SoK: Privacy-Preserving Computing in the Blockchain Era

Ghada Almashaqbeh[1] and Ravital Solomon[2]

[1] University of Connecticut, ghada.almashaqbeh@uconn.edu
[2] NuCypher, ravital@nucypher.com

Abstract. Cryptocurrency and blockchain continue to build on an innovative computation model that has paved the way for a large variety of applications. However, privacy is a huge concern as most (permissionless) blockchains log everything in the clear. This has resulted in several academic and industrial initiatives to address privacy. Starting with the UTXO model introduced by Bitcoin, initial works brought confidentiality and anonymity to payments. Recent works have expanded to support more generalized forms of private computation. Such solutions tend to be highly involved as they rely on advanced cryptographic primitives and creative techniques to handle issues related to dealing with private blockchain records (e.g. concurrency, private coin tracking to prevent double spending, efficiency). This situation makes it hard to comprehend the current state-of-the-art, much less build on top of it.
To address these challenges, we provide a systematization of knowledge for privacy-preserving solutions in blockchain. To the best of our knowledge, our work is the first of its kind. After motivating design challenges, we provide an overview of the zero-knowledge proof systems used in supporting blockchain privacy, focusing on their key features and limitations. Then, we develop a systematization of knowledge framework using which we group the state-of-the-art privacy preserving solutions under three categories: private payments, computation with input/output privacy, and function privacy. We briefly touch upon challenges and implications including misuse, regulations and compliance, usability, and limited functionality. Our work seeks to highlight open problems and research questions to guide future work directions.

1 Introduction

Following their revolutionary economic impact, cryptocurrencies and blockchain technology continue to build on an innovative computation model. Researchers and practitioners alike are racing to build new applications and transform existing systems into fully decentralized ones by utilizing the unique features of blockchain. While early systems focused mainly on payment transfer, newer smart contract-enabled blockchains allow individuals to build applications processing highly sensitive data, such as those involved in medical records tracking, trading, and voting.

However, lack of privacy is a huge concern. Popular systems like Bitcoin and Ethereum do not support privacy out of the box. All records are logged in the

clear on the blockchain, allowing anyone to read their contents. Moreover, while no real identities are required, several studies have shown how seemingly random-looking addresses can be linked to the real identities of their owners [1,2,3,4]. This is problematic; users do not want their payment activity to be disclosed, not to mention their more sensitive data such as votes or health-related information. Another issue caused by lack of privacy is susceptibility to front-running attacks [5]. That is, a malicious actor monitors broadcast transactions and races to issue her own transaction and have it be confirmed before the observed transaction (e.g. racing to win an auction). At the same time, revealing a coin's history may result in "tainted" currency; these are coins that no one wants to own or accept as a payment due to some undesired coin history (e.g. being used in some illegal trade).

All these concerns resulted in several academic and industrial initiatives to bring privacy to blockchains [6,7,8,9,10,11,12,13,14]. Starting with the simpler UTXO model from Bitcoin, early works aimed to provide confidential (hiding transfer amount) and anonymous (hiding user addresses) payments. Focus shifted to supporting privacy in the account-based model, with a desire to build private smart contracts. Such smart contracts would allow for arbitrary computation to be performed on the blockchain with input/output privacy. Extending privacy goals further, interest has emerged in supporting function privacy so that even the computation itself is hidden.

A common theme to these solutions is their reliance on advanced cryptographic primitives such as homomorphic commitments/encryption [7,8] and various zero knowledge proof systems [15,16,17]. This is in addition to the need for creative techniques to handle issues arising from supporting privacy, such as resolving concurrent transactions operating on private account state, dealing with anonymity sets, tracking private coins to prevent double spending, and addressing efficiency problems. Furthermore, determining regulation and compliance, especially with the increased interest in using blockchains in the financial sector, complicates the matter and calls for more attention towards potential misuses of privacy. All these facets make it hard to comprehend the current state-of-the-art, much less build on top of it.

Contributions. To address this challenge, we conduct a systematization of knowledge of privacy-preserving solutions in cryptocurrencies and blockchains, which (to the best of our knowledge) is the first of its kind. On one hand, such a study makes it easier for newcomers to understand the landscape. On the other hand, our work paves the way towards more advancements by identifying missing features and open questions. After motivating design challenges and providing a brief background on the main cryptographic building blocks used in supporting privacy (Section 2), we develop two systematization frameworks. The first is employed to survey zero-knowledge proof (ZKP) systems that blockchain privacy-preserving solutions adopt, with a focus on impactful facets—including their flexibility, security, and efficiency (Section 3). The second framework is used to survey state-of-the-art privacy-preserving solutions for

blockchain and map them into three categories (Section 4). The first category is private payments, which cover payments that do not disclose their currency amounts and/or the addresses of the sender and the recipient. The second category is private computation, which allows for arbitrary (user-specified) computation operating on private inputs and producing private outputs. Lastly, we look at function privacy, which also hides the computation itself. For the second category, we further classify the solutions based on the design paradigm they follow—homomorphic encryption-based or ZKP-based—and assess benefits and tradeoffs of each paradigm. After that, we briefly discuss implications of privacy-preserving payments and smart contracts such as misuse, regulation/compliance, and usability, in addition to technical proposals to address private computing for multi-user inputs (Section 5). Throughout these discussions, we identify open research questions and useful insights for the community to guide future work directions.

Scope. The blockchain space encompasses a massive body of work taking on several forms (e.g. peer-reviewed papers, white papers, technical reports, or even blog posts). Vetting and describing all these works is clearly infeasible. Thus, we study influential and representative solutions; we focus on peer-reviewed works (where possible) that have been used in operational projects or have served as the basis for new and creative design paradigms. This paper is not an attempt to describe in detail how each system or solution works, nor it is about providing formal definitions of protocols and their security aspects. Instead, its goal is to describe design paradigms and explore their key features, limitations, and tradeoffs. Moreover, anonymity is not the main focus; we target confidentiality (hiding inputs/outputs) and function privacy (hiding the computation itself). We discuss anonymity only for systems that support the other forms of privacy (the interested reader may consult [18] for a SoK on anonymity in blockchains). Furthermore, we do not focus on privacy for specific use-case blockchain-based systems like private decentralized exchanges or private voting applications. Instead, we study solutions that support privacy-preserving payments or arbitrary computation for a variety of applications. In particular, we study the following schemes:

- **Payments**: Monero [19] is one of the first private deployed cryptocurrencies, with numerous subsequent academic works on it. Zerocash [7] is a peer-reviewed paper with a resulting operational project Zcash. Quisquis [20] is a peer-reviewed work proposing a unique system model (account UTXO hybrid).
- **Computation**: Hawk [21] is one of the first peer-reviewed works in private blockchain computing. Zether [8] is a peer-reviewed work supporting confidential transactions on Ethereum; it is the first to build on the homomorphic encryption-based approach we introduce in Section 4. Zexe [9] is a peer-reviewed work, resulting in the operational project Aleo. Zkay [10] is a peer-reviewed work and operational project with an extensive open-source codebase to support private smart contracts on Ethereum. Kachina [14] is a peer-reviewed work from IOHK, one of the first theoretical works to for-

mally model private smart contracts. Arbitrum [13] is a peer-reviewed work, with a resulting operational project from Offchain Labs. Ekiden [12] is a peer-reviewed work, with a resulting operational project from Oasis Labs.

We note that privacy-preserving solutions are generally designed to be modular, meaning that they are not explicitly tied down to using a specific proof scheme. Hence, in implementing these schemes, some operational projects switched to more efficient building blocks (e.g. ZKP systems) than what was originally proposed in their peer-reviewed papers. This is a natural result of technical advancements, leading to more optimized cryptographic primitives than what was originally available. We describe schemes based on the peer-reviewed papers to unify the discussion and, when appropriate, mention the changes that operational projects adopted.

2 Background

To facilitate the discussion, we first present an overview of blockchain design, differentiate between the UTXO and account-based model (along with simple payments and smart contracts), and informally examine the notion of privacy in the context of blockchain. Finally, we review the main cryptographic building blocks used in the surveyed solutions.

2.1 Blockchain Components

A blockchain is an append only log (usually referred to as a distributed ledger), representing the backbone of any cryptocurrency. This ledger records all transactions in the system, allowing mutually trustless parties to exchange payments. A blockchain is maintained and extended by miners who compete to win the rights of mining the next block and, hence, collecting the mining rewards. The current state of a blockchain is agreed upon through the consensus protocol that these miners run. End users (lightweight clients) can then use the service by broadcasting transactions to the miners and tracking only their own records, rather than the full blockchain.

