SoK: Privacy-Preserving Computing in the Blockchain Era

Ghada Almashaqbeh[1] and Ravital Solomon[2]

[1] University of Connecticut, ghada@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. Privacy is a huge concern for cryptocurrencies and blockchains as most of these systems log everything in the clear. This has resulted in several academic and industrial initiatives to address privacy. Starting with the UTXO model of 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 records (e.g. concurrency and private coin tracking to prevent double spending). 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 study the zero-knowledge proof systems used in supporting blockchain privacy, categorizing them based on their key features and limitations. Then, we develop a systematization of knowledge framework—using which we classify the state-of-the-art privacy preserving solutions based on several dimensions such as supported functionality, work model, where the private computation is performed, and cryptographic primitives used. Finally, we touch upon challenges including limited functionality, practicality, and accommodating new developments.

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 [66,67,51,13]. 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. Although front-running is a fairness issue, lack of privacy also makes users susceptible to front-running attacks [32]. 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 [54,68,22,18,71,76,26,45,48]. 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 [68,22] and various zero knowledge proof systems [60,41,23]. 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. 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 com-

putation, 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 (1) where the private computation is performed and (2) the design paradigm it uses to enable the private computation. Most schemes follow one of two paradigms that we call *homomorphic encryption-based* or *zero knowledge proof-based*. We go on to assess the benefits and tradeoffs of these paradigms.

After that, we briefly discuss technical proposals to address private computing for multi-user inputs, along with efficiency and trust issues (Section 5). Throughout these discussions, we seek to provide 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 [8] 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 [59] is one of the first private deployed cryptocurrencies, with numerous subsequent academic works on it. Zerocash [68] is a peer-reviewed paper with a resulting operational project Zcash. Quisquis [34] is a peer-reviewed work proposing a unique system model (account UTXO hybrid).
- Computation: Hawk [49] is one of the first peer-reviewed works in private blockchain computing. Zether [22] 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 [18] is a peer-reviewed work, resulting in the operational project Aleo. Zkay [71] is a peer-reviewed work and operational project with an extensive open-source codebase to support private smart contracts on Ethereum. Kachina [48] is a peer-reviewed work from IOHK, one of the first theoretical works to formally model private smart contracts. Arbitrum [45] is a peer-reviewed work,

with a resulting operational project from Offchain Labs. Ekiden [26] 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—such 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 [58]. About half a decade later, Ethereum went on to pioneer an account-based model [74]. 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 a set of security properties [15,36,61]. Informally, a ledger $\mathcal{L}$ is secure if it satisfies the following [36]:

- 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.
- 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.

Persistence covers what is called consistency and future self-consistency in [61]. Liveness includes chain growth and quality (i.e., a ledger $\mathcal{L}$ records only valid transactions and blocks), defined in [61]. [15] also adds fairness, which states that miners will collect rewards in proportion to the mining power they put in the system.

2.2 Privacy Domain in Blockchains

Privacy shares the general theme of hiding user data. However, in the context of blockchain, 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 [68,22]:

- 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 [39].

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. An additively homomorphic commitment is valuable in private payments as it allows for updating a private (committed) account balance by adding and subtracting (committed) currency amounts. Pedersen commitments [62] 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 one 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 purpose as homomorphic commitments—updating an encrypted account balance by adding and subtracting (encrypted) currency amounts. 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 [37] and was followed up by a large body of works devising more optimized constructions [20,33,57,73,38,21,25]. 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,

[3] Note that for simplicity we represent homomorphic addition and multiplication using '+' and '·'. 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 $m_1 + m_2$).

a statement x, and a witness ω for x and outputs a proof π proving correctness of x (that it satisfies the NP relation). Lastly, Verify takes pp, statement x, and π and outputs 1 if the proof is valid (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—such that proof size is constant and verification time is linear in the size of the input, regardless of the circuit size representing the underlying NP relation. A ZKP system that satisfies all four of these properties is denoted as a zk-SNARK (zero knowledge succinct non-interactive argument of knowledge). Formal definitions of ZKP systems and zk-SNARKs can be found in [39,14].

ZKPs are heavily utilized in private cryptocurrency 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 [12] or garbled circuits [11].

