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. Moreover, verification should be efficient as well.

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[b], the proof should be **much** shorter than n operations. State-of-art

solutions can provide proofs that are $O((\log n)^c)$ (polylogarithmic) is size! Moreover, verification time of such proof is also typically polylogarithmic (or even $O(1)$ in some cases).

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

[b] Note that "number of operations" is very vague term. One way to measure the "size" of some computational problem is specifying the number of gates in the arithmetical circuit $C(x, w)$, representing the computation of this problem (denoted as $|C|$, respectively).

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 introduce 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 idea is that relation is a natural way to formalize the notion of a statement and witness. Namely, we denote the elements of $\mathcal{X}$ as statements and the elements of $\mathcal{Y}$ as witnesses.

Further, we denote by 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 $x \in \mathbb{Z}_N^{\times}$[a] is a **quadratic residue** modulo N, meaning there exists some $w \in \mathbb{Z}_N^{\times}$ such that $x \equiv w^2 \pmod{N}$ (naturally, w is called a *square root* of x). The relation in this case is:

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

In this case, $\mathcal{L}_{\mathcal{R}} = \{x \in \mathbb{Z}_N^{\times} : \exists w \in \mathbb{Z}_N^{\times} \text{ such that } x \equiv w^2 \pmod{N}\}$. Here, our square root of x modulo N, that is w, is the witness for the statement $x \in \mathcal{L}_{\mathcal{R}}$. 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.

[a] By $\mathbb{Z}_N^{\times}$ we denote the multiplicative group of integers modulo N. In other words, this is a set of integers $\{x \in \mathbb{Z}_N : \gcd\{x, N\} = 1\}$.

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 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 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 to make proving NP statements easier.

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 $\mathcal{R} = \{(x, w) \in (\mathbb{Z}_N^\times)^2 : x \equiv w^2 \pmod N\}$ and corresponding language $\mathcal{L}_{\mathcal{R}} = \{x \in \mathbb{Z}_N^\times : \exists w \in \mathbb{Z}_N^\times \text{ such that } x \equiv w^2 \pmod N\}$ the prover $\mathcal{P}$ wants to convince the verifier that the given x is in language $\mathcal{L}_{\mathcal{R}}$. Again, sending w 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 $r \xleftarrow{R} \mathbb{Z}_N^\times$, calculate $a \leftarrow r^2 \pmod N$ and say to the verifier $\mathcal{V}$:

- Hey, I could give you the square roots of both a and $ax \pmod N$ and that would convince you that the statement is true! But in this case, you would know w[2].
- So instead of providing both values simultaneously, you will choose which one you want to see: either r or $r \times w \pmod N$. This way, after a couple of such rounds, you will not learn w 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] 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 $rw \pmod N$, he can divide the latter by former and get w

