Privacy and zero-knowledge proofs

6.1600 Course Staff

Fall 2023

So far, we have discussed several cryptography primitives, from hash functions to encryption schemes, and explored many applications of those primitives to systems security. These primitives have provided security, but in a very all-or-nothing sense: in order to provide broadly applicable security, our definitions required that a certain party (with the key) could either completely decrypt the message, learning the message contents, or cannot do anything with the message at all.

Disclaimer: This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

1 *Zero-knowledge proofs*

Zero-knowledge proofs allow one party (called a prover) to convince another party (called a verifier) that something is true, without revealing any information. More formally, a prover might have some value x, and claims that $y = f(x)$ for some publicly known function f and public value y. The proof takes the form of an interaction between the prover and the verifier, after which the verifier either accepts the proof or rejects it. This is an *interactive proof*, which is quite different from the idea of a *classical proof* that you might write down on paper in a math class.

We expect such a proof to have two properties:

- First, the proof should be *sound*: if the verifier accepts the proof, it should be convinced that the prover really does "know" x.
- Second, the proof should be *zero-knowledge*: the verifier should "learn nothing" about x as a result of participating in the proof.

How can we define the notion of knowledge? That is, what does it mean for the prover to really know x? A naïve definition might say that x is stored somewhere in the prover's memory, but that might be too strong: a prover might "know" x without storing it literally in its memory, perhaps storing an alternate representation. A more general definition is to think of knowledge in terms of extractability: a prover knows x if there exists an efficient algoritm that can deduce the value of x by observing the state of the prover.

A related notion we need to pin down is what it means for the verifier to learn nothing. A good way to define this, it turns out, is to think of "learning nothing" in terms of the verifier being able to simulate its interaction with the prover without actually talking to

the real prover. Specifically, if the verifier can produce a transcript of its hypothetical interaction with the prover, without actually communicating with the prover, we say the verifier learned nothing about x.

Perhaps surprisingly, we can show (though not in this lecture) that any proof at all can be converted into a zero-knowledge proof. In fact, interactive proofs can be used to efficiently prove anything in PSPACE, which is believed to be much larger than NP.

We can think of using zero-knowledge proofs in authentication. The value x known to the prover is the user's secret key, and the public value of y is the user's public key. The authentication protocols we've seen so far, such as challenge-response protocols using MACs or signatures, are not quite zero-knowledge: the server learns the signature of a server-chosen challenge value under the client's (prover's) secret key, and they could not produce a simulated transcript that has a valid signature without actually interacting with the real prover.

2 *Discrete log problem and Schnorr signatures*

As an example of zero-knowledge proofs, we will use the discrete log problem: given a group $\mathcal{G}$, a generator g of order q for $\mathcal{G}$ (where q is a big prime), and some value $g^x \in \mathcal{G}$, what is the value of $x \in \mathbb{Z}_q$? We have already seen the discrete log problem in the context of Diffie-Hellman key exchange and elliptic-curve signatures.

Imagine a prover that knows $x \in \mathbb{Z}_q$, where x might be chosen randomly, and a verifier that knows $X = g^x \in \mathcal{G}$. How can the prover convince the verifier it knows the x corresponding to X? One protocol for doing this, called Schnorr's protocol, is as follows:

- Prover picks a random $r \in \mathbb{Z}_q$, and sends $R = g^r \in \mathcal{G}$ to the verifier.
- The verifier picks a challenge bit c at random, either 0 or 1, and sends the challenge to the prover.
- The prover sends back a response z, chosen as follows: if $c = 0$, the prover sets z to r, and if $c = 1$, the prover sets z to $r + x \in \mathbb{Z}_q$. Mathematically, $z = r + cx \in \mathbb{Z}_q$.
- The verifier checks that $g^z = RX \in \mathcal{G}$. If equal, the verifier accepts, otherwise the verifier rejects.

This protocol critically depends on c not being known to the prover in advance. If the prover knew c in advance, the prover could pick z at random, and just compute R as $(g^r)(X^c)^{-1} \in \mathcal{G}$. The protocol also depends on the verifier not asking for answers to both values

of c: otherwise, the verifier can learn x by subtracting the responses for the two values of c. However, running the protocol just once does not reveal information about x: even if $c = 1$, adding a random value r to x modulo q makes z effectively random as well.