An MPC protocol is secure if it satisfies three properties: correctness, privacy, and fairness. Correctness ensures 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 ensures 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 properties 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 [24] 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 ZKPs) [19,7]. As we will discuss later, MPC can also be utilized to execute off-chain private computations over inputs coming from multiple users.

Proof System	**PHGR13** [60]	**PHGR13+ Kosba** [50]	**Bulletproofs** [23]	**GM17** [43]
Used in	Zerocash	Hawk*	Quisquis, Zether*	Zexe, Zkay
Universal	X	X	✓	X
Transparent	X	X	✓	X
Prover Time	Quasilinear	Quasilinear	Linear	Quasilinear
Verifier Time	Linear	Quasilinear	Linear	Linear
Size	Constant	Quasilinear	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 (specifically Hawk uses a variant of PHGR13 and Zether uses a variant of Bulletproofs).

3 A Bird's Eye View of 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. Accordingly, research in ZKPs has exploded in the past years, with a goal of constructing lightweight ZKPs for the blockchain setting. Of the 10 works we survey in this paper, only 2 of them do not make use of ZKPs.[4]

These proof systems share many features in common as they 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 discuss how these features affect deployment of privacy-preserving computing solutions in blockchain.

3.1 Proof Systems Used

We examine three major proof systems ([60], [23], [43]) 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 [60] 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 non-falsifiable security assumption [29].

[4] Kachina [48] 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.

PHGR13 was first used in Zerocash to achieve succinct proofs for private currency transfer [68]. 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 [41]. Hawk [49], a private smart contract platform, requires this feature and applies Kosba's transformation ([50]) to achieve this.

Bulletproofs and variants. Bulletproofs [23] 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 [34] (and eventually in the operational project for Monero [59]). 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 [22].

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 [31]. Unlike PHGR13, GM17 provides simulation extractability out of the box. The private computation schemes Zexe [18] and Zkay [71] both use GM17 as the basis of their constructions.

3.2 Flexibility

In looking at flexibility, we emphasize universality—such that a single reference string be used to prove any NP statement [75]. However, a challenges arises in providing lightweight ZKPs; the most lightweight constructions in practice are non-universal.

Universality. 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 [68], adopted a non-universal ZKP scheme. Subsequent private payment works (like Quisquis [34] 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 each new application. This setup process can be expensive to perform, calling into question the practicality of supporting arbitrary applications via non-universal ZKPs. 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.

3.3 Security

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. 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.

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 [68]. A popular mitigation strategy is to distribute trust by employing multi-party computation so that many parties can participate in the setup process [19,7]. 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.

One of the first documented MPC ceremonies for generating system parameters was for Zcash [6] (the operational project implementing Zerocash). The six participants went to great lengths to ensure honest generation—including air gapping their machines, recording protocol communication via append-only DVDs, and physically destroying their hardware after the process was complete. More recent MPC ceremonies [1,4] to generate parameters have involved larger numbers of participants and often volunteers.

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.

3.4 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—time and space overheads—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 [60] 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 [68]. Later constructions employed proof systems with both linear and quasi-linear proving times. In terms of proof verification, almost 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 [23]. 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 [23].

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 Ze-

rocash takes over 4 minutes (using a modest machine) [18,68]. 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 gravitated towards 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 [68].

Private computing solutions with universal and transparent ZKPs, such as Quisquis and Zether, produce logarithmic-sized proofs [23]. 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) [23].

4 The Landscape: Existing Privacy-Preserving Solutions

In this section, we review the solutions 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 blockchain records.

4.1 Systematization Framework

We develop a systematization of knowledge framework to fit surveyed solutions into relevant categories based on distinguishing features. As shown in Figure 1, our framework follows the evolution of these solutions with respect to supported functionality. It also classifies them based on other dimensions such as work model (UTXO vs. account model), where a private computation (if any) is performed (off-chain vs. on-chain), along with the core cryptographic building blocks used to enable privacy. We utilize the main three categories (with respect to the supported functionality dimension) to facilitate discussion in this section, with

