Decentralizing Authorities into Scalable Strongest-Link Cothorities

Ewa Syta, Iulia Tamas, Dylan Visher, David Wolinsky, Bryan Ford

Yale University
Computer Science Department
Talk Outline

• Trouble with Authorities
• Towards Cothorities: Large-scale Collective Authorities
• A First Step: Scalable Collective ElGamal Log-Signing
• The Availability Problem, and Two Solutions
• Prototype and Preliminary Results
• Future Work: Potential Applications
“Authorities” are Everywhere

Conceptually simple but security-critical services

• Logging, Time-stamping Authorities

• Naming Authorites

• Certificate Authorities

• Randomness Authorities (e.g., Lotteries)

• Digital Notaries
Tamper-Evident Logging

- Authority services often produce, rely on tamper-evident public logs
  - Each log entry signed by the authority
  - Hash chains for consistency verification

- But hashes don't solve the **forking problem**...

- Or the **freshness problem**...

```
1 record 2 record 3 record
   ^                  ^      ^
  Head               Head 1  Head 2
```

```
1 record 2 record 3 record
   ^                  ^      ^
  "Head"             3 record
```
When authorities go bad...

Compromised authority services can:

- Tamper with history: e.g., forge log entries
- Pre-date or post-date a timestamp
- Equivocate: customize history for each user
- Impersonate names and MITM attack
- Look into the future: e.g., win the lottery

And usually you're trusting one entity to be good
If we trust many authorities...

Attacker gets to choose which authority to attack

→ Weakest-link security overall
Example: Certificate Authorities

EFF SSL Observatory:

- ~650 CAs trusted by Mozilla or Microsoft
- Any CA can issue certs for any domain name
- Prime key theft target
  - MITM attack power
- Breaches do happen
  - DigiNotar, Comodo, CNNIC/MCS
Talk Outline

• Trouble with Authorities

• **Towards Cothorities: Large-scale Collective Authorities**

• A First Step: Scalable Collective ElGamal Log-Signing

• The Availability Problem, and Two Solutions

• Prototype and Preliminary Results

• Future Work: Potential Applications
Splitting Trust in Authorities

We know how to:

• Split trust across a few servers, typically <10
  – “Anytrust”: only 1-of-k servers need be honest, but all k servers need to remain live
  – Byzantine Fault Tolerance (BFT): 2/3 of k servers need to be honest, 2/3 need to be live

• Split cryptographic keys, operations
  – Threshold cryptography, multisignatures

Example: Tor directory authority (8 servers)
Small-Scale Trust-Splitting

Is splitting trust across 5-10 replicas “enough”?  

• Who owns/controls these replicas?  
  – Truly independent operators (decentralized), or within one organization (merely distributed)?  
  – All in same country? All in “five-eyes” territory?  

• What is the real cost of targeted attacks?  
  – 5 Tor directory server private keys might be well worth the cost of a 0-day exploit or two  

• Who chooses the 5-10 replicas?  
  – Why should “everyone” trust them?
Large-Scale Trust Splitting

Main proposition:

*We can and should build authority services to split trust across large-scale collectives*

- e.g., thousands of replicas/monitors or more

Result:

**Collective Authorities or Cothorities**
Why Large-Scale Trust Splitting

Basic goals:

- Transform authorities from “weakest-link” to “strongest-link” security model
  - Remain secure unless many nodes compromised

- Split trust across broad diversity of servers, operators, organizations, countries, interests, alternative software implementations, ...
  - Every user can find someone they really do trust

- Make adding participants cheap and always beneficial → can only increase security
Talk Outline

• Trouble with Authorities
• Towards Cothorities: Large-scale Collective Authorities
• A First Step: Scalable Collective ElGamal Log-Signing
• The Availability Problem, and Two Solutions
• Prototype and Preliminary Results
• Future Work: Potential Applications
Scalable Collective Signing

Basic primitive: a tamper-evident logging cothority

Simple operation model (for now):

- **Leader** server generates log entries, timeline
- **Follower** servers (e.g., thousands) collectively witness and “sign off” on log entries
- Each log entry gets **single collective signature**: small, quick and easy for anyone to verify

→ Leader cannot roll back or rewrite history, or equivocate, without *many* colluding followers
  - Can't sign valid log entries without followers!
Scalable Collective Signing

- Public log
  - Records signed collectively by leaders and followers

Collective Authority (cothority)

1. Record
2. Record
3. Record

Each record collectively signed
Basic Operation

For each protocol round:

• Servers collect items to be signed.

