A-Trust Gesellschaft für Sicherheitssysteme
im elektronischen Datenverkehr GmbH
Landstraßer Hauptstraße 5
A-1030 Wien

https://www.a-trust.at
E-Mail: office@a-trust.at

a.sign RK COM/DLL

Developer Manual

Version: 2.2
Datum: 23. Dezember 2016

Copyright

A-Trust
Gesellschaft für Sicherheitssysteme im elektronischen Datenverkehr GmbH
A-1030 Wien

Leistungsbeschreibung

A-Trust stellt das Produkt a.sign RK COM/DLL zur Verfügung welche die Funktionen zum Zugriff auf die a.sign RK CHIP vereinfacht. Dieses Produkt hat sowohl eine DLL-Schnittstelle als auch eine COM-Schnittstelle (Component Object Model [Mic16]) Es werden Funktionen für den AES-Schlüssel Generierung und Verschlüsselung, Base64 Kodierung , Base32 Kodierung, Sha256 Hash, QR-Code und OCR-Code Generierung bereitgestellt, welche für die Implementierung der Registrierkassensicherheitsverordnung [Bun15] benötigt werden.

Bereitgestellte Funktionen und Programme:

- COM-Schnittstelle für die einfache Einbindung in VB6 und VBA
- DLL-Schnittstelle für die Verwendung in C/C++
- Schnittstelle zum Zugriff auf die Chipkarte
- Funktion für JWS-Signatur der aufbereiten Datenstruktur des Belegs
- Funktion für Signaturprüfung einer DEP Zeile
- Funktion zum Base64 und Base64-URL kodieren von String-Werten
- Funktion zum Base32 kodieren
- Funktionen zum Generieren und Verwenden des AES-Schlüssel zur Verschlüsselung des Umsatzzählers.
- Sha256 Hash Funktion
- Funktionen zum Generieren eines QR-Codes
- Funktionen zum Generieren eines OCR-Codes
- Developer Handbuch mit Funktionsbeschreibung inkl. Visual Basic 6 und C++ Quellcode
- Visual Basic 6 Testprogramm welches den Einsatz der Funktionen des COM Objektes demonstriert
- C++ Testprogramm welches den Einsatz der DLL-Schnittstelle demonstriert
- durchgängiges Beispiel in Visual Basic 6 für die Erstellung beliebiger Belegzeilen mit erfolgreicher Verifikation über A-SIT Plus Testtool
- Sowohl als 32-bit als auch als 64-bit Verfügbar
- Für den Einsatz unter Windows XP wird ein spezieller a.sign Client XP benötigt

Inhaltsverzeichnis

Datum	Rev	Autor	Änderungen
23.12.2016	2.2	Patrick Hagelkruys	Überarbeitung Kapitel 2.1 „Registrierung im Betriebssystem“
06.12.2016	2.1	Patrick Hagelkruys	Fehlerkorrekturlevel für QR-Code Beschreibung der DLL Schnittstelle
11.11.2016	2.0	Patrick Hagelkruys	Farbtiefe für QR-Code
24.10.2016	1.9	Patrick Hagelkruys	AES Prüfsumme hinzugefügt Funktion zum Umkodieren von Base64 und Base64-URL auf Base32 Fehlerbehebung bei Base32 Kodierung
13.06.2016	1.8	Patrick Hagelkruys	OCR Funktionen hinzugefügt
30.05.2016	1.7	Patrick Hagelkruys	FAQ hinzugefügt
17.05.2016	1.6	Patrick Hagelkruys	Umbenennung der Produkte
09.05.2016	1.5	Patrick Hagelkruys	Fehler in Beispielcode ausgebessert
13.04.2016	1.4	Patrick Hagelkruys	Padding in Base64Url
30.03.2016	1.3	Patrick Hagelkruys	QR-Code Funktionen
29.03.2016	1.2	Patrick Hagelkruys	Funktionen für Base64 Dekodierung
23.03.2016	1.1	Patrick Hagelkruys	Umsatzzähler als String für AES Funktionen
15.02.2016	1.0	Patrick Hagelkruys	Rückgabewerte der Funktion erweitert Funktion zum Umkodieren von Base64Url zu Base64
05.02.2016	0.9	Patrick Hagelkruys	Tippfehler in Code-Beispiel a.sign Client Version angepasst
01.02.2016	0.8	Patrick Hagelkruys	Hash Signatur voriger Beleg
28.01.2016	0.7	Patrick Hagelkruys	Review
26.01.2016	0.6	Patrick Hagelkruys	Typo in Quellcodebsp. Returncodes für alle Funktionen Funktion: Software prüfen Funktion: Karte prüfen Schnittstelle Sha256 Leistungsbeschreibung
21.01.2016	0.5	Patrick Hagelkruys	textuelle Änderungen
20.01.2016	0.4	Patrick Hagelkruys	JWS Signatur hinzugefügt
20.01.2016	0.3	Patrick Hagelkruys	neue Schnittstellen Logging
20.01.2016	0.2	Ramin Sabet Robin Balean Patrick Hagelkruys	Review ToBeSigned Data korrigiert Kartenwechsel, neue Schnittstellen
18.01.2016	0.1	Patrick Hagelkruys	Erste Version

Tabelle 1: Dokumentenhistorie

1 Überblick

1.1 Zusammenfassung

Ziel dieses Dokumentes ist die Beschreibung der Schnittstelle des Produktes a.sign RK COM/DLL. Dieses Produkt stellt zwei Schnittstellen zur Verfügung, COM und DLL-Aufrufe.

COM (Component Object Model [Mic16]) ist eine Technologie von Microsoft und ermöglicht die Kommunikation zwischen Softwarekomponenten.

DLL-Aufrufe entsprechen Standard C/C++ Aufrufen.

Das a.sign RK COM/DLL kapselt die Aufrufe zu Erstellung von digitalen Signaturen wie diese in der österreichischen Registrierkassen Sicherheitsverordnung [Bun15] benötigten werden.

Diese Dokumentation entspricht der Version 2.6.0.0 des a.sign RK COM/DLL.

1.2 Voraussetzungen

Für die Verwendung des a.sign RK COM/DLL sind folgende Voraussetzungen zu erfüllen:

- Windows basiertes Betriebssystem (Windows Vista oder neuer)
- a.sign Client in der Version 1.3.2.29c oder neuer
- Kartenleser
- aktivierte a.sign RK CHIP

TEIL I

COM SCHNITTSTELLE

2 Verwendung des a.sign RK COM Objektes

2.1 Registrierung im Betriebssystem

Vor der Verwendung muss des COM Objekt im Betriebssystem registriert werden, dazu ist ein einfacher Kommandozeilenbefehl notwendig. Das a.sign RK COM wird in drei Varianten ausgeliefert:

- 32bit COM Objekt (x86). Für die Verwendung in 32bit Betriebssystemen und für 32bit Applikationen in 64bit Betriebssystemen (z.B.: VB6 Applikationen)
- 64bit COM Objekt (x64). Für die Verwendung in 64bit Applikationen
- Windows XP kompatibles 32bit COM Objekt (xp). Zusätzlich wird der a.sign Client XP benötigt.

