A Novel Architecture for Enterprise Blockchains using Trusted Execution Environments

Kai Mast
Cornell University

Lequn Chen
University of Washington

Emin Gün Sirer
Cornell University

January 16, 2019

Abstract

Blockchains have emerged as a potential mechanism to enable immutable and consistent sharing of data across organizational boundaries. While much of the discussion on blockchains to date has been structured around public versus permissioned blockchains, both of these architectures have significant drawbacks. Public blockchains are energy inefficient, hard to scale and suffer from limited throughput and high latencies, while permissioned blockchains depend on specially designated nodes, potentially leak meta-information, and also suffer from scale and performance bottlenecks.

This paper introduces *autonomous blockchains*, a new blockchain architecture based on free-standing, immutable, eidetic databases that implement independent timelines, linked together through interactions. Autonomous blockchains can be realized inside trusted execution environments, to provide not only blockchain-like integrity and auditability guarantees, but also to support the storage and querying of private data. Further, multiple autonomous blockchains can be linked together through federated transactions to exchange data and order mutual operations. These transactions are amenable to audits and yield tamper-proof witnesses. The paper describes these mechanisms, and the applications they collectively enable in detail. Evaluation shows that this design can achieve high throughput while providing stronger integrity guarantees than other NoSQL stores.

1 Introduction

Many high-value applications require the reliable and immutable storage of data across multiple distrusting parties [41, 14, 45]. These applications are characterized by integrity requirements wherein each party must abide by pre-defined policies. Conventional databases cannot live up to this challenge, as they require full trust in the database application and host operating system.

Blockchains have recently emerged as a potential platform to address the needs of these applications. Public/permissionless blockchains [31, 42], based on proof-of-work or proof-of-stake consensus, maintain an immutable log of events distributed across all participants of the system. As a result, they are energy inefficient, hard to scale and suffer from limited throughput and high latencies [12]. Further, due to their open and distributed setting, they cannot be used to store private or confidential data. Private/permissioned blockchains [7, 34, 30] employ a committee consensus protocol [8, 25, 20] to maintain the log and append updates in an orderly fashion. Changes to the data are only possible after a valid quorum of the committee agrees to do so. This approach necessarily requires specially designated committee nodes, often leaks business information and meta-information, such as which clients interact with which others, at what frequency, and is limited in performance by bottlenecks in geo-replicated, large quorums, typically built on quadratic consensus protocols. Earlier work focused on accountable systems [24, 43], which ensure integrity by allowing clients to audit the log with respect to states they have observed previously. But accountability mechanisms by themselves can only enforce fork consistency [28, 15], a weaker security property than strong consistency.

This work presents *autonomous blockchains*, a novel class of data stores that provide a self-standing, permanent, tamper-proof record of all data to facilitate computations over integrity and confidentiality protected data across organizational boundaries. A system is self-standing if it does not rely on external validators or a quorum of replicated machines to ensure its correctness. The central abstraction provided by autonomous blockchains is an append-only log [10] of partially-ordered states. Past states are unchangeable, and new states are only appended to the log. Building on this foundation, autonomous blockchain instances can operate independently, as a "blockchain of one," and offer strong integrity and confidentiality guarantees.

Autonomous blockchains can be linked together through transactions spanning multiple chains. A new federated transaction primitive enables connection blockchains. Autonomous blockchains are thus able to create networks that share designated data items and invokes computations on each other. This is in stark contrast to conventional blockchains, which replicate all data across all nodes, as it minimizes inter-node communication and enables the network to scale with the number of blockchain nodes.

Autonomous blockchains support application-defined policies [40, 21] as well as introspection mechanisms to enable trust in the policies to allow distrusting parties to interact across multiple chains. Specifically, autonomous blockchains allow every object to be associated inseparably with an associated *semantic security policy*. Policies are encoded symbolically as abstract syntax trees, which enables applications to analyze the policy and establish trust in the future behavior of that object. Policy enforcement allows guaranteeing confidentiality as well as integrity. For instance, blockchains may restrict modifications to a bank account those issued by the specified owner and ones that do not result in a negative balance.

To enable privacy preserving data queries, autonomous blockchains support *protected function evaluations*, read-only transactions that compute functions over remote private data [18, 44]. The primary use of this functionality is to compute a vetted function over private data without revealing the input data to the remote party. Like with any other transaction, the holder of the data retains full control over what can be done with the data, and both parties, the invoker, and only functions can be executed on the data if the data holder agrees to do so.

