewise split for robustness and security. The key slice could be distributed such that the user has a slice on an external device such as a mobile phone or smartcard and a software program running on a computer such that the combination of sources would be required for them to create a partial signature.

It is important that a unique random ephemeral key D_k is produced or it would be possible to use the information to recreate the private key d_A.

Public signing

The primary purpose of transactional signing using this protocol is to enable the distributed signing of the bitcoin transaction. Any transaction that has not been published to the Blockchain can be maintained privately by the participants. Therefore, if a coordinating participant $p_{(c)}$ on any occasion has not been able to achieve the required level of votes to sign a transaction successfully, it is not necessary to create a new bitcoin transaction. The ownership of any settled transaction remains secure if the key slices are themselves secure to the threshold value.

If the system is deployed well, the ability to compromise up to (k-1) participants leaves the system secure to attack below the threshold value. When coupled with a periodic rekeying protocol (*Algorithm 2*), our system can withstand side channel attacks and memory disclosures.

Method and implementation

As the protocol encrypts the secret information required to be sent between participants using ECC based on a hierarchical derivation [Wright, 2016] it is both possible and advisable to collate all messages into a single packet sent to all users such that validation can be done against potentially compromised or hostile participants when necessary.

Signature generation is proposed by a coordinating participant $p_{(c)}$. By default, any key slice can act as the coordinating participant and the requirements come down to the individual implementation of the protocol. The algorithms used are documented below, and a later section provides detail as to their deployment.

Algorithm 1 Key Generation

Domain Parameters (CURVE, Cardinality n, Generator G)

Input: NA

Output: Public Key Q_A

Private Key Shares $d_{A(1)}, d_{A(2),\ldots,} d_{A(j)}$

For our threshold of k slices from (j) participants, we have a constructed key segment $d_{A(i)}$ which is associated with participant (i) and $(j-1)$ participants nominated as participant (h) that are the other parties that participant (i) exchanges secrets with to sign a key (and hence a Bitcoin transaction).

- In the scheme, j is the total number of participants where $k \leq j$ and hence $h = j - 1$
- Hence, we have a (k, j) - threshold sharing scheme.

The method for *algorithm 1* follows:

1) Each participant $p_{(i)}$ of (j) where $1 \leq i \leq j$ exchanges an ECC public key (or in this implementation, a Bitcoin address) with all other participants. This address is the Group identity address and does not need to be used for any other purpose. This can be formed of a "Type 42" address of any form or level of the hierarchy (Patent 222). The exchange of symmetric keys for participants is completed using the method in patent 42.

Note this is a derived address [Wright, 2016] and key based on a shared value between each of the participants from the process of "Determining a common secret for two Blockchain nodes for the secure exchange of information".

2) Each participant $p_{(i)}$ selects a polynomial $f_i(x)$ of degree $(k-1)$ with random coefficients in a manner that is secret from all other parties.

This function is subject to the participant's secret $a_0^{(i)}$ that is selected as the polynomial free term. This value is not shared. This value is calculated using a derived private key [Wright, 2016].

We define $f_i(h)$ to be the result of the function, $f_{(x)}$ that was selected by participant $p_{(i)}$ for the value at point $(x=h)$, and the base equation for participant $p_{(i)}$ is defined as the function:

$$f_{(x)} = \Sigma_{\mathrm{p}=0}^{(k-1)} \; a_p x^{\mathrm{p}} \bmod n$$

In this equation, a_0 is the secret for each participant $p_{(i)}$ and is not shared.

Hence, each participant $p_{(i)}$ has a secretly kept function $f_i(x)$ that is expressed as the degree (k-1) polynomial with a free term $a_0^{(i)}$ being defined as that participant's secret such that:

$$f_{i(x)} = \Sigma_{\gamma=0}^{(k-1)} \; a_\gamma x^{\gamma} \bmod n$$

3) Each participant $p_{(i)}$ encrypts $f_i(h)$ to participant $P_{(h)}$ $\forall$ h= $\{1,\ldots,(\mathrm{i}-1),(\mathrm{i}+1),\ldots,j\}$ using $P_{(h)}$'s public key[2] [Wright, 2016] as noted above and exchanges the value for $P_{(h)}$ to decrypt.

Note that $n \times G = \varnothing$ for any basic point $G \in E(\mathrm{Z}_p)$ of order n for the prime p[3].

As such for any set of integers B: $\{b_i \in \mathbb{Z}_n\}$ that can be represented as $(b, b_1, b_{2,\ldots})$, if $bG = [b_1G + b_2G + \ldots] \bmod p$, then $b = [b_1 + b_2 + \ldots] \bmod n$. Further, if $bG = [b_1 b_2 \ldots] G \bmod p$ then $b = [b_1 b_2 \ldots] \bmod n$.

---

[2] A related research paper will detail how the process of "Determining a common secret for two Blockchain nodes for the secure exchange of information" and the sharing of keys may be integrated.

[3] In the case of bitcoin the values are:

Elliptic curve equation: $y^2 = x^3 + 7$

Prime modulo: 2256 – 232 – 29 – 28 – 27 – 26 – 24 - 1

= FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F

Base point =

04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

Order = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141

Given that $\mathbb{Z}_n$ is a field and we can validly do Lagrange interpolation modulo n over the values selected as ECC private keys, we have a condition which leads to the conclusion that Shamir's Secret Sharing Scheme SSSS [5] can be implemented over $\mathbb{Z}_n$.

4) Each participant $P_{(i)}$ broadcasts the values below to all participants.

a) $a_\kappa^{(i)}G \qquad \forall \kappa = \{0,\ldots,(k-1)\}$

b) $f_i(h)G \qquad \forall h = \{1,\ldots,j\}$

The value associated with the variable h in the equation above can either be the position of the participant $P_{(h)}$ such that if participant $P_{(h)}$ represents the third participant in a scheme, then h=3 or equally may represent the value of the ECC public key used by the participant as an integer. Use cases and scenarios exist for either implementation. In the latter implementation, the value $h = \{1,\ldots,j\}$ would be replaced by an array of values mapped to the individual participant's utilised public key.

