Subliminal Channels in the Private Information Retrieval Protocols

Meredith L. Patterson
The University of Iowa
Department of Computer Science
Iowa City, Iowa
USA
mlpatter@cs.uiowa.edu

Len Sassaman
Katholieke Universiteit Leuven
ESAT-COSIC
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee
Belgium
len.sassaman@esat.kuleuven.be

Abstract

Information-theoretic private information retrieval (PIR) protocols, such as those described by Chor et al. [5], provide a mechanism by which users can retrieve information from a database distributed across multiple servers in such a way that neither the servers nor an outside observer can determine the contents of the data being retrieved. More recent PIR protocols also provide protection against Byzantine servers, such that a user can detect when one or more servers have attempted to tamper with the data he has requested. In some cases (as in the protocols presented by Beimel and Stahl [1]), the user can still recover his data and protect the contents of his query if the number of Byzantine servers is below a certain threshold; this property is referred to as *Byzantine-recovery*.

However, tampering with a user's data is not the only goal a Byzantine server might have. We present a scenario in which an arbitrarily sized coalition of Byzantine servers transforms the userbase of a PIR network into a signaling framework with varying levels of detectability by means of a subliminal channel [11]. We describe several such subliminal channel techniques, illustrate several use-cases for this subliminal channel, and demonstrate its applicability to a wide variety of PIR protocols.

1 Introduction

Since the seminal paper by Chor, Goldreich, Kuzhelevitz and Sudan in 1995 [5], information-theoretic private information retrieval protocols have received considerable attention in the information theory community [2, 3, 1, 7, 12, 9, 8, 13]. However, comparatively little work has been done to evaluate these privacy primitives within the broader scope of a secure privacy system. We examine one consideration that designers of systems based on information-theoretic PIR may need to consider within their threat model: subliminal channels between independent PIR databases.

1.1 Background

There are two primary classes of PIR schemes: *information-theoretic* PIR, and *computational* PIR. In the case of the former, an attacker is unable to learn any information about the user's query, even with unlimited computing power. In the latter, the privacy of the query is preserved only against adversaries restricted to polynomial-time computations. In this paper, we consider information-theoretic PIR protocols exclusively.

2 Examples of Subliminal Channels in Several PIR Schemes

2.1 A simple subliminal channel in a Chor et al. PIR Service

Let us examine the simple XOR-based PIR scheme presented in the seminal paper on PIR [5], and observe the potential for covert communication between nodes in a multiple-server PIR system.

Suppose there are three servers: an honest server A and Byzantine servers B and C in collusion. Out of n buckets, the bucket to be retrieved is bucket b_1. The user sends bit vectors $\vec{v}_A$, $\vec{v}_B$, $\vec{v}_C$ to A, B and C such that $\vec{v}_A \oplus \vec{v}_B \oplus \vec{v}_C = \langle 1\,0\,0\,0\,\ldots \rangle$, e.g.:

A	1	1	1	0
B	0	1	1	1
C	0	0	0	1

Thus A should send the user $b_{A1} \oplus b_{A2} \oplus b_{A3}$, B should send $b_{B2} \oplus b_{B3} \oplus b_{B4}$, and C should send b_{C4}. However, suppose B and C agree to flip the same bit in the vectors they received — in this case, bucket 3:

A	1	1	1	0
B	0	1	0	1
C	0	0	1	1

The XOR of all three vectors, and thus the XOR of the returned data, still unveils the contents of b_1. More than two servers can collude in this fashion, although an odd number of colluding servers must flip bits for at least two buckets (e.g., B and C flip bits for bucket j while C and D flip bits for bucket k).