We have fully implemented CreDB, a prototype autonomous blockchain, backed by secure hardware. Each CreDB node in the system runs in a *trusted execution environment (TEE)*, provided by the system's hardware. The usage of TEEs enables nodes to trust another participant's computation without trusting the administrator of that system. Because each service can run their own blockchain backed by a TEE, the throughput scales with the number of nodes in the system. Experiments show that, depending on the workload, CreDB can process up to 50k ops on a single machine, and approximately 500 tps on the TPC-C benchmark.

The rest of this paper is structured as follows. The next section provides the overarching data and computation model for autonomous blockchains (Section 2). The following two sections evaluate a full prototype of the system. The evaluation contains a qualitative part, explaining how to implement several sample applications (Section 3), and a quantitative component, describing the impact our execution environment has on the performance of the system (Section 4).

2 The Autonomous Blockchain Model

At a high level, every autonomous blockchain implements a secure database using a trusted execution environment (TEE) that clients, as well as other nodes, can connect to. Each blockchain instance has its own timeline, datastore, and set of connected nodes. An autonomous blockchain connects to other chains to create a network across which data can be shared, and functions can be invoked, securely. Clients connect through one or multiple such blockchains and do not need special hardware support. This enables porting legacy database applications to this new abstraction and provide them with stronger integrity guarantees. Nodes and clients rely on a public attestation service to ensure integrity and authenticity of database nodes. Attestation services provide a public-key infrastructure to ensure the authenticity of parties but do not gain access to private data. This architecture, as well as permissioned and permissionless architectures, are visualized in Figure 1.

Applications are written against an API that is a superset of a transactional key-value storage API. The most notable additions are *timeline inspection*, *secure semantic policies*, *witness generation*, and *protected function evaluation*, which we discuss below after we provide the basic object and event model on which the system is based.

2.1 Assumptions and Attack Model

Following previous work [3] we assume a powerful attack model: an adversary might have root access to the database server. This includes full control over the scheduler, the file system, and network communication. The attacker may tamper with the hardware, except for the CPU itself.

We further assume that clients mistrust and database operators may distrust other parties in the system. This means principals need assurance that data can be modified only by parties they specify. Further, they demand control over what information is leaked to other parties, including the database administrator.

Autonomous blockchains do not rely on a single implementation or a single source of trusted hardware. An autonomous blockchain can be vetted by auditing their source code and verifying the running binary to correspond to this source code. The federated model allows for multiple implementations of the same protocol to coexist on the same network.

However, unlike previous work, we assume that participants are economically incentivized to keep their systems up and running. We believe this is a realistic assumption most businesses already provide strong liveness guarantees for their services.

2.2 Objects and Transactions

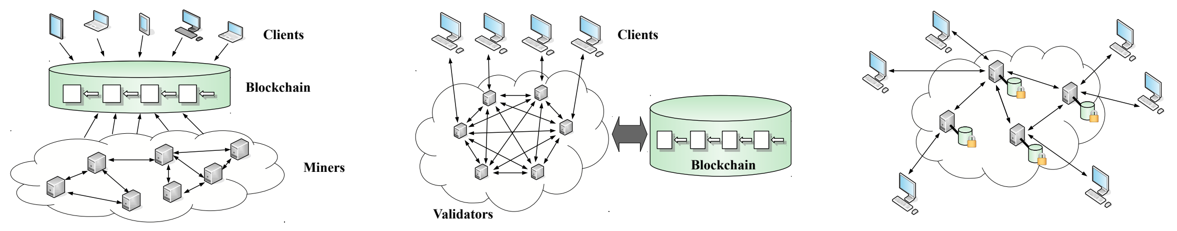
Autonomous blockchains expose a flexible object model that accommodates unstructured, as well as structured, data. Objects are collections of attribute-value pairs, where attributes can have types such as lists, dictionaries, binary data, and primitive values, which consist of integers, floating point numbers, and strings. Binary data can contain executable code representing stored procedures. Each object belongs to a *collection* (similar to tables in relational datastores).

Each blockchain maintains a partially-ordered log of *transactions*, each relating to one or more *objects*. As such, for each creation, update, or deletion of an object, the ledger holds a record of a corresponding transaction. Transactions store the new values of all updated objects. In the case of a deletion, the new value is a tombstone entry $\perp$. Transactions relating to the same object are arranged in a total order to guarantee linearizability [17]. Further, in case events are created by a transaction that spans multiple objects, an event may also capture the dependencies between versions of different objects. Transactions are further able to encode read and write sets of objects on remote timelines, which is a necessity for cross-node collaboration. Crucially, events that are unrelated are not ordered with respect to each other.

