HyperZEXE

Recursive HyperPlonk for fully function-private smart contract

Tianyi Liu Ye Zhang Yupeng Zhang **Zhenfei Zhang**

June 29, 2023

Zero knowledge proofs

Attest a statement while hidding some inputs of the statement

- Zcash: prove the validity of UTXO without leaking the ID info
- zkRollups: prove the soundness of a list of transactions
- zkBridge: prove the soundness of a list of transactions from another chain
- zkDID: prove who you are without revealing who you are
- zkOracle: attest historical data

Function Privacy

Execute a function with additional guarantees:

- The inputs and outputs to the function remain hidden.
  - Zcash: execute a Layer 1 transaction where sender/receiver and amount are hidden
  - Tornado cash: execute a smart contract transaction where sender/receiver and amount are hidden
- Above, and the function itself is also secret.
  - Aleo (earlier version of testnet): smart contract 1 is IND from SC 2

Function-private smart contract

Applications

- Distributed private computations.
- Miner-extractable values (MEV).
  - all info. w.r.t the smart contract are hidden, no MEV to extract.
- Plausible deniability.
  - Miners do not see if a smart contract is sanctioned.

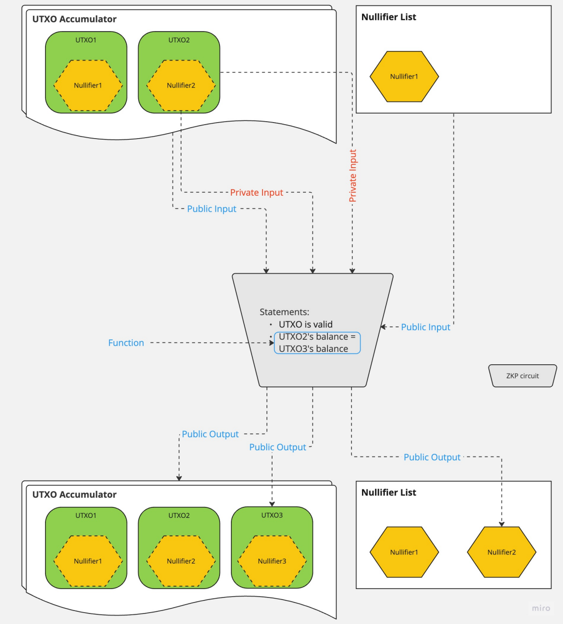
Function-private smart contract

Applications

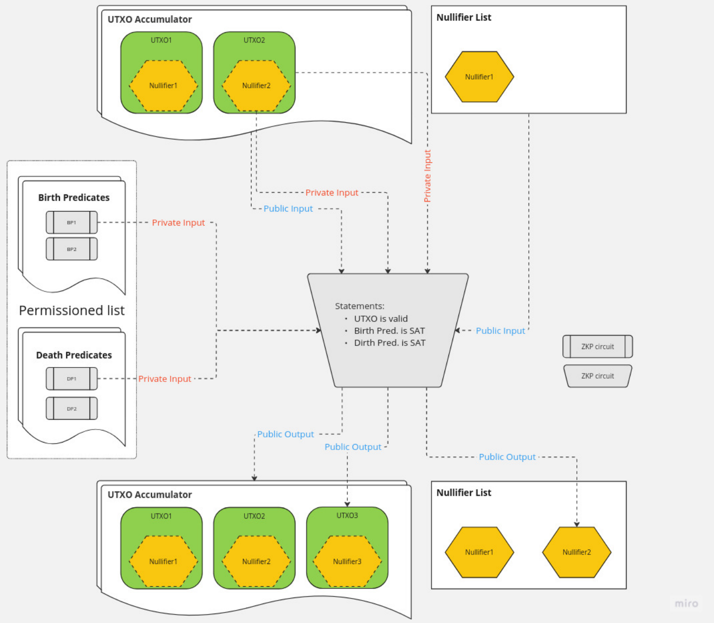
- Distributed private computations.
- Miner-extractable values (MEV).
  - all info. w.r.t the smart contract are hidden, no MEV to extract.
- Plausible deniability.
  - Miners do not see if a smart contract is sanctioned.

Bonus

- An efficient recursive prover.



