

## Private Anonymous Messaging

Ruchith Fernando, Bharat Bhargava  
 Department of Computer Science  
 Purdue University  
 United States of America  
 {rfernand, bb}@cs.purdue.edu

Mark Linderman  
 Air Force Research Laboratory  
 Rome, NY  
 United States of America  
 mark.linderman@rl.af.mil

**Abstract**—Messaging systems where a user maintains a set of contacts and broadcasts messages to them is very common. We address the problem of a contact obtaining a message that it missed, from other contacts of the user while maintaining anonymity of all parties involved. We identify a set of requirements in addressing this problem and propose a modification to the hierarchical identity based encryption scheme proposed by Boneh et. al [2]. We briefly present an implementation of the proposed cryptographic primitives as a proof of concept.

**Keywords:** Anonymity, Privacy, Peer-to-peer messaging, hierarchical identity based encryption

### I. INTRODUCTION

Social media includes services such as weblogs, online social networks and microblogging are being used in broadcasting opinions. This leads to censorship and suppression. This is feasible due to the current architecture of these services where they primarily adopt a client-server model. Different approaches can maintain the power of social media while being able to resist control by third parties.

We consider a peer-to-peer setup where the peers are *only* connected to other peers who they personally trust. These are called *contacts* of a peer. A peer distributes a message to its contacts (which we call an *update*) and all the peers are expected to receive this update. A peer may directly communicate the message when a contact is available online. We address the problem where there is only a sub set of contacts available to a peer to directly communicate a message and the other unconnected peers needs to obtain this message from those who already possess it without compromising their privacy. We present a novel cryptographic approach as a solution to this problem.

Following section introduces the setup of the network of peers and how they are connected and the requirements that we try to satisfy with our scheme. Preliminary notions and our solution is that meets the stated requirements is described. Implementation of the proposed cryptographic primitives is briefly presented after a highlevel evaluation of the scheme.

### II. PROBLEM

A peer in this system is a user who has a set of other peers registered with it as contacts. This peer registration is bi-directional. In other words when peer A becomes a contact of peer B, peer B becomes a contact of peer A.

A peer intends to send messages to all its contacts. All of these messages are to be delivered to the peer's contacts at that point of time. This is similar to the notion of microblogging. (Example: Twitter). Such a message is identified as an *update*.

We denote the peer generating an *update* as  $P$  and its contacts as the set  $C = \{C_{P_i}\}$  where  $i \in \{1, \dots, n\}$  where  $n$  is the number of contacts of  $P$ .

We formalize the problem of a contact obtaining an update that it missed anonymously with the following requirements.

- A peer  $P$  should be able to simply send its *update*  $M_P$  only to those contacts who are available online at the point of time it sends the update using direct connections to those peers. We denote the set of online contacts as  $C^+ \subseteq C$  where  $|C^+| \geq 1$ .
- Those other contacts of  $P$  who were off-line at when  $P$  sent  $M_P$  should be able to obtain  $M_P$  when they are available online. We denote these contacts as  $C^- \subset C$ .
- Any  $C_{P_i} \in C^-$  will be able to publish a query requesting an update of  $P$ . This is called an update request and is denoted by  $Q_P$ .
- Any  $C_{P_i} \in C^+$  will be able to publish a response to a  $Q_P$ . This response is denoted by  $S_P$  and an eavesdropper with polynomially bounded resources should not be able to compute the original  $M_P$  using  $S_P$ .
- The contact who provides  $S_P$  should not be able to learn who generated  $Q_P$ .
- The contact who generates  $Q_P$  and receives the corresponding  $S_P$  should be able to extract  $M_P$  but should not be able to learn who generated  $S_P$ .
- When the composition of  $C$  changes to new set of peers  $C'$ ,  $P$  should be able to update private configuration of the members of  $C'$  with the issue of a public message.
- After such an update those peers in the set  $C - C'$  should not be able to obtain an *update* of  $P$ .

The scheme proposed in section IV addresses all these

requirements.