32bit Betriebssystem

Für 32bit Windows Betriebssysteme ist folgender Befehl in einer Administrator Konsole auszuführen:

```
regsvr32.exe x86/asignRKCom.dll
```

Registrierung in 32bit Systemen

64bit Betriebssystem jedoch 32bit Applikation

Für das 32bit COM Objekt unter 64bit Windows Betriebssysteme ist folgender Befehl in einer Administrator Konsole auszuführen:

```
c:\Windows\SysWOW64\regsvr32.exe x86/asignRKCom.dll
```

Registrierung in 32bit COM unter 64bit Systemen

64bit Betriebssystem und 64bit Applikation

Für 64bit Windows Betriebssysteme und der Verwendung einer 64bit Applikation ist folgender Befehl in einer Administrator Konsole auszuführen:

```
regsvr32.exe x64/asignRKCom.dll
```

Registrierung in 64bit COM unter 64bit Systemen

Windows XP

Für Windows XP Betriebssysteme ist folgender Befehl in einer Administrator Konsole auszuführen:

```
regsvr32.exe xp/asignRKCom.dll
```

Registrierung in Windows XP

Nach erfolgreicher Registrierung wird der Dialog aus Abbildung 1 angezeigt.

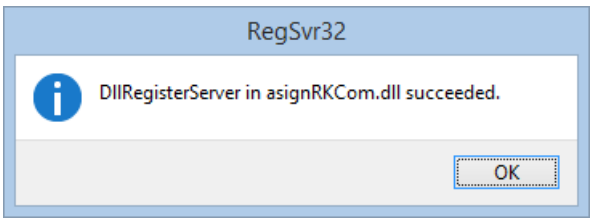

Abbildung 1: COM Objekt Registrierung erfolgreich

2.2 Deregistrierung im Betriebssystem

Zum Entfernen des COM Objektes muss zusätzlich der Parameter `/u` übergeben werden.

```
regsvr32.exe /u asigRKCom.dll
```

Deregistrierung in 32bit Systemen

Die Deregistrierung der 32bit und 64bit COM Objekte erfolgt analog zu der Registrierung in Kapitel 2.1.

Die erfolgreiche Deregistrierung wird durch den Dialog aus Abbildung 2 bestätigt.

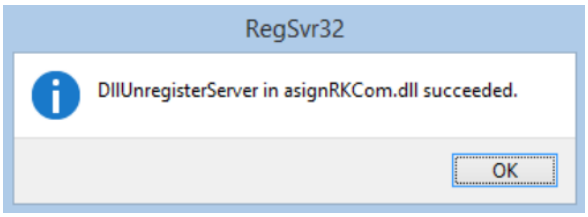

Abbildung 2: COM Objekt Deregistrierung erfolgreich

2.3 Kartenwechsel

Bei einem Kartenwechsel muss das COM Objekt neu geladen werden. Es reicht nicht die LoadInfo Methode (Kapitel 2.4.4) aufzurufen.
Es wird empfohlen das Registrierkassen Programm zu beenden, die Karte zu wechseln und erst dann das Registrierkassen Programm zu starten.

2.4 Schnittstelle Registrierkassen Karte - Methoden und Eigenschaften

2.4.1 Software prüfen

Dieser Befehl prüft ob die notwendige a.sign Client Software in der richtigen Version installiert ist.
Rückgabewert: Long

- **0:** OK
- **2:** Registry Einträge des a.sign Client fehlen. Fehlerhafte Installation?
- **3:** a.sign Client Version nicht ausreichend, bitte aktualisieren
- **4:** a.sign Client kann nicht geladen werden. Fehlerhafte Installation?
- **5:** Allgemeiner Fehler

2.4.2 Initialize

Laden der PKCS#11 Datei und Initialisierung der internen Datenstrukturen im COM Objekt. Dieser Befehl muss nur einmal beim Start des Programmes ausgeführt werden.
Rückgabewert: Long

- **0:** OK
- **1:** a.sign Client konnte nicht geladen werden.
- **2:** a.sign Client DLL ist falsch bzw. Funktionen fehlen?
- **3:** a.sign Client kann nicht initialisiert werden bzw. wurde schon initialisiert.

2.4.3 Karte prüfen

Dieser Befehl prüft ob eine Karte im Kartenleser ist.
Rückgabewert: Long

- **0:** OK
- **1:** Keine aktivierte Karte gefunden

2: Keine Karte gefunden.

2.4.4 LoadInfo

Laden der Zertifikatsdaten von der Karte. Die geladenen Daten werden über die Eigenschaften `ZdaId`, `CertificateSerial`, `Certificate` und `IssuerCertificate` ausgelesen. Dieser Befehl muss nur einmal (z.B.: Programmstart) ausgeführt werden, die Daten der Karte werden im Speicher gehalten.
Rückgabewert: Long

0: OK

1: a.sign Client nicht initialisiert

2: Fehler in a.sign Client

2.4.5 ZdaId

String Wert welcher die ZDA-ID enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.
Rückgabewert: String

2.4.6 CertificateSerial

String Wert welcher die Zertifikatsseriennummer enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.
Rückgabewert: String

2.4.7 CertificateSerialHex

String Wert welcher die Zertifikatsseriennummer im HEX-Format enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.
Rückgabewert: String

2.4.8 Certificate

String Wert welcher das Base64 kodierte Zertifikat enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.
Rückgabewert: String

2.4.9 IssuerCertificate

String Wert welcher das Base64 kodierte Ausstellerzertifikat enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.
Rückgabewert: String

2.4.10 Sign

Durchführen einer Signatur auf der Karte. Der zurückgegebenen Wert `Signature` ist bereits Base64-URL kodiert.
Parameter:

[in]: String; zu signierendes JSON (header + Payload)

[out]: String; Signatur

Rückgabewert: Long

0: OK

1: a.sign Client nicht initialisiert

2: Fehler beim Signieren

2.4.11 SignJWS

Durchführen einer Signatur auf der Karte. Die Funktion bereitet die eingegebenen Daten nach dem JWS Standard auf, d.h. es wird der entsprechende JWS-Header mit dem Algorithmus erzeugt und sowohl Daten als auch Header Base64-URL kodiert. Der zurückgegebene Wert entspricht der JWS Signatur bestehend aus Protected Header, Payload und Signatur jeweils Base64-URL kodiert und durch Punkt getrennt.
Parameter:

[in]: String; zu signierende Belegzeile

[out]: String; Signatur

Rückgabewert: Long; 0=OK

2.4.12 Verify

Durchführen einer Verifikation einer DEP Zeile.
Parameter:

[in]: String; Belegzeile welche signiert wurde

[in]: String; Signatur

Rückgabewert: Long

0: OK

1: a.sign Client nicht initialisiert

-34: Belegzeile startet nicht mit „_R1-AT1“

-35: Zertifikat konnte nicht geladen werden

-36: Fehler bei Hash Berechnung

-37: Signatur ungültig

-38: Fehler beim Parsen des öffentlichen Schlüssel

-39: Fehler beim Parsen des Zertifikates

-40: Fehler beim Parsen des Signaturewerte

-41: Fehler Signaturüberprüfung

2.4.13 VerifyJWS

Durchführen einer Verifikation einer DEP Zeile.
Parameter:

[in]: String; DEP Zeile (JWS)

Rückgabewert: Long

0: OK

1: a.sign Client nicht initialisiert

-33: JWS Header ungültig

-34: Belegzeile startet nicht mit „_R1-AT1“

-35: Zertifikat konnte nicht geladen werden

-36: Fehler bei Hash Berechnung

-37: Signatur ungültig

-38: Fehler beim Parsen des öffentlichen Schlüssel

-39: Fehler beim Parsen des Zertifikates

-40: Fehler beim Parsen des Signaturewerte

-41: Fehler Signaturüberprüfung

2.4.14 Finalize

Freigeben der internen Datenstrukturen im COM Objekt und entladen der PKCS#11 Datei. Dieser Befehl muss nur einmal beim Beenden des Programmes ausgeführt werden.

