0.1 Motivation

Finally, we came to the most interesting part of the course: zero-knowledge proofs. Before we start with SNARKs, STARKs, Bulletproofs, and other zero-knowledge proof systems, let us first define what the zero-knowledge is. But even before that, we need to introduce some formalities. For example, what are "proof", "witness", and "statement" — terms that are so widely used in zero-knowledge proofs.

Let us describe the typical setup. We have two parties: **prover** $\mathcal{P}$ and a **verifier** $\mathcal{V}$. The prover wants to convince the verifier that some statement is true. Typically, the statement is not obvious (well, that is the reason for building proofs after all!) and therefore there might be some "helper" data, called **witness**, that helps the prover to prove the statement. The reasonable question is whether the prover can simply send witness to verifier and call it a day. Of course since you are here, reading this lecture, it is obvious that the answer is no. More specifically, by introducing zk-SNARKs, STARKs, and other proving systems, we will try to mitigate the following issues:

- **Zero-knowledge:** The prover wants to convince the verifier that the statement is true without revealing the witness.
- **Argument of knowledge:** Moreover, typically we want to make sure that the verifier, besides the statement correctness, ensures that the prover **knows** such a witness related to the statement.
- **Succinctness:** The proof should be short, ideally logarithmic in the size of the statement. This is crucial for practical applications, especially in the blockchain space where we cannot allow to publish long proofs on-chain.

Example. Suppose, given a hash function[a] $H : \{0,1\}^* \to \{0,1\}^\ell$, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that he knows the preimage $x \in \{0,1\}^*$ such that $H(x) = y$ for some given public value $y \in \{0,1\}^\ell$. The properties listed above are interpreted as follows:

- **Zero-knowledge:** The prover $\mathcal{P}$ does not want to reveal *anything* about the pre-image x to the verifier $\mathcal{V}$.
- **Argument of knowledge:** Given a string $y \in \{0,1\}^\ell$ it is not sufficient for a prover to merely state that y has a pre-image. The prover $\mathcal{P}$ must demonstrate to a verifier $\mathcal{V}$ that he **knows** such a pre-image $x \in \{0,1\}^*$.
- **Succinctness:** If the hash function takes n operations to compute, the proof should be **much** shorter than n operations. State-of-art solutions can provide proofs that are $O(\log n)$ is size!

[a] The notation $\{0,1\}^*$ means binary strings of arbitrary length

0.2 Relations and Languages

Recall that **relation** $\mathcal{R}$ is just a subset of $\mathcal{X} \times \mathcal{Y}$ for two arbitrary sets $\mathcal{X}$ and $\mathcal{Y}$. Now, we are going to intrduce the notion of a *language of true statements* based on $\mathcal{R}$.

Definition 0.1 (Language of true statements)**.** Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be a relation. We say that a statement $x \in \mathcal{X}$ is a **true** statement if $(x, y) \in \mathcal{R}$ for some $y \in \mathcal{Y}$, otherwise the statement is called **false**. We define by $\mathcal{L}_{\mathcal{R}}$ (the language over relation $\mathcal{R}$) the set of all true statements, that is:

$$\mathcal{L}_{\mathcal{R}} = \{x \in \mathcal{X} : \exists y \in \mathcal{Y} \text{ such that } (x, y) \in \mathcal{R}\}.$$

Now, what is the purpose of introducing relations and languages? The answer is simple: suppose we have a prover $\mathcal{P}$ and verifier $\mathcal{V}$. The prover wants to convince the verifier that a statement $x \in \mathcal{X}$ is true, that is $x \in \mathcal{L}_{\mathcal{R}}$. However, the verifier does not trust the prover and wants to be sure that the statement is indeed true. This is where the notion of a *proof* comes into play. Moreover, typically while proving the statement, the prover does not want to reveal any information about the *witness* $y \in \mathcal{Y}$.

Further, we denote by $w \in \mathcal{W}$ the witness for the statement $x \in \mathcal{L}_{\mathcal{R}}$. Oftentimes, one might also encounter notation ϕ to denote the statement, but we will stick to x for simplicity.

Example. Suppose we want to prove the following claim: number $n \in \mathbb{N}$ is the product of two large prime numbers $(p, q) \in \mathbb{N} \times \mathbb{N}$. Here, the relation is the following:

$$\mathcal{R} = \{(n, p, q) \in \mathbb{N}^3 : n = p \cdot q \text{ where } p, q \text{ are primes}\}$$

In this particular case, the language of true statements is defined as

$$\mathcal{L}_{\mathcal{R}} = \{n \in \mathbb{N} : \exists p, q \text{ are primes such that } n = pq\}$$

Therefore, our initial claim we want to prove is $n \in \mathcal{L}_{\mathcal{R}}$. The witness for this statement is the pair (p, q), where p and q are prime numbers such that $n = p \cdot q$ and typically (but not always) we want to prove this without revealing our witness: p and q. For example, one valid witness for $n = 15$ is $(3, 5)$, while $n = 16$ does not have any valid witness, so $16 \notin \mathcal{L}_{\mathcal{R}}$.

