# INTERACTIVE PROOF SYSTEMS AND ZERO-KNOWLEDGE

BURTON ROSENBERG
UNIVERSITY OF MIAMI

## Contents



## 1. Euclidean Proof System

The ideal of proof appears in Euclid's *The Elements*.

*The Elements* is a collection of 13 books written in Greek by Euclid. It is strongly believed to be written in 300 BC in Alexandria, Egypt.

There are no surviving copies from then, but there are commentaries by Proclus in the 5-th century and Pappus of Alexandria from 290–350 AD that confirm its existence.

The oldest full surviving example is the Codex Vaticanus Graecus 190, from the 9th century, and is the Vatican in Rome.

*Date*: November 12, 2025.

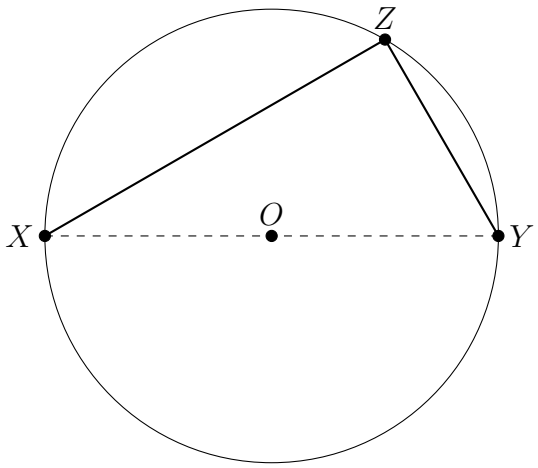


FIGURE 1. Thales' Theorem: For $Z$ on the circle, $Z$ is a right angle.

The notion of proof is much older. In 600 BC Thales of Miletus proved the theorem appearing in *The Elements*, Book III, Proposition 31,

> *For a triangle with two points lying diametrical on a circle, the angle at the third point when lying on the circle is a right angle.*

See Figure 1 for an illustration of this theorem.

*The Elements* also included an algorithm for determining the GCD of two lengths. This is illustrated in Figure 2 for $gcd(5, 3) = 1$.

(1) The algorithm starts with segments of size 5 and 3.
(2) Subtracting lengths of 3 from 5, this is done one time with a the remainder segment of length 2.
(3) Then subtracting lengths from 2 from 3, this is done one time with a remainder segment of length 1.
(4) Then subtracting lengths of 1 from 2, this is done two times with no remainder. Because there is no remainder, the algorithm halts.
(5) The answer is the length of the last segment.

In the commentary by Proclus it was noted that if the GCD algorithm was performed on two line segments of which the ratio of the lengths is not rational, the algorithm would not terminate.

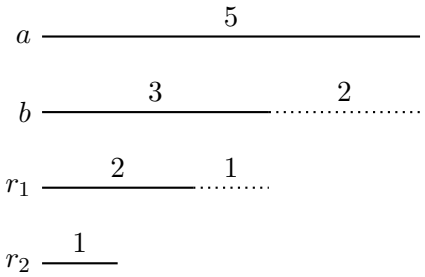


FIGURE 2. Geometric illustration of Euclid's GCD algorithm: each line shows the division into integer parts and a remainder.

## 2. Interactive Proof Systems

Both the proof of Thales Theorem and the GCD algorithm are proofs in the Euclidean proof system. They are a finite number of steps, according to a system of derivation, leading to the conclusion. In response to the difficulties of certain proofs, such as the proof that two graphs were non-isomorphic, Goldwasser, Micali and Rackoff introduced a different sort of proof system[1] that depended on the discussion between two computational entities, the *prover* and the *verifier*. It is an extension of Euclid's system in two ways,

(1) Rather than a "proof" which is verified, which is the same as a deterministic algorithm which is carried out, the verifier can ask the prover questions.
(2) Euclid's proof systems were perfectly sound. It is not possible to prove a false theorem. In an interactive proof there is an arbitrarily small probability that the system will conclude falsely that an untrue statement is true.