Rückgabewert: Long

0: OK

1: Fehler

2.5 Schnittstelle AES ICM - Methoden und Eigenschaften

2.5.1 GenerateKey

Generieren eines AES Schlüssel. Dieser Befehl muss nur einmal pro Kasse durchgeführt werden und das Ergebnis durch den Aufrufenden gespeichert werden.
Parameter:

[out]: String; AES Schlüssel

Rückgabewert: Long

0: OK

1: Fehler

2.5.2 Encrypt

Verschlüsselung des Umsatzzählers
Parameter:

[in]: String; AES Schlüssel

[in]: String; Umsatz

[in]: String; KassenId

[in]: String; Belegnummer

[out]: String; Verschlüsselter Umsatzzähler

Rückgabewert: Long

0: OK

1: Fehler

2.5.3 Decrypt

Entschlüsselung des Umsatzzählers. Diese Funktion wird im Regelfall nicht benötigt.
Parameter:

[in]: String; AES Schlüssel

[in]: String; Verschlüsselter Umsatzzähler

[in]: String; KassenId

[in]: String; Belegnummer

[out]: String; Umsatz

Rückgabewert: Long

0: OK

1: Fehler

2.5.4 AesKeyCheckSum

Generieren der Prüfsumme über den AES-Schlüssel. Diese Funktion generiert die optionale Prüfsumme für die Anmeldung des AES-Schlüssels in Finanzonline.
Parameter:

[in]: String; AES Schlüssel

[out]: String; Prüfsumme

Rückgabewert: Long

0: OK

1: Fehler

2.6 Schnittstelle Base64 - Methoden und Eigenschaften

2.6.1 Encode

Base64 Encoding eines String
Parameter:

[in]: String; zu encodierende Daten

[out]: String; encodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.2 EncodeUrl

Base64-URL Encoding eines String
Parameter:

[in]: String; zu encodierende Daten

[in]: Long; Padding, 0=Nein, 1=Ja

[out]: String; encodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.3 ReencodeUrlToNormal

Decodiert einen Base64-URL kodierten String und kodiert diesen neu als Base64 (Normal)

Parameter:

[in]: String; zu reencodierende Daten

[out]: String; reencodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.4 ReencodeNormalToUrl

Decodiert einen Base64 (Normal) kodierten String und kodiert diesen neu als Base64-URL

Parameter:

[in]: String; zu reencodierende Daten

[in]: Long; Padding, 0=Nein, 1=Ja

[out]: String; reencodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.5 Decode

Base64 decoding eines String

Parameter:

[in]: String; zu decodierende Daten

[out]: String; decodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.6 DecodeUrl

Base64-URL Decoding eines String
Parameter:

[in]: String; zu decodierende Daten

[out]: String; decodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.7 ReencodeBase64ToBase32

Decodiert einen Base64 kodierten String und kodiert diesen neu als Base32
Parameter:

[in]: String; zu reencodierende Daten

[out]: String; reencodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.6.8 ReencodeBase64UrlToBase32

Decodiert einen Base64-URL kodierten String und kodiert diesen neu als Base32
Parameter:

[in]: String; zu reencodierende Daten

[out]: String; reencodierte Daten

Rückgabewert: Long

0: OK

1: Fehler

2.7 Schnittstelle Sha256 - Methoden und Eigenschaften

2.7.1 HashString

Sha256 eines String
Parameter:

[in]: String; zu hashende Daten

[out]: String; Daten nach Hashfunktion

Rückgabewert: Long

0: OK

1: Fehler

2.7.2 HashBytes

Sha256 eines Byte Array
Parameter:

[in]: Byte ; zu hashende Daten

[in]: Long ; Länge der zu hashenden Daten

[out]: String; Daten nach Hashfunktion

Rückgabewert: Long

0: OK

1: Fehler

2.7.3 HashSigVorigerBeleg

Sha256 des vorigen Belegs wie in [Bun15, Z4, Sig-Voriger-Beleg] verlangt
Parameter:

[in]: String; zu hashende Daten

[in]: Long ; Länge der zu extrahierenden Bytes (derzeit 8)

[out]: String; Base64 codierte Daten nach Hashfunktion

Rückgabewert: Long

0: OK

1: Fehler

2.8 Schnittstelle QR-Code - Methoden und Eigenschaften

2.8.1 Encode

Erzeugt aus der übergebenen Belegzeile einen QR-Code uns speichert diesen in der Ausgabedatei im BMP Format.
Parameter:

[in]: String; Daten für QR-Code

[in]: String; Ausgabedatei (bmp)

Rückgabewert: Long

0: OK

1: Fehler

2.8.2 EncodeFromJWS

Erzeugt aus der übergebenen JWS-Zeile einen QR-Code uns speichert diesen in der Ausgabedatei im BMP Format.
Parameter:

[in]: String; JWS-Zeile für QR-Code

[in]: String; Ausgabedatei (bmp)

Rückgabewert: Long

0: OK

1: Fehler

2.8.3 SetScaleFactor

Skalierungsfaktor für QR-Code. Der QR-Code wird bei einem Skalierungsfaktor von 1 als 77x77 Pixel ausgegeben und Entsprechend des Faktors vergrößert.
Parameter:

[in]: Long; Skalierungsfaktor

2.8.4 SetMargin

Rand für QR-Code, entsprechend dem übergebenen Wert werden weiße Pixel an allen Seiten eingefügt.
Parameter:

[in]: Long; Margin

2.8.5 SetBitDepth

Farbtiefe für den QR-Code in bit, mögliche Werte sind 1,4,8,16,24,32.
Parameter:

[in]: Long; bit (1,4,8,16,24,32)

2.8.6 SetErrorCorrectionLevel

Fehlerkorrekturlevel für den QR-Code, mögliche Werte sind L,M,Q,H. (siehe auch 6.1)
Parameter:

[in]: String; (L,M,Q,H)

2.9 Schnittstelle OCR-Code - Methoden und Eigenschaften

Für den OCR-Code ist in der RKSV [Bun15, Detailspezifikation Kapitel 14] beschreiben, dass die Base64 Werte im Base32 Format kodiert werden müssen.