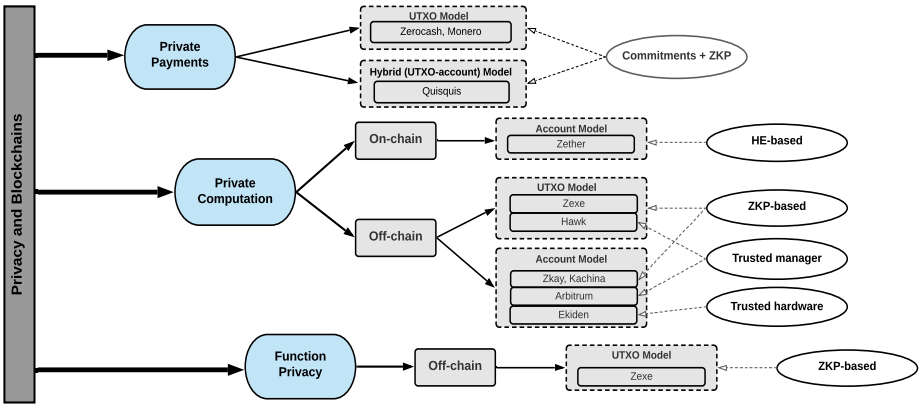

Fig. 1: Systematization framework diagram. ZKP-based and HE-based refer to approaches we define in Section 4.3.

subcategories based on other dimensions presented under each of the main categories.

Private payments. The earliest category came as a response to Bitcoin's lack of privacy. It focuses on providing confidentiality and sender/recipient anonymity. We review three schemes [68,34,59] here.

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 to support private computation. The first is on-chain private computation (via what we call the *homomorphic encryption-based approach*), where users instruct the miners to execute arbitrary computations over private inputs and produce private outputs. The second is off-chain via what we call the *ZKP-based approach*, where the work is offloaded to the user who implements the computation locally and produces ZKPs that miners must verify before accepting the output and updating the blockchain state. Other constructions under the off-chain category do not follow the ZKP-based approach; instead, they rely on alternative techniques (i.e. trusted hardware/managers) to achieve their goal. The private computation category encompasses seven schemes [22,18,71,45,26,48,49].

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). Function privacy can also be useful to hide the exchange of user-defined assets since leaking the function may reveal the token type. To the best of our knowledge, there are only two schemes that belong to this category [18,48]; both of them follow the off-chain ZKP-based 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 of these solutions center around hiding the user's payment activity. This translates to hiding transaction amounts and issuer/recipient addresses to provide confidentiality, anonymity, and transaction unlinkability. We study three schemes in the private payments category: Zerocash [68], Monero [59], and Quisquis [34]. 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 (besides meeting other system specific conditions), and, lastly, rely on anonymity sets to disguise the issuer and recipient's 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 (e.g. handling double spending, output range checking, and storage/memory considerations) while showing how they are handled in these schemes. Please note that most of these issues and techniques are common to the private computation category as well, 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 [34], 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. 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, Quisquis adopts an account notion and needs additive homomorphisms to update account balances based on private currency transfers. Also, 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 she owns. 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).

Handling double spending. Another challenge is preventing 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 has already been spent by checking the ledger. In private cryptocurrencies, additional machinery is required.

The core idea employed by private cryptocurrencies 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—whether unique value or capability based—require searching publicly published data on the blockchain to prevent double spending, similar to how double spending in prevented for public cryptocurrencies.

Output range checking. [5] 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's outputs 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). In contrast, Monero uses ring signatures [59] 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.

[5] Input value checking, i.e., total input value equals total output value (which is not range checking), was discussed under confidentiality.

Anonymity and transaction linkability. Anonymity can be 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—as the probability of guessing the actual input coin (and, thus, the owner's 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 (i.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. 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. Newly added private coins automatically become part of this set (called a shielded pool). Spent coins cannot be removed since they cannot be identified (otherwise anonymity would be compromised). When a sender wants to spend her coins, she provides a proof that she owns coins in the shielded pool without specifying which coins. Hence, this sender can be any private coin owner in the set. Although Monero and Quisquis can support a large anonymity set (covering all private coins in the system), 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 reuse of public keys in Monero's ring signatures allow for breaking its anonymity [55]. Additionally, the anonymity set of a deployed version of Zcash (the operational project built upon Zerocash) can be shrunk considerably using heuristics based on identifiable usage patterns [46].

