**Publicly Verifiable Distributed ORAM with application in Metadata**

**Metadata Storage Structure**

The main goal of the project is to store telecommunication metadata in the public cloud securely and give permission to legitimate users to make restricted query from stored data such as neighbourhood query and next hop query.

In order to achieve the secure storage not only the privacy of the data should be considered but also the access pattern is very important security disclosure. Therefore, we consider a storage model that can address both problems. Each telephone number and its neighbours are assumed as a block of data and with leverage of secret sharing, we provide privacy. We also store opening of the commitments of the data along with shares. The commitments are also kept in public cloud and for each block of data if we say #A is the telephone number and neighbours are B,C and D the commitment is g^r\* h^A\*h\_1^B\*h\_2^C\*h\_3^D. in order to avoid access pattern disclosure we use path ORAM on shares and commitments.

**Commitments Structure**

The main reason of using commitments is to prove that the telecommunication company cannot modify or change the data after storing it. But part of the commitments is changing after each query, because of the ORAM structure. In Path ORAM structure, after each accesses to the storage all buckets along the read path decrypt and re \_encrypt when pushing to the new path.

In order to overcome this problem, we don’t modify the commitments along the read path and just re-randomize them and write them in the same path. In this way, we can use proof of the shuffle to show that Telco haven’t changed anything after committing. But this scheme cannot avoid cheating before committing, it means that Telco can easily generate two commitments on each block of the data (telephone number and its neighbours) and store it in the ORAM structure and use proof of the shuffle that show nothing has changed.

**Merkle Tree Structure**

We use Merkle tree to avoid publishing all the commitments in public and checking them after each query to ensure that apart the set of the commitments that we re-randomized them nothing has changed on commitments.

Telco store the ORAM tree of the commitments as a Merkle tree as well. It means the buckets in the ORAM tree are contained commitments and hash of the children commitment with commitment of the bucket. After each query, a new root hash of the Merkle tree is published and sent to the legitimate users then she can check that the Merkle tree before query and after query are equivalent and hasn’t changed.

If we assume that C :={c1,c2,c3,…,cn} is the set of the commitments in the read path of the ORAM tree and C’:={ c’1,c’2,c’3,…,cn} is the set of the commitments after re\_ randomizations, h1 and h2 are the hash of the untouched parts in the Merkle tree and H is the root hash before query and H’ is root hash after query then it is enough to show that

1. The commitment of queried block is in the C.
2. C and h1 and h2 results H
3. C’ and h1 and h2 results H’

In order to achieve proof of the shuffle we need to publish C and C’ after each re-randomization of the ORAM path. Therefore the protocol for covering the three mentioned steps is as following:

1. Publishing C, C’ and permutation commitment (depends to the proof of the shuffle we choose).
2. Sending C, h1 and h2 to the legitimate users in order to verify and recompute the H.
3. In order to hide the level of the queried block, we need to do re-randomization and proof of the shuffle again. Therefore we need to publish C’ and C” which are two sets after re-randomization.
4. Sending C”, h1 and h2 to the legitimate user in order to recompute and verify the H’.

**Stash Structure**

In ORAM structure, stash stores the blocks which are overflowed from the buckets. In path ORAM structure, this would happen when the queried block is either elevated or descended through the old path in order to set in the new path. Therefore, in this repositioning of the queried block the stash could store the block which cannot find empty bucket through the old path in desired level of ORAM tree.

In this scheme, the stash could have some queried buckets of the commitments from C”. So the stash should be published in public as a part of C” in order to perform the proof of shuffle hence the commitments in the stash re-randomize along with commitments in the accessed path in path ORAM structure. To show the integrity of the commitments in the Merkle tree, the commitments in the stash should be included in C, C” and the root hash which is sent to the legitimate user.

**Directedness of the graph**

Start by assuming that all edges are directed and that g^r\* h^A\*h\_1^B\*h\_2^C\*h\_3^D represents A calling B, C and D. Then we can ask the second-hop query of who was called by someone whom A called.

Other kinds of second-hop query are harder in this data structure, for example who called someone whom A also called. We considered some options for including the list of people who called A.

Option 1: uncommitted crib notes. Note that this does not need to be committed to, assuming that the final proof is based on the main data structure. It does need to be remembered by the data owner. One possibility is to secret-share this information – when A calls B, that fact is recorded in the main data structure record representing A’s calls, and also included in a “crib notes” form that Telstra uses to answer reverse connection queries. The “crib notes” could be secret-shared directly among the cloud authorities. When the Telco wants to know who called #A, it retrieves the secret-shared data and then the relevant commitments for each of the calling parties.

Question: how do we stop it from returning only a subset of the true answer?

Option 2: Committed list of incoming calls. This gets rid of the question of leaving out some subset – just use the same mixing proof as for outgoing calls.

Questions: how do we ensure consistency with the outgoing call commitments? Do we need to?

**Second –hop Query**

The graph we are working on is a directed graph as the Telco just keeps the metadata which is related to the outgoing calls for billing purposes. Therefore, if we assume the neighbours of #A are X and Y and neighbours of X are X1 and X2 and neighbours of Y are Y1 and Y2 then related commitments for second-hop query are as following:

CA=g^ra\*h^A\*h1^X\*h2\*^Y

CX=g^rx\*h^X\*h1^X1\*h2^X2

CY=g^ry\*h^Y\*h1^Y1\*h2^Y2

Telco sends the X1,X2,Y1 and Y2 as the second-hop neighbours of the A to the querier and needs to prove that the answer of the query is correct and untouched.

In order to prove the correctness and integrity of the answer, Telco needs to show: step 1-CA, CX and CY are in the Merkle tree

Step 2-CA, CX and CY are related by using known elements of X1, X2, Y1 and Y2 without revealing X and Y.

To show the CA, CX and CY is in the Merkle tree, Telco should follow the Protocol which described for neighbourhood query, in section one. To show the second part, Telco should use Zero-knowledge proof to show CA, CX and CY are related and are correct commitments of A, X and Y without reviling anything about neighbours of A which are X and Y. Therefore, Telco needs to prove that it knows the X,Y, ra, rx and rY for recomputing the CA, CX and CY.

In this scheme we use Zero-knowledge proof scheme to show that Telco knows the entire secret to recompute the commitments. If we assume X’ ,Y’, r’a, r’X and r’Y are random values which are generated randomly by Telco and *e* is a challenge generated by querier then following steps are needed to show the step 2:

1. Telco sends C’A= g^r’a \*h1^X’\*h2\*^Y’

C’X =g^r’x\*h^X’

C’Y = g^r’y\*h^Y’

1. Querier sends challenge *e* to the Telco
2. Telco sends fX=X’+eX , fY=Y’+eY , fra=r’a+e ra ,frx=r’x+e rx , frY=r’Y+e r’Y to the querier.
3. For proof of the step 2 , querier needs to show

g^fra\* h1^fX\* h2^fY == C’A\*(CA/h^A)e

g^frx\*h^fX== C’X\*(CX/h1^X1\*h2^X2)e

g^frY\*h^fY==C’Y\*(CY/h1^Y1\*h2^Y2)e