Modern ZK Crypto

Introduction to Zero Knowledge

Today

- Why is modern ZK interesting?
- What is this program about?
- Program logistics
- Program structure/schedule
- Conceptual intro to ZKPs and zkSNARKs (may be review for some of you)
- A few examples of ZK apps

Wednesday

- Writing your first ZK circuits

Why are we interested in modern ZK?

Thesis

vitalik.eth
@VitalikButerin

Replying to @tarunchitra

I expect ZK-SNARKs to be a significant revolution as they permeate the mainstream world over the next 10-20 years.

5:40 PM · Sep 1, 2021 · Twitter Web App

537 Retweets **198** Quote Tweets **2,732** Likes

Thesis

ZK crypto (specifically SNARKs, STARKs, etc. - ZKPs for arbitrary computation) is more important and general than people think it is.

Thesis

ZK crypto is easier to use than people realize it is, and new entrants can rapidly start making real contributions.

Thesis

ZK crypto is a fun and challenging problem space that rewards breadth and creative problem-solving.

What is this course about?

“Full stack” of zero-knowledge

- Theoretical underpinnings of modern ZK crypto
  - Focus is on building intuition, rather than strict mathematical rigor.
- Modern ZK toolstacks, and how to use them
  - Computers with F_p instead of {0, 1}, and F_p[x] instead of arrays.
  - We’ll mostly be using the circom + snarkjs stack.
- Applications of ZK, and application design patterns
  - These applications are often decentralized apps (“dapps”) for reasons we’ll get into.

Session 1 (Monday 1/9) Introduction to ZK (Brian Gu)

We'll give an overview of the course, and a whirlwind tour of modern zero-knowledge techniques and applications. This session will approach the "why" of the course: why has ZK been such an exciting topic lately, and why do we think that it has the potential to one of the biggest technology stories of the next decade?

Session 2 (Wednesday 1/11) Circom 1 (Brian Gu)

This session focuses on practical zkSNARK circuit engineering: using a toolstack (circom/snarkjs/zkREPL) for the groth16 zkSNARK protocol to build simple zero-knowledge proofs. We'll discuss the R1CS programming model (and cost model), and simple circuit components such as bit operators, range checks, and more.

Session 3 (Friday 1/13) Mathematical building blocks (Yufei Zhao)

In this session, we'll discuss some of the basic "building blocks" of modern proof systems, including: formalization of zero-knowledge, discrete logarithm and other common cryptographic sources of "hardness," elliptic curve cryptography, and pairing-based cryptography.

Session 4 (Tuesday 1/17) Circom 2 (Harish Bhaskaranjan)

Note that this session will take place on Tuesday, as Monday is MLK day.

Building off "Circom 1," we'll write and discuss more complex circuits: inclusion proof verification, hash functions, signature and encryption verification.

Session 5 (Wednesday 1/18) Commitment Schemes (Ying Tong Lai)

We'll build off of the "mathematical building blocks" session to construct vector, univariate polynomial, and multivariate polynomial commitment schemes.

Session 6 (Friday 1/20) Algorithms for Efficient Cryptographic Operations (Jason Morton)

We'll discuss techniques for efficient openings and polynomial arithmetic, including number-theoretic transform (NTT); multi-scalar multiplication (MSM); fast elliptic curve double-and-add operations.

Session 7 (Monday 1/23) Arithmetizations (Ying Tong Lai)

We discuss a few examples of arithmetizations—intermediate representations of ZK programs and circuits which can be consumed by a proving system.

Session 8 (Wednesday 1/25) PLONK and polynomial identities. (Jason Morton)

We dive into the PLONK zkSNARK protocol—a zkSNARK construction based on polynomial commitment schemes, and a particular PLONK-style arithmetization. We also discuss arguments like LOOKUP, built from polynomial identities.

Session 9 (Friday 1/27) Proving systems stack; recursion and composition. (Ying Tong Lai)

Based on the learnings from previous four sessions, we'll give an overview of the zkSNARK protocol landscape, and build up a taxonomy of proving systems. We'll also discuss proof system recursion and composition.

Session 10 (Monday 1/30) Applied ZK Constructions 1 (Ying Tong Lai)

We'll discuss ZK constructions in the wild: membership proofs for pseudonymous messaging, nullifier-based constructions for private digital currency transfers, and incomplete information games with zkSNARKs.

Session 11 (Wednesday 2/1) Applied ZK Constructions 2 (Brian Gu)

We'll discuss more advanced uses of zkSNARKs: encrypted data marketplaces, ZKML, ZKVMs, recursive ZK proofs, proof-of-email, and more.

Session 12 (Friday 2/3) Student and Staff Demos

In our final session, students and staff will demonstrate projects and ZK applications that they've been working on over IAP!

- Theoretical underpinnings of modern ZK crypto
  - Focus is on building intuition, rather than strict mathematical rigor.

Session 1 (Monday 1/9) Introduction to ZK (Brian Gu)

We'll give an overview of the course, and a whirlwind tour of modern zero-knowledge techniques and applications. This session will approach the "why" of the course: why has ZK been such an exciting topic lately, and why do we think that it has the potential to one of the biggest technology stories of the next decade?

Session 2 (Wednesday 1/11) Circom 1 (Brian Gu)

This session focuses on practical zkSNARK circuit engineering: using a toolstack (circom/snarkjs/zkREPL) for the groth16 zkSNARK protocol to build simple zero-knowledge proofs. We'll discuss the R1CS programming model (and cost model), and simple circuit components such as bit operators, range checks, and more.

In this session, we'll discuss some of the basic "building blocks" of modern proof systems, including: formalization of zero-knowledge, discrete logarithm and other common cryptographic sources of "hardness," elliptic curve cryptography, and pairing-based

Session 4 (Tuesday 1/17) Circom 2 (Vivek Bhupatiraju)

Note that this session will take place on Tuesday, as Monday is MLK day.

Building off "Circom 1," we'll write and discuss more complex circuits: inclusion proof verification, hash functions, signature and encryption verification.

We'll build off of the "mathematical building blocks" session to construct vector, univariate polynomial, and multivariate polynomial commitment schemes.

Session 6 (Friday 1/20) Algorithms for Efficient Cryptographic Operations (Jason Morton)

We'll discuss techniques for efficient openings and polynomial arithmetic, including number-theoretic transform (NTT); multi-scalar multiplication (MSM); fast elliptic curve double-and-add operations.

Session 7 (Monday 1/23) Arithmetizations (Ying Tong Lai)

We discuss a few examples of arithmetizations—intermediate representations of ZK programs and circuits which can be consumed by a proving system.

Session 8 (Wednesday 1/25) PLONK and polynomial identities. (Jason Morton)

We dive into the PLONK zkSNARK protocol—a zkSNARK construction based on polynomial commitment schemes, and a particular PLONK-style arithmetization. We also discuss arguments like LOOKUP, built from polynomial identities.

Session 9 (Friday 1/27) Proving systems stack; recursion and composition. (Ying Tong Lai)

Based on the learnings from previous four sessions, we'll give an overview of the zkSNARK protocol landscape, and build up a taxonomy of proving systems. We'll also discuss proof system recursion and composition.

