Bundesamt
für Sicherheit in der
Informationstechnik

Eine Studie im Auftrag des
Bundesamtes für Sicherheit in der Informationstechnik

Projekt 154

Quellcode-basierte Untersuchung von kryptographisch relevanten Aspekten der OpenSSL-Bibliothek

Arbeitspaket 1:
Dokumentation von OpenSSL

Version 1.0.1 / 2015-11-03

Sirrix AG
security technologies

Zusammenfassung

Die Kryptobibliothek OpenSSL wird seit mehreren Jahren in verschiedenen kryptographischen Systemen eingesetzt. Sie ist besonders wichtig bei gesicherten Datenübertragungen in Computernetzwerken und spielt aufgrund ihrer großen Verbreitung im Internet eine besonders wichtige Rolle bei der Verwendung von HTTPS (HTTP over SSL/TLS). OpenSSL bietet eine große Zahl von kryptographischen Funktionen an; dazu gehören z.B. symmetrische und asymmetrische Datenverschlüsselung, digitale Signaturen, Authentisierung, Hashfunktionen, Erstellung und Verifikation von Zertifikaten inklusive Zertifikatsketten, Mechanismen zum Integritätsschutz, Schlüssel- und Zufallszahlenerzeugung. Die wichtigste Aufgabe von OpenSSL ist es jedoch, eine Implementierung des TLS-Protokolls (früher SSL-Protokoll) zu sein. Damit stellt die Bibliothek sowohl grundlegende Kryptofunktionen als auch das High-Level-Protokoll TLS zur Verfügung.

Dieser Bericht ist das Ergebnis der Untersuchung der Schnittstellen der OpenSSL-Bibliothek. Es werden die Architektur sowie die Abhängigkeiten zwischen den Komponenten und Funktionen beschrieben, um ein möglichst vollständiges Verständnis der Bibliothek zu erhalten.

Autoren

Wolfgang Meyer zu Bergsten (MzB), Sirrix AG
René Korthaus (RK), Sirrix AG
Jörg Schwenk (Jsc), 3curity GmbH
Juraj Somorovsky (Jso), 3curity GmbH

Quellcode-basierte Untersuchung von kryptographisch relevanten Aspekten der OpenSSL-Bibliothek

OpenSSL 1.0.1g

Inhaltsverzeichnis

1 Beschreibung der Komponenten von OpenSSL

In diesem Kapitel wird der Sourcecode der OpenSSL-Bibliothek analysiert und eine Dokumentation der enthaltenen Komponenten und Module erstellt. Zunächst werden statistische Eigenschaften des Quelltextes dargestellt. Diese Darstellung hilft bei einer ersten groben Einschätzung der Komplexität der Software und ermöglicht erste Einschränkungen bezüglich der zu analysierenden sicherheitskritischen Komponenten. Im zweiten Teil wird diese Einteilung bezüglich der Verzeichnisse und Softwarekomponenten verfeinert, woraus sich ein weiter verschärfter Überblick und daraus der Fokus ergibt. Der dritte Abschnitt beschreibt die Komponenten auf der Architekturebene und geht auf grundsätzliche Softwareblöcke ein. Der vierte Teil beschreibt schließlich die wichtigen Komponenten der OpenSSL-Bibliothek auf der Schnittstellen-Ebene und soll damit zum Verständnis der Programmierphilosophie sowie der API beitragen.

Es wird, wo es sinnvoll erscheint, auf Abhängigkeiten untereinander und das Zusammenspiel der Komponenten miteinander eingegangen. Auffällige Eigenschaften der Software sowie Empfehlungen für den Einsatz oder Verbesserungsmöglichkeiten werden deutlich erkennbar angesprochen.

1.1 Allgemeine Quelltext-Eigenschaften

Das OpenSSL Projekt beinhaltet insgesamt rund 570000 Zeile Code.

Die Funktionalität ist zum größten Teil in C implementiert (rund 390000 Zeilen Code), kleine Teile aber auch in Assembler (rund 13600 Zeilen) sowie C++ (rund 11300 Zeilen).

Das verwendete Build System ist “make” [MAKE] mit einem OpenSSL-spezifischen, in Perl geschriebenen Konfigurationsskript (“Configure”). Hierauf wird im folgenden genauer eingegangen (s. Abschnitt 1.1.1). Für ein Testprogramm (demos/tunala/) werden die GNU autotools [GNUAM] genutzt.

Sprache	Dateien	Quelltext	Kommentare	Kommentare %	Leer	Gesamt	Kommentar
C	1081	271994	82404	23,3	38901	393299	s. Abschnitt 1.2
Perl	518	69597	34071	32,9	21280	124948	s. Abschnitt 1.1.3
Make	75	13606	426	3	1707	15739	Build-System
Assembler	13	10864	1439	11,7	1311	13614	Nur nicht untersucht Architekturen
C++	36	7144	1690	19,1	2522	11356	Nur MacOS, ansonsten “C Code”[1]

Sprache	Dateien	Quelltext	Kommentare	Kommentare %	Leer	Gesamt	Kommentar
Dcl	39	6658	5440	45.0	297	12395	Nur VMS
Shell	35	1443	390	21,3	376	2209	Buildsystem, Utilities, Demos
Bat	33	1254	133	9,6	417	1804	Build Windows
Scheme	7	83	0	0	0	83	Fehlerhaft kategorisiert
Emacslisp	1	24	19	44,2	2	45	Indent Style
Autoconf	1	15	8	34,8	6	29	Demo “tunala”
Automake	1	4	1	20	2	7	Demo “tunala”
HTML	1	3	2	40	2	7	OpenSSL Website Button
Summe	**1841**	**382689**	**126023**	**24,8**	**66823**	**575535**	

1.1.1 OpenSSL Build-System

Das Buildsystem von OpenSSL wird durch das Perl-Script “Configure” konfiguriert. Darauf folgend erstellt “make depend” die Abhängigkeiten des Projekts und “make” schlussendlich die Bibliothek.

1.1.2 Konfiguration in Ubuntu 14.04 (OpenSSL 1.1.0f)

Für das Ubuntu-Paket werden vor dem Build verschiedene Patches angewendet:

ca.patch	gnu_source.patch	openssl_fix_for_x32.patch
config-hurd.patch	c_rehash-compat.patch	fix-pod-errors.patch
debian-targets.patch	libdoc-manpgs-pod-spell.patch	req_bits.patch
engines-path.patch	libssl-misspell.patch	perlpath-quilt.patch
man-dir.patch	openssl-pod-misspell.patch	ppc64-support
man-section.patch	pod_req_misspell2.patch	CVE-2014-0076.patch
no-rpath.patch	pod_pksc12.misspell.patch	CVE-2014-0160.patch
no-symbolic.patch	pod_s_server.misspell.patch	CVE-2010-5298.patch
pic.patch	pod_x509setflags.misspell.patch	CVE-2014-0198.patch
valgrind.patch	pod_ec.misspell.patch	CVE-2014-0195.patch
rehash-crt.patch	pkcs12-doc.patch	CVE-2014-0221.patch
rehash_pod.patch	dgst_hmac.patch	CVE-2014-0224-1.patch
shared-lib-ext.patch	block_diginotar.patch	CVE-2014-0224-2.patch
stddef.patch	block_digicert_malaysia.patch	CVE-2014-3470.patch
version-script.patch	defaults.patch	CVE-2014-0224-3.patch

1 Der C++ Anteil beschränkt sich auf die Definition von “extern C” Interfaces in Header-Files sowie den Dateinamen (Endung .cpp)

Das Paket wird in Ubuntu wie folgt konfiguriert:

```
./Configure --prefix=/usr --openssldir=/usr/lib/ssl --libdir=lib/x86_64-linux-gnu no-idea no-mdc2 no-rc5 no-zlib enable-tlsext no-ssl2 enable-ec_nistp_64_gcc_128 enable-ec_nistp_64_gcc_128 debian-amd64
```

1.1.3 Einsatz von Perl

Der Perl-Code besteht aus 518 Dateien.

- Hiervon befinden sich 336 Dateien Dokumentation im ".pod"-Dateiformat [POD], und sind für die Prüfung von OpenSSL nicht relevant.
- 32 Dateien sind Hilfsfunktionen im "util"-Verzeichnis. Dies sind für die Prüfung von OpenSSL nicht relevant.
- 97 Dateien befinden sich in "asm"-Verzeichnissen unterhalb des "crypto"-Verzeichnisses und erzeugen Assembler-Code für Kryptografische Primitiven (Symmetrische Chiffren, Operationsmodi, Hashfunktionen) und BigNum-Arithmethik.
- 20 weitere Dateien im "crypto" Verzeichnis stellen Hilfsfunktionen für die genannten Code-Generatoren bereit, wie z.B. Erzeugung von verschiedenen Assemblerformaten oder architekturspezifischen Zugriff auf Cycle Counter.
- Die restlichen 33 Dateien sind plattformspezifische Hilfsskripte sowie Tests von Basisfunktionalität der Umgebung und für die Prüfung von OpenSSL nicht relevant.

1.1.4 Organisation des Quelltexts

Der OpenSSL-Code enthält die folgenden Top-Level-Directories:

Verzeichnis	Bemerkung	relevant
crypto	Kryptografie-Implementierungen, siehe Abschnitt 1.2	Ja
ssl	TLS/SSL-Implementierung, siehe Abschnitt 1.2	Ja
engines	Kryptografie-Module, siehe Abschnitt 1.2	Ja
include	Links auf header-Files in crypto	Referenz
apps	OpenSSL Kommandozeilenwerkzeuge	Eventuell
test	Testprogramme, Testdaten	Referenz
doc	OpenSSL Dokumentation	Referenz
bugs	Liste von Bugs	Referenz
times	Benchmarks von Chiffren auf verschiedene Systemen	Nein
certs	Testzertifikate	Nein
demos	Alte Demo-Programme, nicht funktional	Nein
tools	c_rehash Tool	Nein
util	Hilfsfunktionen für Build	Nein
shlib	Plattformspezifische Skripte zum Erstellen von Shared Libraries	Nein
MacOS	Systemspezifisch MacOS – pre Mac OS X	Nein
Netware	Systemspezifisch Netware	Nein
VMS	Systemspezifisch VMS	Nein
ms	Systemspezifisch Windows – Visual C++ 4	Nein
os2	Systemspezifisch OS/2	Nein

Der weitere Fokus liegt auf der Untersuchung der “crypto”-Module sowie der “ssl”-Implementierung.

1.2 Implementierung von OpenSSL

OpenSSL ist vor allem in C implementiert. Ein weiterer bedeutender Teil besteht aus Assemblercode, der von Perlskripten generiert wird. Sofern es relevant ist, wird die Hardwarearchitektur auf Intel x86 und AMD64 beschränkt.

