Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection

Peter Scholl (University of Bristol)
Michele Orrù (ENS Paris)
Emmanuela Orsini (University of Bristol)
Motivation: Private Set Intersection

No-fly lists

Private contact discovery
Blueprint for private set intersection protocols

1-out-of-n oblivious transfer

From [PSZ14,PSSZ15]
1-out-of-2 Oblivious Transfer

\[ x_b \] \quad \text{for} \quad b \in \{0, 1\}

\[ x_0, x_1 \in \{0, 1\}^k \]
1-out-of-n Oblivious Transfer

$\forall b \in \{0, \ldots, n-1\} \quad x_b \in \{0, \ldots, n-1\}$

$\forall x_0, \ldots, x_{n-1} \in \{0,1\}^k$
OT requires public-key primitives, so is inherently expensive

Obf. >> FHE >> PKE >> SKE >> OTP

Less efficient  More efficient
OT extension creates many OTs from just a few, at very low cost

k seed OTs

Symmetric crypto

m OTs

k=128

m = 1 billion
Warm-up: OT on long strings is easy

1-2 OT

- First transfer a key
- Then encrypt long messages under the keys
[IKNP03] OT extension: “Turn your head”
IKNP: a few OTs on long strings $\Leftrightarrow$ many OTs on short strings, with sender/receiver reversed

\[ y = x + b \cdot c \]

\[ 1 \times x \]

1-2 OT

\[ x = x + c \]
IKNP: a few OTs on long strings $\Leftrightarrow$ many OTs on short strings, with sender/receiver reversed

$k \times 1$-2 OT

Sender’s strings are correlated by $c$
IKNP: a few OTs on long strings $\Leftrightarrow$ many OTs on short strings, with sender/receiver reversed.

\[ k \] \[ Y \] = \[ k \] \[ X \] + \[ b_1 \] \[ c \] \[ \vdots \] \[ b_k \] \[ c \] \[ m \]
IKNP: a few OTs on long strings $\Leftrightarrow$ many OTs on short strings, with sender/receiver reversed
IKNP: a few OTs on long strings $\Leftrightarrow$ many OTs on short strings, with sender/receiver reversed
IKNP: a few OTs on long strings $\Leftrightarrow$ many OTs on short strings, with sender/receiver reversed
1-out-of-n OT extension, with active security
[KK13] gives you passive security

Replace IKNP correlation:

• Sender has an XOR share of each row of C
• Active adversary: how to guarantee correct encodings?

\[ c_1 = \text{Encode}(x_i), \text{ with error-correcting code} \]
New consistency check for the receiver’s codewords

Sender sends challenge:

Receiver opens both shares of:

Sender checks each column is a codeword
Intuition behind the consistency check

- Takes $k$ random linear combinations of $C$ over $F_2$.
- Except with prob. $2^{-k}$, receiver gets challenged on every codeword.
- ECC must be a linear code and have a high minimum distance.

Inspired by [FJNT16]
Overhead of active security is very small
Faster private set inclusion from 1-out-of-n OT

- 1-out-of-n OT on random strings => private set inclusion on elements of \{1, ..., n\}
- Previous works: several OTs for 1 PS inclusion on long strings
- **Observation**: our 1-out-of-n random OT works for \(n = O(2^k)\)
- Only need **one OT** for each PS inc.

\[ \in \mathbb{C} ? \]
Conclusion

- 1-out-of-n OT extension is very cheap
  - With active security
  - Even for very large n (on random strings)
  - Around 3x improvement to PSI

- Open problems:
  - Fast, actively secure PSI
  - Fast 1-out-of-2 OT on bits with $O(1)$ communication
Thank you!

Full paper: http://ia.cr/2016/933
Code: https://github.com/mmaker/oos16
Low-Leakage Secure Search for Boolean Expressions

Fernando Krell (Dreamlab Technologies)
Gabriela Ciocarlie (SRI International)
Ashish Gehani (SRI International)
Mariana Raykova (Yale University)
Searchable Encryption

Data Owner

Setup

Enc(D), Enc(Index(D))

Search

Enc(query)

Enc(result)

Output results

Server

Solution 1: Retrieve all documents and compute query locally
Solution 2: Build secure index for efficient search

Issue: Search pattern and access pattern leakage
Delegated Queries

1. Enc(D), Enc(Index)

2. Access params
5. Decryption keys

3. Query token
4. Encrypted Results

Solutions:
- OSPIR-OXT (CCS’13)
- Blind Seer (S&P’14 ’15)
- IKLO (CT-RSA’16)
Leakage

- Leakage allows for efficiency
- What does this leakage means in practice?
- Attacks:
  - IKM NDSS’12
  - LZWT Inf. Sci. ‘14
  - CGPR CCS ’15
  - KKNO CCS ’16