For example let Alice be  $P$ . Alice has four contacts: Bob, Charlie, David and Nancy, out of which only Bob gets the *update*  $M_P$ . Therefore:  $C = \{Bob, Charlie, David, Nancy\}$ ,  $C^+ = \{Bob\}$  and  $C^- = \{Charlie, David, Nancy\}$ .

Note that a peer only trusts and has knowledge of its immediate contacts and is not aware of connections among those peers and their contacts. There are practical implementations of the notion of friend-only networks such as Freenet/Darknet [4] and GNUUnet [5]. We only consider cryptography related aspects of the solution to the identified problem.

### III. PRELIMINARY NOTIONS

This section introduces the necessary background.

#### A. Hierarchical Identity Based Encryption (HIBE) [2]

Identity based encryption first proposed by Shamir [8] is a public key encryption scheme where the identity of an entity can be used as the public key. The first complete solution for this was presented by Boneh and Franklin [3]. Any party who intends to send a message to another will simply use a set of public parameters of a trusted authority along with the identity of the recipient will encrypt using this scheme. The recipient of the cipher text will be able to obtain the corresponding private key from the third party (who executes private key generation algorithm for the given identity after authenticating the requester) and decrypt the cipher text to obtain the plain text.

This idea of identity based encryption was extended to a hierarchy of identities in [6] and [2], where at each level the private key is used as the input to the key generation algorithm along with the global parameters defined by the root. The HIBE system is defined in [2] as follows (which we modify in deriving out scheme):

Let  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$  be a bilinear map where  $\mathbb{G}$  is a group of prime order  $p$ . An identity is defined as  $ID = (I_1, \dots, I_k) \in (\mathbb{Z}_p^*)^k$  where  $k$  is the depth of the hierarchy that the  $ID$  belongs to.

There are four algorithms: *Setup*, *KeyGen*, *Encrypt* and *Decrypt*.  $l$  is the maximum depth of the hierarchy allowed.

- *Setup*( $l$ ), generates the public parameters and the master key as follows:
  - Select a generator  $g \in \mathbb{G}$  and a random  $\alpha \in \mathbb{Z}_p$
  - Set  $g_1 = g^\alpha$
  - Pick random  $g_2, g_3, h_1, \dots, h_l \in \mathbb{G}$
  - $params = (g, g_1, g_2, g_3, h_1, \dots, h_l)$
  - $master-key = g_2^\alpha$

- *KeyGen*( $d_{ID_{k-1}}, ID$ ), generates the private key of the given  $k^{th}$  level  $ID$  using a  $k - 1$  level private key ( $k \leq l$ ).

First suppose the  $k - 1$  level private key was generated using the master key :

- Select a random  $r \in \mathbb{Z}_p$
- Output  $d_{ID_{k-1}} = (a_0, a_1, b_k, \dots, b_l) = (g_2^\alpha \cdot (h_1^{I_1} \dots h_{k-1}^{I_{k-1}} \cdot g_3)^r, g^r, h_k^r, \dots, h_l^r)$

Now the  $k^{th}$  level private key:

- Select a random  $t \in \mathbb{Z}_p$
- Output  $d_{ID_k} = (a_0 \cdot b_k^{I_k} \cdot (h_1^{I_1} \dots h_k^{I_k} \cdot g_3)^t, a_1 \cdot g^t, h_{k+1}^t, \dots, h_l^t)$

- *Encrypt*( $params, ID, M$ ), encrypts a message  $M \in \mathbb{G}$  using the public key  $ID = (I_1, \dots, I_k)$  :

- Select a random  $s \in \mathbb{Z}_p$
- Output  $CT = (A, B, C) = (e(g_1, g_2)^s \cdot M, g^s, (h_1^{I_1} \dots h_k^{I_k} \cdot g_3)^s)$

- *Decrypt*( $d_{ID}, CT$ ), decrypts a given cipher text of the above form  $(A, B, C)$  using the given private key of the form  $(a_0, a_1, b_k, \dots, b_l)$ .

$$(A \cdot e(a_1, C)) / (e(B, a_0)) = M$$

### IV. PROPOSED SOLUTION

Here we present the scheme that addresses the requirements identified in section II. We discuss the details of setting up the parameters of a peer, registering contacts, contacts generating update requests to be processed by other contacts, update response to such a request and re-key of the system at a peer and how contacts update themselves.