2.3 Protected Function Evaluation

Another key primitive supported by autonomous blockchains is *protected function evaluation (PFE)*. PFE enables parties to invoke a custom function on a remote node in a secure execution environment guarded by the TEE. This way, data protected by the TEE remains private to the trusted environment, and only the designated result of the function call is revealed to the caller.

Since computations on private data have the intended goal of retrieving some information extracted from that data, they need to be vetted to ensure that this leakage is permissible to all parties. Autonomous blockchains employ two mechanisms to perform this vetting. First, before execution of a function, both the calling and the executing parties must approve the function. The executing party needs to ensure that no private data is leaked and that the function execution does not take up an unreasonable amount of resources. This can be done by checking the functions hash against a whitelist or by analyzing the AST of the function. Much past work concerns itself with the analysis of function



(a) Permissionless/public Blockchains (b) Permissioned/private Blockchains (c) Autonomous Blockchains

Figure 1: The autonomous architecture compared to permissioned and permissionless blockchains. Instead of one global ledger, autonomous blockchains form a network of independent ledgers.

properties, including for information leakage [13] and information flow [26], so the mechanisms of this vetting are beyond the scope this paper. In addition, every single object retrieved during a PFE has its semantic security policy checked on every access.

After successful execution, the calling party receives a witness containing a function identifier and its result. Autonomous blockchains identify functions through the hash of their bytecode. The witness is signed by a persistent key associated with the blockchain instance of the executing party.

This design imposes minimal structure on witnesses. In particular, it deliberately leaves freshness guarantees up to applications – autonomous blockchains do not purport to provide a global clock or a total order of events. The critical observations behind this decision are threefold. First, no single notion of time can serve every application. Some applications may operate on a sub-microsecond granularity, which could entail inordinate overheads, while others keep track of events in a more coarse-grained manner. Second, even if there was a time granularity that one could pick for most applications, current technologies for providing a trusted time source into a secure execution environment provide much weaker guarantees than the TEE itself, because they rely on additional hardware outside of the CPU die [11]. Finally, it has been our experience that most applications can be implemented using simple happens-before relationships between affected objects.

2.4 Semantic Security Policies

Semantic security policies enable applications to associate application-specific constraints with an object. These policies are inseparable from the object to which they belong and inviolable even by the principal controlling the database instance. To access the database, a user must necessarily go through the blockchain's policy enforcement engine mandated by the TEE.

Thus, even an attacker who takes over the database cannot subvert the access policies associated with objects. In case of accessing a previous version of the object, that version's policy and state will be used to make an access control decision.

Each autonomous blockchain maintains a registry of identities, which can be leveraged by policies to make an access decision. Identities are tuples consisting of a human-readable name and a public key. This registry is used to prevent man-in-the-middle and impersonation attacks. We assume a public key infrastructure (PKI) that nodes can rely on when connecting to previously unknown parties.

Identities are inseparable from the associated authenticated communication channel. In particular, nodes cannot change their identity after a connection has been set up. Costly authentication and attestation have to be performed only once when setting up the channel. After successful attestation, policies can always rely on the authenticity of the referenced identities.

Policies are specified at the time of an object's creation and can be modified after the fact only if the policy permits it. And changes to a policy are stored in the object's timeline just like changes to all other fields of the object. Accesses to an object's value in the timeline leading to the evaluation of the object's policy at that point in time. To enable this, autonomous blockchains require policies to be idempotent, i.e., they may not have any side effects or reference the state of other objects.

Policies use the current state of the object, as well as information about the attempted operation, to make access control decisions. Similar to conventional stored procedures policies have access to several modules that hold information about the attempted operation and the object to be accessed. Two modules, in particular, are only available to secure semantic policies: First, op_info contains information about the operation itself, such as the kind of operation and the proposed change. Second, op_context enables to retrieve information about the issuing party, such as their identity.

Autonomous blockchains further allow associating a policy with a collection to enable richer application semantics. Such collection policies may, for example, specify who can create or modify any object in the collection. Further, they can enforce a schema on the data, by rejecting all updates that miss required fields or contain fields in an invalid format. Collection policies thus allow to break down application logic into multiple concurrent objects without sacrificing integrity.