1.2.1.1 Verzeichnis crypto

In “crypto” gibt es die folgenden Verzeichnisse:

Verzeichnis	Bemerkung	relevant
Arithmetik Bibliothek (1)		
bn	BigNum-Arithmetik auf beliebig langen Integer-Werten	Ja
Blockchiffren (9)		
aes	AES	Ja
bf	Blowfish	Nein
camelia	CAMELIA	Nein
cast	CAST	Nein
des	DES	Nein
idea	IDEA	Nein
rc2	RC2	Nein
rc5	RC5	Nein
seed	SEED	Nein
Stromchiffren (1)		
rc4	RC4	Nein
Hashfunktionen (7)		
md2	MD2	Offen (evtl. Zertifikatsverifikation)
md4	MD4	Nein
md5	MD5	Offen (evtl. Zertifikatsverifikation)
ripemd	RIPEMD	Nein
whrlpool	Whirlpool	Nein
sha	SHA-1, SHA-2	Ja
mdc2	MDC2	Nein
MAC Konstruktion (2)		
cmac	CMAC	Nein
hmac	HMAC	Ja
Asymmetrische Primitiven (6)		
dh	Diffie Hellman	Ja

Verzeichnis	Bemerkung	relevant
dsa	DSA Signatur	Ja (Zertifikatsverifikation)
rsa	RSA	Ja
ec	Elliptic Curve	Ja
ecdh	Elliptic Curve Diffie Hellman	Ja
ecdsa	Elliptic Curve DSA	Ja
Operationsmodi für Blockchiffren (1)		
modes	CBC, CCM, CFB, CTR, CTS, GCM, OFB, XTS	Ja (CBC, GCM)
RNG (1)		
rand	Kryptografisch sicherer RNG	Ja
Zertifikate (6)		
asn1	Verarbeitung von ASN.1-kodierten Daten	Ja
pem	Verarbeitung von PEM-kodierten Daten	Ja
pkcs12	Verarbeitung von PKCS12-Containern	Nein
pkcs7	Verarbeitung von PKCS7-Containern (S/MIME)	Nein
x509	Verarbeitung von Zertifikaten	Ja
x509v3	Verarbeitung von Zertifikaten	Ja
Programmierhilfsmittel (17)		
threads	Multithreading Tests	Nein
store	STORE Typ	Nein
stack	STACK Datenstruktur	Nein
pqueue	DTLS pqueue	Nein
objects	Objekt-Bibliothek	Nein
buffer	Buffer Datenstruktur	Nein
conf	Lesen und Verarbeiten von	Nein

Verzeichnis	Bemerkung	relevant
	Konfigurationsdateien	
dso	Shared-Object Abstraktionslayer	Nein
engine	Integration externer Kryptografie-Module	Ja
err	Genereller Code für Fehlerbehandlung	Nein
perlasm	Hilfsfunktionen für Assembly-Erzeugung	Nein
bio	I/O Abstraktion	Nein
evp	High-Level kryptografische Funktionen	Ja
comp	Kompressions-Bibliothek	Nein
lhash	Implementierung einer Hash-Table	Nein
txt_db	Datenbank	Nein
ui	Textbasierte User Interface Bibliothek	Nein
Kommandozeilenwerkzeuge (2)		
cms	S/MIME	Nein
ts	Time Stamp Authority	Nein
Protokolle (4)		
krb5	Kerberos 5 ASN1 Wrapper	Nein
ocsp	Online Certificate Status Protocol	Eventuell
jpake	J-Pake Key Exchange Protokoll	Nein
srp	Secure Remote Password Protocol	Nein

Arithmetikbibliothek

Die BigNum-Bibliothek stellt die grundlegenden Funktionen für die Ganzzahl-Arithmetik mit beliebig großen Zahlen bereit.

Blockchiffren

OpenSSL bietet für die AES-Chiffre sowohl eine auf allen Plattformen lauffähige C-Implementierung als auch verschiedene Assembler-Implementierungen. Hierbei sind die Intel AES-NI Implementierung sowie optimierte Assembler-Implementierungen (AMD64, x86) hervorzuheben.

Die Einbindung in OpenSSL ist in Abschnitt 1.4.10 beschrieben.

Hashfunktionen

Für die relevanten Hashfunktionen (md2, md5, sha) existiert jeweils eine C-Implementierung. Darüber hinaus gibt es für MD5 und die SHA-Familie optimierte Assembler-Implementierungen (AMD64, x86).

Die Einbindung in OpenSSL ist in Abschnitt 1.4.10 beschrieben.

Asymmetrische Primitiven

Die relevanten asymmetrischen Primitiven (DSA, RSA, EC, ECDH, ECDSA) sind mit Hilfe der BN-Bibliothek in C implementiert.

Die Einbindung in OpenSSL ist in Abschnitt 1.4.10 beschrieben.

Operationsmodi für Blockchiffren

Von den implementierten Modes of Operation sind lediglich CBC und GCM relevant. Diese sind in C implementiert. Zusätzlich gibt es für die GHASH-Funktion des GCM optimierte Assembler-Implementierungen (AMD64, x86).

Die Einbindung in OpenSSL ist in Abschnitt 1.4.10 beschrieben.

Zufallszahlengenerator

Der PRNG ist Betriebsystemabhängig in C implementiert. Unter anderem werden Linux (rand_unix.c) und Windows unterstützt (rand_win.c).

Die Einbindung in OpenSSL ist in Abschnitt beschrieben.

Zertifikatsverarbeitung

Die Zertifikatsverarbeitung (asn1, pem, x509v3) ist in C implementiert.

Die Einbindung in OpenSSL ist in Abschnitt 1.4.6 beschrieben.

Integration externer Kryptografie-Module

Die Integration externer Kryptografie-Module (Engines) wird über ein C-Interface ermöglicht. Im Verzeichnis “engines” werden Kryptografie-Module mitgeliefert. In der Standardkonfiguration mit “./config; make” (was zu “./Configure linux-x86_64” evaluiert) werden keine Engines installiert.

Die Einbindung in OpenSSL ist in Abschnitt 1.4.9 beschrieben.

Bemerkung: In der Ubuntu-Standardinstallation werden verschiedene Engines installiert (lib4758cca.so, libaep.so, libatalla.so, libcapi.so, libchil.so, libcswift.so, libgmp.so, libgost.so, libnuron.so, libpadlock.so, libsureware.so, libubsec.so)

High-Level Kryptografiefunktionen

Die Bibliothek für das Erstellen von High-Level Kryptografiefunktionen (evp) ist in C implementiert.

Die Einbindung in OpenSSL ist in Abschnitt 1.4.10 beschrieben.

1.2.1.2 Verzeichnis ssl

Das Verzeichnis „ssl“ beinhaltet die Implementierung der TLS- und SSL-Protokollstacks. Für dieses Protokoll liegt der Fokus auf dem TLS-1.2-Stack.

Die Einbindung in OpenSSL ist in Abschnitt 1.4.7 beschrieben.

1.3 Test Suite

Die Test-Suite kann durch “make test” ausgeführt werden. Es wird vornehmlich die Funktionalität der Implementierung der Low-Level Funktionen getestet. Es gibt wenig Systemtests und keine Tests, bei denen die Software aus der Sicht eines Angreifers betrachtet wird.

1.3.1 Blockchiffren und deren Modes of Operation

Für die zu untersuchende Blockchiffre AES in den Betriebsmodi CBC testet das Programm “test/evp_test” die Ver- und Entschlüsselung im Betriebsmodus CBC mit Testvektoren aus [SP800]. Darüber hinaus wird die Basiskomponente ECB mit Testvektoren aus [FIPS197] und [SP800] getestet.

Testprogramm	Getestete Chiffren
test/evp-test evptest.txt	AES-[128|192|256]-[CBC|ECB]

Die vorhandenen Tests stellen sicher, dass die Implementierungen der AES-Varianten mit unterschiedlichen Schlüssellängen im CBC- und ECB-Modus korrekt sind. Für den GCM-Modus gilt dieses jedoch nicht.

Bemerkung: Die Nichtexistenz von Tests erschwert die kontinuierliche Sicherstellung der Softwarequalität. Darüber hinaus legt es die Schwelle für eine Bemerkung der Implementierung durch Dritte höher, da diesen keine Grundlage für die Funktionalität der Software gegeben wird.

Empfehlung: Wir empfehlen, Tests für den Betriebsmodus AES-[128|192|256]-GCM hinzuzufügen. Diese sollten auf Basis der Spezifikation implementiert werden [GCM] und als Basis den exis-

tierenden Selbst-Test nutzen.

1.3.2 Hashfunktionen

Für die SHA2-Produktfamilie, SHA1, MD2 und MD5 existieren Testprogramme.

Testprogramm	Getestete Hashfunktionen
`test/shatest`	SHA1
`test/sha256t`	SHA224, SHA256
`test/sha512t`	SHA384, SHA512
`test/md5test`	MD5
`test/md2test`	MD2
`test/evp-test`	MD2, MD5, SHA1

Diese Tests stellen sicher, dass die Implementierung der Hashfunktionen funktional korrekt sind.

1.3.3 MAC Konstruktion

OpenSSL testet die HMAC-Konstruktion mit der MD5 Hashfunktion.

Testprogramm	Getestete MAC-Konstruktion
`test/hmactest`	HMAC mit MD5

Bemerkung: Der Test unterstützt nur HMAC-MD5.

Empfehlung: Erweiterung des Test auf HMAC-SHA[224|256|384|512].

1.3.4 Asymmetrische Primitiven

Die Asymmetrischen Primitiven werden mit den folgenden Programmen getestet.

Testprogramm	Getestete Primitiven
`test/rsa_test`	RSA
`test/dsatest`	DSA
`test/dhtest`	Diffie Hellman
`test/ectest`	Elliptic Curves
`test/ecdhtest`	EC Diffie Hellman
`test/ecdsatest`	EC DSA

1.3.4.1 RSA

Es wird die Ver- und Entschlüsselung mit OAEP und PKCS#1 v1.5 Padding getestet. Für das Padding wird SHA1 als Hashfunktion genutzt. Die Testschlüssel sind 400, 512 sowie 1024 Bit lang.

Signatur- und Verifikationsfunktionen werden nicht getestet.

1.3.4.2 DSA

Es wird die Erzeugung von DSA-Parametern (Primfaktoren p und q, sowie ein Generator G), die Erzeugung einer Signatur und deren Verifikation getestet.

1.3.4.3 Diffie Hellman

Es wird die Erzeugung des primen Modulus p, zweier zufälliger Exponenten und die damit erfolgende Schlüsselvereinbarung durchgeführt. Als Generator wird fest die „5“ gewählt.

1.3.4.4 Elliptic Curves

Die Standardkonfiguration von OpenSSL beinhaltet alle Kurven aus RFC 4492, sowie weitere Kurven. Getestet werden die grundlegenden Operationen und Eigenschaften wie Addition, Multiplikation, Invertierung eines Erzeugers, sowie die Zyklizität einer Untergruppe. Es werden verschiedene Darstellungen von Punkten getestet. Für die unterstützen Kurven wird der Grad und die Ordnung überprüft.

1.3.4.5 EC Diffie Hellman

In den EC Diffie Hellman Tests wird die Vereinbarung eines Schlüssels mit den NIST-Kurven aus RFC 4492 getestet.

1.3.4.6 EC DSA

Die EC DSA Tests führen eine Signatur und Verifikation auf einer Untermenge der Kurven aus RFC 4492 durch.

1.3.5 Zufallszahlengenerator

Vom Zufallszahlengenerator wird ausschließlich die pseudozufällige Version getestet.

Testprogramm	Getestete Funktion
test/randtest	Pseudozufällige Datenerzeugung

Bemerkung: Da kein Test für den kryptografisch sicheren Zufallszahlengenerator RAND_pseudo_bytes existiert, wird nicht sichergestellt, dass dessen Fail-Safe-Funktionalität wie erwartet arbeitet. So könnte beispielsweise Schlüsselmaterial erzeugt werden, obwohl der Entropie-

pool erschöpft ist. Daher ist die Verifikation dieser Funktionalität von großer Bedeutung.

Empfehlung: Implementierung eines deterministischen Zufallszahlentests, bei dem die Menge der Entropie vorgegeben wird und daraus die damit zu erzeugende Menge an Zufallsdaten berechnet werden kann. Es muss überprüft werden, ob RAND_bytes einen Fehlerwert zurückgibt, sobald diese Menge überschritten wurde.

1.3.6 Zertifikatsverarbeitung

Grundsätzlich ist festzustellen, dass Tests der Zertifikatsverarbeitung keine vollständig Abdeckung erreichen können, da die Möglichkeiten für die Erstellung von Zertifikaten nahezu unbegrenzt sind.

Testprogramm	Getestete Funktion
test/tx509	Zertifikatskonvertierung
test/testgen	Zertifikatserstellung, -verifizierung
test/testss	Zertifikatserstellung, -verifizierung

In OpenSSL wird die Konvertierung von X.509v3 Zertifikaten mit 2 festen Zertifikaten getestet sowie die Erstellung und Verifikation von durch OpenSSL erstellten Zertifikaten.

Bemerkung: Es wird die Verifikation weniger Zertifikate getestet. Vor dem Hintergrund erfolgreicher Angriffe auf Verifikations-Implementierungen [GTOF, CVE-2014-0092] mittels speziell geformter böswilliger Zertifikate sollte hier eine höhere Testabdeckung angestrebt werden, auch wenn eine vollständige Abdeckung nicht möglich ist.

