TrustDBle: Towards Trustable Shared Databases

Muhammad El-Hindi
TU Darmstadt

Simon Karrer
TU Darmstadt

Gloria Doci
TU Darmstadt

Carsten Binnig
TU Darmstadt

Research Short Paper

ABSTRACT

In this paper, we present *TrustDBle* [ˈtrʌstəbəl], a trustable DBMS which can be used for shared access by multiple parties. *TrustDBle* is based on our previous work on *BlockchainDB* which uses blockchains as an auditable storage to guarantee transparency and auditability of any data change. *TrustDBle* extends this work with a secure OLTP engine that implements verifiable ACID-compliant transaction execution on shared data while preserving scalability. In this work we discuss the main design choices and considerations for building trustable data management systems and show first results from our work on *TrustDBle*.

1. INTRODUCTION

Motivation. Databases (DBs) and database management systems (DBMSs) are a proven technology to efficiently store and query data. They scale very well to support a large number of nodes and amounts of data. Further, they offer standardized interfaces, such as SQL, and their use is well established among developers and operation teams. However, DBMSs are centralized solutions that assume a single database owner who solely can query and modify the data. Yet, in many use cases the data might belong to several different entities. For example, in typical supply chain scenarios the information about goods along the supply chain needs to be tracked by many different companies.

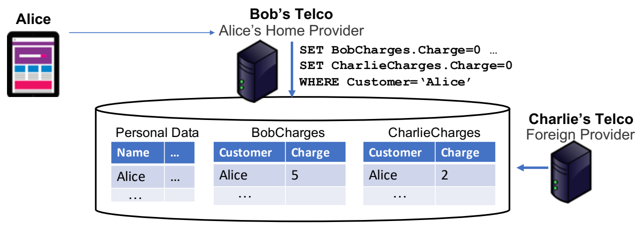
The traditional solution for shared data access is that each company keeps a local copy of the database and uses custom-developed data integration solutions to synchronize state between the different instances. However, this way of data sharing comes with many different drawbacks. For example, once data leaves the database of the data producer it is not transparent for this data producer to whom the data will be made available or how the data might be altered. This opens up the door for data misuse in different directions. For instance, in a food supply chain, the best-before date might be "faked" to resell products even after they expired.

Third International Symposium on Foundations and Applications of Blockchain (FAB '20) May 1, 2020, Santa Cruz, California, USA.

A solution to this problem can be achieved with blockchains (BCs) (or distributed ledgers in general) that can be used as shared databases. First, BCs record all updates in an immutable manner and thus provide an auditable storage that can be used to find out how data was changed over time by whom. Second, BCs require consensus of the participants before a data update can be committed. While BCs thus seem to address the aforementioned issues, they lack the performance and scalability required for many use cases. For example, BCs offer transaction rates of $100's$ or $1000's$ transactions per second, while databases can achieve $100,000's$ transactions per second. Further, BCs introduce new interfaces and programming models that many organizations are not familiar with. Due to the lack of standards almost every blockchain platform even uses different interfaces, which increases the complexity and risk for adopting this technology in enterprises.

Contribution. This paper presents *TrustDBle* [ˈtrʌstəbəl], a trustable DBMS which can be used for shared access by multiple parties. *TrustDBle* is based on our previous work [4] that shows how a scalable storage manager can be built on top of blockchains to provide high performance and audibility of all data changes at the same time. However, the shared storage manager in [4] is limited to simple put/get-operations. *TrustDBle* thus extends our previous work [4] with a secure execution engine that implements verifiable transaction execution on shared databases while preserving the scalability of [4]. Verifiable transaction execution means that the DBMS engine itself guarantees a correct (i.e., ACID-compliant) execution of transactions. For example, to achieve isolation *TrustDBle* introduces a lock manager that resides in a trusted execution environment and thus guarantees correct serializable execution of multi-statement transactions from different parties. In this work, we discuss the main design choices of all involved components when building a trustable DBMS and show first results from our work on *TrustDBle*. This includes the aforementioned verifiable transaction execution engine, but also other components needed to establish trust. For example, in many data sharing scenarios it is important for a data owner/producer to track who accessed data during which queries.

