clearmatics

Leveraging ZKPs to Design Privacy Preserving State Transitions on Ethereum

Antoine Rondelet

October 29th, 2019
ZKProof Community Event – Amsterdam, The Netherlands

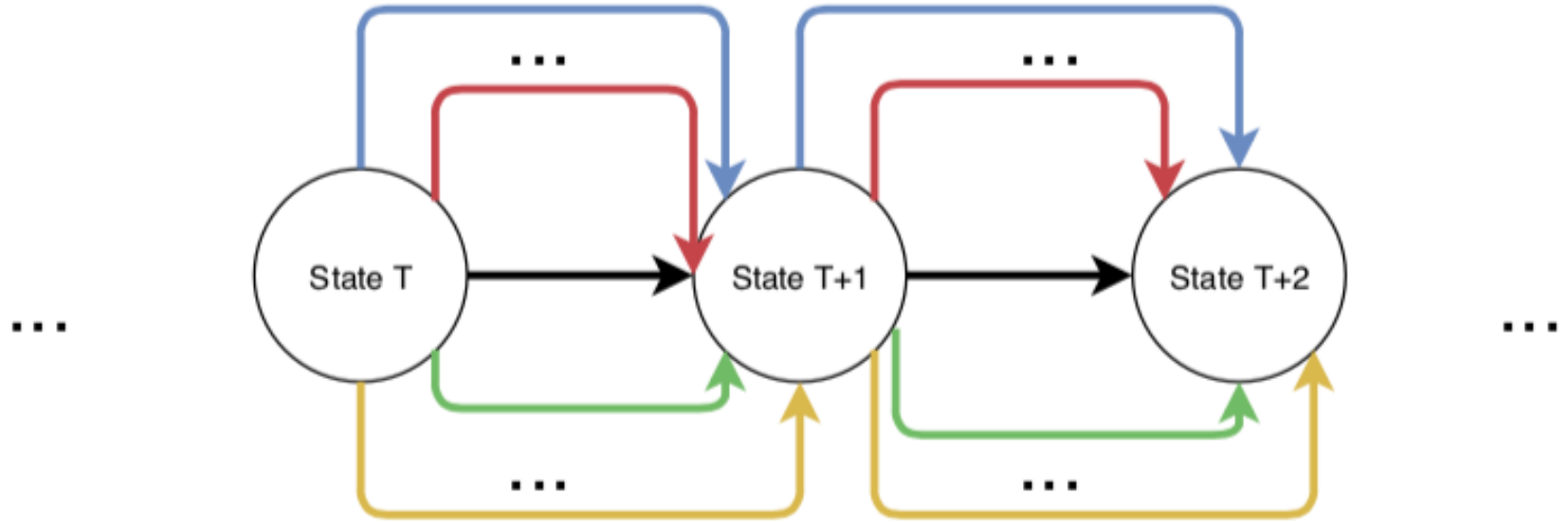
clearmatics.com



@antoineRnd (Telegram)
@itranneo (Twitter)
https://keybase.io/antoinerondelet

Ethereum: Distributed state machine

clearmatics



"Smart contracts"
≈
State transition description

Smart contract: Specify state transitions

clearmatics

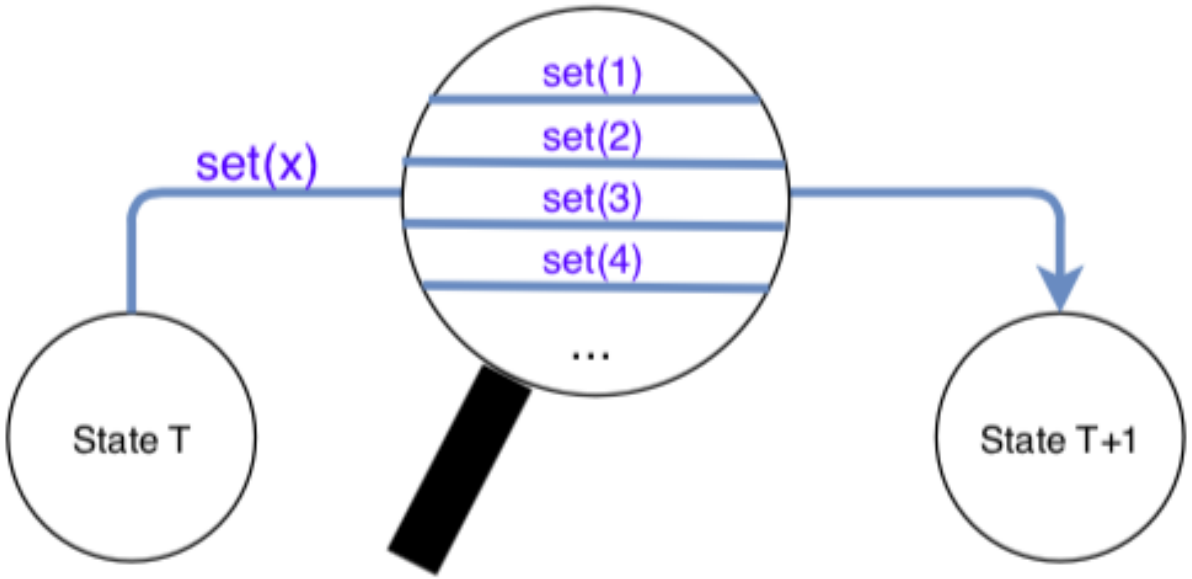
```
pragma solidity ^0.4.0;

contract SampleContract {
    uint storageData;

    function set(uint x) {
        storageData = x;
    }

    function get() constant returns (uint) {
        return storageData;
    }
}
```

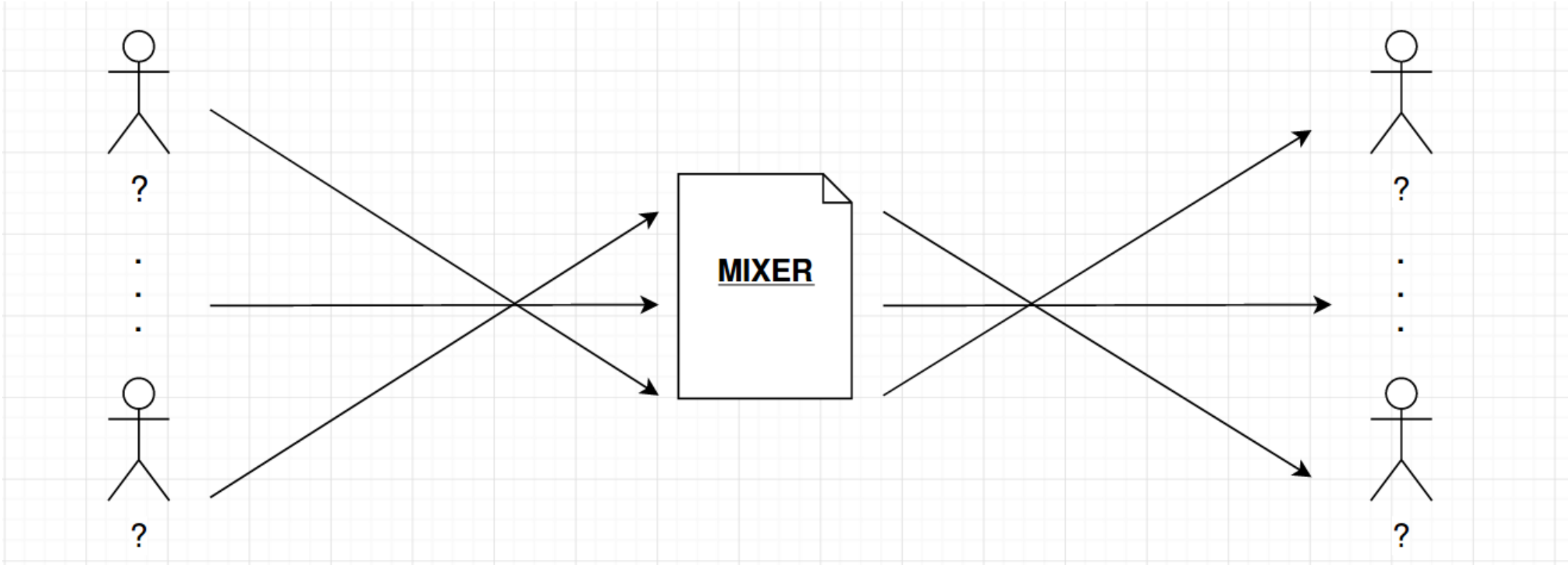
Set() can be called with 2^{256} values (2^{256} possible state transitions)



Ethereum what?