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

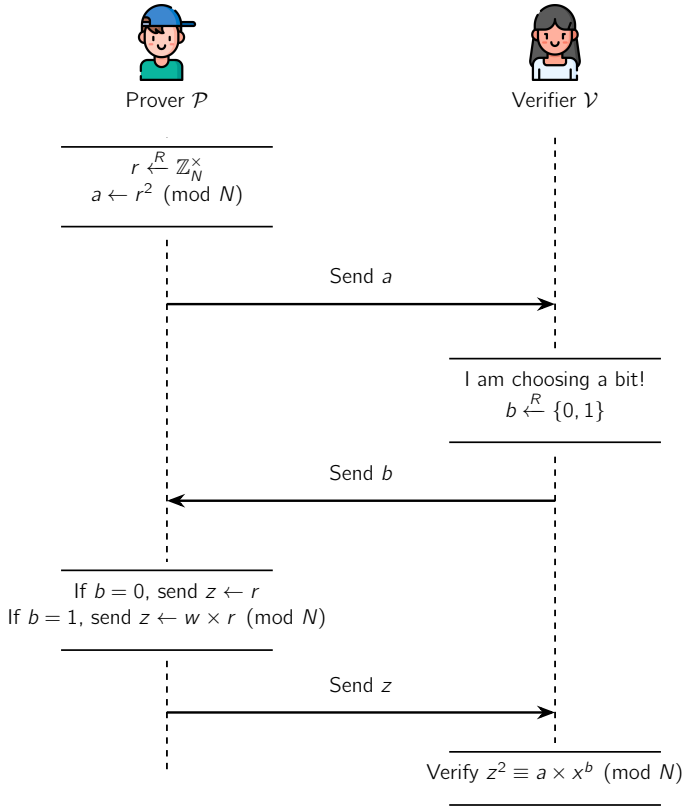

Figure 0.1: The interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$ for the quadratic residue test.

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

Completeness. Suppose the verifier chose $b = 0$ and thus the prover has sent $z = r$. The check would be $r^2 \equiv a \times x^0 \pmod N$ which is equivalent to $r^2 \equiv a \pmod N$. This obviously holds.

If, in turn, the verifier chose $b = 1$ and the prover sent rw, the check would be $(rw)^2 \equiv ax^{1-0} \pmod N$ which is equivalent to $r^2w^2 \equiv ax \pmod N$. Since $a = r^2 \pmod N$ and $x = w^2 \pmod N$, this check also obviously holds.

Soundness. Here, we need to prove that for any dishonest prover who does not know w, the verifier will reject the claim with overwhelming probability. One can show the following, which we are not going to prove (yet, this is quite easy to show):

Proposition 0.3. If $x \notin \mathcal{L}_\mathcal{R}$, 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 $x \in \mathcal{L}_\mathcal{R}$ 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 Zero-Knowledge

Turns out that defining the zero-knowledge to even such a simplistic interactive proof system is not that easy. Informally, we give the following definition.

Definition 0.6. An interactive proof system $(\mathcal{P}, \mathcal{V})$ is called **zero-knowledge** if for any polynomial-time verifier $\mathcal{V}^*$ and any $x \in \mathcal{L}_\mathcal{R}$, the interaction $\langle \mathcal{P}, \mathcal{V}^* \rangle(x)$ gives nothing new about the witness w.

Definition 0.7. The pair of algorithms $(\mathcal{P}, \mathcal{V})$ is called a **zero-knowledge interactive protocol** if it is *complete*, *sound*, and *zero-knowledge*.

Basically, the specified interaction is a **proof**! The prover $\mathcal{P}$ can convince the verifier $\mathcal{V}$ that the statement is true without revealing the witness – that is what we need (quite of).

Remark. The above definition is very informal and, for the most part, complete for the purposes of this course. If you do not want to dive into the formalities, you can skip the next part of this section. However, if you are curious about some technicalities, feel free to continue reading.

0.4.1 The Verifier's View

Suppose that the interaction between $\mathcal{V}$ and $\mathcal{P}$ has ended with the successful verification. What has $\mathcal{V}$ learned? Well, first things first, he has learned that the statement is true, that is $x \in \mathcal{L}_{\mathcal{R}}$. However, he has also learned something more: he has learned the transcript of the interaction, that is the sequence of messages between $\mathcal{P}$ and $\mathcal{V}$.

Definition 0.8. Interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of prover's messages (e.g., commitments and responses) $(m_1, m_2, \ldots, m_\ell)$, verifier's queries $(q_1, q_2, \ldots, q_\ell)$, and $\mathcal{V}$'s random coins $(r_1, r_2, \ldots, r_\ell)$. The view of the verifier $\mathcal{V}$, denoted as $\mathsf{view}_{\mathcal{V}}(\mathcal{P}, \mathcal{V})[x]$, is a random variable $(m_1, r_1, q_1, m_2, r_2, q_2, \ldots, m_\ell, r_\ell, q_\ell)$ that is determined by the random coins of $\mathcal{V}$ and the messages of $\mathcal{P}$ after the interaction with the statement x. See Figure below.

