SafeAPI - v1.20
Documentation Utilisateur
27 janvier 2005

SafeLogic S.A.R.L au capital de € 30 489
R.C.S Paris B 422 461 848
27-29, rue Raffet - 75016 Paris
Tél : (33) (0)1 45 72 25 15 - Fax : (33) (0)1 45 72 14 06 - Web : www.safelogic.com

Table des Matières

1 INTRODUCTION

1.1 Contenu du document

Ce document constitue la documentation d'utilisation de l'API de Cryptographie SafeAPI sous Windows NT/2000/2003/XP et Unix

Sous Windows, il traite de l'utilisation de cette bibliothèque via une DLL avec Visual Basic comme langage d'illustration :

- Paramètrage des projets Visual Basic 6.0.
- Description des API : fonctions, arguments, retour et codes erreur.

1.2 Destinataires du document

Ce document s'adresse aux développeurs Java (ou Visual Basic, C/C++, Delphi, PowerBuilder, etc.) qui souhaitent incorporer des appels aux API de Cryptographie de SafeAPI. Les API de Cryptographie SafeAPI ont pour but de faciliter considérablement le développement d'applications qui utilisent des fonctions de hash, signature et chiffrement symétrique et asymétrique.

Une connaissance des notions de cryptographie de base est néanmoins indispensable pour la lecture de ce document.

1.3 Environnement technique de fonctionnement

Le noyau SafeAPI est écrit en 100 % Java et fonctionne de façon identique sous Windows NT/2000/XP, Linux et tous les Unix qui supportent Java et JDBC.

Voici les environnements supportés dans cette version :

Environnement technique	Versions supportées
Windows NT	• Windows NT Server V 4.0 SP 4 ou supérieure. • Windows NT Workstation V 4.0 SP 4 ou supérieure.
Windows 2000/XP	• Windows 2000 Server SP 2 ou supérieure. • Windows 2000 Professionnal SP 2 ou supérieure. • Windows XP Professionnal SP 1 ou supérieure.
Unix	• Sun Solaris 7.x/8.x/9.x (SPARC et Intel). • Linux (Intel) RedHat 6.1/7.1/8.0/9.0. • Autres Unix qui supportent Java.
JVM (Java Virtual Machine)	• JDK 1.4.1 de Sun Microsystems ou supérieur.

L'API est supportée pour toutes les versions égales ou supérieures de ces environnements Java.

2 INSTALLATION DE SAFEAPI

2.1 Installation du JDK 1.4 de Sun Microsystems

SafeAPI nécessite l'installation du JDK 1.4 de Sun Microsystems.

Le JDK pour Windows/Unix/Linux est disponible en téléchargement sur le site de Sun à l'adresse :
http://java.sun.com/j2se.

2.2 Installation des fichiers binaires de SafeAPI

2.2.1 Téléchargement de SafeAPI

Téléchargez le fichier `safeapi_v1.20.zip` à l'adresse :
http://www.safelogic.com/safeapi/download/safeapi_v1.20.zip

2.2.2 Installation standard

Par défaut nous suggérons les répertoires d'installation suivants :

Windows NT/2000/XP	Dézipper le contenu du fichier `safeapi_v1.20.zip` dans **`c:\.`** Cela va créer l'arborescence **`c:\safelogic`**
Unix/Linux	Créer un user safelogic et dézipper le contenu du fichier `safeapi_v1.20.zip` dans **`/home`**. Cela va créer l'arborescence **`/home/safelogic.`**

On obtient l'arborescence Windows :

Nom répertoire Windows (par défaut)	Commentaires
`c:\safelogic\lib`	Bibliothèques Java communes de SafeLogic.
`c:\safelogic\safeapi`	Répertoire de base de SafeAPI.
`c:\safelogic\safeapi\bin`	Scripts de lancement de SafeAPI.
`c:\safelogic\safeapi\conf`	Fichiers divers d'aide à la configuration de SafeAPI. (Classpath).
`c:\safelogic\safeapi\doc`	Documentation d'installation et d'utilisation.
`c:\safelogic\safeapi\examples`	Programmes d'exemples en Java.
`c:\safelogic\safeapi\keys`	Stockage des clefs SafeAPI et de la license.
`c:\safelogic\safeapi\lib`	Bibliothèques Java spécifiques de SafeAPI. Inclut les DLL.

On obtient l'arborescence Unix/Linux :

Nom répertoire Unix (par défaut)	Commentaires
`/home/safelogic/lib`	Bibliothèques Java communes de SafeLogic.
`/home/safelogic/safeapi`	Répertoire de base de SafeAPI
`/home/safelogic/safeapi/bin`	Scripts de lancement de SafeAPI
`/home/safelogic/safeapi/conf`	Fichiers divers d'aide à la configuration de SafeAPI. (Classpath).
`/home/safelogic/safeapi/doc`	Documentation d'installation et d'utilisation
`/home/safelogic/safeapi/examples`	Programmes d'exemples en Java
`/home/safelogic/safeapi/keys`	Stockage des clefs SafeAPI et de la license.
`/home/safelogic/safeapi/lib`	Bibliothèques Java spécifiques de SafeAPI.

2.2.3 Installation personnalisée

SafeAPI peut être installé librement dans un répertoire racine différent de `/home/safelogic` ou `c:\safelogic.`

Il faut et il suffit simplement de respecter les contraintes de CLASSPATH décrites ci-dessous.

2.3 FICHIER DE LICENCE

SafeAPI utilise un fichier de licence `safeapi_license.txt` - **fourni séparément et non inclus dans `safeapi_v1.20.zip`** - qui comporte plusieurs lignes avec dans l'ordre :

- La chaîne « safeapi ».
- Un code de catégorie suivant le type de sous-catégorie du produit.
- La liste des nom de serveurs (HOSTNAME) d'installation, séparés par des « ; ».
- Une date de fin d'évaluation de la version d'évaluation ou 9999999 pour les versions achetées.
- Un email d'identification de l'utilisateur.
- Une chaîne hexadécimale qui correspond à la signature des éléments précédents avec une clé privée RSA.

Par défaut, `safeapi_license.txt` s'installe par défaut dans le sous répertoire `keys`. (`c:\safelogic\safeapi\keys` ou `/home/safelogic/keys`)

En cas d'installation personnalisée, la localisation du fichier doit répondre aux deux contraintes :

- `safeapi_license.txt` doit être situé dans un répertoire défini/déclaré dans la variable d'environnement CLASSPATH.
- `safeapi_license.txt` doit être situé dans le même répertoire que le fichier clé `license@safelogic.com_1_RSA.pkf`. Ce fichier est aussi localisé par défaut dans le répertoire `keys` .

2.4 Mise à jour CLASSPATH Java

La variable d'environnement Java CLASSPATH doit contenir :

1. Le chemin d'accès aux deux librairies `cryptix.jar` et `safeapix.jar`.
2. Le chemin d'accès au répertoire `keys` qui contient `safeapi_license.txt` et le fichier clé `license@safelogic.com_1_RSA.pkf`.
3. Le chemin d'accès au sous répertoire `examples` qui contient les exemples Java.

2.4.1 CLASSPATH sous Windows NT/2000/XP

Pour l'installation standard, ajouter la chaîne suivante au CLASSPATH :

```
c:\safelogic\lib\cryptix.jar;c:\safelogic\safeapi\lib\safeapix.jar;
c:\safelogic\safeapi\keys;c:\safelogic\safeapi\examples\java;
```

2.4.2 CLASSPATH sous Unix/Linux

Pour l'installation l'installation standard, ajouter la chaîne suivante au CLASSPATH :

```
/home/safelogic/lib/cryptix.jar;/home/safelogic/safeapi/lib/safeapix.jar;
/home/safelogic/safeapi/keys;/home/safelogic/safeapi/examples/java;
```

2.5 Installation des DLL (Windows)

Sous Windows, SafeAPI peut être utilisée sous forme de DLL à partir de nombreux langages de programmation (C/C++, Visual Basic, Delphi, etc.).

Copiez les deux DLL `safejnidll.dll & safeapix.dll` du répertoire `c:\safelogic\safeapi\lib` dans un dossier « système » du disque dur.
Exemple : `c:\WINDOWS\system32.`

Enregistrement de la DLL safeapix.dll

safeapix.dll doit être enregistrée comme suit:

- Ouvrez une session MS-DOS locale dans le répertoire d'installation de safeapix.dll.
- Passez la commande `regsvr32` sur la dll.

Exemple :

```
C:\WINDOWS\System32>regsvr32 safeapix.dll
```

2.6 Vérifications et tests de l'installation

Ouvrir une session ligne de commande et lancer le programme Java d'exemple `TestHash` avec comme paramètre la chaîne "abc":

Windows :

```
C:\>java com.safeapi.test.TestHash "abc"
```

Unix/Linux :

```
$ java com.safeapi.test.TestHash "abc"
```

Le programme renvoie comme résultat à l'écran les deux valeurs MD5 et SHA-1 de « abc » :

```
MD5: 900150983CD24FB0D6963F7D28E17F72
SHA-1: A9993E364706816ABA3E25717850C26C9CD0D89D
```

2.7 Documentation Javadoc

La documentation Javadoc de toutes les API SafeAPI est contenue dans le répertoire **`/safelogic/safeapi/javadoc`** du fichier `safeapi_v1.20.zip` .

2.8 Documentation Javadoc en ligne

La documentation Javadoc des APIs est aussi accessible en ligne à l'adresse : http://www.safelogic.com/safeapi/v1.20/javadoc.

3 PARAMETRAGE DE L'ENVIRONNEMENT TECHNIQUE SOUS WINDOWS

3.1 Généralités

SafeAPI se présente sous la forme de fonctions appelables à partir de Java et de tout environnement de développement qui supporte l'utilisation de DLL. Se référer à la documentation technique de votre environnement de développement pour pouvoir appeler des fonctions d'une DLL.

Nous avons choisi un paramètrage sous Microsoft Visual Basic 6.0 à titre d'exemple et pour décrire les mécanismes généraux.

3.2 Exemple : Microsoft Visual Basic 6.0

Dans l'environnement Visual Basic, l'utilisation de l'API de cryptographie nécessite d' « importer » les composants de la DLL. Ceci se fait en incluant la DLL **safeapix.dll** comme référence du projet.

