# Efficient Data Structures for Tamper-Evident Logging

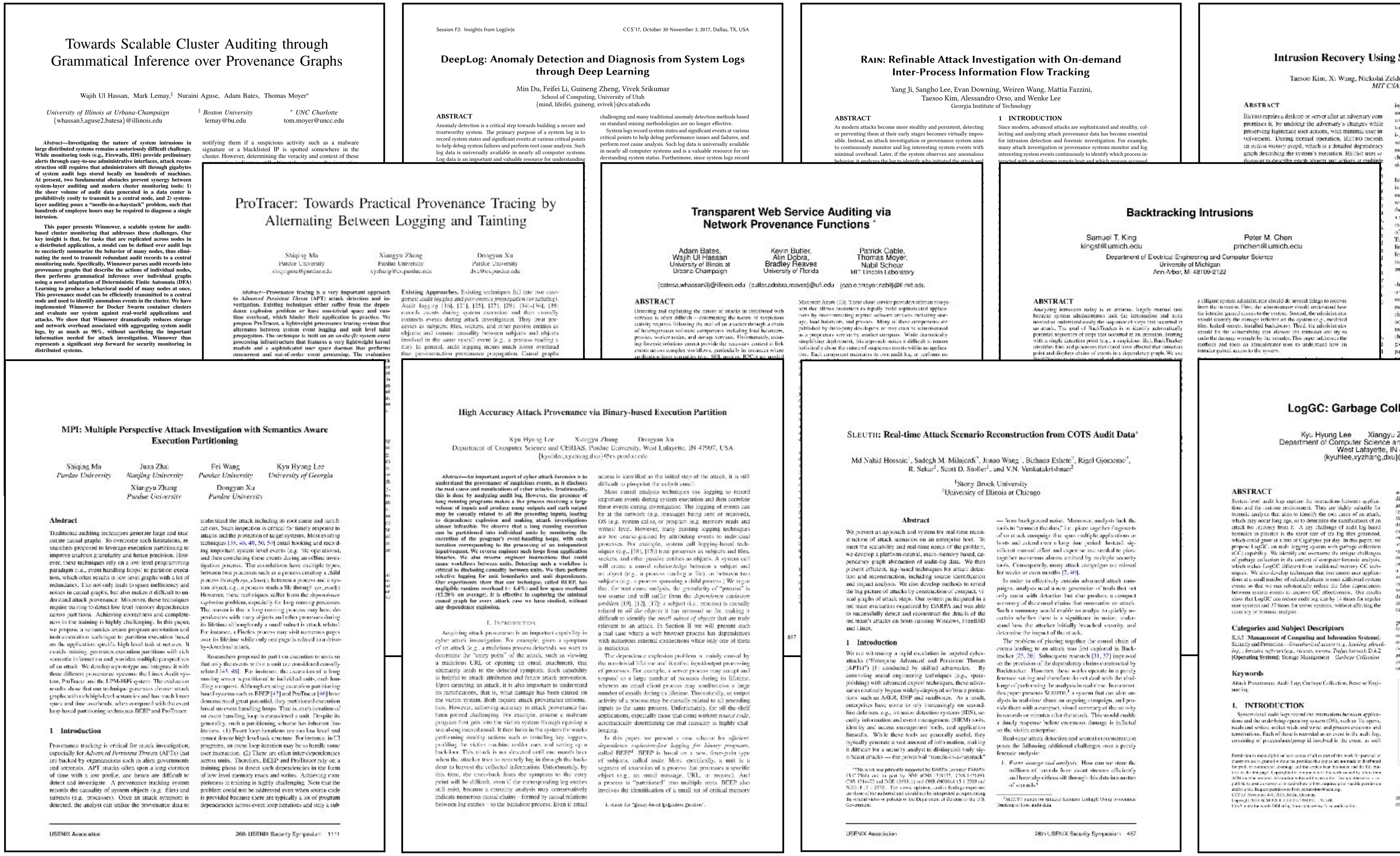
Scott A. Crosby        Dan S. Wallach

*Rice University*

Presented at USENIX Security 2009

# Reliance on logs

Assume the adversary doesn't tamper with the logs.



Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs

Wajih Ul Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, Thomas Moyer

DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning

Min Du, Feifei Li, Guineng Zheng, Vivek Srikumar

RAIN: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking

Yang Ji, Sangho Lee, Evan Downing, Weiren Wang, Mattia Fazzini, Taesoo Kim, Alessandro Orso, and Wenke Lee

Intrusion Recovery Using Selective Re-execution

Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek

ProTracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting

Shiqing Ma, Xiangyu Zhang, Dongyan Xu

Transparent Web Service Auditing via Network Provenance Functions

Adam Bates, Wajih Ul Hassan, Kevin Butler, Alin Dobra, Bradley Reaves, Patrick Cable, Thomas Moyer, Nabil Schear

Backtracking Intrusions

Samuel T. King, Peter M. Chen

MPI: Multiple Perspective Attack Investigation with Semantics Aware Execution Partitioning

Shiqing Ma, Juan Zhai, Fei Wang, Kyu Hyung Lee, Xiangyu Zhang, Dongyan Xu

High Accuracy Attack Provenance via Binary-based Execution Partition

Kyu Hyung Lee, Xiangyu Zhang, Dongyan Xu

SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data

Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R. Sekar, Scott D. Stoller, and V.N. Venkatakrishnan

LogGC: Garbage Collecting Audit Log

Kyu Hyung Lee, Xiangyu Zhang, Dongyan Xu

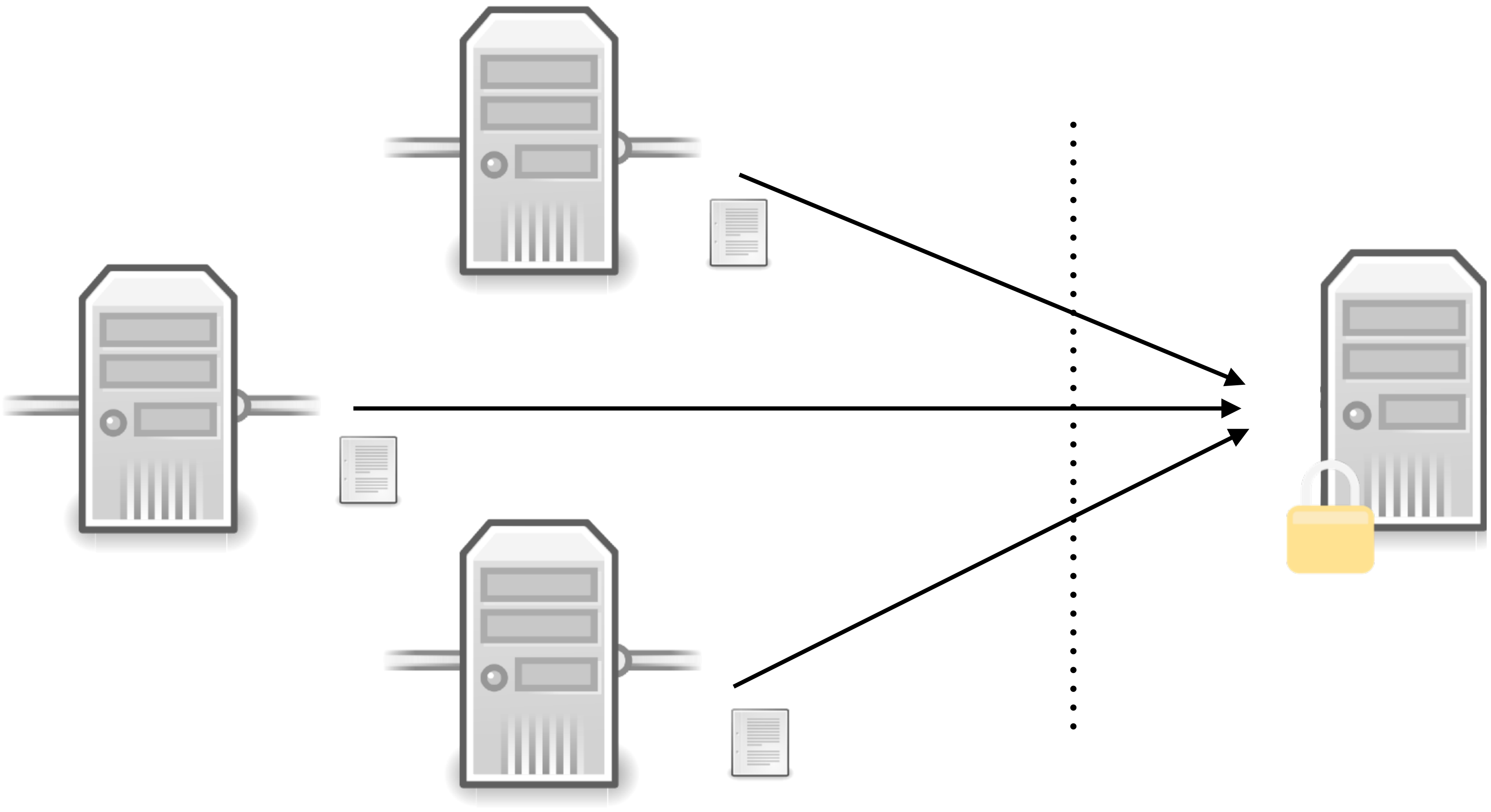
# Can we trust the logs?

