Zero Knowledge What? An Introduction to Zero Knowledge

An Introduction to Zero Knowledge Proofs

One of my instructors shared that he would have a funny daily conversation with his dad:

- Dad: “How was school today?”
- Middle Schooler: “Fine.”

Understandably, a father might be frustrated with the same response each day. Why? One could argue that this father might not be learning much about his son’s day. Indeed, this idea captures a fundamentally important idea in complexity theory and cryptography: zero-knowledge proofs.

Proof Systems

First, let’s define how proofs work in computer science. We will define two parties: a *prover* P P and a *verifier* V V. P P and V V can communicate and send messages to each other. V V can output “accept” if V V is convinced by P P and “reject” otherwise. V V can only run in polynomial in the length of the statement in question. P P and V V can also use randomness. Here are some example statements a prover may try to prove to the verifier:

- Is a boolean formula satisfiable?
- Can a graph’s nodes be assigned three different colors such that no two adjacent nodes are the same color?
- Given a set of integers, is there a subset whose sum is zero?
- Given a group GG with a generator $g \in G$g∈G, what is the discrete logarithm of g^xgx for some $x \in G$ x∈G?

Note

The first three examples are all NP-Complete problems, so it is widely believed that V V will not be able to determine the answer in polynomial time themselves. This is why P P is needed in the first place.

Note

This fourth problem is a different form than the previous three. In fact, it is a *search problem* (“what is the answer”) as opposed to a decision problem (“is this true or false”). This problem doesn’t have as strong computational hardness as the previous three but is also assumed to be hard for classical (non-quantum) computers. If you have a way to solve this problem efficiently for elliptic curve groups with classical computers, contact your nearest cryptographer immediately.

Note

For budding complexity theorists, this definition of a proof system loosely encapulates the complexity class IP.

Definining Zero Knowledge

In what sense can a proof system be zero knowledge? At first glance, proving information without giving away information sounds like a paradox. Let’s say in English that a zero knowledge proof should prove that a statement is true **without giving away information beyond the fact that the statement is true**.

Let’s ground this idea with some examples:

- Prove that a graph is 3-colorable without giving away any information about how to color it.
- Prove that a boolean formula has a satisfying assignment without giving away anything about the assignment.
- Prove that some cryptocurrency was given to you without giving away anything about who previously owned that token.
- Prove that you know the discrete logarithm of g^xgx without giving away anything about xx.

Note

Similarly to our decision versus search problem discussion, the first three proofs are proofs of the truth of a statement. The last proof isn’t about whether a statement is true but instead is defined as a “zero-knowledge proof of knowledge.”

A Cool Application

Let's see which of your friends know you super well: Who knows your birthday?

Now, imagine that you were with a group of friends who can all listen in. Some of your friends claim that they know the answer. You would like to allow these arithmetic pros to prove to you that they know the answer without giving away the answer to your other friends who are listening. Even more, you want to allow your friends that *know* your birthday to convince your friends that *don't* know your birthday that they know your birthday without giving away your birthday.

The crazy thing: this can be done!

Schnorr's Sigma Protocol

First, let's encode your birthday as an integer: MMDDYYYY. For example, if you birthday is Jan 10, 1938, then your encoding would be 01101938. Choose some super large cyclic group $Z_p = \{0, 1, \ldots, p-1\}$ Zp={0,1,…,p−1} where $p = 2q + 1$ p=2q+1 for some *huge* primes qq and pp (let's say around 2^{2048}22048). Since our P P and V V are humans, let $p = 2000000579$ p=2000000579.

Note

If you want to know how I picked pp, check out this page. This careful choice of pp is to make the discrete logarithm for this group difficult.

Choose any $g \neq 0, 1 \in Z_p$ g≠0,1∈Zp as our generator for Z_p Zp. Next, let xx the encoding of our birthday. Publish $h = g^x$ h=gx to your friends. We assume it is hard for them to be able to find xx given g^x gx beyond brute forcing the solution. If your appearance doesn't give away your age, let's assume that there are about $36,500$36,500 possibilities and that a human thus would really struggle to brute force guess the answer.

Note

In this toy example, brute forcing the answer (guessing a a's and cheking if $g^a = g^x$ ga=gx) via a computer is easy. Most computer science problems have a much larger output space such that brute forcing a solution would be infeasible. As such, our goal will be to make brute forcing be a forgetful friend's best strategy to figure out the answer.