2.9.1 Encode

Erzeugt aus der übergebenen Belegzeile eine OCR-Code Zeile.
Parameter:

[in]: String; Daten für OCR-Code

[out]: String; Kodierter Daten OCR-Code

Rückgabewert: Long

0: OK

1: Fehler

2.9.2 EncodeFromJWS

Erzeugt aus der übergebenen JWS-Zeile eine OCR-Code Zeile.
Parameter:

[in]: String; JWS-Zeile für OCR-Code

[out]: String; Kodierter Daten OCR-Code

Rückgabewert: Long

0: OK

1: Fehler

3 Beispiel Verwendung in Visual Basic 6

3.1 Schnittstelle Registrierkassen Funktionen (Kartenzugriff)

```
1  Dim reg As Object
2  Set reg = CreateObject("ATrustRegistrierkasseCom.Registrierkasse")
3  Dim ret As Long
4
5  ret = reg.CheckSoftware
6
7  ret = reg.Initialize
8
9  ret = reg.CheckCard
10
11 ret = reg.LoadInfo
12 Dim sZdaId As String: sZdaId = reg.ZdaId
13 Dim sCertSerial As String: sCertSerial = reg.CertificateSerial
14 Dim sCertificate As String: sCertificate = reg.Certificate
15 Dim sIssuer As String: sIssuer = reg.IssuerCertificate
16
17 Dim ToBeSigned As String
18 Dim Signature As String
19 ToBeSigned = "eyJhbGciOiJFUzI1NiJ9.UjEtQVQxX0RFTU8tQ0FTSC1CT1...A6MjNfMA=="
20 ret = reg.Sign(ToBeSigned, Signature)
21
22 Dim ToBeSigned2 As String
23 Dim Signature2 As String
24 ToBeSigned2 = "_R1-AT1_DEMO-CASH-B...P6PGD2KOQ2==="
25 ret = reg.SignJWS(ToBeSigned2, Signature2)
26
27 If (0 = ret) Then
28   Dim jws_parts() As String
29   jws_parts = Split(Signature2, ".")
30   ' protected_header = jws_parts(0)
31   ' payload = jws_parts(1)
32   ' jws_signature_value = jws_parts(2)
33 End If
34
35 ret = reg.Finalize
```

Zeile 1-2 In den ersten beiden Zeilen wird das COM Objekt erstellt.

Zeile 5 Überprüfen ob die richtige Version des a.sign Client installiert ist.

Zeile 7 Initialisierung der Datenstrukturen im COM Objekt und in der PKCS#11 Schnittstelle zur Karte ist.

Zeile 9 Überprüfen ob eine Karte vorhanden ist.

Zeile 11 Laden der Informationen von der Karte

Zeile 12 - 15 Auslesen der in Zeile 7 geladenen Daten

Zeile 20 Durchführen einer Signatur

Zeile 25 Durchführen einer Signatur nach JWS Standard

Zeile 28-32 Parsen der JWS-Signatur in die einzelnen Teile

Zeile 35 Freigeben der internen Datenstrukturen im COM Objekt und in der PKCS#11 Schnittstelle

3.2 Schnittstelle AES-ICM (Umsatzzähler verschlüsseln)

```
1  Dim comAes As Object
2  Set comAes = CreateObject("ATrustRegistrierkasseCom.AesIcm")
3
4  Dim ret As Long
5  Dim aeskey As String
6  comAes.GenerateKey aeskey
7
8  Dim Umsatz As Long: Umsatz = 2349
9  Dim KassenId As String: KassenId = "Register3874"
10 Dim Belegnummer As String: Belegnummer = "39920034"
11 Dim Encrypted As String
12
13 ret = comAes.Encrypt(aeskey, Umsatz, KassenId, Belegnummer, Encrypted)
14
15 Dim Umsatz2 As String
16 ret = comAes.Decrypt(aeskey, Encrypted, KassenId, Belegnummer, Umsatz2)
17
18 Dim checksum As String
19 res = comAes.AesKeyCheckSum(aeskey, checksum)
```

Zeile 1-2 In den ersten beiden Zeilen wird das COM Objekt erstellt.

Zeile 5-6 Erstmaliges Erstellen eines AES Schlüssels

Zeile 8-11 Initialisieren der Variablen für die Verschlüsselung des Umsatzzählers

Zeile 13 Verschlüsselung des Umsatzzählers

Zeile 15-16 Test durch Entschlüsselung, Umsatz und Umsatz2 sollten jetzt gleich sein

Zeile 18-19 Generieren der Prüfsumme für den AES Schlüssel

3.3 Schnittstelle Base64

```
1  Dim comBase64 As Object
2  Set comBase64 = CreateObject("ATrustRegistrierkasseCom.Base64")
3
4  Dim ret As Long
5  Dim inData As String: inData = "_R1-AT1_DEMO-CASH-B...P6PGD2KOQ2==="
6  Dim outData as String
7  Dim outDataUrl as String
8
9  ret = comBase64.Encode(inData, outData)
10
11 ret = comBase64.EncodeUrl(inData, 0, outDataUrl)
```

Zeile 1-2 In den ersten beiden Zeilen wird das COM Objekt erstellt.

Zeile 4-7 Initialisieren der Variablen für die Base64 Kodierung

Zeile 9 Base64 kodieren eines Strings

Zeile 11 Base64-URL kodieren eines Strings

3.4 Schnittstelle Sha256

```
1  Dim comSha256 As Object
2  Set comSha256 = CreateObject("ATrustRegistrierkasseCom.Sha256")
3
4  Dim ret As Long
5  Dim indata As String: indata = "TEST STRING"
6  Dim inbytes() As Byte
7  Dim outdata As String
8  Dim outdata2 As String
9  Dim outdata3 As String
10 Dim bytesExtrahiert As Long: bytesExtrahiert = 8
11 ' inbytes muss befuellt werden
12
13 ret = comSha256.HashString(indata, outdata)
14
15 ret = comSha256.HashBytes(inbytes(1), UBound(inbytes), outdata2)
16
17 ' indata ist die gesamte jws Signatur des vorigen Belegs
     ([jws_header].[payload].[sig])
18 ret = comSha256.HashSigVorigerBeleg(indata, bytesExtrahiert, outdata3)
```

Zeile 1-2 In den ersten beiden Zeilen wird das COM Objekt erstellt.

Zeile 4-11 Initialisieren der Variablen für die Sha256 Funktion

Zeile 13 Sha256 eines Strings

Zeile 15 Sha256 eines Byte Array

Zeile 18 Sha256 der Signatur des vorigen Belegs

3.5 Schnittstelle QR-Code

```
1  Dim comQr As Object
2  Set comQr = CreateObject("ATrustRegistrierkasseCom.QrCode")
3
4  Dim ret As Long
5  Dim sf As Long: sf = 2
6  Dim margin As Long: margin = 3
7
8  Dim indata As String: indata = "R1-AT1_DEMO-CASH-BOX426_776730..."
9  Dim indata2 As String: indata2 = "eyJhbGciOiJFUzI1NiJ9.UjEtQVQxX0RFTU..."
10 Dim outfile As String: outfile = "c:\\temp\\qr1.bmp"
11 Dim outfile2 As String: outfile = "c:\\temp\\qr2.bmp"
12
13 comQr.SetScaleFactor (sf)
14 comQr.SetMargin (margin)
15
16 ret = comQr.Encode(indata, outfile)
17
18 ret = comQr.EncodeFromJWS(indata2, outfile2)
```

Zeile 1-2 In den ersten beiden Zeilen wird das COM Objekt erstellt.

