Mutual Contact Discovery

Jaap-Henk Hoepman

Privacy & Identity Lab
iHub
Radboud University
Karlstad University
University of Groningen

jhh@cs.ru.nl // www.cs.ru.nl/~jhh // blog.xot.nl // @xotoxot

Contact discovery: preliminaries

- There is an underlying, existing, social graph \((V, E)\)
  - Unique identifiers \(A, B, \ldots\)
    (e.g., phone numbers).
  - Low entropy (enumerable; guessable).
  - User/device \(A\) maintains contact list

\[
\text{contacts}_A = \{B \in V \mid (A, B) \in E\}
\]

Contact discovery: goal

- Users join a new (messaging) service
  - And what to learn who is also a member of the service

- Requirement
  - If \(B \in \text{contacts}_A\) then \(A\) gets notified that \(B \in \mathcal{U}\) when \(B\) joins (or when \(A\) adds \(B\) to \(\text{contacts}_A\) when \(B \in \mathcal{U}\) already).
The classic (WhatsApp) approach

Matching takes place here.

Contacts stored for later matching, also for "non-users".

Privacy issue

- Matching service learns all contacts of a user on the underlying social graph, including the identifiers of "non users".

Solutions

- Hashing
  - Store the contacts as hashes on the matching database.
  - Improvement: hash the contactlist before sending it to the matching service.
  - Even better: use a key derivation function (KDF).

- Problem
  - NL: $2^{24}$ phone numbers.
  - Dictionary easily computed in seconds, and storable on disk.

- Trusted hardware (Signal)
  - Use hashing like above, but
  - Run matching code on server in trusted hardware.

- Private Set Intersection
  - To compute $\mathcal{U} \cap \text{contacts}_A$.
  - Problem
    - Expensive.
    - Needs to be done for every user whenever a new user joins.

Still some privacy issues remain

- Not mutual
  - If $A \in \text{contacts}_A$ then $B$ gets notified that $A \in \mathcal{U}$ even if $B \notin \text{contacts}_A$.

- This is problematic
  - Ex-es, former bosses, doctors.
Mutual contact discovery: requirements

- **Correctness**
  - Output $\text{out}_A$
  - $B \in \text{out}_4$ if ($B \notin \mathcal{U}$) and ($A \in \text{contact}_B$) and ($B \in \text{contact}_B$).

- **Security**
  - Let $B \in \text{contact}_A$, if $B \notin \mathcal{U}$ then $X$ cannot force $B \notin \text{out}_4$.
  - ($X$ stands for the matching server or a user respectively.)

- **Membership privacy**
  - If $X \notin \text{contact}_A$, then $X$ does not learn whether $A \in \mathcal{U}$ (for any $A \neq X$ of its choosing).

- **Contact privacy**
  - $X$ does not learn whether $B \in \text{contact}_B$ (for any $A, B$ both unequal to $X$ of its choosing).

Threat model

- **Active adversary**
  - May behave arbitrarily.
  - May block and observe messages (as channels are secure, eavesdropping, replaying or modifying messages is prevented).

- **May use prior knowledge to maximise chance of success**
  - Knows list of identifiers in use,
  - Knows identifiers for persons of interest, and
  - May have knowledge of potential contacts.

Protocol #1

- **Notation**
  - Time divided into slots $T$, starting at 0.
  - Epoch $T = T \div 2$.
  - Token $v_{AB} = KDF(T | A | B)$.

- **Submission phase, even epoch**
  - Each user $B$ sends $v_{AB}$ to the server, for all $A \in \text{contact}_B$.
  - The server stores these values in $S$.

- **Query phase, odd epoch**
  - Each $B$ now sends $v_{BA}$ to the server, for all $A \in \text{contact}_B$.
  - The server returns whether $v_{BA} \in S$.
  - $B$ adds $A$ to $\text{out}_A$.

- **After every odd epoch**
  - The server deletes $S$.

Protocol #1 example
Key derivation function

- Like a (cryptographic) hash function
  - Takes a ‘significant’ time to compute (to make constructing a dictionary expensive).
- Hashing single phone numbers offers no significant protection
  - Small dictionary can be computed in seconds
- But, what about the concatenation of two phone numbers
  - NL: 2 phonenumbers.
  - Input to KDF then 48 bits; dictionary 2^48 - 32 TB.
  - Assume average customer hardware 2^20 times slower than attacker computing power.
  - If average contact list contains 2^8 (hundreds) of contacts, then work for attacker compared to work for user is factor 2^28 / (2^20 * 2^8) = 2^20 higher.

Analysis

- Correctness
  - Yes
- Security
  - Server: can always reply yes in query phase.
  - User: can guess A, B such that B ∈ contacts_A, then compute v_A and send it in the submission phase. A will query this value in the query phase and (wrongly) conclude B ∈ out_A

A more formal treatment

- We have social graph (V, E)
  - Stored by clients: contact_A = {B ∈ V | {A, B} ∈ E}
  - Split in visible_A + hidden_A
  - hidden_A = ∅ for honest A
- Define A ⇔ B if
  - (A ∈ visible_B) ∧ (B ∈ contacts_A)
- Is only symmetric for honest users
- Correctness
  - B ∈ out_A if B ∈ U ∧ A ⇔ B
The ideal model

Protocol #2: using a key server

1. Submission phase

2. Query phase

Definitions

- Let $e: G_1 \times G_1 \rightarrow G_2$ be a pairing function: $e(aP, bQ) = e(P, Q)^b$ for any points $P, Q$
- Let $H_1: V \rightarrow G_1$ be a cryptographic hash function
- Define $P_k = H_1(P)$
- Let $s$ be a secret of the certificate authority
- Define the certificate for $A$ as $C(A) = sP_k$

- Define a token $T_{ab} = e(C(A), P_k)$
- Then $T_{ab} = T_{ba}$
- Only $A$ and $B$ can create it
- Define another cryptographic hash function $H_2: G_2 \times G_1 \rightarrow \{0, 1\}^n$
- Let $< \!$ be a total order over $G_1$
- Define $H_2^s(X, Y, Z) = \begin{cases} H_2(X, Y, Z) & \text{if } Y < Z \\ H_2(X, Z, Y) & \text{otherwise} \end{cases}$
- Then $H_2^s(X, Y, Z) = H_2^s(X, Z, Y)$

Certificate authority

Certificate authority

Certificate authority
Analysis

Sketch
- Only X knows $C(X)$
- So, only A and B can construct $T_{AB}$, by Bilinear Diffie-Hellman (BDH) problem
- A can pretend that B in contacts$ _B$, but this is modelled as if $B \in$ contacts$ _A$
- $T_{AB}$ only sent as $H_2^X(T_{AB}, X, Y)$, so no other party learns it
- So, meaningful $H_2^X(T_{AB}, X, Y)$ can only be constructed by A or B
- B adds A to out$ _A$ only if it receives $H_2^Y(T_{AB}, P_A, P_B), H_2^Y(T_{AB}, P_B, P_A)$
  - which is/can only be constructed by A
- The server can prevent A from adding B to out$ _A$ by not sending $H_2^Y(T_{AB}, P_A, P_B), H_2^Y(T_{AB}, P_A, P_B)$ but this is modelled as $A \in hidden _A$

Generalise to dynamic setting

Make asynchronous
- Note how in query phase members submit the same information as in the submission phase
- Therefore, omit submission phase
- Members only execute the query phase, regularly
- Query tuples added to database
- Responses sent when a match is detected

Support deletion
- Add delete command, sending same tuple as in query command
- Server is supposed to honestly delete the tuple from its database

Discussion

[Monty Python’s Argument Clinic sketch]