The Seven-Layer Magic Trick: A Guide to How Zero-Knowledge Proofs Work

Introduction: The Promise of Provable Secrets

Any sufficiently advanced technology is indistinguishable from magic. This observation, famously made by science fiction writer Arthur C. Clarke, perfectly captures the essence of one of today's most transformative cryptographic tools: the Zero-Knowledge Proof (ZKP). At its heart, a ZKP is a method that allows one person to prove to another that they know a secret, without revealing the secret itself.[1] This capability seems paradoxical, akin to a kind of digital magic.

To demystify this technology, it is helpful to think of a ZKP not as a complex mathematical formula, but as an elaborately staged magic trick.[3] Imagine a magician—let's call her the "Prover"—who wants to convince an audience—the "Verifier"—that she knows the secret method to a complex card trick. She could, of course, just tell them the secret. But that would ruin the magic and devalue her unique knowledge. Instead, she performs the trick. The performance itself is the proof. If successful, the audience is convinced she knows the secret, yet they learn nothing about her method. This performance has two key properties that mirror those of a ZKP: it is

zero-knowledge because the secret method remains hidden, and it is *succinct* because it is far faster for the audience to watch the trick than it would be for them to learn and master the secret method themselves.[3]

This "magic" is not just for entertainment; it is a foundational technology that solves critical challenges in the digital world. It enables private financial transactions where the amounts and participants can be hidden, as pioneered by Zcash.[5] It powers a new generation of blockchain scaling solutions called ZK-Rollups, which make networks like Ethereum faster and cheaper to use.[7] And it is paving the way for secure digital identity systems where you could, for example, prove you are over 18 without ever revealing your date of birth.[10]

But this magic trick is not arbitrary. It is a highly engineered process, constructed in a series of distinct stages. To truly understand how it works, we must deconstruct the performance layer by layer. Much like the Open Systems Interconnection (OSI) model provides a framework for understanding computer networks, a seven-layer conceptual stack can be used to dissect a modern ZKP system.[13] This report will guide you through that stack, from the initial design of the trick's rules to the final, public judgment of the performance. Each layer will be explained with a consistent analogy tied to our magic show, revealing how these components work together to create a seamless and secure illusion.

To provide a clear roadmap for this journey, the following table gives a high-level overview of the entire system. It acts as a mental scaffold, summarizing what each layer does and how it fits into the grand performance.

Layer #	Layer Name	Analogy: The Magic Show Component	Core Question it Answers
1	Setup	Designing the Rules & Game Board	What are the fundamental rules of this proof?
2	Language	Writing the Script for the Secret Action	How do we tell the magician exactly what to do?
3	Witness Generation	Performing the Secret Action Backstage	How does the magician perform the secret and record their steps?
4	Arithmetization	Creating the Unsolvable Puzzle	How do we turn the record of the action into a math problem?
5	Proof System	The Unforgeable Photograph	How do we create a tiny, verifiable snapshot of the solution?

6	Cryptographic Primitives	The Laws of Physics & Magic Materials	What unbreakable principles is this all built on?
7	Verifier Deployment	The Public Judge & The Award Ceremony	How is the proof finally judged and accepted by everyone?

With this map in hand, let us begin by examining the very foundation upon which our magic trick is built: the rulebook itself.

Part I: The Blueprint and the Performance

Layer 1: Setup - Forging the Rules of the Game

The lifecycle of a zero-knowledge proof begins not with a computation or a secret, but with the establishment of a root of trust.[13] Before any proof can be generated or verified, both the Prover and the Verifier must agree on a common set of public parameters that define the system's security and operational rules. This foundational layer is where the cryptographic social contract of the system is forged.

The most effective analogy for this layer is **an impartial committee designing the official rulebook and specialized game board for a global tournament**.[13] Before any match can be played, all competitors and judges must agree on the exact rules, the dimensions of the field, and the properties of the equipment. In the world of ZKPs, this "game board" is a set of cryptographic numbers called a Structured Reference String (SRS) or Common Reference String (CRS).[13] The method used to create this rulebook and game board is a defining architectural choice that splits the ZKP world into two distinct philosophies.

The Two Kinds of Rulebooks

The landscape of setup ceremonies is defined by a fundamental dichotomy: the presence or absence of secrets in the creation process.[13]

A **Trusted Setup**, common to highly efficient systems like zk-SNARKs, is analogous to a rulebook designed in secret by the tournament committee. This secret design process allows for the creation of an incredibly elegant and efficient game board, one that enables very fast and simple games—in ZKP terms, this translates to remarkably small proofs and extremely fast verification times.[13] However, this performance comes with a critical condition. The committee had to use a secret ingredient to craft the board, and they solemnly swear they destroyed the recipe afterward. This secret recipe is colloquially known as "toxic waste".[13] If the committee lied and kept a copy of the recipe, they could create a counterfeit game board that allows them to cheat in any game, and no one would be the wiser. They could generate fraudulent proofs for false statements that would appear perfectly valid, catastrophically breaking the system's integrity.[13] Therefore, using this type of system requires trusting that the "toxic waste" was properly handled and destroyed during the setup phase.

In stark contrast, a **Transparent Setup**, the hallmark of systems like zk-STARKs, is like a rulebook designed in a massive, open stadium with full public oversight.[13] The process is "transparent" because it uses only publicly verifiable components, analogous to standard, fair dice rolls instead of a secret recipe. In cryptographic terms, this means relying on the properties of well-understood hash functions.[13] By eliminating the concept of "toxic waste," the system's integrity rests solely on open, auditable mathematics. This approach offers superior security guarantees and is even plausibly resistant to the "super-cheating" methods of future quantum computers.[13] The trade-off for this robustness is a clunkier and slower game; STARK proofs are substantially larger, and their verification is more computationally intensive.[13]

Making the Secret Committee Trustworthy

Given the significant risk posed by "toxic waste," modern trusted setups employ a powerful mitigation strategy: they decentralize the trust. Instead of a small, secret committee, they organize a **Multi-Party Computation (MPC) ceremony**.[13] The analogy here is that the rulebook is created not by a single entity, but by thousands of independent people from all over the world, participating in a collaborative protocol. Each participant contributes their own small, secret ingredient to the master recipe and then securely destroys their personal contribution.

The security of this process relies on what is known as the "1-of-N" trust model. For the final, composite recipe to be compromised, *every single one* of the N participants would have to collude and share their secret ingredient. As long as just one participant acts honestly and destroys their contribution, the final game board is secure and cannot be used to cheat.[13] This elegant mechanism transforms an absolute trust requirement in a single entity into a much weaker and more plausible assumption about the lack of total collusion among a large and diverse group.