5) Each participant $P_{(h\neq i)}$ verifies the consistency of the received shares with those received from each other participant.

That is: $\sum_{\kappa=0}^{(k-1)} \mathrm{h}^\kappa a_\kappa^{(i)} G \stackrel{?}{=} f_i(\mathrm{h})$

And that $f_i(h)G$ is consistent with the participant's share.

6) Each participant $P_{(h\neq i)}$ validates that the share owned by that participant ($P_{(h\neq i)}$) and which was received is consistent with the other received shares:

$a_0^{(i)}G = \Sigma_{h\in \mathrm{B}} b_h f_i(\mathrm{h})G \qquad \forall P_{(h\neq i)}$

If this is not consistent, the participant rejects the protocol and starts again.

7) Participant $p_{(i)}$ now either calculates their share $d_{A(i)}$ as:

SHARE($p_{(i)}$) $= d_{A(i)} = \Sigma_{\mathrm{h}=1}^{j} f_h(i) \bmod n$

Where: SHARE($p_{(i)}$) $\in \mathbb{Z}_n$ and $d_{A(j)}$

and

Where: $Q_A = Exp - Interpolate(f_1,\cdots,f_j) \quad \triangleright [= G \times d_A]$

Return $(d_{A(i)}, Q_A)$

Participant $p_{(i)}$ now uses the share in calculating signatures. This role can be conducted by any participant or by a party $p_{(c)}$ that acts as a coordinator in the process of collecting a signature. The

participant can vary and does not need to be the same party on each attempt to collect enough shares to sign a transaction.

Hence private key shares $d_{A(i)} \leftarrow \mathbb{Z}_n^*$ have been created without knowledge of the other participant's shares.

Algorithm 2 **Updating the private key**

Input: Participant P_i 's share of private key d_A denoted as $d_{A(i)}$.

Output: Participant P_i's new private key share $d_{A(i)}$.

Algorithm 2 can be used to both update the private key as well as to add randomness into the protocol. In a follow-up paper related to this one, we will demonstrate how this algorithm can be extended to allow group additive and subtractive processes. In this manner, it will be possible to create distributed key sets where the private key is never created or issued and yet transactional signatures can be completed. More importantly, this system will be extensible through the addition and removal of members within a hierarchy of groups.

Using US patent number 15087315 [Wright, 2016] format keys, this process can lead to the recalculation of hierarchical sub-keys without the reconstruction or even calculated existence of the private keys. In this manner, we can construct hierarchies of bitcoin addresses and private key slices that when correctly deployed will remove any large-scale fraud or database theft as has occurred in the past.

1) Each participant selects a random polynomial of degree $(k-1)$ subject to zero as it's free term.

   This is analogous to *Algorithm 1* but that the participants must validate that the selected secret of all other participants is zero.

Note that: $\varnothing G = nG = 0$ where 0 is a point at infinity on the elliptic curve.

Using this equality, all active participants validate the function:

$$a_0^{(i)} G = \varnothing \quad \forall i = \{1, \cdots, j\}$$

See Feldman (1987) for an analogy.

Generate the zero share: $z_i \leftarrow \mathbb{Z}_n^*$

2) $d_{A(i)}{}' = d_{A(i)} + z_i$

3) **Return:** $d_{A(i)}{}'$

The result of this algorithm is a new key share that is associated with the original private key. A variation of this algorithm makes the ability to both increase the randomness of the first algorithm or to engage in a re-sharing exercise that results in new key slices without the need to change the bitcoin address possible. In this way, our protocol allows a group to additively mask a private key share without altering the underlying private key. This process can be used to minimise any potential key leakage associated with the continued use and deployment of the individual key shares without changing the underlying bitcoin address and private key.

Algorithm 3 Signature Generation

Domain Parameters: CURVE, Cardinality n , Generator G

Input: Message to be signed $e = H(m)$

Private Key Share $d_{A(i)} G \mathbb{Z}_n^*$

Output: Signature $(r,s) \in \mathbb{Z}_n^*$ for $e = H(m)$

A) Distributed Key Generation

1) Generate the ephemeral key shares using *Algorithm 1*:

$$D_{k(i)} \leftarrow \mathbb{Z}_n^*$$

2) Generate Mask shares using *Algorithm 1*:

$$\alpha_i \leftarrow \mathbb{Z}_n$$

3) Generate Mask shares with *Algorithm 2*:

$$b_i, c_i \leftarrow \mathbb{Z}_n^2$$

B) Signature Generation

4) $e = H(m)$ Validate the hash of the message m

5) Broadcast

$$\vartheta_i = D_{k(i)} \alpha_i + \beta_i \bmod n$$

And

$$\omega_i = G \times \alpha_i$$

6) $\mu = Interpolate(\vartheta_i, \ldots, \vartheta_n) \bmod n$

$$\triangleright \left[= D_k \alpha \bmod n \right]$$

7) $\theta = Exp - Interpolate(\omega_1, \ldots, \omega_n)$

$$\triangleright \left[= G \times \alpha \right]$$

8) Calculate (R_x, R_y) where $r_{x,y} = (R_x, R_y) = \theta \times \mu^{-1}$

$$\triangleright \left[= G \times D_k^{-1}] \right]$$

9) $r = r_x = R_x \bmod n$

If $r = 0$, start again (i.e. from the initial distribution)

10) Broadcast $S_i = D_{k(i)} \left(\mathrm{e} + D_{A(i)} r \right) + C_i \bmod n$

11) $S = Interpolate(s_i, \ldots, s_n) \bmod n$

If $s = 0$ redo Algorithm 3 from the start (A.1).

