Modern ZK Crypto

Circom 1 (Brian Gu)

Housekeeping

Schedule updates/reminders

- Thursday - ZKML Office Hours with Prof. Jason Morton (5-7PM, 2-136)
- Friday - ZK Math Building Blocks with Prof. Yufei Zhao

- **[UPDATE]** Monday - No activities
- **[UPDATE]** Tuesday - Project Ideation Office Hours (10AM-12PM, 2-136)
- Tuesday - Circom 2 with Vivek Bhupatiraju

Course materials + communication

- Session 1 recording, slides, lecture notes, exercises are at zkiap.com
  - Today's lecture notes: https://hackmd.io/@gubsheep/Hyx1hho5o
- We'll also send out program announcements over email, in addition to Discord
  - This is probably about one email per class - unsubscribe whenever!

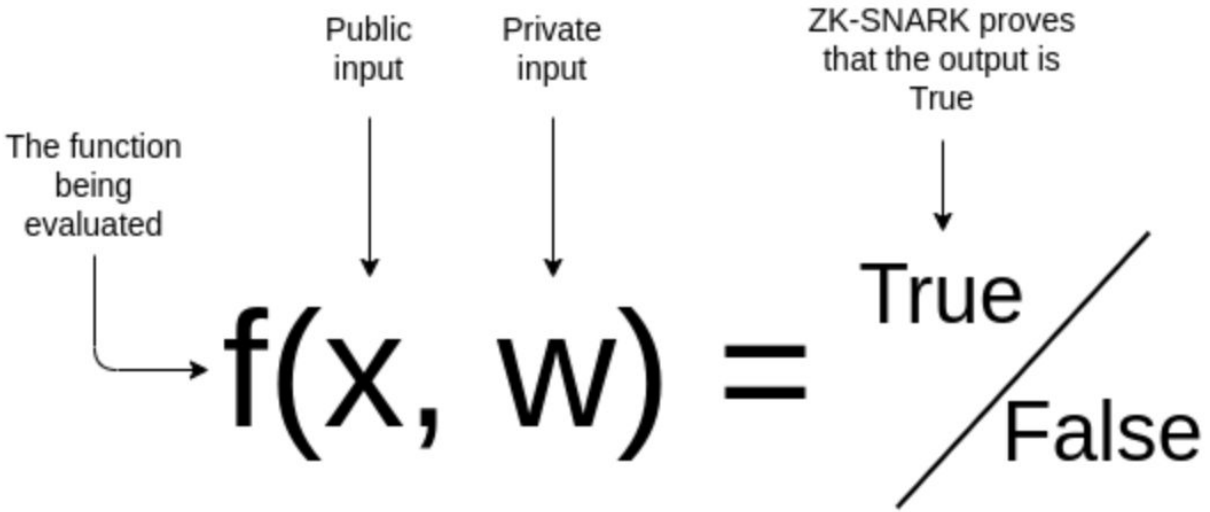
- On Friday we'll collect Discord handles of folks who want to join for the rest of the month

zkSNARKs

What are zkSNARKs?

- A new cryptographic tool that can efficiently generate a zero-knowledge protocol for *any problem or function*.
- Properties:
  - **zk**: hides inputs
  - **Succinct**: generates short proofs that can be verified quickly
  - **Noninteractive**: doesn't require a back-and-forth
  - **ARgument of Knowledge**: proves you know the input

What are zkSNARKs?



What are zkSNARKs?

