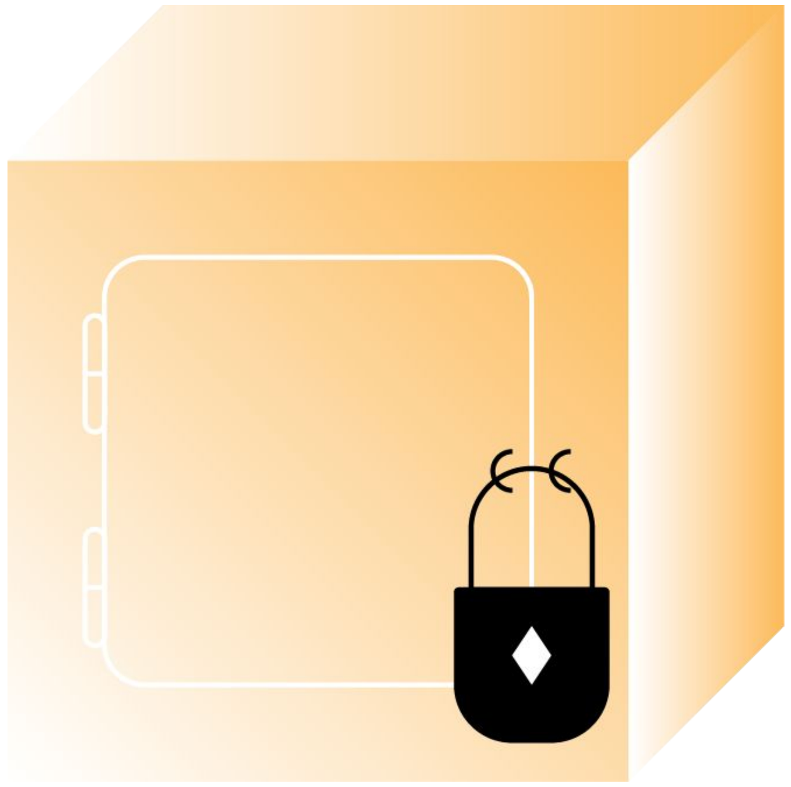
Centrifuge



zkNFT: our adventures with zkSNARKs

2019-10-29, ZKProof Community Event, Amsterdam

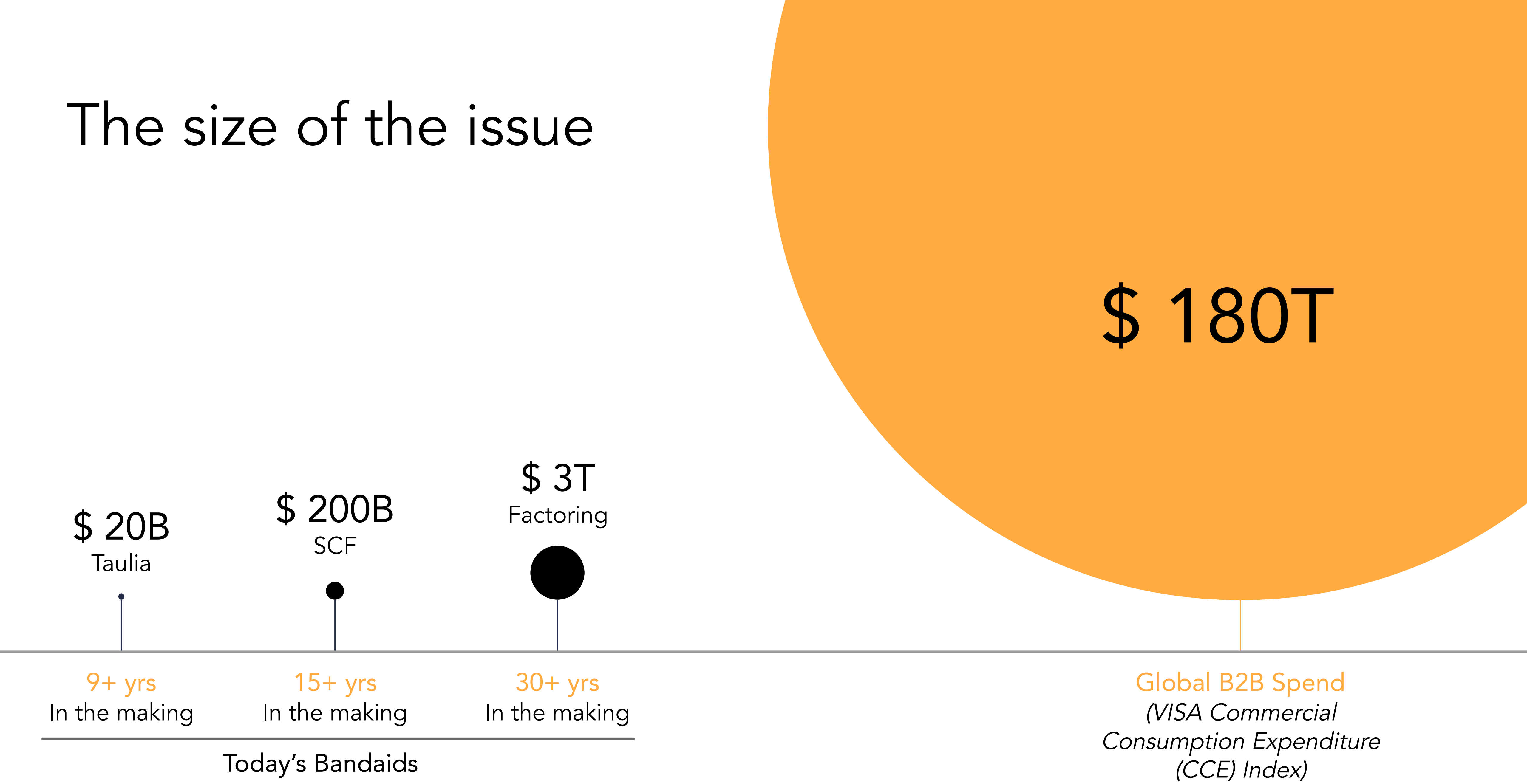
Lucas Vogelsang, @lucasvo // @centrifuge

We ❤️ Supply Chain Finance

Years of working in this space

taulia
Last quarter we delivered a record-breaking $4.5 billion in early payments to suppliers