12) Return (r, s)

13) In Bitcoin, reconstruct the transaction with the (r, s) pair to form a standard transaction.

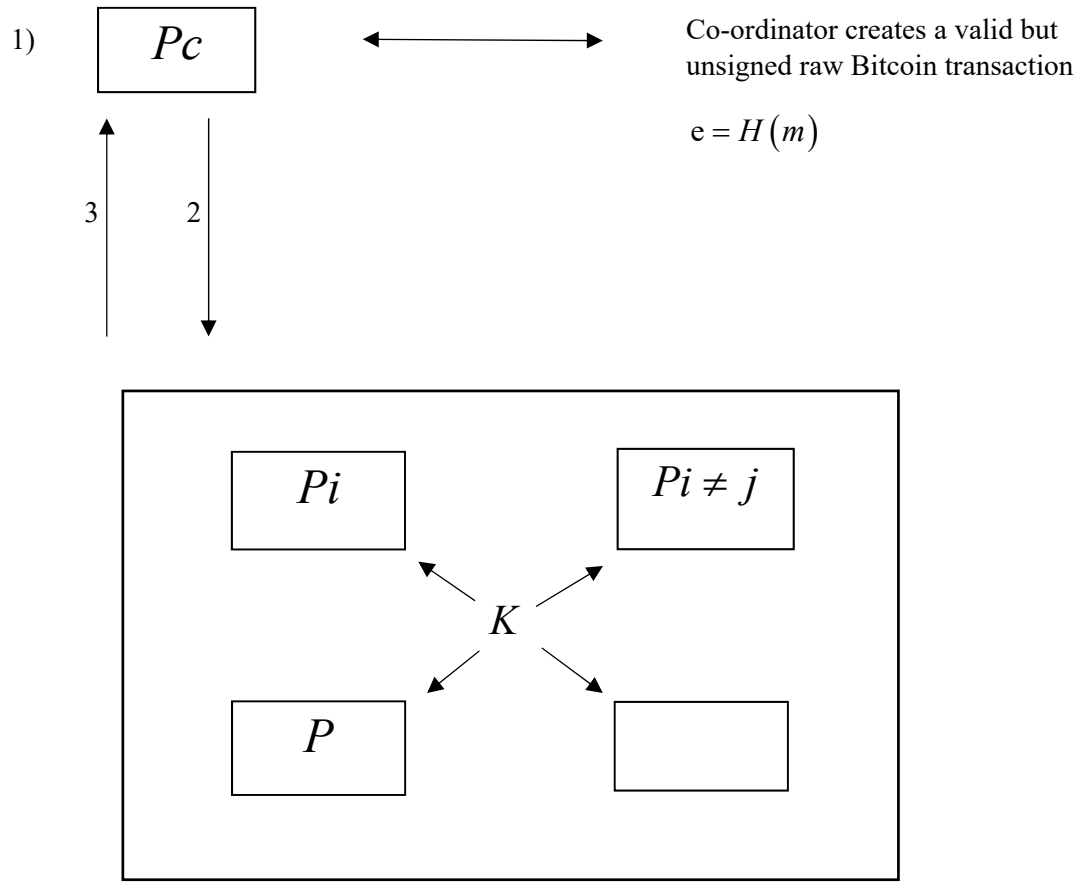
IV. The Model - Threshold ECDSA (T.ECDSA)

In our model, we allow for the system of n groups or individuals that we designate as participants. Each player can be an individual is a soul participant or a group or combination of the above. Participant $p_{(i)}$ may be mapped against an identity using a commonly derived public key calculation or participant $p_{(i)}$ may be left as a pseudonymous entity with the public key of the participant used only for this protocol without being mapped back to the individual.

This system introduces a dedicated broadcast channel allowing for the recognition of other participants as a valid player and a member of the scheme while simultaneously allowing members within the group to remain unidentified. When a message is broadcast from participant $p_{(i)}$, the members within the group will recognise the message as coming from an authorised party without necessarily being able to identify the end user or individual that is associated with the key. It is also possible to link the identity of the key to an individual is such a system were warranted.

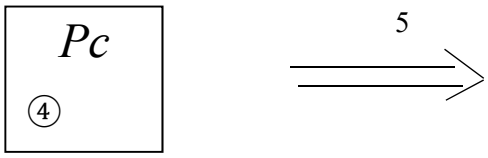
We see this process flow summarised below:

Step 1)



Step 2) *Pc* sends the raw transaction to the Group. If this validates (i.e. the raw transaction matches the Hash to be signed), the participant votes by signing it.

Step 3) If Yes, each Participant returns the partially signed transaction.



Step 4) *Pc* (or any other Participant) reconstructs the complete signature if a threshold of partially signed transactions are received.

Step 5) *Pc* broadcasts the transaction as a signed Bitcoin transaction.

The calculation of a message signature can be initiated by an individual who does not change, or through a temporary broadcasting party. The role of the protocol coordinator can be conducted by any participant or by a party $p_{(c)}$ that acts as a coordinator in the process of collecting a signature. The individual scenario can be constructed ahead of time in a manner that allows for any member of the system as participant $p_{(i)}$ to step forth as the coordinating party $p_{(c)}$ in proposing a scenario that leads to a transaction signature if the required votes are obtained or may be based on the necessity to get authorisation through a set coordinator that does not change throughout the process. A subsequent paper will document this process in greater detail.

Key Generation

We use a modified ECDSA key generation algorithm to make a signature scheme that is fully distributed. In this scheme, the private key is communally selected by the distributed group using a combination of hidden random secrets.

The threshold key derivation algorithm is given in *Algorithm 1*.

The algorithm is extensible, and each step of the algorithm can be executed by every participant synchronously without a dealer or in groups or individuals or dealers. This implementation is fully compatible with the current bitcoin protocol. Any signatories will appear to an outside observer or verifier as if they were signed in the standard manner. Consequently, there is no way to tell if a key has been generated in the standard format or using our enhanced protocol.