In any given interaction of the protocol, the prover could falsely convince the verifier that it knows x, using the attack we just described, but by repeating this protocol λ times, the verifier can reduce the probability of an unsound proof (called the "soundness error") to $2^{-\lambda}$.

We will now flesh out this argument—that Schnorr's protocol is a zero-knowledge proof—in more detail. This will amount to showing that the protocol protects the two parties in the protocol (the prover and the verifier) from each other. A verifier can be sure that the protocol is sound, regardless of what adversarial prover it interacts with: if the verifier accepts, the prover really knew x. A prover can also be sure that the protocol is zero-knowledge, regardless of what adversarial verifier it interacts with: if the prover participates in the protocol with an arbitrary verifier, it can be sure that the verifier did not learn anything about its secret x.

2.1 *Soundness using an extractor*

To show that Schnorr's protocol is a zero-knowledge proof, we need to first convince ourselves that the protocol is *sound*: that is, if the verifier accepts, the prover really knows x. We can prove soundnes by showing that, if there is a (possibly adversarial) prover P that can convince our verifier V using Schnorr's protocol, there exists an efficient extractor E that can return x based on P's interaction with V, with some high probability (say, $\geq 1/2$).

The extractor E for Schnorr's protocol can be constructed as follows:

- E runs P to get some initial transcript with challenge 0. The transcript consists of $\langle R, c = 0, z \rangle$.
- Rewind the state of the prover to just after the point when it sent R to V.
- Run P again to get the transcript $\langle R, c = 1, z' \rangle$.
- Output $z' - z$ as the discrete log x.

By our assumption—namely, that P can convince our verifier V—both runs of P by the extractor must convince V to accept (otherwise, P would not convince V in at least one of the two c choices). That means that both $g^z = R$ and that $g^{z'} = RX$. Thus, $g^{z'-z} = X$, which

is exactly the definition of what it means for something ($z' - z$ in particular) to be the discrete log of X.

Here we have glossed over the details of what happens if the prover convinces the verifier with some non-negligible probability that is nonetheless less than 1.

2.2 *Zero-knowledge using a simulator*

The second part of showing that Schnorr's protocol is zero-knowledge requires us to show that the verifier learns nothing as a result. To do this, we construct a *simulator* S. The simulator's job is to convince an arbitrary verifier V to accept, without actually knowing x, thereby producing a transcript that looks indistinguishable from V interacting with the real prover P.

We can construct this simulator S as follows:

- Run the cheating strategy of P with V, where P guesses the challenge upfront.
- If the guess turns out to be correct, output the transcript. This happens with probability $1/2$.
- If the guess was wrong, retry. This happens with probability $1/2$ as well.

Both the extractor and simulator constructions rely crucially on being able to rewind the prover or verifier and try again. This enables us to convince ourselves that the protocol is sound and zero-knowledge, but in a real-world interaction, the verifier cannot rewind the prover and the prover cannot rewind the verifier.

3 *Fiat-Shamir heuristic and Schnorr signatures*

One challenge with zero-knowledge proofs is that they are interactive, requiring the prover and verifier to send messages back and forth in order for the verifier to be convinced of the proof. A clever trick, called the Fiat-Shamir heuristic, allows us to turn interactive zero-knowledge proofs into non-interactive proofs. The idea is that, whenever the protocol expects the verifier to send a challenge to the prover, the challenge should just be the hash of the transcript so far in protocol's execution. The prover can then execute the zero-knowledge protocol against a synthetic verifier (the hash function), and produce a transcript. The prover can now send this transcript to the verifier, and if the verifier can confirm that indeed all of the challenges were correctly computed using a collision-resistant hash function (like SHA-256), it's sound to accept this transcript as a proof.

We can use this Fiat-Shamir heuristic to construct a (regular, non-interactive) signature scheme from an (interactive) zero-knowledge proof of the discrete log problem. This requires a slight modification of the protocol we described above, where instead of sending a single challenge bit c, the verifier sends a 256-bit vector of bits $\overrightarrow{c}$. This also requires a slight modification to the zero-knowledge definition, requiring an honest verifier, which we won't discuss the details of. The elliptic-curve signatures used in practice today are effectively variants of this Schnorr signature scheme.