A Succinct Story of Zero Knowledge

NALIN BHARDWAJ, University of California, San Diego, USA

> *If I had but the time and you had but the brain ...*
>
> Lewis Carroll, The Hunting of the Snark

We study the development of "production" zero-knowledge proofs starting from their theoretical roots in computational complexity theory and the PCP theorem, ultimately creating a summary of the most important developments in the study of zero-knowledge proofs.

Key Words and Phrases: complexity theory, cryptography, zero knowledge

1 INTRODUCTION

Zero-knowledge proofs are extremely interesting primitives to study from a practical perspective. zk-SNARKs and zk-STARKs have captured a lot of attention from the blockchain/crypto space in recent years, going from theoretical research paper constructions to programmable privacy-preserving production circuits in just a few years. Starting with [Sasson et al. 2014]'s Zcash in 2014, over the past 5 years we've seen the advent of several decentralized blockchain applications that would be otherwise infeasible without ZK proofs. These include DarkForest: a completely decentralized strategy game using ZK proofs to create a "fog-of-war" [Dark Forest Team 2020] and MACI: a primitive for collusion resistant digital voting [Buterin 2019] among many others. Deeper than just the application layer, there have also been fundamental protocol-layer ideas such as the Mina protocol [Bonneau et al. 2020], StarkNet [StarkWare [n.d.]] and Cairo [Goldberg et al. 2021] from StarkWare, ZKSync [Gluchowski 2021], etc. that are blockchain scaling solutions using ZK SNARK/STARKs to securely scale transaction throughput in a decentralized ledger. Unsurprisingly, Ethereum creator Vitalik Buterin, too, expects "...ZK-SNARKs to be a significant revolution as they permeate the mainstream world over the next 10-20 years." [Buterin 2021]

On the theoretical counterpart, looking at the subject from the perspective of complexity theory, cryptographic zero-knowledge proofs that exist in production now are very much a consequence of a lot of early work in complexity theory, starting from Interactive Protocols (IP) leading up to the seminal PCP theorem, which directly translated into strategies for creating SNARGs, and further, led advances in the follow-up zero-knowledge SNARK/STARKs, the current state of the art in practical zero-knowledge proofs.

2 INTERACTIVE PROOFS: *IP*

2.1 Deterministic $IP = NP$

Informally, deterministic IP define the class of problems where a *prover* can convince a *verifier* of a *witness*, i.e. that the *prover* knows an input that satisfies a particular language in an (interactive) environment where they can communicate with each other.

Formally, let there exist a language L and a pre-image $x \in L$. Let the prover and verifier have a *transcript* (a log of messages back and forth) π. Then, an interactive proof system is given by a verification algorithm V (run by the verifier) with the properties:

1. **Completeness**: True assertions have valid proofs. That is, if $x \in L$, then $\exists \pi$ such that $V(x, \pi)$ accepts.
2. **Soundness**: False assertions have no proof. That is, if $x \notin L$, then $\forall \pi V(x, \pi)$ rejects.
3. **Efficiency**: $V(x, \pi)$ stops in time $poly(|x|)$.

From the definition, it is clear that $NP \subseteq$ Deterministic IP, since with simply $\pi = \{x\}$, for all languages in NP, V can verify the witness in poly-time.

We can also show the other direction: Deterministic $IP \subseteq NP$. Assume there exists $L \in$ Deterministic IP. Then, there exists a transcript $\pi = (\pi_1, \pi_2, \ldots \pi_k)$ between the prover and the verifier for some k. Then, since we know that the verifier runs in poly-time, $|\pi|$ must also be poly-time. Further, since the verifier is deterministic, the prover can just simulate the verifier to generate the transcript between them, and with this transcript as witness, the verifier can verify the solution in poly-time. So, in totality, we have shown deterministic IP's equivalence to NP. This characterisation of IP is so popular now that even NP is sometimes defined this way (c.f. [Sipser 1996]).

Given that deterministic IP is *just* NP, it's not immediately clear why we should care about it. What if we add randomness, allowing the verifier to be probabilistic? That is, letting the verifier produce false positives or true negatives with some small (practically negligible) probability. Formally, we let the prover P encode the previously mentioned conditions of deterministic IP as follows:

1. **Completeness**: If $x \in L$, then the transcript π is accepted with at least $2/3$ probability by V.
2. **Soundness**: If $x \notin L$, then for all possible provers P, the transcript π is rejected with at least $2/3$ probability by V.
3. **Efficiency**: $V(x, \pi)$ stops in time $poly(|x|)$.

Intuitively, one might compare this probabilistic IP class with other complexity classes that use randomness. For instance, with BPP, the class of problems solvable in polynomial time by a probabilistic Turing Machine. For BPP, [Bennett and Gill 1981] proved that $BPP \subseteq P/poly$ and in fact, it is conjectured that $P = BPP$ (for instance, in [Goldreich 2011]). Given this information about BPP, intuitively, one might guess that adding randomness to IP is not significantly helpful.

But, on the contrary, we will show that adding this randomness actually makes IP quite powerful.

Author's address: Nalin Bhardwaj, nalinbhardwaj@nibnalin.me, University of California, San Diego, USA.

2.2 $IP = PSPACE$[1]

Not only is the entire polynomial hierarchy PH contained in IP (as shown by [Lund et al. 1992]), but IP is $PSPACE$! To that end, first, we show that $IP \subseteq PSPACE$. Let there exist a language $L \in IP$ and a verifier V for it. We show that a prover can compute the maximum probability with which some element $x \in L$ is accepted by the prover in $PSPACE$, and therefore, $IP \subseteq PSPACE$. Consider a decision tree of possibilities of the interactive protocol where each path of length i from the root to the node corresponds to some prefix $\pi_{1,\ldots,i}$ of a transcript π. Since V runs in polynomial time, the depth of the tree is at most polynomial, and each node has at most $O(2^{poly(n)})$ children since each message has at-most polynomial length. In this tree, let's define an auxiliary attribute A: for each node i, A_i is the maximum probability with which the prover can make the verifier accept a proof such that the prefix of the transcript corresponds to the path that ends at node i, i.e. $Path_{root,\ldots i} = \pi_{1,\ldots,i}$. We can compute this using dynamic programming on trees: For each leaf, we assign $A_i = 1$ if V accepts at that leaf or $A_i = 0$ otherwise. For all internal nodes, let C_i = the set of children of a node i. For an internal node where the next message is a response from the prover, we can set $A_i = max(\{A_c \ \forall c \in C_i\})$ and otherwise to $A_i = mean(\{A_c \ \forall c \in C_i\})$. Therefore, in this setup, A_{root} is the maximum probability with which the prover can make the verifier accept the input, and since we can run this computation in $PSPACE$, $IP \subseteq PSPACE$.

Now, we show $PSPACE \subseteq IP$ by showing that a $PSPACE-complete$ language is in IP. This was originally proven by [Shamir 1992], but here we demonstrate a simpler proof presented in [Shen 1992].

We work with the $PSPACE-complete$ language of totally quantified boolean formulae($TQBF$) which consists of quantified formula of the form $\forall x_1 \exists x_2 \ldots Q_n x_n \phi(x_1, \ldots x_n)$ where ϕ is a 3CNF and $Q_i = \forall$ if i is odd and $Q_i = \exists$ if i if even. (Note that we actually only require Q to be an alternating sequence of $\exists$ and $\forall$, a symmetric proof if we switch the parity of the starting element.)

Arithmetization: We convert ϕ into an arithmetic polynomial using the following transformation:

1. *Variable*: $\overline{x_i} \longrightarrow 1 - x_i$
2. *Clause*: $C_j = (x_a \vee \overline{x_b} \vee x_c) \longrightarrow P_j = 1 - (1 - x_a)x_b(1 - x_c)$
3. *Formula*: $\phi(x_1, x_2, \ldots x_n) = C_1 \wedge C_2 \wedge \ldots C_m \longrightarrow P(x_1, x_2, \ldots x_n) = \prod_{j=1}^{m} P_j(x_1, x_2 \ldots x_n)$

Clearly, the outputs of the polynomial P are equivalent to that of the formula ϕ on binary inputs $\{0,1\}^n$ and P.