This work: Remove leakage to server for delegated queries.
Oblivious Random Access Machines
- GO J. ACM ‘96
- Outsource memory to server
- Hide access patterns
- Polylog overhead for access
- Polynomial space
Other Tools

1. Set representation via Bloom Filters
   - Bit array
   - Hash functions
   - Set bits pointed by hash of elements

2. Oblivious PRF
Construction Sketch

- Secure Index outsourced to server in ORAM
- Give index key $s$ to server and ORAM access params to client
- Encrypted index bits are retrieved by client
- Client and server perform secure computation to evaluate encrypted index bits against query

What to we gain?
- Server doesn’t learn any access pattern
- Client doesn’t learn partial value of index bits
The Index

- Use Blind-Seer index:
  - Search tree: nodes store searchable keywords of subtree
  - Leaves point to records.
  - Nodes: Bloom filters.
  - In Blind Seer, BF are One-time padded.
  - Use of 2PC to remove pad and evaluate formula.
  - Result is leaked to both client and server.
Our Approach

- Store Bloom filter tree in ORAM
  - Remove leakage to server.
- Give access parameters to client
  - Avoid performing secure computation on all ORAM retrieved elements.
- Encode bloom filter bits such that
  - Client doesn’t learn bits or partial query evaluation.
  - Client only learns result of evaluation
Oblivious Bloom Filter Evaluation

- Bit encoding: \( b \rightarrow \tilde{b} = (g^b \times F_s(r), r) \)
- BF eval: client has \( \tilde{b}_1, \tilde{b}_2, ..., \tilde{b}_h \), server has secret \( s \)
  1. Obtain \( \prod F_s(r_i) \)
  2. Compute \( g^{\sum b_i} \)
  3. Compare against \( g^h \)
- Issues:
  - Client can compare values of different evaluations.
  - Solution: client gets masked PRF value \( \prod F_s(r_i) \times R \)
  - Client can compute \( g^{\sum b_i} \times g^j \), and learn \( \sum b_i \) exactly
  - Solution: Hide generator \( g \) from client.
Bloom filter Evaluation Protocol

\[ i = H_j(query) \mid j \in [h] \]
\[ \tilde{b}_i = \langle g^{b_i} \times F_s(r_i), r_i \rangle \]
Oblivious PRF

**Hashed DH PRF**

\[ X = H(r)\alpha \]

\[ Y = X^s \]

Output \( Y^{\alpha^{-1}} \)

**Multiplicative Masked OPRF**

\[ W = g^\beta \]

\[ X = (W \cdot \prod H(r_i))^{\alpha} \]

\[ Y = X^s \]

Output \( Y^{\alpha^{-1}} \)

\[ \Pi F_s(r_i) \cdot R^{-1} = \Pi H(r_i)^s \cdot g^{\beta \cdot s} \]
Boolean Formulas

- **Conjunctions:** Trivial
  - Single term queries are conjunctions over BF bits
  - Just add ORAM lookups for each term
  - Evaluate as single term case

- **Disjunctions:** \( q = C_1 \lor C_2 \lor \cdots \lor C_{|q|} \)
  - Use set of possible matching values
  - \(|q|\) terms \( \Rightarrow \approx h^{|q|}\) possible values
  - Idea: client obtains

\[
(g \sum b_i \cdot R_1)^{L_1-1} \cdot (g \sum b_j \cdot R_2)^{L_2} \cdot \cdots \cdot (g \sum b_k \cdot R_{|q|})^{L_{|q|}}
\]
Empirical Results 1/2

![Graph showing time (secs) for different scenarios: Single term, 3 Conjunction, 3 Disjunction, 3-DNF 2, and 3-DNF 3. The x-axis represents the scenarios, and the y-axis represents time in seconds. The graph includes bars for $10^3$, $10^4$, and $10^5$.](image)
Empirical Results 2/2

The graph shows the relationship between query size and time (seconds). The y-axis represents time in seconds, ranging from 10 to 10,000. The x-axis represents query size, ranging from 0 to 6.

There are four lines representing different methods:
- Conjunction
- Conjunction-Multithreaded
- Disjunction
- Disjunction-Multithreaded

As the query size increases, the time taken increases significantly, with the Disjunction-Multithreaded method showing the steepest rise.
Conclusion

- Security: Removed all important leakage to server.
- Functionality: Support for small DNF formulas.
- Efficiency: Seconds per record.
- Introduced protocol for Oblivious bloom filter evaluation via oblivious PRFs
Directions for future work

- Protect index access pattern leakage to client.
- Index space is $x \times 10^3$ of plaintext solution.
- Improve efficiency for complex Boolean formulas.
  - Use circuit approach (e.g. YAO's GC).
- Use of specially designed ORAMs to store index (Oblivious data structures WNLCSSH CCS ’14, KS ASIACRYPT ‘14).
- Include robust access control for queries.