The MuSig Schnorr Signature Scheme

December 7, 2018

This report investigates multi (MuSig), which is provably secure in the *plain public-key model*. However, the case of interactive signature aggregation where each signer signs their own message must still be proven by a complete security analysis.

Multi-signatures are a form of technology used to add additional security for cryptocurrency transactions. A multi-signature protocol which allows a group of signers to produce a short, joint signature on a common message.

Contents

1 Introduction

1.1 Schnorr signatures

Schnorr signatures produce a smaller on-chain size, support faster validation and have better privacy. They natively allow for combining multiple signatures into one through aggregation. They permit more complex spending policies.

Signature aggregation also has its challenges. This included the rogue-key attack, where a participant steals funds using a specifically constructed key. This is easily solved for simple multi-signatures, however, through an enrollment procedure, where the keys sign themselves, supporting it across multiple inputs of a transaction requires *plain public-key security*, meaning there is no setup.

There is an additional attack, termed the Russel attacks, after Russel O'Connor, who was discovered that for multi-party schemes a party could claim ownership of someone else's key and so spend their other outputs.

That being said, Peter Wuille has been able to address some of these issues and has provided a solution which refines the Bellare-Neven (BN) scheme. He also discussed the performance improvements that were implemented for the scaler multiplication of the BN scheme and how they enable batch validation on the blockchain.[1]

1.2 MuSig

A multi-signature scheme is a combination of a signing and verification algorithm, where multiple signers (each with their own private/public key) jointly sign a single message, resulting in a single signature. This can then be verified by anyone knowing the message and the public keys of the signers.

MuSig is a simple multi-signature scheme that is novel in combining:

1. Support for key aggregation;
2. Security in the *plain public-key model*.

There are two versions of MuSig, that are provably secure, which differ based on the number of communication rounds:

- Three-round MuSig only relies on the Discrete Logarithm (DL) assumption, on which ECDSA (Elliptic Curve Digital Signature Algorithm) also relies
- Two-round MuSig instead relies on the slightly stronger One-More Discrete Logarithm (OMDL) assumption

1.3 Key aggregation

The term *key aggregation* refers to multi-signatures that look like a single-key signature, but with respect to an aggregated public key that is a function of only the participants' public keys. Thus, verifiers do not require the knowledge of the original participants' public keys- they can just be given the aggregated key. In some use cases, this leads to better privacy and performance. Thus, MuSig is effectively a key aggregation scheme for Schnorr signatures.

There are other multi-signature schemes that already exist that provide key aggregation for Schnorr signatures, however they come with some limitations, such as needing to verify that participants actually have the private key corresponding to the pubic keys that they claim to have. *Security in the plain public-key model* means that no limitations exist. All that is needed from the participants is their public keys. [2]

2 Applications of multi-signatures

Recently most obvious use case for multi-signatures, is with regards to Bitcoin, where it can function as a more efficient replacement of n-of-n multisig scripts and other policies that permit a number of possible combinations of keys. For these, a native multi-signature scheme means that what is left is one signature per transaction input.

A key aggregation scheme also lets one reduce the number of public keys per input to one, as a user can send coins to the aggregate of all involved key, rather than including them all in the script. This leads to smaller on-chain footprint, faster validation, and better privacy.

Instead of creating restrictions with one signature per input, one signature can be used for the entire transaction. Key aggregation cannot be used across multiple inputs, as the public keys are committed to by the outputs, and those can be spent independently. MuSig can be used here (with key aggregation done by the verifier).

On a technical standing, in order to combine all the transaction inputs' signatures, a multi-signature scheme is not necessary, instead an aggregate signature scheme can be used. The distinction is simply that in an aggregate signature, each signer has their own message, instead of one message shared by all.

Aggregate signatures can be categorized as being:

- Interactive: Interactive aggregate signatures (IAS) require the signers to cooperate, while non-interactive schemes all the aggregation to be done by anyone
- Non-interactive: These allow the aggregation to be done by anyone

No non-interactive aggregation schemes are known that only rely on the DL assumption, but interactive ones are trivial to construct: where a multi-signature scheme has every participant sign the concatenation of all messages. The paper by Blockstream, focusing on key aggregation for Schnorr Signatures shows that this is not always a desirable construction, and gives an IAS variant of BN with better properties instead. [2]

3 Simple Schnorr Multi-Signatures

The paper describes a new Schnorr-based multi-signature scheme called MuSig, which is provably secure in the *plain public-model*. This means that signers are only required to have a public key, but they do not have to prove knowledge of the private key corresponding to their public key to some certification authority or to other signers prior to engaging the protocol.

This new scheme provides improvements to Bellare and Neven (ACM-CCS 2006) and its variants by Bagherzandi *et al.* (ACM-CCS 2008) and Ma *et al.* (Des. Codes Cryptogr., 2010) in two respects:

Write Sections on Bagerzandi and Ma

1. It is simple and efficient, as it has the same key and signature size as standard Schnorr signatures;
2. It allows *key aggregation*, where the joint signature can be verified just as a standard Schnorr signature with respect to a single "aggregated" public key which can be computed from the individual public keys of the signers. [3]

3.1 Multi-signatures

Introduced by Itakura and Nakamura [4], multi-signature protocols allow a group of signers (that individually possess their own private/public key pair) to produce a single signature σon a message m. Verification of the given signature σcan be publicly performed given the message and the set of public keys of all signers.

