A Vision for Autonomous Blockchains backed by Secure Hardware

Kai Mast
Cornell University

Lequn Chen
University of Washington

Emin Gün Sirer
Cornell University

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 raises the question if blockchains, in their current form, are the only class of datastores that can provide such strong integrity guarantees.

We introduce *autonomous blockchains*, an architecture based on free-standing, immutable, eidetic databases that implement independent timelines, linked together through interactions. Autonomous blockchains can be realized using trusted execution environments in combination with audit mechanisms. This architecture does not only provide blockchain-like integrity and auditability guarantees but also supports storing and querying 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. Evaluation shows that this design can achieve high throughput while providing stronger integrity guarantees than conventional datastores.

CCS Concepts • **Security and privacy → Management and querying of encrypted data**; **Information accountability and usage control**.

SysTEX '19, October 27, 2019, Huntsville, ON, Canada

ACM ISBN 978-1-4503-6888-9/19/10…$15.00
https://doi.org/10.1145/3342559.3365333

ACM Reference Format:
Kai Mast, Lequn Chen, and Emin Gün Sirer. 2019. A Vision for Autonomous Blockchains backed by Secure Hardware. In *4th Workshop on System Software for Trusted Execution (SysTEX '19), October 27, 2019, Huntsville, ON, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3342559.3365333

1 Introduction

Many high-value applications require reliable and immutable storage of data across multiple distrusting parties [13, 35, 39]. 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 trust in the entire application stack and host operating system(s) by all parties. Earlier work focused on accountable systems [23, 37], which ensure integrity by allowing clients to audit the log for states they have observed previously. But audit mechanisms by themselves cannot protect against third-parties accessing the data.

Blockchains have recently emerged to fill this void by providing an *immutable*, i.e. append-only, data feed across a trustless network of peers. Public/permissionless blockchains [29, 36] operate across an open network and achieve consensus through mechanisms known as proof-of-work or proof-of-stake, both of which require massive replication of data and computation in their current form. Thus, public blockchains are energy inefficient, hard to scale and suffer from limited throughput and high latencies [11]. Further, due to their open and distributed setting, they cannot be used to store private or confidential data. Private/permissioned blockchains [7, 28], on the other hand, achieve consensus across a pre-defined committee [8, 19, 24]. This approach necessarily requires specially designated committee nodes, often leaks meta-data, such as which clients interact with which others, at what frequency, and is limited in performance as existing protocols require all-to-all communication across participants.

The main contribution of this paper is to outline our vision of *autonomous blockchains*, a class of data stores that provide a self-standing, permanent, tamper-proof event-log of all transactions and data, and allow to facilitate confidential computations on such data. We consider a system as system as self-standing if it does not rely on an external consensus mechanism to operate. The central abstraction provided by autonomous blockchains is an append-only log [9]: 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, optionally, allow data sharing with other blockchain instances.

Data and computation across different autonomous blockchains can be linked together through *federated transactions*. This architecture thus enables creating networks that share designated data items and executing computations on remote parties. This is in stark contrast to conventional blockchains, which replicate all data across all nodes. As a result, communication between autonomous blockchain nodes is kept minimal.

Autonomous blockchains support application-defined policies [20, 34] and introspecting functions, which, in turn, enables mutually-distrusting parties to interact. Specifically, autonomous blockchains allow every object to be inseparably associated with a *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 can be leveraged to guarantee 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 [16, 38]. The primary use of this functionality is to execute a vetted function 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 can vet the function a priori.

The usage of hardware-based *trusted execution environments (TEEs)* enables nodes to trust another participant's computation without trusting the administrator of that system. Audit mechanisms prevent malicious actors from rolling back the TEE's state and ensure that each blockchain node only runs a single TEE. Because each service can run their own blockchain backed by a TEE, the throughput scales with the number of nodes in the system.

2 The Autonomous Blockchain Model

At a high level, every autonomous blockchain implements a secure database that clients, as well as other nodes, can connect to. Each blockchain instance maintains a distinct ledger, comprising a timeline of events and a datastore. 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 to the network through one or multiple such blockchain instances and do not need special hardware support. Providing a conventional database interface allows for straightforward porting of existing applications to this new abstraction. 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.

