An Overview of Zero-Knowledge Proofs

How They Work and Why They Matter

Based on https://ethereum.org/en/zero-knowledge-proofs/

What are zero-knowledge proofs?

A zero-knowledge proof is a way of proving the validity of a statement without revealing the statement itself. The ‘prover’ is the party trying to prove a claim, while the ‘verifier’ is responsible for validating the claim.

Zero-knowledge proofs first appeared in a 1985 paper, “The knowledge complexity of interactive proof systems” which provides a definition of zero-knowledge proofs widely used today:

> A zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that something is true, without revealing any information apart from the fact that this specific statement is true.

Zero-knowledge proofs have improved over the years and they are now being used in several real-world applications.

Why do we need zero-knowledge proofs?

Practical issues today

- Proving age: Showing drivers license. Only the birth date being BEFORE a certain date matters, but you're sharing much more than that.
- Proving citizenship: Many will copy your ID or passport, even though most of it is none of their business and can even leak and be abused.
- Proving funds: Right now we share a bank statement with current balance, even though the only info needed is that we have funds ABOVE a certain amount.
- Diplomatic: Countries that have to prove they have no nuclear weapons, without needing to disclose their actual arsenal

Zero-knowledge proofs represented a breakthrough in applied cryptography, as they promised to improve security of information for individuals. Consider how you might prove a claim (e.g., “I am a citizen of X country”) to another party (e.g., a service provider). You’d need to provide “evidence” to back up your claim, such as a national passport or driver’s license.

But there are problems with this approach, chiefly the lack of privacy. Personally Identifiable Information (PII) shared with third-party services is stored in central databases, which are vulnerable to hacks. With identity theft becoming a critical issue, there are calls for more privacy-protecting means of sharing sensitive information.

Zero-knowledge proofs solve this problem by eliminating the need to reveal information to prove validity of claims. The zero-knowledge protocol uses the statement (called a ‘witness’) as input to generate a succinct proof of its validity. This proof provides strong guarantees that a statement is true without exposing the information used in creating it.

Going back to our earlier example, the only evidence you need to prove your citizenship claim is a zero-knowledge proof. The verifier only has to check if certain properties of the proof hold true to be convinced that the underlying statement holds true as well.

How do zero-knowledge proofs work?

A zero-knowledge proof allows you to prove the truth of a statement without sharing the statement's contents or revealing how you discovered the truth. To make this possible, zero-knowledge protocols rely on algorithms that take some data as input and return 'true' or 'false' as output.

A zero-knowledge protocol must satisfy the following criteria:

1. **Completeness**: If the input is valid, the zero-knowledge protocol always returns ‘true’. Hence, if the underlying statement is true, and the prover and verifier act honestly, the proof can be accepted.
2. **Soundness**: If the input is invalid, it is theoretically impossible to fool the zero-knowledge protocol to return ‘true’. Hence, a lying prover cannot trick an honest verifier into believing an invalid statement is valid (except with a tiny margin of probability).
3. **Zero-knowledge**: The verifier learns nothing about a statement beyond its validity or falsity (they have “zero knowledge” of the statement). This requirement also prevents the verifier from deriving the original input (the statement’s contents) from the proof.

In basic form, a zero-knowledge proof is made up of three elements: **witness**, **challenge**, and **response**.

- **Witness**: With a zero-knowledge proof, the prover wants to prove knowledge of some hidden information. The secret information is the “witness” to the proof, and the prover's assumed knowledge of the witness establishes a set of questions that can only be answered by a party with knowledge of the information. Thus, the prover starts the proving process by randomly choosing a question, calculating the answer, and sending it to the verifier.

- **Challenge**: The verifier randomly picks another question from the set and asks the prover to answer it.

- **Response**: The prover accepts the question, calculates the answer, and returns it to the verifier. The prover's response allows the verifier to check if the former really has access to the witness. To ensure the prover isn't guessing blindly and getting the correct answers by chance, the verifier picks more questions to ask. By repeating this interaction many times, the possibility of the prover faking knowledge of the witness drops significantly until the verifier is satisfied.

