# Yale University
# Department of Computer Science

## Java Implementation of a Single-Database Computationally Symmetric Private Information Retrieval (cSPIR) protocol

Felipe Saint-Jean [1]

YALEU/DCS/TR-1333
July 28, 2005

[1] This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-04-1-0725.

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*

1. REPORT DATE
**28 JUL 2005**

2. REPORT TYPE

3. DATES COVERED
**00-07-2005 to 00-07-2005**

4. TITLE AND SUBTITLE
**Java Implementation of a Single-Database Computationally Symmetric Private Information Retrieval (cSPIR) protocol**

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
**Yale University,Department of Computer Science,PO Box 208285,New Haven,CT,06520-8285**

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR’S ACRONYM(S)

11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:

| a. REPORT | b. ABSTRACT | c. THIS PAGE | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| **unclassified** | **unclassified** | **unclassified** |  | **12** |  |

# Java Implementation of a Single-Database Computationally Symmetric Private Information Retrieval (cSPIR) protocol

Felipe Saint-Jean *

## 1 Motivation

Picture the following scenario. Alice is looking for gold in California. What Alice does is look for a place with a little gold and follow the trace. Now, Alice wants to find gold in a place where no mining patent has been awarded, but many patents have been awarded in California during the gold rush. What Alice does is to walk around California with a GPS and a notebook computer. Whenever she finds a trace of gold she follows it querying if any patent has been awarded in that location. If she finds a trace of gold in a piece of land with no issued patent she can request the patent and start mining for gold.

The problem is that she is worried that Bob's Mining Patents Inc., the service she queries the patents from, might cheat on her. Because Bob's knows she is looking for gold in California (Alice said so when signing up for Bob's service), he knows that, if she queries from some location, then there is gold there. So, if she queries a location and there is no patent awarded, Bob may run to the patent office and get the mining patent for that location.

Depending on privacy and economic constraints, a few solutions come to mind. Alice might buy from Bob the whole database for California. Alice then can make all the queries to her own database, and Bob will never find out where Alice is looking for gold. But this might be very expensive, because Bob charges per query; what he charges for the whole database will probably be more than what Alice is willing to

*This work was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-04-1-0725.

pay. Alice can also, with each query, perform a collection of fake queries so that Bob can't figure out which is the real query (this leaks information unless she queries the whole database!), but that still makes Alice pay for more queries that she would like.

This Alice-and-Bob scenario is a basic motivation for Private Information Retrieval: a family of two-party protocols in which one of the parties owns a database, and the other wants to query it with certain privacy restrictions and warranties. Since the PIR problem was posed, different approaches to its solution have been pursued. In the following sections, we will present the general ideas of the variations and proposed solutions to the PIR problem. Then, we will present a collection of basic protocols that allow the implementation of a general-purpose PIR protocol. Finally, we will show details of a particular PIR protocol we have implemented.

# 2 Introduction

As mentioned in the motivation, the goals of PIR would be realized if Bob were to send the whole database to Alice. That would be satisfactory if Bob didn't care whether Alice learned more than what she queried. In that situation, the challenge is to devise a protocol that reduces the amount of data Bob has to send in order for Alice to learn the answer to her query without Bob's learning what the query was. In general, that can only be done by having replicated, non-communicating databases.Going back to the Alice-and-Bob scenario, there is not one Bob but a collection of them with identical databases. In this way, the query can be hidden if Alice interacts with all the Bobs in such a way that each Bob is never sure whether the query he receives is the real one. The solutions proposed in this line of work achieve a lower number of queries (sublinear in the database's size) if more replicated Bobs are available. In this kind of solution, Alice's privacy is protected, and the objective is to reduce the number of queries needed. Most of the protocols in this line of work present solutions that are private from an information-theoretic point of view. For example Chor*et al.* [6] show that, if the database is replicated two or more times, sublinear communication complexity can be obtained. Sublinear communication means that, in the execution of the protocol, less that the complete database is transfered. As mentioned above, the whole database is a trivial upper bound on the communication of the PIR problem. In the information-theoretic approach, the protocols are composed of many single-element queries, each taking with a cost of 1, because exactly one element of the database is transfered in each response to each query.