Outline. The remainder of this paper is organized as follows: The next section discusses the guarantees and requirements that a trustable data management system should provide. Then, we present our architecture of *TrustDBle* and the main design choices to establish trust in DBMSs. After-



```
SET BobCharges.Charge=0 …
SET CharlieCharges.Charge=0
WHERE Customer='Alice'
```

Name	…
Alice	…
…	

Customer	Charge
Alice	5
…	

Customer	Charge
Alice	2
…	

Figure 1: Shared DB of Two Telco Providers

wards, initial results of running a OLTP benchmark (SmallBank) simulating a typical data sharing scenario is shown in our evaluation of *TrustDBle*. Finally, we conclude with the discussion of related work and a summary.

2. PROPERTIES OF A TRUSTABLE DBMS

In telco scenarios, mobile phones of users are often being used outside the range of their home networks and can also connect to an available cell network of another provider. While this causes charges for users in different cell networks, a user still pays her charges only to the telco provider of her home network, who clears the debts of the user with the second provider. Figure 1 depicts such a scenario where Alice has a home provider (Bob) but was also using the network of another provider (Charlie). Through the abstraction of the shared database, Alice could not only settle her charges with Bob, but also Bob could clear the debts of Alice with Charlie using transactions on the shared database.

However, in a setup in which the involved parties do not trust each other, it is required that the DBMS providing the shared database abstraction guarantees that data can only be accessed and manipulated as agreed by all parties (e.g., if Alice clears her charges with Bob, Bob has to clear the debts with Charlie based on the agreed terms of the mobile contracts). Further, any data modification (this includes malicious modifications) should be recorded in a tamper-proof manner. For example, if Charlie manipulates the phone charges of Alice to his own favor (e.g., by increasing his charge state for Alice and decreasing Bob's) it should be possible for Alice to detect this malicious change using the tamper-proof history of all changes in the DBMS.

To achieve such guarantees the following three criteria must be met by a trustable DBMS:

Requirement 1 - Data Sovereignty. While multiple Telco providers could use the same shared database, Alice should be able at any time to control who has access to her (personal) data. For example, only the home provider can read information such as the personal address.

Requirement 2 - Verifiable Processing. As shown in the example, transactions are used to execute complex data manipulations that consist of multiple operations and access multiple records. Especially, in a collaborative setting where multiple parties control different parts of the data, it must be guaranteed that all parties execute transactions in a correct and verifiable way. In this context, correct and verifiable execution means that a DBMS is able to prove that a transaction was executed in an ACID-compliant way by all involved parties. In the above scenario, for example, this means that Alice's settlement transaction is correctly executed on both Bob's and Charlie's charge tables.

Requirement 3 - Auditable Storage. The existence of verifiable processing as described above, is often not sufficient if one could potentially tamper with or remove data from the DBMS after processing. For example, as mentioned earlier, Charlie might tamper with the data to manipulate the current state of charges to his advantage. Hence, auditability requires that all information needed to validate how data was updated over time must be tamper-proof. Further, data management systems must provide new interfaces to make databases easily auditable. This could be done by users directly or by additional applications or services.

3. ARCHITECTURE & KEY CONCEPTS

TrustDBle is a distributed database that is build around the key requirements of a trustable DBMS as discusseed before. Similar to a traditional database, it offers users a standard SQL-Interface and provides ACID properties. However, it also guarantees sovereignty, verifiable processing and auditable storage.

Figure 2 shows an overview of *TrustDBle*'s architecture. Similar to our work in [4], we build a database layer on top of blockchains as a storage layer. Thereby, *TrustDBle* also makes use of sharding in the storage layer to enable scalability. In this work, we extend the database layer with a secure execution engine that enables verifiable processing and sovereignty.