#### A. Peer Setup

A peer  $P$  will have a two level HIBE system parameters. This is by calling *setup*(2). This will generate generates the public parameters and the master key of the peer as follows:

- Select a generator  $g \in \mathbb{G}$  and a random  $\alpha \in \mathbb{Z}_p$
- Set  $g_1 = g^\alpha$
- Pick random  $g_2, g_3, h_1, h_2 \in \mathbb{G}$
- $params = (g, g_1, g_2, g_3, h_1, h_2)$
- $master-key = g_2^\alpha$

#### B. Registering a Contact

The main idea is to setup a two level ( $l = 2$ ) HIBE system at each peer. When a peer  $P$  registers a  $C_{P_i}$  it will create a new random first level identifier  $I_{r_i} \in \mathbb{Z}_p$  and corresponding private key ( $d_{I_{r_i}}$ ). The private key and the identifier will be

communicated to  $C_{P_i}$  using a private channel.  $d_{I_{r_i}}$  is of the form  $(g_2^\alpha \cdot (h_1^{I_{r_i}} \cdot g_3)^r, g^r, h_2^r)$ , where  $r \in \mathbb{G}$  is random.

- $C_{P_i}$  keeps both  $I_{r_1}$  and  $d_{I_{r_1}}$  private along with the public parameters of  $P$
- $P$  stores the tuple  $\langle I_{r_i}, r \rangle$ ,<sup>1</sup>

### C. A Contact Requesting an Update

When  $P$  sends an *update* message it may send the update directly to available contacts by encrypting the message using their corresponding identifiers. The interesting case is when a contact  $C_{P_{req}}$  needs to obtain the latest *update* of  $P$  and  $P$  is no longer available online. In such a situation, as highlighted by in the requirements,  $C_{P_{req}}$  will be able to generate a request for  $P$ 's update,  $Q_p$ . This is generated as follows:

Suppose the identifier assigned to  $C_{P_{req}}$  by  $P$  is  $I_{r_1}$

- Select a random  $I_{r_2} \in \mathbb{Z}_p$
- Set  $ID_{req} = h_1^{I_{r_1}} \cdot h_2^{I_{r_2}}$
- Update Request to be published:  
 $Q_P = \langle PID, ID_{req} \rangle$ , here  $PID$  is an identifier string of  $P$  known to all  $P$ 's contacts.

$C_{P_{req}}$  publishes  $\langle PID, ID_{req} \rangle$  and any of  $P$ 's other contacts will be able to respond to this request. This request information can be made publicly available using a common medium. The steps in creating the response is described next.

### D. Encryption and Update Response

When a contact of  $P$  observes the tuple  $\langle PID, ID_{req} \rangle$  and decides to serve this request it will first encrypt the latest *update* message  $M_P$  from  $P$  using the following modified encryption function (*Encrypt'*) and  $P$ 's public parameters  $params_P$ .

$Encrypt'(params_P, ID_{req}, M_P)$  :

- Select a random  $s \in \mathbb{Z}_p$
- $CT_{resp} = (e(g_1, g_2)^s \cdot M, g^s, (ID_{req} \cdot g_3)^s) = (A, B, C)$

The contact publishes the tuple  $\langle PID, ID_{req}, CT_{resp} \rangle$  as the response  $S_P$ .

### E. Decryption of the Update

The contact that generated the update request will obtain the response available and do the following to obtain the plain update message  $M_P$ . Now it can generate the corresponding private key using the first level private key it possesses, using  $I_{r_2}$  (used to generate  $ID_{req}$ ) as the second level identifier. Suppose the first level private key is  $d_{I_{r_1}} = (a_0, a_1, b_2)$ , then:

- Private key for  $ID_{req}$  :  $d_{ID_{req}}$   

$$= (a_0 \cdot b_2^{I_{r_2}} \cdot (h_1^{I_{r_1}} \cdot h_2^{I_{r_2}} \cdot g_3)^t, a_1 \cdot g^t)$$

$$= (a_0', a_1')$$

<sup>1</sup>This is used to update contact parameters in the case of a re-key.

