zk-proofs - from novice to master

Ariel Gabizon

Aztec

*"If encryption is a light switch - on or off,
zero-knowledge proofs are a dimmer allowing you to
control exactly how much you information expose"*

The deck of cards:

A full deck with red and black cards, face down.

I take out a red three of hearts. How to

convince you I took a red card, without showing which one

Proving color to the color blind:

A red and green ball, otherwise indentical

How to convince a color-blind friend they are different?.

Counting leaves in a tree:

How to prove you can instantly count the number of leaves on a tree, without disclosing the number of leaves?

Visual example: Where's Waldo?

3-coloring

How can we prove to someone we can color a graph with 3 colors without leaking the coloring?

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

Chicken and egg problem: Alice has sudoku puzzle solution, Bob wants to buy it - who goes first?.

ZKCP: Protocol where money and solution change hands at exactly same time.

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

1. Alice chooses cryptographic key $\mathbf{K}$, sends $\mathbf{h} = \mathbf{HASH}(\mathbf{K})$.

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

1. Alice chooses cryptographic key $\mathbf{K}$, sends $\mathbf{h} = \mathbf{HASH}(\mathbf{K})$.
2. Alice sends encrypted solution $\mathbf{C} = \mathbf{E_K}(S)$ to Bob; and proves in ZK: "C is encryption of sudoku solution under key who's hash is $\mathbf{h}$.

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

1. Alice chooses cryptographic key $\mathbf{K}$, sends $\mathbf{h} = \mathbf{HASH}(\mathbf{K})$.
2. Alice sends encrypted solution $\mathbf{C} = \mathbf{E_K}(\mathbf{S})$ to Bob; and proves in ZK: "C is encryption of sudoku solution under key who's hash is $\mathbf{h}$.
3. Bob makes bitcoin "hash-locked-transaction" to Alice with $\mathbf{h}$.

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

1. Alice chooses cryptographic key $\mathbf{K}$, sends $\mathbf{h} = \mathbf{HASH}(\mathbf{K})$.
2. Alice sends encrypted solution $\mathbf{C} = \mathbf{E}_{\mathbf{K}}(S)$ to Bob; and proves in ZK: "C is encryption of sudoku solution under key who's hash is $\mathbf{h}$.
3. Bob makes bitcoin "hash-locked-transaction" to Alice with $\mathbf{h}$.
4. Alice reveals $\mathbf{K}$ to unlock her funds.

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

1. Alice chooses cryptographic key $\mathbf{K}$, sends $\mathbf{h} = \mathbf{HASH}(\mathbf{K})$.
2. Alice sends encrypted solution $\mathbf{C} = \mathbf{E}_{\mathbf{K}}(\mathbf{S})$ to Bob; and proves in ZK: "C is encryption of sudoku solution under key who's hash is $\mathbf{h}$.
3. Bob makes bitcoin "hash-locked-transaction" to Alice with $\mathbf{h}$.
4. Alice reveals $\mathbf{K}$ to unlock her funds.
5. Bob can now use $\mathbf{K}$ to decrypt solution.

ZK + bitcoin: Zero-Knowledge contingent payments (by Greg Maxwell)

1. Alice chooses cryptographic key $\mathbf{K}$, sends $\mathbf{h} = \mathbf{HASH}(\mathbf{K})$.
2. Alice sends encrypted solution $\mathbf{C} = \mathbf{E}_{\mathbf{K}}(\mathbf{S})$ to Bob; and proves in ZK: "C is encryption of sudoku solution under key who's hash is $\mathbf{h}$.
3. Bob makes bitcoin "hash-locked-transaction" to Alice with $\mathbf{h}$.
4. Alice reveals $\mathbf{K}$ to unlock her funds.
5. Bob can now use $\mathbf{K}$ to decrypt solution.

More on the mathy side: Schnorr's discrete log protocol

Given g^x, prove you know x without revealing it.

More on the mathy side: Schnorr's discrete log protocol

Given $X := g^x$, prove you know x without revealing it.

1. Prover chooses random r, sends $R := g^r$.

More on the mathy side: Schnorr's discrete log protocol

Given $\mathbf{X} := \mathbf{g}^{x}$, prove you know x without revealing it.