Next, we note that if we fix values $x_1, \ldots x_{n-1}$, then $P(x_1, \ldots x_{n-1}, 0) \cdot P(x_1, \ldots x_{n-1}, 1)$ evaluates 1 if and only if $\forall x_n \phi(x_1, \ldots x_n)$ is true. Similarly, if Q_n was $\exists$, we note that $1 - (1 - P(x_1, \ldots x_{n-1}, 0)) \cdot (1 - P(x_1, \ldots, x_{n-1}, 1))$ evaluates to 1 iff $\exists x_n \phi(x_1, \ldots x_n)$ is true. With these observations, let $\prod_{x_n \in \{0,1\}} P(x_1, \ldots x_n)$ be defined as $P(x_1, \ldots x_{n-1}, 0) \cdot P(x_1, \ldots x_{n-1}, 1)$ and let $\coprod_{x_n \in \{0,1\}} P(x_1, \ldots x_n)$ be defined as $1 - (1 - P(x_1, \ldots x_{n-1}, 0)) \cdot (1 - P(x_1, \ldots, x_{n-1}, 1))$. Then, we can rewrite the original quantified boolean formula $\forall x_1 \exists x_2 \cdot$

Protocol 1: Naive TBQF

Prover **Verifier**

$F_1 \leftarrow \coprod_{x_2} \cdots \prod_{x_n} \phi(x_1, \ldots x_n)$ $\longrightarrow$

Check

$F_1(0) \cdot F_1(1) = 1$

Select random r_1

$\xleftarrow{r_1}$

$F_2 \leftarrow \prod_{x_3} \cdots \prod_{x_n} \phi(r_1, \ldots x_n)$ $\longrightarrow$

Check

$F_1(r_1) =$

$1 - (1 - F_2(0)(1 - F_2(1)))$

Select r_2

•

•

•

$\forall x_n \phi(x_1, \ldots x_n)$ is true iff

$$\prod_{x_1 \in \{0,1\}} \coprod_{x_2 \in \{0,1\}} \cdots \prod_{x_n \in \{0,1\}} \phi(x_1, \ldots x_n) = 1 \tag{1}$$

Based on this, we describe a naive protocol in **??**. Immediately, it's clear that the protocol is not efficient: The degree of each polynomial F_i is $O(2^m \cdot 3m)$. This follows from the observation that for x_1, the degree doubles with each $\prod / \coprod$).

To address this issue, we observe that in equation (1), each $x_i \in \{0,1\}$. But this implies, for any $a \in \mathbb{Z}^+$, $x_i^a = x_i$.[2] So by reducing the power of any intermediate polynomials to 1, we obtain a simplified representation of equation (1) of the form:

$$\prod_{x_1 \in \{0,1\}} R_{x_1} \coprod_{x_2 \in \{0,1\}} R_{x_2} \cdots \prod_{x_n \in \{0,1\}} R_{x_n} \phi(x_1, \ldots x_n) = 1 \tag{2}$$

where R_{x_i} is the reduced degree polynomial corresponding to the variable x_i. Notice that for all the reduced R_{x_i} except the innermost n R_{x_i}, the degree of the intermediate polynomials is at most 2, and for the innermost R_{x_i}, the degree is upper bounded at $3m$.

With this clever trick, we can create a modified version of the Naive protocol that successfully proves $TBQF$: At the ith step in the protocol, we prove the ith polynomial in our equation. If it is of the form of a $\prod$ or $\coprod$, we proceed similarly to the Naive protocol. Otherwise, if it's of the form R_{x_j} (for some j), we simply check if $F_{i-1}(r_i) = F_i$, with the variables F_i and r_i being analogous to the Naive protocol as well.

We now show that this satisfies all conditions to be a IP protocol:

1. *Completeness*:, if the prover is not cheating, they can simply compute the polynomial with the randomness and send it back to the verifier and the equations will hold for any future values of randomness r_i.

[1] This section is based on [Katz 2011].

[2] This observation is also the primary contribution made by [Shen 1992] in his variant of [Shamir 1992]'s original proof.

(2) *Soundness*: By the Schwartz–Zippel lemma, the probability that the verifier's check holds true but the polynomials are not actually equal is d/q, where d is the degree of the polynomial and q is the size of the prime field we're working in. We now compute a strict upper bound for d: For each operator $\prod$ or $\coprod$, there is error $1/q$, error $3m/q$ for the last n operators, and error $2/q$ for all operators in the form of R_i. Overall, we have error $\frac{n}{q} + \frac{3mn}{q} + \frac{2}{q} \cdot O(n) = O(\frac{mn+n^2}{q})$, which is clearly sufficient to obtain negligible error with the size of the field being $poly(m, n)$.

(3) *Efficiency*: Efficiency follows since all polynomials have length $poly(n)$, and therefore any comparisons, multiplications etc. can be computed symbolically in $poly(n)$ time.

This finishes our proof that IP is, indeed, extremely powerful. Essentially, we have derived a protocol for $TQBF$ by transforming a boolean formula into a polynomial. The key idea, then, is to "strip" each polynomial in the equation, randomly fixing a variable in the polynomial and checking if previous answers agree with the answers in the future.

2.3 Arthur-Merlin Games

We now look at an interesting variant of the IP class, AM: Arthur-Merlin games. This is a restricted class of IP in which the verifier cannot use "private" randomness, i.e. a verifier's messages are simply completely random messages in some distribution. [3]

2.3.1 $AM = IP$. It is trivially seen that $AM \subseteq IP$ since it's simply an additional restriction on any language in IP. But, in fact, $AM = IP$. While we won't formally prove this, we present an intuitive explanation of this: Notice that in the proof for $PSPACE \subseteq IP$ in 2.2, we described a protocol that is already in AM (since all verifier messages are random). This is a clear indication that AM is *at least* as powerful as IP, but in fact, since $PSPACE$ is an "upper bound" for IP (and consequently AM), $AM = PSPACE$ and thus $IP = AM$.

Arthur-Merlin Games are a precursor to SNARGs in that they show us that, in theory, powerful zero-knowledge proofs can exist in a decentralised environment like a public blockchain, where all computational intermediates are readable by an adversary prover/verifier. Using an AM protocol allows the prover and verifier to leave their internals public, and they can safely use public randomness oracles (for instance, recent blockchain block hashes or randomness from a prior trusted setup).

3 ZERO KNOWLEDGE[4]

While interactive protocols (and Arthur Merlin Games) are extremely interesting as standalone constructs, we now turn our focus to protocols that possess the "zero-knowledge" property. Informally, a zero-knowledge proof is a proof that does not allow the verifier to learn anything other than the fact that a particular assertion is being proven true.

More formally, we define a zero-knowledge proof system as one where there exists some probabilistic poly-time, non-interactive algorithm S(called the *simulator*) which, when given any verifier and any pre-image $x \in L$, produces output that is indistinguishable from the verifier's point of view of the interaction with the prover. Therefore, the zero-knowledge property in an interactive proof allows a prover to not leak any unnecessary information about its pre-image while proving the assertion securely via the witness. Thus, zero-knowledge is a *security property*([noa 2021]), and just like other security properties, there are two variants of zero-knowledge:

(1) *Computational security*: This type of security relies on the hardness of computational problems (for instance, problems in NP). These rely on the idea that it is computationally infeasible for the verifier to brute-force and find a pre-image that corresponds to a particular witness/transcript.

(2) *Statistical security*: Such security relies on the idea that the distribution of the witness/transcript generated for an arbitrary pre-image is statistically "close" to that of any other pre-images, virtually making it indistinguishable from those pre-images for the verifier. This notion of proximity is formalised by the notion of *negligible function*([noa 2020]): the distance function between the witnesses/transcripts in the domain of pre-images is a negligible function.