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 [34]. 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 as soon as she owns a private coin.

Space/permanent memory. 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 a UTXO is not identified when it is spent to preserve anonymity), 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 such, 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.

Discussion. The updatable key primitive allows for the hybrid UTXO-account model of Quisquis which offers additional advantages. One advantage is simplifying how to handle double spending and private key bookkeeping for lightweight clients (since 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 states does not impact validity of pending transactions' ZKPs. However, in contrast to a pure account model, a client's wallet must track all UTXOs tied to an account (including all their updated public keys) 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 (one representative arithmetic circuit). Quisquis and Monero use separate techniques to ensure that the 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. Example applications of where I/O privacy is valuable include auctions, voting, user-defined assets, and decentralized exchanges.

Depending on where the private computation is performed, existing schemes can be grouped into two major categories: on-chain in which the miners operate over users' private data or off-chain in which the users themselves (or trusted third parties) compute over their inputs locally. We classify each category further

based on the approach used to enable the private computation. The on-chain category relies on one approach that we term the *homomorphic encryption (HE)-based approach*. It extends ideas from Quisquis and Monero to support more complex private computations using homomorphic operations performed by the miners on-chain.[6] The off-chain category relies on two different approaches; the first, which we call *ZKP-based approach*, builds on ideas from Zerocash by asking the user to produce a ZKP to prove she performed the off-line computation correctly—without the need for homomorphic operations. The second makes use of trusted managers/hardware to facilitate the off-chain computation (without any ZKPs). Both 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.

In what follows, we discuss each of these categories (marked by C1 for on-chain and C2 for off-chain) along with design issues common for the on-chain HE-based approach and off-chain ZKP-based approach. At the end, we discuss the off-chain non-ZKP based schemes.

(C1) On-chain: HE-based approach. The goal of this category is to support arbitrary computation with I/O privacy on-chain. All of the schemes (that we are aware of) under this category employ the HE-based approach in the account-based model.

At a high level, the HE-based paradigm works as follows. The user provides encrypted inputs for the desired computation, along with a ZKP proving that necessary (application-specific) conditions have been satisfied on her inputs. These encrypted inputs and the ZKP are posted on-chain. Miners verify the ZKP and then perform the requested computation directly on the encrypted inputs. The types of computation supported over ciphertexts are determined by the homomorphic properties of the chosen encryption scheme. If an additively homomorphic encryption scheme is used, then private computation is restricted to addition only (i.e. users can only request additive computations). To support I/O privacy for *arbitrary* computation, an encryption scheme that is both additively and multiplicatively homomorphic (i.e. FHE) is required.

Zether [22] was the first construction to employ the HE-based approach, encrypting a user's account balance and updating it via homomorphic addition. While the primary goal of Zether is to support confidential payments atop Ethereum via a new token, Zether 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 the few applications Zether considers—hiding a bid on a fixed number of items (sealed-bid auctions) or hiding a vote (confidential voting by stake size).

smartFHE [70] takes this further and uses an FHE scheme to enable building private smart contracts with arbitrary computations on users' encrypted

[6] ZKPs are still used here to prove well-formedness of encrypted inputs, but not to attest for the computation result as in the off-chain category.

inputs. In particular, it employs the BGV scheme [20], in conjunction with Bulletproofs [23] and a proof system suitable for proving lattice-based relations [65]. Building on ideas from Ethereum and Zether, smartFHE permits arbitrary public and private smart contracts.[7]

(C2.1) Off-chain: ZKP-based approach. This paradigm 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, the ledger state update (if any), were done correctly. The encrypted inputs, encrypted outputs, and ZKP are posted on chain. The miners' role here is to simply check the ZKP and then update the blockchain state accordingly. The schemes under this category include Zexe [18], Zkay [71], and Kachina [48].

Seeking to extend the limited scripting capability of Zerocash, Zexe adopts the ZKP-based approach to allow for flexible conditional payments in the UTXO model. Zexe generalizes the idea of coins as *records* with some data payload (similar to coins 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. In other words, the ZKP attests to the validity of the conditional computation performed off-chain and allows for spending the private coins. Unfortunately, it is unclear how to support loops based on private control conditions using their system.

