Privacy Protection for P2P Publish-Subscribe Networks

Marek Klonowski, Miroslaw Kutylowski, Bartlomiej Rozanski

2005
Privacy Protection for P2P Publish-Subscribe Networks

Marek Klonowski, Mirosław Kutylowski, and Bartek Różański
Wrocław University of Technology

Abstract. P2P systems are mainly used as distributed file systems; until recently no P2P version of newsgroups protocol existed. A DHT-based Publish-Subscribe (Pub-Sub) system by Aekaterinidis and Triantafillou is the first such a solution: a user interested in a particular topic subscribes it so that later he receives messages on new events on this topic once they occur.

Pub-Sub systems as proposed cause severe privacy problems - a user who posts a new message on some subject has to fetch the list of all nodes who subscribed to this content. This in turn means that an adversary has a straightforward and legal tool to get information who is interested in a particular topic.

In this paper we show how to enhance the system in order to protect user privacy. This includes anonymizing the users as well as hiding information on system dynamics and delivery of messages through anonymous paths. We also show how to protect such a system from spam activity.

1 Publish-Subscribe Systems

In client-server architectures users with common interest frequently form discussion groups or forums. Messages related to a particular topic can be left there so that a group member can check this place later for a new content. The problem with above scenario is limited scalability and a fair chance of spamming by unrelated news. Indeed, everybody could post new information and there was no control over message contents.

Publish-Subscribe (Pub-Sub) systems are based on another principle. The users specify (more or less precisely) what contents are they interested in. This definition, called subscription, is published somewhere in the network. Upon arrival of an event its scope is resolved (see [1]) and the server submitting the information on the event receives a list of all subscribers who are interested in this event. Since Pub-Sub is a distributed system it does not suffer from scalability problems and allows for more and larger groups of interests. Moreover, the users no longer receive unrelated information.

Privacy Problems It has been observed that Pub-Sub systems face many security problems (see for instance [2]). One of them is privacy violation that is inherent in the system architecture: a node submitting a message $E$ triggers subscription resolution

mechanism, which ends up with an up-to-date list of users who subscribed to the contents of E. So any user which has (or just claims to have) a message on some topic can fetch the list of nodes interested in it. Now this is a straightforward method for powerful attacks against Pub-Sub architectures, because adversaries interested in blocking particular content are easily able to do so.

This paper addresses the problem of subscribers’ anonymity in Pub-Sub protocols. We show a modification of the original architecture (a new anonymity layer), which prevents privacy threats mentioned. Despite modification, the (now anonymous) delivery of messages is still fast and guaranteed. The modified protocol is immune against repetitive attacks, with adversaries trying to discover subscribers’ identities by fetching lists of subscribers over and over. The solution is based on universal re-encryption [7], anonymous communication based on URE-Onions [6] and signatures for ciphertexts that can be universally re-encrypted [9].

This paper is organized as follows: in Section 2 we present anonymity tools used in our solution. Section 3 presents our anonymity layer for Pub-Sub systems and a short discussion on its features.

2 Technical Tools

2.1 Universal Re–Encryption:

let us recall ElGamal based universal re-encryption scheme of Golle et al. [7]. It is based on a cyclic group $G$ such that discrete logarithm problem is computationally hard for $G$. Let $g$ be a generator of $G$. The protocol consists of the following subprocedures:

Key setup: a random $x < |G|$ is a private key, the corresponding public key is $y$ such that $y = g^x$.

Encryption of message $m$: at the beginning numbers $k, k’ < |G|$ are chosen uniformly at random. Then we put $\alpha_0 = m \cdot y^k, \beta_0 := g^k, \alpha_1 = y^{k’}, \beta_1 := g^{k’}$. The tuple $(\alpha_0, \beta_0, \alpha_1, \beta_1)$ is a URE-ciphertext of $m$.

