Implementing Information-Theoretically Secure Oblivious Transfer from Packet Reordering

Paolo Palmieri and Olivier Pereira

Université catholique de Louvain,
UCL Crypto Group,
Place du Levant 3, B-1348 Louvain-la-Neuve, Belgium
{paolo.palmieri,olivier.pereira}@uclouvain.be

Abstract. If we assume that adversaries have unlimited computational capabilities, secure computation between mutually distrusting players can not be achieved using an error-free communication medium. However, secure multi-party computation becomes possible when a noisy channel is available to the parties. For instance, the Binary Symmetric Channel (BSC) has been used to implement Oblivious Transfer (OT), a fundamental primitive in secure multi-party computation. Current research is aimed at designing protocols based on real-world noise sources, in order to make the actual use of information-theoretically secure computation a more realistic prospect for the future.

In this paper, we introduce a modified version of the recently proposed Binary Discrete-time Delaying Channel (BDDC), a noisy channel based on communication delays. We call our variant Reordering Channel (RC), and we show that it successfully models *packet reordering*, the common behavior of packet switching networks that results in the reordering of the packets in a stream during their transit over the network. We also show that the protocol implementing oblivious transfer on the BDDC can be adapted to the new channel by using a different sending strategy, and we provide a functioning implementation of this modified protocol. Finally, we present strong experimental evidence that reordering occurrences between two remote Internet hosts are enough for our construction to achieve statistical security against honest-but-curious adversaries.

Keywords: Oblivious transfer, secure multi-party computation, noisy channels, packet reordering, delay.

1 Introduction

When a source transmits information over a packet-switching network, it produces an in-order sequence of packets. However, depending on the network properties and the communication protocol used, the sequence received at the destination might be a different one. In this paper we show how this noise introduced by the network can be used in practice to achieve oblivious transfer and, more generally, secure computation.

H. Kim (Ed): ICISC 2011, LNCS 7259, pp. 332–345, 2012.

At the network level of the ISO/OSI model, the Internet Protocol (IP) offers no guarantee that packets are received at destination in the same order in which they were sent at the source. This task is taken on by some of the protocols at the transport layer, most notably the Transmission Control Protocol (TCP), while others leave the problem unaddressed, as in the case of the User Datagram Protocol (UDP). The phenomenon for which the ordering of a sequence of packets in a stream is modified during its transit on a network is commonly known as *packet reordering*. Common causes of packet reordering are packet striping at the data-link and network layers [2,11], priority scheduling and route fluttering [16,3]. Reordering is a common behavior over the Internet. For instance, tests conducted in [1] for 50 hosts, 35 of which chosen randomly, show an occurrence rate of over 40%, with a mean reordering rate roughly fluctuating between 10 and 20% per occurrence. Current Internet trends, like increasing link speeds and increased parallelism within routers, wireless ad hoc routing, and the widespread use of quality of service (QoS) mechanisms and overlay routing, all indicate an expected increase in packet reordering occurrences.

For its ability to deteriorate the responsiveness and quality of data transmission, especially in applications featuring real time communication or streaming of multimedia content, packet reordering is generally treated as any other form of noise: a problem that needs to be solved. However, cryptographers have a history in transforming noise into something useful and desirable. It is the case of secure multi-party computation, that can be achieved only through the use of noisy channels when the adversaries are computationally unbounded. Multi-party computation deals with the problem of performing a shared task between two or more players who do not trust each other. Security is achieved when the privacy of each player's input and the correctness of the result are guaranteed [4]. A basic primitive and a fundamental building block for any secure computation is Oblivious Transfer (OT), introduced by Rabin in 1981 [19]. In fact, when oblivious transfer is available, any two-party computation can be implemented in a secure way [13]. A commonly used variant of the primitive is the 1-out-of-2 oblivious transfer, proposed by Even, Goldreich and Lempel [10], and later proved to be equivalent to the original OT by Crépeau [5]. In this protocol, a sender Sam knows two secrets, and is interested in transmitting one to a receiver Rachel without disclosing the other. Rachel wants to choose which secret to receive, but does not want to reveal her choice. Privacy of the inputs and correctness of the result are achieved without implying any degree of mutual trust between the players.

