Rendezvous Tunnel for Anonymous Publishing

by

Ofer Hermoni, Niv Gilboa, Eyal Felstaine, Yuval Elovici and Shlomi Dolev

Technical Report #10-05

September 2010

Rendezvous Tunnel for Anonymous Publishing

Abstract

Many anonymous peer-to-peer (P2P) file sharing systems have been proposed in recent years. One problem that remains open is how to protect the anonymity of *all* participating users, namely, reader, server and publisher. In this work we propose a novel solution for a P2P file sharing system. **Our solution provides overall anonymity to all participating users**.

The novelty of this work is threefold. First, we introduce an anonymous key exchange protocol secure against an honest but curious adversary. The anonymity of the protocol is proved on the basis of the Decisional Diffie Hellman (DDH) problem. Second, we propose two solutions to build the rendezvous tunnel: basic and advanced. The basic solution is straightforward, while the advanced solution is based on the key exchange protocol. In the advanced solution, the key exchange is done between the publisher and each user along the rendezvous tunnel that preserve anonymity in the presence of limited traffic recording. Third, the rendezvous tunnel is used as a building block for an anonymous P2P file sharing system that provides anonymity to **all** participating users.

I. Introduction

On-line communication occupies a great part of the daily lives of many people. Peer-to-peer (P2P) systems have recently become more popular, as we use them to chat, talk, share files etc. Anonymous P2P systems have thus become an important area of research [1], [3], [5], [6], [8], [11], [17], [18] and implementation [7]. Anonymity in P2P file sharing means that an adversary cannot link participating users to the content they share. Note that an adversary can act as a user, and thus the users also have to remain hidden with respect to one another.

A weakness common to many existing anonymity solutions is that they do not provide (or even consider) anonymity for *all* participating users, namely, publisher, server and reader. Solutions such as those given in [3], [16], [19] provide anonymity to the reader but do not protect the server that stores the shared document. Publishing solutions such as [6], [12] provide publisher and reader anonymity, but do not protect the servers or the creator of an index.

In our solution, anonymity for all users is achieved by using three anonymity tunnels (Figure 1). Anonymity tunnels are widely used in theory and in practice, e.g., Tor [7] uses tunnels to provide anonymity. We use two types of anonymity tunnels. The publishing and reading tunnels are sender anonymity tunnels. A sender anonymity tunnel is designed to protect the anonymity of the message's sender. In a sender anonymity tunnel, the sender knows the identity of the recipient, and anonymity is achieved by using a Mix [3] based protocol. The rendezvous tunnel is quite different. In this tunnel, the initiator of the tunnel does not know the identity of the user at the end of the tunnel. This tunnel is built by using a random walk. The reading and the publishing tunnels protect the anonymity of the reader and the publisher, respectively, whereas the rendezvous tunnel protects the anonymity of the server.

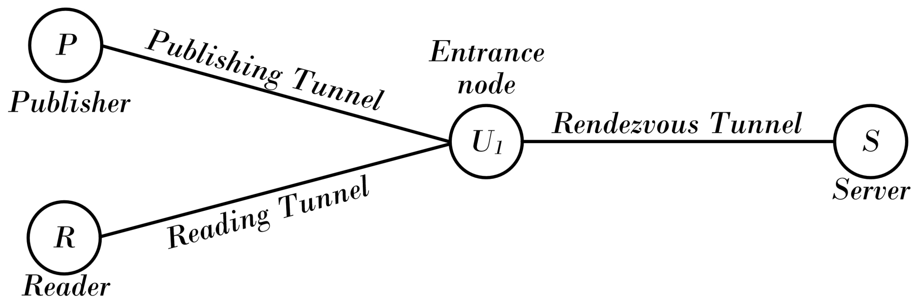


Fig. 1. Three-tunnels system

Servers in our system store shares of documents, and each share is reached through a *rendezvous tunnel* between the server and an address given by a hash of the document's name. To publish a document, the publisher first divides the document into shares, for each share finds the address of the entrance to the tunnel by hashing the document's name. Next, the publisher uses anonymous communication to reach the entrance of the rendezvous tunnel. We then use a random walk and an anonymous key exchange scheme to set keys along the rendezvous tunnel. The publisher finishes by inserting the shares into the servers through the rendezvous tunnels. A reader wanting to retrieve the document operates in a similar manner. The reader finds the address of the entrance to the rendezvous tunnels by hashing the document's name. Then, the reader uses anonymous communication to reach the entrance of the tunnels, retrieves the shares anonymously and reconstructs the document.

A. *Related work*

The core concept of providing Internet anonymity goes back to the early days of the public network and has been extensively studied since then. Chaum [3] proposed to use an intermediary proxy (relay server or Mix) whose aim is to hide the identity of the reader from the server. Later works developed Chaum's approach by deploying predefined or ad-hoc paths.

One can divide previous work into two categories. The first category includes approaches that provide a high degree of anonymity, but with the cost of high communication overhead [1], [4], [8] ensuring a predefined traffic statistics. Those solutions require continuous communication in practice, and in many cases one can assume that the statistics do not reveal activities.

The Dinning Cryptographers network [4] is an anonymous broadcast protocol based on the dining cryptographers problem. DC-net provides a very strong form of sender and receiver anonymity at the cost of broadcast. Due to this expensive broadcast communication, the scheme suffers from poor scalability and hence is unsuitable for large-scale use. Xor-Trees [8] reduces the amount of communication in comparison to DC-nets.

Buses [1] is an anonymous scheme inspired by the observation that public transit buses hide the movement patterns of passengers. Passengers are pieces of information that are allocated seats in a bus that traverses the network. Buses aims at hiding the traffic patterns as so to prevent an external adversary from linking between two communicating parties. This scheme provides sender-receiver unlinkability in the cost of traffic even when on information is transferred. However, the sender must know the intended receiver and vice versa, hence anonymity of both the sender and the receiver with respect to each other is not supported.

The second category consists of approaches that are based on the assumption that the traffic pattern does not reveal information. Those works provide different levels of anonymity with different levels of communication overhead. Onion Routing [19] uses a fixed, predefined path, which is essentially a list of intermediate proxies leading to the destination. The major advantage of onion routing is that relays cannot unravel the information received or determine destination address. Tor [7] is an advanced low-latency scheme that improves the original Onion Routing. Tor is a tunnel-based system that constructs circuits in stages, extending the circuit one hop at a time. Tor utilizes directory servers to manage available services and the access to them. Public-key and symmetric key encryption are used to encrypt messages that traverse the circuits. Tor provides sender anonymity and sender-receiver unlinkability. In addition, Tor provides receiver anonymity through *rendezvous points* and *hidden services*. The main shortcoming of Tor's hidden services is that they are provided by the content owner, tor does not consider publishing. In the present work we address this limitation by publishing the content. Another related solution is the tunnel based scheme CIAS [18]. CIAS is a low latency P2P scheme that provides anonymity for senders and receivers while adhering to strict and low bounds of delay, communication and bandwidth overheads. However, CIAS does not consider publisher anonymity either.

All works described above are aimed to provide anonymity and unlinkability to the reader and the server in the message. Non of them take into account P2P network with anonymous publishing phase. Freenet

[5] is a solution that addresses this problem. Freenet provides some degree of a server and publisher anonymity. When a user requests a document, it uses a document identifier to send the query without being aware of the server's identity or location. The weakness of Freenet is that the length of the tunnel is not controlled by the users. The tunnel might be as short as one node, and an adversary that controls the tunnel can revoke the anonymity of the reader, server and publisher.

A different technique used to provide anonymity employs secret sharing schemes to break data items into several parts and distribute them among different server. In Publius [12], the content is encrypted by a key and stored in a fixed set of servers. The encryption key is shared by Shamir's secret sharing and distributed to the servers. Retrieving is carried out by reconstructing the key, retrieving the encrypted document and decrypting it. Although Publius does concern publisher anonymity, the documents are stored in a static list of available servers, and the index is not protected. Another approach that uses secret sharing is Deniability [11]. This technique takes a different approach: instead of trying to hide the identity of the users, it blurs the connection between pieces of information and their meaning.

