mas.s62
lecture 18
confidential transactions

2018-04-18

Tadge Dryja

today

hiding output amounts

commitments

Pedersen commitments

range proofs

confidential transactions

coinjoin

last class, looked at combined transactions

one issue: output amounts reveal who's sending what where

coinjoin tx

amounts reveal connections...

input 0 user A signature 10 coins	output 0 address C 2 coins
input 1 user B signature 2 coins	output 1 address D 10 coins

output amounts

wouldn't it be great if we could hide the amounts?

hidden amount tx

no longer linkable

input 0 user A signature 10 coins	output 0 address C _ coins
input 1 user B signature 2 coins	output 1 address D _ coins

no output amounts

So that solves the coinjoin issue

Also, really useful!

If people can see how many coins you have, they could:

charge you more / try to rob you etc...

amount privacy

we can try to improve privacy by making it hard to link outputs together, or hard to link people and outputs

Hiding amounts makes outputs very hard to distinguish

amount privacy

OK I'm sold! How do we do it?

First, what are we even trying to do?

What are we hiding, and from whom?

hidden amount tx

long term state

input 0 user A signature _ coins	output 0 address C _ coins
input 1 user B signature _ coins	output 1 address D _ coins

amount privacy

People receiving payments should probably know how much they're receiving. And how much they have.

People sending should also know how much they're sending.

hidden amount tx

only sender / receiver know

network view:

input 0 user A signature _ coins	output 0 address C _ coins
input 1 user B signature _ coins	output 1 address D _ coins

hidden amount tx

only sender / receiver know

sender view:

input 0 user A signature 2 coins	output 0 address C 7 coins
input 1 user B signature 7 coins	output 1 address D 2 coins

hidden amount tx

only sender / receiver know

receiver view:

input 0 user A signature _ coins	output 0 address C _ coins
input 1 user B signature _ coins	output 1 address D 2 coins

disclosure

May want to hide per-output.

Some kind of encryption? Hide the amounts so that only people with the right private key can see the numbers?

But then...

hidden amount tx

only sender / receiver know

participant view:

input 0 user A signature 2 coins	output 0 address C 70 coins
input 1 user B signature 7 coins	output 1 address D 2000 coins

hidden amount tx

only sender / receiver know

network view:

input 0 user A signature _ coins	output 0 address C _ coins
input 1 user B signature _ coins	output 1 address D _ coins

disclosure

if the network sees nothing, easy to create coins.

If those coins are later used, you can't tell they were made up.

Unless you trace all encrypted parent transactions back to before the encryption.

disclosure

doesn't work: either you allow people to create coins, or you reveal ~all previous amounts to eventually everyone.

disclosure

doesn't work: either you allow people to create coins, or you reveal ~all previous amounts to eventually everyone.

Need to prevent coin creation while still keeping amounts secret...

hidden amount tx

network view:

input 0 user A signature w coins	output 0 address C y coins
input 1 user B signature x coins	output 1 address D z coins

hidden amount tx

network view:

proof: w+x = y+z

input 0 user A signature w coins	output 0 address C y coins
input 1 user B signature x coins	output 1 address D z coins

commitments

how will we do this? commitments.

simplest form:

commitments

how will we do this? commitments.

simplest form:

commit(value) -> c

commitments

how will we do this? commitments.

simplest form:

commit(value) -> c

reveal value

commitments

how will we do this? commitments.

simplest form:

commit(value) -> c

reveal value

verify(c, value) -> bool

commitments

a hash function is a commitment

hash(5) -> 68fde0b7

commit to 68fde0b7

commitments

a hash function is a commitment

hash(5) -> 68fde0b7

commit to 68fde0b7

reveal 5

commitments

a hash function is a commitment

hash(5) -> 68fde0b7

commit to 68fde0b7

reveal 5

verify: hash(5) == 68fde0b7? True

commitments

This is binding (computationally)

hash(5) -> 68fde0b7

I can't find another number that will get me to 68fde0b7. (Maybe if I try 2^{256} of them.)