High-level idea:

- Turn your problem (graph isomorphism, discrete log, etc.) into a function whose inputs you want to hide.
- Turn that function into an equivalent set of “R1CS” (or other) equations
  - Basically, an arithmetic circuit - a bunch of + and * operations on prime field elements
  - Simplified: equations of the form $x_i + x_j = x_k$, or $x_i * x_j = x_k$
- Generate a ZKP for satisfiability of the R1CS

zkSNARK Properties

- A new cryptographic tool that can efficiently generate a zero-knowledge protocol for *any problem or function*.
- Properties:
  - **zk**: hides inputs
  - **Succinct**: generates short proofs that can be verified quickly
  - **Noninteractive**: doesn’t require a back-and-forth
  - **ARgument of Knowledge**: proves you know the input

zkSNARKs

- Function inputs: x1, x2, x3, x4
- OUT = f(x) = (x1 + x2) * x3 - x4

- zkSNARK: I know some secret (x1, x2, x3, x4) such that the result of this computation is OUT. Here's a signature that proves that I know such a tuple, without telling you what the tuple actually is.

zkSNARKs prove constraints

- Function inputs: x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 * x3
- OUT := y2 - x4

- SNARK prover inputs: x1, x2, x3, x4, y1, y2, OUT
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
  - y1 == x1 + x2
  - y2 == y1 * x3
  - y2 == OUT + x4

zkSNARKs prove constraints

- Function inputs: 02, 04, 08, 05
- 06 := 02 + 04
- 48 := 06 * 08
- 043 := 48 - 05

- SNARK prover inputs: 02, 04, 08, 05, 06, 48, 043
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
  - 06 == 02 + 04
  - 48 == 06 * 08
  - 48 == 043 + 05

zkSNARKs prove constraints

- Function inputs: x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 * x3
- 043 := y2 - x4

- SNARK prover inputs: x1, x2, x3, x4, y1, y2, 043
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
  - y1 == x1 + x2
  - y2 == y1 * x3
  - y2 == 043 + x4

zkSNARKs

- Function inputs: x1, x2, x3, x4
- OUT = f(x) = (x1 + x2) / x3 - x4

- zkSNARK: I know some secret (x1, x2, x3, x4) such that the result of this computation is OUT. Here's a signature that proves that I know such a tuple, without telling you what the tuple actually is.

zkSNARKs prove constraints (only + and *)

- Function inputs: x1, x2, x3, x4
- y1 := x1 + x2
- y2 := y1 / x3
- OUT := y2 - x4

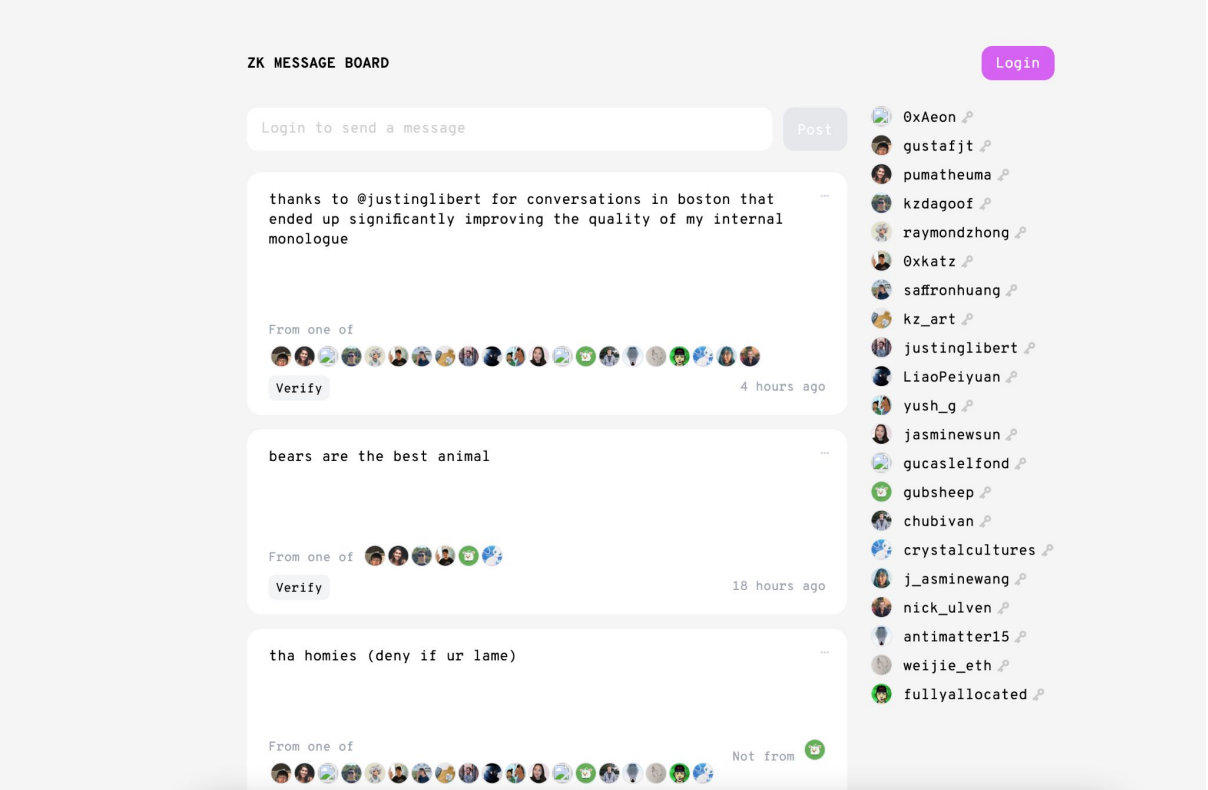
- SNARK prover inputs: x1, x2, x3, x4, y1, y2, OUT
- SNARK prover output: a “signature” that only verifies if the following constraints are satisfied:
  - y1 == x1 + x2
  - y1 == y2 * x3
  - y2 == OUT + x4