The attacker may modify the log file to cover their traces!

**Goal**: An event, once correctly logged, cannot be *undetectably* hidden or modified.

# Industry practice

Send logs to a **trusted** central server



# This paper

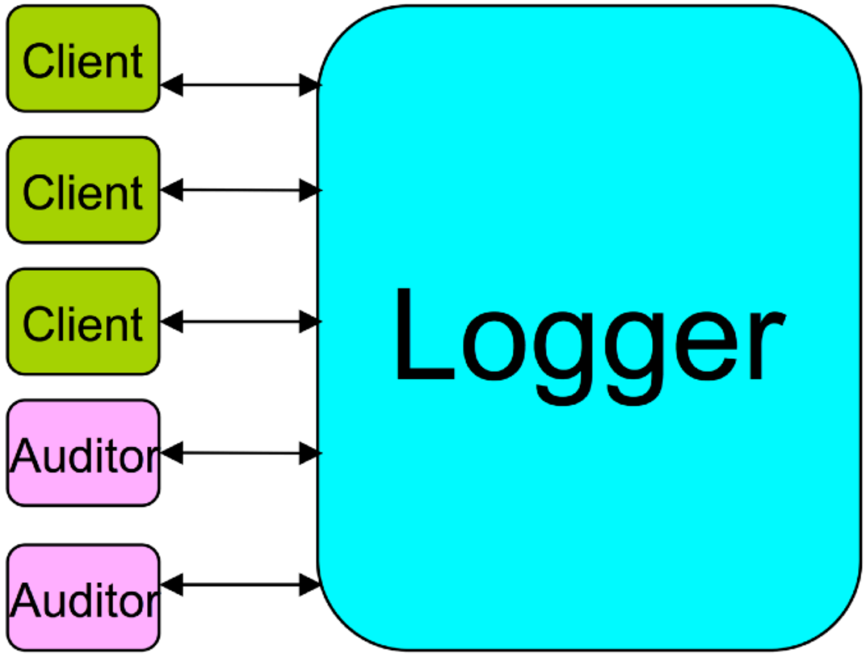
Allow the central server to be **untrusted.**

Ingredients:

1. Auditing
2. History Tree

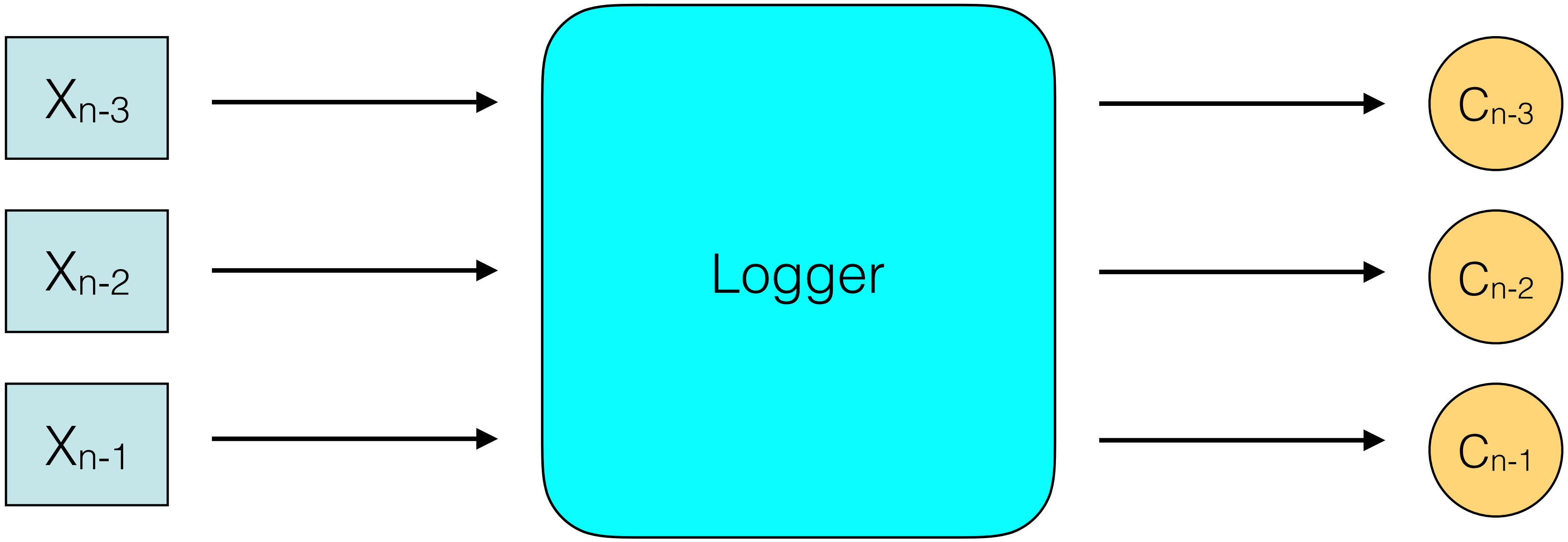
# High-level design

- Logger (central server)
  - Stores logs
- Clients
  - Generate logs
- Auditors
  - Verify the correct operation of the logger



