A SECURE MOBILE SYSTEM TO SUPPORT CITIZEN JOURNALISM

CSISO801 Final Year Project FYP11015

Final Report

Supervisor:
Dr. S.M. Yiu

Student:
Lam Harry To Chun
(2009177272)
Lam Wan Yin, May
(2009148398)
Li Ka Yau, Ray
(2009566813)
Lee San Tai, Eric
(2009565273)
# Table of Contents

Introduction .................................................................................................................. 3

1. Our Scheme ............................................................................................................... 4
   A. Design Flow ........................................................................................................... 4
   B. Comparison to existing scheme ......................................................................... 8
   C. Other functions .................................................................................................... 9

2. Assumptions ............................................................................................................. 9

Progress ......................................................................................................................... 10

1. Windows Azure Cloud Server ................................................................................. 10
   A. Setting up Azure (Handled by May and Harry) ................................................. 10
   B. Setting up SQLAzure (Handled by May) .......................................................... 11
   C. Cloud database connection, SQLservice.svc (Handled by Harry) .............. 14
   D. Setting up image service (Handled by May) ..................................................... 16
   E. Push Notification (Handled by Ray) ................................................................. 17

2. Windows Phone Application ...................................................................................... 20
   A. User Interface (Handled by Ray) ...................................................................... 20
   B. Application login and access control (Handled by May) ................................. 24
   C. Session Key generation (Handled by May) ....................................................... 26
   D. Watermark and thumbnail generation (Handled by Eric) ............................... 27
   E. Photo Upload Service, UploadHelper.cs (Handled by Harry) ....................... 28
   F. Building the crypto class (Handled by Harry) ................................................... 29
   G. Ring Signature Scheme (Handled by Harry) ..................................................... 32
   H. Photo Selling Scheme (Handled by Harry) ....................................................... 35
   I. Map Page (Handled by Eric) ............................................................................. 39
   J. Album Page (Handled by Eric) .......................................................................... 41

3. Web portal ................................................................................................................ 42
   A. Access Control (Handled by May) .................................................................. 42
   B. Home Page (Handled by May) .......................................................................... 44
   C. Private Page (Handled by May) ....................................................................... 45
   D. Map Page (Handled by May) ........................................................................... 47

Testing and Performance Evaluation ............................................................................ 49

1. Photo Encryption (AES) .......................................................................................... 50
2. Photo Decryption (AES) ......................................................................................... 51
3. Ring Signing Time ................................................................................................... 52
4. Ring Verification Time ............................................................................................. 53

Project Management .................................................................................................... 54
Project Timeline

Challenges

1. Problems on WCF services
2. Encryption on phone
3. Immature SDK

Further Development

1. Anonymous Photo Selling
2. News Agencies Subscription
3. Map Navigation
4. Blacklisting and Whitelisting
5. Anonymous Polling using Ring Signature

Conclusion
Introduction

Nowadays, many people have smartphones in their pocket. They use the phone camera to capture bits and pieces of everyday life. However, few people know that their photos are of much news value. In a vibrant and dynamic city like Hong Kong, news is always happening around us. News agencies want to get the first hand news and pictures of the news, but there is always a time lag between the happening of the event and the reporters arrive. In this situation, the smartphones of people near the site of event can help to capture the news in a much faster way. After that, they can sell the photos to news agencies. This is a win-win transaction as news agencies can get the latest news from the citizen photographers, and the photographers can have reward selling the photos.

Although photo sharing is easy on smartphones, it is often hard to have a secure channel to sell photos. Current implementation of photo sharing software doesn’t support encryption and digital signature. Photos may be leaked and tampered during the upload process. In sending photos to news agencies, one may even need to send photos through email. The one way process means people uploading the photo may not be guaranteed to have a reward if the photos are used in publication.

Furthermore, sometimes people may encounter dangerous situations that uploading photos may put the photographer at risk. To solve this problem, we need a tool to upload photos anonymously so that the photos are published but the identity of the photographer is not leaked.

Our project is to implement a smartphone photo-sharing platform that supports transaction of photos. There are 4 main functions of the platform:

1. A two-way process selling photos with encryption and digital signature
2. Ring signature for uploading photos anonymously
3. A map to let people know the news around them
4. Categorization of photos to improve organization

In the project, we aim to solve the problem of complicated photo selling process, as well as promoting citizen journalism to report breaking news in a faster way.
1. Our Scheme

In our project, we aim to speed up the process of photo transaction by using smartphone application and cloud technology. We also try to protect the interest of the seller and buyer by encryption and digital signature. Our project consists of three main pieces: the Windows Azure cloud server, the Windows Phone 7 application and the web portal hosted on Windows Azure.

A. Design Flow

First of all, when a user encountered an event that he thinks is of news value, he can immediately take a photo using the smartphone application. Our application design ensures that the user can finish the photo capture as quickly as possible.

Figure 1 - Capture photo of the event
After taking the photo, the user can categorize the photo by adding a tag so that people who are interested in such an event can find the photo quickly. The user can also add comments into the photo as a description to help the buyers to choose. When the tags and comments are set, the user can then upload the photo. The user can choose to encrypt the photo in here. If the user chooses not to encrypt it, everyone can download the photo. If he does, other users have to buy the photo in order to download it.

![Figure 2 - Adding tags and comments to photo](image)

After clicking the “upload” button, the photo will be uploaded to Windows Azure Blob storage. If the user intended to sell the photo, the photo will be encrypted with 1024bit AES encryption, while the key is stored in the phone waiting for other users to purchase. A thumbnail with watermark will also be uploaded to serve as a preview for customers to choose. The location of the photo taken is uploaded along with the photo.
Next, other users can view the thumbnail of photo along with its comments and tags in the application or web portal. If a user is interested in buying the photo, the user can click the "buy" button, along with the price he wants to give. A push notification will be sent to the seller of the photo for him to choose whether to sell the photo with the price or not.

If the seller agrees the sell the photo, the system will send a push notification back to the buyer, to notify him that the photo is ready to download.
When the buyer click the download button, the aforementioned amount of e-cash\(^1\) will be sent from the buyer to the seller, encrypted by the public key of the seller using 1024bit RSA encryption. At the same time, the AES key of the photo will be sent to the buyer, encrypted by the public key of the buyer, using 1024bit RSA encryption.

\(^1\) The e-cash platform is considered as an external system and is not being implemented in the project.

Finally, after the buyer received the AES key, he can decrypt the original photo and save it into the photo library. There will be a check in hash value of the photo so as to ensure the photo is the correct photo the buyer is supposed to buy.

---

**Figure 5 - Decryption of photo and transfer of E-cash**
B. Comparison to existing scheme

The table below shows a sketch of flow in selling photo using our platform, compared to the traditional way of selling photos to news agencies:

<table>
<thead>
<tr>
<th>Traditional approach</th>
<th>Our scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Steps</strong></td>
<td><strong>Steps</strong></td>
</tr>
<tr>
<td>1 Witness the event</td>
<td>1 Witness the event</td>
</tr>
<tr>
<td>2 Take a photo</td>
<td>2 Take a photo</td>
</tr>
<tr>
<td>3 Attach the photo to email</td>
<td>3</td>
</tr>
<tr>
<td>4 Send the photo to news agencies and wait for response</td>
<td>4</td>
</tr>
<tr>
<td>5 News agencies checked the photo and decided to use in publication</td>
<td>5</td>
</tr>
<tr>
<td>6 News agencies contact the photo seller by email/phone</td>
<td>6</td>
</tr>
<tr>
<td>7 Photo seller receives an award</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As compared to the original photo selling mechanism, the photo selling process using our scheme is speeded up by a wide margin and is much more secure. The seller does not need to wait for the news agencies for response (step 4 of traditional approach) and can guarantee to receive an award after buyer downloaded the photo (Step 6 of our scheme). Also, through the control in the application, we can limit the photos sold so that it cannot be sold twice. The notification and transaction is fully automatic so that the news agencies do not need to care about contacting the photo seller.
C. Other functions

In uploading the photo, the user can choose to do a ring signature. The system will not upload the identification of the user who uploaded the photo, so as to protect him from being found by malicious people. The ring signature allows us to ensure the person who uploaded the photo is a valid user of our platform.

In addition to the photo selling mechanism, our platform allows users to find out the news happening around them. In uploading photos, no matter they are encrypted or not, the location of the photo will also be uploaded onto the server. Users can search on the map in the phone application or web portal for news nearby. For news agencies, this would be a handy function as they can find out the accurate location of the news event from the map.

The server application and database are built on Windows Azure cloud platform. This requires us to build services in order to allow communication between the smartphone application and the server. We are also building the application on Windows Phone 7 platform in order to take advantage of the sophisticated access control on Microsoft platforms.

2. Assumptions

During our project, we utilize 3G network and GPS technology. We have made the following assumptions in order to make our project possible:

i) The city has 3G network coverage with fast internet speed

ii) The firewall in the internet connection can allow communication between phone and cloud server

iii) Users has obtained digital signature from trusted authority

iv) The Windows Phone installed with the application is with the following minimum requirement:

   CPU: 1 GHz
   RAM: 512 MB
   Camera: 5MP
   OS: Windows Phone 7.0 or above
Progress

As aforementioned, our project consists of three main pieces: the Windows Azure cloud server, the Windows Phone 7 application and the web portal hosted on Windows Azure. The three components of our project can be further divided into sub-parts and each member of our group is responsible to different parts.

