zkEVM: Zero-Knowledge Ethereum Virtual Machine

Tal Derei

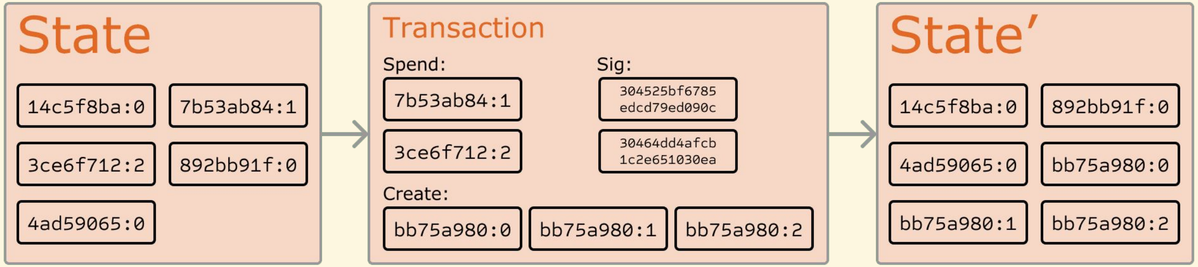
Terms

- **L1** = Layer-1 (Main Chain)
- **L2** = Layer-2 (ZK-Rollups)
- **ZK** = Zero-Knowledge
- **zk-SNARKS** = Zero-Knowledge Succinct Non-Interactive Argument of Knowledge Proofs

Background

- Ethereum = Transaction-Based State Machine
  - EVM traverses blockchain starting from genesis block

- EVM = Stack Machine
  - Processes transactions

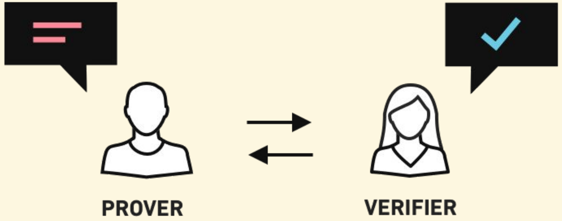


Problem?

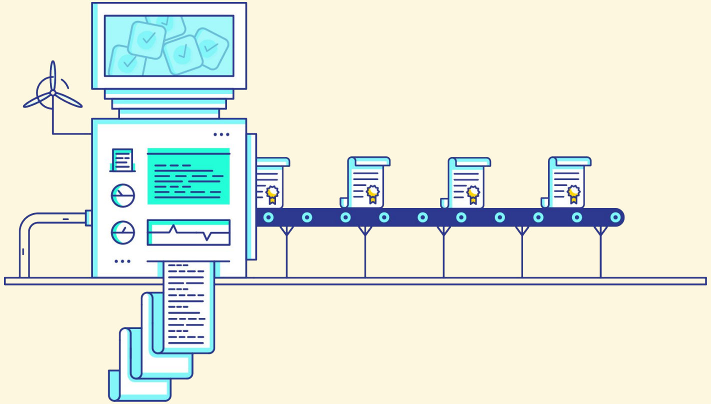
ZK-Rollups:

- Generate zero knowledge-proofs on L2
- Pass back proof on L1 for verification
- ZK proofs (and the EVM) need to conform to zk-circuit proof specifications

…And the problem is the EVM wasn't designed with zero-knowledge in mind!



What is zkEVM?



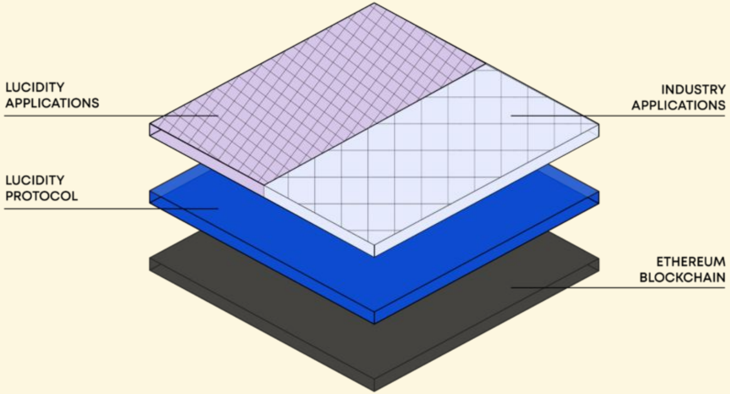
zkEVM is the ***key*** to scaling Ethereum blockchain in the future!

Vitalek Buterin: “In the medium to long term, zk-rollups will win out in all use cases over Optimistic Rollups as ZK-SNARK technology improves”

What is zkEVM?

L2 ZK-Rollup for Payments and **Generic Smart Contracts**!

EVM on L2!



What is zkEVM?

zkEVM is a “**A turing-complete virtual machine that executes smart contracts on a zk-Rollup (Layer-2) network, is EVM-compatible and zero-knowledge (SNARK) friendly**”’

- Key to building ZK-Rollups compatible with the EVM
  - Easily port DAPs and DAOs written in solidity on L2

- zkEVM keeps EVM semantics (e.g. gas fee structure and security properties of the main-chain)

- Based on traditional CPU architectures

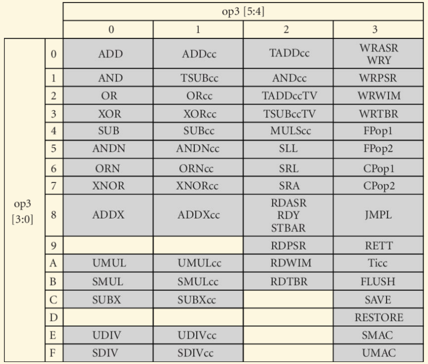
Competitors

- **Hermez**: L2 ZK-Rollup based on SNARKs
- **zkSync 2.0**: L2 ZK-Rollup based on SNARKs
- **Starkware**: L2 ZK-Rollup based on STARKs
- **Loopring**: L2 ZK-Rollup protocol for DEXs
- **Ethereum Foundation**

