

# **Spezifikation verschlüsselte Konfigurationsdateien .seb für Safe Exam Browser**

## **Inhalt**

<b>1. Grundsätzliche Überlegungen</b>	<b>2</b>
<b>2. Basic file structure with 4 char prefix</b>	<b>3</b>
Procedure to encrypt and save a .seb file	4
Procedure to load and decrypt a .seb file	4
<b>3. Encryption with public/private key</b>	<b>5</b>
Encrypting	6
Decrypting	6
<b>4. Encryption with password</b>	<b>7</b>
Explanation of terms used	7
Data format	8
Encrypting	8
Decrypting	8
<b>5. Structure and list of keys in the plain XML configuration file before encryption</b>	<b>9</b>
<b>6. Changes</b>	<b>35</b>

## 2. Grundsätzliche Überlegungen

Die .seb-Konfigurationsdateien müssen verschlüsselt sein, damit sie einerseits von den Prüfungsteilnehmern nicht verändert werden können, andererseits sollen gewisse Konfigurationsdetails geheim gehalten werden, um Manipulationsmöglichkeiten zu erschweren.

Die Verschlüsselung muss plattformunabhängig umsetzbar sein, damit sowohl Windows, Mac OS X als auch zukünftig möglicherweise Linux-, iOS-, Android-Clients mit den .seb-Dateien arbeiten können. Ziel ist eine möglichst sichere Verschlüsselung unter Beachtung der heutigen Kryptografie-Standards (und nicht eine möglichst einfache Umsetzbarkeit auf Code-Seite).

Die eigentlichen Konfigurationsdaten sollen als XML-Struktur mit Key/Value-Paaren definiert sein, weil die Konfiguration der Betriebssysteme Mac OS X und Windows ebenfalls auf XML basiert und XML einfach und unabhängig von Plattformen und Programmiersprachen umsetzbar ist. Die Key-Strings sollen möglichst selbsterklärend sein (Abkürzungen vermeiden, Länge ist irrelevant). Human-Readability der gesamten Konfigurationsdateien ist aber weitgehend irrelevant, weil die Benutzer wegen der Verschlüsselung der .seb-Dateien zukünftig nie direkt mit der XML-Datenstruktur in Berührung kommen werden.

Die .seb-Dateien müssen plattformunabhängig zur Konfiguration aller Versionen der Clients geeignet sein. Möglichst viele Key/Value-Paare sollen auf allen Plattformen dieselbe Funktion haben, wo dies prinzipbedingt nicht möglich ist, muss ein kompatibler Weg für die Konfiguration auf unterschiedlichen Plattformen gefunden werden. Beispiel: Genereller Meta-Key zum Ein-/Ausschalten einer Funktionalität und individuelle Detail-Keys pro Plattform für die spezifische Konfiguration.

Es gibt zwei Verschlüsselungsszenarien:

- Beim Einlesen der .seb-Datei wird ein Passwort abgefragt mit dem die Konfigurationsdaten verschlüsselt sind. Dieses Verfahren trennt verschlüsselte Daten und Schlüssel sehr zuverlässig, sofern ein gutes Passwort vom Prüfungsveranstalter gewählt wurde und bis unmittelbar vor Prüfungsbeginn geheim gehalten wird. Damit ist das Verfahren auch ideal für den Einsatz auf studentischen (nicht zentral verwalteten) Rechnern.
- Für das Verschlüsseln der .seb-Datei wird ein Schlüssel verwendet, der zuvor auf dem Rechner sicher gespeichert wurde (im Certificate-Store oder Keychain abgelegt/installiert). Auf zentral verwalteten Rechnern, auf denen die Benutzer/Prüfungsteilnehmer keinen Administrator-Zugang haben, ist der zur Entschlüsselung notwendige Schlüssel sicher von den zu entschlüsselten Daten getrennt.  
Gleichzeitig existiert mit diesem Verfahren kein „Generalschlüssel“, der innerhalb der SEB-Programmdaten abgelegt wäre und damit potenziell extrahiert und publiziert werden könnte, womit dann die gesamte Installationsbasis von SEB kompromittiert wäre.  
Die Verschlüsselung per vorher installiertem Schlüssel kann mit der Verschlüsselung per Passwort kombiniert werden, z.B. für höhere Sicherheit bei einzelnen Prüfungen.

Beim zukünftigen SEB-Server wird ein Verfahren mit noch höherer Sicherheit verwendet werden, bei dem der private Schlüssel für die Entschlüsselung der Konfigurationsdaten und die Kommunikation zwischen SEB-Server und Prüfungsrechnern auf dem SEB-Server liegt, der vor dem Zugriff von Prüfungsteilnehmern und Hochschulpersonal abgeschottet ist.

### 3. Basic file structure with 4 char prefix

A .seb file always begins with a 4 char prefix, which specifies the kind of the following data. The four chars correspond 4 bytes. In general, all text information in .seb files, means prefixes and the XML settings data (keys and values) are coded in Unicode **UTF8**. Encrypted data is in binary format. In the illustration below the .seb file length is n+1 bytes, the lower line shows the byte numbers (ranging from 0 to n).

	prefix		... data ...	
	0 - 3		4 - n	

Valid prefixes:

- pkhs** Public key hash, stands for encrypted data using a cryptographic identity, identified by a 20 bytes hash of the encrypting public key.
- pswd** Password, stands for encrypted data using a password.
- plnd** Plain data, stands for a unencrypted data, see below.
- pwcc** Password, configuring a client. Currently Mac OS X version only (this kind of .seb file is used for overwriting the local preferences of the Mac SEB client). The Windows version can ignore configuration files containing this prefix (abort interpreting the .seb file).

Since a .seb file can be encrypted with both a cryptographic identity and a password, there can be an outer block and an encapsulated inner block, both identified by their own prefixes. If the .seb file is encrypted by both methods, then the outer block is always a “pkhs” block, the inner is a “pswd” block (on Mac OS X the inner block could also be a “pwcc” block). If a .seb file is only encrypted by an identity, there is a outer “pkhs” block and an inner “plnd” block (means the outer block is encrypted with an identity and the inner block is prefixed by “plnd” and its data is not encrypted). If a .seb file is solely encrypted by a password, there is only one “pswd” block (there is no outer/inner block).

### **Procedure to encrypt and save a .seb file**

(This is done on Windows with the SEB configuration tool, on the Mac inside SEB in the preferences window's "Exam" pane)

1. The SEB configuration tool checks if at least a password or a cryptographic identity for encryption is selected. If none is selected, then it aborts saving the .seb file, requesting the user to enter/choose at least one of these security elements.
2. If a password is entered, encrypt the XML settings data with that password and prefix it with "pswd" (store those four bytes in front of the encrypted data).
3. If no password is entered, prefix the plain XML settings data with "plnd".
4. If a cryptographic identity is selected, encrypt the whole resulting data from step 2 or 3 (including the prefix!) with the public key. Store the prefix "pkhs", store the 20 bytes public key hash value (see next chapter) and append the encrypted data to that.
5. Save the resulting (binary) data in the .seb file.

### **Procedure to load and decrypt a .seb file**

(This happens in SEB Starter)

1. Load the whole .seb file into memory. Check for the first four bytes prefix.
2. If the prefix is not "pkhs": Skip to step 4.
3. If the prefix is "pkhs": The 20 bytes public key hash following this prefix (see next chapter) is read and used to retrieve the identity to which the public key is belonging to from the certificate store or keychain. The private key belonging to this identity is retrieved and used to decrypt the encrypted data.
4. Check for the prefix of the data resulting from step 2 or 3. If it is "plnd", then strip the 4 bytes prefix, the remaining data is the XML settings data.
5. If the prefix is "pswd": Request the user to enter a password. Decrypt the data with this password. If decryption fails with a "wrong password" error, ask again for the password (max. 5 times, then abort).
6. If decryption was successful, the resulting data is the XML settings data.

#### 4. Encryption with public/private key

In this scenario, RSA public-key cryptography is used to encrypt and decrypt the plain XML settings data or the XML settings data already encrypted by password (see chapter “basic file structure”). In detail, a X.509 certificate with an embedded public key is used to encrypt the exam settings and the associated private key is used to decrypt the settings in the SEB exam clients (the combination of such a certificate containing a public key together with the according private key is called cryptographic identity). This encryption/decryption scenario is a bit an unusual use of the asymmetric RSA cryptography (because the private key has to be deployed to all the exam clients), but it was chosen because it's easy to deploy the cryptographic identity in form of PKCS #12 data in a password-protected \*.p12 file (which can be renamed to .pfx as used by Windows) to all the exam clients. The private key is stored securely in the Certificate Store in Windows or the Keychain in Mac OS X, so this scenario is quite secure if examinees don't have administrator rights on the (centrally managed) exam computers (means they cannot extract the key from the secure store). Currently SEB is not using the certificate to verify the identity of the originator of the exam settings; so self-signed identity certificates can be used. The asymmetric encryption/decryption methods will be reused for communication with the SEB Sever, which will be implemented later.