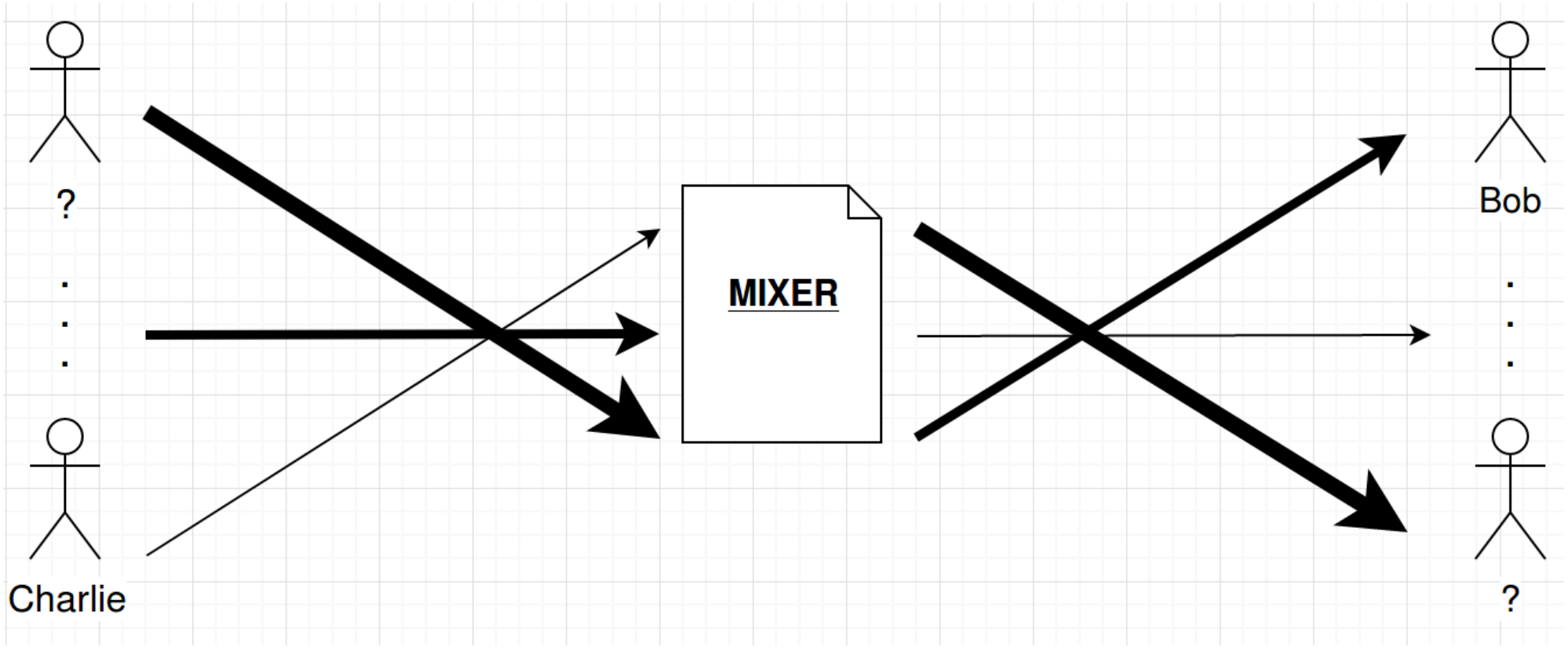
- Distributed state machine
- Account model
- Smart contracts: Encode custom state transitions
  - “gas” to avoid the halting problem → “You pay for the storage cost and the computation cost of the state transition you execute”
- Every transaction is stored on-chain and needs to be verified before being executed (signature verification and account nonce check) → Every transaction has a default cost of 21k gas
  - Problem for privacy! We can’t submit a transaction from a newly generated account! (If an account is funded, we tend to assume the identity of the owner is known)

The “Mixer Ideal”



Do we really live in this Ideal world?

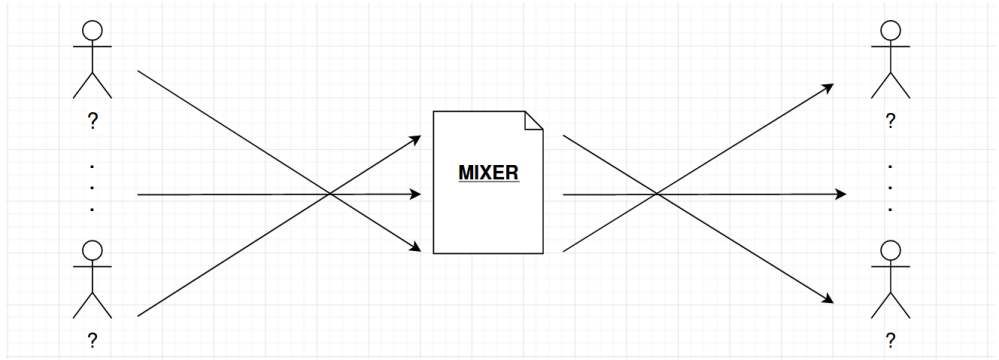
clearmatics



“Indistinguishability game” for mixers

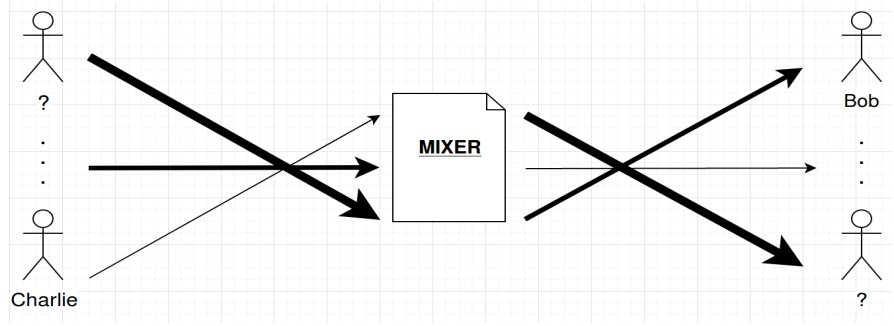
clearmatics

IDEAL world



- What is the gap between IDEAL and REAL worlds?
- Can we easily infer what’s going through the Mixer we designed?