Implementations

- Hermez (by **Polygon**)
  - Opcode-Based Implementation
    - Implement full set of EVM opcodes DIRECTLY in ZK-Rollup
    - Uses the **SAME** EVM compiler as L1 under the hood

- zkSync 2.0 (by **Matter-Labs**)
  - Compiler-Based Implementation
    - Building a newly designed EVM to maintain solidity compatibility

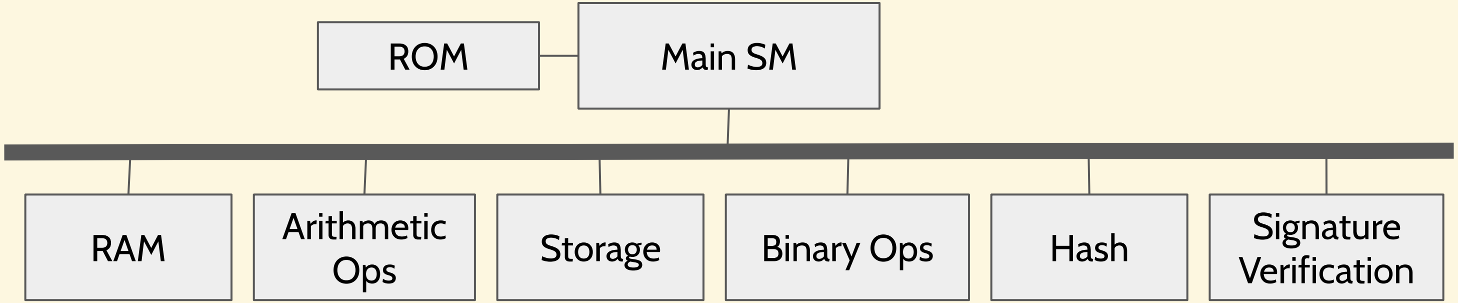


		op3 [5:4]			
		0	1	2	3
op3 [3:0]	0	ADD	ADDcc	TADDcc	WRASR WRY
	1	AND	TSUBcc	ANDcc	WRPSR
	2	OR	ORcc	TADDccTV	WRWIM
	3	XOR	XORcc	TSUBccTV	WRTBR
	4	SUB	SUBcc	MULScc	FPop1
	5	ANDN	ANDNcc	SLL	FPop2
	6	ORN	ORNcc	SRL	CPop1
	7	XNOR	XNORcc	SRA	CPop2
	8	ADDX	ADDXcc	RDASR RDY STBAR	JMPL
	9			RDPSR	RETT
	A	UMUL	UMULcc	RDWIM	Ticc
	B	SMUL	SMULcc	RDTBR	FLUSH
	C	SUBX	SUBXcc		SAVE
	D				RESTORE
	E	UDIV	UDIVcc		SMAC
	F	SDIV	SDIVcc		UMAC

Hermez

OPCODE-BASED APPROACH

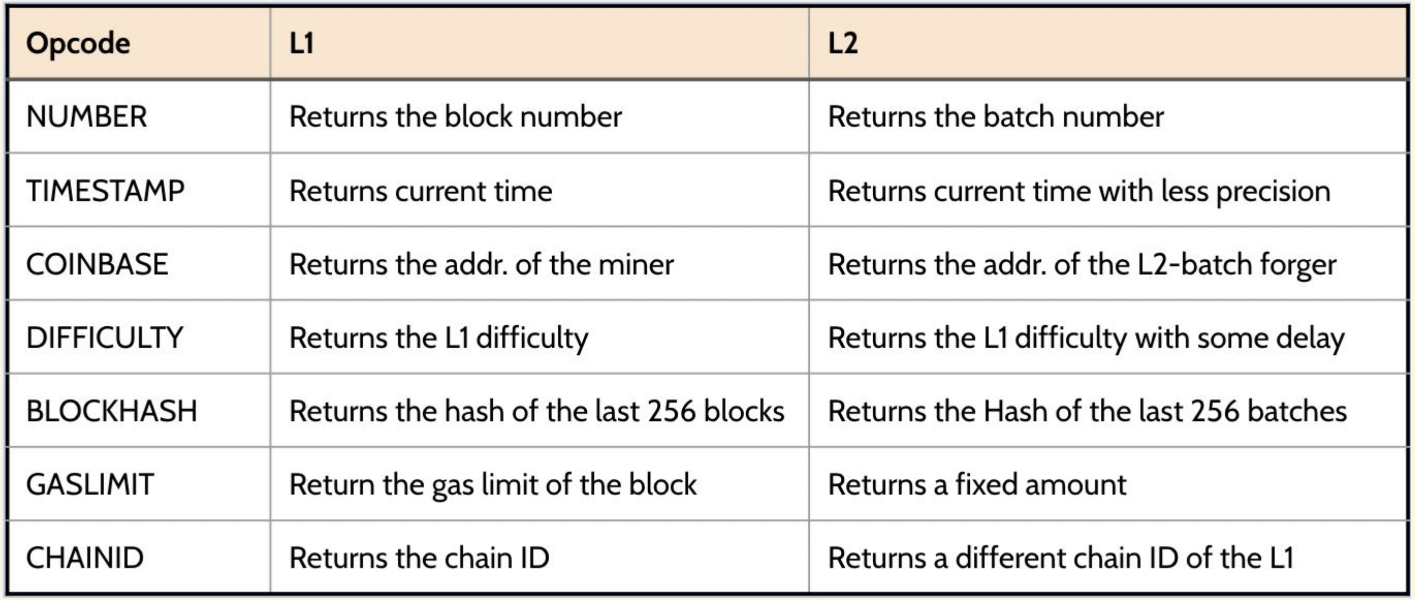
- Translates the entire set of EVM opcodes into micro opcodes
  - Micro opcodes interpreted in special VM (uVm)



polygon

EVM Opcodes

L1 EVM-Specific Opcodes



Opcode	L1	L2
NUMBER	Returns the block number	Returns the batch number
TIMESTAMP	Returns current time	Returns current time with less precision
COINBASE	Returns the addr. of the miner	Returns the addr. of the L2-batch forger
DIFFICULTY	Returns the L1 difficulty	Returns the L1 difficulty with some delay
BLOCKHASH	Returns the hash of the last 256 blocks	Returns the Hash of the last 256 batches
GASLIMIT	Return the gas limit of the block	Returns a fixed amount
CHAINID	Returns the chain ID	Returns a different chain ID of the L1

