Received October 16, 2021, accepted November 9, 2021, date of publication November 18, 2021, date of current version December 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3129314

Verifiable Computing Applications in Blockchain

SILVIO ŠIMUNIĆ[1], DALEN BERNACA[1], AND KRISTIJAN LENAC[1,2], (Member, IEEE)
[1]Faculty of Engineering, University of Rijeka, 51000 Rijeka, Croatia
[2]Center for Artificial Intelligence and Cybersecurity, University of Rijeka, 51000 Rijeka, Croatia

Corresponding author: Kristijan Lenac (klenac@riteh.hr)

This work was supported in part by the European Regional Development Fund under grant KK.01.1.1.01.0009 (DATACROSS), and in part by the Research Project Embedded System for 3D Perception by the University of Rijeka, Rijeka, Croatia.

ABSTRACT From weak clients outsourcing computational tasks to more powerful machines, to distributed blockchain nodes needing to agree on the state of the ledger in the presence of adversarial nodes, there is a growing need to efficiently verify the results of computations delegated to untrusted third parties. Verifiable computing is a new and interesting research area that addresses this problem. Recently, new applications of verifiable computing techniques have emerged in blockchain technology for secure key management, sybil-resistance and distributed consensus, and smart contracts, while providing desired performance and privacy guarantees. In this paper, we provide an overview of common methods for verifying computation and present how they are applied to blockchain technology. We group the presented verifiable computing applications into five main application areas, i.e., multiparty approval for secure key management, sybil-resistance and consensus, smart contracts and oracles, scalability, and privacy. The main contribution of this survey is to answer two research questions: 1) what are the main application areas of verifiable computing in blockchain technology, and 2) how are verifiable computing techniques used in major blockchain projects today.

INDEX TERMS Verifiable computing, blockchain, zero-knowledge proof, verifiable random function, multiparty computation, trusted execution environment, smart contract, privacy.

I. INTRODUCTION

Verifiable computing (or verifiable computation) allows a client to delegate computation to possibly untrusted clients while retaining the ability to verify the results. This solves the problem of dishonest clients returning false results without actually performing the assigned work. Verifiable computing is not only about getting the correct result, but also about verifying that result with significantly less computational effort than was required to compute the result itself. There are several methods that can be used for verifiable computing [1], such as interactive proofs (IP) [2], zero-knowledge proofs (ZKP) [3], multiparty computation (MPC) [4], trusted execution environment (TEE) [5] and others. These concepts found their application in cloud computing, volunteer computing projects, and blockchain technology. For example, the Enigma [6] project, a decentralized computing platform with guaranteed privacy, uses MPC to jointly work on data while maintaining privacy. The Folding@home project [7], a distributed computing platform that aims to develop new treatments for a variety of diseases by simulating protein dynamics, uses a simple technique of verification by replication to check the correctness of results. The use of verifiable computing and blockchain enables new mechanisms for privacy preservation, for example of patient healthcare data in an e-Healthcare system [8] or for a secure and efficient management of data during a federated learning process [9]. Particularly interesting application areas of verifiable computing can be found in blockchain technology and its applications, which have motivated and challenged research in this field in recent years.

The associate editor coordinating the review of this manuscript and approving it for publication was Pedro R. M. Inácio.

Consider a public blockchain. It is a distributed, decentralized, and shared ledger that requires no central authority and eliminates the need for third-party verification [10]. It was first introduced for the Bitcoin cryptocurrency [10] as a public ledger to record all transactions. It provides trust when the parties involved do not trust each other. Instead of trusting an authority or intermediary, trust is transferred to a computer code. This code runs on computers called nodes that are connected in a peer-to-peer (P2P) network. The need for an intermediary is avoided by using a distributed, cryptographically secured ledger that everyone agrees to. The ledger can only be updated by consensus between all parties involved, i.e., all nodes. The data that is written to the blockchain is mathematically encrypted and digitally signed. It is grouped into blocks that are appended and cryptographically linked to a previous block, creating a chain of blocks, a growing ledger of historical records. Once the data is written and accepted by the nodes, no deletions or changes are possible, making the blockchain practically immutable. In order to participate in consensus, where identities can be anonymous, and in the presence of malicious nodes that do not follow the rules, a sybil-resistance mechanism must be in place. One such mechanism is Proof-of-Work (PoW) [10], [11], where participants must perform some computational work to validate new data with all other participants before it can be appended to the chain of blocks [12]. In PoW, network participants typically perform cryptographic computations to solve puzzles (see subsection III-B1 for more details). The high energy consumption of the PoW consensus protocol and limited transaction throughput motivated the development of alternative mechanisms such as Proof-of- Useful-Work (PoUW) [13], which focuses on solving practical problems instead of cryptographic puzzles, and Proof-of-Stake (PoS) [14], which uses stake in the blockchain, i.e. cryptocurrency supporting the blockchain network, as voting power. All of the above consensus mechanisms use some form of verifiable computing.

Some blockchains are also capable of running smart contracts, which further enrich the functionality of the blockchain. A smart contract is a program that runs on the blockchain and whose methods, when invoked, perform actions that execute logic and change the state of the blockchain. To confirm correct execution, each participant in the network must re-execute the same method with the same input, which is a waste of compute cycles and requires that all nodes have access to the same input data, which can be a problem with data outside the blockchain. Verifiable computing methods can circumvent both problems, i.e., they can reduce the need to re-execute the same computations and guarantee that the execution is done with the same data even in a distributed environment.

In this paper, we provide an overview of verifiable computing techniques, focusing on their use in the decentralised setting of blockchain technology. The main research questions we address are: 1) what are the main application areas of verifiable computing in blockchain technology, and 2) how are verifiable computing techniques used in major blockchain projects today.

In section II, we provide an overview of common concepts and technologies for verifying computation. Then, in section III, we present how they are being used today with blockchain technology in some notable projects that use these methods. Finally, we conclude with a discussion and summary of the work presented.

II. VERIFIABLE COMPUTING

In the blockchain context, where the parties involved in a computation cannot be implicitly trusted, verification of a computation may generally involve verification of the participants, the program executed to perform the computation, the data and results of the computation, and any messages exchanged between the participants during the process. In this section, we describe selected methods and concepts related to verifiable computing currently used in the blockchain implementations under consideration, with a focus on performance and privacy. First, digital signatures are discussed as the primary scheme for verifying the authenticity and integrity of a message, document, or transaction, as well as ensuring non-repudiation and accountability of participants in the blockchain system. In the next two subsections, we present various methods for verifying that the computation was performed correctly and that the results are as claimed. We then present techniques and schemes that allow multiple parties to perform the computation jointly, even when unknown adversarial participants are present. We present methods for verifiable generation of random numbers, which are an essential tool in many security protocols and especially in blockchain applications. Finally, we describe how specially designed hardware can facilitate the task of verifying computations.

A. DIGITAL SIGNATURE SCHEMES

A digital signature scheme typically consists of a triplet of efficient algorithms (G, S, V) [15]. A generation algorithm G takes no input and generates a key pair (sk, pk) where sk is the private key selected uniformly at random from a set of all possible private keys and pk is the corresponding public key. A signing algorithm $S(sk, m)$ returns a signature σ when given the sk and a message m as input. A verification algorithm $V(pk, m, \sigma)$, given the message m, public key pk and signature σ, outputs *true* if the signature is valid and *false* otherwise. The private key sk is therefore used as the signing key and the public key pk is used as the verification key. For any practical signature scheme it must be computationally infeasible to generate a valid signature t without knowing the private key sk.

Digital signatures are widely used in certificates and public key infrastructure (PKI). In any blockchain, they are used to verify the integrity and authenticity of digital messages and ensure non-repudiation [16]. Every transaction on a blockchain is digitally signed and every node must verify all transactions. Therefore, signature verification should be as fast as possible. Other desirable properties of signature schemes include small signature size, the ability to aggregate multiple signatures to save both space and time, and speed of the signing process. The following three digital signature schemes, compared in Table 1, are commonly used in blockchain implementations:

1) ECDSA SIGNATURES

ECDSA stands for Elliptic Curve Digital Signature Algorithm and is the signature scheme currently used in Bitcoin [10]. It is based on the discrete logarithm problem and uses elliptic curves over finite fields. The generated keys are shorter than keys in RSA signatures, which are often used in certificates and PKI. A public key pk is generated as a point on an elliptic curve (the curve used by Bitcoin is $secp256k1$ as standardized by the National Institute of Standards and Technology (NIST) [17]) by using the private key sk as a scalar.

2) SCHNORR SIGNATURES

Similar to ECDSA, the Schnorr signature scheme is based on the discrete logarithm problem and uses elliptic curves, but the signature is computed differently [18]. Also, with Schnorr signatures, it is possible to aggregate all signatures and public keys in the transaction into a single key and signature. Signature verification can be performed only once. This makes Schnorr multisignature transactions (MultiSig) indistinguishable from regular signatures, which has a positive impact on privacy and transaction size. The disadvantage of the Schnorr signature scheme is that it requires a random number generator for signature aggregation and an additional round of communication between each signer and the aggregator, due to the interactivity between signers. This is not practical for blockchain applications like MultiSig with cold wallets [19]. Also, $t - of - n$ multisig schemes become complex for large t and n [19] and it is not possible to combine all signatures in the block into a single signature.

3) BLS SIGNATURES

BLS (Boneh-Lynn-Shacham) signature scheme [20] uses the properties of bilinear pairing of elliptic curves and is based on the computational hardness assumption over the Diffie-Hellman problem [21]. It has the unique property that all BLS primitives (secret keys, public keys, signatures) are aggregable and can be combined in any order to form a single primitive of the same type, with no practical limit on the number of aggregated primitives. BLS primitives that correlate with each other still correlate with each other when the same arithmetic operations are performed on each of them. For example, if two signatures are aggregated that were created using the two previously aggregated secret keys and the same message hash, the new signature will also be validated against the aggregated public key. Also, unlike ECDSA, where the use of randomness within the signature results in multiple possible signatures for the same public key and message, BLS signatures are unique and deterministic. For any given combination of public key and message, there can be only one valid signature. These properties are especially relevant for various blockchain implementations [22], [23]. The disadvantage of BLS signatures is that signature verification is an order of magnitude slower than ECDSA or Schnorr signatures. However, if selected optimization techniques are implemented, the signature verification time can be significantly reduced [23].

B. VERIFICATION BY REPLICATION

The easiest way to verify a computation is to simply repeat the computation. Given a computation method C and an input I, the prover can compute the result $R = C(I)$ and send it to the verifier. Assuming that the verifier also has access to C and I, acquired either publicly or privately, he can then compute $R' = C(I)$ and evaluate $R' \stackrel{?}{=} R$. In an untrusted environment, with multiple participants, n verifiers can be randomly chosen to perform the verification. Assuming that the majority of participants are honest, there is usually a threshold t that marks the minimum quorum (often $t \geq 2$) of verifiers that must agree on the result R. If this condition is met, the verifiers can then confidently assume that the result R is correct.

The advantage of this approach is its simplicity and the technical implementation is straightforward compared to other approaches. However, this comes at the cost of a major weakness - the waste of energy caused by the repeated computation itself.

Replication can be done always or only under certain conditions. A system that needs a quick decision on whether the computation was performed correctly usually has the recomputation performed immediately by a certain number of verifiers. Other types of systems, based on game-theoretic principles [24], require recomputation only when one participant disagrees with the solution submitted by another participant. In these systems, the computation is performed only once unless the result is disputed. In case of a dispute, an application-specific dispute resolution protocol is started, which can be a kind of interactive game between a prover and a verifier. To prevent participants in these systems from making false claims, they must put down a deposit, which they lose along with their reputation if they are found to have acted maliciously.