REAL world



IND-MXR Game:

(Chain, Mxr, {A.pk, A.sk}, {Users.pk, Users.sk}) ← Gen(1^{λ})

(Pay_0, Pay_1) ← A(Chain, {A.pk, A.sk}, {Users.pk})

b ←$-- {0,1}

If b == 0:

    Chain ← Execute(Chain, Pay_0, {Users.pk, Users.sk})

Else:

    Chain ← Execute(Chain, Pay_1, {Users.pk, Users.sk})

b’ ← A(Chain, {A.pk, A.sk}, {Users.pk})

Return b == b’

A first Mixer attempt: Mobius

clearmatics

Linkable Ring Signatures

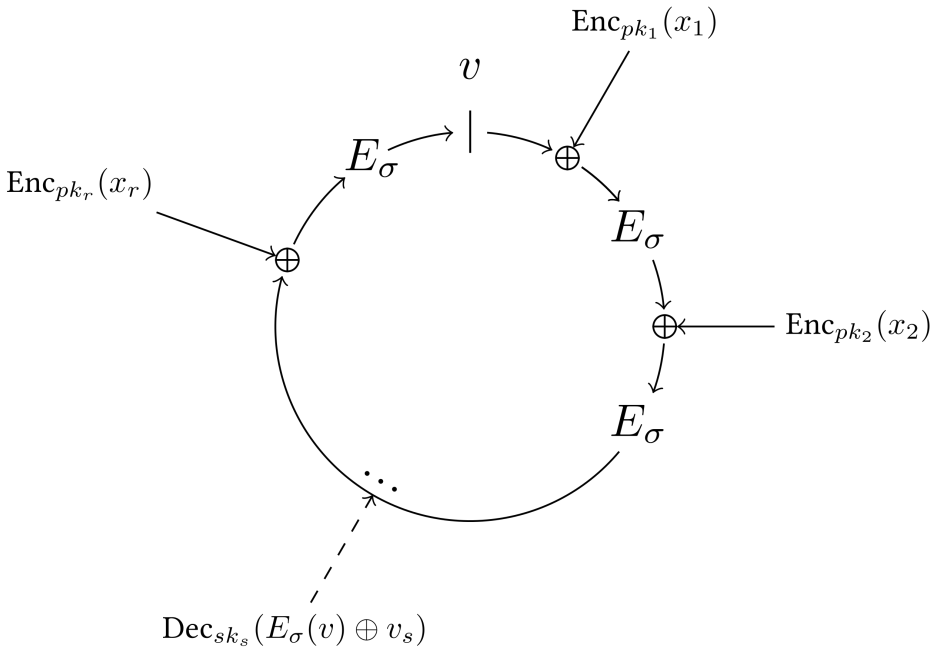


Image from:
https://en.wikipedia.org/wiki/Ring_signature

Detect if 2 signatures have been produced by the under the same private key (without revealing the signer)

“Zero-Knowledge proof of membership of a set”

A first Mixer attempt: Mobius (2)

- Alice wants to pay Bob
  - Exchange secret s
  - Bob shares his mpk_{bob}
  - Alice derives a spk for Bob, $spk_{bob} = mpk_{bob} * g^{Hash(s \,||\, nonce)}$
  - Alice deposits on the Mixer which add spk_{bob} to the list of pub keys that compose the ring R on the Mixer smart contract
  - When R is full, the contract emits a message m
    - Bob can generate a ring sig on m using the set of pub keys in the ring and his $ssk_{bob} = msk_{bob} + Hash(s \,||\, nonce)$
    - If the ring sig verifies correctly, the funds can be withdrawn to Bob's stealth address (the signer was a recipient of a payment).

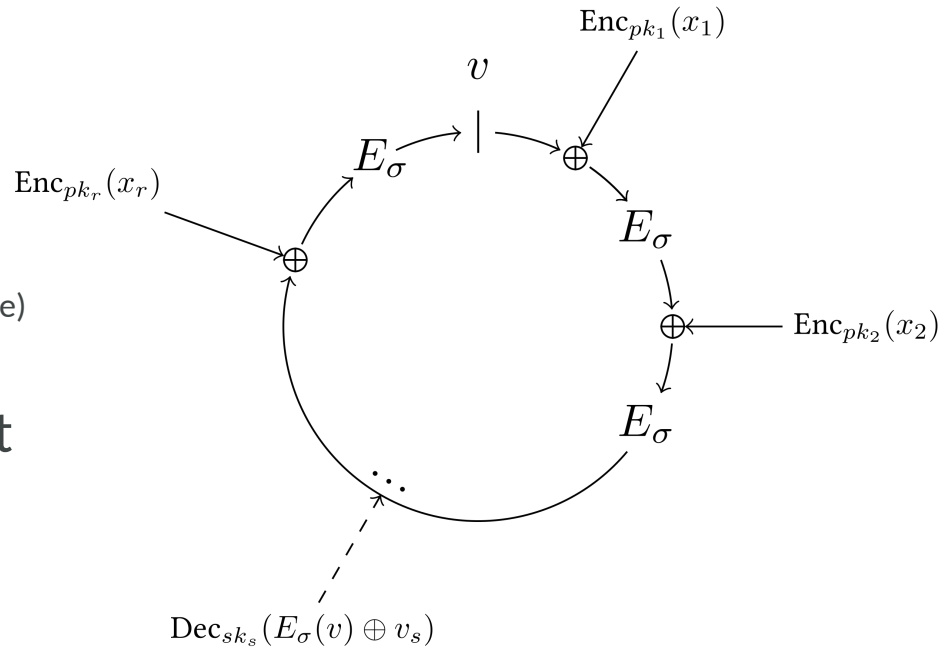


Image from:
https://en.wikipedia.org/wiki/Ring_signature

Not viable in practice. Introduces a big gap between IDEAL and REAL

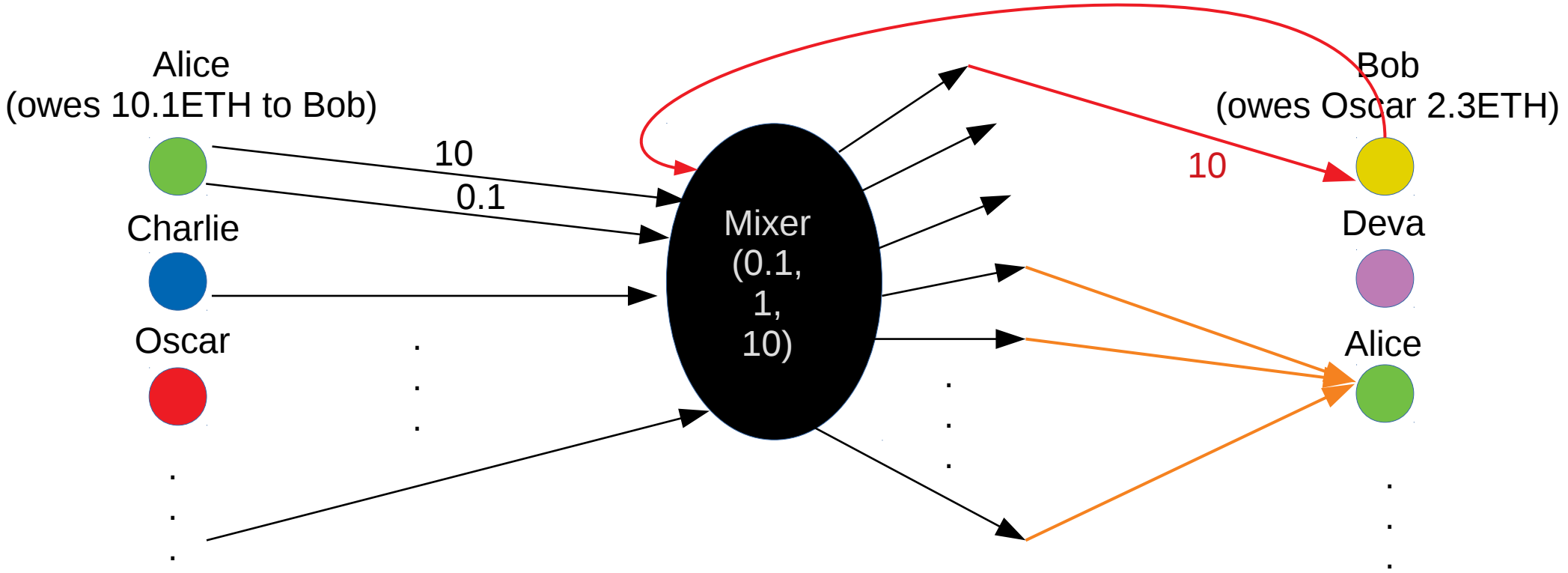
A first Mixer attempt: Mobius (3)

