ECE 443/518 – Computer Cyber Security
Lecture 14 Secure Collaborations

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

October 17, 2022

Outline

Reading Assignment

- This lecture: Secure Collaborations
- Next lecture: Consensus

Outline

Secure Collaborations

Coin Flipping

Collaborative Data Management

Collaborations

- Parties collaborate to achieve a common objective.
  - For purposes like doing business, gambling, decision making.
  - Involve both computation and communication.
- Lack of trust: what if someone cheats?
  - Leakage of sensitive data.
  - Manipulation toward unfair or incorrect results.
- Laws help to protect against such issues in our daily life.
  - Need enforcement, by some party that is trusted by everyone.
  - Only for deterrence.
- What about reliability issues like corrupted computation or communication?

Secure Collaborations

- Solve collaboration as a security problem.
- Threats: everyone will cheat whenever possible.
  - There is no trusted third party.
  - This also models failures in communication channels – either the sender sends bad messages or the receiver claims to receive bad messages.
- Policy: objective of collaboration as security properties.
  - E.g. authentication, integrity, and confidentiality.
  - For our lectures, we assume authentication is supported by digital signatures, and focus on different collaborations that may require different levels of integrity and confidentiality.
- Mechanism and protocol design to enforce policy.
  - Allow parties to participate if they behave well.
  - Reject parties whenever they cheat.

Outline

Secure Collaborations

Coin Flipping

Collaborative Data Management

Coin Flipping

- The most primitive (true) random number generator.
  - Widely used in dispute resolution.
- The coin flipping game between Alice and Bob.
  1. Alice calls the coin flip $C \in \{heads, tails\}$.
  2. Bob flips the coin and report the result R.
  3. Alice wins if $C == R$; otherwise Bob wins.
- Fair coin: 50/50 chance for heads/tails.
  - If we assume that Alice can observe Bob's coin flipping results to decide the chances of heads/tails, then Bob have to use a fair coin to avoid losing money in the long run.
- What if Alice and Bob need to play the game over phone?
  - No trusted third party.

Cheaters

- Bob cheats by knowing Alice's call C and reporting $R \neq C$.
  - Bob can further provide a video of R.
  - If you are thinking about live streaming, why you believe the streaming is live?
- If we modify the game to ask Bob to flip and report R first, then Alice may cheat by calling $C = R$.

Mechanism Design: Commitment Scheme

- Commitment scheme: allow one to publish a secret message that will be revealed at a later time.
  - Commitment: the message cannot be modified once published.
- Commitment scheme can be implemented via hash $h()$.
  1. Alice chooses a random number k and sends Bob $a = h(C||k)$.
  2. Bob sends R.
  3. Alice reveals C and k for Bob to verify $h(C||k) == a$.
  4. Alice wins if $C == R$; otherwise Bob wins.
- Bob cannot cheat as long as $h()$ is preimage resistant.
  - Otherwise Bob can recover C from a and report $R = C$.
- Alice cannot cheat as long as $h()$ is collision resistant.
  - Otherwise Alice can find k_1 and k_2 to satisfy $h(heads||k_1) == h(tails||k_2)$, and reveal k_1 or k_2 depending on Bob's R.

Dice Roller and Mental Poker

- How can Alice and Bob roll a dice over the phone?
- A real challenge: how can Alice and Bob play poker over the phone?

Outline

Secure Collaborations

Coin Flipping

Collaborative Data Management

Collaborative Data Management

- Data management
  - Data set: other's public keys, business transactions, etc.
  - Operations: CRUD (Create, Read, Update and Delete).
- Collaborative
  - Everyone is allowed to modify the data.
- Security property
  - Ignore confidentiality.
  - Integrity and nonrepudiation: integrity is violated if someone modifies data without following a protocol; optionally, nonrepudiation helps to identify who modifies the data.
  - Version control and auditing: track history of how the data set changes, and know who made the change via nonrepudiation.
- Integrity also detects data corruption.
  - E.g. due to faults in memory, hard drive, and networks.
  - It is possible to recover corrupted data, though the techniques are out of the scope of this course.

A Naive Protocol

- Append to the data set its own hash.
  - Sign the hash if nonrepudiation is required.
- Store all versions of the data set and the hash.
  - As well as the signatures for auditing.
- Issues
  - Performance: not efficient to hash a large data set whenever it is modified.
  - Storage: cannot afford to store all versions of a large data set.
  - Auditing: we need to prove that the two versions is indeed before and after a change – storing all versions does not help.