The above describes the structure of an ‘interactive zero-knowledge proof’. Early zero-knowledge protocols used interactive proving, where verifying the validity of a statement required back-and-forth communication between provers and verifiers.

A good example that illustrates how interactive proofs work is Jean-Jacques Quisquater's famous Ali Baba cave story. In the story, Peggy (the prover) wants to prove to Victor (the verifier) that she knows the secret phrase to open a magic door without revealing the phrase.

Non-interactive zero-knowledge proofs

While revolutionary, interactive proving had limited usefulness since it required the two parties to be available and interact repeatedly. Even if a verifier was convinced of a prover’s honesty, the proof would be unavailable for independent verification (computing a new proof required a new set of messages between the prover and verifier).

To solve this problem, Manuel Blum, Paul Feldman, and Silvio Micali suggested the first non-interactive zero-knowledge proofs where the prover and verifier have a shared key. This allows the prover to demonstrate their knowledge of some information (i.e., witness) without providing the information itself.

Unlike interactive proofs, noninteractive proofs required only one round of communication between participants (prover and verifier). The prover passes the secret information to a special algorithm to compute a zero-knowledge proof. This proof is sent to the verifier, who checks that the prover knows the secret information using another algorithm.

Non-interactive proving reduces communication between prover and verifier, making ZK-proofs more efficient. Moreover, once a proof is generated, it is available for anyone else (with access to the shared key and verification algorithm) to verify.

Non-interactive proofs represented a breakthrough for zero-knowledge technology and spurred the development of proving systems used today. We discuss these proof types below:

Types of zero-knowledge proofs

ZK-SNARKs

ZK-SNARK is an acronym for **Zero-Knowledge Succinct Non-Interactive Argument of Knowledge**. The ZK-SNARK protocol has the following qualities:

- **Zero-knowledge**: A verifier can validate the integrity of a statement without knowing anything else about the statement. The only knowledge the verifier has of the statement is whether it is true or false.
- **Succinct**: The zero-knowledge proof is smaller than the witness and can be verified quickly.
- **Non-interactive**: The proof is ‘non-interactive’ because the prover and verifier only interact once, unlike interactive proofs that require multiple rounds of communication.

- **Argument**: The proof satisfies the ‘soundness’ requirement, so cheating is extremely unlikely.
- **(Of) Knowledge**: The zero-knowledge proof cannot be constructed without access to the secret information (witness). It is difficult, if not impossible, for a prover who doesn’t have the witness to compute a valid zero-knowledge proof.

The ‘shared key’ mentioned earlier refers to public parameters that the prover and verifier agree to use in generating and verifying proofs. Generating the public parameters (collectively known as the Common Reference String (CRS)) is a sensitive operation because of its importance in the protocol’s security. If the entropy (randomness) used in generating the CRS gets into the hands of a dishonest prover, they can compute false proofs.

Multi-party computation (MPC) is a way of reducing the risks in generating public parameters. Multiple parties participate in a trusted setup ceremony, where each person contributes some random values to generate the CRS. As long as one honest party destroys their portion of the entropy, the ZK-SNARK protocol retains computational soundness.

Trusted setups require users to trust the participants in parameter-generation. However, the development of ZK-STARKs has enabled proving protocols that work with a non-trusted setup.

ZK-STARKs

ZK-STARK is an acronym for **Zero-Knowledge Scalable Transparent Argument of Knowledge**. ZK-STARKs are similar to ZK-SNARKs, except that they are:

- **Scalable**: ZK-STARK is faster than ZK-SNARK at generating and verifying proofs when the size of the witness is larger. With STARK proofs, prover and verification times only slightly increase as the witness grows (SNARK prover and verifier times increase linearly with witness size).
- **Transparent**: ZK-STARK relies on publicly verifiable randomness to generate public parameters for proving and verification instead of a trusted setup. Thus, they are more transparent compared to ZK-SNARKs.

ZK-STARKs produce larger proofs than ZK-SNARKs meaning they generally have higher verification overheads. However, there are cases (such as proving large datasets) where ZK-STARKs may be more cost-effective than ZK-SNARKs.