Session 10 (Monday 1/30) Applied ZK Constructions 1 (Aayush Gupta)

We'll discuss ZK constructions in the wild: membership proofs for pseudonymous messaging, nullifier-based constructions for private digital currency transfers, and incomplete information games with zkSNARKs.

Session 11 (Wednesday 2/1) Applied ZK Constructions 2 (Brian Gu)

We'll discuss more advanced uses of zkSNARKs: encrypted data marketplaces, ZKML, ZKVMs, recursive ZK proofs, proof-of-email, and more.

Session 12 (Friday 2/3) Student and Staff Demos

In our final session, students and staff will demonstrate projects and ZK applications that they've been working on over IAP!

- Modern ZK toolstacks, and how to use them
  - Computers with F_p instead of {0, 1}, and F_p[x] instead of arrays.
  - We'll mostly be using the circom + snarkjs stack.

Recommended Project

We highly encourage interested to participate in the optional project component, to solidify their understanding of the material. Course staff will provide mentorship for students interested in building a ZK project over the course of the month. Projects may include:

- A full-stack application of ZK crypto, such as an anonymous voting app, a p2p/decentralized game, a cryptocurrency mixer, etc.
- A library of useful ZK primitives, such as ZK circuits for a ZK-friendly encryption scheme.
- An implementation of a zero-knowledge proof system or some key component parts, along with a series of tutorials or writeups.
- Documentation or educational material, such as a series of blog posts or tutorials explaining a ZK proof system.

Projects from teams that have participated in past 0xPARC educational programs have included:

- zkREPL, an in-browser collaborative development environment for writing ZK circuits.
- circom-ecdsa, an implementation of Ethereum's signature algorithms in zkSNARK circuits.
- zkmessage.xyz, a demonstration of how zkSNARKs can be used to emulate and extend other cryptographic primitives, such as ring signatures.
- Zordle, a webapp that allows you to generate zero-knowledge proofs that your Wordle guess diagram is legitimate. A subproject of Zordle involved porting the Halo2 ZK proving system to WASM.

Session 1 (Monday 1/9) Introduction to ZK (Brian Gu)

We'll give an overview of the course, and a whirlwind tour of modern zero-knowledge techniques and applications. This session will approach the "why" of the course: why has ZK been such an exciting topic lately, and why do we think that it has the potential to one of the biggest technology stories of the next decade?

Session 2 (Wednesday 1/11) Circom 1 (Brian Gu)

This session focuses on practical zkSNARK circuit engineering: using a toolstack (circom/snarkjs/zkREPL) for the groth16 zkSNARK protocol to build simple zero-knowledge proofs. We'll discuss the R1CS programming model (and cost model), and simple circuit components such as bit operators, range checks, and more.

Session 3 (Friday 1/13) Mathematical building blocks (Yufei Zhao)

In this session, we'll discuss some of the basic "building blocks" of modern proof systems, including: formalization of zero-knowledge, discrete logarithm and other common cryptographic sources of "hardness," elliptic curve cryptography, and pairing-based cryptography.

Session 4 (Tuesday 1/17) Circom 2 (Vivek Bhupatiraju)

Note that this session will take place on Tuesday, as Monday is MLK day.

Building off "Circom 1," we'll write and discuss more complex circuits: inclusion proof verification, hash functions, signature and encryption verification.

Session 5 (Wednesday 1/18) Commitment Schemes (Ying Tong Lai)

We'll build off of the "mathematical building blocks" session to construct vector, univariate polynomial, and multivariate polynomial commitment schemes.

Session 6 (Friday 1/20) Algorithms for Efficient Cryptographic Operations (Jason Morton)

We'll discuss techniques for efficient openings and polynomial arithmetic, including number-theoretic transform (NTT); multi-scalar multiplication (MSM); fast elliptic curve double-and-add operations.

Session 7 (Monday 1/23) Arithmetizations (Ying Tong Lai)

We discuss a few examples of arithmetizations—intermediate representations of ZK programs and circuits which can be consumed by a proving system.

Session 8 (Wednesday 1/25) PLONK and polynomial identities. (Jason Morton)

We dive into the PLONK zkSNARK protocol—a zkSNARK construction based on polynomial commitment schemes, and a particular PLONK-style arithmetization. We also discuss arguments like LOOKUP, built from polynomial identities.

Session 9 (Friday 1/27) Proving systems stack; recursion and composition. (Ying Tong Lai)

Based on the learnings from previous four sessions, we'll give an overview of the zkSNARK protocol landscape, and build up a taxonomy of proving systems. We'll also discuss proof

Session 10 (Monday 1/30) Applied ZK Constructions 1 (Aayush Gupta)

We'll discuss ZK constructions in the wild: membership proofs for pseudonymous messaging, nullifier-based constructions for private digital currency transfers, and incomplete information games with zkSNARKs.

Session 11 (Wednesday 2/1) Applied ZK Constructions 2 (Brian Gu)

We'll discuss more advanced uses of zkSNARKs: encrypted data marketplaces, ZKML, ZKVMs, recursive ZK proofs, proof-of-email, and more.

Session 12 (Friday 2/3) Student and Staff Demos

In our final session, students and staff will demonstrate projects and ZK applications that they've been working on over IAP!

- Applications of ZK, and application design patterns
  - These applications are often decentralized apps ("dapps") for reasons we'll get into.

Recommended Project

We highly encourage interested to participate in the optional project component, to solidify their understanding of the material. Course staff will provide mentorship for students interested in building a ZK project over the course of the month. Projects may include:

- A full-stack application of ZK crypto, such as an anonymous voting app, a p2p/decentralized game, a cryptocurrency mixer, etc.
- A library of useful ZK primitives, such as ZK circuits for a ZK-friendly encryption scheme.
- An implementation of a zero-knowledge proof system or some key component parts, along with a series of tutorials or writeups.
- Documentation or educational material, such as a series of blog posts or tutorials explaining a ZK proof system.

Projects from teams that have participated in past 0xPARC educational programs have included:

- zkREPL, an in-browser collaborative development environment for writing ZK circuits.
- circom-ecdsa, an implementation of Ethereum's signature algorithms in zkSNARK circuits.
- zkmessage.xyz, a demonstration of how zkSNARKs can be used to emulate and extend other cryptographic primitives, such as ring signatures.
- Zordle, a webapp that allows you to generate zero-knowledge proofs that your Wordle guess diagram is legitimate. A subproject of Zordle involved porting the Halo2 ZK proving system to WASM.