The Euclidean proof system is not randomized, so it must have perfect soundness — it has exactly one conclusion, true or false. The Goldwasser-Micali system is randomized, and it uses this randomization to sometimes fail. But this also opens up the prospect for faster proofs, or easier proofs, and most significantly, for *zero-knowledge proofs*.

In the IP proof system there are two interacting algorithms, the verifier $V$ and the prover $P$, and only the verifier is required to be a PPT algorithm, We place no restriction on $P$. Given a statement $x$ the machines interact to determine the truth of $x$. We will write this computation as $[P, V](x)$.

[1] *The Knowledge Complexity of Interactive Proof-Systems*, 1985 STOC.

They interact by taking turns a writing on a pair of correspondence tapes. In each round of communication $V$ writes a query on the query tape, which $P$ reads. Then $P$ writes its response o the answer tape, which $V$ reads.

Consider a language $\mathcal{L} \subseteq \Sigma^*$, from some universal set $\Sigma^*$. An *interactive proof system* gives a decision in polynomial time with these rules,

- *Completeness:*
$$\forall x \in \mathcal{L},\ Prob\big([P,V](x) = T\big) \geq 2/3.$$
- *Soundness:*
$$\forall x \notin \mathcal{L}, \tilde{P},\ Prob\big([\tilde{P},V](x) = T\big) < 1/3.$$

The prover can be all powerful, but a proof is not an all powerful machine claiming something is true. The all powerful machine can explain why something is true, but the verifier must accept that explanation, and must accept it reasonably efficiently.

Hence in the soundness requirement, no prover can make a verifier accept a false claim with greater than $1/3$ probability. This is independent of $x \notin \mathcal{L}$. So re-running the interaction a large number of times will discover whether the statement is true or false negligible probably of error.

**Definition 2.1.** Define IP as the class of all languages that have IP proof systems.

**Theorem 2.1.** $IP \supseteq NP$.

**Proof:** NP is the class of languages that have a polynomial time verifiers. For NP language $\mathcal{L}$ and its verifier $V(x,y)$,

$$\forall x \in \mathcal{L}, \exists y,\ V(x,y) = T$$
$$\forall x \notin \mathcal{L}, \forall y,\ V(x,y) = F$$

The IP proof system is built from this verifier which takes $y$ from the unsolicited response of the prover. For $x \in \mathcal{L}$ there is a $y$ such that $V(x,y) = T$. The correct prover $P$ provides that $y$ on the answer tape. For $x \notin \mathcal{P}$, there is no $y$ such that $V(x,y)$. So the arbitrary prover $\tilde{P}$ provides an arbitrary $y$. □

Note that in this case the proof system is deterministic, and has *perfect completeness* and and *perfect soundness*. In general, the benefit of perfect completeness is not significant for the language class IP. However, if an IP language has perfect soundness then the language is certainly in NP.

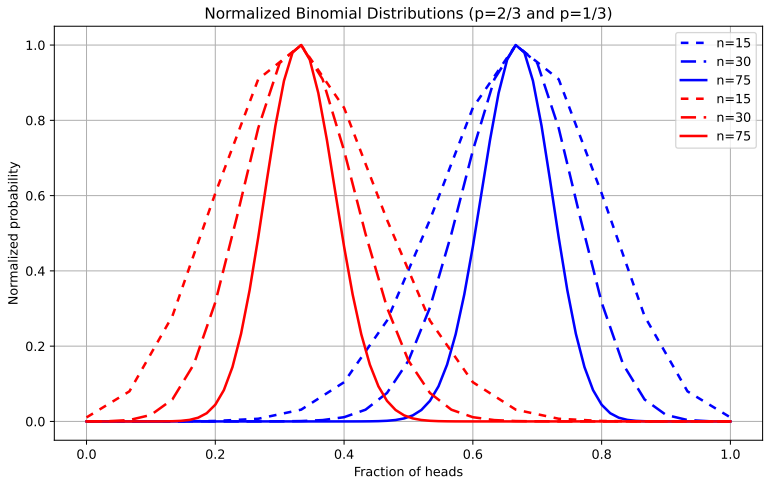


FIGURE 3. Probability amplification for symmetric probabilities.