A simple way to change a standard signature scheme into a multi-signature scheme is to have each signer produce a stand-alone signature for m with its private key and to then concatenate all individual signatures.

The transformation of a standard signature scheme to a multi-signature scheme needs to useful and practical, thus the newly calculated multi-signature scheme must produce signatures where the size is independent of the number of signers and similar to that of the original signature scheme. [3]

3.1.1 Rogue attacks

Rogue attacks are a significant concern when implementing multi-signature schemes. Here a subset of corrupted singers, manipulate the public keys computed as functions of the public keys of honest users, allowing them to easily produce forgeries for the set of public keys (despite them not knowing the associated secret keys.

Proposals from [5], [6], [7], [8], [9], [10], [11] were thus undone before a formal model was put forward along with a provably secure scheme from Micali, Ohta, and Reyzin. [12]Unfortunately, despite being provably secure this scheme is costly and an impractical interactive key generation protocol. [3]

A means of generically preventing rogue-key attacks is to make it mandatory for users to prove knowledge (or possession [13]) of the secret key during public key registration with a certification authority. Certification authority is a setting known as the knowledge of secret key (KOSK) assumption. The pairing-based multi-signature schemes by Boldyreva [14] and Lu *et al.* [15] rely on the KOSK assumption in order to maintain security. However, this as can be seen from [16]and [13] this assumption is problematic.

As it stands, the Bellare and Neven [16] provides the most practical multi-signature scheme, based on the Schnorr signature scheme, which is provably secure that does not contain any assumption on the key setup. Since the only requirement of this scheme is that each potential signer has a public key, this setting is referred to as the *plain-key model.*

3.1.2 Schnorr signature scheme

The Schnorr signature scheme uses:[17]

- A cyclic group G of prime order p
- A generator $g$of G
- A hash function H
- A private/public key pair is a pair $(x, X)\epsilon\{0, ..., p-1\}\times G$ where $X = g^x$
- To sign a message m, the signer draws a random integer $r$in Z_p, computes $R = g^r$, $c = H(X, R, m)$, and $s = r + cx$
- The signature is the pair (R, s) , and its validity can be checked by verifying whether $g^s = RX^c$

The above described is referred to as the so-called "key-prefixed" variant of the scheme, which sees the public key hashed together with R and m[18]. This variant was thought to have a better multi-user security bound than the classic variant [19], however in [20] the key-prefixing was seen as unnecessary to enable good multi-user security for Schnorr signatures.

For the development of the new Schnorr-based multi-signature scheme [3], key-prefixing seemed a requirement for the security proof to go through, despite not knowing the form of an attack. The rationale also follows the process in reality, as messages signed in Bitcoin always indirectly commits to the public key.

3.1.3 Design of a Schnorr multi-signature scheme

The naive way to design a Schnorr multi-signature scheme would be as follows:

- A group of $n$signers want to cosign a message m
  - Let $L = \{X_1 = g^{x_1}, ..., X_n = g^{x_n}\}$ be the multi-set of all public keys [1]
  - Each cosigner randomly generates and communicates to others a share $R_{i=}g^{r_i}$
  - Each of the cosigners them computes $R = \Pi_{i=1}^{n} R_i$, $c = H(\tilde{X}, R, m)$
  - Where $\tilde{X} = \Pi_{i=1}^{n} X_i$ is the product of individual public keys, and a partial signature $s_i = r_i + cx_i$
  - Partial signatures are then combined into a single signature (R, s) where $s = \Sigma_{i=1}^{n} si \bmod p$
  - The validity of a signature (R, s) on message m for public keys $\{X_1, ...X_n\}$,is equivalent to $g^s = R\tilde{X}^c$ where $\tilde{X} = \Pi_{i=1}^{n} X_i$ and $c = H(\tilde{X}, R, m)$
  - Note that this is exactly the verification equation for a traditional key-prefixed Schnorr signature with respect to public key $\tilde{X}$, a property termed *key aggregation*

However, as mentioned above, [7], [9], [10] and[12] these protocols are vulnerable to a rogue-key attack where a corrupted signer sets its public key to $X_1 = g^{x_1}(\Pi_{i=2}^{n} X_i)^{-1}$, allowing the signer to produce signatures for public keys $\{X_1, ...X_n\}$ by themself.

3.1.4 Micali-Ohta-Reyzin Multi-signature scheme

The Micali-Ohta-Reyzin multi-signature scheme [12]solves the rogue-key attack using a sophisticated interactive key generation protocol.