Logistics - Main Sessions

- Lectures are **MWF 2 - 3:30** in **room 4-237**
  - Except for MLK weekend, where the usual Monday session will take place on Tuesday
  - That week, we'll have an informal office hours session on Monday.
- Communication is primarily on **Discord**
  - Discord will be the home of async discussion as well
  - We'll open up the "full Discord" later this week!
- Resources are on the **program homepage**: zkiap.com
  - Sessions will be recorded and uploaded (generally within 1-2 days)
  - Slides, lecture notes, psets, and supplementary material is hosted here
  - Streaming link (?)
- Add the **Google Calendar**

Logistics - Additional Programming

- Office Hours
- Optional Problem Sets
- PLONKathon
- Student Projects

Logistics - Office Hours

- Office Hours
  - **Room 2-136, Tuesdays 10AM - 12PM and Thursdays 5PM - 7PM**.
  - **First Thursday has no OH; second week OH is on Monday/Thursday (MLK week)**
  - Come by to talk about psets, lecture content, projects, or co-work + discuss ZK generally.
  - Staff are also generally available during weekdays, on campus or on Discord. After class is also a good time to catch us.
  - We'll have food :)

Logistics - Problem Sets

- Optional Problem Sets
  - Instructors will post optional psets along with each lecture.
  - Coding exercises, math problems, understanding questions.
  - Come to OH / Discord if you have questions.

Logistics - Student Projects

- Projects
  - Optional but highly-encouraged!
  - Spend the last two weeks doing something hands-on
  - Most project-related activities will take place in OH (finding teams, brainstorming, getting help from mentors, etc.)
- Some examples
  - Build a ZK app
  - Build developer tool(s)
  - (Re)-build a ZK library
  - Educational material or documentation

Logistics - PLONKathon

- PLONKathon
  - Third weekend (1/28 - 1/29), all day Saturday and Sunday
  - A “learning hackathon” where you’ll either:
    - Build PLONK (mostly) from scratch
    - Implement a core circuit library in circom
    - Work on your final project

Logistics - Additional Programming

- Optional Problem Sets
- Office Hours
- PLONKathon
- Student Projects

You'll get the most out of this program if you're able to spend 10hr/wk+ outside of main lectures.

We highly recommend for you to show up in person to programming! The program community has historically been a major component of the experience for past students.

Pre-requisites

- Elementary number theory and group theory
- Basic cryptographic primitives
- Basic algebraic concepts, esp. polynomials

Course structure / schedule

Structure / Schedule

- Week 1: Introduction
  - Today: Program Intro
  - Tuesday: Office Hours - discuss projects, psets, general ZK questions, etc.
  - Wednesday: Circom 1
  - Thursday: Office Hours - ZKML
  - Friday: ZK Math Building Blocks
- Use this week to figure out if you want to participate in the full program!

Structure / Schedule

- Week 2
  - Monday (MLK Day): Unofficial Office Hours - Project Ideation session
  - Tuesday: Circom 2
  - Wednesday: Commitment Schemes
  - Thursday: Office Hours - Team and project matching
  - Friday: Algorithms for Efficient Cryptographic Operations
- Submit project proposals by Sunday 1/22

Structure / Schedule

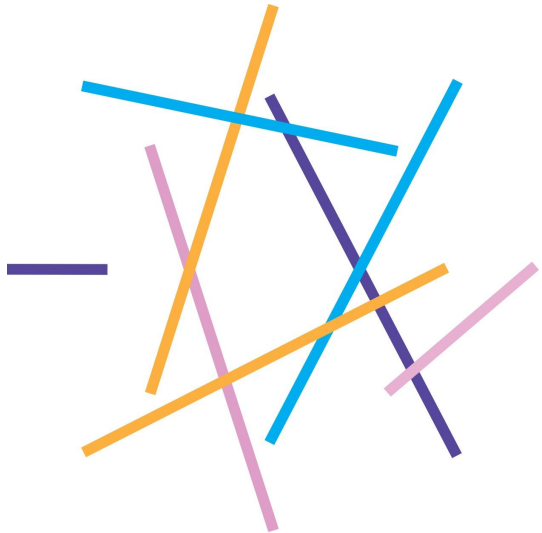
- Week 3
  - Monday: Arithmetizations
  - Tuesday: Office Hours - Project Review/Feedback
  - Wednesday: PLONK and polynomial identities
  - Thursday: Office Hours - Projects, psets, general questions
  - Friday: Proving Systems Stack; Recursion/Composition
  - Saturday + Sunday: PLONKathon

Structure / Schedule

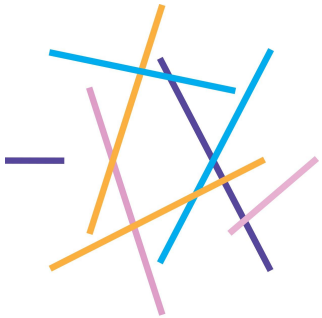
- Week 4
  - Monday: Applied ZK Constructions 1
  - Tuesday: Office Hours - Projects, psets, general questions
  - Wednesday: Applied ZK Constructions 2
  - Thursday: Office Hours - Projects, psets, general questions
  - Friday: Final demos + public ZK talks

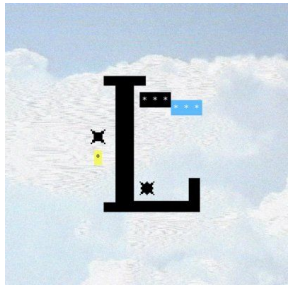
About 0xPARC

- Open-source, application-level R&D
- How do we use these powerful new tools in production?
- What infrastructure will we need to enable them?



0xPARC Community





PL

Intro

- Brian: brian@0xparc.org
  - 0xPARC; ZK at the application level
- Michael: michael@0xparc.org
  - 0xPARC; Course logistics and resources, ZK security
- Ying Tong: yingtong@0xparc.org
  - Geometry + 0xPARC; Proving systems and ZK recursion
- Aayush: aayushg@mit.edu
  - Personae Labs; ZK circuits and ZK for identity
- Vivek: vivek@personaelabs.org
  - Personae Labs; ZK circuits and ZK for identity
- Jason: jason@zkonduit.com
  - ZKonduit + Penn State Mathematics; ZKML
- Yufei: yufeiz@mit.edu
  - MIT Mathematics
- Izabella: izabella@0xparc.org
  - 0xPARC; Course operations

Intro

- I’m Brian
- Studied math at MIT
- Worked with Ethereum Foundation on R&D and ecodev since 2018
- Worked on various experimental ZK R&D projects under the Thiel Fellowship
- Started 0xPARC, supporting applied crypto R&D

What is a Zero Knowledge Protocol?

Zero Knowledge Proofs / Protocols

Zero-Knowledge (ZK) crypto lets me **prove to you that I know a fact, without telling you the fact**.