Signature Generation

The concept of threshold signature generation is described in [Shamir, 1979]. Algorithm 3 is related to a procedure reported in [Feldman, 1987] that was based on DH based systems and has been modified to allow for ECDSA.

We've extended this process such that it is fully compatible with both Bitcoin transaction processing and signing. This also extends to multisig transactions where it is possible to require distributed keys for each of the multiple signatures that are necessary.

Re-sharing the Private Key

This process can be extended to introduce an entirely distributed key re-sharing scheme. This re-distribution is completed when the current participants execute one round of *Algorithm 2* adding the resulting zero-share to the participant's private key share. The new shares will be randomly distributed if one participant has introduced a random value.

This process allows us to additively mask the private key share while not altering the actual private key.

Threshold ECDSA Signature Derivation

The threshold ECDSA signature creation system derived using the ideas related to the threshold DSS signature generation protocol found in [Feldman, 1987] which followed the scheme developed in [Shamir, 1979].

Verification

Our system allows for the off-line signing and verification of messages before any value being transferred to a known bitcoin address. Each of the parties can calculate and validate an address independently using the processes noted in *Algorithm 1*. Hence, all participants can are aware that their share is valid before any exercise that requires funding a bitcoin address. For this process, although verification schemes are possible, they are unnecessary. Any threshold participant who chooses to send an invalid signature slice is in effect voting for the negative. That is, a vote to not sign the message and hence not complete the transaction in bitcoin is achieved from inaction. The impact is as if they did not sign a message at all.

Algorithm 2 provides a method where participants can have their share consistency verified. If a threshold of non-malicious participants has been maintained, it is possible to exclude any known malicious participants on rekeying. Hence key slices can be updated while not allocating fresh slices to known malicious participants, allowing for the refreshing of the key in a manner that also allows for reallocations of slices.

In environments where trust is particularly scarce and malicious adversaries are to be expected as the norm, we can further enhance the robustness of our verification process increasing the ability to defend against an $j/2$ passive and an $j/3$ active adversary [Ben-Or, 1989; Rabin, 1988] when completing secure multiparty computations.

We can enhance the robustness of the system using the additional process:

1. Let D_a be the secret shared among the j participants on a polynomial $\mathrm{A}(x)$ of degree $(k-1)$.
2. Separately participants $p_{(i)}$ have a share $D_{a(i)}$ of D_a and $D_{a(i)}G\forall\left(i\in 0,...,j\right)$ which are made available to the group.
3. All participants next share a secret b using *Algorithm 2* such that each participant $p_{(i)}$ has a new hidden share $D_{b(i)}$ of D_b on a polynomial of degree $(k-1)$.

Note: $D_{b(i)}=\sum_{h=1}^{j}D_{b(i)}^{(h)}$, where $D_{b(i)}^{(h)}$ is the sub-share submitted to participant $p_{(i)}$ from participants $p_{(h\neq i)}$.

4. The participants use *Algorithm 2* so that each participant $p_{(i)}$ has a new hidden share $\mathrm{Z}_{(i)}$ on a polynomial of degree $(2k-1)$ of which the free term equals zero.
5. Each participant $p_{(i)}$ publishes $D_{a(i)}D_{b(i)}^{(h)}G\forall\left(h\in 0,...,j\right)$ and $D_{a(i)}D_{b(i)}G$ to the group.
6. Each participant $p_{(h\neq i)}$ can verify the validity of $D_{a(i)}D_{b(i)}^{(h)}G$ as they have $D_{b(i)}^{(h)}$ and $D_{a(i)}G$.
7. Also, participant $p_{(i)}$ can further verify that $D_{a(i)}D_{b(i)}G=\sum_{h=1}^{j}D_{a(i)}D_{b(i)}^{(h)}$.

Any participants can determine if other participants are acting maliciously with this system.

V. Distributed Key Generation

It is possible to complete the implementation of both distributed autonomous corporations (DACs) and distributed autonomous social organisations (DASOs) in a secure manner through this scheme. We have shown that any k members can represent such a group through an identification scheme (including through digital certificates signed and published by a certification authority) and that any k members can construct a digital signature on behalf of the organisation. This system extends to the signing of bitcoin transactions that verify without any distinguishing feature and provide for the transfer of value. These authentication schemes are proven secure.

Method and implementation

As the protocol encrypts the secret information required to be sent between participants using ECC based on a patent 42 derivation, it is both possible and advisable to collate all messages into a single packet sent to all users such that validation can be done against potentially compromised or hostile participants when necessary.

Signature generation is proposed by a coordinating participant $p_{(c)}$. By default, any key slice can act as the coordinating participant and the requirements come down to the individual implementation of the protocol. On the creation of a valid raw transaction by $p_{(c)}$, the transaction and the message hash of the transaction are broadcast to all participants $p_{(i\neq c)}$ using an encrypted channel.

A. Generate ephemeral key shares $\mathbf{D}_{k(i)}$

The participants generate the ephemeral key $\mathbf{D}_k$, uniformly distributed in $\mathbb{Z}_n^*$, with a polynomial of degree *(k-1)*, using *Algorithm 1*, which creates shares $\left(D_{k(1)},\cdots,D_{k(j)}\right) \overset{((k-1),j))}{\leftrightarrow} D_k \bmod n$.

Shares of D_k are maintained in **secret** being held individually by each participant.

B. Generate mask shares α_i

Each participant generates a random value α_i, uniformly distributed in $\mathbb{Z}_n^*$ with a polynomial of degree (k-1), using *Algorithm 1* to create shares $\left(\alpha_1,\cdots,\alpha_j\right) \overset{((k-1),j)}{\leftrightarrow} \alpha \bmod n$. These are used to multiplicatively mask $\mathbf{D}_{k(i)}$.

The shares of α_i are **secret** and are maintained by the corresponding participant.