The first protocol to implement oblivious transfer over a noisy channel used the well-known Binary Symmetric Channel (BSC) [6]. The BSC is a simple binary channel that flips with probability p each bit passing trough it. While being a common reference in information theory, the BSC proved not to satisfy cryptographers, more interested in modeling advantages a potential adversary might have. Many modifications of the channel were consequently proposed, in the direction of allowing dishonest players an edge over honest ones. An Unfair Noisy Channel (UNC) let the adversary choose the crossover probability within a

specific (narrow) range [9,8], while the Weak Binary Symmetric Channel (WBSC) introduces the possibility for the dishonest player to know with a certain probability if a bit was received correctly [20]. The aim of these constructions is to gain generality by easing the security assumptions, but, while we know that OT can be built on any non-trivial noisy channel [7], none of these proved to be suitable for actual implementation in a real world communication scenario, due to the lack of flexibility and strong requirements imposed by the channel model.

A different approach was taken in [15], where the proposed oblivious transfer protocol uses a different source of noise: communication delays. The protocol is built over a new noisy channel model, called Binary Discrete-time Delaying Channel (BDDC), that accepts binary string inputs at discrete times, and output each string at the following discrete time. Strings passing through the channel have a probability p of being delayed, and therefore being kept in the channel until the following output time.

1.1 Contribution

In this paper we present a new channel model, the Reordering Channel (RC). The reordering channel is a modified version of the BDDC, that modifies the concept of delay from a temporal one to that of shifting positions in a sequence. We observe that the RC provides enough ambiguity (noise) for building an oblivious transfer protocol, and we show that using a different strategy for sending packets at the sender's end we are able to build oblivious transfer on the channel using a modified version of the same protocol used on the BDDC. Since the reordering channel models the behavior of packet reordering over the Internet, we provide an actual, functioning implementation of the protocol based on the transmission of UDP packets. The source code of the application is provided.[1] Finally, we present strong experimental evidence supporting the effectiveness and security of the construction and we show how different specific packet reordering behaviors can be used to improve the efficiency of the protocol.

To the best of our knowledge, this is the first actual implementation of oblivious transfer over the Internet that provides security against computationally unbounded adversaries, that is, adversaries with unlimited computational capabilities.

1.2 Outline of the Paper

In section 2 we introduce some preliminary notions and definitions relative to oblivious transfer that will be useful in the following. The definition of BDDC and the protocol implementing OT over it are also presented. In section 3 we discuss the implementation of oblivious transfer from packet reordering. We initially introduce the reordering channel and we show how the OT protocol for the BDDC can be modified to work on the new channel. In Section 3.1 we discuss

[1] The source code of the latest version of the program is available for download over the internet at the address: `http://www.uclouvain.be/crypto/ifyd-latest.tar.gz`.

common reordering behaviors that influence the design of our implementation of the protocol, which is presented in Section 3.2. In Section 3.3 we introduce some metrics used to analyze the data gathered during the testing of our application, of which we relate in Section 3.4, where we present statistical evidence of the security of our construction.

2 Preliminaries

Our construction is based on the chosen 1-out-of-2 binary oblivious transfer (in the following simply called oblivious transfer). A protocol implements oblivious transfer in a secure fashion when three conditions are satisfied after a successful execution: the receiver party learns the value of the selected bit b_s (correctness); the receiver party gains no further information about the value of the other bit b_{1-s} (security for Sam); the sender party learns nothing about the value of the selection bit s (security for Rachel) [6].

The behavior of the players defines the level of security that a protocol can achieve. The oblivious transfer protocol for the BDDC is secure against *honest-but-curious* players. In practice, the player strictly follows the protocol but tries to gain extra information from her inputs and output, in order to gain an advantage in guessing the other player's secret.

2.1 Binary Discrete-Time Delaying Channel

A protocol for achieving oblivious transfer from communication delays in the information-theoretic model has been proposed in [15]. The noisy channel used to model delay is the Binary Discrete-time Delaying Channel (Figure 1).