Empfehlung: Erstellen eines Testprogramms für die generelle Verifikation von Zertifikaten. Für die Tests muss eine Datenbank mit Testzertifikaten erstellt werden, die Rahmenparameter zum jeweiligen Zertifikat enthält (insbes. ob es erfolgreich verifiziert werden soll und welcher Zeitpunkt für die Verifikation angenommen werden soll). In diese Zertifikatsdatenbank werden zum einen bekannte böswillige Zertifikate und zum anderen eigene Zertifikate eingepflegt, mit denen versucht wird, Randfälle im Code anzugreifen. Diese Datenbank wäre darüber hinaus für andere TLS-Implementierungen hilfreich.

1.3.7 Integration externer Kryptografie-Module

Der Test der engine-Komponente testet das Hinzufügen und Entfernen von Modulen. Die mitgelieferten Module werden nicht getestet.

Testprogramm	Getestete Funktion
test/enginetest	engine-Interface

1.3.8 High-Level Kryptografie-Funktionen

OpenSSL testet verschiedene Hashfunktionen und Blockchiffren in verschiedenen Kombinationen. Die Auflistung erfolgt in den vorigen Abschnitten (Abschnitt 1.2).

1.3.9 TLS/SSL

OpenSSL liefert Tests für die TLS-1.2-Implementierung mit.

Testprogramm	Getestete Funktion
test/testssl	TLS 1.2

Bemerkung: Die Bewertung kann erst im weiteren Verlauf des Projekts erfolgen und wird nachgetragen.

1.4 Software-Interfaces

1.4.1 Interne Funktionen

OpenSSL nutzt Bibliotheken für die Verwaltung interner Strukturen. Diese sind:

lhash	objects	stack
txt_db	threads	buffer

Da diese für ein Verständnis der externen Schnittstellen nicht relevant sind, verweisen wir den interessierten Leser auf die zugehörige OpenSSL-Dokumentation [SSL-crypto].

1.4.2 Fehlerbehandlung: err

OpenSSL bietet für die Analyse von Fehlern die “err”-Bibliothek. Wenn von einer OpenSSL-Funktion ein Fehler zurückgegeben wird, ermöglicht diese Bibliothek diesen Fehler weitergehend zu untersuchen und eine Fehlerbeschreibung als Text zu erhalten.

Weitere Details sind in [SSL-err] zu finden.

1.4.3 Arithmetikbibliothek: bn

OpenSSL nutzt für die Verwendung von Ganzzahl-Arithmetik mit Zahlen beliebiger Länge die interne BN-Bibliothek. Die Bibliothek wird insbesondere für Public-Key-Algorithmen wie RSA und Diffie-Hellman genutzt. Die Zahlen werden in einer “BIGNUM”-Struktur gespeichert, für deren Verwaltung grundlegende Funktionen bereitgestellt werden. Für Berechnungen werden Zuweisungsoperationen, grundlegende arithmetische Operationen und für die Kryptografie Operationen in Restklassen, die Erzeugung von Zufalls- und Primzahlen sowie Primzahltests angeboten. Weitere

spezielle Operationen sind die Montgomery-Multiplikation und Blinding-Operationen.

Diese Operationen werden intern von OpenSSL verwendet. Daher verweisen wir für weitere Information auf die OpenSSL-Dokumentation [SSL-bn].

Bemerkung 1: Die Auswertung des Rückgabewertes der verschiedenen Operationen muss überprüft werden um eventuelle Fehler bei der Allokation von Speicher oder anderer Fehler abzufangen und daraus resultierende kryptografischer Schwächen zu verhindern.

Bemerkung 2: Wichtig für die Verarbeitung von kryptografischen Daten ist die korrekte Nutzung und Funktion der Freigabe- und Überschreib-Funktionen der Bibliothek, da ansonsten solche Daten im Arbeitsspeicher verfügbar und für andere Prozesse lesbar verbleiben könnten.

1.4.4 Elliptische Kurven: ec

OpenSSL nutzt für Berechnungen auf Elliptischen Kurven die interne “ec”-Bibliothek. Es werden Kurven über endliche Körper mit einer entsprechenden Primzahl sowie Kurven der Charakteristik 2 mit entsprechenden irreduziblen Polynomen unterstützt. Es wird sowohl die Erzeugung neuer EC-Gruppen unterstützt als auch die Nutzung der zu untersuchenden, für TLS 1.2 vorgegebenen, benannten Kurven.

Das externe Interface der Bibliothek arbeitet vornehmlich mit affinen Koordinaten, unterstützt aber den Import projektiver und komprimierter Darstellung von Punkten. Die Parameter der Kurve werden in EC_GROUP-Strukturen gespeichert, Punkte einer Kurve in einer EC_POINT-Struktur. Die Bibliothek hat optimierte Implementierungen für Operationen auf speziellen Kurven, die in einer EC_METHOD-Struktur gespeichert werden. Schlüssel werden in der EC_KEY-Struktur gespeichert.

Bemerkung 1: Wie die BN-Bibliothek bietet auch die ec-Bibliothek Funktionen zum Überschreiben von verwendeten Daten, was für kryptografisch relevante Daten notwendig ist.

Es werden die typischen Operationen für Punkte auf Kurven angeboten (Addition, Verdopplung, Multiplikation, Invertierung, Vergleich). Für die Beschleunigung der Punkt-Multiplikation können Vielfache des Erzeuger-Punktes vorberechnet werden.

Die Bibliothek bietet Funktionen zum Erzeugen von Punkten.Die Zuordnung eines Punktes zu einer Kurve erfolgt bei der Erzeugung des Punktes. Es existieren Tests, ob ein gegebener Punkt auf einer Kurve liegt.

Bemerkung 2: Tests auf Zugehörigkeit eines Punktes zu einer Kurve müssen bei Verwendung dieses Interfaces manuell durchgeführt werden. Dadurch besteht das Risiko, dass durch Implementierungsfehler in der aufrufenden Funktion durch Berechnungen mit nicht zu der Kurve gehörenden Punkten die Sicherheit des privaten Schlüssels nicht gewährleistet ist. Dies gilt für alle Operationen mit Elliptischen Kurven.

Empfehlung 1: Es sollte überprüft werden, ob ein impliziter Test auf Gruppenzugehörigkeit von Punkten zur zugrunde liegenden Kurve vor allen Rechenoperationen möglich ist und wie hoch die Kosten zur Laufzeit für einen solchen Test sind.

Für die Schlüsselerzeugung stellt die Bibliothek die Funktion "EC_KEY_generate_key()" bereit.

Die für Berechnungen auf der Kurve zu nutzenden Funktion kann über die EC_METHOD spezifiziert werden. Damit ist es möglich, optimierte Implementierungen für bestimmte Kurven zu nutzen. OpenSSL bietet auf 64-Bit Plattformen optimierte Implementierungen für die TLS-Kurven secp224r1, secp256r1, secp521r1. Um die Grundfunktionalität bereitstellen zu können bietet die EC-Bibliothek eine für alle Kurven verwendbare EC_METHOD („EC_GFp_simple_method()“).

Diese Darstellung der EC-Bibliothek ist nicht vollständig. Da diese Bibliothek nicht Kern der Untersuchungen ist, verweisen wir für weitere Details auf die OpenSSL-Dokumentation [SSL-ec].

Allgemeine Empfehlung: Bei der Nutzung der EC-Bibliothek gibt es viele Freiheitsgerade und damit Möglichkeiten für Fehler in einer Implementierung. Daher sollte Code, der diese Bibliothek nutzt, sorgfältig überprüft werden.

1.4.5 Zufallszahlengenerator: rand

OpenSSL bietet einen Entropiepool und Verwaltungsfunktionen für diesen.

Die nur auf Microsoft Windows verfügbaren Funktionen (RAND_screen(), RAND_event()) werden ebenso wenig betrachtet wie die Interaktion mit dem „Entropy Gathering Daemon“ (RAND_egd()), der nur auf Plattformen ohne guten System-RNG sinnvoll ist.

Die OpenSSL-Verwaltungsfunktionen ermöglichen das Hinzufügen von Entropie zum Entropiepool, die Erzeugung von Zufallsdaten, und das Speichern von Entropie über verschiedene Läufe hinweg (z.B. bei einem Webserver, der vom Entropiepool abgeleitete Informationen beim Herunterfahren des Systems speichert um beim Neustart bereits ausreichend Entropie zur Verfügung zu haben, obwohl das Betriebssystem noch keine ausreichende Entropie liefern konnte). Die Funktionen zur Verwaltung des Entropiepools können ausgetauscht werden.

Referenz	https://www.openssl.org/docs/crypto/rand.html, https://www.openssl.org/docs/crypto/RAND_set_rand_method.html, https://www.openssl.org/docs/crypto/RAND_load_file.html, https://www.openssl.org/docs/crypto/RAND_add.html, https://www.openssl.org/docs/crypto/RAND_bytes.html, https://www.openssl.org/docs/crypto/RAND_cleanup.html

Spezifizieren einer RNG-Engine	int RAND_set_rand_engine()
Setzen einer alternative RAND-Funktion	void RAND_set_rand_method()
Lesen der aktuellen RAND-Funktion	RAND_METHOD *RAND_get_rand_method()
Standard-RAND-Funktion	RAND_METHOD *RAND_SSLeay()
Speichern von Entropie in Datei	int RAND_write_file()
Laden von Entropie aus Datei	int RAND_load_file()
Seeden des Entropiepools	void RAND_seed()
Hinzufügen von Entropie mit Entropieschätzung	void RAND_add()
Feststellen, ob genug Entropie im Pool	int RAND_status()
Blockierendes Lesen von Zufallsdaten	int RAND_bytes()
Nicht-blockierendes Lesen von Zufallsdaten	int RAND_pseudo_bytes()
Deinitialisierung des Entropiepools	void RAND_cleanup()

Bemerkung 1: Die RAND_set_rand_method() sollte nicht mehr genutzt werden und durch eine Engine-Implementierung ersetzt werden, da eine Engine mehr Möglichkeiten für die Entropie-Erzeugung bietet.

Bemerkung 2: Mit den RAND_set_rand_engine()- und RAND_set_rand_method()-Funktionen kann eine schwache Funktion für die Erzeugung von Zufallsdaten spezifiziert werden. Dies kann auch bei ansonsten korrekter Nutzung der API zu Schwächen im praktischen Einsatz führen.

Empfehlung 1: Sicherheitskritische Software muss dahingehend untersucht werden, welche RAND-Funktion genutzt wird. Die Nutzung der Software ist erst anzuraten, wenn die Qualität aller genutzten RAND-Funktionen geprüft wurde.

Bemerkung 3: Die RAND_pseudo_bytes()-Funktion liefert auch Daten, wenn der Entropiepool erschöpft ist. Diese Daten sind nicht mehr kryptografisch sicher.

Empfehlung 2: Bei Erzeugung von kryptografisch sensitivem Material muss die Funktion RAND_bytes() genutzt werden. Diese kann blockieren. Entwürfe sowie Implementierungen von Krypto- und Software-Systeme müssen auf diesen Umstand Rücksicht nehmen.

1.4.6 Zertifikatsverarbeitung: x509

Dieser Abschnitt behandelt vornehmlich die Verarbeitung von Zertifikaten bezüglich der Verifikation.

Über die hier dokumentierten Funktionen hinaus gibt es verschiedene Module, die Certificate Revocation Lists, Certificate Requests, Im- und Exportfunktionen für Zertifikate und darin enthaltene Strukturen anbieten. Diese arbeiten mit Ein- und Ausgabeoperationen, die von der BIO-Bibliothek bereitgestellt werden. Weiterhin existieren Funktionen zum Verändern von X.509-Strukturen.

Eine vollständige Beschreibung dieser Funktionen würde den Rahmen dieses Arbeitspaketes sprengen und wurde daher nicht durchgeführt. Für mehr Details verweisen wir auf die OpenSSL-Dokumentation [SSSl-x509, SSL-pem].