Example. Suppose that for the aforementioned protocol with $N = 3 \times 2^{30} + 1$, the conversation between the prover $\mathcal{P}$, who wants to convince that $1286091780 \in \mathcal{L}_R$, and $\mathcal{V}$ is the following:

1. During the first round, $\mathcal{P}$ sends 672192003 to $\mathcal{V}$.
2. $\mathcal{V}$ sends $b = 0$ to $\mathcal{P}$.
3. $\mathcal{P}$ sends 2606437826 to $\mathcal{V}$.
4. $\mathcal{V}$ verifies that indeed $2606437826^2 \equiv 672192003 \pmod{N}$.
5. During the second round, $\mathcal{P}$ sends 2619047580 to $\mathcal{V}$.
6. $\mathcal{V}$ chooses $b = 1$ and sends to $\mathcal{P}$.
7. $\mathcal{P}$ sends 1768388249 to $\mathcal{V}$.
8. $\mathcal{V}$ verifies that $1768388249^2 \equiv 2619047580 \times 1286091780 \pmod{N}$.
9. Conversation ends.

The view of the verifier $\mathcal{V}$ is the following:

$$\mathsf{view}_{\mathcal{V}}(\mathcal{V}, \mathcal{P})[1286091780] = (672192003, 0, 2606437826, 2619047580, 1, 1768388249)$$

Essentially, the view that you currently has witnessed is the same as one that $\mathcal{V}$ has seen. After this interaction, you have not learned anything about the witness w that prover $\mathcal{P}$ knows and which we, as of now, has not revealed to you.
In fact, you can verify by yourself, that the witness was $w = 3042517305$ and two randomnesses were $r_1 = 2606437826$ and $r_2 = 3023142760$.
One final note that is essential for the further discussion: variable $\mathsf{view}_{\mathcal{V}}(\mathcal{P}, \mathcal{V})[x]$ is a random variable. For example, for our particular case, both bits could be 0 or both bits could be 1.

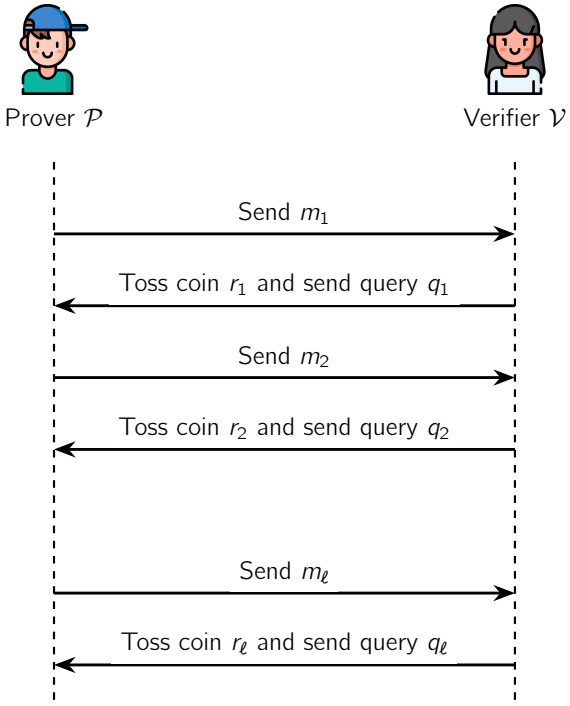

Figure 0.2: The interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$. Prover's messages consist of messages $\{m_k\}_{k=1}^{\ell}$, verifier's messages consist of queries $\{q_k\}_{k=1}^{\ell}$, and additionally verifier samples random coins $\{r_k\}_{k=1}^{\ell}$.

0.4.2 The Simulation Paradigm

The key idea is the following: $\mathsf{view}_{\mathcal{V}}(\mathcal{V}, \mathcal{P})[x]$ gives nothing new to the verifier $\mathcal{V}$ about the witness w. But it gives nothing new, if he could have *simulated* this view on his own, without even running the interaction. In other words, the "simulated" and "real" views should be *computationally-indistinguishable*. But let us define the computational indistinguishability first.