1. Windows Azure Cloud Server

   A. Setting up Azure (Handled by May and Harry)

Since cloud server is a relatively new idea to us, we have to gather more information in order to understand how the server works. We have been using a month in testing the configurations of the server in order to meet our requirement.

![Figure 6 - Windows Azure Platform Control Panel](image-url)
After several trials, we decided to deploy 3 services on Azure, for handling the web portal, the image upload service and the SQL Azure database respectively.

![Figure 7 - Cloud services deployment status](image)

**B. Setting up SQLAzure (Handled by May)**

SQLAzure is the Database which host on Windows Azure Platform. We save all the user and photo records on the database. The advantage of using Cloud database is that it has a strong firewall that blocks external access. We can configure the database so that only services we hosted on Azure can access the database. This can prevent malicious attack on our database.

We have created 6 tables in total in the database.

i. **photo table**

This table is used to store the record of the photo, which includes url of the photo, uploader record, signatures and location records.

**Table 2 - Design of table “photo”**

<table>
<thead>
<tr>
<th>Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>photo</strong> id (int)</td>
<td>uploaderid (int)</td>
</tr>
<tr>
<td>security (int)</td>
<td>longitude (text)</td>
</tr>
<tr>
<td>hash (text)</td>
<td>ring (int)</td>
</tr>
<tr>
<td>z2 (varchar (max))</td>
<td>z3 (varchar (max))</td>
</tr>
<tr>
<td>h3 (varchar (max))</td>
<td>y1 (int)</td>
</tr>
</tbody>
</table>
ii. **tags table**

The tags table stores all the grouping information of photos. It also stores the number of views and rating of the photos.

<table>
<thead>
<tr>
<th>Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>tags</strong></td>
<td>photoid (int)</td>
</tr>
<tr>
<td></td>
<td>likenum (int)</td>
</tr>
</tbody>
</table>

### Table 3 - Design of “tags” table

iii. **user_table**

The table stores the Live ID, unique device ID, public key and push channel of the user. We check this table to ensure that one phone is only paired with one Live ID, in order to enable user access control. This can also ensure consistency when doing digital signature.

<table>
<thead>
<tr>
<th>Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>user_table</strong></td>
<td>id (int)</td>
</tr>
<tr>
<td></td>
<td>publickey (varchar(max))</td>
</tr>
</tbody>
</table>

### Table 4 - Design of “user_table”

iv. **buy table**

The table stores all the information of a transaction. The user ID of sellers are buyers are stored here, and the ciphers of AES keys are stored here.

<table>
<thead>
<tr>
<th>Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>buy</strong></td>
<td>buyerid (int)</td>
</tr>
<tr>
<td></td>
<td>id (int)</td>
</tr>
<tr>
<td></td>
<td>money (varchar(max))</td>
</tr>
</tbody>
</table>

### Table 5 - Design of “buy” table
v. **passtable**
   The table stores used session keys of users. Session keys allow the users to access other services.

   **Table 6 - Design of “passtable”**

<table>
<thead>
<tr>
<th>Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>passtable</td>
<td>id (int)</td>
</tr>
</tbody>
</table>

vi. **Push table**
   The table stores the push messages sent to users.

   **Table 7 - Design of “Push” table**

<table>
<thead>
<tr>
<th>Name</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>buy</td>
<td>id (int)</td>
</tr>
</tbody>
</table>

We have also created table views so that data can be easily obtain the data. The three (buysummary, summary and topview) table views link the tables of user_table, tags and photo for some features in the phone application and for the data collection in web portal.
C. Cloud database connection, SQLservice.svc (Handled by Harry)

SQLservice.svc receives the database queries from the phone and then executes them on SQL Azure (Cloud database). Originally there’s a service called Odata, which is provided by Microsoft to allow easy access to the SQL Azure. Odata would allow Windows Phone applications to send SQL queries directly to SQLAzure server and receive the returned result. However, the registration for the service is temporarily down, and yet to be resumed on late March. Since our project is ended in early April, we decided to create a workaround. We have implemented the interface ourselves. We decided to let the WCF service to handle all the database queries.

There are two kinds of methods in SQLservice.svc. One kind of them is of general use that receives queries from the phone application and execute on database. The other kind of functions performs specific functions related to the database to suit the needs of different functions on the phone.

i. Methods that receive queries

In creating the functions that receive queries from the phone, we need to preload the database structure into the service so that it can handle different data type. In order to make the data serializable, we have to fit the table returned into a list so that it can be transferred through the network. Also due to this problem, we have to separately create functions for executing queries for different database tables. Very unfortunately, sub-queries are not allowed using this scheme. Here is an example of the method:

```
public List<photo> selectPhotoQuery(string query)
{
  using (var context = new MMEEntities())
  {
    ObjectQuery<photo> reQuery = context.CreateQuery<photo>(query);
    List<photo> result = new List<photo>();
    foreach (photo temp in reQuery)
    {
      if (temp != null)
      {
        result.Add(temp);
      }
    }
    return result;
  }
}
```

Figure 9 - Method to retrieve data from "photo" table
In this function, we can execute a query on the table "tags", and the function will return the result in a list. The method call from the phone would be as follow:

```csharp
SqlHelper.selectPhotoQueryAsync("SELECT value photo FROM FMREEntities.photo AS photo ORDER BY photo.id DESC");
SqlHelper.selectPhotoQueryCompleted += new EventHandler<SqlServicePhotoQueryCompletedEventArgs>(selectPhotoQueryCompleted);
```

**Figure 10 - Method call for selecting data from "photo" table**

For queries that have no returning object, we can generalize it to be run on the same method. It is made available by using a different way to connect to the database and execute the queries. Implementation is shown below:

```csharp
public int executeQueryNoReturn(string query)
{
    string conn_str = ConfigurationManager.ConnectionStrings["FMREConnectionString"].ConnectionString;
    SqlConnection conn = new SqlConnection(conn_str);
    conn.Open();
    SqlCommand sql = new SqlCommand(query, conn);
    int row = sql.ExecuteNonQuery();
    conn.Close();
    return row;
}
```

**Figure 11 - Method for executing query with no return value**

**ii. Methods of specific use**

In creating methods that perform specific functions, we accept parameters from the method call and return specific value that the caller wants to get. Here is an example of this kind of functions that returns the cipher of the photo from photo ID provided: The calling of the method is as follow:

```csharp
SqlHelper.getKeyCipherFromIDCompleted += new EventHandler<SqlServiceGetKeyCipherFromIDCompletedEventArgs>(getKeyCipherFromIDCompleted);
SqlHelper.getKeyCipherFromIDCompleted(new EventArgs());
```

**Figure 12 - Calling of the method to retrieve cipher of photo of corresponding ID**

```csharp
public string getKeyCipherFromID(int photoID)
{
    string conn_str = ConfigurationManager.ConnectionStrings["FMREConnectionString"].ConnectionString;
    SqlConnection conn = new SqlConnection(conn_str);
    conn.Open();
    string sql = "SELECT keycipher FROM buy WHERE photoid = " + photoID + ";
    SqlCommand cmd = new SqlCommand(sql, conn);
    SqlDataReader reader = cmd.ExecuteReader();
    reader.Read();
    return (string)reader["keycipher"]
}
```

**Figure 13 - Method to retrieve cipher of photo with corresponding ID**
There are both advantages and disadvantages in using these two kinds of methods. The former one is more like traditional SQL queries but cannot handle sub-queries. Complex queries cannot be run. Also, there is always more overhead in data transfer as the table is returned from the cloud server to the phone application and let the phone process on the returned table.

For the later kind of methods, they lack flexibility as each method can only serve one purpose, but it can handle more complex queries. The cloud server handles the returned tables so that we can save more resources on the phone. There are fewer overheads in data transfer, as only a value is returned.

We think that the later kind of methods are more suitable to be used in companion with a phone application, as data transfer is expensive on mobile phones, and phones often have limited resources. In later development stage, most of the functions in the later development stage are designed in the second way.

D. Setting up image service (Handled by May)

Image Service is the Windows Communication Foundation for connecting the Windows Phone app with the Azure Blob. Azure Blob is a storage space on Azure that allows us to store uploaded files. The image service is deployed on Azure and added to the app as a service for handling the upload process of the photos. Here is the flow of photo uploading process:

1. The user takes a photo and the photo will be saved at Isolated Storage which is allocated on the phone
2. The phone app creates a new class called ‘CloudBlobUploader’ and save the photo as a stream file in it.
3. Then the phone app calls the Image Service (which should be deployed on Azure already) and passes the stream file to it.
4. The Image Service connects with the Azure Blob and uploads the stream file.
Here is the code for handling the upload process in the UploadService.svc:

```csharp
public Uri UploadUriWithSharedAccessSignature(string userId)
{
    try
    {
        InitializeStorage();
        var container = this.cloudBlobClient.GetContainerReference("imageservice");
        container.CreateIfNotExist();
        var sas = container.GetSharedAccessSignature(new SharedAccessPolicy()
        {
            Permissions = ContainerSharedAccessPermissions,
            SharedAccessExpireTime = DateTime.UtcNow + TimeSpan.FromMinutes(5)
        });
        var uriBuilder = new UriBuilder(container.Uri + "/" + userId) { Query = sas.TrimStart('?') };
        return uriBuilder.Uri;
    }
    catch (Exception exception)
    {
        throw new WebFaultException<ServerMessage>(exception.Message, HttpStatusCode.InternalServerError);
    }
}
```

![Figure 14 - Method for handling upload](image)

The container of the cloud blob is set to public. All people in the public can download the file on the Azure Blob as long as they have the URL of the file. Implementing session keys in the communication between the app and the Azure Blob will solve the security problems come along with this setting. The security problems aroused will be are discussed in the “Security Analysis” part in the report.