3 Example Applications

CreDB provides a novel programming model for building high-integrity applications. This section describes multiple applications to demonstrate the practicality of this model. These applications leverage server-side program evaluation, both in the form of stored procedure and policies, to provide rich application logic on top of CreDB. Together with stored procedures, policies enable to implement state machines in the form of an object.

Stored procedures define possible state transitions, while the policies restrict when and by whom these transitions can be invoked.

3.1 Checking Credit History

Financial institutions often check the credit history of a potential new client by querying their banks. However, both the client as well as their bank might not want to reveal the credit history for both privacy and business reasons. Currently, third-party credit score agencies, required to be trusted by all parties, are used to circumvent this issue. Such an approach might not always be feasible as it needs such a common trusted party to exist and usually creates additional fees and pose other vulnerabilities.

CreDB's PFE mechanism natively accommodates such blind checks. We built a credit score checker that computes on the bank's data without revealing the data itself. This functionality is stored on the bank's blockchain instance in the form of a callable function, that can be vetted by third parties. The bank further implements a policy that ensures customer data is stored in a specified format. In particular, the policy ensures that customer data cannot be arbitrarily changed to ensure the credit checkers input is authentic.

Both parties check the trusted function for correctness. The bank ensures that it solely returns the credit score, without leaking any sensitive parts of the client's credit history. The party requesting the credit score inspects the program together with the database policies to certify no data can be omitted or changed from the credit check.

The credit checker function first queries the client's bank to retrieve all transactions corresponding to the client. After executing PFEs on the client's banks, it will check the client's overall balance. In our implementation, this checker will calculate the overall balance of the account and further look for any periods of bad liquidity. In a real-world system would use a more sophisticated mechanism in place.

3.2 Micropayments

Payment services can wire micropayments using a network of CreDB nodes. In such setup, each service maintains its own blockchain instance and connects it to nodes of banks it conducts business with. Each blockchain implements a simple double-entry booking system, which requires two collections: liabilities and assets. The liabilities collection keeps a record of all value stored on the local blockchain by clients and remote banks. assets reference remote accounts of that bank stored at other institution. An additional collection, programs, holds stored procedures that allow authorized parties to move value as well as data analysis. Policies further ensure that assets and liabilities can only be modified by authorized programs and the authorized programs themselves cannot be modified at all.

Money can then be moved between accounts on the same service and accounts on remote services, without the reliance on a third party. To achieve this, two programs are stored on each blockchain: move_locally and move_remotely. move_locally directly modifies two entries in the liabilities-collection to transfer value. The program executes in the form of a server-side transaction to ensure atomicity and isolation of account balance changes.

Moving money between accounts on two different payment services is achieved through the move_remotely-program. The program requires three arguments, the source account, the target account, and the target payment service. It then checks the source account's balance before looking up the target bank in the assets-collection. If such an entry exists it will invoke move_locally on the remote bank, moving money between the source bank's account on the remote bank and the target clients account. Like its local counterpart, the program relies on a transaction ensure isolation and atomicity.

4 Experimental Evaluation

We implemented and evaluated fully-functional prototype of an autonomous blockchain in the form of CreDB. The prototype is implemented in about 25k lines of C++ code. The main takeaway from the result in this section is that while the overheads associated with this kind of secure hardware are significant, they can be mitigated using efficient implementation and paging techniques.

The prototype uses version 2.1.2 of the Intel SGX SDK and is compiled using GNU g++7. Evaluation is done using two kinds of hardware. First, a big configuration that provides 32GB of RAM and an Intel Core i7 6700K CPU offering 8 logical cores. Second, a medium configuration providing 16GB of RAM and an Intel Xeon E5420 CPU offering 8 logical cores. Both configurations run Ubuntu 18.04 based on Linux 4.15. For all experiments, except 4.2, a single server is hosted on the big configuration while clients execute across multiple medium configurations.

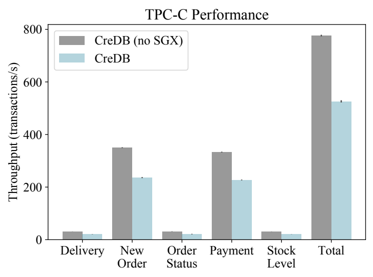
4.1 Transactional Performance

We expose CreDB to a TPC-C workload and compare it to a version of CreDB that doesn't run in SGX. The experimental setup contains four warehouses and a dataset of about one gigabyte. Intel will most likely provide hardware with much larger EPC sizes in the future. We thus assume that the chosen dataset size is indicative of how future versions of CreDB will perform on larger datasets. We use py-tpcc[1], a Python implementation of TPC-C, for all measurements. For CreDB, data is stored normalized. In particular, each order is a distinct object and not part of the client's record.

