11

Zero-Knowledge Proofs and Arguments

11.1 What is Zero-Knowledge?

The definition of a zero-knowledge proof or argument captures the notion that the verifier should learn nothing from the prover other than the validity of the statement being proven.[1] That is, any information the verifier learns by interacting with the honest prover could be learned by the verifier on its own without access to a prover. This is formalized via a simulation requirement, which demands that there be an efficient algorithm called the *simulator* that, given only as input the statement to be proved, produces a distribution over transcripts that is indistinguishable from the distribution over transcripts produced when the verifier interacts with an honest prover (recall from Section 3.1 that a transcript of an interactive protocol is a list of all messages exchanged by the prover and verifier during the execution of the protocol).

Definition 11.1 (Informal definition of zero-knowledge)**.** A proof or argument system with prescribed prover $\mathcal{P}$ and prescribed verifier $\mathcal{V}$ for a language $\mathcal{L}$ is said to be zero-knowledge if for any probabilistic polynomial time verifier strategy $\hat{V}$, there exists a probabilistic polynomial time algorithm S (which can depend on $\hat{V}$), called the simulator, such

[1] Recall that a *proof* satisfies statistical soundness, while an argument satisfies computational soundness. See Definitions 3.1 and 3.2.

that for all $x \in \mathcal{L}$, the distribution of the output $S(x)$ of the simulator is "indistinguishable" from $\mathrm{View}_{\hat{\mathrm{V}}}(\mathcal{P}(\mathrm{x}), \hat{\mathrm{V}}(\mathrm{x}))$. Here, $\mathrm{View}_{\hat{\mathrm{V}}}(\mathcal{P}(\mathrm{x}), \hat{\mathrm{V}}(\mathrm{x}))$ denotes the distribution over transcripts generated by the interaction of prover strategy $\mathcal{P}$ and verifier strategy $\hat{V}$ within the proof or argument system.

Informally, the existence of the simulator means that, besides learning that $x \in \mathcal{L}$, the verifier $\mathcal{V}$ does not learn anything from the prover beyond what $\mathcal{V}$ could have efficiently computed herself. This is because, conditioned on x being in $\mathcal{L}$, $\mathcal{V}$ cannot tell the difference between generating a transcript by interacting with the honest prover versus generating the transcript by ignoring the prover and instead running the simulator. Accordingly, any information the verifier could have learned from the prover could also have been learned from the simulator (which is an efficient procedure, and hence the verifier can afford to run the simulator herself).

In Definition 11.1, there are three natural meanings of the term "indistinguishable".

- One possibility is to require that $S(x)$ and $\mathrm{View}_{\hat{\mathrm{V}}}(\mathcal{P}(\mathrm{x}), \hat{\mathrm{V}}(\mathrm{x}))$ are literally the same distribution. In this case, the proof or argument system is said to be *perfect-zero knowledge*.[2]

- Another possibility is to require that the distributions $S(x)$ and $\mathrm{View}_{\hat{\mathrm{V}}}(\mathcal{P}(\mathrm{x}), \hat{\mathrm{V}}(\mathrm{x}))$ have negligible *statistical distance*. In this case, the proof or argument system is said to be *statistical zero-knowledge*.

  Here, the statistical distance (also called total variation distance) between two distributions D_1 and D_2 is defined to be

$$\frac{1}{2} \sum_{y} |\Pr[D_1(x) = y] - \Pr[D_2(x) = y]|,$$

[2] In the context of perfect zero-knowledge *proofs*, it is standard to allow the simulator to abort with probability up to 1/2, and require $S(x)$ to be distributed identically to $\mathrm{View}_{\hat{\mathrm{V}}}(\mathcal{P}(\mathrm{x}), \hat{\mathrm{V}})$ conditioned on $S(x)$ not aborting. This is because, if the simulator is not permitted to abort, no perfect-zero knowledge proofs are known for any non-trivial problems (meaning problems not known to be in **BPP**, the class of problems solvable in randomized polynomial time). This subtlety will not be relevant to this survey.

and it equals the maximum over all algorithms $\mathcal{A}$ (including inefficient algorithms) of

$$|\Pr_{y \leftarrow D_1}[\mathcal{A}(y) = 1] - \Pr_{y \leftarrow D_2}[\mathcal{A}(y) = 1]|,$$

where $y \leftarrow D_i$ means that y is a random draw from the distribution D_i. Hence, if two distributions have negligible statistical distance, then no algorithm (regardless of its runtime) can distinguish the two distributions with non-negligible probability given a polynomial number of samples from the distributions.