Hermez

OPCODE-BASED APPROACH

- Bottom-Up Construction: each opcode needs to be reconstructed for L2
  - ~50 opcodes
  - Some opcodes more complex and expensive to compute on L2
    - (e.g. SHA256 or Keccak Hash Functions)

- Guarantees
  - Ethereum-level security guaranteed by zkSNARKs
  - Throughput of approximately 2000 TPS

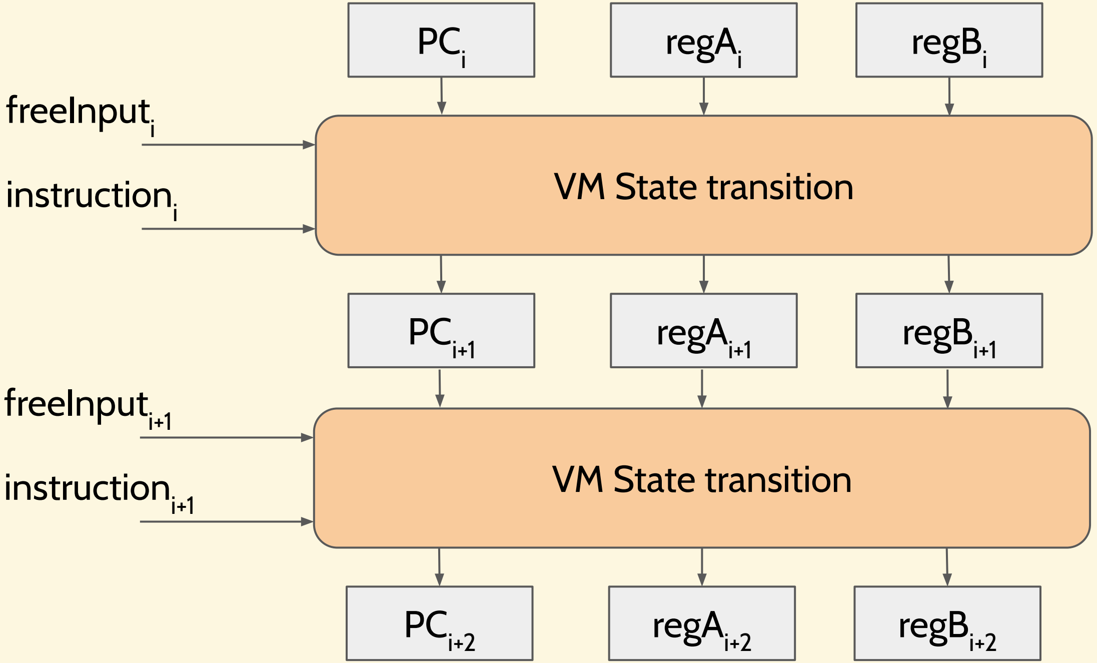
polygon

Proof Generation

Generating zkSNARK Proof

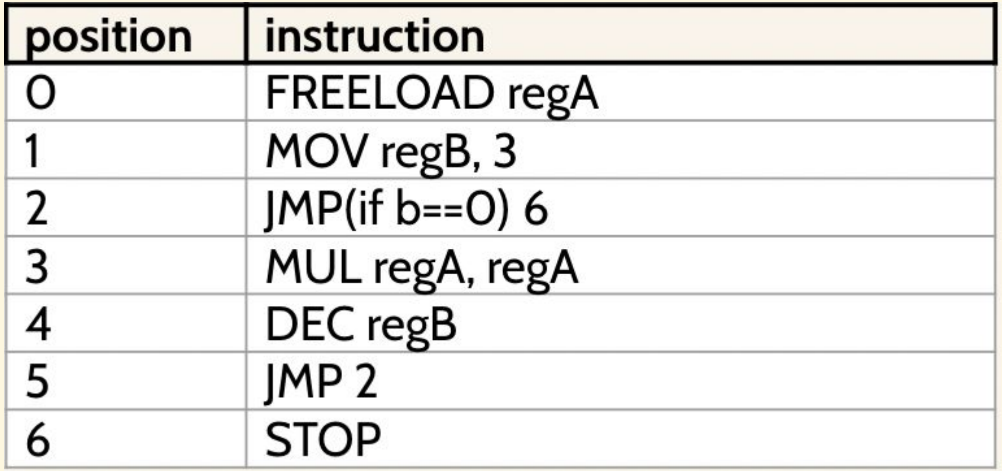
State Machine

The main state machine of a simple VM



Program

Program Opcodes



position	instruction
0	FREELOAD regA
1	MOV regB, 3
2	JMP(if b==0) 6
3	MUL regA, regA
4	DEC regB
5	JMP 2
6	STOP

polygon

Compilation

Program Compilation

FREELOAD regA	->	0x00010000
MOV regB, <n>	->	0x00020000 + n
JMP(if b==0) <n>	->	0x00040000 + n
JMP <n>	->	0x00080000 + n
MUL regA, regA	->	0x00100000
DEC regB	->	0x00200000
STOP	->	0x00400000

Field Elements

position	instruction	inst
0	FREELOAD regA	0x00 0000
1	MOV regB, 3	0x00 20003
2	JMP(if b==0) 6	0x00 040006
3	MUL regA, regA	0x00100000
4	DEC regB	0x00200000
5	JMP 2	0x00080002
6	STOP	0x00400000