Decryption: Using private key $x$ it is possible compute

$$\frac{\alpha_0}{\beta_0^x} = m \cdot y^k / g^{kx} = m$$

Re-encryption Random values $k_0’$ and $k_1’$ are chosen. Re-encrypted ciphertext obtained from $(\alpha_0, \beta_0, \alpha_1, \beta_1)$ takes the following form:

$$\left( \alpha_0 \cdot \alpha_1^{k_0’}, \beta_0 \cdot \beta_1^{k_0’}, \alpha_1^{k_1’}, \beta_1^{k_1’} \right)$$

The above scheme has a number of useful properties:

Key-privacy The same message encrypted for the second time yields a different ciphertext. Moreover, given two ciphertexts, it is impossible to say whether they were encrypted under the same key without private keys.

Homeomorphic property: Having two ciphertexts $(\alpha_0, \beta_0, \alpha_1, \beta_1)$ of a message $m$ one can change it to a ciphertext of $m \cdot m’$ by substituting $\alpha_0 := \alpha_0 \cdot m’$. So if $m = 1$, then one can “inject” a message $m’$ into an “empty ciphertext” encoding 1.
Re-encryption property: re-encryption does not require any knowledge on the public and private key corresponding to the ciphertext.

During re-encryption all four components of a ciphertext change in a provably secure way – no one can distinguish two ciphertexts without appropriate private keys.

Extended Universal Re–Encryption One can generalize the scheme described so that decryption has to be performed by multiple parties before the plaintext is obtained [6]. Simply, if \( x_1, x_2, \ldots, x_\lambda \) are private keys and \( y_1, y_2, \ldots, y_\lambda \) are the corresponding public keys, then we form the following ciphertext

\[
E_{y_1, y_2, \ldots, y_\lambda}(m) = \left( m \cdot (y_1 y_2 \ldots y_\lambda)^{k_0} \cdot g^{k_0}; (y_1 y_2 \ldots y_\lambda)^{k_1} \cdot g^{k_1} \right)
\]

So \( E_{y_1, y_2, \ldots, y_\lambda}(m) \) is a ciphertext with decryption key \( \sum_{i=1}^{\lambda} x_i \). It can be re-encrypted as a regular URE-ciphertext, but it can also be partially decrypted. If we start with a \( E_{y_1, y_2, \ldots, y_\lambda}(m) = (\alpha_0, \beta_0; \alpha_1, \beta_1) \), then

\[
(\alpha_0 / \beta_0^{x_1}, \beta_0; \alpha_1 / \beta_1^{x_1}, \beta_1)
\]

is a ciphertext with decryption key \( \sum_{i=2}^{\lambda} x_i \).

URE-Signatures One can extend URE-encryption so that a ciphertext can be signed so that the signature can be re-encrypted together with the URE-ciphertext [9].

The first remark is that all algorithms described in this section can be implemented for a subgroup \( G \) of order \( p'q' \) of RSA group \( \mathbb{Z}_N^* \), where \( N = pq, p = 2p' + 1 \) and \( q = 2q' + 1 \), for appropriate prime numbers \( p' \) and \( q' \). (For a discussion on security of the El-Gamal over composite groups can be found in papers [3], [11].)

Key setup: The public key of a signer is a random \( e \), which is co-prime with \( \phi(N) \). The private signing key is \( d \) such that \( e \cdot d = 1 \mod \phi(N) \).

For generating an encryption key pair for a server, some cooperation between the signer holding \( d \) and the server is necessary. The parameter \( \hat{g} := g^d \) is computed by the signer. The server chooses \( x \) uniformly at random and computes \( y := g^x \). Finally, the signer computes \( \hat{y} = y^d \) (so \( \hat{y} = g^{dx} = \hat{g}^x \)). Then the public encryption key for the server is the tuple \((g, \hat{g}, y, \hat{y}, N)\) and the corresponding private decryption key is \( x \).

