kleiner kommunikation
max@kleiner.com

1 OpenSSL mit Delphi

OpenSSL ist eine freie Implementierung des SSL/TLS-Protokolls mit deren Chiffrierung und bietet darüber hinaus Funktionen zur Zertifikatsverwaltung einer PKI (Public Key Infrastructure) und zu unterschiedlichen kryptographischen Methoden und Standards. In diesem Bericht erfahren Sie Anwendung und Möglichkeiten mit Delphi und weitere Geheimnisse.

1.1 Secret Stories XXL

OpenSSL wurde ursprünglich im Jahre 1995 durch eine Initiative der Mozilla Foundation im Netscape Browser freigegeben. Seit der Lockerung der US-Exportbestimmung darf man ja auch Browser einsetzen, die starke Kryptographie unterstützen, konkret AES 256 bit.

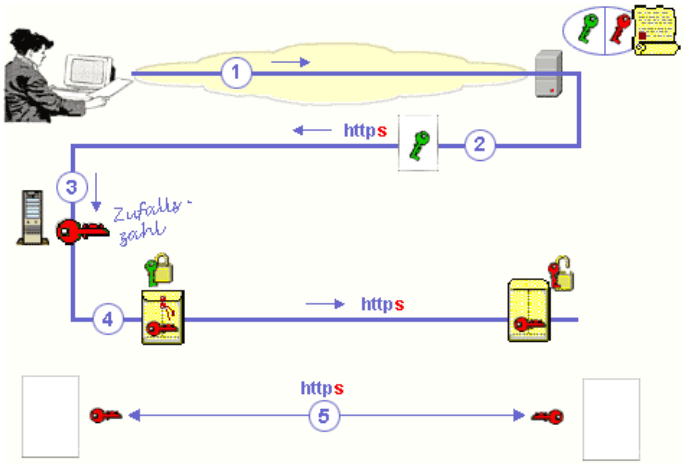
SSL-Chiffrierung wird heute vor allem mit https eingesetzt, jedoch ist auch eine Client/Server Verbindung als TCP-Kanal ohne Browser möglich, wie das Projekt DWS auf sourceforge zeigt.[i] Die meisten Webserver oder Tools unterstützen SSLv3 mit einer Vielzahl von Kryptomethoden, einige Browser / Server setzen jedoch vermehrt den Nachfolger TLS mit RSA- und AES-Verschlüsselung ein, (TLS 1.0 steht neu für SSLv3.1). Das heißt, SSL als Verschlüsselungsprotokoll ist ein hybrides Verfahren kombiniert mit symmetrischen (AES oder DES) und asymmetrischen (RSA) Schlüsseln.

Anhand einer idealisierten Sequenz will ich nun das hybride Verfahren auflisten, welches täglich ohne viel Wissen im Browser transparent funktioniert:

1. Aufbau einer Verbindung https://www.irgendwo.ch mit Port 443
2. Übertragen des Webserver-Zertifikats zum Browser (Maschinenzertifikat)
3. Prüfen der Signatur des Zertifikats anhand des von einer CA hinterlegten Schlüssels, bei Erfolg ist die Identität des Webservers bestätigt
4. Generieren des SessionKey (Zufallszahl) und chiffrieren mit dem erhaltenen Public Key (Zertifikat).
5. Senden des chiffrierten SessionKey zum Webserver
6. Dechiffrieren des SessionKey mit dem Private Key auf Server Seite.

Bis hierhin wirkt das asymmetrische Verfahren (Public/Private Key) wo es eigentlich darum geht, einen symmetrischen SessionKey sicher und authentifiziert zwischen zwei Parteien auszuhandeln und zu transportieren. Schlußendlich folgt:

7. Symmetrische Ver- und Entschlüsselung der Nutzdaten im Client wie beim Server. [1]



Der Vorteil des SSL-Protokolls ist die Möglichkeit, jedes höhere Protokoll (http oder smtp) auf Basis von SSL zu implementieren um eine abgesicherte und integre Verbindung zu gewährleisten. Damit ist

[1] Vor allem aus Performancegründen wirkt ein symmetrischer Schlüssel

eine Unabhängigkeit von Anwendungen und Systemen gewährleistet, jedoch muß eine Exe, Server oder Browser das höhere Protokoll wie https oder ftp over SSL auch implementiert haben. [2] Und dies zeige ich exemplarisch mit dem folgenden https-Server.