UTXO Accumulator
UTXO1
Nullifier1
UTXO2
Nullifier2
Nullifier List
Nullifier1
Private Input
Public Input
Private Input
Statements:
UTXO is valid
UTXO2's balance = UTXO3's balance
Function
Public Input
ZKP circuit
Public Output
Public Output
Public Output
UTXO Accumulator
UTXO1
Nullifier1
UTXO2
Nullifier2
UTXO3
Nullifier3
Nullifier List
Nullifier1
Nullifier2
miro



ethereum foundation Scroll TEXAS A&M UNIVERSITY

Recursive proof

- Input a proof π_1 that is valid w.r.t. verification key vk
- Generate a new proof π_2 asserting $\mathtt{verify}(\pi_1, vk) == 1$

Recursive proof

- Input a proof π_1 that is valid w.r.t. verification key vk
- Generate a new proof π_2 asserting $\texttt{verify}(\pi_1, vk) == 1$

EC-based provers: $E : y^2 = x^3 + b \bmod q$

- Proves relations over the scalar field $\mathbb{F}_{|\mathbb{G}|}$
- Produces a proof over the base field $\mathbb{F}_q$

Recursive proof

- Input a proof π_1 that is valid w.r.t. verification key vk
- Generate a new proof π_2 asserting $\texttt{verify}(\pi_1, vk) == 1$

EC-based provers: $E : y^2 = x^3 + b \bmod q$

- Proves relations over the scalar field $\mathbb{F}_{|\mathbb{G}|}$
- Produces a proof over the base field $\mathbb{F}_q$

EC1: Two chain proofs

- Two curves `CurveA` and `CurveB`
- `CurveA::BaseField` = `CurveB::ScalarField`
- e.g.: `ZEXE`

ethereum foundation Scroll TEXAS A&M UNIVERSITY

Recursive proof

- Input a proof π_1 that is valid w.r.t. verification key vk
- Generate a new proof π_2 asserting $\texttt{verify}(\pi_1, vk) == 1$

EC-based provers: $E : y^2 = x^3 + b \bmod q$

- Proves relations over the scalar field $\mathbb{F}_{|\mathbb{G}|}$
- Produces a proof over the base field $\mathbb{F}_q$

EC2: Cyclic curves

- Two curves `CurveA` and `CurveB`
- `CurveA::BaseField` = `CurveB::ScalarField` and `CurveB::BaseField` = `CurveA::ScalarField`
- e.g.: `Halo2-Pasta`, `Nova`, etc.

Recursive proof

- Input a proof π_1 that is valid w.r.t. verification key vk
- Generate a new proof π_2 asserting $\texttt{verify}(\pi_1, vk) == 1$

EC-based provers: $E : y^2 = x^3 + b \bmod q$

- Proves relations over the scalar field $\mathbb{F}_{|\mathbb{G}|}$
- Produces a proof over the base field $\mathbb{F}_q$

EC3: Non-native arithmetics

- Single Curve `BN254`
- Use $\mathbb{F}_{|\mathbb{G}|}$ to emulate $\mathbb{F}_q$
- Penalty: 30× larger circuit (`Halo2-lib`)
- e.g.: zkEVM via `Halo2-KZG`

Recursive proof

- Input a proof π_1 that is valid w.r.t. verification key vk
- Generate a new proof π_2 asserting $\texttt{verify}(\pi_1, vk) == 1$

Code based provers

- Relation and proof uses a same field
- FRI, Breakdown, etc...

ZEXE paradigm

	Inner Prover		Outer Prover	
	Scheme	Curve	Scheme	Curve
`ZEXE`	Groth16	BLS12-377	Groth16	CP6-782
`SnarkVM`	Marlin	BLS12-377	Groth16	BW6-761
`VeriZEXE`	TurboPlonk	BLS12-377	UltraPlonk	BW6-761

Table: 2-Chain recursive proof systems in ZEXE

ethereum foundation Scroll TEXAS A&M UNIVERSITY

ZEXE paradigm

	Inner Prover		Outer Prover	
	Scheme	Curve	Scheme	Curve
`ZEXE`	Groth16	BLS12-377	Groth16	CP6-782
`SnarkVM`	Marlin	BLS12-377	Groth16	BW6-761
`VeriZEXE`	TurboPlonk	BLS12-377	UltraPlonk	BW6-761

Table: 2-Chain recursive proof systems in ZEXE

- Plonk arithmetization is $10 \sim 30\times$ more expressive than R1CS

ZEXE on-chain

Challenges 1

- The outer proof has to be on BN254 curve
- Ethereum does not support other popular ZK-friendly curves or fields
  - BN254 curve group mul: 6K Gas
  - Pasta curves group mul: 3M Gas
  - BW6-761 curve group mul: ??? Gas

