Decentralizing Authorities into Scalable Strongest-Link Cothorities

Ewa Syta, Iulia Tamas, Dylan Visher, David Wolinsky – Yale University
Bryan Ford, Linus Gasser, Nicolas Gailly – Swiss Federal Institute of Technology (EPFL)

Stanford University – October 9, 2015
We depend on many authorities

**Conceptually simple** but **security-critical** services

- Logging, Time-stamping Services, Digital Notaries
- Naming Authorites (ICANN logo)
- Certificate Authorities
- Randomness Authorities (e.g., Lotteries)
- Software Update Services
But are authorities trustworthy?

WIRED

Hack Obtains 9 Bogus Certificates for Prominent ...

HACK OBTAINS 9 BOGUS CERTIFICATES FOR PROMINENT WEBSITES; TRACED TO IRAN
But are authorities trustworthy?

This Dude Hacked Lottery Computers To Win $14.3M Jackpot In U.S.

By Waqas on April 14, 2015  Email  @hackread
But are authorities trustworthy?

D-LINK ACCIDENTALLY LEAKS PRIVATE CODE-SIGNING KEYS

by Michael Mimoso  
Follow @mike_mimoso
Talk Outline

• The Need to Decentralize Internet Authorities
• Witness Cothorities: Transparency via Collective Signing
• Timestamp Cothorities: Collectively Attesting Freshness
• Randomness Cothorities: Scalable Unbiased Randomness
• Conclusions and Future Work
Talk Outline

• The Need to Decentralize Internet Authorities
• Witness Cothorities: Transparency via Collective Signing
• Timestamp Cothorities: Collectively Attesting Freshness
• Randomness Cothorities: Scalable Unbiased Randomness
• Conclusions and Future Work
Why do we have authorities?
Why do we have authorities?

Respect my Authoritah!

What is:
- Gmail's SSL public key?
- Bob's IM public key?
- Latest version of App?
When authorities go bad

Respect my Authoritah!

Fake Gmail
Fake Bob
Fake Software Update

Gmail
Bob
Software Update
Key Problem #1

Authorities (and their private keys) are powerful

- Bad CA → MITM any web site
- Bad keyserver → impersonate any user
- Bad update server → instant backdoor

Attractive targets for hackers, criminals, spy agencies
Key Problem #2

There are many authorities: e.g., hundreds of CAs trusted by web browsers

- Any CA can issue cert for any domain name

Hacker (or spy agency) needs only one CA key

- Weakest-link security
- @$% happens

DigiNotar, Comodo, CNNIC/MCS
Challenge: Decentralize Authorities

Split important authority functions across multiple participants (preferably independent)

- So authority isn't compromised unless multiple participants compromised

From *weakest-link* to *strongest-link* security
Decentralizing Trust

We have many technical tools already

- “Anytrust”: 1-of-k servers honest, all k live
- Byzantine replication: 2/3 honest, 2/3 live
- Threshold cryptography, multisignatures

Example: Tor directory authority (8 servers)
Limitations of Trust-Splitting

Trust-splitting is rare, challenging to implement, usually scales only to small groups.

- Is splitting across 5-10 servers **enough**?
- Are they **truly independent** and **diverse**?
- Who **chooses** the composition and how?

Are we convinced there is no adversary powerful enough to hack 5 of 8 directory servers?
Grand Challenge: Trust Scaling

Large-scale collective authorities: “Cothorities”

- Split trust over hundreds, thousands of parties
- Correct unless large fraction compromised

E.g.: replace hundreds of CAs with one CA with authority split across hundreds of parties

- Diversity of servers, operators, organizations, countries, interests, software, hardware, ...
- Make adding participants cheap, efficient
- Ensure security scales with size and sensitivity
Talk Outline

• The Need to Decentralize Internet Authorities
• **Witness Cothorities: Transparency via Collective Signing**
  • CoSi: Scalable Collective Multisignatures
  • Implementation and Preliminary Experimental Results
  • Applications: Secure Logging, Proactive Transparency
• Timestamp Cothorities: Collectively Attesting Freshness
• Randomness Cothorities: Scalable Unbiased Randomness
• Conclusions and Future Work
A First-Step Goal

Generically improve security of any authority, independent of authority type or semantics

Introducing Witness Cothorities...
Witness Cothorities

“Who watches the watchers?”
Public witnesses!