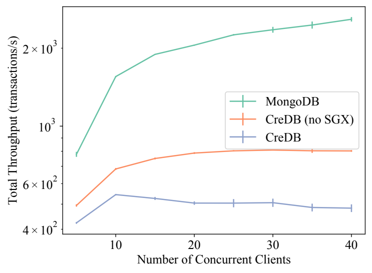
Figure 2a visualizes the observed performance, broken down for each query type. Each setup was evaluated under the number of clients that yielded the highest overall throughput. As expected the version of CreDB that runs without a trusted environment performs significantly better.

Figure 2b shows both systems, as well as MongoDB, over a changing number of clients to visualize the impact of limited amounts of protected memory. We evaluated MongoDB on denormalized data, as we observed it to perform better than MongoDB on normalized data. In these experiments, MongoDB uses the WiredTiger storage engine writing to a memory-mapped

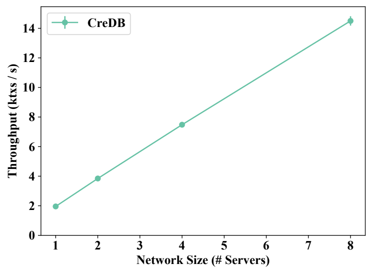
[1] https://github.com/apavlo/py-tpcc



(a) The performance of CreDB under a TPC-C workload compared to CreDB without SGX



(b) CreDB's performance on TPC-C under a changing number of clients compared to CreDB without SGX and MongoDB



(c) Performance of a network of CreDB nodes: Throughput scales linearly with the number of servers in the network

Figure 2: Macrobenchmark results

location. Note that MongoDB has no support for ACID transaction and its performance is therefore not degraded by transaction aborts. The comparison thus solely serves as an ideal baseline. We observe that each system scales up with an increasing number of clients. However, CreDB's throughput quickly reaches its peak of about 500tx/s. We pinpoint this limitation to the fact that once the EPC memory size is exhausted, threads will start competing for memory. The variant of CreDB without SGX yields in about twice the performance until concurrent transactions become the main bottleneck. MongoDB outperforms both implementations of CreDB due to the differences in application semantics described in the previous section. We attribute the better performance of MongoDB further to a lack of optimizations for secondary indexes in CreDB.

4.2 Network Scaling

We evaluated how a network of CreDB nodes scales with the number of servers. Our benchmark is built on top of the micropayment application sketched in Section 3.2. We allocate four physical machines of the medium configuration to host clients for this benchmark. Up to eight server partake in the network, each hosted on a dedicated machine with a medium configuration. Because these machines don't come with the most recent Intel CPU generation, we run the network scaling benchmark in simulation mode. Thus, absolute numbers are not reflective of what is achievable on hardware, but the overall scaling trend is reflective of the system's behavior.

In Figure 2c we gradually increase the number of banks and plot the overall throughput of the system. Client issue requests, consisting of transferring money from one to another account. Accounts are located at a remote service with a 20% probability and thus require cross node collaboration. To support such remote transfers, all banks in this setup are connected to all other banks. We observe that the throughput of the system increases with the number of CreDB nodes. Because the collection of liabilities and assets can quickly grow large, a single bank can only perform so many operations. Thus, while federated transactions incur an overhead when the number of banks is high, the benefit of having the set of clients split across multiple servers outweighs the cost of remote function evaluation.

5 Discussion and Future Work

More expressive object semantics The evaluated prototype is built to scale with large-scale datasets but does not accommodate large individual objects. In particular, large objects are not able to fit in enclave memory and cannot be efficiently sharded into multiple blocks. CreDB currently mitigates this problem by supporting collection-wide policies, which allow to break down application logic into multiple objects that are all part of the collection.

Another common problem in databases that this paper does not address is schema consolidation, where different parties may not structure their stored objects in the same manner. We envision a simple type system where objects, both locally and remote, can be checked against a specification. This can then be used to not only check the object for a specific structure but also to verify its policy. Similarly, specifications can check other functions that are part of the object and speed up PFE, by defining a standard protocol between multiple nodes.