Definition 0.9 (Computational Indistinguishability)**.** Given two random distributions D_0 and D_1, define the following challenger-adversary game:

1. The challenger randomly samples $x_0 \xleftarrow{R} D_0, x_1 \xleftarrow{R} D_1$ and a bit $b \xleftarrow{R} \{0,1\}$.
2. The challenger sends (x_0, x_1, b) to the adversary.
3. The adversary $\mathcal{A}$ outputs a bit $\hat{b}$.

We define the advantage of the adversary $\mathcal{A}$ in distinguishing D_0 and D_1 as

$$\mathsf{Indadv}[\mathcal{A}, D_0, D_1] = \left| \Pr[b = \hat{b}] - \frac{1}{2} \right|$$

Distributions D_0 and D_1 are called **computationally indistinguishable**, denoted as $D_0 \approx D_1$, if for any polynomial-time adversary $\mathcal{A}$ and polynomial number of rounds in the game, the advantage $\mathsf{Indadv}[\mathcal{A}, D_0, D_1]$ is negligible.

Finally, we are ready to define the **zero-knowledge**.

Definition 0.10 (Zero-Knowledge)**.** An interactive protocol $(\mathcal{P}, \mathcal{V})$ is **zero-knowledge** for a language $\mathcal{L}_{\mathcal{R}}$ if for every poly-time verifier $\mathcal{V}^*$ there exists a poly-time simulator Sim such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$:

$$\mathsf{view}_{\mathcal{V}^*}(\mathcal{P}, \mathcal{V}^*)[x] \approx \mathsf{Sim}(x)$$

However, the condition that verifier might be arbitrary is rather strong. Therefore, we introduce the notion of **honest-verifier zero-knowledge**.

Definition 0.11. Honest-Verifier Zero-Knowledge (HVZK) An interactive protocol $(\mathcal{P}, \mathcal{V})$ is **honest-verifier zero-knowledge** for a language $\mathcal{L}_{\mathcal{R}}$ if there exists a probabilistic poly-time simulator Sim such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$[a]:

$$\mathsf{view}_{\mathcal{V}}(\mathcal{P}, \mathcal{V})[x] \approx \mathsf{Sim}(x)$$

[a] Below, we assume that the verifier $\mathcal{V}$ is honest: he is following the protocol.

0.5 Proof of Knowledge

Now, the main issue with the above definition is that *we have proven the statement correctness, but we have not proven that the prover* **knows** *the witness*. These are completely two different things. Let us demonstrate why.

Example. Suppose that the prover $\mathcal{P}$ wants to convince the verifier that he knows the discrete logarithm of a given point $P \in E(\mathbb{F}_p)$ on a cyclic elliptic curve $E(\mathbb{F}_p)$ of order r. This corresponds to the relation and the corresponding language:

$$\mathcal{R} = \{(P, \alpha) \in E(\mathbb{F}_p) \times \mathbb{Z}_r : P = [\alpha]G\},$$
$$\mathcal{L}_{\mathcal{R}} = \{P \in E(\mathbb{F}_p) : \exists \alpha \in \mathbb{Z}_r \text{ such that } P = [\alpha]G\}$$

But here is the catch: actually, $\mathcal{L}_{\mathcal{R}} = E(\mathbb{F}_p)$ since any point P has a witness α such that $P = [\alpha]G$ (recall that the curve is cyclic)! So proving that $P \in \mathcal{L}_{\mathcal{R}}$ is completely useless! Rather, we want to prove that the prover knows α, not the fact that the point has a discrete logarithm.

That is why instead of **proof**, we need a **proof of knowledge**. This leads to even another weird paradigm used for the rigorous definition: the **extractor**. Basically, the knowledge of witness means that we can *extract* the witness while interacting with the prover. Yet, the *extractor* can do more than the verifier: he can call the prover however he wants and he can also rewind the prover (for example, run some pieces multiple times). This sometimes is referred to as "extractor $\mathcal{E}$ uses $\mathcal{P}$ as an oracle".