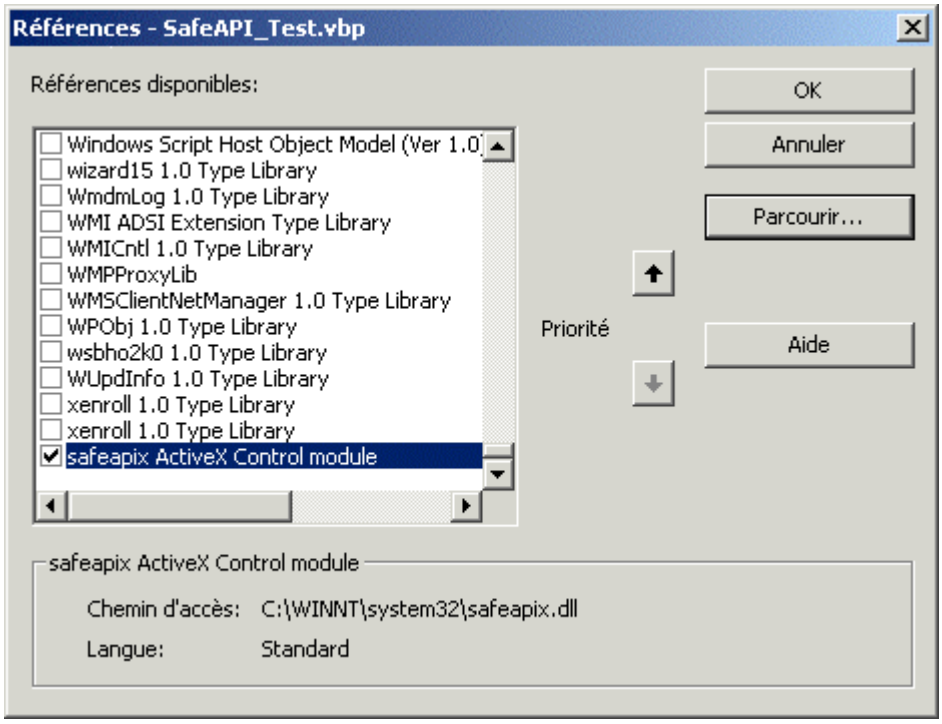
Rappelons rapidement la démarche à suivre pour cela :

- Créer un nouveau Projet VB.

- Menu **Projet** - sous-menu **Références**. Dans la liste des Références disponibles, **safeapix** doit apparaître. Cocher la checkbox puis valider le choix en cliquant le bouton OK.

Si la DLL n'apparaît pas spontanément dans liste **références disponibles**, il faut la récupérer sur l'ordinateur avec le bouton **Parcourir** (Cf. image ci-dessous).

Le résultat final de cette opération ressemble à la capture suivante :

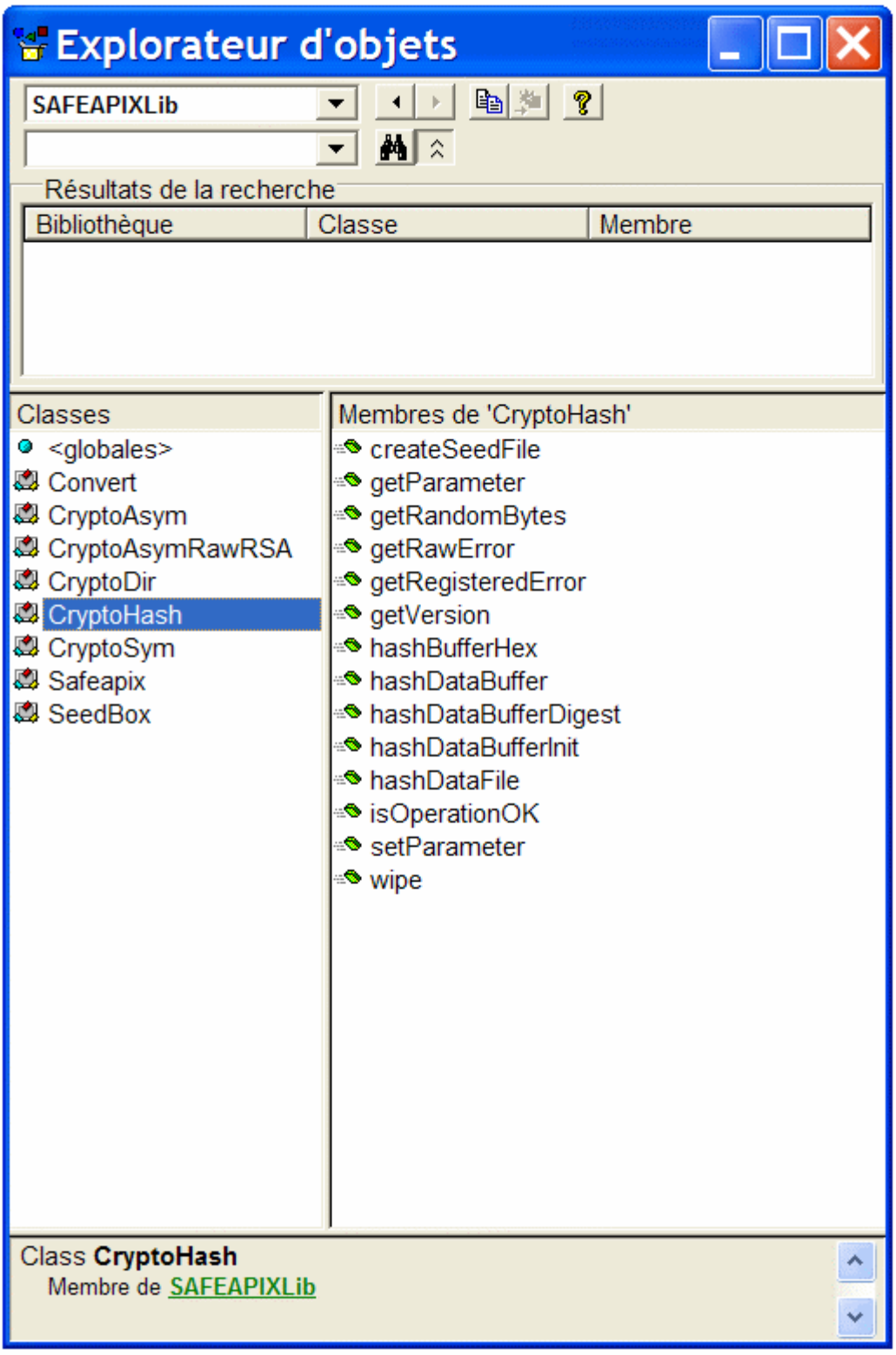


Vous pouvez vérifier que la DLL est bien disponible en ouvrant l'explorateur d'objet.

Celui-ci est disponible en tant qu'icône de la barre d'outils, ou par le raccourci clavier **F2**, ou encore par le menu **Affichage**.

Si le composant est correctement intégrée au projet vous devez pouvoir sélectionner **safeapix** dans la liste des bibliothèques.

Voici ce qu'affiche l'explorateur d'objet pour cette librairie :


Explorateur d'objets
SAFEAPIXLib
Résultats de la recherche
Bibliothèque
Classe
Membre
Classes
<globales>
Convert
CryptoAsym
CryptoAsymRawRSA
CryptoDir
CryptoHash
CryptoSym
Safeapix
SeedBox
Membres de 'CryptoHash'
createSeedFile
getParameter
getRandomBytes
getRawError
getRegisteredError
getVersion
hashBufferHex
hashDataBuffer
hashDataBufferDigest
hashDataBufferInit
hashDataFile
isOperationOK
setParameter
wipe
Class CryptoHash
Membre de SAFEAPIXLib

4 UTILISATION DE LA LIBRAIRIE SAFEAPI

4.1 Notion de Classe

SafeAPI est composée de **classes**, suivant la terminologie Java et/ou C++, qui doivent être **instanciées** avant leur utilisation. Le principe d'utilisation est équivalent sous tous les langages qui appellent SafeAPI sous forme de DLL.

Chaque classe regroupe des **méthodes (ou fonctions).**

Par exemple, la **classe** `CryptoHash` permet d'effectuer des opérations de hachage sur des fichiers avec La **méthode** `hashDataFile.` `hashDataFile` retourne un tableau de bytes.

Voici comment calculer la valeur de hachage MD5 d'un fichier sous plusieurs langages :

Java

```
    // Loads an instance of CryptoHash
    CryptoHash cryHash = new CryptoHash();

    // Compute the Hash value of safeapix.dll with MD5 algorithm
    byte[] hash = cryHash.hashDataFile("MD5",
                          "c:\\windows\\system32\\safeapix.dll");
    // Etc...
```

Visual C++ 6.0

```
    // Loads an instance of CryptoHash
    ICryptoHash cryHash;

    if (!cryHash.CreateDispatch("safeapix.CryptoHash"))
        AfxMessageBox("Error! SafeAPI CryptoHash  not loaded!" ,0 ,0);

    // Compute the Hash value of safeapix.dll with MD5 algorithm
    COleVariant hash = cryHash.hashDataFile("MD5",
                              "c:\\windows\\system32\\safeapix.dll");
    // Etc...
```

Visual Basic 6.0

```
    ' Loads an instance of CryptoHash
    Dim cryHash As New CryptoHash

    ' Compute the Hash value of safeapix.dll with MD5 algorithm
    Dim hash As Variant
    hash = cryHash.hashDataFile("MD5",
                  "c:\\windows\\system32\\safeapix.dll")
    ' Etc...
```

4.2 Liste des Classes de SafeAPI

Le tableau suivant détaille les classes de SafeAPI par ordre alphabétique :

Classe	Méthodes
Convert	Méthodes de conversion de buffers.
CryptoAsym	Méthodes de chiffrement asymétrique (RSA) de buffers et de fichiers.
CryptoAsymRawRSA	Méthodes pour le chiffrement asymétriques de buffers avec des clés « natives » RSA.
CryptoCommon	*Classe non instanciable et invisible dans la DLL.* *Regroupe les méthodes communes aux classes de cryptographie :* *CryptoAsym, CryptoAsymRawRSA, CryptoDir, CryptoHash, CryptoSym*
CryptoDir	Méthodes de chiffrement symétrique et asymétrique de répertoires.
CryptoHash	Méthodes de hachage de buffers et avec MD5 et SHA-1. Méthodes de hachage avec MD5 et SHA-1.
CryptoSym	Méthodes de chiffrement symétrique.
SeedBox	Méthodes de génération de valeurs aléatoires (« seed ») pour les interfaces graphiques.
Status	*Classe non instanciable et composée uniquement de propriétés.* Contient les valeurs de Code Retour des méthodes sous forme de chaînes de caractères (String).