Observe that if it is desirable for each server to have a uniquely identifying set of bits to flip, then the length of the string describing which bits are flipped and which ones are not must be sufficiently long. In particular:

Observation 1 (Lower bound on flipped-bit string lengths) *If there are n colluding servers, then the length of B_n, the set of bits which an individual server will flip or not flip, is bounded such that $|B_n| > \lceil \lg n \rceil$.*

Proof 1 *First, note that in order to assign a unique bitstring to each server, each string must be at least $\lceil \lg n \rceil$ bits long, otherwise there will be overlap. But it is not enough that the bitstrings be unique: they must also* XOR *to the all-zero string. No server can be assigned an all-zero bitstring (otherwise, that server would be instructed to never flip any bits whether it was transmitting or not, and thus could never transmit anything). Thus there are actually $2^{\lceil \lg n \rceil} - 1$ strings from which to choose if each string is $\lceil \lg n \rceil$ bits long. If n is a power of 2, then each bitstring* must *be at least $(\lceil \lg n \rceil) + 1$ bits long, otherwise at least one string will have to overlap.*

Furthermore, suppose that $n = 2^{\lfloor \lg n \rfloor} + 1$. Define the parity string *for each bit in the set of n bitstrings as $P_m = B_{1m} \cdot B_{2m} \cdot \ldots \cdot B_{nm}$, where m is the position of the bit. For example, the parity strings for the set of all 3-bit strings in increasing order, less the all-zero string, are $\{0001111\}, \{0110011\}, \{1010101\}$. In order for all n strings to* XOR *to 0, the result of* XOR*ing the bits of each parity string must also be 0; this will always be the case when examining the parity strings for all 2^m strings of m bits. There will be $m-1$ parity strings of parity 0 for 2^{m-1} strings (including the all-zero string); adding another string to the set requires the mth bit and thus the mth parity string. In order to not throw off the first $m-1$ parity strings, the first $m-1$ bits of the nth string must be 0. However, an $m-1$-bit string of 0s is already present in the*

$n-1$ strings already selected. Since an m-bit all-zero string cannot be present, both the already-present string and the nth string being added must have a 1 as their mth bit, but then a string has been duplicated. It is further impossible to manipulate the set of m-bit strings such that the m parity strings are all of parity 0.

Through this channel, colluding servers could communicate potentially sensitive information even if the network channel were monitored for such communication. When one server flips its prearranged bit(s) and another does not, the data which the user retrieves consists of more than one bucket XORed together, and thus in the case of non-Byzantine-robust PIR protocols, the user must make another attempt to retrieve his data. (Byzantine-robust PIR systems are discussed below.) Colluding servers can therefore use the user as an oracle which reveals the transmission of a single bit: if the user's request succeeds, a 0 has been transmitted, and if the request fails, a 1 has been transmitted.

One might ask why colluding servers would prearrange a set of bits to flip, as this is an additional step. Indeed, if all servers preserve the original bit-string to communicate a 0 and change a random bit to communicate a 1, the same effect is preserved. However, prearranging the flipped bits provides for some security against outside tampering: if the presence of servers using this channel has been detected by an adversary who subsequently changes the responses of one or more other servers to correct for the responses sent by the colluding servers, it will be easier for the colluding servers to detect this if they already know what variations to expect from the other parties on the subliminal channel. In effect, the search space for the origin of an error decreases significantly when the set of flipped bits is prearranged. Later, when we address the multi-party use case, we will also demonstrate how a prearranged set can quickly pinpoint the identity of a malfunctioning or inoperative server within the channel participants.

2.1.1 A simple use-case: the dead-man switch

Suppose that N individuals living in a police state are unable to communicate regularly without fear of letting the authorities know that they are in communication. They all wish to keep the others apprised of their safety, so they agree to announce their status to one another every day by means of a dead-man switch over the subliminal channel described above. Each participant runs a PIR database server modified such that flipping bits is a manual operation (i.e., requires some approval input from the server's operator in order to occur). Suppose that one of the participants is then picked up by the secret police. He will be unable to approve any subsequent bit-flips—in effect, his hand has fallen off the switch—so all subsequent PIR queries will fail and the other participants will know that something has happened to one of their number.

However, if multiple colluding servers attempt to transmit simultaneously, they are guaranteed to fail in the case of a two-party collusion. If A and B both attempt to signal each other by not flipping their prearranged bits, then the user's original set of bit vectors is used unmodified and the user's request succeeds. When simultaneous transmissions occur in multi-party collusions, transmission failure is not guaranteed—if A and C are not supposed to flip the same bits, and both fail to perform the bit-flip, then the PIR request fails as planned—but it is also impossible to determine who initiated the transmission. Thus, we extend this approach to remove the possibility of collision and enable colluding servers to identify which party is sending a signal.