**Theorem 2.2.** For a language $\mathcal{L} \in \text{IP}$, if the soundness is perfect then $\mathcal{L} \in \text{NP}$.

**Proof:** Let $[P, V](x)$ be the prover verifier system. Let $V^*(x, \tau)$ be the polynomial time algorithm that verifies the conversation $\tau$ that was had between $P$ and $V$. That is, it is programmed exactly like $V$, share the random coins (if any) of $V$, accepts as the witness the concatenation of all replies from $P$. Note that $|\tau|$ is of polynomial length.

This is a verifier for the language. Since if $x \notin \mathcal{L}$, for any conversation $\tau$ will have $V^*(x, \tau) = F$. And if $x \in \mathcal{L}$, there is a conversation $\tau$ such that $V^*(x, \tau) = T$. □

## 3. Probability Amplification

The answer that the IP protocol gives is a coin of one of two biases. In our case, if the answer is yes, it gives us a coin with bias 2/3. If no, it gives us a coin with bias 1/3. Our decision is to decide yes or no by experimenting with the coin to estimate its bias. We flip the coin a number of times, compute the fraction of heads, and use this as a sample of the bias.

Figure 3 shows how increasing trials (15, 30 and 75 trials) sharpens the sample around the true bias. We've take a special case where the two biases are symmetric

around 1/2. Therefore our decision criteria is whether the majority of flips was heads or tails.

If instead of 2/3–1/3 we chose any other pair of probabilities, symmetric around 1/2, with sufficient trials we would be able to discriminate between the two coin biases. Using *Chernoff Bounds*[2], which give approximations for the probability of the sample being on the wrong side of the majority, the chance of this error is shown to decrease exponentially in the number of trials.

As long as we choose two fixed probabilities symmetric around 1/2, this graph tells the story, With sufficient trials the majority rule will separate the coins with negligible error. So the 2/3–1/3 is just my favorite numbers, but other numbers are found in the definition of IP. In general, the idea is you are given the result not as a single answer, but as a coin you will experiment with to determine its bias.

The class IP is not changed if we require perfect completeness. That is, if $x \in \mathcal{L}$ then $V(x) = T$, always. In this case we are given one of two coins: one which always gives heads, and other which mostly gives tails, but can give heads. Our situation is now, we flip the coin waiting for a tails. The moment we see a tails, we know that $x \notin \mathcal{L}$. Else, for $k$ flips were are $q^k$ certain that $x \in \mathcal{L}$. This is the error that we were given the biased coin and misidentify it as a coin that only gives heads.

## 4. When proofs are not perfectly sound

The proper reading of,

$$\forall x \notin \mathcal{L}, \forall y,\ V(x, y) = F$$

is not we have proven $x \notin \mathcal{L}$, rather it means we have failed to prove $x \in \mathcal{L}$. This distinction is clarified when one considers the possibility of the complement of $\mathcal{L}$ being in NP.

[2]**Chernoff Bound:** Let $X_1, X_2, \ldots, X_n$ be independent Bernoulli random variables with $\Pr[X_i = 1] = p_i$ and let

$$S_n = \sum_{i=1}^{n} X_i, \quad \mu = \mathbb{E}[S_n] = \sum_{i=1}^{n} p_i.$$

For any $0 < \delta < 1$,

$$\Pr[S_n < (1 - \delta)\mu] \leq \exp\Big( - \frac{\delta^2 \mu}{2} \Big).$$

Suppose in addition to the $V$ that proves when $x \in \mathcal{L}$ we have an $V'$ that proves when $x \notin \mathcal{L}$. That is, there exists a $V'(x, y)$ such that $\forall x \notin \mathcal{L}, \exists\, y',\ V'(x, y') = T$.