An important improvement in PIR was put forth by Kushilevitz and Ostrovsky [2]; they presented a PIR protocol that requires no replication. Their protocol, based on the hardness of the Quadratic Residuosity problem, is private from a computational complexity point of view; so, to distinguish it from the information-theoretical approach, it is known as cPIR, for "computational PIR". This idea was first considered in [7]. One variation of the standard PIR scenario, in which only Alice's privacy is safeguarded, is the SPIR scenario (Symmetric PIR). In SPIR, we not only care about Bob's not learning anything about Alice's query, but we also want Alice not to learn anything about other entries in Bob's database other than the one she queried. This is very similar to the one-out-of-N Oblivious Transfer problem, and, as we will see later they are closely related.

In this work, we will focus on the implementation of a specific SPIR protocol proposed by Naor and Pinkas [3] that uses Oblivious Transfers as a building block. One of the properties of this protocol is that it requires only one initialization phase for a sequence of queries, thus amortizing the cost of the initialization phase. We will also show a variation of the protocol, proposed by Boneh, that eliminates the initialization phase by introducing a cPIR query as part of the protocol.

# 3 Description of protocols and other tools

In this sections, we present a collection of protocols that are required for the SPIR implementation. In all of them, the Sender (Bob) owns a database, and the Receiver (Alice) wants to get the $i$-th value in this database.

## 3.1 One-out-of-two Oblivious transfer $OT_1^2$

In a one-out-of-two Oblivious Transfer (abbrev. $OT_1^2$), the Sender holds two values. As a result of the protocol, the Receiver learns a value of his choice and nothing about the other. The Sender learns nothing about the choice made by the Receiver. The implementation of $OT_1^2$ that we used is:

**Initialization**: The Sender and Receiver agree on a large prime $q$ and a generator $g$ for $Z_q^*$. In the actual implementation, the Sender generates them and sends them to the Receiver. The pair $(q, g)$ can be used in several transfers. That is because we want the Receiver not to be able to compute the discrete log efficiently and no extra information that enables him to do so is given as part of the protocol. $H(.)$ is a random oracle– in practice, a hash function.

| Receiver | | Sender |
|---|---|---|
| | | Sender chooses a random element $C$ in $Z_q$ and sends it to the receiver<br>$C \leftarrow_r Z_q$ |
| | $\leftarrow C$ | |
| $k \leftarrow_r Z_q^*$ ; Choose $k$ random in $Z_q$<br>Let $PK_\sigma = g^k$ and $PK_{1-\sigma} = \frac{C}{PK_\sigma}$<br>Send $PK_0$ to the Sender | | |
| | $PK_0 \rightarrow$ | |
| | | $PK_1 = \frac{C}{PK_0}$ ; Sender computes $PK_1$<br>$r_0 \leftarrow_r Z_q^*$<br>$r_1 \leftarrow_r Z_q^*$<br>$E_0 =< g^{r_0}, H(PK_0^{r_0}) \oplus M_0 >$ ; Encryption of $M_0$<br>$E_1 =< g^{r_1}, H(PK_1^{r_1}) \oplus M_1 >$ ; Encryption of $M_1$ |
| | $\leftarrow E_0E_1$ | |
| $M_\sigma = H((g^{r_\sigma})^k) \oplus M_\sigma$ | | |

In our implementation, the random oracle $H$ is implemented as a Hash function. Thus, the size of $M_i$ has to be smaller than the size of the output of $H$. We used $sha - 256$; so, $M_i$ can be up to 256 bits.