Next, we present a classic example of statistical zero-knowledge proofs to show a flavour of the ideas underlying practical constructions of such proof systems:

3.1 Graph 3-coloring[5]

A 3-coloring of an undirected graph $G = (V, E)$ is an assignment $C : V \longrightarrow \{R, G, B\}$ (say "Red," "Green," and "Blue" colors) such that no pair of adjacent vertices are assigned the same colour. We show that there exists a statistical zero-knowledge proof system for the language of graphs G that are 3-colorable.

Assume there exists a one-way function f that both the verifier and prover initially agree on using. Then we describe a protocol as follows:

(1) *Prover*: Say, the prover has a 3-coloring C it wants to prove. Let τ be a permutation of $\{R, G, B\}$ selected uniformly at random, and let $C' = \tau \cdot C$. Then, for all $v \in V$, let boxes $D_v = f(C'_v)$. Send the boxes D to the Verifier.

(2) *Verifier*: Choose a random edge $(u, v) \in E$, and send it to the prover.

(3) *Prover*: Send C'_u and C'_v to the verifier, essentially unboxing D_u and D_v.

(4) *Verifier*: Check that C'_u and C'_v are distinct, and that the prover's commitments match the claimed pre-images, i.e. $D_u = f(C'_u)$ and $D_v = f(C'_v)$. Reject if not.[6]

We now prove that this is a zero-knowledge interactive protocol:

[3] The name for this class originates from the folk tale of King Arthur and Merlin the Magician — The Merlin (the prover) is an all-powerful being trying to convince King Arthur (the verifier) of an assertion, but given Merlin's psychic powers, King Arthur cannot possibly keep his randomness secret from him.

[4] This section borrows from [Vadhan 2007] and its references.

[5] This section is due to [Goldreich et al. 1991]

[6] This application of a one-way function is formally a cryptographic primitive called a *commitment scheme*. A commitment scheme is said to be *hiding*, i.e. it does not reveal anything about the input in the commitment D and *binding*, i.e. the prover cannot cheat by using multiple inputs that generate the same commitment.

Completeness: Note that if C is a valid 3-coloring, then C' is a valid 3-coloring as well for any τ. Thus, if C is a valid 3-colouring, the verifier definitely accepts.

Soundness: If C is not a valid coloring, then there must exist some edge $(u, v) \in E$ such that $C'_u \neq C'_v$, so the verifier must reject with probability at least $1/|E|$. If we simply repeat this protocol $O(|E|)$ times, the probability that the verifier accepts an invalid coloring is $(1 - |E|)^{O(|E|)} < 1/3$.

Efficiency: Efficiency follows from a similar argument as Soundness, and thus details are omitted for brevity.

Zero-Knowledge: Now, we look at the most interesting property of this protocol. Intuitively, let's consider what the verifier learns in the course of the protocol: Since D is established via a one-way function, the only information the verifier learns is C'_u and C'_v for some $(u, v) \in E$. But then, $C' = \tau \cdot C$, i.e. it is simply an independent, random permutation of the complete colouring — all the information learnt is something the verifier could have simulated on its own. More substantially, we notice that this information learnt by the verifier is, in fact, something "close" to the information the verifier would have learnt for *all* other possible colourings (including invalid ones), implying that the protocol does indeed satisfy the statistical zero-knowledge condition.

And now, for the big reveal, we note that standard complexity theory results tell us that graph 3-colouring is an *NP*-complete language! This means that we have just shown that if one-way functions exist, all languages in *NP* have zero-knowledge proof systems. This is quite powerful, and a more general result follows:

If one-way functions exist, then every language in NP has both a computational zero-knowledge proof system and a statistical zero-knowledge argument system.

We skip the counterpart proof for computational zero-knowledge proof systems since the statistical zero-knowledge argument system already provides a strong intuition for the ideas used in demonstrating both kinds of zero-knowledge proof security.

Later, it turned out that even the assumption that one-way functions exist is extraneous. [Ben-Or et al. 1988] first proved this result by defining a new *multi-prover* model for interactive proofs. While their original application for this model was only to derive a proof free of the one-way function requirement, it turned out that their model for interactive proofs was significantly more powerful than even they originally expected. Let's see how.

4 *MULTIPROVER* INTERACTIVE PROOFS: *MIP*

MIP is a variant of *IP* defined by [Ben-Or et al. 1988] where instead of just one prover, two independent provers initialise on some common input but are unable to communicate with each other after starting communication with the verifier, essentially allowing the verifier to cross-check any assertions between the two provers. [Ben-Or et al. 1988] were able to show that with this variant, the intractability assumptions of one-way functions were unnecessary to prove the existence of zero-knowledge proofs for all languages in *NP*.

In fact, as [Babai et al. 1991] later show, *MIP* is as powerful, if not more, than just *IP*, $MIP = NEXPTIME$, i.e. the class of problems solvable by nondeterministic Turing Machines in exponential time. While the relationship between *NEXPTIME* and *PSPACE* itself is not well known, certainly $PSPACE \subseteq NEXPTIME$ since standard results tell us $PSPACE \subseteq EXP \subseteq NEXPTIME$. Note, however the strictness of both the results is unknown (*PSPACE* vs. *EXP* and *EXP* vs. *NEXPTIME*).

Another consequential result (that may be seen as a weak precursor to the *PCP* theorem) was $MIP(poly) = MIP(1)$, i.e. a system with polynomially many rounds can be converted into one with a constant number of rounds and further, a system with any constant number of provers can be converted to one with 2 provers. This is quite a powerful result since it implies all generalisations, at least on the axes of provers and rounds collapse to $MIP = NEXPTIME$. This was shown by [Lapidot and Shamir 1997].

While on one hand, as with *MIP*, complexity theory saw progress by loosening constraints and defining new generalisations of the original *IP* protocol, on the other end of the spectrum, progress was made by considering restrictions on *IP*, such as with the *PCP* theorem.

5 PCP: SUCCINCTNESS AND THE HARDNESS OF APPROXIMATION

A probabilistically checkable proof (*PCP*) is a type of proof that can be checked by a randomized algorithm using a bounded amount of randomness and reading a bounded number of bits of the proof. Much like randomised *IP*, a *PCP* verifier is an algorithm that accepts correct proofs and rejects incorrect ones with high probability. While there is no obvious connection between *PCP*s and interactivity, most research into *PCP* started from the point of view of interactivity, and protocols like *MIP* and randomised algorithms were at the intersection of complexity theory and cryptography at the time.

5.1 PCP theorem: $NP = PCP[O(\log n), O(1)]$

Formally, the PCP Theorem states that $NP = PCP[O(\log n), O(1)]$, where the first part($O(\log n)$) refers to the number of bits of randomness required and the second part($O(1)$) refers to the bits of proof the verifier algorithm needs to read. In English, this means that for any language in *NP*, it is possible to generate a proof such that to verify it with high probability, a verifier only needs to generate logarithmic bits of randomness and only needs to read a constant number of bits in the order of the length of the proof.

The *PCP* theorem has been one of the crowning achievements of complexity theory. Consequences of the *PCP* theorem not only had a strong impact on cryptographers' quest for succinct zero-knowledge proofs but also complexity theory more widely since they provided very strong bounds for approximating *NP* languages in general.

It is hard to overstate the seeming impossibility of this result. In some metaphysical sense, it means that we can write math proofs millions or billions of pages long that only need a small amount of computation for anyone else to verify. Unsurprisingly, this theorem was the culmination of a series of clever ideas and observations over the span of many papers, authors and years of hard work.

5.2 Simpler *PCP* results

As a warm-up, we take a quick look at how the framing of *PCP* relates to the usual complexity theoretical classes.

5.2.1 $coRP = PCP[poly(n), O(1)]$. This follows from the definition of *coRP*: it is the set of languages for which there exists a probabilistic Turing Machine that runs in polynomial time, is allowed to generate randomness, and correctly detects false instances and detects correct instances with high probability.