- The third possibility is to require that all *polynomial time* algorithms $\mathcal{A}$ cannot distinguish the distributions $S(x)$ and $\text{View}_{\hat{\mathrm{V}}}(\mathcal{P}(\mathrm{x}), \hat{\mathrm{V}}(\mathrm{x}))$ except with negligible probability, when given as input a polynomial number of samples from the distributions. In this case, the proof or argument system is said to be *computational zero-knowledge*.

Accordingly, when someone refers to a "zero-knowledge protocol", there are actually *at least* 6 types of protocols they may be referring to. This is because soundness comes in two flavors—statistical (proofs) and computational (arguments)—and zero-knowledge comes in at least 3 flavors (perfect, statistical, and computational). In fact, there are even more subtleties to be aware of when considering how to define the notion of zero-knowledge.

- **(Honest vs. dishonest verifier zero-knowledge).** Definition 11.1 requires an efficient simulator for *every possible* probabilistic polynomial time verifier strategy $\hat{V}$. This is referred to as malicious- or dishonest-verifier- zero knowledge (though papers often omit the clarifying phrase malicious or dishonest-verifier). It is also interesting to consider only requiring an efficient simulator for the prescribed verifier strategy $\mathcal{V}$. This is referred to as *honest-verifier* zero-knowledge.

- **(Plain zero-knowledge vs. auxiliary-input zero-knowledge).** Definition 11.1 considers the verifier $\hat{V}$ to have only one input, namely the public input x. This is referred to as

plain zero-knowledge, and was the original definition given in the conference paper of Goldwasser *et al.* [133] that introduced the notion of zero-knowledge (along with the notion of interactive proofs). However, when zero-knowledge proofs and arguments are used as subroutines within larger cryptographic protocols, one is typically concerned about dishonest verifiers that may compute their messages to the prover based on information acquired from the larger protocol prior to executing the zero-knowledge protocol. To capture such a setting, one must modify Definition 11.1 to refer to verifier strategies $\hat{V}$ that take two inputs: the public input x known to both the prover and verifier, and an *auxiliary input* z known only to the verifier and simulator, and insist that the output $S(x, z)$ of the simulator is "indistinguishable" from $\text{View}_{\hat{\text{V}}}(\mathcal{P}(\text{x}), \hat{\text{V}}(\text{x}, \text{z}))$. This modified definition is referred to as *auxiliary-input* zero-knowledge. Of course, the distinction between auxiliary-input and plain zero-knowledge is only relevant when considering dishonest verifiers.

An added benefit of considering auxiliary-input computational zero-knowledge is that this notion is closed under sequential composition. This means that if one runs several protocols satisfying auxiliary-input computational zero-knowledge, one after the other, the resulting protocol remains auxiliary-input computational zero-knowledge. This is actually not true for plain computational zero-knowledge, though known counterexamples are somewhat contrived. The interested reader is directed to [54] and the references therein for a relatively recent study of the composition properties of zero-knowledge proofs and arguments.

The reader may be momentarily panicked at the fact that we have now roughly 24 notions of zero-knowledge protocols, one for every possible combination of (statistical vs. computational soundness), (perfect vs. statistical vs. computational zero-knowledge), (honest-verifier vs. dishonest-verifier zero-knowledge), and (plain vs. auxiliary input zero-knowledge). That's $2 \cdot 3 \cdot 2 \cdot 2$ combinations in total, though for honest-verifier notions of zero-knowledge the difference between auxiliary-input and plain zero-knowledge is irrelevant. Fortunately for us, there are

only a handful of variants that we will have reason to study in this monograph, summarized below.

In Sections 11.2–11.4, we briefly discuss statistical zero-knowledge proofs. Our discussion is short because, as we explain, statistical zero-knowledge proofs are not very powerful (e.g., while they are capable of solving some problems believed to be outside of **BPP**, they are not believed to be able to solve **NP**-complete problems). Roughly all we do is describe what is known about their limitations, and then give a sense of what they are capable of computing by presenting two simple examples: a classic zero-knowledge proof system for graph non-isomorphism due to [131] (Section 11.3), and a particularly elegant protocol for a problem called the Collision Problem (this problem is somewhat contrived, but the protocol is an instructive example of the power of zero-knowledge).