In this case, your friend is the prover, and you are the verifier. In fact, even your friends who **don't** know the answer could be the verifier themselves! Here is the protocol:

$P(x, h = g^x)$ P(x,h=gx)		$V(h = g^x)$ V(h=gx)
$r \rightarrow^R Z_p$ r→RZp		
$u \leftarrow g^r$ u←gr	$\rightarrow u$ →u	
	$c \leftarrow$ c←	$c \rightarrow^R Z_p$ c→RZp
$z \leftarrow r + cx$ z←r+cx	$\rightarrow z$ →z	
		"Accept" if $g^z == u * h^c$ gz==u∗hc

How to Try It Using the Python Interpreter

Note: This tutorial is an instructive toy example. This code should not be used for any real application. For example, this code does not protect against side channel attacks and stores private values in terminal memory.

Pick some friends and try this out! You can easily do all this arithmetic using the Python Interpreter, which we will demonstrate below.

For the person who wants to poll who knows their birthday, they should compute their public value as follows (without anyone looking at their screen):

```
>>> g = 5 # public parameter
>>> p = 2000000579 # public parameter
>>> x = 1101938 # Don't let anyone peek! Jan 10, 1938
>>> h = pow(g,x,p) # more efficient than (g**x) % p
1880666247
```

For the friend P P that wants to prove that they know your birthday, have them begin with this code (do not let anyone take a peek at your screen!).

```
>>> from random import SystemRandom
>>> gen = SystemRandom()
>>> g = 5 # public parameter
>>> p = 2000000579 # public parameter
>>> r = gen.randrange(p)
>>> u = pow(g,r,p)
>>> print(u) # send to V
1706406692
```

Send uu to the verifier friend VV. VV should then run this code on their computer.

```
>>> from random import SystemRandom
>>> gen = SystemRandom()
>>> g = 5 # public parameter
>>> p = 2000000579 # public parameter
>>> u = 1706406692 # From prover, let's keep this for later
>>> c = gen.randrange(p)
>>> print(c) # send to P
107041050
```

Next, P should receive cc and run this code:

```
>>> c = 107041050 # from V
>>> z = r+c*x
>>> print(z) # Send to V
1181831844243911
```

Finally, V should perform this final check:

```
>>> h = 1880666247 # from person who's birthday I should know but don't.
>>> z = 1181831844243911 # from prover
>>> print("They know that person's birthday" if pow(g,z,p) == (u * pow(h,c,p)) % p else "They're lying! They don't know their birthday")
They know that person's birthday
```

Zero Knowledge Formalisms

This is a more formal treatment of a zero knowledge proof. Feel free to skip this section. Let (P, V)(P,V) be an interactive proof system for a language LL (a "language" is a complexity theory formalism. Essentially, treat $x \in L$ x∈L as "x is true"). This proof system is *zero-knowledge* if it satisfies the following properties $\forall x$∀x:

h

- ***Completeness.*** $x \in L \implies Pr_{P,V\text{'s randomness}}[V \text{ accepts running}(P, V)] = 1$ x∈L⟹PrP,V's randomness [V accepts running(P,V)]=1

  In English, this means that an honest prover and verifier should cause the verifier to accept if x is true.

- ***Soundness.*** $\forall P^*(x \notin L \implies Pr_{P,V\text{'s randomness}}[V \text{ accepts } (P^*, V)]) < \frac{1}{3}$ ∀P∗(x∉L⟹PrP,V's randomness [V accepts (P∗,V)])<13.

  In English, this means that malicious provers should only be able to trick an honest verifier that a statement is true with low probability.

- ***Perfect Zero Knowledge.***
  $\forall V^* \exists$ efficient S such that $\forall x \in L : \{Sim(y)\} \approx \{View_{V^*}((P, V^*)(x))\}$ ∀V∗∃ efficient S such that ∀x∈L:{Sim(y)}≈{ViewV∗((P,V∗)(x))}

  In English, this means an efficient algorithm should be able to generate the transcript of the protocol without even knowing whether x x is true or not. If the transcript can be generated without the knowledge of the answer, then the transcript must give away zero knowledge! In order to preserve completeness and soundness, the resulting proof system **must** be randomized.

There are many variants of this definition of zero knowledge:

- ***Honest-Verifier ZK***

  only requires the simulator to produce identical distributions for the honest verifier V V for the protocol. In other words, the verifier is expected to behave honestly, but malicious verifiers may be able to learn information by deviating from the protocol.

- ***Statistical ZK***

  relaxes the requirement that the transcript/view distributions are identical and instead makes the similarity function allow for some negligible difference (as a function of some security parameter) in probability for any given state in the distribution.

- ***Computational ZK***

  requires that all polynomially bounded algorithms should be unable to distinguish the transcript/view distributions (relying on some security assumptions).