State Trace

Main State Machine Trace

step	instruction	inst	freeLoad	PC$_i$	regA$_i$	regB$_i$	PC$_{i+1}$	regA$_{i+1}$	regB$_{i+1}$
0	FREELOAD regA	0x00010000	44	0	0	0	1	44	0
1	MOV regB, 3	0x00020003	0	1	44	0	2	44	3
2	JMP(if b==0) 6	0x00040006	0	2	44	3	3	44	3
3	MUL regA, regA	0x00100000	0	3	44	3	4	1936	3
4	DEC regB	0x00200000	0	4	1936	3	5	1936	2
5	JMP 2	0x00080002	0	5	1936	2	2	1936	2
6	JMP(if b==0) 6	0x00040006	0	2	1936	2	3	1936	2
7	MUL regA, regA	0x00100000	0	3	1936	2	4	85184	2
8	DEC regB	0x00200000	0	4	85184	2	5	85184	1
9	JMP 2	0x00080002	0	5	85184	1	2	85184	1
10	JMP(if b==0) 6	0x00040006	0	2	85184	1	3	85184	1
11	MUL regA, regA	0x00100000	0	3	85184	1	4	3748096	1
12	DEC regB	0x00200000	0	4	3748096	1	5	3748096	0
13	JMP 2	0x00080002	0	5	3748096	0	2	3748096	0
14	JMP(if b==0) 6	0x00040006	0	2	3748096	0	6	3748096	0
15	STOP	0x00400000	0	6	3748096	0	6	3748096	0

Zero-Knowledge Proofs on Hermez

Now we relate opcodes to Zero-Knowledge!

Polynomials

Polynomial representations:

Coefficients representation

$$p(x) = a_0 + a_1x + a_2x^2 \ldots. + a_nx^n$$

Evaluation representation

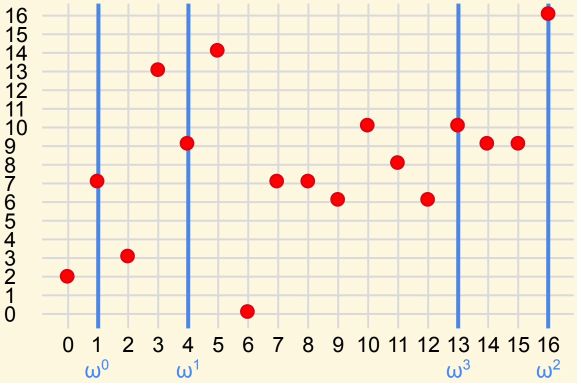
$$p(x) = A_0L_0(x) + A_1L_1(x) + A_2L_2(x) + \ldots. + A_nL_n(x)$$

$$\begin{tikzcd} \lbrack a_0, a_1, a_2, \ldots. , a_n \rbrack \arrow[r, shift left, "\text{fft}"] & \langle A_0, A_1, A_2, \ldots. , A_n \rangle \arrow[l, shift left, "\text{fft}^{-1}"] \end{tikzcd}$$

Numerical Example

x	f(x)
0	2
1 = ω^0	7
2	3
3	13
4 = ω^1	9
5	14
6	0
7	7
8	7
9	6
10	10
11	8
12	6
13 = ω^3	10
14	9
15	9
16 = ω^2	16

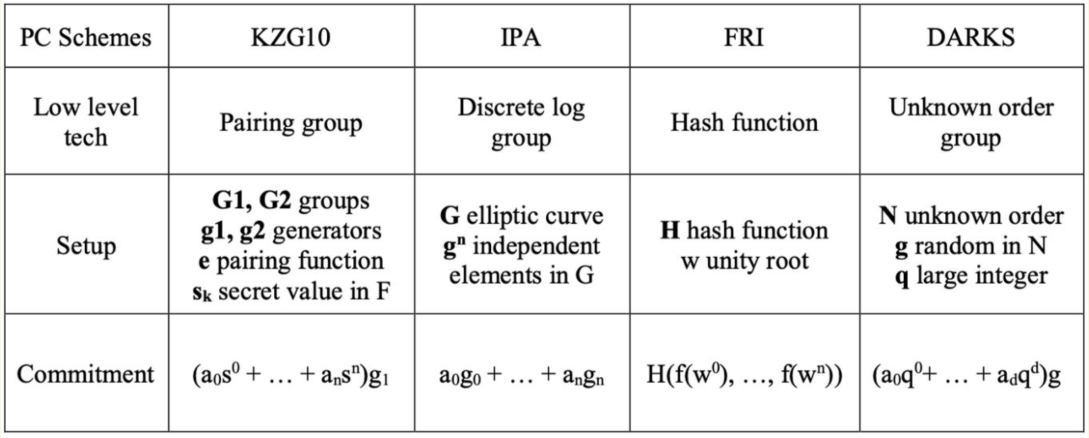
$f(x) = 2 + 3x + x^2 + x^3$



Coefficients representation: [2,3,1,1]

Polynomial Commitments

- **Polynomial Commitments** allow a prover to publish a public value / statement (*commitment*), while keeping it hidden to others (*hiding*), and ability to reveal the committed value later



PC Schemes	KZG10	IPA	FRI	DARKS
Low level tech	Pairing group	Discrete log group	Hash function	Unknown order group
Setup	**G1, G2** groups **g1, g2** generators **e** pairing function $\mathbf{s_k}$ secret value in F	**G** elliptic curve $\mathbf{g^n}$ independent elements in G	**H** hash function w unity root	**N** unknown order **g** random in N **q** large integer
Commitment	$(a_0s^0 + \ldots + a_ns^n)g_1$	$a_0g_0 + \ldots + a_ng_n$	$H(f(w^0), \ldots, f(w^n))$	$(a_0q^0 + \ldots + a_dq^d)g$

Different Commitment Schemes for different ZKPs:

FRI-Proofs

Hermes uses FRI-based zkZSNARKs

- **FRI-Proofs**
  - Verify all the polynomial commitments based on hash functions