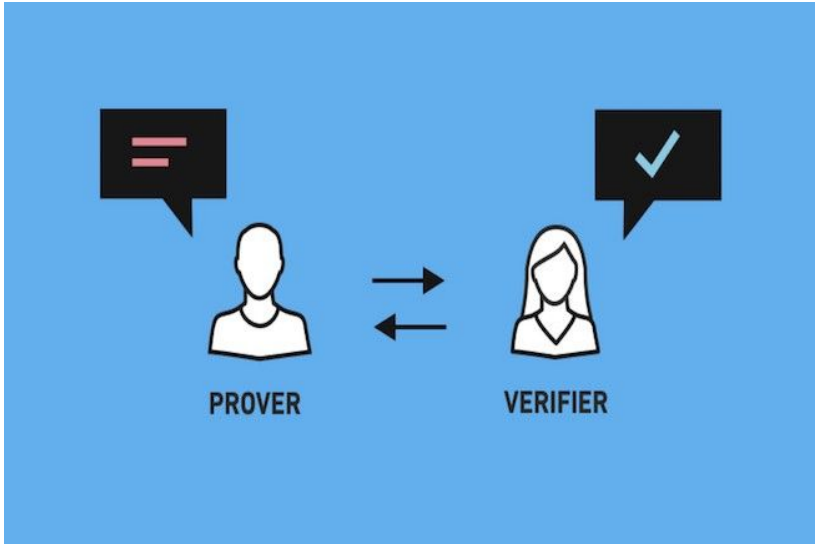
- I know the private key corresponding to an Ethereum account - but I won't tell you what my private key is!
- I know a way to fill in a map with 3 colors such that no two adjacent regions are the same color - but I won't tell you the coloring!
- I know a number x such that SHA256(x) = 0x77af… - but I won't tell you x!

Zero Knowledge Proofs / Protocols

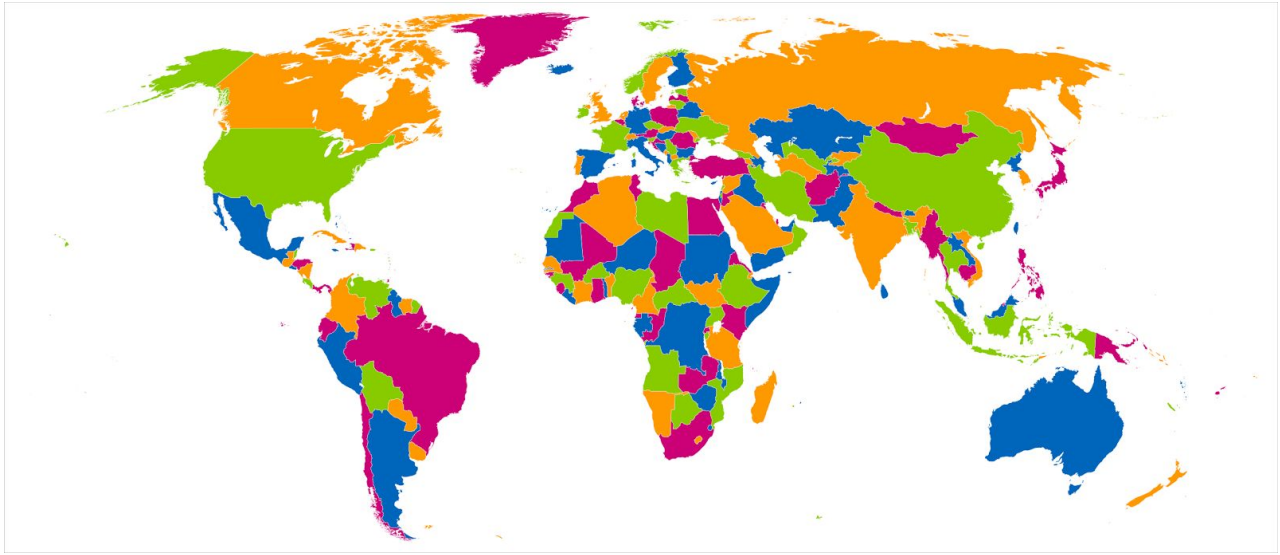
Setup: a **prover** wants to convince a **verifier** that they know something, without revealing the underlying information.

Verifier: Asks questions / issues challenges to the prover, and checks responses.

Prover: Responds to verifier questions or challenges.