The world's largest volunteer computing project, Folding@home [7], and many others [25], [26], use this approach. The architecture of such projects usually involves a client and many workers [27]. The client sends identical units of work from (C, I) to multiple workers $W = \{w_1, \ldots, w_n\}$, which then compute the results $R_W = \{R_{w_1}, \ldots, R_{w_n}\}$. Once the quorum is reached with a certain threshold for R_W, the result is marked as final. It is also important that these projects have some kind of reputation system to discourage dishonest actors and prevent them from submitting false results. The reputation metric can be a combination of the time taken to complete the work and the correctness of the results returned [28].

C. VERIFICATION BY PROOFS

An alternative to recomputation to verify the result of the computation is to use a proof which asserts that the computation was performed and that the result is correct. Generally, the prover has a proving method P which runs a computation method C on input I and produces $(R, \pi) = P(C, I)$, where R is the result and π is the proof of the computation. Given a computation method C, an input I, a result R, a proof of the computation π, and a verification method V, anyone can check the validity of the result by $V(C, I, R, \pi) \stackrel{?}{=} true$. In some cases $\pi = R$ holds, where the result R can serve as a proof π and its validity proves that the computation has been performed. Consider an example of searching for an occurrence of a substring that matches a regular expression in a large string. The prover can compute the result $R = (i, j)$ using the computation method C on a string S and a regular expression re, where i and j are starting and ending positions of an occurrence. Given the result (i, j) the verifier can run a separate verification method $V(S, re, i, j)$ to quickly

TABLE 1. Properties of notable signature schemes with reference to ECDSA.

	Signature size	Verification time	Signature aggregation	Unique	Deterministic
ECDSA	-	-	No	No	No
Schnorr	Long	Normal	Interactive only	No	No
BLS	Short	Slow	Yes	Yes	Yes

check whether the substring in that range matches a regular expression without having to perform the entire computation C again. If privacy is a concern and the data being worked on must remain private, the proof of the computation can be constructed in a zero-knowledge way (subsection II-C2). In this case, the verifier does not need to know the input I and can simply check $V(C, R, \pi)$. In the next subsections, we present common methods that rely on proving rather than recomputing to verify a computation.

1) INTERACTIVE PROOF

An interactive proof (IP) system is an abstract machine where computation is modeled as an exchange of messages between the prover P and the verifier V [2]. The prover has unlimited computational power and cannot be trusted, while the verifier has limited computation power and is always honest. Messages are exchanged between the verifier and the prover until the verifier is convinced by the prover that a statement is true, resulting in the verifier accepting the statement. If the prover failed to convince the verifier that a statement is true, the verifier rejects the statement. Interactive proof systems must satisfy the following two requirements [29]:

- **Completeness** – If the statement is true, the honest verifier will be convinced that the statement is true by an untrusted prover.
- **Soundness** – If the statement is false, no cheating prover can convince the honest verifier that the statement is true, except with some small probability.

An interactive proof can also be a proof of knowledge by which the prover can convince the verifier that he knows something [30]. Most frequently used interactive ways to prove knowledge are the Schnorr protocol [18] and the Σ-protocol [31]. Proofs of knowledge are also used in signature schemes, such as group signatures [32] and Schnorr signatures [18].

2) ZERO-KNOWLEDGE PROOF

A zero-knowledge proof (ZKP) is a method by which the prover can prove to the verifier that some statement is true without revealing any information other than the fact that the statement is true [33]. Assume the prover wants to prove to the verifier that he knows the value x. The challenge is to prove the knowledge of x without revealing the x itself. Besides the requirements of interactive proof systems **completeness** and **soundness**, any ZKP must also satisfy the requirement of **zero-knowledge** – if the statement is true, no verifier learns anything other than the fact that the statement is true.

There are two types of ZKP: interactive and non-interactive. In an interactive ZKP, the prover performs series of actions to convince the verifier that a particular statement is true, with some probability p. The more rounds of interaction there are, the higher the probability p will be. A non-interactive zero-knowledge proof (NIZKP) does not require any interaction between the prover and the verifier, which provides greater flexibility since the verifier can verify the proof itself at any time. It is also possible to transform some interactive ZKPs to NIZKPs using a Fiat-Shamir heuristic, which is a way of creating a signature based on an interactive proof of knowledge [34], [35].

One of the NIZKP is zk-SNARK [3], [36]–[38], which stands for "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge". *Succinct* means that the proof is just a few hundreds bytes in size, even for statements about computations that are very large, and that it can be verified quickly, usually within a few milliseconds. Every zk-SNARK consists of three algorithms [39]: generator G, prover P, and verifier V. Generator G takes as an input a secret parameter λ and a program (computation) C, from which it derives proving key pk and verification key vk. This is a one time process for any program C, and the keys are made public. The prover P takes as an input proving key pk, a public input x and a private input w, also known as a witness. The proving algorithm then generates a proof $\pi = P(pk, x, w)$ which asserts that the prover knows a witness w which satisfies the program. The verifier V can then use verifying key vk, public input x and a proof π to compute $V(vk, x, \pi)$ which returns *true* if the prover knows a witness that satisfies $C(x, w) = true$. Downside and common criticism of zk-SNARK protocol is the need for a trusted setup. A secret parameter λ is required during generation phase, which if unveiled, could allow a cheating prover to generate fake proofs without knowing a witness w. To prevent this from happening it is not unusual to use MPC (subsection II-D) during generation of the public parameters [40], [41], which is secure as long as one of the participants is honest.

This drawback in zk-SNARKs has led to the development of other NIZKPs that do not require trusted setup, such as zk-STARKs [42] and Bulletproofs [43]. While zk-STARKs are faster than zk-SNARKs during proving phase, they are slower during verification phase [44]. Bulletproofs are smaller in size than zk-STARKs, however, they are much larger than zk-SNARKs [44]. Different protocols have different limitations in terms of proof size, proving time and verification time. These are the properties that need to be considered when deciding which NIZKP to use, as also shown in Table 2 [45]–[47].

D. MULTIPARTY COMPUTATION

Multiparty computation (MPC) is a method by which multiple parties $P = \{p_1, \ldots, p_n\}$ can jointly compute a function F of their individual inputs $X = \{x_1, \ldots, x_n\}$, without any party revealing their inputs to the other parties [4], [48]. The generated output $Y = F(x_1, \ldots, x_n)$ is public and can be viewed by all parties. For example, in the blockchain context the parties are often nodes and it is assumed that some of the nodes may be potentially adversarial [49]. The nodes wish to perform a joint computation of a function while preserving basic security properties such as **correctness** and **privacy**. The privacy requirement states that nothing beyond what is absolutely necessary may be learned; more specifically, the parties may learn only the intended output and nothing else. The requirement of correctness states that each party should produce its correct output. Therefore, the adversary must not be able to cause the result of the computation to be different from the function that the parties intended to compute.

Most MPC protocols use secret sharing. Secret sharing allows a secret to be shared among multiple parties, with each party receiving a portion of the secret [50]. No single party owns the entire secret, minimizing the risk that the secret will be revealed if one party is compromised. Besides some trivial secret-sharing schemes where all shares are necessary to recover the secret, the most common schemes are those where t of n secrets are shared and where a certain threshold of adversaries can be tolerated [51]. A notable secure and efficient threshold sharing scheme based on the Lagrange interpolation polynomial is Shamir Secret Sharing (SSS) [52]. SSS is based on the premise that the dealer and the parties are honest. It is not possible to verify that (i) a dealer is sending correct shares to some or all participants and (ii) participants are not submitting fake shares during the reconstruction process [53]. Verifiable secret sharing (VSS) includes auxiliary information that allows parties to verify their shares as consistent, even if the dealer is malicious [53]. A commonly used example of a simple VSS scheme is the Feldman scheme for non-interactive VSS [54]. Publicly Verifiable Secret Sharing (PVSS) [55] explicitly requires that not only the parties involved in the scheme, but anyone can verify that the parties have received correct shares. The original secret sharing schemes [52], [56] required a dealer to generate and distribute shares of a secret to the participants. To initialize the cryptosystem securely without requiring a dealer, a Distributed Key Generation protocol (DKG) [57], [58] can be used. A group of n participants can jointly generate a secret whose shares are distributed among the participants such that any subset larger than t can use the secret. No trusted party is required.

E. VERIFIABLE RANDOMNESS

Many applications require random values that cannot be predicted in advance and must be published. The source of the randomness may be from the system itself or from a trusted third party. However, when participants do not trust either the system or a third party, a provably fair and publicly verifiable method is needed. We describe common methods that can be used to generate randomness and publicly verify its fairness.

1) VERIFIABLE RANDOM FUNCTION

A verifiable random function (VRF) is a pseudo-random function that provides a verifiable proof of its correctness [59]. For a given input x, used as a seed for random number generation, a party with private key Pr and public key Pu can compute the result $y = F_{Pr}(x)$ and the proof $\pi_{Pr}(x)$. The result is a random number that other parties can verify by computing the verification function with the known seed x, the proof π_{Pr} and the public key Pu. The result of the verification is *true* if the random number was generated correctly, *false* otherwise. In other words, evaluating the output of the VRF function for a given input does not allow the adversary to distinguish the output of VRF from a random source. Moreover, given the output of the VRF function and a prover's public key, the verifier can be non-interactively convinced that the output was correctly generated using the prover's secret key. The prover cannot say for arbitrary random looking number that for that input and his private key the same number will be generated.

VRF can be used to provide deterministic precommitments that will be revealed at a later time [60]. For example low entropy inputs (e.g., "John Doe") can be mapped to a random number and committed in advance on a public blockchain. A notable use case for VRFs is the Internet draft for NSEC5 [61] which proposes a scheme that uses VRFs to prevent offline DNSSEC zone brute-force enumeration and guarantee the integrity of zone contents even if an attacker compromises the authoritative nameserver responsible for responding to DNS queries for the zone. Some traditional applications of VRFs are resettable zero-knowledge proofs [62], micropayment systems [63], and privacy-preserving transaction escrow schemes [64]. Some prominent blockchain applications of VRFs are described in the next section.

2) VERIFIABLE DELAY FUNCTION

A verifiable delay function (VDF) is a function that takes a prescribed minimum time to compute a unique output that can be efficiently verified in a public setting [65], [66]. More precisely, for an input $x \in X$ and a VDF function $y = F(x)$, anyone can compute $F(x)$ in t sequential steps, but no adversary, even with a parallel computer with many processors, can compute the output of $F(x)$ in substantially fewer steps. Given the output y, any observer can quickly verify that $y = F(x)$. Moreover, every input $x \in X$ must have a unique valid output $y \in Y$.

VDFs were introduced as a way to prevent malicious actors from influencing an output by predicting future values.

F. TRUSTED EXECUTION ENVIRONMENT

A trusted execution environment (TEE) is a secure area on the main processor that is separate from the main operating

TABLE 2. Properties of different NIZKPs.

	Proving time	Verifying time	Proof size	Trusted setup	Quantum secure
zk-SNARK	Quasilinear	Constant	Constant	Yes	No
zk-STARK	Quasilinear	Polylogarithmic	Polylogarithmic	No	Yes
Bulletproofs	Linear	Linear	Logarithmic	No	No

system. It is an execution environment that provides features such as isolated execution of programs, integrity of computations, and confidentiality of data [5].

An application using TEE generally consists of two components. An untrusted component, a host, runs on the untrusted operating system, while a trusted component, an enclave, runs inside an isolated TEE container. TEE enables the storage of secret data that cannot be accessed by attackers, ensuring confidentiality. To ensure security and integrity, TEE only executes code authorized by other authorized code. To prove its trusted state, TEE is able to generate a signed proof, called an attestation, which can be used later for verification. This makes it impossible for other applications to alter the state or tamper with the code executed within TEE. As such, TEE provides features that can help verify computations.