4.3 Liste des Classes par licence SafeAPI

Les classes accessibles de SafeAPI dépendent de la licence choisie et/ou achetée.

Licence SafeAPI	Classes accessibles
Licence Evaluation	Toutes les classes (avec limitation dans le temps).
Licence Hash	Convert, **CryptoHash,** Status
Licence Chiffrement Symétrique	*Classes de Licence Hash,* **CryptoSym, SeedBox**
Licence Chiffrement Asymétrique	*Classes de Licence Chiffrement Symétrique,* **CryptoAsym, CryptoAsymRawRSA**
Licence Chiffrement Répertoires	Toutes les classes : *Classes de Licence Chiffrement Asymétrique,* **CryptoDir**

5 MANUEL DE REFERENCE DES CLASSES

5.1 Classe Convert : méthodes de conversion de buffers

Convert est une classe d'utilitaires de conversion de type destinée à faciliter l'utilisation des autres classes de l'API :

- Conversion de bytes en chaîne.
- Conversion de bytes en valeur hexadécimale.

Méthode	**bytesToHexString**		
Description	Convertit une valeur binaire (buffer de bytes) en format de chaîne de caractères hexadécimale.		
Arguments	byte[ ]	bBuffer	Buffer en bytes.
Retour	String		Conversion du buffer en chaîne de caractères hexadécimale.

Méthode	**bytesToString**		
Description	Convertit un buffer de bytes en String (chaîne de caractères).		
Arguments	byte[ ]	bBuffer	Buffer en bytes.
Retour	String		Conversion du buffer en chaîne de caractères.

Méthode	**hexStringToBytes**		
Description	Convertit une String *avec une représentation hexadécimale* en byte[ ].		
Arguments	String	sHex	Représentation hexadécimale de la chaîne.
Retour	byte[ ]		Conversion de la chaîne hexadécimale en bytes.

Méthode	**stringToBytes**		
Description	Convertit une String en buffer de bytes.		
Arguments	String	sString	Chaîne de caractères.
Retour	byte[ ]		Conversion de la chaîne de caractères en bytes.

5.2 Classe SeedBox : méthodes visuelles de génération de seed

Classe qui permet la génération de « seed » de façon aléatoire à l'aide d'une boîte de dialogue demandant à l'utilisateur d'effectuer une saisie clavier.

La création de valeur de « seed » sert à initialiser un générateur de nombre pseudo-aléatoires. En particulier, les valeurs « seed » sont très utilisées pour être certain de générer des clés symétriques ou asymétriques dont les valeurs ne peuvent être détectées par un attaquant.

Méthode	**seedDialog**		
Description	Génère un seed aléatoire de la longueur désirée en ouvrant une fenêtre de dialogue demandant une saisie au clavier. Retourne ce seed.		
Arguments	String	sTitle	Titre de la fenêtre de dialogue [facultatif]
	String	sCaption	Texte de la boîte de dialogue [facultatif]
	int	lSeedSize	Longueur du seed (en bits)
Retour	byte[ ]		Seed sous forme de suite d'octets

Méthode	**getSeedValue**	
Description	Renvoie la valeur du seed généré.	
Arguments	Aucun	
Retour	byte[ ]	Seed sous forme de suite d'octets

Méthode	**seedCanceledByUser**	
Description	Indique si l'utilisateur a annulé la génération du seed (fermeture de la fenêtre de dialogue).	
Arguments	Aucun	
Retour	Boolean	True : l'utilisateur a fermé la fenêtre lui-même avant la fin False : la génération du seed s'est déroulée entièrement

Un exemple d'utilisation de cette classe en VB est la méthode **bt_GenSeed_Click** dans l'exemple d'implémentation. Cette méthode se trouve dans le fichier Visual Basic **Form_GenKeyTest.frm** qui correspond à la fenêtre GenKeyTest.

5.3 Classe CryptoCommon : méthodes communes aux classes CryptoXxxx

CryptoCommon est une *classe non instanciable et invisible dans la DLL*.

Elles regroupe les méthodes communes aux classes de cryptographie :

- **CryptoAsym,**
- **CryptoAsymRawRSA,**
- **CryptoDir,**
- **CryptoHash,**
- **CryptoSym.**

Les méthodes de CryptoCommon sont accessibles et visibles via ces classes.

5.3.1 CryptoCommon : Méthodes de Paramétrage dynamique de l'API

Les méthodes de paramètrage dynamiques sont utilisées pour spécifier l'environnement système d'utilisation des fonctions de cryptographie directement depuis votre environnement de développement.

Cette implémentation permet d'éviter les paramètrages en statique et/ou les paramètrages sous forme de fichier ".ini" qui peuvent être sources d'erreurs et d'attaques.

Le paramétrage concerne les données relatives au :

- Répertoire de stockage des clefs.
- Au fichier de rapport d'erreur (fichier dump).

Méthode	**setParameter**		
Description	Permet de changer la valeur des paramètres de l'API.		
Arguments	String	ParamName	Nom du paramètre à modifier (Cf. Liste ci-dessous)
	String	Value	Nouvelle valeur du paramètre indiqué.
Retour	Boolean		True : le changement a été réalisé False : le changement n'a pas été réalisé

Méthode	**getParameter**		
Description	Permet de récupérer la valeur courante des paramètres de l'API.		
Arguments	String	ParamName	Nom du paramètre (Cf. Liste ci-dessous)
Retour	String	La valeur du paramètre ParamName.	

5.3.2 Liste des paramètres

Paramètre	Rôle
DEFAULT_SIGN_ALGO	Algorithme de signature utilisé par défaut
DEFAULT_SYM_ALGO	Algorithme symétrique utilisé par défaut en cryptographie asymétrique et « mixte ».
DUMP_ENABLED	**True** ou False : indique si l'API doit dumper dans un fichier le détail des erreurs inconnues de code erreur **CRYPTO_UNKNOWN_ERROR.**
DUMP_FILEPATH	Nom complet du fichier de dump (écriture en mode append).
KEY_DIRECTORY	Répertoire de stockage des clefs.
MIN_PASSPHRASE_SIZE	Taille minimale autorisée pour une passphrase en nombre de caractères
RAND_SEED_DIR	Répertoire de stockage des fichiers de « seed ».

5.3.3 Liste des valeurs par défaut des paramètres

Paramètre	Valeur par défaut
DEFAULT_SIGN_ALGO	RSA
DEFAULT_SYM_ALGO	`Blowfish`
DUMP_ENABLED	`False`
DUMP_FILEPATH	`Répertoire user.home`
RAND_SEED_DIR	`Répertoire de stockage de safeapi_license.txt (keys).`
MIN_PASSPHRASE_SIZE	8
KEY_DIRECTORY	`Répertoire de stockage de safeapi_license.txt (keys).`
RAND_SEED_DIR	identique à **KEY_DIRECTORY**

5.3.4 Méthodes de détection des erreurs

CryptoCommon comporte des méthodes de diagnostic du fonctionnement de chaque méthode de cryptographie :

- 1) **isOperationOK()** : Vérification du bon déroulement de la dernière méthode.
- 2) **getRegisteredError** : Récupération du Code Erreur de base sous forme de String.
- 3) **getRawError** : Récupération du Code Erreur détaillé sous forme de String.

Méthode	**isOperationOK**	
Description	Indique si la dernière méthode de cryptographie appelée s'est terminée correctement.	
Arguments	Aucun.	
Retour	Boolean	True : la dernière méthode est un succès False : la dernière méthode a échoué

Méthode	**getRegisteredError**	
Description	Renvoie le Code Erreur renvoyé par la dernière méthode de cryptographie qui a échoué.	
Arguments	Aucun	
Retour	String	Message d'erreur (Code erreur)

Méthode	**getRawError**	
Description	Renvoie les informations détaillées pour la dernière erreur sous forme de chaîne.	
Arguments	Aucun.	
Retour	String	Message d'erreur détaillé.

La liste des messages d'erreurs retournés par cette fonction est présentée dans un tableau fourni en annexe (**Annexe A: Tableau des Codes Erreur par méthode).**

Le tableau décrit aussi pour chaque méthode de cryptographie la liste des erreurs possibles (valeurs de **getRegisteredError & getRawError).**

5.3.5 Méthodes de création de seed ou valeurs aléatoires

5.3.5.1 Création initiale d'un fichier de nombre aléatoires (seed)

Méthode	**createSeedFile**		
Description	Crée (ou remplace) le fichier servant au seed du générateur de nombre aléatoire. A cet effet, cette méthode prompte une fenêtre de génération de données aléatoires à l'utilisateur.		
Arguments	String	Title	Le titre de la fenêtre de seed
	String	Caption	La légende de la fenêtre de seed

Notes :

- Il est obligatoire de créer un fichier de seed avant l'utilisation des fonctions de chiffrement symétrique et asymétrique.
- Les données *aléatoires* servent en input d'un générateur de nombres *pseudo-aléatoires* qui repose sur la norme ANSI X9.17 et implémenté sous la forme de la méthode **getRandomBytes()**
- L'appel de **setParameter()** avec « RAND_SEED_DIR » permet de choisir le répertoire de stockage du fichier de seed.

5.3.5.2 Génération dynamique d'un seed

Méthode	**getRandomBytes()**		
Description	Génère un seed ou valeur pseudo-aléatoire de 24 octest.		
Argument	String	sMsg	Message d'initialisation sous la forme d'une chaîne de caractères quelconques
Retour	byte[ ]	valeur du hash du buffer mémorisé (suite de bits).	

Notes :

- Il est obligatoire de créer un fichier de seed avec **createSeedFile()** avant de pouvoir utiliser **getRandomBytes().**
- les valeurs de seed servent à alimenter les fonctions de génération de clefs symétriques ou asymétriques (paramètre **bSeed**).