E. Push Notification (Handled by Ray)

Since there is no existing service provided by Microsoft to push notifications to the subscribed windows phone, we have to implement a WCF service ourselves. In order to build a WCF push notification service, we first created a WCF contract, which is an interface that controls the service of pushing notification to phones. The WCF contract is shown below:

```csharp
[OperationContract]
void SubscribeMyPhone(int phoneID, string channelURI);

[OperationContract]
void PushSurveyData(string rawMessage, int receiverID);

[OperationContract]
void PushToast(string toastTitle, string toastMessage, int receiverID);

[OperationContract]
void PushTileUpdate(string tileTitle, int tileCount, string tileImageURI);
```

![Figure 15 - Operation Contract for push notification](image)
After created the contract, we implemented the SubscribeMyPhone function. SubscribeMyPhone is a function that registers the phone to an URI that we sends push notifications to. When message is pushed to the URI, the phone will receive a push notification. As we only want to push notifications to subscribed user who installed our application, this function also restricts the subscribed phones to receive the notifications. Every phone-URI pair will be saved in a dictionary that is stored in our azure database. The implementation of SubscribeMyPhone function is shown below:

```csharp
public void SubscribeMyPhone(int phoneID, string channelURI)
{
    Uri thisURI = new Uri(channelURI);
    if (SubscriptionURIs.ContainsKey(phoneID))
    {
        SubscriptionURIs[phoneID] = thisURI;
    }
    else
    {
        SubscriptionURIs.Add(phoneID, thisURI);
    }

    // Then need to save in a database, otherwise it will lost when restart app
    string conn_str = ConfigurationManager.ConnectionStrings["NHREConnectionString"].ConnectionString;
    SqlConnection conn = new SqlConnection(conn_str);
    conn.Open();

    string sqluse = "UPDATE user_table SET channel = " + thisURI + " WHERE (id = " + phoneID + ")";
    SqlCommand cadchann = new SqlCommand(sqluse, conn);
    int row = cadchann.ExecuteNonQuery();

    conn.Close();
}
```

*Figure 16 - Method for subscribing the phone*
After subscribing the phone, other users can now send push notifications to the user. The cloud server will use XML to construct the push notifications, which contain the title and the message field. Below is an example of code creating the XML message:

```
MemoryStream myStream = new MemoryStream();
XmlWriter myWriter = XmlWriter.Create(myStream);
myWriter.WriteStartDocument();
myWriter.WriteStartElement("mp", "Notification", "mpNotification");
myWriter.WriteStartElement("wp", "Toast", "wpNotification");
myWriter.WriteStartElement("wp", "Text1", "wpNotification");
myWriter.WriteValue(ToastTitle);
myWriter.WriteEndElement();
myWriter.WriteStartElement("wp", "Text2", "wpNotification");
myWriter.WriteValue(ToastMessage);
myWriter.WriteEndElement();
myWriter.WriteEndElement();
```

After building the xml object, we create a HTTP Web Request instance with HTTP Request Method “Post”, content type “text/xml” and a header containing unique notification ID. After that, the header is merged with the push payload and sends the notification to the user communication channel (user’s URI) that created at the very beginning. The figure below shows the construction of HTTP request.

```
var myRequest = (HttpWebRequest)WebRequest.Create(receiveURI); //Push Client's channelURI
myRequest.ContentType = "text/xml";
myRequest.ContentLength = pushPayloadLength;

//Customize or exclude the X-WindowPhone-Target header based on noti
switch (notificationType)
{
    case "toast":
        break;
    case "tile":
        break;
    case "raw":
        break;
}

//Merge headers with payload
using (var requestStream = myRequest.GetRequestStream())
{
    requestStream.Write(pushPayload, 0, pushPayloadLength);
}
//Send notification to this phone
var response = (HttpWebResponse)myRequest.GetResponse();
```

![Figure 17 - Construction of push notification message](image1)

![Figure 18 - Construction of HTTP request for push notification](image2)
Nevertheless, as the Windows Phone 7 operating system do not support the function of saving received push notifications, the notifications will be disappeared after users click on the messages. In order to make the application user friendly, we had made a notification history page in our application to solve the aforementioned problem. We save the push notifications sent in our database so that the user can check the messages sent to them.

2. Windows Phone Application

A. User Interface (Handled by Ray)

The phone application mainly contains six pages. They are the main page, camera page, map page, photo album page, photo page and notification page.

The figure below shows the main page of our application:

![Figure 19 - Design of main page](image)
In the main page, users can choose the corresponding icons in order to launch camera page, map page, album page and notification page.

The figure below shows the design of the camera page:

Once the user enters the camera page, the application will launch the camera. This is to speed up the process of photo capturing. In the camera page, users can choose to recapture the photo, categorize the photo with tags and add comments. The users can choose to upload their photo in public or private. Uploading in public means the original photo is open to all users, while for private uploading, the other users have to buy the photo in order to obtain the full resolution picture.
The figure below shows the design of the map page and photo album page:

For the map page, the photos in the database will be shown as pins on the map. When the user clicks on the map, the page will be directed to a specific page of the photo. The map will be centered to the current location of the user, so as to assist the user finding what is happening around them. The user can also adjust the zoom level of the map to search in a wider range.

Similar to the map page, the album page shows the photo taken by other users. However, the photos are displayed as lists in chronological order. This is to assist users to find out the latest events regardless of the location. When the users click on the photos, the page will be directed to the specific page of the photo. The users are able to filter the photos according to tags in this page.

More detailed functions and construction of the map page and photo album page will be described later in the report, handled by Eric.
The figure below shows the design of the photo page:

![Design of photo page](image)

The photo page will show the thumbnail of the photo selected. There are text fields showing the tag and comments of photo, the number of like, dislike and view of the photo. There is also a text box for photo buyers to enter the price they want to pay for the photo. The “buy”, “accept”, “deny” and “DL (Download) button will be enabled in different stages of photo purchase, which will be described later in the report, handled by Harry.

The figure below shows the design of the notification page:

![Design of notification page](image)
The notification page stores will retrieve all the push notifications sent to the user, sorted in chronological order. When the user clicks the message, the application will jump to the corresponding photo that is related to the message. Since the Windows Phone 7 OS does not support saving the push notifications, the notifications page will greatly enhance the user experience by reminding the user about notifications and direct user to the related pages.

**B. Application login and access control (Handled by May)**

In order to enhance the user control on the app, users have to use their Windows Live ID to login before using the app. The user is required to login to the Live ID account in Microsoft server. After the user logged in, the app will give return a token to the application as the identification of the user. We cannot recover the Live ID of the user from the token so that the confidentiality of the log in process is kept. After receiving the token, the application will check the user records on the database to see whether the user has logged in before. If it is a first time user, new user record will be added to the database.

If the user is not a new user, the device id of the phone will be recorded in the database. Then the app will further check whether the user has logged in with the same Live ID before. If the user is not using the same Live ID as before on the device, the application will prompt the user to use the same Live ID. This is to ensure that other user cannot obtain the private key stored inside the application. If the user used the same Live ID to login before, the user can get into the app successfully and a new user records will be added to the database.

![Image of login page](image1.png)

**Figure 25 - Page for Live Account login**

![Image of access control violation](image2.png)

**Figure 26 - Prompt for access control violation**
The getid method that returns the token to the application on SQLservice.svc is implemented as below:

```csharp
public string getid(String deviceid) {
    string conn_str = ConfigurationManager.ConnectionStrings["MYSQLConnectionString"].ConnectionString;
    SqlConnection conn = new SqlConnection(conn_str);
    conn.Open();

    string sqlview = "SELECT liveID FROM user_table WHERE name = "+ deviceid + ";"
    SqlCommand cmdview = new SqlCommand(sqlview, conn);
    SqlDataReader reader = cmdview.ExecuteReader();
    bool getliveid = false;
    string liveid = "null";
    while ((reader.Read()) && (!getliveid))
    {
        liveid = reader[0].ToString();
        getliveid = true;
    }
    conn.Close();
    return liveid;
}
```

Figure 27 - Method for getting the token from Windows Live server

On the phone, an event handler SQLhelper_getidCompleted will check whether the Live ID get from function getid is the same as current Live ID or not.

```csharp
private void SQLhelper_getidCompleted(object sender, SQLservice.getidCompletedEventArgs e)
{
    var simplewebtokenStore = Application.Current.Resources["swtStore"] as SimpleWebTokenStore;
    string userNameIdentifier = simplewebtokenStore.SimpleWebToken.NameIdentifier;
    if ((e.Result.ToString() == userNameIdentifier) || (e.Result.ToString() == "null"))
    {
        returnmain();
    }
    else
    {
        MessageBox.Show("Please use the same LiveID as before");
    }
}
```

Figure 28 - Method to compare Live ID token

A new user record should be created while every time the user reinstalls the app because the private key of the user will be deleted when the user uninstall the app. A new set of public key and private key should be generated and a new user record should be created.