- Finally to decrypt  $CT_{resp} = (A, B, C)$  :

$$(A \cdot e(a_1', C)) / (e(B, a_0')) = M_P$$

### F. Peer Re-key

The set of contacts at a peer  $C$  can change in two ways:

- When a new contact joins
- when an existing contact is removed

When a new contact ( $C_{P'}$ ) joins the peer  $P$  can carryout new contact registration without and this doesn't require any changes to the parameters. The new contact will be able to request updates of the peer from its other contacts in the set  $(C - C_{P'})$ .

However when  $P$  needs to remove a contact  $C_{P'}$  from the list of contacts, it has to update its parameters. We present an approach where we generate public information that the set  $C - C_{P'}$  will be able to use to configure themselves.

In peer setup, the generated HIBE configuration if of the form  $params = (g, g_1, g_2, g_3, h_1, h_2)$  and  $master-key = g_2^\alpha$  where  $g_1 = g^\alpha$  and  $\alpha \in \mathbb{Z}_p$  is random. In the case of re-key a peer :

- Generates a new random  $\alpha' \in \mathbb{Z}_p$
- Sets  $master-key = g_2^{\alpha'}$
- Set  $g_1 = g^{\alpha'}$

With this change  $P$  will have to update the private keys of the contacts. Note that in contact registration process  $P$  stored the tuple  $\langle I_{r_i}, r \rangle$  for each contact  $C_{P_i}$ .

To update contacts:

First generate a random  $u \in \mathbb{Z}_p$

Initialize a list  $\langle id'_i, A_i \rangle$  and for each contact  $C_{P_i} \in C$ :

- generate the first component of the private keys of the contacts as  $g_2^{\alpha'} \cdot (h_1^{I_{r_i}} \cdot g_3)^{r_i} = A$ . This  $r$  value is from the stored  $\langle I_{r_i}, r \rangle$ .
- Add  $\langle u^{I_{r_i}}, A \rangle$  to the  $\langle id'_i, A_i \rangle$  list.

Finally the complete re-key information to be published is

$$\langle PID, g_1, u, [\langle id'_1, A_1 \rangle, \dots, \langle id'_n, A_n \rangle] \rangle,$$

where  $n = |C|$ . Note that  $id'_i$  is the identifier of  $C_{P_i}$  blinded using  $u$  where  $id'_i = u^{I_{r_i}}$ .

When a peer  $C_{P_i} \in C$  obtains this information it will do the following :

- Update  $P$ 's public parameters by replacing the  $g_1$  value with received value.
- Retrieve its identifier issued by  $P$  ( $I_{r_i}$ ) and compute  $id' = u^{I_{r_i}}$
- Obtain the updated first component of its private key from the list  $[\langle id'_1, A_1 \rangle, \dots, \langle id'_n, A_n \rangle]$  using  $id'$ .

Evaluation section, discusses how this scheme meets the identified requirements.

## V. EVALUATION

We present a high level theoretical evaluation here. Formal proof of security is to be included in a extended version of this paper.

### A. Update Request

A contact of peer  $P$  generates a random identifier for any other party to use in encryption of an *update* message (which is included in the update request  $Q_P$ ). As described in section IV.C this takes the form :

$$ID_{req} = h_1^{I_{r_1}} \cdot h_2^{I_{r_2}}$$

Here  $h_1$  and  $h_2$  are public values but  $I_{r_1}$  and  $I_{r_2}$  values are only known to the contact who generates the request. Therefore it is clear that for an eavesdropper with computationally bounded resources, it is infeasible to evaluate  $ID_{req}$  and obtain the two values  $I_{r_1}$  and  $I_{r_2}$ .