- Several issues:
  - 1 ring – 1 denomination
  - Latency issues (the rings needs to be full – we need to reach a threshold of deposited pub keys before we can withdraw)
  - Payment atomicity issues (if base X decomposition of payment value)
  - Small anonymity sets (cf: Latency)
  - … and Stealth addresses are not viable on Ethereum

May be possible to improve, but things get complicated very quickly and it becomes hard to argue about the actual privacy level…

Careful with note denominations!

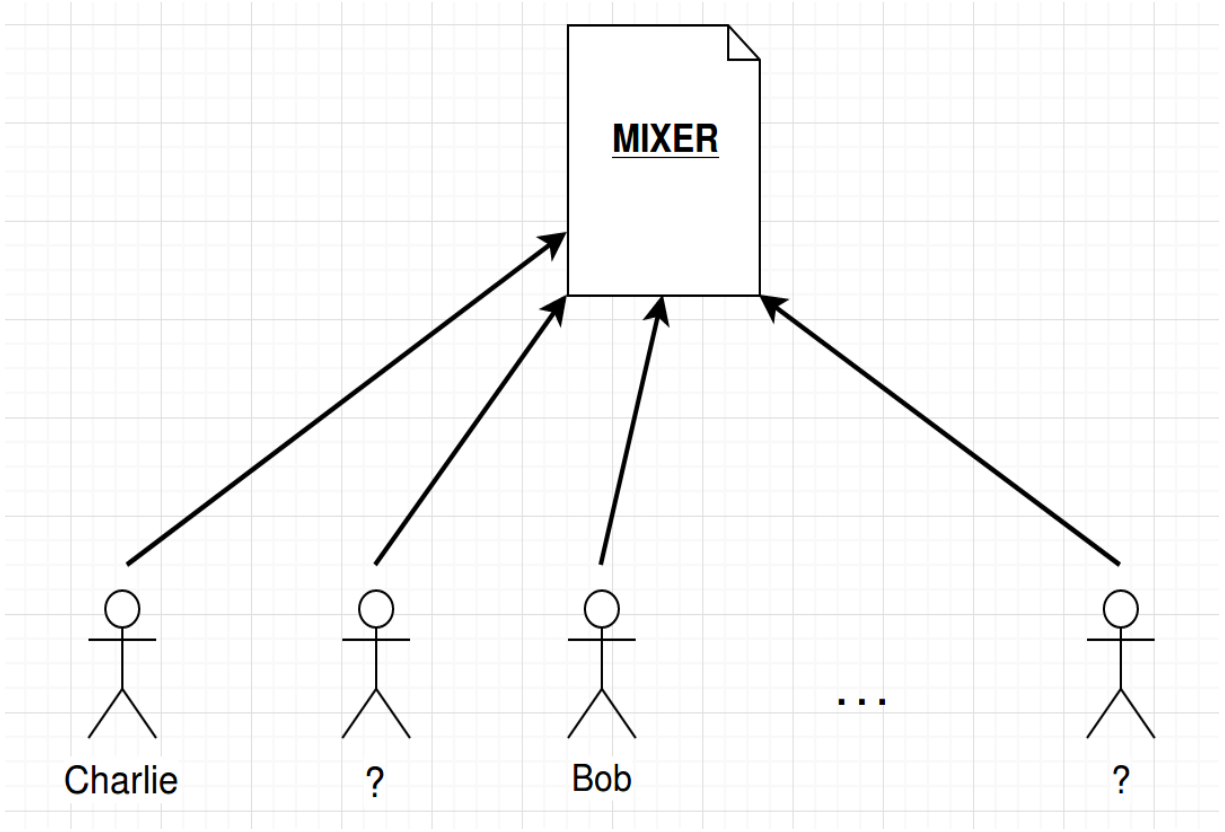
- Fixed denominations are BAD:
  - Increase number of communications with the “Mixing contract” (users are forced to create “the change” the need by themselves → Forced to withdraw and re-deposit → deadly metadata)
  - Increase cost (gas and proofs generation) or the solution (gas paid for each tx)