Zeile 4-11 Initialisieren der Variablen für die QR-Code Funktion

Zeile 13 Setzen des Skalierungsfaktor

Zeile 14 Setzen der Margin

Zeile 16 QR-Code erstellen aus Belegzeile

Zeile 18 QR-Code erstellen aus JWS-Zeile

Abbildung 3: QR-Code Ausgabe

3.6 Schnittstelle QCR-Code

```
1  Dim comOcr As Object
2  Set comOcr = CreateObject("ATrustRegistrierkasseCom.OcrCode")
3
4  Dim ret As Long
5
6  Dim indata As String: indata = "R1-AT1_DEMO-CASH-BOX426_776730..."
7  Dim indata2 As String: indata2 = "eyJhbGciOiJFUzI1NiJ9.UjEtQVQxX0RFTU..."
8  Dim outdata As String: outdata = ""
9  Dim outdata2 As String: outdata2 = ""
10
11 ret = comOcr.Encode(indata, outdata)
12
13 ret = comOcr.EncodeFromJWS(indata2, outdata2)
```

Zeile 1-2 In den ersten beiden Zeilen wird das COM Objekt erstellt.

Zeile 4-9 Initialisieren der Variablen für die OCR-Code Funktion

Zeile 11 OCR-Code erstellen aus Belegzeile

Zeile 13 OCR-Code erstellen aus JWS-Zeile

TEIL II

DLL SCHNITTSTELLE

4 Verwendung des a.sign RK COM/DLL Objektes

4.1 Kartenwechsel

Bei einem Kartenwechsel muss die DLL neu geladen werden. Es reicht nicht die LoadInfo Methode (Kapitel 4.2.4) aufzurufen.
Es wird empfohlen das Registrierkassen Programm zu beenden, die Karte zu wechseln und erst dann das Registrierkassen Programm zu starten.

4.2 Schnittstelle Registrierkassen Karte - Methoden und Eigenschaften

4.2.1 Software prüfen

Dieser Befehl prüft ob die notwendige a.sign Client Software in der richtigen Version installiert ist.

```
long CheckSoftware();
```

Rückgabewert:

- **0:** OK
- **2:** Registry Einträge des a.sign Client fehlen. Fehlerhafte Installation?
- **3:** a.sign Client Version nicht ausreichend, bitte aktualisieren
- **4:** a.sign Client kann nicht geladen werden. Fehlerhafte Installation?
- **5:** Allgemeiner Fehler

4.2.2 Initialize

Laden der PKCS#11 Datei und Initialisierung der internen Datenstrukturen im COM Objekt. Dieser Befehl muss nur einmal beim Start des Programmes ausgeführt werden.

```
void* Initialize();
```

Rückgabewert: Handle der Registrierkassen Klasse, dieses Handle wird für die nachfolgenden Befehle benötigt.

4.2.3 Karte prüfen

Dieser Befehl prüft ob eine Karte im Kartenleser ist.

```
long CheckCard(void* handle);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

Rückgabewert:

0: OK

1: Keine aktivierte Karte gefunden

2: Keine Karte gefunden.

4.2.4 LoadInfo

Laden der Zertifikatsdaten von der Karte. Die geladenen Daten werden über die Eigenschaften `ZdaId`, `CertificateSerial`, `Certificate` und `IssuerCertificate` ausgelesen. Dieser Befehl muss nur einmal (z.B.: Programmstart) ausgeführt werden, die Daten der Karte werden im Speicher gehalten.

```
long LoadInfo(void* handle);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

Rückgabewert:

0: OK

1: a.sign Client nicht initialisiert

2: Fehler in a.sign Client

4.2.5 ZdaId

String Wert welcher die ZDA-ID enthält.
Kann erst nach einem Aufruf 4.2.4 verwendet werden.

```
long ZdaId(void* handle, char* buffer, long *buffersize);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

buffer: Bereits reservierter Speicher für die ZdaId, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

buffersize: Speichergröße von buffer bzw. benötigte Speichergröße

Rückgabewert:

0: OK

100: Buffer ist NULL

101: Buffer zu klein

4.2.6 CertificateSerial

String Wert welcher die Zertifikatsseriennummer enthält.
Kann erst nach einem Aufruf 4.2.4 verwendet werden.

```
long CertificateSerial(void* handle, char* buffer, long *buffersize);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

buffer: Bereits reservierter Speicher für die Zertifikatsseriennummer, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

buffersize: Speichergröße von buffer bzw. benötigte Speichergröße

Rückgabewert:

0: OK

100: Buffer ist NULL

101: Buffer zu klein

4.2.7 CertificateSerialHex

String Wert welcher die Zertifikatsseriennummer im HEX-Format enthält.
Kann erst nach einem Aufruf 4.2.4 verwendet werden.

```
long CertificateSerialHex(void* handle, char* buffer, long *buffersize);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

buffer: Bereits reservierter Speicher für die Zertifikatsseriennummer, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

buffersize: Speichergröße von buffer bzw. benötigte Speichergröße

Rückgabewert:

0: OK

100: Buffer ist NULL

101: Buffer zu klein

4.2.8 Certificate

String Wert welcher das Base64 kodierte Zertifikat enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.

```
long Certificate(void* handle, char* buffer, long *buffersize);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

buffer: Bereits reservierter Speicher für die Zertifikat, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

buffersize: Speichergröße von buffer bzw. benötigte Speichergröße

Rückgabewert:

0: OK

100: Buffer ist NULL

101: Buffer zu klein

4.2.9 IssuerCertificate

String Wert welcher das Base64 kodierte Ausstellerzertifikat enthält.
Kann erst nach einem Aufruf 2.4.4 verwendet werden.

```
long IssuerCertificate(void* handle, char* buffer, long *buffersize);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

buffer: Bereits reservierter Speicher für die Ausstellerzertifikat, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

buffersize: Speichergröße von buffer bzw. benötigte Speichergröße

Rückgabewert:

0: OK

100: Buffer ist NULL

101: Buffer zu klein

4.2.10 Sign

Durchführen einer Signatur auf der Karte. Der zurückgegebenen Wert `Signature` ist bereits Base64-URL kodiert.