Example. Another example of claim we want to prove is the following: number $y \in \mathbb{Z}_N^*$ is a quadratic residue modulo N, meaning there exists some $x \in \mathbb{Z}_N^*$ such that $y \equiv x^2 \pmod{N}$. The relation in this case is:

$$\mathcal{R} = \{(x, y) \in (\mathbb{Z}_N^*)^2 : y \equiv x^2 \pmod{N}\}$$

In this case, $\mathcal{L}_{\mathcal{R}} = \{y \in \mathbb{Z}_N^* : \exists x \in \mathbb{Z}_N^* \text{ such that } y \equiv x^2 \pmod{N}\}$. Here, our *square root of y modulo N*, that is x, is the witness for the statement $y \in \mathcal{L}_{\mathcal{R}}$. For example, for $N = 7$ we have $4 \in \mathcal{L}_{\mathcal{R}}$ since 5 is a valid witness: $5^2 \equiv 4 \pmod{7}$, while $3 \notin \mathcal{L}_{\mathcal{R}}$ since there is no valid witness for 3.

However, we want to limit ourselves to languages that has reasonably efficient verifier (since otherwise the problem is not really practical and therefore of little interest to us). For that reason, we define the notion of a *NP Language* and from now on, we will be working with such languages.

Definition 0.2 (NP Language)**.** A language $\mathcal{L}_{\mathcal{R}}$ belongs to the NP class if there exists a polynomial-time verifier $\mathcal{V}$ such that the following two properties hold:

- **Completeness:** If $x \in \mathcal{L}_{\mathcal{R}}$, then there is a witness $w \in \mathcal{W}$ such that $\mathcal{V}(x, w) = 1$ with $|w| = \mathsf{poly}(|x|)$. Essentially, it states that true claims have *short*[a] proofs.
- **Soundness:** If $x \notin \mathcal{L}_{\mathcal{R}}$, then for any $w \in \mathcal{W}$ it holds that $\mathcal{V}(x, w) = 0$. Essentially, it states that false claims have no proofs.

[a] "Short" is a pretty relative term which would further differ based on the context. Here, we assume that the proof is "short" if it can be computed in polynomial time. However, in practice, we will want to make the proofs even shorter: see SNARKs and STARKs.

However, this construction on its own is not helpful to us. In particular, without having any randomness and no interaction, building practical proving systems is very hard. Therefore, we need some more ingredients!

0.3 Interactive Probabilistic Proofs

As mentioned above, we will bring two more ingredients to the table: **randomness** and **interaction**:

- **Interaction:** rather than simply passively receiving the proof, the verifier $\mathcal{V}$ can interact with the prover $\mathcal{P}$ by sending challenges and receiving responses.
- **Randomness:** verifier can send random coins (challenges) to the prover, which the prover can use to generate responses.

Remark. For those who have already worked with SNARKs, the above introduction might seem very confusing. After all, what we are aiming for is to build **non-interactive** zero-knowledge proofs. However, as it turns out, there are a plenty of ways to make *some* interactive proofs non-interactive. We will discuss this in more detail later.

Now, one of the drastic changes is the following: if $x \notin \mathcal{L}_{\mathcal{R}}$, then the verifier $\mathcal{V}$ should reject the claim with overwhelming[1] probability (in contrast to strict probability of 1). Let us consider the concrete example.

0.3.1 Example: Quadratic Residue Test

Again, suppose for relation $\mathcal{R} = \{(x, y) \in (\mathbb{Z}_N^*)^2 : y \equiv x^2 \pmod{n}\}$ and corresponding language $\mathcal{L}_{\mathcal{R}} = \{y \in \mathbb{Z}_N^* : \exists x \in \mathbb{Z}_N^* \text{ such that } y \equiv x^2 \pmod{n}\}$ the prover $\mathcal{P}$ wants to convince the verifier that the given y is in language $\mathcal{L}_{\mathcal{R}}$. Again, sending x is not an option, as we want to avoid revealing the witness. So how can we proceed? The idea is that the prover $\mathcal{P}$ should prove that he *could* prove it if he felt like it.