## 3.2 One out of N Oblivious Transfer $OT_1^N$

One-out-of-N Oblivious Transfer (abbrev. $OT_1^N$) is a generalization of $OT_1^2$. In $OT_1^N$, the Sender holds a list of $N$ elements instead of 2. Here also desired properties are that the Receiver learn only the $i$-th value in the database and that the Sender learn nothing about $i$. Our implemented $OT_1^N$ protocol is the following:

**Initialization**: The Sender holds values $X_1, X_2, ..., X_N$ with $X_i \in \{0,1\}^m$ and $N = 2^l$. The Receiver wants to learn $X_i$.

| Receiver |  | Sender |
|---|---|---|
|  |  | Prepares $l = \lceil log_2 N \rceil$ random pairs of keys $(K_1^0, K_1^1), (K_2^0, K_2^1), ..., (K_l^0, K_l^1)$, where each $K_j^b$ is a $t$-bit key to the pseudorandom function $F_k$.<br>For all $1 \leq I \leq N$, let $(i_1, i_2, ..., i_l)$ be the bits of $I$; compute $Y_I = X_I \oplus \bigoplus_{j=1}^{l} F_{K_j^{i_j}}(I)$<br>Sender and Receiver engage in an $OT_1^2$ for the strings $< K_j^0, K_j^1 >$ with $j = 0, ..., l$. |
| In the $OT_1^2$ picks; $K_j^{i_j}$ to learn $X_I$. |  |  |
|  | $\leftarrow Y_1, .., Y_N$ |  |
| $X_I = Y_I \oplus \bigoplus_{j=1}^{l} F_{K_j^{i_j}}(I)$ |  |  |

The paper by Naor and Pinkas that proposed this protocol [4] is ambiguous in defining the pseudorandom function $F$ as $F_K : \{0,1\}^m \rightarrow \{0,1\}^m$ but using it as $F_k(I)$; where $1 \leq I \leq N$; for our implementation we needed to use a pseudorandom function $F_K : \{0,1\}^l \rightarrow \{0,1\}^m$; where $l = \lceil log_2 N \rceil$.

In relation to the domain of this $OT_1^N$ implementation, $K_j^b$ will be input of a $OT_1^2$, so the $X_i$ have to be the same size as the output of the pseudorandom function used.

## 3.3 PIR

In a PIR protocol, the Sender holds a database of size $N$. The Receiver wants the $i$-th value in this database. As a result of the protocol; the Receiver must learn the $i$-th entry in the database, but the Sender must learn nothing about $i$. In a general PIR protocol, by "learn nothing", we mean that a computationally unbounded Sender can learn nothing about $i$. That means privacy is preserved from an information-theoretic point of view. We mention this kind of protocol for clarity, but it is not used in our implementation. There is no restriction on what the Receiver can learn as a result of the protocol.

## 3.4 cPIR

A cPIR protocol is similar to a PIR protocol. The only difference is that privacy is safeguarded against a polynomially bounded Sender.

## 3.5 Other tools and protocols required in the way

A few additional cryptographic techniques will be needed to implement the SPIR protocol.

### Random Oracle

A random oracle is a protocol-design tool that gives all the parties in the protocol a common source of random bits. In practice, the shared randomness is provided by a cryptographically strong hash function like SHA-1.

### Sum-consistent synthesizer

A sum-consistent synthesizer is a function $S$ for which the following holds:

- $S$ is a pseudo-random synthesizer.
- For every $X, Y$ and $X', Y'$, if $X + Y = X' + Y'$, then $S(X, Y) = S(X', Y')$.

where a pseudo-random synthesizer is basically a pseudorandom function on many variables that is pseudorandom on each one of them. Pseudo-random synthesizers where introduced by Naor and Reingold in [1].

# 4 cSPIR Implementation details

The scenario in which an SPIR protocol is used is similar to that in which a cPIR protocol is used. However, at the end of the execution of an SPIR protocol, the Receiver should have learned nothing about values in the database other than the $i$-th one. Note that SPIR is the most constrained of all PIR variations and that an $OT_1^N$ is a SPIR protocol.

