JSONStore Security Utilities

Overview

The MobileFirst client-side API provides some security utilities to help protect your user's data. Features like JSONStore are great if you want to protect JSON objects. However, it is not recommended to store binary blobs in a JSONStore collection.

Instead, store binary data on the file system, and store the file paths and other metadata inside a JSONStore collection. If you want to protect files like images, you can encode them as base64 strings, encrypt it, and write the output to disk. When it is time to decrypt the data, you can look up the metadata in a JSONStore collection, read the encrypted data from the disk, and decrypt it using the metadata that was stored. This metadata can include the key, salt, Initialization Vector (IV), type of file, path to the file, and others.

At a high level, the SecurityUtils API provides the following APIs:

- Key generation - Instead of passing a password directly to the encryption function, this key generation function uses Password Based Key Derivation Function v2 (PBKDF2) to generate a strong 256-bit key for the encryption API. It takes a parameter for the number of iterations. The higher the number, the more time it takes an attacker to brute force your key. Use a value of at least 10,000. The salt must be unique and it helps ensure that attackers have a harder time using existing hash information to attack your password. Use a length of 32 bytes.
- Encryption - Input is encrypted by using the Advanced Encryption Standard (AES). The API takes a key that is generated with the key generation API. Internally, it generates a secure IV, which is used to add randomization to the first block cipher. Text is encrypted. If you want to encrypt an image or other binary format, turn your binary into base64 text by using these APIs. This encryption function returns an object with the following parts:
    - ct (cipher text, which is also called the encrypted text)
    - IV
    - v (version, which allows the API to evolve while still being compatible with an earlier version)
- Decryption - Takes the output from the encryption API as input, and decrypts the cipher or encrypted text into plain text.
- Remote random string - Gets a random hex string by contacting a random generator on the MobileFirst Server. The default value is 20 bytes, but you can change the number up to 64 bytes.
- Local random string - Gets a random hex string by generating one locally, unlike the remote random string API, which requires network access. The default value is 32 bytes and there is not a maximum value. The operation time is proportional to the number of bytes.
- Encode base64 - Takes a string and applies base64 encoding. Incurring a base64 encoding by the nature of the algorithm means that the size of the data is increased by approximately 1.37 times the original size.
- Decode base64 - Takes a base64 encoded string and applies base64 decoding.

Setup

Ensure that you import the following files to use the JSONStore security utilities APIs.

iOS

```
#import "WLSecurityUtils.h"
```

Android

```
import com.worklight.wlclient.api.SecurityUtils
```

JavaScript

No setup is required.

Examples

iOS

Encryption and decryption

```
// User provided password, hardcoded only for simplicity.
NSString* password = @"HelloPassword";

// Random salt with recommended length.
NSString* salt = [WLSecurityUtils generateRandomStringWithBytes:32];

// Recomended number of iterations.
int iterations = 10000;

// Populated with an error if one occurs.
NSError* error = nil;

// Call that generates the key.
NSString* key = [WLSecurityUtils generateKeyWithPassword:password
                              andSalt:salt
                              andIterations:iterations
                              error:&error];

// Text that is encrypted.
NSString* originalString = @"My secret text";
NSDictionary* dict = [WLSecurityUtils encryptText:originalString
                                 withKey:key
                                 error:&error];

// Should return: 'My secret text'.
NSString* decryptedString = [WLSecurityUtils decryptWithKey:key
                                      andDictionary:dict
                                      error:&error];
```

Encode and decode base64

```
// Input string.
NSString* originalString = @"Hello world!";

// Encode to base64.
NSData* originalStringData = [originalString dataUsingEncoding:NSUTF8StringEncoding];
NSString* encodedString = [WLSecurityUtils base64StringFromData:originalStringData length:originalStrin
g.length];

// Should return: 'Hello world!'.
NSString* decodedString = [[NSString alloc] initWithData:[WLSecurityUtils base64DataFromString:encode
dString] encoding:NSUTF8StringEncoding];
```

Get remote random

```
[WLSecurityUtils getRandomStringFromServerWithBytes:32
              timeout:1000
              completionHandler:^(NSURLResponse *response, NSData *data, NSError *connectionError) {

  // You might want to see the response and the connection error before moving forward.

  // Get the secure random string.
  NSString* secureRandom = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
}];
```

Android

Encryption and decryption

```
String password = "HelloPassword";
String salt = SecurityUtils.getRandomString(32);
int iterations = 10000;

String key = SecurityUtils.generateKey(password, salt, iterations);

String originalText = "Hello World!";

JSONObject encryptedObject = SecurityUtils.encrypt(key, originalText);

// Deciphered text will be the same as the original text.
String decipheredText = SecurityUtils.decrypt(key, encryptedObject);
```

Encode and decode base64

```
import android.util.Base64;

String originalText = "Hello World";
byte[] base64Encoded = Base64.encode(text.getBytes("UTF-8"), Base64.DEFAULT);

String encodedText = new String(base64Encoded, "UTF-8");

byte[] base64Decoded = Base64.decode(text.getBytes("UTF-8"), Base64.DEFAULT);

// Decoded text will be the same as the original text.
String decodedText = new String(base64Decoded, "UTF-8");
```

Get remote random

```
Context context; // This is the current Activity's context.
int byteLength = 32;

// Listener calls the callback functions after it gets the response from the server.
WLRequestListener listener = new WLRequestListener(){
  @Override
  public void onSuccess(WLResponse wlResponse) {
    // Implement the success handler.
  }

  @Override
  public void onFailure(WLFailResponse wlFailResponse) {
    // Implement the failure handler.
    }
};

SecurityUtils.getRandomStringFromServer(byteLength, context, listener);
```

Get local random

```
int byteLength = 32;
String randomString = SecurityUtils.getRandomString(byteLength);
```

JavaScript

Encryption and decryption

```
// Keep the key in a variable so that it can be passed to the encrypt and decrypt API.
var key;

// Generate a key.
WL.SecurityUtils.keygen({
  password: 'HelloPassword',
  salt: Math.random().toString(),
  iterations: 10000
})

.then(function (res) {

  // Update the key variable.
  key = res;

  // Encrypt text.
  return WL.SecurityUtils.encrypt({
    key: key,
    text: 'My secret text'
  });
})

.then(function (res) {

  // Append the key to the result object from encrypt.
  res.key = key;

  // Decrypt.
  return WL.SecurityUtils.decrypt(res);
})

.then(function (res) {

  // Remove the key from memory.
  key = null;

  //res => 'My secret text'
})

.fail(function (err) {
  // Handle failure in any of the previously called APIs.
});
```

Encode and decode base64

```
WL.SecurityUtils.base64Encode('Hello World!')
.then(function (res) {
  return WL.SecurityUtils.base64Decode(res);
})
.then(function (res) {
  //res => 'Hello World!'
})
.fail(function (err) {
  // Handle failure.
});
```

Get remote random

```
WL.SecurityUtils.remoteRandomString(32)
.then(function (res) {
  // res => deba58e9601d24380dce7dda85534c43f0b52c342ceb860390e15a638baecc7b
})
.fail(function (err) {
  // Handle failure.
});
```

Get local random

```
WL.SecurityUtils.localRandomString(32)
.then(function (res) {
  // res => 40617812588cf3ddc1d1ad0320a907a7b62ec0abee0cc8c0dc2de0e24392843c
})
.fail(function (err) {
  // Handle failure.
});
```