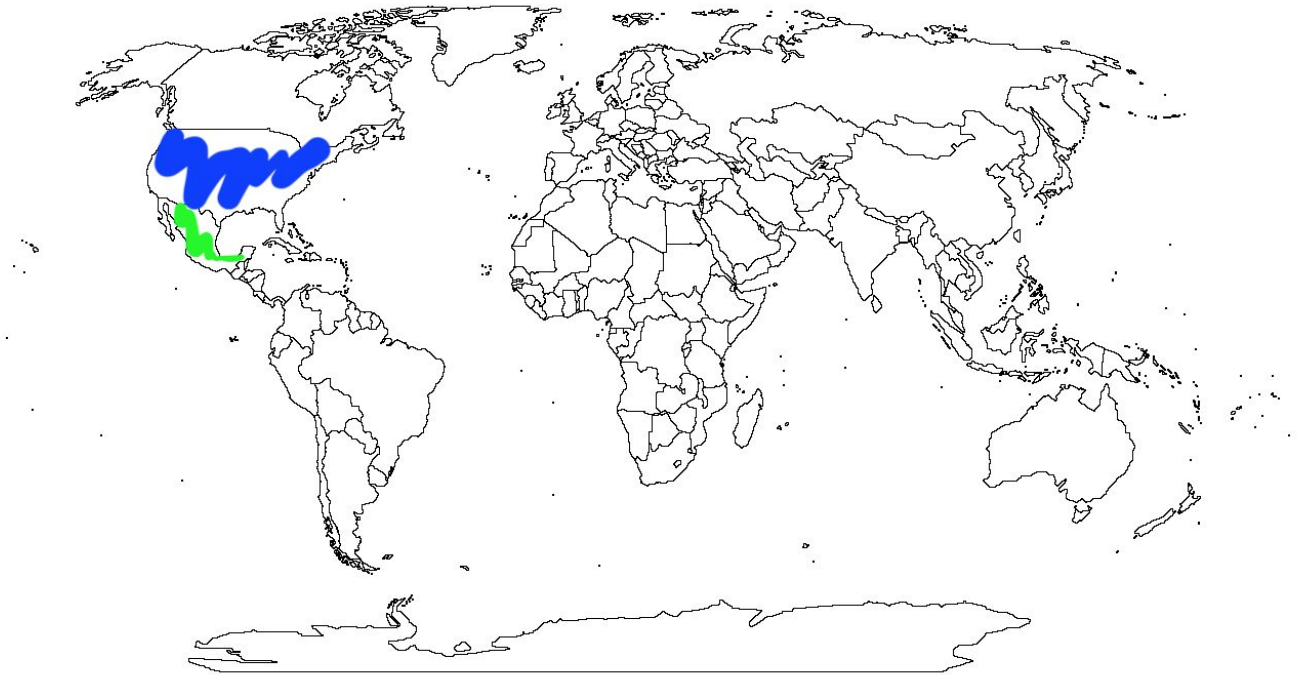


Example: ZK protocol for knowledge of map (3-)coloring

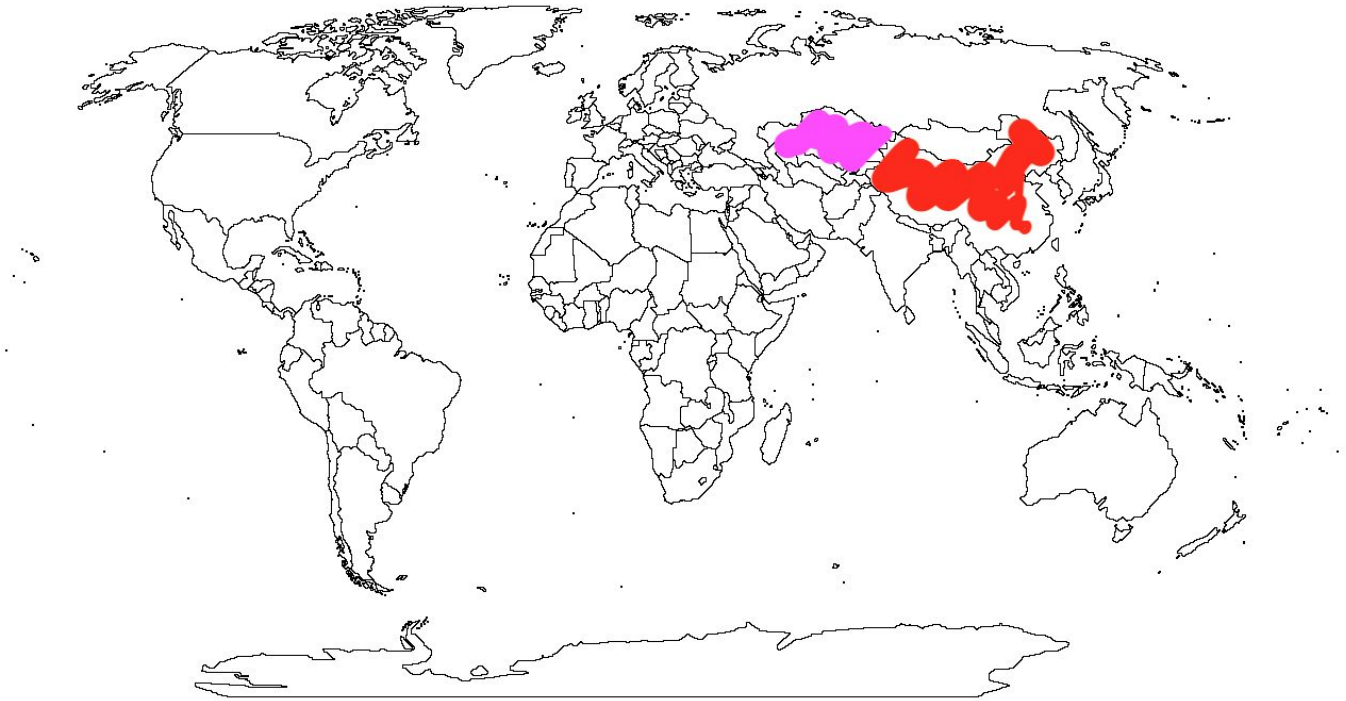


Example: map 3-coloring

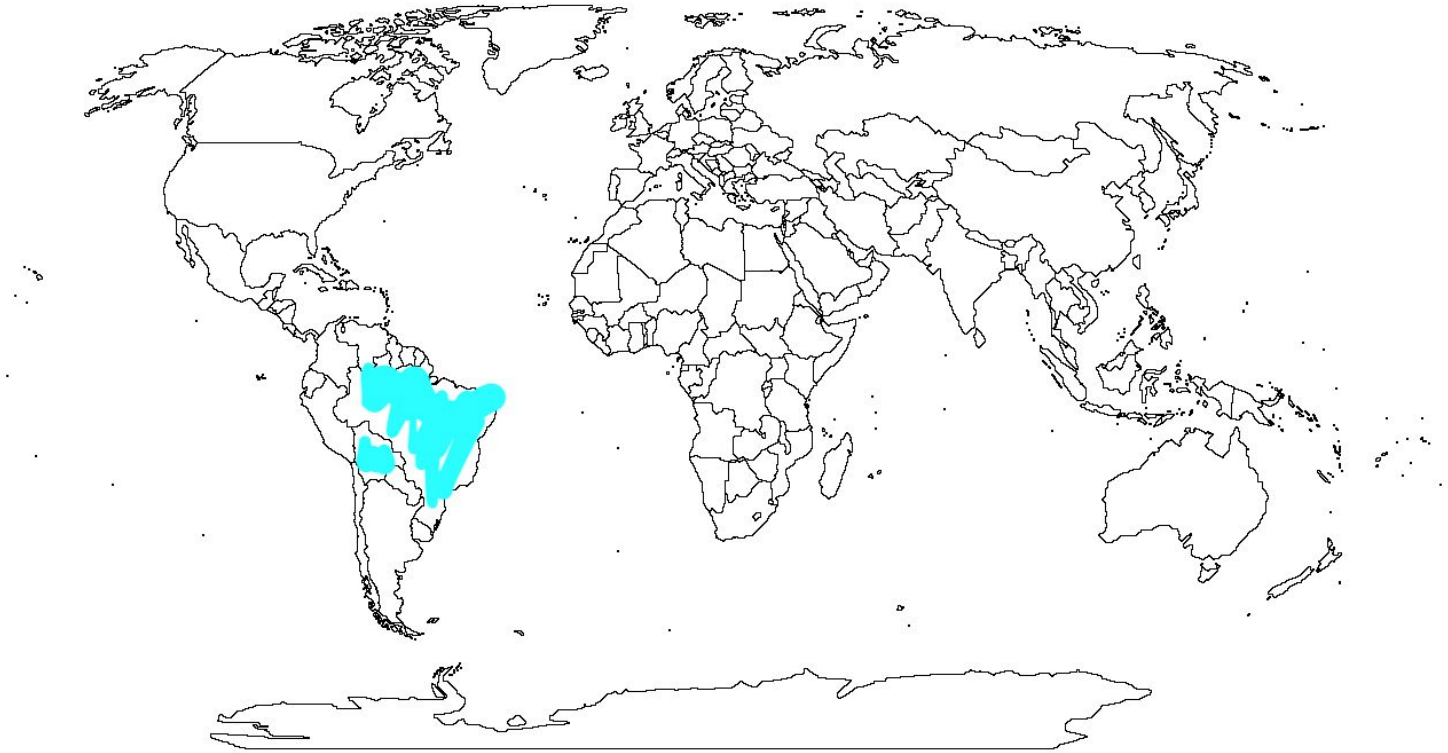
Example: map 3-coloring



Example: map 3-coloring



Example: map 3-coloring



Example: map 3-coloring

http://web.mit.edu/~ezyang/Public/graph/svg.html

Zero Knowledge Proofs / Protocols

ZK Protocols have 3 properties:

- The Prover’s responses don’t reveal the underlying information.

Zero Knowledge Proofs / Protocols

ZK Protocols have 3 properties:

- The Prover’s responses don’t reveal the underlying information.
- If the Prover knows the underlying information, they’re always able to answer satisfactorily.

Zero Knowledge Proofs / Protocols

ZK Protocols have 3 properties:

- The Prover’s responses don’t reveal the underlying information.
- If the Prover knows the underlying information, they’re always able to answer satisfactorily.
- If the Prover doesn’t know the underlying info, they’ll eventually get caught.