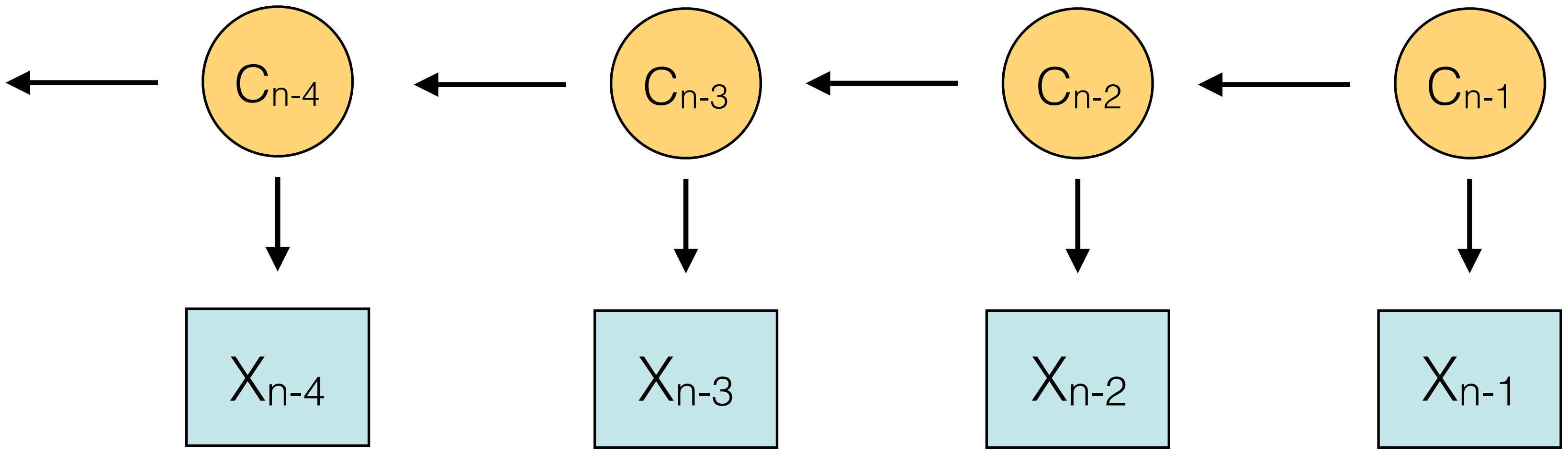
# Logger

- Logs come in
- Commitments go out



# Commitments

- Each commits to the entire past. Example construction [Kelsey, Schneier]:
    - $C_n = H(C_{n-1} \| X_n)$
- They are signed by the logger