5.2.2 $NP = PCP[0, poly(n)]$. This, too, follows from the definition of *NP*, when framed in the manner we did in section 2.1.

5.2.3 $PCP[0, O(\log n)] = P = PCP[O(\log n), 0]$. . In the first part of the equality, since the machine can only read a logarithmic order of bits, these bits can be brute-forced by a polynomial-time TM anyway. Similarly, the latter part follows since a polynomial-time TM can try all possible strings of randomness of length $O(\log n)$ in polynomial time, not requiring the randomness at all.

5.3 $NP = PCP[O(\log n), O(1)]$[7]

Finally, we now look at the ideas behind the core *PCP* theorem and its consequences. This is meant to be an informal introduction to the core ideas of the proof, a formal reference can be found in [Arora et al. 1998] and [Dinur 2007].

Let us turn to our old example problem of graph 3-colouring from section 3.1. Since we know graph 3-colouring is an *NP*-complete problem, it is sufficient to come up with a scheme to solve it in $PCP[O(\log n), O(1)]$ to prove our assertion. So, let's look at our original solution. The verifier had to make $O(|E|)$ queries and generate $O(|E|)$ random bits, so clearly that solution is in $PCP[poly(n), poly(n)]$.

How can we improve it? It seems like we *need* to make $O(|E|)$ queries because the prover could have found some colouring that is *almost* a 3-colouring, with maybe a couple faulty edges (like the graph in **??**). The only way to gain confidence about discovering such faults is to query almost all edges. The core implication of the *PCP* theorem is that there is a way to *amplify* such errors, to a point where we only need to make a constant number of queries to detect any such errors with high probability. Essentially, we want to find a transformation from any graph G into another graph H such that if G fails to have a 3-colouring, H fails to have a 3-colouring **badly**.

Formally, we will find a way to transform any graph G to a new graph H such that if G has a 3-colouring, H has a 3-colouring, but if G has no 3-colouring, i.e. for any 3-colouring there exists at least one edge such that it does not satisfy the colouring constraint, in H, for any 3-colouring, a significant fraction (say 10%) of the edges do not satisfy the colouring constraint. This 10% fraction is called the "gap" between a correct 3-colouring and any other 3-colouring.

Let's skeletonize our variables. Let $f : V \longrightarrow \{R, G, B\}$ be a 3-coloring of the nodes. Next, define $g((u,v)) = \begin{cases} 1 & \text{if } f(u) = f(v) \\ 0 & \text{if } f(u) \neq f(v) \end{cases}$ for an edge $(u,v) \in E$. Finally, let $\delta_f = \frac{\sum_{(u,v)\in E} g((u,v))}{|E|}$ for some 3-coloring f. Further, let $\pi_G = \min_{\forall f} \delta_f$, i.e. π_G is the fault rate of the least faulty 3-coloring of a graph G.

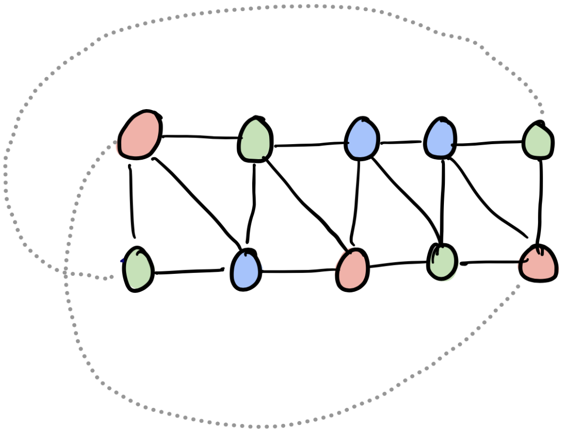

Fig. 1. Example of a graph that cannot be 3-colored but has an almost perfect coloring with just one fault. The dotted grey lines denote that the circles are copies of the same node.

5.3.1 Next, we claim that we can create a transformation $G \longrightarrow H$ such that $\pi_H \geq 2 \cdot \pi_G$. (Provided $\pi_G < c$ for some arbitrary large constant c and G is an expander graph.)

If we succeed in creating such a general transformation, we can double the fault rate repeatedly to obtain a graph with a "high" fault rate. There are multiple ways to create such a transformation. The original paper ([Arora and Safra 1998], [Arora et al. 1998]) used special error-correcting codes, whose roots lie in the field of coding theory. At this point, the connection between the problem we're solving and coding theory should be somewhat intuitive, but due to the complexity of this solution, we do not study it here. Instead, we turn to a newer, more understandable solution by [Dinur 2007] based on ideas of gap amplification with expander graphs[8]. Here, we present some intuition for the core ideas of [Dinur 2007]'s proof:

5.3.2 Step 0: Preprocess. As a first step, we take any graph G and embed it in a larger expander graph H such that $|V_H| \leq c \cdot |V_G|$ for some constant c. There are multiple simple ideas to accomplish this, so we omit too much detail. One example method for a constant-degree graph G would be to take the union of edges in G and some other constant degree expander graph G' with the same set of vertices.

5.3.3 Step 1: Inhale. Let $G = (V_G, E_G)$. Define $H = (V_H, E_H)$. $V_H = V_G$ and $E_H = \{(u,v) : dist_G(u,v) \leq t\}$ for some constant c. Further, for each node $v \in V_G$, let COL_v be the set of "local" colouring of v and its neighbourhood N_v: i.e. a colouring of just the neighbourhood of the node v. Informally, a "local" colouring of a node is some valid colouring of the subgraph of the node and its neighbours in the larger graph. Then, for each edge $(u,v) \in E_H$, we can check the consistency of the local colourings COL_u and COL_v. Two neighbouring local colourings are "consistent" if the colourings

[7] This section is primarily based on a Lecture Series by Irit Dinur, starting with [Irit Dinur 2019].

[8] Informally, edge expander graphs refer to undirected multigraphs in which every subset of vertices that is not "very large" itself has a "large" neighbourhood. While details vary by use case, "very large" can be intuitively thought of as $poly(|V|)$ and "large" as at least a constant in our case.

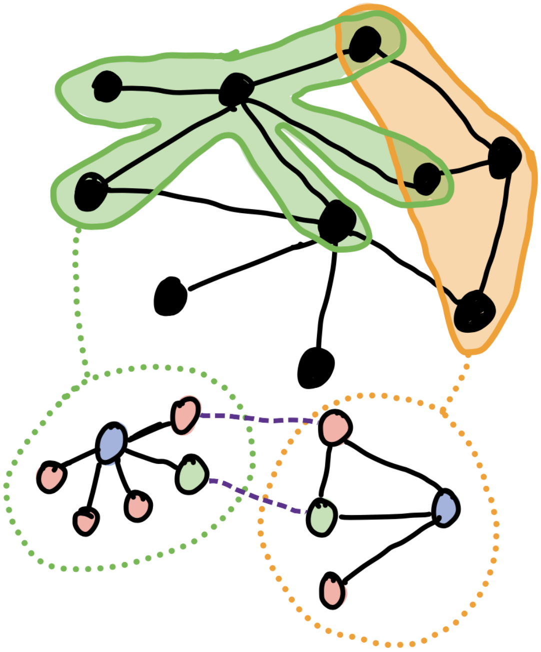

Fig. 2. Example of a sheaf over a graph: The green and orange highlights show two neighbourhoods with some overlapping nodes. The colourings in the dotted green and orange circles are both "local" colourings of the neighbourhoods, and we look for consistencies in the local colourings (specifically in the colourings of the nodes connected by purple lines, since those represent the same nodes in the original graph)

of the nodes in the intersection $N_u \cap N_v$ can be assigned in a manner consistent with each other. Similar to before, in this new graph

$$g((u,v)) = \begin{cases} 0 & \text{if } COL(u) \text{ is consistent with } COL(v) \\ 1 & \text{otherwise} \end{cases} \quad \forall (u,v) \in E_H$$

and we have properties similar to the ones previously described, but on H. Notably, we can prove that $\delta(H) \geq 2 \cdot \delta(G)$. An intuitive explanation of why this happens is based on the properties of expander graph. In essence, we've taken a sheaf over balls of radius t in G, and any faulty edge in this ball is propagated to balls around it. This way, we're "spreading" the errors in any individual edge to edges in the vicinity.