• Servers aggregate all items into a Merkle tree, which cryptographically summarizes all items.

• Servers collectively sign the root of the tree using a Schnorr multisignature.

• A signed root becomes a new log entry.
CoSi Crypto Primitives

Builds on well-known primitives:

• Merkle Trees
• Schnorr Signature and Multisignatures

CoSi builds upon existing primitives but makes it possible to scale to thousands of nodes

• Using communication trees and aggregation, as in scalable multicast protocols
Merkle Trees

- Every non-leaf node labeled with the hash of the labels of its children.
- Efficient verification of items added into the tree
- Authentication path - top hash and siblings hashes

```
E = H(H(A) | H(B))
F = H(H(C) | H(D))
G = H(H(E) | H(F))
```

Diagram:
- Top hash node
- Intermediate hash nodes
- Leaf nodes: A, B, C, D
Schnorr Signature

- Generator $g$ of prime order $q$ group
- Public/private key pair: $(K=g^k, k)$

<table>
<thead>
<tr>
<th>Signer</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commitment</td>
<td>$V=g^v$</td>
</tr>
<tr>
<td>Challenge</td>
<td>$c$</td>
</tr>
<tr>
<td>Response</td>
<td>$r = (v - kc)$</td>
</tr>
</tbody>
</table>

Signature on $M$: $(c, r)$

Commitment recovery
$V' = g^rK^c = g^{v-kc}g^{kc} = g^v = V$

Challenge recovery
$c' = H(M|V')$

Decision
$c' = c$? ✔
Collective Signing

• Our goal is collective signing with N signers
  • Everyone produces a signature
  • N signers -> N signatures -> N verifications!
  • Bad for thousands of signers!

• Better choice – a multisignature
Schnorr Multisignature

- Key pairs: \((K_1=g^{k_1}, k_1)\) and \((K_2=g^{k_2}, k_2)\)

<table>
<thead>
<tr>
<th>Signer 1</th>
<th>Signer 2</th>
<th>Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commitment</td>
<td>(V_1=g^{v_1})</td>
<td>(V_2=g^{v_2})</td>
</tr>
<tr>
<td>Challenge</td>
<td>(c)</td>
<td>(c)</td>
</tr>
<tr>
<td>Response</td>
<td>(r_1 = (v_1 - k_1c))</td>
<td>(r_2 = (v_2 - k_2c))</td>
</tr>
</tbody>
</table>

**Signature on M:** \((c, r_1)\)  Same signature!

Commitment recovery  Same verification!  \(V' = g^{rK^c}\)  \(K=K_1*K_2\)
Challenge recovery Done once!  \(c' = H(M|V')\)
Decision  \(c' = c\)?
CoSi Protocol

1. Announcement Phase

2. Commitment Phase

3. Challenge Phase

4. Response Phase
CoSi Protocol

1. Make multisignatures scale to thousands of nodes
   • Limit the computation and network bandwidth costs of each server
   • Tree-based communication structure

2. Handle server failures
   • Rare but non-negligible
   • DoS attacks possible if not addressed
   • Two approaches
Talk Outline

• Trouble with Authorities
• Towards Cotherorities: Large-scale Collective Authorities
• A First Step: Scalable Collective ElGamal Log-Signing
• The Availability Problem, and Two Solutions
• Prototype and Preliminary Results
• Future Work: Potential Applications
Approach 1: Exceptions

- If node A fails, the remaining nodes can provide a valid signature but
  - For a modified collective key: $K' = K \times K^{-1}_A$
  - Using a modified commitment: $V' = V \times V^{-1}_A$
  - And response: $r' = r - r_A$