.seb settings structure when using a cryptographic identity:

“pkhs”	public key hash	... encrypted data ...
0 - 3	4 - 23	24 - n

Total length of the .seb file is  $n+1$  bytes. After the 4 bytes prefix containing the string “pkhs”, a 20 bytes public key hash follows and after that the encrypted data. In X.509 certificates, the hash of the public key can be used to identity both the certificate (with its embedded public key) and the associated private key. Therefore the SEB on an exam client computer checks if a cryptographic identity with the hash contained in the .seb file is stored in the Certificate Store or Keychain and uses its private key to decode the settings.

In the Mac OS X version of SEB the system provided Common Security Services Manager (CSSM) APIs are used for encryption with public/private key. CSSM belongs to the cross platform, open source Common Data Security Architecture (CDSA), which “is a set of layered security services and cryptographic framework that provide an infrastructure for creating cross-platform, interoperable, security-enabled applications for client-server environments. CDSA covers all the essential components of security capability ... with security services that provide facilities for cryptography, certificate management, trust policy management, and key recovery.” For more information see the official documentation at <http://www.opengroup.org/security/cdsa.htm> and <http://pubs.opengroup.org/onlinepubs/9629299/toc.htm>. Although other compatible security frameworks and higher-level APIs may also be used on other platforms, below the CDSA/CSSM API methods are listed which are used by SEB on OS X.

## Encrypting

1. The exam administrator can choose the identity for encryption from a list of cryptographic identities. These have to meet the following conditions:
  - They need to be complete identities; means there must be a certificate with embedded public key and an associated private key.
  - In the CSSM\_KEY structure of both keys, *KeyHeader.AlgorithmId* must be CSSM\_ALGID\_RSA (they have to be RSA keys)
  - The public key's (its *KeyHeader.KeyClass* is CSSM\_KEYCLASS\_PUBLIC\_KEY) *KeyHeader.KeyUsage* property must have at least one of these 3 flags set: CSSM\_KEYUSE\_ENCRYPT, CSSM\_KEYUSE\_WRAP, CSSM\_KEYUSE\_ANY.
  - The private key's (its *KeyHeader.KeyClass* is CSSM\_KEYCLASS\_PRIVATE\_KEY) *KeyHeader.KeyUsage* property must have at least one of these 3 flags set: CSSM\_KEYUSE\_DECRYPT, CSSM\_KEYUSE\_WRAP, CSSM\_KEYUSE\_ANY.
2. Fetch certificate and its public key from the identity selected for encryption.
3. Get CSSM\_ACCESS\_CREDENTIALS for operation CSSM\_ACL\_AUTHORIZATION\_ENCRYPT with the selected public key.
4. Get CSSM\_CSP\_HANDLE and CSSM\_KEY for the selected public key.
5. Call CSSM\_CSP\_CreateAsymmetricContext with algorithm ID = CSSM\_ALGID\_RSA and Padding = CSSM\_PADDING\_PKCS1. This creates the new context handle CSSM\_CC\_HANDLE.
6. Call CSSM\_EncryptData, the result is the encrypted data.
7. Free the context with CSSM\_DeleteContext.

## Decrypting

1. Using the public key hash find the according certificate -> identity -> private key in the certificate store/keychain on the exam client computers.
2. Get CSSM\_ACCESS\_CREDENTIALS for operation CSSM\_ACL\_AUTHORIZATION\_DECRYPT with the selected private key.
3. Get CSSM\_CSP\_HANDLE and CSSM\_KEY for the selected private key.
4. Call CSSM\_CSP\_CreateAsymmetricContext with algorithm ID = CSSM\_ALGID\_RSA and Padding = CSSM\_PADDING\_PKCS1. This creates the new context handle CSSM\_CC\_HANDLE.
5. Call CSSM\_DecryptData, the result is the decrypted plain XML settings data.
6. Free the context with CSSM\_DeleteContext.

## 5. Encryption with password

In this encryption scenario we use a symmetric AES cipher algorithm, the key is derived from a password. To do this properly, so that we get a secure AES key, is not trivial. The Mac OS X version of SEB uses the open source [RNCryptor](#) Objective-C framework, which provides a secure implementation of AES encryption including correct handling of password stretching (PBKDF2), salting, IV and HMAC (see explanations below). The author also provides good documentation about [properly encrypting with AES](#) and explains, why this effort is necessary. The description below explains generally how the encryption and decryption is done by RNCryptor, which is roughly the procedure that should be used for *any* secure use of AES (it is not specific to RNCryptor). For Windows/.NET and other platforms/languages the encryption has to be re-implemented using the platform provided *standard low-level crypto algorithms, namely PBKDF2, AES-256-CBC and SHA-256*. The OS X version uses the open source C crypto framework [CommonCrypto](#), which could also be used on Linux, although many other open and closed source frameworks also provide these crypto algorithms. [.NET also includes AES](#).

### Explanation of terms used

(Taken from [Wikipedia](#), see the links for the full description)

- AES** Advanced Encryption Standard, specification for the [encryption](#) of electronic data using a [symmetric-key algorithm](#). See also [list of implementations](#).
- HMAC** A hash-based message authentication code is a specific construction for calculating a [message authentication code](#) (MAC) involving a [cryptographic hash function](#) in combination with a secret [cryptographic key](#). As with any MAC, it may be used to simultaneously verify both the [data integrity](#) and the [authenticity](#) of a [message](#).
- IV** Initialization vector, an additional random or pseudorandom input value, which is required to be mixed with the first block when using a block-based encryption algorithm like AES-CBC.
- PBKDF2** Password-Based Key Derivation Function 2 is a [key derivation function](#) that is part of [RSA Laboratories' Public-Key Cryptography Standards](#) (PKCS) series. PBKDF2 applies a [pseudorandom function](#), such as a [cryptographic hash](#), [cipher](#), or [HMAC](#) to the input [password](#) or [passphrase](#) along with a [salt](#) value and repeats the process many times to produce a *derived key*, which then can be used as a [cryptographic key](#) in subsequent operations.
- PRF** Pseudo random function. A hash algorithm like SHA-256 can be used as a PRF.
- SHA-256** [Secure Hash Algorithm](#) belonging to the SHA-2 set of [cryptographic hash functions](#).

## Data format

All data in network order (big-endian).

version	options	encryption salt	HMAC salt	IV	... encrypted data ...	HMAC
0	1	2-9	10-17	18-33	34 - n-33	n-31 - n

1. version (1 byte): Data format version. For now always 1.
2. options (1 byte): Reserved. Always 0.
3. encryption salt (8 bytes)
4. HMAC salt (8 bytes)
5. IV (16 bytes)
6. encrypted data/ciphertext (variable) -- Encrypted with CBC mode.
7. HMAC (32 bytes)

Details:

- EncryptionKey = PBKDF2 (with parameters encryption salt, 10k iterations, password)
- HMACKey = PBKDF2(HMAC salt, 10k iterations, password)
- Encrypted data (ciphertext) is AES-256-CBC encrypted using the given IV and the EncryptionKey (above).
- HMAC is generated using the encrypted data and the HMACKey (above) and the SHA-256 PRF.

## Encrypting

1. Generate a random encryption salt
2. Generate the encryption key using PBKDF2 (see your language docs for how to call this). Pass the password as a string, the random encryption salt, and 10,000 iterations.
3. Generate a random HMAC salt
4. Generate the HMAC key using PBKDF2 (see your language docs for how to call this). Pass the password as a string, the random HMAC salt, and 10,000 iterations.
5. Generate a random IV
6. Encrypt the data using the encryption key (above), the IV (above), AES-256, and the CBC mode. This is the default mode for almost all AES encryption libraries.
7. Pass your ciphertext to an HMAC function, along with the HMAC key (above), and the PRF "SHA-256" (see your library's docs for what the names of the PRF functions are; this might also be called "SHA-2, 256-bits").
8. Put these elements together in the format given above.

## Decrypting

1. Pull apart the pieces as described in the data format.
2. Generate the encryption key using PBKDF2 (see your language docs for how to call this). Pass the password as a string, the given encryption salt, and 10,000 iterations.
3. Generate the HMAC key using PBKDF2 (see your language docs for how to call this). Pass the password as a string, the given HMAC salt, and 10,000 iterations.
4. Decrypt the data using the encryption key (above), the given IV, AES-256, and the CBC mode. This is the default mode for almost all AES encryption libraries.
5. Pass your ciphertext to an HMAC function, along with the HMAC key (above), and the PRF "SHA-256" (see your library's docs for what the names of the PRF functions are; this might also be called "SHA-2, 256-bits"). Verify that your result matches the result in the message you were sent.