In the following, we provide more details on *TrustDBle*'s different layers and explain how the previously mentioned properties of a trustable DBMS are achieved.

3.1 Auditable Storage

TrustDBle's storage layer utilizes blockchains as a persistent, auditable storage backend. Blockchains enable us to store data in a tamper-proof way and record modifications of the data on the blockchain. Thereby, as shown in [4], database techniques such as sharding help *TrustDBle* to overcome the scalability limitations of blockchains.

Despite sharding, we also aim to implement further optimization such as caching in the storage layer to speed up data access. Caching will enable *TrustDBle* to treat the auditable storage as another layer in the memory hierarchy of the database. With the help of *verifiable processing* it can be guaranteed that data is maintained in a tamper-proof way in memory or on disk until it has been persisted to the auditable storage.

3.2 Verifiable Processing & Data Sovereignty

On top of the auditable storage, we offer components for verifiable transaction processing and sovereignty. In the following, we mainly focus on verifiable transaction processing and only briefly touch on sovereignty.

Verifiable Processing. With verifiable transaction processing we refer to two aspects. First, all parties involved in a shared DBMS can only execute transactions which all parties agreed on. Second, when transactions are executed, parties can make sure that those transactions where executed correctly (i.e., following the ACID properties a non-shared DBMS would give).

One approach to achieve verifiable transaction processing is to use smart contracts to implement the transaction pro-

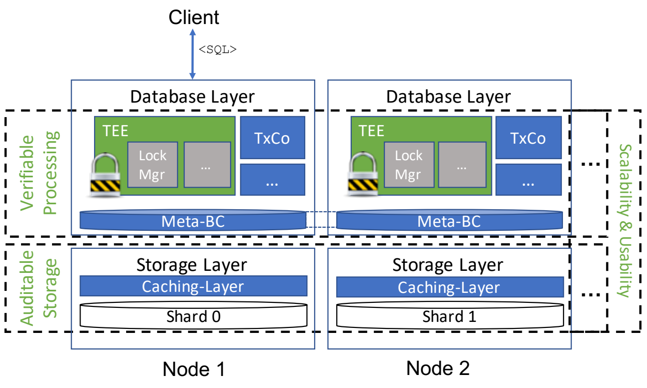


Figure 2: TrustDBle Architecture

cessing logic, e.g., as done in [3]. However, we found that this approach suffers from the same scalability limitations as blockchains themselves and adds additional complexity since new transactions logic needs to be re-implemented as smart contracts.

To overcome these issues, we implement a secure transaction processing engine outside the blockchain (i.e., on top of the auditable storage) with the help of a Trusted Execution Environment (TEE) (green box in Figure 2). In our current prototype we use Intel Software Guard Extensions (SGX) to provision the TEE. Intel SGX establishes TEEs as so called hardware enclaves, that are protected by the CPU. Thereby, the CPU provides the enclave with a special address space that is only accessible by the trusted code inside the enclave.

One major limitation of SGX, however, is that it only provides a small portion of protected memory. To overcome this challenge, *TrustDBle* does not place all transaction execution components in the trusted environment. Instead, we place only those components inside the TEE which are required to verify that the ACID properties have been fulfilled. For example, to provide verifiable isolation, we only implement the lock manager (LockMgr) of our database layer inside the trusted environment.

We use two key criteria to decide which component to implement inside the TEE. First, the component should only maintain a small state inside the protected memory region. In the case of the lock manager, for instance, we only need to maintain the lock table of the lock manager in the TEE. Second, verifiable processing must depend on the correct behaviour of a node. As an alternative approach, we employ verification protocols similar to [4] to verify the correct behaviour of a component. This is for example used to verify the correct behaviour of the transaction coordinator (TxCo) component and achieve verifiable atomicity.

Data Sovereignty. Furthermore, we plan to use TEEs to guarantee sovereignty. The key idea is that *TrustDBle* encrypts data inside the TEE and only provides authorized parties access to the encryption key.