C. Generate mask shares β_i, c_i

Execute *Algorithm 2* twice using polynomials of degrees 2*(k-1)*.

Denote the shares created in these protocols as $\left(\beta_1,\cdots,\beta_j\right) \overset{(2(k-1),j)}{\leftrightarrow} \beta \bmod n$ and $\left(c_1,\cdots,c_j\right) \overset{(2(k-1),j)}{\leftrightarrow} c \bmod n$. These are used as additive masks. The polynomial must be of degree 2*(k-1)* because the numbers being masked involve the products of two polynomials of degree *(k-1)*. This doubles the required number of shares needed to recover the secret.

The shares of *b* and *c* are to be kept **secret** by the participants.

D. **Compute** digest of message *m*: $e = H(m)$

This value is checked against the received hash of the transaction obtained from $p_{(c)}$.

E. **Broadcast** $\upsilon_i = D_{k(i)}\alpha_i + \beta_i \bmod n$ and $\omega_i = G \times \alpha_i$

Participant P_i broadcasts $\upsilon_i = D_{k(i)}\alpha_i + \beta_i \bmod n$ and $\omega_i = G \times \alpha_i$.

If no response is received from P_i, the value used is set to *null*.

Note: $\left(\upsilon_1, \cdots, \upsilon_j\right) \overset{(2(k-1),j)}{\leftrightarrow} D_k \alpha \bmod n$

F. **Compute** μ = **Interpolate** $\left(\upsilon_1, \cdots, \upsilon_j\right) \bmod n$

Interpolate() [2]:

Where $\left\{\upsilon_1, \cdots, \upsilon_n\right\}\left(j \geq (2k-1)\right)$ forms a set, such that at most of (k-1) are *null* and all the residual values reside on a *(k-1)*-degree polynomial $F(.)$, then $\mu = F(0)$.

The polynomial can be computed using ordinary polynomial interpolation. The function "Interpolate()" is the Berlekamp-Welch Interpolation [2] and is defined[4] as an error correcting algorithm for BCH and Reed-Solomon codes.

G. **Compute** θ = **Exp-Interpolate** $\left(\omega_1, \cdots, \omega_j\right)$

Exp-Interpolate() [10]:

If $\left\{\omega_1, \cdots, \omega_j\right\}\left(j \geq (2k-1)\right)$ is a set where at most (k-1) values are *null* and the remaining values are of the form $G \times \alpha_i$, and each α_i *exists on* some *(k-1)*-degree polynomial $H(\cdot)$, then $\theta = G \times H(0)$.

Thi value can be computed by $\theta = \sum_{i \in V} \omega_i \times \lambda_i = \sum_{i \in V} \left(G \times H(i)\right) \times \lambda_i \, \beta$ for which V is a (k)-subset of the correct ω_i *values* and further, λ_i represent the resultant Lagrange interpolation coefficients. The polynomial can be computed by using the Berlekamp-Welch decoder.

H. **Compute** $(R_x, R_y) = \theta \times \mu^{-1}$

I. **Assign** $r = R_x \bmod q$ If $r = 0$, go to step A.

Each participant $p_{(i)}$ computes their slice of r_i in step J. The coordinator $p_{(c)}$ can use these values to reconstruct s if they have received a threshold number of responses.

J. **Broadcast** $s_i = D_{k(i)}\left(e + D_{A(i)} r\right) + c_i \bmod n$

---

[4] http://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html
See also Whittaker, E. T. and Robinson, G. "Lagrange's Formula of Interpolation." §17 in The Calculus of Observations: A Treatise on Numerical Mathematics, 4th ed. New York: Dover, pp. 28-30, 1967.
https://jeremykun.com/2015/09/07/welch-berlekamp/

If a response is not solicited/received from P_i, the values used are set to *null*.

Note: $\left(s_1,\cdots,s_j\right) \overset{(2(k-1),j)}{\leftrightarrow} D_k\left(m+D_A r\right) \bmod n$ $(s_1,...,s_n) \longleftrightarrow k(m + D_A r) \bmod n$.

K. **Compute** s = **Interpolate** $(s_1,...,s_n) \bmod n$

If $s = 0$, go to step I.

See above for the meaning of the function *Interpolate()*.

Each participant $p_{(i)}$ computes their slice of s_i in step J. The coordinator $p_{(c)}$ can use these values to reconstruct s if they have received a threshold number of s_i responses.

L. Return (r,s)
M. Replace the signature section of the raw transaction and broadcast this to the network.

Dealer distribution of slices

The system derived above can be made much more flexible through the introduction of group shares. In this manner, the allocation of shares can be split between a dealer, multiple dealers, a group containing no dealers or any possible combination of the above in any level of hierarchical depth.

By replacing the value d_A and its corresponding key slice $d_{A(i)}$ with a value derived using the same algorithm, we can create hierarchies of votes. For example, we can create a scheme that simultaneously integrates shares that are derived from:

1) Dealer based distributions
2) Multiple Dealers
3) No Dealer

Hence, the scheme is extensible and can be made to incorporate any business structure or organisational system.

The allocation of slices is also extensible. Deploying an uneven allocation process allows us to add weighting on shares. In the scheme displayed in figure 1, we can create a hypothetical organisation with five top-level members. This, however, does not require setting the value of n=5 equally weighted shares. In our hypothetical organisation, we could set the voting structure for the top-level schema as follows:

- Threshold(0) 61 Shares
- D_{L0_2} 15 Shares
- D_{L1} 15 Shares
- D_{L2} 15 Shares
- D_1 45 Shares
- D_2 10 Shares

Here we have set n=100. As noted this is an arbitrary value that can reflect any organisational structure. The organisation in figure 1 allows for a veto scenario (D_1) and through the introduction of multi-layered allocation allows for any voting structure that can be imagined.

