Interoperable privacy preserving user identity and discovery for E2EE messaging

Draft proposal for Mimi
Giles Hogben, Femi Olumofin
Functional Requirements

For a given messaging service identity handle (Phone number or alphanumeric UserID):
1. Retrieve key material and message delivery endpoint
2. Return optional default receiver service ID user preference
Privacy Requirements

- Resolver service should not learn the UserID a client is querying for.
- Resolver service should not learn the public identity of the querying client (i.e. who is sending a message to who).
- Resolver service should not learn the exact timing of when a message is sent.
Privacy non-requirement

**Hiding service reachability**

- All major E2EE messaging services already publish unACL’d reachability information without opt-out. e.g. +16501234567, reachable on Whatsapp.
Registration Phase

Alice → KDS ← Bob
PubInfAlice → ← PubInfBob

Session Setup

Alice → KDS ← Bob
Request for PubInfBob → ← PubInfBob

Symmetric-ratchet

Alice → Bob
Symmetric key derivation → Symmetric key derivation

calculation of shared key
calculation of shared key

Asymmetric-ratchet

Alice ← Bob
Message + new key

generation of new key
Option 1: brute force query - too expensive and leaks private info

Client

Registration of UserID + service ID.

Pubkey bundle request

Messing service KDS 1

Messing service KDS 2

Messing service KDS 3
Option 2: Centralized hub - expensive and organizationally complex
Option 3 (Preferred): Federated with KDS resolver service

Client

KDS resolver

Pubkey bundle request

Messaging service 1
KDS

Messaging service 2
KDS

Messaging service 3
KDS

Resolution of UserID to service ID(s)

Registration of UserID + service ID.
Message delivery (similar to key distribution)
Preferred service integrity
Privacy of resolver queries

● Goal: prevent leakage of the user's social graph to resolvers and other parties

● Setting: User may query a PN/userID in an ad hoc manner or in a batch (e.g., key bundle download for all of a user's address book contacts)

● Our proposal: Private Information Retrieval (PIR)
  ○ Google's PIR framework to transform any standard lattice-based homomorphic PIR scheme into efficient keyword PIR
  ○ Approach is feasible with privacy - cost tradeoff that we consider as reasonable
Homomorphic Encryption

\[
\begin{align*}
a + b &= a + b \\
a \times b &= a \times b
\end{align*}
\]
Private Information Retrieval

I want $M[4] = ?$
Private Information Retrieval

I want $M[4] = ?$

$4 \Rightarrow 00010.$

$$\sum_{i=1}^{5} a_i b_i = 0 \times 0 + 0 \times 0 + 0 \times 0 + 1 \times 1 + 0 \times 0 = 1$$

$d$
Google Keyword PIR framework

- Framework transforms standard lattice-based PIR schemes into keyword PIR
  - User has a query PN/userID, not the index of the DB record
- Encodes a sparse DB as linear combination of its records
  - DB size reduction using small additional client storage
  - Compatible with recursion
  - Ensures minimal noise growth for fully homomorphic encryption
- Performance
  - 2x reduction in response size
  - 2x reduction in response overhead for batch PIR
Google Keyword PIR framework

For client storage:
- Shard hash key K
- Partition hash keys (3)
- Partition boundaries $B_0, ..., B_b$

Offline: download hash keys, $B_0, ..., B_b$
Online: keyword queries

$k_7 = ?$

query $q$, $|q| \leq 8$

response $r = E(v_7)$
Cost estimates

Assumptions:
- 10BN records
- Size 1.28 TB
- 10k shards -> 1M records each

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Cost estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIR Public Key Size Per Device (storage required)</td>
<td>14 MB</td>
</tr>
<tr>
<td>Upload Bandwidth Per Query</td>
<td>14 KB</td>
</tr>
<tr>
<td>Download Bandwidth Per Query</td>
<td>21 KB</td>
</tr>
<tr>
<td>Client Time Per Query</td>
<td>0.1s</td>
</tr>
<tr>
<td>Server Time Per Query (Single Thread)</td>
<td>0.8-1s</td>
</tr>
</tbody>
</table>
Questions
Cross-service identity spoofing

- Alice messages Bob at Bob's preferred service (bob@Threema)
- Eve messages Alice impersonating Bob using bob@FooService
- Alice needs some indicator or UI to know that bob@Threema isn't bob@FooService and that when bob@FooService messages, it should not be assumed that bob@FooService is bob@Threema.