So here how it goes. The prover $\mathcal{P}$ can first sample a random $r \xleftarrow{R} \mathbb{Z}_N^*$, calculate $s \leftarrow r^2 \pmod{n}$ and say to the verifier $\mathcal{V}$:

- Hey, I could give you the square roots of both s and sy and that would convince you that the statement is true! But in this case, you would know x^2.
- So instead of providing both values simultaneously, you will choose which one you want

[1] Some technicality: as you know from the Lecture 2, the value $\varepsilon = \mathsf{negl}(\lambda)$ is called negligible since it is very close to 0. In turn, the value $1 - \varepsilon$ is called *overwhelming* since it is close to 1.

[2] If verifier gets both r and rx, he can divide the latter by former and get x

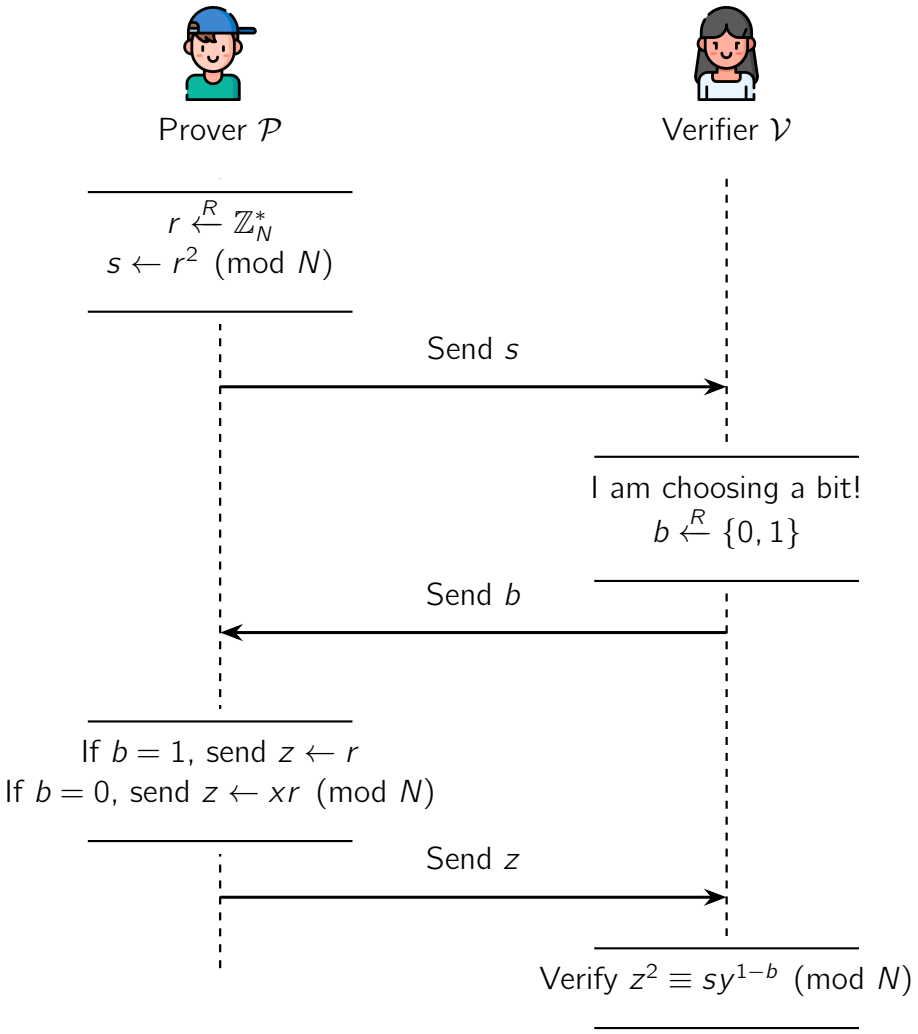

Figure 1: The interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$ for the quadratic residue test.

to see: either r or ry. This way, after a couple of such rounds, you will not learn x but you will be convinced that I know it.

That being said, formally the interaction between prover $\mathcal{P}$ and verifier $\mathcal{V}$ can be described as follows:

1. $\mathcal{P}$ samples $r \xleftarrow{R} \mathbb{Z}_N^*$ and sends $s \leftarrow r^2 \pmod n$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random bit $b \xleftarrow{R} \{0,1\}$ to $\mathcal{P}$.
3. If $b = 1$, the prover sends $z \leftarrow r$, otherwise, if $b = 0$, he sends $z \leftarrow rx \pmod n$.
4. The verifier checks whether $z^2 \equiv sy^{1-b} \pmod n$.
5. Repeat the process for $\lambda \in \mathbb{N}$ rounds.

Now, let us show that the provided protocol is indeed **complete** and **sound**.