Side-channel Attacks Side-channel attacks, which is attacks that observe the application's behavior through non-standard communication, such as looking at its CPU or cache usage, are of constant interest in the security community. Thus, several papers have addressed how the confidentiality of trusted hardware enclaves can be broken using such attacks [36]. Most of these attacks benefit from the fact that weak cryptographic code, e.g., where application secrets modify the control flow, is executed inside the enclave. While preventing CreDB nodes from side-channel attacks is beyond the scope of the paper, all cryptographic code in the enclave is implemented using constant-time libraries. Still, we expect future versions of CreDB will need to be amended as other such side-channel attacks are discovered.

Further, in scenarios where we envision the deployment of autonomous blockchains, there is typically a well-known counterparty, in a legal relationship, that can be held accountable. We propose the usage o this system for applications where values do not exceed the minimum of the cost for an SGX attack (in the hundreds of millions of dollars) and the value of the counterparties assets.

6 Related Work

Enforcing Policies on Data TEEs are one specific instance of secure hardware, which has thoroughly been leveraged by previous work to provide high integrity applications. The Nexus Operating System and its associated authorization logic [38] allow enforcing policies on applications, by providing an operating system rooted in a trusted platform module (TPM). TPMs require full trust in the computation stack. TEEs, on the other hand, provide a "reverse sandbox" that shield enclave code from potentially malicious host operating systems. Thus, CreDB only requires trust in the CPU and enclave code.

Information flow control is another common technique to enforce data policies on a programs execution. SIF [9] and Fable [35] use a combination of static and dynamic information flow tracking to enforce policies through compiler and runtime. Fabric [26] extends this paradigm to the distributed setting. While such techniques can protect data from malicious code, they cannot defend from other attackers, and are, thus, orthogonal to the mechanisms described in this paper.

In a distributed setting, specific Byzantine fault-tolerant consensus protocols can be used to shield a system from misbehaving principals [8, 6, 29]. In such an environment, the trust lies in the network itself and a large fraction of nodes behaving honestly. Permissioned blockchains have adopted these protocols, which careful selection of committee members and need a higher number of replicas than the approach described in this paper. Further, they do not shield from data leakage and cannot enforce access controls without substantial additional measures.

Ensuring Data Integrity Tamper-evident logs allow detecting Byzantine behaviors of storage servers [24, 43] and more complex applications [16]. While most of these mechanisms only provide fork consistency, A2M [10] uses trusted hardware to achieve strong consistency in such a setup. However, even if an audit mechanism provides strong consistency, to ensure detection of misbehavior, it requires that clients are honest and communicate with each other. Further, misbehavior can be detected only after the fact which is not a strong enough guarantee for many applications.

TrInc [22] provides a monotonic incrementer implemented in trusted hardware. Systems like CreDB can benefit from TrInc as it helps to protect from staleness attack, for example after an enclave is restarted. TrInc can also be used as a primitive to build Byzantine fault-tolerant systems with fewer replicas. However, it cannot be used to enforce access control to data.

Proof of Retrievability (PoR) [19, 37] allows verifying that a remote service indeed holds a dataset. PoR assumes a single client and thus is not suitable for some of CreDB's use cases. One possible application for PoR in CreDB would be to verify replication of encrypted data on a third party.

Concerto [2] is a datastore that achieves strong consistency using server-side integrity verification. Due to batch verification, this approach achieves much higher performance than other mechanisms [23]. However, Concerto ensures only data integrity and does not guard the data from unwanted accesses. Guardat [40] shields data from malicious applications by enforcing policies in the storage layer. CreDB takes this concept a step further and enforces policies using trusted hardware.

Encrypted Databases If policy enforcement is not a requirement, i.e., users trust each other, operating on encrypted data might be sufficient to achieve confidentiality. Maheshwari et al. [27] presented one of the first encrypted databases. Their system stores hashes of the encrypted data in a small trusted hardware module to protect from tampering.

CryptDB [32] and Monomi [39] rely on homomorphic encryption of data. To make such a scheme efficient CryptDB does not encrypt all data and only supports a subset of the SQL language. TrustedDB [4] and Cipherbase [1] overcome this limitation by running queries on encrypted data using a trusted hardware module. All of these systems, to our knowledge, assume that clients trust each other. In contrast, the policy enforcement and accountability features in CreDB are designed with multiple distrusting clients in mind.

Protecting Applications and Data using TEEs Previous work demonstrated how to run mostly unmodified applications in trusted virtual machines [5] or containers [3] executing in a TEE. On a high level, the main difference between these systems and CreDB is the choice of abstraction. CreDB can provide applications with a trusted storage system, without having to execute the application itself entirely in a trusted environment. Systems build on top of the CreDB API may thus yield in higher performance, with the tradeoff that the application has to be ported to this new API.