ZKRepl demo #1

ZKRepl demo #2: Num2Bits

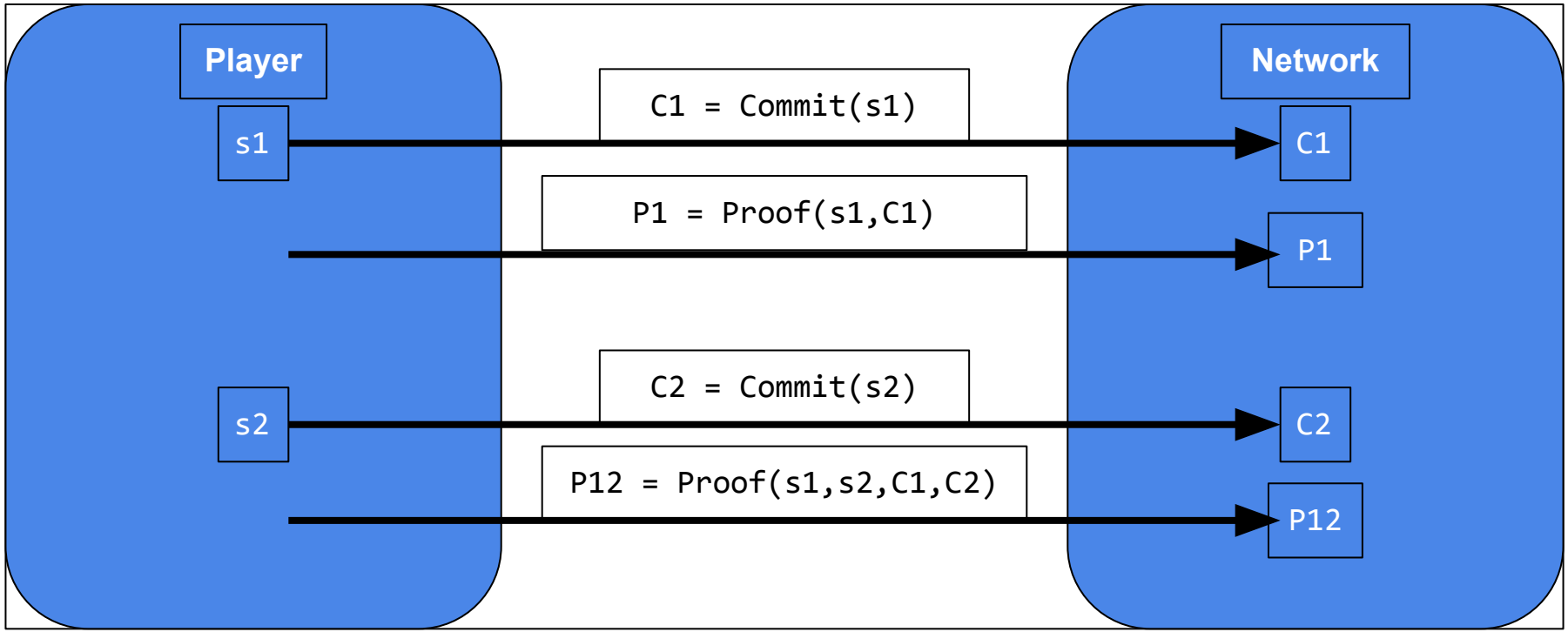
ZKRepl demo #3: IsZero

Applications



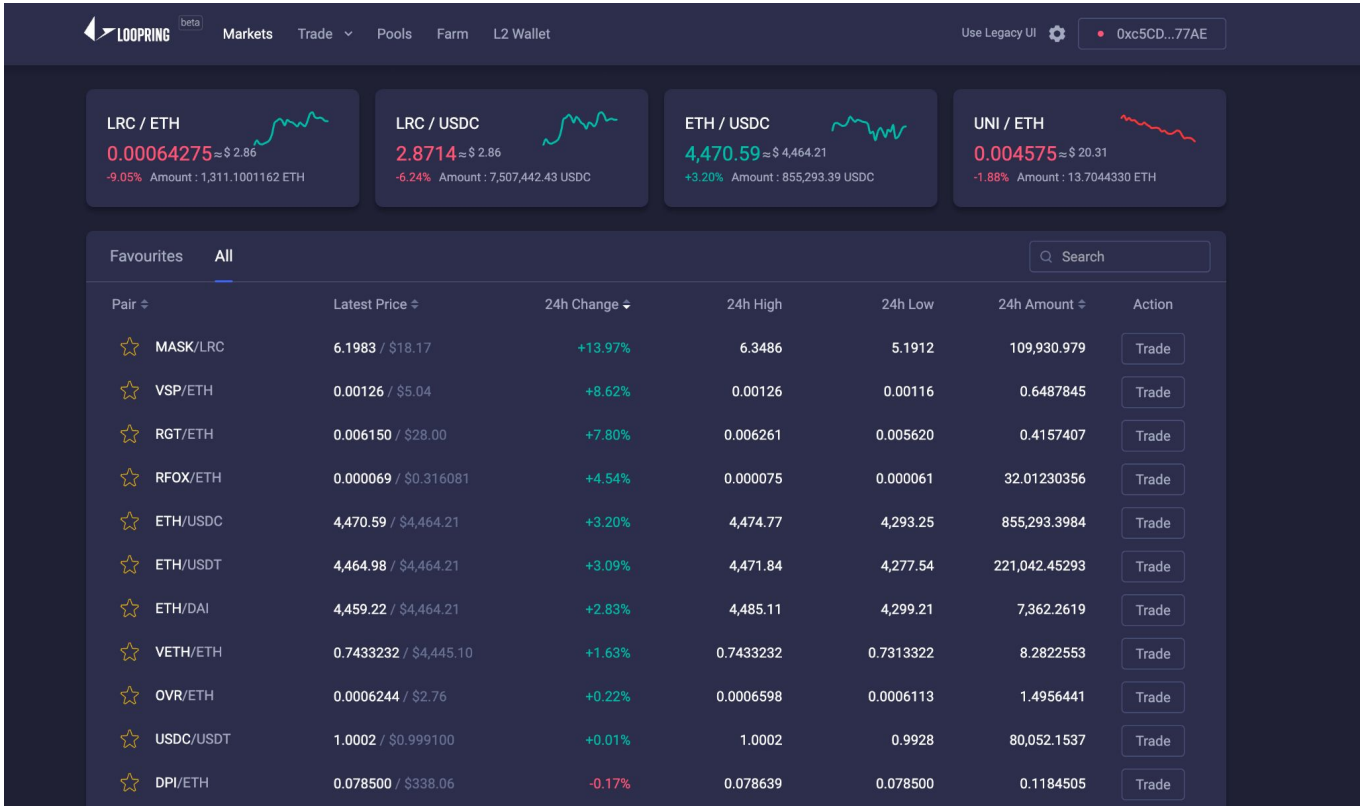
Applications

Applications

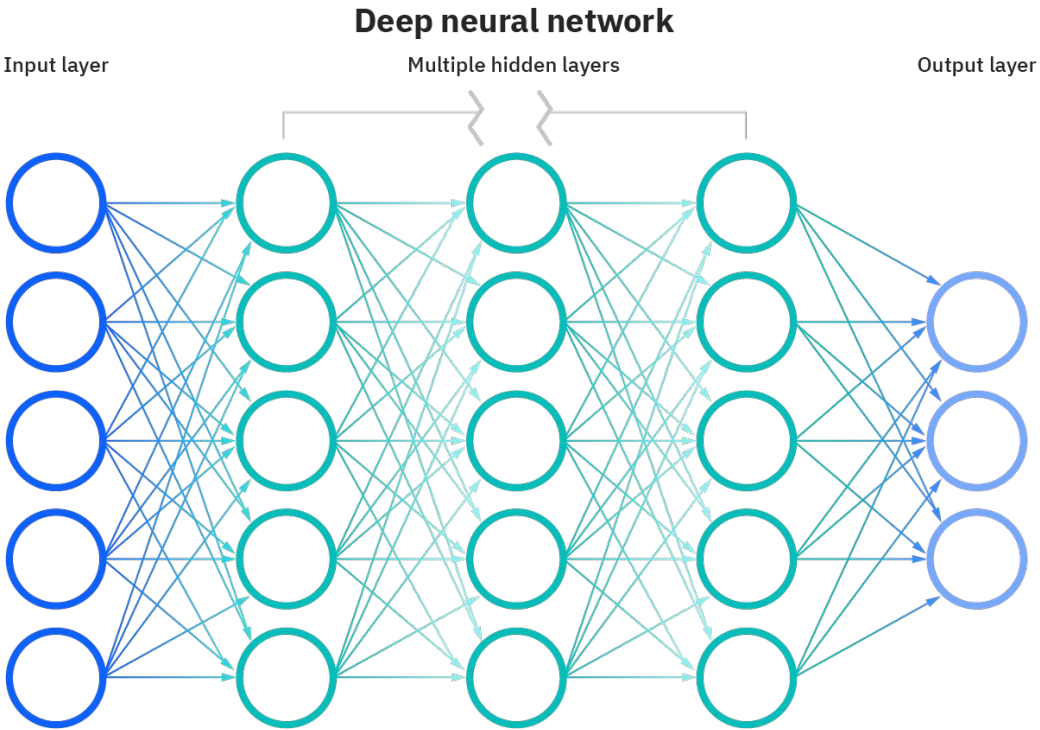


Applications

Applications



Applications



Open areas of exploration

Open areas of exploration

- Building better circuits

Open areas of exploration

- Building better circuits
- Building better protocols

Open areas of exploration

- Building better circuits
- Building better protocols
- Uncovering new use cases

Open areas of exploration

- Building better circuits
- Building better protocols
- Uncovering new use cases
- Building infrastructure, securing existing circuits, etc…