accumulators
efficient set algorithms

Tadge Dryja

2019-09-09

edge / dev++ / scaling

Tel Aviv University

intro

Hi I'm Tadge Dryja

worked on / authored lightning network & discreet log contracts

today: accumulators, utreexo

For questions, interrupt? or come bug me after

what's this about "accumulators"?

Cryptography has lots of fun tools, not just encryption, or signing.

Cryptography to increase the scalability of Bitcoin? Sounds good!

As with all engineering, lots of trade-offs

goal

We want to keep track of, like, billions of things.

And it should only take up 1KB or so.

Maybe sounds crazy? But doable.

hash functions

you may have heard of them!

h(x) = y; y looks unrelated to x

"one way": given y, can't find x

"collision free", can't find x, x'
such that x != x', h(x) = h(x')

file digest

an important property is that x and y aren't the same length. x can be much longer than y.

This is useful: take a big file, and only save the hash of it. Can verify the file was unmodified later.

set digest

what if we want more flexibility? Like a set of n files, instead of just 1 file?

We could just save the hashes of each file, which works OK, but takes up O(n) space. (32 bytes * n)

set digest

We want to store less than O(n); maybe O(log(n)) or O(1).

Also, it would be cool if we could add and remove from these sets later on, and prove membership or non-membership.

accumulators

This set digest is called an accumulator. The basic functions are

```
gen() -> acc

add(acc, element) -> acc, proof

verify(acc, element, proof) -> bool
```

accumulators

Merkle tree

can classify it as a "static" accumulator

You can add many elements, but only once. Can prove inclusion, but given the root, you can't add more elements

accumulators

accumulator terms

"Dynamic": There's a Remove() function in addition to Add(), which does what you'd think

"Universal": There's a Prove() and Verify() for elements not in the set.

RSA accumulator

RSA-based accumulators...
Wait, RSA? Tough to cover in a few minutes, but a quick refresher!

The original digital signature algorithm. Also does encryption.
Powerful, but a bit of a minefield.

Implement with caution!

RSA

pubkey: make 2 prime numbers, p, q.

n = pq. n is the public key.

private key: phi = (p-1)(q-1)

e = 3 or something (65537 is safer I guess)

m = some message (hash)

d = computed from phi such that

$(m^e)^d \equiv (m^d)^e \equiv m \mod n$

RSA

encrypt: $c \equiv m^e \mod n$

decrypt: $m \equiv c^d \mod n$

sign: $s \equiv m^d \mod n$

verify: $m \equiv s^e \mod n$

cool!

RSA accumulating

for the accumulator $n = pq$, but there is no d and no e.

Start with $v = 3$ or some other starter prime.

Every element x in the set must be prime, so need to hash onto primes

RSA accumulating

Add(x, v): $v' \equiv v^{x} \bmod n$

keep doing that for x_1, x_2, x_3 ...

Prove(x, v): an inclusion proof p is the accumulator v with every element *except* x added

Verify(x, p, v): $p^{x} \equiv ?\ v \bmod n$

RSA accumulator properties

constant size: v, p, x -- everything's the same length as n, regardless of number of elements

Can prove many inclusions at once, again same size

RSA accumulator issues

p, q are trusted setup. Anyone who knows p, q can create false proofs

while proofs are aggregatable, proof updates are not

RSA proof updates

many proofs $p_1=v^{X-x1}$, $p_2=v^{X-x2}$, $p_3=v^{X-x3}$...

add single element x8

must to compute p_1^{x8}, p_2^{x8}, p_3^{x8}

adding multiple elements x8, x9

must compute $(p_1^{x8})^{x9}$, $(p_2^{x8})^{x9}$, $(p_3^{x8})^{x9}$

What do we want to accumulate?

How about accumulating some bitcoins?

If proof updates are few / infrequent, then we're OK.
But if we're looking at the UTXO set, proof updates happen every 10 minutes.

What do we want to accumulate?

If we wanted to prove every bitcoin:

60M utxos
~6K updates every 600 sec (10/sec)

For individual proofs, 60M * 10 =
600M exponentiations / sec

@1ms per op, need 600K cpu cores!

scalability

Do we need to keep proofs for every possible transaction?

Maybe not; if wallets keep track of their own UTXOs and proofs, it's much more reasonable

scalability

Lightly used wallet: 10 utxos

6K updates per block * 10 txos =
60K exponentiations per block

@1ms each, that's 1 minute of CPU
time per block
Doable, but still lots of work

RSA setup vs class groups

If we can deal with the CPU load, we still have the trusted setup

Possible solution: Class groups of binary quadratic forms

Group of unknown order with no trusted setup. Possibly slower, novel, but could help!

What about those merkle trees?

What about hash based, tree-like accumulators?

Some previous research, but not dynamic without a manager

Next talk: novel hash-based dynamic accumulator optimized for bitcoin: utreexo