5.3.6 Méthodes complémentaires

Méthode	getVersion	
Description	Récupère la Version de SafeAPI	
Arguments	Aucun	
Retour	String	Informations de version de SafeAPI.

Méthode	wipe		
Description	Détruit le fichier spécifié de façon sécurisée et définitive en effectuant plusieurs passes d'écriture de données aléatoires sur le contenu de celui-ci.		
Arguments	String	FilePath	Nom complet du fichier à détruire.
	int	Level	Niveau de sécurité, de 1 à 3 (8 à 20 passes)

Note :

- **wipe** constitue un moyen d'effacer de façon très sûre un fichier.
- **Attention** : un fichier effacé par **wipe** est totalement irrécupérable.

5.4 Classe CryptoHash : méthodes de hachage

CryptoHash contient trois types de méthodes :

- Méthodes de hash de buffer.
- Méthode de hash de fichiers.
- Méthode de hash de buffer court avec retour au format hexadécimal.

5.4.1 Hash de buffer long

Méthode	**hashDataBufferInit**		
Description	Initialise l'outil et le buffer de hash.		
Arguments	String	Algorithm	Algorithme de hash : • `"MD5"` • `"SHA-1"`

Méthode	**hashDataBuffer**		
Description	Ajoute les octets passés en argument au buffer qui sera haché au moment de l'appel à hashDataBufferDigest. Cette méthode ne doit être utilisée qu'une fois le buffer initialisé.		
Arguments	byte[ ]	SubBuffer	Octets à ajouter au buffer à hasher

Méthode	**hashDataBufferDigest**	
Description	Calcule et retourne le hash du buffer qui doit préalablement être initialisé et rempli avec les méthodes adéquates.	
Arguments	Aucun.	
Retour	byte[ ]	Valeur du hash du buffer en mémoire (suite de bits)

Notes :

Le hash de buffer/chaîne est effectué par « accumulation » et permet ainsi de hacher des chaînes **sans limite de taille**. Il faut respecter la séquence :

- 1) Initialisation : **hashDataBufferInit.**
- 2) Alimentation du buffer à hacher : **hashDataBuffer.**
- 3) Calcul final de la valeur hash : **hashDataBufferDigest.**

5.4.2 Hash de fichier

Méthode	hashDataFile		
Description	Calcule et retourne la valeur du hash associé au fichier spécifié à l'aide de l'algorithme passé en paramètre.		
Arguments	String	sAlgorithm	Algorithme de hash : • "MD5" • "SHA-1"
	String	sFilePath	Nom complet du fichier
Retour	byte[ ]	valeur du hash du buffer mémorisé (suite de bits).	

Notes :

- Le terme **nom complet** désigne le nom complet d'un fichier avec les directories et unités. Exemples :

  - Windows : `c:\safelogic\lib\cryptix.jar`
  - Unix : `/home/safelogic/lib/cryptix.jar`

5.4.3 Hash de buffer court

Méthode	hashBufferHex		
Description	Calcule et retourne la valeur hash du buffer à l'aide de l'algorithme passé en paramètre.		
Arguments	String	sAlgorithm	Algorithme de hash : • `"MD5"` • `"SHA-1"`
	String	bBuffer	Buffer à hacher.
Retour	String	Valeur du hash du buffer mémorisé sous la forme d'une String hexadécimale.	

Notes :

- Permet de hacher un buffer en une seule opération.

- La valeur de hachage est retournée au format hexadécimal, ce qui permet son affichage et/ou le stockage sans opération supplémentaire dans un fichier ou une database SQL.

5.5 Classe CryptoSym : méthodes de chiffrement symétrique

5.5.1 Méthode genSecretKey : génération de clé secrète

Méthode	genSecretKey		
Description	Génère une clef secrète de la longueur désirée (si valable) avec l'algorithme spécifié à partir du seed fourni et la retourne.		
Arguments	char[ ]	caPassphrase	Passphrase de protection de la clé.
	String	sAlgorithm	Algorithme de chiffrement symétrique. (Cf. « Tableau des algorithmes de chiffrement symétriques»)
	int	Length	Longueur de la clef (en bits)
	byte[ ]	bSeed	Valeur du seed.
	String	sEmail	E mail de référence de la clé.
	int	lKeyIndex	Numéro de cette clé (facultatif).
Retour	Aucun.		

Notes :

La clé secrète générée est stockée dans un fichier dont le nom est composé de la concaténation dans l'ordre des objets suivants séparés par le caractère « _ »:

- De l'e mail identifiant/de référence.
- De l'indice passé à **genSecretKey.**
- Du nom de l'algorithme choisi : Blowfish, CAST-128, IDEA.
- De l'extension « .skf ».

Le nom générique en minuscules du fichier de stockage, sans la directory ni l'extension, constitue le « **Key ID** » de la clé secrète.

Exemples de **Key ID** :

- **user@domain.com_1_ cast-128**
- **user@domain.com_2_ blowfish**

Rappel :
Le répertoire de stockage est défini par la méthode **setParameter()** avec le paramètre **KEY_DIRECTORY.**

Avant enregistrement sur le disque dur, la clé secrète est chiffrée avec :

- L'algorithme choisi par l'utilisateur dans l'API : Blowfish, CAST-128, IDEA.
- Le mode CFB.
- Une clé de 128 bits qui est la valeur hash MD5 de la passphrase passée comme paramètre à **genSecretKey.**

La clé secrète est donc stockée de façon fortement sécurisée sur le disque dur et aucun stockage de la passphrase n'est effectué.

5.5.2 Méthode getSecretKey : Extraction de la clé secrète

Méthode	**getSecretKey**		
Description	Extrait - de son fichier de stockage - une clef secrète générée par genSecretKey et le retourne au format hexadécimal		
Arguments	char[ ]	caPassphrase	Passphrase de protection de la clé.
	String	sAlgorithm	Algorithme de chiffrement symétrique. (Cf. « Tableau des algorithmes de chiffrement symétriques»)
	int	Length	Longueur de la clef (en bits)
	String	sEmail	E mail de référence de la clé.
	int	lKeyIndex	Numéro de cette clé (facultatif).
Retour	byte[ ]		Valeur de la clé secrète au format hexadécimal (lettres en minuscules).

Note :

- Cette méthode est à utilisée pour communiquer à une autre application une clef secrète générée par SafeAPI.

- **Attention** : la clef récupérée n'est plus protégée par sa passphrase : elle doit être véhiculée sur un canal sur.

5.5.3 Chiffrement de buffer

Méthode	encryptBuffer		
Description	Chiffre le buffer passé en argument à l'aide de la clef associée à la passphrase spécifiée.		
Arguments	String	sKey_ID	Key ID de la clé secrète.
	char[ ]	caPassphrase	Passphrase de protection de la clé sKey_ID.
	byte[ ]	bBuffer	Buffer à chiffrer.
Retour	byte[ ]		Buffer chiffré précédé du IV (Initialisation Vector) qui est généré en interne par la méthode.

Rappel :

Le **Key_ID** de la clé secrète est un identifiant de la forme :

```
<email>_<index>_<algorithme>
```

avec :
`e_mail` : compte pop associé à la clé de type **user@domain.com.**
`index` : Indice numérique.
`Algorithme` : algorithme parmi la liste Blowfish, CAST-128, IDEA.

Méthode	decryptBuffer		
Description	Déchiffre le buffer passé en argument à l'aide de la clef associée à la passphrase spécifiée.		
Arguments	String	sKey_ID	Key_ID de la clé secrète.
	char[ ]	caPassphrase	Passphrase de protection de la clé sKey_ID.
	byte[ ]	bBuffer	Buffer *chiffré avec* ***encryptBuffer*** à déchiffrer.
Retour	byte[ ]		Buffer déchiffré.

Notes :

Cette méthode est à utiliser *conjointement* avec **encryptBuffer** pour chiffrer et déchiffrer des buffers. On a la cinématique :

- **encryptBuffer :** chiffrement d'un buffer qui retourne un buffer chiffré précédé sur les 64 premier bits par un Initialisation Vector.
- **decryptBuffer :** déchiffrement d'un buffer *chiffré avec* ***encryptBuffer*** , et donc composé d'un IV sur les 64 premiers bits.

L'IV représente les 64 premiers bits du buffer retourné. Cette concaténation est complètement transparente dans une utilisation normale : chiffrement puis déchiffrement ultérieur d'un buffer.

5.5.4 Chiffrement de buffers: méthodes avec passage de clé et IV

Les méthodes SafeAPI de chiffrement/déchiffrement symétrique de buffers sont surchargées pour permettre aux utilisateurs avertis de gérer directement les paramètres d'un chiffrement symétrique :

- Valeur de clé en format binaire.
- Valeur de IV (Initialisation Vector) en format binaire.

Méthode	**encryptBufferRawKey**		
Description	Chiffre le buffer passé en argument à l'aide de la clef et du IV spécifiés en bytes.		
Arguments	String	sAlgorithm	Algorithme de chiffrement (Cf. « Tableau des algorithmes de chiffrement symétriques »)
	byte[ ]	bKey	Clé de 128 bits en format binaire.
	byte[ ]	bIV	IV de 64 bits (8 bytes) à utiliser.
	byte[ ]	bBuffer	Buffer à chiffrer.
Retour	byte[ ]		Buffer chiffré

Méthode	**decryptBufferRawKey**		
Description	Déchiffre le buffer passé en argument à l'aide de la clef et du IV spécifiés en bytes.		
Arguments	String	sAlgorithm	Algorithme de chiffrement (Cf. « Tableau des algorithmes de chiffrement symétriques »)
	byte[ ]	bKey	Clé de 128 bits en format binaire.
	byte[ ]	bIV	IV de 64 bits (8 bytes) utilisé pour chiffrer.
	byte[ ]	bBuffer	Buffer à déchiffrer.
Retour	byte[ ]		Buffer déchiffré.

Notes :

- Les trois algorithmes sont toujours utilisés en mode CFB (Cipher Feedback Mode).

- La surcharge des fonctions permet aussi de tester le « fonctionnement » conforme de SafeAPI et des programmes VB par rapport aux jeux d'essais normalisés. (Cf. le chapitre « FAQ IMPLEMENTATION/CRYPTOGRAPHIE » pour plus de précisions.)