[1] No constraints are imposed on the key setup, the adversary thus can choose corrupted public keys at random, hence the same public key can appear more than once in L

3.1.5 Bellare and Neven signature scheme

Bellare-Neven (BN) [16]proceeded differently in order to avoid any key setup. It is a more widely known plain public-key multi-signature scheme, that does not support key aggregation. It is possible to use BN multi-signatures where the individual keys are MuSig aggregates. BN multi-signature scheme is secure without such assumptions. Below are details:

- Call $L = H(X_{1,}X_{2...})$
- Each signer chooses a random nonce r_i and shares $R_i = r_iG$ with the other signers
- Call R the sum of the R_i points
- Each signer computes $s_i = r_i + H(L, X_{i,}R, m)x_i$
- The final signature is (R, s) where $s$is the sum of the s_i values
- Verification requires $sG = R + H(L, X_{1,}R, m)X_2 + ...$

Technically, BN has a pre-commit round, where the signers initially reveal $H(R_i)$to each other, prior to revealing the R_i points themselves. This step is a requirement in order to prove security under the DL assumption, but it can be dismisses if instead the OMDL assumption is accepted.

Furthermore, when an IAS is desired (where each signer has their own message), $L = H((X_1, m_{1)}, (X_2, m_2), ...)$ and $s_i = r_i + H(L, R, i)x_i$ is used for signing (and analogous for verification).

The resulting signature does not satisfy the normal Schnorr equation anymore, nor any other equation that can be written as a function of a combination of the public keys; the key aggregation property is lost in order to gain security in the plain public-key model.

This is where MuSig comes in. It recovers the *key aggregation property without losing security:*

- Call $L = H(X_1, X_2...)$
- Call X the sum of all $H(L, X_i)X_i$
- Each signer chooses a random nonce r_i, and shares $R_i = r_iG$ with the other signers
- Call R the sum of the R_i points
- Each signer computes $s_i = r_i + H(X, R, m)H(L, X_i)x_i$
- The final signature is (R, s) where $s$is the sum of the s_i values
- Verification again satisfies $sG = R + H(X, R, m)X$

So what was needed was to define X not as a simple sum of the individual public keys X_i,but as a sum of multiples of those keys, where the multiplication factor depends on a hash of all participating keys. [2]

Their main idea is to have each cosigner use a distinct "challenge" $c_i$when computing their partial signature $s_i = r_i + c_i x_i$,defined as $c_i = H(< L > X_i, R, m)$,where as before

Bellare and Neven showed that this yields a multi-signature scheme provably secure in the *plain public-key* model under the Discrete Logarithm assumptions, modeling $H$and H'as random oracles. However, this scheme does not allow key aggregation anymore since the entire list of public keys is required for verification.

3.1.6 The new scheme

The new proposed Schnorr-based multi-signature scheme can be seen as a variant of the BN scheme, allowing key aggregation in the *plain public-key model.* This scheme consists of three rounds, the first two being exactly the same as in BN. Challenges c_i are changed from $c_i = H(< L > X_i, R, m)$ to $c_i = H_{agg}(< L > X_i).H_{sig}(\tilde{X},R,m)$, where $\tilde{X}$is the so-called aggregated public key corresponding to the multi-set of public keys $L = \{X_1, ...X_n\}$, defined as $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i c} = R\tilde{X}^c$ where $c = H_{sig}(\tilde{X}, R, m)$.

Basically , the key aggregation property has been recovered and can now be enjoyed by the naive scheme, which respect to a more complex aggregation key $\tilde{X} = \prod_{i=1}^{n} X_i^{a_i c}$. $c = H_{sig}(< L >, R, m)$ yields a secure scheme, however does not allow key aggregation since verification is impossible without knowing all the individual singer keys.

3.1.7 The benefits of key aggregation

- If a group of $n$signers want to authorize which all of them agree, but do not necessarily wish to reveal their individual public keys
- They can privately compute the aggregated key $\tilde{X}$ corresponding to their multi-set of public keys and publish it as an ordinary (non-aggregated) key.
- Signers are ensured that all of them will need to cooperate to produce a signature which is valid under $\tilde{X}$, whereas verifiers will not even learn that $\tilde{X}$is in fact an aggregated key.
- Moreover, $\tilde{X}$can be computed by a third party just from the list of public keys, without interacting with the signers.
- This property will prove instrumental for obtaining a more compact and privacy-preserving variant of so-called n-of-n multi-signature transactions in Bitcoin.

Two variants of the BN multi-signature scheme have been previously proposed.

3.1.8 The Bagherzandi *et al.* scheme

Bagherzandi *et al.*[21] reduced the number of rounds from three to two using an homomorphic commitment scheme. Unfortunately, this increases the signature size and the computational cost of signing and verification.

3.1.9 The Ma *et al.* scheme

Ma *et al.*[22]proposed a variation based on Okamoto's signature scheme[23], which involved the "double hashing" technique, which sees the reduction of the signature size compared to [21]while using only two rounds.

However, neither of these two variants allow for key aggregation.

Multi-signature schemes supporting key aggregation are easier to come by in the KOSK model. In particular, Syta *et al.* proposed the CoSi scheme which can be seen as the naive Schnorr multi-signature scheme described earlier where the co-signers are organized in a tree structure for fast signature generation.

4 Interactive Aggregate Signatures

In some situations, it may be useful to allow each participant to sign a different message rather than a single common one. An interactive aggregate signature (IAS) is one where each signer has its own message m_i to sign, and the joint signature proves that the i-th signer has signed m_i . These schemes are more general than multi-signature schemes, however they are less flexible than non-interactive aggregate signatures [24, 25] and sequential aggregate signatures [LMRSO4].