Now, let us move to the formal definition.

Definition 0.12 (Proof of Knowledge)**.** The interactive protocol $(\mathcal{P}, \mathcal{V})$ is a **proof of knowledge** for $\mathcal{L}_{\mathcal{R}}$ if exists a poly-time extractor algorithm $\mathcal{E}$ such that for any valid statement $x \in \mathcal{L}_{\mathcal{R}}$, in expected poly-time $\mathcal{E}^{\mathcal{P}}(x)$ outputs w such that $(x, w) \in \mathcal{R}$.

Lemma 0.13. The protocol from Section 0.3.1 is a proof of knowledge for the language $\mathcal{L}_{\mathcal{R}}$.

Proof. Let us define the extractor $\mathcal{E}$ for the statement x as follows:

1. Run the prover to receive $a \equiv r^2 \pmod{N}$ (r is chosen randomly from $\mathbb{Z}_N^*$).
2. Set verifier's message to $b = 0$ to get $z_1 \leftarrow r$.

3. **Rewind** and set verifier's message to $b = 1$ to get $z_2 \leftarrow rw \pmod N$.
4. Output $z_2/z_1 \pmod N$

The extractor $\mathcal{E}$ will always output w if $x \in \mathcal{L}_{\mathcal{R}}$. □

Remark. Note that extractor is given much more than the verifier: he can call the prover multiple times and he can also rewind the prover. This is the main difference between the verifier and the extractor.

0.6 Fiat-Shamir Heuristic

0.6.1 Random Oracle

In cryptography, one frequently encounters the term *cryptographic oracle*. In this section, we are not going to dive into the technical details of what that is, yet it is useful to have a general understanding of what it is.

Definition 0.14 (Cryptographic Oracle)**.** Informally, *cryptographic oracle* is simply a function $\mathcal{O}$ that gives in $O(1)$ an answer to some typically very hard problem.

Example. Consider the Computational Diffie-Hellman (CDH) problem on the cyclic elliptic curve $E(\mathbb{F}_p)$ of prime order r with a generator G. Recall that such problem consists in computing $[\alpha\beta]G$ given $[\alpha]G$ and $[\beta]G$ where $\alpha, \beta \in \mathbb{Z}_r$.
Typically, it is believed that the Diffie-Hellman problem is hard (meaning, for any adversary strategy the probability of solving the problem is negligible). However, we *could* assume that such problem can be solved in $O(1)$ by a cryptographic oracle $\mathcal{O}_{\text{CDH}} : ([\alpha]G, [\beta]G) \mapsto [\alpha\beta]G$. This way, we can rigorously prove the security of some cryptographic protocols *even* if the Diffie-Hellman problem is suddenly solved.

One of the most popular cryptographic oracles is the **random oracle** $\mathcal{O}_{\text{R}}$. Let us define how the random oracle works.

Suppose someone is inputting x to the random oracle $\mathcal{O}_{\text{R}}$. The oracle $\mathcal{O}_{\text{R}}$ does the following:

1. If x has been queried before, the oracle returns the same value as it returned before.
2. If x has not been queried before, the oracle returns a randomly uniformly sampled value from the output space.

Remark. Of course, the sudden appearance of the random oracle is not a magic trick. In practice, the random oracle is typically implemented as a hash function. Of course, formally, the hash function is not a random oracle, yet it is a very good approximation and it is reasonable to assume that the hash function behaves like a random oracle.

0.6.2 Fiat-Shamir Transformation

Now, the main issue with the interactive proofs is that they are... Well, *interactive*. Ideally, we simply want to accumulate a proof π, publish it (say, in blockchain) so that anyone (essentially, being the verifier) could check its validity. So we need some tools to make *some* interactive protocols non-interactive. This is, of course, not always possible, but there are some ways to achieve this.