- Client gets a signature under $K'$ along with an exception $e_A$
  - $e_A$ also lists conditions under which it was issued

- Client accepts **only** if a quorum of nodes maintained
Approach 2: Life insurance

- Node "insures" its private key by depositing the key shares with other signers (insurers)
- If node fails, others recover the key and continue
- Use Shamir verifiable secret sharing (VSS)
Talk Outline

• Trouble with Authorities
• Towards Cothorities: Large-scale Collective Authorities
• A First Step: Scalable Collective ElGamal Log-Signing
• The Availability Problem, and Two Solutions
• **Prototype and Preliminary Results**
• Future Work: Potential Applications
Implementation

• Implemented in Go with dedis crypto library

• Schnorr multisignatures on Ed25519 curve
  – AGL's Go port of DJB's optimized code

• Run experiments on DeterLab
  – Up to 4096 virtual CoSi nodes
  – Multiplexed atop up 32 physical machines
  – Latency: 100ms roundtrip between two servers
Preliminary Results

Latency vs. Number of Hosts

Per-Round Signing Latency (s)

Number of Timestamp Servers

- min
- max
- avg
Preliminary Results

System and User Time vs. Number of Hosts

System and User Time (s)

Number of Timestamp Servers

64 256 1024 4096
Talk Outline

• Trouble with Authorities
• Towards Cothorities: Large-scale Collective Authorities
• A First Step: Scalable Collective ElGamal Log-Signing
• The Availability Problem, and Two Solutions
• Prototype and Preliminary Results
• Future Work: Potential Applications
Already (or close to) usable for:

- Tamper-evident logging
  - History rewriting protection
  - Equivocation protection
- Secure timestamping
  - Pre/post-dating protection
- Large-scale Byzantine Consensus
  - Propose/commit, view changes implemented
  - Still need validation, evaluation, optimization
Secure Randomness/Lotteries

Current version with exceptions for availability:

- Protects from anyone predicting the future
- Protects from anyone rigging the outcome
- *Not yet* fully bias-protected if leader malicious

Shamir secret-sharing version can fix bias risk

- Collective commits to *single unknown value*
- Ensures *exactly that value* as ultimate output
Certificate Cothorities

Proactive protection against fake certs, MITM

• Ideal: browser vendor leads a cothority
  – CAs join, check and collectively sign all certs
  – Any CA can block signature if cert violates policy
    • e.g., only Google CA can sign 'google.com' cert

• Alternative: root CA leads a cothority
  – Migrates sub-CAs into cothority membership, phases out availability of delegated authority

• Alternative: based on Certificate Transparency
  – Log servers as cothorities, collectively signed SCTs
Certificate Transparency

Fake Google.com
(MITM Attacker)

DigiNotar

google.com

Auditors

Log Servers

Monitors

CAs

Client
Improving CT

Fake Google.com (MITM Attacker)

Collective Certificate Signing

DigiNotar

CAs

Log Servers

Auditors

Monitors
CT as a Couthority

Fake Google.com (MITM Attacker)

Collective Certificate Signing

Go Daddy
DigiNotar
Google CA
DigiCert
GlobalSign
VeriSign
Comodo
Handy Software Labs
GeoTrust

Client
A Better Blockchain?

Decentralized consensus, secure ledgers

- Without proof-of-work, massive power waste
- Without risk of temporary forks
- Without 51% attack vulnerability
- Stronger protection for clients, “light” nodes
  - Just check *one* log-head signature for correctness
- Efficient: with FawkesCoin hash-based ledger, just *one* public-key crypto operation per round
- Scalable: every server need not store, verify every record throughout blockchain history
Conclusion

Cothorities build on old ideas

- Distributed/Byzantine consensus protocols
- Threshold cryptography, multisignatures

Show that they can scale to thousands of servers

- Strongest-link security among many witnesses
- Practical: demonstrated for 4000+ servers
- Efficient: 1.5-second signing latency at scale