Completeness. Suppose the verifier chose $b = 1$ and thus the prover sent r. The check would be $r^2 \equiv sy^{1-1} \pmod N$ which is equivalent to $r^2 \equiv s \pmod N$. This obviously holds.

If, in turn, the verifier chose $b = 0$ and the prover sent rx, the check would be $(rx)^2 \equiv sy^{1-0}$

(mod N) which is equivalent to $r^2x^2 \equiv sy \pmod{N}$. Since $s = r^2 \pmod{N}$ and $y = x^2 \pmod{N}$, this check also obviously holds.

Soundness. Here, we need to prove that for any dishonest prover who does not know x, the verifier will reject the claim with overwhelming probability. One can show the following fact.

Proposition 0.3. If $y \notin \mathcal{L}_{\mathcal{R}}$, then for any prover $\mathcal{P}$, the verifier $\mathcal{V}$ will reject the claim with probability at least $1/2$.

By making λ rounds, the probability of rejection is $\left(\frac{1}{2}\right)^{\lambda} = \mathsf{negl}(\lambda)$ and therefore the verifier can be convinced that $y \in \mathcal{L}_{\mathcal{R}}$ with overwhelming probability of $1 - 2^{-\lambda}$.

To denote the interaction between algorithms $\mathcal{P}$ and $\mathcal{V}$ on the statement x, we use notation $\langle \mathcal{P}, \mathcal{V} \rangle(x)$. Finally, now we are ready to define the notion of an **interactive proof system**.

Definition 0.4. A pair of algorithms $(\mathcal{P}, \mathcal{V})$ is called an **interactive proof** for a language $\mathcal{L}_{\mathcal{R}}$ if $\mathcal{V}$ is a polynomial-time verifier and the following two properties hold:

- **Completeness:** For any $x \in \mathcal{L}_{\mathcal{R}}$, $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle(x) = \mathsf{accept}] = 1$.
- **Soundness:** For any $x \notin \mathcal{L}_{\mathcal{R}}$ and for any prover $\mathcal{P}^*$, we have

$$\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(x) = \mathsf{accept}] \leq \mathsf{negl}(\lambda)$$

Definition 0.5. Besides classes **P** and **NP**, we now have one more class: **the class of interactive proofs** (**IP**):

$$\mathbf{IP} = \{\mathcal{L} : \text{there is an interactive proof } (\mathcal{P}, \mathcal{V}) \text{ for } \mathcal{L}\}.$$

0.4 Schnorr's Identification Protocol

One very useful protocol for demonstration purposes is Schnorr's identification protocol. It is a simple and elegant protocol that allows one party to prove to another party that it knows a discrete logarithm of a given element.

Suppose $\mathbb{G}$ is a cyclic group of prime order q with generator $g \in \mathbb{G}$. Suppose prover $\mathcal{P}$ has a secret key $\alpha \in \mathbb{Z}_q$ and the corresponding public key $u = g^{\alpha} \in \mathbb{G}$ and he wants to convince the verifier $\mathcal{V}$ that he knows α corresponding to the public key u.

Well, the easiest way how to proceed is simply giving α to $\mathcal{V}$, but this is obviously not what we want. Instead, the Schnorr protocol allows $\mathcal{P}$ to prove the knowledge of α without revealing it.

Let us finally describe the protocol. The schnorr identification protocol $\Pi_{\mathsf{Schnorr}} = (\mathsf{Gen}, \mathcal{P}, \mathcal{V})$ with a generation function Gen and prover $\mathcal{P}$ and verifier $\mathcal{V}$ is defined as follows:

- $\mathsf{Gen}(1^{\lambda})$: As with most public-key cryptosystems, we take $\alpha \xleftarrow{R} \mathbb{Z}_q$ and $u \leftarrow g^{\alpha}$. We output the *verification key* as $\mathsf{vk} := u$, and the *secret key* as $\mathsf{sk} := \alpha$.
- The protocol between $(\mathcal{P}, \mathcal{V})$ is run as follows:
  - $\mathcal{P}$ computes $\alpha_T \leftarrow \mathbb{Z}_q$, $u_T \leftarrow g^{\alpha_T}$ and sends u_T to $\mathcal{V}$.
  - $\mathcal{V}$ sends a random challenge $c \xleftarrow{R} \mathbb{Z}_q$ to $\mathcal{P}$.
  - $\mathcal{P}$ computes $\alpha_C \leftarrow \alpha_T + \alpha c \in \mathbb{Z}_q$ and sends α_C to $\mathcal{V}$.

– $\mathcal{V}$ accepts if $g^{\alpha_C} = u_T \cdot u^c$, otherwise it rejects.