Proving the Security of the Mini-APP Private Information Retrieval Protocol in EasyCrypt

Alley Stoughton

Dagstuhl Seminar on *The Synergy Between Programming Languages and Cryptography*
November 30–December 5, 2014

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

This work is sponsored by the Director of National Intelligence under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author, and are not necessarily endorsed by the United States Government.

Mini-APP Protocol

- Mini-APP is a three-party private information retrieval (PIR) protocol
- It's my simplification of a protocol developed by cryptographers at the University of California, Irvine, as part of IARPA's APP (Advanced Privacy Protection) program
- Mini-APP features a very simple kind of database

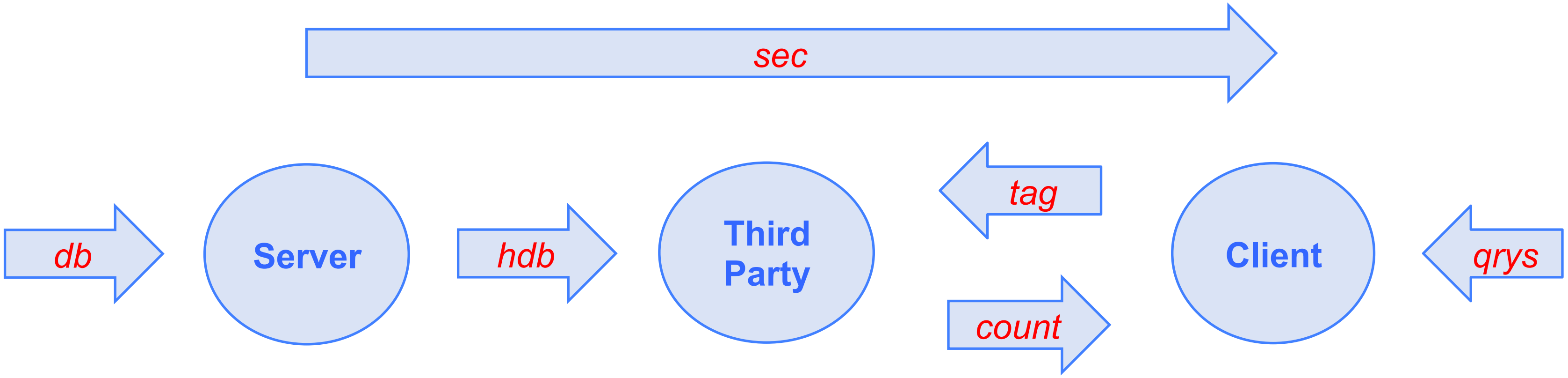
Mini-APP Protocol

- A *database* is one-dimensional: it consists of a list of *attributes*
- Each *query* is also an attribute—a request for the *count* of the number of times it occurs in the database
- The Client should only learn the counts for its queries
  - E.g., it shouldn't learn the order of the database, or counts of attributes it doesn't ask about
- The Server shouldn't learn what queries the Client makes
- To make this work, the protocol uses an untrusted Third Party, which should learn nothing about the database and queries other than *patterns*

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Mini-APP Protocol

- The Server randomly generates a secret, *sec*, and shares it with the Client
- The Server turns its database, *db*, into a hashed database, *hdb*, and sends *hdb* to the Third Party (TP)
  - Each attribute, *attr*, of *db* is turned into the hash of (*attr*, *sec*)
- For each query, *qry*, the Client hashes (*qry*, *sec*), and asks the TP for the number of occurrences of this hash tag in *hdb*