3.3 Scalability & Usability

Despite trust achievement being the core of *TrustDBle*, we do not want to scarify scalability and usability. *TrustDBle* achieves this with the help of two main ideas:
DB optimizations & interfaces: Scalability is achieved by making use of classical database techniques, such as sharding, distributed locking and caching. While these techniques introduce new challenges such as cross-shard transactions and cache-coherence, we want to address them without compromising trust.

Further, we want to provide users with familiar database interfaces and abstractions (e.g., ACID transactions and isolation levels). This way, *TrustDBle* can be integrated in existing enterprise architectures and serve as drop-in replacement for traditional databases that do not provide trust guarantees.
Adaptivity & Flexibility: Also, we want to achieve usability through adaptivity. As mentioned previously, *TrustDBle* supports different auditable storage backends. Therefore, *TrustDBle* can be used with any blockchain network that users might already be familiar with. Moreover, we plan to support connecting to local databases via an ODBC interface. This way, users can run queries that target local and *TrustDBle*'s data.

4. CURRENT STATE OF TRUSTDBLE

In the following, we report on the current state of *TrustDBle* and show our initial performance results of running a simple OLTP benchmark on *TrustDBle*.

4.1 Implementation Details

TrustDBle is still in its early stage and we mainly focused on verifiable transaction processing and not on data sovereignty so far. Our verifiable transaction processing engine currently provides verifiable isolation and atomicity guarantees (i.e., the A and I of ACID) which are based on a secure locking scheme and an auditable two-phase-commit (2PC) protocol for cross-shard transactions.

The secure locking scheme is implemented via sharded lock managers that run inside the TEE of a *TrustDBle* node. Only, transaction managers (TxMgrs) with a valid lock (i.e., signed by the LockMgr) are allowed to modify or access data. Sharding the lock manager enables us to prevent the lock manager from becoming a bottleneck. Availability, is achieved by persisting the state of lock managers to auditable storage. This way, other nodes in *TrustDBle* can recover from a lock manager fault. Moreover, our locking scheme supports the execution of transactions under different isolation levels. To that end, the trusted lock managers enforce that locks are acquired and released according to the specified isolation level.

Similarly, we utilize the auditable storage to log 2PC messages and detect or recover from a faulty transaction coordinator. Thereby, local transaction managers apply a *trust, but verify* strategy with respect to a transaction coordinator: While the transaction coordinator is responsible to collect and forward 2PC messages and decisions to local TxMgrs, each local TxMgr also logs messages to a shared meta-blockchain which all TxMgr can access. Messages in this meta-blockchain are used to verify decisions of the transaction coordinator. Similarly to our previous work [4] we make use of deferred verification techniques to mask verification in the case of no failures.

4.2 Initial Results

Since *TrustDBle* combines database, BC technology and TEEs to overcome the limitations of blockchains, the focus of our evaluation is currently mainly on scalability. In particular, we want to show that our approach to verifiable

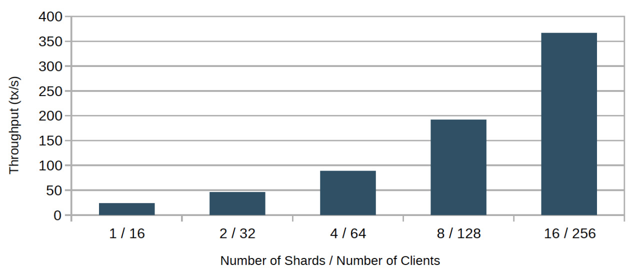


Figure 3: Scalability with the number of shards

processing can execute cross-shard transactions in a scalable way. To that end, we implement the Smallbank OLTP benchmark [1] and measure the performance of *TrustDBle* in different settings.