While different protocols use different ways to achieve this, one of the most popular methods (which, in particular, is used in STARKs) is the **Fiat-Shamir heuristic**. The idea is the following: instead of verifier sending the challenges, we can replace them with the random oracle applied to all the previous messages.

Here how it goes. Suppose we have an interactive protocol $(\mathcal{P}, \mathcal{V})$ for the statement x. As previously defined, the interaction between $\mathcal{P}$ and $\mathcal{V}$ consists of prover's messages $(m_1, m_2, \ldots, m_\ell)$, verifier's queries $(q_1, q_2, \ldots, q_\ell)$, and verifier's random coins $(r_1, r_2, \ldots, r_\ell)$. In case all the queries are public random coins, such interactive protocol is called **public-coin protocol** (or, more formally, **Arthur-Merlin protocol**). However, as it turns out, when all the verifier's queries are simply uniformly sampled random values, it is an overkill to use the interactive protocol. Instead, suppose at some point the verifier got messages $m_1, m_2, \ldots, m_{\ell'}$ $(\ell' \leq \ell)$ from the prover. Then, instead of verifier sampling some random value $r_{\ell'}$, we can simply use the random oracle $\mathcal{O}_\mathsf{R}$ as follows: $r_{\ell'} \leftarrow \mathcal{O}_\mathsf{R}(x, m_1, m_2, \ldots, m_{\ell'})$. Practically, instead of random oracle $\mathcal{O}_\mathsf{R}$ we use the hash function H, and use: $r_{\ell'} \leftarrow H(x \parallel m_1 \parallel m_2 \parallel \cdots \parallel m_{\ell'})$.

Remark. Sometimes, to simulate the "interaction" with the verifier, one uses the "Fiat-Shamir Channel". Its main purpose is to simulate the verifier's queries and random coins. For example, one might implement it as a class/struct with the following methods:

1. `send_message(`m`)`: "sends" the message m to the verifier. Under the hood, the proof stream accumulates the current state s and appends m to it.
2. `sample()`: returns the challenge r from the random oracle $\mathcal{O}_\mathsf{R}$, applied to the current state s.
3. `get_proof()`: returns the proof π, being the history of interaction, that the prover can publish.

One can check the `winterfell` Rust library or a simpler non-production implementation of the Fiat-Shamir Channel in Golang for more details.