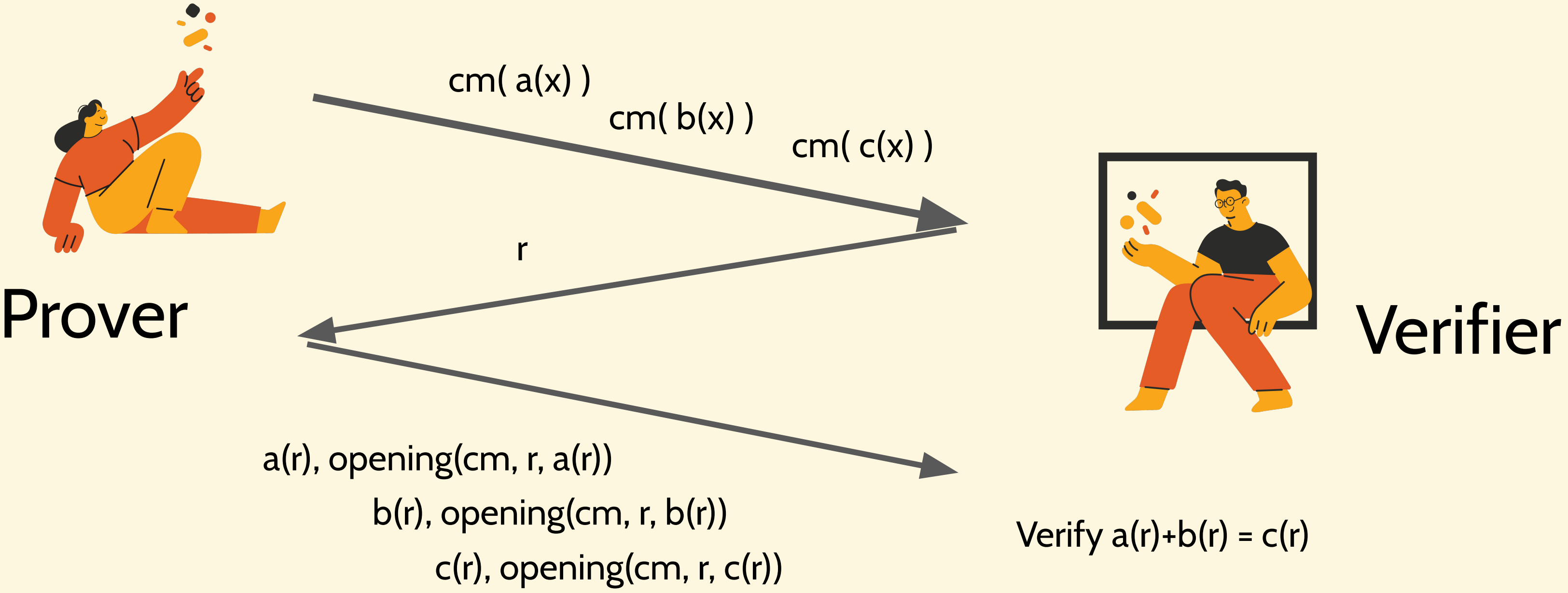
Polynomial Commitment

→ prover commits to certain polynomial P (bind original message with public polynomial)

→ prover proves value of polynomial at certain point Z satisfies P(z) through proof without revealing the polynomial

$$\mathbf{P(Z) = z}$$

Polynomial Protocol



cm(a(x))

cm(b(x))

cm(c(x))

r

Prover

Verifier

a(r), opening(cm, r, a(r))

b(r), opening(cm, r, b(r))

c(r), opening(cm, r, c(r))

Verify a(r)+b(r) = c(r)

Polynomial Relationships

There exists **verifiable relationships** between polynomials!

Program Verification

position(x)	instruction	inst(x)	rom(x) = 2^{16}·position(x) + inst(x)
0	FREELOAD regA	0x00010000	0x000000010000
1	MOV regB, 3	0x00020003	0x000100020003
2	JMP(if b==0) 6	0x00040006	0x000200040006
3	MUL regA, regA	0x00100000	0x000300100000
4	DEC regB	0x00200000	0x000400200000
5	JMP 2	0x00080002	0x000500080002
6	STOP	0x00400000	0x000600400000

Use Plookup to verify that:

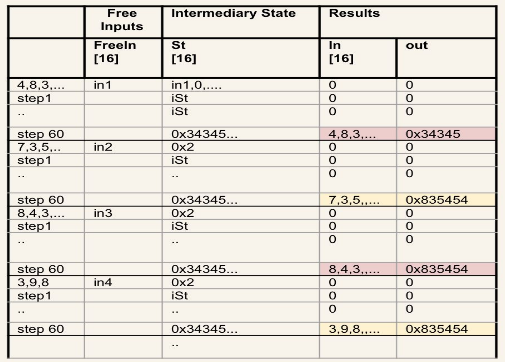
instTrace(x) ⊂ rom(x)

step(x)	instruction	inst(x)	freeLoad(x)	PC(x)	regA(x)	regB(x)	PC_{i+1}	$regA_{i+1}$	$regB_{i+1}$	instTrace(x)= 2^{16}PC(x)+ inst(x)
0	FREELOAD regA	0x00010000	44	0	0	0	1	44	0	0x000000010000
1	MOV regB, 3	0x00020003	0	1	44	0	2	44	3	0x000100020003
2	JMP(if b==0) 6	0x00040006	0	2	44	3	3	44	3	0x000200040006
3	MUL regA, regA	0x00100000	0	3	44	3	4	1936	3	0x000300100000
4	DEC regB	0x00200000	0	4	1936	3	5	1936	2	0x000400200000
5	JMP 2	0x00080002	0	5	1936	2	2	1936	2	0x000500080002
6	JMP(if b==0) 6	0x00040006	0	2	1936	2	3	1936	2	0x000200040006
7	MUL regA, regA	0x00100000	0	3	1936	2	4	85184	2	0x000300100000
8	DEC regB	0x00200000	0	4	85184	2	5	85184	1	0x000400200000
9	JMP 2	0x00080002	0	5	85184	1	2	85184	1	0x000500080002
10	JMP(if b==0) 6	0x00040006	0	2	85184	1	3	85184	1	0x000200040006
11	MUL regA, regA	0x00100000	0	3	85184	1	4	3748096	1	0x000300100000
12	DEC regB	0x00200000	0	4	3748096	1	5	3748096	0	0x000400200000
13	JMP 2	0x00080002	0	5	3748096	0	2	3748096	0	0x000500080002
14	JMP(if b==0) 6	0x00040006	0	2	3748096	0	6	3748096	0	0x000200040006
15	STOP	0x00400000	0	6	3748096	0	6	3748096	0	0x000600400000