Das Aushandeln der Algorithmen und Definieren eines gemeinsamen funktionalen Nenners im Netz ist das eigentliche Geheimnis des Erfolgs von SSL, da in der Regel, außer der Zertifikatsverwaltung, ein Nutzer kaum Wissen benötigt. Jedoch darf man den Mißbrauch durch Täuschung auch nicht von der Hand weisen, hier hilft dann ab und zu ein Blick in das Zertifikat bezüglich Vertrauen und Gültigkeit oder konkreter: man notiert sich den fingerprint (Hashwert).

Das Open-Source-Projekt [ii] wurde als erstes nach FIPS 140-2 zertifiziert. Als SSL von der IETF im RFC 2246 als Standard festgelegt wurde, benannte man es im Jahre 1999 um zu Transport Layer Security (TLS). Die Unterschiede zwischen SSLv3 und TLSv1 jedoch sind nicht groß. Es entstanden aber Versionsmängel. So meldet sich TLS 1.0 im Header als Version SSL 3.1.

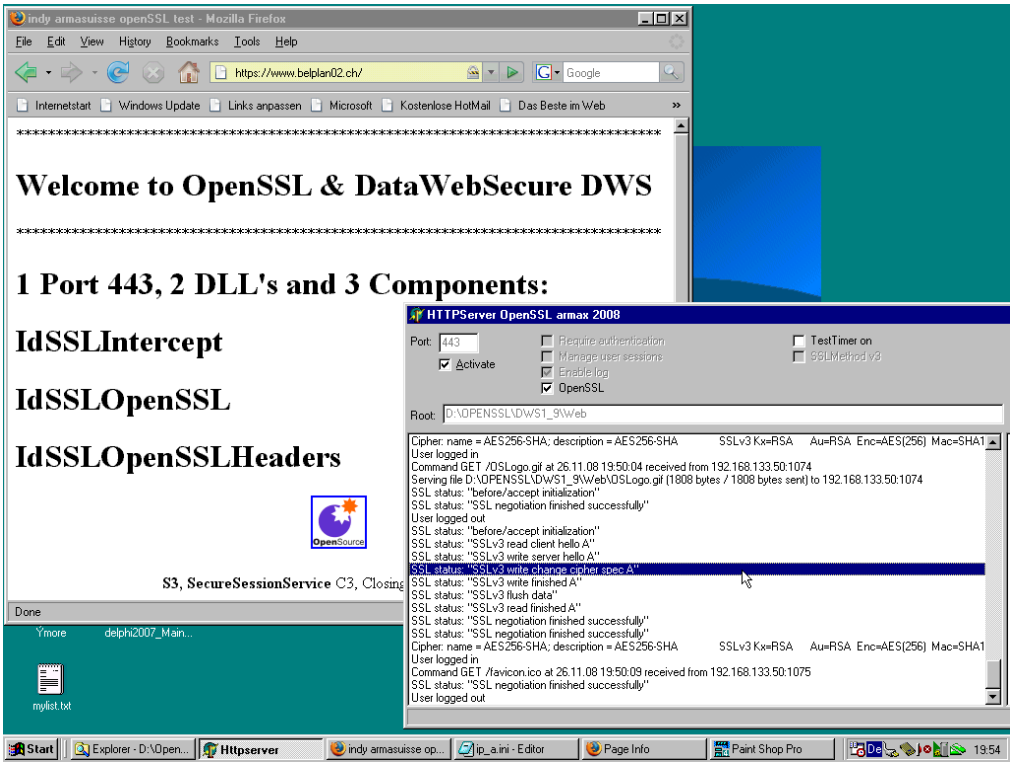


Abb. 1: Der Server verfolg das Protokoll im Monitor

Die in Delphi eingesetzten Komponenten stammen aus den Indy Sockets. Für die Arbeit mit Indy und OpenSSL benötigt man grundlegend zwei DLL's (*ssleay32.dll* und *libeay32.dll*), welche man auch vom DWS-Projekt auf der Basis 0.9.8g beziehen kann oder man compiliert die Sourcen selbst unter folgender Adresse: http./www.intelicom.si/. Ein Geheimnis bleibt unter Indy 10 das Umbenennen der DLL auf libssl32.dll, siehe