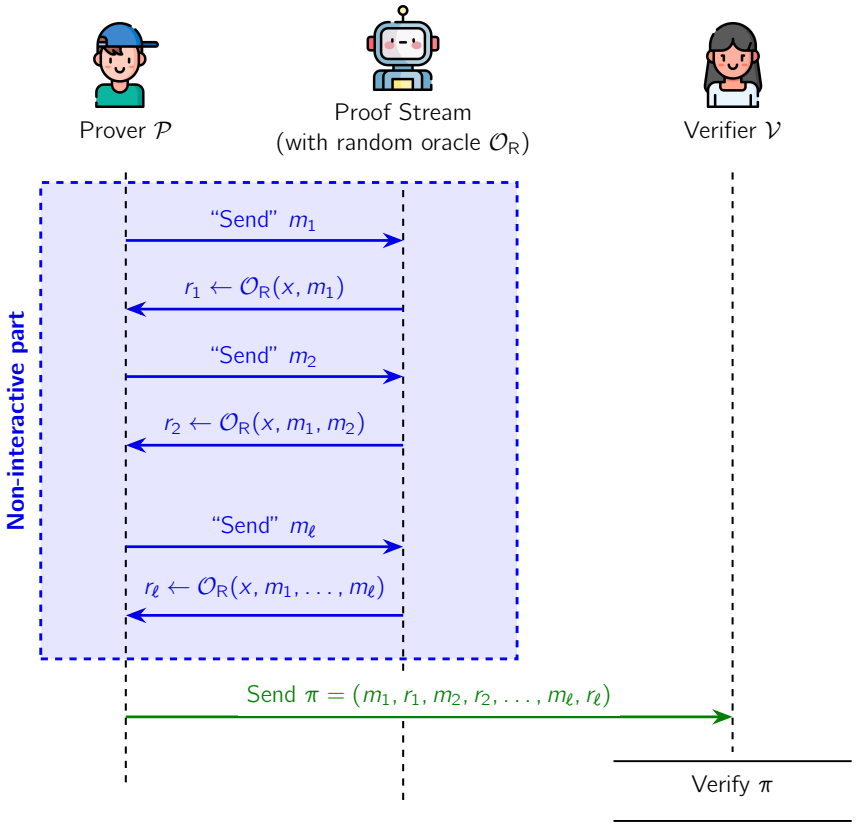

Figure 0.3: The non-interactive protocol between prover $\mathcal{P}$ and verifier $\mathcal{V}$ using Fiat-Shamir Transformation. In **blue** we marked a non-interactive part of the protocol, being the "communication" between a prover and a proof stream. In **green** we marked the final proof π that is sent to the verifier.

The process is illustrated in Figure 0.3. The Fiat-Shamir looks as follows:

1. First, the prover $\mathcal{P}$ "sends" the first message m_1 to the verifier $\mathcal{V}$. Here, "sending" is not an actual sending, but rather its simulation.
2. If we had an interactive protocol, the verifier $\mathcal{V}$ would send the random challenge r_1 to the prover $\mathcal{P}$. Instead, we use the random oracle $\mathcal{O}_\mathsf{R}$ to get $r_1 \leftarrow \mathcal{O}_\mathsf{R}(x, m_1)$.

3. Then, using this challenge, prover does his part in the protocol, and sends the next message m_2.
4. Again, if we had an interactive protocol, the verifier would send the next challenge r_2 to the prover. Instead, we use the random oracle $\mathcal{O}_{\mathsf{R}}$ to get $r_2 \leftarrow \mathcal{O}_{\mathsf{R}}(x, m_1, m_2)$, which gets "sent" to the prover.
5. The process continues until the protocol is finished.

Note that the whole process can be done by a prover with no interaction with the "verifier". In this case, one of the ways to represent the proof π is to publish the transcript of the interaction (that is, all the messages sent by the prover and challenges computed using the random oracle). This is exactly what is done in STARKs.

The reason why such transformation works is that the random oracle $\mathcal{O}_{\mathsf{R}}$ is a *random* function. Therefore, the challenges $r_1, r_2, \ldots$ are *random* values, and the prover cannot predict them (for example, by fabricating messages to have some specific output). That being said, the following theorem holds (which, of course, we are not going to prove since the proof is complicated).

Theorem 0.15. Suppose that $(\mathcal{P}, \mathcal{V})$ is a public-coin interactive argument of knowledge for some language $\mathcal{L}_{\mathcal{R}}$ with a negligible soundness error. Then, the Fiat-Shamir transformation of $(\mathcal{P}^{\mathcal{O}_{\mathsf{R}}}, \mathcal{V}^{\mathcal{O}_{\mathsf{R}}})$ is a non-interactive argument for $\mathcal{L}_{\mathcal{R}}$ with negligible soundness error in the random oracle model $\mathcal{O}_{\mathsf{R}}$.

0.7 Exercises

Exercise 1. When dealing with RSA protocol, one frequently encounters the following relation where e is a prime number and $n \in \mathbb{N}$:

$$\mathcal{R} = \left\{(w, x) \in \mathbb{Z}_n^\times \times \mathbb{Z}_n^\times : w^e = x\right\}$$

Which of the following is the language $\mathcal{L}_\mathcal{R}$ that corresponds to the relation $\mathcal{R}$?