Applications are written against an interface that is a superset of a transactional key-value storage API. They either interface with the datastore through a secure network connection or execute inside the datastore's TEE in the form of policies and stored procedures.

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, including 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 and database administrators 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.

We assume that every autonomous blockchain implementation has access to a TEE. In particular, we assume that as long as the CPU itself is not tampered with, TEEs have the following functionality. First, application data is protected from third-party access. Second, a third party cannot influence the execution of the TEE, except for how many CPU cycles are allocated to it. Third, the TEE can prove remotely that it is functioning correctly and has not been compromised (*remote attestation*).

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 have types such as lists, dictionaries, strings, binary data, and numeric values. 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 [15]. 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. Crucially, unrelated events 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. This can be done by checking the functions hash against a whitelist or by formally analyzing the program code. Much past work concerns itself with the analysis of function properties, including for information leakage [12] and information flow [25], so the mechanisms of this vetting are beyond the scope of 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 the function identifier and a certified result. Autonomous blockchains identify functions through the hash of their bytecode, similar to how the Ethereum Virtual Machine operates. The witness is signed by a persistent key associated with the blockchain instance of the executing party, which allows any third party to verify the authenticity of the result.

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 [10]. 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 allow associating 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 the 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 leads 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 refer to the state of other objects. Policies have access to an interface that exposes information about the attempted operation and the affected object(s).

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.

2.5 Witnesses and Fraud Proofs

Even backed by a TEE, autonomous blockchains are still beholden to the *database administrator (DA)*, which we address in this section. DAs are entities that configure and runs a specific blockchain instance. Aside from configuring the TEE itself, the DA is also in charge of replicating the on-disk state of the blockchain to provide fault-tolerance. In particular, if the machine running the TEE becomes unavailable, they can restart the blockchain instance using the replicated disk-state on the same or different physical machine.

While the DA cannot arbitrarily change stored data, access to the machine which the enclave runs on enables them to pause, rollback, or clone the TEE at any point in time. The database state may be encrypted, cryptographically signed,

and replicated, but is stored on untrusted hardware, which can be exploited by the DA to roll back to a previous state. Similarly, a DA may attempt to run multiple instances of the same blockchain node to provide different views of the world to different parties. Mitigations for these attacks exist, such as monotonic hardware counters or Byzantine fault-tolerant replication schemes, but are too slow or require trust in third parties.

The autonomous blockchain abstraction includes the notion of fraud proofs, which demonstrate misbehavior of a DA. Entities that interact with the blockchain instance collect records of its behavior in the form of witnesses. If malicious behavior is detected, it can be demonstrated using two conflicting witnesses. For example, if the DA runs more than one TEE with the same key, the witnesses might certify two different database states at the same point in time. A fraud proof then is merely a certificate containing the set of conflicting witnesses.

The autonomous blockchain architecture leverages economic incentives in combination with fraud proofs to enforce the correct behavior of DAs. In scenarios where we envision the deployment of autonomous blockchains, the DA is typically a well-known counterparty, in a legal relationship, that can be held accountable. Aside from allowing for legal actions using fraud proofs, we envision a *kill switch* in the form of a fraud proof. Here, an enclave will shut down after receiving a fraud proof, which provides a strong negative incentive for a DA that wants to keep the blockchain running to maintain their revenue stream, thus they are incentivized to not conduct fraudulent behavior.

Clients may further leverage witnesses to overcome the lack of availability of a blockchain node. Witnesses enable to bootstrap a self-standing, read-only snapshot of the user data. For data items that are not protected by security policies, witnesses can provide the data in the form of a digital signature. For other data, witnesses allow bootstrapping a local trusted execution environment, where the data is protected by the same policies as in the original datastore. This scheme is analogous to clients storing blocks locally in conventional blockchain systems.

3 Prototype Evaluation

We implemented and evaluated a fully-functional prototype of an autonomous blockchain in the form of CreDB. 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, a single server is hosted on the big configuration while clients execute across multiple medium configurations.