The size of the issue



LIBOR +50bps

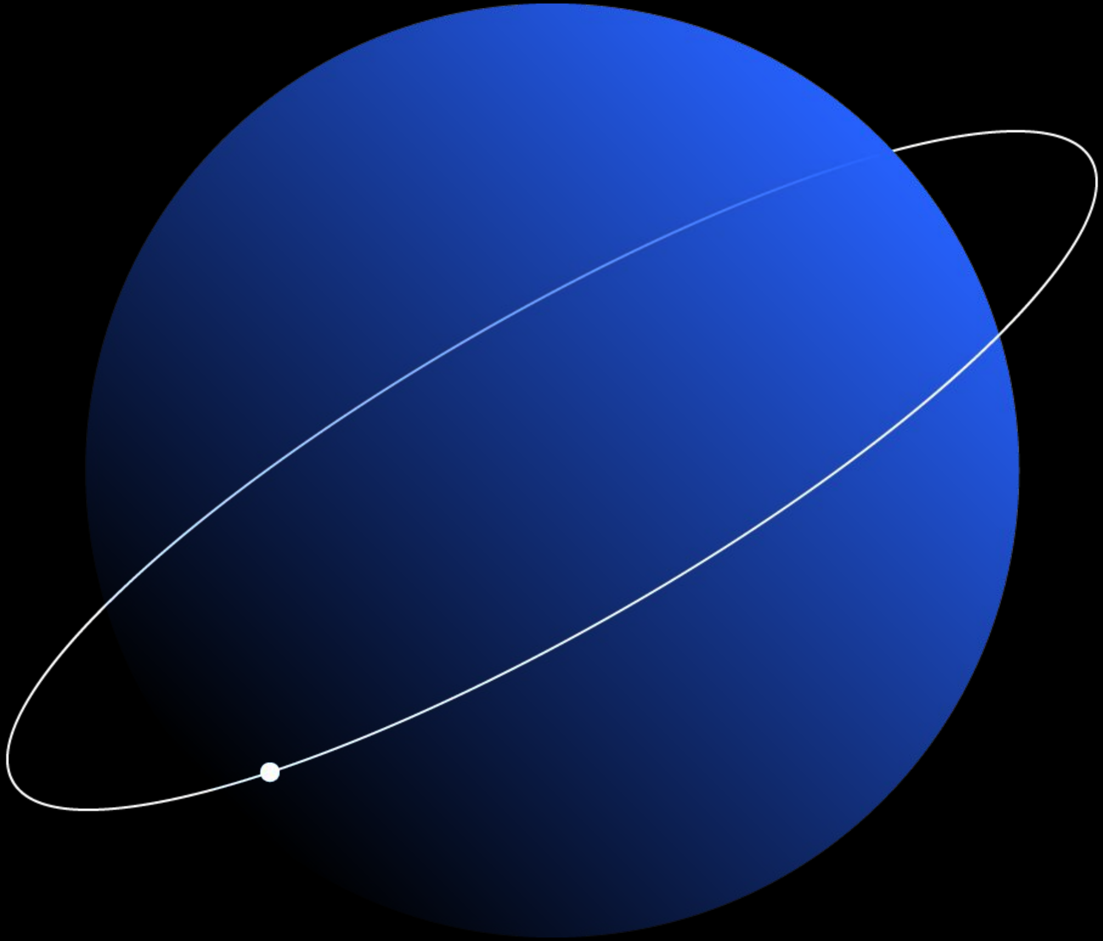
The Global 2000

15% APR

Lisa’s Pizza Joint

Centrifuge

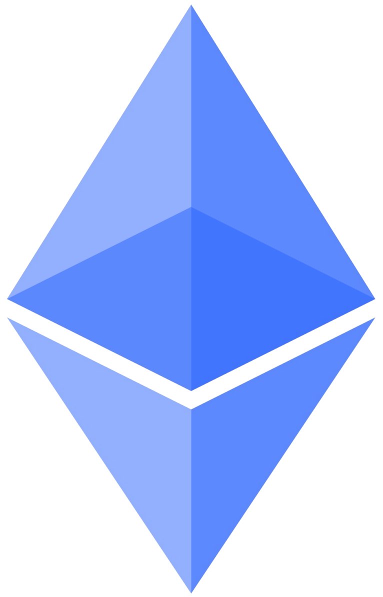
1, 2, 3



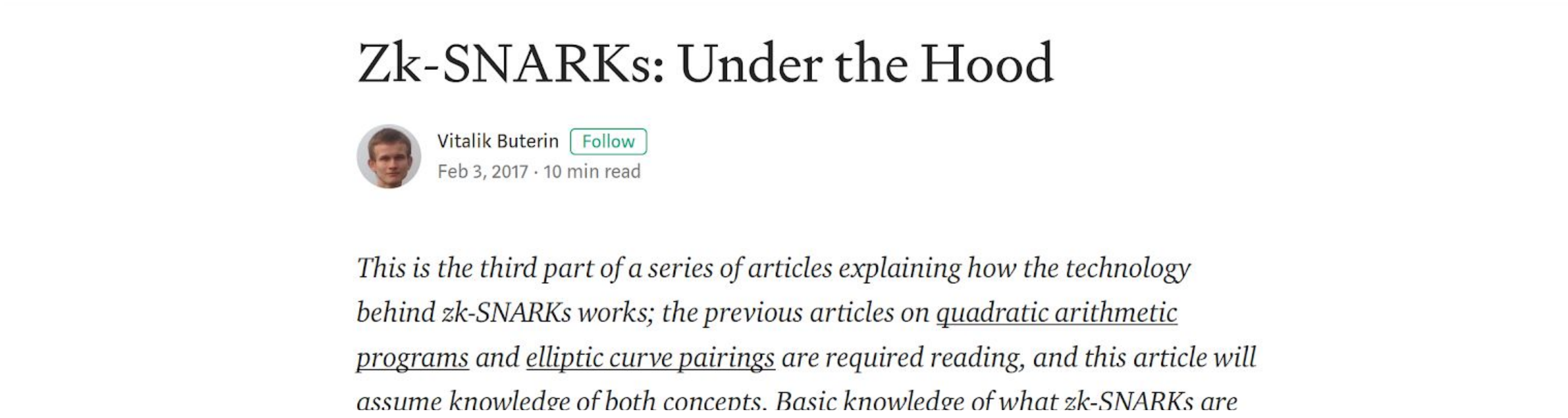
1. My adventures with snarks
2. Practical Use Case
3. Where to from here

Disclosure: I’m not a cryptographer

My perspective: get something into Ethereum Mainnet



Feb 2017


Zk-SNARKs: Under the Hood

Vitalik Buterin Follow
Feb 3, 2017 · 10 min read

This is the third part of a series of articles explaining how the technology behind zk-SNARKs works; the previous articles on quadratic arithmetic programs and elliptic curve pairings are required reading, and this article will assume knowledge of both concepts. Basic knowledge of what zk-SNARKs are

Nov 2017: Let's try ZoKrates

First example:

```
prove that you know secret values a, b
such that a+b < 15
```

As easy as pie

zokrates.github.io

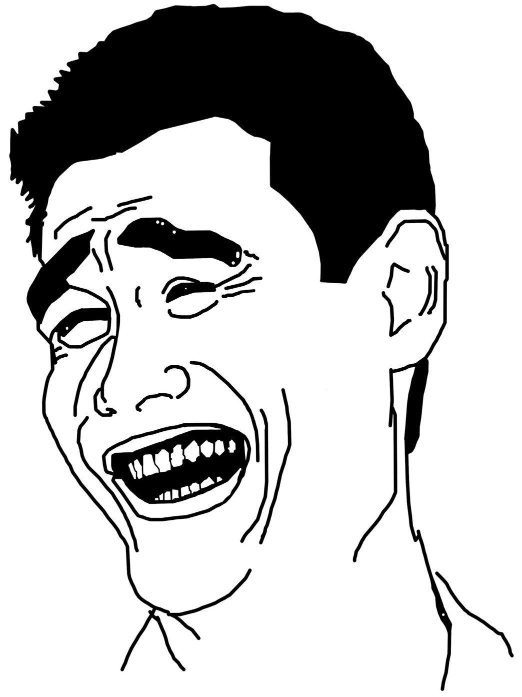
Nov 2017: Let’s try ZoKrates

Second example:

```
prove that you know secret value a such that

a < b and sha256(a) = b
```

As easy as pie? In 2017: NOT.



Jun 2018: Still trying to get SHA256 into a SNARK

Presented at Zcon0 in Montreal:

SHA256 now ~27’000 Constraints instead of ~270’000

Almost optimal.

Jun 2018: JubJub & Embedded Curves

Pedersen Hashes? Embedded Curves? EDDSA?