Figure 3 shows an experiment in which we increased the number of participants in a *TrustDBle* network. Thereby, the number of participants matches the number of shards, since each node was responsible for a separate shard. We further scale the number of clients from 16 to 256. Each client runs an update heavy workload of 1000s of transactions per shard under a read committed isolation level. All nodes are running as virtual machines in Microsoft Azure with 16 vcpus, 32 GB memory and Ubuntu 16.04 LTS as operating system. Hyperledger Sawtooth 1.04 is used as blockchain platform in the storage layer. This experiment shows that *TrustDBle* can execute cross-shard transactions in a scalable manner for a network with up to 16 nodes and multiple shards. In future, we plan to perform a more detailed evaluation and, e.g., study *TrustDBle*'s behaviour for larger network sizes. Moreover, we will analyze other scalability aspects (such as scalability with data size) and evaluate *TrustDBle*'s behaviour under attacks.

5. RELATED WORK

Several other work makes use of TEEs inside databases. EnclaveDB [7], for example, implements an entire signle-node database with the help of a TEE. *TrustDBle* is a distributed database, and extends the use of TEEs to a distributed setup. Further, EnclaveDB focuses on privacy and integrity, but not on a collaborative setup in which multiple parties share access to data. Further, all sensitive data is stored inside the TEE and therefore EnclaveDB requires support for large TEEs with hundred gigabytes of memory. In contrast, *TrustDBle* uses TEE only to secure few critical components of the system like the LockMgr.

Another area of related work are hybrid approaches that combine TEEs with blockchains to address the performance limitations of blockchains. Ekiden [2] uses a hybrid approach to get confidentiality and use TEEs to improve smart contract computation and scalability of the underlying blockchain. It provides privacy, but lacks usability, since it implements a new blockchain platform with custom interfaces and programming abstractions. TrustDBle is rather a database than a blockchain and provides standard database interfaces (e.g., SQL) and abstractions (e.g., ACID transactions and isolation levels). Another system using a hybrid approach is Hyperledger Avlon[8]. It combines trusted execution environments, such as Intel SGX, and blockchains. Howerever, ACID guarantees are not provided and instead it uses trusted oracles to offload certain computations from a blockchain.

Further, Blockchain databases like Veritas [5] as well as BigchainDB [6] aim to provide a similar abstraction of a shared database as *TrustDBle*. However, they differ in how they implement this abstractions. *TrustDBle* focuses on how the complexity of existing blockchains can be overcome with the help of an additional database layer, while relying on the auditability and tamper-proofness characteristics of blockchains.

6. CONCLUSION

We presented *TrustDBle*, which combines database, blockchain and secure hardware technology to implement a trustable data management system. It extends our previous work with a secure OLTP engine that implements verifiable ACID-compliant transaction execution on shared data while preserving scalability. In this paper, we discussed the properties that are required to build trustable DBMSs and explained how these guarantees are implemented in *TrustDBle*. In contrast to native blockchains, *TrustDBle* is able to fulfill these guarantees without scarifying scalability and usability. Our initial experiments show that *TrustDBle* successfully supports cross-shard transactions and allows us to scale the performance of the system by increasing the number of shards.

7. ACKNOWLEDGEMENTS

This research work has been funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. We also thank Microsoft Research for providing us with Azure resources for our evaluation.

8. REFERENCES

[1] M. J. Cahill et al. Serializable isolation for snapshot databases. *ACM Transactions on Database Systems (TODS)*, 34(4):1–42, 2009.

[2] R. Cheng et al. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 185–200, June 2019.

[3] T. T. A. Dinh et al. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD 17, page 10851100, New York, NY, USA, 2017. Association for Computing Machinery.

[4] M. El-Hindi et al. Blockchaindb: a shared database on blockchains. *Proceedings of the VLDB Endowment*, 12(11):1597–1609, 2019.

[5] J. Gehrke et al. Veritas: Shared verifiable databases and tables in the cloud. In *CIDR*, 2019.

[6] T. McConaghy et al. BigchainDB: a scalable blockchain database. *white paper, BigChainDB*, 2016.

[7] C. Priebe et al. EnclaveDB: a secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 264–278, May 2018.

[8] The Linux Foundation. Hyperledger Avalon. https://www.hyperledger.org/projects/avalon, 2020.