Definition 1. [15] *A* Binary Discrete-time Delaying Channel *with delaying probability p consists of: an input alphabet $\{0,1\}^n$, an output alphabet $\{0,1\}^n$, a set of consecutive input times $T = \{t_0, t_1, \ldots\} \subseteq \mathbb{N}$, a set of consecutive output times $U = \{u_0, u_1, \ldots\} \subseteq \mathbb{N}$ where $\forall u_i \in U, t_i \in T, u_i \geq t_i$. Each input admitted into the channel at input time $t_i \in T$ is output once by the channel, with probability of being output at time $u_j \in U$*

$$\Pr[u_j] = p^{(j-i)} - p^{(j-i+1)} \quad . \tag{1}$$

2.2 Oblivious Transfer over a BDDC

The following protocol, also proposed in [15], implements oblivious transfer over a BDDC with error probability p. The sender party, Sam, inputs two secret bits b_0, b_1 and gets no output; the receiver Rachel inputs the selection bit s and receives output b_s. In the following, we introduce a modified version of this protocol, which serves as the base for our construction over packet reordering.

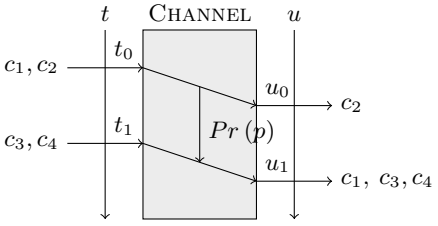

Fig. 1. A schematization representing a Binary Discrete-time Delaying Channel accepting two strings at time t_0, one of which gets delayed once, and two at time t_1, none of which gets delayed. This results in the channel emitting one string at time u_0 and three at u_1.

Protocol 1. [15] *Before starting the communication between the parties, Sam selects two disjoints sets E, E' each composed of n distinct binary strings of length l: $e_1, \ldots, e_n$ and $e'_1, \ldots, e'_n$. Then, Sam builds the sets $C = \{c_1, \ldots, c_n\}$ and $C' = \{c'_1, \ldots, c'_n\}$, where $c_i := e_i \| i$ and $c'_i := e'_i \| i$.*

1. *Sam sends to Rachel the set C at instant t_0, and C' at t_1, using a p-BDDC.*
2. *Rachel receives over the BDDC the strings in $\{C \cup C'\}$, in the order produced by the channel. She keeps listening on the channel at instants $u_2, u_3, \ldots$ until all the delayed strings have been received.* [2]
3. *Rachel selects the set I_s, where $s \in \{0, 1\}$ is her selection bit, such that $|I_s| = \frac{n}{2}$ and so that $i \in I_s$ only if $c_i \in C$ has been received at u_0. Then she selects $I_{1-s} = \{1, \ldots, n\} \setminus I_s$ and sends I_0 and I_1 to Sam over a clear channel.*[3]
4. *Sam receives I_0, I_1 and chooses two universal hash functions f^0 and f^1, whose output is 1-bit long for any input. Let $E_j \subset E$ be the set containing every $e_i \in E$ corresponding to an $i \in I_j$, such that*

$$e_i \in E_j \Leftrightarrow i \in I_j \ . \tag{2}$$

For each set I_j, Sam computes the string g_j by concatenating each $e^j_k \in E_j$, ordering them for increasing binary value, so that

$$g_j = \left(e^j_1 \parallel \ldots \parallel e^j_{\frac{n}{2}}\right) \qquad \text{with } e^j_1, \ldots, e^j_{\frac{n}{2}} \in E_j \ . \tag{3}$$

Sam computes $h_0 = f^0(g_0)$, $h_1 = f^1(g_1)$ and sends to Rachel the functions f^0, f^1 and the two values

$$i_0 = (h_0 \oplus b_0) \ , \qquad i_1 = (h_1 \oplus b_1) \ . \tag{4}$$

[2] If less than $\frac{n}{2}$ strings are received at u_0 Rachel instructs Sam to abort the communication.

[3] Or Rachel can just send one of these two sets in order to save bandwidth as Sam can easily reconstruct the other.

5. Rachel computes her guess for b_s

$$b_s = f^s(g_s) \oplus i_s \ . \tag{5}$$

The internal working of the protocol is easily explained: when listening on the BDDC, Rachel receives at instant u_0 all the strings in C that have not been delayed by the channel. This subset will constitute the correct information she needs to decode the selected bit. Any string received at u_1 or at a later time is instead ambiguous, and guarantees that Rachel can not decode both b_0 and b_1. At u_1, the strings from C delayed once and the strings of set C' that have not been delayed can not be distinguished, and so on and so forth for $u_2, u_3, \ldots$.