We implemented a variation of version 3 of the protocol presented in [3]. Version 3 of the protocol is not secure, because, after several queries (3 to be exact) it leaks information. The authors of [3] propose a high-cost fix. In our implementation, we lower the cost by modifying a step in the protocol. The original protocol as described in [3] is

**Initialization**: The Sender prepares $2\sqrt{N}$ random keys $(R_1, R_2, ..., R_{\sqrt{N}}, C_1, C_2, ..., C_{\sqrt{N}})$. For every pair $1 \leq i, j \leq \sqrt{N}$, the Sender prepares a commitment $Y_{ij}$ of $X_{ij}$, $Y_{ij} = commit_{K_{ij}}(X_{ij})$. It sends all of the commitments to the Receiver.

If the Receiver wants to learn $X_{ij}$, then the protocol proceeds as follows:

| Receiver | Sender |
|---|---|
| | Choose $r_C$ random in the same space the keys $R$ and $C$ are chosen from. Set $r_R = -r_C$ so $r_C + r_R = 0$<br>Sender and Receiver engage in a $OT_1^{\sqrt{N}}$ protocol for the values $R_1 + r_R, R_2 + r_R, ..., R_{\sqrt{N}} + r_R$ |
| Receiver gets $R_i + r_R$ | |
| | Sender and Receiver engage in a $OT_1^{\sqrt{N}}$ protocol for the values $C_1 + r_C, C_2 + r_C ..., C_{\sqrt{N}} + r_C$ |
| Receiver gets $R_j + r_C$<br>$K_{ij} = S(R_i + r_R, C_j + r_C)$; Receiver opens the commitment $Y_{ij}$ and reveals $X_{ij}$. | |

In the modified version, if the Receiver wants to learn $X_{ij}$, then the protocol proceeds as follows:

| Receiver | Sender |
|---|---|
| | The Sender prepares $2\sqrt{N}$ random keys $(R_1, R_2, ..., R_{\sqrt{N}}, C_1, C_2, ..., C_{\sqrt{N}})$. For every pair $1 \leq i, j \leq \sqrt{N}$, the Sender prepares a commitment $Y_{ij}$ of $X_{ij}$, $Y_{ij} = commit_{K_{ij}}(X_{ij})$ |
| The Receiver makes a PIR query over the $Y$s and learns $Y_{ij}$ | |
| | $r_C \leftarrow_r$ and $r_R = -r_C$; so $r_C + r_R = 0$<br>Sender and Receiver engage in an $OT_1^{\sqrt{N}}$ protocol for the values $R_1 + r_R, R_2 + r_R, ..., R_{\sqrt{N}} + r_R$ |
| Receiver gets $R_i + r_R$ | |
| | Sender and Receiver engage in an $OT_1^{\sqrt{N}}$ protocol for the values $C_1 + r_C, C_2 + r_C ..., C_{\sqrt{N}} + r_C$ |
| Receiver gets $R_j + r_C$<br>$K_{ij} = S(R_i + r_R, C_j + r_C)$ Receiver opens the commitment $Y_{ij}$ to obtain $X_{ij}$. | |

The main difference is that the original protocol requires $\Omega(n)$ initialization network traffic to send the commitments. In the modified version, that is replaced by a PIR query. One can randomize the commitments in each step to solve the information-leakage problem. The price of this is $O(N)$ local computation, which is better for overall protocol efficiency than $\Omega(N)$ communication.

The *commit* function was implemented with the symmetric cryptosystem $AES$; $commit_{K_{ij}}(X_{ij}) = AES - ENC_{K_{ij}}(X_{ij})$. The Sum-consistent synthesizer $S$ was implemented with the hash function $sha256$, $S(A, B) = sha256(A + B)$.