## 6. Structure and list of keys in the plain XML configuration file before encryption

The plain settings are saved as serialized objects in a XML format. As base for this the Apple property list ([plist](#)) format was chosen, because it's a) having a simple structure using only a few basic tags, b) being directly supported on Mac OS X and iOS and easy to parse on other systems.

Property list data types and XML tags

Abstract type	XML Tag	Storage format
array	<array>	Indexed collections of values: Can contain any number of child elements
dictionary:	<dict>	Collections of values each identified by a key: Alternating <key> tags and plist element tags
string	<string>	UTF-8 encoded string
number - integer	<integer>	Decimal string
number - floating point	<real>	Decimal string
boolean	<true />, or <false />	No data (tag only)
date	<date>	<a href="#">ISO 8601</a> formatted string
data	<data>	<a href="#">Base64</a> encoded data

Since human readability is not relevant for the .seb configuration file as users will never see it unencrypted, structure of the XML configuration files is reduced to the basic, essential plist-structure. Therefore there are **no sections** as in the old ini files. All keys in the root-level dictionary are unique (each key can only occur once).

### Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>allowDownUploads</key>
  <true/>
  <key>allowFlashFullscreen</key>
  <false/>
  <key>allowPreferencesWindow</key>
  <true/>
  <key>allowQuit</key>
  <true/>
  <key>allowSwitchToApplications</key>
  <true/>
  <key>blockPopUpWindows</key>
  <false/>
  <key>chooseFileToUploadPolicy</key>
  <integer>0</integer>
  <key>downloadDirectoryOSX</key>
  <string>~/Downloads</string>
  <key>downloadPDFFiles</key>
  <false/>
  <key>allowBrowsingBackForward</key>
  <false/>
  <key>enableJava</key>
```

```

    <false/>
    <key>enableJavaScript</key>
    <true/>
    <key>enablePlugIns</key>
    <true/>
    <key>examKeySalt</key>
    <data></data>
    <key>hashedAdminPassword</key>
    <string>155290511d5c4bfb1369217d6846c8eef1ed6a564579516eaf36cf5598ac92de</
string>
    <key>hashedQuitPassword</key>
    <string>8577da2ea54085708b3b851bc50315a36bb740ba5135e747cfb12457b5d3060f</
string>
    <key>newBrowserWindowByLinkBlockForeign</key>
    <false/>
    <key>newBrowserWindowByLinkPolicy</key>
    <integer>2</integer>
    <key>newBrowserWindowByScriptBlockForeign</key>
    <false/>
    <key>newBrowserWindowByScriptPolicy</key>
    <integer>2</integer>
    <key>openDownloads</key>
    <true/>
    <key>quitURL</key>
    <string>http://www.safeexambrowser.org/exit</string>
    <key>sebServicePolicy</key>
    <integer>2</integer>
    <key>startURL</key>
    <string>http://www.safeexambrowser.org/exams</string>
    <key>permittedProcesses</key>
    <array>
      <dict>
        <key>title</key>
        <string>SEB</string>
        <key>nameWin</key>
        <string>xulrunner.exe</string>
        <key>pathWin</key>
        <string>../xulrunner/</string>
        <key>arguments</key>
        <array>
          <string>-app "..\xul_seb\seb.ini"</string>
          <string>-profile "%LOCALAPPDATA%\ETH Zuerich\xul_seb\Profiles"</
string>
            </array>
        </dict>
        <dict>
          <key>title</key>
          <string>Calculator</string>
          <key>nameWin</key>
          <string>calc.exe</string>
        </dict>
        <dict>
          <key>title</key>
          <string>Notepad</string>
          <key>nameWin</key>
          <string>notepad.exe</string>
        </dict>
      </array>
    </dict>
  </plist>

```

## List of possible keys

Some keys are not available on all platforms, see remarks. Keys not available on a platform should be ignored by SEB Starter when parsing and using the settings. The SEB Windows configuration tool and the preferences system in SEB MacOSX should read and keep the key/values defined for another platform untouched and save them together with the configured key/values for the own platform. Like this it will be possible to create a “merged” .seb file with specific settings for both platforms.

**If keys demanded on one platform aren’t found in the settings file (because it has been created for another platform), then the default value has to be used.** The default value therefore always is the “safer” or “safest”, most restrictive option.

### **allowBrowsingBackForward**

Boolean indicating if browsing back to previously visited pages (and forward again) according to the browser history of this browser session (since SEB was started) is allowed or not.

Default value: `<false/>`

*Currently Mac only*

### **allowDownUploads**

Boolean, indicating if downloading and uploading files is allowed.

Default value: `<true/>`

*Currently Mac only*

### **allowFlashFullscreen**

Boolean, indicating if Flash is allowed to switch on fullscreen presentation (mainly used in Flash video players).

Default value: `<false/>`

*Mac only*

### **allowPreferencesWindow**

Boolean indicating if users are allowed to open the preferences window on exam clients. Usually it should be disabled besides for debugging purposes.

Default value: `<true/>`

*Mac only*

### **allowQuit**

Boolean indicating if quitting SEB by key combination, menu entry or window closing button is allowed. This flag does not affect the *quit link feature* (if *quitURL* is set and detected, SEB quits regardless of this flag).

Default value: `<true/>`

*Currently Mac only*

### **allowSwitchToApplications**

*(replaces key ShowSebApplicationChooser on Windows)*

Boolean indicating if users are allowed to switch to permitted applications. This also indicates if the application chooser is displayed or not.

Default value: `<false/>`

### **allowUserSwitching**

Boolean indicating if fast user switching is allowed. When using SEB on students' own computers (or where examinees have access to other user accounts on the exam Macs) and especially when using third party applications during the exam, this key should be set to false. Otherwise when the computer is put to sleep and woken up again, in the lock screen "Switch user" is shown and when using a third party application the fast user switching option is displayed on the right side of the menu bar.

This feature requires the Mac OS X SEB Service being installed (this is why its default value is false).

Default value: `<true/>`

*Mac only*

### **allowVirtualMachine**

Boolean indicating if SEB is allowed to run on a virtual machine (e.g. for exams in virtual desktop environments) or not (in order to prevent potential manipulations).

Default value: `<false/>`

### **blockPopUpWindows**

Boolean indicating if pop-up windows (often advertisement) opened by JavaScript without an user action such as a button click are blocked.

Default value: `<false/>`

*Currently Mac only*

### **browserViewMode**

Integer with a value representing one of the *browserViewModes*: Either use a window for the SEB browser or display the browser full screen.

Possible values:

```
enum {  
    browserViewModeWindow          = 0,  
    browserViewModeFullscreen     = 1  
};  
typedef NSUInteger browserViewModes;
```

Default value: `<integer>0</integer>` (browserViewModeWindow)

*Currently Mac only*

### chooseFileToUploadPolicy

Integer with a value representing one of the *chooseFileToUploadPolicies*: SEB can let the user choose the file to upload manually (as usual) or automatically choose the same file which was downloaded before. There are three possible policies to choose the file to upload:

- **manually with file requester**
- **by attempting to upload same file downloaded before**: If the file is not found, a file requester is presented and the user can choose some other file manually.
- **by only allowing to upload the same file downloaded before**: If the file is not found, an error message is presented. This setting might bring additional security, because only files which have been downloaded before (in the same browser session, means since SEB was started) can be uploaded. If several files have been downloaded, pressing the choose file (or similarly named) button in the browser window will first choose the file most recently downloaded, pressing the button several times will cycle through all the files downloaded in this session.

Possible values:

```
enum {
    manuallyWithFileRequester                = 0,
    attemptUploadSameFileDownloadedBefore    = 1,
    onlyAllowUploadSameFileDownloadedBefore  = 2
};
typedef NSUInteger chooseFileToUploadPolicies;
```

Default value: `<integer>0</integer>` (manuallyWithFileRequester)

*Currently Mac only*

### copyBrowserExamKeyToClipboardWhenQuitting

Boolean indicating that the Browser Exam Key should be copied to the clipboard when quitting SEB MacOSX. Since normally SEB clears the clipboard when quitting, with this option the exam admin finds the current exam key in the clipboard and can paste it into the settings of his exam system. This option is significant only for the SEB MacOSX preferences window.

Default value: `<false/>`

*Mac only*

### createNewDesktop

Boolean indicating if SEB should be executed in a newly created desktop window (in fullscreen mode), such that e.g. the task bar and the start menu at the bottom edge of the screen are blanked out.

Default value: `<true/>`

*Windows only*

### downloadDirectoryOSX

String representing the path of the directory to which downloaded files will be saved. Paths containing the home directory of the current user are abbreviated with the tilde symbol ~.

Default value: `<string>~/Downloads</string>`

*Mac only. Windows equivalent: downloadDirectoryWin*

### downloadDirectoryWin

String representing the path of the directory to which downloaded files will be saved. Paths containing the home directory of the current user should contain a Windows compatible placeholder for the home directory (eventually also for other special system directories).

Default value: *not specified yet*

*Windows only. Mac equivalent downloadDirectoryMac*

### **downloadPDFFiles**

Boolean indicating if PDF files should be downloaded or displayed online inside the browser window.

Default value: `<false/>`

*Currently Mac only*

### **embeddedCertificates**

Array of dictionaries which contain SSL client certificates and cryptographic identities with their properties which are embedded into settings. When SEB loads a .seb settings file with embedded certificates or identities, then it installs them into the OS X Keychain or into XULRunner certificate database (SSL client certificates) or Windows Certificate Store (identities).

Keys in the *embeddedCertificates* dict:

- **certificateData**  
Data (Base64 encoded) of the certificate/identity.
- **name**  
String containing the name of the certificate/identity. The name might be just the common name, the Email address, a combination of both and the public key hash value.
- **type**  
Integer with a value representing the type of the certificate. User interface strings (English): “SSL Certificate”, “Identity”.

Possible values for the *type* key:

```
enum {  
    certificateTypeSSLClientCertificate    = 0,  
    certificateTypeIdentity                = 1,  
};  
typedef NSUInteger certificateTypes;
```

### **enableBrowserWindowToolbar**

Boolean indicating if a toolbar is displayed on top of the browser window which can also be hidden by the user if it's not used.

Default value: `<false/>`

*Currently Mac only*

### **enableJava**

Boolean indicating if Java Applets are enabled. Starting SEB 2.0 this option is disabled by default because Java applets are considered a potential security risk.

Default value: `<false/>`

*Currently Mac only*

### **enableJavaScript**

Boolean indicating if JavaScript is enabled. Please note that most modern websites need JavaScript for full functionality.

Default value: `<true/>`

*Currently Mac only*

**enableLogging**

Boolean indicating if SEB writes a log.

Default value: `<false/>`

**enablePlugins**

Boolean indicating if web plugins like Flash are enabled. For security reasons it's recommended to disable this option if you don't use any plugin content.

Default value: `<true/>`

*Currently Mac only*

**enableSebBrowser**

Boolean indicating if the SEB browser should be used. If you don't want to use any browser in SEB, because SEB Starter should only act as a kiosk application starting up another application in a kiosk mode (for example a virtual desktop infrastructure client), then set *enableSebBrowser* = *false*.

Default value: `<true/>`

**enableUrlContentFilter**

Boolean indicating if not only URLs are filtered using the *urlFilterRules* dictionary, but also all embedded resources.

Default value: `<false/>`

**enableUrlFilter**

Boolean indicating if URLs are filtered using the *urlFilterRules* dictionary.

Default value: `<false/>`

**examKeySalt**

Data representing a random salt value which is used to generate the browser exam key.

No Default value.

**exitKey1**

*(replaces key B1 in MsgHook.ini on Windows)*

Integer value representing a [virtual key code](#) of the first function key to be pressed and held down together with two other keys in the right order to exit SEB.

No default value

*Windows only*

**exitKey2**

*(replaces key B2 in MsgHook.ini on Windows)*

Integer value representing a [virtual key code](#) of the second function key to be pressed and held down together with two other keys in the right order to exit SEB.

No default value

*Windows only*

**exitKey3**

*(replaces key B3 in MsgHook.ini on Windows)*

Integer value representing a [virtual key code](#) of the third function key to be pressed and held down together with two other keys in the right order to exit SEB.

No default value

*Windows only*

**hashedAdminPassword**

String containing Base16 encoded data representing a SHA256 hash of the password required to enter the preferences window.

Default value: `<string></string>` (empty string = no admin password set)

*Mac only*

**hashedQuitPassword**

String containing Base16 encoded data representing a SHA256 hash of the password which is prompted when users try to quit SEB.

Default value: `<string></string>` (empty string = no quit password set)

*Mac OS X: When this key isn't available in a .seb file while parsing it, but the keys exitKey1-3 are defined, then the quit password is generated from those keys, for example „F4+F11+F6“. Such a password isn't really safe, so even when generating a .seb file on Windows and wanting to use only exit keys in the Windows version of SEB, a secure quit password should be defined.*

**hideBrowserWindowToolbar**

Boolean indicating if the browser window toolbar should be hidden by default. Users can unhide the toolbar in the view menu or the contextual menu on the browser window title bar. In full screen browser view mode, the toolbar is auto hidden with this setting and appears when users move the mouse towards the screen's top border.

Default value: `<false/>`

*Mac only*

**hookMessages**

Boolean indicating if SEB should intercept input functions like key combinations (for example Alt+F4) or right mouse click.

Default value: `<true/>`

*Windows only*

**hookedMessages**

Dictionary containing keys representing intercepted input functions and boolean values (on the second line).

The setting for the function keys doesn't affect the exit keys.

- **enableEsc**  
`<false/>`
- **enableCtrlEsc**  
`<false/>`
- **enableAltEsc**  
`<false/>`
- **enableAltTab**  
`<true/>`
- **enableAltF4**  
`<false/>`
- **enableRightMouse**  
`<false/>`



- **enableStartMenu**  
<false/>
- **enableF1**  
<false/>
- **enableF2**  
<false/>
- **enableF3**  
<false/>
- **enableF4**  
<false/>
- **enableF5**  
<true/>
- **enableF6**  
<false/>
- **enableF7**  
<false/>
- **enableF8**  
<false/>
- **enableF9**  
<false/>
- **enableF10**  
<false/>
- **enableF11**  
<false/>
- **enableF12**  
<false/>

*All keys Windows only*

### **ignoreQuitPassword**

Boolean indicating if SEB should ignore the quit password and should only be quit by pressing and holding down three function keys in a specific order (which need to be defined with *exitKey1-3*). If you want to use just a quit password also in SEB Windows, then don't define any *exitKeys*.

Default value: <false/>

*Windows only*

### Keys for Windows Security Screen registry values

These boolean values indicate if the respective options are visible and active on the Windows Security Screen which appears when the key combination Ctrl + Alt + Del is pressed.

There is an *insideSeb...* and an optional ***outsideSeb...*** variant of each of these flags. The options *insideSeb...* define the values inside of SEB (which are thus valid during an exam, when SEB is running). The options *outsideSeb...* define the values outside of SEB (which are thus valid in normal use, when SEB is not running). The standard defaults are to disable (= <false/>) all options inside SEB.

Normally exam admins should not have to use any *outsideSeb...* options. Especially on unmanaged computers (like student's own notebooks) it should not be assumed that all computers have the standard windows setting (all options enabled)! The *outsideSeb...* options could however be used in a special "recovery" .seb settings file for cases when setting and resetting the registry options didn't work properly.

**There is no default value for the options outside SEB** (at least if nothing went wrong badly, in that case the default value would be assumed *enabled*/true/>, see below):

**Instead, SEB Windows 2.0 assures to set and reset each registry values strictly in an atomic way:**

1. When SEB starts: Check if there is already a persistently saved registry option
  - If yes: Something went wrong, possibly SEB crashed or it has been killed instead of quitted properly last time. Don't overwrite the persistently saved registry option. Proceed with step 2.
  - if not: Save the current value of the registry option persistently (in a file inside it's settings directory), this saved value therefore represents the outside SEB value.
2. Set the registry option to the value of the *insideSeb...* option.
3. Perform a check to assure that the registry option was correctly set to the *insideSeb...* value (read it again and compare if the value is what it should be). If not, then react depending on the *sebServicePolicies*:
  - *ignoreService* -> no reaction (just write it into log if it's active),
  - *indicateMissingService* -> display error message, but offer option to continue exam
  - *forceSebService* -> display error message with only option to quit SEB
4. When SEB quits, then reset the registry values:
  - If there is an *outsideSeb...* option flag in the currently active settings, then reset the registry value to the value of the *outsideSeb...* flag
  - Otherwise:
    - If there is a persistently saved registry option value (there should be one, otherwise something went wrong badly...) reset the registry option value to the persistently saved value.
    - If there is no persistently saved registry option value (something went wrong badly), reset the registry option to *enabled/true*. Write a warning into the log.
  - Check if the registry option value was correctly reset (read it again and compare if the value is what it should be).
    - If yes: delete the persistently saved registry option value.
    - If not, don't delete the persistently saved registry option value, display an error message saying something like "The registry setting for <name of the key> could not be reset. Try to restart and quit SEB again, possibly after restarting your computer."
  - Quit SEB.

### **insideSebEnableSwitchUser / outsideSebEnableSwitchUser**

Boolean indicating if the button "Benutzer wechseln" or "Switch User" is activated.

### **insideSebEnableLockThisComputer / outsideSebEnableLockThisComputer**

Boolean indicating if the button "Computer sperren" or "Lock this computer" is activated.

### **insideSebEnableChangePassword / outsideSebEnableChangePassword**

(replaces keys *insideSebEnableChangeAPassword* and *outsideSebEnableChangeAPassword* on Windows)

Boolean indicating if the button "Kennwort ändern..." or "Change a password..." is activated.

**insideSebEnableStartTaskManager / outsideSebEnableStartTaskManager**

Boolean indicating if the button "Task-Manager starten" or "Start Task Manager" is activated.

**insideSebEnableLogOff / outsideSebEnableLogOff**

Boolean indicating if the button "Abmelden" or "Log off" is activated.

**insideSebEnableShutDown / outsideSebEnableShutDown**

Boolean indicating if the button "Herunterfahren" or "Shutdown" in the lower right corner is activated.

**insideSebEnableEaseOfAccess / outsideSebEnableEaseOfAccess**

Boolean indicating if the button "Erleichterter Zugriff" or "Ease of Access" in the lower left corner is activated, which offers help e.g. to visually or aurally handicapped persons, like the Magnifier Glass.

**insideSebEnableVmWareClientShade / outsideSebEnableVmWareClientShade**

Boolean indicating if the "Shade" bar at the upper edge of a virtual desktop is activated, if existent.

Default value for all the *insideSEB...* keys: `<false/>`

Default value for all the *outsideSEB...* keys: `<true/>`

(used only in rare cases when resetting keys didn't work properly)

*Windows only*

**logDirectoryOSX**

String representing the path of the directory to which log files will be saved. Paths containing the home directory of the current user are abbreviated with the tilde symbol ~. The special value `<string>NSTemporaryDirectory</string>` will use the temporary directory for the current user.

Default value: `<string>NSTemporaryDirectory</string>`

*Mac only. Windows equivalent: logDirectoryWin*

**logDirectoryWin**

String representing the Windows formatted path of the directory to which log files will be saved. Paths containing the home directory of the current user should contain a Windows compatible placeholder for the home directory (eventually also for other special system directories).

Default value: *not specified yet*

*Windows only. Mac equivalent: logDirectoryOSX*

**mainBrowserWindowHeight**

String indicating the height in pixels of the main browser window (if it's not in full screen mode). A height of 0 means that SEB opens the window with the full usable screen height (full height of the current screen minus the height of the SEB dock/task bar if *showTaskBar = true* and minus the height of the menu bar if *showMenuBar = true*).

Default value: `<integer>0</integer>`

### **mainBrowserWindowPositioning**

Integer with a value representing one of the *browserWindowPositionings*.

Possible values:

```
enum {  
    browserWindowPositioningLeft           = 0,  
    browserWindowPositioningCenter        = 1,  
    browserWindowPositioningRight         = 2  
};  
typedef NSUInteger browserWindowPositionings;
```

### **mainBrowserWindowWidth**

Integer indicating the width in pixels of the main browser window (if it's not in full screen mode). A width of 0 means that SEB opens the window with the full width of the current screen.

Default value: `<integer>0</integer>`

### **monitorProcesses**

Boolean indicating if SEB is monitoring which processes (and applications) are running during an exam. Third party applications and other processes which are not permitted to run during an exam (not having an entry in the permittedProcesses dictionary or being explicitly blacklisted in prohibitedProcesses or being an exception like some specific system processes) are killed by SEB if they start up during an exam. If they are running when SEB is started, then an alert/dialogue window is displayed to tell the user to quit the not permitted (prohibited) applications and to restart SEB afterwards or to let SEB kill the applications risking that there could be data loss. SEB kills not permitted background processes itself, without user confirmation. Applications which allow to be terminated nicely in OS X are automatically terminated (also not asking the user).

Default value: `<false>`

### **newBrowserWindowByLinkBlockForeign**

Boolean indicating if hyperlinks which direct to a different host than the one of the current page should be ignored.

Default value: `<false>`

*Currently Mac only*

### **newBrowserWindowByLinkHeight**

Integer indicating the height in pixels of browser windows opened by a link requesting to be opened in a new browser window (target="\_blank" or target="\_new"). A height of 0 means that SEB opens the window with the full usable screen height (full height of the current screen minus the height of the SEB dock/task bar if *showTaskBar = true* and minus the height of the menu bar if *showMenuBar = true*).

Default value: `<integer>0</integer>` (= full usable screen height)

### **newBrowserWindowByLinkPolicy**

Integer with a value representing one of the *newBrowserWindowPolicies*.

Possible values:

```
enum {
    getGenerallyBlocked                = 0,
    openInSameWindow                  = 1,
    openInNewWindow                    = 2
};
typedef NSUInteger newBrowserWindowPolicies;
```

Default value: <integer>2</integer> (openInNewWindow)

### **newBrowserWindowByLinkPositioning**

Integer with a value representing one of the *browserWindowPositionings*.

Possible values:

```
enum {
    browserWindowPositioningLeft       = 0,
    browserWindowPositioningCenter     = 1,
    browserWindowPositioningRight      = 2
};
typedef NSUInteger browserWindowPositionings;
```

### **newBrowserWindowByLinkWidth**

String containing the width in pixels or as percentage (followed by the % sign) of screen width of browser windows opened by a link requesting to be opened in a new browser window (target="\_blank" or target="\_new").

Default value: <string>1000</string>

### **newBrowserWindowByScriptBlockForeign**

Boolean indicating if hyperlinks which direct to a different host than the one of the current page should be ignored.

Default value: <false/>

*Currently Mac only*

### **newBrowserWindowByScriptPolicy**

Integer with a value representing one of the *newBrowserWindowPolicies* (see *newBrowserWindowByLinkPolicy*) for hyperlinks opened from JavaScript or plug-ins (like Flash).

Default value: <integer>2</integer> (openInNewWindow)

*Currently Mac only*

### **proxySettingsPolicy**

Integer with a value representing one of the *proxySettingsPolicies*

Possible values:

```
enum {
    useSystemProxySettings             = 0,
    useSEBProxySettings                = 1,
};
typedef NSUInteger proxySettingsPolicies;
```

Default value: <integer>0</integer> (useSystemProxySettings)

### openDownloads

Boolean indicating if downloaded files will be opened (with the according application, which currently has to be set correctly in the system for each used file type, in a future SEB version it will be possible to include file/MIME types and according applications in the .seb file).

Default value: `<false/>`

*Currently Mac only*

### quitURL

String containing the full URL (starting with `http://` or `https://`) of the link to quit SEB after exam.

Default value: `<string></string>` (empty string = quit link feature not active)

### sebBrowser

String containing the executable's file name of the SEB browser application, which is specified with an entry in the *permittedProcesses* dictionary (*nameWin* key). To be started automatically, it also needs to be added to the array *autostartApplications*. If you don't want to use any browser in SEB, because SEB Starter should only act as a kiosk application starting up another application in a kiosk mode (for example a virtual desktop infrastructure client), then you need to set *enableSebBrowser* = *false*.

Default value: `<string>xulrunner.exe</string>`

*Windows only*

### sebMode

Integer with a value representing one of the *sebModes* which indicate if SEB will use local settings and load the start URL or if it will try to connect to the SEB Server.

Possible values:

```
enum {
    sebModeStartURL                = 0,
    sebModeSebServer               = 1
};
typedef NSUInteger sebModes;
```

Default value: `<integer>0</integer>` (*sebModeStartURL*)

### sebConfigPurpose

Integer with a value representing one of the *sebConfigPurposes* which indicate for what the SEB settings file will be used. This option is significant only for the SEB MacOSX preferences window.

Possible values:

```
enum {
    sebConfigPurposeStartingExam    = 0,
    sebConfigPurposeConfiguringClient = 1
};
typedef NSUInteger sebConfigPurposes;
```

Default value: `<integer>0</integer>` (*sebConfigPurposeStartingExam*)

*Mac only*

### sebServerFallback

Boolean indicating if SEB should connect to the start URL and use local settings in case it cannot connect to the SEB Server.

Default value: `<false/>`

### sebServerURL

String containing the full URL (starting with http:// or https://) of a SEB Server. See key *sebMode*.

Default value: `<string></string>`

### sebServicePolicy

(replaces key *ForceWindowsService* on Windows)

Integer with a value representing one of the *sebServicePolicies* which indicate whether SEB Starter is allowed to run without the SEB Service (background process), if a warning is displayed when the service is not running or if SEB Starter is only allowed to run when the service is running.

Possible values:

```
enum {
    ignoreService                = 0,
    indicateMissingService       = 1,
    forceSebService               = 2
};
typedef NSUInteger sebServicePolicies;
```

Default value: `<integer>2</integer>` (*forceSebService*)

### sendBrowserExamKey

Boolean indicating if the *Browser Exam Key* should be send in a custom HTTP request header, combined with the URL of the HTTP request into a SHA256 hash. The exam key is generated dynamically using the application's code signature, a special check sum over all relevant settings of the SEB client and a random salt, saved in key *examKeySalt* when the settings have been created.

Default value: `<false/>`

### showMenuBar

Boolean indicating if the Mac OS X menu bar including all menus should be displayed or not. The menu bar and the menus are not at all required for using SEB and might distract examines, but from SEB MacOSX version 2.x this option allows to use menus in SEB similar to standard Mac OS X applications.

Default value: `<false/>`

*Mac only*

### showTaskBar

Boolean indicating if the SEB dock/task bar should be displayed when using third party applications. When *allowSwitchToApplications* = *false*, then the task bar is not displayed, regardless of the value of *showTaskBar*.

Default value: `<true/>`

### startURL

String containing the full URL (starting with http:// or https://) of the page to open when SEB is started. This can be for example the URL of an exam or of a exam portal page.

Default value: `<string>http://www.safexambrowser.org</string>`

### permittedProcesses

(replaces key *permittedApplications* on Windows)

Array of dictionaries containing the properties of permitted third party applications and processes which are permitted to run during an exam. Permitted applications (which have a *title* value) show up

in the application chooser, they can be used during an exam in addition to the SEB browser. Permitted processes (which don't have a *title* value) don't appear in the application chooser, but they are allowed to run in background even when **monitorProcesses** is true (they are on the whitelist).

Keys in the *permittedProcesses* dict:

- **active**  
Boolean indicating if the permitted process is active.
- **os**  
Integer with a value representing on which operating system the permitted process runs.

Possible values for the *os* key:

```
enum {  
    operatingSystemOSX          = 0,  
    operatingSystemWin          = 1,  
};  
typedef NSUInteger operatingSystems;
```

- **title**  
String of application title which is displayed in the application chooser. Background processes don't have a *title* value, because they don't need to be chosen by users.  
*Optional*
- **description**  
String containing a description of the process. This is only displayed in the SEB configuration tool, preferences window and in logs. It should explain what kind of process this is, because this might not be obvious only from the *name*.  
*Optional*
- **name**  
String of the process name (usually the file name of the executable).
- **identifier**  
String of the process identifier in reverse domain notation, for example <string>com.apple.mail</string>.  
*Optional. Mac only*
- **autostart**  
(replaces root level key *AutostartProcess* on Windows)  
Boolean indicating whether the process is started automatically together with SEB. Usually, the SEB browser component is started automatically.  
Default value: <false/>
- **hideProcess**  
Boolean indicating whether a process gets hidden if it shows its menu bar, an alert, dialogue or other window (this means in OS X the process tries to become „active“). Usually both permitted applications and processes are allowed to display windows and become active, with this flag set to true permitted processes are only allowed to run in background (when **monitorProcesses** is true), but not to display any user interface elements.  
Default value: <true/>  
*Optional. Currently Mac only*



- **path**  
String of path to the application executable's directory (excluding the file name, see key *name*). If the path is not given or relative (not absolute from the root directory/drive), then SEB searches the current and system provided paths for applications, see flag *allowUserToChooseApp*.  
*Optional*

- **allowUserToChooseApp**  
Boolean indicating if the user is presented a requester/dialog window allowing to choose the third party application if it cannot be found at the paths specified (instead of just displaying an error message). Only applications matching the other criteria specified in the *permittedProcesses* dictionary (like *name*, *identifier*, *signature*) are accepted.  
Default value: `<false/>`  
*Optional*

- **arguments**  
Array of dictionaries containing the arguments to append to the *path+name* of the application/process.

Keys in the *arguments* dict:

- **active**  
Boolean indicating if the argument will be used/appended to the *path+name* of the executable (meant for testing).
- **argument**  
String representing one argument to append to the *path+name* of the application/process executable.

*Optional*

- **signatures**  
Array of dictionaries containing metadata and the actual signatures of the application/process executable. Used to identify the process binary/application securely. Multiple signatures are possible to identify several versions of a binary. The signature algorithm is platform specific and has yet to be defined.

Key in the *signatures* dict:

- **description**  
String containing a description of the signature. For example it should be mentioned which application/binary version is covered by the signature.
- **relativePath**  
By default the signature is calculated over the executable file of the process (identified by the key *name*). If you want the signature be calculated using for example the directory the executable lies in or you want an additional signature of another binary or a subdirectory to be used to identify the application this process belongs to, then specify a relative path (starting from the directory the process executable *name* lies in).
- **signature**  
Data (Base64 encoded) representing the signature of the binary/directory/bundle .

*Optional.*

Example:

```
<key>permittedProcesses</key>
<array>
  <dict>
    <key>title</key>
    <string>SEB</string>
    <key>nameWin</key>
    <string>xulrunner.exe</string>
    <key>pathWin</key>
    <string>../xulrunner/</string>
    <key>arguments</key>
    <array>
      <string>-app "../xul_seb\seb.ini"</string>
      <string>-profile "%LOCALAPPDATA%\ETH Zuerich\xul_seb\Profiles"</string>
    </array>
  </dict>
  <dict>
    <key>title</key>
    <string>Calculator</string>
    <key>nameWin</key>
    <string>calc.exe</string>
  </dict>
  <dict>
    <key>title</key>
    <string>Notepad</string>
    <key>nameWin</key>
    <string>notepad.exe</string>
  </dict>
</array>
```

Default value: **Important:** When SEB Starter on Windows loads a .seb settings file, it should not only check if there is a value (array) for the *permittedProcesses* key, but also if there is an entry (array item) for the standard process for XULRunner. If not, it has to save this default entry representing *xulrunner.exe* inside the *permittedProcesses* array. The reason for this is that a .seb file saved by SEB on Mac OS X might contain some permitted processes (so the key/value exists) but there won't be any *xulrunner.exe* process saved.

```
<array>
  <dict>
    <key>title</key>
    <string>SEB</string>
    <key>nameWin</key>
    <string>xulrunner.exe</string>
    <key>pathWin</key>
    <string>../xulrunner/</string>
    <key>arguments</key>
    <array>
      <string>-app "../xul_seb\seb.ini"</string>
    </array>
  </dict>
</array>
```

## prohibitedProcesses

Array of dictionaries which contain the properties of processes which are prohibited to run during an exam (they are on the blacklist) when **monitorProcesses** is true. This blacklist of processes makes sense because SEB on both platforms usually allows to run system processes (SEB MacOSX allows all processes and applications signed by Apple to run), but some of them might not be wanted during an exam. With *prohibitedProcesses* you can prevent some specific background processes and applications from running together with SEB. Use this with care, test if the system continues to run safely when the blacklisted processes are killed by SEB.

Keys in the *prohibitedProcesses* dict:

- **active**  
Boolean indicating if the prohibited process is active.
- **os**  
Integer with a value representing on which operating system the permitted process runs.

Possible values for the *os* key:

```
enum {  
    operatingSystemOSX          = 0,  
    operatingSystemWin          = 1,  
};  
typedef NSUInteger operatingSystems;
```

- **name**  
String of the process name (usually the file name of the executable).
- **description**  
String containing a description of the process. This is only displayed in the SEB configuration tool, preferences window and in logs. It should explain what kind of process this is, because this might not be obvious only from the *name*.  
*Optional*
- **identifier**  
String of the process identifier in reverse domain notation, for example <string>com.apple.mail</string>.  
*Optional. Mac only*
- **user**  
String with the user identifier under which this process is running. If no user is indicated, then the process is killed regardless under which user it's running. Instead of this identifier also the key *currentUser* can be used.  
*Optional*
- **currentUser**  
Boolean indicating that the prohibited process has to run under the currently logged in user (not system users). Use it instead of indicating the user identifier (*user* key).  
*Optional*

- **strongKill**

Boolean indicating whether an application (or process) may be killed in a not-nice way, what may cause data loss if the application had unsaved data in memory or was just writing to a persistent memory/drive. If this application is safe to be killed anytime, then setting this flag to true helps to avoid bothering users: If this flag is set to false and the application is running when SEB is started, then an alert/dialogue window is displayed to ask the user to quit this prohibited application together with other not permitted applications and to restart SEB afterwards (or to let SEB kill the applications risking that there could be data loss).

Setting this flag to false does not mean that processes (and applications) are not killed: Depending on the platform's capabilities, SEB first tries to terminate prohibited processes and applications nicely or asking the user to do it themselves, if this doesn't work then it kills them strongly anyways (as long as **monitorProcesses** is set to true). But setting **strongKill** will speed up this process. This flag should not be set for OS X applications which allow to be terminated nicely (they are anyways automatically terminated, without asking the user).

Default value: `<false/>`

*Optional*

- **signatures**

Array of dictionaries containing metadata and the actual signatures of the process executable.

Used to identify the process binary/application securely. Multiple signatures are possible to identify several versions of a binary. The signature algorithm is platform specific and has yet to be defined.

Keys in the *signatures* array dictionaries:

- **description**

String containing a description of the signature. For example it should be mentioned which application/binary version is covered by the signature.

- **relativePath**

By default the signature is calculated over the executable file of the process (identified by the key *name*). If you want the signature be calculated using for example the directory the executable lies in or you want an additional signature of another binary or a subdirectory to be used to identify the application this process belongs to, then specify a relative path (starting from the directory the process executable *name* lies in).

- **signature**

Data (Base64 encoded) representing the signature of the binary/directory/bundle .

*Optional.*

## **proxies**

Dictionary containing key/values of proxy settings for the exam client computers which override the system's proxy settings on the clients if *proxySettingsPolicy* is set to *useSEBProxySettings*.

**Note: The key names in the proxies dictionary are taken from the proxies dictionary in the OS X network service dictionary. That's why they follow a different notation (first letter is capital).**

Keys in the *proxies* dictionary:

- **ExceptionsList**

Array of strings containing host and domain names of network resources that should be accessed without a proxy server.

- **ExcludeSimpleHostnames**

Boolean indicating whether simple host names are excluded.

- **AutoDiscoveryEnabled**  
Boolean indicating if automatic proxy discovery is used.
- **AutoConfigurationEnabled**  
Boolean indicating if automatic proxy configuration is used.
- **AutoConfigurationJavaScript**  
String containing the full JavaScript source for the proxy autoconfiguration (PAC) file.
- **AutoConfigurationURL**  
String specifying the location of the proxy autoconfiguration (PAC) file.
- **FTPPassive**  
Boolean indicating if passive FTP mode (PASV) is used.
- **FTPEnable**  
Boolean indicating if the FTP proxy is enabled.
- **FTPPort**  
Integer representing the port number (between 1 and 65535) of the FTP proxy.
- **FTPProxy**  
String containing either the hostname or IP number of the FTP proxy host.
- **FTPRequiresPassword**  
Boolean indicating if a password (and username) is required when contacting the proxy.
- **FTPUsername**  
String containing the username to be used when contacting the proxy.
- **FTPPassword**  
String containing the plain password for this proxy server.
- **HTTPEnable**  
Boolean indicating if the FTP proxy is enabled.
- **HTTPPort**  
Integer representing the port number (between 1 and 65535) of the FTP proxy.
- **HTTPProxy**  
String containing either the hostname or IP number of the FTP proxy host.
- **HTTPRequiresPassword**  
Boolean indicating if a password (and username) is required when contacting the proxy.
- **HTTPUsername**  
String containing the username to be used when contacting the proxy.
- **HTTPPassword**  
String containing the plain password for this proxy server.
- **HTTPSEnable**  
Boolean indicating if the HTTPS proxy is enabled.
- **HTTPSPort**  
Integer representing the port number (between 1 and 65535) of the HTTPS proxy.

- **HTTPSProxy**  
String containing either the hostname or IP number of the HTTPS proxy host.
- **HTTPSRequiresPassword**  
Boolean indicating if a password (and username) is required when contacting the proxy.
- **HTTPSUsername**  
String containing the username to be used when contacting the proxy.
- **HTTSPassword**  
String containing the plain password for this proxy server.
- **RTSPEnable**  
Boolean indicating if the RTSP proxy is enabled.
- **RTSPPort**  
Integer representing the port number (between 1 and 65535) of the RTSP proxy.
- **RTSPProxy**  
String containing either the hostname or IP number of the RTSP proxy host.
- **RTSPRequiresPassword**  
Boolean indicating if a password (and username) is required when contacting the proxy.
- **RTSPUsername**  
String containing the username to be used when contacting the proxy.
- **RTSPPassword**  
String containing the plain password for this proxy server.
- **SOCKSEnable**  
Boolean indicating if the SOCKS proxy is enabled.
- **SOCKSPort**  
Integer representing the port number (between 1 and 65535) of the SOCKS proxy.
- **SOCKSProxy**  
String containing either the hostname or IP number of the SOCKS proxy host.
- **SOCKSRequiresPassword**  
Boolean indicating if a password (and username) is required when contacting the proxy.
- **SOCKSUsername**  
String containing the username to be used when contacting the proxy.
- **SOCKSPassword**  
String containing the plain password for this proxy server.

*All keys are optional.*

## urlFilterRules

Array of dictionaries each containing a set of URL filter rules.

Keys in the *urlFilterRules* array dictionaries:

- **active**  
Boolean indicating if the rule is active.
- **expression**  
String containing a description of the rule set. This is only displayed in the SEB configuration tool, preferences window and in logs. It should explain what kind of rule this is. If no description is entered, the configuration tool should generate a generic “description”/name like “rule 1”, “rule 2” etc.
- **ruleActions**  
Array containing the actual actions of the rule. Each rule action is a dictionary with the keys below.

Keys in the *ruleActions* dict:

- **active**  
Boolean indicating if the action is active.
- **regex**  
Boolean indicating if the action rule is a regular expression. If *regex* is set to *false*, then the rule is formatted using the wildcard \* (stands for an arbitrary string of any length).
- **action**  
Integer with a value representing one of the *urlFilterRuleActions*. The URL filter processes the actions and expressions inside each rule one by one. If the URL doesn't match one expression, then the next expression is processed. The actions described below are used to decide what to do if the URL matches the expression.  
Possible actions:
  - **block**  
If the URL matches the expression, then it is rejected and processing of following actions inside this rule and processing of rules following the current one is stopped.
  - **allow**  
If the URL matches the expression, then it is accepted and processing of following actions inside this rule and processing of rules following the current one is stopped.
  - **skip**  
If the URL matches the expression, then processing of the remaining actions inside this rule is stopped and the URL filter continues processing the next rule following the current one.
  - **and**  
If the URL matches the expression, then the URL filter continues to process the next action inside this rule. If the URL doesn't match the expression, then the URL filter skips the remaining actions in this rule and continues to process the next rule. Like this you can realize a logical AND dependency of two or more expressions.
  - **or**  
If the URL matches the expression, then the URL filter executes the next action inside this rule which is not another *or* action (the expression of that not-*or* action following the *or*-action is not evaluated, the action is processed regardless of that expression, because the expression of the preceding *or*-action already matched) If the URL doesn't match the expression, then the next action is processed. Like this you can realize a logical OR dependency of two or more expressions.

If the URL filter reaches the end of the last rule (means there was no matching *block* or *allow* expression found), then the URL is discarded. If you want the URL to be accepted if no matching *block* (or *allow*) expression was found, then add an *<allow \*>* action/expression as the last rule.

Possible values for the *action* key (in the user interface only the actual action name is shown, see above):

```
enum {
    urlFilterActionBlock           = 0,
    urlFilterActionAllow          = 1,
    urlFilterActionSkip           = 2,
    urlFilterActionAnd            = 3,
    urlFilterActionOr             = 4
};
typedef NSUInteger urlFilterRuleActions;
```

- **expression**

String containing the filtering expression or pattern, either in the regular expression format (*regex = true*) or (*regex = false*) a simpler filter expression containing the wildcard char *<\*>*. A filter expression can filter against all elements of a URL/URI according to [RFC 3986](#):

`scheme://host:port/path?query_string#fragment_id`

Format for a filter expression in the non-regex format:

- Scheme can be http or https. This field is optional, and must be followed by '://'.
- The host field is required, and is either a partial or full hostname or an IP address. It can also contain or be replaced completely with the wildcard '\*' char, see below for details. The URL filter doesn't resolve hostnames itself, so if you allow everything and only block *<hostname.com>*, then that host could still be reached using its IP address. You should therefore mainly use whitelisting to allow accessing only specific sites during an exam.
- An optional '.' (dot) can prefix the host field to disable subdomain matching, see below for details.
- An optional port can come after the host and always has to start with the character ':'. It must be a valid port value from 1 to 65535.
- An optional path can come after the host or after the port and always has to start with the character '/'. Parts of the path can be replaced with the wildcard char '\*'.
- URL parameters like a query string can be indicated and always have to start with the character '?'. Parts of the query can be replaced with the wildcard char '\*'.

Examples for filter expressions:

- *<example.com>* matches *<example.com>*, *<www.example.com>* and *<www.mail.example.com>* (internally processed as a host name search for *example.com* and a search for *\*.example.com*)
- *<.www.example.com>* matches exactly *<\*.://www.example.com>* (no other subdomains)
- *<mail.\*>* matches all hosts having a subdomain or domain 'mail', like *<mail.ethz.ch>*, *<www.mail.gov.to>*, *<mail.com>*
- *<\*:8088>* matches all requests to port 8088
- *<example.com/stuff/\*>* matches all requests to any subdomain of *<example.com>* that have *<stuff>* as the first segment of the path
- *<example.com/images\*.png>* matches all requests to any subdomain of *<example.com>* that have *</images>* as the first segment of the path and *<.png>* as the path extension of an file URL (means it matches all PNG images in the *</images>* directory or its subdirectories)
- *<\*.net>* matches all host with any kind of subdomains in the .net top level domain like *<example.net>*, *<www.example.net>*, *<www.mail.example.net>*
- *<\*/\*.net>* matches all files with a *<.net>* file extension on any host

You should consider using a regular expression if performing complex filtering (when a simple filter doesn't cover all possible cases how that complex URL could be formatted) or split the expression and combine the expression parts using the *and* action.



Don't forget that the order of the filter actions and rule sets is very much relevant. If you would put rule 2 first in the example below, then .seb files would also be allowed from the host <maps.google.\*>.

Example:

The urlFilterRules which are shown as key/values in the XML format below look (similar to) as follows in the UI of the preferences/settings tool:

- **Rule 1: allow .seb files only from safeexambrowser.org and https://exams.ethz.ch**

• safeexambrowser.org	or
• https://exams.ethz.ch	skip
• */*.seb	block
- **Rule 2: allow maps.google.\*, block google.\* and all its other subdomains**

• maps.google.*	allow
• google.*	block
- **Rule 3: allow \***

• *	allow
-----	-------