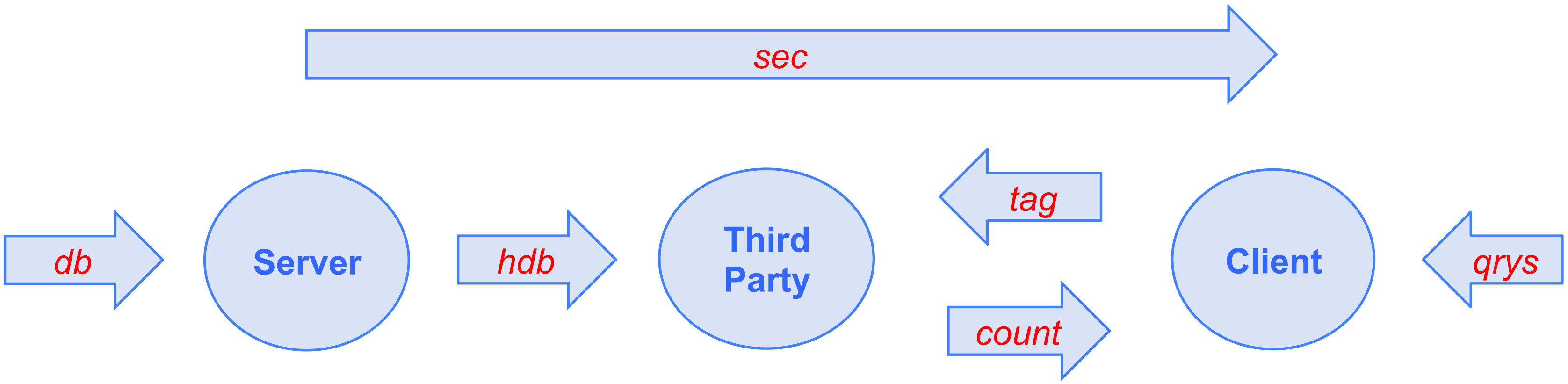


Secrets and Random Oracle

- A secret is a bit string of length *secLen*
- A hash tag is a bit string of length *tagLen*
- Hashing is done using a random oracle, consisting of a map to which new elements are added, dynamically
  - Attribute/secret pairs are mapped to hash tags
  - Stand-in for collision-avoiding hash function
- If a hash collision occurs, the Client's results may be inflated
  - E.g., if the database consists of attributes *x* and *y*, but (*x*, *sec*) and (*y*, *sec*) hash to the same hash tag, then the count for query *x* will be 2 not 1
  - But—under reasonable assumptions—hash collisions will be very unlikely

Protocol Security

- Informally, from an honest but curious perspective:
  - The protocol is secure against the Server, as the Server doesn't receive anything from the TP or Client
  - The protocol is secure against the Client, because the Client only learns the counts of the queries it makes
  - The protocol is secure against the TP, because hashing isn't efficiently invertible, and the TP will be very unlikely to guess *sec*—so the TP will only learn attribute *patterns*



Real and Ideal Games

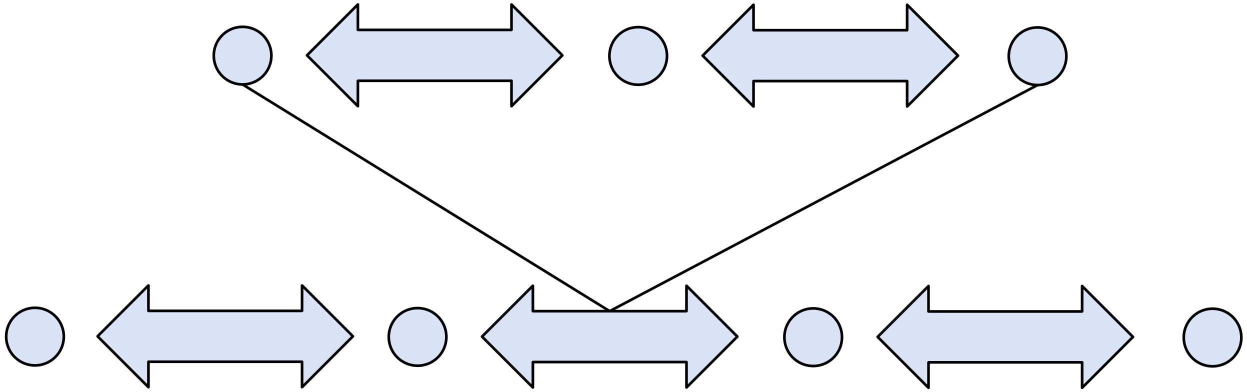
- **We formalize security of the protocol using pairs of cryptographic games—one pair for each protocol party (Server, Client, Third Party (TP))**
- **The "real" games are based on the protocol as described above**
  - **Everything the party sees is recorded in its *view***
- **The "ideal" game for a given party is based on a variant of the protocol in which it is obvious the party doesn't learn anything it shouldn't**
  - **The party's view must still be constructed**