What is often missed in a multilevel hierarchical structure is that although we have allocated slices of the secret these do not need to be evenly distributed and further, the ownership of subgroups does not

need to mirror that of other levels. In figure 1 we have a seemingly powerful block controlling 45% of the total number of shares in 75% of the threshold. If we then look at the lower-level allocation of the shares, the scenario becomes far more complex. We can create cross-ownership with individuals holding voting shares in multiple levels and positions on the table.

The distributions in Table 3 are defined as (Shares held, Threshold, Allocation {n}).

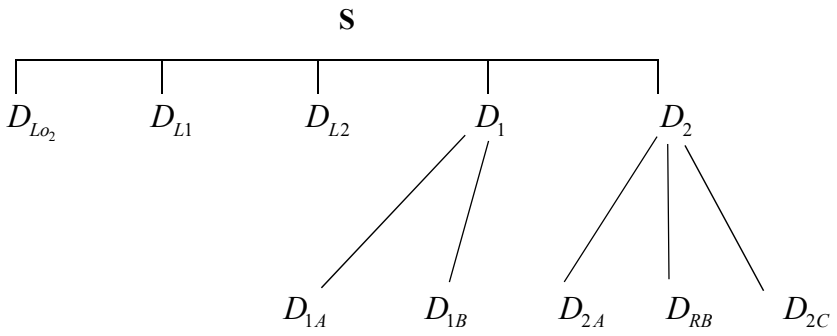


Figure 1. Secrets can be distributed in reconstructive hierarchies.

From the table above (3), we see that participants P1 and P2 each hold sway over the votes but that they coalition with participant P4 provides either P1 or P2 with a sufficient voting block as long as P1 or P2 does not veto the vote.

As there are no limits to the implementation and structure of the voting format in our system, we can use this to create any organisational hierarchy that can be imagined as well as ensuring secure backup and recovery methodology.

Table 3 Hypothetical organisation structure

Participant	Level 0	Level 1	S.Votes (Max)	S.Votes (min)
P 1	D_{L0_2} (15,61,100)	D_{1A} (5,6,10) D_{2A} (3,8,10)	70	15
P 2	D_{L1} (15,61,100)	D_{1B} (5,6,10) D_{2B} (3,8,10)	70	15
P 3	D_{L2} (15,61,100)		15	0
P 4		D_{2C} (6,8,10)	10	0

The result is that we can have veto powers and voting rights that are assigned to higher level signing shares. In our example ownership of the shares in S could be held at, D_{L0_2}, D_{1A} and D_{2A}.

Secure Multi-Party Computation

Secure multi-party function computation with n participants, $p_{(1)},...,p_{(i)},...,p_{(n)}$ is a problem based on the need to assess a function $\mathbb{F}\left(x_1,...,x_i,...,x_n\right)$, involving $x_{(i)}$, a secret value provided by $p_{(i)}$ that

is required to be maintained in confidence such that no participant $p_{(j\neq i)}$ or external party gains any knowledge of $x_{(i)}$. Hence, the objective is to preserve the confidentiality of each participant's values while being able to guarantee the exactness of the calculation.

In this scenario, the trusted third-party T collects all the values $x_{(i:1\ldots n)}$ from the various participants $p_{(i:1\ldots n)}$ and returns the calculation. This design works only in an idealised world where we can implicitly trust T. Where there is any possibility that T could either be malicious, rogue or compromised, the use of a trusted third party becomes less viable. This scenario mirrors existing elections where the participants are the voters, and the trusted third party is played by government.

It has been proven [Bar-Ilan, 1989] that any value that can be computed in a secure manner using a trusted third-party may also be calculated without a trusted party while maintaining the security of the individual secrets $x_{(i)}$. The protocols presented in this paper are secure against private computation and provide secure computation even where a non-threshold group of compromise participants can collaborate.

Simple multiplication

Where we have two secret values, x and y that a distributed among n participants $p_{(i:1\ldots n)}$, it is possible to compute the product xy while simultaneously maintaining the secrecy of both input variables x and y as well as ensuring that the individual secrets $x_{(i:1\ldots n)}$ and $y_{(i:1\ldots n)}$ are maintained by participant $p_{(i)}$ retaining the confidentiality.

In this scheme, x and y are each shared between a threshold group of participants using a polynomial of degree (k-1). Each participant $p_{(i)}$ can multiply their share of $x_{(i:1\ldots n)}$ on a polynomial of degree (k-1) of x and $y_{(i:1\ldots n)}$ in a polynomial of degree (k-1) on y.

Introducing *Algorithm 2*, returns the participant $p_{(i)}$ share of $z_{(i)}$, a polynomial of degree (2k-1). With this value, each participant $p_{(i)}$ calculates the value $x_{(i)}y_{(i)} + z_{(i)}$.

The return value for $x_{(i)}y_{(i)} + z_{(i)}$ represents a valid share of the calculation for $x.y$ on a polynomial of degree (2k-1). Any participant or a coordinator acting for the threshold number of shares can use the return value held by each participant to calculate the true value of $x.y$ without obtaining any knowledge of the individual shares.

Simple addition

Where we have two secret values, x and y that are distributed among n participants $p_{(i:1\ldots n)}$, it is possible to compute the product $x + y$ while simultaneously maintaining the secrecy of both input variables x and y as well as ensuring that the individual secrets $x_{(i:1\ldots n)}$ and $y_{(i:1\ldots n)}$ that are maintained by participant $p_{(i)}$ retain the confidentiality.

As per the process for simple multiplication, each participant $p_{(i)}$ calculates the value $x_{(i)} + y_{(i)} + z_{(i)}$. The calculation of $z_{(i)}$ is not necessary, but adds a further level of randomness and confidentiality to the process.

The return value for $x_{(i)} + y_{(i)} + z_{(i)}$ represents a valid share of the calculation for $x + y$ on a polynomial of degree (2k-1). Any participant or a coordinator acting for the threshold number of shares can use the return value held by each participant to calculate the true value of $x + y$ without obtaining any knowledge of the individual shares.