# 5 Network Layer

All of these protocols involve two-party computations. To implement them, we needed a network layer. Because the implementation was done in Java, a natural choice would have been RMI. The problem with RMI is that it does not fit well with the kind of message-driven way protocols are usually described. That means that using

RMI forces the code to be structured differently from the way protocols are specified. That doesn't seem too important at first glance, but it makes a big difference when checking and going through code. It is much easier to go over a piece of code that looks like the protocol written in the original paper. For that, reason our network layer was implemented by means of messages. Another important feature of the network layer is that it requires support for nested calls. The SPIR implementation is built upon other two-party protocols; so we needed to have persistent connection in all the nested protocol calls. For a simple solution that combined both requirements, we implemented the NetworkBroker class. The NetworkBroker class is a symmetric class written over TCP sockets that allows the sending of serializable objects over a network in a way that is very natural for implementing protocols involving two parties. Here is a short code sample showing an object message passing through a pair of threads:

```
public void testSimpleCase() throws IOException, ClassNotFoundException{
    int port = 40000;
    final NetObjectBrokerServer server = new NetObjectBrokerServer(port);
    final Integer sc = new Integer(900);
    final Integer cs = new Integer(901);


    Runnable r = new Runnable(){
        public void run() {
            try {
                server.accept();
                System.out.println("Got a connectoon");
                Integer cs1 = (Integer)server.waitObject();
                assertEquals(cs1.intValue(),cs.intValue());
                server.sendObject(sc);
                server.close();
            } catch (Exception e) {
                e.printStackTrace();
                fail();
            }

        }};
    // Server is waiting
    new Thread(r).start();
    System.out.println("Here!!");
    NetObjectBrokerClient client =
       new NetObjectBrokerClient("localhost",port);
    System.out.println("Connected, sending obj");
    client.sendObject(cs);
    Integer sc1= (Integer)client.waitObject();
    assertEquals(sc1.intValue(),sc.intValue());
    client.close();
```

# 6 Conclusions and extensions

The main objective of this work is to see how applicable PIR protocols are in practice. A typical database-oriented application would benefit from a few more features of the query engine. Interesting extensions, from that point of view, would include the ability to query for existence of an entry and the ability to query for string-valued or non-sequential keys. That can be easily implemented if the Sender maintains two tables. More ambitiously, we would like to be able to compute joins in a private way without $NM$ complexity, where $N$ and $M$ are the sizes of the joint tables.

An $OT_1^N$ is an SPIR query. It has the same privacy properties. It would be interesting to identify the cases in which an SPIR query is more efficient than an $OT_1^N$; then in many places where a $OT_1^N$ is done, it might be replaced by a recursive SPIR query.

Many optimizations can be done. One of them is to replace the basic implementation of $OT_1^2$ with a more efficient one.

# References

[1] M.Naor and O. Reingold, “Synthesizers and their application to the parallel construction of pseudo-random function,” *Proc. 36th IEEE Symp. on Foundations of Computer Science,* 1995:170-181.

[2] E. Kushilevitz, R. Ostrovsky, “Replication is not needed: Single database, computationally private information retrieval,” *Proc. 38th IEEE Symp. on Foundations of Computer Science,* 1997:364-373.

[3] M. Naor, B. Pinkas, “Oblivious Transfer with Adaptive Queries,” *Advances in Cryptology-Crypto'99, Lecture Notes In Computer Science; Vol. 1666* , 1999: 573 - 590

[4] M. Naor, B. Pinkas, “Oblivious Transfer and Polynomial Evaluation,” *Proc. 31st ACM Symp. on Theory of Computing,* 1999: 245 - 254

[5] M. Naor, B. Pinkas, “Efficient Oblivious Transfer Protocols,” *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms,* 2001: 448 - 457

[6] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, “Private Information Retrieval” *Journal of the ACM (JACM),* Volume 45 , Issue 6 , 1998: 965 - 981

[7] B. Chor , N. Gilboa, “Computationally Private Information Retrieval” *Proc. 29th ACM Symp. on Theory of Computing,* 1997:304-313