1. Prover chooses random $\mathbf{r}$, sends $\mathbf{R} := \mathbf{g}^{r}$.
2. Verifier chooses random $\mathbf{c}$

More on the mathy side: Schnorr's discrete log protocol

Given $X := g^x$, prove you know x without revealing it.

1. Prover chooses random r, sends $R := g^r$.
2. Verifier chooses random c
3. Prover sends $u := x \cdot c + r$

More on the mathy side: Schnorr's discrete log protocol

Given $\mathbf{X} := \mathbf{g}^{x}$, prove you know x without revealing it.

1. Prover chooses random $\mathbf{r}$, sends $\mathbf{R} := \mathbf{g}^{\mathbf{r}}$.
2. Verifier chooses random $\mathbf{c}$
3. Prover sends $\mathbf{u} := \mathbf{x} \cdot \mathbf{c} + \mathbf{r}$
4. Verifier checks $\mathbf{X} \cdot \mathbf{R} = \mathbf{g}^{\mathbf{u}}$.

In parallel to ZK, a big breakthrough: succinct verification

- 1990, sumcheck - "Can prove a sudoku *doesn't* have a solution without verifier going through all options"

In parallel to ZK, a big breakthrough: succinct verification

- 1990, sumcheck - “Can prove a sudoku *doesn’t* have a solution without verifier going through all options”
- 1998, PCP theorem - “The proof that a sudoku puzzle has a solution can be encoded such that the verifier only needs to read three bits”

Zero-knowledge and succinctness - a love story

- Succinct verification+merkle trees $\longrightarrow$ small proofs
- When the proof is small easier to make it zk - less places information can hide.

A note on efficient zero-knowledge proofs and arguments.

(extended abstract)

Joe Kilian
NEC Research Institute
Princeton, NJ 08540

Abstract

In this note, we present new zero-knowledge interactive proofs and arguments for languages in NP. To show that $x \in L$, with an error probability of at most 2^{-k}, our zero-knowledge proof system requires $O(|x|^{c_1}) + O(\lg^{c_2}|x|)k$ ideal bit commitments, where c_1 and c_2 depend only on L. This construction is the first in the ideal bit commitment model that achieves large values of k more efficiently than by running k independent iterations of the base interactive proof system. Under suitable complexity assumptions, we exhibit a zero-knowledge arguments that require $O(\lg^{c}|x|)kl$ bits of communication, where c depends only on L, and l is the security parameter for the prover.[1] This is the first construction in which the total amount of communication can be less than that needed to transmit the NP witness. Our protocols are based on efficiently checkable proofs for NP [4].

1 Introduction.

1.1 The problem of efficient security amplification.

The standard definition of interactive proofs[7] requires that the verifier accept a correct proof and reject an incorrect assertion with probability at least $\frac{2}{3}$. As there are few applications where a 1/3 error probability is acceptable, one usually tries to obtain an error probability less than 2^{-k}, where k is some easily adjustable security parameter. The most obvious way of achieving this security amplification is take a protocol with a 1/3 error probability, run it $O(k)$ times, and have the verifier accept or reject by majority vote.[2] Are there any more efficient ways of achieving security than by this simple technique? As we will show, the answer is yes, for a wide variety of languages, in a well known model for which no other amplification technique was previously known.

Succinct arguments in a nutshell

Public program $\mathbf{T}$, public output z.

Succinct arguments in a nutshell

Public program $\mathbf{T}$, public output z.

Want to prove "I know input x for program $\mathbf{T}$ that generates output z.

Succinct arguments in a nutshell

Public program $\mathbf{T}$, public output z.

Want to prove "I know input x for program $\mathbf{T}$ that generates output z.

Want proof size and verification time to be much smaller than run time of $\mathbf{T}$.

(SNARK:=Succinct Non-Interactive Argument of Knowledge)

Succinct arguments in a nutshell

Public program $\mathbf{T}$, public output z.

Want to prove "I know input x for program $\mathbf{T}$ that generates output z.

Want proof size and verification time to be much smaller than run time of $\mathbf{T}$.
(SNARK:=Succinct Non-Interactive Argument of Knowledge)

Arithmetization [LFKN,......]: Reduce claim to claim of form "I know polynomials that satisfy some identity"

Succinct arguments in a nutshell

Public program $\mathbf{T}$, public output z.

Want to prove "I know input x for program $\mathbf{T}$ that generates output z.

