“Unfairly Linear Signatures”

Adam Gibson

June 6, 2018

Outline

Commitments Key properties (Hash, Pedersen)

Outline

Commitments Key properties (Hash, Pedersen)

ZKPOK ZKPs, Sigma protocol, Schnorr protocol

Commitments Key properties (Hash, Pedersen)

ZKPOK ZKPs, Sigma protocol, Schnorr protocol

Schnorr sigs Multisig, Adaptor sig

Outline

Commitments Key properties (Hash, Pedersen)

ZKPOK ZKPs, Sigma protocol, Schnorr protocol

Schnorr sigs Multisig, Adaptor sig

Security cf with ECDSA

Outline

Commitments Key properties (Hash, Pedersen)

ZKPOK ZKPs, Sigma protocol, Schnorr protocol

Schnorr sigs Multisig, Adaptor sig

Security cf with ECDSA

CoinSwaps

Outline

Commitments Key properties (Hash, Pedersen)

ZKPOK ZKPs, Sigma protocol, Schnorr protocol

Schnorr sigs Multisig, Adaptor sig

Security cf with ECDSA

CoinSwaps

ECDSA multisig With Paillier; adaptor

Commitments - 1

The telephone game: Alice chooses heads or tails, Bob tosses a coin. Alice wins if she guesses right.
🤔

Commitments - 1

The telephone game: Alice chooses heads or tails, Bob tosses a coin. Alice wins if she guesses right.
🤔

Alice hashes her choice: $\mathbb{H}(\text{“tails”} + \text{long random})$ before giving to Bob.

Commitments - 1

The telephone game: Alice chooses heads or tails, Bob tosses a coin. Alice wins if she guesses right. 🤔

Alice hashes her choice: $\mathbb{H}(\text{“tails”} + \text{long random})$ before giving to Bob.

She can't lie - she can't make up any random string that gives the same hash with “heads”.

Commitments - 1

The telephone game: Alice chooses heads or tails, Bob tosses a coin. Alice wins if she guesses right.
🤔

Alice hashes her choice: $\mathbb{H}(\text{“tails”} + \text{long random})$ before giving to Bob.
She can't lie - she can't make up any random string that gives the same hash with “heads”.
Bob calls “heads” (he can't know what Alice chose, so can't get an advantage).

Commitments - 1

The telephone game: Alice chooses heads or tails, Bob tosses a coin. Alice wins if she guesses right. 🤔

Alice hashes her choice: $\mathbb{H}(\text{“tails”} + \text{long random})$ before giving to Bob.

She can't lie - she can't make up any random string that gives the same hash with "heads".

Bob calls "heads" (he can't know what Alice chose, so can't get an advantage).

This way Alice lost in a fair game.

Commitments - 2

Our toy example illustrated the two key properties of a commitment scheme:

- **Binding** - Alice can't go back on her word (hash function properties)

Our toy example illustrated the two key properties of a commitment scheme:

- **Binding** - Alice can't go back on her word (hash function properties)
- **Hiding** - Bob can't know Alice's choice from the commitment (randomized)

Commitments - 2

Our toy example illustrated the two key properties of a commitment scheme:

- **Binding** - Alice can't go back on her word (hash function properties)
- **Hiding** - Bob can't know Alice's choice from the commitment (randomized)

Can get the same effect using Elliptic Curve points, or numbers $\in \mathbb{Z}_N$, instead of hash functions. Add randomness and use hardness of (elliptic curve) discrete log.

Pedersen commitment

$C_x = rH + xG$

x is the message we commit to, r is the randomness, C is the commitment, G is the elliptic curve “generator” point.

Pedersen commitment

$C_x = rH + xG$

x is the message we commit to, r is the randomness, C is the commitment, G is the elliptic curve “generator” point.

But what the heck is H?

Pedersen commitment

$C_x = rH + xG$

x is the message we commit to, r is the randomness, C is the commitment, G is the elliptic curve "generator" point.

But what the heck is H?

"Nothing Up My Sleeve" numbers.

Pedersen commitment

$C_x = rH + xG$

x is the message we commit to, r is the randomness, C is the commitment, G is the elliptic curve "generator" point.

But what the heck is H?

"Nothing Up My Sleeve" numbers.

But what happens to **hiding** and **binding** if something is up my sleeve?