However, the old user record is not deleted as that might be used as part of the ring signature in other photo. Deleting the old user record will cause the ring signature can’t function correctly.
C. Session Key generation (Handled by May)

To enhance the security level on the app, we implemented the session key system in the app. For every time the user opens the app, a session key will be generated on the database and saved on the “passtable” on the database. After verifying the identity of the user, the app will get a valid session key on the passtable by calling the function “getpass” on SQLAzure service. Then for every request from the app to the services, a valid session key will be required for the purpose of ensuring the request is from a trusted authority.

```csharp
public string getpass(string liveid)
{
    string conn_str = ConfigurationManager.ConnectionStrings["WIREConnectionString"].ConnectionString;
    SqlConnection conn = new SqlConnection(conn_str);
    conn.Open();
    //Select the records for specific user
    string sqlview = "Select * from user_table WHERE liveid = " + liveid + ""
    SqlCommand cmdview = new SqlCommand(sqlview, conn);
    SqlDataReader reader = cmdview.ExecuteReader();
    bool hassid = reader.HasRows;
    if (hassid)
    {
        //Select a valid sessionkey
        string sqlpass = "Select sessionkey from passtable WHERE (used = 0)"
        SqlCommand cmdpass = new SqlCommand(sqlpass, conn);
        SqlDataReader reader2 = cmdpass.ExecuteReader();
        bool gotpass = false;
        string s = "";
        while ((reader2.Read()) && (!gotpass))
        {
            s = reader2[0].ToString();
            gotpass = true;
        }
        string sqluse = "UPDATE passtable SET used = 1 WHERE (sessionkey = " + s + ""
        SqlCommand cmduse = new SqlCommand(sqluse, conn);
        int row_count = cmduse.ExecuteNonQuery();
        string sqladd = "INSERT into passtable (used, sessionkey) values (0, newid())"
        SqlCommand cmdadd = new SqlCommand(sqladd, conn);
        int run = cmdadd.ExecuteNonQuery();
        conn.Close();
        return s;
    }
}
```

**Figure 29 - Method for retrieving session key**

In our implementation, the session key is a uniqueidentifier. In further stages of development, the session key can be generated as other more secure crypto algorithm.
D. Watermark and thumbnail generation (Handled by Eric)

When the user captures a photo, a thumbnail of the photo will be generated, masked by a colourful watermark. The thumbnail is of low resolution and quality, and is used as a preview of the original photo. In the photo-selling scheme, the watermarked thumbnail is an important piece of the solution. This is because the watermark can prevent malicious users use the resized photo on unauthorized purposes. Users have to buy the photo in order to remove the watermark and obtain the authorization of the seller to use on publications. In the design of the watermark, the text of the watermark is designed with changing colours. This is to make the watermark visible on any background colour.

The watermark function is called just after a photo is captured. The function will render a text block with specified font size, font family, and font colour to the photo before uploading.

```
private void RenderString(Bitmap bgm, string strRender)
{
    TextBlock textBlock = new TextBlock();
    textBlock.Text = strRender;
    // colored the watermark text
    LinearGradientBrush brush = new LinearGradientBrush();
    GradientStop a = new GradientStop();
    a.Color = Colors.Red;
    a.Offset = 0.0f;
    GradientStop b = new GradientStop();
    b.Color = Colors.Yellow;
    b.Offset = 0.2f;
    GradientStop c = new GradientStop();
    c.Color = Colors.Blue;
    c.Offset = 0.5f;
    GradientStop d = new GradientStop();
    d.Color = Colors.Green;
    d.Offset = 0.75f;
    brush.StartPoint = new Point(0, 0);
    brush.EndPoint = new Point(8, 3);
    brush.GradientStops.Add(a);
    brush.GradientStops.Add(b);
    brush.GradientStops.Add(c);
    brush.GradientStops.Add(d);
    textBlock.Background = brush;
    // set font, size, etc. on TextBlock
    textBlock.Margin = new Thickness(50, 50, 50, 50);
    textBlock.Height = 100;
    textBlock.Width = 100;
    textBlock.FontStyle = FontStyles.Bold;
    textBlock.Text = strRender;
    // render textBlock
    bitmap = new Bitmap(textBlock, null);
    bitmap.Invalidate();
    // render background
    background = new DrawImage();
    background.Opacity = 0.5f;
    background.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
    textBlock.Invalidate();
}
```

Figure 31 - Method for rendering watermark
E. Photo Upload Service, UploadHelper.cs (Handled by Harry)

To allow the smartphone application to call the WCF services, we have built a class, UploadHelper.cs, to handle the upload of photo. The class handles the creation of blob on Azure Blob Storage, initializes the storage using the WCF service, and uses the returned URI to upload files.

```csharp
public void uploadObject()
{
    var client = new UploadServiceClient();
    client.UploadUrlWithSharedAccessSignatureCompleted += uploadUrlWithSharedAccessSignatureCompleted;
    client.UploadUrlWithSharedAccessSignatureAsync(new Uri().ToString());
}

private void uploadUrlWithSharedAccessSignatureCompleted(object sender,
                                                      UploadUrlWithSharedAccessSignatureCompletedEventArgs e)
{
    if (e.Error != null)
        MessageBox.Show(e.Error.Message.ToString());
    if (e.Error == null)
    {
        // Get the path to the blob.
        var builder = new StringBuilder(e.Result);
        builder.Path = builder.Path + root + filename;
        // Open the image file from the file system.
        var bmmFile = new BmmFile()
        var file = BmmFile.Open?
        // Create the uploader and kick off the uploader.
        var uploader = new CloudBlobUploader(file.url.AbsoluteUri);
        var pos = uploader.UploadPos;
        fileurl = uploader.UploadPos?
        // Delete the file from the file system.
        fileurl = fileurl.DeleteFile(filename);
        uploadCompleted(this, e);
    }
    uploader.StartUpload();
}
```

Figure 32 - Methods for uploading photos

The complication of such a method is that it is not like uploading files to ordinary file servers. We have to first create a blob on the storage, generate an URI, request for permission to upload, and then actually uploading to the storage.

We can also see in the code that the read and write of file in Windows Phone applications are not the same as other phones. We have to use an Isolated Storage to store the photo taken, and then call the method to upload the photo to the storage. We are taking this approach as this makes the method and WCF service highly reusable. Next time when we need to upload files to another Azure Blob Storage, we just need to change the connection string. This will be beneficial to the further development of our projects as well as others trying to use Azure cloud servers.
F. Building the crypto class (Handled by Harry)

In order to support the encryption and decryption of photos using AES, as well as the encryption and decryption of AES keys using RSA, we have created a crypto helper class to serve the purpose. The crypto class also provides ring signature and verification services. In here, we break down the class into 3 parts:

i. RSA security service

At first we need to generate RSA key pairs. We cannot find keys issued by trusted authorities so the generated keys are only serving for testing purposes. The keys are generated with 1024 bit strength. The key pairs generated are compatible to be used in .NET 4.0 Framework RSACryptoServiceProvider. In fact, real RSA keys can be easily imported to the library as it follows the same set of standards.

```csharp
public CryptoHelper(int mode) {
    RSA = new RSACryptoServiceProvider(1024);
    RSA.KeySize = 1024;
    RSA.KeyUsage = KeyUsageFlags.Signature | KeyUsageFlags.EncryptData;
    // Use default values
    if (mode == 0) {
        // Attach the KeysGenerated event handler
        RSA.KeyExchange = true;
        RSA.KeysEncrypted = true;
        RSA.KeySize = 1024;
    }
    // Generate key pair
    RSA.GenerateKeyPair();
}
```

Figure 33 - Generate RSA key pairs

The keys are generated at the first launch of the application. As long as the application is not uninstalled, the key will be stored in isolated storage inside the application. The key cannot be regenerated so each installation is stick to one user account.
For RSA encryption and decryption, we use the following methods:

```java
public byte[] RSACrypt(string obj)
{
    // Convert your text to a byte array
    byte[] rawBytes = System.Text.Encoding.UTF8.GetBytes(obj);
    // Encrypt your raw bytes and return the encrypted bytes
    byte[] encBytes = dRSA.Encrypt(rawBytes);
    return encBytes;
}

public string RSAEncrypt(byte[] encBytes)
{
    // Now decrypt your encrypted bytes
    byte[] decBytes = dRSA.Decrypt(encBytes);
    // Convert your decrypted bytes back to a string
    return System.Text.Encoding.UTF8.GetString(decBytes, 0, decBytes.Length);
}
```

**Figure 34 - Methods for RSA encryption and decryption**

The RSA encrypt method receives a Base64 string and encrypt it into byte array with imported public key, and the RSA decrypt method decrypts the byte array into Base64 string with the imported private key.