If the participants are less hostile, this can be simplified as an $x_{(i)} + y_{(i)}$ addition without the additional step.

Inverse or reciprocal

For a distributed secret value, $x \bmod n$ which is distributed confidentially between j participants as $x_{(i:1\ldots j)}$, it is possible to generate shares of the polynomial associated with the value for $x^{-1} \bmod n$ while not revealing any information that could disclose the values $x_{(i)}$, x or x^{-1} [Gennaro, 1996]. Again, each participant $p_{(i)}$ maintains a share of the value x represented by $x_{(i)}$ over a polynomial of degree (k-1).

Using Algorithm 1, each participant creates a share $x_{(i)}$ of an unknown secret $x.y$ on a polynomial of degree (k-1). Each participant then runs Algorithm 2 to calculate $(k-1)$ of a zero secret on a polynomial of degree (2k-1). Each participant $(2k-1)$ performs the calculation to compute the value $x_{(i)}y_{(i)} + z_{(i)}$.

Using the ***Interpolate()*** routine presented above, each participant can calculate the value of $\mu = x_{(i)}y_{(i)} + z_{(i)}$ returning the value μ from the collected values of μ_i. Each participant can then calculate the value of $\mu^{-1} \bmod n$.

These values are sufficient such that any participant $p_{(i)}$ can compute the associated share of x_i^{-1} using $\zeta_i = \gamma_i \mu^{-1}$ one a polynomial of degree (2k-1). The Berlekamp-Welch decoding scheme [Berlekamp, 1968] provides one of several methods that can be used to complete this process.

Assignment

The ability to sign a transaction in a verifiable and provable manner provides the opportunity to prove ownership privately and even relinquish or exchange ownership of the bitcoin private key and associated bitcoin address without publicly moving anything on the Blockchain. In this manner, a bitcoin address can be funded, and the contents of that address may be transferred or sold without leaving a public record. As this process is a threshold system, the assignment of key slices can be achieved securely without further settlement recorded on the Blockchain.

In this way, we can separate the ownership of a note already settled on the Blockchain from the process of transacting that note.

CLTV

A bitcoin message or in more common parlance, transaction, can be created with the inclusion of a CLTV [BIP 65] entry. With this addition, the transaction can be made recoverable even in the catastrophic loss of all key slices or if multiple slices from an entity are deemed untrustworthy or lost in a manner that does not allow for the secure reconstruction of a signature with a minimum threshold.

This is further possible were an entity is using a third-party service and desires to ensure that that service cannot hold or deny access to the keys. In constructing a bitcoin transaction with a time based fail safe, the user knows that a malicious third-party or a compromised exchange site or bank cannot extort them for access to their keys. As a worst-case scenario, the compromise to a catastrophic level would lead to the time-based reversal of a transaction to a predefined address based on a CLTV condition. This predefined address can be created using the protocols noted within this paper. As such, it is possible to construct a series of transactions and keys that cannot be readily compromised.

VI. Security considerations

Benger et. al. (2014) offered one example of ECDSA private key recovery using a Flash and reload methodology. This occurrence is but one example of attacks against system RAM and Cache. These methods leave the use of procedures such as that of Shamir's SSS [1979] wanting as they reconstruct the private key. Moreover, in any scenario with a private key is reconstructed that any amount of time a requirement for trust is introduced. It is necessary in this scenario to rely on the systems and processes of the entity holding the private key.

Even if the trusted party is not malicious, there is a necessity to rely on their processes. As we have seen from many recent compromises, this reliance on reconstructing the private key leaves avenues of attack.

As both a drop-in replacement for the existing ECDSA implementations as well as being completely transparent and compatible with the current bitcoin protocol, no hard fork or soft fork is required for its implementation, and the implementation is indistinguishable from any current transaction. Our application can treat individuals as separate participants allowing for the group signing of keys with a recovery function. As an example, a two of two scheme can be implemented using four key slices where the online wallet provider or exchange maintains two key slices and the end user maintains two slices. The exchange and the user would each have a two of two process over their key slices which would then be used in conjunction with each other for the secure signing of a message when required.

VII. Future Work

This paper will be extended to allow for the automated group allocation of shares to patent number 15087315 format hierarchical wallets [Wright, 2016]. This previous group calculation will be extended from simple signing into a hierarchical group structure allowing for the automated creation of change addresses and hierarchical sub-keys that are associated with an initial shared secret group.

Additionally, it will be possible to incorporate key reconstruction within a bitcoin message and transaction. This system would allow the creation of a P2SH address associated directly with a standard single address transaction. The merging of these forms of transactions will open many opportunities for smart contracts.

The full ramifications of this paper will be extended in a series of daughter papers. In these, we will introduce a fully distributed ecosystem that can coexist within the Bitcoin Blockchain as a fully distributed allocation system based on blind signatures that extends the work of Chaum [1983] in creating a fully distributed model of what was sought with digi-cash.

A further paper will introduce a series of multi-party computations allowing bitcoin to act as a black box computational algorithm. The use of zero-knowledge proofs for programming when coupled with our group distributed share system allows for a flexible design of computational systems and smart contracts that can be stored in the Blockchain while also being run without knowledge of other parties.

The final edition will be the extension into a dynamic threshold group management structure for ewallets and calculational entities. This will allow for the secure addition and removal of parties

involved in the sharing and distribution of keys in a manner that provides external group allocations without the need to enact settlement on the Bitcoin Blockchain.

VIII. Conclusions

The scheme forms the foundation of what bitcoin sought to achieve with the introduction of a group signature process. The addition of a fault tolerable signing system with the coupling of a distributed key creation system removes all centralisation and trust requirements. Many systems will evolve with the need for trust. The removal of the need to trust the underlying infrastructure and powers these enabling them to differentiate themselves based on their real merits.