Integrity and Data Structure

- For performance concerns, it is perferable to only hash the modification but not the whole data set.
  - Need to understand how the data set manages data.
- A popular choice of data structure is a tree.
  - Unsorted to represent hierarchical data, e.g. files and directories.
  - Sorted to represent key-value associations, e.g. map/dictionary or database tables.
  - Other data structures can be treated as a tree, e.g. linked list as a tree without branches.
- We can hash the tree nodes instead of the whole tree.
  - Modification to a tree is limited to the path from the node being modified to the root – to hash all nodes along the path is efficient!
  - But how can the relations between the nodes be protected?

Hashes as 'Cryptographic' Pointers

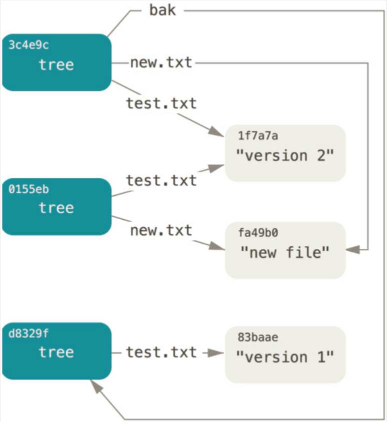
- Tree uses pointers to maintain relations between nodes.
  - Pointers cannot be reused in different programs, not to mention on different computers used for collaboration.
  - Anyone can modify a node and then the whole subtree, without being caught.
  - We need pointers that provide cryptographic guarantees.
- Merkle hash tree: hash of a node can work as its address.
  - Does not rely on a particular program or computer.
  - Practically, collision resistant implies that two nodes will have different hashes if they have different content.
- Nonrepudiation can be achieved by signing the hashes and store the signatures with the hashes.

Example: Git

(Pro Git)

- Git is a popular software for version control.
- Data set as a Merkle hash tree.
  - Two types of nodes: blob for files, tree for directories.
  - Each node is hashed with SHA-1 (only first 20 bits are shown).
- Integrity is guaranteed since modification of node content without changing it hash in the parent node will be detected.

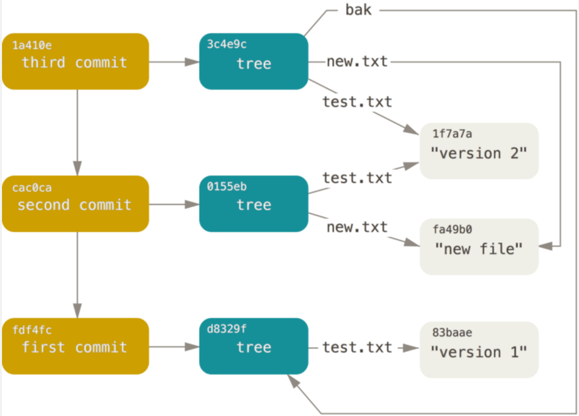
Modification and Storage



(Pro Git)

- Each modification results in a new root node.
- Replaced nodes are preserved for version control.
- There is no need to store any node more than once.
  - No matter how many times it appears in the history.

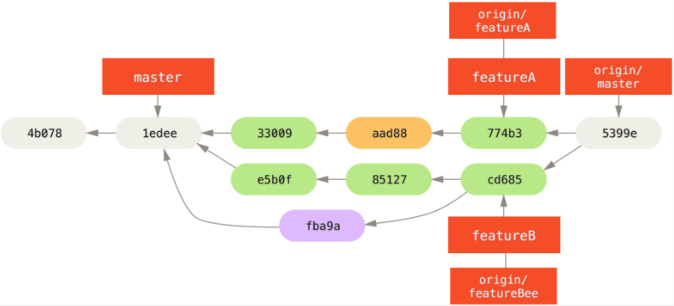
Integrity of History



(Pro Git)

- Use another Merkle hash tree consisting of 'commit' nodes to protect integrity of history.
  - The data structure is actually a directed acyclic graph (DAG), though the idea of using hashes to replace pointers is the same.
- What if multiple parties modify the data set at the same time?

Branches



(Pro Git)

- Git allows simultaneous modifications to happen on different branches of the tree consisting of commits.
  - With some efforts, branches can be merged to incorporate changes together.
- What if we need to apply similiar ideas to an application where branches are not allowed?
  - Then multiple parties need to agree on what 'main branch' to modify and who makes the modification.
  - That is another difficult secure collaboration problem.

Summary

- Seemingly impossible secure collaborations, like coin flipping, can be implemented via cryptographic constructions.
- Merkle hash tree provides an all-in-one solution for complex data management tasks with integrity guarantee.