A Survey on the Applications of Zero-Knowledge Proofs

Ryan Lavin, Xuekai Liu, Hardhik Mohanty, Logan Norman, Giovanni
Zaarour, Bhaskar Krishnamachari

Department of Electrical and Computer Engineering

Viterbi School of Engineering

University of Southern California

Los Angeles, CA, USA

{rlavin, xuekaili, hmohanty, fnorman, gzaarour, bkrishna}@usc.edu

Abstract

Zero-knowledge proofs (ZKPs) represent a revolutionary advance in computational integrity and privacy technology, enabling the secure and private exchange of information without revealing underlying private data. ZKPs have unique advantages in terms of universality and minimal security assumptions when compared to other privacy-sensitive computational methods for distributed systems, such as homomorphic encryption and secure multiparty computation. Their application spans multiple domains, from enhancing privacy in blockchain to facilitating confidential verification of computational tasks. This survey starts with a high-level overview of the technical workings of ZKPs with a focus on an increasingly relevant subset of ZKPs called zk-SNARKS. While there have been prior surveys on the algorithmic and theoretical aspects of ZKPs, our work is distinguished by providing a broader view of practical aspects and describing many recently-developed use cases of ZKPs across various domains. These application domains span blockchain privacy, scaling, storage, and interoperability,

as well as non-blockchain applications like voting, authentication, timelocks, and machine learning. Aimed at both practitioners and researchers, the survey also covers foundational components and infrastructure such as zero-knowledge virtual machines (zkVM), domain-specific languages (DSLs), supporting libraries, frameworks, and protocols. We conclude with a discussion on future directions, positioning ZKPs as pivotal in the advancement of cryptographic practices and digital privacy across many applications.

I. INTRODUCTION

Zero-knowledge proofs (ZKPs) are a set of cryptographic methods allowing one party to prove the validity of a claim to another without disclosing any of the claim's underlying details. The seminal work by Goldwasser, Micali, and Rackoff laid the groundwork for ZKPs in the 1980s [1]. It introduced the principle of knowledge complexity as a metric for quantifying the information transferred from the prover to the verifier. Subsequently, Goldreich and others [2], [3] published several works until the 1990s that expanded the scope of ZKPs to a broader range of computational problems. They demonstrated their applicability to NP-complete problems under specific cryptographic assumptions. Another significant advancement in the development of ZKP's came with the introduction of succinctness in [4], [5] to ZKP's following the design paradigm of Kilian's seminal 1992 paper [6]. The succinct and non-interactive properties of zk-SNARKs greatly enhance their practical applicability and versatility in blockchains and non-blockchain systems. The foundational principles of ZKPs offer innovative solutions to digital systems that prioritize both security and privacy [7].

The advent of ZKPs has represented a significant leap forward in the convergence of privacy and verifiability, which has captivated significant interest from both the cryptographic scholarly community and the industry. It introduced a unique approach in the field of cryptography, differentiating itself from other privacy-sensitive computational innovations for distributed systems, such as homomorphic encryption and

secure multiparty computation. While these methods are also being actively developed and advanced, each serves a specific purpose in information verification and privacy preservation. Homomorphic encryption enables computations on encrypted data without needing to decrypt, thereby preserving confidentiality while allowing the derivation of useful insights [8]. Secure multiparty computation enables trustless collaborations and allows parties to jointly compute a function over their inputs while keeping those inputs private [9]. From Table I, we observe ZKPs offer several advantages, such as universality and minimal security assumptions over other techniques by proving the truth of a statement without disclosing any other information.

In modern digital systems, a trade-off exists between openness and privacy. Blockchains, for example, prioritize transparency to ensure trust and prevent fraud, with every transaction openly verifiable. However, this transparency can compromise privacy [10]. Despite the pseudonymous nature of blockchain transactions, advanced analytics can de-anonymize users by correlating on-chain and off-chain data. Such exposure can reveal a user's entire transaction history, leading to privacy breaches and potential targeted threats. As digital infrastructures become increasingly complex, striking the right balance between transparency for security and preserving user privacy becomes a vital research challenge. In this context, ZKPs emerge as a robust solution addressing the challenges posed by such trade-offs in digital systems.

ZKPs enable users to verify private data such as bank balances or credit scores without revealing specifics. Furthermore, they can ensure anonymity in authorization processes by allowing access to restricted areas or proving regional origin without disclosing detailed credentials. In the financial domain, ZKPs can allow for identity-free payments and tax submissions without revealing exact earnings. Additionally, they facilitate trustless outsourcing by letting organizations validate results without redoing the entire operation in the computational field. Furthermore, ZKPs can modify blockchain operations, shifting from collective to singular computation with network-wide verification. Overall, ZKPs

Cryptographic Method	Universality	Security Assumptions	Computational Complexity
Zero-Knowledge Proofs	**High:** Versatile across various protocols. Supports a broad spectrum of functions from authentication to smart contracts.	**Minimal:** Relies on the complexity of creating a proof without exposing any of the underlying data. Assumes the security of cryptographic primitives.	**High:** The prover and verifier may perform significant computational work, which might include prover creating a polynomial time computation representing the original problem and generating a proof of this computation
Fully Homomorphic Encryption	**High:** Best suited for secure processing of data. Allows for computation on encrypted data, making it ideal for cloud computing environments.	**Scheme Dependent:** Security is contingent on the difficulty of algebraic challenges.	**Very High:** FHE operations are computationally intensive because they involve complex algebraic operations on encrypted data. The overhead associated with maintaining homomorphism over computations makes these operations costly.
Secure Multiparty Computation	**Medium:** Geared towards collaborative tasks in data analysis. Permits multiple parties to compute over data without revealing individual inputs.	**Varied:** Security often needs an honest majority within the group of participants, assuming computational hardness and sometimes the use of random oracles.	**High:** Depends on protocol complexity and the number of participants. Each participant in an SMC protocol must perform computations on their part of the data as well as on data received from others.

TABLE I: Comparison of Zero-Knowledge Proofs with other cryptographic methods

exemplify the merging of verification with privacy across diverse applications [11].

In exploring the realm of ZKPs, it is fundamental to recognize verifiable computation as the cornerstone upon which the practical applications of ZKPs are built. Verifiable computation, at its core, is the ability to prove that an external computation was performed correctly without revealing the inputs or the computation process. This foundational attribute of ZKPs serves as a gateway to their two practical value propositions: succinctness and privacy. In the remainder of this survey, we will see how each of these two values contributes to the practical applications of ZKPs.

Succinctness in ZKPs allows for the quick verification of the correctness of a computation without the extensive resources typically required for direct computation execution. This is crucial in contexts where computational resources are limited or expensive, such as in blockchain networks. For instance, ZKPs enable the validation of transaction blocks or smart contract executions without necessitating each node to replay the entire sequence of transactions or computations. This aspect dramatically reduces the computational load on the network, enhancing scalability and throughput.

Privacy, the second major value proposition, emerges from the intrinsic nature of ZKPs to prove the correctness of information without revealing the information itself. This characteristic is particularly transformative in scenarios where sensitive or confidential data is involved. This capability opens the door to a wide range of applications, from private voting systems to confidential financial transactions, where the integrity of the process is maintained without compromising the privacy of the individuals or entities involved.

In the ZKP literature, there exist two key survey articles relevant to our work. The first survey by Morais *et al.* [12] dives deep into Zero-Knowledge Range Proofs (ZKRP) with a particular emphasis on the algorithmic details required for the implementation of ZKRPs. Furthermore, it defines the fundamental theoretical background behind ZKRPs and describes possible generic and blockchain-specific use cases of ZKRPs. In contrast,

our survey covers a broader range of application categories and use cases. Further, we provide an overview of the underlying theoretical components, tools, and infrastructure needed for ZKP development. Additionally, we enlist and discuss the currently deployed and in-deployment application software that fall under the different blockchain and non-blockchain categories. The second survey by Sun *et al.* [13] takes a narrower approach, concentrating on the intersection of ZKPs with blockchain technology. It outlines the security challenges within blockchain's public and transparent nature and discusses how ZKPs can mitigate risks of private data exposure. Our work differentiates itself from these prior surveys by providing a more comprehensive practical application-oriented overview that covers both blockchain and non-blockchain applications of ZKPs in a broader context complemented by extensive analysis of the ZKP infrastructure being developed, including zkVMs, DSLs, and supportive technologies to guide developers to essential tools for building ZK-enabled applications efficiently.

This survey article serves as a comprehensive guide for practitioners and researchers, focusing on the broad range of practical applications and use cases of ZKPs. It systematically highlights the best practices, potential challenges, and the underlying impact of ZKPs on diverse application fields. Practitioners will find it a valuable roadmap that offers insights into incorporating ZKPs into various sophisticated digital systems. For researchers, it provides an overview of the current state of ZKP deployment, aiming to catalyze further research and development. By bridging the gap between theoretical cryptographic knowledge and practical application domains of ZKPs, this survey intends to enable professionals across disciplines to leverage the nuanced benefits of ZKPs in their work.

We organize the remainder of this article to methodically illustrate the intricacies of zkSNARKs and their wide range of applications. Section II characterizes the foundational components and requisite properties of SNARKs. Section III describes the infrastructure underpinning Zero-Knowledge Proofs, encompassing virtual machines, domain-specific
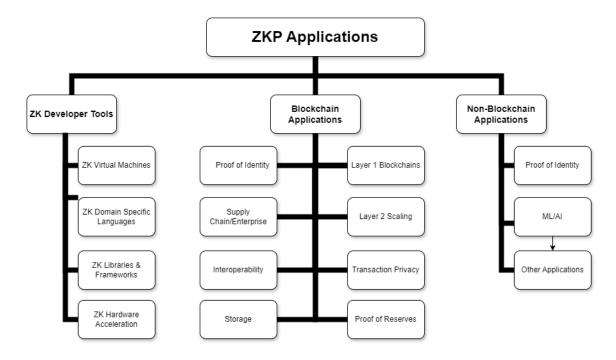
Fig. 1: Survey Structure

languages, and pertinent libraries. Next, Section IV provides an extensive dissection of blockchain applications, which reviews the role of ZKPs across various blockchain layers and functionalities. Section V extends the discussion to non-blockchain application domains, detailing ZKPs' utility in diverse contexts such as machine learning and digital identity verification. Finally, Section VI puts forward the future directions and discusses the expanding scope of ZKPs, then wraps up with a summary of our discussion and gives a reflective analysis of the presented content.

II. COMPONENTS OF ZKSNARKS

Zero-Knowledge Proofs (ZKPs) are a cryptographic tool that enables one party, known as the prover, to prove to another party, the verifier, that a certain statement is true without revealing any information beyond the validity of the statement itself [14]. A

"statement" in the context of ZKPs refers to a claim that can be mathematically verified, such as "I know two factors to a large number" or "I have executed a set of instructions from a virtual machine (VM) instruction set correctly". This introduction aims to clarify the relationship between the broader category ZKPs and a particular subset known as Succinct Non-interactive Arguments of Knowledge (SNARKs), which aims to simplify the verification process by eliminating the need for interaction between two parties[1].

There are several useful abstractions with which one could view the components that enable SNARKs. We will explore two such abstractions. First, we will follow the life-cycle of compiling high-level code into a provable form. Once the computational representation becomes provable, we will work with an interactive protocol, a process where two parties, referred to as a prover and verifier, exchange a series of messages to form a SNARK. First, we will explain the properties that a SNARK must adhere to.

A. Definition and Properties of SNARKs

SNARKs, an acronym that stands for Succinct Non-interactive Arguments of Knowledge, represent a subset of zero-knowledge proofs with unique characteristics. Defining the acronym can help in highlighting these attributes:

- **Succinctness**: SNARKs are distinguished by their compact proof size, which remains small regardless of the computational complexity or the size of the input data. This property is instrumental in environments where bandwidth or storage is limited, such as blockchain networks, ensuring that proofs can be transmitted and verified efficiently.

- **Non-interactivity**: Unlike some Zero-Knowledge Proofs that require back-and-forth communication between the prover and the verifier, SNARKs are non-interactive.

---

[1]Our introduction is necessarily brief, for a more thorough discussion, we refer the reader to more detailed explainers on zk-SNARKs such as [11].

This means that the prover can generate a single proof that the verifier can independently check without further communication. Non-interactivity is achieved through the use of a common reference string shared between the prover and verifier.

- **Arguments of Knowledge**: SNARKs assure not only that a certain statement is true but also that the prover possesses explicit knowledge of the information substantiating that statement. This aspect of SNARKs ensures that the proof carries with it a guarantee of knowledge, which is crucial for applications that require authentication or verification of authority.

SNARKs have several defining properties that ensure their functionality and security. These properties include:

1) **Completeness**: If a statement is true and both the prover and verifier follow the protocol honestly, the verifier will be convinced of this truth by a valid proof. Mathematically, for every valid deterministic input x and corresponding non-deterministic input w, often referred to as a witness, there exists a proof π such that the verifier accepts π as valid.

2) **Soundness**: If a statement is false, no cheating prover can convince the verifier of its truth. This is often a computational property, meaning that no efficient (polynomial time) prover can create a proof for a false statement that the verifier would accept.

3) **Knowledge Soundness**: Ensures that if a prover can convince a verifier of the truth of a statement, then the prover actually knows a private input for that statement. This distinguishes SNARKs from other non-interactive proof systems where a prover might be able to convince a verifier without actually knowing the private input.

4) **Zero Knowledge**: The proof does not reveal any information about the witness or the statement beyond its validity. There are different levels of zero-knowledge:

- *Computational Zero Knowledge*: The verifier learns nothing from the proof that they couldn't compute on their own.
- *Statistical Zero Knowledge*: The verifier learns almost nothing from the proof, except with a small statistical difference.
- *Perfect Zero Knowledge*: The proof gives absolutely no information to the verifier about the witness or statement, beyond the fact that the statement is true.

It's worth noting that the differences between computational, statistical, and perfect properties revolve around the nature of the information leakage (or lack thereof) and the strength of the security guarantee.

B. Lifecycle of a SNARK: From Python to Polynomials

1) Frontends: From High-level code to circuits

This section explores the transition from high-level programming constructs to the more abstract representation as circuits, which are instrumental to the construction of a SNARK.

Consider the following Python function[2]:

```
def function(a, b, c):
    square_a = a * a
    square_b = b * b
    square_c = c * c
```

---

[2]**A Practical Caveat:** While the transition from Python code to an arithmetic circuit illustrates the conceptual process underlying the construction of SNARKs, it's important to note that, in practice, directly translating high-level programming languages to circuit representations may not be computationally efficient. To address this, developers often utilize zero-knowledge domain-specific languages (often referred to as zk-DSLs) or similar frameworks specifically designed for creating zero-knowledge proofs. These will be explored later in section III-B.

```
multiply_a2b2 = square_a * square_b
multiply_b2c2 = square_b * square_c
sum_terms = multiply_a2b2 + multiply_b2c2
return sum_terms
```

This function calculates the evaluation of a three-variate polynomial $f(a, b, c) = a^2b^2 + b^2c^2$. Next, we represent this function as an arithmetic circuit. Arithmetic circuits are structured mathematical representations that decompose more-complex computations (such as the Python function above) into simple arithmetic operations (like addition and multiplication). This transformation, when coupled with the later transformations described below, will enable the properties of privacy and succinctness that enable the practicality of SNARKs.
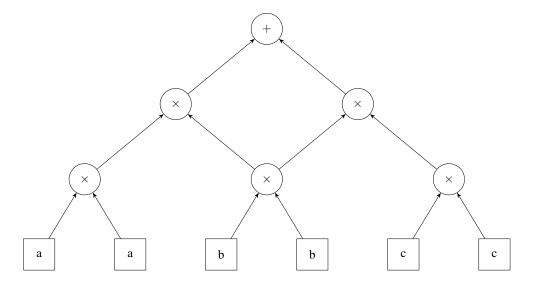
Fig. 2: An arithmetic circuit representation

a)

In the context of cryptographic circuits like the one in Figure 2, the elements a, b, and c are typically considered to be elements of a finite field $\mathbb{F}_p$, specifically numbers modulo a large prime number p. This use of a finite field ensures that all operations are performed within a closed, discrete set of numbers in the set $\{0, \ldots, \text{p-1}\}$, providing properties that help maintain cryptographic security.