```
long Sign(void* ptr, char* toBeSigned, long toBeSignedLen, char*
    Signature, long *SignatureLen);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

toBeSigned: Zu signierendes JSON (header + Payload)

toBeSignedLen: Länge der Eingabedaten

Signature: Bereits reservierter Speicher für die Signatur, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

SignatureLen: Speichergröße von `Signature` bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: a.sign Client nicht initialisiert

2: Fehler beim Signieren

100: Handle ist NULL

101: SignatureLen zu klein

4.2.11 SignJWS

Durchführen einer Signatur auf der Karte. Die Funktion bereitet die eingegebenen Daten nach dem JWS Standard auf, d.h. es wird der entsprechende JWS-Header mit dem Algorithmus erzeugt und sowohl Daten als auch Header Base64-URL kodiert. Der zurückgegebene Wert entspricht der JWS Signatur bestehend aus Protected Header, Payload und Signatur jeweils Base64-URL kodiert und durch Punkt getrennt.

```
long SignJWS(void* ptr, char* toBeSigned, long toBeSignedLen, char*
    Signature, long *SignatureLen);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

toBeSigned: Zu signierende Belegzeile

toBeSignedLen: Länge der Eingabedaten

Signature: Bereits reservierter Speicher für die Signatur, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

SignatureLen: Speichergröße von `Signature` bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: a.sign Client nicht initialisiert

2: Fehler beim Signieren

100: Handle ist NULL

101: SignatureLen zu klein

4.2.12 Verify

Durchführen einer Verifikation einer DEP Zeile.

```
long Verify(void* ptr, char* toBeSigned, long toBeSignedLen, char*
    Signature, long SignatureLen);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

toBeSigned: Belegzeile welche signiert wurde

toBeSignedLen: Länge der Eingabedaten

Signature: Signatur der Belegzeile

SignatureLen: Länge der Signatur

Rückgabewert:

0: OK

1: a.sign Client nicht initialisiert

-34: Belegzeile startet nicht mit „_R1-AT1“

-35: Zertifikat konnte nicht geladen werden

-36: Fehler bei Hash Berechnung

-37: Signatur ungültig

-38: Fehler beim Parsen des öffentlichen Schlüssel

-39: Fehler beim Parsen des Zertifikates

-40: Fehler beim Parsen des Signaturewerte

-41: Fehler Signaturüberprüfung

100: Handle ist NULL

101: SignatureLen zu klein

4.2.13 VerifyJWS

Durchführen einer Verifikation einer DEP Zeile.

```
long VerifyJWS(void* ptr, char* JWSSignature, long JWSSignatureLen);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

toBeSigned: DEP Zeile (JWS)

toBeSignedLen: Länge der DEP Zeile

Rückgabewert:

0: OK

1: a.sign Client nicht initialisiert

-33: JWS Header ungültig

-34: Belegzeile startet nicht mit „_R1-AT1“

-35: Zertifikat konnte nicht geladen werden

-36: Fehler bei Hash Berechnung

-37: Signatur ungültig

-38: Fehler beim Parsen des öffentlichen Schlüssel

-39: Fehler beim Parsen des Zertifikates

-40: Fehler beim Parsen des Signaturewerte

-41: Fehler Signaturüberprüfung

100: Handle ist NULL

101: SignatureLen zu klein

4.2.14 Finalize

Freigeben der internen Datenstrukturen im COM Objekt und entladen der PKCS#11 Datei. Dieser Befehl muss nur einmal beim Beenden des Programmes ausgeführt werden.

```
long Finalize(void* ptr);
```

Paramter:

handle: Handle welches von Initialize zurückgeliefert wurde (siehe 4.2.2)

Rückgabewert: Long

0 OK

1 Fehler

4.3 Schnittstelle AES ICM - Methoden und Eigenschaften

4.3.1 GenerateKey

Generieren eines AES Schlüssel. Dieser Befehl muss nur einmal pro Kasse durchgeführt werden und das Ergebnis durch den Aufrufenden gespeichert werden.

```
long GenerateKey(char* newkey, long *newkeyLen);
```

Paramter:

newkey: Bereits reservierter Speicher für den AES Schlüssel, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

newkeyLen: Länge des AES Schlüssel bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

102: newkeyLen zu klein

4.3.2 Encrypt

Verschlüsselung des Umsatzzählers

```
long Encrypt(char* aeskey, char* Umsatz, char* KassenId, char*
    Belegnummer, char* Encrypted, long* EncryptedLen);
```

Paramter:

aeskey: AES Schlüssel

Umsatz: Umsatz

KassenId: KassenId des Belegs

Belegnummer: Belegnummer des Belegs

Encrypted: Bereits reservierter Speicher für den verschlüsselten Wert, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

EncryptedLen: Länge des verschlüsselten Wertes bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: EncryptedLen zu klein

4.3.3 Decrypt

Entschlüsselung des Umsatzzählers. Diese Funktion wird im Regelfall nicht benötigt.

```
long Decrypt(char* aeskey, char* encryptedData, char* KassenId, char*
    Belegnummer, char* Umsatz, long* UmsatzLen);
```

Paramter:

aeskey: AES Schlüssel

encryptedData: Verschlüsselte Daten

KassenId: KassenId des Belegs

Belegnummer: Belegnummer des Belegs

Umsatz: Bereits reservierter Speicher für den entschlüsselten Umsatz, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

UmsatzLen: Länge des verschlüsselten Wertes bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: UmsatzLen zu klein

4.3.4 AesKeyCheckSum

Generieren der Prüfsumme über den AES-Schlüssel. Diese Funktion generiert die optionale Prüfsumme für die Anmeldung des AES-Schlüssels in Finanzonline.

```
long AesKeyCheckSum(char* aeskey, char* checksum, long* checksumLen);
```

Paramter:

aeskey: AES Schlüssel

checksum: Bereits reservierter Speicher für die AES Checksumme, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

checksumLen: Länge der AES Checksumme bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: checksumLen zu klein

4.4 Schnittstelle Base64 - Methoden und Eigenschaften

4.4.1 Encode

Base64 Encoding eines String

```
long Base64Encode(char* input, long inputLen, char* ouput, long* ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.2 EncodeUrl

Base64-URL Encoding eines String

```
long Base64UrlEncode(char* input, long inputLen, long padding, char*
    ouput, long* ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

padding: Padding, 0=Nein, 1=Ja

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.3 ReencodeUrlToNormal

Decodiert einen Base64-URL kodierten String und kodiert diesen neu als Base64 (Normal)

```
long Base64ReencodeUrlToNormal(char* input, long inputLen, char* ouput,
    long* ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.4 ReencodeNormalToUrl

Decodiert einen Base64 (Normal) kodierten String und kodiert diesen neu als Base64-URL

```
Base64ReencodeNormalToUrl(char* input, long inputLen, long padding, char*
    ouput, long* ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

padding: Padding, 0=Nein, 1=Ja

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.5 Decode

Base64 decoding eines String

```
long Base64Decode(char* input, long inputLen, char* ouput, long* ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.6 DecodeUrl

Base64-URL Decoding eines String

```
long Base64UrlDecode(char* input, long inputLen, char* ouput, long*
    ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.7 ReencodeBase64ToBase32

Decodiert einen Base64 kodierten String und kodiert diesen neu als Base32

```
long ReencodeBase64ToBase32(char* input, long inputLen, char* ouput, long*
    ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.4.8 ReencodeBase64UrlToBase32