The evolution of these ceremonies illustrates a growing understanding of this social dimension of security. Zcash, a pioneer in this field, conducted its first "Sprout" ceremony in 2016 with just six expert participants.[13] For its "Sapling" upgrade in 2018, it conducted a much larger, two-phase ceremony that involved over 170 contributions from a more diverse group.[13] This culminated in the 2023 Ethereum KZG "Summoning" ceremony for the EIP-4844 upgrade, which was designed for mass participation. Through a user-friendly browser interface, it attracted an unprecedented 141,416 contributions.[13] This event transformed the trusted setup from a cryptographic necessity into a powerful social ritual. Participants contributed randomness from creative sources, including radioactive decay and even "the dynamics of a dog's dinner dance," reinforcing the social consensus and providing assurance against even esoteric systemic failures.[13] The security guarantee of "at least one of 141,416 global participants was honest" is vastly stronger and more credible than relying on the honesty of a handful of experts.

Building a Universal Game Board

A major drawback of early trusted setups was their specificity. The rulebook created for one game, like Chess, was useless for any other game, like Checkers. In ZKP terms, the setup was "circuit-specific," meaning any change to the application's logic required a whole new, resource-intensive MPC ceremony.[13]

Subsequent innovations introduced the concept of a **Universal** SRS. Systems like PLONK and Marlin use a setup that is not tied to any single program.[13] This is analogous to creating a master game board that can be quickly adapted for any game up to a certain complexity. This amortizes the high social and financial cost of the MPC ceremony across an unlimited number of future applications, turning it from a recurring project-specific burden into a one-time public good.[13] Some systems even feature an

Updatable SRS, which allows anyone, at any time, to contribute fresh randomness, perpetually strengthening the security of the original setup.[13]

The choice between these setup models is not merely technical; it is a profound socio-economic decision. It reveals a trade-off between a high upfront capital expenditure (Capex) and ongoing operational expenditure (Opex). A trusted setup, with its complex and costly MPC ceremony, represents a massive one-time Capex.[13] The return on this investment is extremely low Opex; the resulting zk-SNARKs have tiny proofs and fast, cheap on-chain verification, which translates directly to low, predictable transaction fees for users.[13] Conversely, a transparent setup like a STARK has zero Capex, as no ceremony is needed.[13] The cost is instead paid continuously as higher Opex. STARK proofs are significantly larger and have more computationally intensive verification, consuming more network bandwidth and on-chain resources with every proof.[13] This economic lens explains the strategic divergence in the blockchain ecosystem. Ethereum, with its extremely high on-chain transaction costs, made a rational high-Capex investment in the KZG ceremony to drastically lower the Opex for its entire Layer 2 ecosystem via EIP-4844.[13] In contrast, a system like Starknet, which prioritizes massive off-chain computation and long-term, post-quantum security, opts for the zero-Capex, higher-Opex STARK model.[13]

This layer is necessary because without the agreed-upon Rulebook and Game Board, there is no shared context for the proof. The magician and the audience would be playing by different rules, and the performance would be meaningless. Layer 1 establishes the fundamental reality upon which all subsequent cryptographic claims are built.

Layer 2: The Language - Writing the Play-by-Play Script

Once the fundamental rules of the game are established in Layer 1, the next step is to define the specific action that needs to be proven. Layer 2 serves as the interface between a human developer's intent and the formal, mathematical world of the proving system. Its purpose is to translate a high-level program into a precise and unambiguous set of instructions.

The analogy for this layer is a **playwright translating a story concept into a detailed script**, complete with specific dialogue, actions, and stage directions for the actors to follow.[13] The developer is the playwright, and the Zero-Knowledge programming language is the structured format of the script.

From Vague Idea to Specific Instructions

A developer might start with a general idea, such as, "I want to prove I have sufficient funds in my bank account to make a purchase, without revealing my total balance." This is the story concept. A ZK language, such as Circom, Noir, or Cairo, is the tool used to write this concept down in a formal way that a computer can interpret and ultimately prove.[7]

The script must make a clear distinction between two types of information. First are the **private inputs**, also known as the "witness." These are the secret values known only to the prover. In our analogy, this is the actor's secret motivation or backstory—for example, the *exact* balance in the bank account.[13] The zero-knowledge property of the final proof guarantees these private inputs are never revealed. Second are the

public inputs, or the "instance." These are values known to both the prover and the verifier that parameterize the computation. In the analogy, this is the public setting of the play—for example, the *price* of the item being purchased.[13] The script, or ZK program, defines the relationship between these inputs that must hold true:

private_balance >= public_price.

Two Kinds of Theaters

Just as plays can be written for different kinds of stages, ZK programs can be developed for different computational environments. This choice represents a key trade-off between performance and developer convenience.

One approach is to use **Direct-to-Circuit Languages**. These are like writing a script for a unique, custom-built stage designed to be perfectly optimized for one specific play.[13] Languages like Circom and Noir fall into this category. This method generally produces the most efficient and performant proofs (in our analogy, the tightest and most compelling performance) because every element is tailored to the specific logic being proven. However, it often requires the playwright to also be a skilled stage-builder, demanding specialized knowledge of how to construct cryptographic circuits efficiently.[12]

The other approach is to use a **Zero-Knowledge Virtual Machine (zkVM)**. This is like writing a script for a standard, pre-existing theater stage, such as a Broadway theater that hosts many different shows.[13] The "stage" is a circuit that emulates a generic computer processor, with a standard Instruction Set Architecture (ISA) like RISC-V or the Ethereum Virtual Machine (EVM). This paradigm allows developers to be just playwrights, using familiar, general-purpose programming languages like Rust, C++, or Go without needing to worry about the complex details of building the stage itself.[8] This dramatically lowers the barrier to entry, but it comes with a performance cost often called the "abstraction tax".[13] The final performance must not only prove that the play was performed correctly but also that the standard stage itself was constructed and operated according to its rules, adding overhead to every proof.

The Compiler's Role: The Director

Between the playwright's script and the actors' performance is the director. In the ZKP stack, this role is played by the **compiler**. The compiler takes the high-level code written by the developer and "lowers" it into an even more precise, formal, and unambiguous set of instructions for the cryptographic machinery. This intermediate format is known as an **Intermediate Representation (IR)**.[13]

The design of the IR is critical. A well-designed IR can enable powerful optimizations and, importantly, modularity. For example, Noir's Abstract Circuit Intermediate Representation (ACIR) is designed to be "backend-agnostic".[13] It separates standard arithmetic from complex "black box functions" like a SHA256 hash. This is like a director's script notation that is universal. It allows the same play to be staged in different theaters (i.e., run on different proof systems like PLONK or Groth16) simply by providing a new set of stagehands (a new backend) that knows how to interpret the universal notation.[13] This decoupling of the application logic (the play) from the underlying cryptography (the theater) is a sign of the ZKP ecosystem's maturation, mirroring the evolution of traditional software development with tools like LLVM IR.[11]

