DATABASE AND INFORMATION SYSTEMS

BLOCKCHAIN DATABASE

DETECTING INSIDER ATTACKS ON DATABASES USING BLOCKCHAIN

Introduction (Adapted from here)

Applications that rely on centralized databases are often vulnerable to insider attacks. Any user with administrative privileges on the Database system, is capable of modifying (inserting, updating or even deleting) the database entries. We try to counter this problem using the tamper resistance property of Blockchain. We have implemented and tested our solution using Multichain on an academic grading application.

WORKING OF OUR APPLICATION

The main database of student grades has five fields namely the **userID, course name, grade, transactionID** (its record in the Blockchain) **and identifier** (the instructor who issued it).

The Multichain network provides stream functionality, which allows for data insertion and retrieval on the blockchain network. Streams provide a natural abstraction for blockchain use cases which focus on general data retrieval, timestamping and archiving, rather than the transfer of assets between participants. Any number of streams can be created in a MultiChain blockchain, and each stream acts as an independent append-only collection of items.

Streams can be used to implement three different types of databases on a chain:

- A key-value database or document store
- A time series database, which focuses on the ordering of entries
- An identity-driven database where the entries are classified according to their author.

We create three streams for our use - one for storing the database state, another for storing the public keys and identifier (instructorID) pairings and the third one for storing the instructor-course pairings.

- **SIGNUP and LOGIN**

We have implemented our application using Python Tkinter based GUI. Instructors can sign up and then log in to the application. When they sign up, an **RSA key pair** is generated. The **Private Key** is encrypted using the *passphrase* that the instructor enters and is stored on the person's system. The **Public Key** is sent to the server where it is broadcast as a transaction on a *public key* stream.

Needless to say, the instructor's Password is *salted* and stored on the database using **PBKDF2** password hashing using SHA256.

- **INSERT QUERY**

An instructor can enter grades of his students in the courses that he teaches. The courses that a particular instructor teaches is not stored on the database but present as a transaction on an *instructor* stream.

An instructor enters a space-separated string of a student's userID, course and grade with each such string separated by a newline. On submitting a bunch of grades, each line of **grade is digitally signed** using the instructor's Private Key and sent to the server. The server retrieves the public key of the instructor from the *public key* stream and verifies the signature.

For each grade about to be entered, the database is retrieved* (ordered by txID) and each tuple (*txID+uid+course+grade+identifier*)$^{\#}$ is concatenated and a hash is obtained. To this hash the present grade is concatenated (*uid+course+grade+identifier*) and again hashed. This final hash is then published to the stream which gives us a transaction for that grade.

Note that, the actual implementation is actually optimized to query the database only once for each group of inserts.

** : It is constructed as an array of strings where each string is #.*

- **UPDATE QUERY**

The instructors also have an option to update a student's grade. Unlike Inserts, updates happen one at a time. When an update is issued, the whole database is retrieved and the old grade tuple is removed from it. It is now hashed, the new data is concatenated and again hashed (similar to the insert query above) and this is sent as a transaction to the stream. The old grade is then updated with the new grade and the new transactionID.

DETECTION OF AN INSIDER ATTACK

Before every query (SELECT, INSERT, UPDATE) issued to the database, a check is performed to see if the database is consistent with what is present in the Multichain stream.

The stream is queried to obtain the latest transaction. The transactionID and the corresponding data (which is a hash) is retrieved. The whole database is now queried and all the grades are concatenated[#] except the one corresponding to the latest transactionID. A hash is calculated, then concatenated with this leftover grade *(uid+course+grade+identifier)* and a hash is calculated over the resultant string. If the obtained hash is equal to the one stored in the stream corresponding to the latest transaction, then the database is consistent otherwise it has been accessed illegally, indicating insider access.

Once a grade has been inserted into the database, our protocol can detect illegal modification and deletion of any of the grades from the database.

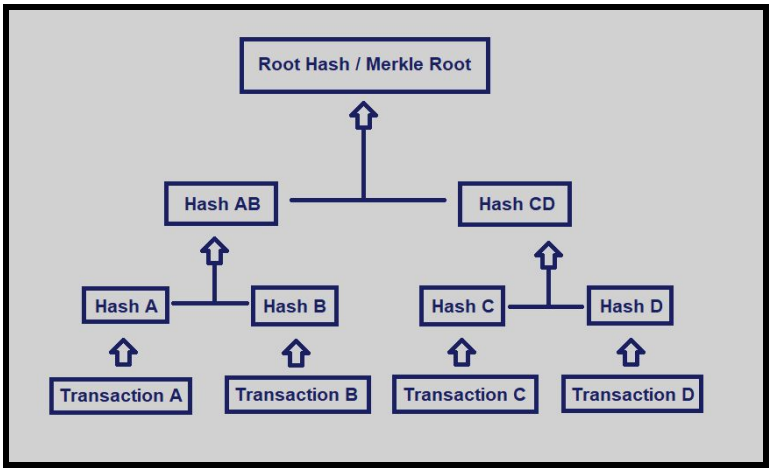
SOME NOTABLE FEATURES

We have tried our best to implement as many security checks as possible as well as a user friendly interface. Some of them are :

1. A status bar on the top of the window to tell one about the ongoing process.
2. Whenever a batch of grade inserts fail, the user is notified as to which grades failed to insert (either due to primary key violation or incorrect format of data)
3. An instructor can modify (insert or update) grades of only the courses that he teaches (of course!) and not of others
4. A student is only allowed to see his own grade. He does not have the right to perform any other actions.

POTENTIAL FEATURES

1. ***Separate streams for each course***. As each stream acts as an independent append-only collection of items, this is a fairly natural abstraction.
2. ***A separate stream for database statements instead of hashes***. This allows for quick recovery in case the transaction was pushed to the stream but there was an error in pushing it to the database
3. ***Using Merkle Trees***. Merkle trees have widespread usage in BlockChains. It is a structure that allows for efficient and secure verification of content in a large data body. They are created by repeatedly hasing pairs of nodes until there is only one hash left, called the Root of the Merkle.



The Merkle Root *summarizes all of the data in the related transactions*, and is stored in the block header. It maintains the integrity of the data. If a single detail in any of the transactions or the order of the transactions changes, so does the Merkle Root. Using a Merkle tree allows for a quick and simple test of whether a specific transaction is included in the set or not.

SOFTWARES USED

Python, MySQL, PHP, Multichain

REFERENCES

[1] Shubham Sharma, Rahul Gupta, Shubham Sahai Srivastava and Sandeep K. Shukla, Detecting Insider Attacks on Databases using Blockchains

[2] Dr Gideon Greenspan, Founder and CEO, Coin Sciences Ltd, MultiChain Private Blockchain - White Paper, url : https://www.multichain.com/

[3] Image on Merkle Tree taken from https://hackernoon.com/merkle-trees-181cb4bc30b4