2.2 A multi-party subliminal channel

In addition to choosing a set of bits to flip, let each colluding server S_i also choose a particular user M_i from the userbase of the PIR system. We refer to M_i as the *mark*

for S_i, and every $S \in \{S \setminus S_i\}$ as the *shills* for S_i. Server S_i will always perform its bit-flip operation for every $M_j \in \{M \setminus M_i\}$, and will fail to flip a bit in its mark's request when S_i wants to transmit a 1. S_i's shills receive S_i's signal by observing M_i's behaviour, as in the single-user case.

Now, instead of a simple on-off signal shared by all participants in the channel, each server controls a stream of bits. Note, however, that if one participant suddenly drops out of the channel ("falls off the dead-man switch", as above), then *all* streams will produce failing PIR requests, and it will not immediately be evident which server has dropped out. We address this problem with a systematic approach for constructing the bit-strings and rotating their usage so that this failure mode can be detected. We assume an *atomic broadcast* paradigm for PIR requests; that is, for a set of servers S and a set of marks M, every server receives the requests from the marks in the same order.

2.2.1 Failure detection in the multi-party model

Note that the *entire* set of flipped bits must XOR to the all-zero string so that a normal PIR transaction is not interrupted. When one colluding server stops communicating, the XOR of the remaining participants' strings will contain some 1s and all subsequent transactions will fail. Since each server has a mark allocated to it, however, a server can easily identify that queries for which it did not intend to transmit a 1 are failing anyway. Thus, we design the sets of bits to flip such that the bitstrings describing these sets can be systematically truncated and rotated through in order to identify the failed server. Recall from Observation 1 that for n servers, each bitstring must be at least $\lceil \lg n \rceil + 1$ bits long. Again, an atomic broadcast paradigm is assumed.

Example 1 (A three-party approach) *For servers A, B and C, assign the initial set of strings describing which bits to flip as follows:*

A	*1*	*1*	*0*
B	*1*	*0*	*1*
C	*0*	*1*	*1*

(In this example, buckets beyond the third are disregarded.)

For the first PIR *query that the servers receive, each uses the bit-string that was first assigned to it, e.g., server A uses bit-string A. For the second query, each server uses the bit-string assigned to the next server in the list, e.g., server A uses bit-string B. For the third query, each server uses the following bit-string (server A uses string C); for the fourth, the rotation returns to the top of the queue; and so on and so forth.*

Now, suppose that server B fails. In order to initiate the failure-detection process, each of the remaining servers must be aware that a failure has occurred somewhere on the channel (i.e., each server must notice that it transmitted a 1 when it did not intend to) and *they must recognise that the other servers are aware of the failure, so that they can synchronize their efforts. As soon as a server recognises that a failure has occurred, it immediately stops flipping any bits. So long as any servers continue to flip bits, however, all transactions will continue to fail. Once A and C have both recognised this, subsequent queries will succeed; thus, once the marks assigned to A, B and C have performed successful queries, recovery can begin.*

As the first step, for the next query, all remaining servers revert to their initially assigned bit-strings, but truncate the last bit. Thus A uses $\langle 1\,1 \rangle$ *and C uses* $\langle 0\,1 \rangle$*. This query fails. Each server rotates to the next string in the queue (A uses string B and C uses string A), truncated; i.e., A uses* $\langle 1\,0 \rangle$ *and C uses* $\langle 1\,1$*. This still fails. On the third rotation, A uses* $\langle 0\,1 \rangle$ *and C uses* $\langle 1\,0 \rangle$*. This also fails, so another bit is dropped. In round 4, A uses* $\langle 1 \rangle$ *and C uses* $\langle 0 \rangle$*. This fails as well. In round 5, A uses* $\langle 1 \rangle$ *and C uses* $\langle 1 \rangle$*. This query succeeds, indicating that B has failed (since during round 4, B was supposed to flip the first bit in its string, and both A and C know this.*

Developing a systematic method of producing bit-strings suitable for failure detection in larger multi-party models is an obvious avenue for further work. Prefix coding may present useful techniques for constructing such bit-strings quickly.

One drawback to this approach is that every colluding server must observe as many marks as there are colluding servers. Network traffic between the colluding servers and the marks must therefore increase by a factor of n^2, though in a sufficiently noisy system this increase may well go unnoticed.