Use-cases for zero-knowledge proofs

- Anonymous payments
- Identity protection
- Authentication
- Verifiable computation
- Reducing bribery and collusion in on-chain voting

Anonymous payments

Credit card payments are often visible to multiple parties, including the payments provider, banks, and other interested parties (e.g., government authorities). While financial surveillance has benefits for identifying illegal activity, it also undermines the privacy of ordinary citizens.

Cryptocurrencies were intended to provide a means for users to conduct private, peer-to-peer transactions. But most cryptocurrency transactions are openly visible on public blockchains. User identities are often pseudonymous and either wilfully linked to real-world identities (e.g. by including ETH addresses on Twitter or GitHub profiles) or can be associated with real-world identities using basic on and off-chain data analysis.

There are specific “privacy coins” designed for completely anonymous transactions. Privacy-focused blockchains, such as Zcash and Monero, shield transaction details, including sender/receiver addresses, asset type, quantity, and the transaction timeline.

By baking in zero-knowledge technology into the protocol, privacy-focused blockchain networks allow nodes to validate transactions without needing to access transaction data.

Zero-knowledge proofs are also being applied to anonymizing transactions on public blockchains. An example is Tornado Cash, a decentralized, non-custodial service that allows users to conduct private transactions on Ethereum. Tornado Cash uses zero-knowledge proofs to obfuscate transaction details and guarantee financial privacy. Unfortunately, because these are "opt-in" privacy tools they are associated with illicit activity. To overcome this, privacy has to eventually become the default on public blockchains.

Identity protection

Current identity management systems put personal information at risk. Zero-knowledge proofs can help individuals validate identity whilst protecting sensitive details.

Zero-knowledge proofs are particularly useful in the context of decentralized identity. Decentralized identity (also described as ‘self-sovereign identity’) gives the individual the ability to control access to personal identifiers. Proving your citizenship without revealing your tax ID or passport details is a good example of how zero-knowledge technology enables decentralized identity.

Authentication

Using online services requires proving your identity and right to access those platforms. This often requires providing personal information, like names, email addresses, birth dates, and so on. You may also need to memorize long passwords or risk losing access.

Zero-knowledge proofs, however, can simplify authentication for both platforms and users. Once a ZK-proof has been generated using public inputs (e.g., data attesting to the user's membership of the platform) and private inputs (e.g., the user's details), the user can simply present it to authenticate their identity when they need to access the service. This improves the experience for users and frees organizations from the need to store huge amounts of user information.

Verifiable computation

Verifiable computation is another application of zero-knowledge technology for improving blockchain designs. Verifiable computing allows us to outsource computation to another entity while maintaining verifiable results. The entity submits the result along with a proof verifying that the program was executed correctly.

Verifiable computation is critical to improving processing speeds on blockchains without reducing security. Understanding this requires knowing the differences in proposed solutions for scaling Ethereum.

On-chain scaling solutions, such as sharding, require extensive modification of the blockchain's base layer. However, this approach is highly complex and errors in implementation can undermine Ethereum's security model.

Off-chain scaling solutions don't require redesigning the core Ethereum protocol. Instead they rely on an outsourced computation model to improve throughput on Ethereum's base layer.

Here’s how that works in practice:

- Instead of processing every transaction, Ethereum offloads execution to a separate chain.
- After processing transactions, the other chain returns the results to be applied to Ethereum’s state.

The benefit here is that Ethereum doesn’t have to do any execution and only needs to apply results from outsourced computation to its state. This reduces network congestion and also improves transaction speeds (off-chain protocols optimize for faster execution).

The chain needs a way to validate off-chain transactions without re-executing them, or else the value of off-chain execution is lost.

This is where verifiable computation comes into play. When a node executes a transaction outside of Ethereum, it submits a zero-knowledge proof to prove the correctness of off-chain execution. This proof (called a validity proof) guarantees that a transaction is valid, allowing Ethereum to apply the result to its state—without waiting for anyone to dispute it.

Zero-knowledge rollups and validiums are two off-chain scaling solutions that use validity proofs to provide secure scalability. These protocols execute thousands of transactions off-chain and submit proofs for verification on Ethereum. Those results can be applied immediately once the proof is verified, allowing Ethereum to process more transactions without increasing computation on the base layer.