3 Packet Reordering as a Noisy Channel

While the BDDC is able to model discrete delays in a communication, it is not suitable to simulate the delaying behavior of packet switching networks, which is usually visible in the form of packet reordering. Therefore, we introduce a new channel model, called Reordering Channel (RC), that redefines the concept of delay using the relative position of a packet in a stream.

Definition 2. *A* Reordering Channel *consists of: an input sequence of binary strings $T = (t_1, \ldots, t_n)$, an output sequence of binary strings $U = (u_1, \ldots, u_n)$, a sequence of identically distributed discrete random variables $\{X_n\}$ over $\mathbb{N}$ and its probability distribution $P_X : \mathbb{N} \to [0, 1]$, with $\sum_{m \in \mathbb{N}} P_X(n) = 1$. Each string in T is output once by the channel in U, i.e. $\{t_1, \ldots, t_n\} \equiv \{u_1, \ldots, u_n\}$. The ordering of the output sequence is determined by the channel, that selects for the next available position in the output sequence the $t_i \in T$ not already selected with the smallest value $v = i + X_i$. In case more than one string shares the same value v, the channel selects among them the one with the smallest value i.*

In practice, the channel takes a stream of packets as input, and outputs the same packets in a reordered fashion. For an appropriate distribution function, where the probability of a packet not to be delayed ($X = 0$) is high enough, this channel simulates accurately the reordering behavior of standard Internet connections. We discuss experimental results regarding the amount of reordering that is observed on the Internet in Section 3.4.

With an appropriate discrete probability distribution, that is, a distribution that respects (1) for a value p, the delaying behavior of the reordering channel follows that of the corresponding BDDC with probability p. However, the reordering channel, due to its continuous nature, opposed to the discrete one of the BDDC, lacks the reference points in time that are needed by the receiver to make sure that a string has not been delayed. Therefore, the protocol that implements oblivious transfer over the BDDC will not work on the RC. To adapt the protocol to the new channel, we need to use a different sending strategy at the sender's end. Instead of sending the two sets C and C' from step 1 of the protocol into the reordering channel sequentially, we can send them as a stream,

by interleaving the strings in the two sets. We observe, in fact, that we obtain an ambiguity similar to the one of two strings output at the same time by the BDDC, if two strings with the same value happen to be received consecutively as a result of reordering by the RC. As illustrated in Figure 2, we can start sending a first batch of i strings form C, where i is the arbitrarily selected interleaving value. After c_i, we interleave the strings from C with the ones from C'. Using this sending sequence, the receiver is unable to distinguish between two strings c_i and c'_i when c_i is reordered, and received at least $i-1$ positions far from its original place. Adopting this sending strategy, we can implement the oblivious transfer protocol for the BDDC on the reordering channel and, therefore, on the Internet.

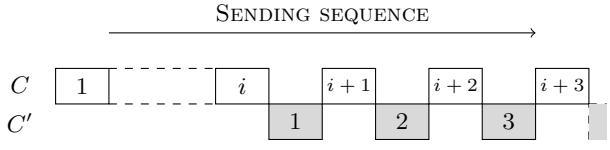

Fig. 2. Package interleaving

3.1 Reordering Dynamics

For an implementation of the protocol to be effective, the selected value of i must be consistent with the actual amount of reordering observed over the Internet. However, it is not the only parameter that will affect reordering.

The probability of occurrence of packet reordering depends on a number of factors, such as physical distance and number of intermediate hops between the hosts, transmission medium, quality and speed of hop-to-hop links, traffic on the network and so on. Packet reordering also frequently displays a consistent behavior over time between two given hosts.

As already experimentally observed by Bellardo and Savage, the inter-spacing of packets effectively reduces the reordering probability [1]. In the test they conducted, the probability is significantly reduced when adding an inter-packet gap of 100 microseconds (μs), while a longer spacing of 500 μs brings the number of reorderings close to 0. An increase in the size of the packets has the same effect, since the longer serialization delay increases the delay between the leading edge of each subsequent packet. This, in turn, decreases the possibility that two packets will be reordered if assigned to different queues, when subject to parallelization during routing. We can actively use this property by adding an inter-packet gap to stabilize a path affected by a high reordering probability.