Two notable hardware technologies that support TEE are ARM TrustZone [67] and Intel SGX [68]. TEEs are usually shipped with a unique private key from the manufacturer, which is then used to generate attestations. This means that in the case of a malicious or hacked manufacturer, we can no longer trust the TEE. In addition, numerous vulnerabilities have been discovered in TEEs in recent years [69]–[73]. Some of these risks can be mitigated by distributing the work among multiple workers and using threshold cryptography for secrets. TEE can be used to store encryption keys to verify the integrity of the operating system, user data for biometric authentication, and other forms of sensitive data [5]. TEE can also provide a hardware-protected random number generator with special instructions [74].

III. BLOCKCHAIN APPLICATIONS

In this section, we provide an overview of the main applications of verifiable computing in blockchain systems. We describe how the selected verifiable computing methods are used, what functionality they enable, and for what purpose. Where possible, we also list examples of practical implementations of blockchain systems that use these methods. For better readability and structure, we group the applications of verifiable computing into five thematic blockchain areas: key management, consensus mechanisms, smart contracts, scalability, and privacy.

The verifiable computing methods and their applications are summarised in Tables 3 and 4. Table 3 maps the desired goal/functionality to one or more specific verifiable computing techniques used to achieve that goal, while Table 4 provides an overview of current blockchain applications that rely on verifiable computing techniques, with notable system examples for each application.

A. SECURE KEY MANAGEMENT

Blockchain protocols are based on public-key cryptography where pairs of a public key and a private key are used to perform different tasks. Any person with the private key is able to sign transactions and there is no way to recover it if it is lost. It is therefore of utmost importance to protect the private key and keep it secret.

1) MULTIPARTY APPROVAL SCHEMES

To reduce the risk of an adversary gaining access to the private key, multiparty approval schemes are often used where there is no single point of failure.

A traditional solution for multiparty approval is to use a multisignature transaction (MultiSig) [101]. To be considered valid, a multisignature transaction must be signed by multiple private keys from different participants.

A different approach with several important advantages over MultiSig is the use of verifiable computing techniques in secure multiparty approval schemes. By using a Threshold Signature Scheme (TSS) [102]–[104] it is possible to define a (t, n)-threshold signature scheme, where at least t parties of n are required to create a signature. To sign a transaction, multiple parties participate in a secure offline protocol where each party uses its secret to generate a partial signature. When enough partial signatures have been collected, the transaction can be signed. Anyone with a public key can then verify that the transaction is valid, i.e., that it was signed with the corresponding private key. However, the signature is the result of an MPC computation and the private key is never generated. It can therefore not be stolen, destroyed or used to sign another transaction. Once the offline signature process is complete, the threshold signature appears as a single standard signature regardless of the number of approvers. This is an advantage over MultiSig because no meta-information about public key shares or signers needs to be stored when sending to a $t - of - n$ address or spending it's output. No additional software modification is required by a particular blockchain implementation to support TSS approval schemes and privacy is also improved as the address is indistinguishable from a normal address. In addition, TSS provides additional privacy and security as individual approvers can be removed, changed, or added at any time without changing the resulting signature. In contrast, MultiSig records the signature of each approver on the public blockchain for successful transactions.

2) SECRET SHARES MANAGEMENT

One of the common security practices in key management is key updating, where the user is required to change the

TABLE 3. Blockchain uses of verifiable computing methods.

Goal	Method
Hashcash solution verification	Replication, hash function
Computation integrity, solution verification	Replication, TEE
Validator selection	VRF
Transaction existence verification	Replication
Constant sized blockchain recursive verification	NIZKP
Committee selection for shards	VRF, VSS, DKG
Opening and closing state channels	MultiSig
Sidechain state verification	Replication, NIZKP
Verification of off-chain transactions	Interactive game, NIZKP
Verification of external data (oracles)	Replication, TEE, MPC
Provable and fair generation of random numbers	VRF, VDF
Off-chain computation verification in smart contracts	Replication, NIZKP, interactive game
Private computation in smart contracts	NIZKP, MPC, TEE
Threshold signature scheme	MPC
Masternode quorum selection	DKG, MPC
Private transactions	NIZKP
General private computation	MPC, TEE

TABLE 4. Verifiable computing applications examples in blockchain networks.

Method	Blockchain application	System example
Replication	PoW	Bitcoin [10], PoW blockchains
	SPV	Merkle proofs, NIPoPoWs [75]
	Sidechains	BTC Relay [76], NIPoPoWs [75]
	Smart contract data	Chainlink [77]
	Dispute resolution	Optimism [78], Truebit [24], Arbitrum [79]
Proofs (IP/NIZKP)	Succint blockchain	Mina [80]
	Sidechains	Zendoo [81], zkRelay [82]
	Rollups	Loopring [83], zkSync [84]
	Smart contract computation	ZoKrates [85], Cairo [86]
	Smart contract privacy	Hawk [87], Zether [88]
	Private transactions	Zcash [89], [90], Monero [91]
MPC	Shard committee selection	RapidChain [92]
	Smart contract data	DECO [93]
	Smart contract privacy	zkHawk [94]
	Multiparty approval schemes	(multiple projects)
	Instantly settled transactions	Dash [23]
	Chainlocks (51% attack prevention)	Dash [23]
	Private computation	ARPA [95]
VRF	Shard committee selection	OmniLedger [96]
	Smart contract randomness	ChainLink [77]
TEE	PoUW	REM [97]
	Private computation	Enigma [6]
	Smart contract data	Town Crier [98]
	Smart contract privacy	Ekiden [99], ShadowEth [100]

private keys regularly to reduce the risk of exposure. The same practice can be used in multiparty approval schemes where shares of the secret are held by n different parties. It should be possible for each party to replace the individual share of the secret sharing scheme with a new share, so that when the shares are added together, e.g. for use in the signing protocol, they are still equal to the value of the private key.

a: SECURE KEY ROTATION

In TSS this can be achieved as it is possible to generate many different and random combinations of secret distributed shares that can still be combined to generate the same secret key [105]. Key rotation can be performed proactively at regular intervals or on demand if deemed necessary to address a potential security risk. For example, an attacker might attempt to compromise one party at a time until it has the required number of shares. In a secure key rotation, the attacker must in practice compromise at least t parties in a (t, n) secret sharing scheme within the key rotation interval. For a short key rotation interval, this task is likely to be much more difficult. In a traditional MultiSig system, if one party is compromised and the adversary learns the value of the key share, a key update is required in which entirely new private keys are generated and an on-chain transaction is performed to synchronize the public and private keys and accounts. Changing a private key in a cryptocurrency, for example, results in a changed account or payment address, which is often undesirable. This is not the case with TSS, as it is possible to perform secure key rotation while maintaining a static account address.

b: USER REPLACEMENT

There are many scenarios in which it may be necessary to replace, remove, or add a party participating in a multiparty approval scheme. For example, if an employee who holds a secret share leaves the company or is replaced by another employee. In this case, his or her secret share must be invalidated and a new share generated for the new employee. Preferably, this should be done without changing the address or account protected by the secret sharing scheme. Again, both of these processes can be easily accomplished using TSS. To remove a party from a secret sharing scheme, the secure key rotation mentioned above can be performed between the remaining users. Once the key rotation has been performed, the secret share of the leaving party can no longer be used in any way, effectively removing the party from the sharing scheme. To add another party, all that is necessary is to issue another share to the new party. For example, if Lagrange interpolation is used in TSS, the share is derived from a point on the polynomial corresponding to a number associated with the new party, e.g., a hash of ID.

B. CONSENSUS AND SYBIL-RESISTANCE

In a distributed consensus, participants propose a value and later agree on that value. To be fair, consensus protocols involving anonymous identities must ensure that each participant does not cheat by having more influence on the outcome than they should [106]. A malicious participant could hide behind multiple identities and have an unfair amount of voting power. This is also called a sybil attack [107] and requires special mechanisms to either reduce the impact or ensure that this does not happen. In the next subsections, we present common sybil-resistance mechanisms used in the blockchain and explain how they use verifiable computing concepts.

1) PROOF OF WORK

To participate in the consensus on the blockchain, computing power can be used. This is the approach taken by PoW-based blockchains such as Bitcoin [10]. In a distributed system [108], the ability of nodes to reach consensus on a value is called liveness [109]. Conversely, safety is a guarantee that nodes will not come to a consensus with different values [109]. Bitcoin uses the Nakamoto consensus, which emphasises liveness over safety, and the longest chain with the most accumulated work is the source of truth. Participants in the network, known as *miners*, have an incentive to expend computational power for the chance to publish a block and thus secure the block reward. The PoW algorithm that Bitcoin uses is called Hashcash [110]. Miners who want to compete for the right to publish the next block try to find a value called *nonce*, which serves as one of the inputs to a cryptographic hash function. The output of such a function must yield a value h, such that $h < d$, where d is the current difficulty of the network. Once a *miner* finds the satisfying *nonce* value, it is broadcast to the network as part of the new block and others verify its correctness by recomputing h. In this context, *nonce* serves as a proof of the computation itself. What guarantees sybil-resistance is the fact that finding *nonce* is computationally intensive and requires a brute force approach. However, once the *nonce* is found, its verification is cheap and occurs in constant time. Therefore, the cost relation between the computation method C and the verification method V is $V \ll C$. As the blockchain grows, the computational cost of rewriting history increases exponentially [10], while the cost of verification remains constant.

2) PROOF OF USEFUL WORK (PoUW)

One of the biggest criticisms of the PoW mechanism is that it is wasteful in terms of energy [111]. Due to environmental concerns, there is active research into how to replace it with alternative algorithms, puzzles, or problems. There have also been successful attempts to replace SHA256 *mining* with solving NP-complete problems [112], [113] to achieve PoUW. To keep the protocol fair, it is important that no participant has an advantage in solving these problems because they know the solution beforehand. For this reason, randomly generated problems can be used, but their solutions are of limited use because they may have no practical applications. Other works [114]–[116] describe systems and frameworks that solve problems from specific domains such as artificial intelligence, in particular machine learning. These limitations have also led to the use of trusted hardware, i.e., TEE (subsection II-F), to perform more general computations.

One of the first algorithms to use trusted hardware for sybil-resistance was Proof of Elapsed Time (PoET), proposed by Intel for the Sawtooth Lake [117] project and based on the SGX module. Instead of performing cryptographic computations, *miners* run code that idles for a random amount of time, and the *miner* with the shortest wait time gets to lead the consensus round and reaps the reward. PoET has been shown to have some vulnerabilities and limitations. An attacker is able to successfully attack the network by compromising only a small fraction of the participants [118]. PoET also incentivizes the use of cheap and outdated hardware specifically designed for *mining*, which leads to waste, i.e., all the hardware collected does not perform useful work [97]. Moreover, reliance on a single manufacturer can be considered centralized. The Resource-Efficient Mining (REM) framework [97], which also relies on trusted hardware, addresses the risk of compromised SGX CPUs and achieves more promising PoUW. REM measures computation in the number of instructions executed for useful work and is therefore more flexible in the variety of work that can be used for PoUW. It allows users to use their CPUs for any workload, such as protein folding, machine learning, data compression, and hashing [97]. Thus, REM enables even less wasted computing power in *mining*, while maintaining security guarantees similar to PoW.

3) PROOF OF STAKE

Instead of using PoW for sybil-resistance, some blockchains opt for the PoS mechanism, where the voting power is based on participants' stake in the network [119]. This approach is very effective as it consumes a fraction of energy compared to PoW-based blockchains and helps in scaling the network. In PoS, participants can take a special role as proposers or validators who are responsible for growing the blockchain. A proposer creates a new block, which must then be checked by validators before it is appended to the blockchain. A naive approach is to use the block hash as a source of randomness and derive the probabilities for selecting proposers and validators [14]. Since transactions in the block can be reordered or omitted, this allows *miners* to keep doing so until they find the hash that would most favor them in the next round of consensus. To overcome this vulnerability, another source of randomness should be used.