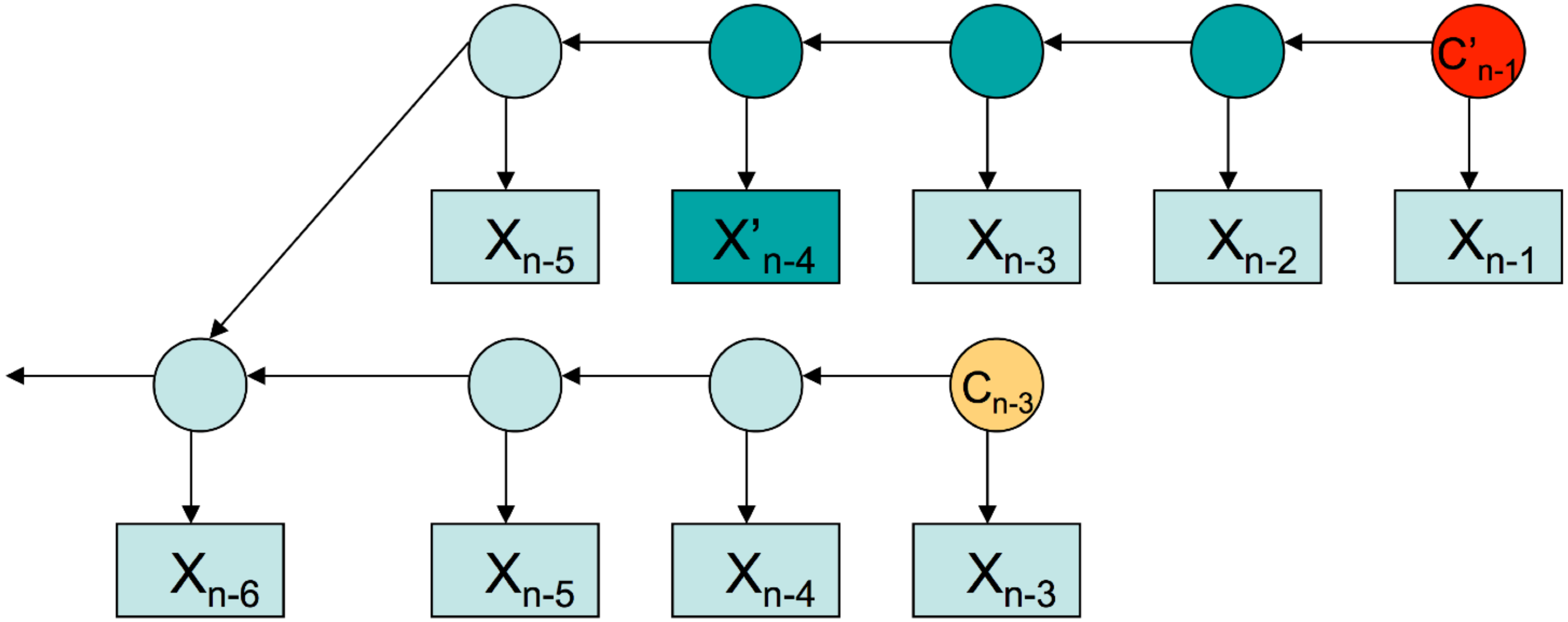


# We don’t trust the logger!

- Does $C_{n-3}$ really contain $X_{n-3}$ ?
- Do $C_{n-2}$ and $C_{n-1}$ commit the same historical events?
- Is the event at index *i* in the log defined by $C_n$ really $X_i$ ?

# Example: log forks

- What if the logger rolls back the log and adds on different events?

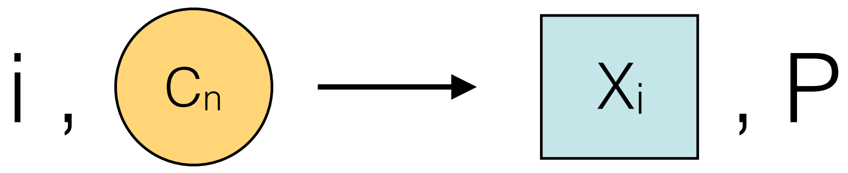


# Solution: Auditing

- Check the returned commitments
  - For correct event lookup
  - For consistency