Zkay takes the ZKP-based approach further by applying it in the context of smart contracts (i.e. account model). They observe that writing private smart contracts is an error prone process that can be difficult for developers. For this reason, they propose a language to enable developers to indicate which data is private and to which account it belongs. They also build a compiler to transform contracts written in this language into ones that can be deployed atop Ethereum (while incorporating ZKPs as appropriate for computations over private data). The compiler enforces several conditions needed to make compilation feasible, such as operating on private inputs belonging to only a single user, preventing loops with private control conditions, or requiring certain inputs to be public to allow building the underlying ZKP circuit.

Unfortunately, the ZKP-based approach can also be very computationally intensive for the user. Producing the needed ZKP can easily take the user over a minute, even on a powerful machine [18,71,49]. One potential solution to this problem is to delegate proof generation to some semi-trusted third party. 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

[7] 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.

for correct execution of the computation. By delegating proof generation to a semi-trusted manager, the user loses some privacy by sharing her data with some third party. Lastly, it should be noted that Hawk focuses on extending the Zerocash protocol and thus uses the UTXO model.

Kachina [48] lays down a theoretical foundation for privacy preserving smart contracts by presenting an ideal functionality-based definition. They realize this functionality using the ZKP-based approach. In particular, a contract creator will divide the contract state into two parts: a public on-chain state and a (local) private off-chain state maintained by the user. Kachina introduces the concept of state oracles (i.e., a way to query the ledger for particular state information) to reduce proof generation costs by permitting users to involve only the relevant parts of the public ledger state when generating the necessary ZKP proofs for off-chain computations.

Concurrency. 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. Each user maintains an account with an associated private (encrypted) balance. 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.

Handling concurrency in Kachina is more complicated; it encompasses more than account balances since their scheme involves computations that may change the ledger state of a smart contract. They introduce a function to specify dependencies between transactions. Dependencies could be an application-dependent; hence, each contract will define its own function (as part of its public state). It is the user's responsibility to produce a sequence of transactions that do not conflict.

Unfortunately, Zkay does not discuss how to resolve concurrency conflicts. This is likely due to the fact that they are focused on automating the ZKP-based approach for smart contracts by providing a language and a compiler.

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 address. Unfortunately, users can only initiate one anonymous transaction per epoch to prevent double-spending attacks. Zexe, on the other hand, assumes a model in which each user has a private anonymous channel [44] 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, many of these operations should ideally be supported as precompiles to reduce the overall cost. Nonetheless, introducing new precompiles requires the community consent as these are considered core changes to the Ethereum network protocol. Unfortunately, many of the necessary cryptographic operations used by Zether are not currently offered as precompiles; 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 cost the sender around $1.5 USD [22]. With more recent gas prices,[8] 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 cost depends 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 [71]. At the time of Zkay's proposal, this worked out to around $0.50 USD [71]; this same transaction now costs over $165 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 sup-

[8] 68 gwei/gas, 1 ETH = $2445 USD

porting 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 [56]. 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 [69]. 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) [68,34,22]. 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 systems (such as those with constant proof size) can also be used to manage ledger growth but at the cost of expensive proof generation for the user. Generating the ZKP for even a simple computation can easily take over a minute for the user on a powerful machine [18,71,49]. In Zexe, Zkay, and Hawk, proof setup would need to be repeated for new applications as they use proof systems with non-universal reference strings and trusted setups. 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.

(C2.2) Off-chain: Solutions without ZKPs. The two remaining privacy-preserving solutions (Arbitrum [45] and Ekiden [26]) fall under the off-chain category. However, instead of asking the user to perform the private computation off-chain, they rely on trusted managers or trusted hardware to do so instead (thus eliminating the need for a ZKP).

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). 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; the secret key used to produce the signature is only known to the trusted hardware. Thus, miners only need to verify the resulting signatures to ensure that the trusted hardware produced the ledger state changes. 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 [40,72].

In Arbitrum, smart contracts are implemented as standalone virtual machines (VMs) which are managed by some pre-determined set of managers. 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 unless there is dispute among 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 managers' signatures when receiving ledger state updates (assuming no disagreement between managers has occurred). However, it may be non-trivial to implement all smart contracts as VMs. It is also unclear how managers would be chosen for each VM and how many would be needed to guarantee that at least one manager is honest.

