Master thesis

Distributed and Secure Social Network Mobile Application

Master student at KTH
Tien Dat Le,  
NordSecMob programme
letiendat3012@gmail.com

Supervisor: Ali Yavari
Examiner: Konrad Tollmar
Abstract

As one of the twenty-first century greatest invention, online social network (OSN) is now the central information hubs of million lives, storing and controlling data flow of millions terabytes, changing the way information is generating, searching and sharing among people. However, due to the more and more dependence of people to some OSN providers, the data of users are violated and manipulated by providers publicly and aggressively.

Many researches recently pinpoint that decentralizing and mobile is inevitable trend of OSN. In this thesis, we explore the neat requirements of DOSN in mobile context and propose a solution satisfying them. We first examine potential weakness and problems in some novel DOSN solutions. Thereafter, the deployment of identity-based cryptography and free cloud storage service is introduced to remedy the existing challenges in existing solutions. In addition, we developed proof-of-concept implementations for the system with a clear API and expandable abilities.

The experimental evaluation on the system shows that the approach of using identity-based cryptography for privacy enhancement and identity and trust management is feasible in mobile context. Moreover, the free storage clouds assists content serving well and hence reduces energy and computing resource spent on peer-to-peer mobile networks without adding extra cost.
## Contents

1. Introduction .......................................................................................................................... 7  
   1.1 Problem statement ........................................................................................................ 7  
   1.2 Objectives and research goals ..................................................................................... 8  
   1.3 Contribution .................................................................................................................. 8  
   1.4 Structure of the thesis ................................................................................................. 9  

2. Background ........................................................................................................................... 10  
   2.1 Online social network .................................................................................................. 10  
      2.1.1 Definition ............................................................................................................. 10  
      2.1.2 The security and privacy requirement ................................................................. 11  
      2.1.3 The flexibility in content sharing requirement .................................................... 11  
   2.2 Distributed Hash Table ................................................................................................. 11  
      2.2.1 Definition ............................................................................................................. 12  
      2.2.2 Properties of DHTs ............................................................................................ 12  
      2.2.3 Pastry .................................................................................................................. 12  
   2.3 Identity-based cryptography ......................................................................................... 17  
      2.3.1 History of identity-based cryptography .............................................................. 17  
      2.3.2 Basic concepts of Identity-based encryption and signature .............................. 18  
      2.3.3 Security of identity-based cryptography ............................................................ 19  
      2.3.4 Security of identity-based cryptographic schemes in this thesis ....................... 20  

3. Review of current DOSN solutions ..................................................................................... 22  
   3.1 Pure p2p architecture ................................................................................................. 22  
      3.1.1 PeerSoN ............................................................................................................... 22  
      3.1.2 LifeSocial.KOM ................................................................................................. 22  
      3.1.3 Analysis .............................................................................................................. 24  
   3.2 Distributed server architecture .................................................................................... 25  
      3.2.1 Diaspora .............................................................................................................. 25  
      3.2.2 Analysis .............................................................................................................. 25  
   3.3 Pure clouds architecture ............................................................................................... 26  
      3.3.1 Vis-à-Vis project ................................................................................................. 26
7.3 Limitations.................................43
8. Conclusion........................................45
1. Introduction
Twenty-first century has seen a huge boost in the field of Internet applications, changing the way information is generating, searching and sharing among people. One of such significant applications is online social network (OSNs) that is very popular among Internet users these days, revolutionizing our ways of interacting. They help us stay in touch with our friends, widen our circle of relationship, follow social events of famous celebrities and entertain in our free time.

Free OSNs such as Facebook, Twitter and Foursquare are now the central information hubs of million lives. With the exponential growth of OSN’s users, it is no doubt that OSN now become the most popular platform for people to communicate and share their data. Facebook alone now is reported to have 802 million active users and 609 million mobile users each day [1]. The volume of data processed by OSNs is astonishing: Facebook receives more than 2.5 billion shared contents [2] while Twitter receives more than 55 million tweets each day [3], and the number is increasing daily.

1.1 Problem statement
With such enormous amount of data owned by OSNs, mostly private and sensitive, it is crystal clear that OSN providers will find a way to make money out of the data. Many reports recently have revealed the tragic situations. Due to a report in [4], Facebook and other providers are blamed to monetize by selling the private data of users to marketers. The trend is aggravated as have shown in [5] by the constantly changing in privacy policies where the right of users to control their private information diminished during the time.

Recently, there is a cautious research experiment has been conducted by Facebook raised even more concern to its users. In the experiment, the users’ news feed content was altered to show only happy or sad contents in order to tinker users’ emotion [6]. The result of the experiment has implied an anxious fact that OSN providers, by violating users privacy, have even the capable of manipulating them.

Hence, an urgent question is how to protect OSN’s users’ privacy while keeping its appealing features. There are numerous efforts thus have recently been paid to decentralize online social network (DOSN) such as Safebook, Foaf, LifeSocial.KOM [7], PeerSoN [8], Likir [9], DECENT [10], Easier and Twister [11]. However, the challenges still remain and none of existing solutions are widely adopted.

Moreover, the switch from desktop users to mobile users within OSNs [12] [13] put even more challenges to the exist architecture of DOSN and raise a demand of proposing a new architecture that could solve the challenge in energy and resource constrains in mobile context.
To this end, the target of this master thesis is to develop a new system that allows decentralizing OSN in the resource constrains mobile context.

1.2 Objectives and research goals
The main goals of the thesis are to: 1) study the current OSN literature and analyze the technical and non-technical requirements of DOSN, 2) study current DOSN solutions and analyze their pros and cons plus their satisfactory of the requirements, 3) propose and implement an energy-efficient, privacy-enhanced and user friendly solution for DOSN on Android, 4) evaluate the performance of our proposed solution.

1.3 Contribution
DOSN is relatively new and challenge area that requires a broad knowledge to understand. The latest solutions of DOSN are rather complex, ranging from peer-to-peer technology, distributing server to cloud computing. In this thesis, the objective is to study the DOSN literature from a broad perspective, including all of present architectures and propose a new architecture that solves the existing challenges.

The thesis work successfully conceptualizes and implements a new DOSN model where a security provider and storage provider is introduced. By separating the provider of storage and of security that was associated in previous work, we eventually solve the challenge of keeping privacy while keeping the operation cost “free” for users.

Through this thesis work, we prove that a secure DOSN is now feasible on mobile context thanks to the peer-to-peer technology and identity-based cryptography. My experiment shows that the new architecture satisfies the neat requirement of both energy consumption and processing time in the mobile context.
In this thesis, we also discuss about the role of each party, i.e. users, storage provider and security provider, in operating of the system. Furthermore, we provide a programming API for developers to later extend our work to support new storage providers.

1.4 Structure of the thesis
The rest of the thesis is organized as follows. Chapter 2 starts by discussing the background needed to understand what is presented in this thesis, including online social network, distributed hash table and identity-based cryptography. Chapter 3 walks through state-of-the-art solution in DOSN along with the appropriate analysis. Chapter 4 presents our contribution to a new DSNA architecture and explains its core components, that is, structural peer-to-peer network, free cloud storage interface and identity-based cryptography system. Chapter 5 describes the detailed implementations and programming APIs. Chapter 6 discusses the weak points and strong points of our architecture compare to current solutions and Chapter 7 concludes the thesis.
2. Background

This chapter begins by describing OSN and its core elements, followed by the analysis of its security and privacy requirements. Thereafter, we discuss Distributed Hash Table and Identity-based cryptography, which are the core components used in the DOSN architecture proposed in this thesis.

2.1 Online social network

2.1.1 Definition

The online social network application has been defined in [14] as: “a system where (a) users are first class entities with a semi-public profile, (b) users can create explicit links to other users or content items, and (c) users can navigate the social network by browsing the links and profiles of other users as in previous studies [15].”

Online social networks serve a number of purposes, but three primary roles stand out as common across all sites. First, online social networks are used to maintain and strengthen existing social ties, or make new social connections. The sites allow users to “articulate and make visible their social networks”, thereby “communicating with people who are already a part of their extended social network” [15]. Second, online social networks are used by each member to upload his own content. Note that the content shared often varies from site to site, and sometimes is only the user’s profile itself. Third, online social networks are used to find new, interesting content by filtering, recommending, and organizing the content uploaded by users.