According to Bellare and Neven [BN06], they suggested a generic way to turn any multi-signature scheme into an IAS scheme by the signer running the multi-signature protocol using as message the tuple of all public keys/message pairs involved in the IAS protocol.

For BN's scheme and the scheme of the Blockstream signature scheme, this does not increase the number of communication rounds as messages can be sent together with shares R_i.

However, the problem arise with the generic construction in the *plain public-key model.* Below is a closer look at transformation and show that is subtly fails to provide the strongest security guarantees in the *plain public key model.*

4.1 Syntax and security model

The syntax is adapted as follows. An IAS scheme Π is a triple of algorithms ($\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}$). The randomization key generation algorithm takes no input and returns a private/public key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow_{\$} \mathsf{KeyGen}()$. The signature algorithm Sign is run by each participant on input its key pair $(\mathsf{sk}, \mathsf{pk})$, a message m, and a set of public key/message pairs for other cosigners $S' = \left\{(pk_1^{'}, m_1^{'}), ...(pk_{n-1}^{'}, m_{n-1}^{'})\right\}$ such that (pk,m)$\notin S'$; it returns a signature $\sigma$for

the set of public key/message pairs $S = S' \cup \{(\mathsf{pk},m)\}$.[2] The deterministic verification algorithm Ver takes as input a set of public key/message pairs $S = \{(\mathsf{pk}_1,m_1), ..., (\mathsf{pk}_n,m_n)\}$ and a signature σ, and returns 1 if the signature is valid for S and 0 otherwise.

As another means, one can specify the signature algorithm to take as input a key pair

4.2 Revisions

A previous version of this paper, dated January 15, 2918, proposed a 2-round variant of MuSig, where the initial commitment round is omitted claiming a security proof unsure the One More Discrete Logarithm (OMDL) assumptions [BP02][BNPS03]. However, a flaw in the security proof was discovered by Drijvers *et al.* [DEFN18], how also showed through a meta-reduction that this scheme cannot be proved secure using an algebraic black box reduction under the DL or OMDL assumption (assuming the OMDL problem is hard).

In more details, observe that in the 2-round variant of MuSig, an adversary (controlling public keys $X_2, ..., X_n$) can impose the value of $R = \Pi_{i=1}^{n} R_i$ used in signature protocols since he can choose $R_2, ..., R_n$after having received $R_1$from the honest signer (controlling public key $X_1 = g^{x_1}$). This forbids to use the textbook way of simulating the honest signer in the Random Oracle model without knowing x_1 by randomly drawing s_1 and c, computing

There is no attach currently known against the 2-round variant of MuSig and that it might be secure although thais is not provable under standard assumptions from existing techniques.

5 The Discrete Logarithm Problem

Definition 1 (DL problem)

- Let $\mathbb{G}$, p, g be group parameters- it is fixed, but the bit length k and $p$can be regarded as a security parameter if necessary.
- An algorithm $\mathcal{A}$ is said to $(t, \varepsilon solve$the DL problem with with respect to $\mathbb{G}, p, g$ if on input a random group element X, it runs in time at most t and returns $x \in \{0, ..., p-1\}$ such that

6 Elliptic Curve Digital Signature Algorithm

Currently in Bitcoin ECDSA is implemented. To sign a message m we hash it and treat this hash as a number: $z = hash(m)$. We also need a random or random-looking number k. We prefer not to trust random number generators

[2] S is a set, i.e., no public key/message pair repeats, even though the same public key might appear mulitple times. This is without loss of generality since any repeated public key/message pair in S can be deleted by the signature/verification algorithm.

(too many failures and vulnerabilities are related to bad random number generators) so we usually use RFC6979 to calculate deterministic k based on our secret and the message we are signing.

Using a private key pk we can generate a signature for message m consisting of two numbers: r($x$coordinate of the random point $R = kG$ and $s = (z+rpk)/k$. Then, using our public key $P = pkG$ anyone can verify our signature by checking that point $(\frac{z}{s})G + (\frac{r}{s})P$ has $x$coordinate equal to r.

Insert Image

This algorithm is very common, however it can be improved. Firstly, signature verification includes inversion $(1/s)$ and two points multiplications an these operations are very computationally heavy. In Bitcoin every node has to verify all the transactions. This means that when you broadcast a transaction, thousands of computers will have to verify your signature. Making verification process simpler will be very beneficial even if signing is more difficult.

Secondly, every node has to verify every signature individually. In the case of m-of-n multisig transaction node may even have to verify the same signature several times. For example, transaction of 7-of-11 multisig input will contain 7 signatures and require from 7 to 11 signature verifications on every node in the network. Also such transaction will take a huge amount of space in the block and you will have to pay large fees for that.

7 Schnorr signatures

Schnorr signatures are generated slightly differently. Instead of two scalars (r, s)we use a point R and a scalar . Like ECDSA, R is considered a random point on the elliptic curve ($R = kG$). Second part of the signature is calculated slightly differently: $s = k + hash(P, R, m)pk$. Here pk is your private key, $P = pkG$ is your public key, m is the message. Then once can verify this signature by checking that $sG = R + hash(P, R, m)P$

Insert image: Visualization of the Schnorr signature verification

This equation is linear, so equations can be added and subtracted with each other and still stay valid. This leads to a nice feature of Schnorr signatures.

