Concise UC Zero-Knowledge Proofs for Oblivious Updatable Databases

Jan Camenisch*, Maria Dubovitskaya*, Alfredo Rial†

*Dfinity, {jan, maria}@dfinity.org
†SnT, University of Luxembourg, alfredo.rial.work@gmail.com

IEEE CSF 2021
22/06/2021
Motivation

In commit-and-prove protocols, a prover P commits to her input and then proves in zero-knowledge (ZK) to a verifier V statements about the committed values. These steps are repeated and intertwined, i.e., commitments are updated, new ones formed, and additional proofs executed.

We regard commitments as a tool to maintain a database between P and V with read and write operations.

- **Write**: When P commits to a value, the value is written into the database.
- **Read**: When P proves a statement about a committed value, the value is read from the database.

A database constructed with commitments guarantees the following properties.

- **Hiding Property**: values stored in the database are hidden from V.
- **Binding Property**: after a value is written into the database at a certain position, P cannot read a different value. ZK proofs for reading and writing values ensure that those values remain hidden from V.
Motivation: Modularity

In commit-and-prove protocols, the task of maintaining a database between P and V and reading and writing values into it is not separated from the task of proving statements about the values read or written. I.e., typically, P computes a ZK proof to prove a statement about a committed value, which involves both reading a value from the database and proving a statement about it.

To improve modularity, we propose to separate the task of maintaining a database between P and V from the task of proving statements about the values read or written (or about the positions where the values are stored). This has the following advantages:

• Simpler and more structured security proofs.
• Study the task of maintaining a database between P and V in isolation, which allows an easy comparison of different techniques to maintain a database.
Motivation: Database Positions

If Pedersen-like commitments alone are used to construct a database, it is not possible to hide from $V$ the database positions where data is read or written. However, this is necessary in some protocols.

For example, in [Herrmann et al., WiSec 14], a protocol for a location-based service between a user and a service provider is presented where the database consists of pairs

$$[\text{position}, \text{value}] = [\text{location}, \text{counter}]$$

When a user visits a location, the counter for that location needs to be incremented. User privacy requires that the location remains hidden from the service provider. Therefore, in this protocol it is necessary to both:

- Read, write and prove statements about the counter (the value stored)
- Read, write and prove statements about the location (the database position where the value is read or written.)

We would like to construct a database in which hiding the database position and proving statements about can be done, and with cost independent of the database size.
Contribution

- UC functionality $F_{CD}$ for an oblivious and updatable committed database.
- Modular design of protocols using $F_{CD}$.
- Construction $\Pi_{CD}$ for $F_{CD}$. 
Functionality $F_{CD}$

- We consider a simple database $DB$ with entries of the form

\[
[position, value] = [i, v]
\]

We want a functionality $F_{CD}$ in which
- $F_{CD}$ interacts with a prover $P$ and a verifier $V$.
- $F_{CD}$ allows $P$ to perform two operations.
  - **Read**: $P$ reads an entry $[i, v]$ from the database.
  - **Write**: $P$ writes an entry $[i, v]$ into the database.

Both $i$ and $v$ must remain hidden from $V$.

- For modularity, the tasks of proving statements about the position $i$ or the value $v$ must be done by other functionalities $F^{R}_{ZK}$ parameterized by the appropriate relations $R$.
- In a protocol that uses $F_{CD}$ along with $F^{R}_{ZK}$, we need to ensure that the position $i$ and the value $v$ read or written by $P$ are equal to $i$ and $v$ sent to $F^{R}_{ZK}$ by $P$.
- We used the method in [Camenisch et al., CRYPTO 2016] to ensure that the prover sends the same $i$ and $v$ to $F_{CD}$ and to $F^{R}_{ZK}$.
- This method consists in sending committed inputs to the functionalities, where the commitments are computed by a functionality $F^{NIC}$ for non-interactive commitments.
$F_{CD}$: Write Operation

Input: $(\text{write}, \text{com}_i, i, \text{open}_i, \text{com}_w, v, \text{open}_w)$

$F_{CD}$

- Verify commitments
- $cp \leftarrow cp + 1$
- Store $(qid, \text{com}_i, \text{com}_w, i, v, cp)$