NUMS and Binding

Suppose Alice knows h s.t. $H = hG$, and she committed $C_x = rH + xG$

NUMS and Binding

Suppose Alice knows h s.t. $H = hG$, and she committed $C_x = rH + xG$

Now she wants to cheat and pretend she committed to y not x

NUMS and Binding

Suppose Alice knows h s.t. $H = hG$, and she committed $C_x = rH + xG$

Now she wants to cheat and pretend she committed to y not x

Sets

$$C_x = yG + rH + (x - y)G = yG + \left(r + (x - y)h^{-1}\right) H$$

NUMS and Binding

Suppose Alice knows h s.t. $H = hG$, and she committed $C_x = rH + xG$

Now she wants to cheat and pretend she committed to y not x

Sets

$C_x = yG + rH + (x - y)G = yG + \left(r + (x - y)h^{-1}\right) H$

Pedersen commitments suffer from non-perfect binding as shown; but are **perfectly** hiding for the same reason.

Imperfection

- **Perfect** hiding and **Perfect** binding are incompatible

Imperfection

- **Perfect** hiding and **Perfect** binding are incompatible
- Best we can do? One perfect, one computational

Imperfection

- **Perfect** hiding and **Perfect** binding are incompatible
- Best we can do? One perfect, one computational
- Pedersen are perfect hiding (see previous slide)

Imperfection

- **Perfect** hiding and **Perfect** binding are incompatible
- Best we can do? One perfect, one computational
- Pedersen are perfect hiding (see previous slide)
- If you want perfect *binding*, cannot use compression (function not injective)

Pedersen commitment miscellany

Pedersen commitments are homomorphic - I can give you a commitment to 2 and to 3 and the sum of the commitments is a commitment to 5. Very useful! But not here.

Pedersen commitments are homomorphic - I can give you a commitment to 2 and to 3 and the sum of the commitments is a commitment to 5. Very useful! But not here.

$C_{x_A+x_B} = (r_A + r_B)H + (x_A + x_B)G$

Pedersen commitments are homomorphic - I can give you a commitment to 2 and to 3 and the sum of the commitments is a commitment to 5. Very useful! But not here.

$C_{x_A+x_B} = (r_A + r_B)H + (x_A + x_B)G$

Keeping perfect hiding, you can extend to commitment to a tuple with multiple NUMS basepoints:

Pedersen commitment miscellany

Pedersen commitments are homomorphic - I can give you a commitment to 2 and to 3 and the sum of the commitments is a commitment to 5. Very useful! But not here.

$C_{x_A+x_B} = (r_A + r_B)H + (x_A + x_B)G$

Keeping perfect hiding, you can extend to commitment to a tuple with multiple NUMS basepoints:

$C_{\mathbf{x}} = rH + x_1 G_1 + x_2 G_2 + \ldots x_n G_n$

We can use a commitment scheme as a way to prove knowledge of a secret, **without revealing it**. (Notice in the telephone game, we revealed it at the end).

Zero Knowledge Proof of Knowledge

We can use a commitment scheme as a way to prove knowledge of a secret, **without revealing it**. (Notice in the telephone game, we revealed it at the end).
How?

We can use a commitment scheme as a way to prove knowledge of a secret, **without revealing it**. (Notice in the telephone game, we revealed it at the end).

How?

Commit to a random, then take a challenge, and respond to the challenge in a way that only knower of secret can do. This basic "game" is called a **Sigma Protocol**

We can use a commitment scheme as a way to prove knowledge of a secret, **without revealing it**. (Notice in the telephone game, we revealed it at the end).

How?

Commit to a random, then take a challenge, and respond to the challenge in a way that only knower of secret can do. This basic "game" is called a **Sigma Protocol**

Σ

Sigma protocol

Game setup: Alice has x s.t. $P = xG$, Bob has only P

Sigma protocol

Game setup: Alice has x s.t. $P = xG$, Bob has only P

Choose $k \leftarrow \$$, send $R = kG \quad \Longrightarrow$

Game setup: Alice has x s.t. $P = xG$, Bob has only P

Choose $k \leftarrow \$$, send $R = kG$ $\quad\Longrightarrow$

$\Longleftarrow \quad$ Choose $e \leftarrow \$$

Game setup: Alice has x s.t. $P = xG$, Bob has only P

Choose $k \leftarrow \$$, send $R = kG$ $\implies$

$\impliedby$ Choose $e \leftarrow \$$

Calculate $s = k + e \times x$ $\implies$

Sigma protocol

Game setup: Alice has x s.t. $P = xG$, Bob has only P

Choose $k \leftarrow \$$, send $R = kG \quad \Longrightarrow$

$\Longleftarrow \quad$ Choose $e \leftarrow \$$

Calculate $s = k + e \times x \quad \Longrightarrow$

Game ends with Bob verifying $sG ?= R + eP$

Would it work without the first step?

