Ebblink Documentation

version 1 – January 2017

Table of Contents

Introduction

Ebblink provides increased control over image sharing. Ebblink uses image encryption technology to store photos in secure servers, so that only authorized users can view a shared image. Additionally, Ebblink's technology enables a user to make images available for only a limited time period. In order to provide enhanced control, Ebblink users can revoke all access to an image at any time. Secure technology locks images in Ebblink's servers, so that they cannot be viewed or analyzed by FiveOpenBooks, LLC. The application currently runs on iOS-supported mobile devices with planned future support of Android mobile devices.

Ebblink's base technology is P3, which ensures privacy while being efficient in terms of storage and bandwidth. The security provided by Ebblink's P3 patented technology stems from the separation of a "secret part" and "public part" during the encryption process. The public part is a JPEG-compliant file, from which the most important image content has been removed. The private part is encrypted and will be needed to unlock the full image content. Only authorized viewers having access to the encryption keys can recombine both parts in order to view the original image.

How to test the technology

The easiest ways to test the Ebblink technology is to download our iOS app:

Link to AppStore

Alternatively, the technology can be tested on the web:

Link to web app

Key technology used to build these apps is available in our SDK and API.

Use cases

Ebblink technology provides access control by creating rules that are applied by our servers in order to determine if an image can be opened. This provides maximum flexibility to developers so that they can create access control that best fit their application. For example:

- A password can be embedded in the URL generated by our service or, for additional privacy, it may be sent separately to the user so that it can be shared directly with the intended recipients;
- Automated expiration time can be set by the application (arbitrary times of up to 1 year);
- Images can be deleted at any time;
- Other rules can be applied to make an image expire (for example, based on the number of times it has been accessed)

Example Scenarios

Time limit

A user has just completed a confidential prototype and now needs to send some quick photos to her overseas team members. Due to the sensitivity surrounding the project, the user cannot send the photos directly via text message. Instead the user opens Ebblink, knowing the application will keep her images secure and private (they will be encrypted so that even the Ebblink service provider cannot open them). The user chooses to enhance the control she has by setting a 1 hour time lifespan for her images, long enough to share the concept yet short enough to guarantee the image's confidentiality.

Unshare Image

Last night, a user shared an image in his fraternity's group chat of him completing an impressive round of shots. This morning, the user opens Ebblink to relive his late night glory. While reminiscing, the user notices the number of viewers has jumped from the hundred making up his fraternity to a couple thousand, far beyond his intended audience. Though still a bit hazy, the user recalls he has a job interview this coming Thursday. Instead of panicking, the user simply chooses Unshare Image, erasing the photo everywhere, and continues to brew his very black cup of coffee.

Ebblink API

In order to begin implementing Ebblink API, sign up as a developer, read the documentation, and begin making requests against the API. An API key will be provided once a Developer Account has been created by contacting FiveOpenBooks, LLC. Authorization will require all queries to contain an API key header followed by the provided API key.

Quick Start

- Sign up for a developer account and get an API Key: info@fiveopenbooks.com
- Make requests against the API: https://sandbox.ebb.pics

Create an Image

```
POST /images/upload
```

View an Image

```
GET /images/view
```

Get Image Information

```
GET /images/status
```

Unshare an Image

```
DELETE /images/kill
```

Example Scenarios

Time limit

Sharing an image with a 1-hour time lifespan

Flow chart:

1. Use **Create Image** using a 1 hour time limit, by sending a `POST` request with parameter `duration=3600seconds`.
2. Response to **Create Image** is successful, `Response (200)`.

Unshare Image

Unsharing an image after it has been shared

Flow chart:

1. Application uses **Create Image** call.
2. Image is successfully created and Application receives `POST` `Response (200)`.
3. Application sends **Get Image Information** request, `GET`.
4. `GET` Response shows the number of views.

