Collaborative, Privacy-Preserving Data Aggregation at Scale

Michael J. Freedman
Princeton University

Joint work with: Benny Applebaum, Haakon Ringberg, Matthew Caesar, and Jennifer Rexford
Problem:
Network Anomaly Detection
Collaborative anomaly detection

• Some attacks look like normal traffic
  – *e.g.*, SQL-injection, application-level DoS  [Srivatsa TWEB ‘08]

• Is it a DDoS attack or a flash crowd?  [Jung WWW ‘02]
Collaborative anomaly detection

• Targets (victims) could correlate attacks/attackers

[Katti IMC ’05], [Allman Hotnets ‘06], [Kannan SRUTI ‘06], [Moore INFOC ‘03]

“Fool us once, shame on you. Fool us N times, shame on us.”
Problem:  
Network Anomaly Detection

Solution:
- Aggregate suspect IPs from many ISPs
- Flag those IPs that appear $> \text{threshold } \tau$
Problem: Distributed Ranking

Solution:

• Collect domain statistics from many users
• Aggregate data by domain
Problem:

... 

Solution:

• Aggregate (id, data) from many sources
• Analyze data grouped by id
But what about privacy?

What inputs are submitted?

Who submitted what?
Data Aggregation Problem

- Many participants, each with (key, value) observation
- Goal: Aggregate observations by key

<table>
<thead>
<tr>
<th>Key</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$A(v_a, v_b)$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$A(v_i, v_j, v_k)$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$k_n$</td>
<td>$A(v_x)$</td>
</tr>
</tbody>
</table>
Data Aggregation Problem

- Many participants, each with (key, value) observation
- Goal: Aggregate observations by key

<table>
<thead>
<tr>
<th>Key</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$F(A(v_a, v_b))$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$F(A(v_i, v_j, v_k))$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$k_n$</td>
<td>$F(A(v_x))$</td>
</tr>
</tbody>
</table>

PDA: Only release the value column

CR-PDA: Plus keys whose values satisfy some function
Data Aggregation Problem

• Many participants, each with (key, value) observation
• Goal: Aggregate observations by key

<table>
<thead>
<tr>
<th>Key</th>
<th>Values</th>
<th>≥ τ</th>
</tr>
</thead>
<tbody>
<tr>
<td>k₁</td>
<td>Σ (1, 1)</td>
<td>?</td>
</tr>
<tr>
<td>k₂</td>
<td>Σ (1, 1, 1)</td>
<td>≥ τ</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>kₙ</td>
<td>Σ (1)</td>
<td>≥ τ</td>
</tr>
</tbody>
</table>

PDA: Only release the value column
CR-PDA: Plus keys whose values satisfy some func
Goals

• **Keyword privacy:** No party learns anything about keys

• **Participant privacy:** No party learns who submitted what

• **Efficiency:** Scale to many participants, each with many inputs

• **Flexibility:** Support variety of computations over values

• **Lack of coordination:**
  – No synchrony required, individuals cannot prevent progress
  – All participants need not be online at the same time
# Potential solutions

<table>
<thead>
<tr>
<th>Approach</th>
<th>Keyword Privacy</th>
<th>Participant Privacy</th>
<th>Efficiency</th>
<th>Flexibility</th>
<th>Lack of Coord</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garbled Circuit Evaluation</td>
<td>Yes</td>
<td>Yes</td>
<td>Very Poor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multiparty Set Intersection</td>
<td>Yes</td>
<td>Yes</td>
<td>Poor</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

- **Decentralized**
• Weaken security assumptions?
  – Assume honest but curious participants?
  – Assume no collusion among malicious participants?

• In large/open setting, easy to operate multiple nodes (so-called “Sybil attack”)

Security

Efficiency
Towards Centralization?

Participants

DB
## Potential solutions

<table>
<thead>
<tr>
<th>Approach</th>
<th>Keyword Privacy</th>
<th>Participant Privacy</th>
<th>Efficiency</th>
<th>Flexibility</th>
<th>Lack of Coord</th>
</tr>
</thead>
<tbody>
<tr>
<td>Garbled Circuit Evaluation</td>
<td>Yes</td>
<td>Yes</td>
<td>Very Poor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multiparty Set Intersection</td>
<td>Yes</td>
<td>Yes</td>
<td>Poor</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Hashing Inputs</td>
<td>No</td>
<td>No</td>
<td>Very Good</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Network Anonymization</td>
<td>No</td>
<td>Yes</td>
<td>Very Good</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Towards semi-centralization

Assumption: Proxy and DB do not collude
### Potential solutions

<table>
<thead>
<tr>
<th>Approach</th>
<th>Keyword Privacy</th>
<th>Participant Privacy</th>
<th>Efficiency</th>
<th>Flexibility</th>
<th>Lack of Coord</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decentralized</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Garbled Circuit Evaluation</td>
<td>Yes</td>
<td>Yes</td>
<td>Very Poor</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Multiparty Set Intersection</td>
<td>Yes</td>
<td>Yes</td>
<td>Poor</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Centralized</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hashing Inputs</td>
<td>No</td>
<td>No</td>
<td>Very Good</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Network Anonymization</td>
<td>No</td>
<td>Yes</td>
<td>Very Good</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>This Work</td>
<td>Yes</td>
<td>Yes</td>
<td>Good</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Privacy Guarantees