Adversary Model

- Our games are parameterized by an Adversary with access to the random oracle
- Both real and ideal games begin with an initial call to the Adversary in which the Adversary picks a database and list of queries (possibly hashing various attributes in the process)
- Both games end with a final call to the Adversary, in which the Adversary is called with the view produced by running the protocol. The Adversary returns a boolean judgment (possibly hashing various attributes in the process), which is returned as the result of the game
- Adversary allowed to maintain state between its calls
- The protocol is said to be *secure against* the given party iff the Adversary can't distinguish the real and ideal games, i.e., the probabilities of the games returning true differ by a negligible amount

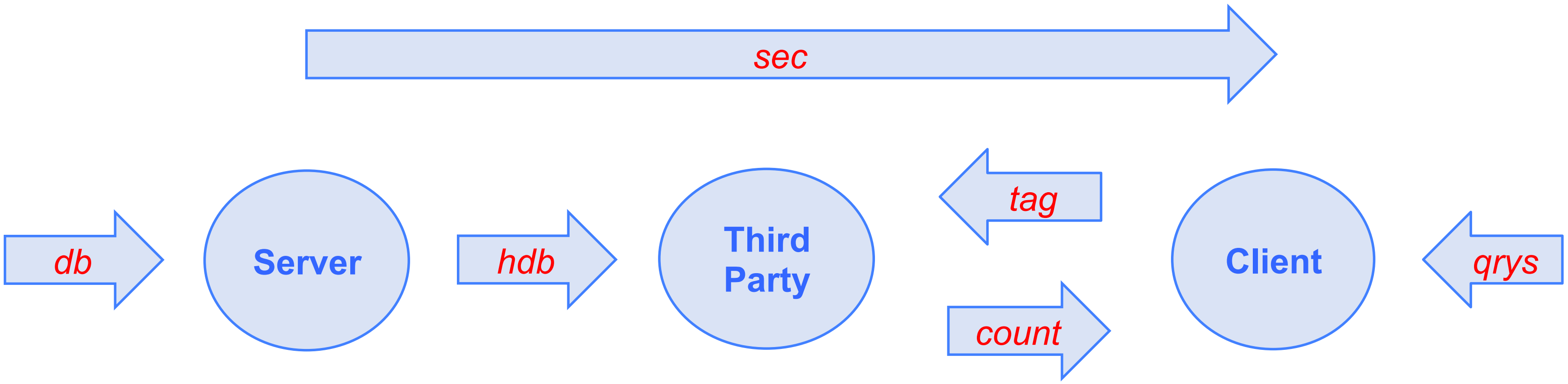
EasyCrypt Proof

- **~300 lines of definitions, game descriptions, top-level lemmas**
- **~4,900 lines of proof script**
- **Proofs make heavy use of abstraction—reusable lemmas**
- **Proofs are both *horizontal* and *vertical***
  - ***Horizontal*: sequences of games, connecting real and ideal games**
    - **In each step, we upper-bound the distance between the games**
  - ***Vertical*: reductions**



Security Against Server

- In the Server's ideal game, the Client and TP are absent
  - Nothing is done with *qrys*—in particular, the queries aren't hashed
- Because the Server generates *sec*, its choice of *db* may be influenced by *sec*
- Consequently, the initial call to the Adversary is given *sec*, so its choice of *db* and *qrys* may be a function of *sec*—giving us a strong security guarantee



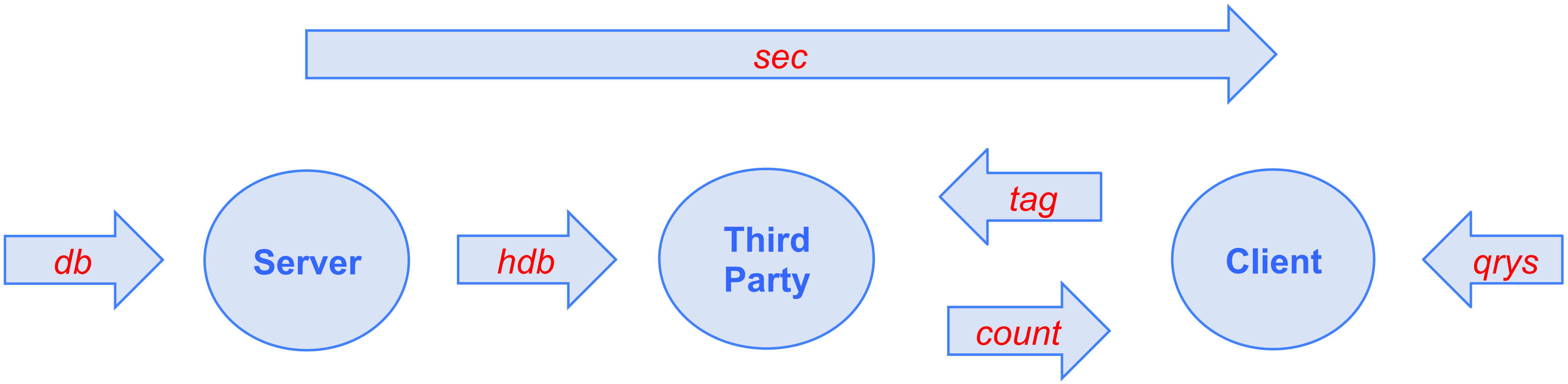
LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Security Against Server