3.2 Protocol Implementation

In our implementation of the protocol, the receiver acts as a server, waiting online for a client (the sender) to connect. The protocol used to transmit packets is the User Datagram Protocol (UDP). UDP provides no guarantees of message

delivery to the upper layer, offers no reordering detection or correction mechanism, and retains no state of the messages once sent. The simple structure of a packet, called datagram and defined in RFC 768, minimizes the size and does not include any sequence number [18]. The structure of the packets sent follows the structure of the strings in C and C' as defined in Protocol 1. Each packet p_j is composed of the sequence number j and an unique identification value e.

To select the receiver mode of operation, the option `-r {S}` must be specified, where `S` is the selection bit. The receiver algorithm is structured as follows. After network initialization, the program waits for incoming connections on port 9930. Once packets are received, they are put in the arrival order in a buffer. The number of packets to be received is determined in advance with the sender. The buffer is then read packet by packet, and each pair of packets sharing the same sequence number j and satisfying any of the following conditions is marked as ambiguous: the first packet with sequence number j is found in a position higher than $(j + i - 1)$; the two packets sharing sequence number j are less than $\frac{i}{2}$ position apart. Sets I_s and I_{1-s} are created by putting all the sequence numbers corresponding to an ambiguous pair of packets in I_{1-s}, plus enough non-ambiguous sequence numbers to reach half the total, and putting the remaining values in I_s. The two sets are then sent back to the sender's address. The software then waits for the encoded bits, and the chosen bit b_s is decoded using (5).

The sender mode of operation is selected by using the option `-s {B0:B1}`. The two secret bits are passed in the argument, separated by a colon character. The mandatory option `-a {IP_ADDRESS}` is used to specify the receiver's IP address. `-w USEC` can be used to add a `USEC` microseconds long waiting gap between packets. The algorithms is structured as follows. Two sets P^0 and P^1 of n packets each are created, with each packet p composed of an increasing sequence number j and a randomly selected unique identifier e. The two sets are then sent to the receiver's address, using the sending sequence described in Section 3 for a predetermined i. Once all the packets have been sent, the software starts waiting for the sets I_0, I_1 from the receiver. Using the information received, the secret bits are encoded according to (4), using an hash function, and sent to the receiver.

The reference platform for our implementation of the protocol is Linux. The programming language used is C++. Only standard POSIX libraries have been used. The compiler of choice is the GNU Compiler Collection (gcc), version 4. Full logging capabilities are implemented, including on-screen and file logging.

3.3 Metrics

In order to measure the incidence and relevance of the reorderings observed in our tests, we use the metrics proposed in RFC 5236 [12] and in [21], adapting them to the needs of our specific application when necessary.

When packets are received at the destination they are assigned a *receive index* (RI), according to the order of arrival. *Displacement* (D) of a packet is defined as the difference between RI and the sequence number of the packet. For example, the displacement of packet p_j^0 from the first set is $RI\left(p_j^0\right) - j$,

while $D\left(p_j^1\right) = RI\left(p_j^1\right) + i - j$. Therefore, a negative displacement value indicates the earliness of a packet and a positive value the lateness. We call *absolute displacement* the modulus of D. The *displacement frequency* $FD\left(k\right)$ is the number of received packets having a displacement of k. The *reorder density* (RD) is the distribution of the displacement frequencies, normalized with respect to the number of received packets, ignoring lost and duplicate packets. The *mean displacement of packets* (M_D) is defined as

$$M_D = \left| \sum_{i=-D_r}^{i=+D_r} \left(|i| \times RD\left[i\right] \right) \right| / \left| \sum_{i=-D_r}^{i=+D_r} RD\left[i\right] \right| \ , \tag{6}$$

while the *mean displacement of late packets* (M_L) is

$$M_L = \left| \sum_{i=1}^{i=+D_r} \left(|i| \times RD\left[i\right] \right) \right| / \left[\sum_{i=1}^{i=+D_r} RD\left[i\right] \right] \tag{7}$$

in the case of packets with positive displacement. The *reorder entropy* (E_R) is an indicator of the reorder density (a discrete probability distribution) to be concentrated or dispersed. It is defined as $E_R = (-1) \times \sum_{i=-D_r}^{i=+D_r} \left(RD\left[i\right] \times \ln RD\left[i\right] \right)$.