Zero Knowledge Proofs / Protocols

ZK Protocols have 3 properties:

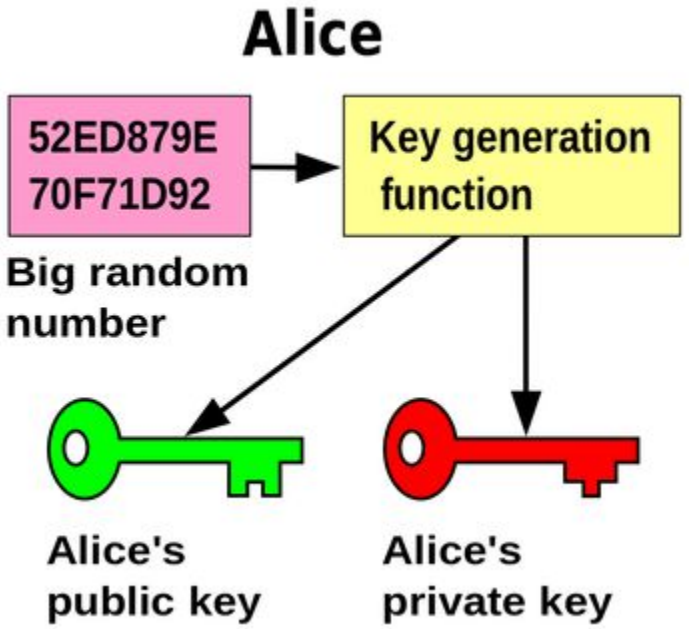
- **[Zero Knowledge]** The Prover's responses don't reveal the underlying information.
- **[Completeness]** If the Prover knows the underlying information, they're always able to answer satisfactorily.
- **[Soundness]** If the Prover doesn't know the underlying info, they'll eventually get caught.

Example 2: Digital Signatures

In Ethereum, all transactions are signed with **public-key cryptography**.

Every public account is associated with a secret, private key.

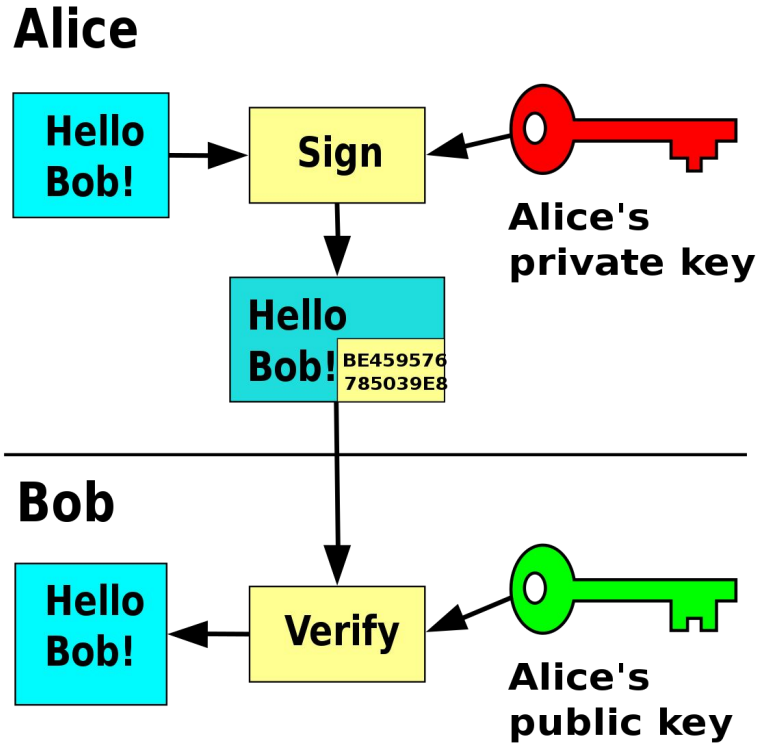
You shouldn't be able to send funds out of an account unless you know the private key to that account.



Example 2: Digital Signatures

A “**signature”** is attached to every transaction.

Under the hood, a signature is (essentially) a **zero knowledge proof** that you know the private key corresponding to the public key you’re sending funds from.



ZKPs are not new!

- Digital signature schemes have been around for decades.
- Zero Knowledge protocols for specific problems have been known for decades as well.
  - Map 3-coloring
  - Graph isomorphism
  - Discrete logarithm
  - Hash pre-images

ZKPs are not new!

- For each of the above problems, researchers would have to come up with a special-purpose / specific ZK protocol.
- The holy grail: “Here’s an output *y* and an arbitrary function *f*. I know a secret value *x* such that *f(x) = y*”
  - A technique to do this would allow us to verify arbitrary computation (for example, money or digital ownership transfers) with complete privacy

Digression: Anonymous Voting

- Setup: Five people with known public/private keys, voting on “is pineapple good on pizza”

Digression: Anonymous Voting

- Setup: Five people with known public/private keys, voting on “is pineapple good on pizza”
  - {sk1, pk1}, {sk2, pk2}, {sk3, pk3}, {sk4, pk4}, {sk5, pk5}

Digression: Anonymous Voting

- Setup: Five people with known public/private keys, voting on “is pineapple good on pizza”
  - {sk1, pk1}, {sk2, pk2}, {sk3, pk3}, {sk4, pk4}, {sk5, pk5}

Along with my vote, I’ll attach a proof:

- I know some secret sk, such that:
  - pubkeygen(sk) = pk
  - (pk - pk1)(pk - pk2)(pk - pk3)(pk - pk4)(pk - pk5) = 0

Digression: Anonymous Voting

- Setup: Five people with known public/private keys, voting on “is pineapple good on pizza”
  - {sk1, pk1}, {sk2, pk2}, {sk3, pk3}, {sk4, pk4}, {sk5, pk5}

Along with my vote, I’ll attach a proof:

- I know some secret sk, such that:
  - pubkeygen(sk) = pk
  - (pk - pk1)(pk - pk2)(pk - pk3)(pk - pk4)(pk - pk5) = 0
- Is this enough?

Digression: Anonymous Voting

- Setup: Five people with known public/private keys, voting on “is pineapple good on pizza”
  - {sk1, pk1}, {sk2, pk2}, {sk3, pk3}, {sk4, pk4}, {sk5, pk5}

Along with my vote, I’ll attach a proof:

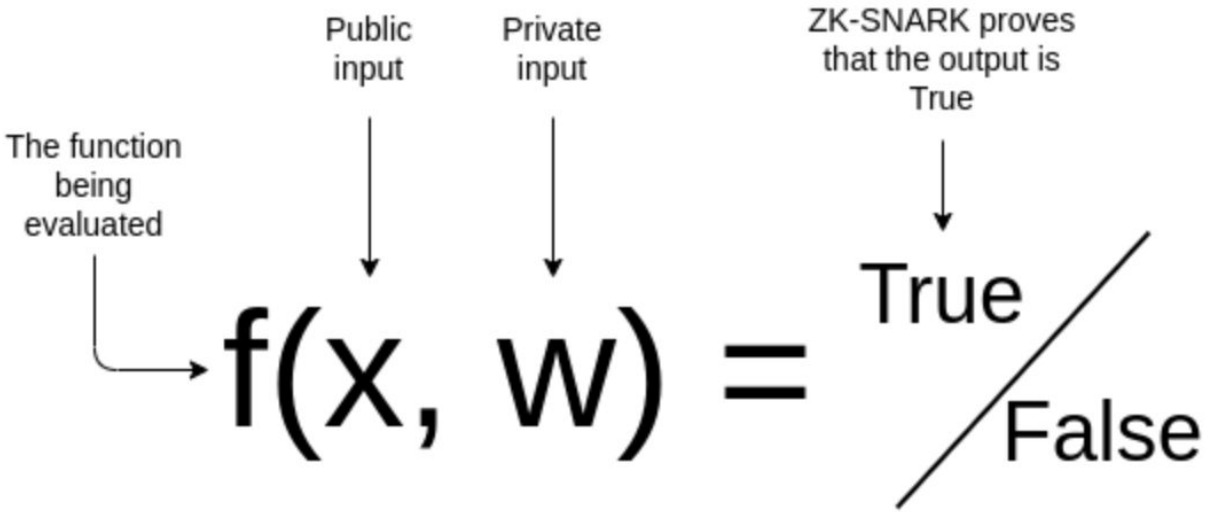
- **For some public nf,** I know some secret sk, such that:
  - pubkeygen(sk) = pk
  - (pk - pk1)(pk - pk2)(pk - pk3)(pk - pk4)(pk - pk5) = 0
  - **hash(sk) = nf**

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

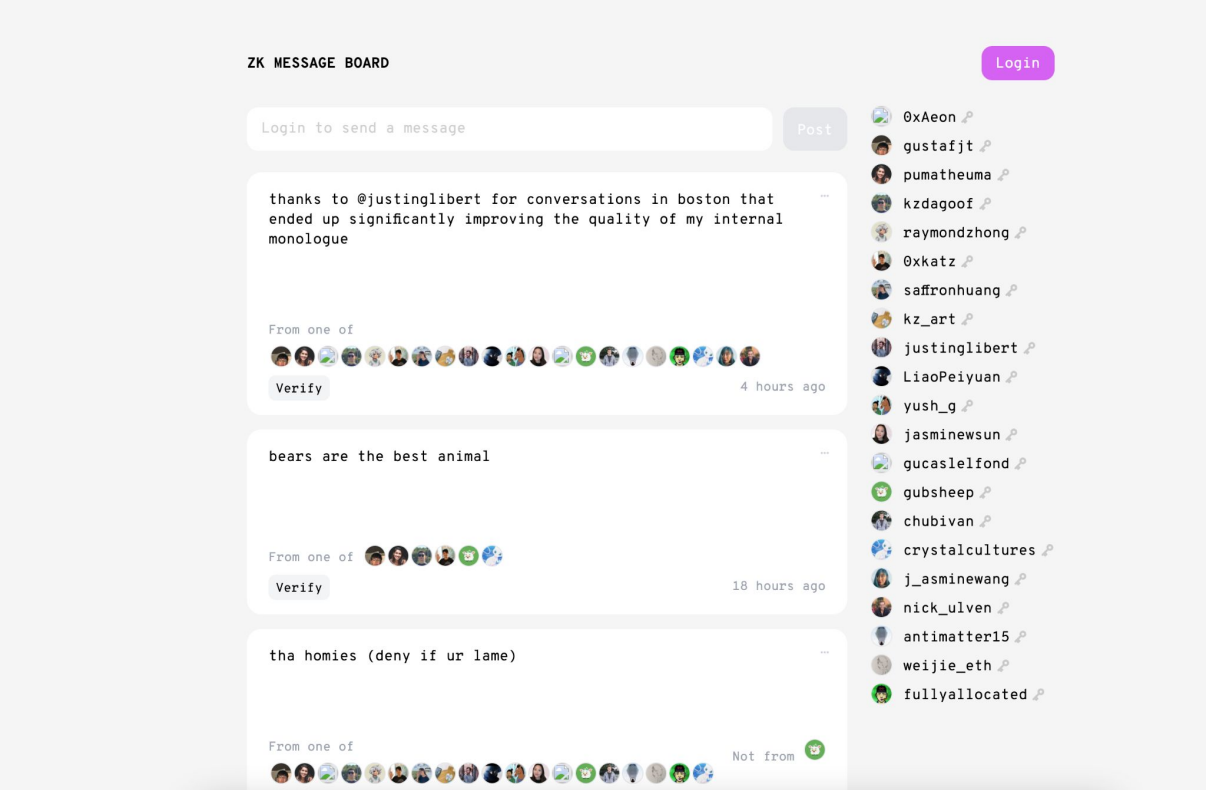
zkSNARKs prove constraints (only + and *)