To serve these purposes, there are numerous core elements in online social network, namely profile, social connections, resource contents, communications and messages. The hierarchy of those elements is defined in [16] as follow:

- **Profile**: The profile presents the identity of an individual user in an OSN. During the time of sign-up, the profile is generated and contains personal information, a list of social connections, resources, etc. that is everything the particular user wants to share through OSN.
  - **Social connections**: A list of people such as friends or family to indicate existing social relationships of users. Almost all OSNs offer special functionalities so that a user can connect with new people based on different but related categories such as common interest, same school, etc.
  - **Resources**: Different kind of resources such as pictures, videos, events, documents, groups, wall (space to broadcast messages to all or selected social connections, public messages), which are stored under specific user profile.
o Communication: Since communication is the main motivation behind the development of online social networks, they offer different ways to communicate for the users.

- Public message: Interaction through a “wall” (like a noticeboard), which may be public or semipublic depending on users’ settings.
- Private message: Same as existing electronic mail, only visible to the sender and the receiver.
- Instant message: Synchronous text-based messaging among users.

2.1.2 The security and privacy requirement

In this section we analyze the security and privacy requirement of OSN, which is the main purpose of decentralizing OSN.

Since the users want to access their hot new share events and comments, the failure to access to content resources is unacceptable and will definitely downgrade the usability of the service. Also, the resources should only be accessed and read by the chosen users, which have the links and should be sensor even with the service providers to satisfy privacy requirement.

Moreover, for the sensitivity of resources and connection in OSN, it is mandatory that users should have the mechanism to recover their password and take back their account no matter what kind of secret have leaked.

In short, the requirement could be summarized in these words of “maximum availability”, “complete privacy” and “recovery ability”.

2.1.3 The flexibility in content sharing requirement

In the very beginning of OSN, the way of communication and interaction is somehow static via wall and resources. People have to walk through their friends’ wall to update new events or to see new pictures.

However, with the evolution of technology in OSN, the content sharing ways is now different. Users now just have to choose to follow celebrities, content providers or favorite channels, and the OSN will take care of pushing new attractive contents to their new feeds.

Hence, the OSN should support dynamic pushing data models like publish/subscribe rather than the simple pull data model for users. The content sharing mechanisms should include following, publishing in groups, publishing contents through hash-tags, etc.

2.2 Distributed Hash Table

With a glance to the privacy and content sharing requirement, it is mandatory to a DOSN is to have the capable of handling large scale publish/subscribe data flow while keeping the
“maximum availability” and “complete privacy”. The traditional servers architecture is infeasible due to privacy concern and operation cost, leading us to the sole path of deploying peer-to-peer technology for message delivering. Since the peer-to-peer network will have to handle a tremendous amount of messages, we select to use a class of structured peer-to-peer systems named Distributed Hash Table (DHT) that is scalable for millions of nodes and supports topic-based publish/subscribe. Consequently, the section will be devoted to DHT definition, properties and the set of DHT algorithm that is used in this thesis.

2.2.1 Definition
A distributed hash table (DHT) [17] is a class of a decentralized distributed system that provides a lookup service similar to a hash table; (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.

2.2.2 Properties of DHTs
There are three essential properties that most of DHTs possess make them scalable [18]:

- Routing is scalable. The typical number of hops required to find an item is less than or equal to $O(\log(n))$ and each node stores $O(\log(n))$ routing entries, for $n$ nodes.
- Items are dispersed evenly. Each node stores on average $\frac{d}{n}$ items, where $d$ is the number of items in the DHT, and $n$ is the number of nodes.
- The system scales with dynamism. Each join/leave of a node requires redistributing on an average $\frac{d}{n}$ items, where $d$ is the number of items in the DHT, and $n$ is the number of nodes.

DHT self-manage items and routing information when:

- Node join: Routing information is updated and reflect new nodes and items are redistributed
- Node leave: Routing information is updated and reflect departure of nodes, and items are redistributed before a node leave
- Node fail: Failures are detected and routing information is repair to reflect that. Items are automatically replicated to recover from failure

2.2.3 Pastry
There are various proposals for DHTs such as Chord, CAN, Kademlia, Pastry, Tapestry. After meticulously walking through all the DHT proposals and framework mentioned in [19], a decision have made to pick up Pastry due to its good performance and the FreePastry java library that supports both storage service and publish/subscribe multicast service. This section
is, therefore, a brief introduction to Pastry routing algorithm and its two application in this section.

2.2.3.1 Pastry node state

Each node in the Pastry peer-to-peer overlay network is assigned a 128-bit node identifier (nodeId). The nodeId is used to indicate a node’s position in a circular nodeId space, ranging from 0 to $2^{128}-1$. The nodeId is randomly assigned using a hash function to ensure that nodeIds are uniformly distributed and diverse in geography, ownership, jurisdiction, network attachment, etc.

Providing that a network consist of N nodes, Pastry can route a object with old to the node with id numerically closest to old in less than $\left\lceil \log_2 N \right\rceil$ steps under normal operation (b is a configuration parameter with typical value 4). The delivery is guaranteed unless $L/2$ with adjacent nodeIds fails simultaneously.

Each Pastry node maintains a routing table, a neighborhood set and a leaf set as in the figure below.

![Pastry node state diagram](image)

Fig 2. A Pastry node with nodeId 10233102, b=2, and l=8

A node routing table, $R$, is organized in $\left\lceil \log_2 N \right\rceil$ with $2^b - 1$ entries each. The $2^b - 1$ entries at row $n$ refer to a node whose nodeIds shares the prefix of $n$ first digits with the present node’s
nodeld, but the n+1 th digit will have $2^b - 1$ possible values rather than the n + 1 th digit of present node’s id.

The choice of b decide the trade-off between the size of the routing table (approximately $\lceil \log_2 b N \rceil \times (2^b - 1)$ entries) and the maximum number of hops required to route between any pair of nodes ($\lceil \log_2 b N \rceil$)

The neighborhood set M contains the nodelds and IP addresses of the M nodes that are closest (according to the proximity metric) to the local node. The neighbor set is not normally used in routing messages but is useful in maintaining locality properties.

### 2.2.3.2 Pastry routing algorithm

Let start by defining some notation:

\( R^i_l \): The entry in the routing table R at column \( i \), \( 0 \leq i < 2^b \) and row \( l \), \( 0 \leq l < \lfloor 128 / b \rfloor \)

\( L_i \): The i-th closest nodeld in the leaf set \( L \), \(-\lfloor L_i/2 \rfloor \leq i \leq \lceil L_i/2 \rceil \), where negative/positive indices indicate nodelds smaller/larger than the present nodeld, respectively

\( D_i \): The value of \( l \)'s digit in the key D.

\( shl(A, B) \): The length of prefix shared among A and B, in digits

Then the Pastry routing procedure is defined in [20] as in the following table

| 1. | if \( (L - \lfloor L_i/2 \rfloor \leq D \leq L \lfloor L_i/2 \rfloor) \) \{ |
| 2. | //D is within range of our leaf set |
| 3. | forward to \( L_i \), s.th. \( |D - L_i| \) is minimal; |
| 4. | \} else \{ |
| 5. | //use the routing table |
| 6. | Let \( l = shl(D, A); \) |
| 7. | if \( (R^D_i \neq \text{null}) \) \{ |
| 8. | forward to \( R^D_i \); |
| 9. | \} |
| 10. | \} else \{ |
| 11. | //rare case |
| 12. | forward to \( T \in L \cup R \cup M \), s.th. |
| 13. | \( shl(T, D) \geq l, \) |
| 14. | \( |T - D| < |A - D| \) |
| 15. | \} |
| 16. | \}

Table 1. Pseudo code for Pastry core routing algorithm. Source: [20]
2.2.3.3 Pastry performance

The performance comparison of different DHTs proposal in the table below illustrates that Pastry has a good performance and is scalable.

<table>
<thead>
<tr>
<th>P2P system</th>
<th>Node degree</th>
<th>Network diameter</th>
<th>Node congestion</th>
</tr>
</thead>
<tbody>
<tr>
<td>small-worlds</td>
<td>(O(1))</td>
<td>(O(\log^2 n))</td>
<td>(O((\log^2 n)/n))</td>
</tr>
<tr>
<td>Chord</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O((\log n)/n))</td>
</tr>
<tr>
<td>CAN</td>
<td>(O(d))</td>
<td>(O(dn^{1/d}))</td>
<td>(O(dn^{1/d-1}))</td>
</tr>
<tr>
<td>Pastry</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O((\log n)/n))</td>
</tr>
<tr>
<td>Tapestry</td>
<td>(O(\log n))</td>
<td>(O(\log n))</td>
<td>(O((\log n)/n))</td>
</tr>
<tr>
<td>D2B</td>
<td>(O(1))</td>
<td>(O(\log n))</td>
<td>(O((\log n)/n))</td>
</tr>
<tr>
<td>Viceroy</td>
<td>(O(1))</td>
<td>(O(\log n))</td>
<td>(O((\log n)/n))</td>
</tr>
<tr>
<td>Koorde cfg. 1</td>
<td>(O(1))</td>
<td>(O(log n))</td>
<td>(O((\log n)/n))</td>
</tr>
<tr>
<td>Koorde cfg. 2</td>
<td>(O(\log n))</td>
<td>(O((\log n)/\log \log n))</td>
<td>(O((\log n)/n \log \log n)))</td>
</tr>
</tbody>
</table>

Table 2: Comparison between expected performances of several p2p systems. Source: [19]

2.2.3.4 PAST

PAST [21] is a distributed file system layered on top of Pastry. A file is stored in the system will be assigned with a fileid computed from its filename or its content.