Enforce two security properties:

• Any signed authoritative statement has been widely witnessed
• Any signed authoritative statement conforms to checkable standards

Respect my Authoritah!

Witnesses
CoSi: Collective Signing

Operation:

• **Authority** server generates statements

• **Witness** servers collectively sanity-check and *contribute* to authority's signature

• Each statement gets a *collective signature*: small, quick and easy for anyone to verify

→ Authority (or key thief) can't sign anything in secret without *many* colluding followers
CoSi: Collective Signing

"The time is 3PM."

"Gmail's public key is X."

"The latest version of Firefox is Z."

"Bob's public key is Y."
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
# 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: $V' = g^r K^c = g^{v-kc} g^{kc} = g^v = V$
- Challenge recovery: $c' = H(M | V')$
- Decision: $c' = c$?
Collective Signing

- Goal: collective signing with $N$ signers
  - Strawman: everyone produces a signature
  - $N$ signers -> $N$ signatures -> $N$ verifications
  - Bad if we have thousands of signers

- Better choice: multisignatures
Schnorr Multisignature

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

<table>
<thead>
<tr>
<th></th>
<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>
<td>$V_1$</td>
</tr>
<tr>
<td>Challenge</td>
<td>$c$</td>
<td>$c$</td>
<td>$c = H(M</td>
</tr>
<tr>
<td>Response</td>
<td>$r_1 = (v_1 - k_1 c)$</td>
<td>$r_2 = (v_2 - k_2 c)$</td>
<td>$r_1$</td>
</tr>
</tbody>
</table>

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

- Commitment recovery  
  Same verification!  
  $V' = g^{rK^c}$  
  $K = K_1 \cdot K_2$

- Challenge recovery  
  Done once!  
  $c' = H(M | V')$

- Decision  
  $c' = c$?
CoSi Protocol Setup

Merkle tree containing:

- Public keys $K_i$ (discrete-log)
- Self-signed Certificates
- Aggregate keys $K_i$

$O(n)$ one-time verify cost
$O(|n'-n|)$ group change
CoSi Protocol Rounds

1. Announcement Phase

2. Commitment Phase

3. Challenge Phase

4. Response Phase
CoSi Commit Phase

Merkle tree containing:

- Commits $V_i$
- Aggregate commits $V_i$

Collective challenge $c$ is root hash of per-round Merkle tree

$$V_1 = g^{v_1}, \quad V_2 = V_1 V_2 V_3 V_4$$
CoSi Response Phase

Compute

- Responses $r_i$
- Aggregate responses $r_i$

Each $(c, r_i)$ forms valid **partial** signature

$(c, r_1)$ forms **complete** signature

$\begin{align*}
  r_3 &= v_3 - k_3 c, \\
  r_3 &= r_3 \\
  r_4 &= v_4 - k_4 c, \\
  r_4 &= r_4 \\
  r_1 &= v_1 - k_1 c, \\
  r_1 &= r_1 + r_2 + \ldots + r_N
\end{align*}$
The Availability Problem

Assume server failures are rare but non-negligible

• Availability loss, DoS vulnerability if not addressed
• But *persistently bad* servers administratively booted

Two approaches:

• Exceptions – currently implemented, working
• Life Insurance – partially implemented, in-progress
Simple Solution: Exceptions

• If node A fails, remaining nodes create signature
  • For a modified collective key: \( K' = K \cdot K^{-1}_A \)
  • Using a modified commitment: \( V' = V \cdot V^{-1}_A \)
  • And modified response: \( r' = r - r_A \)

• Client gets a signature under \( K' \) along with exception metadata \( e_A \)
  • \( e_A \) also lists conditions under which it was issued

• Client accepts **only** if a quorum of nodes maintained
Talk Outline

• The Need to Decentralize Internet Authorities

• **Witness Cothorities: Transparency via Collective Signing**
  • CoSi: Scalable Collective Multisignatures

• **Implementation and Preliminary Results**
  • Applications: Secure Logging, Proactive Transparency

• Timestamp Cothorities: Collectively Attesting Freshness

• Randomness Cothorities: Scalable Unbiased Randomness

• Conclusions and Future Work
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 8192 virtual CoSi nodes
  – Multiplexed atop up 64 physical machines
  – Latency: 100ms roundtrip between two servers
Results: Collective Signing Time

The graph shows the collective signing time for different numbers of co-signing hosts. The x-axis represents the number of co-signing hosts, ranging from 2 to 8192. The y-axis represents the seconds per round.

