3/8 Integrity

Lecture

Consider an environment where we have multiple users reading and writing to an untrusted server. In today's lecture we explore some ideas on how can we build some integrity guarantees into this model.

Append-only Log

We are going to be working with an insert-only multiset with the following operations:

- `is x in S?`
- `T := add x to S`
- `is S a subset of T?`

How can we support this kind of data structure? We will go through a variety of approaches and evaluate them on based on their runtime and space complexity attestation given by the server.

Solution 1: Hash-chaining

We could use a hash chain, which is a data structure similar to a linked list, where the data in each node contains a hash of the previous node.

Operation	Runtime	Space
`is x in S?`	O(n)	O(n)
`T := add x to S`	O(1)	N/A
`is S a subset of T?`	O(n)	O(n)

Solution 2: Merkle Tree, in order they were inserted

Operation	Runtime	Space
`is x in S?`	O(log n) if the item is in the tree, O(n) otherwise	O(log n) if the item is in the tree, O(n) otherwise
`T := add x to S`	O(log n)	N/A
`is S a subset of T?`	O(log n) if S is a subset of T, O(n) otherwise	O(log n) if S is a subset of T, O(n) otherwise

Solution 3: Merkle Tree, in sorted order

Operation	Runtime	Space
`is x in S?`	O(log n)	O(log n)
`T := add x to S`	O(log n)	N/A
`is S a subset of T?`	O(n)	O(n)

Discussion

What if we used solution 2 and 3 simultaneously? This creates a consistency requirement. The server has to prove to us that the two data structures have the same data.

If there is one client, every time we insert, we can check that they are in sync. If there are multiple clients, however, we are not aware of all inserts; we use the subset operation to ensure that our prior view of the system is a subset of the current system state. If we use both solutions, solution 2 has an efficient subset operation while solution 3 doesn't so we still have a problem.

We could also go the Certificate Transparency (CT) route, where external auditors check for consistency rather than the client. Moreover, CT uses an auxilliary data structure called a sparse merkle tree. Spare merkle trees provide a convenient way to map from 2^256 items to single bits.

- Proof of membership is very efficient for both if an element is or isn't contained in the set.
- Easy to do an insert in log(n) time.
- Subset is really slow.

This approach has similar security properties to using solutions 2 and 3.

Equivocation

Now what if the server provides a different view of the log to different users? This presents a problem since the server can be honest w.r.t to the auditors but mess with the clients. CT uses gossip between browsers, where the clients effectively issue a subset query between two clients views of the log.

What if clients lie about server hashes? Well we just make sure that the server signs its hashes.

Some issues with this approach:

- How do clients find each other?
- Certificates are the root of trust
- What if an oppresive regime wants to attack a handful of users? If those people don't check with anyone else, then they will never resolve the fork. This is tough with firewalls. CT wanted to set up trusted web servers that does the check but this has privacy issues.

Discussion: Secure History Preservation Through Time Entanglement

Secure history preservation is useful technique in attesting state in a tamper evident manner. Some applications of secure history preservation include:

- Secure time stamping
- Cryptographic, end-to-end voting systems. Voters can check that the encrypted version of the ballot is uploaded and never removed.

Definitions

- A **secure timeline** is a tamper-evident history record of the states through which a system goes throughout its operational history.

- A **service domain** is some system offering a service to some clients.
- An **authenticated logical clock** is a clock in which each time step is annotated with the state in which the service is in at the time, and an authenticator.

Scheme

- Let `h` be a one way hash function.
- At time `i`, the clock has authenticator `T_i`.
- At time `i+1`, `T_i_1 = h(i, T_i, S_i)`

The problem with this scheme is consistency between different timestamp services. Timeline entanglement allows a group of mutually distrustful service domains to collaborate towards maintaining a common, tamper-proof history of the collective timelines. This approach requires coordination between timestamp services and its construction can be found here.