- We prove that the Adversary is equally likely to return true in the two games
- In the sequence of games connecting the real and ideal games, we must get rid of the hashing done by the Client
  - Because we're using a random oracle, this takes some work
  - Proved reusable lemma for removing redundant hashing—using EasyCrypt's eager sampling tactics in a novel way

Security Against Client

- In the Client's ideal game, the TP is absent, and the Server only generates *sec* and gives it to the Client
- Instead, Client is given map containing counts of (only) its queries
  - It consults this map instead of interacting with TP—and thus its counts are never inflated
  - But it still hashes query/*sec* pairs, as the resulting hash tags go in its view



Security Against Client

- Because Client receives *sec*, it could influence its choice of *qrys*
- Server generates *sec*, and could let *sec* influence its choice of *db*
- So for strong security guarantee, we supply *sec* to initial call to Adversary
- If Adversary can do unlimited hashing, it can find and exploit collision
  - It can find distinct $attr_1$, $attr_2$ such that hashing ($attr_1$, *sec*) and ($attr_2$, *sec*) give same tag
  - Then *db* = [$attr_1$], *qrys* = [$attr_2$] will allow it to distinguish real and ideal games

Security Against Client

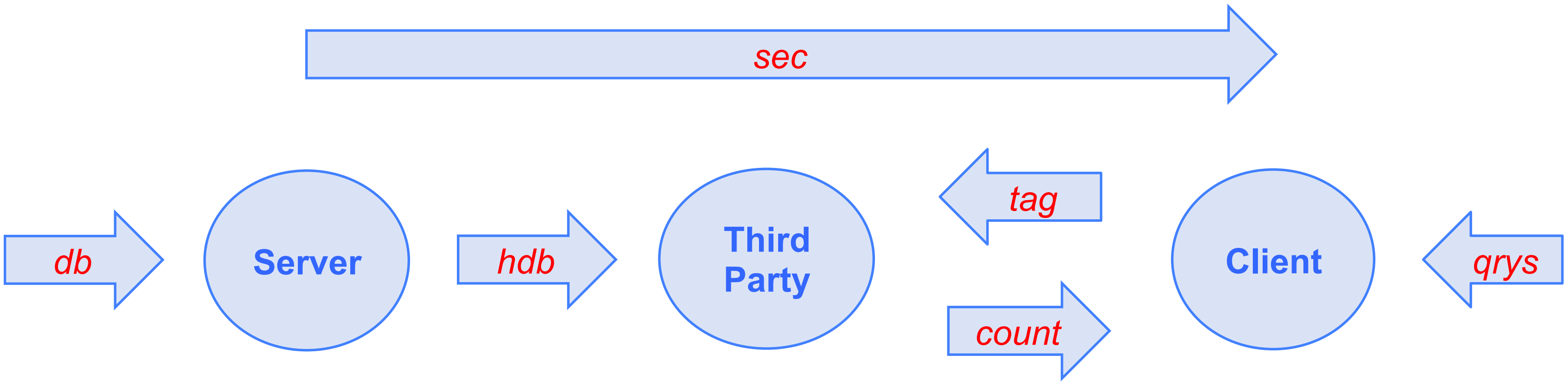
- If Adversary can pick *db* and *qrys* of arbitrary size, it can let *db* = *qrys* = list of distinct attributes of length > number of hash tags
  – Then all query counts in ideal game will be 1, but at least one count in real game will be more than 1
- Consequently, we give Adversary a hashing budget, *budget*, that is ≤ number of hash tags
- If |*db*| + |*qrys*| + hashes done by Adversary in initial call isn't ≤ *budget*, Adversary loses game (is given empty view)
- We prove that the absolute value of the difference between the probabilities that the Adversary returns true in the real and ideal games is upper-bounded by $(\mathit{budget} \times (\mathit{budget} - 1)) / 2^{\mathit{tagLen}}$