In subsequent sections, we present a variety of perfect zero-knowledge arguments. All are non-interactive (possibly after applying the Fiat-Shamir transformation), rendering the distinction between malicious- and honest-verifier (and auxiliary-input vs. plain) zero-knowledge irrelevant.[3, 4, 5]

[3] More precisely, when the Fiat-Shamir transformation is applied to an honest-verifier zero-knowledge proof or argument and is instantiated in the plain model (by replacing the random oracle with a concrete hash function), the resulting non-interactive argument is zero-knowledge so long as the hash family used to instantiate the random oracle satisfies a property called programmability. The result applies even to dishonest verifiers, since non-interactive protocols leave no room for misbehavior on the part of the verifier. Roughly speaking, the simulator for the non-interactive argument obtains a transcript by running the simulator for the interactive argument, and then samples the hash function h used in the Fiat-Shamir transformation at random conditioned on it producing the verifier challenges in the transcript (this ability to perform such conditional sampling of h is what is referred to by programmability). This produces the same distribution over (hash function, transcript) pairs as first selecting h at random, and then having the honest prover use h when applying the Fiat-Shamir transformation to the interactive protocol. See [218] for additional details.

[4] When working in the random oracle model instead of the plain model, there are some subtleties regarding how to formalize zero-knowledge that we elide in this survey (the interested reader can find a discussion of these subtleties in [204], [247]).

[5] For non-interactive arguments that use a structured reference string (SRS), such as the one we describe later in Section 17.5.6, one may consider (as an analog of malicious-verifier zero-knowledge) settings in which the SRS is not generated

Remarks on Simulation. A common source of confusion for those first encountering zero-knowledge is to wonder whether an efficient simulator for the honest verifier's view in a zero-knowledge proof or argument for a language $\mathcal{L}$ implies that the problem can be solved by an efficient algorithm (with no prover). That is, given input x, why can't one run the simulator S on x several times and try to discern from the transcripts output by S whether or not $x \in \mathcal{L}$? The answer is that this would require that for every pair of inputs (x, x') with $x \in \mathcal{L}$ and $x' \notin \mathcal{L}$, the distributions $S(x)$ and $S(x')$ are efficiently distinguishable. Nothing in the definition of zero-knowledge guarantees this. In fact, the definition of zero-knowledge says nothing about how the simulator S behaves on inputs x' that are not in $\mathcal{L}$.

Indeed, it is entirely possible that an efficient simulator S can produce accepting transcripts for a zero-knowledge protocol even when run on inputs $x' \notin \mathcal{L}$. Similarly, in the context of zero-knowledge proofs *of knowledge*, where the prover is claiming to *know* a witness w satisfying some property, the simulator will be able to produce accepting transcripts *without* knowing a witness.

One may initially wonder whether the preceding paragraph contradicts soundness of the protocol: if the simulator can find accepting transcripts for false claims, can't a cheating prover somehow use those transcripts to convince the verifier to accept false claims as valid? The answer is no. One reason for this is that a zero-knowledge protocol may be interactive, yet the simulator only needs to produce convincing *transcripts* of the interaction. This means that the simulator is able to do things like first choose all of the verifier's challenges, and then choose all of the prover's messages in a manner that depends on those challenges. In contrast, a cheating prover must send its message in each round prior to learning the verifier's challenge in that round. So even if the simulator can find accepting transcripts for inputs $x \notin \mathcal{L}$, it will be

properly. For example, the notion of *subversion zero knowledge* demands that zero-knowledge be maintained even when the SRS is chosen maliciously. SNARKs that we describe in this survey that use an SRS can be tweaked to satisfy subversion zero-knowledge [1], [28], [116]. On the other hand, it is not possible for a SNARK for circuit satisfiability to be sound in the presence of a maliciously chosen SRS if the SNARK is zero-knowledge [28].

of no help to a dishonest prover trying to convince $\mathcal{V}$ that $x \in \mathcal{L}$. This will be the situation for the simulators we construct in Section 11.4 for the Collision Problem, and the zero-knowledge proofs of knowledge that we develop in Section 12.2 (e.g., in Schnorr's protocol for establishing knowledge of a discrete logarithm).[6]

Some Final Intuition. Another way of thinking about a zero-knowledge protocol is as follows. If the prover $\mathcal{P}$ convinces the verifier $\mathcal{V}$ to accept, then $\mathcal{V}$ can infer (unless $\mathcal{P}$ got very lucky in terms of the random challenges the verifier happened to send to the prover during the interaction) that $\mathcal{P}$ must have had an effective *strategy* for answering verifier challenges. That is, $\mathcal{P}$ must have been prepared to successfully answer many different challenges that $\mathcal{V}$ *might have* asked (but did not actually ask) during the protocol's execution. If the protocol has low soundness error, this implies that $\mathcal{P}$'s claim is accurate, i.e., that $x \in \mathcal{L}$.