Would it work without the first step? No, because $e \times x$ doesn't blind x to knower of e. k needed for blinding.

Would it work without the first step? No, because $e \times x$ doesn't blind x to knower of e. k needed for blinding.

Would it work without the second step? (e)

Sigma protocol - reasoning

Would it work without the first step? No, because $e \times x$ doesn't blind x to knower of e. k needed for blinding.

Would it work without the second step? (e) No; "key subtraction attack":

$sG? = R + P = (R' - P) + P = k'G$

Sigma protocol - reasoning

Would it work without the first step? No, because $e \times x$ doesn't blind x to knower of e. k needed for blinding.

Would it work without the second step? (e) No; "key subtraction attack":

$sG? = R + P = (R' - P) + P = k'G$

So: k protects Alice, e protects Bob; but extra interaction step $\rightarrow$ Alice "wins" the game without even opening the commitment!

The generic form is:

The generic form is:

(Prover P): Commitment $\Longrightarrow$

$\Longleftarrow$ Challenge (Verifier V)

(Prover P): Response $\Longrightarrow$

The generic form is:

(Prover P): Commitment $\Longrightarrow$

$\Longleftarrow$ Challenge (Verifier V)

(Prover P): Response $\Longrightarrow$

The description of a "Sigma protocol" in the previous *was* exactly the "Schnorr's Identity Protocol" - a method of proving knowledge of a private key corresponding to a public key P in the discrete log setting. This is all very nice but ... is it really secure?

ZKPOK - Definitions

A Zero Knowledge Proof of Knowledge must have 3 characteristics:

Completeness

If I know the secret, I can provide a valid proof

Soundness

If I don't know the secret, I can't.

Zero-Knowledgeness

My proof reveals nothing other than the **single bit** of information that I know the secret.

Soundness

If the Verifier V cheats, can he extract the secret?

Here “cheats” can only mean: cheats with a Prover P that executes as normal; we create different Provers in different universes to find out.

If the Verifier V cheats, can he extract the secret?

Here "cheats" can only mean: cheats with a Prover P that executes as normal; we create different Provers in different universes to find out.

Yes, you read that right ☺

Soundness

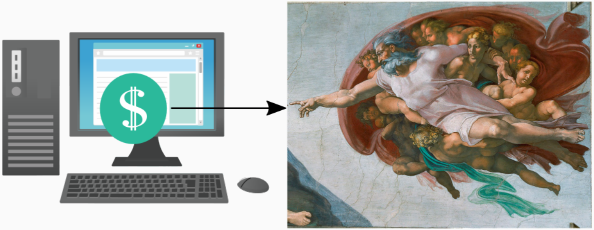
If the Verifier V cheats, can he extract the secret?

Here "cheats" can only mean: cheats with a Prover P that executes as normal; we create different Provers in different universes to find out.

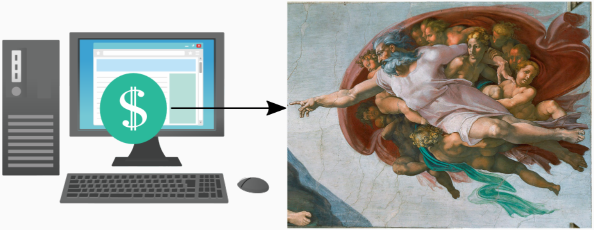
Yes, you read that right ☺

P commits; V branches the Universe and challenges in both; P responds in both.

Soundness - 2

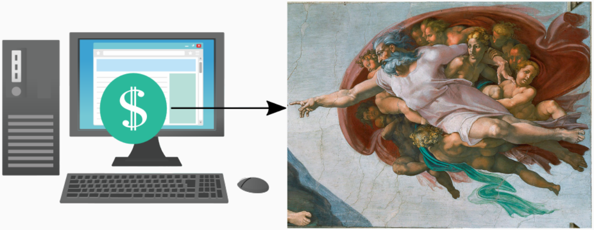


Soundness - 2



$$x = \frac{s_1 - s_2}{e_1 - e_2}$$

Soundness - 2



$$x = \frac{s_1 - s_2}{e_1 - e_2}$$

Works due to k-reuse. The cheating verifier is called an Extractor.

Zero-Knowledgeness (HVZK)

The opposite task: if the Prover P cheats, can he convince the Verifier V? "Simulator": he provides a transcript of the sigma protocol (R, e, s) that verifies correctly, without knowing x.

Zero-Knowledgeness (HVZK)

The opposite task: if the Prover P cheats, can he convince the Verifier V? "Simulator": he provides a transcript of the sigma protocol (R, e, s) that verifies correctly, without knowing x.
This requires getting e from V and then *rewinding*, and cheating by making a new R (see below) that will verify with the given e and a random s.
This requires assuming "Honest Verifier" — the Verifier does not make his challenge choice in any way dependent on the commitment R.