45s to 4s 🤯

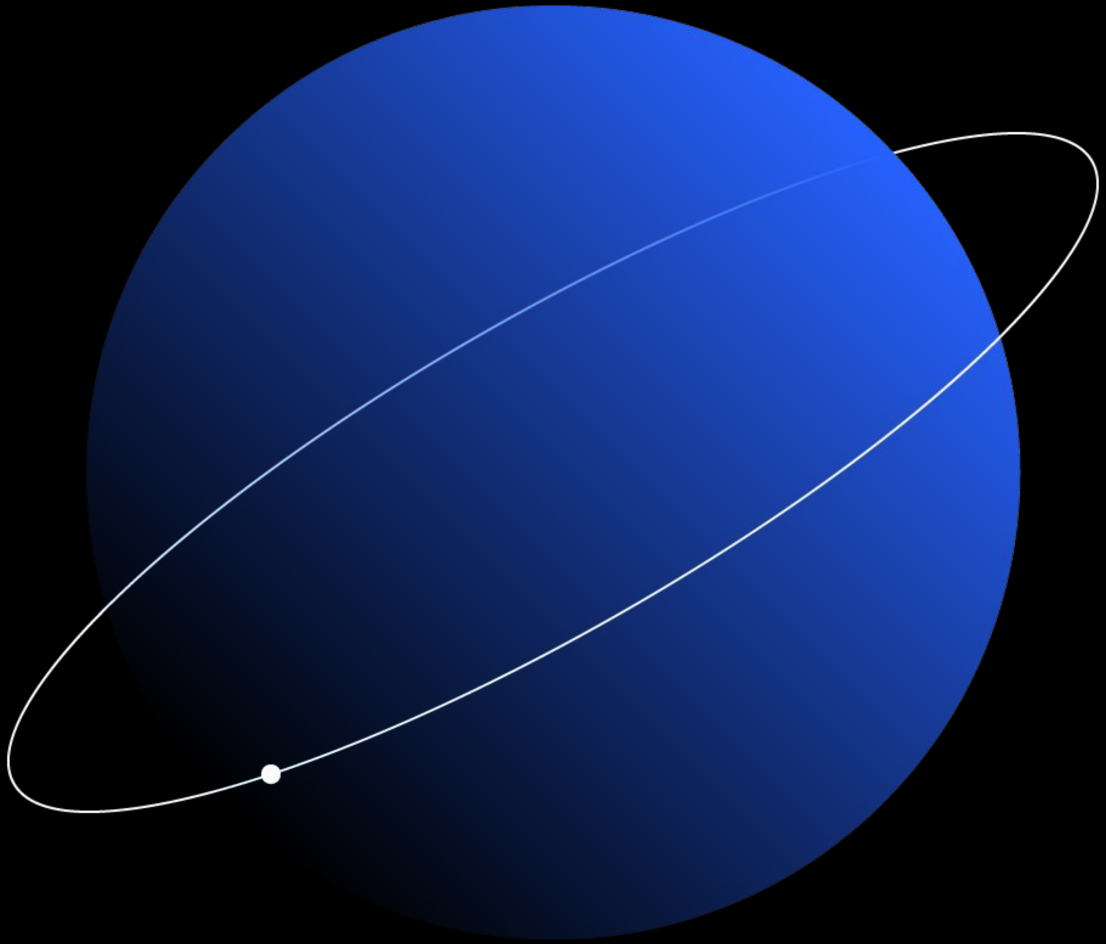
ZcashSapling

Jun 2019: zokrates-pycrypto and stdlib

ZoKrates implements over Baby_jubjub:

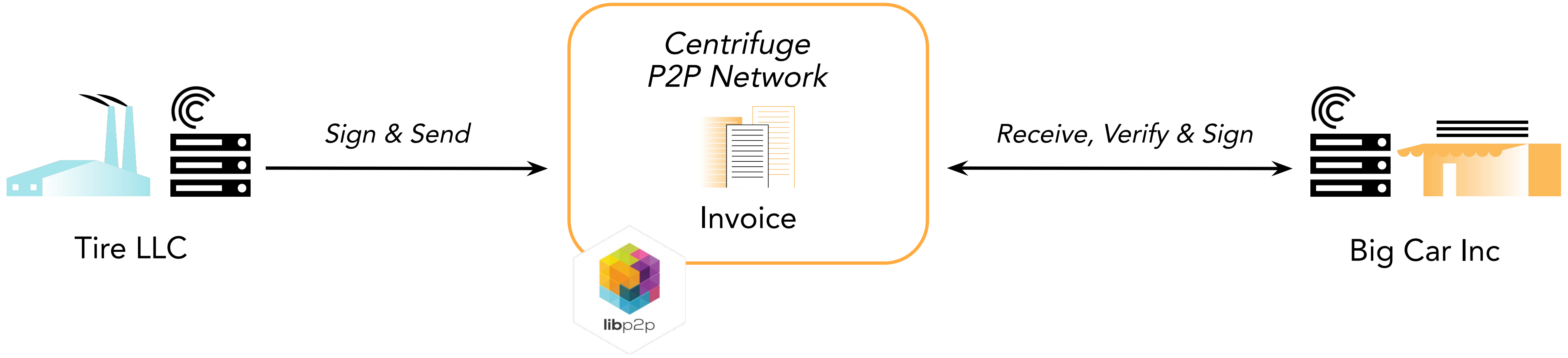
- EDDSA
- Pedersen

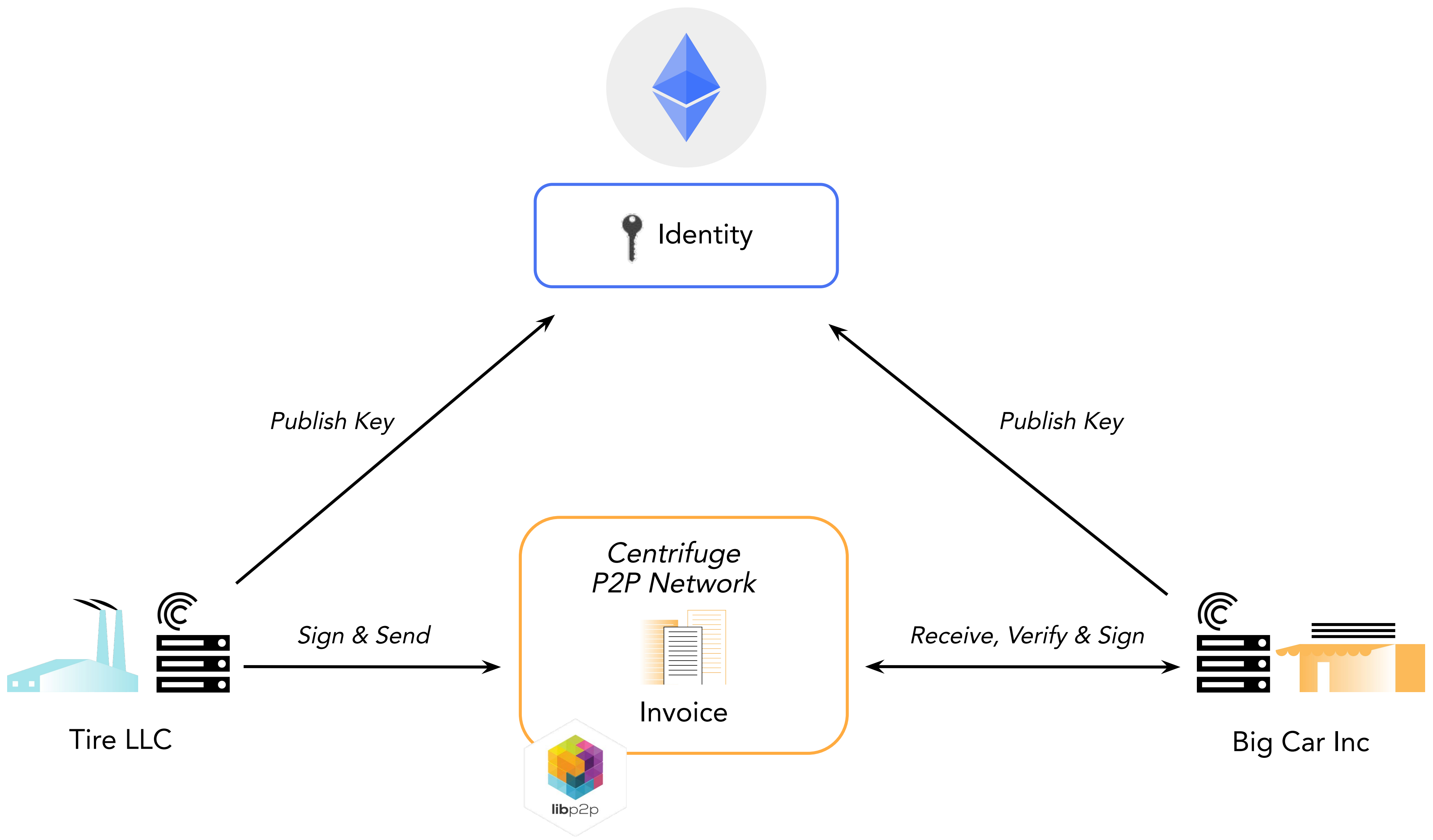
1, 2, 3

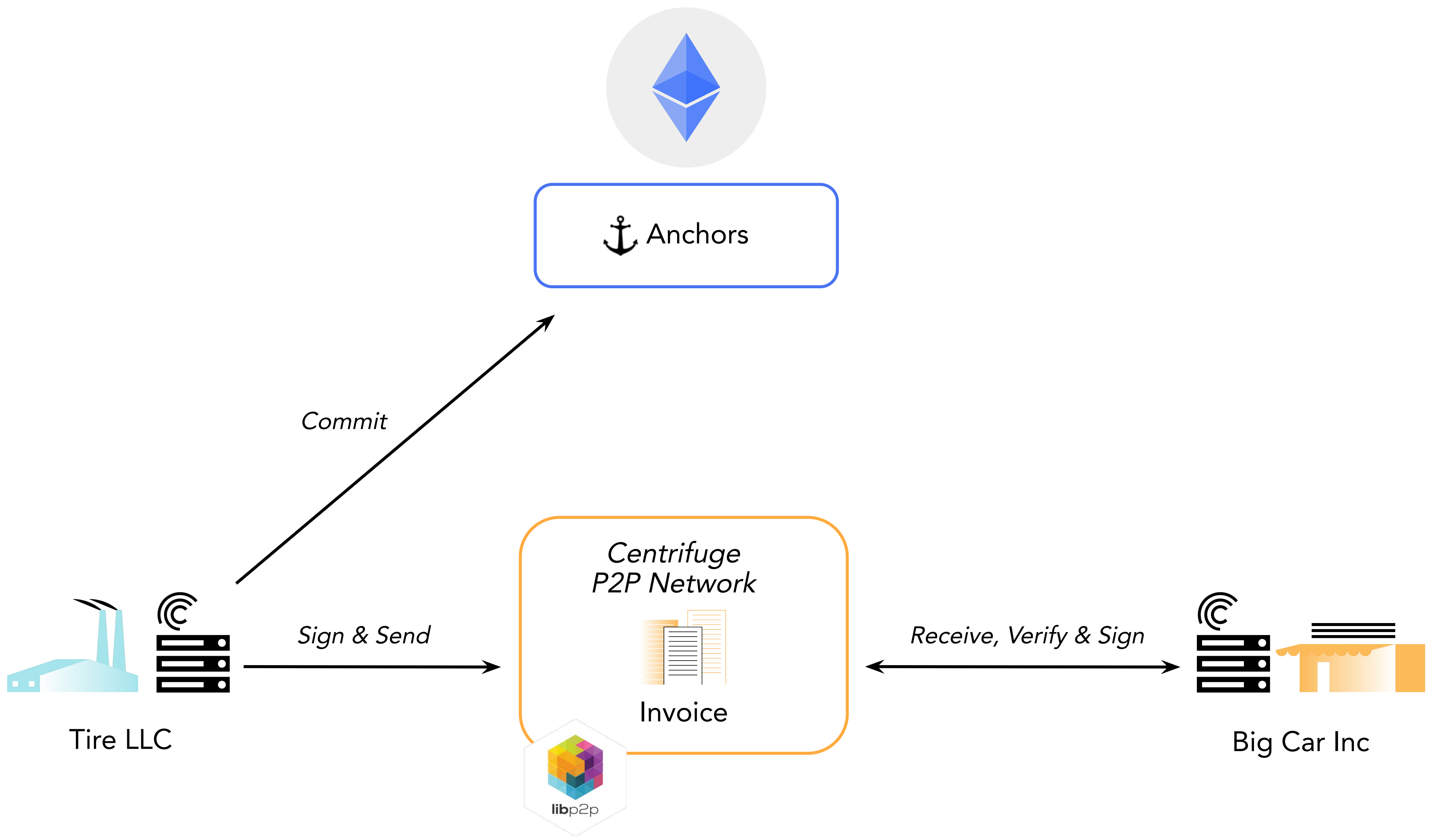


1. My adventures with snarks
2. Practical Use Case
3. Where to from here

Base Protocol



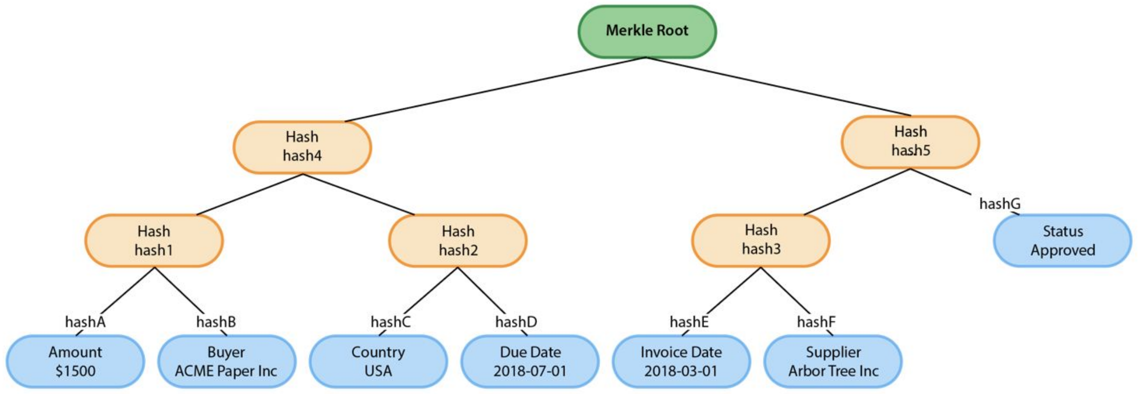

Identity
Publish Key
Publish Key
Centrifuge P2P Network
Invoice
libp2p
Sign & Send
Receive, Verify & Sign
Tire LLC
Big Car Inc


Anchors
Commit
Sign & Send
Centrifuge P2P Network
Invoice
libp2p
Receive, Verify & Sign
Tire LLC
Big Car Inc

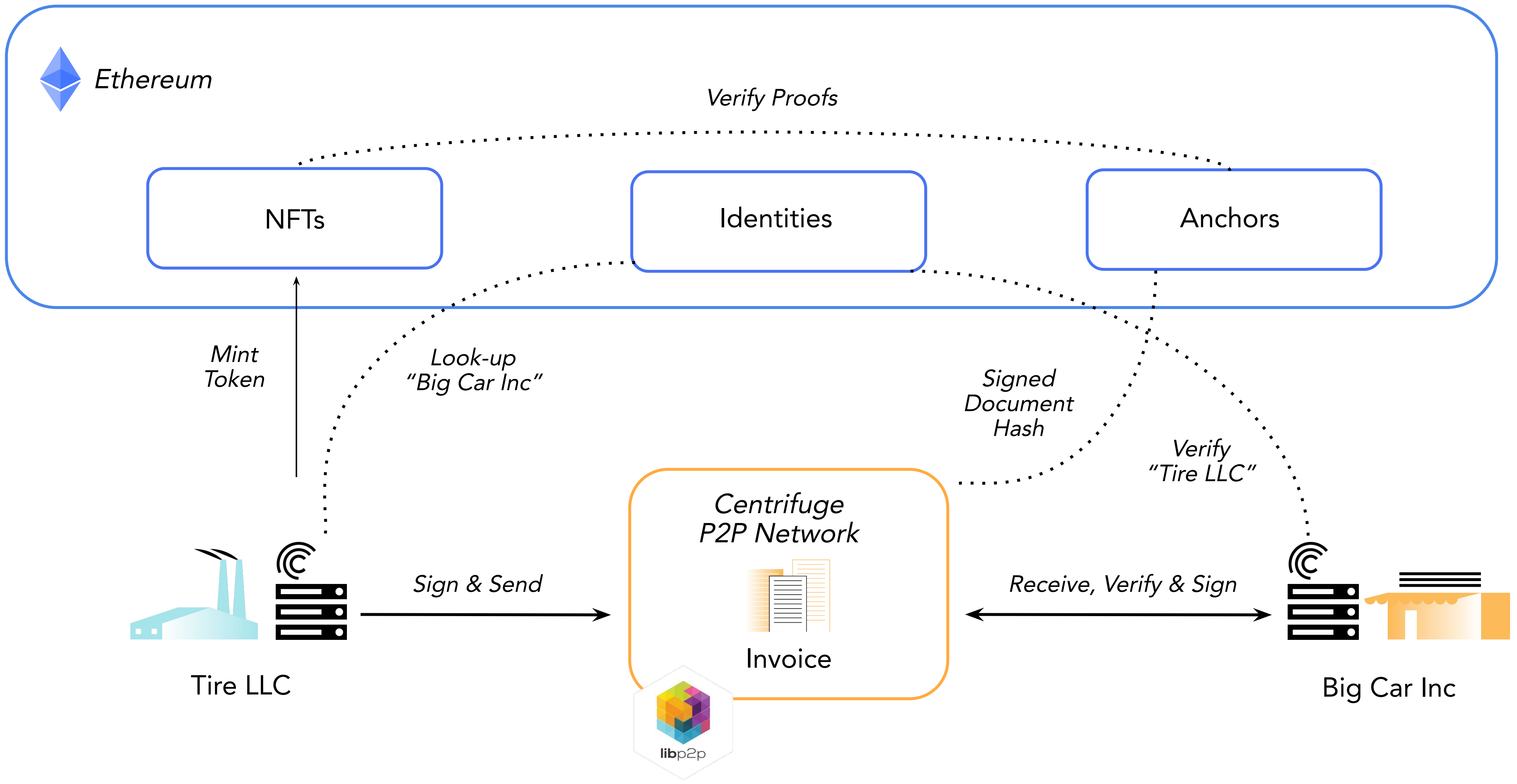
Off Chain Data

invoice

amount: $15000
buyer: Big Car Inc
supplier: Tire LLC
due date: 2019-09-09
...