- 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

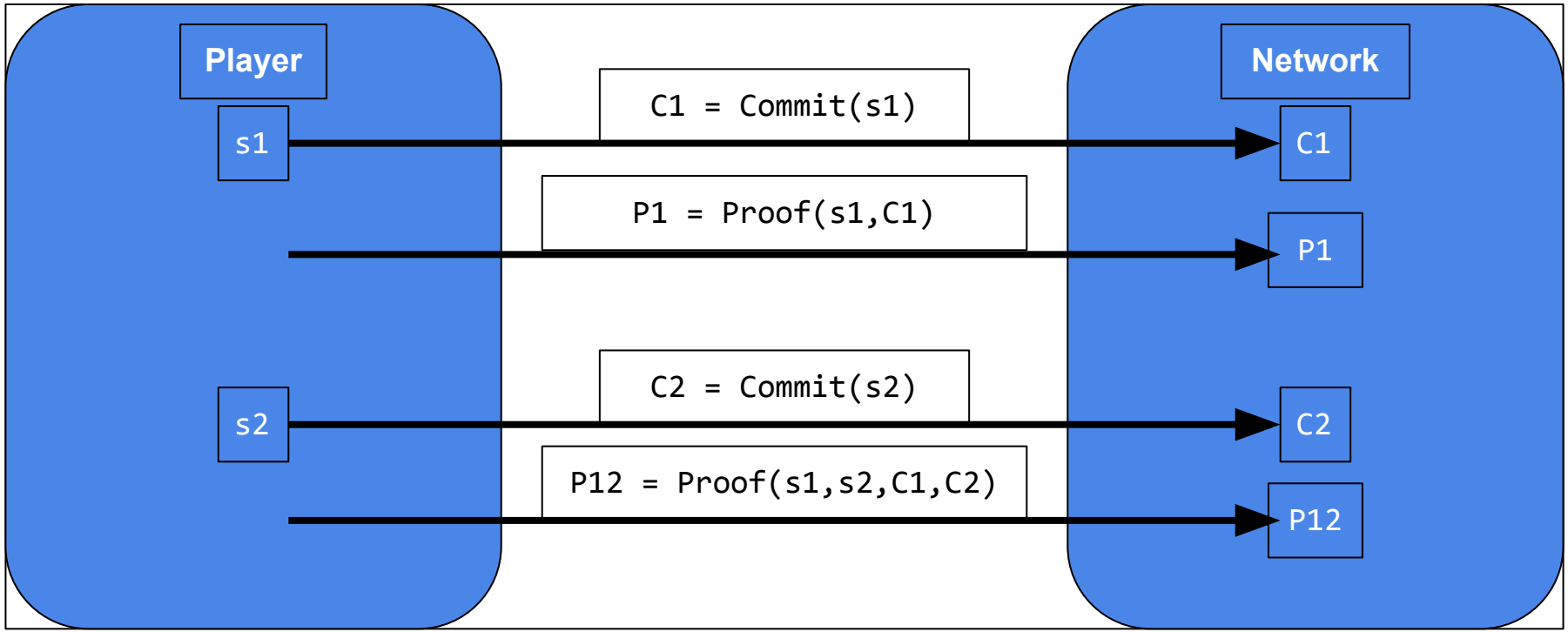
ZKRepl demo #1

Applications



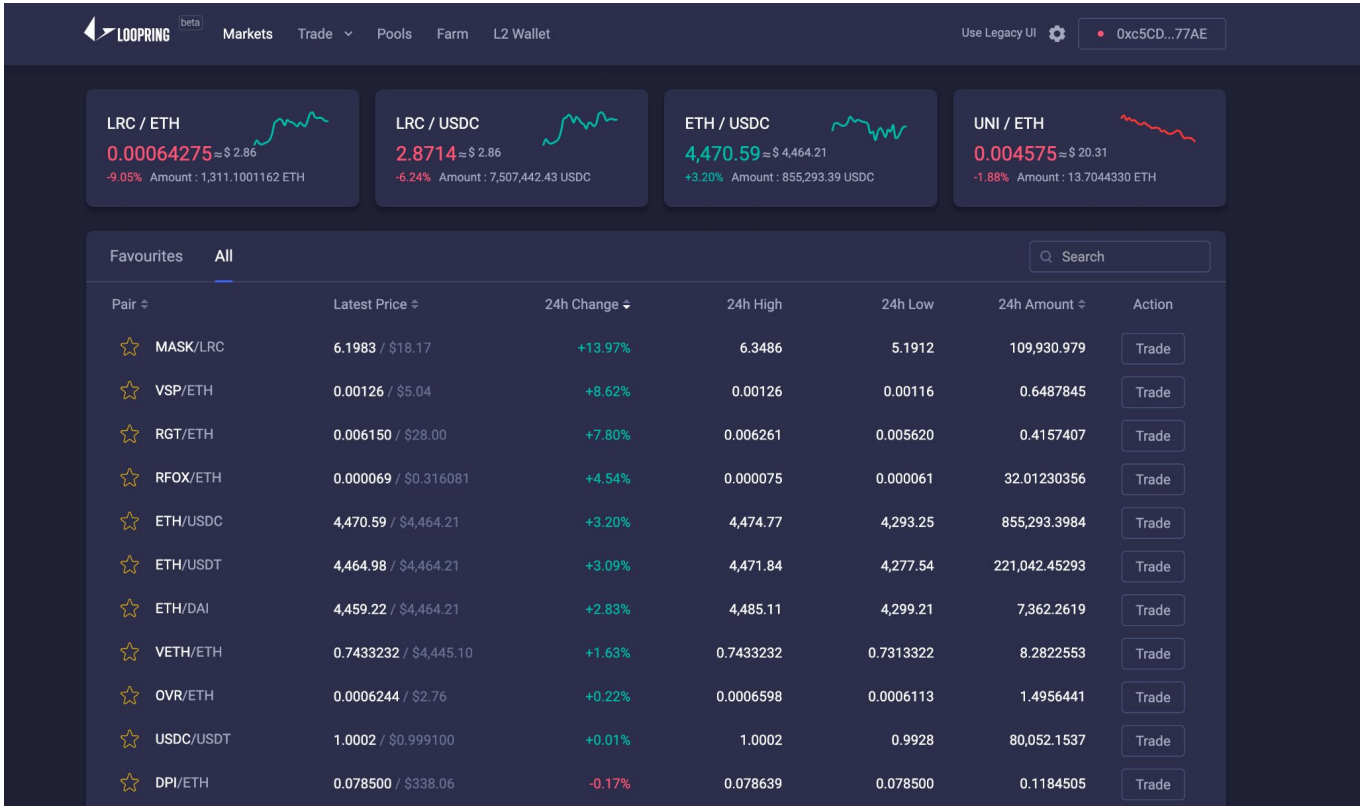
Applications

Applications

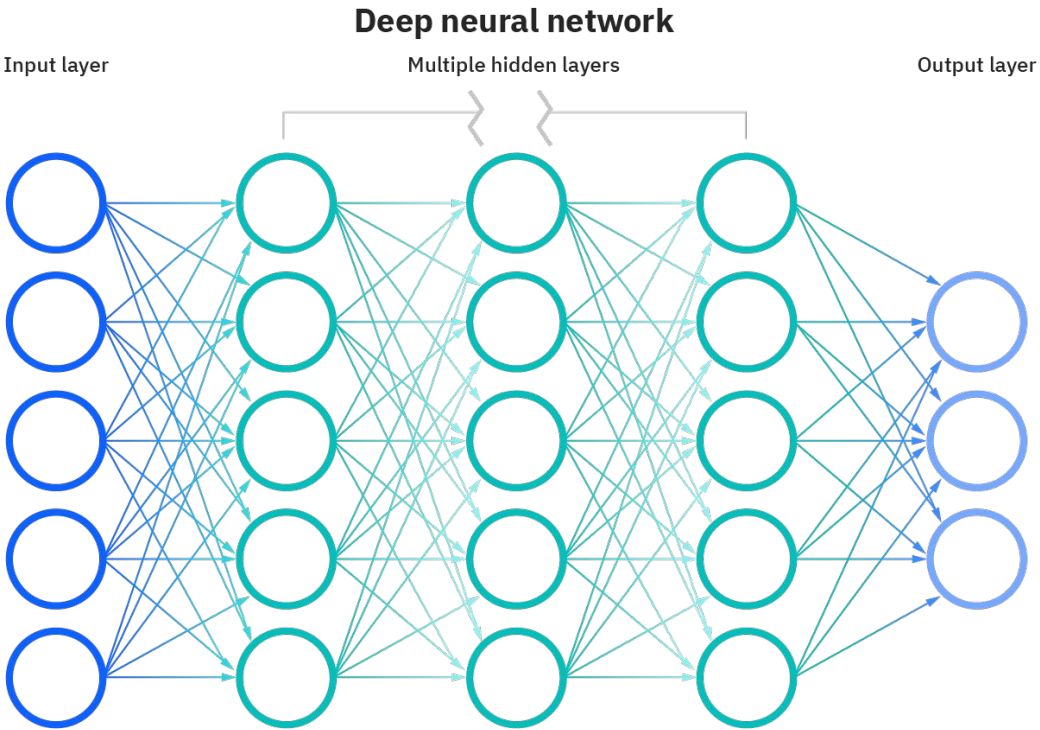


Applications

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…