Meanwhile, zero-knowledge guarantees that $\mathcal{P}$'s answers to the *actual* challenges asked by $\mathcal{V}$ during the protocol reveal no other information whatsoever. Put another way, $\mathcal{P}$'s answers to the challenges sent during the zero-knowledge protocol are only useful for convincing $\mathcal{V}$ that $\mathcal{P}$ was prepared to answer other challenges that were not actually asked. $\mathcal{P}$'s preparation reveals to $\mathcal{V}$ that $\mathcal{P}$'s claim is accurate, but reveals nothing else.

This intuition will become clearer in Section 12.2, when we cover so-called special-sound protocols. These are three-message protocols in which the verifier $\mathcal{V}$ sends a single random challenge to the prover (this challenge is the protocol's second message). Following the prover's first message, if $\mathcal{V}$ were somehow able to obtain the prover's answers to *two* different challenges, then $\mathcal{V}$ would indeed learn information from the two responses. This does not violate zero-knowledge because the verifier

[6] A second possible reason that the existence of a simulator may not help a cheating prover is that, if the protocol is private coin, then the simulator can choose the verifier's private coins and use its knowledge of the private coins to produce accepting transcripts, while a cheating prover does not have access to the verifier's private coins. We only cover one such example of a private-coin zero-knowledge protocol: the graph non-isomorphism protocol given in Section 11.3.

in the protocol only interacts with $\mathcal{P}$ once, and can only send a single challenge during that interaction.

11.2 The Limits of Statistical Zero Knowledge Proofs

It is known that any language solvable by a statistical zero-knowledge proof with a polynomial time verifier is in the complexity class $\mathbf{AM} \cap \mathbf{coAM}$ [3], [112].[7] This means that such proof systems are certainly no more powerful than *constant-round* (non-zero-knowledge) interactive proofs, and such proof systems are unlikely to exist for any **NP**-complete problems.[8] In contrast, the SNARKs we give in this survey with polynomial time verifiers are capable of solving problems in **NEXP**, a vastly bigger class than **NP** (and with linear-time verifiers and logarithmic proof length, the SNARKs in this survey can solve **NP**-complete problems). The upshot is that statistical zero-knowledge proof systems are simply not powerful enough to yield efficient general-purpose protocols (i.e., to verifiably outsource arbitrary witness-checking procedures in zero-knowledge). Accordingly, we will discuss statistical zero-knowledge proofs only briefly in this survey. The reason we discuss them at all is because they do convey some intuition about the power of zero-knowledge that is useful even once we turn to the more powerful setting of (perfect honest-verifier) zero-knowledge *arguments*.

11.3 Honest-Verifier SZK Protocol for Graph Non-Isomorphism

Two graphs G_1, G_2 on n vertices are said to be isomorphic if they are the same graph up to labelling of vertices. Formally, for a permutation $\pi\colon \{1,\ldots,n\} \to \{1,\ldots,n\}$, let $\pi(G_i)$ denote the graph obtained by

[7] **AM** (respectively, **coAM**) is the class of languages, membership (respectively, *non*-membership) in which can be established via a 2-message interactive proofs with a polynomial time verifier. Intuitively, **AM** captures "minimally interactive" proofs. There is evidence that such proof systems are no more powerful than non-interactive proofs [169], [193].

[8] If $\mathbf{AM} \cap \mathbf{coAM}$ contains **NP**-complete problems, then $\mathbf{AM} = \mathbf{coAM}$, which many people believe to be false. That is, the existence of efficient one-round proofs of membership in a language does not seem like it should necessarily imply the existence of efficient one-round proofs of *non*-membership in the same language.

replacing each edge (u, v) with $(\pi(u), \pi(v))$. Then G_1 is isomorphic to G_2 if there exists a permutation π such that $\pi(G_1) = G_2$. That is, π is an isomorphism between G_1 and G_2 so long as $(i, j) \in G_1$ if and only if $(\pi(i), \pi(j))$ is an edge of G_2.