**ii. AES security service**

After initializing the crypto class, we can encrypt the photo using AES. The key is generated with HMACSHA1 pseudo-random number generator and the iteration count is 5000. The method accepts the byte array containing the data to be encrypted and return the string that is the cipher of the AES encryption process.

```csharp
public string AESCrypt(byte[] dataToEncrypt)
{
    AesManaged aes = null;
    MemoryStream memoryStream = null;
    CryptoStream cryptoStream = null;
    try
    {
        // Generate a key based on a Password and HMACSHA1 pseudo-random number generator
        byte[] AESPassword = Encoding.UTF8.GetBytes(AESPassword); // 5000 times
        byte[] aesKey = new HMACSHA1(AESPassword, 32);
        // IV derived from byte array with 16 pseudo-random key bytes
        byte[] aes.IV = new byte[16];
        // Create AES algorithm
        aes = new AesManaged();
        // A new encryption and decryption key with 32 pseudo-random key bytes
        aes.Key = aesKey;
        // IV derived from byte array with 16 pseudo-random key bytes
        aes.IV = aes.IV;
        // Create Memory and Crypto Streams
        memoryStream = new MemoryStream();
        CryptoStream cryptoStream = new CryptoStream(memoryStream, aes.CreateEncryptor(), CryptoStreamMode.Write);
        // Encrypt Data
        cryptoStream.Write(data, 0, data.Length);
        cryptoStream.FlushFinalBlock();
        // Return Base64 String
        return Convert.ToBase64String(memoryStream.ToArray());
    }
    finally
    {
        // Close all streams
        if (cryptoStream != null) cryptoStream.Close();
        if (memoryStream != null) memoryStream.Close();
        if (aes != null) aes.Clear();
    }
}
```

**Figure 35 - Method for encryption using RSA**
The method for decrypting data using AES is similar to the AES encryption method, just that the method is doing the reverse, receiving the cipher as a Base64 string and return a byte array as decrypted data.

```
public byte[] AESdecrypt(string dataToDecrypt)
{
    AESManaged aes = null;
    MemoryStream memoryStream = null;

    try
    {
        //Generate a Key based on a Password and HmacSha1 pseudo-random number generator
        //Size must be at least 8 bytes long
        //Use an iteration count of at least 10000
        byte[] aesKey = rfc2898DeriveBytes(aesPassword, Encoding.UTF8.GetBytes(aesPassword), 5000);
        //Create AES algorithm
        aes = new AESManaged();
        //Key derived from byte array with 32 pseudo-random key bytes
        aes.Key = aesKey;
        //IV derived from byte array with 16 pseudo-random key bytes
        aes.IV = rfc2898DeriveBytes(aesKey, 16);
        //Create Memory and Crypto Streams
        memoryStream = new MemoryStream();
        CryptoStream cryptoStream = new CryptoStream(memoryStream, aes.CreateDecryptor(), CryptoStreamMode.Write);
        //Decrypt Data
        byte[] data = Convert.FromBase64String(dataToDecrypt);
        cryptoStream.Write(data, 0, data.Length);
        cryptoStream.FlushFinalBlock();
        //Return Decrypted String
        byte[] decryptedBytes = memoryStream.ToArray();
        //Dispose
        if (cryptoStream != null)
            cryptoStream.Dispose();
        //Return
        return decryptedBytes;
    }
    finally
    {
        if (memoryStream != null)
            memoryStream.Dispose();
        if (aes != null)
            aes.Clear();
    }
}
```

*Figure 36 - Method for decryption using AES*
G. Ring Signature Scheme (Handled by Harry)

In order to support anonymous upload, we need to apply a ring signature scheme. There are currently no libraries that support ring signature in C# or Silverlight. Therefore, we have to implement the ring signature schemes from mathematical formulas.

In the project, we are implementing a ring signature scheme based on Schnorr’s signature. The mathematical formulas are as follow:

For signing message M,

1. Choose random number r
2. Let \( R = g^k \)
3. Let \( h = H(M \parallel r) \)
4. Let \( z = (k - xe) \)

The signature pair is \((z, h)\), while \( x \) is the private key of the user, \( \parallel \) denotes concatenation and \( H() \) denotes hash function.

For verifying message M with signature pair \((z, h)\),

1. \( z_v = g^{s_y e} \)
2. \( h_v = H(M \parallel r_v) \)

If \( h_v = h \), the signature is verified.

To extend the Schnorr’s signature to a ring signature scheme, we need to perform significant modification.

Suppose we are signing a message M with user key pair \((Y_1, x_1)\),

1. Choose two public key \( Y_2 \) and \( Y_3 \) from a pool of users
2. Generate fake \((z_2, h_2)\) and \((z_3, h_3)\) signature pairs
3. \( R_2 = g^{z_2} Y_2^{h_2} \), \( R_3 = g^{z_3} Y_3^{h_3} \)
4. Generate random number r
5. \( R_1 = g^r \)
6. \( h = H(M \parallel R_1 \parallel R_2 \parallel R_3 \parallel Y_1 \parallel Y_2 \parallel Y_3) \)
7. \( h_1 = (h + h_1 + h_2) \)
8. \( z_1 = r - h_1 x_1 \)
The ring signature would then be \((z_1, h_1, z_2, h_2, z_3, h_3)\). The ring-signing scheme is implemented as follow:

```java
public BigInteger[] ringSign(String m, BigInteger v2, BigInteger v3) {
    // fake signature (z2, h2)
    BigInteger z2 = new BigInteger();
    z2 = genRandomBits(1024, ran);
    BigInteger h2 = new BigInteger();
    h2 = genRandomBits(1024, ran);
    BigInteger r3 = (g.modPow(z2, p) * v2.modPow(h2, p)) % p;

    // fake signature (z3, h3)
    BigInteger z3 = new BigInteger();
    z3 = genRandomBits(1024, ran);
    BigInteger h3 = new BigInteger();
    h3 = genRandomBits(1024, ran);
    BigInteger r3 = (g.modPow(z3, p) * v3.modPow(h3, p)) % p;

    // generate real signature
    BigInteger e = new BigInteger();
    r = genRandomBits(1024, ran);
    BigInteger r1 = g.modPow(r, p);

    byte[] m = encoding.GetBytes(m);
    byte[] k1 = h1.getKs();
    byte[] k2 = h2.getKs();
    byte[] k3 = h3.getKs();
    byte[] y2 = y1.getKs();
    byte[] y3 = y2.getKs();

    byte[] concat = new byte[m.Length + m2.Length + m3.Length + m4.Length + m5.Length + m6.Length + m7.Length];
    System.Buffer.BlockCopy(m, 0, concat, 0, m.Length);
    System.Buffer.BlockCopy(m2, 0, concat, m.Length, m2.Length);
    System.Buffer.BlockCopy(m3, 0, concat, m.Length + m2.Length, m3.Length);
    System.Buffer.BlockCopy(m4, 0, concat, m.Length + m2.Length + m3.Length, m4.Length);
    System.Buffer.BlockCopy(m5, 0, concat, m.Length + m2.Length + m3.Length + m4.Length, m5.Length);
    System.Buffer.BlockCopy(m6, 0, concat, m.Length + m2.Length + m3.Length + m4.Length + m5.Length, m6.Length);
    System.Buffer.BlockCopy(m7, 0, concat, m.Length + m2.Length + m3.Length + m4.Length + m5.Length + m6.Length, m7.Length);

    byte[] h = MD5Core.GetHash(concat);
    BigInteger h1 = new BigInteger(h);

    BigInteger h1 = (h = h2 + h3) % h;
    BigInteger r = r1 % h1;

    BigInteger[] signature = new BigInteger[3, 2];
    for (int i = 0; i < 3; i++)
    {
        signature[i, 1] = new BigInteger();
    }

    signature[0, 0] = z1;
    signature[1, 0] = h1;
    signature[2, 0] = z2;
    signature[2, 1] = h2;
    signature[3, 1] = h3;

    return signature;
}
```

Figure 37 - Method for ring signature

There are minor modifications to meet computing requirements, but the implementation is closely following the mathematical formulas.
To verify message $M$ with signature $(z_1, h_1, z_2, h_2, z_3, h_3)$,

i) $R'_{1} = g^{z_1}Y_{1}^{h_1}$

ii) $R'_{2} = g^{z_2}Y_{2}^{h_2}$

iii) $R'_{3} = g^{z_3}Y_{3}^{h_3}$

iv) $h' = H(M \parallel R'_{1} \parallel R'_{2} \parallel R'_{3} \parallel Y_{1} \parallel Y_{2} \parallel Y_{3})$