commitments

problem: it's binding, but not hiding

Verified can easily guess and check committed value

```
hash(i) -> 68fde0b7

for i = 0; i < 0xffffffff; i++ {}
```

blinded commitments

solution: add a blinding factor

r = b8bc7579

hash(5, r) = 4dd8fa60

to reveal, reveal both 5 and r

note: need to tell people the order of v, r so that you can't claim 5 was your blinding factor

hash commitments

useful, but we need more

want to be able to prove things about commitments

need homomorphic commitments

homomorphic commitments

we want:

```
commit(x) -> a
commit(y) -> b


reveal z = x + y
verify(z, a + b) -> true
```

homomorphic commitments

This would be very useful: can reveal a sum without revealing the constituent parts

How can we build this?

homomorphic commitments

This would be very useful: can reveal a sum without revealing the constituent parts

How can we build this?

Gee...

homomorphic commitments

This would be very useful: can reveal a sum without revealing the constituent parts

How can we build this?

Gee...

G

commitments on a curve

want: commit x, y

reveal $z = x+y$

$X = xG$, $Y = yG$

commitments on a curve

$X = xG$

is this binding?

commitments on a curve

$X = xG$

is this binding?

can I come up with a different x that gets me to X?

commitments on a curve

$X = xG$

is this binding?

can I come up with a different x that gets me to X?

I can't; DLP. This is binding, but...

commitments on a curve

X = 5*G

not blinded, easy to guess 5.

try X = (5+r)G; reveal 5, r

commitments on a curve

X = 5*G

not blinded, easy to guess 5.

try X = (5+r)G; reveal 5, r

why won't this work?

commitments on a curve

X = (5+r)G; reveal 5, r

not binding; find r' = (5+r) - 6

6+r' = 5+r so X is the same

reveal 6, r'

commitments on a curve

X = (5+r)G; reveal 5, r

not binding; find r' = (5+r) - 6

6+r' = 5+r so X is the same

reveal 6, r'

use hash(5, r)G ..?
but then no longer homomorphic...

Pedersen commitments

introducing G's (fraternal) twin, H

H is another generator point distinct from G

Nobody knows n such that nG = H

(pick a random point on the curve)

Pedersen commitments

$$X = rG + vH$$

where:

v is the value committed

r is a blinding factor

Pedersen commitments

$X = rG + vH$

binding

I can't come up with another r, v that gets me to X

(unless I know G/H)

Pedersen commitments

X = rG + vH

hiding

guess that v=5, and you might be right. But 138cbec078*H is also in X so good luck.

Pedersen commitments

$X = r_1G + v_1H$ $\quad$ $Y = r_2G + v_2H$

homomorphic

I want to prove $z = v_1+v_2$ without revealing them individually

Pedersen commitments

$X = r_1G + v_1H$ $\qquad$ $Y = r_2G + v_2H$

$Z = X + Y = (r_1+r_2)G + (v_1+v_2)H$

reveal $r, v = r_1+r_2, v_1+v_2$

Verifier can check if $rG + vH = Z$

Pedersen commitments

$X = r_1G + v_1H \qquad Y = r_2G + v_2H$

$Z = X + Y = (r_1+r_2)G + (v_1+v_2)H$

reveal $r, v = (r_1+r_2), (v_1+v_2)$

binding, hiding, homomorphic

great! We can prove sums

Pedersen amount tx

network view:

proof: W+X = Y+Z

input 0 user A signature $W = r_1G + wH$ coins	output 0 address C $Y = r_3G + yH$ coins
input 1 user B signature $X = r_2G + xH$ coins	output 1 address D $Z = r_4G + zH$ coins

Pedersen amount tx

receiver view:

learn own v, r

input 0 user A signature $W = r_1G + wH$ coins	output 0 address C $Y = r_3G + yH$ coins
input 1 user B signature $X = r_2G + xH$ coins	output 1 address D $Z = r_4G + 2H$ coins

Pedersen amount tx

when making outputs, make all r's but the last random; compute last r

input 0 user A signature $W = r_1G + wH$ coins	output 0 address C $Y = r_3G + yH$ coins
input 1 user B signature $X = r_2G + xH$ coins	output 1 address D $Z = r_4G + zH$ coins

Pedersen amount tx

$$r_1 + r_2 = r_3 + r_4$$

input 0 user A signature $W = r_1G + wH$ coins	output 0 address C $Y = r_3G + yH$ coins
input 1 user B signature $X = r_2G + xH$ coins	output 1 address D $Z = r_4G + zH$ coins

Pedersen amount tx

can prove w+x = y+z

input 0 user A signature $W = r_1G + wH$ coins	output 0 address C $Y = r_3G + yH$ coins
input 1 user B signature $X = r_2G + xH$ coins	output 1 address D $Z = r_4G + zH$ coins

Pedersen txs

can verify that inputs = outputs

just add up all the points on both sides and make sure they're equal

reveal output r, v to person receiving the coins

don't forget r!