PAST is composed of nodes connected to the Internet. Each node can act as a storage node and a client access point and is assigned a 128-bit node identifier (nodelist). Each file that is inserted to PAST is assigned a 160-bit fileid, corresponding to the cryptographic hash of the filename, the owner’s public key and a random salt. Before a file is inserted, a file certificate is created with the fileid, the replicate factor k, the salt, the insert date and the hash of file’s content. The file certificate is then signed by its owner.

When a file is inserted in PAST, Pastry routes the file to the k nodes whose nodelds are numerically closest to the 128 most significant bits of the fileid. Each of these nodes then stores a copy of the file. The replication factor k is decided based on the availability and persistent requirements of the file and may vary between files.

This procedure ensures that (1) a file remains available as long as one of the k nodes that store the file is alive; (2) with high probability, the set of nodes that store the file is diverse in geographic location, ownership, network connectivity, etc.; and (3) the number of file is assigned uniformly between nodes.

2.2.3.5 Scribe

SCRIBE [22] is a decentralized publish/subscribe system that uses Pastry for its underlying route management and host lookup. Users create topics to which other users can subscribe. Once the topic has been created, the owner of the topic can publish new entries under the topic that will
be distributed in a multicast tree to all of the SCRIBE nodes that have subscribed to the topic. The system works by computing the hash of the topic name concatenated with the name of the user who owns the topic. This hash is then used as a Pastry key, and the publisher then routes packets to the node closest to the key using Pastry’s routing protocol to create the root node of the topic on that node. People then subscribe to the topic by computing the key from the topic and publisher’s name and then using Pastry to route a subscribe message to the topic towards the root node. When the root node receives the subscribe message from another node it adds the node ID to its list of children and begins acting as a forwarder of the topic.

```
forward (msg, key, nextid)
  switch msg.type is
    JOIN : if !(msg.group ∈ groups)
      groups = groups ∪ groups
      route(msg, msg.group)
    groups[msg.group].children ∪ msg.source
    nextId = null // Stop routing the original message
```

Table 3. SCRIBE implementation of forward. Source: [22]

```
deliver (msg, key)
  switch msg.type is
    CREATE : groups = groups ∪ group
    JOIN : groups[msg.group].children ∪ msg.source
    UNICAST : ∀ node in groups[msg.group].children
      send(msg.node)
    MULTICAST : if memberOf (msg.group)
      invokeMessageHandler(msg.group, msg)
    LEAVE : groups[msg.group].children = groups[msg.group].children – msg.source
    if (groups[msg.group].children = 0)
      send(msg.groups[msg.group].parent)
```

Table 4. SCRIBE implementation of deliver. Source: [22]

Decentralization is accomplished through having all nodes in the network snoop on subscribe messages going past them on their way to the topics root node. If the topic is one to which the current node subscribes, it will stop forwarding the packet toward the root node and add the node trying to subscribe as one of its children. In this way a treelike structure is formed with the root node at the top sending out to the first few subscriber nodes and then each of these nodes forwarding the messages on to their children, and so on. Because packets from random nodes on the Pastry network destined for the same node often end up traveling along the same path very soon in their journey, they end up attaching to whatever part of the tree is nearest to them in the Pastry network. Since each hop along a pastry route represents what is locally the best route according to the routing metric in use, the subscribe message seeks out the closest portion of the tree and attaches itself there.
Finally fault tolerance among members of the distribution tree is accomplished through the use of timeouts and keep-alives with actual data transmissions doubling as keep-alives to minimize traffic. If a child node does not hear from its parent for a while, it routes a new subscribe message toward the root node of the tree, reattaching itself wherever it bumps into the tree for that topic. If a parent doesn't hear from a child for a timeout period, it drops the child from its list of children. (If this action causes its child list to become empty, the parent stops acting as a forwarder altogether.) The only remaining failure point is that of the root node, and Pastry itself automatically overcomes this. Because Pastry duplicates keys among the few nodes closest to the key's actual value, the root node already has mirrors set up, lying dormant. If the root node goes offline, again detected through timeouts, the next-closest Pastry node will begin acting as the root node. When the creator of the topic tries to publish new material the old root node will be unreachable. The publisher will then fall back on the Pastry network and use it to route its publish message to the new root node. Once this has been done, the publisher caches a copy of the new root node's IP address to reduce the use of the Pastry network for future transmissions.

2.3 Identity-based cryptography
This section discusses Identity-based cryptography (IBC) which is the backbone of the security provider in our model. The revolution of IBC have flattened the public key infrastructure to Private Key Generator (PKG) servers, requiring lesser cost and no effort in certificate issues and certificate management.

2.3.1 History of identity-based cryptography
In 1984, Adi Shamir, a co-inventor of the famous RSA algorithm, introduced the concept of identity-based cryptography [23]. The innovation of this new paradigm of cryptography is its use of users’ identifier such as email addresses or phone numbers, instead of digital certificates, for encryption and signature verification. Thanks to this feature, identity-based cryptography significantly reduces the complexity of a cryptography system and the cost for maintaining
public key infrastructure such as certificate authorities. It also simplifies the scheme for encryption for unprepared users, since messages may be encrypted for users even before they have any contacts with system providers.

At that time, Shamir easily constructed an identity-based signature scheme using existing RSA function, but was unable to propose a scheme for identity-based encryption (IBE). The challenge remained unsolved until 2001 when Boneh and Franklin [24] and Cock [25] independently proposed two schemes using bilinear pairs for IBE, marking a new era of identity-based cryptography in research community.

2.3.2 Basic concepts of Identity-based encryption and signature

The core component of Identity-based cryptosystem is the trusted party called the “Private key generator” (PKG). As mentioned earlier, in the IBE scheme, the sender Alice can use the receiver’s identifier’s information such as e-mail address or phone number, which in string form, to encrypt a message. The receiver Bob, after obtaining the private key associated to his identifier from PKG, can decrypt the message using the key. In case of IBS scheme, the sender Alice will use the private key obtained from PKG to sign her message and the receiver Bob is able to verify it using Alice’s identifier’s information as public key.

To sum up, the IBE scheme and the IBS scheme is described in as follow:

The IBE scheme:

• Setup: The PKG creates its master (private) and public key pair, which we denote by skPKG and pkPKG respectively. (Note that pkPKG is given to all the interested parties and remains as a constant system parameter for a long period.)
• Private key extraction: The receiver Bob authenticates himself to the PKG and obtains a private key skIDBob associated with his identity IDBob.
• Encryption: Using Bob’s identity IDBob and the PKG’s pkPKG, the sender Alice encrypts her plaintext message M and obtains a ciphertext C.
• Decryption: Upon receiving the ciphertext C from Alice, Bob decrypts it using his private key skIDBob to recover the plaintext M.

The IBS scheme:

• Setup: The PKG creates its master (private) and public key pair, which we denote by skPKG and pkPKG respectively. (Note that pkPKG is given to all the interested parties and remains as a constant system parameter for a long period.)
• Private key extraction: The signer Alice authenticates herself to the PKG and obtains a private key sklDAlice associated with
• Signature generation: Using her private key sklDAlice, Alice creates a signature σ on her message M.
• Signature verification: Having obtained the signature and the message M from Alice, the
verifier Bob checks whether σ is a genuine signature on M using Alice's identity IDAlice and the PKG's public key pkPKG. If it is, he returns "Accept". Otherwise, he returns "Reject".

2.3.3 Security of identity-based cryptography
In modern cryptography, cryptosystems and secure schemes are heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions. Identity-based cryptosystems is such systems. The security of identity-based cryptography is based on the assumption that the chosen mathematical functions called bilinear maps are trap-door functions; meaning to compute the inverse bilinear non-degenerate maps is infeasible in polynomial time.

To understand the security of identity-based cryptography, this section is devoted to the bilinear map definition and the two computational problems, namely Bilinear Diffie-Hellman (BDH) and General Diffie-Hellman Exponent (GDHE).