An example of a project that uses verifiable random functions for its sybil-resistance mechanism is Algorand [120]. The VRF takes a private key and a value and produces a pseudorandom output along with a proof that can be used to verify the result. The VRF is used to select the leaders who propose a block and also the committee members who vote on a block [121]. The chance of being selected is proportional to the stake in the network, ensuring that a participant does not gain an advantage by creating multiple identities. Nodes participating in consensus rounds first validate the VRF proof to ensure that the associated output is correct before moving on to the next phase. Other blockchain networks such as OmniLedger [96], Cardano [122], and others [123]–[125] also use VRF as part of their consensus mechanism.

C. SMART CONTRACTS

A smart contract is a computer program running on top of the blockchain [126]. In general, smart contracts only have access to what is already on the blockchain, which was generated or put there by the network participants. The ability to connect real world, i.e. systems external to the blockchain, with smart contracts is of a great value and is one of the major milestones towards increased blockchain adoption. In next subsections, we explain how verifiable computing assists in getting data, randomness or results of general computation to the blockchain and enables to keep private the smart contract code, execution, and data.

1) DATA

Due to its nature, the blockchain can also be defined as a replicated deterministic state machine. This means that any state change at any point in time can be seen and verified by all peers in the network. Suppose there is a method M within a smart contract. When executed, method M makes a request to an external API to obtain and process some data and return the result R. If transaction T which is a call to method M, occurred at time t_i, then we can assert that the blockchain was moved from one state to another, $S \xrightarrow{T} S'_{t_i}$. The problem occurs when this transaction is sent to the network and needs to be verified. Each peer in the network will also execute T once it reaches it, but the time of execution will not be the same. This means that each peer will execute T at a different time t_j and the state transition will be $S \xrightarrow{T} S'_{t_j}$. When the data requested from the API changes or becomes unavailable, the peers will end up in different state S', breaking the invariants of the blockchain and its determinism. The same problem arises with any subsequent verification of state transitions.

This problem is also known as the *oracle problem* and consists of two parts. First, there is the question of how data can be moved between blockchains and external systems, and second, how this can be done in a secure and decentralized manner. To get at the data outside the blockchain, smart contract methods typically send out events that systems outside the blockchain listen for and respond with the data in the form of a new transaction. These systems are known as *oracles* and serve as middleware between blockchains and the real world [127]. This poses the risk of potentially insecure systems passing data to smart contracts on a supposedly highly secure decentralized blockchain network. To solve this problem while still maintaining decentralization and security properties, the approach of verification by replication can be applied. As suggested by Chainlink [77], there are generally two methods by which this can be done: in-contractor aggregation and off-chain aggregation. In the first method, multiple independent actors first fetch the data from an external source and then submit it to a smart contract responsible for reaching a quorum. This is expensive and energy consuming as each actor has to submit a separate transaction. The second method uses Schnorr signatures [18], where aggregation and agreement occurs outside the blockchain and is later submitted as a single transaction.

Other problems that arise are authenticity and confidentiality. A malicious *oracle* can change the retrieved data before it is sent to the blockchain. In a decentralized oracle network, this would require the majority of *oracles* to collaborate and carry out the attack together. Some oracle services, such as Provable [128], provide authenticity proofs along with the data [129]. The Town Crier [98] system also addresses some of these problems by using Intel's trusted hardware SGX. In particular, it guarantees that data supplied by a website has not been tampered with. It also provides confidentiality by enabling private data requests with encrypted parameters. For example, a request might include a password to log into a server from which the data is then retrieved [130]. SGX ensures that such secret data remains hidden from everyone, including Town Crier operators. Other hardware-based distributed oracles also extend this by implementing a reputation system to address problems with overloaded or malicious oracles, favoring those with the shortest response times [131]. Another protocol, DECO [93], achieves most of the same functionality but without trusted hardware. Instead, DECO is making use of a MPC to allow users to prove in zero-knowledge that data accessed via TLS [132] originated from a specific website.

2) RANDOMNESS

Smart contracts sometimes require a random number generator for scenarios such as minting non-fungible tokens (NFT) in games, picking lottery winners, or choosing leaders in consensus rounds.

Similar to Proof of Stake, a naive approach is to use the block hash as a source of randomness. However, this is not secure as it can be manipulated by *miners* [133], [134]. Other solutions, such as decentralized commitments with deposits [135], [136], are complex and costly because they require multiple participants to submit transactions to the blockchain as part of the process of selecting a random number.

For a provably fair and verifiable source of randomness, a verifiable random function can be used. Generally, a random number is generated outside the blockchain and then sent to the network along with the cryptographic proof. Chainlink has implemented VRF [137] for smart contracts, providing developers with both security and ease of use when dealing with random numbers. Each time a random number is requested, a new random number with the associated proof is generated off-chain by oracle operators and then published and verified on-chain before the random number is used in a smart contract to ensure that no tampering has occurred.

A verifiable delay function (VDF) can also be used to generate randomness in smart contracts. The VDF is used to provably delay the revelation of randomness and prevent the randomness from being influenced by the malicious party, who can choose whether or not to submit the generated random value. The delay is designed to allow the seed of the random value to be committed in an earlier block. VeeDo [138], a STARK-based VDF service, allows users to request a random value from an off-chain service, which is then verified on-chain with a corresponding proof. A longer delay increases security, as other parties that may be malicious must also wait the same amount of time to compute the VDF output. When using VeeDo, the user can choose between a 3 minute and 7 minute delay.

3) COMPUTATION

Each network participant that takes part in the smart contract execution process must perform the entire computation again as part of the verification process, which can result in a large computational overhead and network congestion.

Proofs of computation, such as zk-SNARKs, can be used to solve this problem. Since they are hard to construct for non-specialist users, high-level languages have been developed to facilitate their development [85], [86], [139]. An example of such a system for the Ethereum platform is ZoKrates [85]. It allows developers to write their zk-SNARK program in a high-level language from which Solidity code is generated, allowing for easy integration with other smart contracts. A transaction containing the result of the computation and a proof can then be sent to a smart contract, and *miners* only need to verify the proof of correctness. A provable program can also be created using Cairo [86], a high-level language for generating zk-STARKs. Since SNARKs or STARKs cannot be constructed for most programs, this verification method has limited applicability. It is therefore not particularly well suited to complex computations and is best used when some information to be kept secret and a proof of the computation must be immediately available.

Instead of relying on cryptographic proofs, computations can also be verified through an interactive game [140]. TrueBit [24] uses game-theoretic principles as part of its interactive verification protocol. This allows for heavy computations to be performed off the blockchain by a single solver, who must submit a deposit as a guarantee that the computation will be performed correctly. During a certain period of time, any verifier can redo the computation and check if the result is correct. If there are no disputes, the computation is marked as final and the solver receives his deposit back, along with a reward for performing the computation. However, if the verifier and the solver disagree on the result and the adjudication system finds verifier's solution to be correct, the solver loses his deposit and the verifier is entitled to the reward. To incentivize the verifiers' participation, the TrueBit protocol occasionally inserts errors that the verifiers try to find.

D. SCALABILITY

Scalability describes how well a system is able to handle an increased load. A system can be scaled vertically by upgrading hardware and horizontally by distributing the load across multiple machines. By combining these methods, a distributed system can achieve linear scalability. However, this becomes more difficult with blockchains because there may be malicious nodes that do not adhere to the protocol rules. Also, all events must be verifiable by every other node in the network. This raises the question of how blockchains can scale efficiently while maintaining properties such as decentralization and security. One of the metrics used to describe how well a blockchain scales is transactions per second (TPS). In the next subsections, we discuss recent advances in scaling both on-chain and off-chain and how they leverage verifiable computing.

1) ON-CHAIN

The blockchain design as originally proposed by Bitcoin [10] has not proven to be easily scalable in practice. Since then, numerous solutions have been proposed that make changes to the base layer of the blockchain. Some of these solutions include changing the blockchain data structure [80], using smaller signatures for transactions [141], and sharding [142], [143].

a: SIMPLIFIED PAYMENT VERIFICATION

In traditional blockchains, all transactions since the beginning of the network are stored and hosted by individual network participants called *full nodes* [10]. This data can accumulate over time to a size of hundreds of gigabytes. In addition, when other *full nodes* join, they must download all transactions from other participants, which takes up both bandwidth and storage. It is also possible to participate in the network as *thin* or *lightweight node* [10] where only the block headers need to be downloaded to validate transactions using the Simplified Payment Verification (SPV) method. A *thin node* acquires all block headers from a *full node* and, if needed, asks the *full* *node* for a Merkle proof of a particular transaction to verify its existence.

Some of these requirements for *thin nodes* can be relaxed by using Non-Interactive Proofs of Proof-of-Work (NIPoPoWs) [75]. NIPoPoWs enables the verification of transactions on PoW-based blockchains without the need to acquire all block headers. NIPoPoWs nodes only need to download a polylogarithmic number of block headers, which is enabled by the fact that for PoW, statistically half of the blocks produced exceed the network difficulty d [75]. If D is a number of leading zeros in the network difficulty, then half of the block hashes produced start with $D+1$ zeros, a quarter start with $D+2$ zeros, and so on. These blocks are called *superblocks* and form a *superchain*, a compressed blockchain.

b: SUCCINT BLOCKCHAIN

Mina [80] goes a step further and introduces the notion of a succinct blockchain, where the traditional blockchain structure is replaced by an easily verifiable cryptographic proof. Instead of verifying the entire blockchain from the beginning, participants verify the network and transactions with recursive zk-SNARKs. When a new block is created, the proof of computation not only validates the new block, but also the proof of an existing, previous block. This new state is then used as the previous state when another block is created. This keeps the size of the network constant (about 20 kB) and is more accessible to weak clients than a constantly growing ledger, which stores the entire history and has a higher barrier to entry. Smaller devices, such as smartphones, can perform a full verification of the entire network history in only about 200 ms.

c: SHARD COMMITTEE SELECTION

Another technique that changes the blockchain base layer is sharding. Sharding in blockchain means that the network is divided into several parts, called shards, with only a subset of participants responsible for each shard at any given time [143]. The shards operate in parallel and are able to communicate with each other, resulting in higher transaction throughput. Each shard usually has a group of validators, called a committee, that is responsible for consensus and verification of transactions on the same shard. The selection of the committee could be based on a block hash or round robin algorithm. However, in these approaches, an attacker could manipulate or predict the selection and potentially corrupt a particular shard. Therefore, it is important to have a fair, unbiased, and publicly verifiable random method for selecting a committee for each shard. Verifiable random functions (VRF) [59], verifiable secret sharing (VSS) [54], and publicly verifiable secret sharing (PVSS) [55] have been applied to this problem. VRF allows a participant to generate verifiable randomness, while VSS and PVSS allow a group of participants to generate collective randomness. In VSS, each participant can verify the validity of their own share, while in PVSS, everyone can verify that each participant received the correct share through public proofs without direct interaction between participants. Previous works, such as Elastico [144], used the least significant bits of the PoW hash as a source of randomness to determine the committees, which allowed adversaries to influence the selection process in the next round. A more secure approach is taken by other sharded blockchain protocols such as OmniLedger [96] and RapidChain [92], which generate randomness for committee selection using VRF and VSS, respectively.

d: MASTERNODE QUORUMS

Masternodes are incentivised full nodes with a required minimum level of performance (e.g., proof-of-service [23]) that can provide advanced functionality and participate in governance in addition to the regular tasks of a full node. Masternodes have the additional requirement of holding a collateral, which makes them sybil-resistant, i.e., a malicious user cannot spin a number of nodes to gain an advantage. Typical additional tasks include validating blocks proposed by miners, protecting against 51% miner attacks, enabling instantly re-spendable transactions, and enhancing transaction privacy and fungibility for cryptocurrencies [23].