5. Application uses **Unshare an Image**, `DELETE`.

Ebblink application basics

- Standard HTTP verbs
- Standard HTTP error responses
- JSON format for responses
- SSL format for communications

API Responses

Code	Meaning
200	Success
204	No Content
400	Request Malformed
401	Unauthorized
404	Not found
500	Internal server Error

Authentication headers

All requests to the Ebblink API should include headers authentication and userid:

- `Authentication` token provided when registering to the platform
- `Userid` unique id for each end user, this id will be used to identify who is the owner of each image

Error response

```
{
"code": 40401,
"message": "No image found", "key": "IMAGE_NO_FOUND"
}
```

API Parameters

Create Image

`POST` : sends data to a specific resource for processing.

Parameters:

- `image` (`String` , Image file in base 64)
- `duration` (`Integer` , in seconds, maximum 1 year)
- `embeddedPass` (`Boolean` , Optional, by default `1` . If set to `1` the password is embedded in the URL so the viewer will not needed to decrypt the image. This makes things easier and faster. But if there are privacy concerns, the password can be shared separately from the URL by setting this value to `0` . The returned link will ask the viewer for the password to decrypt the image and this call will return the password to be shared with the image recipient.)

Notes:

Maximum duration is `duration=31557600` in seconds (1 year).

`POST` response (200):

- `publicUrl` (`String`), URL pointing to the image
- `password` (`String`), If `embeddedPass=0`
- `imageID` (`String`), Unique identifier for the image

Create Image

```
POST /images/upload
```

View Image

`POST` : sends data to a specific resource for processing.

Parameters:

- `imageId` (`String` , Image id)
- `password` (`String`)

`POST` Response (200):

- `image` (`String`), Image decrypted in base64

View Image:

```
POST /images/view
```

Image Information

`GET` : requests data from a specific resource.

Parameters:

- `imageId`

`GET` Response (200):

- `views` (`Integer` , Number of times the image has been viewed)
- `likes` (`Integer` , Number of likes for that image)
- `thumbs` (`Integer` , Number of thumbs up for that image)
- `smiles` (`Integer` , Number of smiles for that image)
- `remainingTime` (`Integer` , Remaining time in seconds till image expires)
- `link` (`String` , Ebblink URL to reshare)

Image Information:

```
GET /images/status
```

Unshare Image

`DELETE` : deletes a specific resource.

Parameters:

- `imageId`

`DELETE` response (204)

Unshare Image

```
DELETE /images/kill
```

Ebblink iOS SDK

A library to integrate Ebblink private image sharing capabilities into your iOS app.

Quick Start

Requirements

- iOS 8+

Setup

Add the following to your Podfile:

```
pod 'ebblinkSDK', :git => 'https://github.com/ebbapp/ebblinkSDK'
```

Instantiate Ebblink iOS SDK

To get a token write to us at info@fiveopenbooks.com The userId is a parameter that identifies a unique user. You can use any string as the unique identifier, such as for example your email.

```
+ (void)initWithToken:(NSString*)token andUserId:(NSString*)userId;
```

Set processing mode

By default the processing is done on the device. Optionally, you can select to have the processing done on the server.

```
+ (void)setProcessingMode:(EBProcessingMode)processingMode;
```

Upload a new image

```
+ (void) p3ImageUpload:(UIImage*)image duration:(NSNumber*) duration completion:(void (^)
(NSString* imageId, NSString* password, NSString* publicLink, NSString* error))completionBlock;
```

View shared image

```
+ (void) p3ImageView:(NSString*)imageId withPassword:(NSString*)password completion:(void (^)
(UIImage* image, NSString* error))completionBlock;
```

Check image status

```
+ (void) p3ImageStatus:(NSString*)imageId completion:(void (^)(NSDictionary* dictionary,
NSString* error))completionBlock;
```

Delete image shared image

```
+ (void) p3ImageKill:(NSString*)imageId completion:(void (^)(BOOL success, NSString*
error))completionBlock;
```

License