Free Haven [6] is an anonymous publishing system. It is made up of a number of servers, known as servnets, which agree to store and provide documents for anyone. The identities of these servnets are publicly known. Communication is carried out over a Mix-based communication layer. Free Haven provides a certain level of publisher, reader and document anonymity, but lacks in server anonymity.

B. *Organization*

In Section II we construct the system settings. Section III describes two solutions, and gives a running example. We give a thorough analysis of the anonymity of our system in Section IV. We conclude our work in Section V. Through analysis of the solutions appear in the Appendix.

II. Settings and Requirements

A. *Participants*

In P2P file sharing networks, information is stored in units called *documents*. The *publisher* of a document is the entity which places the document into the system. The *server* of a document is an entity that stores and distributes the document. Readers retrieves documents from servers. Retrieving a document in any system requires *name-index mapping*. Name-index mapping enables a user to find the location of a specific document. Usually, the name-index mapping is stored in a database known as index server. Since the index constitutes a threat to users in P2P system, our system replaces the mapping database by a publicly known hash function that hash the document name to an index (or several indexes).

B. *Anonymity Model*

In this paper we design a system that provides *anonymity* to all participants in P2P file-sharing network the publishers, the servers and the readers. One may define several types of anonymity with respect to P2P networks, we adopt the definitions and terms of [6] to define the anonymity of participating users. *Reader anonymity* means that an adversary has no way of knowing which reader on the network has retrieved a particular document. *Server anonymity* means an adversary has no way of knowing which server on the network has served this document or currently stores it. *Document anonymity* means that a server does not know which documents it is storing. *Publisher anonymity* means that an adversary has no way of knowing which user on the network has published a particular document. We also suggest a new definition for the index problem, *Index anonymity* means that an adversary has no way of knowing what was added to or retrieved from the index. Note that the users maintain their anonymity also in respect to each other. For example, the reader maintains its anonymity even when retrieving a document from a server, namely, the server does not know who the reader is, and vice versa. In particular, a server that acts as a reader does not know whether it got the document from itself.

According to the terminology of [14], anonymity means that a user is unidentifiable within a group of users, the *anonymity set*. Unlinkability of items (e.g., users, messages) means that an attacker cannot sufficiently distinguish whether these items are related or not. Sender–receiver unlinkability is provided against an adversary who is neither the sender nor the receiver of the messages.

C. Adversary Model

In anonymous P2P networks, the participating entities, the server, the reader and especially the publisher do not want their identity to be revealed. The adversary's goal is to link specific content to a participating party in the system and thereby to identify the players.

Similarly to other schemes for anonymous P2P networks (e.g. Tor [7]), the adversary in our model is assumed to be *Honest but Curious*, which means that the adversary can control nodes in the network, but is obligated to follow the algorithms.

We assume that the adversary can control at most t network nodes, where t is smaller than the total number of nodes. We also assume that the communication patterns and statistics do not reveal information. As the statistics and pattern of message traffic is distributed in a fixed (say normal) distribution. Such assumption is used when using the well known mixes scheme [3], as for example, when mixes are used and only one message is sent in the entire network the identities of the sender and the receiver are obviously revealed.

D. Index

Retrieving a document in any system relies on the *name-index mapping*. The index mapping enables a user to find the location of a specific document. Usually, the indexing mapping is stored in a database or databases called *index server(s)*. On one hand, every network has to have index mapping. In order to retrieve a document, the reader has to have a way to locate it. On the other hand, index server is of a threat to anonymity in several ways. First, if the adversary controls or eavesdrops the index server, the anonymity of the users that communicate with it (the publisher that publishes a new document in the index and the reader who searches for a document) may be violated. Moreover, if the index mapping includes controversial content, legal procedures may be invoked against the user that holds the index server. Hence, our system design has no index server, the index mapping is performed by the users themselves. Each share in the system has an index entry that includes three parameters. First, the entrance node to a tunnel, denoted as U_1. Second, the reader's seed (which is used in the collective encryption of the document) denoted by s_0. And U_1's session identity, denoted by ID_1. The publisher (and later the reader) calculates the index mapping according to the document name d_{name} (see Algorithm 4 lines 2-4). The publisher uses *Distributed Hash Table* (DHT) and two different known hash functions (or the same hash with different known keys) to compute the index entry. A DHT receives a key as an input, and outputs an ID of a node in the system. The hash functions $h : \{0,1\}^\alpha \rightarrow \{0,1\}^\beta$ receives a variable size input α and outputs, say 128bits, thus ensuring that the probability of a collision is low.

III. Solution Architecture

In this Section we propose two solutions. The basic solution is simple, but still provides anonymity to all participating users. The basic solution appears in Algorithms 1-3. The advanced solution extends the basic solution. In addition to anonymity for all participating users, this solution provides a certain degree of traffic analysis resistance to the publisher. Moreover, the advanced solution conceals the content of the published document from the entrance node during the publishing phase. We explain this solution in details, we give the algorithms that construct the solution, we use figures to follow a running example. Later in section IV we prove the anonymity of this solution. Note that in both solutions, documents are first divided into shares using an (n, k) IDA [15].

A. Basic Solution

There are basically two phases in this solution. First, in the *publication* phase, the publisher sends each share encrypted to a server. Next, in the *retrieval* phase, the reader retrieves the shares from the servers and reconstructs the document.

The overall flow of the basic solution is as follows. First, the publisher divides the document into shares using an (n, k) IDA. Then, for each share, the publisher uses hash functions to build the index mapping. Next, the publisher encrypts the share and sends the encrypted share to the entrance node through a sender anonymity tunnel, the sender tunnel. The entrance node initiates a random walk and builds a rendezvous tunnel. The publisher then sends the encrypted share along the tunnel. Each node along the rendezvous tunnel encrypts the share and forwards it to the server. A reader who wants to retrieve the document operates in a similar manner. For each share, the reader constructs the index in the same way, then the reader uses sender anonymity tunnel (the reader tunnel) and reaches the entrance node. The entrance node forwards a query message to the server through the rendezvous tunnel. Then the server sends the share back along the rendezvous tunnel. Each user along the rendezvous tunnel decrypts the share. The entrance node uses the reader tunnel and sends the share to the reader. The reader then decrypts the share. As soon as enough shares are successfully retrieved, the reader reconstructs the document.

Publication: The publication phase is shown in Algorithm 1. First, by using an (n, k) IDA, the publisher divides the document into shares. Then, for each share, the publisher uses a hash function on the name of the document concatenated to the number of the share ($h(d_{name}||i)$, the publisher creates the index for all the shares of the document. The index of each share includes an entrance node - U_1, a reader seed - s_0 and an ID for the entrance node - ID_1. Then, the publisher encrypts the share sh with the reader seed obtaining $sh \oplus G(s_0)$. The publisher finishes by sending the encrypted share to the entrance node of the rendezvous tunnel via a sender anonymity tunnel. In order to build the rendezvous tunnel, the first node of the rendezvous tunnel (U_1) initiates a random walk (each user chooses randomly, among all network users, the next node in the tunnel) of length ℓ ($\ell > t$). During the construction of the tunnel, each user encrypts the document and forwards it to the server. To encrypt the document, each user creates randomly a session key (s_i) and XORs the received share with $G(s_i)$, where G is a pseudo random generator. When the message arrives at the server, the server saves the encrypted share $sh \oplus \bigoplus_{i=0}^{\ell} G(s_i)$.

In Algorithm 1 line 2, the publisher divides the document into n shares. In lines 4-6, the publisher creates the index for each share and finds the entrance node - U_1. In line 7 the publisher encrypts the share with the reader seed. In line 8 the publisher sends the insert share message to U_1. The insert share message contains the length of the rendezvous tunnel. Each node along the tunnel receives the insert share message in line 11. In line 14-15, each internal node creates a session key (s_i) and encrypts the share by XORing the received share with $G(s_i)$, where G is a pseudo random generator. In lines 16-17, the node chooses the next node along the tunnel and a new identity number for the next node. In line 18, the node forwards the insert share message to the next node. When the message arrives to the server, it saves the encrypted document in line 21.