Reducing bribery and collusion in on-chain voting

Blockchain voting schemes have many favorable characteristics: they are fully auditable, secure against attacks, resistant to censorship, and free of geographical constraints. But even on-chain voting schemes aren't immune to the problem of **collusion**.

Defined as “coordinating to limit open competition by deceiving, defrauding, and misleading others,” collusion may take the form of a malicious actor influencing voting by offering bribes. For example, Alice might receive a bribe from Bob to vote for `option B` on a ballot even if she prefers `option A`.

Bribery and collusion limit the effectiveness of any process that uses voting as a signaling mechanism (especially where users can prove how they voted). This can have significant consequences, especially where the votes are responsible for allocating scarce resources.

For example, quadratic funding mechanisms rely on donations to measure preference for certain options among different public good projects. Each donation counts as a "vote" for a specific project, with projects that receive more votes getting more funds from the matching pool.

Using on-chain voting makes quadratic funding susceptible to collusion: blockchain transactions are public, so bribers can inspect a bribee's on-chain activity to see how they "voted". This way quadratic funding ceases to be an effective means for allocating funds based on the aggregated preferences of the community.

Fortunately, newer solutions such as MACI (Minimum Anti-Collusion Infrastructure) are using zero-knowledge proofs to make on-chain voting (eg., quadratic funding mechanisms) resistant to bribery and collusion. MACI is a set of smart contracts and scripts that allow a central administrator (called a "coordinator") to aggregate votes and tally results *without* revealing specifics on how each individual voted. Even so, it is still possible to verify that the votes were counted properly, or confirm that a particular individual participated in the voting round.

Drawbacks of using zero-knowledge proofs

- Hardware costs
- Proof verification costs
- Trust assumptions
- Quantum computing threats

Hardware costs

Generating zero-knowledge proofs involves very complex calculations best performed on specialized machines. As these machines are expensive, they are often out of the reach of regular individuals. Additionally, applications that want to use zero-knowledge technology must factor in hardware costs—which may increase costs for end-users.

Proof verification costs

Verifying proofs also requires complex computation and increases the costs of implementing zero-knowledge technology in applications. This cost is particularly relevant in the context of proving computation. For example, ZK-rollups pay ~ 500,000 gas to verify a single ZK-SNARK proof on Ethereum, with ZK-STARKs requiring even higher fees.

Trust assumptions

In ZK-SNARK, the Common Reference String (public parameters) is generated once and available for re-use to parties who wish to participate in the zero-knowledge protocol. Public parameters are created via a trusted setup ceremony, where participants are assumed to be honest.

But there's really no way for users to assess the honesty of participants and users have to take developers at their word. ZK-STARKs are free from trust assumptions since the randomness used in generating the string is publicly verifiable. In the meantime, researchers are working on non-trusted setups for ZK-SNARKs to increase the security of proving mechanisms.

Quantum computing threats

ZK-SNARK uses elliptic curve cryptography (ECDSA) for encryption. While the ECDSA algorithm is secure for now, the development of quantum computers could break its security model in the future.

ZK-STARK is considered immune to the threat of quantum computing, as it uses collision-resistant hashes for encryption. Unlike public-private key pairings used in elliptic curve cryptography, collision-resistant hashing is more difficult for quantum computing algorithms to break.

Further reading {#further-reading}

- Computer Scientist Explains One Concept in 5 Levels of Difficulty | WIRED - *Wired YouTube channel*
- Overview of use cases for zero-knowledge proofs — *Privacy and Scaling Explorations Team*
- SNARKs vs. STARKS vs. Recursive SNARKs — *Alchemy Overviews*
- A Zero-Knowledge Proof: Improving Privacy on a Blockchain — *Dmitry Lavrenov*
- zk-SNARKs — A Realistic Zero-Knowledge Example and Deep Dive — *Adam Luciano*

- ZK-STARKs — Create Verifiable Trust, even against Quantum Computers — *Adam Luciano*
- An approximate introduction to how zk-SNARKs are possible — *Vitalik Buterin*
- What is Zero-Knowledge Proof and Its Role in Blockchain? — *LeewayHertz*