Three methods are compared:
- **JVSS**
- **Naive**
- **Couthority**

The graph highlights the performance of each method as the number of co-signing hosts increases. The JVSS method consistently demonstrates lower signing times compared to the Naive and Couthority methods.

Key markers on the graph indicate specific performance benchmarks:
- 8.6 sec for a certain number of hosts
- 38 sec for another set of hosts
- 8 sec for yet another set

These benchmarks provide insights into the scalability and efficiency of each method under varying conditions.
Results: Computation Cost

The graph illustrates the computation cost for different methods as the number of co-signing hosts increases. The y-axis represents the total seconds over all rounds, while the x-axis shows the number of co-signing hosts. The methods compared are JVSS, Naive, Cothority, User time, and System time. The graph shows a clear trend of increasing computation cost with the number of co-signing hosts for all methods.
Talk Outline

• The Need to Decentralize Internet Authorities

• **Witness Cothorities: Transparency via Collective Signing**
  • CoSi: Scalable Collective Multisignatures
  • Implementation and Preliminary Experimental Results
  • **Applications: Logging, Proactive Transparency**

• Timestamp Cothorities: Collectively Attesting Freshness

• Randomness Cothorities: Scalable Unbiased Randomness

• Conclusions and Future Work
Application: Secure Logging

• Many authorities make “public statements”

• Often recorded in **tamper-evident public log**
  - Hash chains for consistency verification

• But hashes don't address **equivocation**...

• Or **freshness**...

“Head”
Witnessing Public Log Servers

- Witnesses collectively verify log structure, Leader can't equivocate without being busted

Each record collectively signed

1 record → 2 record → 3 record

Log Server

Witness Cothority

Witnesses
The Transparency Challenge

Respect my Authoritah!

Fake Gmail

Fake Bob

Fake Software Update

Gmail by Google

Bob

Software Update
Current Transparency Solutions

Respect my Authoritah!

Witnesses
- public logs
- monitors
- auditors

- Perspectives
- Certificate Transparency
- AKI, ARPKI
- CONIKS

Alice

Bob

Gmail by Google

Software Update

!!
An Important Assumption

Respect my Authoritah!

Assumes Alice can, and is willing to, gossip with witnesses.

Takes time, may compromise Alice's privacy.

Freetopia

Witnesses

public logs
monitors
auditors

Gmail
by Google

Software Update
A Different Scenario

Tyrannia

Gen. Rex

Fake CA

Alice

Firewall

Fake Log

Freetopia

Respect my Authoritah!

Witnesses

public logs
monitors
auditors

Bob

Software Update

Fake CA

Gmail by Google
Gossip versus Collective Signing

Gossip can't protect Alice if she...

- Can't (because she's in Tyrannia)
- Doesn't want to (for privacy), or
- Doesn't have time to

cross-check each authoritative statements.

Collective signing proactively protects her from secret attacks even via her access network.

- Attacker can't secretly produce valid signature
Talk Outline

• The Need to Decentralize Internet Authorities
• Witness Cotherities: Transparency via Collective Signing
• Timestamp Cotherities: Collectively Attesting Freshness
• Randomness Cotherities: Scalable Unbiased Randomness
• Conclusions and Future Work
Software Update Scenario

Alice, traveling in Tyrannia, is offered a **software update** for her favorite app

- Claims to be “latest version” - but is it?
- Rex's firewall might inject **authentic** but **outdated, now exploitable** version
- If Alice accepts, she is **instantly Pwned**; retroactive transparency won't help!
Timestamping Cothority

Like classic **digital timestamp** services, only decentralized.