Retrieval: The retrieval process is constructed from two messages - a query message and a response message (see Algorithm 2 and Algorithm 3 respectively). A reader, who wants to retrieve the document, creates the index mapping in the same way as the publisher. Note that the reader needs k shares to be able to reconstruct the document. Then, the reader creates a sender anonymity tunnel, the reading tunnel, to U_1, and sends a query message to the server through the rendezvous tunnel and U_1. Each internal node along the rendezvous tunnel forwards the query message. When the server receives the query message, it sends the encrypted share back along the rendezvous tunnel. Each internal node XORs the message with $G(s_i)$ and forwards the message to the previous node along the rendezvous tunnel. The reader receives the response message, and decrypts the share with the reader seed. As soon as the reader accumulates k shares, the reader reconstructs the document.

In Algorithm 2 lines 3-5, for each share, the reader creates the index mapping. Then, the reader creates

Algorithm 1 - Publication

```
1: Publisher:
2: $\{shares\} \leftarrow IDA(d)$    {dividing the document into $n$ shares}
3: for $x = 1$ to $n$ do
4:     $U_1 \leftarrow DHT(d_{name}||x)$;    {entrance node}
5:     $s_0 \leftarrow h_1(d_{name}||x)$;    {reader seed}
6:     $ID_1 \leftarrow h_2(d_{name}||x)$;    {entrance node's ID}
7:     $\tilde{sh_x} \leftarrow sh_x \oplus G(s_0)$;    {encrypt the the with reader seed}
8:     $send(insertShareMsg, ID_1, \tilde{sh_x}, length)\ to\ U_1$;    {send the encrypted share to $U_1$ through the
       publisher tunnel. The rendezvous tunnel is of length of *length* nodes}
9: end for

10: Node i:
11: $receive(insertShareMsg)$;
12: if $length > 0$ then
13:     $lastNode \leftarrow FALSE$;    {this is not the last node}
14:     $s_i \leftarrow h_1(random())$;    {session key}
15:     $\tilde{sh} \leftarrow \tilde{sh} \oplus G(s_i)$;    {encrypt the document with the session key}
16:     $U_{i+1} \leftarrow DHT(random())$;    {choose in random the next node in the tunnel}
17:     $ID_{i+1} \leftarrow h_2(random())$;    {next node's session ID}
18:     $send(insertDocMsg, ID_{i+1}, \tilde{sh}, length - 1)\ to\ U_{i+1}$;    {forward the message along the tunnel}
19: else
20:     $lastNode \leftarrow TRUE$;    {this is the server}
21:     save $\tilde{sh}$;    {the server saves the encrypted share}
22: end if
```

Algorithm 2 - Query Message

```
1: Reader:
2: for each share $x$ do
3:     $U_1 \leftarrow DHT(d_{name}||x)$;
4:     $s_0 \leftarrow h_1(d_{name}||x)$;
5:     $ID_1 \leftarrow h_2(d_{name}||x)$;    {the index entry}
6:     $send(queryMsg, ID_1)\ to\ U_1$;    {send a query message along the tunnel through $U_1$}
7: end for

8: Node i:
9: $receive(queryMsg)$;
10: if $lastNode == FALSE$ then
11:     $send(queryMsg, ID_{i+1})\ to\ U_{i+1}$;    {forward the query message to the next node in the tunnel}
12: else
13:     GoTo Algorithm 3;
14: end if
```

a sender anonymity tunnel - the reading tunnel, and sends a query message through the rendezvous tunnel to U_1 in line 6. Each internal node along the rendezvous tunnel forwards the query message in line 11. The server in line 13 continues according to Algorithm 3.

In Algorithm 3, the server sends the encrypted share to the previous node along the rendezvous tunnel in line 3. Each internal node in lines 5-7 removes the session key and forwards the message to the previous

Algorithm 3 - Share Response Message

```
1:  Node i:
2:  if $lastNode == TRUE$ then
3:      $send(shareRspnsMsg, ID_i, \tilde{sh})$ to $U_{i-1}$;
4:  else
5:      $receive(shareRspnsMsg)$;
6:      $\tilde{sh} \leftarrow \tilde{sh} \oplus G(s_i)$;                {remove the session key $s_i$}
7:      $send(shareRspnsMsg, ID_i, \tilde{sh})$ to $U_{i-1}$;
8:  end if

9:  Reader:
10: $receive(shareRspnsMsg)$;
11: $sh \leftarrow \tilde{sh} \oplus G(s_0)$;          {the reader reconstructs the shares}
12: insert $sh$ to $\{shares\}$                         {the reader collects the shares}
13: if $|\{shares\}| = k$ then
14:     $d \leftarrow IDA^{-1}(\{shares\})$             {the reader reconstructs the document}
15: end if
```

node along the rendezvous tunnel. The reader, in lines 10-14, receives the share response message, decrypts the share with the reader seed, collects k shares and reconstructs the document.

Discussion on the Basic Solution: The basic solution provides anonymity to *all* participating entities. The anonymity of the publisher and the reader is provided by the publishing tunnel and reading tunnel, respectively. The anonymity of the server is provided by the rendezvous tunnel. The main problem inherent in the basic solution is that the shares are sent to the entrance node U_1 encrypted with a key that is associated with the document's name. Hence, if U_1 is controlled by an active adversary, it can perform a brute force attack on all documents' names and find that U_1 is the entrance node of a specific share.

The advanced solution deals with this drawback.

B. Advanced Solution

The advanced solution extends the basic solution with two main modifications. First, we use a new key exchange protocol to securely exchange keys between the publisher and nodes along the rendezvous tunnels. After the key exchange is complete, the publisher encrypts the shares with the exchanged symmetric keys and sends them to the servers through the rendezvous tunnels. The second modification is that the publisher sends the encrypted shares through node U_2, the second node along the tunnel, and not through U_1. In this way, the brute force attack is not applicable during publication. Some time after the share is safe and securely encrypted in the server, the publisher binds U_1 to U_2 and the rest of the rendezvous tunnel. Another issue that the advanced solution solves is that in order to link between the document and the publisher, the traffic analyzer has to save the traffic analysis information throughout the publishing and the binding process. The time interval between the publishing and the binding may be long enough to frustrate the traffic analyzer. For example, in the basic solution, a traffic analyzer may locate the publisher if the traffic analyzer stores information regarding the traffic during the publication phase. However, in the advance solution, we separate the insertion of the shares and the ability of any user in the system to understand their content. Hence, the traffic analyzer has to store much more information in order to be able to trace the publisher using the stored traffic.

The overall flow of the algorithm is as follows. First, similarly to the basic solution, the publisher divides the document into shares, and uses hash functions over the document name to build the index mapping. Next, for each share, the publisher selects in random the second node along the tunnel, U_2. Then we use random walk and computational theoretic scheme to set symmetric keys between the publisher and each user along the rendezvous tunnel (excluding U_1). After the keys exchange is over, the publisher inserts the document to the server through U_2. Later, after a while, the publisher binds U_1 to the tunnel and the document becomes available to other peers to read. The retrieval process is exactly the same as in the basic solution. The reader who wants to retrieve the document constructs the index in the same way. Then the reader uses anonymous communication to reach U_1, the entrance of the rendezvous tunnel. The reader then retrieves enough shares and reconstructs the document.

In the remaining of this Section, we begin with a general description to the key exchange protocol. Then, for each step we give a brief description followed by a pseudo code and an example. The example follows the Algorithms. In this example, a publisher builds a tunnel of four users (see Figure 2) three of them are participating in the publishing phase $U_2 - U_4$, and U_1 which is binded to the tunnel after the publishing phase. The publisher publishes a share anonymously in the tunnel. We continue the example describing how a reader retrieves the share anonymously through the tunnel. For every algorithm we show a figure containing the important information about the algorithm. For simplicity, we omit the IDA phase, and assume that documents are published in one piece.