The choice of programming language and compiler architecture is governed by a fundamental trilemma, forcing a trade-off between **Performance** (the efficiency and minimality of the final proof), **Developer Experience** (the ease of writing, debugging, and maintaining the code), and **Security** (the safety from common vulnerabilities).[13] It is exceptionally difficult to maximize all three at once. Low-level languages like Circom give developers direct control over the cryptographic "constraints," allowing for hand-optimization that maximizes performance. However, this comes at a steep cost to developer experience and introduces significant security risks, as manual errors can easily lead to critical bugs like "under-constrained" variables, where a malicious prover can cheat the system.[13] At the other end of the spectrum, zkVMs maximize developer experience by allowing the use of familiar languages and tools, but this convenience comes with the "abstraction tax" of lower performance.[8] Modern languages like Noir attempt to strike a balance, offering a safer, high-level syntax that abstracts away the most dangerous parts of circuit writing while still compiling to an efficient representation.[13]

This layer is necessary because without the Script, the magician (Prover) has only a vague idea of what to do. Layer 2 provides the precise, unambiguous instructions needed to ensure the secret action performed in the next layer corresponds exactly to the statement being proven. It is the essential bridge between abstract human intent and concrete mathematical rigor.

Layer 3: Witness Generation - Performing the Secret Action

With a precise script in hand from Layer 2, the next step is the actual performance. Layer 3 is the execution engine of the ZKP system. It is responsible for taking the program's instructions and the prover's secret inputs and running the computation to produce a complete record of every value. This record is the "witness."

The analogy for this layer is the **magician performing the secret steps of the trick backstage, while meticulously documenting every single move, calculation, and prop used in a detailed logbook.** This logbook is the witness, and it serves as the foundational data artifact for the rest of the proof process.[13]

The Performance Log

The witness is not merely the final result of the computation; it is the entire execution trace, containing the complete assignment of values to all variables for that specific run.[13] If the script from Layer 2 specified the calculation

$e = (a * b) + d$, and the secret inputs were $a=3$, $b=4$, and $d=5$, the witness would not just be the final output $e=17$. It would be a complete log: $a=3$, $b=4$, $d=5$, the intermediate value $c = a * b = 12$, and the final value $e=17$. This process is akin to a CPU running a program and logging the state of every register at every clock cycle.[13]

The structure of this logbook is dictated by the format required by the puzzle-maker in the next layer (Arithmetization). For an R1CS-based system, the logbook is a simple, one-dimensional list of all these values.[13] For more modern AIR or PLONKish systems, the logbook is a two-dimensional table, where columns represent different variables (or "registers") and rows represent their state at each consecutive step of the computation.[13] This tabular layout is highly efficient for proving iterative computations like a virtual machine, as it keeps related data physically close in memory, which is beneficial for performance.

For a very long and complex magic trick, the magician's logbook could become enormous, potentially requiring gigabytes of memory to store all at once. This has led to different implementation models for the witness generator. The naive approach, the **Full-Trace Model**, is to perform the entire trick from start to finish and write down the complete logbook before handing it off. This is simple but suffers from prohibitive memory consumption for large computations.[13] A more advanced approach is the

Streaming Model. Here, the magician performs one scene, writes down that part of the log, and immediately passes that page to the puzzle-maker (Layer 4) for processing. Once processed, the page can be discarded, freeing up memory for the next scene. This drastically reduces the peak memory required, enabling proofs of much larger computations.[13]

The Hidden Security Threat of Side Channels

While most of the ZKP stack is concerned with mathematical correctness (soundness), Layer 3 is the primary place where the "zero-knowledge" privacy guarantee can be completely and irrevocably broken. The witness generator is the only component in the entire stack that directly handles the raw, unencrypted secret inputs.[13] Therefore, even if the final cryptographic proof is perfect, information about the secret can be leaked by the physical act of the performance itself.

This is known as a **side-channel attack**.[13] To extend our analogy, imagine an adversary is not in the audience but is an eavesdropper listening outside the magician's backstage dressing room. The adversary cannot see the trick, but they can observe its physical side effects:

- **Timing Attacks:** If the magician takes noticeably longer to perform a calculation with a large secret number compared to a small one, the eavesdropper can infer the magnitude of the secret simply by using a stopwatch. This exact vulnerability was demonstrated in an early version of the Zcash protocol, where the time to generate a proof was correlated with the transaction amount.[13]
- **Branching and Memory Access Attacks:** If the script contains a conditional branch like, "if the secret word is 'abracadabra', shout loudly; otherwise, whisper," the eavesdropper can learn the secret just by listening to the volume of the performance. Similarly, if the secret is used to decide which of two drawers to open, the sound of the specific drawer opening can leak the secret.[13]

The fundamental countermeasure against these attacks is to ensure the performance is **data-oblivious**, meaning its observable physical characteristics are independent of the secret inputs. This is achieved through a discipline called **constant-time programming.**[13] The goal is to eliminate any data-dependent branches or memory access patterns. For example, instead of a conditional "if-else" statement, the script is rewritten using arithmetic. The conditional

if (s==1) {r=a;} else {r=b;} is replaced by the algebraic expression $r = (s \cdot a) + ((1-s) \cdot b)$. This forces the magician to always perform both calculations, blending the results in a way that reveals the correct answer without leaking which path was logically taken. This makes the performance's timing and sound constant, regardless of the secret s, closing the side channel. This elevates witness generation from a simple execution step to a critical security boundary. A leaky witness generator can invalidate the entire system's privacy guarantee, no matter how strong the downstream cryptography is.

This layer is necessary because the proof must be about a *specific* performance of the script with *specific* secret inputs. Layer 3 provides that concrete data—the magician's detailed logbook. Without this witness, there is nothing to turn into a puzzle and, ultimately, nothing to prove.

Part II: The Transformation into a Verifiable Proof

Layer 4: Arithmetization - Creating the Verifiable Puzzle

With the magician's detailed performance logbook (the witness) from Layer 3, the system is ready to transform this raw data into a format that can be cryptographically proven. Layer 4, Arithmetization, is this crucial transformation step. It converts the statement "the computation was performed correctly according to the script" into a formal, algebraic language of polynomial equations.

The analogy for this layer is a **master puzzle-maker who takes the magician's detailed performance log and converts it into a single, complex, and verifiable Sudoku-like grid**.[13] The original claim—that the magician followed the script—is now equivalent to a new, purely mathematical claim: "this entire Sudoku grid is correctly filled according to its rules." This algebraic reframing is the critical bridge between the world of computation (Layer 3) and the world of abstract cryptography (Layer 5).

From Code to Equations

Arithmetization takes each instruction from the script and the corresponding values from the witness log and represents them as a mathematical equation that must be satisfied. A simple computational step like c = a * b is turned into a polynomial constraint a * b - c = 0.[13] The entire program, which could consist of millions of such steps, is converted into a massive system of these polynomial equations. The witness values are then used to fill in the variables of these equations. If the computation was performed correctly, every single equation in the system will evaluate to zero.