Ciphertext and signature creation: For a message \( m \), the signer chooses values \( k_0, k_1 \) uniformly at random. Then it computes the following tuple, which is the signed ciphertext:

\[
(\alpha_0, \beta_0; \alpha_1, \beta_1; \alpha_2, \beta_2; \alpha_3, \beta_3) := \left( m \cdot y^{k_0}, \hat{g}^{k_0}; y^{k_1}, \hat{g}^{k_1}; m^d \cdot \hat{y}^{k_0}, \hat{\hat{g}}^{k_0}; \hat{y}^{k_1}, \hat{\hat{g}}^{k_1} \right).
\]
Re-encryption: values $k'_0, k'_1$ are chosen uniformly at random. Then the re-encrypted message is obtained by the following formula:

$$
\left( \alpha_0 \cdot \alpha_1^{k'_0}, \beta_0 \cdot \beta_1^{k'_0}; \alpha_1^{k'_1}, \beta_1^{k'_1}; \alpha_2 \cdot \alpha_3^{k'_0}, \beta_2 \cdot \beta_3^{k'_0}; \alpha_3^{k'_1}, \beta_3^{k'_1} \right).
$$

One can check that this is another URE-signature of $m$, namely one constructed with random numbers $k_0 + k_1 \cdot k'_0$ and $k_1 \cdot k'_1$.

Signature verification: If a RSA-URE signature is correct, then for some $k$ we have $\alpha_0 = m \cdot y^k$, $\alpha_2 = m^d \cdot y^k$, so $\alpha_2 = \alpha_0 ^d$. Hence the verifier accepts the signature if and only if $\alpha_0 = \alpha_2 ^d$.

2.2 URE-Onions

The most important family of anonymous communication protocols is based on onions routing introduced for the very first time in [12]. For each message to be sent a random “path” of intermediate servers is chosen. Then the message is encoded so that it must be processed (recoded) by all servers from this path. Moreover, the encoding scheme has to guarantee that it is impossible to link the inputs and outputs of each server.

URE Onion Protocol One can base an onion protocol on extended URE-ciphertexts [6]. Assume that we have to send a message $m$. At the beginning a random path of servers is chosen: $s_1, s_2, \ldots, s_\lambda$. Then a modified onion is built with $\lambda$ ciphertexts, called blocks. The $i$th block, for $1 \leq i \leq \lambda - 1$, has following form:

$$
E_{y_{s_1}, \ldots, y_{s_i}}(s_{i+1})
$$

The last block has the form

$$
E_{y_{s_1}, \ldots, y_{s_\lambda}}(m)
$$

Routing At the beginning such an onion is sent to node $s_1$, then it is sent to $s_2$, $s_3$, $\ldots s_\lambda$. When $s_j$ receives an onion, it partially decrypts and re-encrypts all its blocks. During decryption phase each block $(\alpha_0, \beta_0; \alpha_1, \beta_1)$ is replaced by

$$
\left( \frac{\alpha_0}{(\beta_0)^x}, \beta_0; \frac{\alpha_1}{(\beta_1)^x}, \beta_1 \right).
$$

After partial decryption, $s_j$ can read the next destination server $s_{j+1}$ from one of the blocks. During re-encryption phase each block obtained is re-encrypted in a standard way. After partial decryption, $s_j$ can read the next destination server $s_{j+1}$. For more details on the scheme see [6].

4
Extending the Path  A creator of an URE-onion can append to it some data so that the anonymous path determined by the onion can be enlarged – a new server or servers get attached at the beginning. Namely, each URE-ciphertext instead of the form

\((m \cdot (y_{s1} \cdot \ldots \cdot y_{st})^k, g^k), (y_{s1} \cdot \ldots \cdot y_{s1})^{k'}, g^{k'})\)

has now the form:

\((m \cdot (y_{s1} \cdot \ldots \cdot y_{st})^k, g^k), (y_{s1} \cdot \ldots \cdot y_{s1})^{k'}, y_{k1}, \ldots, y_{k1}, y_{k1}', \ldots y_{k1}')\)

In such a case if we wish to extend the path by putting the server with public key \(y_{ju}\) at the beginning of it, then the above URE-ciphertext must be transformed into

\((m \cdot (y_{ju} \cdot y_{s1} \cdot \ldots \cdot y_{st})^k, g^k)((y_{ju} \cdot y_{s1} \cdot \ldots \cdot y_{s1})^{k'}, g^{k'})\)

This can be easily done by multiplying the first component with the component containing \(y_{ju}\) and the third one with the component containing \(y_{j1}'\).

Let us remark that URE-onions are designed so that the changes are not normally possible. For instance, one cannot remove a server from the path.