5.3.4 Step 2: Exhale. One critical mismatch between the properties of H and the aforementioned definitions in G, however, is that we don't naturally have a 3-colouring of H that we can use to repeat the process of doubling δ. The solution to this problem is to use a weaker form of the *PCP* theorem recursively to return H to a 3-colouring. Without getting into the technical details, this is a "solved" problem by the use of PCP encoding in the form of a local replacement gadget. For instance, one method is based on Hadamard encoding/linearity testing ideas (see [Dinur 2007] theorem 1.8 for more information).

In total, we've created a gap amplification algorithm that with each round of inhale and exhale spreads the faulty edges around the graph to obtain a new graph that has $O(c \cdot |G|)$ (for some constant c) vertices. We only need to perform this doubling a logarithmic number of times, so we've obtained a transformation to a new graph where we only need to run the protocol we started within section 3.1 a constant number of times to be confident in the provers witness.

5.4 Hardness of Approximation

We now make a quick note about the consequences of the *PCP* theorem on the *hardness of approximation*. We know that finding a 3-colouring is a hard problem (unless $P = NP$). But perhaps, let's consider the problem of finding a 3-colouring that's *almost* correct, say, it satisfies 99.99% of the edges in the graph. Is this problem also hard?

Say we use the transformation described in the previous section to obtain a representation H of any graph G. If we have an algorithm to correctly color at least 99.99% of the edges, we would have been able to use it on this transformed graph H, and then, since we've obtained a colouring that fails for fewer edges than the previously described "gap", it must be that this colouring of H corresponds to a colouring of G that is perfect. But then, we would have an algorithm for perfectly 3-colouring any graph G. So, even approximating colouring must be hard!

It is quite cool to see that a result that started from the study of vanilla interactive protocols has yielded us an extremely powerful complexity theory result at large: approximation is hard. Interestingly enough, this connection between *PCP* and the hardness of approximation was made in [Feige et al. 1991] even before the actual theorem itself was proven.

6 FROM PCP TO CRYPTOGRAPHIC PROOFS[9]

Now that we know that in theory, it is possible to create efficient proofs that only require reading a few bits of the proof and only require a small amount of randomness, we consider how to construct an actual SNARG — a succinct non-interactive argument of knowledge — that can accomplish this. Then, we'll use this foundation to build up SNARGs into SNARKs/STARKs that additionally achieve knowledge soundness.

6.1 Succinctness: Kilian's Notarised Envelopes

The first attempt at productionising *PCP* was made by [Kilian 1992] in his protocol based on the idea of "Notarised envelopes" powered by Merkle trees.

Essentially, Merkle trees allow a prover to commit to a long string (say π) and generate proofs that $\pi_i = x$ for some index i in the string. Merkle trees depend on collision-resistant hashing and work by constructing a binary tree of commitments of subranges on top of the original string, much like the cumulation in classic data structures like segment trees or Fenwick trees.

Let's say a prover is trying to prove to a verifier that some input $x \in L$ such that L has a *PCP* proof system.

Clearly, this protocol is quite succinct! The verifier makes only a constant number of queries, and each query is answered with Merkle path proofs of logarithmic length of the PCP proof system. Secondly, this protocol is sound. Since the prover commits the entire *PCP* proof π independent of the bits that are checked by the verifier, the

[9] This section is based on [Nitulescu [n.d.]]

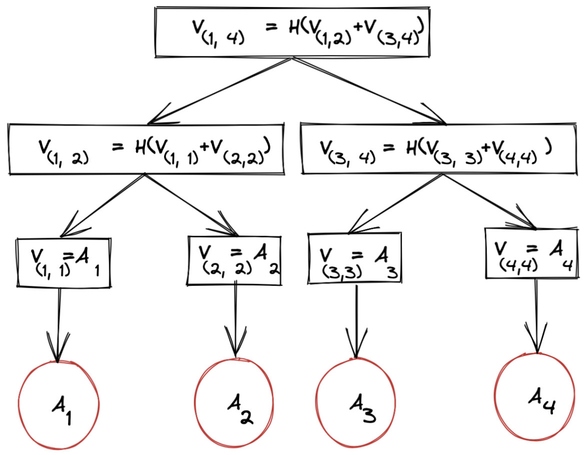
$V_{(1,4)} = H(V_{(1,2)}+V_{(3,4)})$
$V_{(1,2)} = H(V_{(1,1)}+V_{(2,2)})$
$V_{(3,4)} = H(V_{(3,3)}+V_{(4,4)})$
$V_{(1,1)} = A_1$
$V_{(2,2)} = A_2$
$V_{(3,3)} = A_3$
$V_{(4,4)} = A_4$
A_1
A_2
A_3
A_4

Fig. 3. Example construction of a merkle tree from a *PCP* string *A*

1: Verifier sends a collision-resistant hash function H, to be used by the prover for the Merkle tree construction.
2: Prover uses the witness for x to construct a *PCP*π for the input $x \in L$. It then constructs a Merkle tree using π at the leaves and sends the Merkle tree root hash back to the verifier.
3: The verifier then samples $O(\log n)$ randomness bits for the *PCP* verification and sends them to the prover.
4: The prover computes a set of indices, say $r_1, r_2 \ldots r_k$ to query in the *PCP* proof based on the verifier's randomness and opens $\pi_{r_1}, \pi_{r_2}, \ldots \pi_{r_k}$, generating Merkle path proofs for each of the indices, and sends them back to the verifier.
5: The verifier passes if the openings for each of the indices are valid and if a *PCP* verifier running with the generated randomness would have opened those exact indices.

Protocol 2: Kilian's Merkle Tree *PCP* protocol

soundness of this protocol is essentially equivalent to the soundness of the underlying *PCP* proof system. Note that this protocol is a plain SNARG, so it does not attempt to be zero-knowledge.

Notice that this described protocol is not quite *interactive.* While Kilian's protocol is theoretically sufficient to obtain a succinct proof exploiting the locally checkable property of *PCP*s, in practice, having the ability to perform multiple rounds in an interactive proof is quite useful (reasons for this are detailed in [Ben-Sasson et al. 2016]). With this motivation, [Ben-Sasson et al. 2016] described a very natural extension of [Kilian 1992]'s protocol that combines both aspects to create a hybrid construct: a *PCIP*. Like the moniker suggests, a *PCIP* is simply *PCP* followed by an *IP*. Instead of simply sending and querying a single *PCP* proof, the verifier is allowed to respond with random challenges after the first round, and each time the prover generates additional *PCP* strings to convince the verifier more reliably.

6.2 Non-interactive: Fiat-Shamir heuristic

Next, we look at a heuristic that takes any interactive *AM* proof and converts it into a non-interactive proof. Practically, non-interactivity unlocks several powerful use cases. On blockchains, for instance, this allows for a verifiable ledger for the proof since it's significantly easier to verify non-interactive proofs in a distributed system settings.

The original idea of the heuristic was presented in [Fiat and Shamir 1987] for a specific problem and easily generalised to any non-interactive public coin protocol. The primary idea of the heuristic is to use one random number and repeatedly hash it to obtain more randomness and replace the public coins in the original protocol with this new randomness.

To prove the security of such a transformation, the hash function is modelled in the Random Oracle Model (ROM) (from [Bellare and Rogaway 1993] and [Canetti et al. 2004]) as a truly random function. In practice, since such truly random hash functions do not exist, this model of security is somewhat weak and depends on the assumption that there exist "good enough" hash functions in practice that make any attacks infeasible.

With the power of the Fiat-Shamir heuristic in mind, we turn again to Kilian's protocol. Applying the collision-resistant hash function H to the Merkle root hash of the *PCP* string π, we can obtain new randomness under the heuristic. So, we can just use this to generate the verifier's *PCP* queries, converting the original four-round interactive protocol into a non-interactive one! The formalization of this non-interactive construction is credited to [Micali 1994].