Header von *IdSSLOpenSSLHeaders.pas*. Diese beiden DLL kopiert man in den Exe-Ordner. Es gibt auch ältere Versionen, die aber führen bei Indy 10.1.5 in der Regel zur Fehlermeldung „could not load SSL library".

Ich geh mal davon aus, viele nutzen noch Delphi 7, so daß die Quellen auf Indy 9.0.17. basieren, ansonsten sei auch ein aktuelles Beispiel mit Delphi 2007 verfügbar, alles auf der CD mitgepackt. [3] Zu D7 läßt sich übrigens auch Indy 10 in einer Subversion nachträglich einbinden.

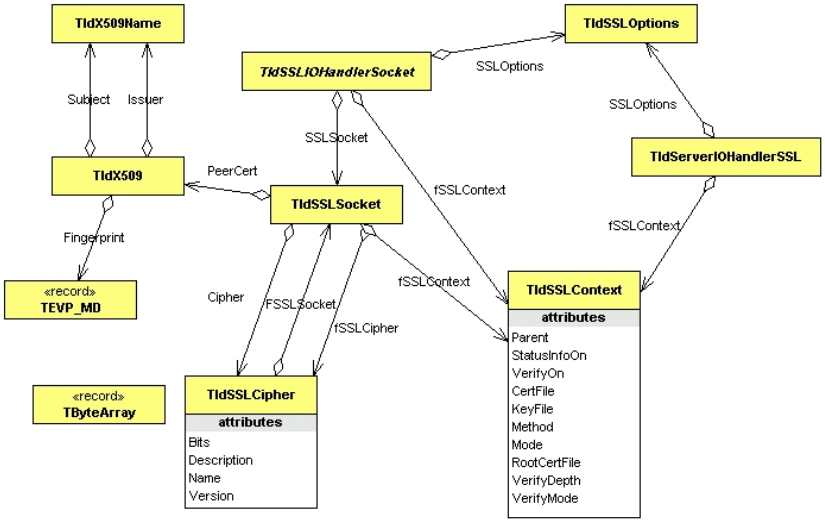
Im Applikationsordner befindet sich eine INI-Datei mit Namen *ip_a.ini*, die verständliche Einträge und Testszenarien aufweist, vor allem den Zertifikatspfad sowie Hostadresse und Portnummer. Die Verwendung von localhost mit 127.0.0.1 ist nicht zwingend erforderlich, ist aber während der Testphase wesentlich einfacher zu handhaben als eine externe URL.

Gehen wir rein in den Digitalen Dschungel. Nachdem die der CD beigelegte Projektdatei im Unterverzeichnis */MySource/httpserver.dpr* kompiliert ist, kopiert man die Exe nach */MyDemo* so daß die

[2] Im Gegensatz zu IPSec, wo der Aufwand eine Ebene tiefer liegt.

[3] openSSL_dws_https_lib_1_9_2.zip

Web und Certs Pfade in der INI-Datei stimmen. Man sollte bereits in der Lage sein, das in Abb. 1 gezeigte Szenario mit localhost aufzurufen!