URE-Navigators and inserting URE-signatures  URE-onions, slightly changed and called navigators in this contexts, were used in [10] to built a layered communication protocol. We shall adapt it for our purposes. A navigator is an URE-onion, where the last block is a URE-ciphertext of 1. This block will be called later navigator-cipherbox.

Such onions can be created in advance by the subscriber and sent to a server of Pub-Sub system handling a particular topic. The server can re-encrypt such a navigator many times and use each version as a container for sending a message to the subscriber.

In the simplest version of the protocol a server generating reporting an event message \(m\) gets a number of (re-encrypted) navigators. Then it inserts a message \(m\) to the navigator-cipherboxes by multiplying them by \(m\). As mentioned above, this creates ciphertexts of \(m\).

One can also prepare a navigator for signing. For this purpose a server holding a private signing key \(d\), after receiving a navigator expands each its URE-ciphertext into a URE-ciphertext with an URE-signature. Simply, each of its components is raised to power \(d\). Then, we get a ciphertext of 1 with an URE-signature. Inserting a message \(m\) to such a navigator is slightly harder (recall that after previous operations the navigator-cipherbox still encodes 1). The message \(m\) has to be presented to the server holding \(d\). Then it computes \(m^d\) and returns the result. Then one can insert \(m\) into the navigator-cipherbox by multiplying the first component by \(m\) and the fifth component by \(m^d\).

3 Protocol Description

Our modified Pub-Sub protocol implements following procedures:

**subscribing** - user subscribes to some content in an anonymous way. After this phase an anonymous paths leading to the subscriber are placed at nodes responsible for subscribed contents;
recoding of the subscription list, when the (encrypted) subscription lists are re-encrypted and permuted at random. The procedure is performed periodically by the nodes storing respective subscription list;

unsubscribing which causes removal of the user from subscription list (upon his anonymous request);

event handling – upon arrival of new message (say: some user $R$ discovered or created new information) respective subscribers $S_j$ have to be notified and the message handed to them. Therefore, $R$ contacts nodes $Q_i$ which store subscription lists and downloads them. Using achieved navigators $R$ sends his message to $S_j$ in secure, anonymous way.

The event handling phase may include some additional procedures that cope with spam. This might be a crucial feature in this case, since spamming becomes very easy in case of anonymous message processing and Pub-Sub systems.

Subscribing In order to subscribe to the content $A$ defined by some predicate $a_i$, user $u$ computes $h_i = H(a_i)$, where $H()$ is some fixed hash function. Values $h_i$ define IDs of the nodes $S_i$ which are responsible for predicates $a_i$. Then, $A$ sends requests to $S_i$ with the following information:

$$A, r_i, N_i$$

where $r$ is a random identifier of the request, and $N$ is a URE-navigator leading to $u$. The request itself has to be sent through an encrypted anonymous communication channel.

Before the appropriate record is inserted in the subscription list, node $S$ verifies if the navigator $N$ indeed leads to the node that requested subscription. For this purpose, following steps are executed:

1. Using navigator $N$ node $S$ sends a confirmation consisting of: a request for $A$, string $r$, and some additional random string $r'$.
2. upon receipt of confirmation user $u$ responds with a message containing $r'$, $r$ sent via anonymous encrypted channel.

After receiving $r'$, $r$ node $S$ uses $N$ and $r$ in order to update the subscription list as described below.

Subscription list Node $S$ stores two lists: a (full) subscription list (FSL) and a reduced subscription list (RSL). The full list is used by node $S$ internally, while an instance of the later (which is evolving) list is eventually passed to the event submitters. The full subscription list contains following data:

1. an identifier $r$,
2. a navigator $N$.

The corresponding record of RSL contains only a navigator $N'$, where:

**option 1:** from each URE-ciphertext the 3rd and 4th components, used for the purpose of re-encryption, are removed.
option 2a: \( N' \) is obtained from \( N \) by appending some additional node on the start of the path, as described in previous section,

option 2b: an entry corresponding to \( N \) has the form:

\[ R, E_{P(R)}(N''), C \]

where \( R \) is some node chosen at random, \( P(R) \) denotes the public key of \( R \), \( E_{P(R)} \) denotes encryption with \( P(R) \) (asymmetric or a hybrid one is preferred choice), \( N'' \) denotes \( N \) with the navigator-cipherbox truncated, and finally: \( C \) is the navigator-cipherbox of \( N \).