In order to provide some of these services, masternodes must collectively reach an agreement that requires the exchange of a large number of messages between them. With a large number of masternodes, this affects the scalability of the network. The solution proposed and used by Dash [23] is to use subsets of masternodes called long-living masternode quorums (LLMQ) [145]. The main task of an LLMQ is threshold signing of consensus-related messages. Multiple LLMQs of different sizes are formed as needed using a DKG protocol and can be active simultaneously for long periods of time, e.g., hundreds of blocks, allowing load balancing between these quorums. A node can be active in multiple quorums simultaneously. LLMQs are selected in a random deterministic manner from all currently active masternodes and the result of this selection, i.e., the quorum participants, is mined on-chain before the LLMQ is actually used. The LLMQs are statistically representative of the entire network. For example, to protect against 51% mining attacks and chain reorganizations, a mechanism called Chainlock [146] is used where masternodes in LLMQ perform a verifiable vote of the ''first seen'' block to checkpoint it. If a threshold of masternodes participating in LLMQ (e.g., 80%) saw the block as first, then statistically about the same threshold of nodes across the whole network should also have seen the node as first and the block can be checkpointed. Similarly, LLMQs enable instantly re-spendable transactions through locking transactions by an LLMQ until miners include them in a block. The miner must include the locked transaction in the next block or the block will not be accepted by the masternodes. All nodes participating in an LLMQ threshold sign the messages using the BLS signature scheme. The result of the signing session can then be aggregated in a single BLS threshold signature, regardless of quorum size, and propagated network-wide. By distributing a load across multiple

LLMQs and aggregating signatures, LLMQs provide a robust solution for scaling on-chain.

2) OFF-CHAIN

Instead of dealing with scalability directly on-chain, another approach is to move parts of the system off-chain and update the data on the chain at a later time, either with partial, compressed, or representative data. A number of off-chain scaling solutions have been proposed for Ethereum and other blockchains, such as state channels [147], [148], side chains [149]–[152], and rollups [153], [154].

a: STATE CHANNELS

State channels allow participants to transact directly outside the blockchain [147]. First, participants open the state channel via multisignature, locking the part of a blockchain state. Participants can then transact with each other. Once this is done, they close the state channel and update the blockchain with the new state, unlocking the state on the blockchain. The disadvantage of this approach is that opening and closing the state channel requires the participants to be fully reachable.

b: SIDECHAINS

A sidechain is another blockchain which is linked to a main blockchain via a two-way peg, providing a way to transfer assets between the two [149]. A two-way peg is a mechanism that allows assets to be locked on one chain which can then be used on the other chain and vice versa [150]. Sidechains can have their own consensus protocols, accounting models, or privacy guarantees that may differ from those of the main blockchain. There is usually a trusted or trustless intermediary, that orchestrates the communication between the main and sidechain. Zendoo [81] and zkRelay [82] leverage zk-SNARKs to generate succinct proofs of the state of the sidechain, allowing efficient verification in constant time on the mainchain. This is an improvement over previous approaches such as BTC Relay [76] and NIPoPoWs [151], which have linear and polylogarithmic costs for the number of blocks, respectively. Since sidechains are separate blockchains, they can lack security and be expensive to maintain.

c: ROLLUPS

Rollups, which allow transactions to be executed off-chain and transaction data to be published later on the chain, are a particularly interesting solution. Since only the Merkle state root and the compressed transactions are stored on-chain, transaction throughput is greatly increased and network fees are significantly reduced. The largest space savings is achieved by using the BLS aggregate signature [20] for multiple transactions. There are two approaches to rollups, optimistic and zero-knowledge. As the names imply, optimistic rollups are based on optimistic assumptions, i.e., transactions are assumed to be valid by default and rely on fraud proofs, while zero-knowledge rollups use ZKPs as validity proofs [153]. In optimistic rollups, the entity responsible for submitting a batch of transactions is required to put down a deposit so incentives are aligned that it does not act fraudulently [154], [155], while zero-knowledge rollups require a deposit to ensure that the relayer eventually updates the state [156]. For optimistic rollups, after a new state root and compressed transactions are sent on-chain, there is a challenge period, usually one week, in which a challenger can submit a proof of fraud. If the proof of fraud shows that the batch was not constructed correctly, the entity that sent the incorrect rollup loses its deposit, the challenger is rewarded, and the on-chain state is reversed. For zero-knowledge rollups there is no challenge period, as a valid zk-SNARK or zk-STARK asserts the correctness and existence of transactions and proves the valid transition from the old to the new state. This leads to much faster finality and even less data stored on-chain. Optimistic rollups are currently better suited for general purpose computation, as NIZKPs can be complex and expensive to construct, generate, or verify. However, as the technology behind NIZKPs improves, zero-knowledge rollups are expected to overcome these issues [153]. There are numerous projects such as Optimism [78], Loopring [83], and others [79], [84], [157] that implement these two approaches for Ethereum.

E. PRIVACY

This section discusses more generally how privacy is achieved for on-chain transactions, general computation, and in smart contracts.

1) TRANSACTIONS

Public blockchains are referred to as pseudo-anonymous. This is because while the identities behind the network participants are not known, the transaction data associated with those identities is public [49]. Once the identity on the blockchain is linked to an individual, there is a risk that someone can trace their entire transaction history.

One of the solutions applied to this problem was zero-knowledge proofs. Zcash [89], [90] was one of the first to use the concept of *shielded transactions* based on zk-SNARKs. This concept allows senders and recipients of transactions to prove that encrypted transactions are valid. The sender, recipient, and transaction data cannot be inferred from the transaction itself, providing confidentiality and strong privacy guarantees.

To address the downsides of zk-SNARKs, such as trusted setup and size, other projects have begun to look at other cryptographic protocols. Projects based on the Mimblewimble [158], [159] protocol, such as Grin [160] and Beam [161], provide not only confidential transactions but also better scalability. Mimblewimble uses Pedersen commitments [57] to prove that transaction inputs and outputs are valid, so $\sum \mathrm{Tx}_{\mathrm{In}} - \sum \mathrm{Tx}_{\mathrm{Out}} = 0$, and Bulletproofs [43] as range proofs to prove that committed values are nonnegative and lie in the range $\in [0, 2^{64}]$. Mimblewimble also does not require all transactions to be stored permanently, resulting in a significant reduction in blockchain size. Monero [91], which used ring signatures [162]–[164], also implemented Bulletproofs, resulting in a drastic reduction in transaction size, faster verification times, and lower fees [165]. Encrypted transactions allow parties to take advantage of public blockchains while protecting their privacy.

2) COMPUTATION

Entities that cannot afford to run their own infrastructure usually delegate their computations to cloud services. However, adding a trusted party to manage their data poses a privacy risk. Computation networks combined with blockchain using a secure MPC can overcome this problem. Not only can a single participant keep their data private, but all participants can collectively compute something with their individual data.

ARPA [95], a privacy-preserving computation network, uses MPC in addition to information-theoretic Message Authentication Codes (MACs). In ARPA, the correctness of computations can be verified with MACs at constant time complexity, while privacy is preserved through a secret sharing protocol. This enables private smart contracts and protection of shared data being worked on.

Enigma [6] proposed a decentralized computation platform with guaranteed privacy based on MPC using blockchain for identities, access control and storage for verifiable trails. However, the originally proposed MPC protocol was impractical due to the computational overhead, so Enigma initially only supported TEEs for their *secret contracts* and deferred the implementation of MPC to future releases. For providing their computational resources, nodes in Engima are compensated with fees paid by those who submit tasks.

3) SMART CONTRACTS

Since smart contracts in most blockchains are stored on a public blockchain, their code, state, and all associated transactions can be viewed by anyone. This presents a hurdle for companies that want to move to blockchain but do not want their assets to be public.

The Hawk framework [87], which uses zk-SNARKs, was one of the first to offer a solution to this problem. Hawk splits the contract into a public contract and a private contract. The public contract is executed on the blockchain, while the private contract is executed off-chain by managers. The drawback of Hask is that the confidentiality of smart contracts depends on a manager being trusted, and each circuit requires a trusted setup for new SNARK. These issues are addressed by zkHawk [94], which replaces SNARKs and the manager with an MPC protocol. Zether [88] proposes the use of Bulletproofs for confidential transactions compatible with existing smart contact platforms. Ekiden [99] and ShadowEth [100] also achieve private smart contracts by using trusted hardware such as Intel SGX, but this imposes additional trust requirements on TEE manufacturers. Arbitrum [79] also allows smart contracts to scale by using off-chain computation, but with privacy guarantees. Arbitrum allows someone to create an Arbitrum Virtual Machine, a special smart contract, and select managers to be responsible for its execution. Arbitrum, with its weaker trust assumptions, guarantees that a single honest manager can ensure that the computations were done correctly. With Arbitrum, the smart contract code does not need to be public, the computation cost is constant, and only the cryptographic hash of the state is stored on-chain.

IV. CONCLUSION

Verifiable computing techniques are used to provide core functions in blockchain projects. They aim to reduce energy requirements while maintaining or increasing security and improving network efficiency, scalability, and privacy. Due to its recent and growing popularity, blockchain technology has been a driving factor for research in this area.

In this paper, we have provided an overview of verifiable computing techniques currently used in popular blockchain projects. We have divided the verifiable computing applications in blockchain into main application areas, i.e., multiparty approval for secure key management, sybil-resistance and consensus, smart contracts and oracles, scalability, and privacy. We have described how and for what purpose verifiable computing is used in large blockchain projects. As far as we are aware, this is the first overview of its kind that aims to map specific verifiable computing techniques to blockchain applications while providing example projects.

Future research should further develop existing verifiable computing methods and propose new ones that are less computationally intensive and enable a broader range of applications. In addition, further research is needed to extensively test the security of existing practical implementations of verifiable computing methods. Part of our current research is devoted to reviewing available state-of-the-art implementations of various verifiable computing techniques, highlighting their unique properties. In an effort to find a sustainable replacement for Proof-of-Work, we are also exploring a solution that uses verifiable computing to verify work in a Proof-of-Useful-Work consensus mechanism without relying on a trusted execution environment.

In summary, verifiable computing applications are being actively explored and improved in a rapidly evolving blockchain technology context. A move towards a sustainable model where one party computes and all others verify efficiently would not only be a technological advance, but also an environmental one.

REFERENCES

[1] X. Yu, Z. Yan, and A. V. Vasilakos, "A survey of verifiable computation," *Mobile Netw. Appl.*, vol. 22, no. 3, pp. 438–453, Jun. 2017. [Online]. Available: http://link.springer.com/10.1007/s11036-017-0872-3

[2] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Proc. 17th Annu. ACM Symp. Theory Comput. (STOC)*, Providence, RI, USA, 1985, pp. 291–304. [Online]. Available: http://portal.acm.org/citation.cfm?doid=22145.22178

[3] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again," in *Proc. 3rd Innov. Theor. Comput. Sci. Conf. (ITCS)*, Cambridge, MA, USA, 2012, pp. 326–349. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2090236.2090263

[4] T. Rabin, ''Verifiable secret sharing and multiparty protocols with honest majority,'' in *Proc. 21st Annu. ACM Symp. Comput. (STOC)*, New York, NY, USA, 1989, pp. 73–85.

[5] M. Sabt, M. Achemlal, and A. Bouabdallah, ''Trusted execution environment: What it is, and what it is not,'' in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Helsinki, Finland, Aug. 2015, pp. 57–64. [Online]. Available: http://ieeexplore.ieee.org/document/7345265/

[6] G. Zyskind, O. Nathan, and A. Pentland, ''Enigma: Decentralized computation platform with guaranteed privacy,'' Jun. 2015, *arXiv:1506.03471*.