Der Namensraum der Unit IdSSLOpenSSL

Wenn nicht, kehren wir zu den Quellen zurück ;). In der Formdatei *main.pas* interessiert uns die uses Anweisung. Die beiden Hauptkomponenten sind `TIdServerIOHandlerSSL` und `TIdHTTPServer`. Die erste Komponente verweist auf die Unit `IdSSLOpenSSL` und die wiederum referenziert die Import Unit, welche die beiden DLL's kapselt (siehe Abb. 2).

Die eigentliche Import Unit *IdSSLOpenSSLHeaders.pas* stammt aus den Quellen der Indy Working Group unter www.indyproject.org. Die Unit ist auch für Linux ausgelegt und prüft in einer strengen Und Verknüpfung in der Funktion `load()` ob jeder Funktionszeiger der DLL geladen werden konnte:

```
result:=
    (@IdSslCtxSetCipherList<>nil) and
    (@IdSslCtxNew<>nil) and
    (@IdSslCtxFree<>nil) and
    (@IdSslSetFd<>nil) and
    (@IdSslCtxUsePrivateKeyFile<>nil) and
   (@IdSslCtxUseCertificateFile<>nil) and
.......
```

Beigefügt ist auch eine Hilfsunit `OpenSSLUtils.pas` von Marco Ferrante mit nützlichen Funktionen zur Zertifikatsverwaltung und Schlüsselgenerierung.

Eine weitere Sammlung mit Delphi SSL einzusetzen, sind die seit langem bekannten ICS Komponenten von http://www.overbyte.be, die aber konzeptionell anders funktionieren und mit ICS-SSL eine eigene Adaption verfolgen.

Die Entscheidung zwischen Indy und ICS erfolgt eher mit der Frage ob ich asynchrones Verhalten oder besser synchrones Verhalten wie bei Indy programmieren will. Soll es protokollnah wie bei ICS oder idealer gut gekapselt und bequem wie bei Indy sein? Für Mail- und FTP-Kommunikation ist ICS eine starke Alternative, weil ich da auch besser verstehe und nachvollziehen kann, was eigentlich während der Kommunikation passiert. Als programminterne Kommunikation auf Basis eines direkt auf TCP oder http aufzusetzenden Protokoll ziehen wir aber klar Indy vor.

Werfen wir einen Blick auf die Hauptfunktion. Hier sind die entsprechende URL mit oder ohne SSL sowie weitere Zugriffsattribute als Optionen zu finden. Die Eigenschaften selbst sind im Code zu finden und bestehen aus den Segmenten Konfiguration, ini-Datei Einlesen, Event-Handler zuweisen, Verbindung öffnen und authentifizieren. Mit dem Zuweisen einer Callback Funktion von Indy ist auch das Auflisten des SSL-Protokolls in einer `TListView` vorhanden:

```
  // open SSL connection
if cbSSL.Checked then begin
  with IdServerIOHandlerSSL1.SSLOptions do begin
    Mode:= sslmServer;
```

```
    Method:= sslvSSLv23;
    VerifyDepth:= 3;
    VerifyMode:= [sslvrfFailifNoPeerCert];
    //VerifyMode:= [sslvrfFailifNoPeerCert, sslvrfPeer]; eg. client!
    AppDir:= ExtractFilePath(Application.ExeName);
    RootCertFile:= AppDir + _IniFile.ReadString('CERT', 'ROOTCERT', '');
    CertFile:= AppDir + _IniFile.ReadString('CERT', 'SCERT', '');
    KeyFile:= AppDir + _IniFile.ReadString('CERT', 'RSAKEY', '');
  end;
  HTTPServer.Bindings.Items[0].port:= 443;
   IdServerIOHandlerSSL1.OnStatusInfo:= MyInfoCallback;
   IdServerIOHandlerSSL1.OnGetPassword:= GetKeyPassword;
   IdServerIOHandlerSSL1.OnVerifyPeer:= IdServerIOHandlerSSL1VerifyPeer;
```

Will man statt SSLv23 nur TLS verwenden, ersetzen wir die Zeilen `SSLOptions.Method` durch `SSLOptions.Method:= sslvTLSv1;` und verwenden noch `UseTLS:= utUseExplicitTLS;` in einem Handgriff. `sslvSSLv23` ist ein weiteres Geheimnis, das bedeutet, die optimale Methode der Version wird mit dieser Einstellung zur Laufzeit ausgehandelt.