6.3 Zero-Knowledge Arguments: PIR + ECRH

Next, let's think about adapting Kilian's protocol to be zero-knowledge. To do so, we first define two primitives:

6.3.1 Private Information Retrieval: PIR. PIR is a protocol that allows a user to retrieve items from a server that possesses some database without revealing information about the item retrieved. For our purposes, it is sufficient to formalise a PIR protocol to consist of 3 algorithms:

(1) *ENC* which inputs some query, say q, and returns an encrypted representation of it, say Q.
(2) *EVAL* which outputs an encrypted answer, say A_Q, to an encrypted database query Q
(3) *DEC* which decrypts an encrypted answer A_Q into the constituent database items

If we assume we can create such algorithms succinctly and securely, a user can encrypt their query using *ENC*, ask the server to generate an answer using *EVAL* and decrypt the answer using *DEC*.

6.3.2 Extractable Collision-Resistant Hash: ECRH. is a collision-resistant hash function for which it is possible to algorithmically "extract" a pre-image for any output in the image of the function. That is, for an ECRH f, given $y \in Im(f)$, we can find x such that $f(x) = y$.

With these two primitives, we turn again to Kilian's protocol and consider how we can use them.

[Di Crescenzo and Lipmaa 2008] first discovered a use-case for *PIR* modifying Kilian's protocol. As a first message, the verifier may send a *PIR*-encrypted encoding of the *PCP* queries. Then, the prover can assemble a large database of every possible query a verifier could have made (along with Merkle path proofs that satisfy their committed root hash), and then use this database to *PIR*-evaluate the verifier's queries and send them back. We can see why this modified protocol is sound: a prover essentially needs to be honest to be able to assemble valid answers to the *PIR* encrypted queries. Notice that even as a standalone, this method can be used to transform Kilian's interactive protocol into a non-interactive one without depending on the security of the Fiat-Shamir heuristic.

Next, we look at the [Bitansky et al. 2017] construction, which builds on [Di Crescenzo and Lipmaa 2008] in an extremely interesting way. Their primary idea is to use a protocol similar to [Di Crescenzo and Lipmaa 2008], but replace the *CRHF* in the Merkle tree construction with an *ECRH*. They also modify the [Di Crescenzo and Lipmaa 2008] protocol to *PIR*-encrypt a verifiers random coins (instead of their actual queries), allowing verifiers to run offline — independent of particular instances of a prover.

Essentially, *ECRH*'s extractability allows their construction to take a root hash for a Merkle tree and extract it out all the way to the leaves, i.e. extract out an entire proof by just recursively applying the extraction algorithm for *ECRH*. Naively doing so, however, would incur an additional polynomial-time blowup with each level, so we can only extract $O(1)$ times in practice. To deal with this, they modify the structure of the Merkle tree, making it so each internal node can have a polynomial number of children rather than just binary.

With this change, we can start to see how we can show Knowledge soundness for the protocol. To convince a verifier, the recursively extracted string must contain valid answers to the PCP queries specified in its PIR queries. If not, we can show by a reduction that we can find collisions within the hash function. In particular, a collision finder could simulate the *PIR*-encryption and obtain two paths that map to the same root but must differ somewhere (as one is satisfying and the other is not), obtaining a collision.

Further, if the verifier extracts a set of arbitrary leaves that are satisfying with respect to the *PIR*-encrypted queries, the same set of leaves must also be satisfying for almost all other possible PCP queries and are thus sufficient for witness-extraction. Formally, we can see a contradiction as if this was not the case then one would be able to use the polynomial-size extraction circuit to break the security of the *PIR*.

The [Bitansky et al. 2017] construction is quite clever overall. It achieves a communication complexity and a verifier time complexity that is polynomial in the security parameter, the size of the instance, and log of the time it takes to verify a valid witness for the instance, obtaining full succinctness!

6.4 ZK-SNARKs

It is no surprise that the bolded letters of previous parts of this section spell out ZK-SNARKS — it is because they have together defined all the pieces necessary to create a fully working zk-SNARK!

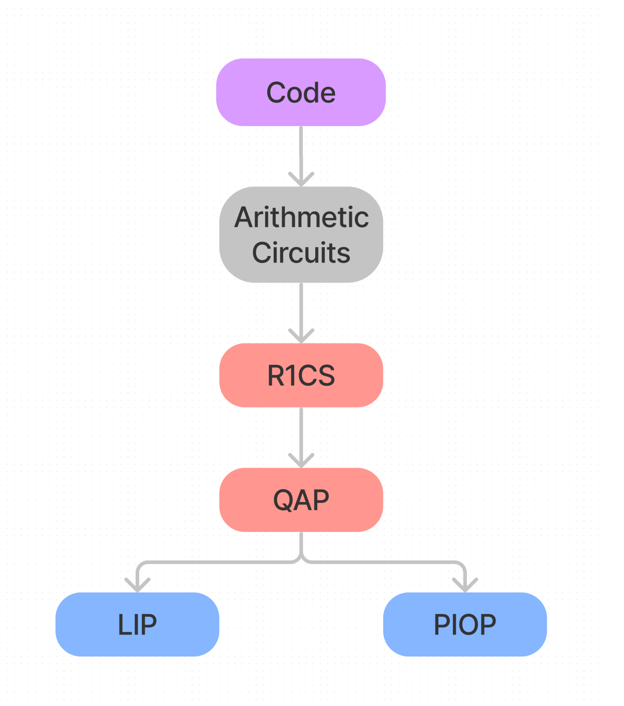

Fig. 4. Steps in the compilation of a zero-knowledge SNARK. Source: Unknown, but appears to be frequently credited to cryptographer Eran Tromer.

While this implementation of zk-SNARKs is not quite the state of the art, it is extremely close and was only put together in its entirety in 2014 (by preprints of [Bitansky et al. 2017]).

Having wade through this long journey between complexity theory and cryptography, we leave the reader with a stanza from Lewis Carroll's *The Hunting of the Snark*:

For the Snark's a peculiar creature, that won't
Be caught in a commonplace way.
Do all that you know, and try all that you don't:
Not a chance must be wasted today!

7 STATE-OF-THE-ART SNARKS: FROM CODE TO PROOFS[10]

At this level of understanding, while it should make sense that at least in theory, we can write succinct zero-knowledge proofs for any problems in *NP*, practically generating zero-knowledge proofs for arbitrary pieces of computation still feels like an opaque problem. So, we now turn to the state-of-the-art production zk-SNARK schemes. Many of these involve quite a few more interesting pieces that come together to create a much more "generalised" zk-SNARK. Given that these constructions involve a decent bit of math/cryptography, we'll only focus on a high-level overview of how these schemes convert code to proofs:

[10] This section is based on [Gabizon 2017].

7.1 Code: Arithmetic Circuits

Of course, first, when we say "code", we need to define a formal encoding of this "code". We start with an *arithmetic circuit* representation of our computation: an arithmetic circuit consists of gates performing addition and multiplication, and wires that carry values in a field $\mathbb{F}$. Notice that the more common boolean circuits can be converted to arithmetic circuits using some simple algebraic manipulation.

Reasons for choosing this particular encoding will be clearer as we go forward, but for now, we note the from standard complexity theory the result that polynomial sized circuits are equivalent to polynomial-time Turing machines (up to a log factor) so, at the very least, circuits are powerful encodings for any computation we might want to do.

Practically, code for zk-SNARKs is most commonly written using the circuit language Circom[11], which was designed by iden3 particularly for this purpose. As one would expect from such novel technology, Circom is quite low-level and feels much like Verilog or other circuit simulating programming languages.