Storing and managing subscription list. Each P2P system has certain degree of dynamics. Therefore, for practical reasons it should be assumed that the subscription changes become effective with periods longer than some \( T \) seconds. During each period new subscription lists are prepared following way:

- each new entry is inserted into the new FSL (i.e. a subscription identifier \( r \) and a navigator \( N \)), the corresponding entry is appended to the RSL, according to the rules discussed above,
- for each navigator \( N \) appearing on previous FSL (has not been removed in the meantime, that is):
  - each component of \( N \) is universally re-encrypted,
  - from the re-encrypted navigator \( N \) a corresponding entry is generated for new RSL.

When a new period begins old subscription lists (FSL and RSL) are replaced by new ones.

Unsubscribing In order to unsubscribe from the event \( A \) user \( u \) sends a request to the \( S_i \)'s containing the same respective random strings \( r_i \) that were used for \( a_i \) subscription. \( r \) strings need not to be stored by \( u \) – for example they can be generated by some \( R(A, K, i) \), where \( R \) is a secure hash function, \( K \) – secret key of \( u \), and \( i \) is some index number that can be easily guessed by \( u \).

Event Processing Option 1: When an event of type \( A \) occurs at node \( X \) (say \( X \) discovers some new information or simply generates new message), it has to be processed and eventually delivered to the respective subscribers. To begin with, \( X \) presents new message \( M \) to the node \( S \) responsible for the subscription list for \( A \) (\( X \) can compute values \( h = H(a_i) \)). If the message \( M \) is valid (according to some publication rules), then \( S \) prepares a list of navigators which will allow \( X \) to broadcast \( M \) to the subscribers. To prepare such a list, \( S \) uses a copy of the RSL and inserts \( M \) into the navigator-cipherbox of each navigator from the list. Then, this list is passed to \( X \).

When \( X \) receives the list he splits it into components and sends each component (which is an URE-Onion) to the first node \( Z_0 \) indicated on the anonymity path. Then it is processed along the anonymity path as described in Section 2. While the message
is processed by consecutive nodes, each can check the URE-signature in navigator-cipherbox, to optionally stop the processing if the signature is invalid.

Option 2: In this case $S$ simply passes the RSL to $X$. Now, $X$ is responsible for splitting the list and inserting message $M$ into each navigator-cipherbox. Afterwards packet delivery is launched - each packet is sent to the first node on respective anonymity path, and the rest unfolds just as in Option 1 description.

In this case however, node $S$ need to add some number of test entries to the RSL at random positions. For $X$ these test entries are indistinguishable from the real ones. When the message $M$ sent by $X$ gets delivered to the nodes defined in RSL, communication will also reach test addresses. This allows our Pub-Sub architecture monitoring of the traffic comming from $X$, and as a result improper behaviour of $X$ can be detected and treated in some way.

3.1 Anonymity Features and Anti-spam Methods

Tracing Subscription Dynamics If the addresses on subscription lists were anonymized in a static way, event submitter $X$ could draw some conclusions on subscribers behavior judging by the information extracted from subscription lists. Namely, $X$ could trace down the patterns of the users (un)subscribing to the topics. For instance if he knew that user $u$ subscribes to a number of topics at given time, he could fetch new RSL’s rightafter, and compare them with the previous versions. As a result, $X$ could guess what topics $u$ subscribed - this is where some new addresses have appeared.

Such an analysis is not straightforward with re-coding of the entries on subscription lists.

Nevertheless, if a small number of users subscribe or unsubscribe at some moment of time, then an adversary may look for subscription lists where the number of subscribers has changed. When the network dynamics is below some threshold, such attack can result in a non-negligible leakage on user’s preferences information. We will not discuss the details of such statistical attacks in our paper, but it should be clear to our reader, that this is possible to do with more or less effort.