An Aside About Completeness and Soundness for Schnorr's Protocol

Completeness and soundness properties are what define an interactive proof system, so I will briefly explain the analysis for Schnorr's protocol. Note that if P P is honest, then $g^z = g^{r+cx} = g^r g^{cx} = u(g^x)^c = u * h^c$ gz=gr+cx=grgcx=u(gx)c=u∗hc, satisfying completness. This part is not vital fo the idea of zero-knowledge, so you can skip this section.

Soundness is somewhat trickier. In fact, our ZK formalisms don't quite apply here, as those definitions were for decision problems, not search problems. Instead, we could say that soundness means "a poly-time adversary without x x should only be able to produce z z where $g^z = u * h^c$ gz=u∗hc with negligible probability." We could prove this definition of soundness by using an adversary that would break soundness for Schnorr's Protocol to create an adversary that would break some widely held security assumption, such as the Decisional Diffie Hellman assumption or discrete log assumption. This proof format is a security reduction and is widely used in cryptgraphy.

Similarly, one could define a variant of soundness for Proof of Knowledge systems. Namely, the scheme should satisfy a proof of knowledge requirement. Formally, this means that an efficient algorithm E E called the "extractor" can, with black box access to P P, determine the hidden value. In our setting:

$\forall x, h, P^* Pr[g^x = h : x \leftarrow E^{P^*}(h)] \geq Pr[(P, V)(h) = 1] - \epsilon$ ∀x,h,P∗Pr[gx=h:x←EP∗(h)]≥Pr[(P,V)(h)=1]−ϵ where ϵ ϵ is considered the *knowledge error*.

Here would be our extractor algorithm:

Run P^* P∗ to determmine u u.
Send $c_1 \rightarrow^R Z_p$ c1→RZp to P^* P∗ and get response z_1 z1
Rewind the prover P^* P∗ to its state after the first message.
Send $c_2 \rightarrow^R Z_p$ c2→RZp to P^* P∗ and get response z_2 z2
Output $\frac{z_1 - z_2}{c_1 - c_2} \in Z_p$ z1−z2c1−c2∈Zp

h

Here, the algorithm fails if $c_1 - c_2 = 0$ c1−c2=0 which happens with probability $\frac{1}{p}$ 1p. As a result, the knowledge error is $\frac{1}{p}$ 1p for provers that always convince the verifier.

Proving that Schnorr's Sigma Protocol is Zero Knowledge

Our goal is to construct an efficient algorithm that will produce a distribution identical to a verifier's view of the protocol. We will only prove HVZK: the non-interactive version of Schnorr's Protocol will automatically become strongly zero knowledge against even malicious verifiers because the protocol transcript will not depend on the verifier at all.

Consider the following algorithm S S:

$z \rightarrow^R Z_p$ z→RZp
$c \rightarrow^R Z_p$ c→RZp
$u \leftarrow \frac{g^z}{h^c}$ u←gzhc
Output (u, c, z) (u,c,z)

Note that we perform the protocol in reverse to guarantee that the verifier's constraints are satisfied without having to perform difficult discrete logarithms. We only satisfy honest-verifier ZK because we implicitly assume that the verifier will generate cc with uniform randomness. Next, note that we arrive at strong zero-knowledge because the transcript/view distributions are identical: $\forall g, h \in G : \{Sim(y)\}$ ∀g,h∈G:{Sim(y)} which equals

$= \{(u, c, z) : c, z \leftarrow^R Z_p, u = \frac{g^z}{h^c}\}$ ={(u,c,z):c,z←RZp,u=gzhc} which equals

$= \{(g^r, c, z) : r, c \leftarrow_R Z_p, z = r + cx\}$ ={(gr,c,z):r,c←RZp,z=r+cx} which equals $= \{View_V((P, V)(g, h))\}$ ={ViewV((P,V)(g,h))}

This holds because each tuple (u, c, z) (u,c,z) is uniformly random such that (u, c, z) (u,c,z) satisfies the constraints $g^z = u * h^c$ gz=u∗hc.

Licensing and Attribution

This tutorial is heavily inspired by the 2019 CS355 course and lecture notes: https://crypto.stanford.edu/cs355/19sp/about/, particularly for the details of Schnorr's Protocol and Proof of Knowledge systems. Special thanks to Floran Tramèr, Dima Kogan, and Henry Corrigan-Gibbs for being such fantastic instructors.

I also found Oded Goldreich's ZK: A Tutorial to be immensely helpful in understanding different variations of zero knowledge. I highly recommend starting there to learn more about zero-knowledge!