The pair $(V, V')$ has the complicated definition,

$$\begin{aligned}
&\forall x, \exists y,\ V(x, y) \neq V'(x, y) \\
&\forall x \left( (\exists\, y\, V(x, y) = T) \implies (\forall y'\, V'(x, y') = F) \right) \\
&\forall x \left( (\exists\, y'\, V'(x, y') = T) \implies (\forall y\, V'(x, y) = F) \right)
\end{aligned}$$

Such languages are in the intersection of NP and co-NP.

We will look at a pair of problems where one is in NP and the other not.

We will show that IP can determine membership for languages not in NP. This makes IP a more powerful proof system. For languages not in NP, the soundness cannot be perfect. The proof system is more powerful because it is a randomized proof system allowing for a small soundness error.

**Definition 4.1.** Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a vertex bijection $\phi : V_1 \cong V_2$ such that the induced map on edges

$$\phi(e) = \phi\big(\{v, v'\}\big) = \{\phi(v), \phi(v')\}$$

is also a bijection.

**Definition 4.2.** The language of graph non-isomorphism is the collect of non-isomorphic graph pairs $(G_1, G_2)$.

Note that graph isomorphism is in NP. The prover sense the vertex bijection, and the verifier verifies that it is a vertex bijection and an edge bijection (in polynomial time).

However it is not known if graph non-isomorphism is in NP.

**Theorem 4.1.** Graph non-isomorphism is in IP.

**Proof:** Chose a prover $P$ be capable of solving graph isomorphism. That is, given isomorphic graphs $G_1$ and $G_2$, $P$ can calculate the vertex bijection $\phi$ demonstrating the isomorphism.

In a single round, the verifier $V$ presents $P$ with the graph,

$$\phi^*(G_b)$$

where $b$ is randomly 1 or 2 with equal probability and $\phi^*$ is a random permutation chosen equally likely from the space of all permutations.

A correct $P$ solves for $b$ and responds with that value. The arbitrary $\tilde{P}$ returns whatever $\tilde{b}$ it chooses.

The verifier accepts if $b = \tilde{b}$ and rejects otherwise.

In the case where the input graphs are not-isomorphic, $P$ will determine with $b$ was chosen by $V$, and have perfect completeness. If the graphs are isomorphic, the correct $P$ will find isomorphisms between the presented graph and both input graphs and will have to return a $\tilde{b}$ which is $b$ only half the time. This is true of the incorrect $\tilde{P}$.

In this argument it is important to note that the random variable $\phi^*(G_1)$ is truly indistinguishable from the random variable $\phi^*(G_2)$ when $G_1 \cong G_2$. This depends on that there is a $\varphi$ such that $G_1 = \varphi(G_2)$ and the class of bijections on the graph is a group,

$$\phi^*(G_1) = \phi^*(\varphi(G_2) = (\phi^* \circ \varphi)(G_2),$$

So as a random variable $\phi^*(G_2)$ is identically distributed as $\phi^*(G_1)$. □

## 5. On the secrecy of the verifier's coins

One might have the impression that the secret choice of the verifier is inevitable in a proof that graph non-isomorphism is in IP. A private coin is the crucial idea in this proof.

However, Babai had already looked at a public coin model in his work on Arthur-Merlin games. In an AM game the verifier flips coins for its working, but presents those coin outcomes to the prover. The prover sees the coins but cannot control them.

Goldwasser and Sipser brought these two models together, showing that any private-coin interactive proof can be transformed into the AM public-coin model with only polynomial overhead. Babai and Moran (1988) later refined this perspective, giving the modern understanding that the public-coin class AM captures the full expressive power of interactive proofs.