4.4 Function Privacy

Zexe [18] is the only concrete scheme providing function privacy.[9] Coins in Zexe have birth and death predicates associated with them (essentially scripts) that control how and when coins are consumed so 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 is unclear how to support interoperation between coins belonging to different users if the scripts associated with them are hidden. Users must 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. 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; hence, 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 Discussion and the Road Ahead

Privacy-preserving solutions encounter several challenges; these include not only technical aspects, but also non-technical ones such as regulatory compliance, usability, and other societal impacts. Nevertheless, in this section, we highlight potential directions for future technical work such as handling multi-user inputs, improving efficiency, and eliminating trusted setups.

[9] Kachina mentions that their private smart contract protocol can realize the functionality of Zexe, and hence, support function privacy. For this reason, we only discuss Zexe in this section.

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, there are two main primitives that 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 via MPC.

Extending the HE-based approach via multi-key FHE. Multi-key FHE supports homomorphic computation over encrypted inputs belonging to different users (hence encrypted with respect to different keys) [57]. 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.[10] 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 others' 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 [57] provide a one-round decryption process. However, to decrypt, each participating party needs to broadcast her share (a partial decryption of the computation output) 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? This require deploying additional techniques to address fairness. Also, as decryption would likely take place off-chain, to preserve privacy of the output, coordinating this process may be non-trivial. Moreover, multi-key FHE schemes with one round decryption rely on either trusted setup [57] or garbled circuits [10]. Finally, multi-key FHE is still fairly inefficient and, thus, currently impractical for the blockchain setting.

Extending the ZKP-based approach via MPC. 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. 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.

[10] This idea is proposed in smartFHE [70].

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 [27] or the honest majority constraint to preserve security [9,52]. On the positive side, MPC continues to witness huge interest and advances, leading to the development of more efficient protocols for various settings [47,30].

5.2 Customized Privacy-Preserving Solutions

There is always an ambition to provide general purpose solutions that can fit any application (one size fits all, so to speak). This is usually viewed as an advantage. However, given the performance constraints of the advanced cryptographic primitives needed to preserve privacy, one might ask: can we develop use case-specific cryptographic solutions that would be significantly more efficient than general purpose ones?

Reflecting on other well-developed privacy-preserving fields, we consider MPC. Many general purpose solutions exist; these can perform arbitrary computation over private inputs such as in garbled circuits or secret sharing-based approaches. However, a large body of works has also developed to handle specific popular functions such as private set intersection, e.g. [63,28,64], optimizing for efficiency.

As another example, we look at FHE. All known FHE schemes are lattice-based. Thus, when using the HE-based approach with an FHE scheme the first thought that comes to mind is to use a lattice-based ZKP to prove relations of FHE ciphertexts. Unfortunately, state-of-the-art lattice-based proofs [17,16] tend to be 3 orders of magnitude bigger in size the those based on elliptic curve cryptography. This problem has motivated efforts to investigate the possibility of using elliptic curve-based ZKPs to prove certain lattice-based relations. One such effort includes short discrete log proofs [65] which achieves significantly shorter proof sizes than lattice-based ZKPs. We anticipate further work on customised cryptographic solutions with respect to privacy-preserving computation.

5.3 The Future of ZKPs

Despite huge gains over the last decade, efficiency continues to be top of mind for ZKP researchers. In exchange for small proof sizes and fast verification, proof generation is often expensive for the user. One solution is to outsource this expensive task to some worker or manager when building a private computation scheme in the blockchain model. As we have briefly mentioned this in Section 4.3 for Hawk, we do not discuss this in further detail here. However, we note that this idea has continued to persist in recent constructions [18] and will likely continue to do so unless proof generation can be made significantly cheaper for private computation schemes (taking the ZKP-based approach).