2.3.3.1 Bilinear non-degenerate pairing map
The bilinear maps is defined in [24] as follow:

Let $G_1$, $G_2$ and $G_T$ be three cyclic groups of prime order $p$.

A bilinear map $e(\cdot, \cdot)$ is a map $G_1 \times G_2 \rightarrow G_T$ such that for any generators $g_1 \in G_1$, $g_2 \in G_2$ and $a, b \in \mathbb{Z}_p$.

- $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ (Bilinearity)
- $e(g_1, g_2) \neq 1$ (Non-degeneracy)

A bilinear map group system $\beta$ is a tuple $\beta = (p, G_1, G_2, G_T, e(\cdot, \cdot))$, composed of object as described above. $\beta$ may also include group generators in its description. We impose all group operations as well as the bilinear map $e(\cdot, \cdot)$ to be efficiently computable, i.e. in time poly ($|p|$).

In practice, the group $G_1, G_2$ is implemented using a group of points on certain elliptic curves, each of which has a small MOV exponent, and the group $G_T$ will be implemented using a subgroup of the multiplicative group of a finite field [26].

2.3.3.2 The Bilinear Diffie-Hellman problem
The BDH problem definition for a symmetric pairing $e: G_1 \times G_2 \rightarrow G_T$ is stated as follows:

**Definition 1: BDH problem**

Given a tuple $(g, g^a, g^b, g^c) \in G^4$ as input where $a, b$ and $c$ chosen at Random from $\mathbb{Z}_p^*$
Output $e(g, g)^{abc} \in G_T$
The BDH assumption signifies that the above problem is computationally intractable.

### 2.3.3.3 The General Diffie-Hellman Exponent problem

Let \( \beta = (p, G_1, G_2, G_T, e(\cdot, \cdot)) \), be a bilinear map group system such that \( G_1 = G_2 = G_T \).

Let \( g_1 \in G \) be a generator of \( G \), and set \( g = e(g_0, g_0) \in G_T \).

Let \( s, n \) be positive integers and \( P, Q \in F_p[X_1, ..., X_n]^s \) be two \( s \)-tuples of \( n \)-variate polynomials over \( F_p \).

Let \( f \in F_p[X_1, ..., X_n] \). It is said that \( f \) depends on \( (P, Q) \), denoted \( f \in \langle P, Q \rangle \) when there exists a linear decomposition

\[
f = \sum_{1 \leq i, j \leq n} a_{i,j} * p_i * p_j + \sum_{1 \leq i \leq n} b_i * q_i, \quad a_{i,j}, b_i \in \mathbb{Z}_p
\]

Given \( P, Q, f \) be as above, the \( (P, Q, f) \)-General Diffie-Hellman Exponent problems as followed

**Definition 2: \( (P, Q, f) \)-GDHE**

*Given the tuple*

\[
H(x_1, ..., x_n) = \left( g_0^{p(x_1, ..., x_n)}, g^{Q(x_1, ..., x_n)} \right) \in G^s \times G_T^s
\]

*compute \( g^{f(x_1, ..., x_n)} \)*

**Definition 3: \( (P, Q, f) \)-GDDHE**

*Given the tuple*

\[
H(x_1, ..., x_n) \in G^s \times G_T^s \text{ as above and } T \in G_T
\]

*decide whether \( T = g^{f(x_1, ..., x_n)} \)*

Noted that the \( (P, Q, f) \)-GDHE and \( (P, Q, f) \)-GDDHE have been proved to have generic security when \( f \notin \langle P, Q \rangle \) in [Boneh8]

### 2.3.4 Security of identity-based cryptographic schemes in this thesis

There are two identity-based cryptographic schemes that we use to secure our system, i.e. the identity-based broadcast encryption with constant size cipher texts and private keys proposed by Cécile Delerablée and the efficient identity-based signatures secure in the standard model proposed by Kenneth G. Paterson and Jacob C. N. Schuldt.
We shall prefer the proof to [24], [27] and [28] and claimed that the identity-based broadcast encryption scheme is secure under the GDHE and GDDHE assumption meanwhile the identity-based signature scheme is secure under the computational Diffie-Hellman assumption.
3. Review of current DOSN solutions

In this chapter, we would like to give a brief introduction to the state of the art solution for DOSN by reviewing their architecture and how far they could reach to meet OSN’s requirement, analyzing the strength and weakness of each solution and based on that proposing our new architecture which might remedy those problems. Although there are a lot of publications related to decentralizing OSN, the architecture of these solutions could be categorized into 3 main categories: pure p2p, distributed servers and pure cloud solutions. Hence, each section will be the presentation and analysis of a specific architecture respectively.

3.1 Pure p2p architecture

3.1.1 PeerSoN

PeerSoN is one of the most outstanding project aims at decentralizing OSN using pure p2p architecture leaded by Sonja Buchegger.

In the initial proposal, PeerSoN offers 2-tier p2p-based architecture for social networks where in one tier distributed hash table (DHT) is used for lookup services and another tier consists of peers to communicate and store users’ profiles [29]. The primary focus is to describe basic functionalities of OSN in p2p infrastructure to remove the dependency on a central authority (service provider). This infrastructure strengthens the privacy settings. However, due to the absolute decentralized architecture, many research problems require careful consideration.

Additionally, they proposed a mechanism to recover users’ password based-on thresh-hold secret sharing cryptographic scheme [30]. The prevention of unauthorized access to contents is enforced by a number of proposals such as simple digital-envelope based, broadcast encryption based [31] or policy based [16].

To tackle the challenge of searching and retrieving information from the DSN while maintaining privacy and preventing information leakage, an attempt to allow user search with knowledge threshold has been made by Benjamin Greschbach, Gunner Kreitz and Sonja Buchegger of PeerSoN [32] project by linking user’s id with some attributes such as name, city, email, etc and deciding a threshold with which the user’s id could be queried. For protecting information privacy, user’s id will be encrypted with the key compose from those attributes and a salt token before putting to DHT. Hence, with sufficient information, one could retrieve his friend id from the DHT.

3.1.2 LifeSocial.KOM

LifeSocial.KOM [7] is a p2p, plugin-based secure online social network, which shifts load for operating the infrastructure from the service providers to the users. The core plugins provides secure communication and access-controlled storage as well as monitored of quality of service. LifeSocial.KOM uses FreePastry for interconnecting the participating nodes and PAST for
reliable, replicated data storage. For access control, LifeSocial.KOM deploy cryptographic Public Keys as main user IDs in the network to uniquely identify users and to create an anchor for security, allowing the authentication of users, secure communication and user-based access control on the objects in the network.

Figure 4. Plugin-based architecture of LifeSocial.KOM. Taken from: [7]
3.1.3 Analysis

The pure p2p architecture has addressed the main security and privacy requirements in an elegant way. It already offered a mechanism to choose a group of peers to replicate each other’s information while considering these factors: “maximum availability” and “minimum number of replicas” like in [33]. For privacy concerns, both PeerSoN and LifeSocial.KOM come up with the solution of using encryption based access control. While LifeSocial.KOM using a trivial identity-based approach where users public key is considered as identity, PeerSoN have experimented different schemes ranging from digital-envelope based to broadcast encryption. PeerSoN also successfully propose a scheme for password recovery using thresh-hold secret sharing schemes.

However, the drawbacks of the pure p2p architecture lie in its performance as well as its usability. To illustrate, since both using DHT to storage and encryption based access control, the nodes who host the data will have to compute redundancy computing to verify if the request is illegal or not. It seems not to be feasible in mobile context, the new trends of OSN. Once users have to host hot-spot contents, it is likely that their battery will be drained out, and they will stop to use the apps without doubt.

Moreover, this approach also requires users to manage trust and certificate by them-self, which is impractical since users always are the weakest links in security. LifeSocial.KOM even cannot provide a mean to recover users account when the private key is leaked, since the id is the public key.

Overall, we appreciate the privacy protection of the solutions meanwhile pinpoint its drawback in security management and usability.
3.2 Distributed server architecture

3.2.1 Diaspora
Diaspora [34] is the only project so far deployed distributed server architecture to decentralize OSN. In Diaspora, there is no centralization server where data is stored but a federation of open servers (called pods) where users could choose to start with their trusted pods. Each pod will have its own admin and server. Messages between pods are exchanged via SSL to protect them from outsiders. Also, since the project have launched by the money of community, it owned by the Diaspora community uses and do not have any policies that claim on the right to users’ data.