If ($h' + h_2 + h_3 = h_1$) then the signature is verified. The ring verification is implemented as follow:

```
public bool ringverify(string m, BigInteger[] signature, BigInteger v1, BigInteger v2, BigInteger v3)
{
    System.Text.UTF8Encoding encoding = new System.Text.UTF8Encoding();

    BigInteger z1 = signature[0, 0];
    BigInteger h1 = signature[0, 1];
    BigInteger z2 = signature[1, 0];
    BigInteger h2 = signature[1, 1];
    BigInteger z3 = signature[2, 0];
    BigInteger h3 = signature[2, 1];

    BigInteger R1 = (g.modPow(z1, p) * v1.modPow(h1, p)) % p;
    BigInteger R2 = (g.modPow(z2, p) * v2.modPow(h2, p)) % p;
    BigInteger R3 = (g.modPow(z3, p) * v3.modPow(h3, p)) % p;

    byte[] ma = encoding.GetBytes(m);
    byte[] R1a = R1.toByteArray();
    byte[] R2a = R2.toByteArray();
    byte[] R3a = R3.toByteArray();
    byte[] V1a = v1.toByteArray();
    byte[] V2a = v2.toByteArray();
    byte[] V3a = v3.toByteArray();

    System.Buffer.BlockCopy(ma, 0, concat2, 0, ma.Length);
    System.Buffer.BlockCopy(R1a, 0, concat2, ma.Length, R1a.Length);
    System.Buffer.BlockCopy(R2a, 0, concat2, ma.Length + R1a.Length, R2a.Length);
    System.Buffer.BlockCopy(R3a, 0, concat2, ma.Length + R1a.Length + R2a.Length, R3a.Length);
    System.Buffer.BlockCopy(V1a, 0, concat2, ma.Length + R1a.Length + R2a.Length + R3a.Length, V1a.Length);
    System.Buffer.BlockCopy(V2a, 0, concat2, ma.Length + R1a.Length + R2a.Length + R3a.Length + V1a.Length, V2a.Length);
    System.Buffer.BlockCopy(V3a, 0, concat2, ma.Length + R1a.Length + R2a.Length + R3a.Length + V1a.Length + V2a.Length, V3a.Length);

    byte[] h_a = MD5Core.GetHash(concat2);
    BigInteger h_a = new BigInteger(h_a);
    if ((h_a + h2 + h3) % p == h1)
    { return true; }
    else
    { return false; }
}
```

Figure 38 - Method for ring signature verification
H. Photo Selling Scheme (Handled by Harry)

In the design of photo selling scheme, we integrate bits and pieces developed in our project to create a scheme to sell photos securely and efficiently. The steps of the photo selling process are described below:

Step 1: When the user encounters an event, he can take a photo of it. (Camera capture is handled by May) At the same time, a thumbnail of the photo with a watermark is created. (Eric handles watermark and thumbnail creation, mentioned earlier in the report)

Step 2: The user can add comments on the photo as well as categorize them with tags. (Ray handles comments and tags, mentioned earlier in the report) The tags will enable photo buyers to find the photos they are interested quickly.

Step 3: User clicks the “upload” button. The photo is encrypted by AES and uploaded to Azure Blob Storage. The thumbnail is also uploaded but without encryption, for other users to preview. The original photo is not stored in the phone so that the user cannot sell twice.

Figure 39 - AES encryption of photo
Step 4: When another user finds the photo and wants to buy it, he can enter the price he wants to pay in photo page and click the “buy” button. A push notification with price information will be sent to the seller of the photo.

```
private void buyButton_Click(object sender, RoutedEventArgs e)
{
    if (priceBox.Text != null)
    {
        int price = Convert.ToInt32(priceBox.Text);
        if (price > 0)
        {
            buyButton.IsEnabled = false;
            priceBox.IsEnabled = false;
            SQLhelper.buyPhotoCompleted += new EventHandlers_SQLservice.buyPhotoCompletedEventHandler(buyPhotoCompleted);
            SQLhelper.buyPhotosync(userID, photoID, Convert.ToInt32(priceBox.Text));
        }
        else
        {
            MessageBox.Show("Please enter a price");
            priceBox.IsEnabled = true;
        }
    }
}
```

Set offer price

Send push notification to seller

Figure 40 - Methods for photo purchase
Step 5: The seller accepts the deal. The system will obtain the public key of the buyer from the database. The AES key is encrypted by the public key of buyer using RSA and sent to the database. A push notification will be sent to the photo buyer to notify the accepted offer.

Step 6: The buyer encrypts e-cash and sent to the seller. (E-cash is not implemented in the project) At the same time, the seller takes the encrypted AES key from database and decrypts it using his private key.
Step 7: The buyer obtained the AES key and decrypts the photo. The buyer will verify the photo with the hash value of the original photo. The photo will be saved to the picture library of the phone.

![Check the MD5 hash value of original photo and downloaded one](image1)

![Save photo to picture library](image2)

Figure 43 - Method to verify the photo

Figure 44 - Method to save downloaded photo to phone library

to receive an award after the photo is sold. In the buyers’ point of view, they can buy photos without caring about contacting the sellers, and can easily find the photos they need.
I. Map Page (Handled by Eric)

When the user enters the map page, a list of photos captured around the user will be downloaded. They will show as pushpins on the map. They map will then center at the currently location of the user in order to display events happening around. The user can adjust the zoom level of the map by clicking the “+” and “-” sign or using multitouch gestures. With this function, users can search for events in a wider range.

![Screenshot of the map page](image)

The original design for the map page will show the thumbnails of photos on the map. However, the screen will become overcrowded when more and more photos are displayed in the map. This is not convenient for the user to navigate through. Therefore, we changed all thumbnails on the map to pushpins. The pushpins are in red in order to distinguish from the background.

![Change of map design](image)
When a pushpin is clicked, the photo page of the corresponding photo will be shown. Users can view the information about the photo. On the photo page, the user can see the thumbnail, comment, number of like, number of dislike and number of view of the photo. The user can also rate the photo and offer a price to purchase the photo from the seller. The seller can accept or deny the offer here. Moreover, once the user bought the photo, he or she can download the photo in here.

![Figure 47 - Code for loading the information of photos](image)

![Figure 48 - Screenshot if photo page before purchase](image)

![Figure 49 - Screenshot of photo page with photo bought](image)

When the photo page is loaded, the view number of corresponding photo will be increased automatically. As aforementioned, the user can rate the photo on this page by clicking “like” or “dislike”. This rating system can allow user to express their feelings of the photo in a fundamental but convenient way, and also facilitate the photo categorizing function on album page.
J. Album Page (Handled by Eric)

When the album page (bottom-left button on main page) is clicked, the album page with a list of photos and their id will be shown. Album page is implemented in panorama, which allows user to swipe the page to view photos listed in different order. In our application, the album page allows users to view photo listed in the order of date, number of likes, number of dislikes, and number of views.

![Figure 50 - Album page sorted according to date](image)

![Figure 51 - Album page sorted according to like numbers](image)

![Figure 52 - Album page sorted according to dislike numbers](image)

![Figure 53 - Album page sorted according to view numbers](image)

Moreover, the album page has an application bar to allow users viewing photos listed in different categories, namely headline, local, sport, and entertainment. The application bar also enables multiple selections on category, which means users can choose to view photo listed in two or more categories. Both panorama page and application bar can work together for showing photos in different criteria. For example, when a user goes to the “Top Like” page and tick both headline and local in the application bar. Those photos in headline and local category will be shown in descending order of the number of likes.

![Figure 54 - Album page with application bar](image)

Similar to the map page, when a user clicks on a photo listed in the album page, the photo id and the session key will be sent to the photo page for database query. After getting the information of the photo from database, the information is shown on the photo page.
3. Web portal

The web portal is an alternative way for users (even for non-Windows Phone users) to access our platform and browse the photos by date, by categories and by locations. The web portal is implemented in ASP.Net. The web portal also leverages other services such as Access Control on Azure, SQL Azure, Azure Blob and Bing Map.

![Figure 55 - Outlook of web portal](image)

A. Access Control (Handled by May)

Access Control is one of the features provided by Windows Azure platform. It allows user to log in the system by different accounts (e.g. Windows Live ID, Yahoo, Gmail etc.) which most of the users already have created at least one account. We can make the login process more user-friendly as users do not need to create another account and remember the other set of password.

After the users typed in the log in details, which is done on the page of the service provider, the Access Control will validate the user and issue a secure token to the application. In the process for the users to enter the log in details, the application will not and cannot access any information about the log in details. As a result, the confidentiality of the log in details can be kept.

Also, by using the Access Control, the web portal doesn’t need to implement a log in system and keep the confidential data in our system. Thus, this can enhance our security of the system.
The figure below shows an overview of the flow in access control:

1. User goes to the URL of the web portal and request to log in
2. The web portal will check whether the user have a secure token or not. If not, it will request one from the Access Control
3. The Access Control will direct the user to the login page of the service provider (Windows Live Account is used in the system in our project) and ask the user to log in.
4. The Access Control will validate the user
5. If the user is valid, the access control system will return a secure token to the web portal.
6. The web portal will check whether the token is valid or not. If it is valid, the web portal will direct the user to the home page of the web portal.

Figure 56 - Flow of access control
After adding the access control to the web portal, we use the following method to get the carrier information and the user ID to identify the user:

```csharp
protected void Page_Load(object sender, EventArgs e)
{
    var identity = User.Identity;

    // get provider
    provider = string.Empty;

    var claimsIdentity = Identity as ClaimsIdentity;
    if (claimsIdentity == null)
        provider = string.Empty;

    var providerQuery = from c in claimsIdentity.Claims
                        where c.ClaimType ==
                        "http://schemas.microsoft.com/accesscontrolservice/2010/07/claims/identityprovider"
                        select c.Value;
    provider = providerQuery.FirstOrDefault();

    // get nameId
    nameId = string.Empty;

    claimsIdentity = identity as ClaimsIdentity;
    if (claimsIdentity == null)
        nameId = string.Empty;

    providerQuery = from c in claimsIdentity.Claims
                    where c.ClaimType == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
                    select c.Value;
    nameId = providerQuery.FirstOrDefault();

    adduser();
}
```

“nameId” (i.e. the token) is being used for identification. After getting the nameId, the portal will further check if it exists in database or not. If it exists, that means the user has registered his account by logging our app using a Windows Phone. In this case, the user can enjoy all the function of the portal. However, if the nameId is not existed in the database, the user can only have limited functions (e.g. can’t buy photos) on the portal.