ZEXE on-chain

Challenges 1

- The outer proof has to be on BN254 curve
- Ethereum does not support other popular ZK-friendly curves or fields
  - BN254 curve group mul: 6K Gas
  - Pasta curves group mul: 3M Gas
  - BW6-761 curve group mul: ??? Gas

Solution 1

- Use `Grumpkin` $\Longleftrightarrow$ BN254 cyclic curves

ZEXE on-chain

Challenges 2

- `Grumpkin` does not support FFT
- Groth16 and Plonk require FFT

ZEXE on-chain

Challenges 2

- `Grumpkin` does not support FFT
- Groth16 and Plonk require FFT

Solution 2

- Use an FFT-free prover
- Candidates:
  - Nova (R1CS)
  - HyperPlonk

ZEXE on-chain

Challenges 3

- `Grumpkin` does not support pairing
- ML-KZG commitment requires pairing
- Other commitment schemes are less verifier friendly

ZEXE on-chain

Challenges 3

- `Grumpkin` does not support pairing
- ML-KZG commitment requires pairing
- Other commitment schemes are less verifier friendly

Solution 3

- Use Hyrax, verifier does $2\sqrt{n}$ group muls
- Or IPA, verifier does n group muls (deferred and aggregated)

ZEXE paradigm

	Inner Prover		Outer Prover	
	Scheme	Curve	Scheme	Curve
`ZEXE`	Groth16	BLS12-377	Groth16	CP6-782
`SnarkVM`	Marlin	BLS12-377	Groth16	BW6-761
`VeriZEXE`	TurboPlonk	BLS12-377	UltraPlonk	BW6-761
`HyperZEXE`	HyperPlonk	Grumpkin	UltraPlonk	BN254

Table: 2-Chain recursive proof systems in ZEXE

Concrete efficiency

- vs non-native `Halo2-KZG`: saves $30\times$ in # constraints
- vs (`Veri`)ZEXE: saves $5\times$ due to smaller field (254 bits vs 761 bits)
- and more ...

Native ECAdd: $(x_3, y_3) := (x_1, y_1) + (x_2, y_2)$

(Veri)ZEXE: prove Short Weierstrass formula directly

- $x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$
- $y_3 = \frac{(2x_1 + x_2)(y_2 - y_1)}{x_2 - x_1} - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^3 - y_1$

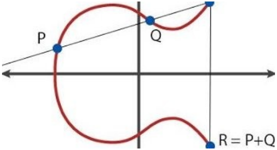
Native ECAdd: $(x_3, y_3) := (x_1, y_1) + (x_2, y_2)$

(Veri)ZEXE: prove Short Weierstrass formula directly

- $x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2$
- $y_3 = \frac{(2x_1 + x_2)(y_2 - y_1)}{x_2 - x_1} - \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^3 - y_1$

HyperZEXE

- (x_3, y_3) is on curve: $y_3^2 = x_3^3 + b$
- (x_1, y_1), (x_2, y_2) and $(x_3, -y_3)$ are on the same line: $(x_1 - x_3)(y_2 + y_3) = (x_2 - x_3)(y_1 + y_3)$
- $(x_1, y_1)! = (x_3, -y_3)$ and $(x_2, y_2)! = (x_3, -y_3)$



Native Double: $(x_2, y_2) := (x_1, y_1) + (x_1, y_1)$

`(Veri)ZEXE`: prove Short Weierstrass formula directly

- $x_2 = \left(\frac{3x_1}{2y_1}\right)^2 - 2x_1$
- $y_2 = \frac{9x_1^3}{2y_1} - \left(\frac{3x^2}{2y_1}\right)^3 - y_1$

Native Double: $(x_2, y_2) := (x_1, y_1) + (x_1, y_1)$

(Veri)ZEXE: prove Short Weierstrass formula directly

- $x_2 = \left(\frac{3x_1}{2y_1}\right)^2 - 2x_1$
- $y_2 = \frac{9x_1^3}{2y_1} - \left(\frac{3x^2}{2y_1}\right)^3 - y_1$