### B. Update response

When a contact of  $P$  responds to a  $Q_P$  with a response  $S_P$  which of the form  $\langle P, ID_{req}, CT_{resp} \rangle$ . Here  $CT_{resp}$  is original HIBE encryption of  $M_P$  using the identity  $I_{r_1}, I_{r_2}$ . This is secure with the security assurances provided by the original HIBE scheme [2]. Hence any other party (with polynomially bounded resources) other than the contact who generated  $ID_{req}$  will not be able to learn any information about  $P$ 's update  $M_P$ . Furthermore process does not leak any information as to who generated  $S_P$  to the contact who generated  $Q_P$ .

### C. Re-key

When a peer  $P$  is re-keyed the information published is :

$$\langle P, g_1, u, [\langle id'_1, A_1 \rangle, \dots, \langle id'_n, A_n \rangle] \rangle$$

Here  $g_1 = g^{\alpha'}$  is a public parameter of  $P$  in the original HIBE scheme and  $\alpha'$  value is safe due to the discrete logarithm problem.

We further utilize the hardness of the discrete logarithm problem to blind the identity values in the map of  $A_i$  values. Here  $u$  value is raised to the power of the first level identity of the contact ( $I_{r_1}$ ). Since the identity values are only known to those corresponding contacts, to an eavesdropper (polynomially bounded)  $A_i$  values in the map are simply indexed by a set of random values.

Finally the  $A_i$  values are of the form  $g_2^{\alpha'} \cdot (h_1^{I_{r_1}} \cdot g_3)^{r_i}$ . Here the  $r_i$  value is private between the peer  $P$  and contact  $C_{P_i}$  and  $\alpha'$  is private to the peer  $P$ . Therefore using this  $A_i$  value is it impossible to obtain the  $r_i$  value (under the same assumptions as above). Therefore no one other than  $P$  will be able to compute the other two components of the private key issued to  $C_{P_i}$ . Therefore the tuple  $\langle id'_i, A_i \rangle$  does not compromise the private key information or the identity of the contact.

After removing a contact and re-keying the parameters of a peer, the removed contact will still be able to issue a request for an update. Even if a current contact of the peer responds to such a request, the removed contact will not be able to decrypt and obtain the message due to the use of the new HIBE parameters.

### D. Peer Unlinkability

In the current scheme, an update request is defined as  $Q_P = \langle PID, ID_{req} \rangle$ , where  $PID$  is an identifier string of  $P$  known to all  $P$ 's contacts.

An observer of the public channel used to communicate may clearly gain information due to the fact that the value  $PID$  is available in plain. We can avoid this by blinding the  $PID$  value in the update request as follows:

- Define  $PID \in \mathbb{Z}_p$
- Select random  $r \in \mathbb{Z}_p$
- $Q_P = \langle PID^r, r, ID_{req} \rangle$

However, this optimization forces the users of this scheme to compute  $PID^r$  value for each of their contacts to identify an update request that they can serve. This may be an expensive operation in a situation where the user has a large set of contacts. This optimization is applicable in the other cases of the scheme where  $PID$  is used.

## VI. IMPLEMENTATION

The proposed scheme was implemented in Java as a library using Java Pairing Based Cryptography [7] library. The proof of concept application uses this library in to demonstrate the features of this library. This work is available under LGPL at anon-encrypt project hosted in Google Code [1]. All the functionality explained here carries unit tests (including application level functionality) and are integrated into the automated build.

### A. Library

The output generated by various components in the library are encoded as XML. Basically certain classes can be serialized to produce the output to be used in as communication payload which is XML. Next we describe main components of the library and their implementation. All classes of the library are contained in *org.ruchith.ae.base* package.

1) *Parameter Generator*: This is implemented in the *AEPParameterGenerator* class and will generate a new set of 2 level HIBE parameters to be used in setting up a peer. An example set of parameters are shown below (in the format used to publish the parameters.)

```
<AEPParameters>
  <Curve>
    type a1
    p 54588247263338484212874980033511711
    n 189542525219925292405802014005249
```

```
n0 4261412863
n1 3221225473
n2 3221225473
n3 4286578687
l 288
```

```
</Curve>
<G>PVkIT8a424kBsKb7MdC2AQ==</G>
<G1>Mqt8WP/2cwGDY18QIXUlw==</G1>
<G2>Bk6UHTx1w99c935Ns4gQ7==</G2>
<G3>mD21ox6XGYE+CkYrRsKmR==</G3>
<H1>gH3aQ/ fJ /h1qRLD09t3RD==</H1>
<H2>YqUaL26J8UPPc0WQKGzUj==</H2>
</AEPParameters>
```