![Diaspora architecture](image)

Figure 6. Diaspora architecture. Source: [35]

3.2.2 Analysis
The distributed server architecture is not theoretically solved the security and privacy of OSN. The communities publish a list of pods along with their IP addresses, their availability and their policies. Users have the right to hand data to their trusted pod among and indeed there is no means provided to protect users from an evil pod.

The strength of this architecture is the usability. Users do not have to configure anything, but registering their information to a pod and could start their OSN activities in any browsers.
However, the privacy is a matter of trust raises concern about its viability, specifically when a few pods dominate others, resulting in switching back the architecture to the notorious centralization.

3.3 Pure clouds architecture

3.3.1 Vis-à-Vis project
Vis-à-Vis [35] project proposes a decentralized framework for OSNs based on the privacy-preserving notation of a Virtual Individual Server (VIS). A VIS is a personal virtual machine running in a paid compute utility where the data will be stored and access by other friends. VISs self-organize into relay networks correspond to social groups.

3.3.2 Analysis
Pure cloud architecture proved to be the superior solution compare to pure p2p and distributed server architecture. Firstly, since all the data is put on users’ cloud, the privacy is ensured without the need of any complex cryptographic scheme. Moreover, the availability of the data and the usability is also high due to natural characteristic of cloud computing.

However, the fact that users have to pay for their own cloud to use this solution puts a huge obstacle in the way of popularizing the architecture. When there is already a out-standing, free of charge service, nobody is eager to switch to a beginning, expensive one.
4. Distributed and Secure Online Social Network Mobile Application Designs

After reviewing the three different architectures of DOSN, there are three key problems of the current solutions that is: The usability of security schemes and the locality of trust (pure p2p), the lack of self-protection means for users from provider (Diaspora) and the operating cost (pure cloud). In this chapter, we present our proposed new architecture for DOSN, which is suitable in mobile context and remedy the aforementioned problems.

4.1 Design goals

The target of our architecture would be to solve these existing problems plus the problem in energy and resource constrains in mobile context. Overall, the design must satisfy the following goals:

- Security usability: there have to be a trusted third party security provider to provide global trust instead of letting users to manage trust and certificate by themselves. The security architecture should simple enough to be used by normal users and administered by trusted party.
- Users protection: the architecture should provide means to assure that users’ privacy is protected from both security provider and storage provider. Even in a context that a few of providers collude.
- Operating cost: the operating cost should be low for the trusted third party provider and should be free for users.

4.2 System architecture

Besides satisfying the three above design goals, the system architecture also has to solve the resource constrains problem and the security problem in mobile device.

Regarding the resources constrains problem, it is naturally that the entire heavy-resource demand task should be outsourced to clouds or servers meanwhile only core task should be keep in the device. With such an analysis, the system will be separated into three main components: the content storage provider component, the security provider component and the communication service component. Since the content storage attaches with data access control and content provider, it is likely that this heavy task should be outsourced somewhere which eager to host the data free and provide data access control. The ideal candidates are the free cloud storage service providers. The security provider component also need to be outsourced to a trusted third party to meet the requirement of global trust. Users, therefore, will need means to protect themselves from security provider and storage providers. Finally, since the communication service is the essential and inseparable component, it should be put in the peer-to-peer network that run by users. At the same time, due to the easy-to-compromised characteristic of secret in mobile device, it is evidently that only session keys should be kept in
mobile device while top private secret would be manage by your storage and security providers.

With the motivation as above, our architecture will consist of three main components illustrated as the figure below.

Fig 7. System architecture

4.3 Entity design

In the proof-of-concept prototype, information that peers in the first prototype system exchange is entity rather than raw message. Although entity introduces more overheads, it supports a strong polymorphism and allows developer to extend the kind of information exchange rapidly.

**BaseEntity:** All the entities are subclass of this abstract class. BaseEntity is “serializable” and contains information about creator identifier id and timestamp that give assistant to the identity-based cryptography.

**SocialProfile:** contains basic information of a user such as the topic to subscribe to users’ publication, the topic that user subscribes to (To send message to him), wall object URL,
personal information. Often be uploaded to DHT and is used to create entity such as notification, message for users.

**NotificationType:** is a enum class, consists of: NEWFEEDS, NEWCOMMENT, NEWLIKE, IPUPDATE, PROFILEUPDATE, SESSION_KEY_CHANGE

**Notification:** contains a NotificationType, a description and a map of location to fileId to provide information about the location of the real entity store on the cloud. Notification is used to exchange location information of the content between peers.

**Message:** an entity represents a text message between one user to another user or a group. Contain a content of the message and conversation name to signify where the message belong to.

**Status:** an entity represents a status of a user. Contain a content of the status and the topic attach with the status to comment to or like it.

**EncryptedEntity:** an entity represents an encrypted entity. Contain the SessionKeyId use to encrypt it, a sealed object of the encrypted entity and file location of the encapsulated key on the cloud. Each SessionKeyId is unique to let user know if he already has the correspond key to decrypt the entity or has to fetch from the cloud.

**KeyHeader:** contains information of the encapsulated key: the SessionKeyId and the EncapsulatedKeyMaterial and an array of legal lds for this key.

**KeyInfo:** contains the genuine value of the session key and the SessionKeyId.

### 4.4 Design justification

#### 4.4.1 The Emergent Trend in Free Cloud Storage Service

It is worth considering the rationale behind choosing free cloud storage service as one component in the system architecture. Indeed, there are four main reasons leading to our decision.

The most important reason comes from the fact that free storage service enhances the usability of the system. Since users want to control over their data, it is impossible for them to check whether the access control list is precisely set or if the data is genuinely deleted in case of using DHT storage. Meanwhile using free cloud storage service, users are provided with a webpage graphical interface to double-check what truly happen to their data.

Secondly, as most of cloud storage service providers supply a moderate amount of free space for users that is sufficient for personal demand, the use of cloud storage service allows our
system outsourcing the costly access control list operation away from p2p network meanwhile keeping the operation cost free for users.

Thirdly, since the cloud storage is high availability and support multi-platform API, it is no doubt that the system can easily extend to support new cloud storage service provider by mapping the new API to the system API.

Last but not least, due to a forecast from HIS as in Fig 8, it is evidently that cloud storage service users will increase dramatically in the near future and making our system widely adopted.

\[\text{Fig 8. The worldwide forecast for Personal Cloud storage users. Data source from: [36]}\]

### 4.4.2 Identity management with Identity-based cryptosystem

In the distributed context of millions of users as in OSN, such a task as identity management and authentication is non-trivial.

At the very beginning of the project, there are two different approaches for security with the use of certificate and public key cryptography has been proposed. In the first approach, each user maintains a group certificate for group communication between his friends meanwhile in the other users will use trivial identity encryption by encrypting the session keys n times with different public key. However, it is infeasible to maintain a certificate authority infrastructure strong enough to provide such a huge amount of certificate without a considerable operation cost because of the neat requirements of certificate authorities [37]. Also, due to the importance of establishing global trust between peers, these two approaches have been excluded.
Another approach, which is to deploy a Kerberos key distribution center to exchange session key between users, also is considered. With the mature in technology, Kerberos proved to be a promise solution for a variety of distributed application. Yet, there is a formidable that is difficult for Kerberos to solve. The idea of Kerberos exchange protocol is that the key distribution center already shared secrets with users. When Alice wants to exchange session key with Bob, she will send request to key distribution; receive the session key and an encrypted version of its with the key is the shared secret between Bob and the server. Alice then sends the encrypted session key to Bob and they establish a share secret. To defend against replay attack, the protocol need to add timestamp and salt to the messages, requiring that both parties should be online within a specific period of time to assure that the receiving message is legal. This is a tight requirement in OSN context where the online periods of users are diverse. In addition, the distribution key center knows every single session keys, which raise a concern to privacy of the solution.

Identity-based cryptography is the only solution that solves the entire problems. On the one hand, it removes the need of heavyweight infrastructure for certificate authorities while provides both a flexible and secure key exchange scheme as well as global trust with identity-based signature. On the other, thanks to the combination with cloud storage access control support, the notorious drawback of identity-based crypto, which is key-escrow problem, also mitigates. By separating encapsulated session key and encrypted content to different clouds, we also claim that unless there is collusion between the cloud store session keys, the cloud store encrypted content and the private key generator; the privacy of users might not be stripped.

4.5 Attacker model
In the attacker model, we assume that trusted party such as cloud provider and private key generator is semi-trusted and will attempt to violate users’ privacy as long as they have chances. Since all the contents is uploaded to cloud storage providers, it is assumed to be safe from the hand of malicious users other than the providers themselves.
5. Implementation

This chapter describes the implementation of the distributed and secure social network mobile application system presented in Chapter 4. The system includes a private key generator server served as the security backbone, an API to extend the cloud free storage with an illustration in GoogleDrive and the Android application for users.