Want proof size and verification time to be much smaller than run time of $\mathbf{T}$.
(SNARK:=Succinct Non-Interactive Argument of Knowledge)

Arithmetization [LFKN,......]: Reduce claim to claim of form "I know polynomials that satisfy some identity"

Succinct arguments in a nutshell

Advantage of claims about polynomials is that suffice to check at one random point

Succinct arguments in a nutshell

Advantage of claims about polynomials is that suffice to check at one random point

But need to solve "chicken and egg problem":
Prover must commit to polynomials before knowing the challenge point.

Polynomial commitment schemes [KZG, 10]

- Prover send short commitment $\mathbf{cm}(\mathrm{f})$ to polynomial.

Polynomial commitment schemes [KZG, 10]

- Prover send short commitment $\mathbf{cm}(\mathbf{f})$ to polynomial.
- Later Verifier can choose value $\mathbf{i} \in \mathbb{F}$.

Polynomial commitment schemes [KZG, 10]

- Prover send short commitment $\mathbf{cm}(\mathbf{f})$ to polynomial.
- Later Verifier can choose value $\mathbf{i} \in \mathbb{F}$.
- Prover sends back $z = \mathbf{f}(\mathbf{i})$; together with proof $\mathbf{open}(\mathbf{f}, \mathbf{i})$ that z is correct.

Polynomial commitment schemes [KZG, 10]

- Prover send short commitment $\mathbf{cm}(\mathbf{f})$ to polynomial.
- Later Verifier can choose value $\mathbf{i} \in \mathbb{F}$.
- Prover sends back $z = \mathbf{f}(\mathbf{i})$; together with proof $\mathbf{open}(\mathbf{f}, \mathbf{i})$ that z is correct.

KZG give us PCS with commitments and openings are practically 32 bytes.
Notation: $[x] = \mathbf{g}^x$ where $\mathbf{g}$ generator of elliptic curve group.

Elliptic curve pairings - some serious math magic

Groups $\mathbf{G}, \mathbf{G_t}$ such that

- $\mathbf{G}$ is an EC with hard discrete log - from $\mathbf{g}^x$ hard to find x, for generator $\mathbf{g} \in \mathbf{G}$.

Elliptic curve pairings - some serious math magic

Groups $\mathbf{G}, \mathbf{G_t}$ such that

- $\mathbf{G}$ is an EC with hard discrete log - from $\mathbf{g}^x$ hard to find x, for generator $\mathbf{g} \in \mathbf{G}$.
- We have a map $e : \mathbf{G} :\longrightarrow \mathbf{G_t}$ such that

$$e(\mathbf{g}^a, \mathbf{g}^b) = \mathbf{g}_t^{a \cdot b}$$

Notation: $[x] = \mathbf{g}^x$ where $\mathbf{g}$ generator of elliptic curve group.

The KZG polynomial commitment scheme

Setup: $[\mathbf{1}], [x], \ldots, \left[x^{\mathbf{d}}\right]$, for random $x \in \mathbb{F}$.

The KZG polynomial commitment scheme

Setup: $[1], [x], \ldots, [x^d]$, for random $x \in \mathbb{F}$.

$\mathbf{cm}(f) := [f(x)]$

The KZG polynomial commitment scheme

Setup: $[1], [x], \ldots, \left[x^{d}\right]$, for random $x \in \mathbb{F}$.

$\mathbf{cm}(f) := [f(x)]$

$\mathbf{open}(f, i) := [h(x)]$, where $h(X) := \frac{f(X) - f(i)}{X - i}$

The KZG polynomial commitment scheme

Setup: $[1], [x], \ldots, [x^d]$, for random $x \in \mathbb{F}$.

$\mathbf{cm}(f) := [f(x)]$

$\mathbf{open}(f, i) := [h(x)]$, where $h(X) := \frac{f(X) - f(i)}{X - i}$

$\mathbf{verify}(\mathbf{cm}, \pi, z, i)$:

$$e(\mathbf{cm} - [z], [1]) \stackrel{?}{=} e(\pi, [x - i])$$

Multiset equality check

Given $\mathbf{a}, \mathbf{b} \in \mathbb{F}^3$, want to check

$$\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\} \stackrel{?}{=} \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$$

Multiset equality check

Given $\mathbf{a}, \mathbf{b} \in \mathbb{F}^3$, want to check
$\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\} \stackrel{?}{=} \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$