• Privacy of PDA against malicious entities and participants
  – Malicious participant may collude with either malicious proxy or DB, but not both
  – May violate correctness in almost arbitrary ways

• Privacy of CR-PDA against honest-but-curious entities and malicious participants
1. Client sends input k
PDA Strawman #1

1. Client sends encrypted input $k$  
2. Proxy batches and retransmits  
3. DB decrypts input

Violates keyword privacy
1. Client sends hashes of $k$
2. Proxy batches and retransmits
3. DB decrypts input

Still violates keyword privacy: IPs drawn from small domains
1. Client sends keyed hashes of k
   - Keyed hash function (PRF)
   - Key s known only by proxy

But how do clients learn $F_s(IP)$?
Our Basic PDA Protocol

1. Client sends keyed hashes of $k$
   - $F_s(x)$ learned by client through Oblivious PRF protocol
2. Proxy batches and retransmits keyed hash
3. DB decrypts input
Basic CR-PDA Protocol

1. Client sends keyed hashes of k, and encrypted k for recovery
2. Proxy retransmits keyed hash
3. DB decrypts input
4. Identify rows to release and transmit $E_{PRX}(k)$ to proxy
5. Proxy decrypts k and releases

<table>
<thead>
<tr>
<th>$F_s(k)$</th>
<th>#</th>
<th>Enc’d k</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_s(1.1.1.1)$</td>
<td>1</td>
<td>$E_{PRX}(1.1.1.1)$</td>
</tr>
<tr>
<td>$F_s(2.2.2.2)$</td>
<td>9</td>
<td>$E_{PRX}(2.2.2.2)$</td>
</tr>
</tbody>
</table>
• **Keyword privacy:** Nothing learned about unreleased keys
• **Participant privacy:** Key $\leftarrow\rightarrow$ Participant not learned

- Any coalition of HBC participants
- HBC coalition of proxy and participants
- HBC database
Privacy Properties

- **Keyword privacy**: Nothing learned about unreleased keys
- **Participant privacy**: Key $\leftrightarrow$ Participant not learned

- Any coalition of HBC participants
- HBC coalition of proxy and participants
- HBC database

\[
E_{DB}(E_{PRX}(k)) \quad E_{DB}(F_s(k)) \quad F_s(k)
\]

Participant $\leftrightarrow$ Proxy $\leftrightarrow$ DB

retransmits $E_{PRX}(k)$
More Robust PDA Protocol

- ORPF → Encrypted OPRF Protocol
- Ciphertext re-randomization by proxy
- Proof by participant that submitted k’s match

- Any coalition of HBC participants
- Malicious participants
- HBC coalition of proxy and participants
- HBC database
- HBC coalition of DB and participants
Encrypted-OPRF protocol

- Problem: in basic OPRF protocol, participant learns $F_s(k)$
- **Encrypted-OPRF protocol**:
  - Client learns blinded $F_s(k)$
  - Client encrypts to DB
  - Proxy can unblind $F_s(k)$ “under the encryption”

\[
(\text{Enc} (\langle F_s(k) \rangle r))
\]

ElGamal $g^{(\pi_{k=1} s_i)} \mod p$
Encrypted-OPRF protocol

- Problem: in basic OPRF protocol, participant learns $F_s(k)$
- Encrypted-OPRF protocol
  - Client learns blinded $F_s(k)$
  - Client encrypts to DB
  - Proxy can unblind $F_s(k)$ “under the encryption”

\[
(Enc((F_s(k)))^r)\]

- OPRF runs OT protocol for each bit of input $k$
- OT protocols expensive, so use batch OT protocol [Ishai et al]
Scalable Protocol Architecture

Participants

Client-Facing Proxies

- Share secret $s$

Front-End DB Tier

- Share DB key

Back-End DB Storage

- Partition $F_s$ keyspace

Proxy Decryption Oracles

- Share PRX key
Evaluation

• Scalable architecture implemented
  – Basic CR-PDA / PDA protocol + and encrypted-OPRF protocol w/ Batch OT
  – ~5000 lines of threaded C++, GnuPG for crypto

• Testbed of 2 GHz Linux machines

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA / ElGamal</td>
<td>key size</td>
<td>1024 bits</td>
</tr>
<tr>
<td>Oblivious Transfer</td>
<td>k</td>
<td>80</td>
</tr>
<tr>
<td>AES</td>
<td>key size</td>
<td>256 bits</td>
</tr>
</tbody>
</table>
Throughput vs. participant batch size

Single CPU core for DB and proxy each
Maximum throughput per server

Four CPU cores for DB and proxy (each)
Throughput scalability

Number CPU cores per DB and proxy (each)
Summary

- **Privacy-Preserving Data Aggregation protects:**
  - Participants: Do not reveal who submitted what
  - Keywords: Only reveal values / released keys

- **Novel composition of crypto primitives**
  - Based on assumption that 2+ known parties don’t collude

- **Efficient implementation of architecture**
  - Scales linearly with computing resources
  - Ex: Millions of suspected IPs in hours

- **Of independent interest...**
  - Introduced encrypted OPRF protocol
  - First implementation/validation of Batch OT protocol