Decodiert einen Base64-URL kodierten String und kodiert diesen neu als Base32

```
long ReencodeBase64UrlToBase32(char* input, long inputLen, char* ouput,
    long* ouputLen);
```

Paramter:

input: Zu encodierende Daten

inputLen: Länge der zu encodierenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.5 Schnittstelle Sha256 - Methoden und Eigenschaften

4.5.1 HashString

Sha256 eines String

```
long HashString(char* indata, long indataLen, char* outdata, long*
    outdataLen);
```

Paramter:

input: Zu hashende Daten

inputLen: Länge der zu hashenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.5.2 HashBytes

Sha256 eines Byte Array

```
long HashBytes(char* indata, long indataLen, char* outdata, long*
    outdataLen);
```

Paramter:

input: Zu hashende Daten

inputLen: Länge der zu hashenden Daten

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.5.3 HashSigVorigerBeleg

Sha256 des vorigen Belegs wie in [Bun15, Z4, Sig-Voriger-Beleg] verlangt

```
long HashSigVorigerBeleg(char* SigVorigerBeleg, long SigVorigerBelegLen,
    long BytesExtrahiert, char* outdata, long* outdataLen);
```

Paramter:

SigVorigerBeleg: Zu hashende Daten

SigVorigerBelegLen: Länge der zu hashenden Daten

BytesExtrahiert: Länge der zu extrahierenden Bytes (derzeit 8)

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

102: ouputLen zu klein

4.6 Schnittstelle QR-Code - Methoden und Eigenschaften

4.6.1 Encode

Erzeugt aus der übergebenen Belegzeile einen QR-Code uns speichert diesen in der Ausgabedatei im BMP Format.

```
long QrCode_Encode(char* data, long datalen, char* filename, long
    scalefactor, long margin, long bitDepth, char errorCorrection);
```

Paramter:

data: Daten für QR-Code

datalen: Länge der Daten für QR-Code

filename: Ausgabedatei (bmp)

scalefactor: Skalierungsfaktor für QR-Code. (scalefactor = 1 entspricht 77x77 Pixel)

margin: Rand für QR-Code, entsprechend dem übergebenen Wert werden weiße Pixel an allen Seiten eingefügt.

bitDepth: Farbtiefe für den QR-Code in bit, mögliche Werte sind 1,4,8,16,24,32.

errorCorrection: Fehlerkorrekturlevel für den QR-Code, mögliche Werte sind L,M,Q,H.

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

4.6.2 EncodeFromJWS

Erzeugt aus der übergebenen JWS-Zeile einen QR-Code uns speichert diesen in der Ausgabedatei im BMP Format.

```
long QrCode_EncodeJWS(char* jwsdata, long jwsdatalen, char* filename, long
    scalefactor, long margin, long bitDepth, char errorCorrection);
```

Paramter:

jwsdata: JWS-Zeile für QR-Code

jwsdatalen: Länge der JWS-Zeile für QR-Code

filename: Ausgabedatei (bmp)

scalefactor: Skalierungsfaktor für QR-Code. (scalefactor = 1 entspricht 77x77 Pixel)

margin: Rand für QR-Code, entsprechend dem übergebenen Wert werden weiße Pixel an allen Seiten eingefügt.

bitDepth: Farbtiefe für den QR-Code in bit, mögliche Werte sind 1,4,8,16,24,32.

errorCorrection: Fehlerkorrekturlevel für den QR-Code, mögliche Werte sind L,M,Q,H. (siehe 6.1)

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

4.7 Schnittstelle OCR-Code - Methoden und Eigenschaften

Für den OCR-Code ist in der RKSV [Bun15, Detailspezifikation Kapitel 14] beschreiben, dass die Base64 Werte im Base32 Format kodiert werden müssen.

4.7.1 Encode

Erzeugt aus der übergebenen Belegzeile eine OCR-Code Zeile.

```
long OcrCode_Encode(char* data, long dataLen, char* outdata, long*
    oudataLen);
```

Paramter:

data: Daten für OCR-Code

datalen: Länge der Daten für OCR-Code

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

4.7.2 EncodeFromJWS

Erzeugt aus der übergebenen JWS-Zeile eine OCR-Code Zeile.

```
long OcrCode_EncodeJWS(char* jwsdata, long jwsdataLen, char* outdata,
    long* oudataLen);
```

Paramter:

jwsdata: JWS-Zeile für OCR-Code

jwsdataLen: Länge der JWS-Zeile für OCR-Code

ouput: Bereits reservierter Speicher für die Ausgabedaten, wenn dieser Parameter NULL ist wird nur die Größe des benötigten Speichers zurückgegeben

ouputLen: Länge der Ausgabedaten bzw. benötigte Speichergröße

Rückgabewert:

0: OK

1: Fehler

101: Fehlende Eingabewerte

5 Beispiel Verwendung in C/C++

5.1 Schnittstelle Registrierkassen Funktionen (Kartenzugriff)

```
1  long res = 0;
2
3  res = CheckSoftware();
4  printf("CheckSoftware: %d\n", res);
5
6  void* handle = Initialize();
7
8  res = CheckCard(handle);
9  printf("CheckCard %d\n", res);
10
11 res = LoadInfo(handle);
12 printf("LoadInfo %d\n", res);
13
14 char* buffer = NULL;
15 long buffersize = 0;
16 res = Certificate(handle, NULL, &buffersize);
17 buffer = new char[buffersize + 1];
18 res = Certificate(handle, buffer, &buffersize);
19 buffer[buffersize] = 0x00;
20 printf("Certificate %d: %s\n", res, buffer);
21 delete[] buffer;
22 buffer = NULL;
23
24
25 char input[] = "_R1-AT1_1_1_2016-02-24T10:42:13_0,00_0,0...=";
26 char* signature = NULL;
27 long signaturesize = 0;
28 res = SignJWS(ptr, input, strlen(input), NULL, &signaturesize);
29 signature = new char[signaturesize + 1];
30 res = SignJWS(ptr, input, strlen(input), signature, &signaturesize);
31 signature[signaturesize] = 0x00;
32 printf("SignJWS %d: %s\n", res, signature);
33 delete[] signature;
34 signature = NULL;
```

Zeile 3-4 Überprüfen ob die richtige Version des a.sign Client installiert ist.

Zeile 6 Initialisierung der Datenstrukturen und in der PKCS#11 Schnittstelle, Rückgabe des Handle für die Karten-Klasse.

Zeile 8-9 Überprüfen ob eine Karte vorhanden ist.

Zeile 11-12 Laden der Informationen von der Karte

Zeile 14 - 22 Auslesen des Zertifikates, in Zeile 16 wird zuerst die Größe des benötigten Speichers gelesen und in Zeile 18 der reservierte Speicher übergeben.

Zeile 25-34 Durchführen einer Signatur nach JWS Standard, in Zeile 28 wird zuerst die Größe des benötigten Speichers gelesen und in Zeile 30 der reservierte Speicher übergeben.