7.1 Batch validation

To verify a block in Bitcoin blockchain we need to make sure that all signatures in the block are valid. If one of them is not valid we don't care which one- we just reject eh whole block and that's it.

With ECDSA every signature has to be verified separately. Meaning that if we have 1000 signatures in the block we will need to compute 1000 inversions and 2000 point multiplication. In total approximately 3000 heavy operations.

With Schnorr signatures we can add up all the signature verification equations and save some computational power. In total for a block with 1000 transactions we need to verify that:

$(s1 + s2 + ... + s1000)G = (R1 + ... + R1000) + (hash(P1, R1, m1)P1 + hash(P2, R2, m2)P2 + ... + hash(P1000, R1000, m1000)P1000$

Here we have a bunch of point additions (almost free in sense of computational power) and 1001 point multiplication. This is already a factor of 3 improvement- we meed to compute roughly one heavy operation per signature.

Insert image: Batch validation of two signatures. As verification equation is linear the sum of several equations is valid as soon as all signatures are valid. We save some computational power as scalar and point additions are much easier than point multiplication.

7.2 Key aggregation

There is a need to keep one's bitcoin safe, so we might want to use at least two different private keys to control bitcoins. Once we will use on a laptop or a phone and another one- on a hardware wallet/cold wallet. So when one of them is compromised we still have control over our bitcoins.

Currently it is implemented via 2-of-2 multi-signature script. This requires two separate signatures to be included in the transaction.

With Schnorr signatures we can use a pair of private keys $(pk1, pk2)$ and generate a shared signature corresponding to a shared public key $P = P1+P2 = pk1G + pk2G$. To generate this signature we need to choose a random number on every device $(k1, k2)$, generate a random point $Ri = kiG$, add them up to calculate a common $hash(P, R1 + R2, m)$ and then get $s1$ and $s2$ from every device $(si = ki + hash(P, R, m)pki)$. Then we can add up these signatures and use a pair $(R, s) = (R1 + R2, s1 + s2)$as our signature for shared public key P. Everyone else won't be able to say if it is an aggregated signature or

7.3 Merkle multi-signatures

MuSig and key aggregation require all signers to sign a transaction. What happens if you want to make a 2-of-3 multisig? Can one use signature aggregation or will we have to use our usual OP_CHECKMULTISIG and separate signatures?

it is possible, but with a small change in the protocol. A new op-code similar to OP_CHECKMULTISIG can be developed that checks if aggregated signature corresponds to a particular item in the Merkle tree of public keys.

For example, if we use a 2-of-3 multisig with public keys $P1$, $P2$ and $P3$, then we need to construct a Merkle tree of aggregated public keys for all combinations we can use: $(P1, P2)$, $(P2, P3)$, $(P1, P3)$ and put the root in the locking script. To spend bitcoins we provide a signature and a proof that our public keys is in the tree. For 2-of-3 multisig there are only 3 elements in the tree and the proof will consist fo two hashes-the one we want to use and its neighbor. For a 7-of-11 multisig there will be already 11!/7!/4!=300 possible key combinations and the proof will require 8 elements. In general the number of elements in the proof scales almost linear with the number of keys in multisig (its $log2(n!m!/(n-m)!)$.

But with the Merkle tree of public keys we are not limited to *m-of-n* multisignatures. We can make a tree with any public keys we want. For example

, if we have a laptop, a phone, a hardware wallet and a recovery seed, we can construct a structure that would allow us to spend bitcoins with a laptop and a hardware wallet, a phone and a hardware wallet or just with a recovery seed. This is currently not possible just with OP_CHECKMULTISIG- only if you construct much more complicated script with branches.

8 Applications

Each input contains a signature of a modified version of the transaction to be validated with its referenced output's key. In fact some outputs even require multiple signatures to be spent. Transactions spending such an output are often referred to as m-of-n multi-signature transactions [And11], and the current implementation corresponding to the trivial way of building a multi-signature scheme by concatenating individual signatures. Additionally, a threshold policy can be enforced where only m valid signatures out of the n possible ones are needed to redeem the transaction (again this is the most straightforward way to turn a multi-signature scheme into some kind of basic threshold signature scheme).

Today Bitcoin uses ECDSA signatures [ANS05][NIS13] over the secp256k1 curve [SEC10] to authenticate transactions. As Bitcoin nodes fully verify all transactions, signature size and verification time are important design considerations while signing time is much less so. Besides, signatures account for a large part of the size of Bitcoin transactions. Because of this, using multi-signatures seems appealing. However, designing multiparty ECDSA signature schemes is notably cumbersome [MR01][GGN16][Lin17]due to the modular inversion involved in signing, and moving to Schnorr signatures would definitely help deploy compact multi-signatures. While several multi-signature schemes could off an improvement over the currently available method, two properties increase the possible impact:

- The availability of key aggregation removes the need for verifiers to see all the involved key, improving bandwidth, privacy, and validation cost
- Security under the *plain public-key model* enables multi-signatures across multiple inputs of a transaction, where the choice of signers cannot be committed to in advance. This greatly increases the number of situations in which mulit-signatures are beneficial.

