Specification Browser Exam Key Request Hash For Validating Safe Exam Browser in the LMS/Exam System

The **Browser Exam Key** feature can be used by the quiz module of a learning management system (LMS) or another exam system to ensure that:

- the right version of Safe Exam **Browser** (SEB) is used to complete the exam;
- the SEB exam client is correctly configured/secured for this specific **exam**.

To achieve this, a unique **key** is generated for each exam when configuring SEB. The exam administrator then has to copy this Browser Exam Key to the according field in the exam/quiz settings of the LMS/exam system.

This feature was possible to be realized in SEB 2.0 together with and thanks to the **.seb** configuration files, which allow to save individual encrypted settings for each exam. By opening a .seb file by double clicking it (or allowing an arbitrary browser to open a downloaded .seb file), the SEB application installed on the exam computer will be started up, reads the .seb file, decrypts it (either requesting the user to enter a password or using a cryptographic identity/key installed on the computer beforehand), reconfigures itself with the settings found in the .seb file and loads the web page at the start URL defined in the .seb file.

When the option “Send Browser Exam Key in HTTP header” is activated, then SEB adds an additional header to every HTTP request which consists of a SHA256 hash of the Browser Exam Key concatenated with the requested URL (which protects the key and keeps it secret, since it changes not only for each exam but also for each requested URL). The header looks like this:

```
"X-SafeExamBrowser-RequestHash" =
81aad4ab9dfd447cc479e6a4a7c9a544e2cafc7f3adeb68b2a21efad68eca4dc;
```

The LMS then can verify the Browser Exam Key received in the HTTP request. For this, the Browser Exam Key saved in the quiz settings has to be concatenated with the requested URL, SHA256 hashed and compared with the value received in the "X-SafeExamBrowser-RequestHash" header.

As we want to validate the SEB version and settings used for the exam, we need to add a bit more complexity, explained in the following section.

How the Browser Exam Key is generated in SEB

To validate the SEB version used for the exam is important: Some security features in older versions can possibly be outdated or haven’t been available at all. So usually it’s a good idea to enforce a specific SEB version to be used for the exam. The second reason for this relates with the nature of SEB being freely available open source software: A technically proficient examinee could compile his own SEB version with manipulated, deactivated or removed security features and use that for an exam.

That’s why the base for a Browser Exam Key is a code signature of the whole SEB application (binaries and resources). How exactly the signature is generated is platform specific and not part of this specification. But the nature of a code signature implicates that it will in any case differ for each SEB version on each platform. Since we want to support at least SEB for Windows and for Mac OS X, we need to implement support for at least two differing Browser Exam Keys for each exam. But it seems better to support an unspecified number of different keys for each exam, in case that in future SEB will support other platforms (for example Linux or some mobile/tablet OS). This also helps in case two different versions of SEB on the same platform have to be used in some case (for example some students use older hardware for which some new SEB version isn’t available), so you can enter the keys for both versions into the quiz settings.

The second reason for supporting several different Browser Exam Keys per exam is that the key should also validate the settings used for the exam. For this, again a signature (a SHA256 hash) for all exam settings is generated. Even though the SEB versions on all platforms should use the same settings options (settings key/values), there always will be some platform specific settings. So this again requires the Browser Exam Key to be platform specific.

As a last security element the Browser Exam Key contains a salt value, which is generated randomly each time a .seb file is saved. This assures the Browser Exam Key to really be unique per exam, even if the same settings would be used for different exams.

The salt value is the only ingredient of the Browser Exam Key, which is saved in the encrypted .seb file (settings key *examKeySalt)*. The other parts (code signature, hash containing all settings) are generated dynamically in each SEB exam client (for performance reasons not with each HTTP request, but at least once before the exam starts and most likely in regular intervals, on a background thread).

Summary Browser Exam Key Generation

1. **Generate code signature of of the whole SEB application (binaries and resources)**. Code signature generation is platform specific. In the case of SEB Windows the code signature includes the XULRunner browser engine (located inside the SEB program directory) and all its profile and configuration files with their default values. The temporary XULRunner config.json, which is created in another directory (with write access for the user process) when starting SEB can be secured with an additional signature contained in the .seb file used for configuring SEB.

2. **Generate a signature (a SHA256 hash) from all exam settings and include the code signature hash** (from 1). Only include settings which are saved in the .seb file, no temporary settings (like system generated UserDefaults key/values in OS X for window positions etc.).