5.2 Schnittstelle AES-ICM (Umsatzzähler verschlüsseln)

```
1  long res = 0;
2  std::string key;
3  {
4    char* buffer = NULL;
5    long buffersize = 0;
6    res = GenerateKey(NULL, &buffersize);
7    buffer = new char[buffersize + 1];
8    res = GenerateKey(buffer, &buffersize);
9    buffer[buffersize] = 0x00;
10   printf("GenerateKey␣%d:␣%s\n", res, buffer);
11   key = buffer;
12   delete[] buffer;
13   buffer = NULL;
14 }
15
16
17 char Umsatz[] = "1267517823";
18 char KassenId[] = "Kasse-1";
19 char Belegnummer[] = "Bel-1928";
20 {
21   char* buffer = NULL;
22   long buffersize = 0;
23   res = Encrypt((char*)key.c_str(), Umsatz, KassenId, Belegnummer, NULL,
       &buffersize);
24   buffer = new char[buffersize + 1];
25   res = Encrypt((char*)key.c_str(), Umsatz, KassenId, Belegnummer, buffer,
       &buffersize);
26   buffer[buffersize] = 0x00;
27   printf("Encrypt␣%d:␣%s\n", res, buffer);
28   delete[] buffer;
29   buffer = NULL;
30 }
```

Zeile 4-13 Erstmaliges Erstellen eines AES Schlüssels

Zeile 17-19 Initialisieren der Variablen für die Verschlüsselung des Umsatzzählers

Zeile 21-29 Verschlüsselung des Umsatzzählers

5.3 Schnittstelle Base64

```
1  long res = 0;
2  std::string input = "test hallo test 1";
3  char* buffer = NULL;
4  long buffersize = 0;
5  res = Base64Encode((char*)input.c_str(), input.length(), NULL,
       &buffersize);
6  buffer = new char[buffersize + 1];
7  res = Base64Encode((char*)input.c_str(), input.length(), buffer,
       &buffersize);
8  buffer[buffersize] = 0x00;
9  printf("Base64Encode %d: %s\n", res, buffer);
10 delete[] buffer;
11 buffer = NULL;
```

Zeile 1-4 Initialisieren der Variablen für die Base64 Kodierung

Zeile 5-11 Base64 kodieren eines Strings

5.4 Schnittstelle Sha256

```
1  long res = 0;
2  std::string input = "test hallo test 1";
3  char* buffer = NULL;
4  long buffersize = 0;
5  res = HashSigVorigerBeleg((char*)input.c_str(), input.length(), 8, NULL,
       &buffersize);
6  buffer = new char[buffersize + 1];
7  res = HashSigVorigerBeleg((char*)input.c_str(), input.length(), 8, buffer,
       &buffersize);
8  buffer[buffersize] = 0x00;
9  printf("HashSigVorigerBeleg %d: %s\n", res, buffer);
10 delete[] buffer;
11 buffer = NULL;
```

Zeile 1-4 Initialisieren der Variablen für die Sha256 Funktion

Zeile 5-11 Sha256 der Signatur des vorigen Belegs

5.5 Schnittstelle QR-Code

```
1 long res = 0;
2 std::string input = "_R1-AT1_demokasse42_0_2016-06-13T14:22:2...mDKw==";
3 res = QrCode_Encode((char*)input.c_str(), input.length(), "qr1.bmp",
      1,3,24,'H');
4 printf("QrCode_Encode %d\n", res);
```

Zeile 1-2 Initialisieren der Variablen für die QR-Code Funktion

Zeile 3 QR-Code erstellen aus Belegzeile

Abbildung 4: QR-Code Ausgabe

5.6 Schnittstelle QCR-Code

```
1  long res = 0;
2  std::string input = "_R1-AT1_demokasse42_0_2016-06-13T14:22:2...mDKw==";
3  char* buffer = NULL;
4  long buffersize = 0;
5  res = OcrCode_Encode((char*)input.c_str(), input.length(), NULL,
       &buffersize);
6  buffer = new char[buffersize + 1];
7  res = OcrCode_Encode((char*)input.c_str(), input.length(), buffer,
       &buffersize);
8  buffer[buffersize] = 0x00;
9  printf("OcrCode_Encode␣%d:␣%s\n", res, buffer);
10 delete[] buffer;
11 buffer = NULL;
```

Zeile 1-4 Initialisieren der Variablen für die OCR-Code Funktion

Zeile 5-11 OCR-Code erstellen aus Belegzeile

TEIL III

ALLGEMEIN

6 Allgemeine Punkte zur Verwendung

6.1 Fehlerkorrekturlevel bei QR-Code

Level L (Low): ca. 7% der Daten können wiederhergestellt werden

Level M (Medium): ca. 15% der Daten können wiederhergestellt werden

Level Q (Quartile): ca. 25% der Daten können wiederhergestellt werden

Level H (High): ca. 30% der Daten können wiederhergestellt werden

6.2 Logging

Zur Fehleranalyse kann das Logging des COM-Objektes bzw. der DLL aktiviert werden, dazu müssen in der Registry die entsprechenden Werte eingetragen werden.

```
HKEY_LOCAL_MACHINE\SOFTWARE\A-Trust GmbH\ATrustRegistrierkasseCom
```

Registry Pfad für 32-bit Systeme

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\A-Trust GmbH\ATrustRegistrierkasseCom
```

Registry Pfad für 64-bit Systeme

Nachfolgend die Werte für das Aktivieren des Logging.

```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\A-Trust GmbH\ATrustRegistrierkasseCom]
"LogPath"="c:\\temp\\registrierkassecom.log"
"Log"=dword:00000001
```

Registry Werte

Zum Aktivieren des Logging Eintrages muss das COM-Objekt bzw. die DLL neu geladen werden, es wird empfohlen das Registrierkassen Programm zu beenden und neu zu starten.

7 Frequently asked questions (FAQ)

7.1 Wann sollen Initialize und Finalize aufgerufen werden?

Initialize (Kapitel 2.4.2) muss nur einmal beim Start Ihres Programmes aufgerufen werden. Dementsprechend soll Finalize (Kapitel 4.2.14) nur beim Beenden des Programmes aufgerufen werden.

Auf keinen Fall sollten die Befehle vor bzw. nach jedem Verkaufsvorgang ausgeführt werden!

7.2 Unterschied zwischen Sign und SignJWS

Der Unterschied sind die Eingabeparameter bzw. der Aufwand zur Vorbereitung der Parameter der vom aufrufenden Programm durchgeführt werden muss.

Im Zweifelsfall verwenden Sie die Funktion SignJWS (Kapitel 4.2.11).

Literatur

[Bun15] Bundesministers für Finanzen: *Verordnung des Bundesministers für Finanzen über die technischen Einzelheiten für Sicherheitseinrichtungen in den Registrierkassen und andere, der Datensicherheit dienende Maßnahmen (Registrierkassensicherheitsverordnung, RKSV)*, 2015. https://www.bmf.gv.at/steuern/RKSV.pdf, besucht: 2015-11-16.

[Mic16] Microsoft GmbH: *COM: Component Object Model Technologies*, 2016. https://www.microsoft.com/com/default.mspx, besucht: 2016-01-26.