```

<key>urlFilterRules</key>
<array>
  <dict>
    <key>active</key>
    <true/>
    <key>description</key>
    <string>Rule 1: allow .seb files only from safeexambrowser.org
and https://exams.ethz.ch</string>
    <key>rules</key>
    <array>
      <dict>
        <key>active</key>
        <true/>
        <key>regex</key>
        <false/>
        <key>action</key>
        <integer>4</integer>
        <key>expression</key>
        <string>safeexambrowser.org</string>
      </dict>
      <dict>
        <key>active</key>
        <true/>
        <key>regex</key>
        <false/>
        <key>action</key>
        <integer>2</integer>
        <key>expression</key>
        <string>https://exams.ethz.ch</string>
      </dict>
      <dict>
        <key>active</key>
        <true/>
        <key>regex</key>
        <false/>
        <key>action</key>
        <integer>0</integer>
        <key>expression</key>
        <string>*/*.seb</string>
      </dict>
    </array>
  </dict>
</array>

```

```

        </dict>
      </array>
    </dict>
    <dict>
      <key>active</key>
      <true/>
      <key>description</key>
      <string>Rule 2: allow maps.google.*, block google.* and all its
other subdomains</string>
      <key>rules</key>
      <array>
        <dict>
          <key>active</key>
          <true/>
          <key>regex</key>
          <false/>
          <key>action</key>
          <integer>1</integer>
          <key>expression</key>
          <string>maps.google.*</string>
        </dict>
        <dict>
          <key>active</key>
          <true/>
          <key>regex</key>
          <false/>
          <key>action</key>
          <integer>0</integer>
          <key>expression</key>
          <string>google.*</string>
        </dict>
      </array>
    </dict>
    <dict>
      <key>active</key>
      <true/>
      <key>description</key>
      <string>Rule 3: allow *</string>
      <key>rules</key>
      <array>
        <dict>
          <key>active</key>
          <true/>
          <key>regex</key>
          <false/>
          <key>action</key>
          <integer>1</integer>
          <key>expression</key>
          <string>*</string>
        </dict>
      </array>
    </dict>
  </array>

```

## 7. Changes

Version 6, February 25, 2013

### Added keys:

- **allowUserSwitching**: Mac only.
- **browserViewMode**: Currently Mac only
- **copyExamKeyToClipboardWhenQuitting**: Mac only, significant only for the SEB MacOSX preferences window
- **enableBrowserWindowToolbar**: Currently Mac only
- **enableSebBrowser**: Added key, changes detail behavior of key *sebBrowser* (no empty string is necessary to disable the SEB browser).
- **enableUrlContentFilter**: Feature will need new code in XULRunner. In the current implementation parameter "seb.trusted.content" would have to be set the in XULRunner's [config.json](#)
- **enableUrlFilter**: Feature will need new code in XULRunner
- **hideBrowserWindowToolbar**: Mac only
- **logDirectoryOSX**: Mac only
- **mainBrowserWindowWidth**: Currently Mac only, would require new feature/code in XULRunner.
- **mainBrowserWindowHeight**: Currently Mac only, would require new feature/code in XULRunner.
- **newBrowserWindowByLinkHeight**: In SEB Windows this value will have to set the parameter "seb.openwin.height" in XULRunner's *config.json*, see <http://www.eqsoft.org/sebian/doku/doku.php?id=config>
- **newBrowserWindowByLinkWidth**: In SEB Windows this value will have to set the parameter "seb.openwin.width" in XULRunner's *config.json*
- **sebPurpose**: Mac only, significant only for the SEB MacOSX preferences window
- **showMenuBar**: Mac only
- **showTaskBar**
- **sendBrowserExamKey**: Implementation of generating Browser Exam Key not yet specified, feature will need code in XULRunner.
- **urlFilterRules**: Might not be final version, should not yet be implemented

### Changed keys:

- **autostartProcesses** removed, replaced with key **permittedProcesses -> autostart**  
Reason: The array with all processes to be started automatically when SEB is started meant unnecessary overhead, it's easier to use the new boolean key *autostart* in the *permittedProcesses* dictionary.
- **prohibitedProcesses -> name** removed/changed to keys **nameWin / nameOSX** to distinguish between Windows and Mac OS X processes/executables.
- **sebBrowser**: Since boolean key *enableSebBrowser* was added to explicitly enable or disable the SEB browser, no empty string in *sebBrowser* is necessary to disable the SEB browser.

Version 7, February 27, 2013

### Added keys:

- **permittedProcesses -> active**
- **prohibitedProcesses -> active**

### Changes:

- **ruleActions** -> or improved explanation of the action, clarified what happens when the URL filter processes an or action (no change in function).

Version 8, April 30, 2013

### Changes:

- **urlFilterRules** added a human readable version to the example of a rule set. Added a missing `<allow *>` rule/action as the last rule so that the example would work as mentioned in the rule descriptions (no change in function).
- Changes in explanations and examples for the **urlFilterRules** dictionary.
- Removed the screenshot of a example for a simple configuration interface for keys not available on a platform. This future feature has been discarded. SEB Server will offer a configuration UI for all key/values on all platforms and the option to merge .seb files, so the resulting file would contain all key/values for all platforms.
- The *plist* headers are now contained in the plain *plist* XML data.

### Added keys:

- **permittedProcesses** -> **os**
- **prohibitedProcesses** -> **os**
- **embeddedCertificates**
- **mainBrowserWindowPositioning**
- **newBrowserWindowByLinkPositioning**
- **downloadDirectoryWin**
- **logDirectoryWin**
- **hookedMessages** dictionary containing all hooked keys and messages (which were single keys before).
- **hookedMessages** -> **enableStartMenu**
- **proxySettingsPolicies**
- **sebMode**
- **sebServerURL**
- **sebServerFallback**
- **proxies**

### Changed keys:

- **permittedProcesses** -> **nameWin / nameOSX** removed/changed to **name**. Now the *os* key is used to distinguish between Windows and Mac OS X processes/executables.
- **permittedProcesses** -> **pathWin / pathOSX** removed/changed to **path**. Now the *os* key is used to distinguish between Windows and Mac OS X processes/executables.
- **permittedProcesses** -> **signatureWin / signatureOSX** removed/changed to **signature**. Now the *os* key is used to distinguish between Windows and Mac OS X processes/executables.
- **permittedProcesses** -> **signature** changed to **signatures**. Changed from data to an array of dictionaries containing metadata and the actual signatures of the process executable. Keys in the dict: **description**, **relativePath**, **signature**
- **permittedProcesses** -> **arguments** changed from array to array of dictionaries with keys **active** and **argument**.
- **prohibitedProcesses** -> **nameWin / nameOSX** removed/changed to **name**. Now the *os* key is used to distinguish between Windows and Mac OS X processes/executables.
- **prohibitedProcesses** -> **signatureWin / signatureOSX** removed/changed to **signature**. Now the *os* key is used to distinguish between Windows and Mac OS X processes/executables.

- **prohibitedProcesses** -> **signature** changed to **signatures**. Changed from data to an array of dictionaries containing metadata and the actual signatures of the process executable. Keys in the dict: **description**, **relativePath**, **signature**
- **copyExamKeyToClipboardWhenQuitting** renamed to **copyBrowserExamKeyToClipboardWhenQuitting** to keep consistent nomenclature.
- **urlFilterRules** -> **description** changed to **urlFilterRules** -> **expression**, to facilitate implementation of the table (NSOutlineView on OS X) displaying the filter rules. The string still contains a description of the URL filter rule set.
- **sebPurpose** changed to **sebConfigPurpose**, represented value constants changed to *sebConfigPurposes*
- **enableBrowsingBackForward** changed to **allowBrowsingBackForward**.
- **enableLog** changed to **enableLogging**.
- **enablePlugins** changed to **enablePlugIns**. (capital 'I').