For simplicity, and without loss of generality, in the following we study only absolute displacements. In fact, for our purposes, displacements values of $\pm i$ are equally ambiguous.

3.4 Experiment

Contrary to what is common in the study of packet reordering, our experiment uses a an active approach, instead of passively monitoring traffic. We do so by using the testing capabilities included in our protocol implementation, which let us produce a stream of UDP datagrams from the sender to the receiver. For testing the software we use two hosts:

- `merlin.dice.dice.ac.ucl.be`, IP `130.104.205.236`, located in Belgium. Debian GNU/Linux (kernel 2.6.32-5, i686);
- `ec2-50-18-108-9.us-west-1.compute.amazonaws.com`, IP `50.18.108.9`, located in Northern California (USA). Ubuntu GNU/Linux (kernel 2.6.35, x86 64-bit).

Both hosts are connected to the Internet through wired, high-speed links. A sample `tracert` shows 18 hops between the two. The mean round trip time (RTT) is 149.6 milliseconds, with a standard deviation of 0.6 ms.

In the following, we base our analysis on two sample traffic datasets, produced by observing the behavior of our protocol implementation in two different settings.

The dataset corresponding to the first experiment is the result of a single prolonged execution of the protocol test routine. In total, 60167 datagrams are recorded. The test session took place in May, 2011. The aim of the test is to

observe behavior under ideal traffic conditions: during the execution, both hosts had no network activity beside the traffic generated by our protocol implementation itself. A first analysis of the data shows a total of 7009 reordering occurrences, equal to 11.65% of all packets received. The maximum displacement value observed is 59. Detailed figures for low displacement values are displayed in Fig. 3. The mean displacement is $M_D = 0.44$, while the mean displacement of late packets is $M_L = 3.75$. The reorder entropy of the set is $E_R = 0.60$, which shows a good variance and therefore a uniformity of displacement frequency and probability. These values appear to be consistent with those observed in [21].

D	FD	%
0	53157	88.35
1	1876	3.12
2	1697	2.82
3	1240	2.06
4	860	1.43
5	468	0.78

D	FD	%
6	246	0.41
7	137	0.23
8	79	0.13
9	59	0.10
10+	347	0.58

Fig. 3. Number of occurrences and percentage for each absolute displacement value

The security of our construction is based on the assumption that the sender can not accurately predict reorderings that will happen during the datagrams transit over the network. Predictions of future reorderings are based on the observation of past occurrences, and the research of patterns in the frequency and magnitude of displacements. In order to show the independence between past reordering occurrences and future ones, we evaluate the *autocorrelation* function. Autocorrelation is the cross correlation of a vector of random variables with itself, and is a useful tool frequently used in signal processing to find repeating patterns. In Fig. 4 the number of reordering occurrences for three different subset into which the first dataset has been divided for a more detailed analysis are provided, along with the relative autocorrelation. It is evident that none of the functions shows any recurring pattern.

The second experiment aims to reproduce reordering behavior under intense traffic over the network. In order to generate traffic we use the software suite composed of RUDE (Real-time UDP Data Emitter) and CRUDE (Collector for RUDE), developed by Laine and Saaristo [14]. This client-server tool allows us to produce UDP traffic from the sender to the receiver with a great deal of precision. In particular, we use the following routine:

- for the first 5 seconds, RUDE sends 1000 packets per second, with a packet size of size 200 bytes, generating a traffic of 200 KB/s;
- for the following 5 seconds, RUDE sends 10000 packets/second with 20 bytes/packet (200 KB/s);
- for the last 5 seconds, RUDE sends 500 packets/second with 2000 bytes/packet (1 MB/s);