Ryoan [18] and Opaque [44] explore protecting data that is processed in the cloud using TEEs. The former allows a static network of enclaves to process each other but does not provide a mechanism for durable tamper-proof storage of data. The latter provides an efficient mechanism for read-only queries on private data. CreDB leverages a policy system similar to Royan with the addition that it has a notion of identities. Unlike both previous systems, CreDB further allows for a dynamic network of enclaves.

EnclaveDB [33] and PESOS [21] provide similar mechanisms as Cipherbase and TrustedDB but implemented using Intel SGX. Similar to our evaluation both systems yield better performance than the usage of dedicated HSMs. EnclaveDB is currently limited to a set of clients and transactions specified at compile time of the transaction. Further, to our knowledge, it does not support federation of database nodes or timeline inspection. PESOS is a low-level object storage system yielding high throughput by relying on trusted storage technologies, a mechanism CreDB could leverage as well.

7 Conclusion

This paper introduced autonomous blockchains, a novel class of datastores providing an immutable, eidetic ledger of all changes made to the data. We believe that autnomous blockchains provide every desired property of conventional blockchains, and do so without reliance on third parties, high energy consumption, or leakage of private data.

Finally, this work demonstrated that CreDB allows building high integrity distributed applications with relatively low effort. Benchmarks show that this approach can handle hundreds of complex transactions a second on a single node. We conclude that CreDB's design yields high performance compared to state-of-the-art permissioned and permissionless blockchains.

References

[1] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. Orthogonal Security with Cipherbase. *CIDR,* 2013.

[2] Arvind Arasu, Ken Eguro, Raghav Kaushik, Donald Kossmann, Pingfan Meng, Vineet Pandey, and Ravi Ramamurthy. Concerto: A High Concurrency Key-Value Store with Integrity. *Proceedings of the 2017 ACM International Conference on Management of Data,* pages 251–266, 2017.

[3] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, Dave Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. SCONE: Secure Linux Containers with Intel SGX. *Symposium on Operating System Design and Implementation,* pages 689–703, 2016.

[4] Sumeet Bajaj and Radu Sion. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering,* 26(3):752–765, 2014.

[5] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding applications from an untrusted cloud with Haven. *ACM Transactions on Computer Systems,* 33(3), 2015.

[6] Iddo Bentov, Rafael Pass, and Elaine Shi. The Sleepy Model of Consensus. *IACR Cryptology ePrint Archive,* 2016.

[7] Christian Cachin. Architecture of the Hyperledger blockchain fabric. *Workshop on Distributed Cryptocurrencies and Consensus Ledgers,* 2016.

[8] Miguel Castro, Barbara Liskov, and others. Practical Byzantine fault tolerance. *Symposium on Operating System Design and Implementation,* pages 173–186, New Orleans, Louisiana, February 1999.

[9] Stephen Chong, Krishnaprasad Vikram, and Andrew C. Myers. SIF: Enforcing Confidentiality and Integrity in Web Applications. *USENIX Security,* 2007.

[10] Byung-Gon Chun, Petros Maniatis, Scott Shenkert, and John Kubiatowicz. Attested Append-only Memory: Making Adversaries Stick to Their Word. *Symposium on Operating Systems Principles,* pages 189–204, Stevenson, Washington, October 2007.

[11] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive,* 2016:86, 2016.

[12] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, and others. On scaling decentralized blockchains. *International Conference on Financial Cryptography and Data Security,* pages 106–125, 2016.

[13] Cynthia Dwork. Ask a better question, get a better answer: A new approach to private data analysis. *ICDT,* pages 18–27, 2007.

[14] Ariel Ekblaw, Asaph Azaria, John D. Halamka, and Andrew Lippman. A Case Study for Blockchain in Healthcare: "MedRec" prototype for electronic health records and medical research data. *Proceedings of IEEE Open & Big Data Conference,* 2016.

[15] Ariel J. Feldman, William P. Zeller, Michael J. Freedman, and Edward W. Felten. SPORC: Group Collaboration using Untrusted Cloud Resources. *Symposium on Operating System Design and Implementation,* Vancouver, Canada, October 2010.

[16] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical Accountability for Distributed Systems. *Symposium on Operating Systems Principles,* pages 175–188, Stevenson, Washington, October 2007.

[17] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.,* pages 463–492, 1990.

[18] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. *Symposium on Operating System Design and Implementation,* pages 533–549, 2016.