1.4.6.1 Verwalten einer Zertifikatsdatenbank

Eine Zertifikatsverifikation in OpenSSL erfolgt grundsätzlich gegen eine Zertifikatsdatenbank.

Referenz	https://www.openssl.org/docs/crypto/X509_STORE_CTX_new.html
Datenstruktur für Certificate Store	X509_STORE_CTX
Erzeugung	X509_STORE_CTX *X509_STORE_CTX_new()
Initialisierung	int X509_STORE_CTX_init()
Leeren der Struktur	void X509_STORE_CTX_cleanup()
Löschen der Struktur	void X509_STORE_CTX_free()
Hinzufügen einer CRL	void X509_STORE_CTX_set0_crls()
Setzen der vertrauenswürdigen Zertifikate	void X509_STORE_CTX_trusted_stack
Hinzufügen einer Zertifikatskette	void X509_STORE_CTX_set_chain

Bemerkung: Es fällt auf, dass bis auf die “_init()”- und “_new()”-Funktion keine Funktion einen Rückgabewert besitzt. Dies überrascht insbesondere beim Hinzufügen von vertrauenswürdigen Zertifikaten und CRLs, da ein Fehler in einer solchen Struktur dem Aufrufer mitgeteilt werden sollte. Das genaue Verhalten der Implementierung sollte im weiteren Verlauf des Projekts analysiert werden.

1.4.6.2 Verifikation eines Zertifikats

Die Verifikation eines Zertifikats setzt eine vertrauenswürdige Zertifikatsdatenbank voraus. Gegen diese kann dann ein Zertifikat geprüft werden. Zusätzlich lassen sich weitere Eigenschaften des Zertifikats spezifizieren, die überprüft werden sollen.

Referenz	https://www.openssl.org/docs/crypto/X509_verify_cert.html, https://www.openssl.org/docs/crypto/X509_VERIFY_PARAM_set_flags.html,

	https://www.openssl.org/docs/crypto/X509_STORE_CTX_get_error.html
Datentyp für Verifikationsparameter	X509_VERIFY_PARAM
Datentyp eines Zertifikats	X509
Setzen des zu verifizierende Zertifikats	void X509_STORE_CTX_set_cert()
Verifikation eines Zertifikats	int X509_verify_cert()
Setzen von Verifikationsflags	int X509_VERIFY_PARAM_set_flags()
Verifikationsflags löschen	int X509_VERIFY_PARAM_clear_flags()
Setzen von Verifikationsattributen	int X509_VERIFY_PARAM_set_[purpose, trust, time, policy, policies, depth, host, hostflags, email, ip, ip_asc]()
Lesen von Verifikationsattributen	int X509_VERIFY_PARAM_get_flags()
Lesen der Verifikationstiefe	int X509_VERIFY_PARAM_get_depth()
Information zum Verifikationsfehler	int X509_STORE_CTX_get_error()
Ebene des Fehlers	int X509_STORE_CTX_get_error_depth()
Zertifikat, das den Fehler verursacht	X509 *X509_STORE_CTX_get_current_cert()
Verifikationskette bei Erfolg	STACK_OF(X509) *X509_STORE_CTX_get1_chain()

Bei erfolgreicher Verifikation des Zertifikats wird von dieser Funktion der Wert “1” zurückgegeben. Ein Rückgabewert von “0” zeigt einen Fehler an. Weitere Informationen zum Fehler können mit der Funktion “X509_STORE_CTX_get_error()” abgefragt werden.

Bemerkung: Die Funktionen zur Verifikation von Zertifikaten sind dokumentiert. Allerdings ist die Dokumentation nicht zusammenhängend. Eine zusammenhängende Dokumentation aller Funktionen, die mit der Verifikation von Zertifikaten zusammenhängen, würde die Einstiegshürde senken und unnötige Fehler bei der Implementierung verhindern.

1.4.7 TLS Bibliothek: SSL

Die High-Level-Bibliothek für TLS-Verbindungen wird von der SSL-Bibliothek bereitgestellt. Die SSL-Struktur ist die Hauptstruktur für die Verwaltung einer TLS-Verbindung. SSL_CIPHER beschreibt eine komplette Ciphersuite, wie z.B. „ECDH-RSA-AES256-GCM-SHA384“

Referenz	https://www.openssl.org/docs/ssl/ssl.html
Dispatch-Struktur	SSL_METHOD
TLS-Ciphersuite-Struktur	SSL_CIPHER
TLS Kontext-Struktur	SSL_CTX

TLS Session-Struktur	SSL_SESSION
TLS Verbindungs-Struktur	SSL
TLS-Methode für Client	TLSv1_client_method()
TLS-Methode für Server	TLSv1_server_method()
TLS-Methode für Client/Server	TLSv1_method()
Erstellen eines Kontextes	SSL_CTX_new()
Setzen der Ciphersuite Liste	SSL_CTX_set_cipher_list()
Erstellen einer Verbindungsstruktur	SSL_new()
Zuweisen eines BIO	SSL_set_bio()
Handshake durchführen	SSL_connect()
Auslesen der ausgehandelten Ciphersuite	SSL_get_cipher()
Schließen der TLS-Verbindung	SSL_shutdown()

Bemerkung: Diese Bibliothek stellt einen Kernteil der Untersuchungen dar. Hier soll zunächst ein allgemeiner Überblick gegeben werden mit den unbedingt notwendigen Elementen für einen Verbindungsaufbau, ohne die Verbindung sicher zu konfigurieren. Im Laufe des Projekts wird dieser Teil erweitert werden.

1.4.8 Input/Output Bibliothek: BIO

Die die BIO-Bibliothek von OpenSSL stellt Funktionen zur Abstraktion von Ein- und Ausgabeoperationen bereit. Die BIO-Bibliothek stellt hierbei nicht nur Abstraktionen von Betriebssystemschnittstellen bereit, sondern auch die Einbindung von kryptografischen Operationen in eine Verarbeitungskette.

Die Auflistung der Funktionen ist unvollständig und dient dazu, einen grundsätzlichen Überblick über die Ein- und Ausgabearchitektur zu schaffen. Die vollständige Dokumentation ist auf der OpenSSL-Seite verfügbar [SSL-bio].

1.4.8.1 Generelle Verwaltung von BIO Strukturen

Die grundsätzliche Initialisierung und das Löschen von BIO-Strukturen erfolgt mit den folgenden Funktionen.

Referenz	https://www.openssl.org/docs/crypto/BIO_new.html,

	https://www.openssl.org/docs/crypto/BIO_push.html, https://www.openssl.org/docs/crypto/BIO_find_type.html
Erzeugen einer BIO-Struktur	BIO *BIO_New()
Freigeben der Struktur	int BIO_free(), void BIO_vfree()
Freigeben einer BIO-Kette	void BIO_free_all()
Zuweisen einer Funktion	int BIO_set()
Zurücksetzen, abhängig vom BIO-Typ	BIO_reset()
Hinzufügen eines Filters zur Verarbeitungskette	BIO_push()
Entfernen eines Filters aus der Verarbeitungskette	BIO_pop()
Navigation in einer BIO-Kette, nach Typ	BIO_find_type()
Navigation in einer BIO-Kette	BIO_next()

Ketten werden mit Hilfe der BIO_push() Funktion aufgebaut. Die so definierte Richtung gilt für das Schreiben von Daten in eine Filterkette. Für lesenden Zugriff wird die Abfolge der Operationen umgekehrt.

Die Bedeutung des Zurücksetzens (“BIO_reset()”) einer BIO-Struktur ist abhängig vom dieser und in der entsprechenden Dokumentation beschrieben.

1.4.8.2 Aufsetzen, Lesen und Schreiben von BIO Sink/Sources

Die BIO-Bibliothek bietet Lese- und Schreibschnittstellen für BIO-Paare, TCP/IP Sockets, Dateien und Dateistreams, Speicher (z.B. für Shared-Memory-Kommunikation), und eine leere (“null”) BIO Sink/Source.

Referenz	https://www.openssl.org/docs/crypto/BIO_s_bio.html, https://www.openssl.org/docs/crypto/BIO_s_accept.html, https://www.openssl.org/docs/crypto/BIO_s_connect.html, https://www.openssl.org/docs/crypto/BIO_s_socket.html, https://www.openssl.org/docs/crypto/BIO_s_fd.html, https://www.openssl.org/docs/crypto/BIO_s_file.html,

Erstellen eines BIO-Paar Methode	BIO_METHOD *BIO_s_bio()
Erstellen eines BIO-Paares	int BIO_new_bio_pair()
Socket aufsetzen	BIO *BIO_new_accept()
Warten auf eingehende Verbindungen	int BIO_do_accept()
Methode für BIO-Verbindungsaufbau	BIO *BIO_new_connect()
Ziel-Host festlegen	long BIO_set_conn_hostname()
Ziel-Port festlegen	long BIO_set_conn_port()
Verbindung aufbauen	int BIO_do_connect()
Erstellen eines BIO-Datei-Methode	BIO *BIO_new_file()
Erstellen einer BIO-Strom-Methode	BIO *BIO_new_fp()
Lesen aus einer BIO Kette	int BIO_read(), int BIO_gets()
Schreiben in eine BIO Kette	int BIO_write(), int BIO_puts()

Das Lesen und Schreiben von Daten einer BIO-Kette wird jedoch nicht vollständig abstrahiert. Die Funktionen arbeiten nicht-blockierend, und daher wird in der OpenSSL-Dokumentation sogar empfohlen, Betriebssystemmittel zu nutzen, um festzustellen, ob die jeweilige Quelle/Ziel bereit ist und dann erst die Lese- und Schreibe-Operationen der BIO-Bibliothek zu nutzen.

1.4.8.3 Behandeln von Lese- und Schreibfehlern

Falls Fehler beim Lesen oder Schreiben aus einer BIO-Struktur auftreten, helfen die folgenden Makros bei deren Analyse.

Referenz	https://www.openssl.org/docs/crypto/BIO_should_retry.html
Lesezugriff erwartet	BIO_should_read()
Schreibzugriff erwartet	BIO_should_write()
Spezial-IO erwartet	BIO_should_io_special()
Temporärer Fehler, später nochmal probieren	BIO_should_retry()
Fehlertyp bei temporärem Fehler	BIO_retry_type()

1.4.8.4 Kryptografische BIO-Filter

Symmetrische Chiffren werden über die folgenden Funktionen in eine Verarbeitungskette eingebunden.

Referenz	https://www.openssl.org/docs/crypto/BIO_f_cipher.html
Erstellen eines BIO für symmetrische Chiffren	BIO_METHOD *BIO_f_cipher()
Setzen der Chiffre	void BIO_set_cipher()
Erfolg einer Entschlüsselung feststellen	int BIO_get_cipher_status()
Referenz auf Chiffre-Kontext	int BIO_get_cipher_ctx()

Hashfunktionen werden über die folgenden Funktionen eingebunden.

Referenz	https://www.openssl.org/docs/crypto/BIO_f_md.html
Erstellen eines BIO für Hashfunktion	BIO_METHOD *BIO_f_md()
Setzen der Hashfunktion	int BIO_set_md()
Lesen des Hash-Wertes	int BIO_get_md()
Referenz auf Hash-Kontext	int BIO_get_md_ctx()

Bemerkung: Es ist auffällig, dass der Rückgabewert beim Setzen der Hashfunktion anders ist als beim Setzen der Chiffre.

1.4.8.5 SSL-Filter

Für die Verbindung mit der TLS-Infrastruktur bietet die BIO-Bibliothek die folgenden Funktionen an.

Referenz	https://www.openssl.org/docs/crypto/BIO_f_ssl.html
Erstellen einer TLS-BIO	BIO_METHOD *BIO_f_ssl()
Erstellen einer BIO-Kette mit TLS-Konfiguration	BIO *BIO_new_ssl()
Erstellen einer BIO- Kette mit TLS-Konfiguration und Connect-BIO	BIO *BIO_new_ssl_connect()
Erstellen einer BIO- Kette mit TLS-Konfiguration, Connect- und Buffer-BIO	BIO *BIO_new_buffer_ssl_connect()
Schließen einer TLS-BIO-Kette	BIO_ssl_shutdown()

Die TLS-BIO führt bei Bedarf automatisch einen TLS-Handshake durch. Es kann die Menge von Daten spezifiziert werden, nach denen eine Neuaushandlung der Verbindung.