Moreover, the introduction of an implicitly decentralised system allows for the creation of more robust and resilient protocols. The compatibility between ECDSA [Johnson, 2001] and Shamir's SSS [Shamir, 1979] has allowed us to introduce a system that extends bitcoin with a new verifiable secret sharing scheme. This system is far more efficient than anything derived by Feldman [Feldman, 1987] or Pedersen [Pedersen, 1992] while losing nothing in security.

In this paper, we have presented a system that extends the functionality of bitcoin without the requirement for a change in the base protocol. Using our system;

1. a trusted third-party is no longer required for the selection or distribution of a key secret,
2. a distributed banking exchange system can be created that does not rely on third-party trust,
3. each member or group of members may independently verify that the share of the secret key that is held corresponds to the bitcoin address and public key advertised,
4. a protocol exists to refresh the private key slices to mitigate the effects of eavesdropping and related attacks, and
5. no trusted third-party is required for the group signing of transactions and messages.

As our system prevents sensitive data from ever appearing in memory, we have completely solved many extant security risks.

IX. References

1) Bar-Ilan, J. Beaver, "Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds", Proc. of 8th PODC, pp. 201-209, 1989.
2) Berlekamp, Elwyn R. (1968), Algebraic Coding Theory, McGraw–Hill, New York, NY.
3) Benger, N., van de Pol, J., Smart, N.P., Yarom, Y.: "Ooh Aah... Just a Little Bit": A Small Amount of Side Channel Can Go a Long Way. In: Batina, L., Robshaw, M. (eds.) Cryptographic Hardware and Embedded Systems | CHES 2014, LNCS, vol. 8731, pp. 75-92. Springer (2014)
4) Ben-Or, M., Goldwasser, S., Wigderson, A.: "*Completeness theorems for noncryptographic fault-tolerant distributed computation*". In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. pp. 1–10. STOC '88, ACM, New York, NY, USA (1988)
5) BIP 65 OP_CHECKLOCKTIMEVERIFY https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki
6) Chaum, David (1983). "*Blind signatures for untraceable payments*" (PDF). Advances in Cryptology Proceedings of Crypto. 82 (3): 199–203.
7) Dawson, E.; Donovan, D. (1994), *"The breadth of Shamir's secret-sharing scheme*", Computers & Security, 13: Pp. 69–78
8) Desmedt. Yuo (1987). "Society and Group Oriented Cryptography: A New Concept". In A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology (CRYPTO '87), Carl Pomerance (Ed.). Springer-Verlag, London, UK, UK, 120-127.

9) Feldman. P. "*A practical scheme for non-interactive verifiable secret sharing*". In Proceedings of the 28th IEEE Annual Symposium on Foundations of Computer Science, pages 427–437, 1987.
10) Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: "*Robust threshold DSS signatures*". In: Proceedings of the 15th Annual International Conference on Theory and Application of Cryptographic Techniques. pp. 354–371. EUROCRYPT'96, SpringerVerlag, Berlin, Heidelberg (1996)
11) Ibrahim, M., Ali, I., Ibrahim, I., El-sawi, A.: "*A robust threshold elliptic curve digital signature providing a new verifiable secret sharing scheme*". In: Circuits and Systems, 2003 IEEE 46th Midwest Symposium on. vol. 1, pp. 276–280 (2003)
12) Johnson, D., Menezes, A., Vanstone, S.: "*The elliptic curve digital signature algorithm (ecdsa)*". International Journal of Information Security 1(1), 36–63 (2001)
13) Kapoor, Vivek, Vivek Sonny Abraham, and Ramesh Singh. "*Elliptic Curve Cryptography*." Ubiquity 2008, no. May (2008): 1-8.
14) Knuth, D. E. (1997), "*The Art of Computer Programming, II: Seminumerical Algorithms*" (3rd ed.), Addison-Wesley, p. 505.
15) Koblitz, N. "An Elliptic Curve Implementation of the Finite Field Digital Signature Algorithm" in Advances in Cryptology — Crypto '98. Lecture Notes in Computer Science, vol. 1462, pp. 327-337, 1998, Springer-Verlag.
16) Liu, C. L. (1968), "Introduction to Combinatorial Mathematics", New York: McGraw-Hill.
17) National Institute of Standards and Technology: FIPS PUB 186-4: "*Digital Signature Standard*" (DSS) (2003)
18) Pedersen, T.: "*Non-interactive and information-theoretic secure verifiable secret sharing*". In: Feigenbaum, J. (ed.) Advances in Cryptology — CRYPTO '91, LNCS, vol. 576, pp. 129–140. Springer (1992)
19) Rabin T. & Ben-Or. M. (1989) "Verifiable secret sharing and multiparty protocols with honest majority". In Proc. 21st ACM Symposium on Theory of Computing, pages 73--85, 1989.
20) Shamir, Adi (1979), "*How to share a secret*", Communications of the ACM, 22 (11): Pp. 612–613
21) Wright, C. & Savanah, S. (2016) "Determining a common secret for two Blockchain nodes for the secure exchange of information" "Application Number: 15087315". 2016: n. pag. UK

X.Appendix

To Compute $\beta = Exp - Interpolate(w_i, \ldots, w_n)$

$Exp - Interpolate(\ ) \quad \rightarrow *$

If $\{w_i, \ldots, w_n\}(n \geq 2t+1)$ is a set of values, such that at most, t are null and the remaining are of the form $G \times a_i$, where the a_i's lie on some $(k-1)-$ Degree Polynomial $H(\bullet)$, then $\beta = G \times H(\varnothing)$. This is computed using:

$$\beta = \Sigma_{i \in v} w_i \times \lambda_i$$

$$= \Sigma_{i \in v} (G \times H(i)) \times \lambda_i$$

where ν is a $k-$ subset of the correct w_i's and λ_i's are the corresponding Lagrange interpolation coefficients.