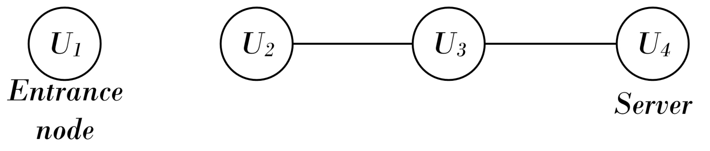


Fig. 2. Example of anonymity tunnel of length four

Anonymous Key Exchange Protocol: The goal of the key exchange protocol is to create symmetric keys in a way that two non-contiguous users along the tunnel will not know that they are taking part of the same key exchange. The key exchange protocol is designed under the assumption of an honest but curious adversary and is based on the Decision Diffie-Hellman (DDH) problem [2]. $Alice$ exchanges keys with Bob, $Carol$ and $David$, (Figure 3). We assume that each node knows only its immediate neighbors. To exchange keys, $Alice$ sends a message, throw a sender anonymity tunnel, the message containing three segments, with each segment destined to a different node. We now focus on the last segment sent from $Alice$ to $David$ (the other segments are built in the same way). $Alice$ generates a random number R_3 ($R_3 \in_R q$), and calculates $g^{R_3} \bmod p$, where p is a large prime number, and q is another large prime such that $q|p-1$ and g is a generator of a multiplicative group of size q modulo p. $Alice$ sends $g^{R_3} \bmod p$ to Bob. Bob selects a random number S_1, calculates $g^{R_3 \cdot S_1} \bmod p$ and forwards the message to $Carol$. $Carol$ does the same as Bob and forwards the message to $David$. $David$ as well selects a random number S_3 and calculates $g^{R_3 \cdot S_1 \cdot S_2 \cdot S_3} \bmod p$. The result is used as the symmetric key of $David$ and $Alice$. $David$ calculates and sends $g^{S_3} \bmod p$ to $Carol$. $Carol$ calculates and sends $g^{S_3 \cdot S_2} \bmod p$ to Bob. Bob calculates and sends $g^{S_3 \cdot S_2 \cdot S_1} \bmod p$ to $Alice$. $Alice$ calculates $g^{S_3 \cdot S_2 \cdot S_1 \cdot R_1} \bmod p$. Now only $David$ and $Alice$ know the symmetric key.

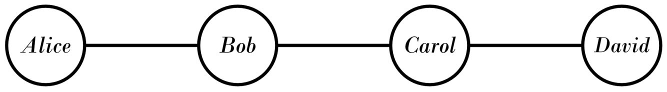


Fig. 3. Alice exchanges symmetric keys

In order to exchange keys the publisher has to send the *preliminary message* and receive the *confirmation message* we describe these algorithms in Algorithm 4 and Algorithm 5 respectively. We describe the algorithms according to Figure 2. Note that when we use the notation of g^k we actually mean $g^k \bmod p$.

Preliminary Message - The goal of the preliminary message is to build the tunnel. Moreover, the preliminary message is the first message of two messages that used to replace symmetric keys between the publisher and the users along the tunnel. The preliminary message is constructed from several segments, each segment is destined to a different user along the tunnel. The content of a segment is a generator g raised in the power of a random number: $\{g^{R_2}, g^{R_3}, \cdots\}$, where g^{R_2} is destined to U_2, g^{R_3} is destined to U_3 and etc. Then, the publisher selects U_2 in random and sends a preliminary message to U_2. Note that the communication between the publisher and U_2 is anonymous by using a sender anonymity tunnel. Node U_2 initiates a random walk in the network, and forwards the message along the rendezvous tunnel to the server. A user along the tunnel that receives the preliminary message extends the random walk and generates keys. These keys are used in the symmetric key exchange protocol. Then the user forwards the preliminary message to the next node along the tunnel. Note that user i forwards only segments that are not destined to it, since they did not arrive at their destination yet. When the server receives the preliminary message, it adds a key and continues according to the confirmation message in Algorithm 5.

Algorithm 4 - Preliminary Message

```
1:  Publisher:
2:  $U_1 \leftarrow DHT(d_{name})$;                              {entrance node}
3:  $s_0 \leftarrow h_1(d_{name})$;                              {reader seed}
4:  $ID_1 \leftarrow h_2(d_{name})$;                             {entrance node's ID}
5:  $U_2 \leftarrow DHT(random())$;                              {second node in the tunnel}
6:  $s_1 \leftarrow h_1(random())$;                              {$U_1$'s seed}
7:  $ID_2 \leftarrow h_2(random())$;                             {$U_2$'s ID}
8:  $Msg \leftarrow Message[length]$;                            {an empty array of messages}
9:  for $i = 2$ to $length$ do
10:     $R_i \leftarrow random()$;                               {random for key exchange}
11:     $m_i \leftarrow g^{R_i}$;                                {g is a generator}
12:     $Msg[i] \leftarrow m_i$;
13: end for
14: $index \leftarrow 2$;                                        {the position of the message along the tunnel}
15: $send(plmryMsg, Msg, ID_2, length, index)\ to\ U_2$;

16: Node $i$:
17: $receive(plmryMsg)$;
18: $i \leftarrow index$                                         {this is the i-th node along the tunnel}
19: for $j = i$ to $length$ do
20:     $k_{i,j} \leftarrow random()$;                           {random key}
21:     $Msg[j] \leftarrow m_j^{k_{i,j}}$;                       {raise $m_j$ to the power of $k_{i,j}$}
22: end for
23: $S_i \leftarrow h_1(m_i)$;                                   {symmetric session key of user $i$}
24: if $i < length$ then
25:     $lastNode \leftarrow FALSE$;                             {this is not the last node}
26:     $Msg[i] \leftarrow NULL$;                                {the message in this position need not to be forward}
27:     $U_{i+1} \leftarrow DHT(random())$;                      {choose in random the next node in the tunnel}
28:     $ID_{i+1} \leftarrow h_2(random())$;                     {next node's session ID}
29:     $send(plmryMsg, Msg, ID_{i+1}, length, index + 1)\ to\ U_{i+1}$;   {forward the preliminary message
        along the tunnel}
30: else
31:     $lastNode \leftarrow TRUE$;                              {this is the server}
32:     GoTo Algorithm 5;
33: end if
```

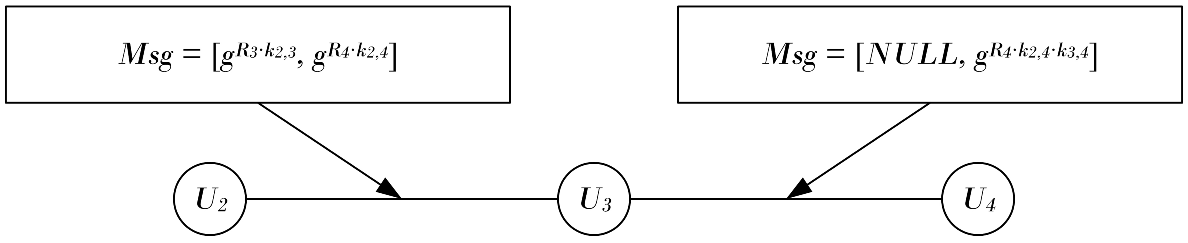

$Msg = [g^{R_3 \cdot k_{2,3}}, g^{R_4 \cdot k_{2,4}}]$
$Msg = [NULL, g^{R_4 \cdot k_{2,4} \cdot k_{3,4}}]$
U_2
U_3
U_4

Fig. 4. Example of preliminary message

Example for the use of the content of the preliminary message is shown in Figure 4. Assume that publisher P wants to publish document d in a tunnel of a size four (see the tunnel in Figure 2). According to Algorithm 4, P generates the index (lines 2-4) and finds that U_1 is the entrance node. Then in lines 4-7, P finds in random the second node along the tunnel (U_2). The publisher in lines 8-12, creates a preliminary message. The message has three segments: $\{g^{R_2}, g^{R_3}, g^{R_4}\}$, one generator in the power of a random number to each user along the tunnel. g^{R_2} is destined to U_2, g^{R_3} to U_3 and g^{R_4} to U_4. Then (line 15) the publisher sends the message along the tunnel. User U_2 receives (line 17) the message from the publisher (since the message was sent through a sender anonymity tunnel, U_2 does not know the identity of the publisher) and generates (lines 19-21) three random keys $\{k_{2,2}, k_{2,3}, k_{2,4}\}$, then U_2 uses those keys to raise the preliminary messages by the power of the keys: $\{g^{R_2 \cdot k_{2,2}}, g^{R_3 \cdot k_{2,3}}, g^{R_4 \cdot k_{2,4}}\}$. U_2 saves (line 23) the symmetric key $s_2 = h_1(g^{R_2 \cdot k_{2,2}})$. Then in lines 25-29, since U_2 removes the segment from the message. U_2 finds the next node U_3 and forwards the message to U_3. Node U_3 does the same and forwards the last segment of the preliminary message to U_4. In Figure 4, the content of the preliminary message sent from U_2 to U_3 appears on the left side, and the content sent from U_3 to U_4 appears on the right side. Node U_4 receives the last segment of the message, generates key, raise the message in the power of the symmetric key and continues according to Algorithm 5.