Build an Anonymous Payment scheme

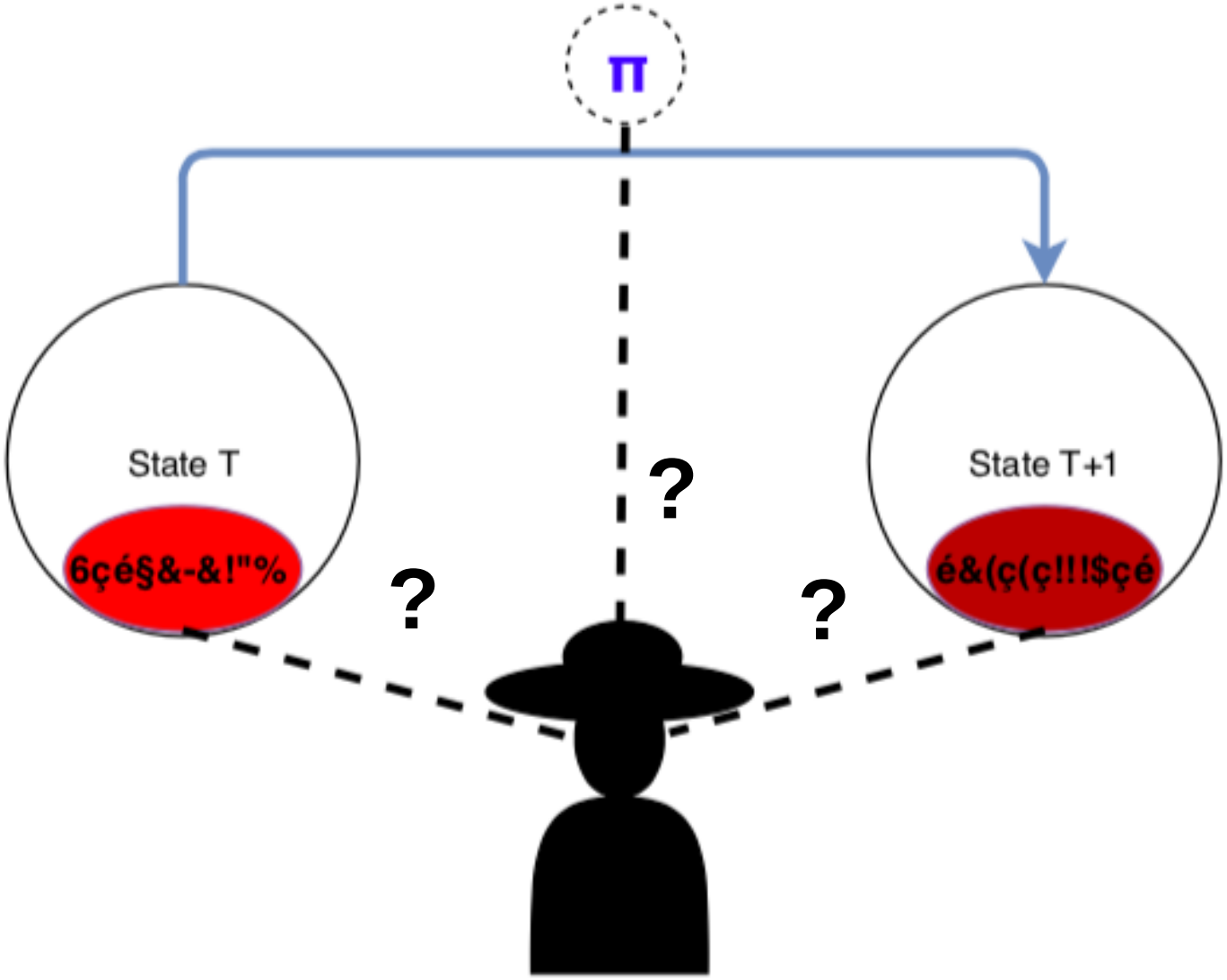
clearmatics

- Provide the same payment possibilities than the base layer
  - “No need to withdraw during the use of the system”
- Can we provide a functionality to change a set of notes to another set of notes (Change 10ETH to 2 notes of 3 and 7ETH for instance)?
  - If so, we address one big inconvenient of fixed denomination-based mixers, and save a lot of information leakages.
  - Funds (could) never leave the Mixer



What about “ZK State transitions”?

clearmatics



Proof systems – Brief background

- Proof ~ Something that convinces someone that a statement is true
- Here, we'll be interested in proving that *x* belongs to *L* (or equivalently, ∃ *w* s.t *(x, w)* ∈ *R)*
- Proof system ~ Procedure that decides which proof are convincing and which are not (acc/rej)
- Different settings of interest:
  - Non-Interactivity (Prover sends proof to Verifier (1 msg), and Verifier decides if acc or rej) → Reference String
  - Zero-knowledge ~ "You (Verifier) know nothing except that *x* ∈ *L*" (*Simulator*)
  - Proof of Knowledge (PoK) ~ "It's not enough to prove to me ∃ w s.t (x, w) ∈ R, I want to have the proof that you know such pair!" (*Extractor*)
- Traditional mathematical proofs (checked line by line) ≠ Probabilistic proofs (use randomness, a false proof is accepted with negligible probability)

(zk)SNARKs: Blockchain friendly

clearmαtics

- **Ar**gument ~ Computationally sound proof (the prover is assumed to be computationally **bounded**)
- Proof of **K**nowledge (PoK) ~ “It’s not enough to prove to me that there exist a pair (x, w) ∈ R, I want to have the proof that you know such pair!” This idea is captured by the notion of an *Extractor*.
- **S**uccinctness ~ We want to make sure that the proof is small compared to the size of (x, w) (“Communication succint”), and that it is quick to verify (“Verification succint”) => Perfect for us! We can do on-chain verifications + we can limit the impact of storing proofs on-chain.

+

- Very fast to verify
- Small proofs
- “Support NP”

-

- “Trapdoored”
- SRS/relation*

*For the most efficient, SNARK – Groth16.

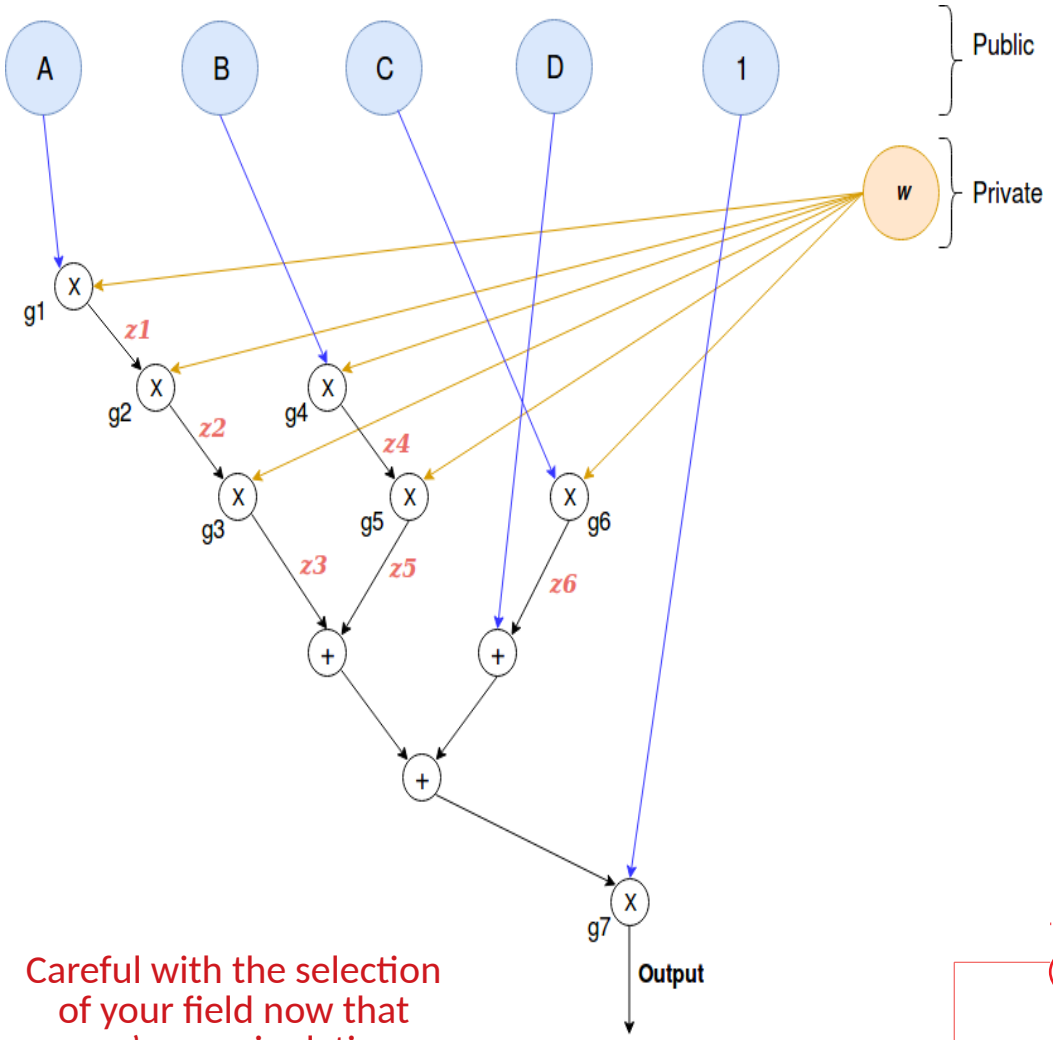
Using zkSNARKs

- Define the NP-statement: “What do you want to prove?”
- Represent your proposition as an arithmetic circuit *C (defined over a f. field F)*
- The prover shows knowledge of a satisfiable assignment to *C* (ie: valid solution to circuit-SAT for *C*)
  - Here the idea is that only *x* in *(x, w)* is used as input of the circuit.
  - The witness *w*, however is used “inside *C*”, as the intermediate wires (*“non-determinism”*)*
  - The goal here for *P* is to show to *V* that on input x, the output of the circuit is 1, WITHOUT leaking any information about the intermediate/inner state of *C*