8.1 Introduction

Bitcoin contains every transaction since the system's inception, resulting in a final state, the set of unspent coins. Each unspent coin has an associated value (expressed as a multiple of the currency unit, 10^{-8} bitcoin) and a programmable public key of the owner. Every transaction consumes one or more coins, providing a signature for each to authorize its spending, and creates one more new coins, with a total value not larger than the value of the consumed coins.

Bitcoins uses a programmable generalization of a digital signature scheme. Instead of a public key, a predicate that determines spend-ability is included in every output (implemented in a concise programming language, called *Bitcoin Script).* When spending, instead of a signature, a witness that satisfies the predicate is provided. In practice, most output predicates effectively correspond to a single ECDSA verification. This is also how Bitcoin supports a naive version of multi-signatures with a threshold policy: coins can be assigned a predicate that requires valid signatures for multiple public keys. Several use cases for the exist, including low-trust escrow services [GBGN17] and split-device security. While using the predicate language to implement multi-signatures is very flexible, it is inefficient in terms of size, computational cost, and privacy.

As a global consensus system, kept in check by the ability for every participant to validate all updates to the ledger, the size of signatures and predicates, and the computational cost for verifying them are the primary limiting factors for its scalability. The computational requirements for signing, or the communication overhead between different signers are far less constrained. Bitcoin does not have any central trusted party, so it is not generally possible to introduce new cryptographic schemes that require a trusted setup. Finally, to function as a currency, a high degree of fungibility and privacy is desirable. Among other things, this means that ideally the predicate of coins fo not lead information about the owner. In particular, if several styles of predicates are in use, the choice may reveal what software or service is being used to manage it.

8.2 Native multi-signature support

An obvious improvement is to replace the need for implementing n-of-n multi-signatures in an ad-hoc fashion with a constant-size multi-signature primitive like Bellare-Neven, While this is on itself an improvement in terms of size, it still requires the predicate itself-whose size also matters- to contain all of the signers' public keys. Key aggregation improves upon this further, as a single-key predicate can be used instead which is both smaller and has lower computational cost for verification. It also improves privacy, as the participant keys and their count remain private to the signers.

When generalizing to the m-of-$n$scenario, several options exist. One is to forego key aggregation, and still include all potential signer keys in the predicates while still only producing a single signature for the chosen combination of keys. Alternatively, a Merkle tree [Mer87] where the leaves are permitted combinations of keys (in aggregated form) can be employed. The predicate in this case would take as input an aggregated public key, a signature with it, and a proof. Its validity would depend on the signature being valid with the provided key, and the proof establishing that the key is in fact one the leaves of the Merkle tree, identified by its root hash. This approach is very generic, as it works for any subset of combinations of keys, and as a result has very good privacy as the exact is not visible from the proof. It is only feasible however when the total number of combinations is tractable.

Some key aggregation schemes that do not protect against rogue-key attacks

can be used instead in the above cases, under the assumption that the sender is given a proof of knowledge/possession for the receivers' private keys. However, these schemes are difficult to prove secure except by using very large proofs of knowledge [MOR01]. As those proofs of knowledge/possession for the receivers' private keys. However, these schemes are difficult to prove secure except by using very large proofs of knowledge/possession do not need to be seen by verifiers they are effectively certified by the sender's signature on the transaction which includes the predicate- they do not burden validation. However, passing them around to senders is inconvenient, and easy to get wrong .Using a scheme that is secure in the *plain public-key model* categorically avoids these concerns.

Another alternative is to use an algorithm whose key generation requires a trusted setup, for example in the KOSK model. While many of these schemes have been proven secure [Bol03, LOS+06], they rely on mechanisms that are usually not implemented by certification authorities [BN06, RY07].

8.3 Cross-input multi-signatures

The previous sections explained how the numbers of signatures per input can generally by reduced to one, but we would like to go further, and replace it with a single signature per transaction. Doing so requires a fundamental change in validation semantics, as the validity of separate inputs is not longer independent. As a result , the outputs can no longer be modeled as predicates. Instead, we model them as functions that return a boolean plus a set of zero or more public keys.

Overall validity requires all returned booleans to be True and a multi-signature of the transaction with L the union of all returned keys.

More concretely, this can be implemented by providing an alternative to the signature checking opcode OP_CHECKSIG and related opcodes in the Script language. Instead of returning the result of an actual ECDSA verification, they always return True, but additionally add the public key with which the verification would have taken place to a transaction-wide multi-set of keys. Finally, after all inputs are verified, a multi-signature present in the transaction is verified against that multi-set. In case the transaction spends inputs from multiple owners, they will need to collaborate to produce the multi-signature, or choose to only use the original opcodes. Adding these new opcodes is possible in backward-compatible way.

8.4 Protection against rogue-key Attacks

When taking cross-input signatures input signatures, there is no published commitment to the set of signers, as each transaction input can independently spend an output that requires authorization from distinct participants. This functionality was not restricted as it would then interfere with fungibility improvements like CoinJoin [Max13]. Due to the lack of certification, security against rogue-key attacks is of great importance.