Pedersen txs

can make invalid outputs which are just points with no known r,v ... but nobody will accept them

Pedersen amount tx

can prove w+x = y+z

input 0 user A signature $W = wG + r_1H$ coins	output 0 address C $Y = yG + r_3H$ coins
input 1 user B signature $X = xG + r_2H$ coins	output 1 address D $Z = W+X - Y$

Pedersen txs

But there's a big problem

Or maybe the opposite of a big problem...

Pedersen txs

But there's a big problem

Or maybe the opposite of a big problem...

no, not a small problem...

Pedersen txs

But there's a big problem

Or maybe the opposite of a big problem...

no, not a <u>small</u> problem...

a big, but negative problem

Pedersen amount tx

can prove w+x = y+z

input 0 user A signature $W = r_1G + 2H$ coins	output 0 address C $Y = r_3G + -99H$ coins
input 1 user B signature $X = r_2G + 7H$ coins	output 1 address D $Z = r_4G + 108H$ coins

Pedersen amount tx

2+7 = -99 + 108

that negative output will be hidden

input 0 user A signature $W = r_1G + 2H$ coins	output 0 address C $Y = r_3G + -99H$ coins
input 1 user B signature $X = r_2G + 7H$ coins	output 1 address D $Z = r_4G + 108H$ coins

confidential txs

we need more than the proof the sums are equal

we also need a proof that they're non-negative

How can we prove something about the number itself without revealing it?

confidential txs

can we sign with one of the points?

$s = k - h(kG, m)a$

$X = r_2G + 7H$

$x = r_2 + 7$? no...

we know the private scalars, but there's H, not G, for the v

confidential txs

$s = k - h(kG, m)a$

what if v is 0? Then

$X = r_2G + 0H$

$x = r_2$

now we can sign a message with key X

confidential txs

proof of zero-value; sign own key

$X = rG + 0H$

$s = k - h(kG, X)r$

$sG = kG - h(kG, X)X$

works, and can't sign if H != 0

confidential txs

proof of v = 1; sign own key

$X = rG + 1H \quad X' = X - H$

$s = k - h(kG, X)r$

$sG = kG - h(kG, X)X'$

works, and can't sign if H != 1

confidential txs

we can prove v is 0. Or 1. Or anything. Without revealing r

But wait. We just revealed v, so what's the point?

ring signatures

introducing: ring signatures

similar to normal signatures, but there is a set of pubkeys

I sign a message with one of the pubkeys, but I don't tell you which

ring signatures

```
keygen() -> priv, pub

sign(msg, priv, []pub) -> sig

verify(sig, msg, []pub) -> bool
```

can verify it's from a key in []pub, but not which

ring signatures

if I can sign with X to prove v=0

or sign with X' (X - H) to prove v=1

A ring signature on (X, X') would prove that v is either 0 or 1, but not which.

ring signatures

make a ring signature from a million public keys, where $Pub_n = Pub_{n-1} - H$

Proves v = 0 ... 999,999

ring signatures

more efficient: ring signature for each bit.

X_0 is 0 or 1

X_1 is 0 or 2

X_2 is 0 or 4

etc...

confidential transactions

A signature per bit, but if your values are not too big, it works.

But a couple KB per output. Used to be 8 bytes.

And not really compatible with bitcoin; a tricky fork.

confidential transactions

private, unlinkable amounts

input 0 user A signature W coins	output 0 address C Y coins
input 1 user B signature X coins	output 1 address D Z coins

confidential transactions

Even more: bulletproofs, more efficient range proofs

Borromean ring signatures

MimbleWimble - when all txs are like this, txs can be cancelled out