## 6. Zero-Knowledge

The introduction of interactive proof systems allowed solutions and characterization of problems outside of NP. It was natural to look at this model when motivated by applications in encryption, as were Goldwasser and Micali. However, they were also after another goal, the convincing of a verifier by a prover with minimal disclosure to the verifier. In the case of their graph non-isomorphism solution, this is achieved. The verifier is convinced the graphs are non-isomorphic, but what proof of this did the verifier really have?

What the verifier had were answers to questions to which it already knew the answers.

There is yet another nuance to this, and that is concerning the length of a proof. There is indeed a proof of graph non-isomorphism. You can list all possible bijections and check that none of them is an isomorphism. The fact is, this is possible. The set of bijections is finite, but exponentially large. The question isn't of proof, but of an efficient proof. The relaxation of soundness isn't about truth, but about short-cuts to the truth.

These ideas play out more fully in Probabilistically Checkable Proofs.

**Theorem 6.1** (PCP Theorem (informal statement))**.** Every decision problem in the class NP has a probabilistically checkable proof (PCP) that can be verified by reading only a constant number of bits of the proof, using a logarithmic number of random bits.

This is a radical departure from the Euclidean Proof Systems. These short proofs are now being used in Etherium for off-chain verifications. A short-proof roll-up of longer off-chain proofs is periodically put on-chain.

**Definition 6.1.** For an IP protocol, a *view* is the transcript of messages that pass between the prover and the verifier. It is denoted $\text{View}[P, V](x)$. The run of the protocol on $x$ is a sampling of a random variable with value in the set of all possible views, with a probability distribution induced by the underlying randomness of the algorithm.

A protocol is zero-knowledge if, when promised that the input is in the language, the view can be efficiently sampled without referring to the prover.

**Definition 6.2** (Zero-Knowledge)**.** For an IP protocol $[P, V](x)$ let,

$$[P, V](x) \in_R \{\, \text{View}[P, V](x) \,\}$$

be the sampling from all possible views of a random run of the protocol. Let $\perp$ be a distinct element representing the failure to sample from this distribution. A protocol is *zero knowledge* if for all $x$ in the language and all verifiers $\tilde{V}$ there is a simulator ,

$$S(x) \in_R \{ \text{View}[P, V](x) \}) \cup \{ \perp \}$$

such that,

(1) The simulator does not fail often,

$$Prob(S(x) = \perp) \leq 1/2$$

(2) Restricted to the case when the simulator does not fail, it generates the view's probability distribution,

$$\{ \text{View}[P, \tilde{V}](x) \} = \{ S(x) \,|\, S(x) \neq \perp \}$$

## 7. Graph Isomorphism

We have a proof system for the language

$$ISO = \{ (G_1, G_2) \,|\, G_1 \cong G_2 \}.$$

We now describe a zero-knowledge proof system for ISO.

**Theorem 7.1.** ISO can be proven in IP.

**Proof:** The input is $(G_1, G_2)$, two graphs each with $n$ vertices and the same number of edges. (Else the verifier determines on its own they are not isomorphic and halts).

Otherwise the Prover determines if the graphs are isomorphic, and if so, determines the isomorphism $\psi(G_1) = G_2$.

Prover picks a random permutation of the vertices $\phi \in \mathcal{S}_n$, and sends the graph $H = \phi(G_1)$ to the verifier.

The verifier sends to the prover $b \in \{ 1, 2 \}$ based on the fair coin flip.

If the graphs are not isomorphic, the prover sends $\phi$, for lack of any better alternative. If the graphs are isomorphic the prover sends,

$$\phi^* = \begin{cases} \phi & b = 1, \\ \phi \circ \psi & b = 2. \end{cases}$$

The verifier then attempts to verify that $H = \phi^*(G_b)$. The verifier halts with true (claiming that the graphs are isomorphic) if this equality holds.

The algorithm is complete. If in fact $G_1 \cong G_2$, then the Prover does derive $\psi$ and $H = \phi^*(G_b)$, with probability one.

