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. **Endpoint discovery**: discover receiver service IDs to retrieve public key material and send message payload e.g. Platform1.org/send/, Platform1.org/kds

2. **Default service discovery**: Discover optional default receiver service ID user preference for a given PN/UserID (e.g. default:Platform1.org)

3. **Global uniqueness**: Fully-qualified service identifiers should be globally unique

4. **(P1) Key verification**: Provide an independently trusted party to assert and verify the association between a public key and a UserID
Privacy Requirements

1. **Social graph**: Resolver or discovery services should not learn the PN/UserID a client is querying for (i.e. who is sending a message to who)

2. **Querying user identity**: A resolver service should not default to sharing the querying user identity with other resolver services when it requires their help for discovery

3. **Metadata**: Resolver service should not learn the exact timing of when a message is sent
1. **Hiding service reachability:** (the link between a UserID and a service). E.g. +16501234567, reachable on Messages
   a. All major E2EE messaging services already publish unACL’d reachability information without opt-out, Whatsapp, Telegram (not including name or any other info)

2. **Hiding the value of UserIDs or public keys:** e.g. the existence of the PN, +16501234567

3. **Hiding the association between a public key and a UserID:** e.g. PN +16501234567 has pubkey x

4. **Contact lookup by name** (or anything except username)
Other non-functional requirements

1. No single entity should be financially responsible for resolving all identity queries (e.g. even within a geographical region)
2. Costs for each participating entity of storing and querying key records should be proportional to their number of participating users.
3. Performance should support each client querying each of their contacts at least once every 24 hours
Registration Phase

Alice → KDS ← Bob
PubInfAlice → PubInfBob

Session Setup

Alice → KDS ← Bob
Request for PubInfBob → PubInfBob

Symmetric-ratchet

Alice → Bob
Symmetric key derivation

Asymmetric-ratchet

Alice ← Bob
Message + new key

calculation of shared key
calculation of shared key
Option 1: brute force query - too expensive and leaks private info

Client

Registration of UserID + service ID.

Pubkey bundle request

Messaging service KDS 1
Messaging service KDS 2
Messaging service KDS 3
Option 2: Centralized hub - expensive and organizationally complex

Client

Public key registration, retrieval

Centralized KDS

Message delivery hub

Messaging service 1

Messaging service 2

Messaging service 3
Option 3 (Preferred): Federated with KDS resolver service

- **Client**
- **KDS resolver**
  - Resolution of UserID to service ID(s)
  - Pubkey bundle request
  - Registration of UserID + service ID
  - Messaging service 1 KDS
  - Messaging service 2 KDS
  - Messaging service 3 KDS
Key distribution

1. Platform1 Client sends a PIR Query PN/UserID.
2. Platform1 Front End receives the query and responds with Supported service IDs + default service.
3. Platform1 Name Server requests Platform2 Name Records.
5. Platform1 Name Server queries PN/UserID and returns the Public Key Bundle.
6. Platform1 Front End encrypts the message.
7. Platform1 Client sends the encrypted message via messaging providers.
8. Platform2 KDS and Platform2 Client receive and process the message.
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

\[ a + b = a + b \]

\[ a \times b = a \times b \]

(Note encrypted)
Private Information Retrieval

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

I want $M[4] = ?$

$4 \Rightarrow 00010.$

$\sum_{d}$

$M = \begin{bmatrix}
1 & a \\
2 & b \\
3 & c \\
4 & d \\
5 & e \\
\end{bmatrix}$

$\begin{bmatrix}
0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}$

$d$
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
Keyword PIR framework

Dense DB (index-based)  Sparse DB (keyword-based)  Shard to sub-DBs  Partition & encode for recursion

1. v_1  k_1  v_1
2. v_2  k_2  v_2
...  ...  ...
 n. v_n  k_n  v_n

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

k_7 = ?

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

query q, |q| ≤ 8
response r = E(v_7)

E(0)  E(0)  E(1)  E(0)
E(0)  v_1  v_2  v_3  v_4
E(1)  v_5  v_6  v_7  v_8
E(0)  v_9  v_{10}  v_{11}  v_{12}
E(0)  v_{13}  v_{14}  v_{15}  v_{16}
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.