2) Arithmetization: From circuits to matrices

Arithmetization is the process of encoding the wiring and gate operations of a digital circuit or model of computation into a mathematical representation. In the context of zero-knowledge proofs, this often involves translating the circuit into a format that is suitable for the generation of a proof that the circuit's execution satisfies certain conditions without revealing the specifics of the inputs or internal states. For brevity, we will focus on Rank-1 Constraint Systems (R1CS), which serve as a common arithmetization strategy for many zk-SNARKs. Some alternative arithmetization methods, namely Plonkish arithmetization, algebraic intermediate representations (AIR), and circuit constraint systems (CCS) have gained popularity in recent years as well [15], [16].

An R1CS is a way to represent a computation in the form of constraints that are linear equations. These constraints can be checked efficiently and are the foundation of various zero-knowledge proof systems. In an R1CS, each constraint is an equation of the form:

$$\mathbf{A} \cdot \mathbf{w} \circ \mathbf{B} \cdot \mathbf{w} = \mathbf{C} \cdot \mathbf{w}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are matrices, $\mathbf{w}$ is a vector of variables that correspond to the transcript of a digital circuit's execution, and $\circ$ denotes the Hadamard (element-wise) product.

As a mental model, one can think of R1CS as a system that verifies all parts of a program by checking individual, linear parts of its computation, ensuring that all constraints are satisfied for the program to be considered correct.

The operations performed by our example circuit can be encoded into an R1CS by expressing each operation as a set of linear constraints. In R1CS, the constraints take the form of equations where the left and right sides must multiply to give the output. For our circuit, we introduce variables and intermediate 'witness' variables to represent the computation steps. Here is how each operation in the circuit translates into constraints:

1) Multiply a by itself to get a^2 results in the constraint $a \times a - a^2 = 0$.

2) Multiply b by itself to get b^2 results in the constraint $b \times b - b^2 = 0$.

3) Multiply c by itself to get c^2 results in the constraint $c \times c - c^2 = 0$.

4) Multiply a^2 by b^2 to get $a^2 \times b^2$ results in the constraint $a^2 \times b^2 - a^2 b^2 = 0$.

5) Multiply b^2 by c^2 to get $b^2 \times c^2$ results in the constraint $b^2 \times c^2 - b^2 c^2 = 0$.

6) Add $a^2 \times b^2$ and $b^2 \times c^2$ to get the final result leads to the constraint $ab^2 + bc^2 - result = 0$.

To encode these constraints into matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, we introduce the following variables: $w_1 = a$, $w_2 = b$, $w_3 = c$, $w_4 = a^2$, $w_5 = b^2$, $w_6 = c^2$, $w_7 = a^2 b^2$, $w_8 = b^2 c^2$, and $w_9 = result$. Each matrix will have rows corresponding to constraints and columns corresponding to these variables, which can be further compressed as an array $\mathbf{w} = [1, a, b, c, a^2, b^2, c^2, ab^2, bc^2, \text{result}]$.

The vector $\mathbf{w}$, often referred to as the 'witness', captures an execution transcript of the circuit. The witness contains the values of all variables and intermediate steps that satisfy all the R1CS constraints for the given inputs. Given that anyone with a witness $\mathbf{w}$ can execute the circuit to verify if $\mathbf{w}$ is valid, a correct execution transcript $\mathbf{w}$ can be viewed as a proof that the computation was carried out correctly and that all of the R1CS's constraints are satisfied. If, at any point, an attacker obtained access to the R1CS or witness vector, access to the inputs, intermediate values, and outputs of the computation would be leaked.

Up to this point, there is no way to prove, in a publicly-verifiable way, that the

computation was performed correctly without allowing others to re-run the computation themselves. The next section, however, converts the R1CS into a form that allows for maintaining privacy while allowing for public-verifiability and a sublinear verifier runtime.

3) Backends: From matrices to polynomials

The backend phase of the SNARK lifecycle involves transforming the Rank-1 Constraint System (R1CS) matrices into a set of polynomial equations that form a Quadratic Arithmetic Program (QAP). This transformation is key to creating a SNARK, as it allows for the efficient verification of the computations represented by the R1CS. Succinctness, a property of SNARKs integral to a sub-linear verifier runtime, is achieved by encoding the matrix operations from the R1CS into polynomial equations. This more succinctly represents the original circuit's execution trace without sacrificing the soundness guarantees of the underlying argument.

Mathematically, a QAP is defined as a set of polynomial equations derived from the R1CS [17]. For matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ of size $m \times n$, where m is the number of constraints and n is the number of variables, we define the QAP as follows: Let $\{A_i(x)\}_{i=1}^m$, $\{B_i(x)\}_{i=1}^m$, and $\{C_i(x)\}_{i=1}^m$ be sets of polynomials where each $A_i(x), B_i(x), C_i(x)$ is a polynomial corresponding to the ith row of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ respectively. Then, the QAP consists of the set of equations $\{A_i(x) \cdot B_i(x) - C_i(x) = h(x) \cdot Z(x)\}_{i=1}^m$, where $h(x)$ is the polynomial that represents a solution to the R1CS, and $Z(x)$ is a verifier-chosen, fixed polynomial that evaluates to zero at $\{x_1, x_2, \ldots, x_m\}$, the set of distinct points associated with each constraint.

The QAP allows for the succinct representation of the R1CS and enables the prover to compute a single proof, which is a small set of polynomial evaluations. This proof attests that there exists a polynomial $h(x)$ that satisfies the QAP equation, implying that the prover knows a witness $\mathbf{w}$ that satisfies the R1CS.

The verifier, given the proof and the public polynomials $A(x)$, $B(x)$, and $C(x)$, can

efficiently check the QAP verification equation without having to perform the full set of R1CS computations. This results in a verification process with a significant speedup over computing the function represented by the circuit directly.

In summary, the conversion from matrices to polynomials and the creation of a QAP forms the bridge between the arithmetic circuit and the final SNARK. This process ensures that a valid execution of the circuit, one that satisfies all the constraints, is encoded into a format that supports the creation of a compact, easily verifiable proof. The integrity of this transformation is fundamental to the trustworthiness and security of the SNARK.

C. Conjoining Information Theory and Cryptography

Interactive proofs (IPs), when not paired with any cryptographic protocol, provide unconditional soundness, where the validity of a statement can only be confirmed if it is indeed true, independent of computational assumptions. Integrating an IP protocol with a cryptographic gadget, however, enhances both the security and other properties of the resulting SNARK. This cryptographic gadget, often referred to as a polynomial commitment scheme (PCS) [18], increases security assumptions but is also a large part of what makes SNARKs practical today through the succinctness property and an improved verifier runtime. While PCSs have become a necessary component in the large taxonomy of SNARK designs today, the necessity for private randomness in PCSs for cryptographic security complicates their integration with an IP protocol. Trusted ceremonies, which are critical for generating this randomness securely and ensuring that no single participant can compromise the setup, thus need to be executed before the SNARK enters production. If a trusted ceremony is executed incorrectly, false proofs will be able to be created and the protocol will be insecure – this happened in 2016 with Zcash's Sapling ceremony and another ceremony was later ran to replace it [19]. Lastly, to ensure that SNARKs are publicly-verifiable, so that collusion between the prover and verifier is not possible,

the SNARK must be made non-interactive. SNARKs achieve non-interactivity through the Fiat-Shamir heuristic [20], which substitutes the verifier's random challenges with outputs from a collision-resistant hash function. This modification not only simplifies the verification process but also makes the proofs publicly verifiable, enhancing both the practicality and security of SNARKs in broad applications.

III. ZKP Software Tools and Platforms

In this section, we explore the underlying infrastructure and tooling currently available to produce ZKP by first examining zero-knowledge virtual machines (zkVM), domain specific languages (DSLs), and supporting libraries, frameworks, and protocols. These core pieces of the ZKP space allow for statements we are familiar with to be translated into an arbitrary computation that is then represented as a verifiable ZK circuit in an efficient manner.

At a broad scale, the process can be split into two sections: the frontend and the backend. Given a program, the process begins with the frontend. The frontend's purpose is to take in a program and transform it into a format that a proving system discussed above can interpret and produce a proof from. The backend handles the proof generation process and does most of the heavy lifting in the process.

After that, the intermediate representation is translated into a constraint system and passed to the backend. The backend begins by putting the circuit representation of the program from the artihmetization scheme into a proving system such as Groth16 [21] or Bulletproofs [22]. Finally, a proof of execution is produced and the user can succinctly verify its correctness.

A. Zero-Knowledge Virtual Machines

1) Motivation and Definition

With the increased dependency on distributed computing in industries such as blockchain and cloud computing, the importance of private computing at a large scale grew exponentially. ZKPs offer a solution for providing an environment that performs computations in a trustworthy manner through a zero-knowledge virtual machine (zkVM). A zkVM is a virtual machine (VM) designed to generate an efficiently verifiable proof for executing an arbitrary computation while preserving the privacy of the program and its data in zero knowledge. The zkVM acts as a programmable ZK circuit that implements a VM, receives public and private inputs, and creates a certificate of valid execution. Similar to a generalized virtual machine, a zkVM maintains its own virtualized components, such as memory management, instruction scheduling, error handling, and others, using a privacy-focused approach with the help of a ZKP.

The motivation for building zkVMs stems from the desire to reduce the complexity of creating a ZKP. Previously, to produce a ZKP, a developer needed to have a strong background in cryptography, develop a circuit representation of their program, set up a backend for each proof, and have a large amount of computing power. To abstract away these steps from the developer, zero-knowledge virtual machines were developed with these features in mind.

2) Methodology

The first step of most zkVMs is to receive an input program from a higher-level language such as C++, Circom [23], Rust, or others. The program is then compiled to bytecode that the VM can translate to an instruction set architecture (ISA) that is generalized, like RISCV, or specialized, such as Miden Assembly [24]. These ISAs are minimal and optimized for cryptographic operations such as hashing to optimize the performance of the zkVM.
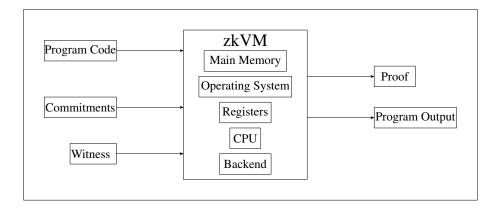
Fig. 3: General zkVM Architecture

Depending on the proving system, the setup phase begins after the program has been interpreted and translated to bytecode. This step involves tasks such as concealing private inputs, initializing a polynomial commitment scheme, key generation, and others. A set of constraints is placed on the program and the witness to enforce computational integrity in the computation trace. Building on what was mentioned in Section 2, constraints ensure the zkVM execution trace follows the guidelines outlined in the ISA. Every constraint is a polynomial for each statement inside the execution trace.

To check the validity of each instruction in the execution trace, a verifier can open any instruction in the zkVM to verify it was computed correctly. The prover defines a polynomial for each statement that the verifier can use the polynomial commitment scheme to verify. The verifier receives this polynomial and responds with a random value for evaluating the prover's polynomial. If the verifier concludes that the output is correct, then the statement in question is also considered correct.

The zkVM distinguishes itself from writing a specialized circuit for computations due to its high optimization level and reduction of complexity from designing a circuit. In a ZK computation, three common overheads must be taken into account:

1) Range Checks: A range check is a way to prove that an element in a finite field falls within a specific interval `[a, b]`. Due to the variance in fields, the importance of correct bit-sized integers in these fields, and the intensive operations throughout them, it is important to verify the inputs are valid. This is done using a range proof to verify that a given integer fits the necessary criteria, such as fitting into 32 bits without overflowing.

2) Bitwise operations: Nearly every computation in a ZK context needs some bitwise operation for tasks such as hashing. On standard computer hardware, these operations are highly optimized for numbers of standard sizes rather than for large elements in a finite field. In a ZK context, every bit inside of a field element must be proved, the decomposition of a field element into its bits must be proved, and finally, the result of a bitwise operation must be aggregated from bits into another field operation. With current hardware that is not optimized for ZK computation, these can incur large overheads in common operations.

3) Hashing. Traditional primitives use existing complex optimizations for bitwise operations as mentioned in 2); however, these bitwise operations take more time because of the above reasoning. Fundamental hash functions such as SHA256 become obsolete on zkVMs unless improvements are made. Considering this, many zkVMs use rivaling primitives such as Rescue or Poseidon or use an entirely different processor to handle the hashing.

By prioritizing efficiency in cryptographic protocols that leverage zero-knowledge proofs, zkVM enhances the performance of applications involved in secure data exchange, anonymous transactions, and confidential smart contracts. This optimization ensures that cryptographic operations, integral to privacy-preserving technologies, are executed with heightened speed and effectiveness.

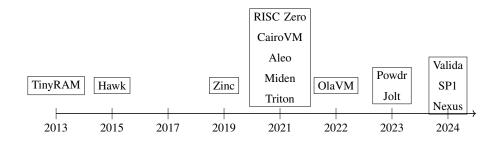The projects in the timeline are an overview of the prominent zkVMs over the last

Fig. 4: Timeline of zkVM Popular Projects

decade such as TinyRAM [25], Hawk [26], Zinc [27] , RISC Zero [28], CarioVM [29], Leo [30], Miden [24], Triton [31], OlaVM [32] , Powdr [33], and Jolt [34]. Some other zkVMs that are on the rise as of writing this paper are SP1 [35], Nexus [36], and Valida [37].

A project in the space is RISC Zero's [28] zkVM which allows users to demonstrate the accurate execution of arbitrary Rust and C++ code. This enables the construction of zero-knowledge applications that leverage the variety of existing Rust packages. Functioning as a verifiable computer, the zkVM mimics the operations of a genuine embedded RISC-V micro-processor, thus allowing any program that can compile down to the RISC-V instruction set to run on the zkVM. This emulation not only facilitates the development process for developers but also simplifies the creation of powerful zero-knowledge applications. RISC Zero also mains a ZK Validity Proof System that empowers users to substantiate the validity of Ethereum blocks. The zkVM is equipped to function on different computer systems, program sizes, hardware targets, and runtime/memory requirements.

Starkware [29], the company behind zk-STARKs (Zero Knowledge Scalable Transparent Argument of Knowledge) applications, created CairoVM. CairoVM is an efficient and practical architecture specifically compatible with the STARK proof system. It

instantiates a STARK based von-Neumann architecture known for its instruction set that allows support for traditional language features such as conditional branching, function calls, recursion, and its ability to have a deterministic or a nondeterministic nature in aspects like its restrictive memory model. The bytecode of the Cairo zkVM, which are generated by the Cairo assembler, represents the input program from a given user. From there, the Cairo Runner is used to obtain the execution trace of the program. The Cairo Runner is a computer program designed for the execution of a compiled Cairo program that operates in a distinctive manner compared to running a typical computer program. The key disparity arises from Cairo's support for nondeterministic code. Finally, a STARK prover for the Cairo AIR mentioned in Section 2 is employed to generate a proof for the assertion from the program.

Polygon's Miden VM [24] is another zkVM built to contribute to a zero-knowledge rollup, which is a modular second layer blockchain built on top of another blockchain such as Ethereum. Miden extends the capabilities of Ethereum by introducing features such as parallel transaction execution and client-side proving. This empowers developers to craft high-throughput and privacy-preserving decentralized applications (dApps) for sectors like DeFi, RWA, and the Metaverse. Popular programming languages like Rust are supported by Miden, which gives developers flexibility and familiarity. Another feature of Miden VM lies in its automatic generation of a STARK-based proof of execution for any program that runs on it. Miden VM operates by consuming programs in the form of a Merkelized Abstract Syntax Tree (MAST), where each node represents a code block in a binary tree structure. Execution begins at the tree's root, recursively processing each required block based on its semantics. If the execution of any code block encounters failure, the VM halts, preventing the execution of further blocks. The program structure involves a binary tree arrangement, with leaves containing linear instruction sequences and internal nodes determining control flow. As a stack machine, Miden VM utilizes a push-down stack of practically unlimited depth, aligning with the

architecture of a STARK. The Miden standard library enhances functionality by providing optimized implementations of commonly-used primitives. This library facilitates the reduction of shared code between parties involved in proving and verifying program execution, achieved by serializing calls to standard library procedures as a fixed 32 bytes, irrespective of the procedure's size. These design goals collectively contribute to Miden VM's efficiency, flexibility, and security.

In conclusion, zkVMs offer a broader spectrum of applicability through its support for general and specialized computation. A zkVM is designed to accommodate a wide range of computations unlike some zero-knowledge proof systems. This flexibility permits developers to implement complex algorithms and diverse computations while benefiting from the privacy and security guarantees inherent in zero-knowledge proofs.

B. Domain Specific Languages

1) Motivation and Definition

A zkDSL (zero-knowledge domain specific language) is a programming language written specifically to provide the programmer a connection between a high-level representation of a program and the low-level intricacies of a ZKP such as writing circuits.

2) Methodology

Some zkDSLs that are listed are O1JS [38], Circom [23], Cairo [29], Noir [39], Juvix [40], and Lurk [41],.

The goal of a zkDSL is to translate a high-level language into an arithmetic circuit that can be passed into a proving system to output a proof on execution of a given program. Some of the most common circuits targeted are R1CS, Plonk, and AIR, as shown in the Arithmetization Schemes table 5. Each language has a niche that it is trying to fulfill. For example, Circom is a language that is focused on writing arithmetic circuits whereas Lurk is meant to be a generalized programming language.

	Proof System(s)	Intermediate Representation	Syntax Comparison	Purpose(s)
O1JS/Snarky	Kimchi	Plonk Variant	Typescript	General, Smart contracts
Circom	Groth16	R1CS	N/A	Circuit development
Cairo	STARK	Algebraic Intermediate Representation	Rust	Smart contracts
Noir	Plonk, Groth16, Marlin	Abstract Circuit Intermediate Representation	Rust	Circuit development
Juvix	Plonk, Halo2	VampIR	Ocaml	Anoma Interts
Lurk	Groth16, Nova	R1CS	Common Lisp, Scheme	General

TABLE II: Comparison of zkDSLs

While zkDSLs achieve the goal of abstraction, some drawbacks must be considered. First, zkDSLs encounter challenges in memory management complexity, primarily due to the inherent intricacies associated with zero-knowledge proofs and cryptographic computations. The nature of zkDSLs, which often involve complex mathematical transformations and intricate circuit representations, demands efficient memory management strategies. The need to handle cryptographic primitives, large-scale boolean circuits, and data structures introduces complexities that traditional memory management systems may find challenging to navigate. Languages like Cairo implement optimization techniques in their compiler or VM to reduce this overhead through constraint reduction and cycle elimination.

Second, for some time, zkDSLs have a history of problems with the developer paradigms of recursion, conventional conditionals (excluding ternaries), mutable variables, and user-defined structures, causing unique coding challenges. Recursion, a staple in conventional programming, faces challenges due to the absence of a direct stack and the deterministic
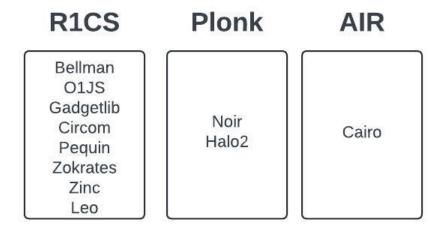
R1CS

Bellman
O1JS
Gadgetlib
Circom
Pequin
Zokrates
Zinc
Leo

Plonk

Noir
Halo2

AIR

Cairo

Fig. 5: Arithmetization Schemes

Code → Constraint System → Cryptographic Compiler → Proving System → Proof

Fig. 6: Overview of zk-DSL Process

path required by circuit-based computation. The representation of if-else statements becomes intricate within algebraic circuits, favoring ternary expressions but sacrificing the straightforwardness of traditional conditionals. The immutability inherent in algebraic circuits complicates the emulation of mutable variables that change state during computation. Furthermore, the structured and hierarchical nature of user-defined structures, commonplace in languages like C++, presents a unique set of challenges. Developers navigating the ZKP circuit structure must consider these drawbacks, as the departure from traditional programming paradigms necessitates a reevaluation and adaptation of coding practices.

3) Applications

A classic example of a zkDSL is the circuit writing language Circom [23], which has been used in applications such as TornadoCash [42] or DarkForest [43]. Circom represents both a programming language and a compiler designed for the creation of arithmetic circuits that are subsequently compiled into Rank-1 Constraint Systems (R1CS). This intricate process allows programmers using Circom to articulate arithmetic circuits at a constraint level, with the compiler generating a file containing the R1CS description, as well as WebAssembly and C++ programs. These output programs facilitate the computation of all circuit values. Additionally, an open-source library, Circomlib [44], features circuit templates to be used in to support writing other circuits. On top of that, O1JS [38] is a library proficient in generating and validating Zero-Knowledge (ZK) proofs derived from R1CS. Collectively, this suite of software tools serves to abstract the intricacies of ZK proving mechanisms, providing an interface for modeling low-level descriptions of arithmetic circuits.

Lurk [41] is a Turing complete LISP-based programming language that autonomously generates zk-SNARKs for arbitrary programs. It presents programs as data to the universal Lurk interpreter circuit to achieve Turing completeness without compromising the size of the proof artifacts generated. This departure from traditional approaches aims to enhance the scalability and capabilities of a ZKP. Further, the need for ad-hoc compilation of programs into flat circuits, a conventional process that imposes significant constraints on the size and complexity of achievable computations, is eliminated.

Leo [45] is a programming language designed for the development of formally verified zero-knowledge applications on the blockchain. Leo establishes a execution environment devoid of restrictions on running time, stack size, or instruction sets, offering flexibility. Beyond the intrinsic benefits of ensuring application privacy and mitigating maximal-extractable value (MEV), Leo has two other helpful properties. Firstly, applications undergo formal verification concerning their high-level specifications. Secondly, the
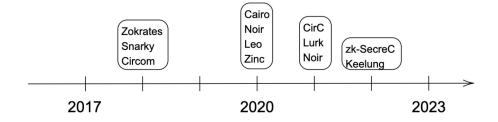
Fig. 7: Timeline of Popular zk-DSLs

succinct verification of applications is made accessible to anyone, irrespective of the application's size. Leo maintains a set of tools, including a testing framework, package registry, import resolver, remote compiler, formally defined language, and theorem prover, specifically tailored for general-purpose zero-knowledge applications. To summarize, zkDSLs emerge as versatile tools with a broad spectrum of applications, ranging from smart contract programming to general computation. These specialized languages are instrumental in the development of privacy-preserving features within smart contracts, facilitating secure and confidential transactions on blockchain networks. Beyond blockchain, zkDSLs find utility in general computation scenarios, where their application extends to fields such as data privacy, machine learning, secure distributed computations, and confidential data analysis. In cryptographic applications and security protocols, zkDSLs play a pivotal role by enabling the creation of proofs for statements related to cryptographic properties, thereby enhancing the security and privacy of digital transactions and communication. This versatility positions zkDSLs as valuable tools for developers working on various applications, providing a user-friendly interface to harness the power of zero-knowledge proofs in diverse domains.

C. Libraries and Frameworks

1) Motivation and Definition

In the domain of zero-knowledge proofs (ZKP), the development process often entails numerous operations characterized by a high barrier to entry, necessitating a profound understanding of computational strategies, cryptographic primitives, elliptic curves, and the intricacies of ZKP mechanisms. For developers, the hardship lies in crafting reusable code and mitigating redundancy, particularly within ZK proofs, where constraints such as circuit size and gas usage can impose significant limitations. To address these challenges, a multitude of libraries have emerged, offering implementations for cryptographic "gadgets" that facilitate the construction of Rank-1 Constraint Systems (R1CS) instances from modular "gadget" classes as listed in Table III. To faciliate the proving and verification process, elliptic curves as utilized, as mentioned in the previous section. Operations on these curves is relatively standardized and does not always need to be recreated, so libraries avoid the issue of duplicated code in this area. The combination of circuit gadgets and elliptic curves allow for the modular composition of an application in conjunction with a compatible proving system such as Groth16 [21] or Bulletproofs [22]. These libraries play a crucial role in abstracting the complexities of preprocessing in a modular manner, enabling developers to focus on higher-level design aspects and promoting code reusability across ZKP applications. This emphasis on modularization and abstraction enhances development efficiency and advances innovation within the ZKP domain by fostering collaboration and knowledge sharing among developers.

2) Methodology

In the design of ZKP frameworks and libraries, several fundamental principles are leveraged to optimize performance and efficiency. One crucial aspect is the operation over small fields, as demonstrated by systems like ethSTARK [46] and Plonky2 [47]. Unlike traditional elliptic curve groups, which require large field sizes (e.g., 256 bits)

Circuit Gadgets	Proving Systems	Elliptic Curves
Arithmetic Operations	Groth16	BLS12-381
Booleans	Plonk	BLS12-377
Range Proofs	Halo	BN254
SHA256	Bulletproofs	MNT-298
MIMC	Pinocchio	MNT-753
Poseidon	Marlin	Jubjub
Pedersen	Gemini	Secp256k1
blake2	Sonic	Curve25519
Lookups	Brakedown	BN256
	Spartan	

TABLE III: Categories of Cryptographic Components

for standard security levels, these systems utilize smaller prime fields (e.g., 64 bits). This approach capitalizes on the efficiency of small-field arithmetic, resulting in state-of-the-art proving performance. Additionally, using small-field elements reduces storage requirements, improving CPUs' cache efficiency. Another critical consideration is the flexibility in field selection. These schemes often use computationally structured fields, providing additional optimization opportunities. Finally, these frameworks and libraries prioritize using cheaper cryptographic primitives, ensuring cost-effective and efficient ZKP implementations [48].

Abstracting away finite field operations holds paramount importance in zero-knowledge proofs (ZKPs). This endeavor necessitates the implementation of fundamental arithmetic operations, including addition, subtraction, multiplication, (modular) exponentiation, and inverse exponentiation, over finite fields. Considering these operations are used so heavily in constructing a ZKP, their modularity and efficiency must be implemented to suffice for any use case. [49]

In elliptic curve cryptography, an elliptic curve is typically defined over a prime

field of a specific order denoted as F_q. The elliptic curve group $(E(F_q))$ comprises the subgroup of points within the field that lie on the curve, including a distinguished point at infinity. While some SNARKs function over elliptic curves without necessitating pairings, others rely on pairings and thus necessitate pairing-friendly elliptic curves. Pairings, a fundamental operation in cryptographic protocols, involve taking an element from the first group (G_1) and another element from the second group (G_2) to compute an element in the target group (G_T), typically denoted as $e(P, Q)$, where P belongs to G_1 and Q belongs to G_2. Efficient implementation of pairings, scalar multiplication, and multi-scalar multiplication (MSM) over pairing-friendly elliptic curves is essential for achieving computational efficiency. Therefore, prioritizing the efficient implementation of these operations is crucial for optimizing the performance of cryptographic protocols relying on pairing-friendly elliptic curves. [49]

To facilitate proof verification, ZKP frameworks often include the specification of cryptographic primitives within their design at the cost of the core SNARK implementation's efficiency, as mentioned above. However, not all computations conform to arithmetic modulo p, necessitating the development of circuits for non-native field operations. For instance, verifying elliptic-curve-based cryptographic primitives, such as ECDSA signatures, requires computations over a different field, such as $\mathbb{Z}_q$, which SNARKs do not natively support. Additionally, SNARKs often exhibit inefficiencies when dealing with traditional hash algorithms and signature schemes, like SHA-2 and ECDSA. To address this, the community has proposed SNARK-friendly alternatives, such as Poseidon Hash, Keccak Hash, Pedersen Hash, MIMC Hash, and Ed25519 (EdDSA signature), which are specifically optimized for use within SNARKs. Despite the benefits of utilizing SNARK-optimized primitives, practical applications often necessitate CPU-optimized primitives due to constraints in non-native field arithmetic. For example, some applications require the verification of ECDSA signatures, which involves non-native field arithmetic, leading to numerous constraints that increase the complexity and

	BN254	BLS12_381	BLS12_377
arkworks	✓	✓	✓
gnark	✓	✓	✓
blstrs		✓	
ffjavascript	✓	✓	
pairing_ce	✓	✓	
zkcrypto		✓	
halo2_curves	✓		
pasta_curves		✓	

TABLE IV: Popular Elliptic Curves in ZK Libraries

verification time of the proof. [49]

By writing these building blocks piece by piece, researchers and developers can create SNARKs across diverse settings, thus facilitating informed decision-making and enhancing the practical utility of SNARK-based cryptographic protocols.

3) Applications

Arkworks [50] is a Rust ecosystem tailored for zkSNARK programming, offering a set of libraries to streamline zkSNARK application development. These libraries implement essential components, including generic finite fields and R1CS constraints, enabling developers to integrate zkSNARK functionalities into their applications seamlessly. Arkworks also provides interfaces for various purposes, such as defining interfaces for SNARKs and relations (e.g., R1CS, AIR) and offering SNARK proving systems like Groth16 and Marlin. Further, the ecosystem includes tools for circuit building and algebraic operations for finite fields and elliptic curves, empowering developers to implement zkSNARKs in their projects efficiently.

Gnark [51] is a library that enables developers to design circuits using the programming language Go. It employs a versatile API and command line interface to accomplish the ZKP process in a familar way to developers. The two proving schemes that Gnark

supports are Groth16 and Plonk, along with a variety of elliptic curves for developers to choose ranging such as BN254, BLS12-381, and BLS12-377. A core focus of the Gnark library is speed of the prover and verifier, since it offers an API for both the frontend and the backend of the proving process. The standard library of Gnark implements common functions such as the MiMC hash function, EdDSA signature verification, Merkle proof verification, and a generalized zk-SNARK verifier. The performance of circuits written in Gnark can be analyzed using the in suite profiling tools available within the library.

CirC [52] is a project dedicated to compiler infrastructure for cryptosystems and verification. It focuses on cryptographic tools such as proof systems, multi-party computation, and fully homomorphic encryption, typically applied to computations expressed as systems of arithmetic constraints. These applications require compilers that can translate high-level programming languages (e.g., C) into such constraints. CirC aims to provide a shared infrastructure for building constraint compilers, offering a valuable resource across various applications. These compilers can translate code into a Rank-1 Constraint System (R1CS), enabling efficient implementation of cryptographic protocols and verification mechanisms. Circify simplifies certain aspects of supporting a new language by transforming stateful programs with complex control flow into flat circuits. However, it does not address language-specific features like type-checking and namespacing. Circify supports various frontends, including C, ZoKrates [53] (Z#) and Circom. This flexibility allows developers to choose or build the frontend that best suits their needs while benefiting from Circify's capabilities in managing program transformations for cryptographic applications.

The TypeScript library O1JS [38] is designed to cater to users with a web development background, offering a user-friendly approach to writing ZK programs and smart contracts for the Mina [54] blockchain. It is described as "an embedded DSL" and is executed as normal Typescript. This library permits developers to write arbitrary ZK programs utilizing many built-in provable operations, including basic arithmetic,

hashing, signatures, boolean operations, and comparators. With o1js, developers can create zkApps on Mina, smart contracts that execute client-side and handle private inputs. The entire o1js framework is packaged as a single TypeScript library, making it accessible in web browsers and Node.js environments, thus allowing developers to integrate their ZK programs into existing web applications.

Various implementations of Zcash's Halo2 [55] proving system, such as those by Axiom and the Ethereum Foundation, provide fundamental primitives for writing zero-knowledge proof circuits. The proving system involves several stages, from committing to polynomials that encode the main components of the circuit, including cell assignments, permuted values, products for lookup arguments, and equality constraint permutations. The next step is constructing the vanishing argument, which constrains all circuit relations to zero, including standard and custom gates, lookup argument rules, and equality constraint permutation rules. The polynomials are then evaluated at all necessary points, including relative rotations used by custom gates and vanishing argument pieces. Finally, the multipoint opening argument is constructed to ensure the consistency of evaluations with their commitments, and the inner product argument is run to create a polynomial commitment opening proof for the multipoint opening argument polynomial. Halo2 is known as a relatively low-level library with high customizability.

Nova [56] is a recursive SNARK that enables incrementally verifiable computation (IVC), a cryptographic primitive that allows a prover to produce a proof of correct execution of a "long-running" sequential computation in an incremental fashion. IVC allows for proofs to build on top of each other in an efficient fashion that speeds up the entire proving and verification process. Nova is implemented in Rust and supports three curve cycles: Pallas/Vesta, BN254/Grumpkin, and secp/secq. It supports frontends such as Bellpepper [57], and its native APIs accept circuits expressed with Bellpepper, Circom, and Lurk.

D. Hardware Acceleration

1) Motivation and Definition

Historically, the speed and memory requirements of ZK proof generation have limited their applicability. The required computations inside of a ZKP, such as hashing, multi-scalar multiplications, and fast-fourier transforms, create a burden for each use case. To reduce the overhead required by ZKPs, various projects have emerged to enhance the performance of ZKPs and their potential implementations.

2) Methodology

Hardware acceleration is defined as the use of optimizing or creating dedicated computer components to improve the performance and efficiency of a specific operation. The main instruments used for this acceleration such as field programmable gate arrays (FPGAs), graphics processing units (GPUs), and application-specific integrated circuits (ASICs) in the ZK world. The limiting factors of hardware acceleration projects are the memory capacity, speed of memory access, speed of data transfer, and speed of arithmetic units. In proof systems where both number theoretic transforms (NTTs) and multi-scalar multiplications (MSMs) are used, the majority of the proof generation time is spent on MSMs, with NTTs accounting for the remainder. Both MSMs and NTTs present performance challenges that can be addressed in several ways. MSMs can be executed on multiple threads, allowing for parallel processing. However, when dealing with large vectors of elements, the multiplications may still be slow and demand considerable memory resources. Additionally, MSMs face scalability issues and can remain sluggish even when extensively parallelized. On the other hand, NTTs involve frequent data shuffling during the algorithm, making them difficult to distribute across a computing cluster. They also require significant bandwidth when run on hardware due to the need to load and unload elements from large datasets. For instance, if a hardware chip has 16GB of memory or less, running NTTs on a dataset larger than 100GB would necessitate data

transfers over the network, significantly slowing down the operations [58].

Both MSM and NTT can be accelerated on GPUs, particularly MSM through an algorithm called 'Pippenger'. This process involves rewriting the computationally intensive tasks from the CPU to the GPU using CUDA or OpenCL, allowing the code to be compiled and executed directly on the GPU. For finer-grained acceleration, developers can optimize memory usage by maximizing the use of fast memory and minimizing slow-access memory to reduce costly data transfers, especially between the CPU and GPU. Additionally, optimizing execution configuration by balancing work across multiprocessors, building concurrent kernels, and allocating resources judiciously can maximize hardware utilization. The goal is to parallelize the entire workflow, minimizing sequential execution where different parts depend on each other's results. Open-source implementations allow developers to quickly start their modifications [59].

FPGAs, or field-programmable gate arrays, offer programmable hardware fabric that can be reconfigured multiple times, cutting manufacturing costs compared to ASICs and providing greater flexibility in hardware resource usage than GPUs. Although optimizing NTTs on GPUs is achievable, frequent data shuffling can lead to significant communication overhead between the GPU and CPU. By implementing the logic directly into the circuit design, FPGAs can potentially perform the task faster. Most open-source implementations for zero-knowledge proofs are written in Rust due to its memory safety and cross-platform compatibility. However, FPGA development tools are typically adapted to C/C++, requiring teams to translate these implementations [60].

GPUs offer fast development times with well-documented frameworks like CUDA and OpenCL and are readily available and cost-effective. However, GPUs are power-hungry, even when exploiting data and thread-level parallelism. In contrast, FPGAs have a more complex development cycle, requiring specialized engineers but allowing for low-level optimizations and providing lower latency, especially for large data streams. FPGAs are more expensive and less readily available than GPUs [58].

ASICs, or application-specific integrated circuits, are customized for particular uses and are permanently etched into silicon, making the design and manufacturing process much more complex and time-consuming compared to FPGAs. Despite this, advancements such as Leo's new integrated chip for accelerated proof generation demonstrate ongoing developments in this area. ASICs are believed to be the most promising hardware acceleration; however, they still have major barriers to entry. Programmability and logic modifications are difficult on ASICS as they possess write-once business logic, necessitating a complete rebuild of the system for any modifications. Conversely, FPGAs and GPUs can be reprogrammed multiple times, allowing the same hardware to be used across various projects with different proof systems or updates. This reprogrammability makes FPGAs a more versatile alternative compared to ASICs. Additionally, the time required for ASIC design, production, and deployment usually spans 12 to 18 months or more [58].

3) Applications

Ingonyama [61] is a hardware acceleration company that integrates chip design with mathematics and advanced algorithms to enhance the performance of compute-intensive cryptography. They maintain a library called ICICLE, a cryptography library for ZKPs using GPUs. ICICLE implements various cryptographic primitives such as elliptic curve (EC) operations, multi-scalar multiplication (MSM), number theoretic transform (NTT), and the Poseidon hash on GPUs. The Polynomial API offers a framework for polynomial operations for developer convenience. Additionally, ICICLE has bindings for Rust and Golang and integrates with projects like Gnark and EZKL. It can also be run in Google Colab. Ingonyama is advancing for a zero-knowledge processing unit (ZPU) defined as "a versatile and programmable hardware accelerator, designed to address the emerging needs of ZKP processing."

Cysic [62] is a ZK accelerator focused on developing ASICs and their accelerated zkVM. The system architecture features an executor responsible for executing programs,

hardware for controlling and distributing segments, and a configurable number of specialized chips to generate ZK proofs for each segment program. Leveraging its expertise in ASIC design and GPU engineering, Cysic aims to overcome challenges by offering ZK compute-as-a-service to various ZK projects. They aim to produce ASICs that specifically target MSMs, NTTs, and other general operations, while maintaining flexibility to adapt to many proving systems. Cysic's ongoing efforts concentrate on specialized ASIC design for real-time ZK proof generation.

Fabric Cryptography [63] introduces The Fabric Verifiable Processing Unit (VPU), a processor designed for cryptography applications ranging from ZKP to FHE. The VPU features a custom instruction set architecture tailored for next-generation cryptography, including ZKP, FHE, MPC, and other algorithms. It offers acceleration for MSM, NTT, polynomial evaluation, as well as Poseidon (1, 2), Blake, and other hash functions. The VPU supports multiprecision vector lanes up to 384-bit and includes a RISC-V core for enhanced programmability. It supports PCIe 4.0 x16 lanes, providing up to 256 Gbps per chip, and is equipped with high-bandwidth DRAM. In addition, Fabric Cryptography offers a PCIe card featuring 3x FC1000 chips for parallel ZK proof generation, a PCIe interface for comprehensive on-chip proving and encryption, and DRAM for recursive ZK proof generation and mining workloads. They also provide server systems and data centers to support larger workloads.

Irreducible [64] offers proving as a service designed for scalability, powered by FPGA-accelerated server clusters. Irreducible supports popular proof systems such as Plonky2 and Polygon zkEVM, with plans to support next-generation systems like Binius and Plonky3. Their FPGA-accelerated server clusters are specifically designed for cryptographic computation at scale. By connecting their FPGAs directly using AMD's high-speed Aurora protocol, they minimize unnecessary data transfers between the CPU and FPGAs. Operating independently of public cloud infrastructure providers like Amazon Web Services and Google Cloud Platform provides redundancy essential for the

decentralized networks they support. Irreducible also features a fully-pipelined FPGA architecture for ZKP-friendly Merkle trees using the Poseidon hash and NTT.

Supranational [65] offers hardware-accelerated cryptography for verifiable and confidential computing. They provide BLST, an IETF-compliant BLS12-381 signature library focused on security and performance. They also implement a simple API for generating VDF and SNARK proofs, powered by fast, open-source implementations running on a high-availability cloud. Additionally, Supranational is developing Sppark, an arbitrary-precision arithmetic accelerator for cryptographic operations, including VDFs, SNARKs, polynomial commitments, and accumulators.

IV. BLOCKCHAIN APPLICATIONS

Use Case	Projects
Layer 1 Blockchains	ZCash, Aleo, Mina
Layer 2 Scaling	Polygon zkEVM, zkSync Era, Scroll, Linea, Starknet, Aztec
Smart Contract/Transaction Privacy	Hawk, Tornado Cash, Privacy Pools, Penumbra, Mina zkApps, Noir
Proof of Identity	Semaphore, WorldID, zPass, Galxe protocol
Supply Chain/Enterprise Blockchain	QEDIT, zk-BeSC
Interoperability	zkBridge, Telepathy,
Blockchain Storage	Herodotus, FileCoin
Proof of Reserves	Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges, Proven: ZeKnow Solv

TABLE V: Overview of blockchain-based applications of Zero-Knowledge Proofs

A. Layer 1 Blockchains

1) Motivation and Definition

Layer 1 blockchains, such as Ethereum and Bitcoin, are foundational blockchain networks that provide a transparent and immutable ledger for data storage. However,

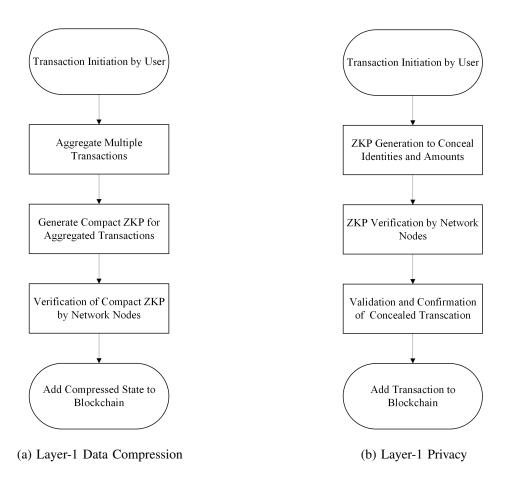(a) Layer-1 Data Compression

(b) Layer-1 Privacy

Fig. 8: Comparison of Layer-1 Data Compression and Privacy Approaches using ZKPs

these platforms face challenges in privacy and scalability. The emergence of zero-knowledge proofs (ZKPs) has induced the development of novel Layer 1 solutions that prioritize privacy and data efficiency. These blockchains integrate ZKPs directly into their base layers, offering a dual advantage: enhanced privacy through transaction concealment and improved scalability via data compression, as illustrated in Figure 8. The primary motivation for ZKP-based Layer 1 blockchain is to bridge the gap between the inherent transparency of conventional blockchains with the increasing demand for data privacy

and efficient on-chain data management.

2) Methodology

Layer-1 blockchains utilize ZKPs to obscure critical transaction details, such as the identities of parties involved and the transaction amounts. This concealment is achieved through advanced cryptographic techniques, including zk-SNARKs or zk-STARKs, that validate transactions without revealing their underlying data. To address scalability challenges, ZKP-based Layer-1 blockchains employ state compression. This technique utilizes ZKPs to create compact proofs that validate large sets of transactions or state transitions, thereby reducing the data volume required on-chain. In these networks, participating nodes are responsible for generating and verifying ZKPs. This ensures transaction integrity while maintaining privacy. The consensus mechanisms of these blockchains are uniquely designed to incorporate ZKP validation. This integration ensures that only transactions authenticated through ZKPs are confirmed and appended to the blockchain.

3) Applications

As a pioneering Layer 1 blockchain, ZCash [66] utilizes zk-SNARKs to offer private transactions, where the details of the sender, receiver, and transaction amounts are encrypted. In ZCash, zk-SNARKs enable the encryption of transaction data, ensuring the anonymity of both the sender and receiver, as well as the confidentiality of the transaction amount. This mechanism allows the network to validate transactions without disclosing the underlying data, thereby preserving the privacy of users. ZCash's implementation of zk-SNARKs is notable for its efficiency, allowing for the verification of transactions in a matter of milliseconds, which is a significant advancement over other cryptographic techniques that were more computationally intensive. The ZCash blockchain leverages a novel cryptographic method known as a "shielded transaction," where the transaction metadata is encrypted, and zk-SNARKs are used to prove that the transaction does not violate the network's consensus rules. This approach enables a high degree of privacy while maintaining the integrity and security of the blockchain. ZCash offers optional

Cryptocurrency	Cryptographic Method	Purpose
ZCash	zkSNARKs	Enables the encryption of transaction data to provide privacy by allowing transaction verification without revealing sender, receiver, or amount.
Monero	Ring Signatures, Stealth Addresses, Ring Confidential Transactions (RingCT)	Ring signatures obscure the sender's identity, stealth addresses hide the receiver's address, and RingCT conceals the transaction amount, ensuring privacy for all transactions.
Aleo	zkSNARKs	Used for creating private smart contracts which allow for verifiable computations without revealing underlying data, enabling privacy in decentralized applications.

TABLE VI: Overview of Cryptographic Methods in Privacy-Centric Cryptocurrencies

privacy features allowing users to choose between shielded and transparent transactions, unlike Aleo and Monero, where privacy is mandatory for all transactions as depicted in Table VI.

Aleo [30] is a distinctive Layer 1 blockchain platform that employs zero-knowledge proofs to enhance privacy and scalability in decentralized applications (dApps). It is designed to facilitate the development and deployment of private applications on the blockchain. Aleo achieves this with the help of a unique framework for constructing dApps that can perform computations in a private and verifiable manner. Its adoption of zk-SNARKs based systems allows developers to create applications where users can interact and transact without revealing sensitive information. This system empowers users to maintain control over their data, ensuring privacy and security in their interactions on the blockchain. Aleo's approach to blockchain architecture is focused on provid-

ing a scalable solution that addresses the common limitations associated with public blockchains, such as privacy concerns and throughput bottlenecks.

The Mina Protocol [54] is an innovative Layer 1 blockchain that introduces a unique approach to scalability and privacy through the use of recursive zk-SNARKs. This innovative protocol is designed to maintain the succinctness of a blockchain by keeping the network size constant (22kB) regardless of the total number of transactions processed. This is achieved through the recursive composition of zk-SNARKs, which allows each new block to contain proof of the validity of the entire blockchain history. As a result, the Mina Protocol maintains a drastically reduced blockchain size compared to traditional blockchains, enhancing scalability and usability.

B. Layer 2 Scaling

1) Motivation and Definition

As aforementioned, ZKP can be used for the property of data succinctness, especially when it comes to blockchain validity proofs. A significant pain point of blockchains is the scarcity and cost of block space, which can be solved using Layer 2 scaling solutions such as rollups. The most prominent types of rollups are optimistic rollups and ZK rollups, but the latter will be our main focus. In ZK rollups, zero-knowledge proofs are used to succinctly prove the validity of state changes to a Layer 1 blockchain without requiring validator nodes on a Layer 1 chain to execute those transactions. In essence, the Layer 2 rollup becomes a cheap modular execution layer that benefits from the security and decentralization of Layer 1, and this enables blockchains to scale by significantly reducing usage costs. Some ZK rollups can also provide privacy-preserving properties using zero-knowledge proofs.

2) Methodology

In Layer 2 rollups [67] (all of which achieve finality on Ethereum), computation is handled outside of Layer 1, and only state changes, deposits, and withdrawals are posted

to Layer 1 through the rollup smart contract. This Layer 1 smart contract contains and maintains a state root, which is the Merkle tree root of the state of the rollup, including accounts and balances. A rollup sequencer is an entity, which could range from a single server to a decentralized network of nodes, which orders transactions, produces L2 blocks, and adds rollup transactions to the ZK-rollup contract with a ZK validity proof. The sequencer publishes a batch of highly compressed transactions as well as the previous state root and the new state root after processing all transactions in the batch. Using the ZK validity proof provided by the sequencer, the rollup contract checks that the new state root is valid, then swaps the old root for the new one. The transaction batches published by sequencers are written to Ethereum in the form of encoded function calls, stored either in the calldata of the EVM, which is a data area used to pass arguments to a function and does not modify the blockchain's state, or in temporary data storage locations called blobs (post-EIP-4844). This serves as a cheap way to store data on-chain, making it possible for individuals to re-construct the state of the rollup using such compressed transactions. Since the Layer 1 rollup contract can quickly verify a zk-SNARK or zk-STARK proof on any amount of large computation, computation is almost fully offloaded to the Layer 2. Through these methodologies, ZK rollups minimize the space and computation restraints of Layer 1 Ethereum, which simply validates state changes and provides inherent decentralization and security [67].

Because Layer 2 scaling solutions are built on top of smart contract-based blockchains, specifically Ethereum, rollups need to provide proofs for state transitions on the Ethereum Virtual Machine (EVM), which is Ethereum's Turing-complete computation engine. Since ZK-rollups must attest to the correctness of computations with ZKP before posting batches to Layer 1, they must go through extra steps in code execution which the EVM does not do. Rather than loading smart contract bytecode into the execution layer and simply posting the result of those computations, ZK-rollups must generate validity proofs for each transaction's state transition.
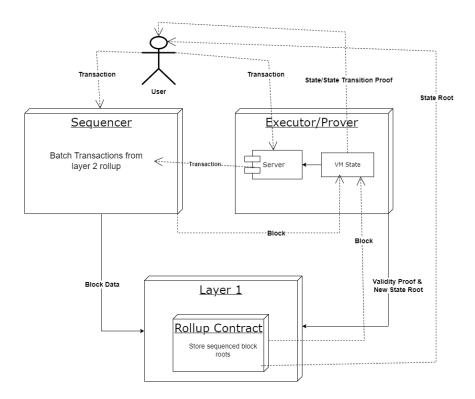
Fig. 9: Generalized ZK Rollup Architecture, redrawn based on [68]

Because EVM opcodes are designed for general-purpose computations, proving EVM computations in ZK circuits is too resource intensive and complex. Consequently, it is very difficult to ensure EVM-compatibility in ZK rollups, resulting in varying architectural designs:

- zkEVM: A zkEVM is a solution which embeds zero-knowledge proofs into EVM smart contract execution by recreating existing EVM opcodes for proving in circuits. A zkEVM computes state transitions just like the typical EVM, however it creates ZK validity proofs to verify the correctness at every operation, including state changes and computations.

- Custom VMs: Some approaches involve creating new high-level or intermediate

languages and virtual machines that are more amenable to ZK proofs but can still support EVM-like operations.

zkEVMs allow easier execution of Solidity smart contracts, while custom VMs may require developers to write smart contracts in some other smart contract language or modify their existing EVM-based implementations due to ZK limitations.

3) Applications

zkSync Era [69] is a zkEVM rollup developed by Matter Labs. Identical to the previously mentioned methodology, zkSync Era utilizes ZK-SNARKs to provide validity proofs of off-chain computation. Smart contracts can be written in Solidity or Vyper and called using the same clients as other EVM-compatible chains, thus making zkSync EVM-compatible. The zksolc compiler used on Era generates bytecode with optimizations in order to make operations more amenable to proof generation, using LLVM as an intermediate representation before executing zkEVM assembly code. Consequently, there are multiple differences from Ethereum, with many EVM opcodes having modified implementation rules. zkSync Era is the first EVM-compatible chain to implement native account abstraction, which is a system of smart-contract-based accounts with arbitrary logic, first introduced in EIP-4337. Thus, every user account on zkSync Era can utilize smart accounts with their existing externally owned account (EOA).

Matter labs also provides an open framework for deploying additional modular chains similar to zkSync Era called the ZK stack [69]. A modular chain from the ZK stack, called a hyperchain, runs a separate instance of the zkSync zkEVM and settles transactions on Ethereum's Layer 1. Hyperchains are linked via Hyperbridges, ensuring asset transfer capabilities. While anyone can deploy Hyperchains, ensuring trust and full interoperability requires using the zkEVM engine from the ZK Stack, which powers zkSync Era. This uniformity in ZKP circuits across Hyperchains guarantees inherited security from Layer 1 without additional trust assumptions. Hyperchains implement a modular approach, allowing developers to choose or create their own blockchain

components, except for the zkEVM core. Various options for sequencing transactions are available, ranging from centralized to decentralized sequencers, and even external protocols for customizing Hyperchain sequencing. Each Hyperchain can also manage its data availability (DA) policy, for example a "Validium" architecture which stores state data off-chain rather than posting the calldata to L1, providing flexibility tailored to specific needs.

Polygon zkEVM [70], built by Polygon Labs, aims to offer full EVM-equivalency, with no separate compiler. Thus, ZK proving circuits verify most EVM opcodes as they are, with a few carrying minor differences that do not impact the developer experience. Because of this inherent equivalency and support of EVM opcodes, developers can deploy their existing L1 smart contracts directly to the Polygon zkEVM rollup, with no necessary tweaks. Much like the zkSync stack, Polygon has the Chain Development Kit (CDK) [71], which allows developers to deploy application-specific chains as validiums using the Polygon zkEVM.

Scroll [72] represents another approach within the zkEVM landscape, focusing on EVM compatibility with necessary adaptations for zero-knowledge proofs. Scroll's zkEVM modifies certain EVM opcode behaviors to fit the ZK-proof framework while maintaining the ability for developers to write and deploy Solidity contracts, with no custom compiler. Although these modifications alter how some operations are handled compared to Ethereum, they are clearly documented, ensuring that developers can account for these changes during smart contract development. The tailored modifications aim to preserve the core experience of Ethereum smart contract interaction within the constraints of a ZK rollup environment.

Linea's zkEVM [73], developed by ConsenSys for the Linea L2, closely mirrors the Polygon zkEVM, providing an EVM-equivalent experience without requiring a custom compiler. Supporting Solidity compilers, Linea enables developers to use well-known Ethereum tools like Hardhat and Foundry seamlessly. This compatibility eases the devel-

oper transition to Linea, with minor considerations for Solidity version compatibility. Additionally, Linea integrates the Canonical Message Service, a system allowing arbitrary data transfer between Linea and other networks, enhancing cross-chain communication and utility.

StarkNet [29], built by StarkWare, stands out as a unique ZK-rollup, fundamentally distinguished by its use of STARKs (Scalable Transparent ARguments of Knowledge), which offer quantum resistance and do not require a trusted setup, over SNARKs. Unlike other ZK-rollups, StarkNet utilizes Cairo – a specialized programming language – instead of Solidity. Cairo programs are compiled into Sierra, a safe intermediate representation, and subsequently into Cairo assembly (Casm) for execution by the StarkNet OS virtual machine. Use of the Cairo programming language and this two-step compilation process is necessary to bridge the gap between smart contract execution and the polynomial constraints of STARK proofs, which in turn validate StarkNet's block execution. Like zkSync, StarkNet's native account abstraction (AA) sets it apart by making all accounts ERC-4337 smart accounts with no externally owned accounts (EOAs) [74].

StarkEx [75], also developed by StarkWare, is a specialized Layer 2 scalability service utilizing STARK proofs for high-throughput, low-latency applications on Ethereum. Unlike StarkNet, StarkEx is not a standalone blockchain, but a service specifically tailored for certain use cases like decentralized exchanges (DEXs) and NFT platforms. It allows these applications to define their own logic off-chain and post transactions to the service, which then generates STARK proofs attesting to the validity of transaction batches. These proofs are submitted and verified on L1. StarkEx also offers various data availability modes - ZK-Rollup, Validium, and Volition. This flexibility allows applications to optimize with any mix of on-chain and off-chain components.

Aztec Network [76] is a Layer 2 rollup which focuses on privacy preservation through the Noir programming language. Noir, a Rust-based DSL for building ZK applications, simplifies the creation of ZKPs by abstracting the cryptographic process while retaining

the robustness of circuit-building languages [77]. Smart contracts in Aztec, utilizing Noir, can have both public and private elements. These contracts are defined as sets of functions, both public and private, written as Noir circuits. Each function, representing a zk-SNARK verification key, operates on the contract's public and private state. Noir's compilation process is unique: it doesn't compile directly to circuits but to an Abstract Circuit Intermediate Representation (ACIR), which can then be compiled into an arithmetic circuit or R1CS, depending on the proving system being targeted [39]. In Aztec, the sequencer aggregates transactions into a block, generates state update proofs, and posts them to the Ethereum rollup contract. This architecture, while similar to other Layer 2 networks, differs notably in its handling of private state. The private execution environment within Aztec safeguards sensitive operations and data, ensuring that private information remains confidential [76]. This architecture is outlined in Figure 10.

C. Blockchain Interoperability

1) Motivation and Definition

With a highly fragmented landscape of different blockchain technologies, including Layer 1 chains and additional modular layers built atop them, there has arisen the need for seamless composability among blockchains. For example, a significant pain point in blockchain user experience is the struggle of trying to bridge tokenized assets from one blockchain to another. This problem extends past cross-chain transactions and financial asset liquidity, as it also concerns general message passing and data storage across fragmented blockchain networks. Even for Layer 2 rollups built atop the same Layer 1, each rollup's state is a separate data moat, which results in the same fragmentation problem even in the case of shared transaction finality and consensus security. Cross-chain composability, also known as blockchain interoperability, has many innovative solutions, including those that utilize ZKP for succinct verifiable computation.
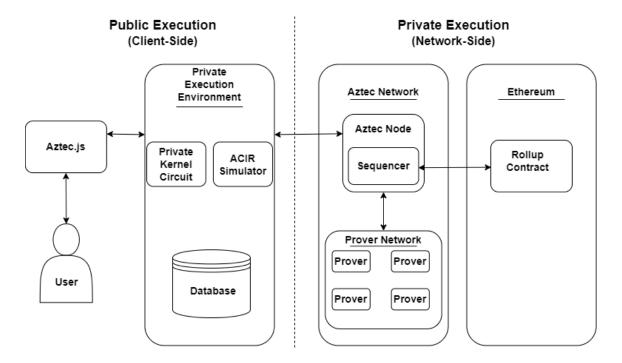
Public Execution
(Client-Side)

Private Execution
(Network-Side)

Fig. 10: Aztec's high-level network architecture, redrawn based on [76]

2) Methodology

Zero-knowledge proofs are used to verify the occurrence of a state change or block execution on one chain to another chain. This property allows protocol developers to coordinate application logic across multiple blockchains, letting users instantly transact and pass data between different networks. Typically, there is some middleware that generates the validity proofs to be verified on the receiving blockchain, where a smart contract will verify the proof and execute corresponding application logic, according to the application's specification. For example, asset bridges lock up tokens on one chain and mint them on another, allowing users to transfer liquidity between chains. Further use cases enable cross-chain DAO voting, Non-Fungible-Tokens (NFTs), and more.

3) Applications

The zkBridge protocol [78], originally published as academic research and later implemented by Polyhedra [79], operates through a modular design that separates application-specific logic from the core functionality of relaying block headers. This core functionality is provided by a block header relay network, which is trusted only for liveness. This network relays block headers of one blockchain along with zk-SNARK correctness proofs to an updater contract on another blockchain. The updater contract is responsible for maintaining a list of recent block headers from the sender chain, verifying proofs submitted by relay nodes, and updating the list accordingly. On the receiver blockchain, the updater contract provides an application-agnostic API that enables application smart contracts to obtain the latest block headers of the sender blockchain. This enables them to build application-specific logic on top of this information. Applications utilizing zkBridge generally deploy a pair of contracts: a sender contract on blockchain 1 and a receiver contract on blockchain 2. The receiver contract can call the updater contract to access block headers of blockchain 1, which it can then use to execute application-specific tasks.

Telepathy [80] is an interoperability protocol that allows arbitrary message passing between Ethereum and other chains. For developers that want to send a cross-chain transaction, they call the Telepathy router contract on Ethereum and must wait until the transaction reaches finality. To verify that a block has been finalized, an off-chain component called the Telepathy operator utilizes a zk-SNARK that proves the block header has signatures from a large percentage Ethereum validators. This proof is passed to the Telepathy light client contract on the destination chain, which verifies proofs and provides access to Ethereum's block headers. The Telepathy relayer accesses that light client data and generates a Merkle proof on the block to verify that the transaction exists and reached finality on Ethereum. This Merkle proof is passed to the Telepathy receiver contract, which verifies it and relays the smart contract call to the receiving contract on
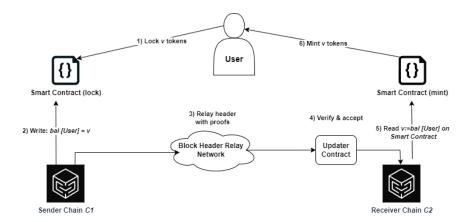
Fig. 11: zkBridge architecture when used for a cross-chain token transfer, redrawn based on [78]

the destination chain, which executes the corresponding application logic as specified by the developer.

D. Blockchain Storage

1) Motivation and Definition

Blockchain storage is essentially a way to save data in a decentralized network using the properties of the blockchain. In order to protect the saved data, blockchain uses data structures such as Merkel trees and Merkel Patricia trees. What is special about these data structures is that specific proofs can be constructed to show that a particular piece of data is contained in the structure and that once a single piece of data is changed, the entire structure changes drastically as well. This essential property of blockchain ensures data integrity and invariance. However, as the quantity of data contained in a proof rises, so does the size of the proof. As a result, the cost of validating such proofs on the chain rises, rendering ordinary inclusion proofs economically unsustainable in most circumstances. A more scalable option is now preferred, whereby the blockchain

does not have to store large amounts of data but only smaller references to data stored in off-chain platforms. Zero-knowledge proofs make this possible: data and computations can be stored off-chain, and ZK Proofs can be used to communicate a summary of these operations to the main chain concisely, efficiently, and without trust.

2) Methodology

Zero-knowledge proofs could be used to minimize the cost of activities associated with verifying the inclusion of data in massive datasets and to validate that the procedure was completed correctly. The prover carries out the necessary calculation and generates proof that proves its correctness. The verifier's job is to validate the proof's validity without redoing the whole analysis. Because of this feature, the prover only requires access to a subset of the data, such as some nodes of a Merkle tree, rather than the entire dataset. This paradigm shift is crucial because it offers a practical solution to lower the costs of employing proofs of inclusion in smart contracts, particularly when massive datasets are involved.

3) Applications

To prove the data is indeed stored in the blockchain, Herodotus [81] gives a new method called Storage Proofs to enable on-chain data access. It is essentially an on-chain accumulator that uses cryptographic means to improve access and verification of historical data on the Ether blockchain. The solution utilizes StarkWare's STARK proofs [82]. This allows users to validate data from any point on the Ethernet blockchain without the need for a third party. It combines proofs of inclusion (for verifying the existence of data) and proofs of computation (for proving the execution of multi-step workflows). These proofs are essential for verifying the integrity and correctness of one or more components of a large data set, thus ensuring that the data has not only been stored, but also remains untampered with and accurate over time.

Filecoin [83] has deployed a considerable zk-SNARK network, which uses the unused hard drive space of users around the world to store files. Using zk-SNARKs to generate

the proofs, the resulting proofs are small, and the verification process is swift (and thus, cheap). For example, proofs that typically would require hundreds of kilobytes to verify can instead be compressed to just 192 bytes with zk-SNARKs. This significant reduction in proof size not only accelerates the verification process but also reduces the computational and financial costs associated with it, making blockchain storage both more scalable and accessible.

E. Smart Contract/Transaction Privacy

1) Motivation and Definition

While creating entirely new blockchain networks or rollups serves as a solution for privacy demands, there are also ZKP-based approaches which offer selective privacy for certain decentralized applications (dApps) and transactions on existing, transparent blockchains such as Ethereum. These applications could be utilized in cases where users want to make certain actions or information private, but continue to use an existing blockchain network. These applications exist on the smart contract execution layer of a blockchain, where dApps are designed to utilize ZKP to enable private transactions in certain contexts, such as transaction mixing, for example. This application domain can be defined as Smart Contract/Transaction Privacy, which mostly benefits from the privacy prong of ZK.

2) Methodology

The core methodology in ZKP smart contract privacy centers around on-chain proof verification within a smart contract framework. Proof generation typically occurs off-chain due to its computationally intensive nature and the need for privacy in computation or the processing of sensitive data. Once a proof is generated, it is submitted to the blockchain. The contract assesses the proof against the protocol's predefined rules, and then acts accordingly if deemed valid, which could involve updating the blockchain's state, executing transactions, or any other protocol-specific actions.

3) Applications

Tornado Cash [42] employs zero-knowledge proofs for enabling transaction privacy and mixing on Ethereum. Its core mechanism revolves around depositing Ether into a smart contract and withdrawing it in a manner that severs the link between the source and destination. The protocol utilizes zk-SNARKS to prove the legitimacy of withdrawals without revealing the original deposit's details. Users deposit Ether, generating a cryptographic commitment added to a Merkle tree within the contract. For withdrawal, a user generates a zk-SNARK proof that they own a leaf in this tree without revealing which one. This proof, once verified by the smart contract, allows the withdrawal of Ether to a new address, ensuring that the transaction's privacy is maintained by obscuring the link between the deposit and withdrawal addresses. This method effectively creates a privacy layer, allowing users to transact anonymously within the public Ethereum blockchain.

Due to its rampant use for illicit financial transactions such as money laundering, Tornado Cash has been heavily sanctioned and banned in countries like the US. This has given rise to the research problem of regulatory-compliant privacy solutions, which has been implemented in the Privacy Pools project, as delineated in [84]. Privacy Pools extends the usage of zk-SNARKs in other privacy solutions like Zcash and Tornado Cash. Rather than simply generating a ZKP to prove that a withdrawal attempt is linked to a specific deposit, users prove membership in a specific association set, which is a collection of transaction references, represented as a Merkle tree, from which a user's funds could have originated. Users define their set by providing the Merkle root as a public input. When withdrawing funds, the user does not directly prove a link to a specific transaction. Instead, they utilize zk-SNARKs to generate proofs that validate their funds' origin from within this predefined, public set of transactions. This approach allows users to demonstrate a connection to a pool of transactions without revealing the exact source, maintaining privacy while introducing an element of transparency.

The use of association sets in Privacy Pools addresses the challenge of aligning

transaction privacy with regulatory compliance. By proving membership in a specific association set, users demonstrate that their funds originate from a group of transactions that are not flagged as high-risk or associated with illicit activities.

Penumbra [85] is a fully private proof-of-stake network and decentralized exchange within the Cosmos ecosystem, offering a unique approach to transaction privacy and cross-blockchain interoperability using zero-knowledge proofs. Penumbra connects to the Cosmos blockchain ecosystem through IBC (the inter-blockchain communication protocol) and maintains all value in a multi-asset shielded pool, inspired by the Zcash Sapling design. This allows for private transactions in any IBC-compatible asset. All transactions on Penumbra are private by default, enabled by zk-SNARKS which validate the correctness and legitimacy of transactions but shield the sender, receiver, and amount. Penumbra's decentralized exchange, called ZSwap, supports sealed-bid batch auctions and concentrated liquidity similar to Uniswap v3. This architecture prevents frontrunning and ensures that only the net flow across a pair of assets is revealed in each block. For cryptographic primitives, Penumbra utilizes BLS12-377 as the proving curve, which is compatible with the Groth16 proving system used. Penumbra may change to PLONK in the future.

A novel privacy problem exists within the space of on-chain decentralized autonomous organizations (DAOs), which are blockchain-based governance systems powered by smart contracts. Because of the transparent nature of public blockchains, DAO treasuries are completely public, which is a big issue for certain types of auctions. In a blog and research project by Griffin Dunaif and Dan Boneh, the authors design a system for a private DAO protocol utilizing ZKP [86]. The protocol utilizes a master contract deployed on the Ethereum network to manage multiple DAOs. This contract allows anyone to send funds to a DAO, but only the DAO manager can withdraw them.

The life cycle of a DAO in this system comprises three steps: creation, deposit, and withdrawal. During DAO creation, the manager establishes the DAO without any on-

chain transactions, and posts a Schnorr public key on the DAO website. To contribute funds to the DAO, users compute the Merkle tree leaf using the DAO public key, in which the deposit is recorded in the master contract. The DAO manager is able to privately monitor deposits and keep track of the DAO treasury using their secret key. When withdrawing funds, the DAO manager again uses this secret key within a SNARK proof to show that a specific batch of deposit leaves in the Merkle tree belong to the DAO, without publicly revealing the secret key. After verifying the proof, the master contract releases the withdrawal amount to the DAO manager.

We have seen niche-specific privacy dApps, but there also exist entirely private smart contract frameworks for private blockchain applications. One example is Hawk [26], which uses off-chain computation to conceal private portions of a smart contract. In Hawk, the programmer writes a smart contract with defined public and private components. The Hawk compiler will subsequently split the computation into pieces, where the private portion of the contract ϕ-priv is executed off-chain and handles sensitive data and computations. This off-chain execution is managed by a trusted party, known as the manager, who can see the users' inputs and is expected to not disclose them. The manager's role is to perform the private computations and generate a ZK-SNARK attesting to the correctness of these computations without revealing any sensitive data inputs. The proofs are then verified on-chain, ensuring the integrity of the private computations while keeping them hidden from the public blockchain. The public portion of the contract ϕ-pub, which executes on the blockchain, handles non-sensitive operations and provides transparency where necessary, but it does not deal directly with private data or currency transactions.

Mina, the previously mentioned Layer-1 blockchain, also provides a privacy-preserving smart contract framework called zkApps [87]. Mina zkApps are developed using the o1js TypeScript library and the Mina zkApp CLI, and comprise of two main components: a smart contract written with o1js, and a user interface (UI) for interaction. Upon zkApp

interaction, the smart contract's code is executed locally in the user's web browser, where it generates a ZK-SNARK proof. This setup allows users to input data into the zkApp, which could be either private or public. Private data remains unseen by the blockchain, while public data may be stored on-chain or off-chain, depending on the zkApp's design. The prover function within the smart contract generates the SNARK proofs, maintaining user privacy by ensuring that sensitive data is processed locally and not disclosed on the blockchain. Once a user decides to submit a transaction to the chain, the transaction, containing the ZKP and associated state updates, is sent to the Mina network. The network verifies that the proof meets all constraints defined in the prover function. The state of a zkApp can be either on-chain or off-chain. On-chain state is stored directly on the Mina blockchain and offers limited storage space, while off-chain state refers to larger data stored elsewhere, like in external storage systems such as IPFS. In scenarios where off-chain storage is used, the zkApp updates an on-chain Merkle tree root of some fully off-chain Merkle tree. This method ensures that the integrity of both the proof and the associated account updates is maintained, allowing the Mina network to confirm the validity of the zkApp transactions and state changes.

F. Blockchain-Based Proof of Identity

1) Motivation and Definition

Because of the aforementioned qualities of ZKP, there exists the unique use-case of a user proving group membership or identity without revealing any sensitive information about their identity, thus marrying the imperatives of authentication and privacy. While this can be implemented outside of the blockchain context (as will be covered in the "Proof of Identity" section below), there exist many identity-proving applications using blockchain smart contracts and execution layers. This application domain can be defined as Blockchain-Based Proof of Identity, which takes advantage of privacy-preserving ZKP.

2) Methodology

The core methodology behind blockchain-based proof of identity systems is the use of zero-knowledge proofs to enable users to prove membership in certain identity groups or ownership of credentials without revealing the actual identity information. This is achieved through cryptographic commitments to a user's information while keeping it secret. A privacy-preserving ZK identity proof can be easily verified by an on-chain smart contract, and this can act as a private credential system secured by the blockchain and open doors to a multitude of decentralized applications, without sensitive information compromise.

3) Applications

Semaphore is a framework for zero-knowledge signaling on Ethereum that allows users to broadcast support for an arbitrary string, without revealing their identity to anyone besides being approved to do so [88]. It uses Pedersen commitments to hide user identities in an incremental Merkle tree stored on-chain. Users generate zk-SNARK proofs showing that:

- Their Pedersen commitment identity is valid, by proving it exists as a leaf in the incremental Merkle tree using the Merkle path.
- They know the secrets behind the Pedersen commitment.
- The unique, pseudorandomly derived nullifier has not been used before, preventing double-spending.
- The signal is properly authorized using an EdDSA signature verification within the circuit.

A smart contract handles logic and state management, such as adding identity commitments to the Merkle tree, updating the nullifier map, and adding successful signals to the signal map. This enables fully on-chain privacy applications with proof verification, such as anonymous voting and reputation systems [88].

Semaphore can serve as a verifiable credential protocol at the base level, as it allows developers to create identities, identity groups, and use identity commitments to prove group membership. An example of a prominent project that utilizes Semaphore for verifiable credentials on Ethereum is World ID, which is the proof-of-personhood verification system for the WorldCoin protocol [89]. When a user creates a unique World ID from their biometric data, their ID is enrolled in a group of verified World ID users. World ID's can then be safely verified without revealing the World ID public key.

The Galxe protocol is a self-sovereign identity service centered around verifiable credentials [90]. It addresses the digital identity multiplicity problem by embedding numerous identity commitments into a single user credential, enabling holders to connect identities across platforms privately. Credential holders can use ZKP to selectively prove requisite information of their identity, while maintaining a pseudonym. Like Semaphore, Galxe constructs identity commitments by computing the Poseidon hash of the private secret identity and a private internal nullifier. This can then be distinguished in a zero-knowledge proof, which is verified by an on-chain smart contract and attests to the user's identity. The internal and external nullifiers generate deterministic nullifiers per verification to prevent double-spending. The current verification stack of the Galxe protocol is BabyZK, which uses the BN254 curve, Groth16 proofs and Poseidon commitments. Use cases of the Galxe protocol could include sybil-resistant reputation systems, access control, achievement aggregation, and personal data markets with privacy.

Aleo, a Layer 1 privacy blockchain mentioned in this paper above, boasts its own ZK identity verification protocol called zPass [91]. zPass is a straightforward implementation due to the nature of enshrined zero-knowledge proofs and private execution available in Aleo. However, zPass also enables users to obtain anonymous credentials from existing identity documents, facilitating real-world adoption without necessitating changes to the protocol. Like Galxe protocol, it allows for selective attribute disclosure, enabling users to prove identity assertions across multiple credentials.

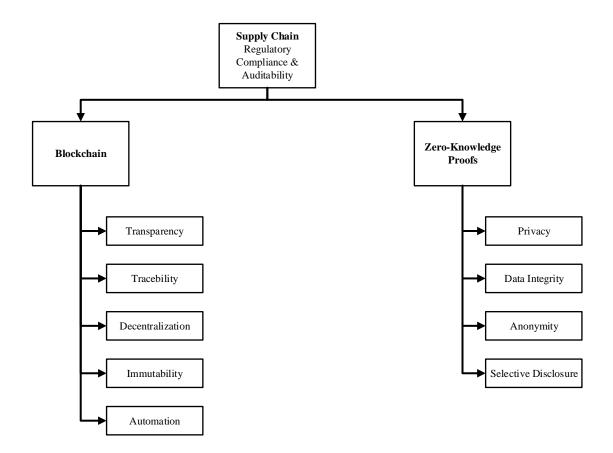*G. Supply Chain/Enterprise Blockchain Privacy*

Fig. 12: Supply Chain features fulfilled by Blockchain and Zero-Knowledge Proofs

1) Motivation and Definition

Supply chains and global enterprises are increasingly recognizing the transformative potential of blockchain technology for enhancing transparency, provenance, immutability, and accountability. However, this shift towards blockchain adoption in supply chain comes with significant privacy concerns. In a typical supply chain, stakeholders, including suppliers, manufacturers, distributors, and retailers, share sensitive data such as

pricing, inventory levels, and production schedules. The public and immutable nature of open blockchains could expose proprietary or sensitive information to competitors or the public. This trade-off between the need for transparency and privacy requirements has led to the exploration of privacy-enhancing technologies within blockchain frameworks for supply chain and enterprise applications.

2) Methodology

There are several strategic approaches for implementing privacy in supply chain and enterprise blockchains. Firstly, the adoption of permissioned blockchain architectures enables the creation of a controlled ecosystem where access is granted only to verified participants. This selective visibility is crucial for maintaining a competitive edge and for ensuring that relevant data is shared only among trusted stakeholders. Furthermore, the integration of ZKPs addresses the necessity for privacy by allowing participants to verify the authenticity and compliance of products without revealing detailed process histories or proprietary information. This technology is instrumental in convincing end consumers of product safety and quality, even in the absence of full process transparency. To tackle the intricate and global nature of modern supply chains, data segmentation and encryption techniques are employed to protect business-sensitive information. By only storing cryptographic hashes of the actual data on-chain and keeping the detailed records off-chain or encrypted, these methodologies significantly reduce the risk of exposing trade secrets and sensitive business strategies. Moreover, the use of smart contracts for privacy enforcement plays a pivotal role in automating compliance and access control based on pre-defined rules. This approach not only streamlines operations but also ensures that data disclosure is strictly governed by necessity and consent in the supply chain scenario. Therefore, incorporating these enhanced privacy measures can achieve the goal of protecting end consumer interests and maintaining mutual trust among supply chain participants, as depicted in Figure 12.

3) Applications

qedit [92] provides privacy-enhancing technology for enterprise blockchains, enabling secure collaboration and data sharing among parties without revealing sensitive information. It utilizes ZKPs to ensure that transactions are valid while keeping the transaction content private. It's designed for enterprises looking to monetize data assets safely, enhance business analytics, and derive actionable insights in a secure manner. The platform is cloud-hosted, highly scalable, and integrates easily with existing database systems for quick deployment. qedit features a configurable dashboard, advanced reporting, and real-time notifications to provide businesses with critical intelligence efficiently. It is aimed at transforming the way companies collaborate on sensitive data, ensuring privacy while enabling data-driven decisions.

zk-BeSC [93] introduces a blockchain-based framework for supply chain management that utilizes polynomial ZKPs to ensure privacy during transactions. The framework is designed to enable confidential transactions among supply chain participants, preserving the privacy of sensitive data while maintaining the traceability and immutability features of blockchain technology. It leverages advanced cryptographic techniques, including homomorphic encryption and elliptic curve pairings, to prove knowledge of polynomials without disclosing them. Implemented on the Ethereum testnet with a web3 application, zk-BeSC demonstrates efficient proof performance and reduced gas consumption for verification, addressing key privacy concerns in supply chain management.

Sahai *et al.* [94] present a blockchain-based solution for improving privacy and traceability in supply chains. This approach leverages Hyperledger Fabric and employs cryptographic tools like ZKPs to protect sensitive business data while ensuring the ability to trace product provenance and contamination. The model supports operations such as product entry, exit, transfer, merge, split, and processing within the supply chain, enabling efficient and private tracing of products from origin to consumer.

H. Proof of Reserves

1) Motivation and Definition

It has always been important in financial markets for companies to demonstrate their reserve assets to prove their solvency to savers. The global financial system often operates in an undercollateralized and highly opaque manner, relying heavily on trust in a central entity (either the system itself or a third-party certifier), but this can create fraud risks, mismanagement, and privacy breaches. Zero-knowledge proof of Reserves (ZK-PoR) (such as [95]) provides a trustless mechanism to verify reserves, enhancing trust and security in decentralized financial systems without sacrificing privacy. This innovation is particularly important for cryptocurrency exchanges and wallets, as users need verifiable assurance that their assets are being held securely without exposing those assets to potential threats. Specific to the scenario of a virtual currency exchange, the proof generated by this method can ensure that the verifier obtains proof of the exchange's repayment ability without knowing the specific amount of the exchange's reserves, the identity of the individual account holder, or any transaction details. Proof, thereby ensuring transparency on reserve adequacy while maintaining privacy and security.

2) Methodology

The methodology behind ZK-PoR involves several key steps to ensure secure and private verification of assets. Initially, the entity holding the reserve constructs a commitment to the asset without revealing its details, using cryptography to generate proof of possession. The proof is rooted in a zero-knowledge proof algorithm and is then transmitted to the verifier. Validators use the same cryptographic algorithm to verify proofs without knowing the actual reserves, the identities of asset holders, or transaction details. The process often employs complex mathematical structures such as homomorphic encryption, elliptic curve encryption, and Merkle trees to ensure the integrity and confidentiality of the proof. The trust established by this approach stems from

mathematical proof rather than the reputation or authority of the entity.

3) Applications

ZK-PoR has numerous applications in the fields of blockchain and financial technology, especially in enhancing privacy and trust. In the financial domain, particularly within cryptocurrency exchanges, proving solvency without compromising sensitive information confidentiality is a critical concern.

Provisions [96] offers a groundbreaking solution to this challenge by employing ZK-PoR. This technology enables a Bitcoin exchange, or any cryptocurrency platform, to transparently demonstrate that it possesses sufficient funds to cover all its obligations to users without disclosing the exact amounts held or the identities of the account holders. Allowing an exchange to prove its liquidity addresses the common concern of potential insolvency. Furthermore, this approach minimizes the risk of sensitive financial data being exposed, which could be detrimental in terms of privacy breaches or malicious exploitation.

Another application, Proven [97], leverages ZK-PoR to provide a decentralized platform that enables companies and financial institutions to verifiably prove their liquidity or asset reserves without revealing specific asset values or compromising the confidentiality of their operations. These applications highlight ZK-PoR's versatility in solving the twin challenges of transparency and privacy in digital finance, providing exchanges and institutions with a powerful solution to build trust with users and regulators while protecting sensitive financial data.

V. NON-BLOCKCHAIN APPLICATIONS

A. Proof of Identity

1) Motivation and Definition

Although we have seen many blockchain-based proof of identity protocols, identity proving as a practical application of ZKP extends past the blockchain domain as well.

Use Case	Projects
Proof of Identity	Zero-Knowledge Proofs of Identity, zk-creds: Flexible Anonymous Credentials from zk-SNARKs and Existing Identity Infrastructure
ML/AI	zkCNN, vCNN, zkDL, Kaizen, zkLLM, Ezkl, Modulus, Giza, TensorPlonk
Other Applications	Non-interactive Zero-Knowledge Arguments for Voting, PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations, Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets, VeeDo, ZKP2P

TABLE VII: Overview of non-blockchain applications of Zero-Knowledge Proofs

Fundamentally, this methodology is derived from the ability to prove a statement about identity, specifically membership in a set or credential verification, without revealing any sensitive information about the identity. As we saw in the blockchain domain, the space of privacy-preserving verifiable credentials is a popular result of these properties. This also extends to non-decentralized technologies, as we will see in the following section. This application domain can be defined as Proof of Identity, utilizing the privacy properties of zero-knowledge verifiable computation.

2) Methodology

These identity proving techniques utilize ZKP to validate a user's identity or membership in a group without revealing any specific, sensitive details about the identity itself. Compared to other application domains, this methodology is far more pure in the sense that the only unifying factor among applications is the ZKP itself; a proof is generated by a prover on some identity credential and given to an identity verifier, who does not get any identity information other than the proof itself.

3) Applications

One of the earliest papers on ZKP, first published in 1988, presents a novel identification scheme based on zero-knowledge proofs, providing a more efficient alternative to RSA-based schemes. [98]. In identification schemes, entity *A* proves their identity to

entity *B* using some constant *S* in the form of a value or physical card, without enabling entity *B* to then falsify themselves as *A* afterwards. Traditional identification schemes utilize encryption and/or hashing along with credentials such as digital passwords, PINs, credit card chips, etc. This paper proposes a practical scheme where no sophisticated adversary is capable of cooperating with a dishonest verifier *B* to produce a falsified credential and pretend to be *A*. The methodology involves interactive proofs where an entity can demonstrate their identity by proving they know a secret key of their credentials without having to reveal the secret; simply submitting a proof of their knowledge of the secret is sufficient, with the secret acting as a digital signature unique to each individual. The paper outlines a directory-less scheme, meaning there is no need for a central repository of public keys or identities. The paper also proposes that such a scheme could be implemented in hypothetical "smart cards", which could act as physical credentials that generate zero-knowledge identity proofs using microprocessors to facilitate everyday identity verification.

The zk-creds protocol leverages zero-knowledge proofs (specifically zk-SNARKs) to convert existing identity documents into anonymous credentials, thereby eliminating the need for credential issuers to hold signing keys [99]. This system contrasts with traditional methods where issuers sign credentials and identity documents for validation. By integrating with existing identity infrastructures like government-issued IDs or university diplomas, zk-creds transforms these traditional credentials into digital, anonymous, yet verifiable formats.

B. Machine Learning

1) Motivation and Definition

In a world where AI-generated material increasingly mimics human-created information, the potential use of zero-knowledge cryptography could assist us in determining that a specific piece of content was produced by applying a specific model to a given

input. If a zero-knowledge circuit representation is built for them, this could give a technique for verifying outputs from large language models like GPT-4 [100], text-to-image models, or any other models. The zero-knowledge quality of these proofs allows us to hide sections of the input or the model if necessary. A good example is enabling users to view the model's inference results without knowing the details of the model and prove that this result really comes from a specific model and input.

Zero-knowledge machine learning (ZKML) is a means to protect data privacy during model training and inference. They enable a data owner or a model owner to demonstrate the accuracy of a computation (such as the prediction of a machine learning model) without exposing any information about the data or the computation itself. This is especially effective in circumstances involving sensitive data.

2) Methodology

In a typical machine learning situation, an application service provider (the prover) wants to provide a machine learning model it owns to the user (the verifier) while keeping the model private. Provers can use zero-knowledge proofs to show that they have indeed performed a computation using a particular model without exposing the model or the computation process itself.

3) Applications

ZKPs are commonly employed in machine learning, notably privacy-preserving and federated learning. service providers (provers) can use ZKP to prove the correctness of their model predictions without revealing the model itself to prevent model theft. The current state of the art in zero-knowledge systems coupled with performant hardware is still a few orders of magnitude shy of proving something as massive as currently available large language models (LLMs), but there has been considerable progress in establishing proofs of smaller models.

In verifying machine learning model predictions, vCNN [101] uses commit-and-prove to combine the typical quadratic arithmetic program (QAP) with the polynomial QAP

in pairing-based zero-knowledge proofs. It supports convolutional neural networks and validates them using polynomial multiplications.

However, by presenting a novel sumcheck technique, zkCNN [102] could prove fast Fourier transformations and convolutions in a linear prover time. It is verified that the convolutions directly using the sumcheck protocol and zkCNN are 34×faster than vCNN.

zkDL [103] is an innovative approach to meet the need for zero-knowledge proofs in deep learning training. The main challenge it addresses is verifying the nonlinear computations inherent in neural networks, especially the ReLU activation function and its backpropagation. By introducing zkReLU, zkDL can efficiently handle these non-arithmetic operations without resorting to polynomial approximations, which are usually computationally expensive and less accurate. The FAC4DNN utilized by zkDL is an arithmetic circuit that aggregates proofs from different layers and training steps. This design bypasses traditional sequential proof generation and greatly reduces computational and communication overheads. zkDL also achieves full compatibility with tensor structures and supports large-scale neural networks, with proofs generated in less than a second for each batch update for networks with up to 10 million parameters.

The research by Sanjam Garg et al. [104] explores the practical application of zero-knowledge proofs in verifying the training of machine learning models. It focuses on experimental settings to verify the feasibility of such proofs, taking into account computational complexity and scalability issues. Their work emphasizes the need to balance strong security guarantees with practical overhead, ensuring that the proofs are concise and the prover's workload is manageable. By experimenting with various configurations and optimization techniques, it provides insights into making zero-knowledge proofs applicable to real-world deep learning applications, which is very suitable as a reference.

Another recent study, Kaizen [105], proposed a zero-knowledge proof system designed for deep neural networks (DNNs). It ensures that the submitted model is correctly trained on the submitted dataset without leaking any other information. Kaizen adopts a sum-

check-based proof system optimized for the gradient descent algorithm and recursively combines these proofs in multiple iterations. This recursive combination ensures that the proof size and verifier time are independent of the number of iterations, making it highly scalable. Kaizen has the ability to handle large models such as VGG-11, and is more practical than general recursive proofs, significantly reducing prover runtime and memory overhead.

In the study of the reasoning process, zkLLM [106] focuses on providing zero-knowledge proofs for large language models, ensuring that the reasoning results are verifiable without revealing the underlying model or data. This is especially important for applications that require strong privacy protection, such as medical or financial fields where data confidentiality is critical. zkLLM uses advanced encryption technology to efficiently generate and verify proofs, and maintains the privacy of model parameters and input data through cryptographic means, enhancing trust in deployed AI systems.

In addition to the issue of the trustworthiness of its predictions, the model's reliance on opaque data sources has become a new challenge. People frequently desire to keep the inputs and parameters of machine learning models private and unknown to the general public. Because the input data may contain sensitive information such as personal finances or biometrics, and the model parameters may also hold critical secrets. To specify provers and verifiers, many ZKML tools represented by the ezkl library [107] can implement higher-level descriptions of machine learning models or other computational graph programs. A prover can demonstrate that a particular output is produced by running a specific neural network on a specific data set.

Modulus Labs [108] is also developing a zero-knowledge machine learning solution, enabling trustful integration of AI outputs into blockchain systems without revealing sensitive data or model details. This allows developers to retain ownership and control over their AI models rather than relying on centralized platforms to host and manage their models. And their new zero-knowledge prover, Remainder [109], is a fast AI prover

for AI inference. It is based on a verifiable decision forest inference circuit using GKR protocol [110].

Giza [111] is a machine learning platform built on StarkNet that can be used to deploy and extend machine learning models, as well as solve the interoperability issues faced in the use of cloud-based machine learning models, performance, and transparency issues. It uses the ONNX open format to improve interoperability. Through a series of common operators and file formats it defines, developers can freely use TensorFlow, PyTorch, Scikit-Learn, and other frameworks and tools. Since StarkNet runs on the Layer2 network, it can enable any decentralized application to achieve unlimited computing scale without affecting the composability and security of Ethereum. Therefore, you don't have to worry about load and architecture issues when using Giza, and can concentrate on model development and iterate. Another benefit of being based on StarkNet is that most functions are managed by the blockchain, which makes it easier to monitor, track, and manage the model, greatly improving transparency.

C. Other Applications

1) Motivation and Definition

Zero-knowledge proofs (ZKPs) have a broad range of applications extending beyond blockchain technology. Their unique ability to verify the accuracy of information without revealing the data itself makes them valuable in various technological fields. This section further progresses into other diverse applications of ZKPs that may not necessarily fit into our predefined categorization. These applications leverage ZKPs to address challenges in other significant areas, including image authentication, decentralized voting systems, and secure multi-party processes by ensuring data privacy and computational integrity.

2) Methodology

In particular, we focus on four different applications: 1) Image Authentication, 2) Secure Electronic Voting Systems, 3) Randomness Generation and Timelocks, and 4)

Collaborative Computations. ZKPs can be used to authenticate images without revealing the original content while allowing certain transformations like cropping or rotation. This is achieved by attaching a zero-knowledge proof to each image, certifying its integrity through any permissible transformations. Secure voting systems can employ ZKPs to validate encrypted votes while simultaneously ensuring voter privacy and integrity. This allows the verification of the correctness of each vote while maintaining the confidentiality of the voter's choice. In applications such as randomness generation and timelocks, ZKPs can provide proof of computational integrity and delay in a verifiable delay function (VDF). These systems combine a delay function with the zkSTARK protocol in order to create a VDF that is slow to compute but fast to verify. Other applications extend the use of zk-SNARKs to collaborative environments where multiple parties jointly produce a single proof over a distributed witness. It addresses the challenge of maintaining data privacy in multi-party computations. The approach involves adapting zk-SNARKs for multi-prover protocols using secret-sharing techniques. This allows the generation of a single proof that validates a computation over data distributed among several parties without revealing individual inputs.

3) Applications

In the realm of digital media, journalism, and legal documentation, the authenticity of images is a critical concern. PhotoProof [112] addresses this by allowing images to be authenticated even after undergoing permissible transformations such as cropping or color adjustments. This application is particularly significant in legal scenarios where image evidence must remain untampered and in journalism, where the integrity of photographic evidence can be crucial in forming public opinion. PhotoProof utilizes ZKPs to ensure that any changes made to an image do not compromise its original authenticity, thereby maintaining the credibility of digital media.

Moreover, ZKPs can address the limitations of electronic voting systems, such as tampering or breaches in voter privacy. The integrity of the voting process in elections

is of utmost importance in maintaining democratic principles. The proposed system for electronic voting using non-interactive zero-knowledge arguments [113] presents a secure method to validate votes while preserving voter anonymity. This application is significant in preventing electoral fraud and ensuring a fair and transparent voting process.

Generating unbiased randomness and implementing secure timelocks is essential in blockchain systems and cryptographic protocols. VeeDo [114] leverages ZKPs in VDFs to provide a reliable source of randomness that is resistant to manipulation, enhancing unpredictability and fairness in blockchain applications. In cryptographic protocols, VeeDo's timelock feature can be utilized to secure information for a predetermined period by adding an extra layer of security to sensitive transactions or processes.

In fields like scientific research, business analytics, and healthcare, there is often a need to perform multi-party computations over shared data without revealing individual inputs. Collaborative zk-SNARKs [115] enables multiple parties to compute a joint result while ensuring that the privacy of individual institutions' data is preserved.

For instance, hospitals and research institutions might collaborate on patient data for medical research while adhering to privacy laws set up by the government. Using collaborative zk-SNARKs, they can analyze aggregate data, such as treatment effectiveness or disease trends, without revealing individual patient details. This has profound implications for collaborative research, where data privacy is paramount, and for businesses that require secure multi-party computations for joint ventures or partnerships.

Next, ZKP2P [116] emerges as an innovative bridge that integrates traditional finance with decentralized finance through a trustless and privacy-centric framework. At its core, ZKP2P employs ZKPs to verify DomainKeys Identified Mail (DKIM) signatures in payment confirmation emails. This use of ZKPs is instrumental in ensuring the authenticity of transactions while maintaining the privacy of sensitive information. The architecture of ZKP2P is characterized by its incorporation of several key components: circuits for the secure verification of transaction details, smart contracts for managing

trustless interactions, and ZK-Email Libraries essential for the generation of private proofs relating to email contents. The ZKP2P protocol is designed to be interoperable with prevalent Web2 payment systems, thereby bridging a significant gap between fiat and cryptocurrencies. By eliminating intermediaries and reducing transaction fees, ZKP2P aims to offer a more inclusive and efficient platform for crypto-fiat conversions with the potential to evolve into a global, on-chain, trustless payment network compatible with diverse financial applications in DeFi, NFTs, and gaming.

VI. CONCLUSION AND FUTURE WORK

In the rapidly evolving world of digital security and privacy, ZKPs have emerged as a revolutionary tool, offering a way to share proofs of computational integrity without revealing the computation's input. This survey explored a wide range of ZKPs' practical applications and use cases, showing their crucial role in advancing cryptographic solutions and enhancing privacy in digital interactions. By examining various use cases, from enhancing privacy in blockchain to securing verification processes, ZKPs have demonstrated their potential to meet some of the most pressing challenges in ensuring digital privacy and security. However, the journey from theoretical constructs to widely applied solutions for ZKPs is still ongoing. The survey uncovered the depth and breadth of ZKPs' applicability, highlighting the need for further research, especially in optimizing their implementation and expanding their use cases.

In concluding our survey on the applications of ZKPs, we identify some promising directions for future research to broaden their practical use and applicability. Future works could explore lightweight ZKP protocols that are feasible for devices with limited computational capabilities, enabling secure, privacy-preserving communication in the IoT landscape. Furthermore, the integration of ZKPs with ML, particularly in the context of bigger and more complex models, presents space for breakthroughs. Investigating the ways in which ZKPs can facilitate privacy-preserving computation and verification of

large machine learning models without exposing the underlying data or the model itself has the potential to transform data privacy in AI.

In the domain of Layer-2 blockchain scalability using ZKPs, future research includes improving SNARK proof generation times in order to enable universal synchronous composability among different Layer-2 rollups. Universal synchronous composability allows different rollups to access and update the same universal blockchain state by sharing a sequencer that posts state changes to Layer-1 – this would defragment the Layer-2 rollup landscape and consolidate blockchain liquidity and state data among all Layer-2 rollups. In order for this to be possible, hardware custom-built for a proving system will likely be required.

Future research could also include finding the implications of merging ZKPs into game theoretic mechanisms. Specifically, there could be more formal research on finding equilibria in privacy-preserving systems augmented by ZK. For example, placing truthful bids in auctions without revealing sensitive information about underlying assets, similar to the private DAO application seen in section IV-E3. Research could also be done on the feasability of financial price discovery in fully or partial private modes, for both auctions and exchanges. To extend from this, there is yet to be a ZK SNARK system which proves the valid execution of a private order-book exchange. This can also be extended to dark pools, which are privatized blockchain exchanges enabled by ZKPs, which can utilize the automated market maker (AMM) exchange model.

Another potential application of ZKPs, specifically within the blockchain space, could be the use of privacy-preserving proofs to mitigate the negative externalities of maximal extractable value (MEV). Some possibilities could include creating encrypted transaction mempools and relays using ZKPs.

Additional future applications and research could extend any of the application categories mentioned in this paper. There are set to be many more uses of ZKP for legal compliance in private financial applications, mitigation of AI risks such as deepfakes, and

additional privacy enhancements in financial applications. As we advance our computational techniques, explore new applications, and deepen our theoretical understanding, ZKPs stand to significantly impact how privacy and security are achieved in the digital age.

ACKNOWLEDGMENTS

REFERENCES

[1] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity and a methodology of cryptographic protocol design," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 1986, pp. 174–187.

[2] ——, "Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems," *Journal of the ACM (JACM)*, vol. 38, no. 3, pp. 690–728, 1991.

[3] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is provable in zero-knowledge," in *Advances in Cryptology—CRYPTO'88: Proceedings 8*. Springer, 1990, pp. 37–56.

[4] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again." *IACR Cryptology ePrint Archive*, vol. 2011, p. 443, 01 2011.

[5] J. Groth, "Short pairing-based non-interactive zero-knowledge arguments," in *Advances in Cryptology - ASIACRYPT 2010*, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 321–340.

[6] J. Kilian, "A note on efficient zero-knowledge proofs and arguments," in *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, 1992, pp. 723–732.

[7] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–34, 2019.

[8] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[9] A. C. Yao, "Protocols for secure computations," in *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE, 1982, pp. 160–164.

[10] O. O'Donoghue, A. A. Vazirani, D. Brindley, and E. Meinert, "Design choices and trade-offs in health care blockchain implementations: systematic review," *Journal of medical Internet research*, vol. 21, no. 5, p. e12426, 2019.

[11] M. Petkus, "Why and how zk-snark works," *arXiv preprint arXiv:1906.07221*, 2019.

[12] E. Morais, T. Koens, C. Van Wijk, and A. Koren, "A survey on zero knowledge range proofs and applications," *SN Applied Sciences*, vol. 1, pp. 1–17, 2019.

[13] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, and X. Peng, "A survey on zero-knowledge proof in blockchain," *IEEE network*, vol. 35, no. 4, pp. 198–205, 2021.

[14] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, ser. STOC '85. New York, NY, USA: Association for Computing Machinery, 1985, p. 291–304. [Online]. Available: https://doi.org/10.1145/22145.22178

[15] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "Snarks for c: Verifying program executions succinctly and in zero knowledge," in *Annual cryptology conference*. Springer, 2013, pp. 90–108.

[16] S. Setty, J. Thaler, and R. Wahby, "Customizable constraint systems for succinct arguments," Cryptology ePrint Archive, Paper 2023/552, 2023, https://eprint.iacr.org/2023/552. [Online]. Available: https://eprint.iacr.org/2023/552

[17] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 626–645.

[18] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*. Springer, 2010, pp. 177–194.

[19] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza, "Secure sampling of public parameters for succinct zero knowledge proofs," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 287–304.

[20] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.

[21] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proceedings, Part II, of the 35th Annual International Conference on Advances in Cryptology — EUROCRYPT 2016 - Volume 9666*. Springer-Verlag, 2016, p. 305–326.

[22] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for

confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*. IEEE Computer Society, 2018, pp. 315–334. [Online]. Available: https://doi.org/10.1109/SP.2018.00020

[23] Circom, "Circom 2 documentation," https://docs.circom.io/, 2024, [Accessed 12-05-2023].

[24] Polygon, "Polygon miden documentation," https://0xpolygonmiden.github.io/miden-base/introduction.html?search=setup, 2024, [Accessed 01-05-2024].

[25] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for c: Verifying program executions succinctly and in zero knowledge," Cryptology ePrint Archive, Paper 2013/507, 2013, https://eprint.iacr.org/2013/507. [Online]. Available: https://eprint.iacr.org/2013/507

[26] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 839–858.

[27] M. Labs, "Zinc," 2020, accessed: July 22, 2024. [Online]. Available: https://blog.matter-labs.io/release-of-zinc-v0-1-8d949aa9a2f2

[28] R. Z. T. Jeremy Bruestle, Paul Gafni, "Risc zero whitepaper," https://dev.risczero.com/proof-system-in-detail.pdf, 2024, [Accessed 02-22-2024].

[29] Starknet, "Cairo and sierra," 2023. [Online]. Available: https://docs.starknet.io/documentation/architecture_and_concepts/Smart_Contracts/cairo-and-sierra/

[30] "Aleo: A new platform for private applications," 2023, accessed: November 24, 2023. [Online]. Available: https://www.aleo.org/

[31] Neptune, "Announcing triton vm," 2022, accessed: July 22, 2024. [Online]. Available: https://neptune.cash/blog/announcing-tvm/

[32] Sin7Y, "Olavm," 2022, accessed: July 22, 2024. [Online]. Available: https://olavm.org/

[33] Powdr, "Powdr zkvm," 2023, accessed: July 22, 2024. [Online]. Available: https://www.powdr.org/

[34] A. Arun, S. Setty, and J. Thaler, "Jolt: SNARKs for virtual machines via lookups," Cryptology ePrint Archive, Paper 2023/1217, 2023, https://eprint.iacr.org/2023/1217. [Online]. Available: https://eprint.iacr.org/2023/1217

[35] S. Labs, "Sp1 zkvm," 2024. [Online]. Available: https://blog.succinct.xyz/introducing-sp1/

[36] D. Marin, M. Abdalla, P. Govereau, J. Groth, S. Judson, K. Sosnin, and G. Vamsi, "Nexus 1.0: Enabling verifiable computation," 2024. [Online]. Available: https://nexus.xyz/

[37] Lita, "Lita zkvm," 2023. [Online]. Available: https://www.lita.foundation/infrastructure#valida

[38] Mina, "O1js," 2021, accessed: July 22, 2024. [Online]. Available: https://docs.minaprotocol.com/zkapps/o1js

[39] Noir, "Introducing noir," 2023. [Online]. Available: https://noir-lang.org/

[40] A. Labs, "Juvix zkdsl," 2017. [Online]. Available: https://github.com/anoma/juvix

[41] N. Amin, J. Burnham, F. Garillot, R. Gennaro, C. Künzang, D. Rogozin, and C. Wong, "LURK:

Lambda, the ultimate recursive knowledge," Cryptology ePrint Archive, Paper 2023/369, 2023, https://eprint.iacr.org/2023/369. [Online]. Available: https://eprint.iacr.org/2023/369

[42] A. Pertsev, R. Semenov, and R. Storm, "Tornado cash privacy solution," 2019. [Online]. Available: https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf

[43] 0xPARC, "Dark forest," https://zkga.me/, 2020, accessed: February 22, 2024.

[44] iden3, "circomlib," 2018, accessed: July 22, 2024. [Online]. Available: https://github.com/iden3/circomlib

[45] C. Chin, H. Wu, R. Chu, A. Coglio, E. McCarthy, and E. Smith, "Leo: A programming language for formally verified, zero-knowledge applications," Cryptology ePrint Archive, Paper 2021/651, 2021, https://eprint.iacr.org/2021/651. [Online]. Available: https://eprint.iacr.org/2021/651

[46] StarkWare, "ethSTARK documentation," Cryptology ePrint Archive, Paper 2021/582, 2021, https://eprint.iacr.org/2021/582. [Online]. Available: https://eprint.iacr.org/2021/582

[47] P. Z. Team, "Plonky2: Fast recursive arguments with plonk and fri," 2022, accessed: July 22, 2024. [Online]. Available: https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf

[48] C. Moore, "Zk bench," 2023, accessed: July 22, 2024. [Online]. Available: https://zkbench.dev/

[49] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orrù, "zk-bench: A toolset for comparative evaluation and performance benchmarking of SNARKs," Cryptology ePrint Archive, Paper 2023/1503, 2023. [Online]. Available: https://eprint.iacr.org/2023/1503

[50] arkworks contributors, "arkworks zksnark ecosystem," 2022. [Online]. Available: https://arkworks.rs

[51] G. Botrel, T. Piellard, Y. E. Housni, I. Kubjas, and A. Tabaie, "Consensys/gnark: v0.9.0," Feb. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.5819104

[52] A. Ozdemir, F. Brown, and R. S. Wahby, "CirC: Compiler infrastructure for proof systems, software verification, and more," Cryptology ePrint Archive, Paper 2020/1586, 2020, https://eprint.iacr.org/2020/1586. [Online]. Available: https://eprint.iacr.org/2020/1586

[53] J. Eberhardt and S. Tai, "Zokrates - scalable privacy-preserving off-chain computations," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1084–1091.

[54] "Mina protocol: The world's lightest blockchain," 2023, accessed: November 24, 2023. [Online]. Available: https://minaprotocol.com/

[55] E. C. Company, "Zcash halo2 book," 2020, accessed: July 22, 2024. [Online]. Available: https://zcash.github.io/halo2/index.html

[56] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," Cryptology ePrint Archive, Paper 2021/370, 2021, https://eprint.iacr.org/2021/370. [Online]. Available: https://eprint.iacr.org/2021/370

[57] L. Labs, "Bellpepper zk-snark library," 2023. [Online]. Available: https://github.com/lurk-lab/bellpepper

[58] D. Boneh, S. Goldwasser, D. Song, J. Thaler, and Y. Zhang, "Zkp mooc," https://zk-learning.org/, 2023.

[59] G. Konstantopoulos, "Hardware acceleration for zero knowledge proofs," 2022. [Online]. Available: https://www.paradigm.xyz/2022/04/zk-hardware

[60] O. Shlomovits, "Revisiting paradigm "hardware acceleration for zero knowledge proofs"," 2023. [Online]. Available: https://medium.com/@omershlomovits/revisiting-paradigm-hardware-acceleration-for-zero-knowledge-proofs-16f717a49555

[61] Ingonyama, "Ingonyama," 2022. [Online]. Available: https://www.ingonyama.com/

[62] Cycisic, "Cysic," 2022. [Online]. Available: https://cysic.xyz/

[63] Fabric, "Fabric," 2022. [Online]. Available: https://www.fabriccryptography.com/

[64] Irreducible, "Irreducible," 2022. [Online]. Available: https://www.irreducible.com/

[65] Supranational, "Supranational," 2023. [Online]. Available: https://www.supranational.net/

[66] "Zcash: Privacy-protecting digital currency," 2023, accessed: November 24, 2023. [Online]. Available: https://z.cash/

[67] E. Foundation, "Zero-knowledge rollups," 2023. [Online]. Available: https://ethereum.org/en/developers/docs/scaling/zk-rollups/

[68] E. Systems, "Zk rollup architecture," 2024, accessed: April 10, 2024. [Online]. Available: https://docs.espressosys.com/sequencer/integrating-a-rollup/integrating-a-zk-rollup/zk-rollup-architecture

[69] M. Labs, "zksync era docs," 2023. [Online]. Available: https://era.zksync.io/docs/reference/

[70] P. Labs, "zkevm wiki," 2023. [Online]. Available: https://wiki.polygon.technology/docs/zkevm/

[71] ——, "Cdk wiki," 2023. [Online]. Available: https://wiki.polygon.technology/docs/cdk/

[72] Scroll, "Ethereum & scroll differences," 2023. [Online]. Available: https://docs.scroll.io/en/developers/ethereum-and-scroll-differences/#evm-opcodes

[73] Linea, "Linea," 2023. [Online]. Available: https://docs.linea.build/overview

[74] StarkWare, "Native account abstraction: Opening blockchain to new possibilities," 2023. [Online]. Available: https://starkware.co/resource/native-account-abstraction-opening-blockchain-to-new-possibilities/

[75] ——, "High-level overview," 2023. [Online]. Available: https://docs.starkware.co/starkex/overview.html

[76] Aztec, "Aztec docs," 2023. [Online]. Available: https://docs.aztec.network/

[77] A. Labs, "Introducing noir: The universal language of zero-knowledge," 2023. [Online]. Available: https://aztec.network/blog/introducing-noir-the-universal-language-of-zero-knowledge/

[78] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkbridge: Trustless cross-chain bridges made practical," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 3003–3017.

[79] Polyhedra, "Polyhedra," 2024, [Accessed 05-08-2024]. [Online]. Available: https://www.polyhedra.network/

[80] Telepathy, "Telepathy," 2024. [Online]. Available: https://docs.telepathy.xyz/

[81] H. D. Ltd., "Secure on-chain data," 2023. [Online]. Available: https://herodotus.dev

[82] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptology ePrint Archive*, 2018.

[83] P. L. Inc., "zk-snarks for the world," 2021. [Online]. Available: https://research.protocol.ai/sites/snarks/

[84] V. Buterin, J. Illum, M. Nadler, F. Schär, and A. Soleimani, "Blockchain privacy and regulatory compliance: Towards a practical equilibrium," https://ssrn.com/abstract=4563364, 2023, sSRN: 4563364.

[85] Penumbra, "Penumbra," 2024. [Online]. Available: https://protocol.penumbra.zone/main/penumbra.html

[86] G. Dunaif and D. Boneh, "How to build a private dao on ethereum," 2021. [Online]. Available: https://hackmd.io/nCASdhqVQNWwMhpTmKpnKQ

[87] M. Foundation, "How zkapps work," 2023. [Online]. Available: https://docs.minaprotocol.com/zkapps/how-zkapps-work

[88] K. Gurkan, K. Wei Jie, and B. Whitehat, "Community proposal: Semaphore: Zero-knowledge signaling on ethereum," March 2020. [Online]. Available: https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-semaphore.pdf

[89] "Intro to zero-knowledge proofs, semaphore and their application in world id," 2023, accessed: January 24, 2024. [Online]. Available: https://worldcoin.org/blog/worldcoin/intro-zero-knowledge-proofs-semaphore-application-world-id

[90] "Galxe/protocol-whitepaper," 2023, accessed: January 24, 2024. [Online]. Available: https://github.com/Galxe/protocol-whitepaper

[91] "zpass," 2024, accessed: January 24, 2024. [Online]. Available: https://zpass.docs.aleo.org/zpass/overview

[92] "Zero-knowledge proofs — qedit," 2023, accessed: February 1, 2024. [Online]. Available: https://qed-it.com/

[93] J. Z. Nasri and H. Rais, "zk-besc: Confidential blockchain enabled supply chain based on polynomial zero-knowledge proofs," in *2023 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE, 2023, pp. 1472–1478.

[94] S. Sahai, N. Singh, and P. Dayama, "Enabling privacy and traceability in supply chains using blockchain and zero knowledge proofs," in *2020 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2020, pp. 134–143.

[95] T. Conley, N. Diaz, D. Espada, A. Kuruvilla, S. Mayone, and X. Fu, "Instant zero knowledge proof of reserve," *Cryptology ePrint Archive*, 2023.

[96] G. G. Dagher, B. Bünz, J. Bonneau, J. Clark, and D. Boneh, "Provisions: Privacy-preserving proofs of solvency for bitcoin exchanges," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 720–731.

[97] Proven, "Cryptographically proving financial health. increasing trust and transparency in markets." 2023, accessed: January 30, 2024. [Online]. Available: https://www.proven.tools/products

[98] U. Fiege, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, 1987, pp. 210–217.

[99] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure," Cryptology ePrint Archive, Paper 2022/878, 2022. [Online]. Available: https://eprint.iacr.org/2022/878

[100] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[101] S. Lee, H. Ko, J. Kim, and H. Oh, "vcnn: Verifiable convolutional neural network based on zk-snarks," *Cryptology ePrint Archive*, 2020.

[102] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985.

[103] H. Sun, T. Bai, J. Li, and H. Zhang, "Zkdl: Efficient zero-knowledge proofs of deep learning training," *Cryptology ePrint Archive*, 2023.

[104] S. Garg, A. Goel, S. Jha, S. Mahloujifar, M. Mahmoody, G.-V. Policharla, and M. Wang, "Experimenting with zero-knowledge proofs of training," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1880–1894.

[105] K. Abbaszadeh, C. Pappas, D. Papadopoulos, and J. Katz, "Zero-knowledge proofs of training for deep neural networks," *Cryptology ePrint Archive*, 2024.

[106] H. Sun, J. Li, and H. Zhang, "zkllm: Zero knowledge proofs for large language models," 2024.

[107] Z. Inc., "What is ezkl?" 2023. [Online]. Available: https://docs.ezkl.xyz/

[108] M. Labs, "Bring powerful ai on-chain with specialized zk," 2023. [Online]. Available: https://www.modulus.xyz/

[109] ——, "Scaling intelligence: Verifiable decision forest inference with remainder," https://github.com/Modulus-Labs/Papers/blob/master/remainder-paper.pdf, Feb 2024.

[110] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," *Journal of the ACM (JACM)*, vol. 62, no. 4, pp. 1–64, 2015.

[111] I. GIZATECH, "Actionable ai for decentralized applications," 2024, accessed: April 3, 2024. [Online]. Available: https://www.gizatech.xyz/

[112] A. Naveh and E. Tromer, "Photoproof: Cryptographic image authentication for any set of permissible transformations," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 255–271.

[113] J. Groth, "Non-interactive zero-knowledge arguments for voting," in *Applied Cryptography and Network Security: Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005. Proceedings 3*. Springer, 2005, pp. 467–482.

[114] Starkware, "Veedo: a stark-based vdf service," 2020. [Online]. Available: https://medium.com/starkware/presenting-veedo-e4bbff77c7ae

[115] A. Ozdemir and D. Boneh, "Experimenting with collaborative {zk-SNARKs}:{Zero-Knowledge} proofs for distributed secrets," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 4291–4308.

[116] "Zkp2p: Exploring zero-knowledge proofs and peer-to-peer technologies," 2023, accessed: January 23, 2024. [Online]. Available: https://zkp2p.xyz/