A worse problem, however, is the fact that each colluding server can only transmit a bit when its mark performs a PIR request. In systems like the Pynchon Gate [10], requests are rate-limited to a fixed number per cycle in order to reduce the effectiveness of long-term intersection attacks and other forms of traffic analysis [6]. If each colluding server can only transmit a few bits per day, the latency of this channel becomes so high as to be useless. (Furthermore, since PIR requests are user-initiated, a colluding server can be silenced if its mark goes offline for a few days.) Fortunately, this problem is easy to address: partition the userbase into sets of marks, such that S_i's set of marks, M_{S_i}, is disjoint with $\bigcup M_{S_j}$, $\forall j \neq i$.

2.3 Subliminal channels in Byzantine-robust PIR

A PIR system which consists of ℓ servers, any k of which need to respond to a client's query, is referred to as a k-out-of-ℓ system. If $t+1$ servers must collude in order to compromise the privacy of a query, the system is termed t-private k-out-of-ℓ. If some v of the k responding servers can return an incorrect answer but the user can still reconstruct the correct response, the system is also called v-Byzantine-robust. v-Byzantine-robust protocols such as that presented in Goldberg [8] rely on polynomial interpolation to reconstruct the contents of the response to a query. When some servers are Byzantine, the set of returned values may correspond to more than one possible polynomial and therefore more than one "valid" data block. Goldberg proposes a list-decoding algorithm which recovers all possible blocks for the values returned, and does so in polynomial time. However, the additional processing time is significant—seconds or even minutes—and thus the user can still function as an oracle for a group of colluding servers.

Note that a colluding server only gets one chance per user to transmit a signal, as once a Byzantine-robust client detects a Byzantine action, it does not request any more blocks from that server. Partitioning the userbase again comes into play here. Overall bandwidth goes down, because a server can only use a mark to transmit a 1 once, and once a server has done so, it cannot use that mark again. However, a server can easily pass an expended mark to another colluding server; i.e., once S_i has transmitted a 1 using mark M_i, all the colluding servers revise their list of marks, assigning M_i to S_{i+1} modulo n, the number of colluding servers.

As a final note, observe that servers establishing a subliminal channel over a v-Byzantine-robust protocol need not (and, in practice, probably should not) prearrange their incorrect answers. Since the colluding servers do not know what seed values have been sent to the non-colluding servers in the network, it is extremely difficult to construct a set of responses which will still interpolate to only one valid polynomial. Thus, servers should always transmit correct responses to their shills' marks, and only transmit incorrect responses to their own marks when they wish to transmit a 1. A v-Byzantine-robust system can still be used as a dead-man switch if a server is configured to require approval to transmit a correct response, and send an incorrect response if no approval is given, but detecting who tripped the switch requires a different (and far simpler) procedure. Since Byzantine-robust clients stop requesting blocks from servers which have taken Byzantine actions, a server which has fallen off the switch will eventually end up blacklisted by all clients in the system. Participants in the channel could simply generate their own dummy traffic and use their own clients'

Byzantine-recovery capabilities to determine which server has stopped transmitting valid traffic.

3 Coercing Collusion from Users

In their paper "A Pact with the Devil", Bond and Danezis discuss ways an adversary can entice third-party users to facilitate his attacks by cooperating in exchange for incentives (and under threat of punishment for failure to cooperate) [4]. A clever attacker employing these methods could significantly broaden the communication channel for the covert communications of the servers; rather than simply relying on an unwitting user's side-channel leakage in the form of success-or-failure detection, users could be drafted into relaying messages encoded in the seemingly random data that PIR clients transmit to the PIR databases. The human factor to the security of such systems should be further explored in light of attack schemes such as those presented in [4].

4 Future Work

4.1 Byzantine-robustness among the Byzantines

The general approach described in this paper requires all colluding servers to be *faithful* to each other, i.e., not interfere with other servers' marks and not alter the data they send to users beyond their manipulation of the request string. A Byzantine shill can produce spurious signals from S_i by neglecting to flip its prearranged bits in a request from M_i, or by flipping random bits in the bit-string or the data sent to M_i. Furthermore, a Byzantine (or merely malfunctioning) server outside the colluding group can also produce a spurious signal. If the only means of determining whether a signal has been sent is whether a mark's request succeeds or fails, then if all colluding servers flip their bits as arranged and a server outside the group takes a Byzantine action, the mark will still have to retry its request or perform a recovery action, thus the mark's behaviour will indicate that a signal was sent.