7.2 Rank-1 Constraint System: R1CS

Next, we take our arithmetic circuit and convert it into an R1CS. An R1CS is a sequence of three vectors $\{a, b, c\}$ and a solution vector s, where the equation $(a \cdot s) \cdot (b \cdot s) = (c \cdot s)$, i.e. the product of the dot products of a and b with s equals the dot product of c with s. You may convince yourself that we can transform any gates into constraints of this form with some simple algebraic manipulation on the inputs and outputs of a gate. By creating such constraints, we'll have a tuple (a, b, c) of vectors. Each vector would have size equal to the total number of inputs to all the gates in the circuit, say N, for future reference.

7.3 Quadratic Arithmetic Programs: QAP

A QAP is pretty much just an R1CS that is represented as a polynomial. We've already seen some ideas to perform such a transformation in 2.2, so we omit much detail, but simply note that this transformation uses Lagrange interpolation to go from groups of three vectors of length n to n groups of three degree-3 polynomials, where evaluating the polynomials at each x coordinate represents one of the gates in the original circuit. Essentially, a QAP is the transformation of a large n constraint R1CS to a single polynomial expressing all the constraints of the circuit, making any checks and further applications much more efficient.

7.4 Linear Interactive Proof/Polynomial Interactive Oracle Proof

At this point, different constructions introduce different ideas for checking a QAP as a zero-knowledge proof. Omitting the mathematical details, every scheme finds a way to efficiently hide s, while proving its existence to the verifier. There are two primary ways to achieve this:

(1) *Linear Interactive Proof (LIP)* is an interactive proof where every message sent by a prover (including malicious ones) must be a linear function of the previous messages sent by the verifier. [Bitansky et al. 2013c] show a transformation from any *PCP* into a Linear *PCP* (a *PCP* where the "honest" function must be a linear function) and further, a transformation from a linear *PCP* into a two-message *LIP*, and then construct a SNARK from *LIP*s with low-degree verifiers. These are essentially generalisations of [Groth 2010] and [Gennaro et al. 2013] constructions.

(2) *Polynomial Interactive Oracle Proof* In a Polynomial Interactive Oracle Proof (PIOP), the prover sends low degree polynomials to the verifier, and rather than reading the entire list of coefficients, the verifier queries evaluations of these polynomials at random points, and combined with the Fiat-Shamir heuristic, this yields a zk-SNARK.

7.5 Kate Commitments

While we have not gone into a lot of detail about *PIOP* proof construction, Kate commitments are the underlying cryptographic primitive that enables them. Even outside of the study of zk-SNARKs, polynomial commitments are highly ubiquitous, so we make a short note on the general construct of polynomial commitments.

Kate commitments, or KZG commitments, are a commitment scheme introduced by [Kate et al. 2010] that are extremely powerful constructs that allow a prover to commit a polynomial and show evaluations of that polynomial on any point with a single group element-sized proof in constant time. As such, Kate commitments and its variants lie at the heart of many *PIOP*-style zk-SNARK constructions (such as Marlin from [Chiesa et al. 2020a] and PLONK from [Gabizon et al. 2019]).

We also write a short note on the use of KZG commitments as a vector commitment in Appendix A.

8 OPEN QUESTIONS & FUTURE WORK

8.1 Fiat-Shamir-Compatible Hash Functions

Earlier, we described the limitation of the Fiat-Shamir heuristic in that its security is so far only proven in the Random Oracle Model, which, of course, are somewhat shaky grounds to build on. It remains an open question whether there exist concrete hash functions that are compatible with the Fiat Shamir heuristic, i.e. if there exist hash functions that guarantee soundness for a transformed proof (and are also compatible with zero-knowledge proofs). There has been a lot of work on this topic, but there is still no known universal hash function that can be proven to be compatible without making strong, somewhat impractical, assumptions yet. On one hand, there has been research such as [Bitansky et al. 2013b] which shows that for three-round proofs, Fiat-Shamir transform using a concrete hash family cannot be proven to be sound via black-box reduction to standard hardness assumptions. On the other hand, recent work such as [Canetti et al. 2019] has shown constructions for a class of protocols whose security can be proven under assumptions stronger than the original hardness assumptions.

8.2 Recursive ZK Proofs

Recursive zero-knowledge SNARKs are one of the most interesting use cases for zero-knowledge technology that are *almost* ready for

[11] https://docs.circom.io

practical use. As the name suggests, recursive ZK SNARKs unlock the ability to take a proof and verify it inside another proof, allowing for a lot more composability without blowing up proving/verifying times. Besides the real-world composability advantages, recursive zero-knowledge SNARKs are also of great interest because of their use case as a blockchain scaling solution, relying on the succinctness feature of SNARKs to *compress* secure verification of long blockchains. Mina Protocol ([Bonneau et al. 2020]) is currently implementing such an approach.

It should, however, come as no surprise that efficient recursive ZK SNARK constructions are quite hard to pull off. With the typical method of SNARK constructions that uses elliptic curve fields, the initial, hairy problem to solve is to find a pairing-friendly curve (see [Bitansky et al. 2013a]). Attempting to solve for this yields many interesting tradeoffs. Many of the options and possibilities have been explored in recent works such as Halo [Bowe et al. 2019], which uses elliptic curve cycles that do not require pairings and Fractal [Chiesa et al. 2020b], that eliminates the use of elliptic curves altogether. Overall, the study of efficient recursive ZK proofs is of great importance, and ripe territory for further exploration.

8.3 Post-Quantum Zero-Knowledge

Many of the techniques described in previous sections depend on the computational hardness of certain problems, such as computing the discrete logarithm, which have been shown solvable by quantum computers in polynomial time by [Shor 1997]. Therefore, quantum computers may pose a significant threat to the security model and practicality of zk-SNARKs that use these techniques. There has been some work on making quantum attack resistant zk-SNARKs. For instance, [Gennaro et al. 2018] recently proposed a lattice-based zk-SNARK starting with SSP — square span programs (an alternative to the QAP intermediate for boolean circuits). Lattice problems are known to hold against quantum attacks [Ajtai 1996], so this is a very promising line of work.

8.4 Optimisations

While this isn't of much complexity theory interest, there has recently been a lot of work on improving the constant factor on verifying/constructing zk-SNARKs. Interestingly, besides just time, the practical use of SNARKs on blockchains such as Ethereum has motivated research aiming to optimise SNARK "gas" consumption. There is a monetary cost, called the gas cost, associated with each instruction executed in Ethereum smart contracts, so it is of significant practical utility to find ways to reduce the gas costs associated with verifying zero-knowledge proofs on blockchains. For instance, [Gabizon and Williamson 2021]'s FFLONK, a modification of the popular PLONK zk-SNARK scheme, was motivated by gas cost reduction. They discovered a modification to KZG commitments that allows for reducing the scalar multiplications necessary to verify the proof by nearly three times, albeit at the cost of almost tripling proof construction time. For reference, each scalar multiplication costs $\sim$ 6000 gas, or $4USD in Ethereum at the time of writing.

ACKNOWLEDGMENTS

I would like to thank Prof. Shachar Lovett and the staff of the UC San Diego class CSE 200: Computational Complexity[12] who allowed me, an undergraduate student, to be part of their graduate class offering in the first place, and motivated and allowed me to work on this research paper as a final project for the class.

Additionally, I would like to thank members of 0xPARC[13] and Ethereum Foundation[14]. While they have not directly supported this research project, they've supported related projects and explorations of mine and I was certainly inspired to study zero-knowledge cryptography due to their work in this field in the first place.

A KATE COMMITMENTS AS VECTOR COMMITMENTS

While we haven't discussed Kate Commitment construction in detail, beyond the math of how they work, I find their use cases significantly more insightful. One way to characterise this power is to consider Kate commitments in comparison with Merkle trees (the construct used in Kilian's protocol). In general, commitments like the Merkle tree are called "vector" commitments. A vector commitment is one that allows someone to commit of a vector, and provides the ability to reveal any bits of this commitment with an "opening" proof.

Let's say we wanted to construct a vector commitment scheme using Kate commitments. So, we have a vector we want to encode as a polynomial. To do so, we can use Lagrange interpolation. Then, using Kate commitments, we can prove the value any *subset* of this vector with just one group element! In contrast, if you're using Merkle trees, you would require $O(\log n)$ size path proofs for each element you want to prove.

