Storing sensitive data in encrypted cache

What is Encrypted Cache?

Encrypted Cache is a mechanism for storing sensitive data on the client side. Encrypted Cache is implemented by using HTML5 web storage technology, which allows data to be saved locally and retrieved on subsequent application use or relaunch.

The Data is encrypted using a combination of a user-provided key and a server-retrieved randomly generated token, which makes it more secure Data is stored in key-value pairs. Encrypted Cache is like a security deposit box – it remains open until you close it, so it is important to remember to close the cache when finished working with it.

Encrypted Cache is similar to technologies such as:

- Local web or DOM storage
- Indexed database API
- Cordova API: Storage API or File API
- JSONStore

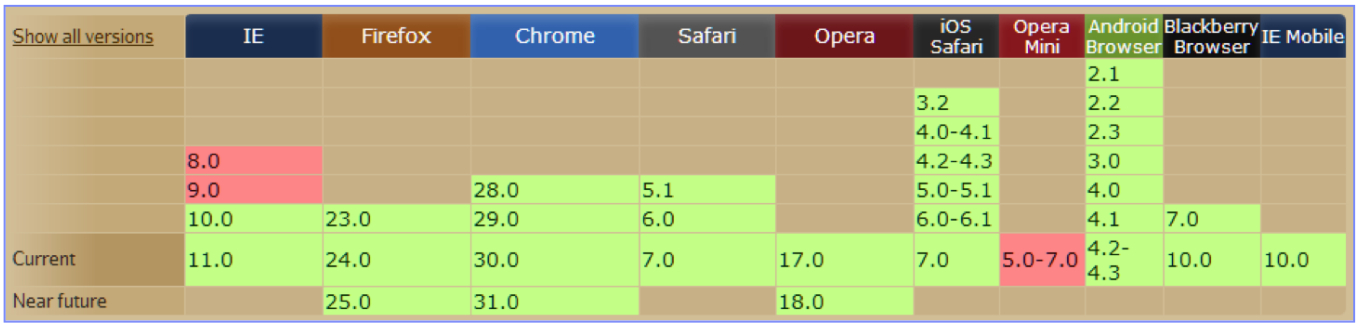
The table on the next slide shows how some features provided by Encrypted Cache compare with other technologies.

	JSONStore	Encrypted Cache	Local Storage	Indexed DB	Cordova Storage	Cordova File
Android Support	Yes	Yes	Yes	Yes	Yes	Yes
iOS Support	Yes	Yes	Yes	Yes	Yes	Yes
Web	Dev. Only (3)	Yes	Yes	Yes	-	-
Data Encryption (1)	Yes	Yes	-	_	_	_
Maximum Storage	Available space	~ 5 MB	~ 5 MB	> 5 MB	Available space	Available space
Reliable Storage (2)	Yes	-	-	-	Yes	Yes
Adapter Integration (1)	Yes	-	-	-	-	-
Multi User Support (1)	Yes	-	-	-	-	-
Indexing	Yes	-	-	Yes	Yes	-
Type of Storage	JSON Documents	Key – value pairs	Key – value pairs	JSON Documents	Relational (SQL)	Strings

(1): These features are further described in the module JSONStore – Common JSONStore usage. (2): Reliable Storage means that your data is not deleted unless the application is removed from the device or one of the methods that removes data is called. (3): Dev. Only means that it is designed only for development. There are no security features and a ~5 MB storage space limit.

Supported browsers and devices

Encrypted cache is implemented by using HTML5 web storage technology. Mobile devices HTML5 web storage support chart:



Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser	IE Mobile
								2.1		
						3.2		2.2		
						4.0-4.1		2.3		
	8.0					4.2-4.3		3.0		
	9.0		28.0	5.1		5.0-5.1		4.0		
	10.0	23.0	29.0	6.0		6.0-6.1		4.1	7.0	
Current	11.0	24.0	30.0	7.0	17.0	7.0	5.0-7.0	4.2-4.3	10.0	10.0
Near future		25.0	31.0		18.0					

= Supported = Not supported

For more information, see http://caniuse.com (http://caniuse.com)

Creating and opening encrypted cache

To create or open previously created encrypted cache, use the following API:

`WL.EncryptedCache.open(credentials, createIfNone, onComplete, onError);`

- `credentials` – string value that represents user-provided password
- `createIfNone` – Boolean value that specifies whether new encrypted cache is created if none is found
- `onComplete` – a callback function to be invoked when cache opening/creating is complete
- `onError` - a callback function to be invoked when cache is not successfully opened/created

```
WL.EncryptedCache.open(key, true, onOpenComplete, onOpenError);
function onOpenComplete(status){
    busyIndicator.hide();
    alert("Encrypted cache succesfully opened");
}
```

Note: The application must be able to connect to the MobileFirst Server to create a new encrypted cache.