In general, cryptocurrencies can be classified into one of two categories based on the way in which they track currency owned by clients. The first of which is the unspent transaction output (UTXO) model, initially proposed by Bitcoin [22]. About half a decade later, Ethereum went on to pioneer an account-based model [23]. In the former model, miners need to maintain all unspent transactions, with a client's currency balance computed as the total value of all unspent transactions destined to her address(es). In the latter, each address has an account on the blockchain associated with a balance that is updated based on the currency transfer transactions this account issues or receives.

Furthermore, cryptocurrencies can be classified into two categories based on the functionality they support. In the first category, only currency transfer operations are supported with limited scripting capability to make conditional payments. We refer to this as the Bitcoin-like model. While in the second category,

the end user can deploy arbitrary programs on the blockchain for the miners to execute on demand in the form of smart contracts. We refer to these as smart contract-enabled cryptocurrencies, where Ethereum was the first to introduce this model.

Security of a blockchain can be defined in terms of satisfying a set of security properties [24,25,26]. Informally, a ledger $\mathcal{L}$ is secure if it satisfies the following:

- Persistence: For any two honest parties P_1 and P_2, and any two time rounds t_1 and t_2 such that $t_1 \leq t_2$, the ledger maintained by P_1 at t_1 is a prefix of the ledger maintained by party P_2 at time t_2 with overwhelming probability.
- u-Liveness: If a transaction tx is broadcast at time round t, then with overwhelming probability tx will be recorded on the ledger $\mathcal{L}$ at time $t+u$, where u is the liveness parameter.
- Chain quality (or correctness): With overwhelming probability, a ledger $\mathcal{L}$ records only valid transactions and blocks.
- Fairness: With overwhelming probability, any miner collects mining rewards in proportion to the mining power it puts in the system.

2.2 Privacy Domain in Blockchains

Privacy shares the general theme of hiding user data. However, in the context of blockchains this can take on several forms. The first type of privacy is input/output privacy (also known as confidentiality), which allows us to hide the inputs and outputs of an operation or function. Confidential currency transfer can be viewed as a restricted form of I/O privacy, since it translates to hiding the amount being sent along with the balances of the sender and recipient addresses. For more general smart contracts, I/O privacy translates to hiding the inputs and outputs of the functions defined within the code of smart contracts. The second type of privacy is function privacy, allowing us to hide the computation itself. Function privacy may be of particular interest for proprietary code or when the code leaks information on the type of processed inputs (potentially having privacy implications). Finally, user anonymity is a form of privacy since it deals with hiding the users' addresses involved in a transaction or a computation. As mentioned earlier, we focus on the former two types of privacy in our work.

In terms of security notions, a privacy-preserving cryptocurrency must satisfy the security properties of a blockchain outlined in the previous section along with additional properties to capture privacy. Informally, these privacy related properties include [7,8]:

- Ledger indistinguishability: An adversary cannot distinguish between two versions of the ledger that differ in at least one transaction with non-negligible probability.
- Balance or overdraft safety: An adversary cannot spend more currency than he rightfully owns, even if the values are hidden and the currency cannot be tracked.

The above notion of ledger indistinguishability can be formulated differently to state that a ledger does not reveal any additional information about private data (or computation) beyond what can be inferred from public records. Such a definition would allow us to cover private computation with I/O privacy and function privacy. The notion of balance/overdraft safety will remain the same for these categories, meaning that even if an adversary requests I/O or function privacy-preserving operations, she will not be able to obtain free coins in the system.

Why is privacy harder for smart contracts than for payments? First, note that a smart contract can be any program of the user's choice. It may involve complex operations (more than just the addition used in currency transfer) and have application-dependent conditions to be checked for the inputs. In the account model, unavoidable concurrency issues occur from trying to operate on encrypted balances using zero-knowledge proofs. Second, smart contracts may operate on inputs from different users, meaning that these inputs are encrypted with respect to different keys. Allowing such interoperation is non-trivial and requires sophisticated cryptographic primitives. Third, efficiency issues from providing privacy (particularly from the use of ZKPs) are compounded in private computation extensions. Lastly, the flexibility provided by smart contracts raises several questions related to correctness and legitimacy of the deployed code. What if this code simply reveals all inputs and/or users addresses? This places a huge burden on the end user to vet such applications and contracts before using them.

2.3 Cryptographic Building Blocks

Privacy-preserving solutions are centered around a handful of cryptographic primitives. These are (informally) defined in what follows.

Commitments. A non-interactive commitment scheme is composed of three efficient algorithms: Setup, Commit, and Open. Setup takes as input a security parameter κ and generates a set of public parameters pp. Commit takes pp, a message m, and randomness r as inputs, and outputs a commitment c to m. Open takes pp and c as inputs and produces a decommitment $d = (m, r)$.

A secure commitment scheme must satisfy two properties: *hiding*, meaning that commitment c does not reveal any information about m, and *binding*, meaning that a commitment c cannot be opened to m' such that $m' \neq m$. Such properties allow for recording private data on the blockchain, hidden in commitments, with the guarantee that the owner cannot change the original data without being detected. A formal definition of a commitment scheme and its security can be found in [27].

Some commitment constructions are additively homomorphic in the sense that given c_1 and c_2, which are commitments to messages m_1 with randomness r_1 and m_2 with randomness r_2, respectively, $c_3 = c_1 + c_2$ is a commitment to $m_1 + m_2$ with randomness $r_1 + r_2$. This allows for operating on committed values (such as account balances) without opening them. This property is valuable

in private payments as it allows for updating a private (committed) account balance by adding and subtracting (committed) currency amounts. Pedersen commitments [28] support such features.

Homomorphic encryption. A homomorphic encryption (HE) scheme, like a regular encryption scheme, is composed of three efficient algorithms: KeyGen which generates encryption/decryption keys (and any other public parameters based on the construction), Encrypt which encrypts a message m to produce a ciphertext ct, and Decrypt which decrypts a ciphertext ct to get the plaintext message m back.

Homomorphic encryption schemes allow for performing computations on ciphertexts such that the output ciphertext will decrypt to the same plaintext output as if we had operated directly on the underlying plaintexts. Additively homomorphic encryption schemes only support addition homomorphisms (i.e. operations). That is, let ct_1 be a ciphertext of m_1, and ct_2 be a ciphertext of m_2, then $ct_1 + ct_2 = ct_3$ is a ciphertext of $m_1 + m_2$. This supports an equivalent utility as homomorphic commitments; updating an encrypted account balance by adding and subtracting (encrypted) currency amounts. At the same time, some encryption schemes can only support homomorphic multiplication, i.e. $ct_4 = ct_1 \cdot ct_2$ is a ciphertext of $m_1 \cdot m_2$. For example, the ElGamal encryption scheme can be either additively homomorphic or multiplicatively homomorphic, based on whether m is encrypted in the exponent, but not both.[3]

Fully homomorphic encryption (FHE) supports both addition and multiplication of ciphertexts. This allows for performing any computation over encrypted inputs to produce encrypted outputs. All currently known schemes rely on lattice-based cryptography, thus providing post-quantum security guarantees. The first FHE scheme was introduced by Gentry [29] and was followed up by a large body of works devising more optimized constructions [30,31,32,33,34,35,36]. FHE continues to be an active research area given increased interest in privacy and computation outsourcing.

Zero knowledge proofs. A (non-interactive) zero knowledge proof (ZKP) system allows a prover to convince a verifier that it knows a witness ω for some statement x without revealing anything about the witness beyond what can be implied by x itself. An example could be proving that a given ciphertext encrypts an integer y that lies in the range $[a, b]$, without revealing the exact value of y.

A ZKP system is composed of three algorithms: Setup, Prove, and Verify. Setup takes as inputs security parameter κ and specifications of the NP relation (which determines the set of all valid statements x) for which proofs are to be generated, and outputs a set of public parameters pp. Prove takes as inputs pp, a statement x, and a witness ω for x and outputs a proof π proving correctness

[3] Note that for simplicity we represent homomorphic addition and multiplication using '+' and '$\cdot$'. Based on the scheme, the exact implementation of each operation may vary, e.g., in additively homomorphic ElGamal encryption, ciphertexts are multiplied with each other to have a ciphertext of $m_1 + m_2$.

of x (that it satisfies the NP relation). Lastly, Verify takes pp, statement x, and π and outputs 1 if the proof is valid, and 0 otherwise.