5.1 Experiment setup

All experiments are performed on two HTC One X (developer’s version) [38]. The devices have public IP to serve the purpose of DHT bootrapping. The code for the desktop application is provided in Appendix A.2. and the Android version is provided in Appendix A.3.

<table>
<thead>
<tr>
<th></th>
<th>HTC One X</th>
<th>Desktop Fujitsu</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Android 4.2.2 Jelly Bean</td>
<td>Ubuntu 12.04 LTS (Precise Pangolin)</td>
</tr>
<tr>
<td>Processor</td>
<td>1.5 GHz, quad-core</td>
<td>Intel Core i7-2600 3.4GHz, quad-core</td>
</tr>
<tr>
<td>RAM</td>
<td>1 GB</td>
<td>8 GB</td>
</tr>
<tr>
<td>Wifi</td>
<td>IEEE 802.11 a/b/g/n</td>
<td>IEEE 802.11 a/b/g/n</td>
</tr>
<tr>
<td>Web-server</td>
<td>Glassfish server 4.0</td>
<td></td>
</tr>
<tr>
<td>JVM version</td>
<td>Java 7 update 40 (7u40)</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Devices use in the experiments

5.2 Reliable publish/subscribe on FreePastry

Since the SCRIBE library only supports best-effort publish/subscribe, the first obstacle in implementation the system is to extend SCRIBE library to support reliable publish/subscribe. We have followed the instruction of the author in the original paper [38] to change the forwardHandler such that the root assigns a sequence number to each message and such that recently multicast messages are buffered by the root and by each node in the multicast tree. Messages are retransmitted after the multicast tree is repaired. The faultHandler add the last sequence number, n, delivered by the node to the JOIN message and the joinHandler retransmits buffered messages with sequence number above the new child.

The sequence number is chosen as the timestamp and each node will synchronize time source using NTP protocol. Thanks to this choice, even in the case when some of the messages lost due to root node and replicate nodes failure, the order of message is still consistency.

To tolerate root failures, the root nodes will be replicated to m nodes with the closest adjunct identifier nodeId. Each time a node become the new root of a topic, it pull a set of root cache from replica and use the Paxos algorithm [38] to ensure strong consistency of the cache.

5.3 Identity-based cryptography components

The identity-based cryptography in the first prototype-system is developed under the consultant of the “Identity-Based Encryption Architecture and Supporting Data Structures” [22].
All the pairing-based operations are implemented in in the Java Pairing-Based Cryptography Library (JPBC) [39].

5.3.1 Secure parameters selection
The system parameters are chosen as follows. Firstly, type A pairings are used, as described in [38] due to the performance requirement. These pairings are constructed from the super singular curve $y^2 = x^3 + x$ over the base field $\mathbb{F}_q$, where $q$ is a prime number. The multiplicative cyclic $\mathbb{G}$ using is a subgroup points of $\mathbb{F}_q$ which is in the elliptic curve $E(\mathbb{F}_q)$. The order of $E(\mathbb{F}_q)$ is $q+1$ meanwhile the order of $\mathbb{G}$ is $p$ where $p$ is a prime factor of $q+1$.

In order to initiate type A pairings in JPBC, we have to choose the size of primes $q$ and $p$. We select the recommendation values of $p$ and $q$, which are 160 bits for $p$ and 512 bits for $q$. With such parameters selection, the discrete logarithm is difficult both in $\mathbb{G}$ and $\mathbb{F}_q^2$. Specifically, the system is claimed to have the discrete logarithm security level of $(L=1024, N=160)$ (The security level is defined as in [41]). This security level is mentioned to be enough until 2015 as in [41]. Moreover, since the certificates of identity-based cryptography should be renewed after a short period as suggested in [42], the time for that the break of the crypto system is meaningful indeed is much more shorter than in the public cryptography case.

5.3.2 Private key generator (PKG) server
The PKG stores and uses cryptographic material, known as a master secret, which is used for generating a user’s IBE private key. For the ease of PKG administration, we also provide a toolkit to generate both master secret key and master public key. All the requests to PGK server are https follow the security suite of RSA encryption with key size 2048 bits and AES-128 symmetric. The code for the PKG server and master secret generate toolkit is provided in Appendix A.5.

5.3.3 Certificate parameters definition

5.3.3.1 Public Parameters definition
The IBE public parameters are defined with the following structure of the forms

```
IBESysParams ::= SEQUENCE {
    version                INTEGER { v2(2) },
    districtName           IA5String,           
    districtSerial         INTEGER,            
    validity               ValidityPeriod,      
    ps06PublicParameters   IBEPublicParameters, 
    cd07PublicParameters   IBEPublicParameters, 
    ibeIdentityType        OBJECT IDENTIFIER,  
}
```
The most important information in the system public parameters is the two public key parameters of two identity-based schemes using in this thesis. The cd07PublicParameters is the public parameters of the broadcast-identity based encryption, i.e. the curve parameters to reconstruct pairings and a public header \( \text{PK1} = (\omega, v, h, h^v, \ldots, h^{v_m}) \) where \( \omega = g^v \), \( v = e(g, h) \), \( g \in \mathbb{G}_1 \), \( h \in \mathbb{G}_2 \) and \( \gamma \in \mathbb{Z}_{p^*} \) are randomly chosen. On the other hand, ps06PublicParameters is the public parameters of the signature scheme which contains another curve parameters and a public header \( \text{PK2} = (g, g_1, g_2, u', U, v', V) \) which is described in [28].

### 5.3.3.2 Private Parameters definition

Similarly, the IBE system master secret parameters and IBE client secret parameters are defined with the following structure of the forms

\[
\text{IBESysMasterParams ::= SEQUENCE }
\]

\[
\begin{align*}
\text{version} & \quad \text{INTEGER \{ v2(2) \},} \\
\text{districtName} & \quad \text{IA5String,} \\
\text{districtSerial} & \quad \text{INTEGER,} \\
\text{validity} & \quad \text{ValidityPeriod,} \\
\text{ps06MasterSecretParameters} & \quad \text{IBEParameters,} \\
\text{cd07MasterSecretParameters} & \quad \text{IBEParameters,} \\
\text{IBEIdentityType} & \quad \text{OBJECT IDENTIFIER,}
\end{align*}
\]

\[
\text{IBEClientSecretParams ::= SEQUENCE }
\]

\[
\begin{align*}
\text{version} & \quad \text{INTEGER \{ v2(2) \},} \\
\text{districtName} & \quad \text{IA5String,} \\
\text{districtSerial} & \quad \text{INTEGER,} \\
\text{validity} & \quad \text{ValidityPeriod,} \\
\text{ps06IdSecretParameters} & \quad \text{IBEParameters,} \\
\text{cd07IdSecretParameters} & \quad \text{IBEParameters,} \\
\text{IBEIdentityType} & \quad \text{OBJECT IDENTIFIER,} \\
\text{clientIdentity} & \quad \text{IA5String,}
\end{align*}
\]

We also shall mention that the cd07MasterSecret MSK1=(g, \gamma), cd07IdSecretParameter IDSK1=(sk_\alpha), ps06MasterSecretParameter MSK2=g^{2\alpha} and ps06IdSecretParameters IDSK2=(d_\alpha) are described meticulously in [27], [28].

The other parameters definition is followed the consultant in [40] and is provided in Appendix B

### 5.4 Cloud storage service API

Our cloud storage API is defined to hide the specific implementation of different cloud storage provider API. As we mentioned from the beginning of the thesis, the aim of the API is to extend
the cloud storage provider on the fly as long as the cloud storage provider API supports the requirement of our system, specifically creating folder, creating files, setting access permission to folder, setting access permission to file, remove permissions, upload content to specific folder.