A standard method of protection against such attacks is to have some small number of dummy users, who subscribe and unsubscribe in a controlled manner. However, there will still be some trace of user behavior for external observer, because in case of low dynamic P2P netowrks stochastic behavior of FSL and RSL changes would be dominated by the changes caused by our dummy users. Fortunately, when re-coding of the subscriber lists takes place, adversaries can no longer associate RSL entries with regular or dummy users. Hence these two types remain undistinguishable and the adversary cannot guess number of subscribers, even when she observes RSL’s for longer period of time.

Tracing paths In case when many events matching $A$ occur at the same period of $S$, many messages to the same subscribers are processed along the same paths at the roughly same time. Then all respective onions have the same encoded addresses, which could pose a threat to the anonymity of subscribers. For protection against such attacks-full navigators (i.e. with all four components of each URE-ciphertext) can be used. In such a case, a repetition will not occur w.h.p. The only remaining attack against
subscriber identities is traffic analysis (which is not easy to perform). Even this attack becomes futile when the anonymity paths are of a logarithmic length (see [5]).

Anti-spam Methods According to the Option 2 above, a spammer using subscription lists would be detected w.h.p.. A different story is modifying the message transmitted by intermediate servers. Only the final recipient can be aware of the modification. However, it is impossible to change a message encrypted in a URE-ciphertext so that it takes a given form.

When the first protocol option is applied (the one with URE signatures), a message injected into the system is controlled by the node responsible for its event. Then each node along the anonymous paths can check the URE signature and stop forwarding in case the signature was invalid. Let us see, that this protocol option is immune against eventual changes introduced to the ciphertext by intermediate nodes, since this would render URE-signature invalid. The only problem here is with repetitions of legitimate messages that have already been processed through the node. Due to a valid URE-signature, such message cannot be discarded in this option. A countermeasure here could be to change the key(s) used for URE-signatures frequently.

Changing of the URE signature keys can be implemented in the following way: From time to time the Pub-Sub system generates a new key pair to be used and distributes it among the servers. It also publishes a certificate of the new public key stating its validity deadline. The Pub-Sub servers uses the private key obtained for URE-signatures until it gets a new, fresh key.

Discussion and Conclusions

The techniques presented fill the privacy gap in the String-DHT protocol for Publish-Subscribe applications in P2P networks. Our modifications adds an important feature, which are missing in the previous protocols – anonymity of subscribers. The modifications are also important from a legal point of view - while running a Pub-Sub system it becomes easier to conform with the private data protection acts.

Our modifications add complexity to the protocol in terms of computational complexity, communication volume and communication latency. Even the size of an encoded message grows something like 8 times. Of course, the Pub-Sub systems should be used to distribute the headlines rather than full files, so the messages are rather small and this is not a very serious issue. Communication latency is also not a serious problem, since in a Pub-Sub system such a delay is usually acceptable. Recall that the main purpose of the Pub-Sub system is to save time wasted for scanning information systems for particular events. The lengths of anonymous paths may be adjusted to current conditions and tracing possibilities by eavesdroppers. Usually, we do not expect that the users will be satisfied with a few intermediate servers. Computational complexity of re-coding the navigators is relatively high, but most of the chores are not executed in real time. Certainly, one has to put balance between a server computational power and the number of lists that it is holding.

Our construction shows how useful and handy is universal re-encryption and related algorithms for design of privacy aware protocols. It seems that many possibilities are still to be explored.
Last not least, our construction integrates privacy protection with the measures against malicious behavior of protocol participants. While the servers of the Pub-Sub system have to be reliable functionally, we do not have to trust them in the sense of malicious behavior – they do not know the subscribers of the topics! Of course the servers might deny service, but this can be detected quite early and appropriate actions can be started.

The subscribers can flood the Pub-Sub system with faulty navigators. However, this problem can be solved as proposed in [10]: the navigators may be produced offline by trusted parties. The initiators of the events cannot insert unrelated messages into the system – the inappropriate messages get either deleted due to invalid signatures or the spammer will be caught.

**Acknowledgment**

We thank Peter Triantafillou for presenting us the topics concerned in this paper.

**References**