1.4.8.6 Zusammenfassung

Die BIO-Bibliothek stellt ein Framework zum Erstellen von Verarbeitungsketten bereit. Ein Teil einer solchen Verarbeitungskette ist bei TLS-Verbindungen ein SSL-Filter. Auf den ersten Blick wirkt die Schnittstelle nicht einheitlich: So ist beispielsweise die BIO_reset()-Funktion für jeden Filtertypen einzeln spezifiziert und hat teilweise eine Bedeutung, die der intuitiven widerspricht. Darüber hinaus ist gerade die SSL-Filter-BIO besonders. Leider ist eine einheitliche, konsistente Darstellung wie in Abschnitt 1.4.10.2 nicht möglich.

Als Resultat ist die BIO-Bibliothek aufgrund ihrer Komplexität schwer zu benutzen. Hier wäre eine Vereinfachung der Schnittstelle, vielleicht mit Einschränkung der Funktionalität auf ein sinnvolles Subset, oder ein zusätzliches Interface mit diesen Eigenschaften, sinnvoll.

1.4.9 Integration externer Kryptografiemodule: Engine

Für die Integration externer Kryptografiemodule wird in OpenSSL das Engine-Interface genutzt. Es werden unter anderem die folgenden Module unterstützt.

Referenz	https://www.openssl.org/docs/crypto/engine.html
Public Key Implementierungen	RSA_METHOD, DSA_METHOD, DH_METHOD
EC Implementierungen	ECDH_METHOD, ECDSA_METHOD
Symmetrische Chiffren	EVP_CIPHER
Hashfunktionen	EVP_DIGEST
RNG Implementierung	RAND_METHOD

Ein Engine-Modul kann alle symmetrischen Chiffren, Hashfunktionen, asymmetrische Algorithmen und den RNG ersetzen. Es ist möglich, eine Engine als “default”-Engine zu spezifizieren, die dann automatisch von allen Funktionen genutzt wird.

Bemerkung: Das Engine-Interface der OpenSSL-Bibliothek stellt ein mächtiges Werkzeug zur Abstraktion der Krypto-Funktionen bereit. Dies kann missbraucht werden, ermöglicht aber auch die Ersetzung aller Kryptofunktionen mit vom Nutzer ausgewählten Implementierungen, die beispielsweise für die Verarbeitung sensitiver Daten geprüft wurden.

1.4.10 High-Level Kryptografiefunktionen: EVP

OpenSSL bietet mit der EVP-Bibliothek einen Abstraktionslayer oberhalb der Implementierung von kryptografischen Primitiven. Die Chiffren werden in verschiedene Klassen aufgeteilt, für die ein

einheitliches Interface bereitgestellt wird [SSL-evp].

Klasse	Interface
Public Key Ver/Entschlüsselung von Daten, “Digital Envelope”	EVP_Seal, EVP_Open
Signaturfunktionen mit Message Authentication Codes	EVP_DigestSign, EVP_DigestVerify, EVP_Sign, EVP_Verify
Symmetrische Verschlüsselung	EVP_Encrypt, EVP_Decrypt, EVP_Cipher
Hashfunktionen	EVP_Digest
Public Key Algorithmen (nativ)	EVP_PKEY
Ableitung von Passwörtern	EVP_BytesToKey
Initialisierung der Bibliothek mit Chiffren	OpenSSL_add_all_algorithms

Die Darstellung soll einen Überblick über die Bibliothek geben und eine grundsätzliches Verständnis schaffen. Daher werden nur die wichtigen Funktionen und deren Eigenschaften dargestellt sowie besondere Eigenschaften, wenn diese dem intuitiven Verständnis widersprechen oder für die Arbeit mit der Bibliothek wichtig sind (beispielsweise überraschende Rückgabewerte). Für weitergehende Details wird in den jeweiligen Abschnitten auf die OpenSSL-Dokumentation verwiesen.

Datentypen werden wie im OpenSSL-Quelltext dargestellt. Bei Funktionen wurden die Parameter entfernt und lediglich eine öffnende und schließende Klammer als Erkennungsmerkmal beibehalten.

1.4.10.1 Initialisierung der Bibliothek mit Chiffren

Für die Initialisierung der Kryptografiefunktionen stellt die OpenSSL-Bibliothek Funktionen bereit. Diese haben keinen Rückgabewert und machen jeweils alle zur Klasse gehörenden Funktionen für die Nutzung verfügbar.

Referenz	https://www.openssl.org/docs/crypto/OpenSSL_add_all_algorithms.html
Hinzufügen aller Chiffren	OpenSSL_add_all_ciphers()
Hinzufügen aller Hashfunktionen	OpenSSL_add_all_digests()
Beide obigen Klassen	OpenSSL_add_all_algorithms()

1.4.10.2 Symmetrische Verschlüsselung

Die EVP-Bibliothek stellt für die Verarbeitung mit Blockchiffren 3 Klassen von Funktionen bereit. Dies sind Funktionen zum Verschlüsseln (EVP_Encrypt...), zum Entschlüsseln (EVP_Decrypt...) sowie zum Ver- und Entschlüsseln (EVP_Cipher...). Sofern nicht anders vermerkt, geben die Funk-

tionen einen “int” zurück. Die _ex-Varianten der Funktionen ermöglichen die Spezifikation einer speziellen Engine, mit der die Operationen durchgeführt werden. Die Auswahl der Chiffre erfolgt über einen Parameter in den “...Init” und “...Init_ex”-Funktionen.

Für “int”-Funktionen signalisiert ein Rückgabewert von “1” erfolgreiche Ausführung, “0” einen Fehler.

Referenz	https://www.openssl.org/docs/crypto/EVP_EncryptInit.html
Funktionspointer-Typ	EVP_CIPHER
Kontext-Typ	EVP_CIPHER_CTX
Initialisierung Kontext	void EVP_CIPHER_CTX_init()
Deinitialisierung Kontext	EVP_CIPHER_CTX_cleanup()
Initialisierung Chiffre	EVP_EncryptInit_ex(), EVP_EncryptInit(), EVP_DecryptInit_ex(), EVP_DecryptInit(), EVP_CipherInit_ex(), EVP_CipherInit()
Verarbeitung von Datenblöcken	EVP_EncryptUpdate(), EVP_DecryptUpdate(), EVP_CipherUpdate()
Verarbeitung des letzten Datenblocks	EVP_EncryptFinal_ex(), EVP_EncryptFinal(), EVP_DecryptFinal_ex(), EVP_DecryptFinal(), EVP_CipherFinal_ex(), EVP_CipherFinal()

Mit dem EVP_CIPHER-Typen werden die NULL-Chiffre, Stromchiffren und Blockchiffren mit ihren Modes of Operation abgebildet. Das Verhalten im GCM- und CCM-Modus weicht leicht vom Verhalten der anderen Modi ab. Dies ist in der Dokumentation beschrieben.

Bemerkung: Das unterschiedliche Verhalten der GCM- und CCM-Modi kann leicht zu Fehlern führen, wenn die unterschiedliche Behandlung von Daten nicht berücksichtigt wird

1.4.10.3 Hashfunktionen

Die EVP-Bibliothek stellt für die Verarbeitung mit Hashfunktionen Funktionstypen (EVP_MD_Digest...) bereit. Sofern nicht anders vermerkt, geben die Funktionen einen “int” zurück. Die _ex-Varianten der Funktionen ermöglichen die Spezifikation einer speziellen Engine, mit der die Operationen durchgeführt werden. Die Auswahl der Hashfunktion erfolgt über einen Parameter in den “...Init” und “...Init_ex”-Funktionen.

Für “int”-Funktionen signalisiert ein Rückgabewert von “1” erfolgreiche Ausführung, “0” einen Fehler.

Referenz	https://www.openssl.org/docs/crypto/EVP_Dige

	stInit.html
Funktionspointer-Typ	EVP_MD
Kontext-Typ	EVP_MD_CTX
Initialisierung Kontext	void EVP_MD_CTX_init(), EVP_MD_CTX *EVP_MD_CTX_create
Kopie Kontext	EVP_MD_CTX_copy_ex(), EVP_MD_CTX_copy()
Deinitialisierung Kontext	EVP_MD_CTX_cleanup(), void EVP_MD_CTX_destroy()
Initialisierung Hashfunktion	EVP_DigestInit_ex(), EVP_DigestInit()
Verarbeitung von Datenblöcken	EVP_DigestUpdate()
Verarbeitung des letzten Datenblocks	EVP_DigestFinal_ex(), EVP_DigestFinal()

1.4.10.4 Hashed MAC

Die Hashed-MAC-Konstruktion ist nicht expliziter Bestandteil der EVP-Bibliothek, da sie für sich schon abstrahiert ist und keine weitere Abstraktion in der EVP-Bibliothek benötigt. Da sie vom Abstraktionslevel vergleichbar mit der EVP-Bibliothek ist und stark mit den Hashfunktionen verzahnt ist, wird sie hier behandelt.

Die HMAC-Bibliothek nutzt die von den Hashfunktionen bekannte EVP_MD-Struktur sowie eine HMAC_CTX Struktur für die Verwaltung einer Instanz.

Sofern nicht anders vermerkt, geben die Funktionen einen “int” zurück. Die _ex-Varianten der Funktionen ermöglichen die Spezifikation einer speziellen Engine, mit der die Operationen durchgeführt werden. Die Auswahl der Hashfunktion sowie die Festlegung des Schlüssels erfolgt über einen Parameter in den “...Init” und “...Init_ex”-Funktionen.

Für “int”-Funktionen signalisiert ein Rückgabewert von “1” erfolgreiche Ausführung, “0” einen Fehler.

Referenz	https://www.openssl.org/docs/crypto/hmac.html
Kontext-Typ	HMAC_CTX
Initialisierung Kontext	void HMAC_CTX_init()
Initialisierung der HMAC-Instanz	HMAC_init(), HMAC_init_ex()
Verarbeitung von Datenblöcken	HMAC_Update()
Verarbeitung des letzten Datenblocks	HMAC_Final()

Deinitialisierung Kontext	HMAC_CTX_cleanup()
Deinitialisierung der HMAC-Instanz	HMAC_cleanup()

1.4.10.5 Public Key Schlüsselverwaltung

Die EVP-Bibliothek stellt für die Verarbeitung von Public Key Schlüsseln generische Funktionen bereit. Um die nativen PK-Algorithmen (z.B. RSA und EC) mit den generischen Funktionen nutzen zu können, müssen die Schlüssel in den Datentypen EVP_PKEY konvertiert werden. Die dafür benötigten Funktionen werden hier dargestellt.

Für "int"-Funktionen signalisiert ein Rückgabewert von "1" erfolgreiche Ausführung, "0" einen Fehler. Die "Pointer"-Funktionen geben im Fehlerfall "NULL" zurück, im Erfolgsfall einen Pointer auf die entsprechende Schlüsselstruktur.

Referenz	https://www.openssl.org/docs/crypto/EVP_PKEY_set1_RSA.html
Datentyp der Schlüsselstruktur	EVP_PKEY
Erzeugung Schlüsselstruktur	EVP_PKEY *EVP_PKEY_new()
Löschen Schlüsselstruktur	void EVP_PKEY_free()
Importieren von Schlüsseln, Konvertierung	int EVP_PKEY_set1_RSA(), int EVP_PKEY_set1_DSA(), int EVP_PKEY_set1_DH(), int EVP_PKEY_set1_EC_KEY()
Re-Konvertierung, Export	RSA *EVP_PKEY_get1_RSA(), DSA *EVP_PKEY_get1_DSA(), DH *EVP_PKEY_get1_DH(), EC_KEY *EVP_PKEY_get1_EC_KEY()
Referenz auf Schlüsselstruktur, Konvertierung	int EVP_PKEY_assign_RSA(), int EVP_PKEY_assign_DSA(), int EVP_PKEY_assign_DH(), int EVP_PKEY_assign_EC_KEY()

Der Unterschied zwischen dem Initialisieren einer Schlüsselstruktur und dem Setzen einer Referenz auf eine Schlüsselstruktur ist, dass beim Initialisieren eine Kopie des Schlüssels erzeugt wird. Bei der Erzeugung einer Referenz wird keine Kopie erzeugt. D.h. insbesondere, dass nach dem Löschen einer Referenz auch alle andere Referenzen auf den gleichen Schlüssel nicht mehr gültig sind.