Contents of the XML elements G, G1, G2, G3, H1, H2 are base 64 encoded, binary representation of elements of group  $\mathbb{G}$ .<sup>2</sup>

2) *Peer Key Generator*: Peer key generator is used to create private keys for contacts by a peer and is implemented in *RootKeyGen*. This generates an instance of *AEPrivateKey* which can be serialized to obtain the certificate that contains the public parameters of the peer, if identifier and the private key created for the contact. A serialized instance of a private key is shown below:

```
<AEPrivateKey>
<C1>Hz7EAfXqjO6r2AZhr0gMbZg==</C1>
<C2>Rs1cmXF1RjIVbmD4aTwZOVG==</C2>
<C3>
<Elem>DJ5vwJdWCwBG3glcb==</Elem>
<Elem>Qf4H7ieFiDFBGvd9F==</Elem>
</C3>
</AEPrivateKey>
```

3) *Contact key generator*: A contact of a peer instantiates *ContactKeyGen* class with the private parameters provided by the peer and calls *getTmpPrivKey()* with a random identifier to obtain the temporary key to decrypt information encrypted using the public identifier that it creates using the given random identifier.

4) *Cipher Implementation*: The modified HIBE encryption and decryption functions are implemented in *Encrypt* and *Decrypt*. The cipher is implemented as a block cipher which encrypts an array of elements ( $\in \mathbb{G}_1$ ) and outputs an instance of *AECipherText* which includes the corresponding instances of *AECipherTextBlock*. The cipher text will be serialized using the *serialize()* methods which produces an XML output. The decryptor will instantiate an *AECipherTextBlock* instance using the serialized cipher text and will be able to decrypt each element and return the plain elements using the given private key. In practice the

<sup>2</sup>An element of group  $\mathbb{G}$  is an elliptic curve element which is of the form (for example)  $\{x=28706359947801324, y=31910506035212, \text{inFlag}=0\}$

cipher text here will be the encryption of the key used to encrypt the payload with a symmetric encryption algorithm such as AES.

5) *Text Encoder*: It is important to note that the cipher implementation works on elements of the group  $\mathbb{G}_1$ . Therefore we need functions that encodes and decodes plain text<sup>3</sup> to and from this group elements.

$$\begin{aligned} \text{encode} &: \{0, 1\}^* \rightarrow \mathbb{G}_1 \\ \text{decode} &: \mathbb{G}_1 \rightarrow \{0, 1\}^* \end{aligned}$$

*TextEncoder* class supports both these functionality and is initialized with the public parameters. The *encode()* method returns an array of elements  $\in \mathbb{G}_1$  given a string value and the *decode()* method return a byte array which is used to construct a string value, given an array of elements.

6) *Re-key*: The functionality required to re-key a peer is encapsulated in the *ReKey* class. This is initialized with the current parameters and *update()* function creates the new master key and  $g_1$ . Then an instance of *ReKeyInformation* is created with a map of given  $\langle I_{r_i}, r \rangle$  values of the contacts. This is the information to publish, and an example (with two contacts) is shown below:

```
<ReKeyInformation>
<G1>GHbxsYEOIliVvk5xVjwxGahKA==</G1>
<Random>NYwJJmRg8JaecVbw==</Random>
<Contacts>
<Contact>
<Id>AJzC1DkYEDYdGPBg==</Id>
<A>hPZ5Gx7/NQn6Ug==</A>
</Contact>
<Contact>
<Id>tF5dFCpi1cOROUC==</Id>
<A>Dz8TQOlycULApw==</A>
</Contact>
</Contacts>
</ReKeyInformation>
```

## B. Proof of Concept Application

The proof of concept application was developed using the above library. This application uses a database with one simple table to hold all information of a contact such as contact's parameters, private key assigned to the peer by the contact, common name for the contact. Apache Derby was used as the DBMS and the database instance is maintained in a configuration directory (called *.ae*) in the user home directory.