5.5.5 Chiffrement de fichiers

Méthode	encryptFile		
Description	Chiffre le fichier dont le nom complet (unité et répertoire inclus) est passé en argument à l'aide de la clef associée à la passphrase spécifiée.		
Arguments	String	sKey_ID	Key_ID de la clé.
	char[ ]	caPassphrase	Passphrase de protection de la clé sKey_ID.
	String	sInputPath	Nom complet du fichier à chiffrer.
	String	sOutputPath	Nom complet du fichier chiffré.

Notes :

- **sInputPath** et **sOutputPath** doivent être différents.
- Rappel : Le terme **nom complet** désigne le nom complet d'un fichier avec les directories et unités.
  - Exemple Windows : **`c:\safelogic\lib\cryptix.jar`**
  - Exemple Unix : **`/home/safelogic/lib/cryptix.jar`**

Méthode	decryptFile		
Description	Déchiffre le fichier dont le nom complet est passé en argument à l'aide de la clef associée à la passphrase spécifiée.		
Arguments	String	sKey_ID	Key_ID de la clé.
	char[ ]	caPassphrase	Passphrase de protection de la clé sKey_ID.
	String	sInputPath	Nom complet du fichier à déchiffrer
	String	sOutputPath	Nom complet du fichier déchiffré

Notes :

- sInputPath et sOutputPath doivent être différents.

5.5.6 Tableau des algorithmes de chiffrement symétrique

Valeur Algorithme	Remarques
`Blowfish`	Mode CFB (Cipher Feedback Mode).
`CAST-128`	Mode CFB.
`IDEA`[1]	Mode CFB.

Note : la casse doit être respectée pour le nom d'algorithme.

[1] IDEA est breveté par ASCOMM et nécessite une licence spéciale auprès de SafeLogic pour être utilisable.

5.6 Classe CryptoAsym : méthodes de chiffrement asymétrique

5.6.1 Méthode genKeyPair : Génération de d'une paire de clés RSA

Méthode	**genKeyPair**		
Description	Génère une paire de clés RSA de la longueur désirée (si valable) avec l'algorithme spécifié à partir du seed fourni et la retourne.		
Arguments	String	sAlgorithm	Algorithme de chiffrement asymétrique : • « RSA ».
	char[ ]	caPassphrase	Passphrase de protection de la clé.
	int	Length	Longueur de la clef (en bits) : doit être un multiple de 512 inférieur ou égal à 4096.
	byte[ ]	bSeed	Valeur du seed.
	String	sEmail	E mail de référence de la clé.
	int	lKeyIndex	Numéro de cette clé (facultatif).
Retour			

Notes :

Le nom du fichier de stockage des deux clés (publique & privée) est composé :

- De l'e mail identifiant/de référence.
- De l'indice passé à **genKeyPair.**
- De la chaîne « **RSA** ».
- De l'extension « **.skf** » pour la clé privée.
- De l'extension « **.pkf** » pour la clé publique.

Le nom générique en minuscules du fichier de stockage, sans la directory ni l'extension, constitue le « **Key ID** » de la paire de clés.

Exemples de **Key ID** :

- **user@domain.com_1_ rsa**
- **user@domain.com_2_ rsa**

Avant enregistrement sur le disque dur, la clé privée est chiffrée avec :

- L'algorithme symétrique Blowfish.
- Le mode CFB.
- Une clé de 128 bits qui est la valeur hash MD5 de la passphrase passée comme paramètre à **genKeyPair.**

La clé privée est donc chiffrée et stockée de façon fortement sécurisée sur le disque dur et aucun stockage de la passphrase n'est effectué.

Rappel :
Le répertoire de stockage est défini par la méthode **setParameter()** avec le paramètre **KEY_DIRECTORY.**

5.6.2 Export de clés RSA générées avec le format de stockage SafeAPI

La classe **CryptoAsymRawRSA** dispose de fonctions pour exporter sous forme de tableau de bytes ou sous forme de chaîne hexadécimale les composants d'une paire de clé générée avec **genKeyPair.**

5.6.3 Chiffrement de buffers avec RSA

Méthode	**encryptBuffer**		
Description	Chiffre un buffer en RSA pour une clé publique dont le Key ID est passé en argument.		
Arguments	String	sKey_ID	Key_ID de la clé publique.
	byte[ ]	bBuffer	Buffer à chiffrer en RSA.
Retour	byte[ ]		Buffer chiffré.

Notes :

Les fonctions de chiffrement/déchiffrement de buffer présentent les caractéristiques suivantes dans SafeAPI Version 1.20 :

- La taille maximale d'un buffer est limitée à 128 bits.
- Le mode utilisé est ECB (Electronic Code Book).
- Le padding utilisé est PKCS#7.

Méthode	**decryptBuffer**		
Description	Déchiffre un buffer chiffré avec clé publique RSA dont le Key ID est passé en argument.		
Arguments	String	sKey_ID	Key_ID de la clé privée.
	char[ ]	caPassphrase	Passphrase correspondant à la clé privée de même Key ID utilisée pour chiffrer le buffer.
	byte[ ]	bBuffer	Buffer *chiffré avec* ***encryptBuffer*** à déchiffrer.
Retour	byte[ ]		Buffer déchiffré.

5.6.4 Chiffrement mixte de fichiers avec RSA : principe

Le chiffrement de fichiers avec RSA est un chiffrement dit « mixte » qui consiste à combiner un chiffrement symétrique avec un chiffrement asymétrique. Cette méthode permet de bénéficier des fonctionnalités d'un chiffrement asymétrique (utilisation de clés publiques/privées), tout en conservant la vitesse de traitement offerte par les algorithmes symétriques.

Dans le cas d'un chiffrement de fichier pour un utilisateur dont on possède la clé publique RSA :

- Le générateur de nombre pseudo-aléatoires fabrique une clé symétrique dite . « clé de session » : Elle ne servira que pour le chiffrement de ce fichier et ne sera jamais plus utilisée.
- Le fichier est chiffré avec un algorithme symétrique et la clé de session.
- La clé de session est elle même chiffrée avec l'algorithme asymétrique (RSA), en utilisant la clé publique de l'utilisateur.
- L'ensemble fichier chiffré + clé de session chiffrée est fusionné dans un nouveau fichier.

Le déchiffrement par un utilisateur s'effectue de façon réciproque :

- L'ensemble fichier chiffré + clé de session est récupéré.
- On extrait la clé de session chiffrée et on la déchiffre avec la clé privée RSA de l'utilisateur.
- La clé de session permet alors de déchiffrer le fichier.

5.6.5 Gestion de Listes de Destinataires

SafeAPI permet de gérer des chiffrements de fichiers pour plusieurs utilisateurs en une seule opération.

Le principe consiste à :

- Créer une Liste de Destinataires (nous utiliserons à présent le terme abrégé Liste).
- A ajouter des identifiant de clé publiques RSA à la Liste. Cela correspond à « ajouter des Destinataires à la Liste ».
- Chiffrer le fichier en une seule opération en passant comme argument à une fonction API le nom de la Liste : le fichier est alors chiffré pour tous les destinataires.

Méthode	**createRecipients**		
Description	Crée une Liste de destinataires pour un chiffrement RSA de fichier.		
Arguments	String	sListName	Nom de la Liste de destinataires.

Méthode	**addRecipient**		
Description	Ajoute un destinataire à une Liste		
Arguments	String	sListName	Nom de la Liste de destinataires.
	String	sKey_ID	Key_ID de la clé publique du destinataire.
Retour	Boolean	True l'ajout a été effectué False sinon	

5.6.6 Chiffrement asymétrique de fichier

Méthode	**encryptFile**		
Description	Chiffre un fichier en RSA pour un destinataire unique ou pour une liste de destinataires passée en argument.		
Arguments	String	sKey_ID *ou* sListName	Key_ID de la clé publique du destinataire unique. *ou* Nom de la Liste de destinataires.
	String	sInputPath	Nom complet du fichier à chiffrer.
	String	sOutputPath	Nom complet du fichier chiffré à créer.

Notes :

- **sInputPath** et **sOutputPath** doivent être différents.

Méthode	**decryptFile**		
Description	Déchiffre un fichier chiffré avec une clé publique et la méthode **encryptFile**		
Arguments	String	sKey_ID	Key_ID de la clé privée de déchiffrement.
	char[ ]	caPassphrase	Passphrase associée à la clé privée.
	String	sInputPath,	Nom complet du fichier chiffré.
	String	sOutputPath	Nom complet du fichier déchiffré à créer.

5.6.7 Signature & vérification de signature

Notes :

- **1)** Les méthodes de signature & vérification de signature s'utilisent par couples :

Méthode de signature	**Méthode correspondant de vérification**
signBuffer()	verifyBuffer()
signFile() Version avec nom complet du fichier	verifyFile() Version avec nom complet du fichier
rawSignFile() Version avec signature en bits	rawVerifyFile() Version avec signature en bits
encryptAngSign() Chiffrement asymétrique et signé de fichier.	decryptAndSign() Déchiffrement asymétrique d'un fichier et vérification de sa signature.

- **2) signBuffer()** et **verifyBuffer()** permettent de signer/vérifier des buffers.

- **3) signFile()** et **verifyFile()** fonctionnent avec une signature contenue dans un fichier.

- **4) rawSignFile()** et **rawVerifyFile()** fonctionnent avec une signature en bits dont la gestion est déléguée au programmeur.

- **5)** Le couple **encryptAngSign() / decryptAndSign()** permet en une seule opération d'effectuer un chiffrement + une signature. Dans ce cas, la signature est toujours stockée dans le fichier crypté.
  La méthode **isFileSigned** permet de savoir si un fichier chiffré avec SafeAPI a été en plus signé avec une clé privée.

Méthode	signBuffer		
Description	Calcule et retourne la signature associée au buffer spécifié à l'aide de la clef privée passée en paramètre.		
Arguments	String	sKey_ID	Identifiant de la clef privée à utiliser pour signer.
	char[ ]	caPassphrase	Passphrase de la clef privée
	byte[ ]	bBuffer	Buffer à signer
Retour	byte[ ]	Valeur de la signature (suite de bits) de bBuffer.	