If it is assumed that transactions used a single multi-signature that was vulnerable to rogue-attacks, like the simpler schemes previously described, an attacker could identify an arbitrary number of outputs he wants to steal, with the public keys $X_1, ..., X_{n-t}$ and then use the rogue-key $X_{n-t+1}, ..., X_n$ such that he can sign for the aggregated key $\tilde{X}$.He would then send a small amount of his own money to outputs with predicates corresponding to the keys $X_{n-t+1}, ..., X_n$. Finally, he can create a transaction that spends all of the victim coins together with the ones he just created by forging a multi-signature for the whole transaction.

It can be seen that in the case of mulit-signatures across inputs, theft can occur by merely being able to forge a signature over a set of keys that includes at least one key not controlled by the attacker, just what the *plain public-key model* considers a win for the attacker. This is in contrast to the single-inout multi-signature case where theft is only possible by forging a signature for the exact (aggregated)keys contained in an existing output. As a result, it is no longer possible to rely on proofs of knowledge/possession that are private to the signers.

To analyze the impact of multi-signatures, a simulation on Bitcoin's historical blockchain was performed to determine the potential space saving. Figure 3 shows the cumulative blockchain size together with what the size would be if all transactions' signatures were replaced with just one per transition, giving an indication of what could have been saved if MuSig had been used since the beginning. Note that this only encompasses the savings from using multi-signatures, and does not induce the savings that are possible from key aggregation.

8.5 $m - of - n$ multi-signatures

Muti-signatures refer to requiring more than one key to authorize a transaction. Currently, standard transactions on the Bitcoin network can be referred to as single-signature transactions, as they require only one signature, from the owner of the private key associated with the Bitcoin address. However, the Bitcoin network supports much more complicated transactions which can require the signatures of multiple people before the funds can be transferred. These are often referred to as m-of-n transactions, where m represents the signatures required to spend, while n represents the signatures possible.[26]

8.5.1 Use cases for multi-signatures

When $m = 1$ and $n > 1$, this shared wallet could be used for small group funds that do not require much security. It is the least secure multi-sig option because it is not multi-factor. Any compromised individual would jeopardize the entire group. Examples of use cases include funds for a weekend or evening event, or a shared walled for some kind of game. Besides being convenient to spend from, the only benefit of this setup is that all but one of the backup/password pairs could be lost and all of the funds would be recoverable.

When $m = n$, there is a partner wallet, which brings with it some nervousness as no keys can be lost. As the number of signatures required increases, the risk also increases. This type of multi-signature can be considered as a hard multi-factor authentication.

When $m < 5n$, it is considered a buddy account, which could be used for spending from corporate group funds. Consequence for the colluding minority need to greater than possible benefits. It is considered less convenient than a shared wallet, but much more secure.

When $m > 5n$, a consensus account is termed. The classic multi-signature wallet is a 2 of 3 and is a special case of a consensus account. 2 of 3 has the best characteristics for creating new bitcoin addresses and for secure storing and spending. One compromised machine does not compromise the funds. A password can be lost and the funds can still be recovered. If done correctly, off-site backups are created during wallet setup. The way to recover funds is known by more than one party. The balance of power with a multi-signature wallet can be shifted by having one party control more keys than the other parties. If one party controls multiple keys, there is a greater risk of those keys not remaining as multiple factors.

When $m = 5n$, it is referred to as a split account, and is an interesting use case, as there would be 3 of 6 where one person holds 3 keys and 3 people hold 1 key. In this way one person could control their own money, but the funds could still be recoverable even if the primary key holder were to disappear with all of his key. As n increases, the level of trust in the secondary parties can decrease. A good use case might be a family savings account that would just automatically become an inheritance account if the primary account holder were to die.[26]

9 Boneh-Lynn-Shacham Signatures

Boneh-Lynn-Shacham signature scheme is based on the computational Diffie-Hellman assumption on certain and hyper-elliptic curves. The signature length is half the size of a DSA signature for a similar level of security.

Schnorr signatures combine all signatures and public keys in the transaction into a single key and a signature and nobody will find out that they correspond to multiple keys. In addition block validation is faster, as all signatures can be validated at once. However there are a few problems with Schnorr signatures:

- Multisig schemes require several rounds of communication. This can be a hindrance with regards to cold storage
- With signature aggregation we have to rely on random number generator- we can't choose random point R deterministically like we do in ECDSA
- m-of-n multisig scheme is tricky- we need to make a merkle tree of public keys that can get pretty large for large m of n
- We can't combine all signatures in the block to a single signature

BLS signatures can fix all of the above. We don't need random numbers at all, all signatures in the block can be combined to a single signature, m-of-n multisig is very simple and we don't need several communication rounds between signers. In addition to that BLS signatures are 2 times shorter than Schnorr or ECDSA-signature is not a pair, but a single curve point

9.1 Hashing to the curve

Normally with ECDSA and Schnorr we hash the message and use this hash in the signing algorithm as a number. For BLS signatures we need a slightly modified hashing algorithm that hashes directly to the elliptic curve. The easiest way is to hash a message as usual and treat the result as an x-coordinate of a point. Elliptic curves usually have about 2^{256} points and SHA-256 hashing algorithm also gives a 256-bit result. But for every valid x-coordinate there are two points with positive and negative y-coordinate (just because if $(x, y)$is on the curve $y^2 = x^3 + ax + b$ then $(x, -y)$is also on the curve). This means that our hash has roughly 50% probability to find two points for some $x$and 50% to find none.

