Schnorr Multi-Signature

Implementing MuSig in CryptoKernel

BenEischen BenFeeser LucasNovak

0 | Schnorr Multi-Signature

Why do we want to implement MuSig?

MuSig allows for key aggregation and increase anonymity

Currently:
Alice, Bob, Carol sign
Tx

Signature:

ABC

With MuSig:
Alice, Bob, Carol sign
Tx

Signature:

D

https://medium.com/@Bitcom21/crypto-innovation-spotlight-schnorr-signatures-a83748f16a4
https://blockstream.com/2018/01/23/musig-key-aggregation-schnorr-signatures.html

1 | Schnorr Multi-Signature

Multi-signature applications

1-of-2: Husband and wife petty cash joint account

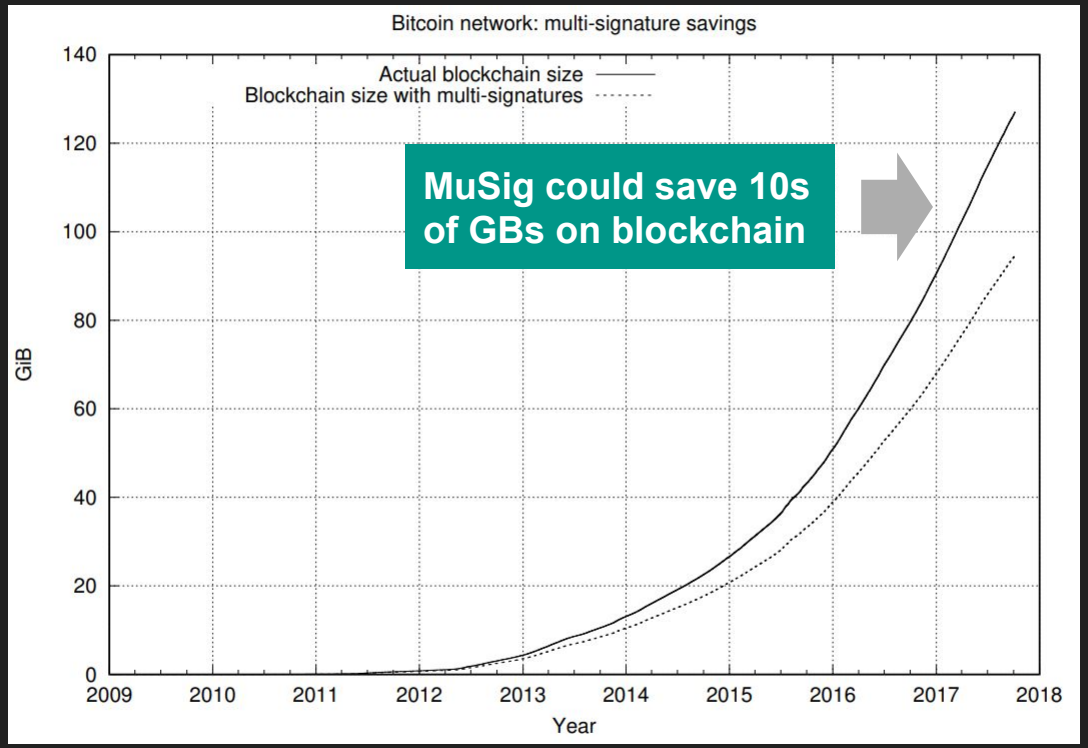
2-of-3: Buyer-seller with trustless escrow

2-of-3: A board of three directors maintaining organization’s funds

2-of-3: Decentralized cold storage vault

https://en.bitcoin.it/wiki/Multisignature

2 | Schnorr Multi-Signature
Implications for Scaling Bitcoin



https://eprint.iacr.org/2018/068.pdf

3 | Schnorr Multi-Signature

Math behind Schnorr signatures

- Signatures are $(R,s) = (rG, r + H(X,R,m)x)$
- Verification requires $sG = rG + H(X,R,m)xG$
  $= R + H(X,R,m)X$

Notation

- x, x_1, x_2, … are private keys with corresponding public keys X, X_1, X_2, … ($X_i = x_iG$, with G the generator)
- The message being signed is m
- $H()$ is a cryptographic hash function
- r, r_1, r_2, … are random nonces chosen by the signer

https://blockstream.com/2018/01/23/musig-key-aggregation-schnorr-signatures.html

4 | Schnorr Multi-Signature

Math behind MuSig

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

Allows for key aggregation, reduces size, and increases privacy

https://blockstream.com/2018/01/23/musig-key-aggregation-schnorr-signatures.html

5 | Libraries

Understanding CryptoKernel and cschnorr

Library	Author	Lang	SLOC	Purpose	URL
CryptoKernel	James Lovejoy	C++, lua	14K	Most alt-coins contain 99% BTC core code; exists to create more flexibility and a standard kernel to be expanded upon for alt-coins	https://github.com/mit-dci/CryptoKernel
cschnorr	James Lovejoy	C	1.5K	Implements Schnorr signatures, multisignature, committed R	https://github.com/metalicjames/cschnorr

6 | Implementation

Modifying CryptoKernel with cschnorr

```
#include <cschnorr/multisig.h>
class Schnorr
 // EC_POINT musig_key->pub->A to base64_encode(buf, 33)
std::string getPublicKey();
 // BIGNUM musig_key->a to base64_encode(buf, 32)
std::string getPrivateKey();
 // base64_decode(pubKey) to EC_POINT musig_key->pub->A
bool setPublicKey(const std::string& publicKey);
 // base64_decode(privateKey) to BIGNUM musig_key->a
bool setPrivateKey(const std::string& privateKey);
```

7 | Implementation

Modifying CryptoKernel with cschnorr

```
// continued
class Schnorr

// musig_sign(ctx, sig, pub, pubkeys, message, ...)
std::string sign(const std::string& message);

// musig_verify(ctx, sig, pub, message, ...)
bool verify(const std::string& message, const std::string& signature);

class SchnorrTest
void testInit();
void testKeygen();
void testSignVerify();
void testPassingKeys();
```

https://github.com/mit-dci/CryptoKernel/pull/27

8 | Key Learnings

- C++ / OpenSSL libraries have a steep learning curve
- Pros and cons of building a system (CryptoKernel) from the ground up with backwards compatibility top of mind
- Understanding multi-signature and Schnorr mathematically
- Potential applications and size savings of multi-signature
- Experience working on open source cryptocurrency project