Méthode	verifyBuffer		
Description	Vérifie si une signature en bits est associée à buffer de façon valide.		
Arguments	String	sKey_ID	Identifiant de la clef publique correspondant à la signature.
	String	bBuffer	Nom complet du buffer à vérifier.
	byte[ ]	bSignature	Signature du buffer bBuffer.
Retour	Boolean	True : la signature est vérifiée (buffer bBfuffer authentifié) False : la signature est invalide.	

Méthode	rawSignFile		
Description	Calcule et retourne la signature associée au fichier spécifié à l'aide de la clef privée passée en paramètre.		
Arguments	String	sKey_ID	Identifiant de la clef privée à utiliser pour signer.
	char[ ]	caPassphrase	Passphrase de la clef privée
	String	sFilePath	Nom complet du fichier à signer.
Retour	byte[ ]	Valeur de la signature (suite de bits)	

Méthode	rawVerifyFile		
Description	Vérifie si une signature en bits est associée à un fichier de façon valide.		
Arguments	String	sKey_ID	Identifiant de la clef publique correspondant à la signature.
	String	sFilePath	Nom complet du fichier à vérifier.
	byte[ ]	bSignature	Signature du fichier FilePath
Retour	Boolean	True : la signature est vérifiée (fichier authentifié) False : la signature est invalide.	

Méthode	signFile		
Description	Calcule et enregistre sur fichier la signature associée au fichier spécifié à l'aide de la clef privée passée en paramètre.		
Arguments	String	sKey_ID	Identifiant de la clef privée à utiliser pour signer.
	char[ ]	caPassphrase	Passphrase de la clef privée.
	String	sFilePath	Nom complet du fichier à signer.
	String	sSigFilePath	Nom complet de la signature du fichier.

Méthode	verifyFile		
Description	Vérifie si la signature associée au fichier est valide.		
	String	sKey_ID	Identifiant de la clef publique correspondant à la signature.
Arguments	String	FilePath	Nom complet du fichier à vérifier.
	String	sSigFilePath	Nom complet du fichier contenant la signature du fichier FilePath.
Retour	Boolean	True : la signature est vérifiée (fichier authentifié) False : la signature est invalide	

Méthode	encryptAndSign		
Description	Signe le fichier spécifié avec la clef privée et le chiffre pour la liste des destinataires établie.		
Arguments	String	sKey_ID	Identifiant de la clef publique correspondant à la signature.
	char[ ]	caPassphrase	Passphrase de la clef privée de signature
	String	sListName	Nom de la liste de destinataires
	String	sInputPath	Nom complet du fichier cible
	String	sOutputPath	Nom complet du fichier destination

Méthode	decryptAndVerify		
Description	Décrypte le fichier spécifié avec la clef privée et vérifie la signature.		
Arguments	String	sKey_ID	Identifiant de la clef privée (de déchiffrement).
	char[ ]	caPassphrase	Passphrase de la clef privée de déchiffrement.
	String	sSignKeyId	Identifiant de la clef publique correspondant à la signature
	String	sInputPath	Nom complet du fichier cible (chiffré)
	String	sOutputPath	Nom complet du fichier dedestination (en clair)
Retour	Int	1 : le fichier a été signé et est authentifié -1 : le fichier a été signé mais n'est pas authentifié 0 : le fichier n'a pas été signé	

Méthode	isFileSigned		
Description	Vérifie qu'un fichier chiffré contient une signature.		
Arguments	String	FilePath	Nom complet du fichier chiffré.
Retour	Boolean	True : le fichier chiffré est signé False : le fichier chiffré n'est pas signé.	

Méthode	getSignKeyIdDigest		
Description	Retourne la valeur du hash de l'identifiant de la clef utilisée pour signer le fichier.		
Arguments	String	sFilePath	Nom complet du fichier
Retour	byte[ ]	La valeur du hash de l'identifiant de la clef de l'expediteur	

Note :

- Usage futur.

5.7 Classe CryptoAsymRawRSA : méthodes de chiffrement asymétrique avec clés RSA « natives »

CryptoAsymRawRSA est destinée aux utilisateurs avertis qui souhaitent utiliser les fonctions avancées de RSA.

L'utilisation de la classe CryptoAsym est recommandée dans la majorité des cas.

Les méthodes décrites dans ce chapitre utilisent toutes des clés RSA au format « natif », i.e. des clés exprimées par des nombres.

Attention :
Dans toutes ces méthodes du chapitre 5.7, les valeurs des clés privées ne sont pas protégées.

Elles doivent rester secrètes pour assurer la sécurité des chiffrements.

5.7.1 Chiffrement de buffers avec clé RSA « native »

Méthode	**encryptBufferRawKey**		
Description	Chiffre un buffer en RSA avec une clé publique dont la forme « native » est passée comme argument.		
Arguments	byte[ ]	bN_Modulus	Modulo N publique de la clé RSA. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bE_Exponent	Exposant E publique. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bBuffer	Buffer à chiffrer en RSA.
Retour	byte[ ]		Buffer chiffré.

Mode & Padding

Les fonctions de chiffrement/déchiffrement de buffer présentent les caractéristiques suivantes :

- Le **mode** utilisé est **ECB** (Electronic Code Book).
- Le **padding** utilisé est **PKCS#7.**

5.7.2 Déchiffrement de buffers avec clé RSA « native »

Méthode	decryptBufferRawKey		
Description	Déchiffre un buffer en RSA avec une clé **privée** dont la forme « native » est passée comme argument.		
Arguments	byte[ ]	bD_Exponent	**D**, exposant **Privé** à conserver secret. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bP_Factor	1er facteur **P Privé** à conserver secret. (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bP_Factor	2ème facteur **Q Privé** à conserver secret. (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bBuffer	Buffer chiffré en RSA à déchiffrer.
Retour	byte[ ]		Buffer déchiffré.

Attention/Rappel :

Les 3 valeurs bD_Exponent, bP_Factor et bP_Factor constituent les éléments « en clair » de la clé privée et doivent rester confidentielles.

Mode & Padding

Les fonctions de chiffrement/déchiffrement de buffer présentent les caractéristiques suivantes :

- Le **mode** utilisé est **ECB** (Electronic Code Book).
- Le **padding** utilisé est **PKCS#7.**

5.7.2.1 Initialisation & chargement de clés RSA au format SafeAPI en mémoire

Les deux fonctions **loadPublicKey** et/ou **loadPrivateKey** permettent de préparer les extractions **et doivent être lancées au préalable** aux autres fonctions pour accéder aux elements.

Méthode	**loadPublicKey**		
Description	Charge en mémoire la clé publique correspondant à une clé générée avec SafeAPI.		
Arguments	String	sKey_ID	Key_ID de la clé publique.
Retour			

Méthode	**loadPrivateKey**		
Description	Charge en mémoire la clé privée correspondant à une clé générée avec SafeAPI.		
Arguments	String	sKey_ID	Key_ID de la clé privée..
	char[ ]	caPassphrase	Passphrase de protection de la clé.
Retour			

5.7.2.2 Méthodes de récupération des éléments des clés RSA

Après le lancement des fonctions **loadPublicKey et/ou loadPrivateKey,** les fonctions de type « get » suivantes permettent de récupérer sous forme de tableau de bytes ou de chaîne hexadécimale (String) les éléments composants la paire de clés RSA :

Méthode	**getPublicKeyExponent**	
Description	Récupère l'exposant E publique de la clé publique	
Arguments	Aucun	
Retour	byte[ ]	Exposant E publique. Représentation du nombre en bits en complément à 2 en *big-Endian.*

Méthode	getHexPublicKeyExponent	
Description	Récupère l'exposant E publique de la clé publique sous forme de chaîne hexadécimale.	
Arguments	Aucun	
Retour	String	Exposant E publique sous forme de chaîne hexadécimale.

Méthode	getModulus	
Description	Récupère le modulo N publique de la clé RSA	
Arguments	Aucun	
Retour	byte[ ]	Modulo N publique de la clé RSA. Représentation du nombre en bits en complément à 2 en *big-Endian.*

Méthode	getHexModulus	
Description	Récupère le modulo N publique de la clé RSA sous forme de chaîne hexadécimale.	
Arguments	Aucun	
Retour	String	Modulo N publique de la clé RSA sous forme de chaîne hexadécimale.

Méthode	getPrivateKeyExponent	
Description	Récupère l'exposant **D** **Privé** à conserver secret.	
Arguments	Aucun	
Retour	byte[ ]	**D**, exposant **Privé** à conserver secret. Représentation du nombre en bits en complément à 2 en *big-Endian.*

Méthode	getHexPrivateKeyExponent	
Description	Récupère l'exposant **D** **Privé** à conserver secret sous forme de chaîne hexadécimale.	
Arguments	Aucun	
Retour	String	**D**, exposant **Privé** à conserver secret sous forme de chaîne hexadécimale.

Méthode	getP	
Description	Récupère le 1er facteur **P** **Privé** à conserver secret (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*	
Arguments	Aucun	
Retour	byte[ ]	1er facteur **P** **Privé** à conserver secret. (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*

Méthode	getHexP	
Description	Récupère le 1er facteur **P** **Privé** à conserver secret (nombre premier) sous forme de chaîne hexadécimale.	
Arguments	Aucun	
Retour	String	1er facteur **P** **Privé** à conserver secret (nombre premier) sous forme de chaîne hexadécimale.

Méthode	getQ	
Description	Récupère le 2ème facteur **Q** **Privé** à conserver secret (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*	
Arguments	Aucun	
Retour	byte[ ]	1er facteur **P** **Privé** à conserver secret (nombre premier) sous forme de chaîne hexadécimale.

Méthode	getHexQ	
Description	Récupère le 2ème facteur **Q** **Privé** à conserver secret (nombre premier) sous forme de chaîne hexadécimale.	
Arguments	Aucun	
Retour	String	2ème facteur **Q** **Privé** à conserver secret (nombre premier) sous forme de chaîne hexadécimale.

Méthode	getInverseOfQModP	
Description	Récupère **U** , l'inverse mathématique de Q modulo P à conserver secret. Représentation du nombre en bits en complément à 2 en *big-Endian.*	
Arguments	Aucun	
Retour	byte[ ]	**U** , inverse de Q modulo P à conserver secret. Représentation du nombre en bits en complément à 2 en *big-Endian.*

Méthode	getHexInverseOfQModP	
Description	Récupère **U** , l'inverse de Q modulo P à conserver secret sous forme de chaîne hexadécimale.	
Arguments	Aucun	
Retour	String	**U** , inverse de Q modulo P à conserver secret sous forme de chaîne hexadécimale.

