Verifiable Databases *

Dmitry Khovratovich

Sikoba Research

21st December 2018

1 Verifiable databases

1.1 Introduction

We consider the following scenario. One or more clients C_i read the database D from an untrusted source. Most queries are READ requests, however, occasionally authorized parties P_i may update the database. The application has the following requirements:

- Integrity: only modifications that satisfy certain rules (internal logic, writer's public keys, etc.) are allowed.
- Authenticity: the responses contain the actual database entries.
- Efficiency: the overhead for response creation, communication, and verification should be minimal.
- Stateless: the clients are not required to store the entire database or most of it between the calls.

1.2 Solutions

Solutions are divided into two groups based on whether the updates are permitted.

1.2.1 Merkle trees

A straightforward solution for the case when the updates are rare or trusted is the Merkle tree:

- Database entries are hashed: first in pairs, then in pairs of hashes from the previous step, and so on till the end. The resulting hash is called a root and is published.
- To add an item to a database of N entries costs $|\log N|$ hash function calls.
- To prove that the item X is part of the database with Merkle root Φ, it is sufficient to provide $|\log N|$ inputs to the hash functions path from the item to the root.

There exist modifications to the Merkle trees such as B^+ trees that are more suitable for update processing [LHKR06]. Here it is assumed that updates are provided and brodacast in plaintext and their authenticity is ensured just by digital signatures.

*Research supported by Fantom Foundation

1.2.2 Public-key based schemes for selections and joins

When certain class of queries such as selections and joins are considered, Merkle tree becomes inefficient as one would have to communicate the entire tree to ensure the correctness of the query. Special methods were developed to reduce overhead, with IntegriDB [ZKP15] and vSQL [ZGK+17]. They have the following properties:

- They need a trusted setup (possibly by clients) for the entire database, which takes days for million-item tables.
- The proof generation, communication and verification time do not depend on the database size, only on the query.
- The proof generation time for real-world million-item queries takes hours.
- Verification is almost instant.

1.2.3 ZKSNARKs

ZKSNARKs [PHGR13] is a generic technique to prove the computational integrity. Though it was primarily designed for zero knowledge proofs of correctness, it can be used to ensure the update correctness without zero knowledge. The procedure is as follows:

1. Encode all possible update logic as arithmetic circuits;
2. Create and publish proving keys for all circuits using some form of a trusted setup.
3. For each update, create a proof of correctness using the proving key for the circuit.
4. Distribute the proof alongside the update.

This technique has the following properties:

- All proofs are compact (<1KB) and instant to verify;
- The proofs are expensive to create, as the time is linear in the circuit size. On real-world scenarios (TPC-H) the proof construction time is estimated as days.
- The proving key must be created by a trusted party or in a secure MPC protocol to prevent proof forgeries.

These properties make ZKSNARKs feasible only for compact circuits.

1.2.4 Other generic proof systems

Bulletproofs [BBB+18] and STARKs [BBHR18, AHIV17] are possible alternatives to ZKSNARKs with respect to generic update proofs. They differ in the following:

- No trusted setup is needed;
- Proofs are not compact but rather logarithmic of the circuit size.
- Verification time is proof-linear in STARKs and circuit-linear in Bulletproofs.

Thus such systems are most relevant for rather large circuits/updates, but the implementations are not production-ready yet.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. IEEE, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018:46, 2018.

[LHKR06] Feifei Li, Marios Hadjieleftheriou, George Kollios, and Leonid Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD Conference*, pages 121–132. ACM, 2006.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society, 2013.

[ZGK+17] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *IEEE Symposium on Security and Privacy*, pages 863–880. IEEE Computer Society, 2017.

[ZKP15] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. Integridb: Verifiable SQL for outsourced databases. In *ACM Conference on Computer and Communications Security*, pages 1480–1491. ACM, 2015.