**B. Home Page (Handled by May)**

In the Home page, users can browse the public photos and give rating to each of it. By clicking on the photo, users can enlarge the photo and the portal will automatically add the number of view of that photo. Users can also like or dislike on particular photos.
After defining the connection string, we can use a List View to show the photos and use it in Configuration Manager to access the database.

To implement the “Like” and “Dislike” functions, we implemented a ListView OnItemClickCommand and execute the following SQL command for each function.

For adding number of like:
"UPDATE tags SET likenum = likenum + 1 WHERE photoid = (SELECT id FROM photo WHERE path = '" + photopath + '");
For adding number of dislike:
"UPDATE tags SET dislikenum = dislikenum + 1 WHERE photoid = (SELECT id FROM photo WHERE path = '" + photopath + '");

Figure 58 - Queries to “like” or “dislike” photos

For filtering the photos by date or by categories, I added selectionchanged event handler on the two calendars SelectIndexChanged function to the ListBox of categories. Whenever the users change the preference by clicking on the calendars or on the ListBox, it will trigger the event handlers and change the condition of the SQL for selecting photos. Then the portal will be reloaded and display the photos under the new condition.

C. Private Page (Handled by May)

Similar to the Home Page, users can browse all the private photos in Private Page and enjoy all the functions in Home Page (e.g. filtering by date or categories, giving rating to the photos and view the photos).

Private photos are the photos for sale. The main difference between the Home Page and the Private is that users can buy the photos on the Private Page. By entering a desired amount of money in the textbox below each photos and click buy button, users can bid the photo. If the photo is already sold, the user will not be able to buy it.
For every successful bid, the following SQL command will be executed and the transaction record will be added to the buy table in database:

```csharp
string sqltran = "SELECT * FROM WHERE photoid = " + photoid;
SqlCommand cmdtran = new SqlCommand(sqltran, conn);
SqlDataReader readertran = cmdtran.ExecuteReader();
bool seen = readertran.HasRows;
if (seen)
{
    LabelBuy.Text = "Photo id: " + photoid + " is sold already";
}
else
{
    TextBox txtprice = (TextBox)e.Item.FindControl("TextBoxprice");
    int price = Int32.Parse(txtprice.Text.ToString());

    string sqlbuy = "INSERT INTO buy (buyerid, photoid, tranid, time, price, status) VALUES (" + buyerid + ", (SELECT id FROM photo WHERE thumbpath = " + photopath + ") , newid(), CURRENT_TIMESTAMP, " + price + ", 0)";
    SqlCommand cmdbuy = new SqlCommand(sqlbuy, conn);
    int row_count = cmdbuy.ExecuteNonQuery();

    string message = "Photoid: " + photoid + " is sold with the price " + price;
    myClient.PushToast("Sold", message, uploaderid);
    LabelBuy.Text = "You have bought Photo " + photoid + " with the price " + price;
}
```

**Figure 60 - Code for adding transaction record to database**

After the bidding is made, the record in the database will be updated accordingly. At the same time, a push notification will be sent to the seller to notice the seller that the photo has an offer.

**Figure 61 - Push notification**
The price and the photoid will be shown in the message by running the following code:

```csharp
string message = "Photoid: " + photoid + " is sold with the price " + price;
myClient.PushToast("Sold", message, uploaderid);
```

**Figure 62 - Code for push notification**

**D. Map Page (Handled by May)**

The web portal also allows users to view photos by locations. It will get the locations of all photos from the records on SQL Azure and get the photo from Azure Blob. Next, it will display those photos on the map by using pushpins on recorded location.

To implement the map page, first, we used the Configuration Manager in the Map.aspx.cs page to retrieve the photos records from SQL Azure and pass them to JavaScript function:
Figure 64 - Passing photo data to JavaScript function

Next, in the JavaScript code, the details of the photos will be saved in arrays in the following format:

```
lats[i] = latitude;
longs[i] = longitude;
paths[i] = URL of the photo;
time[i] = time;
uploaderid[i] = uploaderid;
```

Then in the map.aspx, we used JavaScript to obtain the information of the photos from the arrays and add the pushpin for each photo:

```
function LoadMap() {
    // Define the map
    map = new VEMap('BingMap');
    map.SetDashboardSize(VEDashboardSize.Large);
    map.LoadMap(new VELatLong('22.283868', '114.136587'), 9, VEMapStyle.Road);

    // Add pushpin
    for (var i = 0; i < lats.length; i++) {
        var pin = new VEShape(VEShapeType.Pushpin, new VELatLong(lats[i], longs[i]));
        pin.SetDescription('time[i] + " Uploader is " + uploaderid[i]);
        pin.ShowIcon();
        pin.SetPhotoURL(paths[i]);
        map.AddShape(pin);
    }
    map.ShowDashboard();
}
```

Figure 65 - Adding pushpin to map
Testing and Performance Evaluation

In the performance evaluation plan, we mainly focus on mainly 4 areas:

- Photo Encryption (AES)
- Photo Decryption (AES)
- Photo Signing with Ring Signature
- Ring Signature Verification

In the usage of the application, we found that the above areas are the most time consuming and resource hungry areas. Therefore, through analyzing the performance figures in the above areas, we can find out the bottleneck of the application.

The performance evaluation was carried out on the following 2 phones. Their specifications as shown:

<table>
<thead>
<tr>
<th>HTC 7 Mozart</th>
<th>HTC Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chipset:</strong> Qualcomm QSD8250 Snapdragon</td>
<td>Qualcomm QSD8255 Snapdragon</td>
</tr>
<tr>
<td><strong>CPU:</strong> 1 GHz Scorpion</td>
<td>1 GHz Scorpion</td>
</tr>
<tr>
<td><strong>RAM:</strong> 576 MB</td>
<td>512 MB</td>
</tr>
<tr>
<td><strong>OS:</strong> Windows Phone 7 Build 7740</td>
<td>Windows Phone 7.5 Build 8107</td>
</tr>
</tbody>
</table>

We have tested the phone in the above 4 areas under 4 different resolutions, which are 640x480 (0.3 MP), 1280x960 (1.23 MP), 1600x1200 (1.92 MP) and 2048x1536 (3.14 MP). Each resolution was tested 10 times on each phone, and the average time taken was used to plot the graph to see the performance.
1. Photo Encryption (AES)

Findings:

From the graph, we can see that the photo encryption time increases linearly with the number of pixels. This is a good sign as we can see that there is no bottleneck in encryption process under the tested resolutions. From the figures, we can also see that the newer CPU in Radar has better performance. We can foresee that the time needed for encryption decrease as technology advance.
2. Photo Decryption (AES)

![AES Decryption time Against Resolution](image)

**Findings:**

The AES decryption time increases linearly with the increase in resolution of photo, which is the same as in the findings of AES encryption time.
3. Ring Signing Time

Findings:

From the graph, we can see that the ring signature time mostly remain constant in different resolution. This is expected because the ring signature signs on the 128 bit MD5 hash value of the photo. The figures obtained from performance evaluation confirm the prediction. Moreover, we can see that ring signature is a very slow process on the phone. We believe that the slow performance is due to unoptimized code in the ring signature method. The ring signature scheme is coded directly from mathematical formulas, so hardware accelerations are not supported. We believe that in optimizing the code, the performance of ring signature can be greatly increased.
4. Ring Verification Time

**Findings:**

Similar to signing, the time taken to verify a ring signature is independent to photo resolution. This is because the verification is done on the 128 bit MD5 hash value of photo. In addition, the verification is done in the cloud server, so that it would be less resource-hungry for the phone.
# Project Management

## System Infrastructure and Database Connection

<table>
<thead>
<tr>
<th>Task</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting up Windows Azure</td>
<td>May</td>
</tr>
<tr>
<td>Setting up SQL Azure</td>
<td>May</td>
</tr>
<tr>
<td>Setting up Access Control Service</td>
<td>May</td>
</tr>
<tr>
<td>Connection between Windows Phone to SQL Azure</td>
<td>Harry</td>
</tr>
<tr>
<td>Setting up Image Service</td>
<td>May</td>
</tr>
<tr>
<td>Push Notification</td>
<td>Ray</td>
</tr>
<tr>
<td>Connection between Web Portal to SQL Azure Database</td>
<td>May</td>
</tr>
</tbody>
</table>

## Security

<table>
<thead>
<tr>
<th>Task</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building the crypto class</td>
<td>Harry</td>
</tr>
<tr>
<td>Ring Signature Scheme</td>
<td>Harry</td>
</tr>
<tr>
<td>Session Key</td>
<td>May, Ray</td>
</tr>
<tr>
<td>Water mark</td>
<td>Eric</td>
</tr>
</tbody>
</table>