Choose random $\gamma \in \mathbb{F}$. Check

$$(\mathbf{a}_1+\gamma)(\mathbf{a}_2+\gamma)(\mathbf{a}_3+\gamma) \stackrel{?}{=} (\mathbf{b}_1+\gamma)(\mathbf{b}_2+\gamma)(\mathbf{b}_3+\gamma)$$

Multiset equality check

Given $\mathbf{a}, \mathbf{b} \in \mathbb{F}^3$, want to check
$\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\} \stackrel{?}{=} \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$

Choose random $\gamma \in \mathbb{F}$. Check

$$(\mathbf{a}_1+\gamma)(\mathbf{a}_2+\gamma)(\mathbf{a}_3+\gamma) \stackrel{?}{=} (\mathbf{b}_1+\gamma)(\mathbf{b}_2+\gamma)(\mathbf{b}_3+\gamma)$$

If $\mathbf{a}, \mathbf{b}$ different as sets then w.h.p products different.

Multiset equality check

Given $\mathbf{a}, \mathbf{b} \in \mathbb{F}^3$, want to check
$\{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\} \stackrel{?}{=} \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$

Choose random $\gamma \in \mathbb{F}$. Check

$$(\mathbf{a}_1+\gamma)(\mathbf{a}_2+\gamma)(\mathbf{a}_3+\gamma) \stackrel{?}{=} (\mathbf{b}_1+\gamma)(\mathbf{b}_2+\gamma)(\mathbf{b}_3+\gamma)$$

If $\mathbf{a}, \mathbf{b}$ different as sets then w.h.p products different.

Multiset equality check - polynomial version

Given $f, g \in \mathbb{F}_{<d}[X]$, want to check
$\{f(x)\}_{x \in H} \stackrel{?}{=} \{g(x)\}_{x \in H}$ as multisets

Reduces to:

$H = \{\alpha, \alpha^2, \dots, \alpha^n\}$.

$\mathcal{P}$ has sent $f', g' \in \mathbb{F}_{<n}[X]$.

Wants to prove:

$$\prod_{i \in [n]} f(\alpha^i) = \prod_{i \in [n]} g(\alpha^i)$$

$f := f' + \gamma, g := g' + \gamma$

Multiplicative subgroups:

$\mathbf{H} = \{\alpha, \alpha^2, \ldots, \alpha^n = 1\}$.

L_i is i'th lagrange poly of $\mathbf{H}$:

$$L_i(\alpha^i) = 1, L_i(\alpha^j) = 0, j \neq i$$

Checking products with $\mathbb{H}$-ranged protocols [GWC19]

1. $\mathcal{P}$ computes Z with $Z(\alpha) = \mathbf{1}, Z(\alpha^i) = \prod_{j<i} f(\alpha^j)/g(\alpha^j)$.
2. Sends Z to $\mathbf{I}$.

Checking products with $\mathbb{H}$-ranged protocols [GWC19]

1. $\mathcal{P}$ computes $\mathbf{Z}$ with
   $\mathbf{Z}(\alpha) = \mathbf{1}, \mathbf{Z}(\alpha^i) = \prod_{j<i} \mathbf{f}(\alpha^j)/\mathbf{g}(\alpha^j)$.
2. Sends $\mathbf{Z}$ to $\mathbf{I}$.
3. $\mathcal{V}$ checks following identities on $\mathbf{H}$.
   3.1 $\mathbf{L_1}(\mathbf{X})(\mathbf{Z}(\mathbf{X}) - \mathbf{1}) = \mathbf{0}$
   3.2 $\mathbf{Z}(\mathbf{X})\mathbf{f}(\mathbf{X}) = \mathbf{Z}(\alpha \cdot \mathbf{X})\mathbf{g}(\mathbf{X})$

Checking products with $\mathbf{H}$-ranged protocols [GWC19]

1. $\mathcal{P}$ computes Z with
   $Z(\alpha) = 1, Z(\alpha^i) = \prod_{j<i} f(\alpha^j)/g(\alpha^j)$.
2. Sends Z to $\mathbf{I}$.
3. $\mathcal{V}$ checks following identities on $\mathbf{H}$.
   3.1 $L_1(X)(Z(X) - 1) = 0$
   3.2 $Z(X)f(X) = Z(\alpha \cdot X)g(X)$