5.7.2.3 Exemple complet d'utilisation en Java

Un exemple complet d'export des clés RSA sous forme d'éléments et de chiffrement et de déchiffrement de buffer peut être trouvé dans le programme : **rawRSAExemple.java** dans le répertoire des exemples.

5.7.3 Application directe du chiffrement RSA

CryptoAsymRawRSA permet d'utiliser directement RSA sur un buffer, en utilisant des éléments constitutifs de la clé.

Cette utilisation directe permet des opérations « non conventionelles », non admises dans les autres méthodes de SafeAPI :

- Chiffrement d'un buffer avec une clé privée.
- Déchiffrement d'un buffer avec une clé publique.

Méthode	**rsaWithCrt**		
Description	Applique directement l'algorithme RSA sur un buffer en utilisant les éléments de la clé **privée**. **rsaWithCrt** utilise le Chinese Remainder Theorem (CRT)		
Arguments	byte[ ]	bD_Exponent	**D**, exposant **Privé** à conserver secret. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	Byte[ ]	bP_Factor	1er facteur **P** **Privé** à conserver secret. (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bP_Factor	2ème facteur **Q** **Privé** à conserver secret. (nombre premier). Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bU	**U** , inverse de Q modulo P à conserver **secret**. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bBuffer	Buffer à chiffrer en RSA.
Retour	byte[ ]		Buffer chiffré par RSA.

Attention/Rappel :

Les 3 valeurs bD_Exponent, bP_Factor, bP_Factor et bU constituent les éléments « en clair » de la clé privée et doivent rester confidentielles.

Méthode	rsa		
Description	Applique directement l'algorithme RSA sur un buffer en utilisant les éléments de la clé publique. **rsa** n'utilise pas le Chinese Remainder Theorem (CRT)		
Arguments	byte[ ]	bN_Modulus	Modulo N publique de la clé RSA. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bE_Exponent	Exposant E publique. Représentation du nombre en bits en complément à 2 en *big-Endian.*
	byte[ ]	bBuffer	Buffer à chiffrer en RSA.
Retour	byte[ ]		Buffer chiffré par RSA.

Note pour les deux méthodes rsa :

- Aucun mode n'est appliqué.
- Les méthodes **rsa** découpent le buffer en blocs d'octets de la taille de la clé divisée par 8. Par exemple, si la clé fait 2048 bits, le buffer résultant est constitué de blocs de 2048 / 8 = 256 octets.
- Tout chiffrement inférieur à la taille d'un bloc génère un bloc chiffré qui fait la taille d'un bloc.
- Exemple : avec une clé de 2048 bits, on veut appliquer RSA sur un buffer de 1038 octets. 1038 = 256 + 256 + 256 + 256 + 14. Chaque bloc chiffré sera long de 256 octets. Le buffer résultant sera donc de 1280 octets = 256 + 256 + 256 + 256 + 256, le dernier morceau de 14 octets étant complété après chiffrement jusqu'à 256 octets.

5.7.3.1 Exemple complet d'utilisation en Java

Un exemple complet d'export des clés RSA sous forme d'éléments et de chiffrement et de déchiffrement de buffer peut être trouvé dans le programme : **TestRSA.java** dans le répertoire des exemples.

5.8 Classe CryptoDir : méthodes de chiffrement de répertoires

Les méthodes de la classe **CryptoDir** se divisent en trois groupes principaux.

- 1) Méthodes liées au paramétrage dynamique de l'API.
- 3) Méthodes de chiffrement symétrique de répertoires
- 3) Méthodes de chiffrement asymétrique de répertoires.

5.8.1 Paramétrage dynamique de CryptoDir

Le paramètrage dynamique de Dir Crypto se fait avec les fonctions **setParameter() et getParameter()**, décrites dans 5.3.1 CryptoCommon : Méthodes de Paramétrage dynamique de l'API

On a les paramètres supplémentaires :

Paramètre de CryptoDir	Rôle & plages de valeur
DIR_DELETE_LEVEL	Type de delete de fichiers après chiffrement de répertoire. DELETE_NONE : aucun delete. DELETE_SIMPLE : delete normal. DELETE_WIPE_1 : wipe de niveau 1 (faible) DELETE_WIPE_2 : wipe de niveau 2 (moyen) DELETE_WIPE_3 : wipe de niveau 3 (fort)
VERBOSE	Indique si un chiffrement de fichiers doit être affiché sur la console système. True ou False
NB_RETRY_WHEN_LOCK	Nombre d'essais successifs de chiffrement d'un fichier verrouillé par un autre processus.
SECONDS_PAUSE_WHEN_LOCK	Nombre de secondes de pause entre deux essais de chiffrement d'un fichier verrouillé par un autre processus.

5.8.2 Liste des valeurs par défaut des paramètres

Paramètre de CryptoDir	Valeurs par défaut
DIR_DELETE_LEVEL	DELETE_SIMPLE
VERBOSE	True
NB_RETRY_WHEN_LOCK	3
SECONDS_PAUSE_WHEN_LOCK	1

5.8.3 Chiffrement symétrique de répertoires

Méthode	encryptDirWithPassphrase		
Description	Chiffre tous les fichiers d'un répertoire. Les fichiers chiffrés sont placés dans un répertoire de sortie différent et à spécifier. La clé de chiffrement symétrique est fabriquée dynamiquement en dérivant la passphrase.		
Arguments	String	sAlgorithm	Algorithme à utiliser. (Cf. la liste des valeurs autorisées dans SAPIDOC].).
	char[ ]	caPassphrase	Passphrase de protection du répertoire.
	String	sInputDir	Nom complet du répertoire à chiffrer.
	String	sOutputDir	Nom complet du répertoire de destination des fichiers chiffrés.

Notes :

- **sInputDir** et **sOutputDir** doivent être différents.
- Rappel : Le terme **nom complet** désigne le nom complet d'un répertoire avec les unités.
  - Exemple Windows : **`c:\safelogic\lib`**
  - Exemple Unix : **`/home/safelogic/lib`**
- Les fichiers peuvent être supprimés de **sInputDir** suivant la valeur du paramètre **`DIR_DELETE_LEVEL`.**
- Les deux paramètres NB_RETRY_WHEN_LOCK et SECONDS_PAUSE_WHEN_LOCK ne sont pas applicables à cette API.

Méthode	decryptDirWithPassphrase		
Description	Déchiffre tous les fichiers d'un répertoire. Les fichiers chiffrés doivent avoir été chiffrés avec **encryptDirWithPassphrase**. Les fichiers déchiffrés sont placés dans un répertoire de sortie différent et à spécifier.		
Arguments	String	sAlgorithm	Algorithme à utiliser. (Cf. la liste des valeurs autorisées).
	char[ ]	caPassphrase	Passphrase de protection du répertoire (celle qui a été utilisée avec **encryptDirWithPassphrase**).
	String	sInputDir	Nom complet du répertoire à déchiffrer.
	String	sOutputDir	Nom complet du répertoire de destination des fichiers déchiffrés.

Notes :

- Cf. les notes de **encryptDirWithPassphrase().**

Méthode	encryptDir		
Description	Chiffre tous les fichiers d'un répertoire. Les fichiers chiffrés sont placés dans un répertoire de sortie différent et à spécifier. encryptDir utilise une clé symétrique fabriquée avec **CryptoSym.genSecretKey()**		
Arguments	String	sKey_ID	Key_ID de la clé.
	char[ ]	caPassphrase	Passphrase de protection de la clé sKey_ID.
	String	sInputDir	Nom complet du répertoire à chiffrer.
	String	sOutputDir	Nom complet du répertoire de destination des fichiers chiffrés.

Notes :

- **sInputDir** et **sOutputDir** doivent être différents.
- Les fichiers peuvent être supprimés de **sInputDir** suivant la valeur du paramètre **`DIR_DELETE_LEVEL`.**
- Les deux paramètres NB_RETRY_WHEN_LOCK et SECONDS_PAUSE_WHEN_LOCK ne sont pas applicables à cette API.

Méthode	decryptDir		
Description	Déchiffre tous les fichiers d'un répertoire. Les fichiers chiffrés doivent avoir été chiffrés avec **encryptDir** Les fichiers déchiffrés sont placés dans un répertoire de sortie différent et à spécifier.		
Arguments	String	sKey_ID	Key_ID de la clé.
	char[ ]	caPassphrase	Passphrase de protection de la clé sKey_ID.
	String	sInputDir	Nom complet du répertoire à déchiffrer.
	String	sOutputDir	Nom complet du répertoire de destination des fichiers déchiffrés.

Notes :

- Cf. les notes de **encryptDir().**

5.8.4 Chiffrement asymétrique de répertoires

Le chiffrement asymétrique de répertoires utilise les principes de **chiffrement mixte** et de **gestion de listes de Destinataires** décrits dans 6.2 Utilisation des méthodes en chiffrement asymétrique

Les principes de chiffrement sont identiques à ceux appliqués pour les méthodes de chiffrements asymétriques de fichiers :

- **CryptoAsym.encryptFile().**
- **CryptoAsym.decryptFile().**

En fait, **asymEncryptDir()** appelle pour chaque fichier à chiffrer **CryptoAsym.encryptFile()** et **asymDecryptDir()** appelle pour chaque fichier à déchiffrer **CryptoAsym.decryptFile(),**

Les fonctions **asymEncryptDir()** et **asymDecryptDir()** utilisent des clés RSA fabriquées avec **CryptoAsym.genKeyPair().**

Méthode	**asymEncryptDir**		
Description	Chiffre tous les fichiers en RSA d'un répertoire pour une liste de destinataires passée en argument. Les fichiers chiffrés sont placés dans un répertoire de sortie différent et à spécifier.		
Arguments	String	sListName	Nom de la Liste de destinataires.
	String	sInputDir	Nom complet du répertoire à chiffrer.
	String	sOutputDir	Nom complet du répertoire de destination des fichiers chiffrés.