merkle root


Ethereum
Verify Proofs
NFTs
Identities
Anchors
Mint Token
Look-up "Big Car Inc"
Signed Document Hash
Verify "Tire LLC"
Sign & Send
Centrifuge P2P Network
Invoice
libp2p
Receive, Verify & Sign
Tire LLC
Big Car Inc

From Off-Chain Asset to Financing

invoice

amount: $15000
buyer: Big Car Inc
supplier: Tire LLC
due date: 2019-09-09
status: unpaid
signatures: [....]
…

unpaid invoice NFT

amount: $15000
buyer: Big Car Inc
supplier: Tire LLC
due date: 2019-09-09

Tinlake

Underwriting

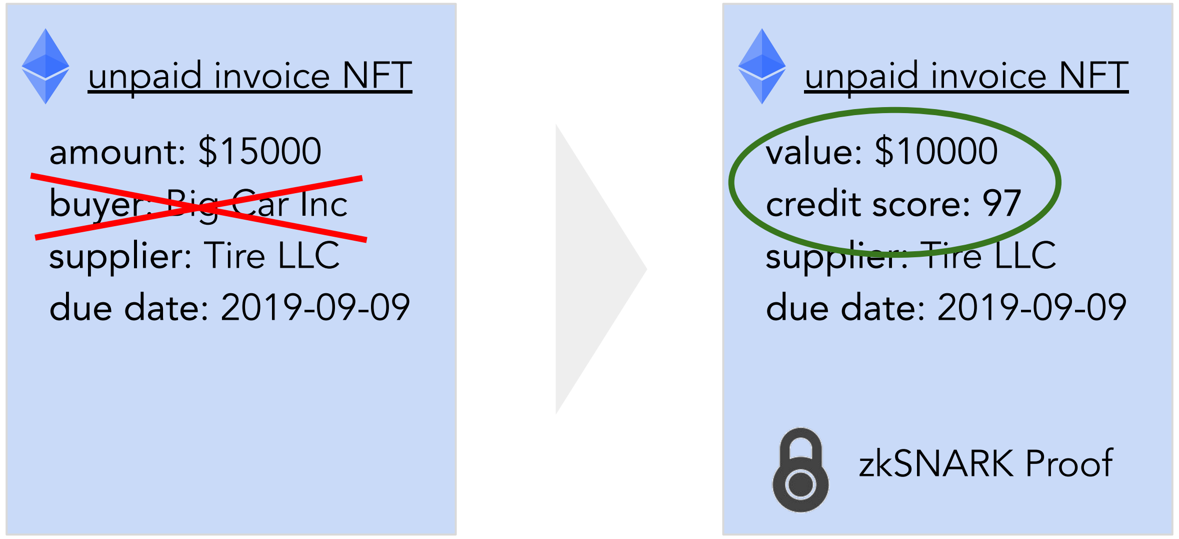
Asset Pricing

Turn NFT into ERC20s

MAKER

Compound

Adding privacy & fungibility



unpaid invoice NFT

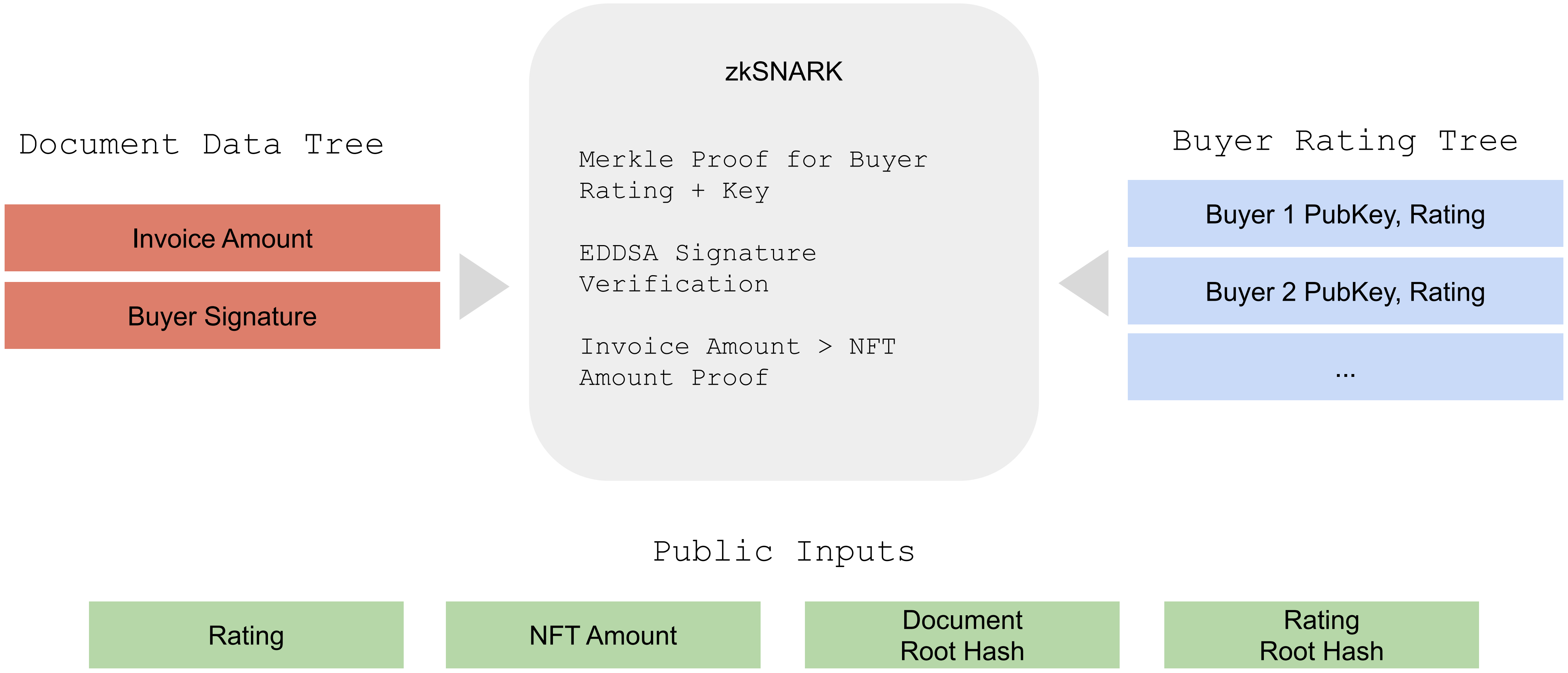
amount: $15000
~~buyer: Big Car Inc~~
supplier: Tire LLC
due date: 2019-09-09