*Note: If there is no witness w, then there is not point in using zero-knowledge since the internal state of the circuit C can be recomputed by the verifer on input x (no source of “non-determinism”)

Representing the relation R: R1CS programming (Example)

clearmatics



Careful with the selection of your field now that you're manipulating polynomials! (ie: (i)FFTs)

Statement: "I know a solution w (the witness) to the cubic equation (E') $Ax^3 + Bx^2 + Cx + D = 0$" where A, B, C, D are public (the instance).
The arithmetic circuit **C** can be represented by the system (S):

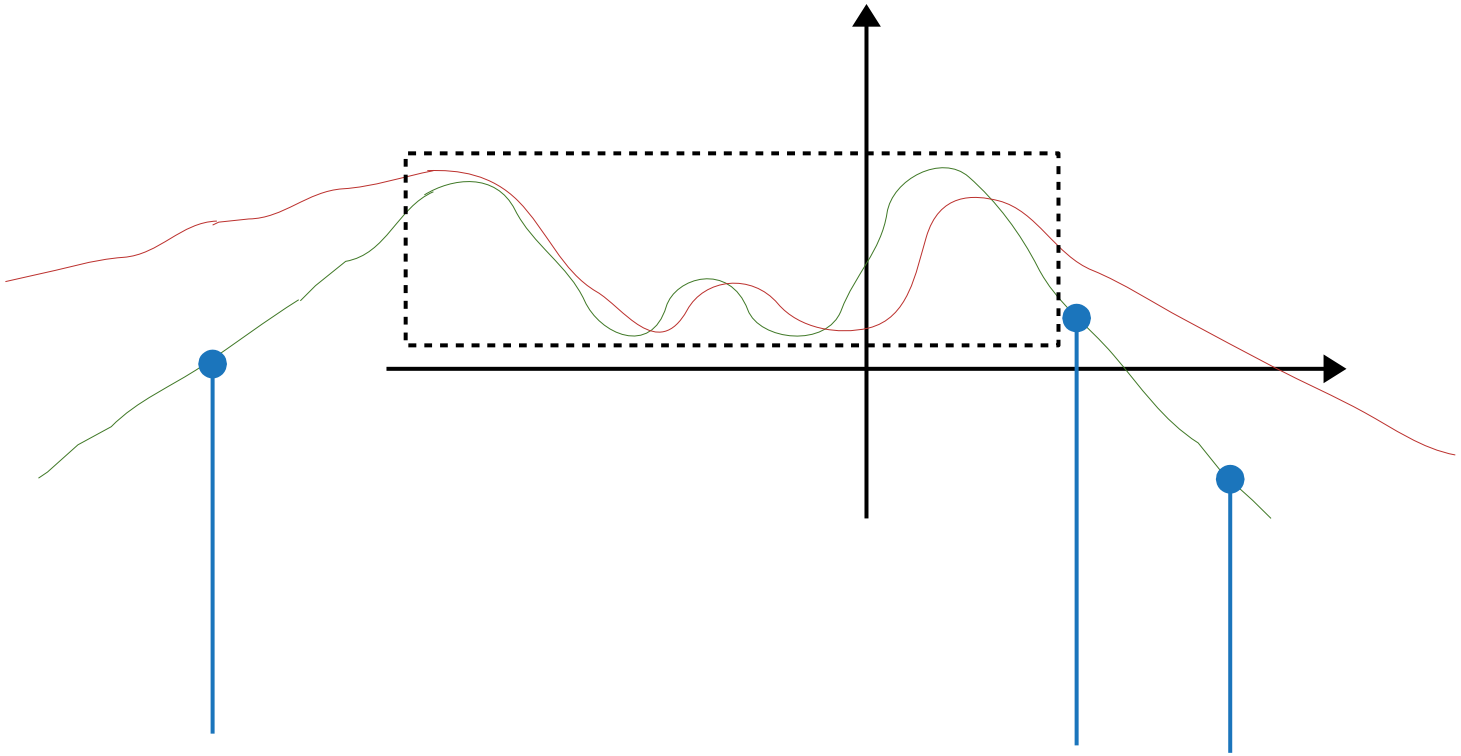
$$
\begin{aligned}
z_1 &= A * w \ (g_1) \\
z_2 &= z_1 * w \ (g_2 = A * w^2) \\
z_3 &= z_2 * w \ (g_3 = A * w^3) \\
z_4 &= B * w \ (g_4) \\
z_5 &= z_4 * w \ (g_5 = B * w^2) \\
z_6 &= C * w \ (g_6) \\
out &= (z_6 + D + z_5 + z_3) * 1 \ (g_7)
\end{aligned}
$$

Let $\vec{X} = (1, w, z1, z2, z3, z4, z5, z6, out)^\top$, and let (S') be the matrix representation of the system (S):

$$
\begin{aligned}
(A,0,0,0,0,0,0,0,0) \cdot \vec{X} * (0,1,0,0,0,0,0,0,0) \cdot \vec{X} &= (0,0,1,0,0,0,0,0,0) \cdot \vec{X} \ (g_1) \\
(0,0,1,0,0,0,0,0,0) \cdot \vec{X} * (0,1,0,0,0,0,0,0,0) \cdot \vec{X} &= (0,0,0,1,0,0,0,0,0) \cdot \vec{X} \ (g_2) \\
\dots &= \dots
\end{aligned}
$$

Now, we have a single check for our circuit **C**: $U \cdot \vec{X} * V \cdot \vec{X} \stackrel{?}{=} W \cdot \vec{X}$ or, equivalently $U \cdot \vec{X} * V \cdot \vec{X} - W \cdot \vec{X} \stackrel{?}{=} \vec{0}$.
To efficiently argue about the satisfiability of this system, we use polynomials interpolated from the matrices and a chosen domain. The check will now consist in checking if the Prover's polynomial vanish on the domain (ie: is a mulitple of a "vanishing polynomial")