Confirmation Message - The second step in the initiation of the anonymous tunnel is sending back the *confirmation message* (see Algorithm 5). The server initiates the confirmation message. When the message leaves the server it contains only the last segment with the server's key. When the message arrives to an internal node in the rendezvous tunnel, the node add its keys to the received segments, then the user returns its segment to the message and forwards the message back along the tunnel. Node U_2 forwards the message to the publisher. The publisher adds its key to the message and recovers the symmetric keys. Then, the publisher continues according to Algorithm 6.

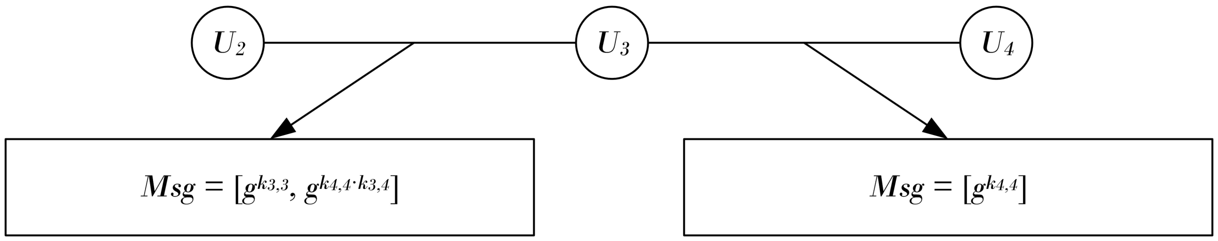


Fig. 5. Example of confirmation message

Example for the use of the content of the confirmation message is shown in Figure 5. The server (U_4 in this example) changes in lines 3-4 the last segment of the previous message to $m_4 = \{g^{k_{4,4}}\}$ and sends the confirmation message to U_3. When the message arrives to U_3, node U_3 (lines 6-11) adds its key to the last segment $m_4 = \{g^{k_{4,4} \cdot k_{3,4}}\}$, creates its segment $m_3 = g^{k_{3,3}}$ and forwards the message to U_2. The message that sent from U_4 to U_3 and message U_3 sends to U_2 are shown in Figure 5. Node U_2 does the same and forwards the message to the publisher. In lines 14-19 the publisher recovers the symmetric keys

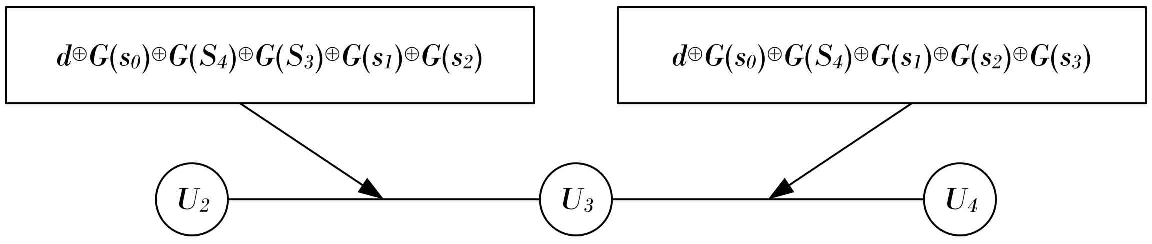
Algorithm 5 - Confirmation Message

```
1: Node i:
2: if $lastNode == TRUE$ then
3:     $Msg[i] \leftarrow g^{k_{i,i}}$;
4:     $send(cnfrmMsg, Msg, ID_i, index - 1)\ to\ U_{i-1}$;   {send a confirmation message back along the tunnel}
5: else
6:     $receive(cnfrmMsg)$;
7:     for $j = i + 1$ to $length$ do
8:         $Msg[j] \leftarrow m_j^{k_{i,j}}$;   {add the random key to the messages}
9:     end for
10:    $Msg[i] \leftarrow g^{k_{i,i}}$;   {adding the message to change keys}
11:    $send(cnfrmMsg, Msg, ID_i, index - 1)\ to\ U_{i-1}$;   {forward the confirmation message back along the tunnel, note that $U_2$ forwards the message back to the publisher}
12: end if

13: Publisher:
14: $receive(cnfrmMsg)$;
15: for $i = 2$ to $length$ do
16:     $Msg[i] \leftarrow m_i^{R_i}$;   {this removes the random from the messages and reveals the shared key}
17:     $S_i \leftarrow h_1(m_i)$;   {symmetric session key of user $i$}
18: end for
19: GoTo Algorithm 6;
```

and continues according to Algorithm 6.

Publication: After the tunnel is built, the publisher can insert (publish) the document in the tunnel (see Algorithm 6). The publisher encrypts the document using one symmetric session key per user along the tunnel and two additional keys - the reader key (s_0) and U_1's key. Then the publisher sends the document to the server through U_2. Each internal node along the rendezvous tunnel removes the symmetric session key, adds new key and forwards the message to the server. The server saves the encrypted document. After the document was inserted to the tunnel, the publisher binds between U_1 to U_2 and the rest of the rendezvous tunnel.



$d \oplus G(s_0) \oplus G(S_4) \oplus G(S_3) \oplus G(s_1) \oplus G(s_2)$
$d \oplus G(s_0) \oplus G(S_4) \oplus G(s_1) \oplus G(s_2) \oplus G(s_3)$
U_2
U_3
U_4

Fig. 6. Example of insert document message

Example for the use of the content of the insert document is shown in Figure 6. In Algorithm 6 lines 2-6, the publisher encrypts the document with the reader's and U_1's session keys: $\tilde{d} \leftarrow d \oplus G(s_0) \oplus G(s_1)$. Then the publisher encrypts the message with the keys of the users and sends the document along the tunnel. Each user along the tunnel (lines 10-15) receives the message, XORs it with the symmetric session key, for example, U_2 does $\tilde{d} \leftarrow \tilde{d} \oplus G(S_2)$. Then U_2 creates new session key s_2 and encrypts the message

Algorithm 6 - Insert Document

```
1:  Publisher:
2:  $\tilde{d} \leftarrow d \oplus G(s_0) \oplus G(s_1)$;   {encrypt the document with reader's and $U_1$'s session keys}
3:  for $i = 2$ to $length$ do
4:      $\tilde{d} \leftarrow \tilde{d} \oplus G(s_{i,i})$;   {encrypt the document with symmetric keys}
5:  end for
6:  $send(insertDocMsg, ID_2, \tilde{d})\ to\ U_2$;   {insert the encrypted document into the tunnel}
7:  $wait\ random\ time$;
8:  $send(bindMsg, ID_1, U_2, ID_2, s_1)\ toU_1$;   {the publisher send anonymously the binding message to $U_1$}

9:  Node $i$:
10: $receive(insertDocMsg)$;
11: $\tilde{d} \leftarrow \tilde{d} \oplus G(s_{i,i})$;   {remove the symmetric key}
12: if $lastNode == FALSE$ then
13:     $s_i \leftarrow h_1(random())$;   {new session key}
14:     $\tilde{d} \leftarrow \tilde{d} \oplus G(s_i)$;   {add new session key}
15:     $send(insertDocMsg, ID_{i+1}, \tilde{d})\ to\ U_{i+1}$;
16: else
17:     $save\ \tilde{d}$;   {the server saves the encrypted document}
18: end if
```

with it: $\tilde{d} \leftarrow \tilde{d} \oplus G(s_2)$. Then U_2 forwards the message to U_3. Node U_3 does exactly the same, removes its symmetric key from the message, and adds a new key. In Figure 6 the content of the messages sent from U_2 to U_3 and U_3 to U_4 are shown. As soon as the server (U_4) receives the encrypted document (line 17), it saves it in its machine. After a random period of time, (lines 7-8) the publisher sends a binding message anonymously to U_1, this message contains the information needed to bind U_1 to U_2 and the tunnel. At this point, the document and the tunnel are ready, and readers can download the document.