- **Each round (e.g., 10 secs):**
  1) Each server collects hashes, nonces to timestamp
  2) Each server aggregates hashes into Merkle tree
  3) Servers aggregate local trees into one global tree
  4) Servers collectively sign root of global tree
  5) Server give signed root + inclusion proof to clients

- **Clients verify signature + Merkle inclusion proof**
Verifiably Fresh Software Updates

Alice accepts only updates with fresh timestamp:

- Knows update can't be an outdated version: tree contains inclusion proof of *her* nonce
- Knows update can't have targeted backdoor: witness couthority ensures *many* parties saw it
Talk Outline

• The Need to Decentralize Internet Authorities
• Witness Cothorities: Transparency via Collective Signing
• Timestamp Cothorities: Collectively Attesting Freshness
• **Randomness Cothorities: Unbiased Public Randomness**
• Conclusions and Future Work
Unbiased Public Randomness

Need authority that can “flip coins” in public, convince everyone result is \textit{fair} and \textit{unbiased}.

- Choose lottery winner
- Sampling ballots in election auditing
- Pick BFT clusters from large pool of servers
- Divide large user network into smaller random anonymity sets
  - e.g., Herbivore [Goel/Sirir '04]
Related: Existing Approaches

Algorithmic work on quorum-building

- e.g., King et al, ICDCN 2011
- Unclear how to implement, apply

Randomness via “slow hashes”

- e.g., Lenstra/Wesolowski, 2015
- New, nonstandard crypto assumptions
CoSi Protocol Responses?

Appealing near-solution:

- Contributions from all participants
- Committed in advance, unpredictable until last phase

But can still be *biased* by leader with $k$ colluders

- Use exceptions to pick “best of” $2^k$ outcomes

$$r_1 = v_1 - k_1c,$$
$$r_1 = r_1 + r_2 + ... + r_N$$

$$r_2 = v_2 - k_2c,$$
$$r_2 = r_2 + r_3 + r_4$$

$$r_3 = v_3 - k_3c,$$
$$r_3 = r_3$$

$$r_4 = v_4 - k_4c,$$
$$r_4 = r_4$$
Availability via “life insurance”

- Node "insures" its private key by depositing the key shares with threshold group of "trustee" servers
  - Shamir verifiable secret sharing (VSS)
- Trustees can sign on behalf of failed node
The Challenge

How to pick set of trustees for given witness?

• All nodes trustees (JVSS): doesn't scale, $O(N^2)$
• Witness-chosen: can pick bad group $\rightarrow$ DoS
• Leader-chosen: pick cronies, get secret early

We need **unbiased public randomness**
to pick these random trustee subgroups,
to get **unbiased public randomness**!
RandHound: Protocol Sketch

Intuition: bootstrap from pairwise unbiased randomness

1) Leader commits to random value $R_L$, each follower $i$ commits to random $R_i$

2) Reveal; follower $i$ picks trustees via $H(R_L, R_i)$, deals secret $S_i$ to picked trustees

3) Leader commits to threshold set of secrets s.t. must include at least one honest follower

4) Followers reveal dealt secret shares
RandHound: Security Properties

Assuming a fraction of participants are honest:

**Unpredictability**: no one can recover the (one) honest follower's secret before final reveal phase

**Unbiasability**: only one possible outcome after leader's threshold-set commit in phase 3

**Availability**: protocol runs to completion w.h.p. unless leaderdishonestly colludes to DoS itself

**Scalability**: \( O(NT) \), where 
\# trustees \( T \) depends only on security parameter
Status

Still preliminary:

- Initial implementation working (code available on DeDiS github)
- Experimentation in-progress
- Cothority integration in-progress
Talk Outline

• The Need to Decentralize Internet Authorities
• Witness Cothorities: Transparency via Collective Signing
• Timestamp Cothorities: Collectively Attesting Freshness
• Randomness Cothorities: Scalable Unbiased Randomness
• Conclusions and Future Work
Ongoing/Future Work

Backward-compatible integration into authorities

- Web PKI: Certificate Authorities, CT, AKI
- Personal PKI: PGP keyservers, CONIKS
- Practical software release, update services

Build more general collective authorities...
Towards Better Blockchains?

Decentralized consensus, secure ledgers

- Without proof-of-work and 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

But demonstrate how to scale trust-splitting

• Strongest-link security among many witnesses
• Practical: demonstrated for 8000+ participants
• Efficient: 1.5-second signing latency at scale