Arithmetization

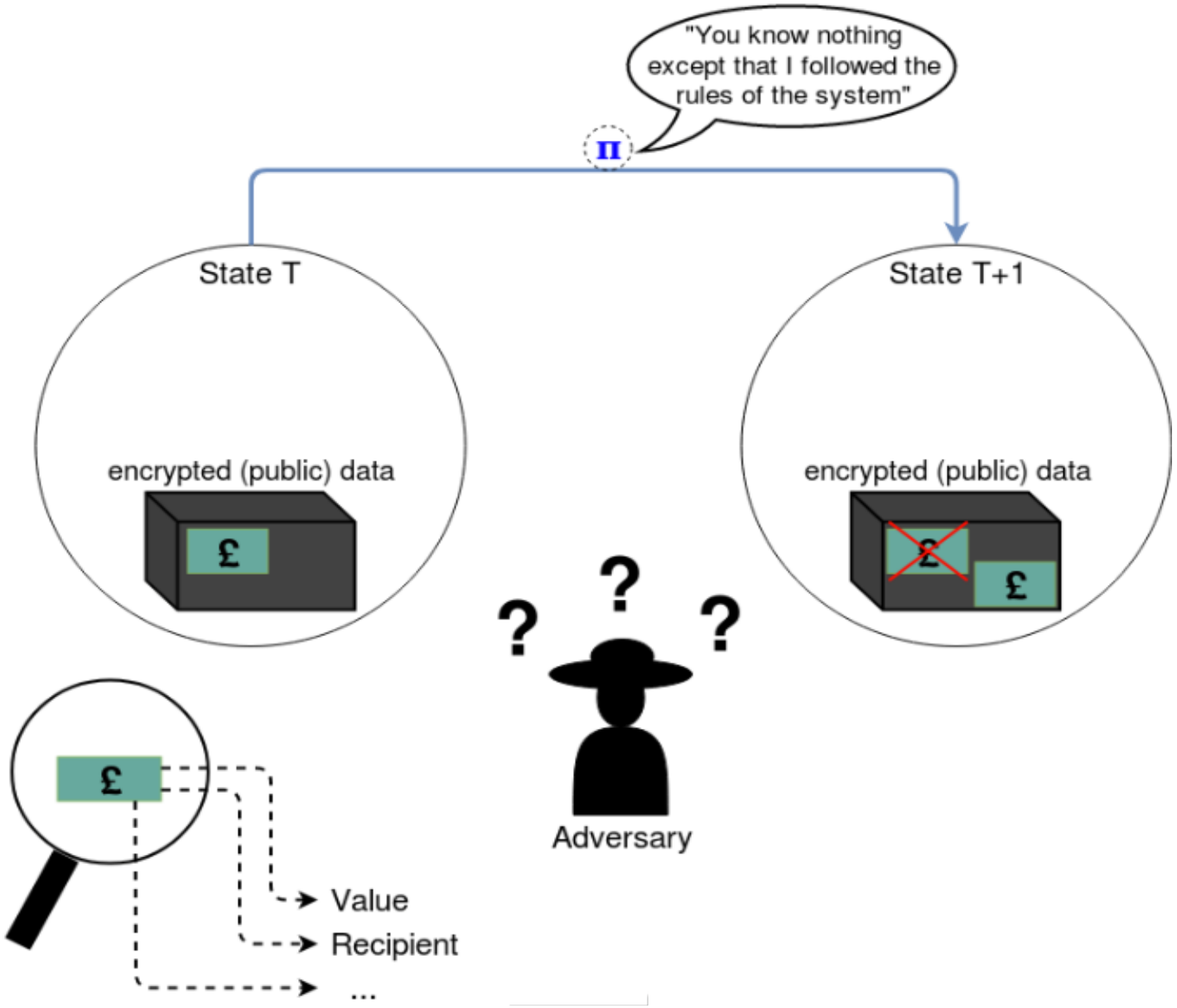


Query/"Probe" the polynomials of the Prover at random points ("toxic waste") to see if they meet the expected requirements. Schwartz-Zippel → if the Prover's polynomials diverge from the expected behavior, they'll be "far" from it and thus It will be "easy to detect".

- Convert the set of constraints representing your NP-statement to an equivalent polynomial p (over some field F) s.t. p has "certain properties" <=> the formula is satisfiable.
- Make sure the "query space" is big compared to the degree of the polynomials (ie: low degree checking for soundness)
- The queries should not be known in advance by the Prover (avoid adaptive behavior for soundness)
- Randomize your polynomials for zero-knowledge

Private payment protocol: Intuition

clearmatics



Zerocash

- Ben-Sasson et al. presented the Zerocash protocol [BCG+14] as a Decentralized Anonymous Payment (DAP) scheme working on top of Bitcoin.
- Deposit public funds in a “shielded pool” and *mint* the equivalent value of “obfuscated funds” represented by notes. These notes are maintained in a Merkle tree in an committed form. Consensus is reached over committed data.
- *“Burn/Destroy/Pour”* note**s** to create new one**s** (of potentially new owner!) such that the sum of the burnt notes = sum of the value of the new ones (“JoinSplit”).
- Uses zkSNARKs to generate proofs that payments are valid (namely, no double spent occurs in the transaction and no value is created “out of thin air”)
- **New payment semantics** added to be able to work on top of a Bitcoin-like system which require a hard-fork!

See this great talk at CESC2017 by A. Chiesa: https://www.youtube.com/watch?v=84Vbj7-i9CI

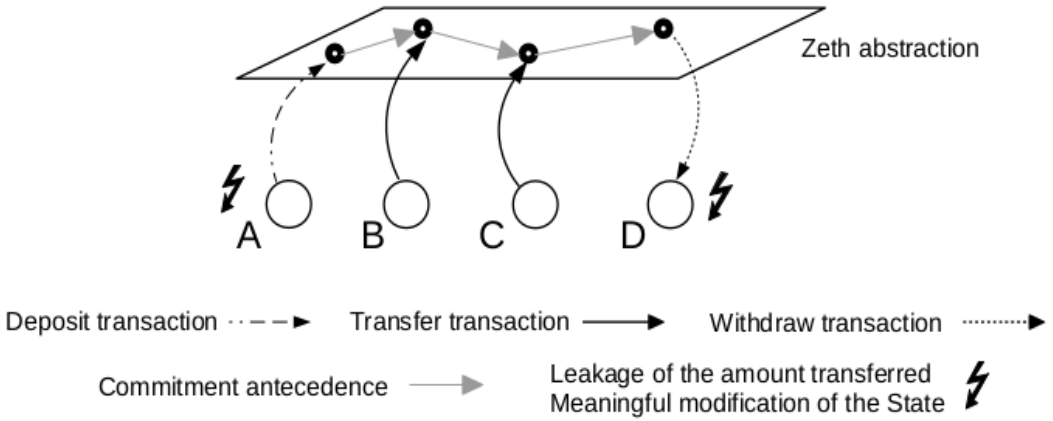
ZETH: Zerocash on Ethereum

clearmatics

- Follows the Zerocash construct and adapts the protocol to work on Ethereum
- Implements the **shielded pool on a smart contract**, and leverage Ethereum events to implement an encrypted broadcast mechanism that breaks the link between sender and recipient.
- Turing completeness of the EVM allows to encode any type of payment semantics. **No modifications needed on the protocol**

ZETH: Zerocash on Ethereum (2)

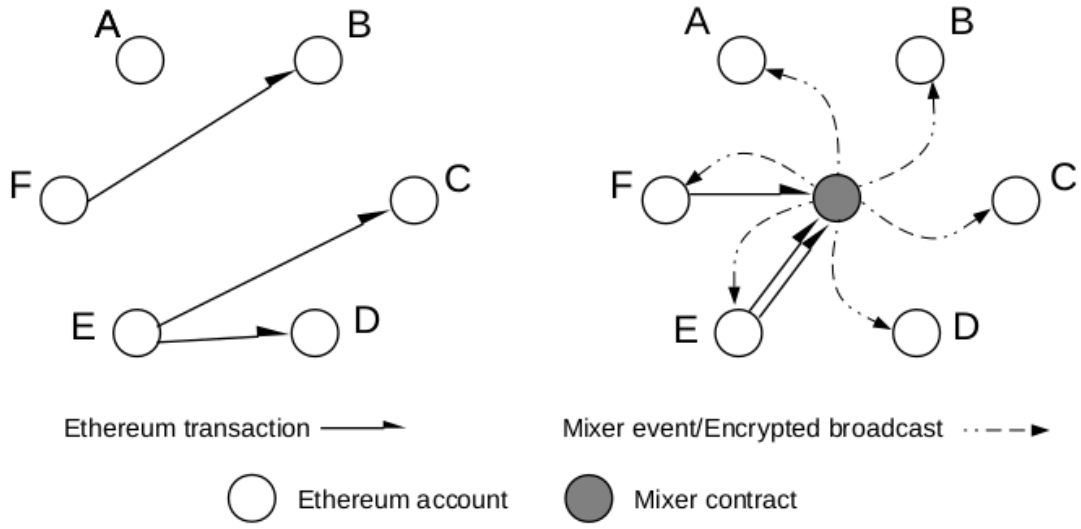
clearmatics



- The Zeth mixer contract provides a layer of abstraction/obfuscation “seating on top of Ethereum”
- **A single anonymity set for all possible denominations!** (To resist SCAs)

$$\mathsf{Mix}(\pi, \mathsf{rt}, v_{\mathsf{in}}, v_{\mathsf{out}}, \{\mathsf{cm}_i^{\mathsf{new}}\}_{i=1}^{M}, \{\mathsf{sn}_i\}_{i=1}^{N}, \{\mathsf{c}_i\}_{i=1}^{M})$$

