Appendix to "Log-Structured Merge Design of Authenticated Data Structures for Efficiently Verifiable Cloud Outsourcing"

A. CONSISTENCY CHECKING

```
1  class store_wrapper{
2    Store store;
3    Att cPut(key,val){
4      prePut(key,val);
5      att(tsw)=store.dPut(key,val);
6      return postPut(key,val,att(tsw));
7    }
8    Crt cGet(key){
9      preGet(<key>);
10     <key,val>,pf(tsrw,tsr*)=store.dGet(key);
11     return postGet(<key>,pf(tsrw,tsr*));
12   }
13
14   mutex State pending_wr, completed_wr,history_w;
15
16   void prePut(<key,val>){
17     pending_wr.add(<key,val,start_rt=now()>);
18   }
19   boolean postPut(<key,val>,att(tsw)){
20     <key,val,start_rt>=pending_wr.remove();
21     completed_wr.addW(<key,val,start_rt,end_rt=now(),tsw>)
       ;
22     ac1 = completed_wr.tryTruncate();
23     if(ac1 != NULL){
24        assertC(ac1,history_w);
25        for(Write w in ac1.trim())
26            history_w.put(w);
27     }
28   }
29   void preGet(key){
30     pending_wr.add(<key,start_rt=now()>);
31   }
32   boolean postGet(r<key,val,tsrw,tsr,pf(tsrw,tsr*)>){
33     r<key,start_rt>=pending_wr.remove();
34     if(r.tsr <= history_w.latest());
35       assertL2(r<key,val,tsr,tsrw,pf(tsrw,tsr*)>,history_w
       );
36     else
37       completed_wr.addR(<key,val,start_rt,end_rt=now(),tsr
       ,tsrw>);
38   }
39 }
```

Listing 2: Interfaces of verified and verifiable Put/Get

```
1  void assertC(ac1,history_w){
2    //L1
3    o0<key,val,ts> = history_w.latest();
4    o1<key,val,ts> = ac1.oldest();
5    do {
6      assertL1pairwise(o0<key,val,ts>,o1<key,val,ts>);
7      o0=o1; o1=o1.next(ac1);
8    } while (o1 != NULL)
9    //L2
10   for (read r in ac1)
11     assertL2(r<key,val,tsr,tsrw,pf(tsrw,tsr*)>,history_w);
12 }
13 void assertL2(r<key,val,tsr,tsrw,pf(tsrw,tsr*)>,history_w)
       {
14   assert(r.tsr <= history_w.latest());
15   assert(verify(pf(tsrw,tsr*))==true);
16   assert(tsr*==tsr);
17 }
```

Listing 3: Linearizability checking

B. FORMAL FRONTEND SECURITY

Intuitively, if a record is fresh in a set of records, the record must be fresh (or absent) in any subset of the set. The correctness comes from the intuition that "a record is the freshest in a dataset iff. it is the freshest or absent in the subsets whose union fully covers the dataset." And the unforgeability comes from the collision-resistance of the hashes and Merkle trees. *F1* can be proven with unforgeability by the authentication path of MHT digesting C_1 and the key ordering among records in C_1 implied Invariant 3.1. *F2* is proven with unforgeability by the authentication path at C_0 and Invariant 3.1. *F3* is implied by invariant 3.3. Generalizing this case gives us a formal proof described below:

PROOF. Consider $\langle k, v, t\rangle$ resides in level C_i of an LSM of n levels, C_j where $j \in [1, n]$.

Record $\langle k, v, t\rangle$ is the freshest (with the largest timestamp value) among all records of key k in the entire LSM tree, iff the following four facts hold:

F1 No record of key k exists in levels $C_0, C_2, \ldots, C_{i1}$.

F2 Record $\langle k, v, t\rangle @ C_i$ is the freshest among all records of key k in level C_i.

F3 Records of key k in levels $C_{i+1}, \ldots, C_n$, if they exist, are older than record $\langle k, v, t\rangle$.

F4 $C_0, C_1, \ldots, C_n$ covers the entire dataset in the LSM tree.

The four facts can be proved by the following proofs, respectively:

F1 is about the non-membership of records $\langle k, \cdot\rangle$ on levels $C_0, \ldots, C_n$. F1 can be verified by the authentication paths of key k of MHTs on these levels.

F2 can be similarly proved by the authentication path of key k of the MHT on level C_i.

F3 can be derived from the inter-level time-ordering (Invariant 3.3), as any record of key k on levels $C_{>i}$ must have their timestamp smaller (older) than any record on level C_i.

F4 can be proved by that the enclave maintains the complete set of per-level Merkle hashes on all levels. □