Social Networking with **Frientegrity**: Privacy and Integrity with an Untrusted Provider

Ariel J. Feldman

Princeton → UPenn

Joint work with:
Aaron Blankstein, Michael J. Freedman, and Edward W. Felten
Online social networks are centralized

Pro: Availability, reliability, global accessibility, convenience

Con: 3rd party involved in every social interaction

Must trust provider for confidentiality & integrity
Threats to confidentiality

- Theft by attackers
- Accidental leaks
- Privacy policy changes
- Government pressure
Threats to integrity

Simple: Corrupting messages

Complex: Server equivocation

Equivocation in the wild:

(e.g. to disguise censorship)

Why do I leave Sina microblogging


http://songshinan.blog.caixin.com/archives/22322 (translated by Google)
Limits of prior work

1. Cryptographic

   Don’t protect integrity

2. Decentralized

   Run your own server (sacrifice availability, convenience, etc.)

   OR

   Trust a provider (who you may not know either)
Frientegrity’s approach

Benefit from a centralized provider

Support common features
(e.g. walls, feeds, friends, FoFs, followers)

Assume untrusted provider
Enforce confidentiality

Provider only observes encrypted data

(Need dynamic access control and key distribution)
Verify integrity

Clients verify that the provider:

• Hasn’t corrupted individual updates
• Hasn’t equivocated
• Enforced access control on writes
Scalability challenges

Long histories; only want tail

Don’t verify whole history each time

Many objects (walls, comment threads, photos, etc.)

Support sharding

Many friends and FoFs

O(log n) “(un)friending”
Frientegrity overview

Server 1

Alice’s profile

Alice’s photo album

Comment thread

Alice’s wall

Alice’s ACL

Server 2

Bob’s profile

Bob

Read Alice’s wall

Verify & decrypt

Server n

Checked for equivocation

Optionally entangled

1. Latest updates
2. Proof of no equivocation
3. Proof of ACL enforcement
4. Decryption keys
Detecting equivocation

**Enforce fork* consistency** [LM07]

- Honest server: linearizability
- Malicious server: Alice and Bob detect equivocation after exchanging 2 messages
- Compare histories

Provider can still fork the clients, but can’t unfork
Comparing histories

Previously: use a hash chain

\[ h_n = H(h_{n-1} \| \text{op}_n) \]

Hash chains are $O(n)$
(and must download the whole history)
Objects in Frientegrity

Let $C_{15}$ be a server-signed commitment to $h_{\text{root}}$ up to op$_{15}$

$h_i = H(h_{\text{leftChild}(i)} \parallel h_{\text{rightChild}(i)})$

$h_{\text{root}}$ commits to entire history

History tree [CW09]
Objects (cont.)

Is $C_8$ consistent with $C_{15}$?
Verifying an object

C_{11}

Alice’s ops
Bob’s ops
Charlie’s ops

Clients collaborate to verify the history

Is C_{11} consistent with C_{15}?
Tolerating malicious users

Tolerate up to $f$ malicious users

Alice's ops
Bob's ops
Charlie's ops
Access control

Prove ACL enforcement

Efficient key distribution

O(log n) “(un)friending”
Proving ACL enforcement

Server

Alice's photo album
Comment thread
Alice's ACL
Alice's wall

h_i = H(h_{leftChild(i)} || h_{rightChild(i)})

h_root signed by Alice

Bob
Charlie
Emma
Alice

David

Bob

Verify & decrypt

Persistent authenticated dictionary
[AGT01]
Efficient key distribution


developed

Alice’s photo album
Comment thread
Alice’s ACL
Alice’s wall

Bob
Verify & decrypt

E_{k3}(k_1) \parallel E_{k4}(k_1)

k_0 = k_{alice\_friend}

David, k_0

Bob, k_1

Alice, k_3

Charlie, k_4

Emma, k_5

E_{charlie\_pk}(k_4)

Key graph
[WGL98]
Adding a friend

Server

Alice’s photo album

Comment thread

Alice’s wall

Alice’s ACL

Bob

Verify & decrypt

E_{k5}(k_2) \oplus E_{k6}(k_2)

E_{zack\_pk}(k_6)
Removing a friend

Server

Alice’s photo album

Comment thread

Alice’s ACL

Alice’s wall

Bob

Verify & decrypt

$\mathbf{k_0'} = k_{alice\_friend'}$

David, $k_0'$

Bob, $k_1'$

Sean, $k_2$

Alice, $k_3$

Charlie, $k_4$

Emma, $k_5$

Zack, $k_6$
Efficient enough in practice?

Setup

- Java client & server
- Simulate basic Facebook features (each user has wall & ACL)
- 2048-bit RSA sign & verify batched via spliced signatures [CW10]
- Experiments on LAN (8-core 2.4 GHz Intel Xeon E5620s, Gigabit network)

Measurements

- Latency of reads & writes to objects
- Latency of ACL changes
- Throughput (in paper)
- Effect of tolerating malicious users
Object read & write latency

Frientegrity
(collaborative verification)

Hash chain

Constant cost of signatures dominates
Latency of ACL changes

The graph shows the response latency (in milliseconds) for adding a user and revoking a user as a function of the ACL size. The latency is generally low and consistent for both operations across different ACL sizes.
Tolerating malicious users

- 50 writers
- 5000 operations
Summary

Both confidentiality & integrity need protection

Benefit from centralization, but provider is untrusted

Clients collaborate to defend against equivocation

Scalable, verifiable access control & key distribution
Thank you

Questions?

http://arifeldman.com
ariel.feldman@cis.upenn.edu