Zero-Knowledgeness (HVZK)

The opposite task: if the Prover P cheats, can he convince the Verifier V? "Simulator": he provides a transcript of the sigma protocol (R, e, s) that verifies correctly, without knowing x.
This requires getting e from V and then *rewinding*, and cheating by making a new R (see below) that will verify with the given e and a random s.
This requires assuming "Honest Verifier" — the Verifier does not make his challenge choice in any way dependent on the commitment R.
$e, s \leftarrow \$, \quad R = sG - eP$

Zero-Knowledgeness (HVZK)

The opposite task: if the Prover P cheats, can he convince the Verifier V? "Simulator": he provides a transcript of the sigma protocol (R, e, s) that verifies correctly, without knowing x.

This requires getting e from V and then *rewinding*, and cheating by making a new R (see below) that will verify with the given e and a random s.

This requires assuming "Honest Verifier" — the Verifier does not make his challenge choice in any way dependent on the commitment R.

$e, s \leftarrow \$, \quad R = sG - eP$

This "proves" that zero information is conveyed, if the distribution of fake transcripts is indistinguishable from the distribution of genuine ones.

Fiat-Shamir transform

- To make the protocol non-interactive, make use of a “random oracle” (the ideal to which a cryptographic hash function aspires)

Fiat-Shamir transform

- To make the protocol non-interactive, make use of a “random oracle” (the ideal to which a cryptographic hash function aspires)
- Hash the transcript up to that point (means R, but ...)

- To make the protocol non-interactive, make use of a “random oracle” (the ideal to which a cryptographic hash function aspires)
- Hash the transcript up to that point (means R, but $\ldots$)
- Schnorr signature on message m therefore becomes: $s = r + H(m|P|R)x$ (we include m to go from Ident. Prot. $\rightarrow$ signature scheme).

- To make the protocol non-interactive, make use of a "random oracle" (the ideal to which a cryptographic hash function aspires)
- Hash the transcript up to that point (means R, but ...)
- Schnorr signature on message m therefore becomes: $s = r + H(m|P|R)x$ (we include m to go from Ident. Prot. $\rightarrow$ signature scheme).
- Hash one-wayness enforces ordering of steps in absence of Verifier enforcement

Fiat-Shamir transform

- To make the protocol non-interactive, make use of a “random oracle” (the ideal to which a cryptographic hash function aspires)
- Hash the transcript up to that point (means R, but ...)
- Schnorr signature on message m therefore becomes: $s = r + H(m|P|R)x$ (we include m to go from Ident. Prot. $\rightarrow$ signature scheme).
- Hash one-wayness enforces ordering of steps in absence of Verifier enforcement
- But - random oracle and zero knowledgeness?

Remember the “Simulator” effectively controls the Verifier’s environment.

ZK with RO

Remember the “Simulator” effectively controls the Verifier’s environment.

So the Simulator gets to cheat and “program” the random oracle (outside Verifier’s env).

ZK with RO

Remember the “Simulator” effectively controls the Verifier’s environment.

So the Simulator gets to cheat and “program” the random oracle (outside Verifier’s env).

Choose $s, e \leftarrow \$$; program RO to output e when input is $sG - eP$; give (R, s) to V.

Reduction to ECDLP

- So far we just assumed that finding x given only $P = xG$ is impossible, but it's "hard".

Reduction to ECDLP

- So far we just assumed that finding x given only $P = xG$ is impossible, but it's "hard".
- "Elliptic Curve Discrete Logarithm Problem"

Reduction to ECDLP

- So far we just assumed that finding x given only $P = xG$ is impossible, but it's "hard".
- "Elliptic Curve Discrete Logarithm Problem"
- It can be shown that: if an attacker can extract the private key from a Schnorr signature, they can also solve the ECDLP

Reduction to ECDLP - 2

Assume we have an adversary "program" that is able to impersonate the holder of x with success probability ϵ:

Reduction to ECDLP - 2

Assume we have an adversary "program" that is able to impersonate the holder of x with success probability ϵ:

Adversary Challenger

Reduction to ECDLP - 2

Assume we have an adversary "program" that is able to impersonate the holder of x with success probability ϵ:

Adversary Challenger

$k \leftarrow \$$, send $R = kG \quad \Longrightarrow$

Reduction to ECDLP - 2

Assume we have an adversary "program" that is able to impersonate the holder of x with success probability ϵ:

Adversary		Challenger
$k \leftarrow \$$, send $R = kG$	$\Longrightarrow$
	$\Longleftarrow e_1 \leftarrow \$$

Assume we have an adversary “program” that is able to impersonate the holder of x with success probability ϵ:

Adversary		Challenger
$k \leftarrow \$$, send $R = kG$	$\Longrightarrow$	
		$\Longleftarrow e_1 \leftarrow \$$
REWIND one step	$\Longrightarrow$	

Assume we have an adversary "program" that is able to impersonate the holder of x with success probability ϵ:

Adversary		Challenger
$k \leftarrow \$$, send $R = kG$	$\Longrightarrow$	
		$\Longleftarrow e_1 \leftarrow \$$
REWIND one step	$\Longrightarrow$	
		$\Longleftarrow e_2 \leftarrow \$$

Reduction to ECDLP - 2

Assume we have an adversary "program" that is able to impersonate the holder of x with success probability ϵ:

Adversary Challenger

$k \leftarrow \$$, send $R = kG \quad \Longrightarrow$

$\Longleftarrow e_1 \leftarrow \$$

REWIND one step $\quad \Longrightarrow$

$\Longleftarrow e_2 \leftarrow \$$

$P(\text{success}) \simeq \epsilon^2$; success $\Longrightarrow$ extract discrete log x.

Digital signature security

- Previous slide(s) only discuss security of scheme against a "total break" - that is to say, the exposure of the private key from the signature.

Digital signature security

- Previous slide(s) only discuss security of scheme against a "total break" - that is to say, the exposure of the private key from the signature.
- But there is also *security against forgery*; in particular we'd like **security against existential forgery under chosen message attack**

Digital signature security

- Previous slide(s) only discuss security of scheme against a "total break" - that is to say, the exposure of the private key from the signature.
- But there is also *security against forgery*; in particular we'd like **security against existential forgery under chosen message attack**
- In English - no matter how many signatures you get me to output for a bunch of messages you maliciously choose, you can't create your own *new* signature on a new message without my key.

ECDSA's weaknesses

No strong security:

ECDSA's weaknesses

No strong security:

$$V: \quad s^{-1}\left(H(m)G + rP\right)\big|_x \; ? = r$$

ECDSA's weaknesses

No strong security:

$V: \quad s^{-1}(H(m)G + rP)|_x \; ? = r$

r is x-coord; there are two points $(Q, -Q)$ with same x-coordinate. So $(r, -s)$ verifies if (r, s) does. This is "intrinsic malleability" (see BIP66).

No strong security:

$V: \quad s^{-1}(H(m)G + rP)|_x \; ? = r$

r is x-coord; there are two points $(Q, -Q)$ with same x-coordinate. So $(r, -s)$ verifies if (r, s) does.

This is "intrinsic malleability" (see BIP66).

Security reduction (see previous) to ECDLP.

No strong security:

$V: \quad s^{-1}(H(m)G + rP)|_x \,?= r$

r is x-coord; there are two points $(Q, -Q)$ with same x-coordinate. So $(r, -s)$ verifies if (r, s) does.

This is "intrinsic malleability" (see BIP66).

Security reduction (see previous) to ECDLP.

Dodgy at best? See e.g. Vaudenay "The Security of DSA and ECDSA".

No strong security:

$V: \quad s^{-1}(H(m)G + rP)|_x \; ? = r$

r is x-coord; there are two points $(Q, -Q)$ with same x-coordinate. So $(r, -s)$ verifies if (r, s) does. This is "intrinsic malleability" (see BIP66).

Security reduction (see previous) to ECDLP. Dodgy at best? See e.g. Vaudenay "The Security of DSA and ECDSA".

No linearity (especially over nonces due to funky use of x-coordinate).

- The Schnorr signature $s = k + ex$ is linear in both the nonce (k) and the key (x)

Leveraging linearity

- The Schnorr signature $s = k + ex$ is linear in both the nonce (k) and the key (x)
- Let's add signatures on a message m to make a joint signature (I AND you sign):

Leveraging linearity

- The Schnorr signature $s = k + ex$ is linear in both the nonce (k) and the key (x)
- Let's add signatures on a message m to make a joint signature (I AND you sign):
- $s_{AB} = s_A + s_B = k_A + k_B + e(x_A + x_B)$

- The Schnorr signature $s = k + ex$ is linear in both the nonce (k) and the key (x)
- Let's add signatures on a message m to make a joint signature (I AND you sign):
- $s_{AB} = s_A + s_B = k_A + k_B + e(x_A + x_B)$
- e is shared; must commit to both nonces like $e = H(R_A + R_B | P_A + P_B | m)$