[7] *Folding@home*. Accessed: Jul. 2021. [Online]. Available: https://foldingathome.org/

[8] M. A. Sahi, H. Abbas, K. Saleem, X. Yang, A. Derhab, M. A. Orgun, W. Iqbal, I. Rashid, and A. Yaseen, ''Privacy preservation in e-healthcare environments: State of the art and future directions,'' *IEEE Access*, vol. 6, pp. 464–478, 2018.

[9] Y. Qu, S. R. Pokhrel, S. Garg, L. Gao, and Y. Xiang, ''A blockchained federated learning framework for cognitive computing in industry 4.0 networks,'' *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2964–2973, Apr. 2021.

[10] S. Nakamoto. (2009). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Jul. 2021. [Online]. Available: https://metzdowd.com

[11] M. Jakobsson and A. Juels, ''Proofs of work and bread pudding protocols(extended abstract),'' in *Secure Information Networks*, B. Preneel, Ed. Boston, MA, USA: Springer, 1999, pp. 258–272. [Online]. Available: http://link.springer.com/10.1007/978-0-387-35568-9_18

[12] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton, NJ, USA: Princeton Univ. Press, Jul. 2016.

[13] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. (Feb. 2021). *Proofs of Useful Work*. [Online]. Available: https://eprint.iacr.org/2017/203

[14] S. King and S. Nadal. (2012). *PPCoin: Peer-to-Peer Crypto-Currency With Proof-of-Stake*. Accessed: Jul. 2021. [Online]. Available: https://people.cs.georgetown.edu/~clay/classes/fall2017/835/papers/peercoin-paper.pdf

[15] D. Boneh and V. Shoup, ''A graduate course in applied cryptography,'' Stanford, CA, USA, Tech. Rep., Jan. 2020, p. 818.

[16] W. Fang, W. Chen, W. Zhang, J. Pei, W. Gao, and G. Wang, ''Digital signature scheme for information non-repudiation in blockchain: A state of the art review,'' *EURASIP J. Wireless Commun. Netw.*, vol. 2020, no. 1, p. 56, Mar. 2020, doi: 10.1186/s13638-020-01665-w.

[17] *National Institute of Standards and Technology*. Accessed: Jul. 2021. [Online]. Available: https://www.nist.gov/

[18] C. P. Schnorr, ''Efficient signature generation by smart cards,'' *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, Jan. 1991. [Online]. Available: http://link.springer.com/10.1007/BF00196725

[19] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, ''Simple Schnorr multi-signatures with applications to bitcoin,'' *Des., Codes Cryptogr.*, vol. 87, no. 9, pp. 2139–2164, Sep. 2019. [Online]. Available: http://link.springer.com/10.1007/s10623-019-00608-x

[20] D. Boneh, B. Lynn, and H. Shacham, *Short Signatures From the Weil Pairing*. Berlin, Germany: Springer, 2001, p. 19.

[21] M. Bellare and P. Rogaway, ''Introduction to modern cryptography,'' Univ. California at Davis, Tech. Rep., 2005, p. 283.

[22] (Jul. 2021). *Chia-Network/Chia-Blockchain*. [Online]. Available: https://github.com/Chia-Network/chia-blockchain

[23] *What is Dash?* Accessed: Jul. 2021. [Online]. Available: https://dashplatform.readme.io/docs

[24] J. Teutsch and C. Reitwießner, ''A scalable verification solution for blockchains,'' Aug. 2019, *arXiv:1908.04756*.

[25] *Rosetta@home*. Accessed: Jul. 2021. [Online]. Available: https://boinc.bakerlab.org/rosetta/

[26] *Einstein@Home*. Accessed: Jul. 2021. [Online]. Available: https://einsteinathome.org/hr/home

[27] *BOINC*. Accessed: Jul. 2021. [Online]. Available: https://boinc.berkeley.edu/

[28] S. P. Muralidharan and V. V. Kumar, ''A novel reputation management system for volunteer clouds,'' in *Proc. Int. Conf. Comput. Commun. Informat.*, Coimbatore, India, Jan. 2012, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/6158811/

[29] M. Furer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, ''On completeness and soundness in interactive proof systems,'' in *Proc. Adv. Comput. Res. A, Res. Annu.*, in Randomness and Computation, vol. 5, S. Micali, Ed., 1989, pp. 429–442.

[30] M. Bellare and O. Goldreich, ''On defining proofs of knowledge,'' in *Advances in Cryptology—CRYPTO' 92* (Lecture Notes in Computer Science), E. F. Brickell, Ed. Berlin, Germany: Springer, 1993, pp. 390–420.

[31] I. Damgård. (2010). *On-Protocols*. [Online]. Available: https://www.cs.au.dk/~ivan/Sigma.pdf

[32] J. Camenisch and M. Stadler, ''Efficient group signature schemes for large groups,'' in *Advances in Cryptology—CRYPTO'97* (Lecture Notes in Computer Science), vol. 1294, G. Goos, J. Hartmanis, J. van Leeuwen, and B. S. Kaliski, Eds. Berlin, Germany: Springer, 1997, pp. 410–424. [Online]. Available: http://link.springer.com/10.1007/BFb0052252

[33] M. Blum, P. Feldman, and S. Micali, ''Non-interactive zero-knowledge and its applications (extended abstract),'' in *Proc. 20th Annu. ACM Symp. Theory Comput. (STOC)*, Jan. 1988, pp. 103–112.

[34] A. Fiat and A. Shamir, ''How to prove yourself: Practical solutions to identification and signature problems,'' in *Advances in Cryptology—CRYPTO'86* (Lecture Notes in Computer Science), vol. 263, A. M. Odlyzko, Ed. Berlin, Germany: Springer, 2006, pp. 186–194. [Online]. Available: http://link.springer.com/10.1007/3-540-47721-7_12

[35] D. Bernhard, O. Pereira, and B. Warinschi, ''How not to prove yourself: Pitfalls of the Fiat–Shamir heuristic and applications to Helios,'' in *Advances in Cryptology—ASIACRYPT 2012* (Lecture Notes in Computer Science), vol. 7658, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, X. Wang, and K. Sako, Eds. Berlin, Germany: Springer, 2012, pp. 626–643. [Online]. Available: http://link.springer.com/10.1007/978-3-642-34961-4_38

[36] B. Parno, J. Howell, C. Gentry, and M. Raykova, ''Pinocchio: Nearly practical verifiable computation,'' *Commun. ACM*, vol. 59, no. 2, pp. 103–112, Feb. 2016.

[37] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, ''Geppetto: Versatile verifiable computation,'' in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2015, pp. 253–270. [Online]. Available: https://ieeexplore.ieee.org/document/7163030/

[38] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, ''Quadratic span programs and succinct NIZKs without PCPs,'' in *Advances in Cryptology—EUROCRYPT 2013* (Lecture Notes in Computer Science), vol. 7881, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. P. Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, T. Johansson, and P. Q. Nguyen, Eds. Berlin, Germany: Springer, 2013, pp. 626–645. [Online]. Available: http://link.springer.com/10.1007/978-3-642-38348-9_37

[39] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky, ''Succinct non-interactive arguments via linear interactive proofs,'' in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 7785, A. Sahai, Ed. Berlin, Germany: Springer, 2013, pp. 315–333. [Online]. Available: http://link.springer.com/10.1007/978-3-642-36594-2_18

[40] S. Bowe, A. Gabizon, and M. D. Green, ''A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK,'' in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 10958, A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, Eds. Berlin, Germany: Springer, 2019, pp. 64–77. [Online]. Available: http://link.springer.com/10.1007/978-3-662-58820-8_5

[41] S. Bowe and A. Gabizon. (May 2019). *Scalable Multi-Party Computation for zk-SNARK Parameters in the Random Beacon Model*. [Online]. Available: https://eprint.iacr.org/2017/1050

[42] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, ''Scalable, transparent, and post-quantum secure computational integrity,'' Cryptol. ePrint Arch., Tech. Rep. 2018/046, 2018. Accessed: Jul. 2021. [Online]. Available: https://eprint.iacr.org/2018/046

[43] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, ''Bulletproofs: Short proofs for confidential transactions and more,'' in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA: May 2018, pp. 315–334. [Online]. Available: https://ieeexplore.ieee.org/document/8418611/

[44] E. Nadilinski. *Demystifying Zero Knowledge Proofs*. Accessed: Jul. 2021. [Online]. Available: https://bit.ly/3BgPkUs

[45] A. Poelstra. *Bulletproofs*. Accessed: Jul. 2021. [Online]. Available: https://diyhpl.us/wiki/transcripts/2018-02-02-andrew-poelstra-bulletproofs/

[46] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. *Bulletproofs, Short Proofs for Confidential Transactions and More*. Accessed: Jul. 2021. [Online]. Available: https://crypto.stanford.edu/~buenz/presentations/bpase18.pptx

[47] *Bulletproofs and Mimblewimble—Tari Labs University*. Accessed: Jul. 2021. [Online]. Available: https://tlu.tarilabs.com/cryptography/bulletproofs-and-mimblewimble/MainReport.html

[48] A. C. Yao, "Protocols for secure computations," in *Proc. FOCS*, 1982, p. 5.

[49] R. Zhang, R. Xue, and L. Liu, "Security and privacy on blockchain," Aug. 2019, *arXiv:1903.07602*.

[50] A. Beimel, "Secret-sharing schemes: A survey," in *Coding and Cryptology* (Lecture Notes in Computer Science), Y. M. Chee, Z. Guo, S. Ling, F. Shao, Y. Tang, H. Wang, and C. Xing, Eds. Berlin, Germany: Springer, 2011, pp. 11–46.

[51] R. Cramer and I. Damg, *Multiparty Computation From Threshold Homomorphic Encryption* (Lecture Notes in Computer Science), vol. 7. 2000, p. 20, doi: 10.7146/brics.v7i14.20141.

[52] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979. [Online]. Available: https://dl.acm.org/doi/10.1145/359168.359176

[53] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, "Verifiable secret sharing and achieving simultaneity in the presence of faults," in *Proc. 26th Annu. Symp. Found. Comput. Sci. (SFCS)*, Portland, OR, USA, 1985, pp. 383–395. [Online]. Available: http://ieeexplore.ieee.org/document/4568164/

[54] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Proc. 28th Annu. Symp. Found. Comput. Sci. (SFCS)*, Los Angeles, CA, USA, Oct. 1987, pp. 427–438. [Online]. Available: http://ieeexplore.ieee.org/document/4568297/

[55] M. Stadler, "Publicly verifiable secret sharing," in *Advances in Cryptology—EUROCRYPT'96* (Lecture Notes in Computer Science), vol. 1070, G. Goos, J. Hartmanis, J. van Leeuwen, and U. Maurer, Eds. Berlin, Germany: Springer, 1996, pp. 190–199. [Online]. Available: http://link.springer.com/10.1007/3-540-68339-9_17

[56] G. R. Blakley, "Safeguarding cryptographic keys," in *Proc. Int. Workshop Manag. Requirements Knowl. (MARK)*. New York, NY, USA: IEEE, Jun. 1979, pp. 313–318. [Online]. Available: https://ieeexplore.ieee.org/document/8817296/

[57] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Advances in Cryptology—CRYPTO'91* (Lecture Notes in Computer Science), vol. 576, J. Feigenbaum, Ed. Berlin, Germany: Springer, 1992, pp. 129–140. [Online]. Available: http://link.springer.com/10.1007/3-540-46766-1_9

[58] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, Jan. 2007. [Online]. Available: http://link.springer.com/10.1007/s00145-006-0347-3

[59] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.* New York, NY, USA: IEEE Computer Society, Oct. 1999, pp. 120–130. [Online]. Available: http://ieeexplore.ieee.org/document/814584/