Eine „zentrale Zeile“ sei jetzt näher beleuchtet. In den Vorgängern von Indy kamen die Interceptoren zum Zug, ab Indy 9 hat sich die I/O Technik eingeschlichen.

```
  HTTPServer.IOHandler:= IdServerIOHandlerSSL1;
```

Die Komponente `TIdServerIOHandlerSSL` wird ja in der Unit `IdSSLOpenSSL` gekapselt. Die Komponente selbst erbt von `class(TIdServerIOHandler)` als Handler mit virtuellen Methoden (Abb. 2) und wir fragen uns was dahinter steckt.

An sich handelt es sich beim IO-Handler von Indy um ein Framework. Man hat hier eine abstrakte Basis um IO (Input/Output) zwischen zwei Indy Komponenten zu realisieren um so den Datenstrom umzulenken. Auch das Protokoll ist ja durch die Komponente gekapselt und so läßt sich mit dem Handler virtuelle Methoden mit eigener Logik überschreiben. Im Framework steckt dahinter eine Indirektion (in einer Tabelle abfragen, welche Methode wirklich verwendet wird) und Indy ruft die eigene Logik auf, ohne daß die restliche Protokoll- und Transport Logik verloren geht.

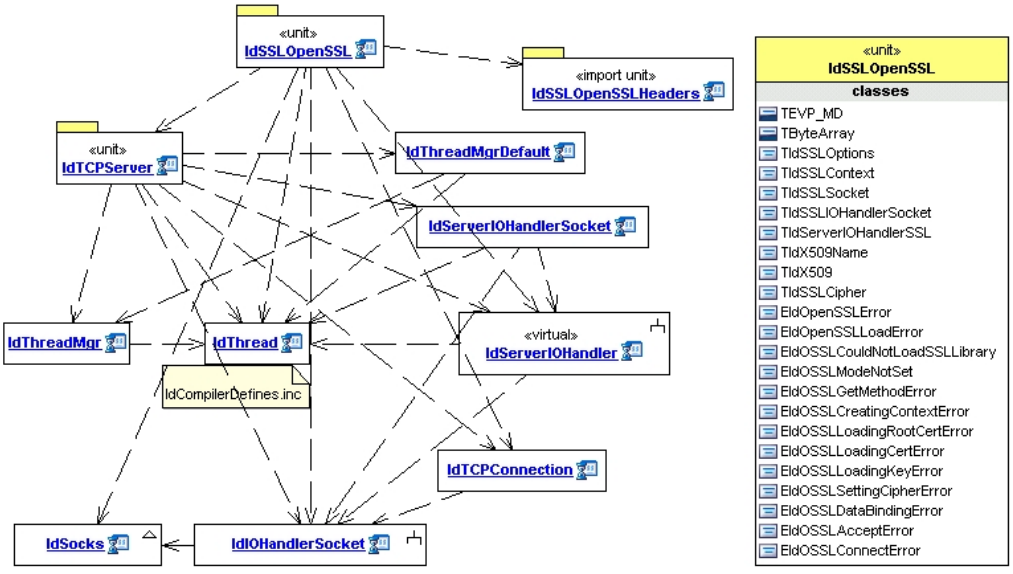


Abb. 2: Die Hauptunit und ihre Abhängigkeiten

1.2 Die Logik der Zertifikate

Ich komme zu den Zertifikaten. Nach dem Starten der Projektdatei ist man ja in der Lage, mit dem Browser eine Verbindung aufzubauen, jedoch sind auf beiden Seiten Zertifikate im Verbund und Einsatz gefragt.