Notes :

- **sInputPath** et **sOutputPath** doivent être différents.
- Les fichiers peuvent être supprimés de **sInputDir** suivant la valeur du paramètre **`DIR_DELETE_LEVEL`.**
- Les deux paramètres NB_RETRY_WHEN_LOCK et SECONDS_PAUSE_WHEN_LOCK permettent de définir les nouvelles tentatives pour chaque fichier verrouillé par un autre processus.

Méthode	asymDecryptDir		
Description	Déchiffre tous les fichiers d'un répertoire. Les fichiers chiffrés doivent avoir été chiffrés avec **encryptDir** Les fichiers déchiffrés sont placés dans un répertoire de sortie différent et à spécifier.		
Arguments	String	sKey_ID	Key_ID de la clé privée de déchiffrement.
	char[ ]	caPassphrase	Passphrase associée à la clé privée.
	String	sInputDir	Nom complet du répertoire à déchiffrer.
	String	sOutputDir	Nom complet du répertoire de destination des fichiers déchiffrés.

Notes :

- Cf. les notes de **asymEncryptDir().**

6 NOTES SUR L'UTILISATION DES METHODES

6.1 Utilisation des méthodes en chiffrement symétrique

Une séquence de chiffrement doit être précédée d'une génération de clé avec **CryptoSym**.**genSecretKey.**

La clé générée est stockée dans un fichier et associée à une « passphrase » qui sert de moyen de protéger la clé secrète : la « passphrase » est elle même transformée en clé de chiffrement pour chiffrer la clé secrète stockée.

La clé générée est valide pour une durée indéterminée : il est donc possible de séparer les programmes VB de génération de clé, chiffrement et déchiffrement avec les méthodes suivantes de **CryptoSym :**

- **encryptBuffer** ou **EncryptFile** pour le chiffrement.
- **decryptBuffer** ou **DecryptFile** pour le déchiffrement.

6.2 Utilisation des méthodes en chiffrement asymétrique

Il est obligatoire en premier lieu de générer un fichier binaire qui contiendra une série de données aléatoires. Ces données sont vitales et servent de base de données *pseudo-aléatoires* pour la génération de clés de session symétriques de 128 bits.

L'API à utiliser est **CryptoAsym**.**createSeedFile()**

La deuxième étape consiste à générer une paire de clés RSA avec **CryptoAsym**.**genKeyPair().**

La paire de clés (privée, publique) RSA est stockée dans deux fichiers, la clé privée étant chiffrée en mode symétrique et associée à une passphrase qui sert de moyen de protéger la clé privée, qui doit en permanence rester confidentielle.

L'étape suivante consiste à générer une liste de destinataires avec :

- **createRecipients**
- **addRecipients**

Enfin, les fonctions de chiffrement/déchiffrement peuvent être utilisées :

- **encryptFile**
- **decryptFile**

Ainsi que les fonctions de signature et vérification de signature :

- **signFile / rawSignFile**
- **verifyFile / rawVerifyFile**

Pour terminer, il est possible d'effectuer les opérations de chiffrement aysmétrique + signature en une seule opération :

- **encryptAngSign**
- **decryptAndSign**

6.3 Programmes sources d'exemples en Java

SafeAPI est fourni avec les programmes d'exemples en Java dont certains s'exécutent en mode « ligne de commande » :

Nom programme exemple Java	Description
`TestHash.java`	Hash de buffer.
`TestHashFile.java`	Hash de fichier.
`TestSym.java`	• Génération de clés symétriques avec Blowfish, CAST128, et IDEA. • Chiffrement symétrique debuffer. • Chiffrement symétrique de fichier.
`TestRSACrypt.java`	• Génération d'une paires de clés RSA. • Chiffrement / déchiffrement asymétrique de buffer. • Chiffrement / déchiffrement asymétrique de fichier.
`TestRSASign.java`	• Signature de buffer. • Signature de fichier (signature inclus ou détachée). • Chiffrement asymétrique et signature de fichier.

Notes :

- Pour exécuter ces programmes ; il faut adosser à leur nom le package java **`com.safeapi.test`**

- Exemple Windows :

```
c:\> java com.safeapi.test.TestHash
```

- Exemple Unix :

```
$ java com.safelogic.test.TestHash
```

Note : les programmes, projets Java etc. sont fournis à titre d'exemple et ne sont pas supportés par SafeLogic.

6.4 Exemples Visual Basic : projet complet SafeAPI_VB

SafeAPI est fourni avec le projet complet Visual Basic 6.0 **SafeAPI_VB** qui contient une série de fenêtres et procédures pour générer un fichier de données « seed », générer des clés symétriques ou RSA, hacher un buffer ou fichier, chiffrer un fichier, etc.

Note : les programmes, projets, fenêtres VB, etc. sont fournis à titre d'exemple et ne sont pas supportés par SafeLogic.

7 FAQ IMPLEMENTATION/CRYPTOGRAPHIE

7.1 Rôle de ce FAQ

Ce document a pour but de « valider » ou « prouver » la qualité de l'implémentation des algorithmes de cryptographie dans SafeAPI. La lecture de ce document suppose la connaissance des notions de base en cryptographie.

7.2 Comment tester l'implémentation de MD5 ?

Le RFC 1321 définit une suite de tests pour MD5.
Cf. http://www.faqs.org/rfcs/rfc1321.html

Voici la suite complète. Le résultat correspond à la conversion en chaîne hexadécimale du résultat en bytes :

```
MD5 ("")    = d41d8cd98f00b204e9800998ecf8427e
MD5 ("a")   = 0cc175b9c0f1b6a831c399e269772661
MD5 ("abc") = 900150983cd24fb0d6963f7d28e17f72
MD5 ("message digest") = f96b697d7cb7938d525a2f31aaf161d0
MD5 ("abcdefghijklmnopqrstuvwxyz") = c3fcd3d76192e4007dfb496cca67e13b
MD5 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789")
      = d174ab98d277d9f5a5611c2c9f419d9f
MD5 ("12345678901234567890123456789012345678901234567890123456789012345
78901234567890") = 57edf4a22be3c955ac49da2e2107b67a
```

7.3 Comment tester l'implémentation de SHA-1 ?

Le FIPS 180-1 définit deux tests pour SHA-1.
Cf. http://www.itl.nist.gov/fipspubs/fip180-1.htm

```
SHA-1( "abc") =  "A9993E364706816ABA3E25717850C26C9CD0D89D
SHA-1( "abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq")
           = "84983E441C3BD26EBAAE4AA1F95129E5E54670F1"
```

7.4 Comment tester l'implémentation de Blowfish ?

Un jeu de test, recommandé par l'auteur de Blowfish, est défini à. ftp://ftp.psy.uq.oz.au/pub/Crypto/Blowfish/test.data

Blowfish est par ailleurs présenté en détail à :
http://www.counterpane.com/blowfish.html

Blowfish est implémenté dans SafeAPI en mode CFB (Cipher Feedback Block) sans padding.

On peut faire le test CFB :

Objet/donnée	Valeur hexadécimale
Clé en hexadécimal :	0123456789ABCDEFF0E1D2C3B4A59687
IV (Initialisation Vector) en hexa :	FEDCBA9876543210
Chaîne hexa en clair :	37363534333231204E6F7720697320746865207469 6D6520666F722000
Chaîne hexa chiffrée :	E73214A2822139CAF26ECF6D2EB9E76E3DA3DE04D1517200519D57A6C3

7.5 Comment tester l'implémentation de CAST-128 ?

Le RFC 2144 définit un jeu de test pour CAST, mais uniquement avec le mode ECB. Cf. http://www.faqs.org/rfcs/rfc2144.html.

CAST-128 est implémenté en mode CFB dans SafeAPI, il n'y a donc pas de jeu de test « normalisé ».

7.6 Comment tester l'implémentation d'IDEA ?

Les tests IDEA son définis par un fichier téléchargeable d'ASCOM à : http://www.ascom.ch/infosec/downloads.html.

IDEA est implémenté en CFB dans SafeAPI. Voici deux tests extrait du jeu d'essai en mode CFB :

Objet/donnée	Valeur hexadécimale
Clé en hexadécimal :	729A27ED8F5C3E8BAF16560D14C90B43
IV (Initialisation Vector) en hexa :	C121A1B050D8286C
Test 1	
Chaîne C1 hexa en clair :	D53FABBF94FF8B5F
Chaîne C1 hexa chiffrée :	E42323CAF932B933
Test 2	
Chaîne C2 hexa en clair :	94FF8B5F
Chaîne C2 hexa chiffrée :	A5E3032A

7.7 Comment sont générés les nombres pseudo-aléatoires avec SafeAPI ?

Les nombres pseudo-aléatoires sont générés en utilisant l'algorithme ANSI X9.17.

X9.17 est un générateur de nombre pseudo-aléatoire « fort » qui prend en input un couple (valeur random, clé) et le crypte avec une clé fournie par l'utilisateur et algorithme symétrique pour produire un nouveau couple (valeur random, clé) imprévisibles pour un attaquant.

7.8 Comment est généré le IV en CFB avec SafeAPI ?

Le IV (Initialisation Vector) pour le mode CFB est généré en utilisant l'algorithme ANSI X9.17.

7.9 Comment sont stockées les clés secrètes ?

Avant enregistrement sur le disque dur, la clé secrète est chiffrée avec :

- L'algorithme choisi par l'utilisateur dans l'API : Blowfish, CAST-128, IDEA.
- Le mode CFB.
- Une clé de 128 bits qui est la valeur hash MD5 de la passphrase passée comme paramètre à **genSecretKey.**

Les clés secrètes ne sont à aucun moment stockées en clair sur le disque dur.

7.10 Comment sont stockées les clés privées RSA ?

Avant enregistrement sur le disque dur, la clé privée est chiffrée avec :

- L'algorithme Blowfish.
- Le mode CFB.
- Une clé de 128 bits qui est la valeur hash MD5 de la passphrase passée comme paramètre à **genKeyPair.**

Les clés privées RSA ne sont à aucun moment stockées en clair sur le disque dur.

8 SUPPORT

Pour toute question technique, veuillez vous adressez à :

SafeLogic
27/29, rue Raffet
75016 Paris

Tél : (33) (0)1 45 72 25 15
Fax : (33) (0)1 45 72 14 06
E mail : contact@safelogic.com
Web : http://www.safelogic.com