There is no known polynomial time algorithm for the problem of determining whether two graphs are isomorphic (though a celebrated recent result of Babai [16] has given a *quasipolynomial* time algorithm for the problem). In Protocol 2, we give a perfect honest-verifier zero-knowledge protocol for demonstrating that two graphs are *not* isomorphic, due to seminal work of Goldreich *et al.* [131]. Note that it is not even obvious how to obtain a protocol for this problem that is not zero-knowledge. While a (non-zero-knowledge) proof that two graphs *are* isomorphic can simply specify the isomorphism π,[9] it is not clear that there is a similar witness for the *non-existence* of any isomorphism.

Protocol 2 Honest-verifier perfect zero-knowledge protocol for graph non-isomorphism

Verifier picks $b \in \{1, 2\}$ at random, and chooses a random permutation $\pi\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$.
Verifier sends $\pi(G_b)$ to prover.
Prover responds with b'.
Verifier accepts if $b' = b$ and rejects otherwise.

We now explain that the protocol in Protocol 2 is perfectly complete, has soundness error at most 1/2, and is honest-verifier perfect zero-knowledge.

Perfect Completeness. If G_1 and G_2 are not isomorphic, then $\pi(G_i)$ is isomorphic to G_i but not to G_{3-i}. Hence, the prover can identify b from $\pi(G_b)$ by determining which of G_1, G_2 it is that $\pi(G)$ is isomomorphic to.

Soundness. If G_1 and G_2 are isomorphic, then $\pi(G_1)$ and $\pi(G_2)$ are identically distributed when π is a permutation chosen uniformly at random over the set of all $n!$ permutations over $\{1, \ldots, n\}$. Hence, statistically speaking, the graph $\pi(G_b)$ provides no information as to the

[9] A perfect zero-knowledge proof for graph isomorphism is also known, an exposition of which can be found in [130, Section 4.3.2].

value of b, which means regardless of the prover's strategy for selecting b', b' will equal b with probability exactly $1/2$. The soundness error can be reduced to 2^{-k} by repeating the protocol k times sequentially.

Perfect honest-verifier zero-knowledge. Intuitively, when the graphs are not isomorphic, the honest verifier cannot possibly learn anything from the prover because the prover just sends the verifier a bit b' equal to the bit b that the verifier selected on its own. Formally, consider the simulator that on input (G_1, G_2), simply chooses b at random from $\{1, 2\}$ and chooses a random permutation π, and outputs the transcript $(\pi(G_b), b)$. This transcript is distributed identically to the honest verifier's view when interacting with the prescribed prover.

Discussion of zero-knowledge. We remark that the protocol is not zero-knowledge against malicious verifiers (assuming there is no polynomial time algorithm for graph isomorphism[10]). Imagine a dishonest verifier that somehow knows a graph H that is isomorphic to one of G_1, G_2, but the verifier does not know which. If the verifier replaces its prescribed message $\pi(G_b)$ in the protocol with the graph H, then the honest prover will reply with the value b' such that H is isomorphic to $G_{b'}$. Hence, this dishonest verifier learns which of the two input graphs H is isomorphic to, and if there is no efficient algorithm for graph isomorphism, then this is information that the verifier could not have computed efficiently on its own.

It is possible to transform this protocol into a proof that is zero-knowledge even against dishonest verifiers. Our sketch of this result follows [130, Section 4.7.4.3]. The rough idea is that if the verifier only sends query graphs H to the prover such that the verifier *already knows* which of G_1, G_2 it is that H is isomorphic to, then the verifier cannot possibly learn anything from the prover's response (as the prover's response is simply a bit b' such that H is isomorphic to $G_{b'}$). Hence, we can insist that the verifier first *prove to the prover* that the verifier knows a bit b such that H is isomorphic to G_b.

For this approach to preserve soundness, it is essential that the verifier's proof not leak any information to the prover about b (i.e.,

[10] Though it may not be terribly surprising if this assumption turns out to be false, in light of the recent result of Babai [16].

the verifier's proof to the prover should itself be zero-knowledge, or at least satisfy a weaker property called witness-independence (see [130, Section 4.6])). This is because, if G_1 and G_2 are isomorphic (i.e., the prover is lying when it claims that G_1 and G_2 are not isomorphic), a cheating prover could use information leaked from the verifier's proof about bit b in order to guess the value of b with probability more than $1/2$.