Security Against Client

- **In the sequence of games connecting the real and ideal games, we transition in and out of games in which the random oracle stays collision-free as long as no more than *budget* distinct hashes are done**
  - **The Adversary's final call uses collision-possible hashing—at that point, it can't learn anything through a collision happening**
  - **Our Switching Lemma is proved using intermediate games**
  - **Distance between games of Switching Lemma upper-bounded by $(budget \times (budget - 1)) / 2^{(tagLen + 1)}$**
- **The transition from relying on the TP for counting occurrences of a query's tag to looking the query's count up in supplied map involves using a complex relational invariant**
  - **Must prove that hashing budget is respected by Server/Client**
- **Before reaching ideal game, must get rid of hashing done by Server—we reuse lemma for removing redundant hashing**

Security Against Third Party

- In the **TP**'s ideal game, **Server** and **Client** do their attribute hashing using a **private** random oracle—for hashing attributes, not attribute/secret pairs
- The **Adversary** can't be given ***sec***, since otherwise it could differentiate real and ideal games
  - See if ***sec*** paired with attributes of ***db*** hashes to tags of hashed database part of view—very unlikely to happen in ideal game



Security Against Third Party

- The Adversary's choice of *db*/*qrys* can't be allowed to depend on *sec*—even if state after generation of *db*/*qrys* doesn't carry over to final call of Adversary
  - *db* could encode the secret: $attr_0$, $attr_0$, $attr_1$, …, for standard pair of attributes, $attr_0$ and $attr_1$ (using extra, leading 0)
  - Then database and secret will be recoverable from the hashed database of the view
- The Adversary can search for value of *sec* that correlates with *db*, *qrys*, the tags of the view, and the (non-private) random oracle
  - Will succeed in real game, but unlikely to succeed in ideal game
- Consequently, we must limit the number of distinct calls the Adversary may make to the random oracle to some limit, *limit*
  - Subsequently, a dummy value is returned, but previously-hashed attribute/secret pairs hash as before

Security Against Third Party

- **We prove that the absolute value of the difference between the probabilities that the Adversary returns true in the real and ideal games is upper-bounded by *limit* / 2^{secLen}**

Security Against Third Party

- We use a reduction to a lemma upper-bounding the distance between games in which an adversary is given access to two versions of a secrecy random oracle:
  - An initialization procedure init, which is given randomly chosen *sec*; adversary not given *sec*
  - A procedure lhash for hashing up to *limit* distinct attribute/secret pairs—excess inputs yield dummy value
  - A procedure hash for hashing attributes
- In first oracle implementation, hashing an attribute *attr* using hash hashes (*attr*, *sec*) in map used by lhash
- In second oracle implementation, hash's map is independent from lhash's map

Security Against Third Party

- If Adversary never calls lhash with (*attr*, *sec*), for some *attr*, then it won't be able to tell the games apart
- Consequently, we are able to upper-bound the distance between the games by *limit* / 2^{secLen}

Security Against Third Party

- Lemma bounding distance between games involving secrecy random oracles is reduced to lemma about secret guessing game, involving a guessing oracle with
  - An initialization procedure init taking in a randomly chosen *sec*
  - A guessing procedure guess lets adversary try to guess secret—but it doesn't learn if it succeeded
- Adversary allowed *limit* calls to guess—after which it's a noop
- We are able to upper-bound probability that adversary guesses secret by $limit\ /\ 2^{secLen}$
  - Uses EasyCrypt's Probabilistic Hoare Logic

Next Steps

- **Now ready to carry out security proof of UCI APP Protocol**
  - **Previous attempt by team at Naval Research Laboratory and MIT Lincoln Laboratory (including me) had only partial success**
  - **At the time, technique repertoire and EasyCrypt tool not sufficiently mature**
- **Plan to implement a tool helping one synthesize and maintain consistency between the real and ideal games of each of the parties of a protocol**