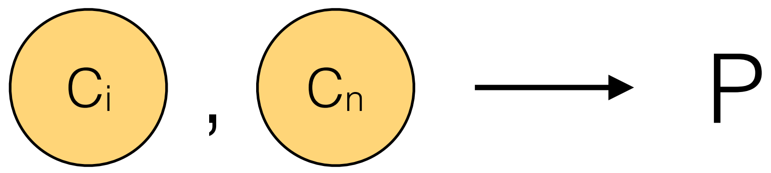
# Two kinds of audits

- Membership auditing



  - Verify proper insertion
  - Lookup historical events

- Incremental auditing



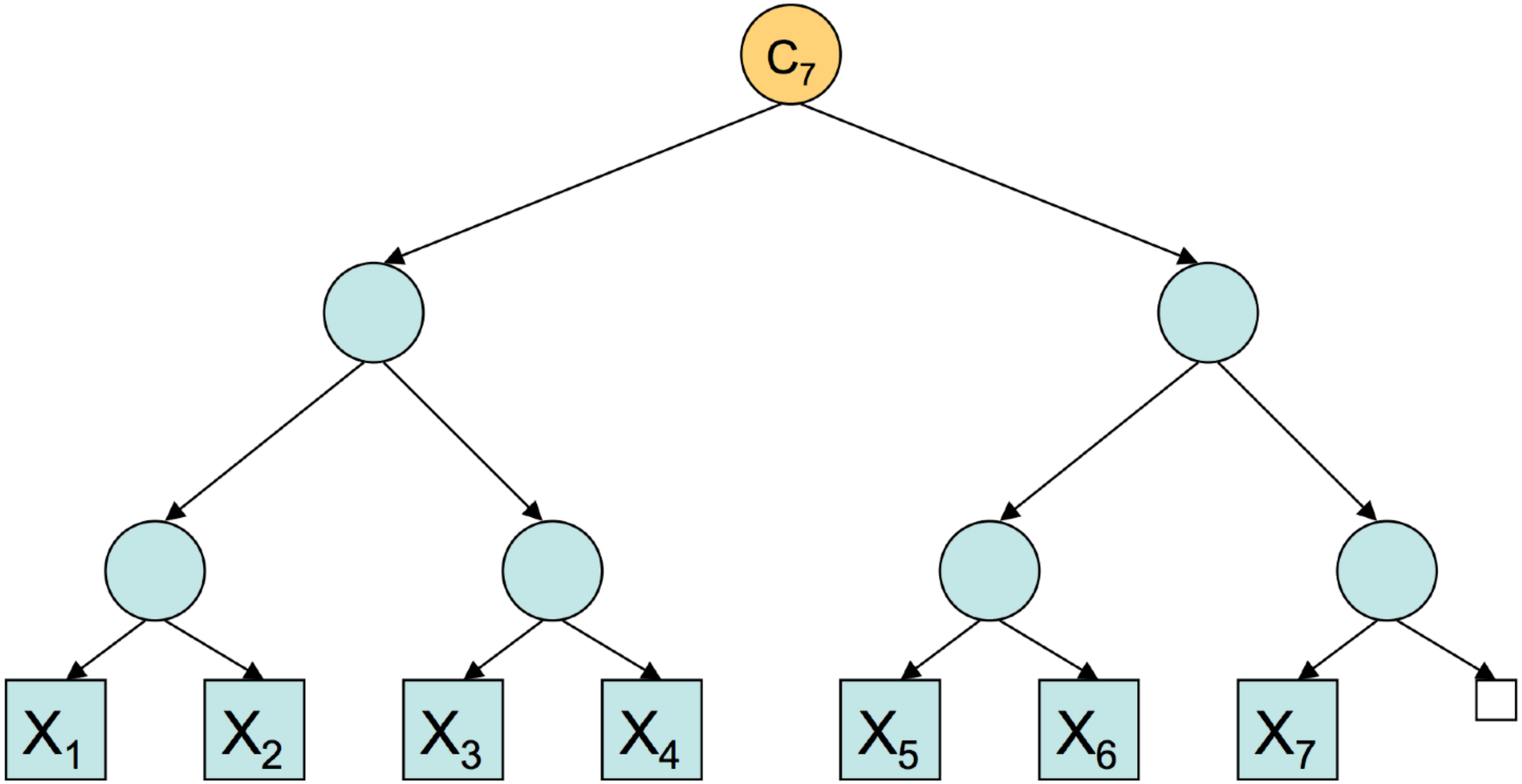
  - Prove consistency between two commitments

# Who does what?

- **Clients must redistribute their received commitments from the logger to auditors.**
- A host can be both client and auditor at the same time.
- Auditing strategies are not discussed in detail.