The algorithm is sound, with soundness bound 1/2. Let $\tilde{P}$ be any prover. Because the graphs $G_1$ and $G_2$ are not isomorphic, no matter how $H$ is created, it can be isomorphic to at most one of these. Without foreknowledge of the Verifier's choice the probability that the verifier asks for the $G_b$ for which there is an answer is most 1/2. □

**Theorem 7.2.** The protocol just described is zero-knowledge..

**Proof:** Let $G_1 = \psi(G_2)$. For any verifier $\tilde{V}$ the view is a triple drawn randomly from the set,

$$\mathcal{D} = \{ \langle H, b, \phi^* \rangle \}$$

We know that $H$ is calculated as $H = \phi_1(G_1)$, where $\phi_1$ is drawn randomly and uniformly from $\mathcal{S}_n$, $b$ is chosen by the Verifier after having seen $H$, and

$$\phi^* = \begin{cases} \phi_1 & b = 1, \\ \phi_1 \circ \psi & b = 2. \end{cases}$$

Since $\psi : \mathcal{S} \to \mathcal{S}$ is a bijection of sets, the distribution of $\phi_1$ is the same as $\phi_1 \circ \psi$. Therefore the distribution of $\phi^*$ is a uniform random draw from $\mathcal{S}_n$.

We create simulator $S$ to generate this distribution. The simulator will need to know the verifier's choice of $b$. We include $\tilde{V}$ as an oracle to $S$, forming $S^{\tilde{V}}$.

The simulation choses $\phi^*$ and $\hat{b}$ uniformly at random and computes. $H = \phi(G_{\hat{b}})$. It then queries $\tilde{V}$ on $H$ and receives its choice of $b$. If $b = \hat{b}$ then the simulator outputs $\langle H, b, \phi^* \rangle$. Else the simulator outputs $\perp$.

The simulation will return $\perp$ with probability 1/2, since the coin flip $\hat{b}$ does not bias the distribution of $H$. Hence the verifier learns nothing about the value of $\hat{b}$ from what it receives. The coincidence of $b$ and $\hat{b}$ is therefore only dependent on the fair coin flip.

Conditioned on $S(x) \neq \perp$, the resulting distribution is the result of gleaning from the triples $\langle H, b, \phi^* \rangle$ those that appear coincident to the success an independent coin flip. This will coincide with the distribution of views from the actual protocol. □

The idea of the simulation argument is to say that whatever a P-time bounded algorithm could compute with with the use of the IP protocol, it could compute without it. The verifier, and all possible outputs of the verifier, are replaced with a P-time sampleable distribution.

7.1. **Further discussion.** What happens if the simulator is given two non-isomorphic graphs, $G_1 \ncong G_2$? Definitions become a bit confused, as we have both a cheating prover and a dishonest verifier.

The simulator will happily select a triple including a random permutation, a bit and a graph isomorphic to $G_b$. In the imperfect soundness of the IP protocol, this is the improbable case of the cheating prover always winning.

The simulator gives one distribution in the case of isomorphic graphs, and another in the case of non-isomorphic graphs. These distributions are distinguished by necessities such as,

$$\begin{aligned} \text{ISO} &: \exists \phi_1, Pr(\langle\, H, 1, \phi_1 \,\rangle) > 0 \implies \exists \phi_2, Pr(\langle\, H, 2, \phi_2 \,\rangle) > 0 \\ \text{non-ISO} &: \exists \phi_1, Pr(\langle\, H, 1, \phi_1 \,\rangle) > 0 \implies \forall \phi_2, Pr(\langle\, H, 2, \phi_2 \,\rangle) = 0 \end{aligned}$$

If evidence of this sort exists to refute non-ISO, then $G_1 = \phi_1^{-1} \circ \phi_2(G_2)$. However finding such evidence through luck is exponentially rare.