However, although servers within a collusion group should technically be considered Byzantine with respect to the PIR system as a whole, the behaviour of faithful colluding servers differs from Byzantine servers outside a collusion group, and this behaviour lends itself to differentiation.

Further work must be done to develop a Byzantine-robust collusion algorithm for Byzantine servers, if needed for a given attacker's own threat model, though solutions to this problem can be found in existing literature, and are out of the scope of this paper.

4.2 Multiple collusion groups

The multi-party approach described in this paper assumes that no mark will be assigned to more than one colluding server. However, when more than one group of servers agrees to collude, it is probable that two servers in different groups will choose the same mark. Thus, it is likely that multiple groups attempting to establish a subliminal channel on the same PIR network will experience irreparable collisions as they attempt to broadcast at the same time, thus rendering both channels unreadable. Investigating techniques for discovering the existence of multiple channels on one PIR network, and possibly establishing non-colliding channels, is also a subject for further work.

5 Conclusion

In this paper, we have demonstrated the potential for a subliminal communication channel in distributed private information retrieval networks which is indistinguishable from ordinary network errors. We have shown its applicability to XOR-based and polynomial-interpolation-based PIR systems, and have shown how an arbitrarily large number of parties can communicate using this channel. Furthermore, we have shown a method for failure detection in the multi-party model and opened avenues for investigation toward making this channel more robust against outside interference. More work must be done to make this a reliable channel for complex, ongoing communication, but for simple use-cases, the necessary tools are already in place.

Acknowledgements Meredith L. Patterson would like to thank the COSIC group at K.U. Leuven, for its hospitality during her visit to Leuven, and for the discussions which led in part to this work.

Len Sassaman's work was supported was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government, by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and by the EU within the PRIME Project under contract IST-2002-507591.

References

[1] A. Beimel and Y. Stahl. Robust information-theoretic private information retrieval. In S. Cimato C. Galdi G. Persiano, editor, *3rd Conf. on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 326–341. Springer-Verlag, 2002.

[2] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. *Lecture Notes in Computer Science*, 2076:89–98, 2001.

[3] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $O(n^{1/(2k-1)})$ Barrier for Information-Theoretic Private Information Retrieval. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[4] Mike Bond and George Danezis. A pact with the devil. In *New Security Paradigms Workshop (NSPW 2006)*, page 13, Schloss Dagstuhl,DE, 2006. ACM.

[5] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[6] George Danezis. Statistical disclosure attacks: Traffic confirmation in open environments. In Gritzalis, Vimercati, Samarati, and Katsikas, editors, *Proceedings of Security and Privacy in the Age of Uncertainty, (SEC2003)*, pages 421–426, Athens, May 2003. IFIP TC11, Kluwer.

[7] Yael Gertner, Shafi Goldwasser, and Tal Malkin. A random server model for private information retrieval or how to achieve information theoretic pir avoiding database replication. In *RANDOM '98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 200–217, London, UK, 1998. Springer-Verlag.

[8] Ian Goldberg. Improving the Robustness of Private Information Retrieval. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, May 2007. `http://www.cypherpunks.ca/~iang/pubs/robustpir.pdf`.

[9] Alexander A. Razborov and Sergey Yekhanin. An $\Omega(n^1/3)$ Lower Bound for Bilinear Group Based Private Information Retrieval. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 739–748, Washington, DC, USA, 2006. IEEE Computer Society.

[10] Len Sassaman, Bram Cohen, and Nick Mathewson. The Pynchon Gate: A Secure Method of Pseudonymous Mail Retrieval. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2005)*, Arlington, VA, USA, November 2005.

[11] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *Advances in Cryptology: Proceedings of Crypto 83*, pages 51–67. Plenum Press, 1984.

[12] David P. Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity (CCC 2005)*, pages 275–284, San Jose, CA, USA, June 2005. IEEE Computer Society.

[13] Sergey Yekhanin. Towards 3-Query Locally Decodable Codes of Subexponential Length. In *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC)*, San Diego, CA, USA, June 2007. ACM Press.