Retrieval: The retrieval of a document is done with two messages in the same way as in the basic solution. First, the reader uses the tunnel and sends a *query message* to the server (see Algorithm 2). Then the server sends back the document in the tunnel in a *document response message* (see Algorithm 3).

Query Message - In Algorithm 2 the reader creates the index entry, and sends anonymously a query message to U_1. Each internal user along the tunnel forwards the message to the next node. When the message arrives to the server, it continues according to Algorithm 3).

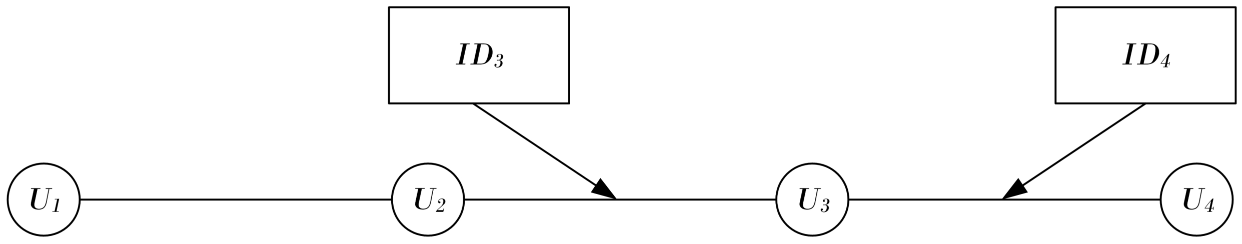


Fig. 7. Example of query message

Example for the use of the content of the query message is shown in Figure 7. In Algorithm 2 lines 2-5, the reader creates the index entry, and sends a query message that includes ID_1 to U_1. Each internal user (lines 7-9) changes the ID in the message and forwards the message to the next node (U_1 changes the ID to ID_2 and forwards to U_2 which in its turn changes the ID to ID_3 and forwards to U_3). As soon as the server receives the message (line 11), it continues according to Algorithm 3.

Document Response Message - In Algorithm 3 the server sends the encrypted document along the tunnel to the reader. Each node in the tunnel removes its session key and forwards the message. When the reader receives the document it removes the reader key and reconstructs the document.

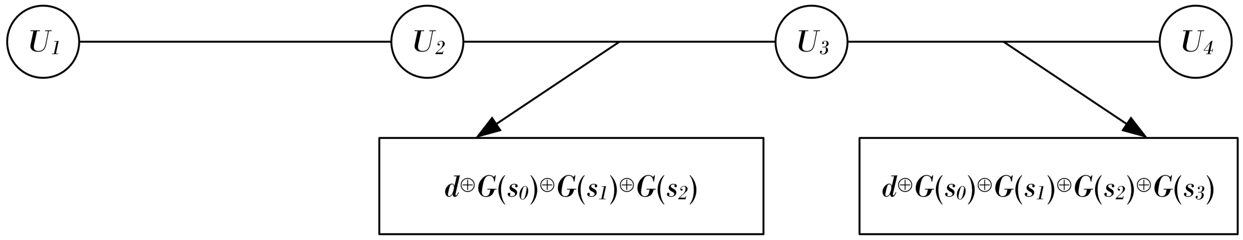

U_1
U_2
U_3
U_4
$d \oplus G(s_0) \oplus G(s_1) \oplus G(s_2)$
$d \oplus G(s_0) \oplus G(s_1) \oplus G(s_2) \oplus G(s_3)$

Fig. 8. Example of document response message

Example for the use of the content of the document response message is shown in Figure 8. The server (U_4) in line 3 send the encrypted document $\hat{d}$ to U_3. While the message arrives at U_3 (lines 5-7) U_3 removes its session key: $\hat{d} \leftarrow \hat{d} \oplus G(s_3)$ and forwards the message to U_2. Node U_2 removes its session key and forwards the message to U_1. Node U_1 removes its session key and forwards the message to the reader. The content of the message the reader receives is $d \oplus G(s_0)$. Then the reader (lines 10-11) XORs the message with the reader seed ($G(s_0)$) and reconstructs the document d.

Throughout the algorithms, any adversary that holds less than the entire tunnel can not link between the messages and document they store. In the next Section we prove the anonymity of our system.

Remarks:

Circles in the tunnel - Essentially the length of the tunnel in our system is $t+1$, where t is the number of nodes that are potentially held by the adversary. An important observation is that regardless the length of the tunnel, the number of distinct nodes must be $t+1$, otherwise the anonymity of the server is revoked. Since nodes along the tunnel are chosen in random, there is a possibility that the same node appears more than once along the tunnel. If a node appears more than once along the tunnel we have a circles tunnel with less than $t+1$ distinct nodes. Hence we will construct a tunnel with more than $t+1$ nodes such that the probability that we have less than $t+1$ distinct nodes tends to zero.

$$\binom{N}{t} t^{\ell} >> \alpha \Rightarrow \binom{N}{t} \frac{t^{\ell}}{\beta} >> \alpha/\beta$$

Note that α/β is the probability to have a tunnel with less than $t+1$ distinct nodes. We show that the expression on the left hand in the above equation (which is greater than our probability) decreases exponentially with the growth of ℓ, this implies that the probability that the tunnel has less than $t+1$ nodes decreases exponentially (or faster) with the growth of ℓ.

$$\binom{N}{t} \frac{t^{\ell}}{\beta} = \binom{N}{t} \frac{t^{\ell}}{N^{\ell}} = \binom{N}{t} (\frac{t}{N})^{\ell}$$

Since $\binom{N}{t}$ is constant and $t << N$, we have expression that decreases exponentially in ℓ. Note that for real life situation, the number of nodes in the system N can be as high as millions and a strong adversary can control about dozens nodes. If we calculate the equation for $N = 1M, t = 20, \ell = 23$ the result is that the probability to have less than $t+1$ distinct nodes is $3.4 \cdot 10^{-7}$.

Fault Tolerance - Let us assume that nodes are expected to leave the network with probability P. If a node leaves the network, all the shares that this node is part of their rendezvous tunnels become unavailable.

Hence, the probability that a tunnel is active is $P_t = (1-P)^\ell$, where ℓ is the tunnel length. Assuming that $N >> \ell \cdot n$ (where N is the number of nodes in the network and n is the number of shares created for each document), for simplicity we can say that with a good approximation there is no intersection between different tunnels. Since we use an (n, k) IDA, unless at least k out of n tunnels are active, the document is not available. Hence, the probability that the document is available is $\sum_{i=k}^{n} \binom{n}{i} {P_t}^i (1-P_t)^{n-i}$. With careful selection of n and k, we can achieve a high probability for document's availability.

IV. Anonymity Analysis

In this Section we define the anonymity of the participating users and prove their anonymity in the settings of the advanced solution. The anonymity definitions are derived from Pfitzmann at al. [14]. Anonymity set is the set of all possible subjects. For example, the anonymity set of the publisher is the group of all publishers in the system.

Publisher Anonymity - The publisher is anonymous if it is not identifiable[1] within the set of the publishers (its *anonymity set*) in the system.