Program Correctness

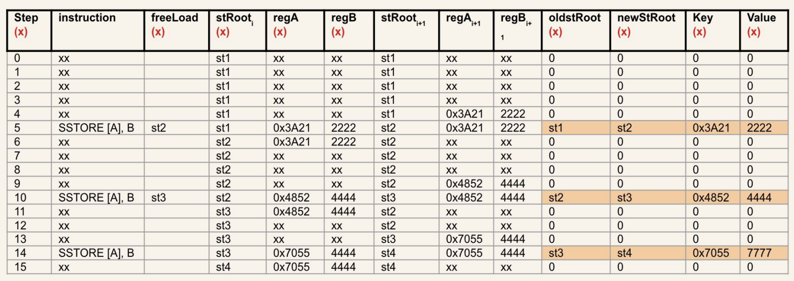
Use Plookup (lookup table) to prove inter-VM correctness

Hash VM trace



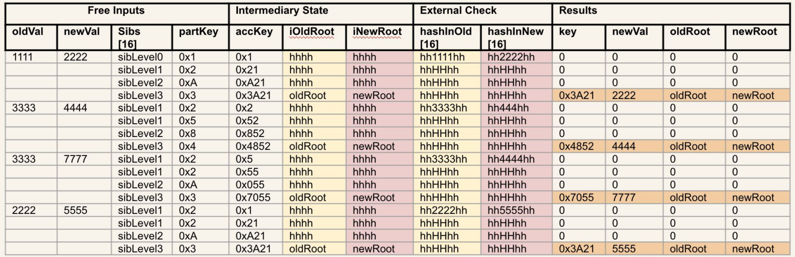
	Free Inputs	Intermediary State	Results	
	FreeIn [16]	St [16]	In [16]	out
4,8,3,...	in1	in1,0,....	0	0
step1		iSt	0	0
..		iSt	0	0
step 60		0x34345...	4,8,3,...	0x34345
7,3,5,..	in2	0x2	0	0
step1		iSt	0	0
..		..	0	0
step 60		0x34345...	7,3,5,,...	0x835454
8,4,3,...	in3	0x2	0	0
step1		iSt	0	0
..		..	0	0
step 60		0x34345...	8,4,3,,...	0x835454
3,9,8	in4	0x2	0	0
step1		iSt	0	0
..		..	0	0
step 60		0x34345...	3,9,8,,...	0x835454
		..		

Main-VM trace



Step (x)	instruction	freeLoad (x)	stRoot$_i$ (x)	regA (x)	regB (x)	stRoot$_{i+1}$	regA$_{i+1}$	regB$_{i+1}$	oldstRoot (x)	newStRoot (x)	Key (x)	Value (x)
0	xx		st1	xx	xx	st1	xx	xx	0	0	0	0
1	xx		st1	xx	xx	st1	xx	xx	0	0	0	0
2	xx		st1	xx	xx	st1	xx	xx	0	0	0	0
3	xx		st1	xx	xx	st1	xx	xx	0	0	0	0
4	xx		st1	xx	xx	st1	0x3A21	2222	0	0	0	0
5	SSTORE [A], B	st2	st1	0x3A21	2222	st2	0x3A21	2222	st1	st2	0x3A21	2222
6	xx		st2	0x3A21	2222	st2	xx	xx	0	0	0	0
7	xx		st2	xx	xx	st2	xx	xx	0	0	0	0
8	xx		st2	xx	xx	st2	xx	xx	0	0	0	0
9	xx		st2	xx	xx	st2	0x4852	4444	0	0	0	0
10	SSTORE [A], B	st3	st2	0x4852	4444	st3	0x4852	4444	st2	st3	0x4852	4444
11	xx		st3	0x4852	4444	st2	xx	xx	0	0	0	0
12	xx		st3	xx	xx	st2	xx	xx	0	0	0	0
13	xx		st3	xx	xx	st3	0x7055	4444	0	0	0	0
14	SSTORE [A], B		st3	0x7055	4444	st4	0x7055	4444	st3	st4	0x7055	7777
15	xx		st4	0x7055	4444	st4	xx	xx	0	0	0	0