## Apps

<table>
<thead>
<tr>
<th>Task</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface</td>
<td>Ray</td>
</tr>
<tr>
<td>Application login and Access Control</td>
<td>May</td>
</tr>
<tr>
<td>Photo Upload Service</td>
<td>Harry</td>
</tr>
<tr>
<td>Transaction</td>
<td>Harry</td>
</tr>
<tr>
<td>Map Page</td>
<td>Eric</td>
</tr>
<tr>
<td>Album Page</td>
<td>Eric</td>
</tr>
<tr>
<td>Photos Storing</td>
<td>Harry</td>
</tr>
</tbody>
</table>

## Web Function

<table>
<thead>
<tr>
<th>Task</th>
<th>Responsible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface</td>
<td>May</td>
</tr>
<tr>
<td>Access Control</td>
<td>May</td>
</tr>
<tr>
<td>Album Page</td>
<td>May</td>
</tr>
<tr>
<td>Transaction</td>
<td>May</td>
</tr>
<tr>
<td>Map Page</td>
<td>May</td>
</tr>
</tbody>
</table>
## Project Timeline

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Plan</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Page</td>
<td>All</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Studies on each topic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study on Security Linkage</td>
<td>Harry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study on Windows Phone 7 apps</td>
<td>Eric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study on Web Portal</td>
<td>Ray</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Study on Windows Azure</td>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation on App</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encryption of the Photo</td>
<td>Harry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Map Function</td>
<td>Eric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other functions</td>
<td>Ray</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frame and the linkage to Azure</td>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation on Windows Azure</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Features</td>
<td>Harry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Azure with Bing Map</td>
<td>Eric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Function</td>
<td>Ray</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Control</td>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Implementation on App and Web Portal</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security Features on App</td>
<td>Harry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photo Page on App</td>
<td>Eric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Interface on App</td>
<td>Ray</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Portal</td>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mid-term Review and interim Report</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction</td>
<td>Harry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Water Mark</td>
<td>Eric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Push Notification</td>
<td>Ray</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Portal And Access Control</td>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Testing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Report</td>
<td>Eric and Harry</td>
<td>All</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation and Exhibition</td>
<td>All</td>
<td>All</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Challenges

We encountered several problems during the course of our project. Problems mainly lies on the WCF services, as the system architecture is relatively new to other platforms, there are not much references that we can make use of. Another problem is about the encryption and decryption process. As we have to take care of the performance of the process on slow phones, the encryption and decryption process on phone are a bit troublesome.

1. Problems on WCF services

At the very beginning, we tried to use the WindowsPhone Azure toolkit to handle the upload of photos. However, the complicated toolkit drives us to use many custom built libraries and we have to amend the flow of our application in order to fit the service. Moreover, the usage of toolkit is overly restricted, so we decided to go for a more time consuming approach. After studying multiple references, we created photo-uploading services that have high reusability. This is one of the most challenging parts in our project.

2. Encryption on phone

Modern CPUs on phones provide hardware acceleration on encryption and decryption. However, slow phones still give very poor performance on the process. The initial implementation of the crypto class generates RSA keys in 30 seconds. The application will hang there for a very long time and being extremely unresponsive. The AES encryption will also make random crashes on our older phone. After tweaking the encryption process for several times, we are able to achieve acceptable performance on the slowest phone without compromising strength of security.

We are also encountering bugs in encryption and decryption process on the crypto library we used. The library randomly crashes while decrypting photos sent from another phone. It is hard to find out the problem until we solved it in a random occasion. This is one of the unfavorable results of not using more reliable and proved library such as Bouncy Castle. However, if the crypto library we see more and more bugs in the crypto library we are using, we may consider changing it.
3. Immature SDK

We encountered problems when using the Windows Phone SDK to develop applications. Our applications are developed using Silverlight but there are many references cannot be used in developing .Net application due to immature SDK. For example there no library supports the conversion between the images and bytes, which is used to be found in .NET applications, so we need to implement this function by our own. The implementation really wastes our time. Microsoft should add back some commonly used methods to the library to improve the experience in developing Windows Phone application.

Furthermore, there are issues in debugging and using emulators. The windows phone emulator does not reflect the true performance of the phone. It is running according to the configuration of your computer, which is not reasonable at all. The emulator also does not support multitouch, which is a popular feature on new applications. Debugging the application on Windows Phone is also troublesome, as we cannot open the camera while it is connected to the computer. That means we either take pictures without debugging, or debugging without taking pictures. This is not desirable for applications that have to make use of the camera. We have devoted more time on debugging because of the flaw of the SDK.
Further Development

The system developed in our project has lots of spaces for improvement. First of all, anonymous photo selling scheme can allow users to upload photos anonymously but also preserve the opportunity to make money out of it. Next, map navigation function can help users to navigate to the site of event in our application. Furthermore, an introduction of blacklisting system can block malicious users uploading fake photos or using photos inappropriately. Last but not least, as a spin off of the project, the ring signature scheme can be applied on the application polling systems so that we can verify a valid user but also protect the identity of the user.

1. Anonymous Photo Selling

The current ring signature scheme supports anonymous upload, but only in an unencrypted way. With little modification in the ring signature scheme, a user should be able to sell a ring signed photo on our system.

Here we provide a sketch of flow in the modified scheme. First of all, when the photo is encrypted, an AES key is generated. At the same time, a new RSA key pair is generated and the public key is sent to the transaction record. We do ring signature on the AES key, as opposed to the hash value of the photo in the original scheme. When a buyer buys the photo, the AES key is sent to the buyer using RSA technology. The buyer can verify the ring signature on the AES key obtained. Consequently, the buyer upload the e-cash encrypted with the new public key on the transaction record. Only the photo seller with the corresponding private key can decrypt the e-cash. Since it is hard to trace the e-cash, anonymity is preserved.

There are few issues remain unsolved in the proposed scheme. First of all, the newly generated RSA key pair is not from trusted authority. There might be security issues aroused by the unsecure RSA key pair. Furthermore, there might be an ethical issue aroused in the scheme, as users can upload forbidden photos in order to make money.
2. News Agencies Subscription

We believe that it is the news agencies that are most interested in citizen journalism platforms. Therefore, one possible improvement in our system is to allow news agencies to subscribe to our system.

The subscribed users can have a dedicated button in the page when photo sellers are choosing the tags and comments. The sellers can choose to notify the subscribed users about the photo uploaded. The subscribed users are then immediately notified of the event and decide to buy the photo or not. This is a win-win situation for the photo sellers and subscribed users as the sellers can increase the probability of selling the photo, and the subscribers can get the information of the event as soon as possible.

3. Map Navigation

The system we have implemented can show the users events happening around them. Some of the users interested in the event may like to go directly to the location to witness the event. To improve user experience, the system should guide the user from the current location to the site of event.

The map navigation can be made possible by integrating navigation services from Google Map or Bing Map.

4. Blacklisting and Whitelisting

We have implemented the “like” and “dislike” function for photos in the system and counting the number of views of the photo. However, there should be a further development in the system in order to rate a photo according to popularity of photos. If a photo is of high popularity, the user should be awarded, and if a user is uploading too many inappropriate photos, we should punish the user.

We can implement a whitelisting scheme, which allows users of high rating to have priority in photo ranking, so that the photos taken by them are more easily seen by others.
We can also implement a blacklisting scheme, which bans those users uploading many inappropriate photos. Since we register users with the Device ID of the phone and the Live ID account, banning the user means the user have to buy a new phone and use a new Live ID in order to join the platform again.

5. Anonymous Polling using Ring Signature

Recently, the Public Opinion Programme of the University of Hong Kong has hosted an online poll on Chief Executive Election. However, to poll using the system, the user has to provide their Hong Kong ID number in order to identify themselves as a valid voter. Some people are concerned that the polling system may link their poll with their Hong Kong ID number and discloses their vote and identity.

As a spinoff of our project, we suggest that the ring signature scheme can solve the aforementioned problem. The system can use ring signature to sign on the vote, so that we can make sure the voter is valid without disclosing their identity.

There are still problems like double voting if we implement ring signature on the voting system, but this is outside the scope of this project.
Conclusion

Recalling the 4 main goals of our project, which are: (1) A two-way process selling photos with encryption and digital signature; (2) ring signature for uploading photos anonymously; (3) a map to let people know the news around them; (4) categorization of photos to improve organization; we have created a photo sharing platform that have met the goals.

The photo selling process has been speeded up so as to encourage people to use our application in everyday life. We help to protect the interests of both the photo buyers and sellers by photo encryption and digital signatures so that the users can trust our platform to do photo transaction.

The ring signature scheme in our project can help to protect the identity of the photo sellers, so that they can share photos freely on our platform.

We have incorporated GPS technologies and map services so that people can quickly find news happening around them.

Last but not least, through categorization of photos, we can allow users to quickly choose the photos according to their own preferences.

Although there still many bugs and performance issues in the photo sharing platform, especially in the newly introduced ring signature scheme, we can foresee that there are huge potentials in further development in the platform.

All in all, we are confident that our application can help solving the existing problem photo sharing mechanism, as well as promoting citizen journalism.