Reader Anonymity - The reader is anonymous if it is not identifiable within the set of the readers (its *anonymity set*) in the system.

Server Anonymity - The server is anonymous if it is not identifiable within the set of the servers (its *anonymity set*) in the system.

Document Anonymity - The server has document anonymity if it can not know the content of the document it stores.

Publisher and reader anonymity are naturally derived from the sender anonymity tunnels they use. The publisher and the reader that communicate with the entrance node U_1, always do so behind the protection of a sender anonymity tunnel.

We prove sender anonymity tunnel based on Theorem 1 (see Appendix). Roughly speaking, Theorem 1 claims that an adversary that does not control the entire tunnel, can not distinguish between the messages of two different documents. Hence, assuming that the adversary does not control the entire tunnel, the identity of the server remains hidden. We obtain server anonymity.

Document anonymity is proved by Theorem 1 and Theorem 2. Theorem 2 claims that the adversary does not obtain any knowledge about the document during the publication phase. Since the server obtains server anonymity, the anonymity of the server is preserved even with respect to itself. Therefore, the server can not obtain more information about the document. Hence we have document anonymity.

A. Limitations

Server anonymity is not maintained in a special case that the adversary control both the publisher and the last node before the publisher. E.g., in Figure 2 the publisher controls the publisher and U_3. In this case, the key that U_3 holds can be matched to the key that the publisher holds. Hence, since U_3 knows the document, and knows that the server is U_4, the server does not obtain server anonymity. To address this problem we can enhance the symmetric key exchange protocol, and use it to change keys between every pair of nodes in the tunnel.

V. Conclusion and Future Work

In this work we introduced a new anonymous key exchange protocol under the assumption of an honest but curious adversary. We showed how to build a rendezvous tunnel. We presented two solutions to anonymous P2P networks. The first solution is simple, as it uses the rendezvous tunnel without key exchange. The second solution enhances the first solution by using the rendezvous tunnel with the symmetric key exchange protocol. Our solutions thus provide overall anonymity to **all** participating users.

[1]By not identifiable we mean that the probability that this is the publisher that published document d, is not higher than the probability of other publishers to be the publishers of the document

The following paragraphs discuss future research directions.

Decoy path - In our system the server knows that it holds a document (although it does not know the content of document). A decoy path is an extension of the original path length ℓ by Λ nodes (a total of $\ell+\Lambda$ nodes in the tunnel). Some of the additional nodes are selected as decoy servers, where each decoy server receives a dummy data item. The size of the dummy data is equal to the size of the document. When a server receives a document in the document response message, it replaces the document received with the document held by it. Neither of the servers knows whether it plays a role of a real or decoy receiver.

Tunnel length - The tunnel length in our tunnel is greater than the number of nodes that the adversary controls. In case of strong adversary we get a very long tunnel. We suggest to build shorter tunnels in a cautious way. The user can calculate the probability that the adversary holds more than t nodes for a parameter ℓ, where ℓ is the tunnel length. According to a threshold probability the user can choose the tunnel length.

Distributed entrance - The Achilles heel of our system is U_1, the entrance node of the rendezvous tunnel. Node U_1 can perform a brute force attack and find the document that was published and is served by it. The problem arises from very basic fact that an independent reader has to have a way to find the entrance node of a specific document. Therefore, if an independent peer can, U_1 can. Although the anonymity of the participating users is not revoked, U_1 can effect the availability of the document. Hence, a different approach is needed. Instead of holding one node as the entrance to the tunnel, we can think of a cloud of nodes that can compute the tunnel together. In this way, the responsibility and hence the ability of a specific user to harm the system is reduced.

Adversary model - The adversary in our model is honest but curious. This assumption limits the capabilities of the adversary. A more powerful system might provide anonymity with a more powerful adversary. An approach that was considered in shadowWalker [13] might be suitable to cope with a more powerful adversary.

References

[1] A. Beimel and S. Dolev. Buses for anonymous message delivery. *Journal of Cryptology*, 16(1):25–39, 2003.

[2] D. Boneh. The decision diffie-hellman problem. In *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.

[3] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), February 1981.

[4] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untranceability. *Communication of the ACM*, 24(2), 1988.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.

[6] R. Dingledine, M. J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*. Springer-Verlag, LNCS 2009, July 2000.

[7] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[8] S. Dolev and R. Ostrovsky. Xor-trees for efficient anonymous multicast and reception. *ACM Transactions on Information and System Security*, 3(2):63–84, May 2000.

[9] O. Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000.

[10] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[11] O. Hermoni, N. Gilboa, E. Felstaine, and S. Shitrit. Deniability - an alibi for users in p2p networks. In *COMSWARE*, pages 310–317, 2008.

[12] A. R. Marc Waldman and L. Cranor. Publius: A robust, tamper-evident, censorship-resistant and source-anonymous web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, August 2000.

[13] P. Mittal and N. Borisov. Shadowwalker: peer-to-peer anonymous communication using redundant structured topologies. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 161–172. ACM, 2009.

[14] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.34.pdf, Aug. 2010. v0.34.

[15] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM*, 36(2):335–348, 1989.

[16] M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[17] A. Serjantov. Anonymizing censorship resistant systems. In *Proceedings of the 1st International Peer To Peer Systems Workshop (IPTPS 2002)*, March 2002.

[18] S. Shitrit, E. Felstaine, N. Gilboa, and O. Hermoni. Anonymity scheme for interactive p2p services. *Journal of Internet Technology*, 10:299–312, 2009.

[19] P. Syverson, D. Goldsclag, and M. Reed. Anonymous connections and onion routing. In *Proceedings of the IEEE 18th Annual Symposium on Security and Privacy*, pages 44–54, Oakland, California, 4–7 1997.

Appendix

We prove that our publishing scheme provides anonymity to publisher, reader and server in the following sense. Assume that an adversary controls at most t nodes along the path between the publisher (or reader) and the server, while the path between the publisher (or reader) and U_1 has at least $t+2$ nodes and the path between U_1 and the server has at least $t+2$ nodes. Assume that the adversary is *honest but curious* and *static*. In other words, the adversary follows protocol specifications exactly and controls the same subset of parties throughout the execution of a protocol. Assume further, that the adversary obtains information only through the execution of the protocols of Section III.

Given such an adversary, we prove the anonymity of a publisher (or reader) even if the adversary controls t nodes, including the server, and we prove the anonymity of the server even if the adversary controls t nodes, including the publisher (or a reader). We prove anonymity by showing that all the traffic that adversarial nodes receive from uncorrupted nodes is computationally indistinguishable from traffic that is sampled uniformly at random from an appropriate domain. Since an adversary can generate such traffic itself, it obtains no information by receiving it from other nodes.

Note that a more powerful adversary, e.g. one that correlates the timing of packets in different parts of a path between publisher, reader and server may be able to learn information that is unavailable to our limited adversary.

Regarding which nodes the adversary controls, consider several nodes which are contiguous along the tunnel. The protocol ensures that if the adversary controls these nodes then it knows that they are part of the same tunnel. However, the tunnel has at least $t+2$ nodes on each of its legs (publisher to U_1, reader to U_1 and U_1 to server). If all the nodes that an adversary controls are contiguous then the adversary can not be sure that two users (either publisher and server or reader and server) are communicating and thus can not link them. Furthermore, in this case the adversary does not have any information on a user beyond the fact that it is acting as a publisher or a reader or a server. Thus, if all corrupted nodes are contiguous then anonymity is maintained.

If the adversary's nodes are not contiguous then there exist at least two nodes that the adversary controls U_i and U_j, $i<j$, such that there is an uncorrupted node, U_ℓ between them, that is $i<\ell<j$. We show below that in this case it seems to the adversary that the traffic flowing through U_j is random and independent of the traffic flowing through U_i.

A. Formal Cryptographic Framework

Let Π be a protocol for T parties to compute a function g. The input of the i-th party is denoted x_i and the output of the i-th party is $g_i(x_1,\ldots,x_T)$. An adversary controls a set I of parties and receives the "view" of every party in I. The view of a party includes its input, output and all intermediate messages that it receives.