# Making audits cheap

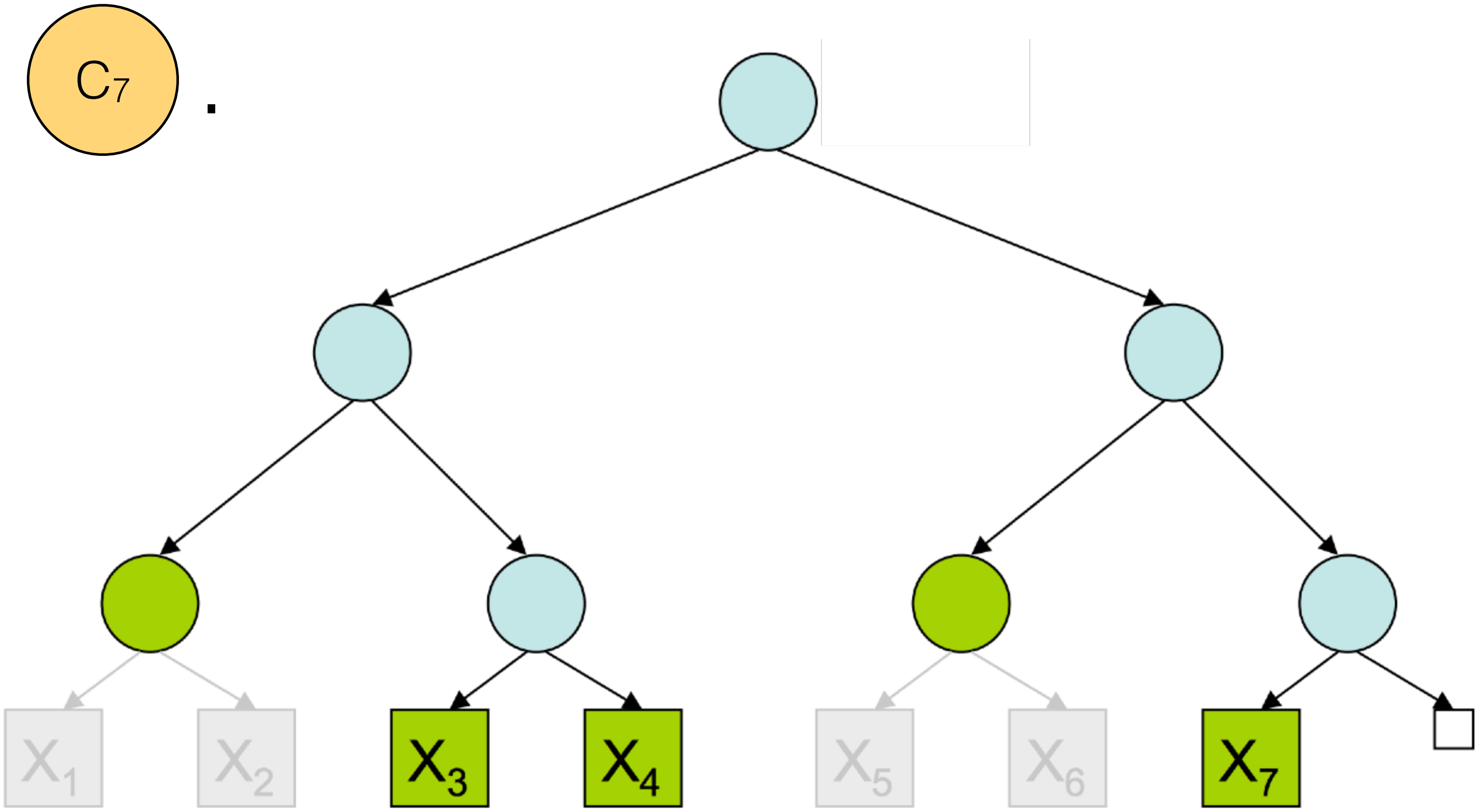
- Logs are stored in a **history tree**



# Membership Auditing

Given (3, $C_7$) return ( $X_3$ , P), where P is:

Valid if root == $C_7$ .

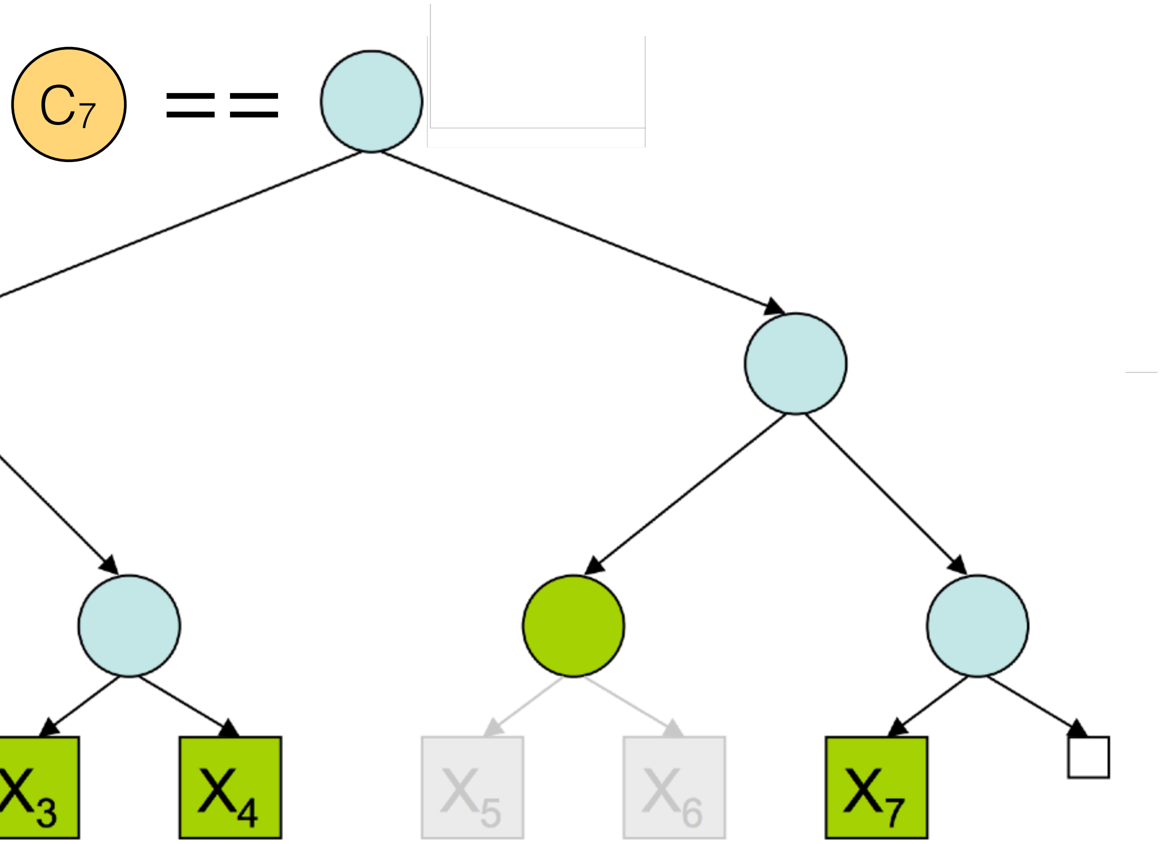


P takes O(log n) to build

# Incremental Auditing

Given ($C_3$, $C_7$) return (P), where P is:

Valid if:
- P is consistent with $C_7$
- P is consistent with $C_3$



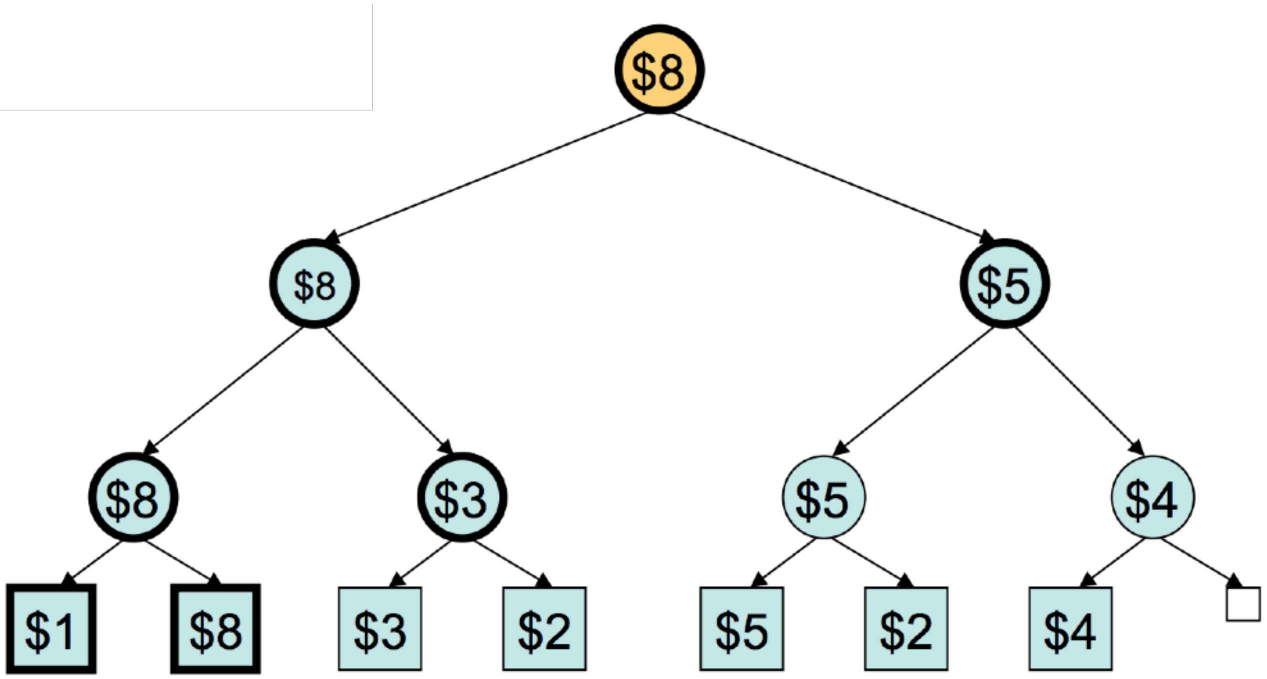
P takes O(log n) to build

# Merkle Aggregation

History trees can be extended to annotate events with attributes.

**Application**: support content searches.

- Max()



Find all transactions over $6

# Performance

- Insert performance: 1,750 events/sec
    - 2.4%: Parse the log event
    - 2.6%: Insert the event to the tree
    - 11.8%: Get root commitment
    - 83.3%: Sign commitment
- Proof generation:
    - With locality (all events in RAM):
        - 10,000-18,000 incremental proofs/sec
        - 8,600 membership proofs/sec
    - Without locality
        - 30 membership proofs/sec

# Recap

- History trees allow the logger to store log events and generate integrity proofs efficiently.
- Other hosts (auditors) need to demand those proofs to ensure the logs are not tampered.
- Result: the logger can be untrusted (but at least one auditor needs to be honest).

# Discussion

- No security analysis: what happens if a client colludes with the logger? What if the secret key of the logger is compromised?
- No full-system evaluation with multiple hosts. Network overhead? Overhead of redistributing commitments with gossip? Scalability?
- No auditing strategies are presented. What kind of audits, from whom and how often should be asked to the logger? What happens when tampering is detected? Lying auditors?