1.4.10.6 Public Key Schlüsselerzeugung

Die EVP-Bibliothek stellt für die Erzeugung von Public Key Schlüsseln generische Funktionen bereit. Der Typ des erzeugten Schlüssels wird über die EVP_PKEY_CTX-Struktur festgelegt. Diese

wird mit der EVP_PKEY_CTX_new...-Funktionen initialisiert, der ein Parameter für den Schlüsseltyp übergeben wird sowie die Angabe der zu implementierenden Engine.

Für "int"-Funktionen signalisiert ein Rückgabewert von "1" erfolgreiche Ausführung, "0" einen Fehler. Die "Pointer"-Funktionen geben im Fehlerfall "NULL" zurück, im Erfolgsfall einen Pointer auf den entsprechenden Schlüsselkontext.

Referenz	https://www.openssl.org/docs/crypto/EVP_PKEY_keygen.html, https://www.openssl.org/docs/crypto/EVP_PKEY_CTX_new.html
Schlüsseltyp	EVP_PKEY
Kontext-Typ der Schlüsselerzeugung	EVP_PKEY_CTX
Erzeugen eines Kontextes	EVP_PKEY_CTX *EVP_PKEY_CTX_new(), EVP_PKEY_CTX *EVP_PKEY_CTX_new_id()
Löschen eines Kontextes	void EVP_PKEY_CTX_free()
Initialisierung Kontext	int EVP_PKEY_keygen_init(), int EVP_PKEY_paramgen_init()
Erzeugung Parameter	int EVP_PKEY_keygen() int EVP_PKEY_paramgen()

1.4.10.7 Public Key Schlüsselherleitung

Die EVP-Bibliothek stellt für die Herleitung von gemeinsamem symmetrischen Schlüsselmaterial zwischen zwei Kommunikationspartnern generische Funktionen bereit. Die EVP_PKEY_derive()-Funktion erzeugt das gleiche symmetrische Schlüsselmaterial bei beiden Kommunikationspartnern. Für die hier beschriebenen Funktionen ist der Rückgabewert vom Typ "int".

Für "int"-Funktionen signalisiert ein Rückgabewert von "1" erfolgreiche Ausführung, "<0" einen Fehler. Insbesondere stellt der Fehlerwert "-2" dar, dass die Operation vom gewählten Public Key Algorithmus nicht unterstützt wird.

Referenz	https://www.openssl.org/docs/crypto/EVP_PKEY_derive.html
Schlüsseltyp	wie Public Key Schlüsselerzeugung (1.4.10.6)
Verwaltung des Schlüssel-Typ	wie Public Key Schlüsselerzeugung (1.4.10.6)
Kontext-Typ	wie Public Key Schlüsselerzeugung (1.4.10.6)
Verwaltung des Kontext-Typ	wie Public Key Schlüsselerzeugung (1.4.10.6)
Initialisierung der Schlüsselherleitung	EVP_PKEY_derive_init()

Setzen des Schlüsselherleitungspartners	EVP_PKEY_derive_set_peer()
Herleitung von Schlüsselmaterial	EVP_PKEY_derive()

1.4.10.8 Public Key Algorithmen (nativ)

Die EVP-Bibliothek stellt für die Ver- und Entschlüsselung sowie Signaturen und deren Verifikation entpsrechende Funktionen bereit. Der jeweilige Algorithmus und Schlüssel wird über die Parameter spezifiziert, mit denen der Kontext initialisiert wurde. Die Signaturverifikation mit Message Recovery ist nur für bestimmte Public Key-Chiffren möglich. Die unterstützten Algorithmen sind in der zugehörigen OpenSSL-Dokumentation spezifiziert.

Für "int"-Funktionen signalisiert ein Rückgabewert von "1" erfolgreiche Ausführung, "<=0" einen Fehler. Insbesondere stellt der Fehlerwert "-2" dar, dass die gewählte Engine das gewünschte Verfahren nicht unterstützt.

Bemerkung: Die alten EVP_SignInit()-, EVP_VerifyInit()- und zugehörige Funktionen werden hier nicht beschrieben.

Referenz	https://www.openssl.org/docs/crypto/EVP_PKEY_sign.html, https://www.openssl.org/docs/crypto/EVP_PKEY_verify.html, https://www.openssl.org/docs/crypto/EVP_PKEY_verify_recover.html, https://www.openssl.org/docs/crypto/EVP_PKEY_encrypt.html, https://www.openssl.org/docs/crypto/EVP_PKEY_decrypt.html
Schlüsseltyp	wie Public Key Schlüsselerzeugung (1.4.10.6)
Verwaltung des Schlüssel-Typ	wie Public Key Schlüsselerzeugung (1.4.10.6)
Kontext-Typ	wie Public Key Schlüsselerzeugung (1.4.10.6)
Verwaltung des Kontext-Typ	wie Public Key Schlüsselerzeugung (1.4.10.6)
Signaturerstellung	EVP_PKEY_sign_init(), EVP_PKEY_sign()
Signaturverifikation	EVP_PKEY_verify_init(), EVP_PKEY_verify()
Signaturverifikation mit Message Recovery	EVP_PKEY_verify_recover()
Public Key Verschlüsselung	EVP_PKEY_encrypt

Public Key Entschlüsselung	EVP_PKEY_decrypt

Bemerkung 1: Statt EVP_PKEY_encrypt/decrypt sollten bis auf spezielle Ausnahmen die EVP_Seal und EVP_Open Funktionen genutzt werden.

Bemerkung 2: Statt EVP_PKEY_sign/verify sollten bis auf spezielle Ausnahmen die EVP_Digest-Sign und EVP_DigestVerify Funktionen genutzt werden.

1.4.10.9 Digital Envelope

Die EVP-Bibliothek stellt für das “Ver- und Entpacken” von kryptografisch sensitiven Daten Seal- und Open-Funktionen bereit (Verschlüsselung von Daten mit einem symmetrischen Schlüssel, der abschließend mit einer asymmetrischen Funktion verschlüsselt dem Datenpaket hinzugefügt wird). Sofern nicht anders vermerkt, geben die Funktionen einen “int” zurück. Die Auswahl der symmetrischen und asymmetrischen Verschlüsselungsfunktionen und Schlüssel erfolgt über Parameter in den “...Init” Funktionen.

Für “int”-Funktionen signalisiert ein Rückgabewert von “1” erfolgreiche Ausführung, “0” einen Fehler. Ausnahme sind hier die EVP_SealFinal()-Funktionen, die im erfolgreichen Fall einen Rückgabewert ”>=1” haben

Referenz	https://www.openssl.org/docs/crypto/EVP_DigestInit.html
Funktionspointer-Typ	wie Symmetrische und Asymmetrische Verschlüsselung (1.4.10.2, 1.4.10.8)
Kontext-Typ	wie Symmetrische Verschlüsselung (1.4.10.2)
Initialisierung Kontext	wie Symmetrische Verschlüsselung (1.4.10.2)
Deinitialisierung Kontext	wie Symmetrische Verschlüsselung (1.4.10.2)
Initialisierung Hashfunktion	EVP_SealInit(), EVP_OpenInit()
Verarbeitung von Datenblöcken	EVP_SealUpdate(), EVP_OpenUpdate()
Verarbeitung des letzten Datenblocks	EVP_SealFinal(), EVP_OpenFinal()

1.4.10.10 Signaturfunktionen mit Message Authentication Codes

Die EVP-Bibliothek stellt für effiziente Signaturen eine Verbindung aus Hashfunktionen und Signaturfunktionen bereit. Sofern nicht anders vermerkt, geben die Funktionen einen “int” zurück. Die Auswahl der Hashfunktion und der Public-Key-Funktion erfolgt über Parameter in den “...Init”-Funktionen.

Für “int”-Funktionen signalisiert ein Rückgabewert von “1” erfolgreiche Ausführung, “<=0”

einen Fehler. Insbesondere stellt der Fehlerwert “-2” dar, dass die gewählte Engine das gewünschte Verfahren nicht unterstützt. EVP_Verify_Final() gibt für eine erfolgreich verifizierte Funktion “1” zurück. Der Rückgabewert “0” spiegelt eine fehlerhafte Verifikation dar, und muss nicht bedeuten, dass ein anderweitiger Fehler in der Verarbeitung der Daten erfolgt ist.

Referenz	https://www.openssl.org/docs/crypto/EVP_DigestSignInit.html, https://www.openssl.org/docs/crypto/EVP_DigestVerifyInit.html
Funktionspointer-Typ	wie Hash- und Signaturfunktionen(1.4.10.3, 1.4.10.8)
Kontext-Typ	wie Hash- und Signaturfunktionen(1.4.10.3, 1.4.10.8)
Initialisierung Kontext	wie Hash- und Signaturfunktionen(1.4.10.3, 1.4.10.8)
Deinitialisierung Kontext	wie Hash- und Signaturfunktionen(1.4.10.3, 1.4.10.8)
Initialisierung Chiffre	EVP_DigestSignInit(), EVP_DigestVerifyInit()
Verarbeitung von Datenblöcken	EVP_DigestSignUpdate(), EVP_DigestVerifyUpdate()
Verarbeitung des letzten Datenblocks	EVP_DigestSignFinal(), EVP_DigestVerifyFinal()

1.4.10.11 Ableitung von Schlüsseln aus Passwörtern

Die EVP-Bibliothek stellt für die Ableitung von Schlüsselmaterial und Initialisierungsvektoren aus Passwörtern die Funktion EVP_BytesToKey() bereit. Diese wird mit einer symmetrischen Chiffre, einer Hashfunktion sowie weiterer Daten, die bei der Herleitung des Schlüsselmaterials genutzt werden können, parametrisiert.

Bei der Funktion signalisiert ein Rückgabewert von “>=1” erfolgreiche Ausführung, “<0” einen Fehler.

Bemerkung: Diese Funktion sollte zugunsten besserer Ableitungsfunktionen wie PBKDF2 nicht mehr genutzt werden.

Referenz	https://www.openssl.org/docs/crypto/EVP_BytesToKey.html
Herleitung von Schlüsselmaterial	EVP_BytesToKey()

1.4.10.12 Zusammenfassung

Die in OpenSSL für die Verwendung verschiedener Chiffren genutzte EVP-Bibliothek stellt generell ein konsistentes Interface zur Verfügung. Dieses besteht aus einem “Init”-Aufruf mit den notwendigen Parametern zur Initialisierung der Krypto-Operation, darauf folgt die Verarbeitung von Daten mit der “Update”-Funktion, und bei Bedarf eine “Final”-Funktion für die Verarbeitung des letzten Datenblocks.

Die Rückgabewerte der Funktionen sind zum größten Teil vom “int”-Datentyp. Bei den betrachteten Funktionen wurde konsequent durchgehalten, dass ein Fehler mit einem Test auf “<=0” festgestellt werden kann, sowie Erfolg mit einem Test auf “>=1”.

Bei Funktionen, die Pointer zurückgeben, signalisiert ein “NULL”-Rückgabewert einen Fehler. Hier muss für korrekte Funktion also ein Test “!=NULL” durchgeführt werden.

Für die von OpenSSL bereitgestellten Chiffren wird in der EVP-Bibliothek jeweils ein Adapter auf die EVP-Funktionstypen für die jeweilige Algorithmenklasse implementiert. Damit kann Code so geschrieben werden, dass bei der Instanziierung eines Kryptosystems dieses unterschiedlich parametrisiert werden kann. Die Implementierung des Kryptosystems für verschieden Kryptoprimitiven kann aber abstrakt gehalten werden. Diese Vorgehensweise wird in der OpenSSL-Dokumentation empfohlen.