unpaid invoice NFT

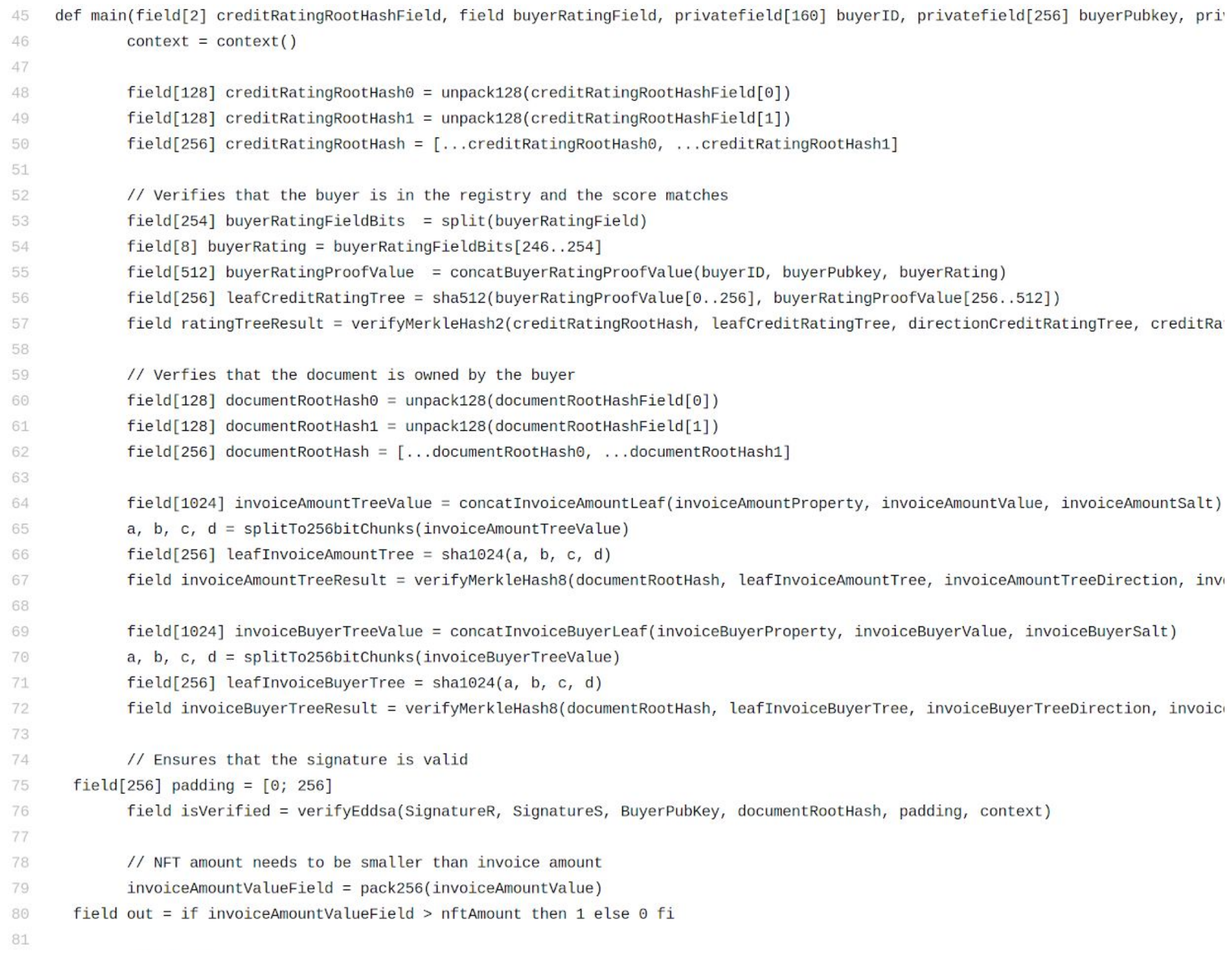
value: $10000
credit score: 97
supplier: Tire LLC
due date: 2019-09-09

zkSNARK Proof

Privacy-Preserving NFT using Trees and Signatures



The good news: it’s possible in 2019



```
def main(field[2] creditRatingRootHashField, field buyerRatingField, privatefield[160] buyerID, privatefield[256] buyerPubkey, pri
        context = context()

        field[128] creditRatingRootHash0 = unpack128(creditRatingRootHashField[0])
        field[128] creditRatingRootHash1 = unpack128(creditRatingRootHashField[1])
        field[256] creditRatingRootHash = [...creditRatingRootHash0, ...creditRatingRootHash1]

        // Verifies that the buyer is in the registry and the score matches
        field[254] buyerRatingFieldBits  = split(buyerRatingField)
        field[8] buyerRating = buyerRatingFieldBits[246..254]
        field[512] buyerRatingProofValue  = concatBuyerRatingProofValue(buyerID, buyerPubkey, buyerRating)
        field[256] leafCreditRatingTree = sha512(buyerRatingProofValue[0..256], buyerRatingProofValue[256..512])
        field ratingTreeResult = verifyMerkleHash2(creditRatingRootHash, leafCreditRatingTree, directionCreditRatingTree, creditRa

        // Verfies that the document is owned by the buyer
        field[128] documentRootHash0 = unpack128(documentRootHashField[0])
        field[128] documentRootHash1 = unpack128(documentRootHashField[1])
        field[256] documentRootHash = [...documentRootHash0, ...documentRootHash1]

        field[1024] invoiceAmountTreeValue = concatInvoiceAmountLeaf(invoiceAmountProperty, invoiceAmountValue, invoiceAmountSalt)
        a, b, c, d = splitTo256bitChunks(invoiceAmountTreeValue)
        field[256] leafInvoiceAmountTree = sha1024(a, b, c, d)
        field invoiceAmountTreeResult = verifyMerkleHash8(documentRootHash, leafInvoiceAmountTree, invoiceAmountTreeDirection, inv

        field[1024] invoiceBuyerTreeValue = concatInvoiceBuyerLeaf(invoiceBuyerProperty, invoiceBuyerValue, invoiceBuyerSalt)
        a, b, c, d = splitTo256bitChunks(invoiceBuyerTreeValue)
        field[256] leafInvoiceBuyerTree = sha1024(a, b, c, d)
        field invoiceBuyerTreeResult = verifyMerkleHash8(documentRootHash, leafInvoiceBuyerTree, invoiceBuyerTreeDirection, invoic

        // Ensures that the signature is valid
  field[256] padding = [0; 256]
        field isVerified = verifyEddsa(SignatureR, SignatureS, BuyerPubKey, documentRootHash, padding, context)

        // NFT amount needs to be smaller than invoice amount
        invoiceAmountValueField = pack256(invoiceAmountValue)
  field out = if invoiceAmountValueField > nftAmount then 1 else 0 fi
```

https://github.com/centrifuge/zk-nft-demo-contract/

Results

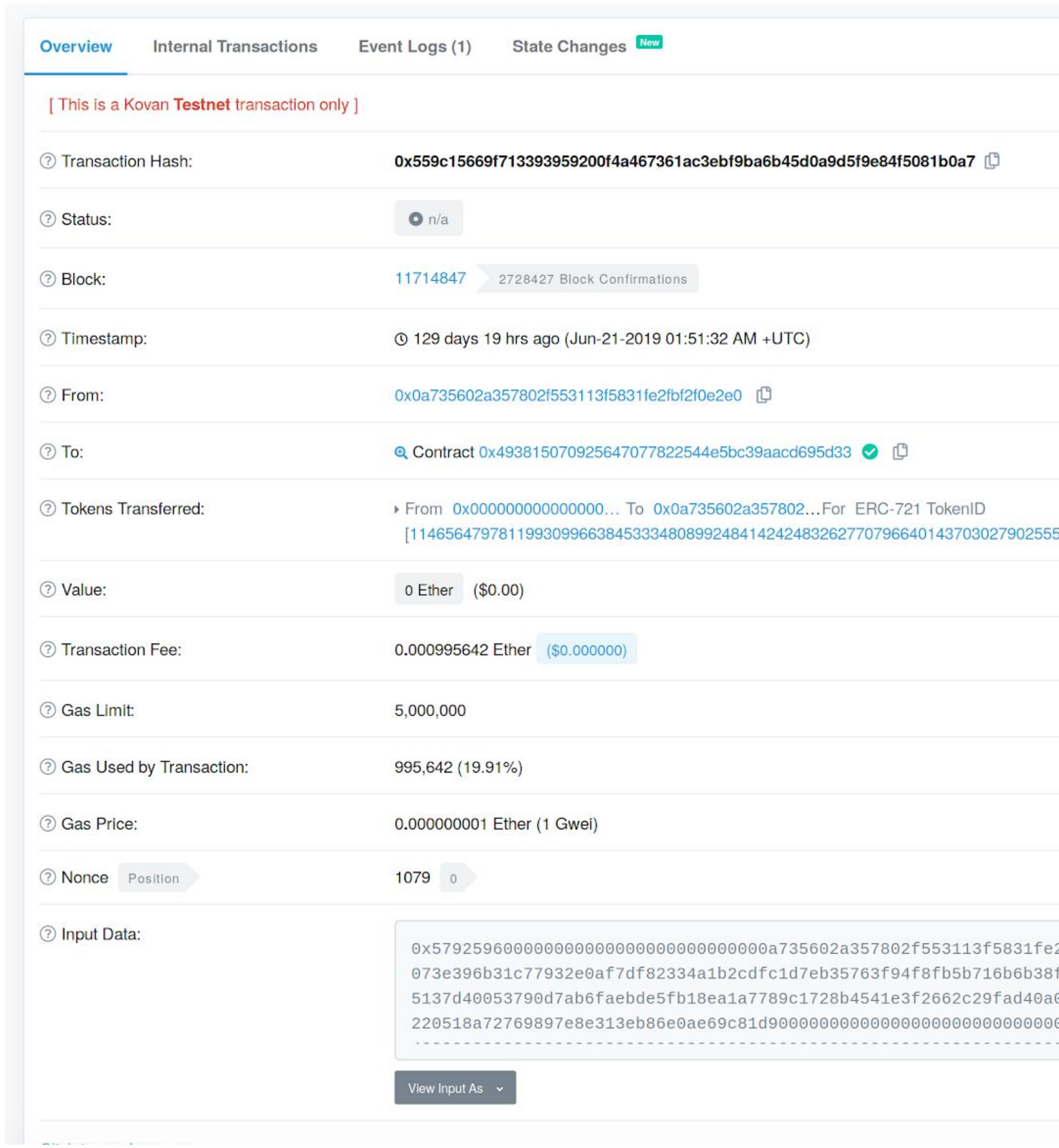
2:30min proving time

~ 900k Gas (less with Istanbul)

already comparable to similar on chain tx

zokrates-pycrypto:

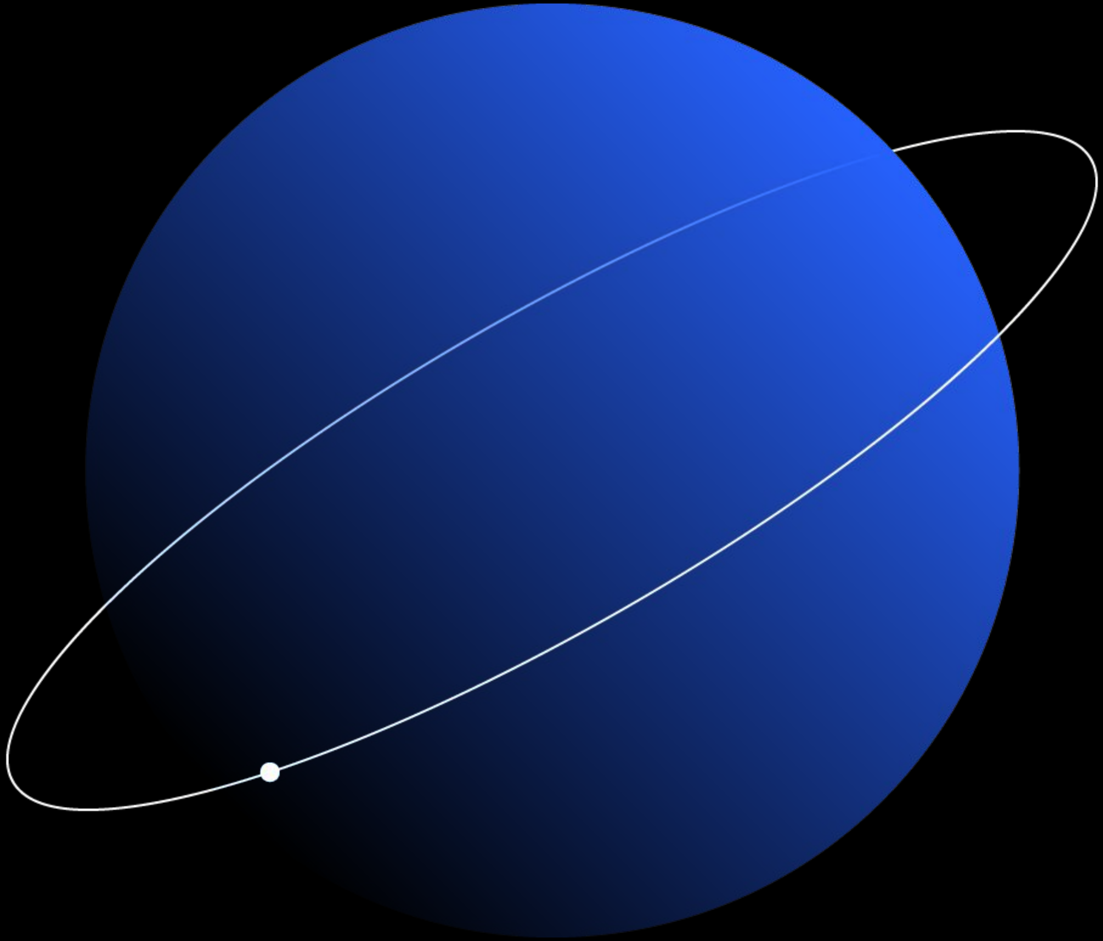
- Pedersen
- EDDSA



[This is a Kovan **Testnet** transaction only]

Transaction Hash:	0x559c15669f713393959200f4a467361ac3ebf9ba6b45d0a9d5f9e84f5081b0a7
Status:	n/a
Block:	11714847 2728427 Block Confirmations
Timestamp:	129 days 19 hrs ago (Jun-21-2019 01:51:32 AM +UTC)
From:	0x0a735602a357802f553113f5831fe2fbf2f0e2e0
To:	Contract 0x493815070925647077822544e5bc39aacd695d33
Tokens Transferred:	From 0x000000000000000... To 0x0a735602a357802...For ERC-721 TokenID [1146564797811993099663845333480899248414242483262770796640143703027902555
Value:	0 Ether ($0.00)
Transaction Fee:	0.000995642 Ether ($0.000000)
Gas Limit:	5,000,000
Gas Used by Transaction:	995,642 (19.91%)
Gas Price:	0.000000001 Ether (1 Gwei)
Nonce Position	1079 0
Input Data:	0x579259600000000000000000000000000a735602a357802f553113f5831fe2 073e396b31c77932e0af7df82334a1b2cdfc1d7eb35763f94f8fb5b716b6b38f 5137d40053790d7ab6faebde5fb18ea1a7789c1728b4541e3f2662c29fad40a0 220518a72769897e8e313eb86e0ae69c81d900000000000000000000000000000

1, 2, 3

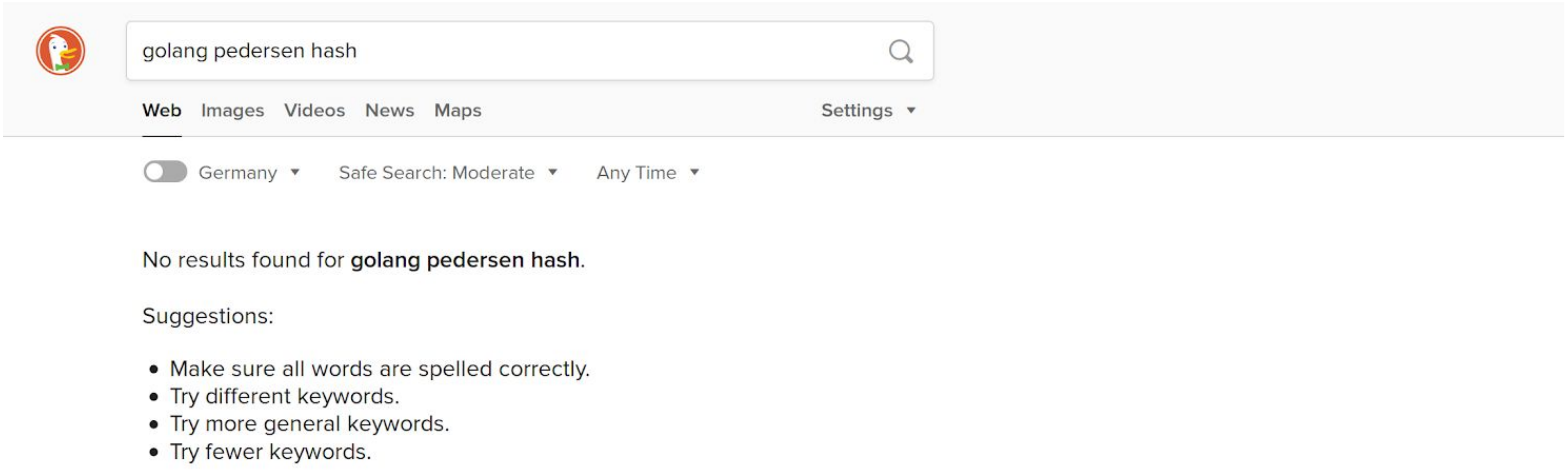


1. My adventures with snarks
2. Practical Use Case
3. Outlook

ZoKrates & baby_JubJub

The future looks bright. ZoKrates shows how easy this can be made. But a lot of it is based on outdated technology.

About those primitives


golang pedersen hash

No results found for **golang pedersen hash**.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.