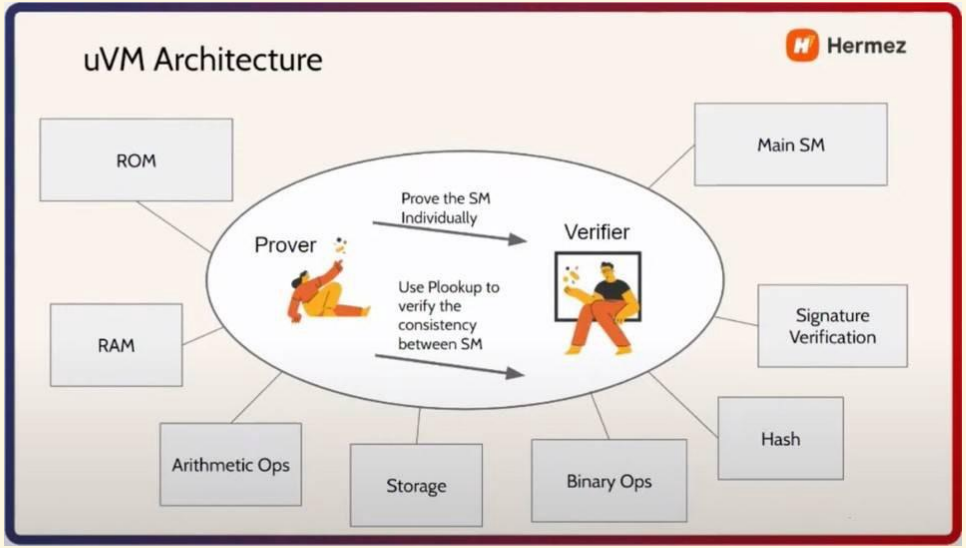
Merkle tree-VM trace



Free Inputs				Intermediary State			External Check		Results			
oldVal	newVal	Sibs [16]	partKey	accKey	iOldRoot	iNewRoot	hashInOld [16]	hashInNew [16]	key	newVal	oldRoot	newRoot
1111	2222	sibLevel0	0x1	0x1	hhhh	hhhh	hh1111hh	hh2222hh	0	0	0	0
		sibLevel1	0x2	0x21	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel2	0xA	0xA21	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel3	0x3	0x3A21	oldRoot	newRoot	hhHHhh	hhHHhh	0x3A21	2222	oldRoot	newRoot
3333	4444	sibLevel1	0x2	0x2	hhhh	hhhh	hh3333hh	hh444hh	0	0	0	0
		sibLevel1	0x5	0x52	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel2	0x8	0x852	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel3	0x4	0x4852	oldRoot	newRoot	hhHHhh	hhHHhh	0x4852	4444	oldRoot	newRoot
3333	7777	sibLevel1	0x2	0x5	hhhh	hhhh	hh3333hh	hh4444hh	0	0	0	0
		sibLevel1	0x2	0x55	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel2	0xA	0x055	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel3	0x3	0x7055	oldRoot	newRoot	hhHHhh	hhHHhh	0x7055	7777	oldRoot	newRoot
2222	5555	sibLevel1	0x2	0x1	hhhh	hhhh	hh2222hh	hh5555hh	0	0	0	0
		sibLevel1	0x2	0x21	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel2	0xA	0xA21	hhhh	hhhh	hhHHhh	hhHHhh	0	0	0	0
		sibLevel3	0x3	0x3A21	oldRoot	newRoot	hhHHhh	hhHHhh	0x3A21	5555	oldRoot	newRoot

Recap

Lots of polynomial relationships!



Proof Verification

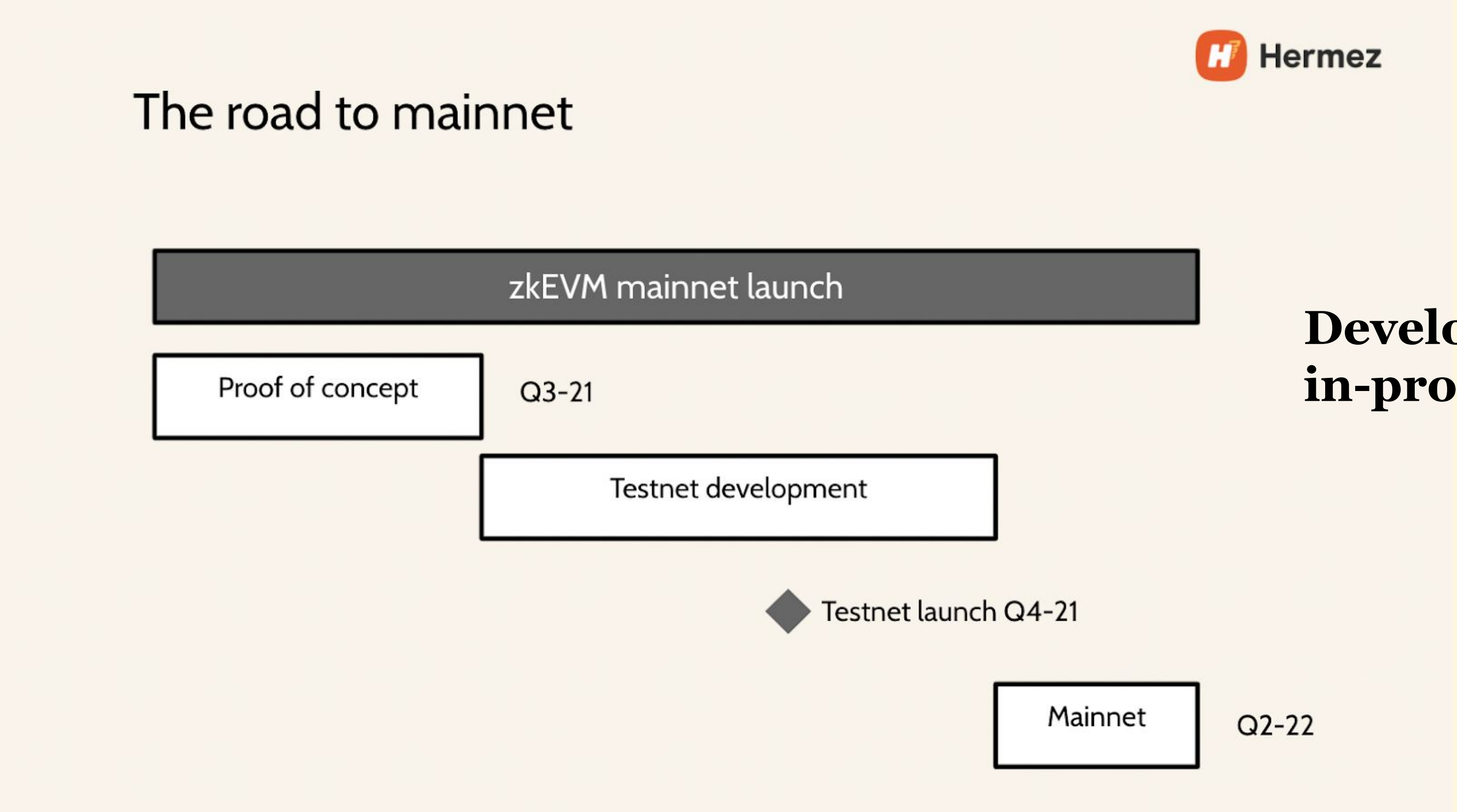
Verifying zkSNARK Proof

Publishing Proofs on L1

- Two-Proof System
    - Generating STARK proof
    - PLONK or GROTH-16 (zero-knowledge proof system / circuit) to generate proof of STARK proof
    - Verify on L1

“Proof of a Proof!”