Bemerkung: Das Interface der EVP-Bibliothek stellt sinnvoll abstrahierte Interfaces für die Erstellung von Kryptosystemen bereit. Die Rückgabewerte der Funktionen sind konsistent gehalten. Damit ist die EVP-Bibliothek für die Implementierung sicherer Kryptosysteme geeignet.

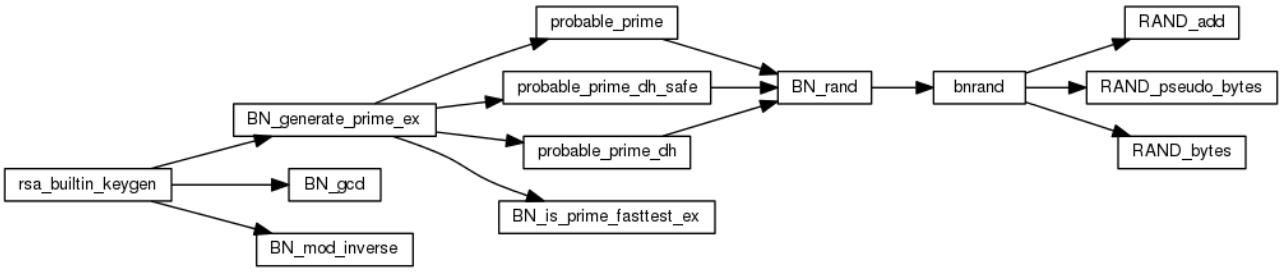
1.5 High-Level Aufrufgraphen für wichtige Funktionen

In diesem Abschnitt werden für die Komponenten und Abhängigkeiten zwischen den Komponenten und Modulen weiter dokumentiert. Es werden Abhängigkeitsgraphen und Aufrufgraphen ausgewählter High-Level Funktionen dargestellt. Der Fokus liegt hierbei auf den in den folgenden Arbeitspaketen näher untersuchten Funktionen.

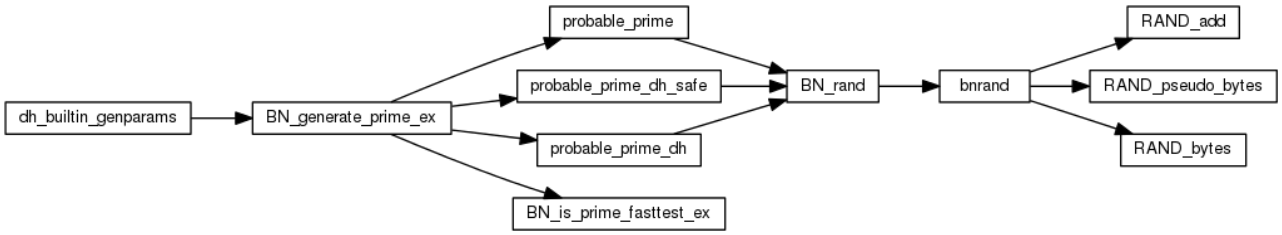
Um das Begreifen der abstrakten Architektur zu vereinfachen wurden die Aufrufgraphen von Verwaltungsfunktionen, trivialen sowie Low-Level Funktionen befreit. Die Graphen sind damit nicht vollständig.

1.5.1 Aufrufgraphen Schlüsselerzeugung

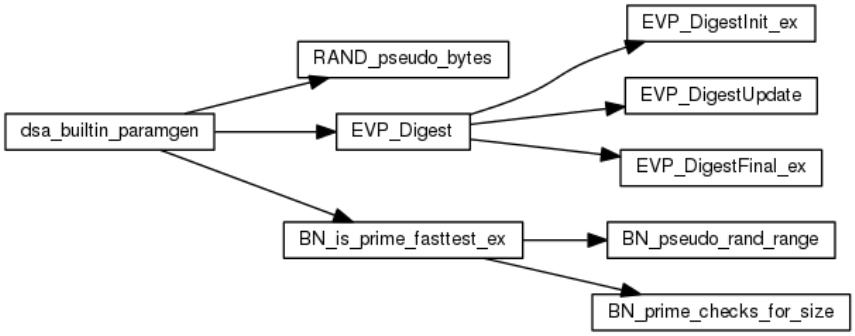
1.5.1.1 RSA Schlüsselerzeugung

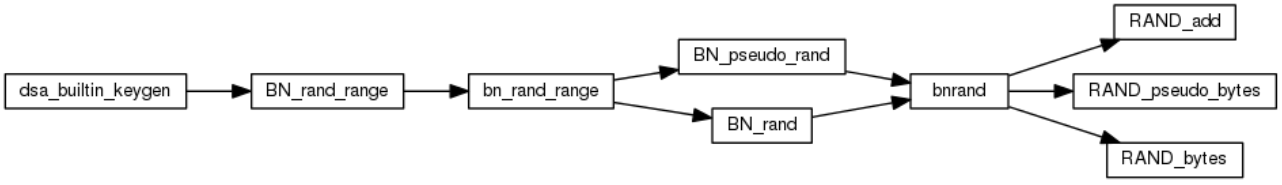


1.5.1.2 Diffie-Hellman Parametererzeugung

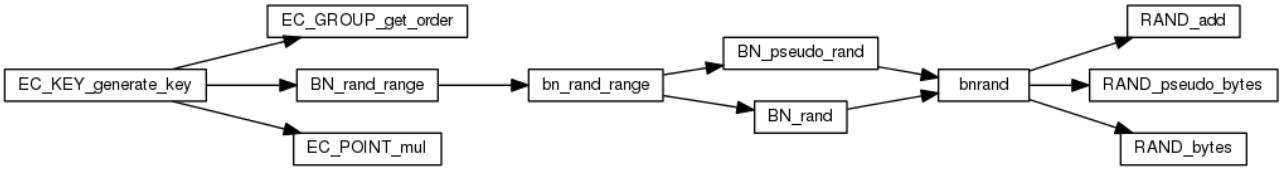


1.5.1.3 DSA Parameter- und Schlüsselerzeugung



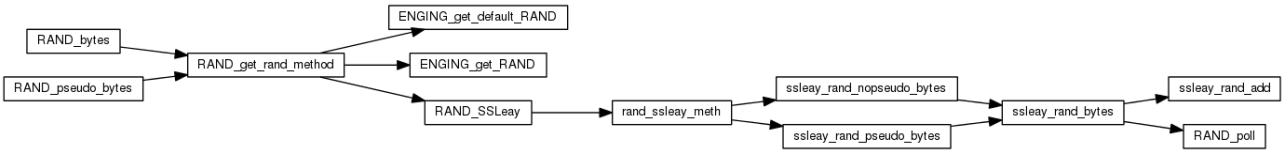


1.5.1.4 EC Schlüsselerzeugung

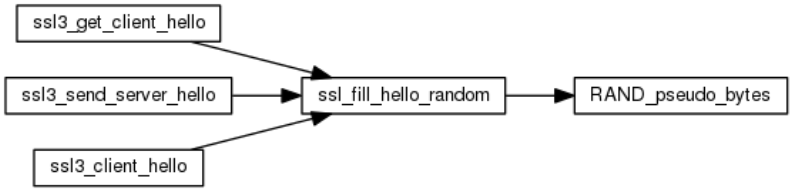


1.5.2 Aufrufgraphen Zufallsdaten

1.5.2.1 Standard-Zufallszahlengenerator

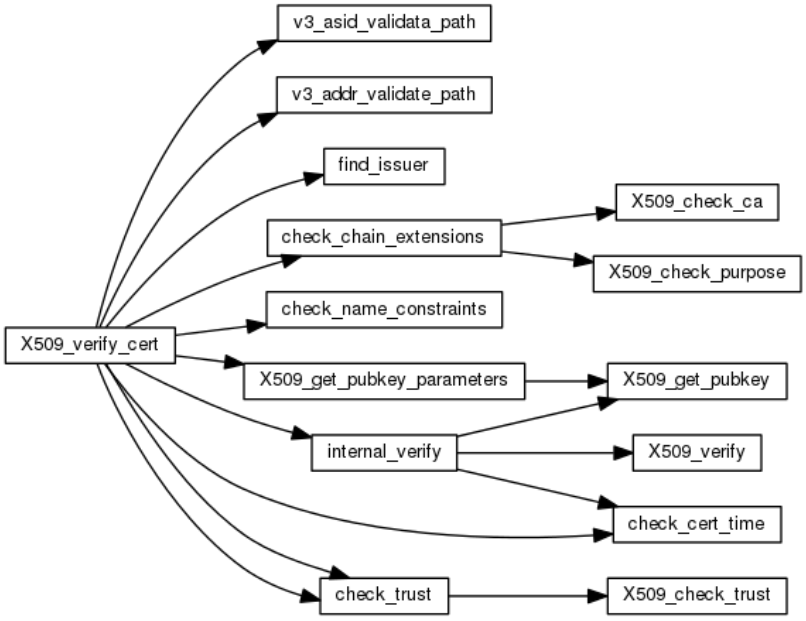


1.5.2.2 TLS Client/Server-Hello Random



1.5.3 Aufrufgraph Zertifikatsverifikation

1.5.3.1 Verifikation eines X.509 Zertifikats



2 Kryptographische Bestandteile der Ciphersuites in TLS 1.2

In diesem Arbeitspaket werden alle zu untersuchenden Ciphersuites aufgelistet, ihre Bestandteile analysiert und im OpenSSL-Quellcode identifiziert.

2.1 Genereller Aufbau der Ciphersuite-Namen

Ciphersuites werden in zwei Byte codiert übertragen; ihre Namen sind nach dem Schema TLS_keyexchangealg_(signalg)_WITH_rl-encryption_rl-keylength_rl-blockmode_rl-hmachash benannt.

Erklärung anhand der Mandatory Ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA.

- TLS: Identischer Präfix für alle in TLS verwendeten Ciphersuites.
- RSA: Gibt an, dass als Schlüsselaustauschalgorithmus (keyexchangealg) RSA Key Encryption eingesetzt wird.
- WITH: Konstanter String, der den Schlüsselaustauschalgorithmus von den im Record Layer (rl) verwendeten Algorithmen abgrenzt.
- AES_128: Die Blockchiffre (rl-encryption) AES wird mit Schlüssellänge 128 Bit (rl-keylength) verwendet.
- CBC: AES wird im CBC-Modus eingesetzt (rl-blockmode).
- SHA: Für die HMAC-Konstruktion des Record Layer wird SHA-1 eingesetzt (rl-hmachash).

2.2 Zu untersuchende Ciphersuites

In unserem Projekt werden folgende Ciphersuites untersucht:

- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_GCM_SHA256
- TLS_DH_RSA_WITH_AES_256_SHA256[2]
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384[3]
- TLS_DHE_PSK_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256

Dazu kommen explizit noch die empfohlenen Ciphersuites aus der TR-02102-2 mit Perfect Forward Secrecy (Tabelle 1):

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 / SHA 384

2 Als Signaturalgorithmus signalg wurde hier zusätzlich zur Leistungsbeschreibung RSA ergänzt
3 Als Signaturalgorithmus signalg wurde hier zusätzlich zur Leistungsbeschreibung RSA ergänzt

- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_DHE_DSS_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA256 / SHA 384

Gleiche Ciphersuites ohne Forward Secrecy aus der TR-02102-2 (Tabelle 2):

- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_ECDH_RSA_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_ECDH_RSA_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_DH_DSS_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_DH_DSS_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_DH_DSS_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_DH_DSS_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_DH_RSA_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_DH_RSA_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_DH_RSA_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_DH_RSA_WITH_AES_256_GCM_SHA256 / SHA 384

Ciphersuites mit Preshared Keys aus der TR-02102-2 (Tabelle 3):

- TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_DHE_PSK_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_DHE_PSK_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_DHE_PSK_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_DHE_PSK_WITH_AES_256_GCM_SHA256 / SHA 384
- TLS_RSA_PSK_WITH_AES_128_CBC_SHA256 / SHA 384
- TLS_RSA_PSK_WITH_AES_256_CBC_SHA256 / SHA 384
- TLS_RSA_PSK_WITH_AES_128_GCM_SHA256 / SHA 384
- TLS_RSA_PSK_WITH_AES_256_GCM_SHA256 / SHA 384

Daraus ergeben sich folgende zu untersuchende Algorithmen:

- keyexchagealg: RSA, DH, DHE, DHE, ECDHE, ECDH, PSK, ECDHE_PSK, DHE_PSK, RSA_PSK
- signalg: RSA, ECDSA, DSS
- rl-encryption_rl-keylength: AES_128, AES_256
- rl-blockmode: CBC, GCM
- rl-hmachash: SHA-1, SHA256 (zusätzlich SHA 384 falls in OpenSSL implementiert)

2.3 Explizit im Namen benannte Kryptofunktionen: Key Exchange

Im Folgenden werden die in den genannten Ciphersuites eingesetzten Kryptofunktionen im Key Exchange beschrieben. Für jede Kryptofunktion werden das Verzeichnis und die relevanten Dateien aufgeführt.