On this train of thought, it is also interesting to note the construction of Verkle Trees([Kuszmaul 2019]), which are essentially a vector commitment scheme built using KZG commitments. They can emulate all features of Merkle tree, but more efficiently. Incidentally, the state trie – the trie containing information about a blockchain block – in the Ethereum blockchain is encoded using a Merkle tree. Given the practical advantages of time and space efficiency, Verkle Trees are also set to replace Merkle trees as the state trie data structure in the upcoming Ethereum upgrades.

REFERENCES

2020. Negligible function. https://en.wikipedia.org/w/index.php?title=Negligible_function&oldid=979710421 Page Version ID: 979710421.

2021. Security parameter. https://en.wikipedia.org/w/index.php?title=Security_parameter&oldid=1026505640 Page Version ID: 1026505640.

Miklós Ajtai. 1996. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 99–108.

Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof verification and the hardness of approximation problems. *Journal of the ACM (JACM)* 45, 3 (1998), 501–555.

Sanjeev Arora and Shmuel Safra. 1998. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM (JACM)* 45, 1 (1998), 70–122.

László Babai, Lance Fortnow, and Carsten Lund. 1991. Non-deterministic exponential time has two-prover interactive protocols. *Computational complexity* 1, 1 (1991), 3–40.

Mihir Bellare and Phillip Rogaway. 1993. Random Oracles Are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security* (Fairfax, Virginia, USA) *(CCS '93)*. Association for

[12] https://cseweb.ucsd.edu//classes/fa21/cse200-a/

[13] https://0xparc.org

[14] https://ethereum.foundation

Computing Machinery, New York, NY, USA, 62–73. https://doi.org/10.1145/168588.168596

Michael Ben-Or, Shafi Goldwasser, Joe Kilian, and Avi Wigderson. 1988. Multi-Prover Interactive Proofs: How to Remove Intractability Assumptions. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (Chicago, Illinois, USA) *(STOC '88)*. Association for Computing Machinery, New York, NY, USA, 113–131. https://doi.org/10.1145/62212.62223

Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive oracle proofs. In *Theory of Cryptography Conference*. Springer, 31–60.

Charles H. Bennett and John Gill. 1981. Relative to a Random Oracle A, PA != NPA != co-NPA with Probability 1. *SIAM J. Comput.* 10 (1981), 96–113.

Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. 2017. The hunting of the SNARK. *Journal of Cryptology* 30, 4 (2017), 989–1066.

Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013a. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 111–120.

Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana Lopez-Alt, and Daniel Wichs. 2013b. Why "Fiat-Shamir for Proofs" Lacks a Proof. In *10th Theory of Cryptography Conference*, Vol. 7785. 181. https://doi.org/10.1007/978-3-642-36594-2_11

Nir Bitansky, Dana Dachman-Soled, Sanjam Garg, Abhishek Jain, Yael Tauman Kalai, Adriana López-Alt, and Daniel Wichs. 2013c. Why "fiat-shamir for proofs" lacks a proof. In *Theory of cryptography conference*. Springer, 182–201.

Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. 2020. Mina: Decentralized Cryptocurrency at Scale. (2020).

Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Recursive proof composition without a trusted setup. *Cryptol. ePrint Arch., Tech. Rep* 1021 (2019), 2019.

Vitalik Buterin. 2019. Minimal anti-collusion infrastructure. https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413

Vitalik Buterin. 2021. vitalik.eth on Twitter. https://twitter.com/VitalikButerin/status/1433228277263462401

Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. 2019. Fiat-Shamir: From Practice to Theory. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Phoenix, AZ, USA) *(STOC 2019)*. Association for Computing Machinery, New York, NY, USA, 1082–1090. https://doi.org/10.1145/3313276.3316380

Ran Canetti, Oded Goldreich, and Shai Halevi. 2004. The random oracle methodology, revisited. *Journal of the ACM (JACM)* 51, 4 (2004), 557–594.

Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. 2020a. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12105)*. Springer. https://doi.org/10.1007/978-3-030-45721-1_26

Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. 2020b. Fractal: Post-quantum and transparent recursive proofs from holography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 769–793.

Dark Forest Team. 2020. Announcing Dark Forest. https://blog.zkga.me/announcing-darkforest

Giovanni Di Crescenzo and Helger Lipmaa. 2008. Succinct NP proofs from an extractability assumption. In *Conference on Computability in Europe*. Springer, 175–185.

Irit Dinur. 2007. The PCP theorem by gap amplification. *Journal of the ACM (JACM)* 54, 3 (2007), 12–es.

U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. 1991. Approximating clique is almost NP-complete. In *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*. 2–12. https://doi.org/10.1109/SFCS.1991.185341

Amos Fiat and Adi Shamir. 1987. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO' 86*, Andrew M. Odlyzko (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 186–194.

Ariel Gabizon. 2017. Explaining SNARKs Part I: Homomorphic Hidings. https://electriccoin.co/blog/snark-explain/

Ariel Gabizon and Zachary J Williamson. 2021. fflonk: a Fast-Fourier inspired verifier efficient version of PlonK. *Cryptology ePrint Archive* (2021).

Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Report 2019/953. https://ia.cr/2019/953.

Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. 2013. Quadratic span programs and succinct NIZKs without PCPs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 626–645.

Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. 2018. Lattice-based zk-SNARKs from square span programs. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 556–573.

Alex Gluchowski. 2021. Introducing ZK Sync: the missing link to mass adoption of Ethereum. https://blog.matter-labs.io/introducing-zk-sync-the-missing-link-to-mass-adoption-of-ethereum-14c9cea83f58

Lior Goldberg, Shahar Papini, and Michael Riabzev. 2021. Cairo–a Turing-complete STARK-friendly CPU architecture. *Cryptology ePrint Archive* (2021).

Oded Goldreich. 2011. In a world of P= BPP. In *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation*. Springer, 191–232.

Oded Goldreich, Silvio Micali, and Avi Wigderson. 1991. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM (JACM)* 38, 3 (1991), 690–728.

Jens Groth. 2010. Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 321–340.

Irit Dinur. 2019. The PCP theorem. https://www.ias.edu/video/HermannWeyl/2019/1118-IritDinur publisher: Institute for Advanced Study.

Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010*, Masayuki Abe (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 177–194.

Jonathan Katz. 2011. Notes on Complexity Theory: Lecture 19. https://www.cs.umd.edu/~jkatz/complexity/f11/lecture19.pdf. [Online; accessed 27-Dec-2021].

Joe Kilian. 1992. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 723–732.

John Kuszmaul. 2019. Verkle trees. *Verkle Trees* (2019), 1–12.

Dror Lapidot and Adi Shamir. 1997. Fully Parallelized Multi-prover Protocols for NEXP-Time. *J. Comput. System Sci.* 54, 2 (1997), 215–220. https://doi.org/10.1006/jcss.1997.1238

Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. 1992. Algebraic Methods for Interactive Proof Systems. *J. ACM* 39, 4 (oct 1992), 859–868. https://doi.org/10.1145/146585.146605

Silvio Micali. 1994. CS proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE, 436–453.

Anca Nitulescu. [n.d.]. zk-SNARKs: A Gentle Introduction. ([n. d.]).

Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.

Adi Shamir. 1992. IP = PSPACE. *J. ACM* 39, 4 (oct 1992), 869–877. https://doi.org/10.1145/146585.146609

A. Shen. 1992. IP = SPACE: Simplified Proof. *J. ACM* 39, 4 (oct 1992), 878–880. https://doi.org/10.1145/146585.146613

Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (oct 1997), 1484–1509. https://doi.org/10.1137/S0097539795293172

Michael Sipser. 1996. *Introduction to the Theory of Computation* (1st ed.). International Thomson Publishing.

StarkWare. [n.d.]. StarkNet. https://starkware.co/starknet/

Salil Vadhan. 2007. The complexity of zero knowledge. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 52–70.