Hermez Roadmap



Hermez

The road to mainnet

zkEVM mainnet launch

Proof of concept Q3-21

Testnet development

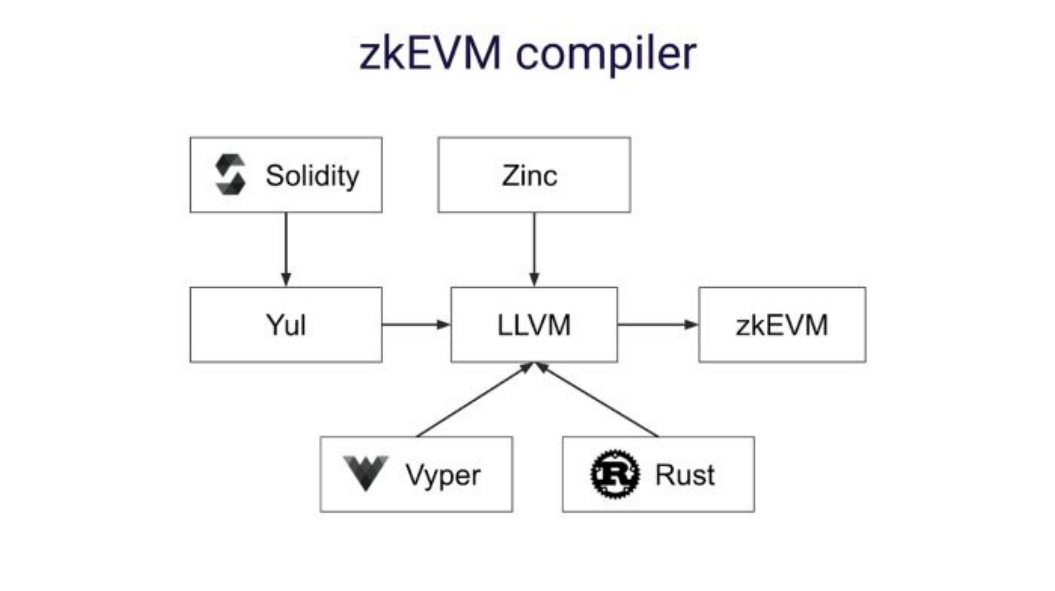
Testnet launch Q4-21

Mainnet Q2-22

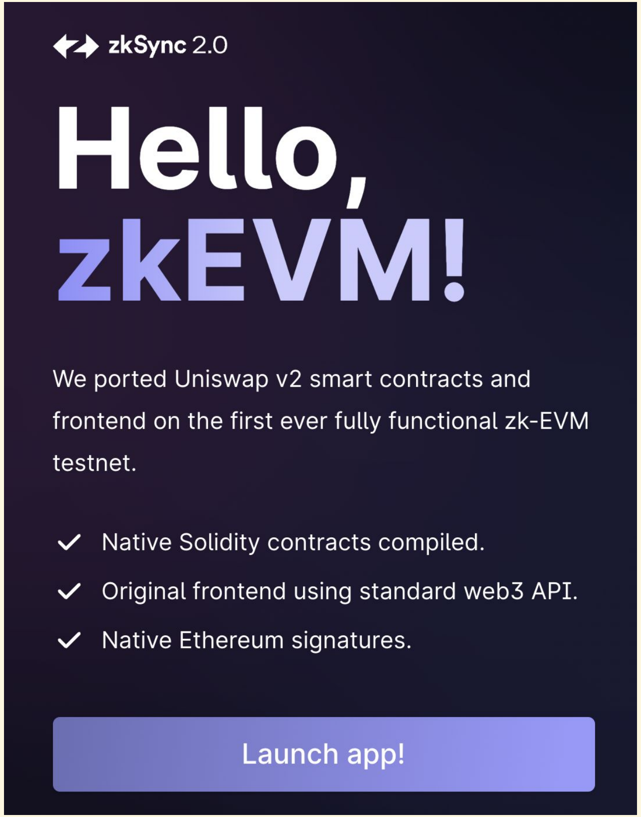
Development in-progress!

zkSync 2.0

- Newly designed virtual machine that runs 99% of Solidity contracts



Demo



zkSync 2.0

Hello, zkEVM!

We ported Uniswap v2 smart contracts and frontend on the first ever fully functional zk-EVM testnet.

- ✓ Native Solidity contracts compiled.
- ✓ Original frontend using standard web3 API.
- ✓ Native Ethereum signatures.

First full application on an EVM-Compatible zkRollup!

Port of the Uniswap V2 frontend on L2!

References

https://medium.com/degate/an-article-to-understand-zkevm-the-key-to-ethereum-scaling-ff0d83c417cc

https://medium.com/@sin7y/an-analysis-of-polynomial-commitment-schemes-kzg10-ipa-fri-and-darks-a8f806bd3e12

https://www.youtube.com/watch?v=17d5DG6L2nw&ab_channel=AmphiMonge

https://www.youtube.com/watch?v=0JIlopu0KIc&ab_channel=ZeroKnowledge

https://t.co/TeCFWGuvSh?amp=1

https://uni.zksync.io/#/

https://www.youtube.com/watch?v=0JIlopu0KIc&ab_channel=ZeroKnowledge

Thank you!

SCALABLE SYSTEMS
& SOFTWARE
RESEARCH GROUP

https://sss.cse.lehigh.edu/