Leveraging linearity

- The Schnorr signature $s = k + ex$ is linear in both the nonce (k) and the key (x)
- Let's add signatures on a message m to make a joint signature (I AND you sign):
- $s_{AB} = s_A + s_B = k_A + k_B + e(x_A + x_B)$
- e is shared; must commit to both nonces like $e = H(R_A + R_B | P_A + P_B | m)$
- Insecure! But manner of insecurity requires thinking about *interaction*

If keys P produced ephemerally, open to direct key subtraction attack; last player can delete everyone else's key; disaster for multisig:

If keys P produced ephemerally, open to direct key subtraction attack; last player can delete everyone else's key; disaster for multisig:
$P_{\mathrm{attack}} = P^* - \Sigma P_i$ where attacker knows privkey of P^*.

If keys P produced ephemerally, open to direct key subtraction attack; last player can delete everyone else's key; disaster for multisig:
$P_{\mathrm{attack}} = P^* - \Sigma P_i$ where attacker knows privkey of P^*.
"Derandomisation": Constructions like
$sG = R + \mathbb{H}(P_{\mathrm{agg}}|R|m)P_{\mathrm{agg}}$

Aggregation schemes

If keys P produced ephemerally, open to direct key subtraction attack; last player can delete everyone else's key; disaster for multisig:
$P_{\mathrm{attack}} = P^{*} - \Sigma P_i$ where attacker knows privkey of P^{*}.
"Derandomisation": Constructions like
$sG = R + \mathbb{H}(P_{\mathrm{agg}}|R|m)P_{\mathrm{agg}}$
Maintain ability to validate using *only the aggregate key* while being safe from key subtraction.

Aggregation schemes

If keys P produced ephemerally, open to direct key subtraction attack; last player can delete everyone else's key; disaster for multisig:
$P_{\mathrm{attack}} = P^{*} - \Sigma P_i$ where attacker knows privkey of P^{*}.
"Derandomisation": Constructions like
$sG = R + \mathbb{H}(P_{\mathrm{agg}}|R|m)P_{\mathrm{agg}}$
Maintain ability to validate using *only the aggregate key* while being safe from key subtraction.
See Musig paper https://eprint.iacr.org/2018/068 for details.

- Bellare-Neven (cleaner security proof but requires all keys for verification)

Aggregation schemes - 2

- Bellare-Neven (cleaner security proof but requires all keys for verification)
- Musig requires only one aggregated key for verification

Aggregation schemes - 2

- Bellare-Neven (cleaner security proof but requires all keys for verification)
- Musig requires only one aggregated key for verification
- Per-input aggregation, per-transaction aggregation, per-block aggregation(?)

Aggregation schemes - 2

- Bellare-Neven (cleaner security proof but requires all keys for verification)
- Musig requires only one aggregated key for verification
- Per-input aggregation, per-transaction aggregation, per-block aggregation(?)
- https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-May/015951.html

Aggregation schemes - 3

Good summary of key facts at https://blockstream.com/2018/01/23/musig-key-aggregation-schnorr-signatures.html

- Break history of coins using atomicity of: spend a coin $\leftrightarrow$ reveal a secret

CoinSwap

- Break history of coins using atomicity of: spend a coin $\leftrightarrow$ reveal a secret
- “Atomic Cross Chain Swap” (see HTLC) not useful for privacy

CoinSwap

- Break history of coins using atomicity of: spend a coin $\leftrightarrow$ reveal a secret
- "Atomic Cross Chain Swap" (see HTLC) not useful for privacy
- Maxwell 2013 CoinSwap (updated) but slow and interactive

CoinSwap

- Break history of coins using atomicity of: spend a coin $\leftrightarrow$ reveal a secret
- "Atomic Cross Chain Swap" (see HTLC) not useful for privacy
- Maxwell 2013 CoinSwap (updated) but slow and interactive
- Schnorr + scriptless scripts (Poelstra); better overall features

CoinSwap in 2017

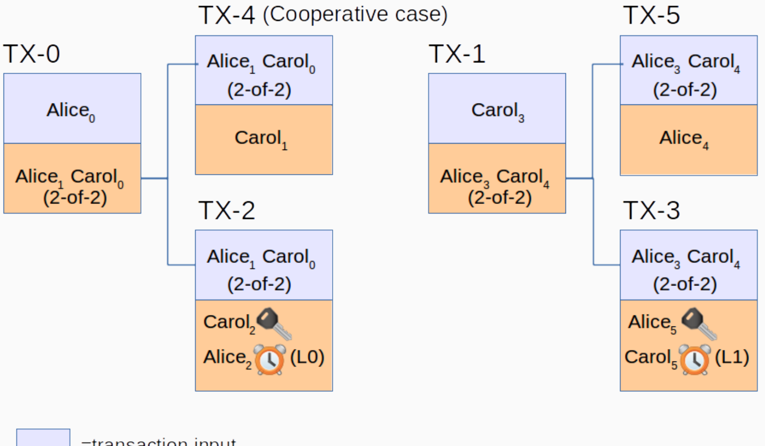
- With segwit; without Schnorr; without taproot