In the following figure, we demonstrate the API of the proof-of-concept system in java language.

```java
public interface CloudStorageService {
    public List<String> initializeDSNAFolders() throws UserRecoverableAuthIOException, IOException;
    public String uploadContentToFriendOnlyFolder(String title, String type, String description, InputStream content) throws UserRecoverableAuthIOException, IOException;
    public String uploadContentToPublicFolder(String title, String type, String description, InputStream content) throws UserRecoverableAuthIOException, IOException;
    public List<String> addPermission(String fileId, List<String> userIds, String type, String role) throws UserRecoverableAuthIOException, IOException;
    public void removePermission(String fileId, String permissionId) throws UserRecoverableAuthIOException, IOException;
    public void removePermission(String fileId, String userId, String permission) throws UserRecoverableAuthIOException, IOException;
    public String createFolder(String title, String description, String parentId) throws UserRecoverableAuthIOException, IOException;
    public String createFile(String title, String type, String description, String parentId, InputStream content) throws UserRecoverableAuthIOException, IOException;
    public void getFile(String fileId, Continuation<InputStream, Exception> action);
}
```

**Fig 9.** The cloud storage service API

At the moment, there is only an implementation of the API for GoogleDrive, which is provided in Appendix A.1. Among various cloud storage services, we select GoogleDrive for the proof-of-concept system as it provides a fine-grain access control, robust network and a well-supported Android API. The extend version should support various different cloud storage such as DropBox, SkyDrive, etc.
Fig 10. Data stores in Google Drive

5.5 Demonstration use case

5.5.1 Joining OSN use case
Use case scenario: Alice wants to join the DOSN and follow her friend, Bob, which is already a user.

1. Alice connect to PKG and get public master key
2. Alice authenticate with PKG and get Alice’s private key
3. Alice use her private key to authenticate by signing a salt and join the p2p network
4. Alice uploads her profile to DHT and query Bob’s profile from DHT (Given that Alice know Bob’s Id)
5. Alice add Bob as friend using information from Bob’s profile
6. Alice follow Bob’s topic to get new contents from Bob

5.5.2 Publish content use case
Use case scenario: Bob wants to publish new contents to his friends.
1. Bob upload the encrypted content to clouds with access control list only to his friends
2. Bob publish the encrypted, signed notification that contains the content’s location in cloud via SCRIBE
3. Alice receive the notification, verify it and decrypt it
4. Alice fetch the EncryptedEntity from cloud
5. Alice check the sessionKeyId to see if she has the key, if not, Alice fetch the KeyHeader from the cloud and use her private key to decapsulate the key, put it to local database
6. Alice decrypt the EncryptedEntity with session key correspond to sessionKeyId
7. Alice view the clear content

5.5.3 Changing session key use case
Use case scenario: Bob wants to change his session key and distribute it to his friends

1. Bob run Identity-based broadcast encryption algorithm and get the two following: a KeyInfo contains the key material and a KeyHeader contains encapsulated key
2. Bob store the KeyInfo to database as his new session key.
3. Bob upload the KeyHeader to clouds with access control list only to his friends
4. Bob publish the signed notification that contains the KeyHeader location in cloud via SCRIBE
5. Bob’s friends receive the notification, verify it
6. Bob’s friends get the KeyHeader from the cloud, check the sessionKeyId to see if they has the key, if not, they decapsulate the key, put it to local database
7. Now Bob and his friends share the same session key.
6. Performance Evaluation

In this chapter, we evaluate the performance of our proposal, and compare the result with different configuration. Since we are among the very first who proposed a DOSN architecture for mobile devices, it is impossible for us to compare our result with related work in term of performance and energy consumption. Therefore, the evaluation is dedicated to measure the overhead added while applying identity-based cryptography to the system. In order to measure the performance in a controlled environment, we build a custom-made mobile application that sends content or encapsulated keys at specific intervals, we measure the power of the standard application in different assumption contexts. Before discussing the results, we present the methodology we follow to obtain the results.

6.1 Methodology

Since we cannot simulate the genuine situation of OSN to evaluate the target overhead, the performance is evaluated using a model to estimate the average core operation a user will use in OSN situation. We will then measure average process time and energy of core operations, namely key decapsulation, encrypted items process, plain items process from empirical data, and apply this figure to the model to estimate the target overhead.

The scenario in our system is users daily distribute their session keys to friends beforehand. And when one wants to publish contents to friends, he merely selects the latest session key to encrypt contents before distributing. In our assumption, the amount of contents a user publishes each day is tiny compare to those receiving from community, meaning the cost for publishing contents is negligible. We, therefore, concern only the cost for contents receiving. Hence, the overhead added while encryption enabled is the cost of session keys decapsulation plus the different cost between encrypted items process and plain items process. The model is built from statistic figures in [43], assuming that an user has average 300 friends in OSN and 200 of them actively publishes averagely 3 contents daily in our model. A user averagely decapsulates 200 key headers and process 600 per day.

To collect the empirical data for measuring average process time and energy of core operations, we implement a custom-made application that sends status and encapsulated keys at fixed interval. Specifically, in the first and second experiment, the application will send 1 status with constant size of 2925 bytes each 2 seconds for 1000 status in two different cases: plain and symmetric encryption status, with the assumption that the session key is distributed before hand. In the second experiment, the application will send 1 encapsulated key each 2 seconds for 1000 keys. Each experiment is conducted when the battery energy is 100% to guarantee that the figures are comparable. After an experiment, the device is charged to full before conducting the next one. We then measure the process time and the power
consumption of the standard application in each case and discuss the practicability of our proposal system in some assumption use case.

It is important to note that our main goal is to measure how efficient the new security scheme is in energy consumption and processing time since it is already proved to provide global trust and keep privacy with the security level of 1024 bits key size in the implementation.

6.2 Result

6.2.1 Power consumption

Figure 12 shows the power consumption in three cases: processing 1000 plain status, processing 1000 encrypted status and decapsulation 1000 key headers. We observed that there is no considerably difference between the power consumption in processing encrypted status and that in processing plain status. Hence, the significant overhead added in encryption mode is the energy spent for session keys distribution. Due to the model that users have averagely 300 friends in OSN and 200 of them actively publishes contents daily; an user, therefore, averagely should decapsulate around 200 key headers from his friends. According to the empirical figure, decapsulating from 150 to 230 key headers only accounts for 3 percentage of the total battery would be the energy overhead added while keeping your content confidentially distribute to your friends.

![Graph: Power consumption over processing plain status](image)

(a). Power consumption over processing plain status
6.2.2 Process time

Fig 13 shows the process time of the 2 different deeds in encryption mode, specifically decrypting an encrypted entity and decapsulating a key header. By measuring overall 1000 samples, we observe that the average time for decryption is around 10 milliseconds meanwhile decapsulating a key header is a resource-consuming task that requires around 1053 milliseconds on average. The process time of decapsulating session keys is 100 times higher than the symmetric decryption due to the heavy-weight of identity-based cryptography. Each decapsulation need to compute $O(m^3)$ multiplications and additions and a pairing operation where $m$ is the number of user identifiers of that session key. Fortunately, the decapsulation task each day is fixed and can be uniformly distributed as we could use an algorithm to assure that users will renew their session key randomly following a uniform distribution within a period of time, given that the renewing session key condition has met. Overall, the overhead in
process time mostly would be the symmetric decryption that is 10 milliseconds on average. (Note that it is empirically computed from our data that is provided in Appendix C)

(a). Process time of encrypted entity in AES 128 bit-mode

(b). Process time of key decapsulation

Fig 12. Process time in encryption mode
7. Discussion

7.1 Summary of result
In this thesis, to gain a general picture about how far DOSN solution have reached, we started by reviewing what is OSN, its security and usability requirement, along with core technology which is used in current DOSN solutions, i.e. Distributed Hash Table and Identity-based cryptography. After that, we chose four current DOSN solutions from three different architectures were proposed, examined their strength and weakness as well as how far they have reached the demand of DOSN. By examining specific solutions, we gathered detailed requirements and needs of users, which lacks in each separate solution. From that, we gained a clear idea of the current security objectives and usability requirements of the system. Based on these security and usability goals and knowledge about properties of DOSN, we chose a suitable type of component and cryptography for the new architecture and designed security protocols for the system. We also developed proof-of-concept implementation for the proposed architecture in this thesis. In addition, we developed an experiment to explore the trade-off between security and energy consumption in our architecture.

7.2 Comparison to existing solutions
Since our system is the first one aim at DOSN in mobile context, it is beyond the bound of possibility to compare our result in terms of energy consumption and processing time with the existing solutions. For this reason, we decide to make comparisons regard of privacy supports, trust management, number of stored certificates and content delivery support between our system versus the existing solutions.