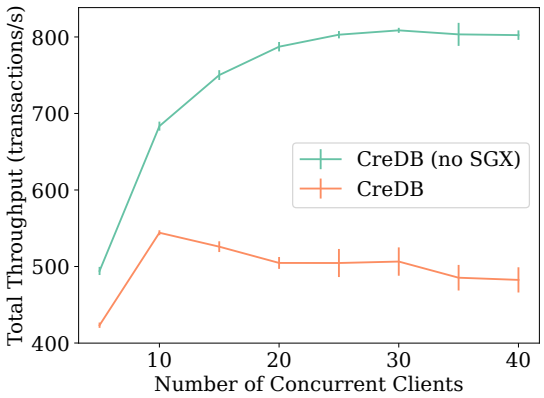

Figure 1. CreDB's performance on TPC-C under a changing number of clients compared to CreDB without SGX

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. For CreDB, data is stored normalized. In particular, each order is a distinct object and not part of the client's record.

Figure 1 shows both systems over a changing number of clients to visualize the impact of limited amounts of protected memory. 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.

4 Related Work

Software-Based Attacks on TEEs 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 [32]. 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, it has been shown that speculative execution on Intel CPUs can be exploited to leak private information of processes and even SGX enclaves [6, 18]. Some of these attacks can be mitigated by upgrading the microcode of Intel CPUs, while others require to disable certain features such as HyperThreading. This means mitigating such attacks comes with a noticeable amount of performance loss. However, an autonomous blockchain is orders of magnitudes faster than other blockchain systems, which means it will still perform well compared to other systems with the mitigations applied. We further assume the usage of open-source TEEs, such as Sanctum or Keystone, will help to avoid similar attacks, as a large developer community can vet the hardware implementation and microcode of the processor.

Ensuring Data Integrity CreDB builds on top of previous work on tamper-evident logs, which allow detecting Byzantine behaviors of storage servers [23, 37] and other applications [14]. While most of these mechanisms only provide fork consistency, A2M [9] and TrInc [21] use trusted hardware to achieve strong consistency in such a setup. However, audit mechanisms by themselves cannot provide data privacy or policy enforcement.

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 [22]. However, Concerto ensures only data integrity and does not guard the data from unwanted accesses. Guardat [34] shields data from malicious applications by enforcing policies in the storage layer.

In a distributed setting, specific Byzantine fault-tolerant consensus protocols can be used to shield a system from misbehaving principals [5, 8, 27]. 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.

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. [26] 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 [30] and Monomi [33] 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.

EnclaveDB [31], Shieldstore [17], and PESOS [20] provide secure storage using Intel SGX instead of dedicated secure hardware, yielding in better performance. While a promising first step, to our knowledge, none of these systems 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.

5 Conclusion

Autonomous blockchains provide a high-level datastore abstraction, including access to an immutable and eidetic ledger of all changes, on top of a low-level trusted execution environment. We believe that this abstraction provides every desired property of conventional blockchains, and do so without reliance on third parties, high energy consumption, or leakage of private data. While our design requires additional trust in the hardware and enclave code, as well as relies on certain economic incentives, we believe this is a valid tradeoff between performance and safety for many use cases.

Finally, we presented a working prototype which demonstrated that this abstraction allows building high-integrity 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 the autonomous blockchain design yields high performance compared to state-of-the-art permissioned and permissionless blockchains, and lays the groundwork for novel designs of trusted storage technologies.

Acknowledgements We are grateful to everyone who provided feedback on the project and earlier versions of the manuscript, especially Soumya Basu, Adem Efe Gencer, Kevin Negy, and the anonymous reviewers.

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

[6] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. *27th {USENIX} Security Symposium ({USENIX} Security 18),* 2018.

[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] 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.

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

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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, Savannah, Georgia, November 2016.

[17] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. ShieldStore: Shielded In-memory Key-value Storage with SGX. *European Conference on Computer Systems,* Dresden, Germany, March 2019.

[18] Paul Kocher, Jann Horn, Anders Fogh, and Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. *40th IEEE Symposium on Security and Privacy (S&P'19),* 2019.

[19] 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.

[20] 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,* Porto, Portugal, April 2018.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

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

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

[32] 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.

[33] 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, Trento, Italy, August 2013.

[34] 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, Bordeaux, France, April 2015.

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

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

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

[38] 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, Boston, Massachusetts, March 2017.

[39] 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.