## 8. ZK Proofs of Knowledge

We will study an application of Zero-Knowledge to the case of proof of identity. An older protocol for proof of identity is the password. The weakness of this scheme is that it necessarily involves shared knowledge between two entities, so it gives little guarantee about possible outcomes. There is the story of *Alibaba and the Forty Thieves* where a password scheme is used, and the weakness of the scheme leads to both fortune and misfortune.

Since Zero Knowledge can prove the truth of a fact without revealing any knowledge other than that the statement "this fact is true", we might look to Zero Knowledge for better identification schemes. One small difference, is what needs to be proved in zero knowledge is not a fact, but knowledge itself.

What we need is a *zero knowledge proof of knowledge*. And this is what we pursue in now.

## 9. Fiat–Shamir identification

Let $\mathbb{Z}_n^\times$ be those integers mod $n$ that have a multiplicative inverse in the system of integers mod $n$. These will be all integers 1 through $n-1$ that are relatively prime to $n$. For $n = 15$ we have,

$$\mathbb{Z}_{15}^\times = \{\, 1, 2, 4, 7, 8, 11, 13, 14 \,\}$$

For instance,

$$7 \cdot 13 = 91 = 6 \cdot 15 + 1 = 1 \pmod{15}.$$

The map $x \mapsto x^2$ has image the *quadratic residues*. A quadratic residue a number that has a square root in $\mathbb{Z}_n^\times$, and in general will have multiple square roots. For instance,

$$\sqrt{4} \bmod 15 = \{\, 2, 7, 8, 13 \,\}$$

The computational complexity of finding square roots depends on $n$ and the knowledge about $n$.

For primes, computing square roots in polynomial. For the case of $n = pq$ for $p$ and $q$ distinct odd primes, computing square roots is computationally equivalent to factoring $n$.

This provides us with a hard problem with an easy inverse usable for a zero knowledge proof of knowledge, for use in an identification scheme.

**Idea for an identification scheme:** Fix an $n = pq$, the produce of two distinct primes. Each entity will chose a secret $x \in \mathbb{Z}_n^\times$, and publish $x^2 \bmod n$. Since taking square roots is hard, publishing $x^2$ does not reveal $x$. An entity is then identified by their knowledge of the corresponding $x$, and this knowledge will be proved in zero knowledge, so that noting about $x$ is revealed by the protocol.

**Idea for a zero knowledge proof of a square root:** While finding a square root in $\mathbb{Z}_n^\times$ is hard for specific quadratic residue, finding a square root of a random quadratic residue is easy. When the goal is the square root of any quadratic residue, pick a random $r \in \mathbb{Z}_n^\times$ and square it. This gives a random $R = r^2$, but it can also be seen as a random $RS = r^2$ for a given $S$, since $RS$ is a random as is $R$.

The essence of the protocol is this ensemble of triples,

$$\{\, \langle\, R, b, t \,\rangle \,\} \text{ where } t^2 \bmod n = \begin{cases} R & b = 0 \\ RS & b = 1 \end{cases}$$

The prover-verifier protocol creates this triples in this order,

(1) The prover picks a random $r \in \mathbb{Z}_n^\times$ and send the verifier $R = r^2 \bmod n$.
(2) The verifier picks a random challenge $b \in \{\, 0, 1 \,\}$.
(3) The prover responds with

$$t = \begin{cases} r & b = 0 \\ rs & b = 1 \end{cases}$$

(4) The verifier checks,

$$t^2 = \begin{cases} R & b = 0 \\ RS & b = 1 \end{cases}$$

and accepts if the equation checks out.

**Completeness:** If the prover knows $s$, the protocol completes with the verifier accepting with probability 1.

**Soundness:** If the prover can answer both $b = 0$ and $b = 1$ that means the prover knows both $r$ and $sr$. Then by simple division the prover knows $s$. Therefore if the prover does not know $s$ it cannot answer both correctly. Therefore the protocol completes with the verifier accepting with probability at most $1/2$.