Find the perfect hash function

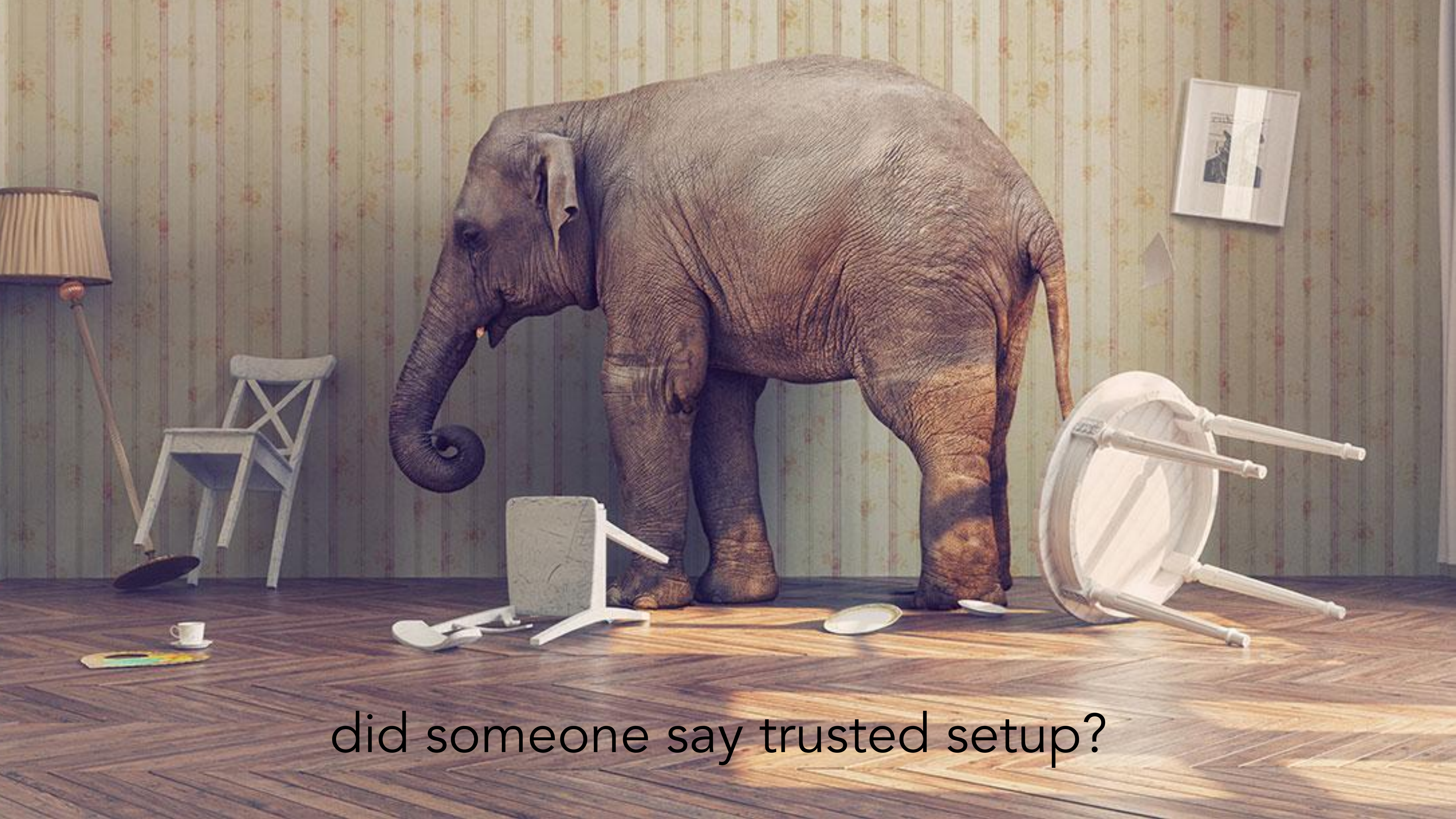
	Ethereum Gas Cost	Constraints	Implementations
SHA256	60+12/word	25.8k	Everywhere
keccak256	30+6/word	-	Everywhere
Blake2	28’000 (now less with Istanbul)	21.5k	In most crypto libs
Pedersen	n/a	2k	Not available, not audited except ZCash’ Rust Impl, Curve dependent
MiMC	16’000	< 1k	Not production ready

Signature Schemes

Standards

Support HSMs

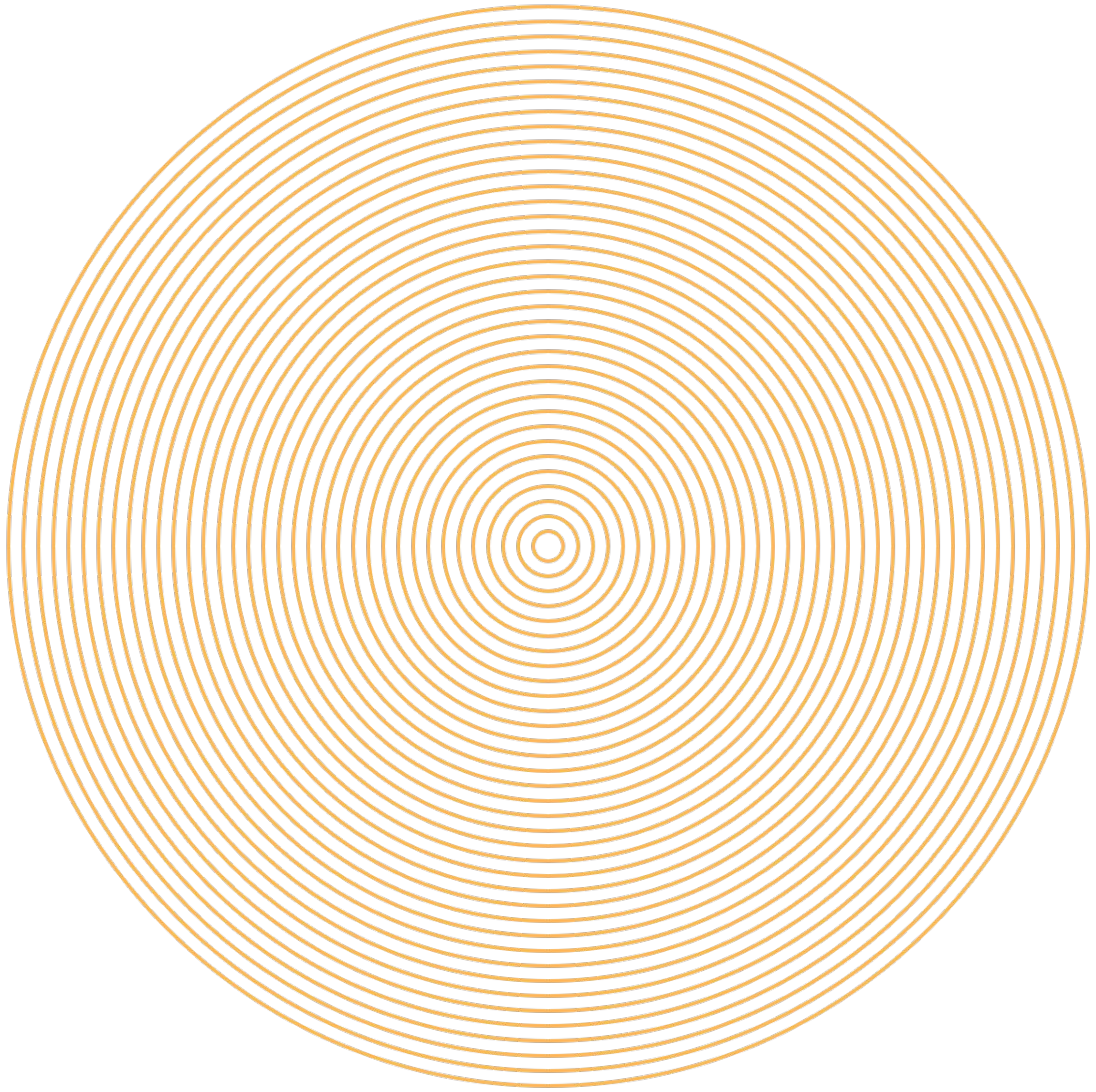
HD Key Derivation


did someone say trusted setup?

Where next?

I can't wait for [insert your favorite proving scheme here].

Questions?



https://centrifuge.io
https://twitter.com/lucasvo
Medium Post on NFT: https://bit.ly/2pj4Yyz