(A) Integers from $\mathbb{Z}_n^\times$ which have a modular root of e-th degree.
(B) Integers from $\mathbb{Z}_n^\times$ which are divisible by e.
(C) Integers x from $\mathbb{Z}_n^\times$ with properly defined expression x^e.
(D) Integers from $\mathbb{Z}_n^\times$ which are prime.
(E) Integers from $\mathbb{Z}_n^\times$ for which e is a primitive root.

Exercise 2. Suppose that for some interactive protocol $(\mathcal{P}, \mathcal{V})$ during one round, the probability that the verifier $\mathcal{V}$ accepts a false statement is $1/8$. How many rounds of interaction are needed to guarantee 120 bits of security? Assume here that n bits of security means that the probability of accepting a false statement is at most 2^{-n}.

(A) 30.
(B) 40.
(C) 60.
(D) 90.
(E) 120.

Exercise 3. Recall that for relation $\mathcal{R} = \{(w, x) \in \mathbb{Z}_N^\times \times \mathbb{Z}_N^\times : x = w^2\}$ we defined the following interactive protocol $(\mathcal{P}, \mathcal{V})$ to prove that $x \in \mathcal{L}_\mathcal{R}$:

- $\mathcal{P}$ samples $r \xleftarrow{R} \mathbb{Z}_N^\times$ and sends $a = r^2$ to $\mathcal{V}$.
- $\mathcal{V}$ sends a random bit $b \in \{0, 1\}$ to $\mathcal{P}$.
- $\mathcal{P}$ sends $z = r \cdot w^b$ to $\mathcal{V}$.
- $\mathcal{V}$ accepts if $z^2 = a \cdot x^b$, otherwise it rejects.

Suppose we use the protocol $(\mathcal{P}, \mathcal{V}^*)$ where the "broken" verifier $\mathcal{V}^*$ always outputs $b = 1$. Which of the following statements is true?

(A) Both the soundness and completeness of the protocol are preserved.
(B) The soundness of the protocol is preserved, but the completeness is broken.
(C) The completeness of the protocol is preserved, but the soundness is broken.
(D) Both the soundness and completeness of the protocol are broken.

Exercise 4. What is the difference between the cryptographic proof and the proof of knowledge?

(A) Cryptographic proof is a proof of knowledge that is secure against malicious verifiers.

(B) Cryptographic proof is a proof of knowledge that is secure against malicious provers.

(C) Cryptographic proof merely states the correctness of a statement, while the proof of knowledge also guarantees that the prover knows the witness.

(D) While cryptographic proof states that witness exists for the given statement, the proof of knowledge makes sure to make this witness unknown to the verifier.

(E) Proof of knowledge does not require verifier to know the statement, while cryptographic proof does.

Exercise 5. What is the purpose of introducing the extractor?

(A) To introduce the algorithm that simulates the malicious verifier trying to extract the witness from the prover.

(B) To define what it means that the prover knows the witness.

(C) To give the verifier the ability to extract the witness from the prover during the interactive protocol.

(D) To define the security of the interactive protocol that uses a more powerful verifier that can extract additional information from the prover.

(E) To give prover more power to extract randomness generated by the verifier.

Exercise 6. What it means that the interactive protocol $(\mathcal{P}, \mathcal{V})$ is a zero-knowledge?

(A) The verifier $\mathcal{V}$ cannot know whether the given statement is true or false.

(B) The verifier $\mathcal{V}$ cannot know whether the prover $\mathcal{P}$ knows the witness.

(C) View of the prover $\mathcal{P}$ in the protocol is indistinguishable from the view of the verifier $\mathcal{V}$.

(D) Any view of any verifier $\mathcal{V}$ can be simulated using some polynomial-time algorithm, outputting computationally indistinguishable distribution from the given view.

(E) The prover $\mathcal{P}$ can convince the verifier $\mathcal{V}$ that the statement is true without knowing the witness.

Hint: View of the participant in the protocol consists of all data he has access to during the protocol execution. For example, verifier $\mathcal{V}$'s view consists of the messages he sends and receives, as well as the random coins he generates.