[19] Ari Juels and Burton S. Kaliski, Jr. Pors: Proofs of Retrievability for Large Files. *ACM Conference on Computer and Communications Security,* pages 584–597, 2007.

[20] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmung Wong. Zyzzyva: Speculative byzantine fault tolerance. *ACM SIGOPS Operating Systems Review,* pages 45–58, 2007.

[21] Robert Krahn, Bohdan Trach, Anjo Vahldiek-Oberwagner, Thomas Knauth, Pramod Bhatotia, and Christof Fetzer. PESOS: Policy Enhanced Secure Object Store. *European Conference on Computer Systems,* 2018.

[22] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. TrInc: Small Trusted Hardware for Large Distributed Systems. *Symposium on Networked System Design and Implementation,* pages 1–14, Boston, Massachusetts, April 2009.

[23] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. *SIGMOD International Conference on Management of Data,* pages 121–132, Chicago, Illinois, June 2006.

[24] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis Shashas. Secure Untrusted Data Repository (SUNDR). *Symposium on Operating System Design and Implementation,* San Francisco, California, December 2004.

[25] Jinyuan Li and David Mazières. Beyond One-Third Faulty Replicas in Byzantine Fault Tolerant Systems. *Symposium on Networked System Design and Implementation,* Cambridge, Massachusetts, April 2007.

[26] Jed Liu, Michael D. George, K. Vikram, Xin Qi, Lucas Waye, and Andrew C. Myers. Fabric: A Platform for Secure Distributed Computation and Storage. *Symposium on Operating Systems Principles,* pages 321–334, Big Sky, Montana, October 2009.

[27] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to build a trusted database system on untrusted storage. *Symposium on Operating System Design and Implementation,* San Diego, California, October 2000.

[28] David Mazières and Dennis Shasha. Building secure file systems out of Byzantine storage. *ACM Symposium on Principles of Distributed Computing,* pages 108–117, Monterey, California, July 2002.

[29] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security,* pages 31–42, 2016.

[30] JP Morgan. Quorum. https://www.jpmorgan.com/global/Quorum, 2017.

[31] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[32] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. CryptDB: A practical encrypted relational DBMS. ACM, Technical Report, 2011.

[33] Christina Priebe, Kapil Vaswan, and Manuel Costa. EnclaveDB: A Secure Database using SGX. *EnclaveDB: A Secure Database using SGX,* 2018.

[34] R3. Corda. https://www.corda.net/, 2017.

[35] Nikhil Sawmy, Brian J. Corcoran, and Michael Hicks. Fable: A language for enforcing user-defined security policies. *Security and Privacy, 2008. SP 2008. IEEE Symposium on,* pages 369–383, 2008.

[36] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware guard extension: Using SGX to conceal cache attacks. *arXiv preprint arXiv:1702.08719,* 2017.

[37] Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical Dynamic Proofs of Retrievability. *ACM Conference on Computer and Communications Security,* pages 325–336, 2013.

[38] Emin Gün Sirer, Willem Bruijn, Patrick Reynolds, Alan Shieh, Kevin Walsh, Dan Williams, and Fred B. Schneider. Logical attestation: an authorization architecture for trustworthy computing. *Symposium on Operating Systems Principles,* pages 249–264, Cascais, Portugal, October 2011.

[39] Stephen Tu, Frans M. Kaashoek, Samuel Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. *International Conference on Very Large Data Bases,* pages 289–300, 2013.

[40] Anjo Vahldiek-Oberwagner, Eslam Elnikety, Aastha Mehta, Deepak Garg, Peter Druschel, Rodrigo Rodrigues, Johannes Gehrke, and Ansley Post. Guardat: Enforcing data policies at the storage layer. *European Conference on Computer Systems,* page 13, 2015.

[41] Shawn Wilkinson, Jim Lowry, and Tome Boshevski. Metadisk a blockchain-based decentralized file storage application. STORJ, Technical Report, 2014.

[42] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper,* 151, 2014.

[43] Aydan R. Yumerefendi and Jeffrey S. Chase. Strong Accountability for Network Storage. *Trans. Storage,* 3(3), 2007.

[44] Wenting Zhang, Dave Ankur, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. *Symposium on Networked System Design and Implementation,* pages 283–298, 2017.

[45] Guy Zyskindand, Nathan Oz, and others. Decentralizing privacy: Using blockchain to protect personal data. *Security and Privacy Workshops (SPW), 2015 IEEE,* pages 180–184, 2015.