To find a point for any message we can try hashing several times by appending a number to the hash and incrementing it on fail.

9.2 Key aggregation and n-of-n multisignature

If we are using multisignature addresses, we are signing the same transaction with different keys. In this case, we can do key aggregation like in Schnorr, where we combine all signatures and all keys to a single pair of a key and a signature. If we consider a common 3-of-3 multisig scheme:

A simple way to combine them is to add all the signatures and all the keys together. The result will be a signature $S = S1 + S2 + S3$ and a key $P = P1 + P2 + P3$. It is easy to see that the same verification equation still works:

$e(G, S) = e(P, H(m))$

$e((G, S) = e(G, S1 + S2 + S3) = e(G, (pk1 + pk2 + pk3)H(m)) = e((pk1 + pk2 + pk3)GH((m)) = e(P1 + P2 + P3, H(m)) = e(P, H(m))$

Similarly to Schnorr there needs to be protection against rogue key attacks. This can be achieved by asking every co-signer to prove that they have private keys for their public keys (by signing their public keys), or that some nonlinearity to the scheme is added making rogue key attacks impossible. Instead of summing up all the keys and signatures, we multiply them by a certain number and then add:

$S = a1S1 + a2S2 + aS3$

$P = a1P1 + a2P2 + a3P3$

Here coefficients of the signatures and keys are calculated deterministically from the public key of the signer and all other public keys:

$ai = hash(Pi, \{P1, P2, P3\})$

References

[1] Blockstream, "Schnorr Signatures for Bitcoin - BPASE '18," 2018. [Online]. Available: https://blockstream.com/2018/02/15/schnorr-signatures-bpase/

[2] P. Wuille, "Key Aggregation for Schnorr Signatures," 2018. [Online]. Available: https://blockstream.com/2018/01/23/musig-key-aggregation-schnorr-signatures/

[3] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr Multi-Signatures with Applications to Bitcoin," pp. 1–34, 2018.

[4] K. Itakura, "A public-key cryptosystem suitable for digital multisignatures," *NEC J. Res. Dev.*, vol. 71, 1983.

[5] C.-M. Li, T. Hwang, and N.-Y. Lee, "Threshold-multisignature schemes where suspected forgery implies traceability of adversarial shareholders," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1994, pp. 194–204.

[6] L. Harn, "Group-oriented (t, n) threshold digital signature scheme and digital multisignature," *IEE Proceedings-Computers and Digital Techniques*, vol. 141, no. 5, pp. 307–313, 1994.

[7] P. Horster, M. Michels, and H. Petersen, "Meta-Multisignature schemes based on the discrete logarithm problem," in *Information Securityâthe Next Decade*. Springer, 1995, pp. 128–142.

[8] K. Ohta and T. Okamoto, "A digital multisignature scheme based on the Fiat-Shamir scheme," in *International Conference on the Theory and Application of Cryptology*. Springer, 1991, pp. 139–148.

[9] S. K. Langford, "Weaknesses in some threshold cryptosystems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 74–82.

[10] M. Michels and P. Horster, "On the risk of disruption in several multiparty signature schemes," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 1996, pp. 334–345.

[11] K. Ohta and T. Okamoto, "Multi-signature schemes secure against active insider attacks," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 82, no. 1, pp. 21–31, 1999.

[12] S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 245–254.

[13] T. Ristenpart and S. Yilek, "The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2007, pp. 228–245.

[14] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *International Workshop on Public Key Cryptography*. Springer, 2003, pp. 31–46.

[15] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2006, pp. 465–485.

[16] M. Bellare and G. Neven, "Multi-Signatures in the Plain Public-Key Model and a General Forking Lemma," *Acm Ccs*, pp. 390–399, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1180453

[17] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[18] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012.

[19] D. J. Bernstein, "Multi-user Schnorr security, revisited." *IACR Cryptology ePrint Archive*, vol. 2015, p. 996, 2015.

[20] E. Kiltz, D. Masny, and J. Pan, "Optimal security proofs for signatures from identification schemes," in *Annual Cryptology Conference*. Springer, 2016, pp. 33–61.

[21] A. Bagherzandi, J. H. Cheon, and S. Jarecki, "Multisignatures Secure under the Discrete Logarithm Assumption and a Generalized Forking Lemma," *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*, p. 449, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1455770.1455827

[22] C. Ma, J. Weng, Y. Li, and R. Deng, "Efficient discrete logarithm based multi-signature scheme in the plain public key model," *Designs, Codes and Cryptography*, vol. 54, no. 2, pp. 121–133, 2010.

[23] **T. Okamoto, "Provably secure and practical identification schemes and corresponding signature schemes," in *Annual International Cryptology Conference*. Springer, 1992, pp. 31–53.**

[24] **D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 416–432.**

[25] **M. Bellare, C. Namprempre, and G. Neven, "Unrestricted aggregate signatures," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2007, pp. 411–422.**

[26] **B. W. Contributors, "Multisignature," 2017.**