CoinSwap in 2017

- With segwit; without Schnorr; without taproot
- “CoinSwapCS” (proof of concept):

CoinSwap in 2017

- With segwit; without Schnorr; without taproot
- “CoinSwapCS” (proof of concept):



- Embed a secret in the nonce; from $s = k + H(m|R|P)x$ to $s = k + t + H(m|R+T|P)x$

Adaptor signatures - 1

- Embed a secret in the nonce; from $s = k + H(m|R|P)x$ to $s = k + t + H(m|R+T|P)x$
- Share T as "hash" of secret

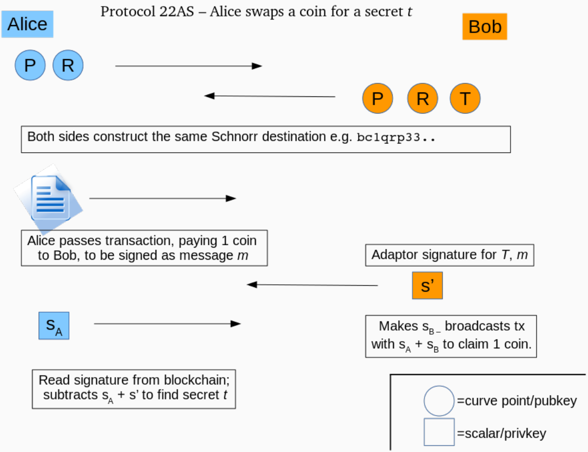
Adaptor signatures - 1

- Embed a secret in the nonce; from $s = k + H(m|R|P)x$ to $s = k + t + H(m|R + T|P)x$
- Share T as “hash” of secret
- Give $s' = k + H(m|R + T|P)x$ as incomplete **adaptor signature**

Adaptor signatures - 1

- Embed a secret in the nonce; from $s = k + H(m|R|P)x$ to $s = k + t + H(m|R+T|P)x$
- Share T as "hash" of secret
- Give $s' = k + H(m|R+T|P)x$ as incomplete **adaptor signature**
- Verifiable; you know it'll be a valid sig if you get preimage of T

A new way to swap a coin for a secret:



Adaptor sigs - 3

1. Prepare: swap keys (Musig etc.), swap txids, swap backouts

Adaptor sigs - 3

1. Prepare: swap keys (Musig etc.), swap txids, swap backouts
2. Pay in (locktime asymmetry as per earlier CoinSwap), confirm

Adaptor sigs - 3

1. Prepare: swap keys (Musig etc.), swap txids, swap backouts
2. Pay in (locktime asymmetry as per earlier CoinSwap), confirm
3. Do 22AS as above; swap Rs, Alice has T

Adaptor sigs - 3

1. Prepare: swap keys (Musig etc.), swap txids, swap backouts
2. Pay in (locktime asymmetry as per earlier CoinSwap), confirm
3. Do 22AS as above; swap Rs, Alice has T
4. There are *2* adaptor sigs with same T

5. When Alice claims her coins, the sig reveals t and Bob completes

Adaptor sigs - 4

5. When Alice claims her coins, the sig reveals t and Bob completes
6. More details at https://joinmarket.me/blog/blog/flipping-the-scriptless-script-on-schnorr/

Adaptor sigs - 4

5. When Alice claims her coins, the sig reveals t and Bob completes
6. More details at https://joinmarket.me/blog/blog/flipping-the-scriptless-script-on-schnorr/
7. Huge advantage in deniability: any sig could be adaptor; Schnorr musig is 1 key

Adaptor sig in ECDSA

Recent work Malavolta et al

https://eprint.iacr.org/2018/472

Adaptor sig in ECDSA

Recent work Malavolta et al

https://eprint.iacr.org/2018/472

Aggregated signature in ECDSA

Recent work Malavolta et al

https://eprint.iacr.org/2018/472

Aggregated signature in ECDSA

Use Paillier's additive homomorphism

$(E(A) + E(B) = E(A + B))$

Adaptor sig in ECDSA

Recent work Malavolta et al
https://eprint.iacr.org/2018/472
Aggregated signature in ECDSA
Use Paillier's additive homomorphism
$(E(A) + E(B) = E(A + B))$
2-party computation $\rightarrow$ single ECDSA signature 2 of 2