Of course, we are omitting many details of how the verifier might prove to the prover in zero-knowledge that it knows a b such that H is isomorphic to G_b. But hopefully this gives some sense of how one might transform honest-verifier zero-knowledge proofs into dishonest-verifier proofs. Clearly, the resulting dishonest-verifier zero-knowledge protocol is more expensive than the honest-verifier zero-knowledge one, because achieving zero-knowledge against dishonest verifiers requires the execution of a second zero-knowledge proof (with the role of prover and verifier reversed).

11.4 Honest-Verifier SZK Protocol for the Collision Problem

In the Collision Problem, the input is a list $(x_1, \ldots, x_N)$ of N numbers from a range of size $R = N$ (while the list length and the range size are equal, it is helpful to distinguish the two quantities, with N referring to the former and R referring to the latter). The goal of the problem is to determine whether every range element appears in the list. Since $R = N$, this holds if and only if every range element appears *exactly* once in the list. However, there is a twist to make the problem easier: it is assumed that *either* every range element appears exactly once in the list (call such inputs YES instances), *or* exactly $R/2$ range elements appear twice in the list (this means, of course, that the other $R/2$ range elements do not appear in the list at all). Call such inputs NO instances. Algorithms for the Collision Problem are allowed to behave arbitrarily on inputs that fail to satisfy the above assumption.

The name Collision Problem refers to the fact that if the input list is interpreted as the evaluation table of a function h mapping domain $\{1, \ldots, N\}$ to range $\{1, \ldots, R\}$, then YES instances have no collisions (i.e., $h(i) \neq h(j)$ unless $i = j$), while NO instances have many collisions

(there are $N/2$ pairs (i, j) such that $h(i) = h(j)$ yet $i \neq j$). This problem was originally introduced as a loose and idealized model of the task of finding collisions in a cryptographic hash function h.[11] With this interpretation as motivation, for each range element $k \in \{1, \ldots, R\}$, we refer to any i with $x_i = k$ as a *pre-image* of k.

In the Collision Problem, since N is thought of as modeling the domain size and range size of a cryptographic hash function, we consider N to be "exponentially large".[12] Accordingly, for this problem, an algorithm should be considered "efficient" (i.e., "polynomial time") only if it runs in time polylog(N).

Fastest Algorithm with no Prover. It is known that the fastest possible algorithm for the Collision problem runs in time $\Theta(\sqrt{N})$ (see Footnote 7 in Section 7.3.2.2), i.e., there is no "efficient" algorithm for the Collision Problem. We briefly sketch how to show this. The best algorithm simply inspects $c \cdot \sqrt{N}$ randomly chosen list elements for a sufficiently large constant $c > 0$, and outputs 1 if they are all distinct, and outputs 0 otherwise. Clearly, when run on a YES instance, the algorithm outputs 1 with probability 1, since for YES instances every list element is distinct. Whereas when run on a NO instance, the birthday paradox implies that for a large enough constant $c > 0$, there will be a "collision" in the sampled list elements with probability at least 1/2.[13] This runtime is optimal up to a constant factor, because it is known that any algorithm that "inspects" $\ll \sqrt{N}$ list elements cannot effectively distinguish YES instances from NO instances (intuitively, this

[11] A key difference between finding collisions in a real-world cryptographic hash function h and the Collision Problem is that in the former task, h will have a succinct implicit description (e.g., via a computer program or circuit that on input i quickly outputs $h(i)$), while in the Collision Problem h does not necessarily have a description that is shorter than the list of all of its evaluations.

[12] Strictly speaking, this is a misnomer, because the size of the input to the Collision Problem is N. But the Collision Problem is modeling a setting where the size of the input (namely, the description of a cryptographic hash function h with domain size and range size N) is really polylog(N).