A secure ZKP system must satisfy several properties including completeness, soundness, and zero-knowledge. *Completeness* ensures that any proof that is generated in an honest way will be accepted by the verifier. *Soundness* (or proof-of-knowledge) states that if a verifier accepts a proof for a statement x then the prover knows a witness ω for x. Put differently, this means that a prover cannot convince a verifier of false statements. Finally, *zero knowledge* ensures that a proof π for a statement x does not reveal anything about the witness ω. An additional efficiency-related property is succinctness—which means that the proof size is constant and the verification time is linear in the size of the input, regardless of the circuit size representing the NP relation underlying the proof. A ZKP system that satisfies all these four properties is denoted as zk-SNARK (zero knowledge succinct non-interactive argument of knowledge). Formal definitions of ZKP systems and zk-SNARKs can be found in [27,37].

ZKPs are utilized heavily in private cryptocurrencies and blockchain applications. They allow for proving that an input satisfies certain conditions, that an operation has been performed correctly, or that the ledger state has been updated successfully, without revealing anything about the underlying private data.

Multiparty computation. A multiparty computation (MPC) protocol allows a set of mutually-distrusted parties to evaluate a function over their private inputs without revealing anything about these inputs beyond what can be inferred from the output. Most MPC protocols in the literature use two approaches: the secret sharing based approach [38] and garbled circuits [39].

An MPC protocol is secure if it satisfies three properties: correctness, privacy, and fairness. Correctness means that an MPC protocol executing a function f will produce the same output that f would produce if it were to operate on the inputs in the clear. Privacy means that no information about the parties' inputs is leaked apart from what the output may reveal. Finally, fairness ensures that either all or none of the parties learn the output. These are often captured using an ideal functionality notion for the intended computation, along with a simulation-based security proof comparing an ideal execution of the protocol with a real world one. The interested reader may consult [40] for further details and formal definitions.

MPC protocols are mainly used to distribute trust, thereby replacing a trusted entity with a set of parties to perform the same functionality in a private way. In blockchain systems, MPC protocols were mainly used to execute a trusted setup when needed (e.g., producing a common reference string for non-interactive ZKP) [41,42]. As we will discuss later, MPC can also be utilized to execute off-chain private computations over inputs coming from multiple users.

	PHGR13 [15]	**Bulletproofs** [17]	**GM17** [43]
Used in	Zerocash, Hawk*	Quisquis, Zether*	Zexe, Zkay
Universal	X	✓	X
Transparent	X	✓	X
Prover Time	Quasilinear	Linear	Quasilinear
Verifier Time	Linear	Linear	Linear
Size	Constant	Logarithmic	Constant

Table 1: Comparison of the major ZKP systems used in private computing for blockchains. Note that the starred schemes use variants of these proof systems.

3 Zero Knowledge Proof Systems

In providing privacy on blockchain, parties often need to prove that conditions on their hidden inputs have been satisfied for the appropriate application. In private currency transfer, for example, this might mean ensuring that the hidden amount being sent is non-negative. Zero-knowledge proofs (ZKPs) provide a cryptographic solution to this problem. The vast majority of blockchain constructions offering privacy rely on ZKPs. Of the 10 works we survey in this paper, only 2 of them do not make use of ZKPs.[4] Accordingly, research in ZKPs has exploded in the past years, with a goal of constructing lightweight ZKPs for the blockchain setting.

In this section, we identify three main features of ZKPs and discuss how these features affect deployment of privacy-preserving computing solutions in blockchain.

3.1 Proof Systems Used

We examine three major proof systems that are used to support privacy-preserving computing in blockchain. All of them use elliptic curves, are (or can be made) non-interactive, and support proving relations for general arithmetic circuits.

PHGR13 and variants. This proof system [15] is a type of zk-SNARK that relies on pairings to produce constant-sized proofs. Its security relies on the knowledge of exponent assumption, a recent non-falsifiable security assumption [44]. PHGR13 was first used in Zerocash to achieve succinct proofs for private currency transfer [7]. Such zk-SNARK proof systems can be transformed to support simulation extractability, thus ensuring that an adversary, who does know a witness, cannot forge a proof despite seeing an arbitrary number of valid proofs [16]. Hawk [21], a private smart contract platform, requires this feature and applies Kosba's transformation ([45]) to achieve this.

[4] Kachina [14] is a theoretical protocol to support private smart contracts using non-interactive zero-knowledge proofs (NIZK). As it does not rely on any particular proof system, we abstain from discussing it further in this section as the following considerations are not applicable.

Bulletproofs and variants. Bulletproofs [17] allow for fairly efficient logarithmic-sized range proofs (in addition to supporting relations for arithmetic circuits). This proof system has the advantage of relying solely on the discrete logarithm assumption. Bulletproofs were initially used to support private currency transfer in Quisquis [20] (and eventually in the operational project for Monero [19]). Zether employs a variant of Bulletproofs called Σ-Bullets that make Bulletproofs interoperable with Sigma protocols, thereby allowing them to efficiently prove that algebraically encoded values lie in a particular range [8].

GM17. This proof system [43] is another type of zk-SNARK that relies on pairings to produce highly succinct (constant-sized) proofs for arithmetic circuits. Its security relies on an assumption similar to the knowledge of exponent assumption [46]. Unlike PHGR13, GM17 provides simulation extractability out of the box. The private computation schemes Zexe [9] and Zkay [10] both use GM17 as the basis of their constructions.

3.2 Categorization of ZKPs

Unfortunately, it is not immediately clear how to best categorize or analyse these zero-knowledge proof systems as there is no general framework to do so. This is, in part, due to the fact that they share a number of features in common; additionally, all proofs were chosen carefully to suit blockchain applications. Thus, we focus on properties that impact practical deployment. In doing so, we identify three main features—flexibility, security, and efficiency—and differentiate between the proof systems according to these measures.

In looking at flexibility, we emphasize universality (i.e. can a single reference string be used to prove any NP statement) [47]. A non-universal proof system will require generating new reference strings for new applications, which impacts practicality especially when it comes to supporting arbitrary applications.

In looking at security, we focus on trust level. A number of the ZKPs require a "trusted setup," in which a trusted party generates some initial parameters used in the proof system. This party must be trusted to behave honestly, otherwise the security (particularly the soundness) of the proof system could be compromised. While it is not the case that a trusted setup implies non-universality, in the body of work we look at, these two features go hand-in-hand.

In looking at efficiency, we discuss time and space overheads. To ensure high throughput and fast response, the ZKPs used in privacy-preserving solutions must be as lightweight as possible; usually, this translates to seeking ZKPs with small proof size and fast verification time. Although fast proof generation is an advantage, it is not quite as critical as proof verification (a task that falls on the miners to perform in the system). As we will see, the majority of constructions utilize trusted setups and non-universal reference strings for efficiency.

3.3 Flexibility

In providing such lightweight ZKPs, a challenges arises; the most lightweight constructions in practice are non-universal (so that the reference string depends on the relation) [47].

Universality. As mentioned above, a proof system is "universal" if a single reference string can be used to prove any NP statement. Non-universality presents no immediate problems for private currency transfer as the construction is usually limited to supporting a fixed number of known relations. Accordingly, the first proposed private currency transfer scheme, Zerocash [7], adopted a non-universal ZKP scheme. Subsequent private payment works (like Quisquis [20] and even Monero itself in practical deployment) moved to using universal proof systems such as Bulletproofs.

Non-universality limits the flexibility of users to engage in more general purpose private computation since a new reference string would need to be generated for new applications. In spite of this challenge, three of the four private computing solutions (utilizing ZKPs) propose using proof systems with non-universal reference strings for maximum efficiency.

3.4 Security

Additionally, the most lightweight proof constructions utilize a trusted setup.

Trust. The earliest works in both private currency transfer (Zerocash) and private computing (Hawk) require a trusted third party to perform the initial setup process. This involves generating the common parameters of the system, as well as a preprocessing step that provides the verifier with a succinct representation of the relation being proved. Preprocessing has a direct impact on efficiency as it significantly cuts down on the proof verification time.

Nonetheless, as the name suggests, trusted setups are a source of security issues if this *trusted* third party does not behave honestly (i.e. reveals the randomness used to generate the reference string or any other secret trapdoors). In particular, this party can use the secret information to potentially break the soundness of the proof system and, hence, spend currency she does not actually own [7]. A popular mitigation strategy is to distribute trust by employing multi-party computation so that many parties can participate in the setup process [41,42]. Thus, as long as at least one party is honest, the whole setup will be secure. If the parties act honestly, any secret information will be destroyed after finishing the setup as instructed.

Due to these security implications, later works such as Quisquis and Zether moved to using transparent proof systems (i.e. proof systems without trusted setups) like Bulletproofs. However, new cryptocurrency designs supporting more general forms of private computation (i.e. Zexe and Zkay) did not adopt this trend. ZKPs with trusted setups continue to be popular because of their efficiency.

3.5 Efficiency

In using ZKPs in a distributed setting like blockchain, efficiency is arguably the biggest concern. Users (who serve as the provers) are often lightweight nodes with limited computational power, thus proof generation cannot be too expensive for them. Although miners (who serve as the verifiers) likely have access to more powerful machines, they may need to verify all proofs produced in the system, so minimizing verification time is important to ensure high throughput. Additionally, miners must track the entire blockchain, which may include all proofs produced in the system. Thus, proof size should be as small as possible, ideally with size independent of the circuit representing the underlying NP statement.

ZKP efficiency has so many facets that it deserves its own SoK to do the topic proper justice. We provide a brief overview of the main considerations and motivate how these factors affect private computing in blockchains.

Theory vs. practice. While the difference between the proof systems may look stark in terms of asymptotic Big O notation, performance in practice is not quite as clear cut. In practice, Bulletproofs tend to be about one order of magnitude larger than the PHGR13's zk-SNARKs [15] when proving statements for confidential payments. Additionally, it can be difficult to compare concrete performance across papers as the authors take advantage of different techniques and provide non-standardized benchmarks. For example, authors may use many cores (Zkay, 12 cores), high RAM (Zexe, 256GB RAM), or manually optimize the arithmetic circuit representing the NP statement to be proved (Zerocash). Thus, we focus primarily on asymptotic behavior with the goal of providing an intuition of the cost. It should be noted that this too can be misleading as various proof systems quote asymptotic behavior with respect to different parameters.

Time. Private operations (whether a transaction or a smart contract function invocation) will be accompanied by a proof, generated upon issuing this operation and verified upon accepting/executing it in the system. Initial schemes, like Zerocash, used zk-SNARKs with quasi-linear proof generation time as these tended to behave like a constant function for most statements the user might need to prove [7]. Later constructions employed proof systems with both linear and quasi-linear proving times. In terms of proof verification, all the proof systems we look at offer linear verification time.

A potential optimization for verification is *batching* (supported in Bulletproofs). Batching makes it cheaper to verify n proofs together than verifying each proof on its own [17]. It relies on the observation that verification is essentially many multi-exponentiations that can be efficiently combined together to reduce the work. With this technique, verification time grows logarithmically initially and then linearly [17].

Another consideration is the time needed to execute the ZKP setup process, especially for non-universal proofs. This is not particularly important for systems that only support private payments as these systems consist of a set of fixed

statements to be proved (so setup will be performed only once when the system is launched). General private computing schemes, on the other hand, could be more sensitive to such factors since a new setup may be needed whenever a new application is deployed. Setup time is non-trivial and often takes on the order of minutes; setup time for the zk-SNARK in Zexe supporting privacy on two inputs takes over 1.5 minutes (using a server), whereas setup for the zk-SNARK in Zerocash takes over 4 minutes (using a modest machine) [9,7]. Compounding the problem further, these non-universal proofs also use trusted setup; recall that, in deployment, a trusted setup is often executed as an MPC ceremony. Repeating this MPC process many times over (when users want to prove new statements for private applications) will be costly and require a non-trivial amount of coordination/cooperation.

Space. Miners must store the full blockchain history; unfortunately, a ZKP can often be the largest contributor to a transaction's size. Using constant-sized proofs can help manage the chain's growth. This observation was realized early on in Zerocash, which pioneered the use of zk-SNARKs with constant-sized proofs. Subsequent works attempting to extend the Zerocash protocol (i.e. Hawk, Zexe) naturally chose to use proof systems with constant-sized proofs as well. Yet, these constant-sized proofs come at a cost. The keys needed to produce the succinct proofs can be very large (i.e. many orders of magnitude larger than the individual proofs themselves in practice). As an example, we look at Zerocash; for the "pour" relation representing the NP statement of the proof needed for a basic private transaction, the proof size is 288 bytes whereas the necessary proving key is 896 MiB [7].

Private computing solutions with universal and transparent ZKPs (such as Quisquis and Zether) produce logarithmic-sized proofs [17]. While these proofs tend to be about one to two orders of magnitude larger in practice than PHGR13 and GM17's zk-SNARKs, they have the advantage of not requiring a proving key (only a small public reference string) [17].

4 The Landscape: Existing Privacy-Preserving Solutions

In this section, we review the solutions that were developed in the past decade to bring privacy to blockchain and cryptocurrencies. We begin by describing our systematization of knowledge framework, followed by a study of these solutions. This study focuses on important features such as work model, security, and efficiency. We also examine how to handle several technical challenges (e.g. double spending and concurrency) that become non-trivial when dealing with private records.

4.1 Systematization Framework

Based on the characteristics and features of the surveyed solutions, we developed a systematization of knowledge framework to fit these solutions into relevant

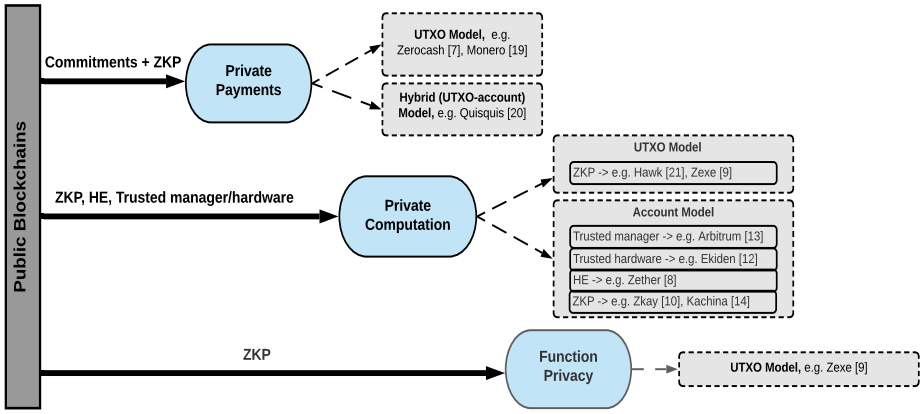

Fig. 1: Systematization framework diagram. Arrows carry the cryptographic building blocks and/or trust assumptions used in the schemes under a category. Arrows' length indicates timeline, where a longer arrow leads to a more recent category.

categories. As shown in Figure 1, our framework follows the evolution of these solutions with respect to supported functionality. It also examines the cryptographic building blocks employed and the trust assumptions needed to support the intended functionality. This process produced three main categories that we discuss in the following paragraphs.

Private payments. The earliest category came mainly as a response to Bitcoin's lack of privacy. It focuses on providing confidentiality as well as anonymity (of the sender and the recipient). We review three schemes [7,20,19] under this category.

Private computation. Private payments are inherently restricted in their functionality; they only support transferring currency from one party to another based on a limited set of conditions encoded in simple system-prescribed scripts. Addressing these limitations is the motivation for the schemes under this category; these schemes mainly took one of two paths. The first enables users to instruct the miners to execute arbitrary computations over private inputs and produce private outputs. The second offloads work to the user who implements the computation locally and produces ZKPs that miners must verify before accepting the output and updating the blockchain state. Some private computation constructions do not follow any of these paths, instead they rely on alternative techniques (e.g. trusted hardware) to achieve their goals. The private computation category encompasses seven schemes [8,9,10,13,12,14,21].

Function privacy. The final category extends the previous two to support a more ambitious goal; hiding the computation itself along with hiding inputs/outputs. This allows for protecting proprietary code or preventing information leaks that may result from knowing the computation (e.g. inferring the data type of inputs/outputs to be medical records or trading information). To the best of

our knowledge, there is only one scheme that belongs to this category [9]; this scheme follows the computation offloading approach mentioned above.

4.2 Private Payments

As the first successful cryptocurrency, Bitcoin served as the starting point (or base system) for private payment solutions. All these solutions center around hiding the user's payment activity. This translates into hiding transaction amounts and issuer/recipient addresses to provide confidentiality, anonymity, and transaction unlinkability. We study three schemes in the private payments category: Zerocash [7], Monero [19], and Quisquis [20]. All of them follow the same generic paradigm; hide a transaction value inside a commitment, provide ZKPs showing that the transaction issuer indeed owns the hidden coins she wants to spend and that she cannot spend more than the input value (beside meeting other system specific conditions), and lastly rely on anonymity sets to disguise the issuer and recipient addresses.

As one may expect, having a fully functional private cryptocurrency is not straightforward. It requires handling several challenges and devising new techniques to process a ledger with private records instead of only public ones. In what follows, we examine such issues while showing how they are handled in the three schemes. It should be noted that most of these issues and techniques are also common to private computation so we only discuss them in detail here.

Work model. Zerocash and Monero are based on Bitcoin, thus inheriting its UTXO-based model. Quisquis combines UTXOs with a notion of accounts, resulting in a hybrid model—that is, each user will have an account but a transaction will contain UTXO inputs rather than accounts. This is made possible thanks to updatable public keys [20], a primitive that allows for having multiple distinct public keys tied to the same private key. All these keys are derived from the original public key generated when creating a specific account. Thus, UTXOs are defined in terms of these updated public keys; a user can spend all UTXOs that belong to her account using the same private key. Quisquis' novel approach makes handling double spending and breaking transaction linkability simpler as we will see shortly.

Confidentiality. As mentioned above, commitments are used to hide (and bind) private data. Additive homomorphisms may be required if a scheme needs to perform a computation over these commitments. For example, transactions in Monero and Quisquis must show that the sum of the input coins equals the sum of the output coins, so that no one can spend more coins than what they own. This can be done without disclosing any of the I/O values by operating on the commitments themselves. Zerocash does not need additive homomorphisms since checking the prior condition is part of the NP statement of the ZKP system it uses instead (i.e. it is part of the arithmetic circuit that the proof must satisfy).

Range checking. Operating on hidden (committed) values introduces another challenge; a malicious transaction issuer can mint free coins. She can create an output to herself with a very small negative value that will be translated into a very large positive value when applying finite field modular operations. To prevent this attack, a transaction issuer must prove that all currency values in a transaction are positive and within the range allowed in the system. Range checking can be part of the same NP statement of the ZKP (as in Zerocash) or proved separately using range proofs (as in Quisquis). Monero, on the other hand, uses ring signatures [19] to handle this task, where values are represented in binary expanded format and a ring signature cannot be produced if the coefficients are not within the allowed range.

Anonymity and transaction linkability. Anonymity is achieved via anonymity sets. Any transaction will be tied to a set of private UTXOs (or coins) such that a spent coin may be any coin within this set. Hence, a transaction issuer needs to prove that she owns one (or more) of the hidden coins in the set without revealing which coin it is. The larger the anonymity set, the better since the probability of guessing the actual input coin (hence the owner address) will become smaller.

Zerocash employs this exact approach, with the proof of ownership as part of the same NP statement of the underlying zk-SNARK proof. Monero instead uses ring signatures; the anonymity set is a public key matrix that is used in the ring signature a party generates when signing a newly issued transaction. This signature proves that the signer is a member of the group (.e. she knows the secret key of one of the public keys in the key matrix) without specifying which member.

Quisquis, on the other hand, combines anonymity sets with updatable keys to support anonymity. All input public keys (the sender, recipient, and anonymity set) will be updated, the balances of the anonymity set accounts stay as they are, while the sender's balance is decremented and recipient's balance is incremented based on the transferred currency amount. Thus, any public key will be used at most twice in the network—once, when it is generated on the output side and, finally, when it is spent on the input side of a transaction. Since the updated keys look like freshly generated ones, Quisquis protects against transaction linkability.

Zerocash provides a higher level of anonymity since its anonymity set includes all private coins in the system. Although Monero and Quisquis can support this, it would greatly impact efficiency since more operations would need to be performed for each additional member in the set (producing a signed commitment in Monero and a key update in Quisquis). In contrast, for Zerocash, the cost of the ZKP does not rely on the size of the anonymity set. Nonetheless, empirical studies reveal weaknesses in these schemes when it comes to anonymity. The multiple usage of public keys in Monero's ring signatures allow for breaking its anonymity [48]. Additionally, the anonymity set of a deployed version of Zcash (the operational project built upon Zerocash) can be considerably shrunk using heuristics based on identifiable usage patterns [49].

Plausible deniability. Plausible deniability allows a party to deny having participated in a private transaction. Both Monero and Quisquis support this feature since a transaction issuer can pick any public key in the system to be part of the anonymity set without the consent of the key owner [20]. Zerocash does not support this property, since the anonymity set includes all private coins in the system. Thus, in Zerocash, a user agrees to be part of the anonymity set the minute she owns a private coin.

Handling double spending. Another challenge is how to prevent a party from spending a hidden coin multiple times. In Bitcoin (and other public cryptocurrencies), this is trivial since all transactions are logged in the clear on the blockchain. Thus, anyone can check if some UTXO is already spent by checking the ledger. In private cryptocurrencies, additional machinery is needed.

The core idea private cryptocurrencies employ is to tie each unspent coin with a unique value (or capability) such that when a coin is spent this value is revealed (or this capability is disabled). For example, Zerocash assigns a unique sequence number to each coin that is published publicly on the blockchain when the coin is spent. Hence, a new transaction that produces a sequence number(s) that has already been revealed indicates a double spending attempt and will be rejected. In Quisquis, the unique value is the public key itself; since a transaction updates all input public keys, any key will appear only once on the input side of any transaction. Thus, the reuse of the same input key indicates double spending. Monero, on the other hand, adopts the unique approach of one time signatures. The key that appears in a UTXO can be used only once to produce a valid ring signature to spend that UTXO. This is due to recording an image (salted hash) of the public key on the blockchain when the UTXO is spent. Therefore, any transaction with a signature tied to an already published image will be rejected since it will be recognized as double spending.

Both techniques require searching publicly published data on the blockchain to prevent double spending, similar to how double spending in prevented for public cryptocurrencies.

Discussion. A distinguishing feature of the schemes discussed in this section is the size of the UTXO set that miners maintain in the system. Both Zerocash and Monero have a growing list of UTXOs (since to preserve anonymity a UTXO is not identified when it is spent), preventing miners from storing a concise version for the blockchain. Although Zerocash computes a Merkle tree over this list, which helps to reduce the cost of generating a ZKP, all UTXOs must still be kept. Quisquis's use of updatable keys solves this problem. Any updated public key with a zero balance will have a proof indicating this fact so that other parties will remove it from the UTXO pool in the system; this advantage is one of the main motivations behind developing Quisquis.

The updatable key primitive allows for the hybrid UTXO-account model of Quisquis, which has many additional advantages. These include simplifying how to handle double spending and private key bookkeeping for lightweight clients

(that is, any account is managed using a single secret key). Moreover, since transactions are still in terms of UTXOs (so all ZKPs are with respects to these UTXOs), account operation concurrency is not an issue. By this, we mean that updating account state does not impact validity of pending transactions' ZKPs. However, in contrast to a pure account model, a client's wallet must track all UTXOs (and so all their updated public keys) tied to an account rather than the account state only.

Another point to highlight is the ZKP NP statement a scheme relies on to ensure validity of private transactions. Zerocash packs all conditions into a single NP statement (hence, one representative arithmetic circuit). Quisquis and Monero use separate techniques to ensure that conditions are satisfied. The latter approach may allow for more performance optimizations, especially when lightweight customized techniques are used. For example, the use of additively homomorphic commitments allows for checking that currency is preserved between inputs and outputs in a much cheaper way than using a ZKP. At the same time, having a single constant-sized ZKP with short verification time may compensate for the performance gains that customized approaches may offer.

4.3 Private Computation

Building on ideas from private currency transfer, private computation schemes sought to provide I/O privacy for arbitrary computations on blockchain. There are two major approaches to private computation in blockchain, where both make use of zero-knowledge proofs but to very different ends. We refer to the first approach as *homomorphic encryption-based* whereas the second approach is *ZKP-based*; these terms are coined based on which cryptographic primitive allows for performing the private computation itself. The former approach extends ideas from Quisquis and Monero to support more complex private computations using homomorphic operations. The latter approach builds on ideas from Zerocash, asking the user to produce a ZKP proving she performed the computation correctly and that all necessary conditions were satisfied—without the need for homomorphic operations. Both of these approaches can be used regardless of the system model (i.e. account vs. UTXO). However, the model chosen to support private computation has important security implications we go on to consider.

Finally, we look at a third private computation approach that diverges from ideas found in confidential transfers. This third category does not make use of ZKPs, instead delegating work to a third party—be it trusted hardware or trusted managers—who will help lower both the users' and miners' overall computational burden.

HE-based approach. The homomorphic encryption-based approach extends ideas from confidential transfers using homomorphic addition. However, here, the goal is to support arbitrary computation with I/O privacy. As this approach is instantiated in the account-based model, a public key encryption scheme will be more suitable and simpler to work with than commitments. The approach works as follows. Users provide encrypted inputs for the desired computation,

along with a ZKP proving that necessary (application-specific) conditions have been satisfied on their inputs. Miners verify the ZKP and can then perform the requested computation directly on the encrypted inputs. The types of computation supported over ciphertexts are determined by the inherent homomorphic properties of the chosen encryption scheme. Thus, we refer to this approach as "homomorphic encryption-based" as the encryption scheme allows for performing the private computation. For example, if an additively homomorphic encryption scheme is chosen, then private computation is restricted to addition only (i.e. users can only request additive computations). Naturally, we might ask if we can support I/O privacy for *arbitrary* computations by using an encryption scheme that is both additively and multiplicatively homomorphic (i.e. FHE). If FHE is supported, a user can ask miners to perform arbitrary computations on her encrypted inputs.

Zether [8] was the first scheme to employ the HE approach, using the ElGamal encryption scheme to encrypt a user's account balance and update it via homomorphic addition. While the primary goal of Zether is to support confidential payments atop Ethereum via a new token, it can also support a restricted class of private smart contracts. As ElGamal encryption is used, Zether can only support I/O privacy for additive computations. Nevertheless, addition suffices for hiding a bid on a fixed number of items or hiding a vote.

smartFHE (as its name might suggest) uses an FHE scheme to support arbitrary computation on a user's encrypted inputs [50]. To do so, it employs the BGV fully homomorphic encryption scheme [30], in conjunction with Bulletproofs and a proof system suitable for proving lattice-based relations. Building on ideas from Ethereum and Zether, it offers dual accounts for public and private operations which can be supported via smart contracts.[5]

ZKP-based approach. The ZKP-based approach asks the user to perform the computation offline on her plaintext inputs to get plaintext outputs. The user then encrypts both the inputs and outputs and produces a ZKP proving that this offline computation/update was done correctly. The encrypted inputs, encrypted outputs, and ZKP are posted on chain. The miners' role here is to simply check the ZKP (as the user performed the private computation herself) and then update the blockchain state accordingly. Thus, we refer to this approach as "ZKP-based" as it relies primarily on the power of ZKPs to perform a private computation. Zexe [9], Zkay [10], and Kachina [14] all take the ZKP-based approach to private computation. Note that Kachina formally models this approach and presents a protocol for constructing such smart contracts, thereby providing a theoretical foundation for future works in the space.

Seeking to extend the limited scripting ability of Zerocash, Zexe adopts the ZKP-based approach to allow for flexible conditional payments. Zexe generalizes the idea of coins as "records" with some data payload (similar to coins

[5] We do not consider this work further since it is not peer-reviewed as of this time. However, it represents an interesting example of the evolution of private computation on blockchain.

with scripts attached to them). Each record has a birth and death predicate to control spending. To spend a coin, a user must show (via a ZKP) that the old death predicate and the new birth predicate have been simultaneously satisfied. Unfortunately, it is unclear how to support loops using their system.

As the ZKP-based approach can be difficult for developers to implement correctly, Zkay proposes a language in which to write smart contracts that better identifies whom private values belong to. Zkay also shows how to transform contracts written in their zkay language into public ones that can be deployed atop Ethereum. While their work uses the zkay language to realize the ZKP-based approach, the language could also be used in the HE-based approach to help developers identify the owners of private data.

Unfortunately, the ZKP-based approach can also be very computationally intensive for the user. Producing the ZKP needed in this approach can easily take the user over a minute, even on a powerful machine [9,10,21]. One potential solution to this problem is to delegate proof generation to some semi-trusted third party; we view this as a modified ZKP-based approach.

Hawk adopts this modified ZKP-based approach. Rather than asking the data owner (i.e. the user) to perform the computation and produce the appropriate ZKP himself, Hawk instead delegates this work to a semi-trusted manager. The manager is trusted with maintaining the privacy of the users' inputs, but not trusted for correct execution of the computation. Note that Hawk focuses on extending the Zerocash protocol and thus uses the UTXO model. Zexe proposes a similar such variant in their work, which they call "delegable decentralized private computation". Users provide their inputs to a worker who is responsible for producing the appropriate ZKP. Unlike Hawk, this variant is limited to operating on a single user's inputs. The user's signature will still be needed to authorize the transaction. As Zexe also seeks to provide anonymity, a randomizable signature scheme is used here to prevent linking across different signatures. Thus, by delegating proof generation to a semi-trusted manager (as in Hawk) or worker (as in the delegable variant in Zexe), the computational burden on the user can be significantly relaxed. In exchange, the user loses some privacy by sharing her data with some 3rd party.

Concurrency. Supporting privacy in the account model creates a concurrency challenge. Each user maintains an account with some private (encrypted) balance associated with it. As part of a confidential transaction, the sender (Alice) will need to produce a ZKP with regards to her current state which includes her private balance. If Bob sends Alice currency after her transaction has been submitted but before being confirmed, her transaction will end up being rejected as the ZKP is no longer valid (since the state has changed). This is particularly problematic when there are fees associated with transactions, as in Ethereum.

To solve this problem, Zether proposes the use of epochs which consist of some fixed number of blocks. Transactions are processed in epochs, with funds rolled over at the end of an epoch to prevent transactions from being rejected due to state changes. Users must submit confidential transactions at the start of

an epoch to ensure they are processed in the same epoch. While this approach suffices for handling confidential transactions, it's unclear how (or if) this rollover process could handle concurrency conflicts for private smart contracts spanning multiple epochs. Kachina instead relies on "state oracle transcripts" to resolve concurrency issues, which are responsible for recording the operations performed on a contract's state. Contracts can make queries to these oracles as needed. Yet, how feasible such oracles would be in practice is unclear. Differently, Zkay does not discuss how to resolve concurrency conflicts in its work at all. This is likely due to the fact that they are focused on providing a language and type system for the ZKP-based approach, rather than on the approach itself.

While the account model may seem like the most natural way to support private smart contracts, it poses a unique concurrency challenge that the UTXO-model does not suffer from.

Anonymity and techniques. Private computation schemes employ two techniques to support anonymity—ring signatures and private anonymous channels. We briefly look at how Zether and Zexe support anonymity in their works. Zether follows a similar approach to Monero, combining ring signatures with range proofs to hide the sender's account. Unfortunately, users can only initiate one anonymous transaction per epoch to prevent double-spending attacks. Zexe, on the other hand, takes an entirely different approach to support anonymity; they assume each user has a private anonymous channel to every other user. However, no discussion around implementing these channels is provided to assess feasibility.

The cost of privacy on Ethereum. Both Zether and Zkay seek to support privacy on Ethereum. Regardless of the approach taken, working on Ethereum presents its own unique set of challenges. Certain operations in Ethereum are offered at a reduced cost via "precompiles." As privacy solutions are heavy on cryptographic operations, ideally many of these operations should already be supported as precompiles to reduce the overall cost. Unfortunately, for Zether, this is not the case. Performing a confidential transaction on Ethereum using Zether costs over 7.1 million gas, with the majority of the cost coming from elliptic curve operations. At the time of Zether's proposal, a confidential transaction costed the sender around $1.5 USD [8]. With current gas prices,[6] the same transaction now costs over $1000 USD. Zkay fairs a bit better; they push computation offline and are poised to take better advantage of precompiles with their pairing-based zk-SNARKs. Their quoted costs depend on the particular contract being implemented (e.g. medical statistics, reviews) along with the proposed transformation. This makes it hard to directly compare with Zether's confidential transaction cost. However, the cost tends to range around 10^6 gas to obtain a transformed private contract, with verification costing roughly the same amount [10]. At the time of Zkay's proposal, this worked out to around $0.50 USD [10]; this same transaction now costs over $165 USD.

[6] 68 gwei/gas, 1 ETH = $2445 USD

Given the high gas costs and the rapidly fluctuating cost of ETH, supporting privacy on Ethereum may not make financial sense until cryptographic operations can be provided at a significantly reduced cost.

Discussion. Advantages depend on the role played in the system (user vs. miner). For users, the HE-based approach reduces their overall computational burden by pushing execution of the private computation to the miners. Consequently, this approach can be expensive for the miners as it requires them to perform the computation (in addition to checking the ZKP). Thus, the HE scheme must be chosen carefully with performance in mind so as to reduce the miners' time spent executing the homomorphic computation.

Extending the HE-based approach to support FHE presents additional efficiency challenges. Recent open-source libraries (such as Microsoft's SEAL supporting the BFV scheme) boast of fast execution time for homomorphic operations, with homomorphic multiplication and refreshing taking less than 1 second on a modest machine [51]. Such results appear promising as they keep the miner's execution time down. However, ciphertext size poses a problem for FHE, particularly when working in the blockchain setting where on chain information must be minimized. The resulting ciphertext of a single homomorphic multiplication operation surpasses over 100 kilobytes in size [52]. For reference, confidential transactions tend to range in size from hundreds of bytes (for Zerocash with highly efficient zk-SNARKs) to a single digit kilobyte (when less efficient Bulletproofs are used) [7,20,8]. Thus, it's unclear how feasible it would be to store FHE ciphertexts on chain.

For miners, the ZKP-based approach can reduce their overall computational burden by pushing a majority of the work client-side and offline. Consequently, this approach prioritizes blockchain throughput. Highly succinct ZKP proof systems (such as those with constant proof size) can also be used to manage ledger growth. Unfortunately, these same proofs can be quite expensive for the user to generate even with powerful machines. For the works surveyed in this paper, the ZKP (proving the update was done correctly) for even a simple computation can easily take over a minute for the user to generate on a powerful machine [9,10,21]. Even more concerning, these times are quoted for some of the most efficient proof systems using trusted setups and non-universal reference strings. As Zexe, Zkay, and Hawk are proposed now, proof setup would need to be repeated for new applications. Ideally, a universal proof system should be chosen here to prevent the need for repeating a costly proof setup process (with an MPC ceremony) for new applications.

Solutions without ZKPs. The two remaining privacy-preserving solutions (Arbitrum and Ekiden) require stronger trust assumptions via trusted managers or trusted hardware. As the two solutions discussed here do not follow a general enough framework, we abstain from comparing them directly with the HE and ZKP-based approaches. Both introduce a third party (the other two parties being

users and miners) to aid in computation and improve scalability. Additionally, both focus on I/O privacy and do not make use of ZKPs in their solutions.

We view users as the data owners who want to run a private computation on their inputs. In Ekiden, users delegate this computation to a third party with a trusted execution environment (TEE) [12]. Ekiden refers to this party as "compute nodes" as they are required to perform the computation and attest to the correctness of the update using digital signatures; this signature is only known to the trusted hardware. Thus, miners only need to check the resulting signatures. Ekiden's approach has the advantage of reducing both the user's and the miner's computational burden by outsourcing the private computation to TEEs. However, their approach requires putting trust in hardware (Intel SGX, for example) which has suffered from various security attacks over the years [53,54].

In Arbitrum, smart contracts are implemented as standalone virtual machines (VMs) which are managed by some pre-determined set of managers [13]. These managers are tasked with ensuring that state changes from the VM are performed correctly and provide a role separate from that of the blockchain miners. Managers behave optimistically; they accept state changes without repeating the computation on-chain each time unless there is dispute between themselves. In the case of a dispute, a bisection protocol occurs with a security deposit. Thus, correctness is guaranteed so long as at least one manager is honest. Unfortunately, these managers are trusted to maintain the privacy of users' inputs and not reveal them to others. Like Ekiden, Arbitrum has the advantage of minimizing miners' work by only requiring them to check signatures (assuming no disagreement between managers has occurred). However, it may possibly be non-trivial to implement all smart contracts as VMs. It's also unclear how managers would be chosen for each VM and how many would be needed for users to feel reasonably assured that at least one manager is honest.

4.4 Function Privacy

Zexe [9] is the only work providing function privacy. Coins in Zexe have birth and death predicates associated with them (essentially scripts) that control how and when coins are consumed [9]. Thus, function privacy translates to hiding the scripts associated with these coins. Users will need to prove in zero-knowledge that both the previous coins' death predicates and the new coin's birth predicates have been satisfied.

While this framework allows a user to hide how her coins were or can be used, it's unclear how to support interoperation between coins belonging to different users if the scripts associated with them are hidden. Users would not know what conditions need to be satisfied to be able to consume a coin. Presumably, parties would coordinate and share such information offline with one another (sharing information in the clear offline but maintaining on-chain privacy). This issue isn't unique to Zexe but exists for providing function privacy more generally. In an account-based model, function privacy may translate to hiding the smart contract's code. Rational users are unlikely to participate in a smart contract when they do not even know what operation is being performed on their data;

thus, function privacy makes sense only when the smart contract author is the sole user of the contract (i.e. the only user providing its inputs) or if some mutually trusted parties determined the computation and inputs to be provided offline.

5 Implications and the Road Ahead

Privacy-preserving solutions encounter several challenges, leading to a number of open problems. We cover both technical (i.e. expanding functionality) and non-technical (i.e. usability and compliance) fronts, touching upon these issues briefly. Our goal is not to provide a comprehensive review (which deserves an SoK in its own right), but to highlight potential directions for future work.

5.1 Privacy for Multi-user Inputs

Neither the HE-based approach nor the ZKP-based approach discussed in Section 4 support arbitrary computation on encrypted inputs belonging to different users out of the box. In cryptography, two primitives can be used to support computation with multi-user input privacy—multiparty computation and multi-key FHE. We consider how the HE-based approach could be extended to support multi-user input privacy using multi-key FHE. Similarly, we also look at how the ZKP-based approach could be extended to support this capability using MPC.

Extending the HE-based approach. Multi-key FHE supports homomorphic computation over encrypted inputs belonging to different users (hence encrypted with respect to different keys) [32]. Any party can perform this homomorphic computation but, to ensure semantic security, the output must be jointly decrypted using all the corresponding secret keys.

In extending the HE-based approach, we could instead request that the encryption scheme used to support private computation be a multi-key FHE scheme.[7] A user could still perform computations on her own inputs as she would if using single-key FHE. However, now, she could also request computations on various combinations of her and other's encrypted inputs. Each user will still need to prove that some conditions on her own inputs hold via a ZKP. Miners will check these ZKPs and then perform the requested computation directly on the encrypted inputs.

Advantages to such an approach include that no coordination is needed for the homomorphic computation since anyone can perform it. Additionally, recent schemes [32] provide a one-round decryption process. However, to decrypt, each participating party needs to broadcast her share (a partial decryption) to the others. This opens up a fairness issue; what if one party observes all the partial decryptions (so that she can decrypt the result) but refuses to share her own partial decryption with the others? Decryption would likely take place offline and coordinating this process may be non-trivial. Moreover, multi-key FHE

[7] This idea is briefly proposed in smartFHE [50].

schemes with one round decryption rely on either trusted setups [32] or garbled circuits [55]. Furthermore, multi-key FHE is still incredibly inefficient and, thus, currently impractical for the blockchain setting.

Extending the ZKP-based approach. In a similar vein, MPC can be used to extend the ZKP-based approach to allow for operating on private inputs coming from several users [9]. These users can perform any MPC protocol offline such that this protocol will not only produce the output (which can be private or public), but also a ZKP attesting to the correctness of this output. MPC literature offers a variety of protocols with different trade-offs in terms of communication complexity, interactivity, and security guarantees.

Nonetheless, this approach increases the load on end users who need to coordinate the computation and stay online during execution. It also inherits any limitations coming from the underlying MPC protocol such as the need for additional machinery to address lack of fairness [56] or the honest majority constraint to preserve security [57,58]. On the positive side, MPC continues to witness huge interest and advances, leading to the development of more efficient protocols for various settings [59,60].

5.2 Usability

Supporting private computation means that regular code (that operates on public inputs) must be converted to a format that operates on private inputs. This can of course be done by utilizing the proper cryptographic primitives and protocols. However, even if these protocols are already implemented, developers who do not have a deep knowledge of cryptography may not invoke the right primitives or use them correctly, leading to privacy violations. Additionally, end users (who are just invoking this code) are still responsible for producing the right format of private input (e.g. ciphertexts accompanied with ZKPs). This is in addition to vetting the code of privacy-preserving smart contracts or applications to ensure their correctness and security.

The above issues trigger several lines of work seeking to improve usability. Code conversion from public to privacy-preserving format can be automated by building compilers. Such compilers could take regular code as input, (e.g. smart contracts in Solidity for Ethereum [61]) and produce a functionality-equivalent one that operates on private inputs and produces private outputs. This approach has deep roots in MPC literature where many compilers, for various MPC paradigms and settings, have been developed (the interested reader can check [62] for a survey of these compilers). Zkay, to the best of our knowledge, is the first scheme to present a compiler for private smart contracts over Ethereum, an important step on this path.

On the end user's side, preparing the right input format can be part of the wallet software. Hence, such a process can be made as transparent as possible to the user. However, automating the analysis of private smart contracts to verify that they preserve privacy is a more challenging task. Although formal verification tools could be extended and used here, in a similar way to those

that were developed to handle security threats in public smart contracts [63], we do not foresee this as being a simple task.

5.3 Regulatory and Societal Considerations

Blockchain and cryptocurrency introduce a new relationship with finance, cultivated with the term of DeFi (decentralized finance). Works within this area often seek to build decentralized blockchain-based versions of conventional financial services, such as loan management or insurance claim processing. Privacy is a natural requirement here; no one wants their activity record to be public. However, this begs the question of how to comply with known regulations that banks adhere to, such as know your customer policy (KYC), taxes, and general data protection regulation (GDPR). Furthermore, privacy-preserving solutions can be utilized by malicious attackers to hide their tracks. For example, the Silkroad website [64] (a darknet drug market) used Bitcoin to handle payments, thinking that it is fully anonymous. Such issues serve as one of the main motivations behind developing permissioned blockchains that allow for interoperability with regulated, conventional financial services [65,66]. Developing powerful privacy-preserving solutions with compliance and cheating detection capabilities is a potential path to get the best of both worlds. This goal adds to the complexity of the problem; addressing privacy in highly distributed environments such as blockchain is challenging, let alone determining how to handle compliance.

6 Conclusion

Having existed for well over a decade, blockchain proves to have a technologically revolutionary role beyond just currency transfer. However, privacy is a huge concern, especially for applications dealing with sensitive data. We present the first systematization of knowledge of privacy-preserving solutions developed thus far for blockchain. Our work provides a critical study of the design paradigms and approaches these solutions followed. It also highlights challenges and open questions related to efficiency, usability, and technical avenues to advance the state-of-the-art. We believe that the knowledge, insights, and perspective provided by this work make for a timely contribution given the increasing interest in addressing privacy for blockchains.

References

1. F. Reid and M. Harrigan, "An analysis of anonymity in the bitcoin system," in *Security and privacy in social networks*. Springer, 2013, pp. 197–223.
2. D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *International Conference on Financial Cryptography and Data Security*. Springer, 2013, pp. 6–24.
3. P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 469–485.

4. A. Biryukov and S. Tikhomirov, "Deanonymization and linkability of cryptocurrency transactions based on network analysis," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 172–184.
5. S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 170–189.
6. I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed e-cash from bitcoin," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 397–411.
7. E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 459–474.
8. B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 423–443.
9. S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 947–964.
10. S. Steffen, B. Bichsel, M. Gersbach, N. Melchior, P. Tsankov, and M. Vechev, "zkay: Specifying and enforcing data privacy in smart contracts," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1759–1776.
11. G. Zyskind, O. Nathan *et al.*, "Decentralizing privacy: Using blockchain to protect personal data," in *2015 IEEE Security and Privacy Workshops*. IEEE, 2015, pp. 180–184.
12. R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200.
13. H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1353–1370.
14. T. Kerber, A. Kiayias, and M. Kohlweiss, "Kachina-foundations of private smart contracts," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 543, 2020.
15. B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 238–252.
16. J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.
17. 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)*. IEEE, 2018, pp. 315–334.
18. N. Alsalami and B. Zhang, "Sok: A systematic study of anonymity in cryptocurrencies," in *2019 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 2019, pp. 1–9.
19. S. Noether, A. Mackenzie *et al.*, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, 2016.
20. P. Fauzi, S. Meiklejohn, R. Mercer, and C. Orlandi, "Quisquis: A new design for anonymous cryptocurrencies," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2019, pp. 649–678.

21. 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)*. IEEE, 2016, pp. 839–858.
22. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
23. G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, 2014.
24. J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 104–121.
25. J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 281–310.
26. R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.
27. O. Goldreich, *Foundations of cryptography: volume 1, basic tools*. Cambridge university press, 2007.
28. T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.
29. C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.
30. Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.
31. J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.
32. P. Mukherjee and D. Wichs, "Two round multiparty computation via multi-key fhe," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2016, pp. 735–763.
33. M. Van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, "Fully homomorphic encryption over the integers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 24–43.
34. C. Gentry, S. Halevi, and N. P. Smart, "Fully homomorphic encryption with polylog overhead," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 465–482.
35. Z. Brakerski and V. Vaikuntanathan, "Efficient fully homomorphic encryption from (standard) lwe," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 831–871, 2014.
36. R. Canetti, S. Raghuraman, S. Richelson, and V. Vaikuntanathan, "Chosen-ciphertext secure fully homomorphic encryption," in *IACR International Workshop on Public Key Cryptography*. Springer, 2017, pp. 213–240.
37. N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, "Succinct non-interactive arguments via linear interactive proofs," in *Theory of Cryptography Conference*. Springer, 2013, pp. 315–333.
38. M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, 1988, pp. 1–10.
39. M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 784–796.

40. R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, no. 1, pp. 143–202, 2000.
41. S. Bowe, A. Gabizon, and M. D. Green, "A multi-party protocol for constructing the public parameters of the pinocchio zk-snark," in *International Conference on Financial Cryptography and Data Security*. Springer, 2018, pp. 64–77.
42. B. Abdolmaleki, K. Baghery, H. Lipmaa, J. Siim, and M. Zajac, "Uc-secure crs generation for snarks," in *International Conference on Cryptology in Africa*. Springer, 2019, pp. 99–117.
43. J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks," in *Annual International Cryptology Conference*. Springer, 2017, pp. 581–612.
44. I. Damgård, "Towards practical public key systems secure against chosen ciphertext attacks," in *Annual International Cryptology Conference*. Springer, 1991, pp. 445–456.
45. A. Kosba, Z. Zhao, A. Miller, Y. Qian, H. Chan, C. PAPAMAN-THOU, R. Pass, S. ABHI, and E. SHI, "C c: A framework for building composable zero-knowledge proofs," *Cryptology ePrint Archive, Report 2015/1093*, 2015.
46. G. Danezis, C. Fournet, J. Groth, and M. Kohlweiss, "Square span programs with applications to succinct nizk arguments," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2014, pp. 532–550.
47. ZKProof, "Zkproof community reference," December 2019.
48. A. Miller, M. Möser, K. Lee, and A. Narayanan, "An empirical analysis of linkability in the monero blockchain," *arXiv preprint arXiv:1704.04299*, 2017.
49. G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn, "An empirical analysis of anonymity in zcash," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 463–477.
50. R. Solomon and G. Almashaqbeh, "smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption."
51. C. Mouchet, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Computing across trust boundaries using distributed homomorphic cryptography." *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 961, 2019.
52. "Microsoft SEAL (release 3.5)," https://github.com/Microsoft/SEAL, Apr. 2020, microsoft Research, Redmond, WA.
53. J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel sgx," in *Proceedings of the 10th European Workshop on Systems Security*, 2017, pp. 1–6.
54. J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 991–1008.
55. P. Ananth, A. Jain, Z. Jin, and G. Malavolta, "Multi-key fully-homomorphic encryption in the plain model," in *Theory of Cryptography Conference*. Springer, 2020, pp. 28–57.
56. A. R. Choudhuri, M. Green, A. Jain, G. Kaptchuk, and I. Miers, "Fairness in an unfair world: Fair multiparty computation from public bulletin boards," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 719–728.
57. P. Ananth, A. R. Choudhuri, A. Goel, and A. Jain, "Round-optimal secure multiparty computation with honest majority," in *Annual International Cryptology Conference*. Springer, 2018, pp. 395–424.

58. Y. Lindell and A. Nof, "A framework for constructing fast mpc over arithmetic circuits with malicious adversaries and an honest-majority," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 259–276.
59. M. Keller, V. Pastro, and D. Rotaru, "Overdrive: Making spdz great again," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 158–189.
60. I. Damgård, C. Orlandi, and M. Simkin, "Yet another compiler for active security or: Efficient mpc over arbitrary rings," in *Annual International Cryptology Conference*. Springer, 2018, pp. 799–829.
61. "Solidity," https://solidity.readthedocs.io/en/latest/.
62. M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic, "Sok: General purpose compilers for secure multi-party computation," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 1220–1237.
63. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.
64. "Silk road marketplace," https://en.wikipedia.org/wiki/Silk_Road_(marketplace).
65. "Hyperledger fabric," https://www.hyperledger.org/use/fabric.
66. "Qourum," https://consensys.net/quorum/.