A additional weakness in early cryptocurrency and blockchain projects was the use of ZKPs with trusted setups. Accordingly, the field has rapidly evolved to address such concerns by introducing new notions of trust. Trust has long been treated as a black and white issue in the ZKP literature—either a ZKP is transparent or it has an initial trusted setup process to generate common parameters. Researchers are often forced to use proofs with trusted setups in their systems for optimal efficiency (e.g. constant proof sizes). However, new notions of trust have been proposed, revealing that trust may be viewed on a spectrum. One such notion can be thought of as *updateable trust*, a hybrid approach achieved via an updateable reference string [42,53,35]. With an updateable reference string the setup process can continue indefinitely, allowing anyone to contribute if she does not trust that the previous parties who generated the parameters were honest. Unlike transparent ZKPs, state-of-the-art ZKPs with updateable reference strings [53] can achieve constant sized proofs. A number of operational projects (e.g., Zcash [5], Mina [3], Aztec [2]) are interested in using or upgrading to ZKPs with updateable reference strings. Such advancements will contribute in changing the landscape to reach a better trade-off between trust and efficiency.

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 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. Announcing aleo setup. `https://aleo.org/post/announcing-aleo-setup`
2. Aztec. `https://aztec.network/`
3. Mina protocol. `https://minaprotocol.com/`
4. Participate in our trusted setup. `https://filecoin.io/blog/posts/participate-in-our-trusted-setup-ceremony/`
5. Zcash. `https://z.cash/`
6. Zcash parameter generation. `https://z.cash/technology/paramgen/`
7. Abdolmaleki, B., Baghery, K., Lipmaa, H., Siim, J., Zajac, M.: Uc-secure crs generation for snarks. In: International Conference on Cryptology in Africa. pp. 99–117. Springer (2019)
8. Alsalami, N., Zhang, B.: Sok: A systematic study of anonymity in cryptocurrencies. In: 2019 IEEE Conference on Dependable and Secure Computing (DSC). pp. 1–9. IEEE (2019)

9. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Round-optimal secure multiparty computation with honest majority. In: Annual International Cryptology Conference. pp. 395–424. Springer (2018)
10. Ananth, P., Jain, A., Jin, Z., Malavolta, G.: Multi-key fully-homomorphic encryption in the plain model. In: Theory of Cryptography Conference. pp. 28–57. Springer (2020)
11. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 784–796 (2012)
12. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 1–10 (1988)
13. Biryukov, A., Tikhomirov, S.: Deanonymization and linkability of cryptocurrency transactions based on network analysis. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 172–184. IEEE (2019)
14. Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: Theory of Cryptography Conference. pp. 315–333. Springer (2013)
15. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: 2015 IEEE symposium on security and privacy. pp. 104–121. IEEE (2015)
16. Bootle, J., Lyubashevsky, V., Nguyen, N.K., Seiler, G.: A non-pcp approach to succinct quantum-safe zero-knowledge. In: Annual International Cryptology Conference. pp. 441–469. Springer (2020)
17. Bootle, J., Lyubashevsky, V., Seiler, G.: Algebraic techniques for short (er) exact lattice-based zero-knowledge proofs. In: Annual International Cryptology Conference. pp. 176–202. Springer (2019)
18. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 947–964. IEEE (2020)
19. Bowe, S., Gabizon, A., Green, M.D.: A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. In: International Conference on Financial Cryptography and Data Security. pp. 64–77. Springer (2018)
20. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
21. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing **43**(2), 831–871 (2014)
22. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
23. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334. IEEE (2018)
24. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of CRYPTOLOGY **13**(1), 143–202 (2000)
25. Canetti, R., Raghuraman, S., Richelson, S., Vaikuntanathan, V.: Chosen-ciphertext secure fully homomorphic encryption. In: IACR International Workshop on Public Key Cryptography. pp. 213–240. Springer (2017)

26. Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 185–200. IEEE (2019)
27. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: 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. pp. 719–728 (2017)
28. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: International Conference on Applied Cryptography and Network Security. pp. 125–142. Springer (2009)
29. Damgård, I.: Towards practical public key systems secure against chosen ciphertext attacks. In: Annual International Cryptology Conference. pp. 445–456. Springer (1991)
30. Damgård, I., Orlandi, C., Simkin, M.: Yet another compiler for active security or: Efficient mpc over arbitrary rings. In: Annual International Cryptology Conference. pp. 799–829. Springer (2018)
31. Danezis, G., Fournet, C., Groth, J., Kohlweiss, M.: Square span programs with applications to succinct nizk arguments. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 532–550. Springer (2014)
32. Eskandari, S., Moosavi, S., Clark, J.: Sok: Transparent dishonesty: front-running attacks on blockchain. In: International Conference on Financial Cryptography and Data Security. pp. 170–189. Springer (2019)
33. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
34. Fauzi, P., Meiklejohn, S., Mercer, R., Orlandi, C.: Quisquis: A new design for anonymous cryptocurrencies. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 649–678. Springer (2019)
35. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR Cryptol. ePrint Arch. **2019**, 953 (2019)
36. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 281–310. Springer (2015)
37. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
38. Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 465–482. Springer (2012)
39. Goldreich, O.: Foundations of cryptography: volume 1, basic tools. Cambridge university press (2007)
40. Götzfried, J., Eckert, M., Schinzel, S., Müller, T.: Cache attacks on intel sgx. In: Proceedings of the 10th European Workshop on Systems Security. pp. 1–6 (2017)
41. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
42. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-snarks. In: Annual International Cryptology Conference. pp. 698–728. Springer (2018)

43. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable snarks. In: Annual International Cryptology Conference. pp. 581–612. Springer (2017)
44. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography from anonymity. In: 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). pp. 239–248. IEEE (2006)
45. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1353–1370 (2018)
46. Kappos, G., Yousaf, H., Maller, M., Meiklejohn, S.: An empirical analysis of anonymity in zcash. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 463–477 (2018)
47. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making spdz great again. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 158–189. Springer (2018)
48. Kerber, T., Kiayias, A., Kohlweiss, M.: Kachina-foundations of private smart contracts. IACR Cryptol. ePrint Arch. **2020**, 543 (2020)
49. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)
50. Kosba, A., Zhao, Z., Miller, A., Qian, Y., Chan, H., Papamanthou, C., Pass, R., Shelat, A., Shi, E.: C c: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093 (2015)
51. Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in bitcoin using p2p network traffic. In: International Conference on Financial Cryptography and Data Security. pp. 469–485. Springer (2014)
52. Lindell, Y., Nof, A.: 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. pp. 259–276 (2017)
53. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2111–2128 (2019)
54. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: 2013 IEEE Symposium on Security and Privacy. pp. 397–411. IEEE (2013)
55. Miller, A., Möser, M., Lee, K., Narayanan, A.: An empirical analysis of linkability in the monero blockchain. arXiv preprint arXiv:1704.04299 (2017)
56. Mouchet, C., Troncoso-Pastoriza, J.R., Hubaux, J.P.: Computing across trust boundaries using distributed homomorphic cryptography. IACR Cryptol. ePrint Arch. **2019**, 961 (2019)
57. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key fhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 735–763. Springer (2016)
58. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
59. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
60. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: 2013 IEEE Symposium on Security and Privacy. pp. 238–252. IEEE (2013)

61. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2017)
62. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
63. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on {OT} extension. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14). pp. 797–812 (2014)
64. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on ot extension. ACM Transactions on Privacy and Security (TOPS) **21**(2), 1–35 (2018)
65. del Pino, R., Lyubashevsky, V., Seiler, G.: Short discrete log proofs for fhe and ring-lwe ciphertexts. In: IACR International Workshop on Public Key Cryptography. pp. 344–373. Springer (2019)
66. Reid, F., Harrigan, M.: An analysis of anonymity in the bitcoin system. In: Security and privacy in social networks, pp. 197–223. Springer (2013)
67. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: International Conference on Financial Cryptography and Data Security. pp. 6–24. Springer (2013)
68. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474. IEEE (2014)
69. Microsoft SEAL (release 3.5). `https://github.com/Microsoft/SEAL` (Apr 2020), microsoft Research, Redmond, WA.
70. Solomon, R., Almashaqbeh, G.: smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption
71. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.: zkay: Specifying and enforcing data privacy in smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1759–1776 (2019)
72. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 991–1008 (2018)
73. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 24–43. Springer (2010)
74. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)
75. ZKProof: Zkproof community reference (December 2019)
76. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: Using blockchain to protect personal data. In: 2015 IEEE Security and Privacy Workshops. pp. 180–184. IEEE (2015)