HyperZEXE

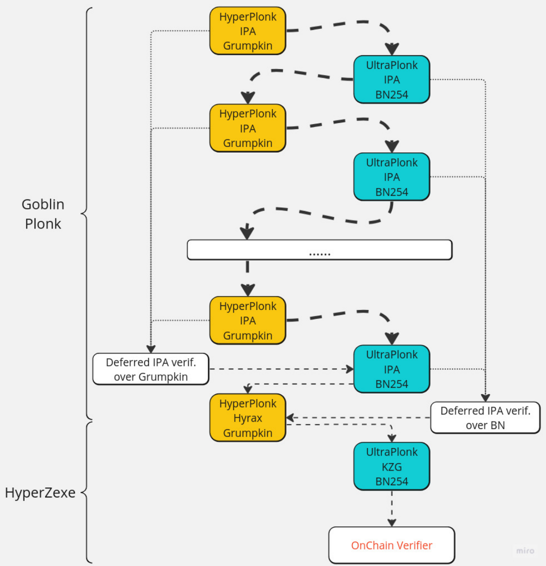
- (x_2, y_2) is on curve: $y_2^2 = x_2^3 + b$
- (x_1, y_1) and $(x_2, -y_2)$ are on a tangential line of the curve $\frac{x_1 - x_2}{y_1 + y2} = \frac{3x_1^2}{2y_1}$
- $(x_1, y_1)! = (x_2, -y_2)$

HyperZEXE's custom gate

OpCodes	Advices		Selectors			
	w_1	w_2	q_{ecc}	q_1	q_2	q_3
On Curve	a_0	b_0	1	0	0	1
EC double	a_1 a_2	b_1 b_2	1	0	1	0
Conditional EC Add	a_3 a_4 cond a_5	b_3 b_4 _ b_5	1	1	0	0

- Custom gate degree: 5 (c.f. 6 for `VeriZEXE` on SW curve)
- Total witness cells per EC mul: 2442 cells
  - c.f., 9325 cells in `VeriZEXE`
  - Further reduced by 5$\times$ via Pippenger and lookups

Beyond ZEXE: infinity recursion for Ethereum applications


Goblin
Plonk
HyperZexe
HyperPlonk
IPA
Grumpkin
UltraPlonk
IPA
BN254
HyperPlonk
IPA
Grumpkin
UltraPlonk
IPA
BN254
......
HyperPlonk
IPA
Grumpkin
Deferred IPA verif.
over Grumpkin
UltraPlonk
IPA
BN254
HyperPlonk
Hyrax
Grumpkin
Deferred IPA verif.
over BN
UltraPlonk
KZG
BN254
OnChain Verifier
miro

Use HyperZEXE as a recursive prover

Target: generate a recursive proof for zkEVM

- Typical circuit size: 2^{20} rows and ≈ 500 columns

Use HyperZEXE as a recursive prover

Target: generate a recursive proof for zkEVM

- Typical circuit size: 2^{20} rows and ≈ 500 columns

baseline

- Single layer, non-native halo2-KZG: BN254 $\rightarrow$ BN254
- Cost: $2^{25} \times 20 \approx 640M$ cells

Use HyperZEXE as a recursive prover

Target: genererate a recursive proof for zkEVM

- Typical circuit size: 2^{20} rows and ≈ 500 columns, or 2^{29} witness cells

HyperZEXE first layer: BN254 $\rightarrow$ Grumpkin

- verify π_1 dominated by batch verifying 500 KZG openings
- requires roughly 1000 ECMULs, or $\approx 2^{19}$ witness cells
- generate a proof π_2 with Hyrax commitment

HyperZEXE second layer: Grumpkin $\rightarrow$ BN254

- verify π_2 dominated by batch verifying 2 hyrax openings
- requires roughly $2 \times \sqrt{2^{19}} = 2^{11}$ ECMULs, or $\approx 2^{20}$ witness cells

	Inner Prover		Outer Prover		Prover time	OnChain Verifier
	Scheme	Setup	Scheme	Setup		
ZEXE	Groth16	Trusted	Groth16	Trusted		
SnarkVM	Marlin	Universal	Groth16	Trusted	150 s	N/A
VeriZEXE	TurboPlonk	Universal	UltraPlonk	Universal	13 s	N/A
HyperZEXE	HyperPlonk	Transparent	UltraPlonk	Universal	$<$ 1s	450K Gas

Progress

- ✓HyperPlonk with Hyrax commitment
- ✓Optimized native-ECC custom gate
- ✓Solidity onchain verifier
- × UltraPlonk verifier circuit
- × HyperPlonk verifier circuit