2.3.1 RSA

Verzeichnis: **crypto/rsa**

Relevante Dateien:

- **crypto/rsa/rsa_pk1.c**: Implementiert die PKCS#1 v1.5 Padding-Funktionalität.
- **ssl/s3_srvr.c**: Implementiert RSA-Entschlüsselung im Rahmen des TLS-Handshakes (Zeilen 2206 – 2258).

Bei dem RSA-basierten Schlüsselaustausch wird der RSA-Algorithmus mit dem PKCS#1 v1.5 Padding für die Verschlüsselung des Premaster Secrets PMS (46 Bytes) eingesetzt. Bei dem Schlüsselaustausch generiert der Client einen PMS und konkateniert es mit zwei Protokollversion Bytes, für TLS 1.2:

0x03 0x03 PMS

Anschließend wendet er auf diese 48 Bytes das PKCS#1 v1.5 Padding an, so dass die Nachricht die Länge des RSA-Schlüssels erreicht:

0x00 0x02 non-zero padding 0x00 0x03 0x03 PMS

Dabei ist es wichtig, dass das zufällig generierte Padding kein 0x00 Byte enthält. Diese Nachricht kann mit dem öffentlichen Schlüssel des Servers verschlüsselt werden.

Nach dem Empfangen des verschlüsselten PMS entschlüsselt zuerst der Server die Nachricht mit seinem privaten Schlüssel. Die PKCS#1 v1.5 Struktur und die Protokollversion werden überprüft. Wenn beide korrekt sind, benutzt der Server den entschlüsselten PMS. Andernfalls benutzt er einen zufällig generierten Wert. Die Nutzung eines zufälligen Wertes ist wichtig, um die Bleichenbacher-

Angriffe zu verhindern (siehe RFC 5246, Abschnitt 7.4.7.1).

2.3.2 DH

Verzeichnis: **crypto/dh**

Relevante Dateien:

- **crypto/dh/dh_check.c**: Enthält eine Funktion, welche die generierte Primzahl und die Wahl des Generators überprüft.
- **crypto/dh/dh_gen.c**: Generierung der Parameter.
- **crypto/dh/dh_ameth.c**, **crypto/dh/dh_pmeth.c**: Verschlüsselung, Entschlüsselung und Schlüsselableitung.
- **ssl/s3_srvr.c**: Implementiert DH-Schlüsselaustausch im Rahmen des TLS-Handshakes (Zeilen 2280 – 2322).

Bei TLS-DH übermittelt der Server sein Zertifikat in der Certificate Nachricht. Dieses Zertifikat enthält die Beschreibung der DH-Gruppe, das erzeugende Element und den DH-Share des Servers.

Der Client muss aus dem Zertifikat die Gruppenordnung q entnehmen, einen zufälligen Wert aus [0,q-1] wählen und das erzeugende Element mit dieser Zahl potenzieren. Der resultierende Wert wird in der ClientKeyExchange-Nachricht an den Server gesandt.

Der Server sollte überprüfen ob das erhaltene Element in der Gruppe liegt. Falls ja muss er dieses Element mit seinem zum Zertifikat gehörenden privaten Schlüssel potenzieren. Die 48 least significant bytes des Ergebnisses bilden das Premaster Secret.

2.3.3 DHE

Verzeichnis: **crypto/dh**

Relevante Dateien:

- **crypto/dh/dh_check.c**: Enthält eine Funktion, welche die generierte Primzahl und die Wahl des Generators überprüft.
- **crypto/dh/dh_gen.c**: Generierung der Parameter.
- **crypto/dh/dh_ameth.c**, **crypto/dh/dh_pmeth.c:** Verschlüsselung, Entschlüsselung und Schlüsselableitung.

Bei TLS-DH übermittelt der Server die Beschreibung der DH-Gruppe in einer Primzahl p, das erzeugende Element g und den DH-Share T_S des Servers. Zur Bildung von T_S wird eine zufällige Zahl s aus [0,q-1] gewählt und T_S wie folgt berechnet:

$$T_S := g^s \bmod p.$$

Dies wird in der ServerKeyExchange-Nachricht zusammen mit einer Signatur sig übermittelt, die wie folgt gebildet wird:

$$sig := SIG.Sign(sk_S , r_C||r_S||p||g||T_S).$$

Der genaue Ablauf der Signaturerzeugung hängt von dem gewählten Signaturverfahren ab. Die Signatur muss mithilfe des Serverzertifikats verifizierbar sein.

Der Client muss die Gruppenordnung q bestimmen, einen zufälligen Wert aus [0,q-1] wählen und das erzeugende Element g mit dieser Zahl potenzieren. Der resultierende Wert T_C wird in der ClientKeyExchange-Nachricht an den Server gesandt.

Der Server sollte überprüfen ob das erhaltene Element in der Gruppe liegt. Falls ja, muss er dieses Element mit dem Wert s potenzieren. Die 48 least significant bytes des Ergebnisses bilden das Premaster Secret.

Das Paar (s, T_S) kann aus Performanzgründen für mehrere Schlüsselvereinbarungen genutzt werden. Dies wird in AP 3 genauer untersucht.

2.3.4 ECDH

Verzeichnisse: **crypto/ecdh**, **crypto/ec**

Relevante Dateien:

- **crypto/ecdh/ech_lib.c**: Hilfsfunktionen fürs Kopieren, Löschen und Allozieren von ECDH-Datenstrukturen.
- **crypto/ec/ec_amethod.c**, **crypto/ec/ec_pmethod.c**: Verschlüsselung, Entschlüsselung und Schlüsselverwaltung.
- **crypto/ec/ec_check.c**: Enthält eine Funktion, welche die Diskriminante und den Generator validiert.
- **crypto/ec/ec_curve.c**: Definiert unterschiedliche Kurven.
- **crypto/ec/ec_mult.c**: Implementiert WNAF-basierte EC-Multiplikation.
- **ssl/s3_srvr.c**: Implementiert ECDH-Schlüsselaustausch und Premaster Secret Ableitung im Rahmen des TLS-Handshakes (Zeilen 2520 – 2672).

Der Ablauf ist hier identisch zu TLS-DH, es wird lediglich eine EC-Gruppe anstelle der Primzahlgruppe verwendet.

2.3.5 ECDHE

Verzeichnisse: **crypto/ecdh**, **crypto/ec**

Der Ablauf ist hier identisch zu TLS-DHE, es wird lediglich eine EC-Gruppe anstelle der Primzahlgruppe verwendet.

2.3.6 PSK

Relevante Dateien:

- **ssl/s3_srvr.c**: Implementiert PSK Premaster Secret Ableitung im Rahmen des TLS-Handshakes (Zeilen 2676 – 2768).

Es wird TLS Preshared Key nach RFC 4279 implementiert.

In allen PSK-Varianten enthält die ServerKeyExchange-Nachricht einen „Identity Hint“, damit der Client das richtige Preshared Secret wählen kann. Das gewählte Preshared Secret wird im Folgenden mit PSK bezeichnet.

Bei Verwendung von TLS-PSK entfallen die Certificate-Nachricht des Servers und die ClientKeyExchange-Nachricht des Servers. Das Premaster Secret pms wird direkt aus dem PSK wie folgt gebildet:

$$pms := N||0\ldots0||N||PSK$$

Hierbei ist N die Länge des PSK, dargestellt in 2 Byte.

2.3.7 RSA-PSK

Bei Verwendung von TLS-RSA-PSK benötigt der Server ein Zertifikat mit einem RSA-Schlüssel, der zur Verschlüsselung verwendet werden darf. Der Client geht wie in TLS-RSA-Ciphersuites üblich vor: er wählt einen zufälligen 46-Byte-Wert R, fügt die Versionsnummer 0x0303 für TLS 1.2 davor, codiert den Datensatz nach PKCS#1 und verschlüsselt ihn mit dem öffentlichen Schlüssel des Servers.

Das Premaster Secret wird wie folgt gebildet:

$$pms := C||V||R||N||PSK,$$

wobei C=48, V=0x0303 und N die Länge des PSK ist.

2.3.8 DHE-PSK

Für TLS-DHE-PSK benötigt der Server kein Zertifikat, da die ServerKeyExchange-Nachricht unsigniert übertragen wird.

Das Premaster Secret pms wird wie folgt gebildet:

$$pms := N_Z||Z||N||PSK,$$

wobei Z = DH(T_S,T_C), NZ die Länge von Z codiert als unsigned integer in 2 Byte und N die Länge des PSK (codiert ebenlfalls als unsigned integer in 2 Byte).

2.3.9 ECDHE-PSK

Der Ablauf ist identisch zu DHE-PSK, mit dem einzigen Unterschied, dass hier EC-Gruppen eingesetzt werden.

2.4 Explizit im Namen benannte Kryptofunktionen: Signatur

Im Folgenden werden TLS-Signaturalgorithmen und die relevanten Implementierungsdateien aufgeführt.

2.4.1 RSA

Verzeichnis: **crypto/rsa**

Relevante Dateien:

- **crypto/rsa/rsa_sign.c**: Implementiert RSA_sign und RSA_verify Funktionen. Beim Aufruf werden weitere unterliegende Funktionen aus anderen Dateien benutzt, z.B. für PKCS#1 Enkodierung und Dekodierung. Die Funktion RSA_sign wird z.B. in der **ssl/s3_srvr.c** (Zeile 1946) direkt ausgeführt.
- **crypto/rsa/rsa_pk1.c**: Implementiert PKCS#1 Enkodierung und Dekodierung.

Zum Verifizieren der Signatur wird der Hashwert h der signierten Werte gebildet. Anschließend wird h PKCS#1-codiert, und dieser Wert dient zusammen mit der Signatur und dem öffentlichen Schlüssel als Eingabe für die Verifikationsfunktion.

2.4.2 ECDSA

Relevante Dateien (zusätzlich zu den Dateien definiert in Kapitel 1.3.3)

- **crypto/ec/ecs_sign.c**: Berechnet EC Signatur.

ECDSA ist die Elliptic-Curve Analogie des DSA-Algorithmus.
Für die genaue Funktionalität des ECDSA verweisen wir auf den ANSI Standard "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", welcher auch in RFC 5246 referenziert wird.

Bei dem ECDSA können außerhalb von SHA-1 andere zusätzliche Hashfunktionen eingesetzt werden.

Es ist sehr wichtig, dass bei der Signaturerstellung kritische Überprüfungen wie bei dem DSA-Algorithmus durchgeführt werden. Zusätzlich muss die ECDSA Signaturerstellung in einer konstanten Zeit erfolgen, sonst kann dies zur Kompromittierung des privaten Schlüssels führen

[Brumley2003].

Enkodierung:
Die Signatur (r,s) wird DER-enkodiert und im Protokoll direkt eingesetzt.

2.4.3 DSS

Verzeichnis: **crypto/dsa**

Relevante Dateien:

- **crypto/dsa/dsa_ameth.c, crypto/dsa/dsa_pmeth.c:** Signaturverfahren und Hilfsfunktionen.
- **crypto/dsa/dsa_sign.c:** