Ben Eischen, Ben Feeser, Lucas Novak
MAS.S62 Cryptocurrency Engineering & Design
14 May 2018

Implementing MuSig in CryptoKernel

I. Schnorr Signatures

Schorr signatures were invented by Claus-Peter Schnorr with his seminal paper “Efficient Signature Generation by Smart Cards” in 1991. Schnorr’s original paper was focused on efficiently generating public keys for communication between smart cards and terminals. The limited computing power of smart cards required a scheme that minimized computational demands. Most of the computational demands required in the signature process are not dependent on the message and therefore can be pre-processed during a processors idle time [Schnorr].

The math behind Schnorr signatures works as follows [Schnorr]:

Let G be a generator
Let x be the private key with a corresponding public key X such that xG=X.
Let m be the message being sent
Let H() be a hash function
Let r be a random nonce

A Schnorr Signature is (R,s) = (rG, r + H(X,R,M)x)

A message and a signature can be verified if the following equation is true:
sG = R + H(X,R,m)X

This verification scheme is correct as:
s = r + H(X,R,M)x
sG = rG + H(X,R,M)xG
sG = R + H(X,R,M)X

Given (R,s) and G one can’t figure out what r or x is:
R = rG -- can’t divide by G to find r
r= s- H(X,R,M)x -- can’t solve for r as r is also in hash function and don’t know x
s-r = H(X,R,M)x -- can’t solve for x as x is in hash function and don’t know r

While a valid way for people to sign individually, Schnorr signatures also allow for signature aggregation. While this can’t be done naively by summing up public keys, as it would allow for rogue key attacks[1], it can be done using a slightly more complex protocol called MuSig [Poelstra].

[1] Given public keys are known by all individuals, one could claim to have a fake public key that is equal to their public key minus the sum of all other public keys. This effectively makes the sum of all public keys equal to their actual public key, giving the individual the ability to sign by themselves [Poelstra].

II. MuSig

MuSig was first proposed by Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille as a Schnorr-based multi-signature protocol that would allow for signature aggregation and security in the plain public-key model[2] [Poelstra].

The logic being MuSig works as follows[3][4]:

Let x_i be the private key of signer i with a corresponding public key X_i such that $x_iG=X_i$.
Let m be the message being sent
Let H() be a hash function
Let r_i be a random nonce chosen by signer i
Let $L = H(X_1, \dots X_n)$
Let $X = \text{sum}(H(L,X_i)X_i)$ for i = 1… n

Each signer creates $R_i = r_iG$ and shares it with the other signers
Each signer then computes $R = \text{sum}(R_i)$ for i = 1 … n
Let $s_i = r_i + H(X,R,m)H(L,X_i)x_i$
Let $s = \text{sum}(s_i)$ for i = 1 … n

The signature is the (R,s)

A signature and message can then be verified with the equation:
$sG = R + H(X,R,m)X$

This verification scheme is correct as:
$s_1 + s_2 + \dots + s_n = r_1 + H(X,R,m)H(L,X_1)x_1 + \dots + r_n + H(X,R,m)H(L,X_n)x_n$
$s = (r_1+\dots +r_n) + H(X,R,m)(H(L,X_1)x_1 + \dots + H(L,X_n)x_n)$
$sG = (r_1+\dots +r_n)G + H(X,R,m)(H(L,X_1)x_1G + \dots + H(L,X_n)x_nG)$
$sG = R + H(X,R,m)X$

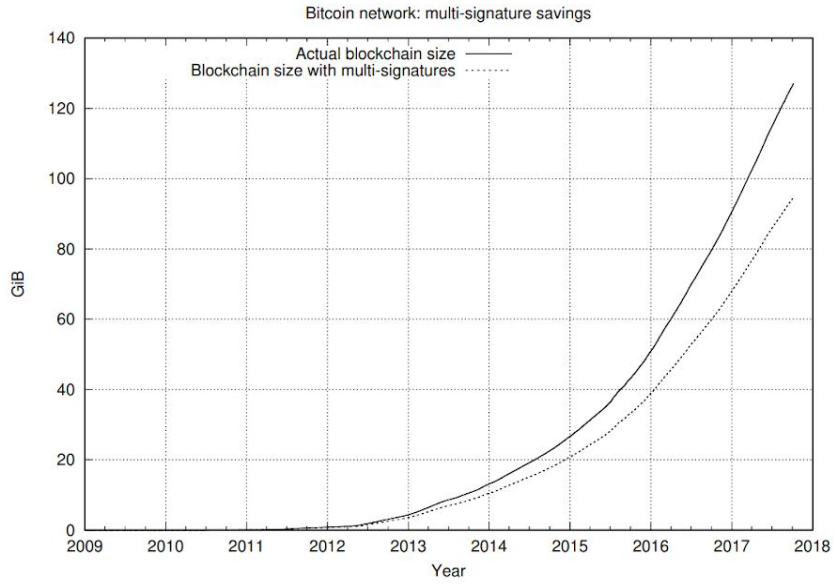
Given R,s, G, L, X, and every r_i ,x_i except r_j,x_j one can't figure out r_j or x_j:
Know: R = Gr
$R = G(r_1 + \dots r_n)$ -- can't divide by G so can't figure out an individual r_j given R

Know: sG = R + H(X,R,m)X
$sG = R + H(\text{sum}(H(X_i,L)X_i), R, m)\text{sum}(H(X_i,L)X_i)$-- the hashing means that someone who knows all other $H(X_i,L)X_i$ can't figure out what X_j is.

[2] Signing only requires a public key rather than proof that one owns the corresponding private key [Poelstra]
[3] For a complete proof see [Poelstra].
[4] Recently the proof that MuSig was completely secure was shown to be false. This come from the sharing of nonces (r,R) byt halting that sharing process part way through. The proof can still be shown to be secure by modifying how these nonces are generated. [Drijvers]

As MuSig is a valid key aggregator, integrating it into a cryptocurrencies signing scheme can give said currency more functionality over traditional multi-signature protocols[5]. First, MuSig reduces the number of bytes when implementing multi-signatures. If three people wanted to sign the same message using the multi-signature methods currently implemented in Bitcoin, it would require three individual signatures on the transaction and thus three times the size of one signature. This means that the size of signatures increase linearly with the number of people signing. With MuSig it would only require one signature, allowing for a constant signature size. One could also aggregate every signature in a block to save space [Poelstra]. Second, aggregating signatures would allow for increased anonymity. Without any form of key aggregation the public addresses for inputs and outputs of transactions are known. If an adversary was paying attention they could keep track of the amount of money being spent from a single address and track it from address to address.[6] Adversaries could also tell if a transaction was a multi-signature transaction. Key aggregation using MuSig prevents people from knowing what public addresses people were spending the money from as the public keys were aggregated together. Finally, a multi-signature using MuSig would look just like a normal signature using Schnorr, preventing observers from knowing whether a transaction was multi-signature [Bogart].

A clear example of how MuSig could improve an existing cryptocurrency is Bitcoin. Anyone viewing the Bitcoin blockchain can instantly tell if multi-signatures were used and what the imputed public addresses were being spent from. MuSig's added privacy would prevent this. Most importantly though, MuSig could save GBs of space if all transaction signatures were replaced by a single Schnorr signature (Figure 1). If each block were to have more space this means more transactions per block, smaller fees, and reducing requirements for Bitcoin network participation [Bogart].



[5] Multi-signature protocols allow a currency can have transactions that require more than one key to authorize a transaction [Wuille].
[6] The real world equivalent of a random person knowing exactly how much money you spent and where you spent it.

Figure 1: Number of GB that could have been saved if MuSig had always been in Bitcoin [Poelstra].

III. Understanding CryptoKernel and cschnorr

Library	Author	Lang	LOC	URL
CryptoKernel	James Lovejoy	C++, lua	14K	https://github.com/mit-dci/CryptoKernel
cschnorr	James Lovejoy	C	1.5K	https://github.com/metalicjames/cschnorr

CryptoKernel is a library written by James Lovejoy that was developed after he noticed that "99% of alt-coins forked Bitcoin core code with minor changes". As of 2018, the Bitcoin core code is nearly ten years old, incredibly layered, and complex. For any novice coming in to create a new coin, making changes to Bitcoin core code required in-depth knowledge of that currency and its system design. Bitcoin was not developed with flexibility in mind. Thus, James decided to develop CryptoKernel with the idea that users could fork CryptoKernel and have flexiblity to set up their coin as they pleased. Per the README.md: "[CryptoKernel] contains modules for key-value storage with JSON, peer-to-peer networking, ECDSA key generation, signing and verifying, big number operations, logging and a blockchain class for handling a Bitcoin-style write-only log. Designed to be object-oriented and easy to use, it provides transaction scripting with Lua 5.3, custom consensus algorithms (e.g Raft, Proof of Work, Authorised Verifier Round-Robin) and custom transaction types" [github.com/mit-dci/CryptoKernel]. Thus, users have the flexibility of choosing their consensus language and a variety of other modules. Our goal with this project was to add MuSig to the CryptoKernel library as another option for users to use in their new coins.

In order to add MuSig to CryptoKernel, James preemptively wrote a library in C called cschnorr that implements Schnorr and MuSig signatures. Our implementation exclusively uses the MuSig functionality of the cschnorr library so that CryptoKernel users can take advantage of MuSig's space savings and added multi-signature privacy.

IV. Modifying CryptoKernel with cschnorr

To implement MuSig in CryptoKernel, needed to statically link the cschnorr library with CryptoKernel. This involves compiling cschnorr and adding it to /usr/lib and a header file. Because James had already implemented a CryptoKernel::Crypto class for ECDSA key generation and signatures, we copied the file structure of crypto.h and crypto.cpp to create schnorr.h and schnorr.cpp respectively in CryptoKernel's src/kernel/ directory.

First, we added "#include <cschnorr/multisig.h>" in schnorr.h. Next, we created a CryptoKernel::Schnorr class that implemented the following:

CryptoKernel::Schnorr::Schnorr()
The constructor function Schnorr, creates the private variables ctx and key of respective types schnorr_context* and musig_key* for the Schnorr class to use.

CryptoKernel::Schnorr::getPublicKey()
Takes the stored public key value, A of type EC_POINT, and returns it as a base64 encoded string.

CryptoKernel::Schnorr::getPrivateKey()
Takes the stored private key value, a of type BIGNUM and returns it as a base64 encoded string.

CryptoKernel::Schnorr::setPublicKey()
Accepts a base64 encoded public key and stores it as a public key value A of type EC_POINT in the musig_pubkey data structure. Returns a bool on success.

CryptoKernel::Schnorr::setPrivateKey()
Accepts a base64 encoded private key and stores it as a private key value a of type BIGNUM in the musig_key data structure. Returns a bool on success.

CryptoKernel::Schnorr::sign()
Wrapper for the musig_sign function, which accepts a message, public key, array of public keys, and returns a signature of the message of type string. For now, only the public key stored in musig_key->pub is used in the array.

CryptoKernel::Schnorr::verify()
Wrapper for the musig_verify function, which accepts a signature, public key, message, and returns a bool upon signature validation.

To test our MuSig functionality we mirrored the tests/CryptoTests.cpp and tests/CryptoTests.h files to create tests/SchnorrTests.cpp,m tests/SchnorrTests.h, and a SchnorrTest SchnorrTest class. The functions the class implemented were as follows:

SchnorrTest::testInit()
Runs the Schnorr::Schnorr() constructor which creates the key and ctx variables and then runs the getStatus() function which always returns true.

SchnorrTest::testKeygen()
Runs Schnorr::getPrivateKey(), Schnorr::getPublicKey(), and then asserts the keys exist.

SchnorrTest::testSignVerify()
Runs Schnorr::sign() on a message and then runs Schnorr::verify() on that text and signature.

SchnorrTest::testPassingKeys()

Creates a tempSchnorr class to test passing public keys between Schnorr classes. A message is signed with tempSchnorr::sign(). Then the original schnorr class has its public key set with schnorr::setPublicKey(tempSchnorr::getPublicKey()). Lastly, schnorr::verify is run on the tempSchnorr signed message.

Please find our full implementation and pull request below:

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

V. Key Learnings

Undertaking this project to implement MuSig in CryptoKernel, we understood multi-signature and Schnorr mathematically. Additionally, we learned of the applications of mutlisignature and its potential size savings. For instance, a husband and wife could create a joint account where only one person would need to sign off on any given transaction. Another application would be a board of directors for a company that requires some quorum of members to sign off on any given expense.

On the engineering side, we all were working with C++ and the OpenSSL libraries for the first time. We found that both have a steep learning curve. We also noticed that CryptoKernel's modularity and built in flexibility allowed us to easily create a MuSig Schnorr class that mirrored the ECDSA Crypto class and respective tests. Finally, we are all excited to have gained experience contributing to and working on an open source cryptocurrency project.

VI. Works Cited

Bogart, Spencer. "Crypto Innovation Spotlight: Schnorr Signatures – Spencer Bogart – Medium." Medium, Augmenting Humanity, 22 Feb. 2018, medium.com/@Bitcom21/crypto-innovation-spotlight-schnorr-signatures-a83748f16a4.

Drijvers, Manu, et al. "Okamoto Beats Schnorr: On the Provable Security of Multi-Signatures." Cryptology EPrint Archive, doi:https://eprint.iacr.org/2018/417.pdf.

Poelstra, Andrew, et al. "Simple Schnorr Multi-Signatures with Applications to Bitcoin." 15 Jan. 2018, eprint.iacr.org/2018/068.pdf.

Schnorr, Claus-Peter. "Efficient Signature Generation by Smart Cards." Journal of Cryptology , vol. 4, no. 3, 1 Jan. 1991, pp. 161–174.

Wuille, Pieter. "Key Aggregation for Schnorr Signatures." Blockstream, 23 Jan. 2018, blockstream.com/2018/01/23/musig-key-aggregation-schnorr-signatures.html.