A callback function can receive one of the following statuses:

- `WL.EncryptedCache.OK` – Encrypted cache was successfully opened or created
- `WL.EncryptedCache.ERROR_CREDENTIALS_MISMATCH` – an attempt was made to open existing encrypted cache by using wrong credentials
- `WL.EncryptedCache.ERROR_SECURE_RANDOM_GENERATOR_UNAVAILABLE` – unable to generate random token due to MobileFirst Server unavailability
- `WL.EncryptedCache.ERROR_NO_EOC` – could not open encrypted cache because it was not previously created
- `WL.EncryptedCache.ERROR_LOCAL_STORAGE_NOT_SUPPORTED` – device does not support HTML5 local storage
- `WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS` – an open() or changeCredentials() request is already running

Example

```
    WL.EncryptedCache.open(key, true, onOpenComplete, onOpenError);
function onOpenComplete(status){
    busyIndicator.hide();
    alert("Encrypted cache succesfully opened");
}
function onOpenError(status){
    busyIndicator.hide();
    switch(status){
        case WL.EncryptedCache.ERROR_KEY_CREATION_IN_PROGRESS:
            alert("ERROR: KEY CREATION IN PROGRESS");
            break;
        case WL.EncryptedCache.ERROR_LOCAL_STORAGE_NOT_SUPPORTED
:
            alert("ERROR: LOCAL STORAGE NOT SUPPORTED");
            break;
        case WL.EncryptedCache.ERROR_NO_EOC:
            alert("ERROR: NO EOC");
            break;
        case WL.EncryptedCache.ERROR_COULD_NOT_GENERATE_KEY:
            alert("ERROR: COULD NOT GENERATE KEY");
            break;
        case WL.EncryptedCache.ERROR_CREDENTIALS_MISMATCH:
            alert("ERROR: CREDENTIALS MISMATCH");
            break;
        default:
            alert("AN ERROR HAS OCCURED. STATUS :: " + status);
    }
}
```

Reading, Writing and Removing data using Encrypted Cache

When the encrypted cache is open, operations such as reading, writing and removing data can be performed.

To store data in encrypted cache, use the following API:

`WL.EncryptedCache.write(key, value, onSuccess, onFailure);`

Example:

```
    WL.EncryptedCache.write(key, value, onWriteSuccess, onWriteFailure);
function onWriteSuccess(status){
    alert("Succesfully encrypted into cache.");
}
function onWriteFailure(status){
    if (status == WL.EncryptedCache.ERROR_EOC_CLOSED)
        alert("Encrypted cache closed, write failed. error code= "+ status);
}
```

To read data from the encrypted cache, use the following API:

`WL.EncryptedCache.read(key, onSuccess, onFailure);`

Example

```
WL.EncryptedCache.read(key, onDecryptReadSuccess, onDecryptReadFailure);
function onDecryptReadSuccess(value){
    alert("Read success. Retrieved value :: " + key + " = " + value);
}
function onDecryptReadFailure(status){
    alert("Encrypted cache closed, reading failed");
}
```

To remove data from the encrypted cache, use the following API:

`WL.EncryptedCache.remove(key, onSuccess, onFailure);`

Example:

```
WL.EncryptedCache.remove(key, onRemoveSuccess, onRemoveFailure);
function onRemoveSuccess(status){
    alert("Succesfully removed from cache.");
}
function onRemoveFailure(status){
    alert("Encrypted cache closed, remove failed");
}
```

Closing and destroying Encrypted Cache

To avoid possible undesired access to Encrypted Cache, close it. After an encrypted cache is closed, access to its data is not possible without the encryption key that was used to create it. To close the encrypted cache, use the following API:

`WL.EncryptedCache.close(onComplete, onFailure);`

Example:

```
function destroyCacheClicked(){
    WL.EncryptedCache.destroy(onDestroyCompleteHandler
,
    onDestroyErrorHandler);
}
function onDestroyCompleteHandler(status){
    alert("Encrypted cache destroyed");
}
function onDestroyErrorHandler(status){
    alert("Error destroying Encrypted cache");
}
```

Changing the encryption key

While Encrypted Cache is in the open state, it is possible to change the encryption key. To do so, use the following API:

```
WL.EncryptedCache.changeCredentials(credentials, onComplete, onError)
```

- `credentials` – new user password to be used
- `onComplete` – a callback function to be invoked when complete
- `onError` – a callback function to be invoked in case of an error
- `Callback - receives a status object with the same structure as WL.EncryptedCache.open()`

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/EncryptedCacheProject.zip) the Studio projet.