Different ZKP systems use different styles of puzzles, or constraint systems, to represent the computation:

- **R1CS (Rank-1 Constraint System):** This is one of the earliest and most fundamental formats, native to zk-SNARKs like Groth16.[13] It breaks the entire computation down into a set of very simple constraints, where each constraint can contain at most one multiplication. This is like a **jigsaw puzzle** made of thousands of tiny, simple pieces. While it can represent any computation, it can be inefficient for certain operations (like bitwise logic) that require a very large number of these simple pieces to express.[13]
- **AIR (Algebraic Intermediate Representation) and PLONKish Systems:** These more modern formats are designed to be much more efficient for iterative computations, such as the execution of a computer program. They organize the witness values into a two-dimensional table, or execution trace.[13] The constraints then define rules about the relationships between values in this table, such as "the value in this cell must be the sum of the two cells above it." This is analogous to a **Sudoku grid**, where the rules govern the relationships between adjacent cells, rows, and columns.[13] This tabular structure is the natural choice for zkVMs. PLONKish systems enhance this model further by allowing for "custom gates" (more complex, specialized rules for a single row) and "copy constraints" (rules that enforce that two distant cells in the grid must be equal, effectively wiring the circuit together).[13]

The Lookup Argument "Cheat Sheet"

Some computational operations are inherently awkward and expensive to express as a set of arithmetic rules. For example, proving that a number falls within a specific range (a range check) can require many individual constraints. To solve this, modern arithmetization schemes introduced a powerful optimization called a **lookup argument**.[13]

This is analogous to giving the puzzle-maker a pre-computed and officially verified **"cheat sheet"** for common but tricky patterns. Instead of laboriously writing out hundreds of Sudoku rules to enforce a complex pattern, the puzzle-maker can simply state, "the values in this section of the grid must match one of the valid rows in this official cheat sheet." The Plookup protocol was a major breakthrough that made this process efficient.[13] This dramatically reduces the number of constraints needed for many common computations, making proofs smaller and faster to generate.

The evolution of arithmetization from unstructured, gate-based systems like R1CS to highly structured, uniform, trace-based representations like AIR and PLONKish is not merely an optimization. This structural uniformity is the key enabler for the next generation of highly efficient proof systems based on folding schemes, which are discussed in the next layer. Folding schemes achieve their remarkable efficiency by requiring the computation to be broken down into a sequence of *identical* steps, each represented by a uniform circuit.[13] The non-uniform circuits produced by R1CS are a poor fit for this model. However, the trace-based approach of AIR, where a VM's execution consists of the repeated application of a single transition function, is a much better match.[13]

This trend reaches its logical conclusion with the "lookup singularity" paradigm, pioneered by systems like Jolt and Lasso.[6] This approach reframes

every possible instruction a computer can execute as a lookup into a single, massive, master cheat sheet. The puzzle-maker's job is no longer to create a puzzle about the magician's specific actions (add, multiply, etc.), but to create a puzzle that simply proves, "For step 1, the action corresponds to line 5,834 of the master cheat sheet. For step 2, it corresponds to line 1,211," and so on.

The profound consequence is that the problem being proven at every single step becomes **perfectly uniform**: it is always the circuit that verifies the lookup argument itself. This innovation in arithmetization at Layer 4 directly unlocked the potential for the massive performance gains of folding schemes at Layer 5. The demand for more efficient proof composition techniques created a need for more structured arithmetizations, and the development of these lookup-centric designs provided the ideal substrate for the new techniques to flourish. This co-evolution is a powerful pattern in the ZKP stack.

This layer is necessary because cryptography cannot work directly on computer code. It needs a mathematical representation of the problem. Arithmetization provides this by translating a statement about computational integrity into a formal algebraic puzzle. Without this Verifiable Puzzle, the cryptographic tools of the subsequent layers would have nothing to apply their "magic" to.

Layer 5: The Proof System - The Cryptographic Camera

At this stage, we have a massive, correctly filled-out mathematical puzzle (the arithmetization from Layer 4) that represents the entire computation. While this puzzle is verifiable, it is far too large to show to the audience directly. Doing so would be slow and would reveal the secret solution contained within it. The role of Layer 5, the Proof System, is to solve this problem. It is the cryptographic heart of the entire system, taking the giant puzzle and producing a tiny, unforgeable, and easily verifiable proof.

The best analogy for the Proof System is a **specialized cryptographic camera that takes a single, unforgeable photograph of the completed puzzle grid**.[13] This photograph has two magical properties: first, it is incredibly small (succinct), and second, it is magically blurred in such a way that it proves the puzzle is solved correctly without revealing any of the actual numbers written in the grid (zero-knowledge).

The Goal: A Tiny, Verifiable Snapshot

The primary job of the proof system is to compress a potentially gigabyte-sized puzzle into a proof that is often just a few hundred bytes.[13] This tiny snapshot can be transmitted quickly and checked by the verifier almost instantly. Different proof systems, or "cameras," offer different trade-offs:

- **Groth16:** One of the earliest and most efficient zk-SNARKs, akin to a camera that produces the smallest possible photograph (constant size) that can be verified with extreme speed. Its main drawback is that it requires a custom-built camera for each specific type of puzzle (it needs a circuit-specific trusted setup).[13]
- **PLONK:** A more flexible system that uses a universal camera. A single setup ceremony can create a camera that can photograph any puzzle up to a certain size. The trade-off is that the photographs are slightly larger and take a little longer to verify than Groth16's.[13]
- **zk-STARKs:** A completely different type of camera that doesn't require any secret components in its construction (it has a transparent setup). It is also resistant to future quantum-based methods of "photo-forgery." However, the photographs it produces are significantly larger (kilobytes instead of bytes), making them more expensive to publish and verify on a blockchain.[13]

Handling Epic Sagas: Proof Composition

What if the magic trick is not a short performance but an epic saga lasting for hours, resulting in an impractically enormous puzzle? Creating and photographing one single, gigantic puzzle is not feasible. To handle such large computations, modern systems use techniques for proof composition, which break the large problem into smaller pieces and then combine the proofs.

- **SNARK Recursion (A Photo of a Photo):** This is a general-purpose technique for combining proofs. It is like taking a photograph of the puzzle for Act 1 of the play. Then, for Act 2, the puzzle-maker creates a new puzzle that includes a small section dedicated to proving, "I possess a valid photograph of the solved puzzle from Act 1." A new photograph is then taken of this combined Act 2 puzzle. This process repeats, creating a chain of proofs. While it can be used to combine proofs from different types of puzzles (heterogeneous computation), it incurs a significant, constant overhead at each step, as proving the validity of a photograph is itself a complex task.[13]
- **Folding Schemes (Blending the Puzzles):** This is a more recent and highly efficient paradigm, particularly for uniform computations like a zkVM. Instead of taking a full photograph at each step, folding uses a special cryptographic **"blending machine"**.[13] After Act 1, the puzzle is put into the machine. After Act 2, its puzzle is added, and the machine magically blends them into a single, still-unsolved puzzle of the same size that represents the accumulated truth of both acts. This process is repeated for every act of the play. The blending process is extremely cheap, orders of magnitude faster than full recursion. Only at the very end of the entire saga is a single photograph taken of the final, blended puzzle. Nova was the first practical folding scheme, and SuperNova extended it to handle programs with multiple different instruction types.[3]

Feature	SNARK Recursion	Folding Schemes (e.g., Nova)
Core Mechanism	"Proof of a proof." The circuit for step i verifies the proof from step i-1.	"Instance of an instance." The instance for step i is folded into the accumulated instance from step i-1.
Per-Step Prover Cost	High: $O(C_{SNARK} + C_V)$,	Low: $O(C_{step}) + O(\log

	where C_V is the verifier circuit size.	C_{step})$
Per-Step Output	A full, self-contained ZK-SNARK proof.	An updated, unproven "running instance" or accumulator.
Final Proof Generation	Occurs at every step.	Occurs only once at the end of the entire computation.
Uniformity Requirement	Not strictly required; can verify proofs from different circuits.	Requires the folded instances to be for the same relation (e.g., R1CS).
Primary Use Case	Proof compression, verifying heterogeneous computations.	IVC for uniform computations (zkVMs), streaming proof generation.

Production systems often create hybrid architectures to get the best of both worlds. The "STARK-in-a-SNARK" pattern is a prime example.[13] This is analogous to using a fast but bulky Polaroid camera (a STARK) to take quick snapshots of each scene backstage. At the end of the play, a very high-end but slower camera (a SNARK) is used to take one, tiny, high-quality photograph of the entire collection of Polaroids. This gives the system a fast proving process (from the STARKs) while producing a tiny final artifact that is cheap to verify on-chain (the SNARK).[13]

The most advanced ZKP architectures elegantly resolve the tension between needing to prove general-purpose, non-uniform computations and the desire to use highly specialized, high-performance tools. They do so by following a powerful, recurring design pattern: **Decompose, Uniformize, Accumulate**. First, a complex, general problem (like a program running in a zkVM) is **decomposed** at Layers 2 and 4 into a series of primitive operations, such as lookups into a master table. This act of decomposition has the powerful side effect of **uniformizing** the problem; every step is now a "lookup proof," which has the same basic structure. Finally, now that the problem has been converted into a long stream of uniform steps, a highly specialized and efficient tool like a folding scheme at Layer 5 can be used to **accumulate** them cheaply.[13] This pattern masterfully combines the layers to transform a difficult, general problem into an easy, specialized one.

This layer is necessary because the puzzle from Layer 4 is still too big and revealing to be practical. The Cryptographic Camera of Layer 5 is what creates the final product that gives ZKPs their most famous properties: a *succinct* (tiny) and *zero-knowledge* (secret-preserving) proof. It is the engine that makes the magic trick both efficient and private.

Layer 6: Cryptographic Primitives - The Laws of Physics & Magic Materials

This is the most fundamental layer of the ZKP stack. It provides the basic, unbreakable cryptographic building blocks and assumptions upon which all the higher layers are constructed. If the proof system is a magical camera, this layer provides the inviolable principles that make the camera work.

The analogy for this layer is the combination of the **fundamental laws of physics (like gravity, which cannot be defied) and the special, magical materials (like un-copyable ink and one-way glass) that are used to build the puzzle grid and the camera itself**.[13] The security of the entire magic trick rests on the assumption that these physical laws are unbreakable and these materials work exactly as advertised.

The Source of Security: Hardness Assumptions

The security guarantees of all cryptographic systems are ultimately derived from the assumed intractability of certain mathematical problems. These are problems that are believed to be impossibly hard for even the world's most powerful computers to solve in any reasonable amount of time.[28] These "hard problems" are the laws of physics for our system.

- **Discrete Logarithm Problem (DLP):** This is a classical hardness assumption used in many zk-SNARKs. It is analogous to trying to figure out the exact number of times you spun a roulette wheel to land on a specific number, given only the starting position and the final position. It is easy to perform the action (spinning the wheel a set number of times), but computationally infeasible to reverse it and find the secret number of spins.[25] This "law of physics" is powerful but is known to be breakable by a sufficiently powerful quantum computer.[20]
- **Collision-Resistant Hash Functions (CRHFs):** This is the assumption that

underpins zk-STARKs. It is analogous to a perfect blender. It is easy to put ingredients in and turn them into a smoothie, but it is computationally impossible to find two different sets of ingredients that produce the exact same smoothie.[13] This property is believed to hold even against quantum computers, making it a "post-quantum" secure assumption.

The Magic Materials: Commitment Schemes

Commitment schemes are the workhorse cryptographic tools used throughout the ZKP process. They are the "magic materials" that allow the magician to perform the trick securely. The primary analogy for a commitment scheme is a **magic envelope**.[45] It has two key properties:

1. **Hiding:** You can put a message (a value) inside the envelope and seal it. Once sealed, no one can see what is inside.
2. **Binding:** Once the envelope is sealed, the message inside cannot be changed or altered.

The magician (Prover) uses these envelopes constantly to "commit" to values before being challenged by the audience (Verifier). This prevents the magician from changing their answers after the fact. Different proof systems use different kinds of magic envelopes, built from different physical laws:

- **KZG Commitments:** These are highly efficient magic envelopes that are built using pairing-based cryptography, which relies on the DLP assumption. They require the special "game board" from a trusted setup (Layer 1) to work. Their most magical property is that they are **additively homomorphic**, which means you can perform mathematical operations on the sealed envelopes themselves. For example, you can take two sealed envelopes, perform a special operation on them, and get a new sealed envelope that contains the sum of the two secret values inside, all without ever opening them.[13] This property is the essential enabler for efficient proof aggregation and folding schemes.
- **FRI (Fast Reed-Solomon IOP of Proximity):** This is the commitment scheme that underlies zk-STARKs. It is a different kind of magic envelope built from the "perfect blender" hash function instead of the DLP.[13] It does not require a trusted setup, but the envelopes are much bulkier, which contributes to the larger proof sizes of STARKs.

The choice of primitives at this foundational layer forces system architects into a design trilemma, requiring a trade-off between **Algebraic Functionality**, **Post-Quantum Resilience**, and **Performance/Succinctness**.[13] This is perhaps the most important trade-off in the entire stack, and it is the root cause of the major architectural divisions in the ZKP ecosystem, such as the split between SNARKs and STARKs.

DLP-based primitives like KZG offer rich **Algebraic Functionality**—specifically, the additive homomorphism that is the direct enabler for the most efficient composition schemes like folding at Layer 5. However, the very algebraic structure that provides this functionality is what makes them vulnerable to quantum attacks, giving them zero **Post-Quantum Resilience**. They also currently offer the best **Performance/Succinctness**, with KZG providing constant-size proofs.[13]

In contrast, hash-based primitives like FRI are built on unstructured functions, which gives them strong **Post-Quantum Resilience**. However, they lack these native algebraic properties, making it much more difficult and less efficient to build techniques like folding directly on top of them. Their performance is also generally worse, with larger proof sizes and higher verification costs.[13]

Lattice-based primitives, built on assumptions like Learning With Errors (LWE), represent a promising research frontier attempting to achieve the best of all worlds: Post-Quantum Resilience combined with Algebraic Functionality. However, these systems are currently less performant and introduce significant implementation complexity.[13]

This trilemma demonstrates that the most consequential architectural decisions are made at the very bottom of the stack. A system designer cannot simply choose to use a high-performance folding scheme at Layer 5; they must have first chosen a Layer 6 primitive that provides the necessary algebraic properties to make that scheme possible and efficient.

This layer is necessary because it provides the ultimate foundation of security and trust for the entire system. Without these assumed-to-be-unbreakable Laws of Physics and reliable Magic Materials, the puzzle could be faked, the camera could be tricked, and the entire magic show would be a house of cards, collapsing under the slightest scrutiny.

Part III: The Final Judgment and Synthesis

Layer 7: Verifier Deployment - The Public Judge & The Award Ceremony

The journey of our zero-knowledge proof culminates in this final layer, where the cryptographic artifact created by the Prover is finally checked, and the result is made public and binding. This layer moves the system from the abstract realm of cryptography into the world of practical engineering, dominated by factors like transaction costs, network latency, and system governance.

The analogy for Layer 7 is the **official tournament judge who receives the unforgeable photograph from the magician, checks it against the official rulebook, and then awards the prize in a public ceremony, stamping the result with their official, unimpeachable seal of approval for all to see.** In a blockchain-based system, this infallible judge is a smart contract.[13]

The On-Chain Judge

In the context of blockchain applications like ZK-Rollups, the proof generated by the off-chain Prover is submitted as part of a transaction to a **verifier smart contract** deployed on the main blockchain (e.g., Ethereum).[7] This smart contract acts as the ultimate, decentralized, and trustless judge. It contains the logic to perform the verification algorithm on the submitted proof. This verification process is designed to be extremely fast and computationally cheap. If the proof is valid, the smart contract accepts the result—for example, by updating the state of the ZK-Rollup—and records this acceptance immutably on the public ledger.[13] This on-chain verification provides the highest degree of transparency and decentralization, as anyone can independently confirm that the judge has validated the proof according to the public rules.

The Economics of Judgment

This public judgment is not free. On a blockchain like Ethereum, every computational step costs a fee, known as "gas." The gas cost of verifying a proof is a dominant engineering constraint and is directly influenced by the choices made in the lower layers.[13] A zk-SNARK proof, being very small, is typically very cheap to verify on-chain. A zk-STARK proof, being much larger, is inherently more expensive to verify.[13] This economic reality is a major driver of system design. To manage the high cost of STARK verification, for instance, systems like StarkNet's SHARP aggregate proofs from many different applications into a single, large STARK proof, thereby amortizing the high verification cost across a large volume of transactions.[13]

For ZK-Rollups, a significant portion of their L1 cost historically came not from the proof verification itself, but from the need to post transaction data to the main chain to ensure data availability. The EIP-4844 "Proto-Danksharding" upgrade on Ethereum fundamentally altered this economic equation.[3] It is analogous to the tournament organizers creating a new, much cheaper, and dedicated courier service (called "blobs") specifically for posting these large data packets. By shifting data from the expensive general-purpose mail (CALLDATA) to this specialized service, rollups have seen their largest operational cost dramatically reduced, leading to substantially lower fees for end-users.[13]

The Need for a Living Rulebook

In the real world, software has bugs, and better techniques are constantly being discovered. This means the judge's rulebook—the verifier smart contract—often needs to be upgradable to allow for bug fixes and performance improvements.[13] This is typically achieved using a "proxy pattern," where a stable, permanent smart contract address (the one users interact with) forwards all calls to a separate logic contract. This logic contract can be replaced by a new version via a governance decision, allowing the system to evolve without requiring users to change the address they interact with.[13]

This necessary feature of upgradability, however, introduces a profound new dimension to the system's security. The ultimate security of a mature, upgradable ZKP system is no longer just a mathematical problem; it becomes a socio-political one. The integrity of the flawless cryptography becomes inextricably linked to the robustness of the human-led governance mechanisms that control these upgrades.

An immutable verifier contract has a static attack surface defined by its code and the underlying cryptographic assumptions. But an upgradable verifier introduces a new, powerful vector of control. An attacker no longer needs to break the "laws of physics" at Layer 6 or find a flaw in the "magic camera" at Layer 5. Instead, they can attack the governance process itself—for example, by acquiring a majority of a project's governance tokens or by compromising the private keys of the multi-signature council that holds administrative power.[13] If an attacker gains control of governance, they can simply vote to replace the correct, honest judge with a fraudulent one that is programmed to accept invalid proofs. This would completely nullify the system's security guarantees, allowing the attacker to, for instance, steal all the funds from a ZK-Rollup.

This reality creates a deep symmetry between the first and last layers of the stack. The security of the system begins with a socio-technical contract established at Layer 1 (the broad participation and trust in the MPC ceremony) and is maintained by an ongoing political process at Layer 7 (the decentralized and resilient governance that secures the verifier). A complete security analysis of a modern ZKP system must therefore be holistic, evaluating not only the mathematical rigor of the proofs but also the socio-political robustness of the governance that secures the final judge.

This final layer is necessary because a proof is useless without a trusted party to verify it. Layer 7 provides the final, trustless arbiter that gives the proof its meaning and economic finality, allowing the system to be securely integrated into a decentralized environment like a blockchain. It is the moment the magic trick is officially validated for the world.

Synthesis: How the Layers Work in Concert & The Two Philosophies of Proof

Having dissected the ZKP system into its seven constituent layers, the true power of this model emerges when we synthesize these parts back into a whole. The layers are not independent components but a tightly coupled, co-evolving system where choices made at the deepest levels create a cascade of constraints and opportunities for the layers above. Understanding this intricate web of interdependencies is the essence of

ZKP systems engineering.[13]

The Cascade of Dependencies

A causal chain flows through the stack, demonstrating how foundational decisions propagate upwards:

- The journey begins at **Layer 6 (Primitives)**. The choice of a hardness assumption dictates the available tools for the entire system. Selecting a DLP-based, additively homomorphic commitment scheme like KZG is the fundamental prerequisite that makes the highly efficient **folding schemes** of **Layer 5 (Proof System)**, such as Nova, possible. This choice, in turn, creates a demand for the highly uniform, lookup-centric **arithmetizations** developed at **Layer 4**, like Jolt, as they provide the ideal, structured input that folding requires. In contrast, choosing a hash-based, non-homomorphic primitive like FRI for post-quantum security would render this entire high-performance architectural path infeasible.[13]
- The choice of **arithmetization** at **Layer 4** directly dictates the required structure of the **execution trace** at **Layer 3 (Witness Generation)**. An AIR or PLONKish arithmetization demands a tabular, 2D witness matrix. This structural requirement then drives the design of the compilers and **languages** at **Layer 2**, which must be capable of efficiently generating this specific trace layout from a developer's high-level code.[13]
- The choice of **Proof System** at **Layer 5** has direct economic consequences at **Layer 7 (Verifier Deployment)**. Opting for a zk-SNARK like Groth16 results in a tiny proof and a cheap on-chain verifier, minimizing ongoing operational costs. Choosing a zk-STARK results in a much larger proof and a more expensive verifier, but in exchange, it eliminates the need for a **trusted setup** (**Layer 1**) and provides **post-quantum security** (**Layer 6**). The hybrid "STARK-in-a-SNARK" architecture is a direct engineering response to this trade-off, attempting to capture the fast prover of STARKs with the cheap verifier of SNARKs.[13]
- Finally, there is a profound symmetry between **Layer 1** and **Layer 7**. The system's security begins with a socio-technical contract (the MPC ceremony) and is ultimately maintained by an ongoing socio-political process (the on-chain governance controlling verifier upgrades). Both layers underscore that the security of a decentralized system is a holistic concern, rooted in human coordination and trust as much as in code and cryptography.[13]

The Two Dominant Philosophies of Proof

This complex web of choices has led to the emergence of two dominant architectural philosophies in the ZKP ecosystem. While many variations exist, most production systems align with one of two major paths, each representing a different set of priorities and trade-offs. The following table crystallizes these two competing styles, providing a powerful framework for analyzing any real-world ZKP system.

Feature	Philosophy A: The SNARK Path (e.g., zkSync, Polygon zkEVM)	Philosophy B: The STARK Path (e.g., StarkNet)
Guiding Principle	Performance & Succinctness	Transparency & Long-Term Security
Layer 1: Setup	Trusted (MPC Ceremony)	Transparent (Public Randomness)
Layer 6: Core "Physics"	Pairing-based Crypto (e.g., DLP)	Hash-based Crypto (e.g., CRHFs)
Layer 5: Proof Size	Very Small ($O(1)$ or $O(\log N)$)	Large ($O(\log^2 N)$)
Layer 7: On-Chain Cost	Low	High (amortized by batching)
Key Advantage	Extreme efficiency for on-chain verification	No trust assumptions; Quantum-resistant
Key Trade-off	Requires trust in the setup ceremony; not quantum-safe	Larger proofs are more expensive in bandwidth/gas
Analogy Summary	An elegant, lightning-fast game on a special, secret-recipe board.	A slightly clunkier but robust game on a board built in the open for all to see.

Conclusion: The Future is Provable

This journey through the seven-layer stack reveals that a Zero-Knowledge Proof is far more than a single cryptographic trick. It is a complex, deeply integrated system where each layer serves a critical function, from establishing a social contract of trust to executing the final, economically-binding verification. The analysis has illuminated a series of fundamental trade-offs that define the ZKP design space. Architects must navigate the tension between trusted and transparent setups at Layer 1; the trilemma of performance, developer experience, and security at Layer 2; and the foundational choice between algebraic functionality, post-quantum resilience, and succinctness at Layer 6.[13]

The evolution of this stack is a story of increasing modularity and abstraction. The clear separation of layers, the development of common interfaces like Intermediate Representations, and the composition of different proof systems are all strategies to manage these complex trade-offs and combine the best properties of different approaches. The trajectory of the field points toward even greater abstraction with the maturation of zkVMs, a relentless pursuit of performance through hardware acceleration, and a long-term shift toward practical post-quantum solutions.[13]

Ultimately, the magic of a ZKP lies not just in the elegance of its mathematics, but in the holistic integrity of its entire architecture. A truly secure and robust system depends on every layer working in concert: the social diversity of its setup ceremony, the formal correctness of its compiler, the side-channel resistance of its witness generator, and the decentralized resilience of the on-chain governance that secures its verifier. As this technology becomes more deeply woven into the fabric of our digital world, this comprehensive, multi-disciplinary understanding will be the key to unlocking its full potential, making the future not just private and scalable, but provably so.

Works cited

1. Zero-knowledge proof - Wikipedia, accessed August 12, 2025, https://en.wikipedia.org/wiki/Zero-knowledge_proof
2. Zero Knowledge Proof (ZKPs) in Blockchain: A Beginner's Guide | Learn - KuCoin, accessed August 12, 2025, https://www.kucoin.com/learn/crypto/zero-knowledge-proof-zkp-explained
3. Secrets, and how to prove them: A magician's guide to zero-knowledge proofs, accessed August 12, 2025,

https://a16zcrypto.com/posts/article/a-magicians-guide-to-zero-knowledge-proofs/
4. a16z: How to Understand Zero-Knowledge Proofs Through a Magic Show - ChainCatcher, accessed August 12, 2025, https://www.chaincatcher.com/en/article/2101307
5. Applications of Zero Knowledge Proofs - 3 Real World Examples - Coin Bureau, accessed August 12, 2025, https://coinbureau.com/adoption/applications-zero-knowledge-proofs/
6. zk-SNARKs and zk-STARKs Explained - Binance Academy, accessed August 12, 2025, https://academy.binance.com/en/articles/zk-snarks-and-zk-starks-explained
7. ZK Rollups: What They Are and Why You Should Care - Changelly, accessed August 12, 2025, https://changelly.com/blog/what-are-zk-rollups/
8. ZK-Rollup Overview - GeeksforGeeks, accessed August 12, 2025, https://www.geeksforgeeks.org/computer-networks/zk-rollup-overview/
9. A Beginners Guide to ZK (Zero-Knowledge) Rollups | by Zeel Gala | Krypcore, accessed August 12, 2025, https://blog.krypcore.com/a-beginners-guide-to-zk-zero-knowledge-rollups-c1850dda4a8c
10. Top 10 Blockchain Zero-Knowledge Proof Use Cases in 2024 | Ultimate Guide, accessed August 12, 2025, https://www.rapidinnovation.io/post/top-10-blockchain-use-cases-of-zero-knowledge-proof
11. Zero Knowledge Proof Identity Management - IAM Concept, accessed August 12, 2025, https://identitymanagementinstitute.org/zero-knowledge-proof-identity-management/
12. Zero-Knowledge Proofs: A Beginner's Guide - Dock Labs, accessed August 12, 2025, https://www.dock.io/post/zero-knowledge-proofs
13. ZKP Seven-Layer Architecture Analysis_.pdf
14. Zinc: a hash-based SNARK without arithmetization overheads - Albert Garreta (Nethermind) - Room 1 - YouTube, accessed August 12, 2025, https://www.youtube.com/watch?v=6BYQUist2vE
15. Decoding ZK-SNARK VS STARK: An In-Depth Comparative Analysis - Calibraint, accessed August 12, 2025, https://www.calibraint.com/blog/zk-snark-vs-stark-differences-comparison
16. zk-SNARK vs zkSTARK - Explained Simple - Chainlink, accessed August 12, 2025, https://chain.link/education-hub/zk-snarks-vs-zk-starks
17. PlonK Deconstructed: 2 Arithmetization - Maya ZK Blog, accessed August 12, 2025, https://www.maya-zk.com/blog/arithmetization
18. zk-STARK vs zk-SNARK : An In-Depth Comparative Analysis - QuillAudits, accessed August 12, 2025, https://www.quillaudits.com/blog/ethereum/zk-starks-vs-zk-snarks
19. Arithmetization in STARKs an Intro to AIR - I - Three Sigma, accessed August 12, 2025,

https://threesigma.xyz/blog/zk/arithmetization-starks-algebraic-intermediate-representation
20. Full Guide to Understanding zk-SNARKs and zk-STARKS - Cyfrin, accessed August 12, 2025, https://www.cyfrin.io/blog/a-full-comparison-what-are-zk-snarks-and-zk-starks
21. Cryptographic Limitations on Learning Boolean Formulae and Finite Automata - CIS UPenn, accessed August 12, 2025, https://www.cis.upenn.edu/~mkearns/papers/cryptojacm.pdf
22. Lecture 11: Key Agreement 1 Hardness Assumptions, accessed August 12, 2025, https://www.cs.cmu.edu/~goyal/s18/15503/scribe_notes/lecture12.pdf
23. The discrete logarithm problem (video) - Khan Academy, accessed August 12, 2025, https://www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/discrete-logarithm-problem
24. Cryptographic Hardness Results for Learning Intersections of Halfspaces - UT Computer Science, accessed August 12, 2025, https://www.cs.utexas.edu/~klivans/hs-hardness.pdf
25. ELI5: Discrete logarithm : r/explainlikeimfive - Reddit, accessed August 12, 2025, https://www.reddit.com/r/explainlikeimfive/comments/11354p/eli5_discrete_logarithm/
26. Top 10 Zero Knowledge-Proof Applications to Know - Infisign, accessed August 12, 2025, https://www.infisign.ai/blog/zero-knowledge-proof-applications
27. Zero-Knowledge Proofs: How it Works & Use Cases in 2025 - AIMultiple, accessed August 12, 2025, https://aimultiple.com/zero-knowledge-proofs
28. Cryptographic Hardness Assumptions - Agnese Gini, accessed August 12, 2025, https://agnesegini.github.io/Sec1/CHA.html
29. Computational hardness assumption - Wikipedia, accessed August 12, 2025, https://en.wikipedia.org/wiki/Computational_hardness_assumption
30. Understanding Zero-Knowledge Proofs in Digital Identity Systems - Vidos, accessed August 12, 2025, https://vidos.id/blog/understanding-zero-knowledge-proofs-in-digital-identity-systems
31. zkSNARKs and zkSTARKs explained (Ultimate Guide) - AMINA Bank, accessed August 12, 2025, https://aminagroup.com/research/unveiling-cryptographic-enigmas-the-magic-of-zksnarks-and-zkstarks/
32. Hello Arithmetization — R1CS | by Emir Soytürk - Medium, accessed August 12, 2025, https://emirsoyturk.medium.com/hello-arithmetization-55e57c8e5471
33. ZK fundamentals: Intermediate representations | Smart contract audits from Veridise, accessed August 12, 2025, https://veridise.com/blog/learn-blockchain/zk-fundamentals-intermediate-representations/
34. Arithmetization schemes for ZK-SNARKs - LambdaClass Blog, accessed August 12, 2025, https://blog.lambdaclass.com/arithmetization-schemes-for-zk-snarks/
35. Top Zero-Knowledge (ZK) Proof Crypto Projects of 2025 | Learn - KuCoin,

accessed August 12, 2025, https://www.kucoin.com/learn/crypto/top-zero-knowledge-zk-proof-crypto-projects

36. The Zero Knowledge Frontier: On SNARKs, STARKs, and Future Applications - The Tie, accessed August 12, 2025, https://www.thetie.io/insights/research/zero-knowledge-starks-snarks/
37. How can I explain "zero knowledge proof" to an end user?, accessed August 12, 2025, https://security.stackexchange.com/questions/86823/how-can-i-explain-zero-knowledge-proof-to-an-end-user
38. Starknet Ecosystem | Discover the future of Ethereum scalability., accessed August 12, 2025, https://www.starknet-ecosystem.com/
39. Proof systems - The halo2 Book - Zcash, accessed August 12, 2025, https://zcash.github.io/halo2/concepts/proofs.html
40. Cryptography: ECDSA — Steemit, accessed August 12, 2025, https://steemit.com/@icostan/cryptography-ecdsa
41. How do you explain the details of something technical to a non-technical audience?, accessed August 12, 2025, https://writing.stackexchange.com/questions/33500/how-do-you-explain-the-details-of-something-technical-to-a-non-technical-audienc
42. ELI5 Blockchain Rollups: Optimistic and Zero-Knowledge - MyEtherWallet, accessed August 12, 2025, https://www.myetherwallet.com/blog/eli5-blockchain-rollups-optimistic-and-zero-knowledge/
43. (ELI5) ZK-Rollups - Medium, accessed August 12, 2025, https://medium.com/cryptolawschool/eli5-zk-rollups-4aebbe55a7fb
44. What Are Zero Knowledge Proofs or ZKPs? | Aleph Zero, accessed August 12, 2025, https://alephzero.org/blog/fundamentals-zero-knowledge-proofs/
45. ZKPDL: A Language-Based System for Efficient Zero-Knowledge Proofs and Electronic Cash - USENIX, accessed August 12, 2025, https://www.usenix.org/legacy/event/sec10/tech/full_papers/Meiklejohn.pdf
46. What is a discrete logarithm problem? How is useful for cryptography? - Quora, accessed August 12, 2025, https://www.quora.com/What-is-a-discrete-logarithm-problem-How-is-useful-for-cryptography
47. Zero knowledge proof explained - Medium, accessed August 12, 2025, https://medium.com/coinmonks/zero-knowledge-proof-explained-1595600ff1cf
48. Accelerating Zero-Knowledge Proofs Through Hardware-Algorithm Co-Design - People | MIT CSAIL, accessed August 12, 2025, https://people.csail.mit.edu/devadas/pubs/micro24_nocap.pdf
49. ELI5: Zero Knowledge Proof : r/explainlikeimfive - Reddit, accessed August 12, 2025, https://www.reddit.com/r/explainlikeimfive/comments/16nvzmd/eli5_zero_knowledge_proof/