Note a further consequence, that if the verifier accepts, the verifier accepts both that $R$ is quadratic residue and that the prover knows on if its square roots.

**Zero-Knowledge:** The simulator selects a random $\hat{b} \in \{\, 0, 1 \,\}$ and a random $r \in \mathbb{Z}_n^\times$. It will attempt to complete the protocol with the triple $T$,

$$T = \begin{cases} \langle\, t^2, 0, t \,\rangle & \hat{b} = 0 \\ \langle\, t^2/S, 1, t \,\rangle & \hat{b} = 1 \end{cases}$$

The simulator then run the verifier $\tilde{V}$ up to its decision on $b$. If $b \neq \hat{b}$ the simulator returns $\perp$. Else the simulator returns $T$.

For a quadratic residue $R$ the map $R \mapsto R/S$ is a bijection, The coin $\hat{b}$ is an independent variable to the verifier's decisions, and

$$Prob(S = \perp) = Prob(b \neq \hat{b}) = 1/2.$$

The distribution of the simulator, conditioned on $S \neq \perp$ should be exactly that of correct prover-verifier runs when the input $S$ is a quadratic residue of which the prover knows a root.

The simulator out will pass verification. For $T = \langle R, b, t \rangle$ if $b = 0$ then $R = t^2$, which is correct for this case. If $b = 1$ then $R = t^2/S$ so $RS = t^2$, which is correct for this case.

By the independence of $b$ and $\hat{b}$,

$$\begin{aligned} Prob(\langle T, \hat{b}, t \rangle | b \neq \perp) &= Prob(\langle T, \hat{b}, t \rangle \,|\, (b = \hat{b})) \\ &= Prob(\langle T, b, t \rangle \ \wedge \ (b = \hat{b}))/Prob(b = \hat{b}) \\ &= Prob(\langle T, b, t \rangle) Prob(b = \hat{b})/Prob(b = \hat{b}) \\ &= Prob(\langle T, b, t \rangle) \end{aligned}$$

So the simulator alone can provide the same distribution as the authentic protocol. □

## 10. Appendix: Quadratic Residues

The difficulty of square roots mod $n = p\,q$ is similar to the RSA problem. To take square roots $Z_n$ will lead to factoring $n$. If the factors of $n$ are available, here is how we take the square root $Z_p$ for a prime $p$. Using the Chinese remainder theorem, a square root mod $n$ can be calculated by combining the square roots mod $p$ and $q$. In doing so, the number of square roots becomes 4.

**Theorem 10.1.** In $Z_n^\times$, with $n = p\,q$, the product of distinct, odd primes, there are four roots of unity.

**Proof:** Let $x^2 = 1 \pmod{n}$. By Bezout's, and the relative primality of $p$ and $q$, for a square root of $x$ there are integers $s$ and $t$ such that.

$$s\,p + t\,q = x$$

Then the four numbers,

$$(\pm\, s)p + (\pm\, t)q$$

are all roots of unity mod $n$. All four are distinct mod $n$, and

$$((\pm\, s)p + (\pm\, t)q)^2 = (sq)^2 + (tq)^2 = (s\,p + t\,q)^2 = x^2 = 1 \pmod{n}$$

□

**Corollary 10.1.** With $n$ as above, every quadratic residue has four roots.

**Theorem 10.2.** Let $p$ be prime, 3 mod 4. And $a$ a quadratic residue $Z_p$. Then the square roots of $a$ are $\pm a^{(p+1)/4} \bmod p$.

**Proof:** Let $p = 4k + 3$, and $x^2 = a$. Then,

$$a^{(p+1)/4} = x^{(p+1)/2} = x^{(p-1)/2+1} = \pm\, x.$$

□

For the case $p$ a prime, 1 mod 4, a more involved algorithm is require for taking the square root. One is called Tonelli–Shanks.