<table>
<thead>
<tr>
<th></th>
<th>Our system</th>
<th>PeerSoN+LifeSocial.KOM</th>
<th>Diaspora</th>
<th>Vis-à-vis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy protection</td>
<td>Identity-based cryptography 1024 bits + AES 128 + Session keys and contents separated</td>
<td>Public key cryptography + AES 128</td>
<td>None</td>
<td>Does not require</td>
</tr>
<tr>
<td>Trust management</td>
<td>By providers</td>
<td>By users</td>
<td>By providers</td>
<td>By users</td>
</tr>
<tr>
<td>Num of stored certificates</td>
<td>2</td>
<td>1+d_friends</td>
<td>0</td>
<td>d_friends</td>
</tr>
<tr>
<td>Num of session keys</td>
<td>d_friends</td>
<td>d_items</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Permission mod cost (add/remove)</td>
<td>O(1) / O(1)</td>
<td>O(1) / O(1)_symm + O(n)_asym</td>
<td>O(1) / O(1)</td>
<td>O(1) / O(1)</td>
</tr>
<tr>
<td>Encryption time</td>
<td>$O(1)$symm</td>
<td>$O(1)$symm + $O(n)$asymm</td>
<td>$O(1)$symm + $O(n)$asymm</td>
<td>Unknown</td>
</tr>
<tr>
<td>----------------</td>
<td>------------</td>
<td>---------------------------</td>
<td>---------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>Decryption time</td>
<td>$O(1)$symm</td>
<td>$O(1)$symm + $O(1)$asymm</td>
<td>$O(1)$symm + $O(n)$asymm</td>
<td>Unknown</td>
</tr>
<tr>
<td>Key distribution time</td>
<td>$O*(d_{friends})$asymm</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Dynamic group key distribution</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Event Delivery</td>
<td>Push model</td>
<td>Pull model</td>
<td>Push model</td>
<td>Pull model</td>
</tr>
<tr>
<td>Friend follow</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Hash-tag follow</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>Secret leaked If device lost</td>
<td>Temporary secret</td>
<td>Permanent secret</td>
<td>Temporary secret</td>
<td>Temporary secret</td>
</tr>
<tr>
<td>Operation cost</td>
<td>Free</td>
<td>Free</td>
<td>Free</td>
<td>Cloud renting cost</td>
</tr>
</tbody>
</table>

Table 5. Functionality and security comparison to existing solution

7.3 Limitations
In this section, we list the main limitations of our DOSN proposal:
Social contents: Contents such as comments, likes, and tags are the challenge in not only our proposal but also the others. On the one hand, the fact that they have different owners leads to a challenge in their access control list, i.e. who should have the privilege to add and remove comments, and who could view the comments. In PeerSoN, the system solve the case by only allowing the main content owner to remove the comments while friends only have privilege to add comment. It is rather awkward solution as there is a case when your friend will be away from OSN and cannot urgently remove your sensitive comment. In our system, we suggest to use a topic attached with the main content to publish notification of friends’ comment. Hence, the privilege to add, remove and allow users to view the content belongs to its owner. Although the solution is more flexible, there exists a drawback in this solution, i.e. the notification does not stay forever since it has limited lifetime on DHT cache. We suggest to implement a solution that owner of the main content should subscribe to his content topic and added the notification to the content itself to assure the consistent of the comments, likes and tags. Users then should synchronize both notification they get from main content and its topic to have the all-inclusive record.
Replication root node: Although we follow the suggestion of Scribe’s authors to implement the root node replication, it still needs further exploration to figure out the optimized configuration for root node replication. Also, the lifetime of publish/subscribe items cache is infinite at the moment and will need further study to determine for how long it should be maintained.
Heavyweight identity-based encryption: The library we used is JPBC which is the java porting version of the well-know PBC library of Stanford university. The benchmark of the library showed a huge-different between the C version and the Java version due to the immature of implementation phase. Therefore, there should be further study to explore new way to optimize performance of identity-based cryptography to boots up the performance on Android platform.
8. Conclusion

DOSN technology is evolving rapidly with the aim to protect users’ privacy from their evil service provider. With the shipping of OSN from traditional desktops to mobile devices, the demand of a DOSN solution for mobile is urgent.

Although a variety of DOSN systems have been proposed, they have many problems, such as poor usability, potential privacy concerns, and unrealistic operation cost as we explained in Chapter 3. One critical challenge in DOSN is to ensure usability and privacy while keeping the operation cost free. However, we have discussed in Chapter 2 that ensuring privacy and manage trust is non-trivial in OSN context because of the tremendous amount of users’ identities.

The thesis illustrated that by separating storage and security provider, we could considerably enhance the usability as well as privacy of the system while keeping the operation cost is free. This model introduces a new structure which have three components: security provider, storage provider and communication provider. While the communication provider is the peer-to-peer network constructed from users, the security provider is proposed using identity-based cryptography and consists of Private Key Generator Servers and KeyToolGenerator and the storage provider is the popular free cloud storage service.

Our implementations in this thesis are working prototypes providing the essential core features which are usable as a solid base for future developments and experiments. This was a challenging and interesting project since it involved research and development in many different areas, including working on DHT and peer-to-peer technology, writing Android application, Java programming, cloud API marshaling, Java web programming, Identity-based algorithm implementation.

The very first experiment result of the system shows that there is no significant difference between symmetric encryption and non-encryption mode. And the overhead cost for distributing session key using identity-based cryptography with key-size 1024 bits shown in empirical data only accounts for 3% of battery on average for a user who has 200 friends actively sending content daily. The overhead process time is just around 13 milliseconds. These results signify a bright future of DOSN on mobile devices.

To the best of our knowledge, this is the first work that presents how to apply the identity-based cryptography and peer-to-peer technology to decentralizing OSN on Android. Possible future work after this thesis should consider the following:

• Discover and study other use cases of OSN that could be supported by the system.
• Discover different DHT publish/subscribe libraries and pair-based cryptography libraries to get further knowledge and comparison on performance on the system and make the optimal choice.

• Improve the applications for real-world deployments. For example, we can add NAT-traversal technique to the current Pastry library or add the bootstrap node information-collecting algorithm to assist users join DHT more flexible.
REFERENCES


[34] “Diaspora (social network),” Wikipedia, the free encyclopedia. 18-Jun-2014.


[37] A. Shakimov, H. Lim, R. Caceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky, “Vis- #x00E0;-Vis: Privacy-preserving online social networking via Virtual Individual Servers,” in 2011 Third International Conference on Communication Systems and Networks (COMSNETS), 2011, pp. 1–10.


Appendix A: Source code

A. 1 DHT-based backbone library
The implementation of the whole library, including: cryptography package, dht package and cloud storage service is provided in
https://github.com/kekkaishivn/DSNA-Application/tree/Reliable-PubSub/FreePastryClient/src/com/dsna

A. 2 Java desktop client source code
The client java with GUI code is provided in
https://github.com/kekkaishivn/DSNA-Application/tree/Reliable-PubSub/FreePastryClient/src/com/dsna/desktop/client/ui

A. 3 Android client source code
Source code of demo Android client is provided in
https://github.com/kekkaishivn/DNSA_MobileCode

A. 4 Key extractor source code
Source code using to generate system public key and master secret key is provided in
https://github.com/kekkaishivn/DSNA-Application/blob/Reliable-PubSub/FreePastryClient/src/com/dsna/crypto/asn1/params/ExtractKey.java

A. 5 Private key generator server source code
Source code of the proof-of-concept of private key generator server is provided in
https://github.com/kekkaishivn/DSNA_PrivateKeyGeneratorServer/tree/master/WebContent
Appendix B
Certificate Parameters definition

The fields of an IBESysParams structure have the following meanings as defined in [44]:

“The version field specifies the version of the IBESysParams format.
The districtName field is an IA5String that must be encoding of an URI.
The districtSerial field is an integer that represents a unique set of IBE public parameters that are available at the URI or IRI defined by the districtName. If new parameters are published for a districtName, the districtSerial MUST be increased to a number greater than the previously-used districtSerial.

The validity field defines the lifetime of a specific instance of the IBESysParams and is defined to be the following:

```
ValidityPeriod ::= SEQUENCE {
    notBefore GeneralizedTime,
    notAfter GeneralizedTime
}
```

IBEPublicParameters is a structure containing public parameters that correspond to IBE algorithms that the PKG supports. This structure is defined to be following:

```
IBEPublicParameters ::= SEQUENCE (1..MAX) OF IBEPublicParameter
```

```
IBEPublicParameter ::= SEQUENCE {
    ibeAlgorithm OBJECT IDENTIFIER,
    publicParameterData OCTET STRING
}
```

A client must verify that the date on which it uses the IBE public parameters falls between the notBefore time and the notAfter time of the IBE public parameters, and it must not use the parameters for IBE encryption operations if they do not”
Appendix C: Empirical data

The battery empirical data collected is automatically upload to
https://drive.google.com/?tab=mo&authuser=0#folders/0B6UVLt6r0KtpTkJQUlg4eGhYSE0

The status and key-header sent is upload to

https://drive.google.com/?tab=mo&authuser=0#folders/0Bww74p4QwAPVbHFKX2VndUxDWkU