- Sender (S) and recipient (R) never talk to each other. S encrypts the note with R’s public key and adds the ciphertext as argument to the “*Mix call*” of the mixer contract (assumption: IK-CCA Enc. Scheme)
- Pros: Resistance against HD failure
- Cons: Ciphertext stored on-chain



ZETH =/=> Anonymity

- Need to pay for the gas of the "*Mix*" function

  => **You can distinguish** between network users using Zeth and those who do not

  => If you know that an address belongs to a user, you can track when the user transacts via Zeth

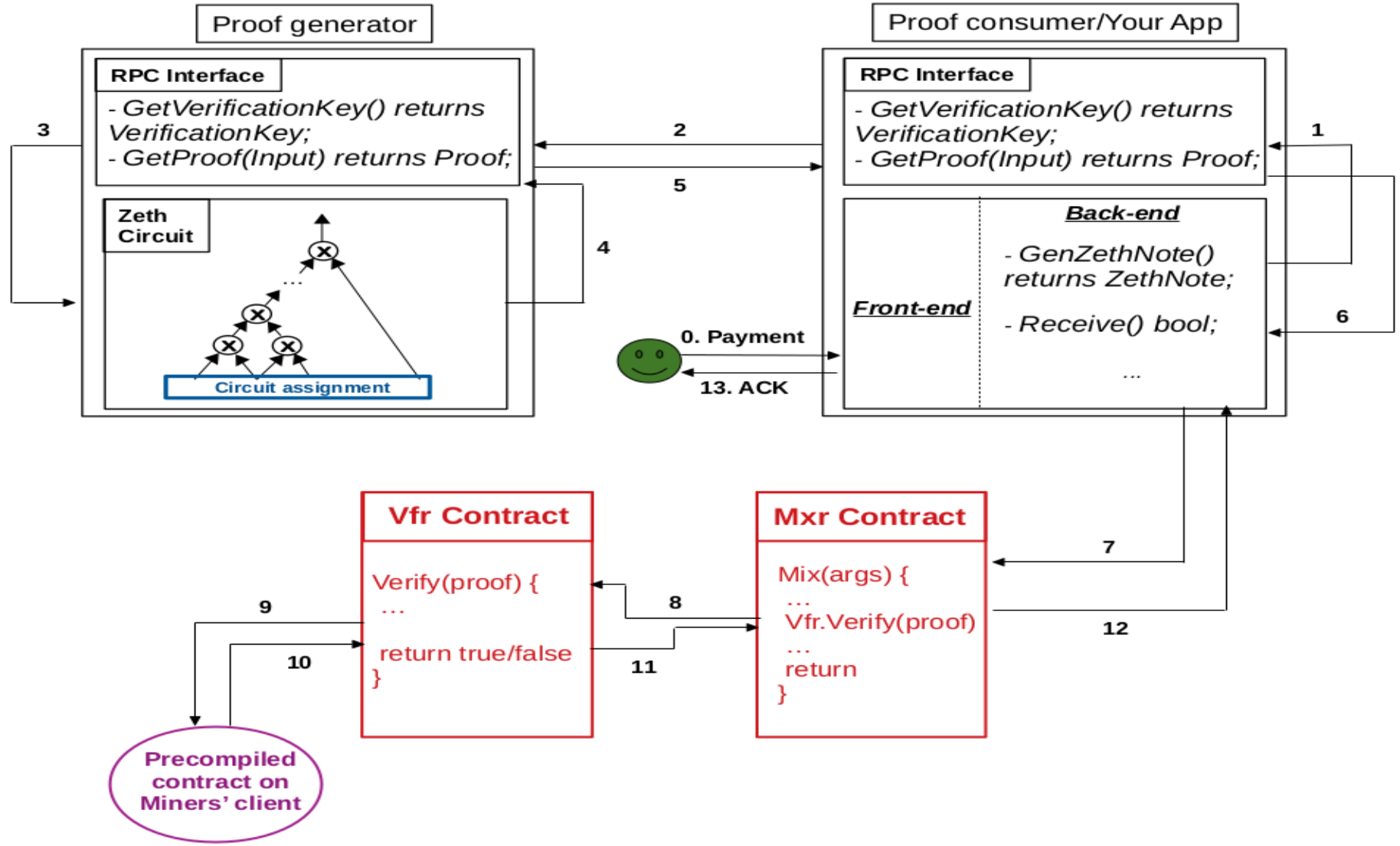
- How to get anonymity?
  - Relay services (relative "anonymity", need for more infrastructure (Tor/I2P, incentive structures...))
  - Changes to the Ethereum protocol - Account abstraction?
  - Be a miner

ZETH software (1)

- Follows a SoC approach
  - Designing secure protocols is hard. Producing secure code is even harder.
    - Isolate sensitive code from the rest (focus your attention/reviews on what matters and remove frictions)
    - Modular code base and easier to maintain/extend
- Unique circuit for all possible actions on the Mixer/Shielded pool
  - Unique statement (maybe more complex than individual statements for unique action), BUT
  - Unique SRS to generate – hence, unique MPC to carry out (you do not want to do multiple MPCs!)
- Client agnostic
  - Works with any client (request and receive proofs via RPC calls)
  - Easy to integrate in wallets
  - Ready to use for ETH2.0 :)

ZETH software (2)

clearmatics



Going further / Zeth is WIP

clearmatics

- MPC for the SRS generation
- Optimizations and security fixes
- Check out our online resources:
  - White-paper: https://arxiv.org/abs/1904.00905
  - Implementation: https://github.com/clearmatics/zeth

Happy to get any feedback and contribution!

ZETH: On Integrating Zerocash on Ethereum

Antoine Rondelet and Michal Zajac

Clearmatics, UK
{ar||michal.zajac}@clearmatics.com

Abstract. Transaction privacy is a hard problem on an account-based blockchain such as Ethereum. While Ben-Sasson et al. presented the Zerocash protocol [BCG+14] as a decentralized anonymous payment (DAP) scheme standing on top of Bitcoin, no study about the integration of such DAP on top of a ledger defined in the account model was provided. In this paper we aim to fill this gap and propose ZETH, an adaptation of Zerocash that can be deployed on top of Ethereum without making any change to the base layer. Our study shows that not only ZETH could be used to transfer Ether, the base currency of Ethereum, but it could also be used to transfer other types of smart contract-based digital assets. We propose an analysis of ZETH's privacy promises and argue that information leakages intrinsic to the use of this protocol are controlled and well-defined, which makes it a viable solution to support private transactions in the context of public and permissioned chains.

Keywords: Ethereum, Zerocash, Zcash, privacy, zero-knowledge proofs

Q&A

Thanks for your attention!

Any question?

@antoineRnd (Telegram)
@itranneo (Twitter)
https://keybase.io/antoinerondelet