[60] R. Goyal, S. Hohenberger, V. Koppula, and B. Waters, "A generic approach to constructing and proving verifiable random functions," Cryptol. ePrint Arch., Tech. Rep. 021, 2017. [Online]. Available: https://eprint.iacr.org/2017/021

[61] D. Papadopoulos, D. Wessels, S. Huque, M. Naor, J. Vcelák, L. Rezyin, and S. Goldberg, "Making NSEC5 practical for DNSSEC," Cryptol. ePrint Arch., Tech. Rep. 2017/099, 2017, p. 19. Accessed: Jul. 2021. [Online]. Available: https://eprint.iacr.org/2017/099

[62] S. Micali and L. Reyzin, "Soundness in the public-key model," in *Advances in Cryptology—CRYPTO 2001* (Lecture Notes in Computer Science), vol. 2139, G. Goos, J. Hartmanis, J. van Leeuwen, and J. Kilian, Eds. Berlin, Germany: Springer, 2001, pp. 542–565. [Online]. Available: http://link.springer.com/10.1007/3-540-44647-8_32

[63] S. Micali and R. L. Rivest, "Micropayments revisited," in *Topics in Cryptology—CT-RSA 2002* (Lecture Notes in Computer Science), vol. 2271, G. Goos, J. Hartmanis, J. van Leeuwen, and B. Preneel, Eds. Berlin, Germany: Springer, 2002, pp. 149–163. [Online]. Available: http://link.springer.com/10.1007/3-540-45760-7_11

[64] S. Jarecki and V. Shmatikov, "Handcuffing big brother: An abuse-resilient transaction escrow scheme," in *Advances in Cryptology—EUROCRYPT 2004* (Lecture Notes in Computer Science), vol. 3027, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, O. Nierstrasz, C. P. Rangan, B. Steffen, D. Terzopoulos, D. Tygar, M. Y. Vardi, C. Cachin, and J. L. Camenisch, Eds. Berlin, Germany: Springer, 2004, pp. 590–608. [Online]. Available: http://link.springer.com/10.1007/978-3-540-24676-3_35

[65] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Advances in Cryptology—CRYPTO 2018* (Lecture Notes in Computer Science), vol. 10991, H. Shacham and A. Boldyreva, Eds. Cham, Switzerland: Springer, 2018, pp. 757–788. [Online]. Available: http://link.springer.com/10.1007/978-3-319-96884-1_25

[66] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions," Tech. Rep. 712, 2018. [Online]. Available: https://eprint.iacr.org/2018/712

[67] Arm Limited. *TrustZone*. Accessed: Jul. 2021. [Online]. Available: https://developer.arm.com/ip-products/security-ip/trustzone

[68] *Intel SGX for Dummies (Intel SGX Design Objectives)*. Accessed: Jul. 2021. [Online]. Available: https://www.intel.com/content/www/us/en/develop/blogs/protecting-application-secrets-with-intel-sgx.html

[69] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on Intel SGX," Jun. 2020, *arXiv:2006.13598*.

[70] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the prevailing security vulnerabilities in trustzone-assisted TEE systems," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2020, pp. 1416–1432. [Online]. Available: https://ieeexplore.ieee.org/document/9152801/

[71] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," Aug. 2017, *arXiv:1703.06986*.

[72] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using SGX to conceal cache attacks," May 2017, *arXiv:1702.08719*.

[73] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "FORESHADOW: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," USENIX Assoc., Baltimore, MD, USA, Tech. Rep., 2018, pp. 991–1008.

[74] *Random Number Generation*. Accessed: Jul. 2021. [Online]. Available: https://www.intel.com/content/www/us/en/develop/documentation/sgx-developer-guide/top/processor-features/random-number-generation.html

[75] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 12059, J. Bonneau and N. Heninger, Eds. Cham, Switzerland: Springer, 2020, pp. 505–522. [Online]. Available: http://link.springer.com/10.1007/978-3-030-51280-4_27

[76] (Jun. 2021). *BTC Relay*. [Online]. Available: https://github.com/ethereum/btcrelay

[77] S. Ellis, A. Juels, and S. Nazarov. (Sep. 2017). *ChainLink: A Decentralized Oracle Network*. [Online]. Available: https://link.smartcontract.com/whitepaper

[78] *Optimism*. [Online]. Available: https://optimism.io/

[79] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten, "Arbitrum: Scalable, private smart contracts," in *Proc. 27th USENIX Secur. Symp. (USENIXSecurity)*. Baltimore, MD, USA: USENIX Association, 2018, pp. 1353–1370.

[80] J. Bonneau, I. Meckler, and V. Rao, "Coda: Decentralized cryptocurrency at scale," Cryptol. ePrint Arch., Tech. Rep. 2020/352, 2020, p. 47. Accessed: Jul. 2021. [Online]. Available: https://eprint.iacr.org/2020/352

[81] A. Garoffolo, D. Kaidalov, and R. Oliynykov, "Zendoo: A zk-SNARK verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains," Feb. 2020, *arXiv:2002.01847*.

[82] M. Westerkamp and J. Eberhardt, "zkRelay: Facilitating sidechains using zkSNARK-based chain-relays," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, Genoa, Italy, Sep. 2020, pp. 378–386. [Online]. Available: https://ieeexplore.ieee.org/document/9229702/

[83] *Loopring—zkRollup Exchange and Payment Protocol*. [Online]. Available: https://loopring.io//

[84] *zkSync—Rely on Math, Not Validators*. Accessed: Jul. 2021. [Online]. Available: https://matter labs.io and https://zksync.io

[85] *ZoKrates*. Accessed: Jul. 2021. [Online]. Available: https://zokrates.github.io/

[86] *Cairo*. Accessed: Jul. 2021. [Online]. Available: https://www.cairo-lang.org/

[87] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA: May 2016, pp. 839–858. [Online]. Available: http://ieeexplore.ieee.org/document/7546538/

[88] B. Bünz, S. Agrawal, M. Zamani, and D. Boneh, "Zether: Towards privacy in a smart contract world," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 12059, J. Bonneau and N. Heninger, Eds. Cham, Switzerland: Springer, 2020, pp. 423–443. [Online]. Available: http://link.springer.com/10.1007/978-3-030-51280-4_23

[89] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2014, pp. 459–474. [Online]. Available: https://ieeexplore.ieee.org/document/6956581/
[90] *What are zk-SNARKs? | Zcash*. Accessed: Jul. 2021. [Online]. Available: https://z.cash/technology/zksnarks/
[91] *Moneropedia: Bulletproofs*. Accessed: Jul. 2021. [Online]. Available: https://www.getmonero.org//resources/moneropedia/bulletproofs.html
[92] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling blockchain via full sharding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Toronto, ON, Canada, Oct. 2018, pp. 931–948. [Online]. Available: https://dl.acm.org/doi/10.1145/3243734.3243853
[93] F. Zhang, D. Maram, H. Malvai, S. Goldfeder, and A. Juels, "DECO: Liberating web data using decentralized oracles for TLS," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1919–1938.
[94] A. Banerjee, M. Clear, and H. Tewari, "zkHawk: Practical private smart contracts from MPC-based hawk," May 2021, *arXiv:2104.09180*.
[95] D. Zhang, A. Su, F. Xu, and J. Chen, "ARPA whitepaper," Dec. 2018, *arXiv:1812.05820*.
[96] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2018, pp. 583–598. [Online]. Available: https://ieeexplore.ieee.org/document/8418625/
[97] F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. van Renesse, "REM: Resource-efficient mining for blockchains," in *Proc. 26th USENIX Conf. Secur. Symp. (SEC)*, 2017, pp. 1427–1444.
[98] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 270–282. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978326
[99] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 185–200.
[100] R. Yuan, Y.-B. Xia, H.-B. Chen, B.-Y. Zang, and J. Xie, "ShadowEth: Private smart contract on public blockchain," *J. Comput. Sci. Technol.*, vol. 33, no. 3, pp. 542–556, May 2018. [Online]. Available: http://link.springer.com/10.1007/s11390-018-1839-y
[101] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie–Hellman-group signature scheme," in *Public Key Cryptography—PKC 2003* (Lecture Notes in Computer Science), vol. 2567, G. Goos, J. Hartmanis, J. van Leeuwen, and Y. G. Desmedt, Eds. Berlin, Germany: Springer, 2003, pp. 31–46. [Online]. Available: http://link.springer.com/10.1007/3-540-36288-6_3
[102] Y. Desmedt, "Society and group oriented cryptography: A new concept," in *Advances in Cryptology—CRYPTO'87* (Lecture Notes in Computer Science), vol. 293, G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. B. Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and C. Pomerance, Eds. Berlin, Germany: Springer, 1988, pp. 120–127. [Online]. Available: http://link.springer.com/10.1007/3-540-48184-2_8
[103] J.-P. Aumasson, A. Hamelink, and O. Shlomovits, "A survey of ECDSA threshold signing," Cryptol. ePrint Arch., Tech. Rep. 2020/1390, 2020, p. 14. Accessed: Jul. 2021. [Online]. Available: https://eprint.iacr.org/2020/1390
[104] M.-S. Hwang and T.-Y. Chang, "Threshold signatures: Current status and key issues," *Int. J. Netw. Secur.*, vol. 1, pp. 123–137, 2005.
[105] D. W. Archer, D. Bogdanov, Y. Lindell, L. Kamm, K. Nielsen, J. I. Pagter, N. P. Smart, and R. N. Wright, "From keys to databases—Real-world applications of secure multi-party computation," *Comput. J.*, vol. 61, no. 12, pp. 1749–1771, Dec. 2018. [Online]. Available: https://academic.oup.com/comjnl/advance-article/doi/10.1093/comjnl/bxy090/5095655
[106] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "SoK: Consensus in the age of blockchains," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, Zürich, Switzerland, Oct. 2019, pp. 183–198. [Online]. Available: https://dl.acm.org/doi/10.1145/3318041.3355458
[107] j. R. Douceur, "The Sybil attack," in *Peer-to-Peer Systems* (Lecture Notes in Computer Science), vol. 2429, G. Goos, J. Hartmanis, J. van Leeuwen, P. Druschel, F. Kaashoek, and A. Rowstron, Eds. Berlin, Germany: Springer, 2002, pp. 251–260. [Online]. Available: http://link.springer.com/10.1007/3-540-45748-8_24
[108] C. Cachin, R. Guerraoui, and L. Rodrigues, Eds., *Introduction to Reliable and Secure Distributed Programming*. Berlin, Germany: Springer, 2011.
[109] B. Alpern and F. B. Schneider, "Recognizing safety and liveness," *Distrib. Comput.*, vol. 2, no. 3, pp. 117–126, Sep. 1987. [Online]. Available: http://link.springer.com/10.1007/BF01782772
[110] A. Back. (Aug. 1, 2002). *Hashcash—A Denial of Service Counter-Measure*. Accessed: Jul. 2021. [Online]. Available: http://www.hashcash.org/hashcash.pdf
[111] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *Proc. 25th IET Irish Signals Syst. Conf., China–Ireland Int. Conf. Inf. Commun. Technol. (ISSC/CIICT)*, Jun. 2014, pp. 280–285.
[112] A. F. Loe and E. A. Quaglia, "Conquering generals: An NP-hard proof of useful work," in *Proc. 1st Workshop Cryptocurrencies Blockchains Distrib. Syst.*, Munich, Germany, Jun. 2018, pp. 54–59. [Online]. Available: https://dl.acm.org/doi/10.1145/3211933.3211943
[113] C. G. Oliver, A. Ricottone, and P. Philippopoulos, "Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems," Sep. 2017, *arXiv:1708.09419*.
[114] H. K. Turesson, H. Kim, M. Laskowski, and A. Roatis. (Nov. 2020). *Proof-of-Useful-Work as Dual-Purpose Mechanism for Blockchain and AI*. Accessed: Jul. 2021. [Online]. Available: https://arxiv.org/abs/1907.08744
[115] A. Baldominos and Y. Saez, "Coin.AI: A proof-of-useful-work scheme for blockchain-based distributed deep learning," *Entropy*, vol. 21, no. 8, p. 723, Jul. 2019. [Online]. Available: https://www.mdpi.com/1099-4300/21/8/723
[116] A. Lihu, J. Du, I. Barjaktarevic, P. Gerzanics, and M. Harvilla, "A proof of useful work for artificial intelligence on the blockchain," Jan. 2020, *arXiv:2001.09244*.
[117] *Proof of Elapsed Time, Sawtooth Lake 0.7 Documentation*. Accessed: Jul. 2021. [Online]. Available: https://sawtooth.hyperledger.org/docs/core/releases/0.7/introduction.html#proof-of-elapsed-time-poet
[118] L. Chen, L. Xu, N. Shah, Z. Gao, Y. Lu, and W. Shi, "On security analysis of proof-of-elapsed-time (PoET)," in *Stabilization, Safety, and Security of Distributed Systems* (Lecture Notes in Computer Science), vol. 10616, P. Spirakis and P. Tsigas, Eds. Cham, Switzerland: Springer, 2017, pp. 282–297. [Online]. Available: http://link.springer.com/10.1007/978-3-319-69084-1_19
[119] C. T. Nguyen, D. T. Hoang, D. N. Nguyen, D. Niyato, H. T. Nguyen, and E. Dutkiewicz, "Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities," *IEEE Access*, vol. 7, pp. 85727–85745, 2019.
[120] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Oper. Syst. Principles*, Shanghai, China, Oct. 2017, pp. 51–68. [Online]. Available: https://dl.acm.org/doi/10.1145/3132747.3132757
[121] *Algorand Consensus—Algorand Developer Portal*. Accessed: Jul. 2021. [Online]. Available: https://developer.algorand.org/docs/algorand_consensus/
[122] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Advances in Cryptology—EUROCRYPT 2018* (Lecture Notes in Computer Science), vol. 10821, J. B. Nielsen and V. Rijmen, Eds. Cham, Switzerland: Springer, 2018, pp. 66–98. [Online]. Available: https://link.springer.com/10.1007/978-3-319-78375-8_3
[123] D. G. Wood. (2016). *POLKADOT: Vision for a Heterogeneous Multi-Chain Framework*. Accessed: Jul. 2021. [Online]. Available: https://polkadot.network/PolkaDotPaper.pdf
[124] J. Burdges, A. Cevallos, P. Czaban, R. Habermeier, S. Hosseini, F. Lama, H. Kilinc Alper, X. Luo, F. Shirazi, A. Stewart, and G. Wood, "Overview of polkadot and its design considerations," May 2020, *arXiv:2005.13456*.
[125] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," May 2018, *arXiv:1805.04548*.
[126] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang, "An overview of smart contract: Architecture, applications, and future trends," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2018, pp. 108–113.
[127] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, "Trustworthy blockchain oracles: Review, comparison, and open research challenges," *IEEE Access*, vol. 8, pp. 85675–85685, 2020.
[128] *Provable—Blockchain Oracle Service, Enabling Data-Rich Smart Contracts*. Accessed: Jul. 2021. [Online]. Available: https://provable.xyz/
[129] *TLSnotary—A Mechanism for Independently Audited*. (Sep. 2014). [Online]. Available: https sessions
[130] *TC—Getting Started*. Accessed: Jul. 2021. [Online]. Available: https://town-crier.org/get-started/
[131] S. Woo, J. Song, and S. Park, "A distributed Oracle using Intel SGX for blockchain-based IoT applications," *Sensors*, vol. 20, no. 9, p. 2725, May 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/9/2725