In the case we investigate, $T \geq 2t+4$, the publisher's input is D, D_{name}, the reader's input is D_{name} and all other parties do not have an input. The output of a reader is D, the output of U_i is s_i for all tunnel nodes U_i and the output of the server is $D \bigoplus_i G(s_i)$. Each tunnel node U_i has as additional output the identification tags, ID_i and ID_{i+1}, the length of the tunnel and its position in the tunnel, i.

Our formal definitions follow the full framework set in [9] and [10]. We use only those elements of the definition framework that are necessary in our setting.

Definition 1: We say that a function $\mu : I\!N \longrightarrow I\!N$ is *negligible* if for every polynomial $p : I\!N \longrightarrow [0,1]$, there exists N such that $\mu(n) < \frac{1}{p(n)}$ for every $n > N$.

Two distribution ensembles are computationally indistinguishable if no efficient algorithm can decide with good probability, whether its input is chosen according to the first distribution or the second distribution. We regard a distribution ensemble as a collection of distributions that are indexed by two parameters: a binary string a and a security parameter n represented in unary form. In our setting, a is the input for a protocol, the distribution is over all the messages of a protocol (for a subset of parties) and is induced by coin tosses of each party as it executes its part of the protocol. The security parameter 1^n determines the required length of cryptographic keys to ensure privacy of the protocol. Formally:

Definition 2: Let $X = \{X(a, 1^n)\}_{n \in I\!N, a \in \{0,1\}^*}$ and let $Y = \{Y(a, 1^n)\}_{n \in I\!N, a \in \{0,1\}^*}$ be two distribution ensembles. We say that X and Y are *computationally indistinguishable* if for every probabilistic, polynomial time algorithm DIS , there exists a negligible function μ such that for every $a \in \{0,1\}^*$:

$$|\Pr[DIS(X(a, 1^n)) = 1] - \Pr[DIS(Y(a, 1^n)) = 1]| < \mu(n).$$

We denote computational indistinguishability of two ensembles by $X \overset{c}{\equiv} Y$.

We use two cryptographic tools to prove that the adversary can't distinguish between real traffic from uncorrupted nodes and random elements of an appropriate domain. The first tool is a Pseudo-Random Generator (see [9] and [10] for formal definitions and constructions). Such a generator G takes as input a string $x \in \{0,1\}^n$ for any n and outputs a string $G(x) \in \{0,1\}^{n^c}$ for a constant $c \geq 1$. If Uni_n and Uni_{n^c} denote the uniform distribution over $\{0,1\}^n$ and $\{0,1\}^{n^c}$ respectively then

$$\{G(Uni_n)\}_n \overset{c}{\equiv} \{Uni_{n^c}\}_n \,.$$

In other words, an efficient algorithm can't tell whether a string is chosen uniformly from the uniform distribution over n^c bits or whether it is generated by the pseudo-random generator from a short (n bit) random string.

The second tool we use is the Decision Diffie-Hellman (DDH) problem (see [2] for formal definitions and a general survey). The DDH problem states that given a large prime number p, another large prime q such that $q|p-1$ and an element g that is a generator of multiplicative group of size q modulo p, it is computationally hard to distinguish between the ensembles: $\{g^x \bmod p, g^y \bmod p, g^z \bmod p\}$ and $\{g^x \bmod p, g^y \bmod p, g^{xy} \bmod p\}$, where x, y and z are chosen uniformly at random from the domain $0, \ldots, q-1$. In other words, any efficient algorithm can't tell whether the third element has a random exponent c or a product of the first two exponents xy.

B. Anonymity Claims

We begin by by analyzing the view of a block of nodes that is preceded by an uncorrupted node. We prove that the view of such nodes can be simulated efficiently without access to the actual messages that these nodes receive. That means that the nodes do not learn any information from the protocol, in the same sense that privacy is proved in multi-party protocols [10].

Let B_i, B_j be two blocks of contiguous nodes that the adversary controls such that $U_i \in B_i$ and $U_j \in B_j$. Denote by $Real^i_D$ and $Real^j_D$ the views of all nodes of B_i and B_j respectively after the execution of all protocols in Section III for a document D.

Let $Ideal^j$ be a view for B_j that is induced by choosing uniformly at random from an appropriate domain and independently any element that uncorrupted nodes generate or have as input and then constructing appropriate messages for the protocol. In the following we assume that the server is corrupt (and thus its messages do not add to the information that the adversary has). Our proofs work with with minor modifications when the server is not controlled by the adversary. Combining all the data elements into one view, we have that $Real^j_D$ is given by:

$$g^{R_j \alpha_j \beta_j} \bmod p, \ldots, g^{R_s \alpha_s \beta_s} \bmod p,$$
$$g^{k_{j,j} \alpha_j \beta_j} \bmod p, \ldots, g^{k_{s,s} \alpha_s \beta_s} \bmod p,$$
$$D \oplus \gamma_j \oplus \delta_j \oplus \gamma_{j,j} \oplus \delta_{j,j}, D \oplus \gamma_j \oplus \delta_j$$

While $Ideal^j$ is given by

$$g^{c_j} \bmod p, \ldots, g^{c_s} \bmod p,$$
$$g^{\overline{C_j}} \bmod p, \ldots, g^{\overline{c_s}} \bmod p,$$
$$D_1 \oplus \gamma_{j,j} \oplus \delta_{j,j}, D_1$$

The elements $c_j, \ldots, c_s$ and $\overline{c_j}, \ldots, \overline{c_s}$ are chosen uniformly at random and independently from $\{0, \ldots, q-1\}$. D_1 is chosen uniformly at random among all binary strings of the same length as D.

Lemma 1: Assume the existence of pseudo-random generators. Let an adversary control at most t nodes in a tunnel to publish a document D and let there be three nodes U_i, U_ℓ and U_j, $i < \ell < j$, such that the adversary controls U_i and U_j, but does not control U_ℓ. Assume that the adversary does not control the publisher. Then,

$$(Real^i_D, Real^j_D) \overset{c}{\equiv} (Real^i_D, Ideal^j)$$

Lemma 2: Assume the existence of pseudo-random generators. Let an adversary control at most t' nodes up to U_i in a tunnel to publish a document D and at most t'' nodes from U_j in a tunnel to publish a document $\overline{D}$. Let $t' + t'' \leq t$ and assume let assume that $i < \ell j$. Assume that the adversary does not control the publisher. Then,

$$(Real^i_D, Real^j_{\overline{D}}) \overset{c}{\equiv} (Real^i_D, Ideal^j)$$

Theorem 1: Assume the existence of pseudo-random generators. Let an adversary control at most t nodes in a network. For any two nodes that the adversary controls, which are not connected by contiguous corrupt nodes in a single tunnel, the adversary can not distinguish whether the nodes are part of the same publishing tunnel for document D or part of two tunnels for a document D and a document $\overline{D}$.

Proof: The theorem is an immediate consequence of Lemma 1 and Lemma 2. The adversary can't distinguish between the view along a single tunnel, $(Real^i_D, Real^j_D)$ and $(real^i_D, Ideal^j)$. Similarly, it can't distinguish between the view along two tunnels, $(Real^i_D, Real^j_{\overline{D}})$ and $(real^i_D, Ideal^j)$. Thus, it can't distinguish between the view from a single tunnel and the view from two tunnels, when U_i and U_j are not connected by contiguous adversarial nodes. ■

Theorem 2: Assume the existence of pseudo-random generators and assume the hardness of the DDH problem. Assume that h_1 uniformly maps elements from $\{0, \ldots, q-1\}$ to seeds for a pseudo-random generator G[2]. Let an adversary control at most t nodes in a network. Given the view of its nodes during the Preliminary, Confirmation and Insert Document messages for a document D, the adversary does not obtain any information on D beyond the information it had prior to the protocol.

[2]By uniform mapping we mean that for any x and y in the range of h_2 we have that $\left|h_1^{-1}(x)\right| = \left|h_1^{-1}(y)\right|$