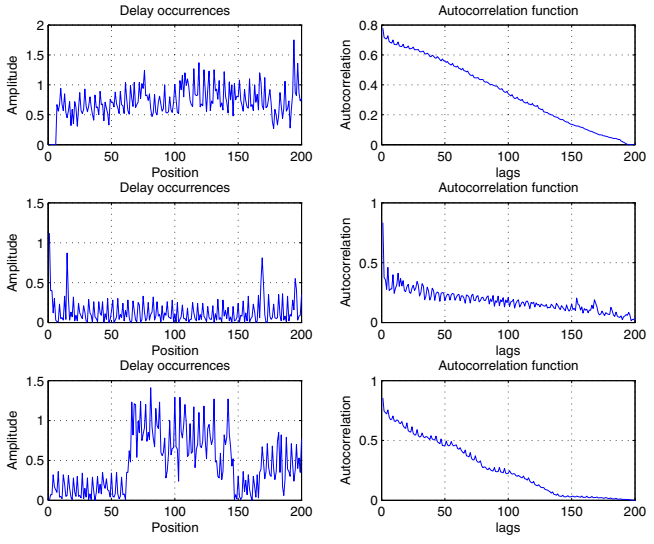

Fig. 4. The number of occurrences (left) and the autocorrelation function (right) for the three subsets composing the first dataset

– for all the duration of the test, RUDE also sends 2500 packets/second with 100 bytes/packet (250 KB/s). These last datagrams have the Type of Service (ToS) priority flag of the IPv4 header set to `LOW_DELAY` (`0x10`).

In addition to the traffic generated by RUDE, the receiver host also performs a large file download from a remote host (`mirror.garr.it`, 131.175.1.35), using the HTTP protocol. The file chosen is large enough for the download to last over the entire execution of the experiment. The average download speed observed is 96.5 KB/s.

During the experiment, a total of 4293 datagrams generated by our protocol implementation are received by the receiver host. About 4 seconds after the testing routine start, a peak of reorderings is recorded (Fig. 5): this is due to a set of packets being lost due to the traffic congestion. The particular routine of external traffic generated with RUDE allows us to analyze how the frequency of reordering occurrences is affected by manipulation of the traffic reaching the receiver host. Fig. 5 clearly shows that the reordering behavior is only marginally affected by external traffic, even when the capacity of the network is almost fully used. In fact, no significant differences both in the number of occurrences and mean displacement value can be seen at the change of bandwidth used at second 10, where the external traffic goes from 200 KB/s to 1 MB/s. The lower values recorded for the first 4 seconds can be instead explained by the progressive increase in TCP traffic generated by the HTTP download, that take full advantage of the available bandwidth only after a few seconds, thanks to the mechanisms regulating TCP traffic.

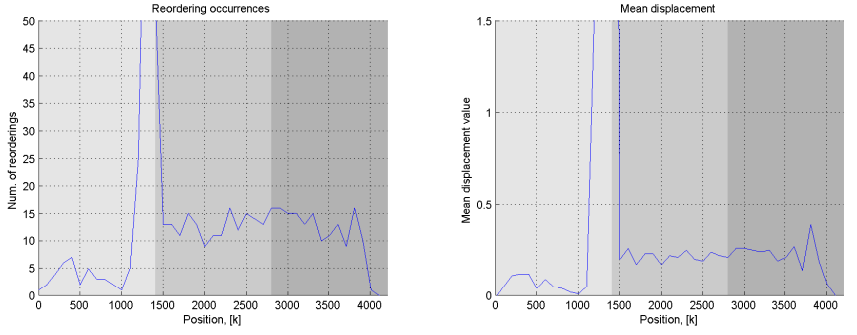

Fig. 5. The number of occurrences (left) and the mean displacement value (right) for the second dataset, calculated over subsets of 100 packets each. The peak of both functions is due to the loss of a set of packets. The three areas colored with different shades of gray picture the three time lapses into which the set is divided (5 seconds each)

4 Conclusion

In this paper we propose an implementation of oblivious transfer over the Internet. The construction we propose is secure against adversaries with unlimited computational capabilities in the honest-but-curious model, and uses packet reordering, a common phenomenon present in any packet-switching network. Reordering of packets in a stream are due to a number of different causes, among which intrinsic network characteristics, parallelism both at the node and the link level, and traffic control and congestion, all of which are increasingly present in today's Internet.