[13] Let $c = 2$. If there is a collision within the first $\sqrt{N}$ samples, we are done. Otherwise, the probability that none of the first $\sqrt{N}$ sampled range elements appear within the second $\sqrt{N}$ sampled range elements is at most $(1 - \sqrt{N}/N)^{\sqrt{N}} = (1 - 1/\sqrt{N})^{\sqrt{N}} \approx 1/e < 1/2$.

is because any algorithm that inspects fewer than $\Theta(\sqrt{N})$ list elements of a random NO instance will with probability $1 - o(1)$ fail to find a collision, and in this case the algorithm has no way to tell the input apart from a random YES instance).[14]

HVSZK Protocol with Efficient Verifier. Here is an honest-verifier statistical zero-knowledge proof for the Collision Problem. The protocol consists of just one round (one message from verifier to prover and one reply from prover to verifier), and the verifier runs in time just $O(\log N)$ (both messages consist of $\log N$ bits, and to check the proof the verifier inspects only *one* element of the input list).

The first message of the protocol, from verifier to prover, consists of a random range element $k \in \{1, \ldots, R\}$. The prover responds with a pre-image i of k. The verifier simply checks that indeed $x_i = k$, outputting ACCEPT if so and REJECT otherwise.

We now explain that the protocol is complete, sound, and honest-verifier perfect zero-knowledge. Recall that this means there is a simulator running in time polylog(N) that, on any YES instance, produces a distribution over transcripts identical to that of that of the honest verifier interacting with the honest prover.

Completeness is clear because for YES instances, each range element appears once in the input list, and hence regardless of which range element $k \in \{1, \ldots, R\}$ is selected by the verifier, the prover can provide a pre-image of k. Soundness holds because for NO instances, $R/2$ range elements do not appear at all in the input list, and hence with probability $1/2$ over the random choice of $k \in \{1, \ldots, R\}$, it will be impossible for the prover to provide a pre-image of k.

To establish honest-verifier perfect zero-knowledge, for any YES instance $(x_1, \ldots, x_N)$, we have to give an efficient simulator that generates transcripts distributed identically to those generated by the honest verifier interacting with the honest prover. The simulator picks a random domain item $i \in \{1, \ldots, N\}$, and outputs the transcript (x_i, i). Clearly,

[14] The expected number of collisions observed on a random NO instance after inspecting at most T items of the input list is $O(T^2/N)$, so if $T \leq o(\sqrt{N})$ this expectation is $o(1)$. Markov's inequality then implies that with probability $1 - o(1)$, no collision is observed by the algorithm.

the simulator runs in logarithmic time (it simply chooses i, which consists of $\log N$ bits, and inspects one element of the input list, namely x_i). Since in any YES instance, each range element appears exactly once in the input list, picking a random domain item $i \in \{1, \ldots, N\}$ and outputting the transcript (x_i, i) yields the same distribution over transcripts as picking a random range element k and outputting (x_i, i) where i is the unique pre-image of k. Hence, on YES instances, the simulator's output is distributed identically to the view of the honest verifier interacting with the honest prover. Put more intuitively, the honest verifier in this protocol, when run on a YES instance, simply learns a random pair (x_i, i) where i is chosen at random from $\{1, \ldots, N\}$, and this is clearly information the verifier could have efficiently computed on its own, by choosing i at random and inspecting x_i.

Discussion. This protocol is included in this survey because it cleanly elucidates some of the counter-intuitive features of zero-knowledge protocols.

- The simulator, even if run on a NO instance, will always output an accepting transcript (x_i, i). This fact may initially feel like it contradicts soundness of the protocol. However, it does not. This is because, if run on a NO instance, the simulator picks the verifier challenge x_i specifically to be an "answerable" challenge, i.e., a range element that appears in the input list. The actual verifier would have chosen a *random* range element as a challenge, which on a NO instance will, with probability 1/2, have no pre-image and hence not be answerable.

- The existence of an efficient simulator is no barrier to intractability of the problem. While the simulator runs in time $O(\log N)$, the fastest algorithm for the problem requires time $\Theta(\sqrt{N})$.

- While the protocol is honest-verifier zero-knowledge, it is not dishonest-verifier zero-knowledge. Indeed, a dishonest verifier can "use" the honest prover to solve the problem of finding a pre-image of a specific range element of the verifier's choosing (a problem that would require $\Theta(N)$ queries without access to a prover). That

is, on a YES instance, if the dishonest verifier sent to the prover a range element k of its choosing (rather than a uniform random range element as the honest verifier does), then the prover will reply with a pre-image of k. The verifier would not have been able to compute such a pre-image on its own in $o(N)$ time, except with probability $o(1)$.