<sup>3</sup>The value encrypted using our scheme here will be a key of a symmetric encryption scheme such as the Advanced Encryption Standard (AES). This symmetric key will be used to encrypt the payload.

1) *Update Request*: The *UpdateRequest* class is used by a contact to generate a request to obtain the latest update of a peer. Given the name of a peer and a random identifier this generates the  $ID_{req} = h_1^{I_{r1}} \cdot h_2^{I_{r2}}$  value and outputs the request as shown below:

```
<UpdateRequest>
  <User>bob </User>
  <ID>fYBD1NAR4F5XPQILewVA==</ID>
</UpdateRequest>
```

2) *Update Response*: When a contact publishes a request for an *update* another contact of the peer will be able to respond to this request and generate a response with the message from the peer. The contact that responds, can simply encrypt the message using the parameters of the peer after encoding the message to element of the group  $\mathbb{G}_1$ . The *UpdateResponse* is used to generate the response message which if of the form:

```
<UpdateResponse>
  <User>bob </User>
  <EncryptedData>
    <CipherText>
      <CipherTextBlock>
        <A>A36dIcExw6eyf7pX5UQ==</A>
        <B>fGKC3aKxQYzSAxN/vpQ==</B>
        <C>jwQvbUwrpI6jMJ4xMAA==</C>
      </CipherTextBlock>
    </CipherText>
  </EncryptedData>
</UpdateResponse>
```

## VII. FUTURE WORK

We have identified the following research questions that contributes to further development of the main concept.

### A. How to minimize the size of re-key information?

Currently we generate a minimum amount of information that is required to re-key a peer and its contacts. But in this scheme the re-key information is of order  $n$  where  $n$  is the number of contacts of the peer. It would be interesting to evaluate the possibility of reducing the size of this public information while maintaining the same properties.

### B. How can we ensure the propagation of messages?

This scheme relies on the fact that the contacts belonging to the set  $C^+$  (those who holds the latest  $M_P$ ) will respond to an *update* request  $Q_P$  with correct a response  $S_P$ . It will be useful to evaluate the possibility of coming up with an incentive scheme for contacts in  $C^+$  to respond to  $Q_P$ s. We need a mechanism where the responses can be evaluated for their correctness with the given context (time of the request etc.).

## C. Formal Proof of Security

Even though we evaluated high level security aspects of this work in section V, we have not provided a formal proof of security. We plan to prove that, given a request message, an adversary with polynomially bounded resources will not be able to:

- Distinguish the contact who generated the request when compared with another request, and
- Distinguish the valid response to the request given two responses (one valid and one not).
- Infer the valid response message to the given response.

Furthermore we will prove that a polynomially bounded contact who was removed before a re-key operation will not be able to derive the new private key based on the public re-key information.

## VIII. CONCLUSION

We proposed the cryptographic primitives to address the problem of distributing a message from a common peer using a pull mechanism where the peers requesting the message can request messages anonymously. Details of the scheme was provided followed by a high level evaluation of security. As a proof of concept of the proposed cryptographic functionality, we developed an implementation and finally identified the possible future improvements and research related to this work.

## REFERENCES

- [1] Project anon-encrypt at Google code. <http://code.google.com/p/anon-encrypt/>.
- [2] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Proceedings of Eurocrypt 2005*, LNCS. Springer, 2005.
- [3] Dan Boneh and Matthew Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32:586–615, March 2003.
- [4] Ian Clarke, Oskar Sandberg, Matthew Toseland, and Vilhelm Verendel. Private communication through a network of trusted connections: The dark freenet. <http://freenetproject.org/papers/freenet-0.7.5-paper.pdf>.
- [5] GNUnet. <https://gnunet.org/>.
- [6] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, EUROCRYPT '02, pages 466–481, London, UK, UK, 2002. Springer-Verlag.
- [7] The Java Pairing Based Cryptography Library (jPBC). <http://gas.dia.unisa.it/projects/jpbc/>.
- [8] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.