[132] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, document RFC 8446, Aug. 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446
[133] E. G. Sirer. *How Not to Run a Blockchain Lottery*. Accessed: Jul. 2021. [Online]. Available: https://hackingdistributed.com/2017/12/24/how-not-to-run-a-blockchain-lottery/
[134] J. Bonneau and B. Bu, "Proofs-of-delay and randomness beacons in Ethereum," presented at the IEEE S&B Workshop, 2016, p. 11. Accessed: Jul. 2021. [Online]. Available: https://jbonneau.com/doc/BGB17-IEEESB-proof_of_delay_ethereum.pdf
[135] (Jul. 2021). *RANDAO: A DAO Working as RNG of Ethereum*. [Online]. Available: https://github.com/randao/randao
[136] *How Can I Securely Generate a Random Number in My Smart Contract?* Accessed: Jul. 2021. [Online]. Available: https://ethereum.stackexchange.com/questions/191/how-can-i-securely-generate-a-random-number-in-my-smart-contract
[137] *Introduction to Chainlink VRF | Chainlink Documentation*. Accessed: Jul. 2021. [Online]. Available: https://docs.chain.link/docs/chainlink-vrf/
[138] (Mar. 2021). *StarkWare VeeDo*. [Online]. Available: https://github.com/starkware-libs/veedo
[139] A. Kosba, C. Papamanthou, and E. Shi, "XJsnark: A framework for efficient verifiable computation," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2018, pp. 944–961. [Online]. Available: https://ieeexplore.ieee.org/document/8418647/
[140] S. Jain, P. Saxena, F. Stephan, and J. Teutsch, "How to verify computation with a rational network," Jun. 2016, *arXiv:1606.05917*.
[141] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology—ASIACRYPT 2018* (Lecture Notes in Computer Science), vol. 11273, T. Peyrin and S. Galbraith, Eds. Cham, Switzerland: Springer, 2018, pp. 435–464. [Online]. Available: http://link.springer.com/10.1007/978-3-030-03329-3_15
[142] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proc. Int. Conf. Manage. Data*, Amsterdam, The Netherlands, Jun. 2019, pp. 123–140. [Online]. Available: https://dl.acm.org/doi/10.1145/3299869.3319889
[143] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "SoK: Sharding on blockchain," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, Zürich, Switzerland, Oct. 2019, pp. 41–61. [Online]. Available: https://dl.acm.org/doi/10.1145/3318041.3355457
[144] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Vienna, Austria, Oct. 2016, pp. 17–30. [Online]. Available: https://dl.acm.org/doi/10.1145/2976749.2978389
[145] (Jun. 2021). *Long-Living Masternode Quorums*. [Online]. Available: https://github.com/dashpay/dips/blob/master/dip-0006.md
[146] (Jun. 2021). *Chainlocks*. [Online]. Available: https://github.com/dashpay/dips/blob/master/dip-0008.md
[147] S. Dziembowski, S. Faust, and K. Hostáková, "General state channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Toronto, ON, Canada, Oct. 2018, pp. 949–966. [Online]. Available: https://dl.acm.org/doi/10.1145/3243734.3243856
[148] A. Miller, I. Bentov, R. Kumaresan, C. Cordi, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," Nov. 2017, *arXiv:1702.05812*.
[149] A. Singh, K. Click, R. M. Parizi, Q. Zhang, A. Dehghantanha, and K.-K.-R. Choo, "Sidechain technologies in blockchain networks: An examination and state-of-the-art review," *J. Netw. Comput. Appl.*, vol. 149, Jan. 2020, Art. no. 102471. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1084804519303315
[150] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. (2014). *Enabling Blockchain Innovations With Pegged Sidechains*. Accessed: Jul. 2021. [Online]. Available: https://blockstream.com/sidechains.pdf
[151] A. Kiayias and D. Zindros, "Proof-of-work sidechains," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 11599, A. Bracciali, J. Clark, F. Pintore, P. B. Rønne, and M. Sala, Eds. Cham, Switzerland: Springer, 2020, pp. 21–34. [Online]. Available: http://link.springer.com/10.1007/978-3-030-43725-1_3
[152] J. Poon and V. Buterin. (2017). *Plasma: Scalable Autonomous Smart Contracts*. Accessed: Jul. 2021. [Online]. Available: https://www.plasma.io/plasma.pdf
[153] V. Buterin. *An Incomplete Guide to Rollups*. Accessed: Jul. 2021. [Online]. Available: https://vitalik.ca/general/2021/01/05/rollup.html
[154] *Layer 2 Rollups*. Accessed: Jul. 2021. [Online]. Available: https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/
[155] *Optimistic Rollups*. Accessed: Jul. 2021. [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups/
[156] *ZK-Rollups*. Accessed: Jul. 2021. [Online]. Available: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/
[157] *Polygon*. Accessed: Jul. 2021. [Online]. Available: https://polygon.technology/
[158] A. Poelstra. (Oct. 2016). *Mimblewimble*. [Online]. Available: https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf
[159] T. E. Jedusor. (Jul. 2016). *Mimblewimble*. [Online]. Available: https://docs.beam.mw/Mimblewimble.pdf
[160] *Grin*. Accessed: Jul. 2021. [Online]. Available: https://grin.mw./
[161] *BEAM Confidential Cryptocurrency and DeFi Platform*. Accessed: Jul. 2021. [Online]. Available: https://beam.mw/
[162] S. Noether and A. Mackenzie, "Ring confidential transactions," *Ledger*, vol. 1, pp. 1–18, Dec. 2016. [Online]. Available: http://ledger.pitt.edu/ojs/ledger/article/view/34
[163] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen, "RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency Monero," in *Computer Security—ESORICS 2017* (Lecture Notes in Computer Science), vol. 10493, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham, Switzerland: Springer, 2017, pp. 456–474. [Online]. Available: http://link.springer.com/10.1007/978-3-319-66399-9_25
[164] N. V. Saberhagen. (Oct. 2013). *CryptoNote Version 2.0*. [Online]. Available: https://bytecoin.org/old/whitepaper.pdf
[165] N. Alsalami and B. Zhang, "SoK: A systematic study of anonymity in cryptocurrencies," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Hangzhou, China, Nov. 2019, pp. 1–9. [Online]. Available: https://ieeexplore.ieee.org/document/8937681/

SILVIO ŠIMUNIĆ received the B.S. degree in computer science from the Faculty of Engineering, University of Rijeka, in 2018, where he is currently pursuing the M.S. degree in computer science. He is also working on blockchain projects as a member of the Laboratory for Applications of Blockchain Technology, Centre for Artificial Intelligence and Cybersecurity (AIRI). His research interests include distributed systems, the IoT, and blockchain.

DALEN BERNACA received the B.S. and M.S. degrees in computer engineering from the Faculty of Engineering, University of Rijeka, in 2012 and 2016, respectively, where he is currently pursuing the Ph.D. degree. He is a member of the Autonomous Systems and Artificial Perception Laboratory (APASLab), Faculty of Engineering, University of Rijeka, and collaborates on projects on other Croatian Universities. His research interests include blockchain, networking systems and protocols, embedded systems, mobile robotics, artificial perception, and human–computer interaction.

KRISTIJAN LENAC (Member, IEEE) received the Ph.D. degree from the University of Trieste, Italy, with a focus on mobile robotics. He is currently an Associate Professor with the Faculty of Engineering, University of Rijeka. He then worked for five years as a Project Manager and a Lead Developer at AIBSLab Engineering Company from Trieste leading several research and development projects commissioned by the industry. On his return to Croatia, he joined the Computer Engineering Department, Faculty of Engineering, University of Rijeka, as a Professor. He is a Founder and currently the Head of the Artificial Perception and Autonomous Systems Laboratory and one of the founders of Riteh Drone Team. At the Centre for Artificial Intelligence and Cybersecurity (AIRI), he is a Founder and currently the Head of the Laboratory for Applications of Blockchain Technology. His research interests include mobile robotics, satellite navigation, embedded systems, and blockchain. He is a member of the Royal Institute of Navigation.