Jede PKI benötigt eine RA (Registrierungsstelle), die umfaßt die Ausgabe digitaler Zertifikate an Benutzer, welche ein Administrator in einem Verzeichnisdienst, z.B. OpenLDAP, verwaltet und geht

nicht selten über die Verwaltung von Mitarbeiterdaten hinaus. Im unserem Beispiel sucht man sich also das Class 3 Zertifikat (root authority) aus und exportiert dieses Base-64-codiert in das Projektverzeichnis. Natürlich nicht irgendein Root-Zertifikat von verisign oder Konsorten, sondern das zum zugehörigen Maschinen-zertifikat *belplan_cert.pem*, das mit dem Private Key des abhängigen Root-Zertifikat *sd_pki_ca.pem* signiert (beglaubigt) wurde.

Eigene Zertifikate sind natürlich mit OpenSSL zu erstellen (siehe *openssl_shell_examples.txt*). Neben dem direkten Aufruf der openssl Kommandos mit Schaltern und Optionen ist noch das Script CA.pl mitgeliefert. Dieses stellt ein vereinfachtes Interface zu den OpenSSL Kommandos dar. CA.pl arbeitet mit einem „Default Configfile“ oder dem File, das in der Umgebungsvariable `OPENSSL_CONF` definiert ist. Eine Root-CA benötigt ein selbstsigniertes Zertifikat, welches man mit folgendem Aufruf erzeugt:

```
  openssl req -new -x509 -keyout ssl_priv.pem -out ca_cert.pem -days 3650 -
config ./openssl.cnf
```

So läßt sich ein Zertifikat mit einem zugehörigen Private Key erzeugen. Das Zertifikat ist für den Zeitraum von 10 Jahren gültig. Man muß sicherstellen, daß der Private Key ausschließlich zum Signieren anderer Zertifikate verwendet wird. Dieser Key ist der kritischste Teil einer CA-Infrastruktur. Der Zertifikatteil (Public Key) dieses Paares wird dann den Clients (Browsern) zur Verfügung gestellt. Die Clients können die Gültigkeit von Zertifikaten, die mit diesem Private Key signiert wurden, eben mit Hilfe des Root-CA Zertifikats (ca_cert.pem) verifizieren.

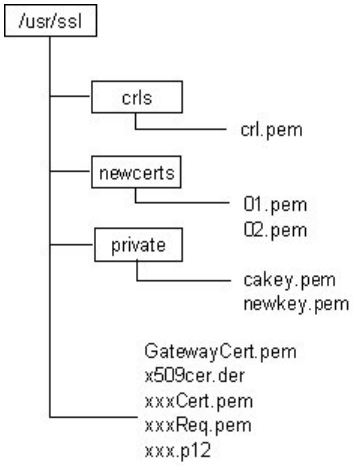


Abb. 3: Verzeichnis einer möglichen Zertifikatsverwaltung

Abschließend ein Tip zur Zertifikatsgenerierung. Wer die Kommandozeilenumgebung scheut, kann mit dem Programm „SSLBuddy“ eigene Zertifikate und Schlüsseldateien generieren. Das Tool unterstützt generell die Technologien um Indy und IntraWeb bei Zertifikaten und deren Aktualisierung. Um das rund 3 MByte große Programm mit Installationsroutine zu beziehen, sei hier der Link:

http://www.arcanatech.com/downloads/SSLBuddySetup.exe

Ich wünsche frohes 2009 und bis zur nächsten Folge „OpenVPN mit Delphi“.

Max Kleiner

Links:

[1] OpenSSL-Vortrag: http:// www.softwareschule.ch/download/openssl_delphi_2009.pdf

[2] DWS-Sourcen: http://sourceforge.net/projects/delphiwebstart

[3] http://de.wikipedia.org/wiki/Transport_Layer_Security

[i] DWS: http://sourceforge.net/projects/delphiwebstart

[ii] OpenSSL-Projekt: http://www.openssl.org