Output: $(\text{write}, \text{qid})$

• Check if stored $(qid, \text{com}_i, \text{com}_w, i, v, cp)$
• Check if $cp = cv + 1$
• Store $[i, v]$ in DB
• $cv \leftarrow cv + 1$

Output: $(\text{write}, \text{com}_i, \text{com}_w)$

• $F_{CD}$ guarantees that the position $i$ and the value $v$ committed to in $\text{com}_i$ and $\text{com}_w$ are written into DB.
$F_{CD}$: Read Operation

**Input:** \((\text{read}, \text{com}_i, i, \text{open}_i, \text{com}_r, v, \text{open}_r)\)

- Verify commitments
- Check if \([i, v] \in DB\)
- Store \((qid, \text{com}_i, \text{com}_r, cp)\)

**Output:** \((\text{read}, \text{com}_i, \text{com}_r)\)

$F_{CD}$ guarantees that the position \(i\) and the value \(v\) committed to in $com_i$ and $com_r$ are stored in DB.
Let's consider a protocol that uses $F_{CD}$ and the functionalities $F_{ZK}^{R_i}, F_{ZK}^{R_v}$. To write an entry into DB the prover $P$ and the verifier $V$ proceed as follows.

- **P** and **V** run setup operations for $F_{CD}$ and $F_{NIC}$. (Steps 1, 2 and 3)
- **P** obtains commitments to a position $i$ and a value $v$ from $F_{NIC}$. (Steps 4 and 5)
- **P** sends those commitments to $F_{CD}$ to write $[i, v]$ into DB. (Step 6)
- **V** validates with $F_{NIC}$ the commitments received from $F_{CD}$. (Steps 7 and 8)
Modular Design with $F_{CD}$: Read Operation

To read an entry from DB and prove statements about it, P and V proceed as follows.

- P obtains commitments to a position $i$ and a value $v$ from $F_{NIC}$. (New commitments are required if it is necessary to hide if the position read is the same as the one previously written.) (Steps 9 and 10)
- P sends those commitments to $F_{CD}$ to read $[i, v]$ from DB. (Step 11)
- V validates with $F_{NIC}$ the commitments received from $F_{CD}$. (Steps 12 and 13)
- P uses $F_{ZK}^{R_i}$, $F_{ZK}^{R_v}$ to prove statements about $i$ and $v$. (Steps 14 and 15)
Construction $\Pi_{CD}$ for $F_{CD}$

$\Pi_{CD}$ is based on vector commitments (VC), which allow committing to a vector $x$ of values.

- **Setup**: An initial DB with entries $[i, v]$ is mapped to a vector $x$ by setting $x[i] = v$ for all entries. P and V compute a vector commitment $vc$ to that vector.
- **Read operation**: To read an entry $[i, v]$, P computes an opening $w$ for position $i$ and proves in ZK that $vc$ commits to $v$ at position $i$.
- **Write operation**: To write an entry $[i, v]$, P updates $vc$ to $vc'$, such that $vc'$ commits to the same vector as $vc$ except that now $v$ is committed at position $i$. P proves in ZK that $vc'$ is an update of $vc$.

VCs have the following efficiency properties:

- The size of $vc$ and of an opening $w$ are independent of the vector size $|x|$.
- The computation cost of updating $vc$ or and opening $w$ is independent of $|x|$.
- The computation cost of $vc$ or and of $w$ grow linearly with $|x|$.
Efficiency of $\Pi_{CD}$

- **Communication cost**: the size of $vc$ and $w$ are independent of the database size $|DB|$, and the size of ZK proofs for read and write operations is also independent of $|DB|$. Therefore, the communication cost is independent of $|DB|$.
- **Computation cost**: $vc$ is computed at setup and later it is only updated.
  - **Worst case**: P needs to read or write all the database positions throughout the protocol execution. The cost of computing the openings $w$ grows quadratically with $|DB|$.
  - **Best case**: The database $|DB|$ is initialized to a vector of 0 and few positions need to be read or written. The computation cost of $vc$ is constant and the computation cost of each $w$ grows linearly with the number of non-zero components in $vc$.

We describe privacy-preserving protocols that use $\Pi_{CD}$ for e-commerce, billing and location-based services in which the best case occurs. Therefore, those protocols handle large databases very efficiently.