Adaptor sig in ECDSA

Recent work Malavolta et al
https://eprint.iacr.org/2018/472
Aggregated signature in ECDSA
Use Paillier's additive homomorphism
$(E(A) + E(B) = E(A + B))$
2-party computation $\rightarrow$ single ECDSA signature 2 of 2
We can recreate adaptor signatures in the above model

Adaptor sig in ECDSA - 2

Original note at https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-April/001221.html

Original note at https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-April/001221.html

1. Share keys, nonce points P, R, Alice sends encrypted privkey $E(x_A)$

Adaptor sig in ECDSA - 2

Original note at https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-April/001221.html

1. Share keys, nonce points P, R, Alice sends encrypted privkey $E(x_A)$
2. ECDH shared nonce $R = k_A k_B G$; x-coord r

Adaptor sig in ECDSA - 2

Original note at https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-April/001221.html

1. Share keys, nonce points P, R, Alice sends encrypted privkey $E(x_A)$
2. ECDH shared nonce $R = k_A k_B G$; x-coord r
3. Bob: $E(k_B^{-1} H)$, $x_B r k_B^{-1} E(x_A)$, add under enc

Original note at https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-April/001221.html

1. Share keys, nonce points P, R, Alice sends encrypted privkey $E(x_A)$
2. ECDH shared nonce $R = k_A k_B G$; x-coord r
3. Bob: $E(k_B^{-1} H)$, $x_B r k_B^{-1} E(x_A)$, add under enc
4. Alice: $k_A^{-1}\left(k_B^{-1}\left(H + x_A x_B r\right)\right) = s$

Adaptor sigs in ECDSA - 3

Previous slide - interactive 2 of 2 multisig for ECDSA with 1 published key – cool!

Adaptor sigs in ECDSA - 3

Previous slide - interactive 2 of 2 multisig for ECDSA with 1 published key – cool!

Although it did miss tech. details - don't do that!

Adaptor sigs in ECDSA - 3

Previous slide - interactive 2 of 2 multisig for ECDSA with 1 published key – cool!

Although it did miss tech. details - don't do that!

How to add in adaptor (T?)

Adaptor sigs in ECDSA - 3

Previous slide - interactive 2 of 2 multisig for ECDSA with 1 published key – cool!

Although it did miss tech. details - don't do that!

How to add in adaptor (T?)

Bob tweaks his $R_B = k_B G$ to $R_B^* = k_B t G$

Adaptor sigs in ECDSA - 3

Previous slide - interactive 2 of 2 multisig for ECDSA with 1 published key – cool!

Although it did miss tech. details - don't do that!

How to add in adaptor (T?)

Bob tweaks his $R_B = k_B G$ to $R_B^* = k_B t G$

Needs to send PoDLE

Adaptor sigs in ECDSA - 4

Next, sends encryption as before with k_B, so
$E(adaptor) = E(s')$

Adaptor sigs in ECDSA - 4

Next, sends encryption as before with k_B, so
$E(adaptor) = E(s')$
Alice decrypts and verifies s'

Adaptor sigs in ECDSA - 4

Next, sends encryption as before with k_B, so

$E(adaptor) = E(s')$

Alice decrypts and verifies s'

Alice returns $s'' = s' \times k_A^{-1}$

Next, sends encryption as before with k_B, so

$E(adaptor) = E(s')$

Alice decrypts and verifies s'

Alice returns $s'' = s' \times k_A^{-1}$

Bob publishes (r, s) where $s = s'' \times t^{-1}$

Next, sends encryption as before with k_B, so $E(adaptor) = E(s')$

Alice decrypts and verifies s'

Alice returns $s'' = s' \times k_A^{-1}$

Bob publishes (r, s) where $s = s'' \times t^{-1}$

Alice gets $t = s'' \times s^{-1}$ from on-chain sig

Other interesting things

- Ring signatures - $s_i = k_i + \mathbb{H}(R_{i-1}| \ldots)x_i$
- AND and ORs of Sigma Protocols
- General ZKP systems - zkSNARKs, Bulletproofs, others
- Blinded Schnorr signatures

Thank you

Contact info:

waxwing (freenode IRC, reddit)

@waxwing__ (twitter)

https://github.com/AdamISZ

A blog: https://joinmarket.me/blog/blog (email in /about-me)

gpg: 4668 9728 A9F6 4B39 1FA8 71B7 B3AE 09F1 E9A3 197A