3. **Add a salt value to the hash generated in 1) and 2)**, which is generated randomly each time a .seb file is saved.

4. **The result should be a SHA256 hash value** (32 Bytes, 64 chars string when encoded Base16).

Sending the Browser Exam Key with HTTP requests
In SEB Windows this is done in XULRunner.

1. **Create a SHA256 hash value from the absolute URL and the Browser Exam Key**. The absolute URL (as a **UTF8 encoded string**) is created by resolving the relative URL against its base according to the algorithm given in RFC 1808. The result is again a SHA256 hash value (32 Bytes, 64 chars string when encoded Base16).

2. **Add an additional header to the HTTP request formatted as follows**:
   `"X-SafeExamBrowser-RequestHash" =` *<hash value created in 1)>*`;`

   This will look for example like:
   ```
   "X-SafeExamBrowser-RequestHash" =
   81aad4ab9dfd447cc479e6a4a7c9a544e2cafc7f3adeb68b2a21efad68eca4dc;
   ```

Creating these hashes needs to be done in the same way and with a standard SHA256 hashing algorithm in the clients and the LMS/exam system, so the resulting hash value send in the HTTP request can be compared properly. Usually the components can be concatenated (in the same order!) and then hashed. Often hashing algorithms offer methods to init, update with some data and finalize a hash value. In OS X the hash is created with the following C/Obj-C code:

```
unsigned char hashedChars[CC_SHA256_DIGEST_LENGTH];

const char *urlString = [absoluteRequestURL UTF8String];

CC_SHA256_CTX sha256;
CC_SHA256_Init(&sha256);
CC_SHA256_Update(&sha256, urlString, strlen(urlString));
CC_SHA256_Update(&sha256, browserExamKey.bytes, browserExamKey.length);

CC_SHA256_Final(hashedChars, &sha256);
```

Checking the Browser Exam Key in the LMS

Configuring keys in quiz settings:

- Best would be a text-area on the quiz editing form, where users can enter as many hashes as they like, separated by newlines.
- For future enhancements (SEB Server) it should be possible to set the Browser Exam Key(s) in the quiz settings using the Webservice API of the LMS (usually REST or SOAP).

During the exam:
When entering an exam and if the feature is enabled, check each HTTP request for the *"X-SafeExamBrowser-RequestHash"* header:

1. Take the requested absolute URL and create a SHA256 hash value from the absolute URL and each Browser Exam Key which is entered in the quiz settings.

2. Compare the hash values from 1) with the value received with the *"X-SafeExamBrowser-RequestHash"* header. If one matches, then continue processing the request.

3. If none matches, then display an error message and don't start the exam. If the exam was already running, then save all results entered until now, stop the exam and inform the user. It should not be possible to leave the quiz module until an exam supervisor/administrator enters some special unlocking password or that user's blocked quiz is reset in the Moodle/LMS backend. Otherwise the user could possibly enter into his Moodle/LMS profile, check some resources prepared beforehand and then inform the exam supervisor that there was a technical problem and ask for getting unlocked and continue the exam.

Instead of calculating the hashes of the absolute URL and all Browser Exam Keys and then comparing them to the received request hash value, we can calculate and compare them one by one: Calculate one hash, compare it to the received request hash, if it matches everything is ok and our check is done, otherwise calculate and compare the next hash and so on. This would be particularly efficient if the Browser Exam Keys in the quiz settings are entered in the same order as the frequency of SEB clients per exam key (most common key first, least common last).

Performance Considerations

One open question was: How costly is this request header check? Will this check create too much server load in complex exam settings? Do we have to compromise to achieve a better performance? Can we improve performance programmatically?

Performance of the request header check is affected by:

1. Performance of the SHA256 hash function (-> looks like this is no real problem, hashing seems to be very performant)
2. Number of Exam Browser Keys entered in the quiz settings, means how many different browser versions are allowed to be used for the exam (for example SEB Windows version 3.0, SEB Windows 2.0 due to legacy PCs being used, SEB MacOSX 3.0, SEB for Linux).
3. Number of HTTP requests per page/per second. Complex web pages with many resources (every source file, image etc. causes a request) create more server load due to the request header check.
4. Number of examinees accessing the exam simultaneously.

There would surely be performance improvements possible by reducing security, but performance test in PHP show that this isn't actually an issue. Even supporting a large number of Browser Exam Keys per exam should not be a problem.

In general we prefer more security, as LMS/exam servers can be scaled if necessary.