Our construction is based on the protocol proposed for the Binary Discrete-time Delaying Channel. In order to adapt the protocol for practical use over the Internet, we introduce a new channel, the Reordering Channel, that models packet reordering. We then build a modified version of the protocol, adapted to work on the RC, and we present a practical implementation of this modified protocol based on the transmission of UPD packets over the Internet. We also present extensive experimental evidence of the security of the implementation: we show that reordering occurrences are found consistently under both intense traffic load and minimal network usage, statistical analysis of the reorderings shows no sign of recurring patterns in frequency or magnitude and no strong statistical correlation is found over reordering occurrences over time.

To the best of our knowledge, our implementation is the first oblivious transfer protocol secure against computationally unbounded capabilities being implemented over the Internet. The novelty of our construction, based on network behavior, opens the way to new security constructions entirely based on channel characteristics.

Acknowledgments. This research work was supported by the SCOOP Action de Recherche Concertées. Olivier Pereira is a Research Associate of the F.R.S.-FNRS.

References

1. Bellardo, J., Savage, S.: Measuring packet reordering. In: Internet Measurement Workshop, pp. 97–105. ACM (2002)
2. Bennett, J.C.R., Partridge, C., Shectman, N.: Packet reordering is not pathological network behavior. IEEE/ACM Trans. Netw. 7(6), 789–798 (1999)
3. Bohacek, S., Hespanha, J.P., Lee, J., Lim, C., Obraczka, K.: A new tcp for persistent packet reordering. IEEE/ACM Trans. Netw. 14(2), 369–382 (2006)
4. Chaum, D., Damgård, I., van de Graaf, J.: Multiparty computations ensuring privacy of each party's input and correctness of the result. In: Pomerance [17], pp. 87–119
5. Crépeau, C.: Equivalence between two flavours of oblivious transfers. In: Pomerance [17], pp. 350–354
6. Crépeau, C., Kilian, J.: Achieving oblivious transfer using weakened security assumptions (extended abstract). In: FOCS, pp. 42–52. IEEE (1988)
7. Crépeau, C., Morozov, K., Wolf, S.: Efficient Unconditional Oblivious Transfer from Almost Any Noisy Channel. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 47–59. Springer, Heidelberg (2005)
8. Damgård, I., Fehr, S., Morozov, K., Salvail, L.: Unfair Noisy Channels and Oblivious Transfer. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 355–373. Springer, Heidelberg (2004)
9. Damgård, I.B., Kilian, J., Salvail, L.: On the (Im)possibility of Basing Oblivious Transfer and Bit Commitment on Weakened Security Assumptions. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 56–73. Springer, Heidelberg (1999)
10. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM 28(6), 637–647 (1985)
11. Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., Towsley, D.: Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. IEEE/ACM Trans. Netw. 15, 54–66 (2007), http://dx.doi.org/10.1109/TNET.2006.890117
12. Jayasumana, A., Piratla, N., Banka, T., Bare, A., Whitner, R.: Improved packet reordering metrics. RFC 5236 (Informational) (June 2008), http://www.ietf.org/rfc/rfc5236.txt
13. Kilian, J.: Founding cryptography on oblivious transfer. In: STOC, pp. 20–31. ACM (1988)
14. Laine, J., Saaristo, S.: RUDE: Real-time UDP data emitter (1999–2002), http://rude.sourceforge.net/
15. Palmieri, P., Pereira, O.: Building Oblivious Transfer on Channel Delays. In: Lai, X., Yung, M., Lin, D. (eds.) Inscrypt 2010. LNCS, vol. 6584, pp. 125–138. Springer, Heidelberg (2011)
16. Paxson, V.E.: Measurements and Analysis of End-to-End Internet Dynamics. Ph.D. thesis, EECS Department, University of California, Berkeley (June 1997), http://www.eecs.berkeley.edu/Pubs/TechRpts/1997/5498.html
17. Pomerance, C. (ed.): CRYPTO 1987. LNCS, vol. 293. Springer, Heidelberg (1988)
18. Postel, J.: User datagram protocol. RFC 768 (Standard) (August 1980), http://www.ietf.org/rfc/rfc768.txt

19. Rabin, M.O.: How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University (1981) (manuscript)
20. Wullschleger, J.: Oblivious Transfer from Weak Noisy Channels. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 332–349. Springer, Heidelberg (2009)
21. Ye, B., Jayasumana, A.P., Piratla, N.M.: On monitoring of end-to-end packet reordering over the internet. In: International Conference on Networking and Services (2006)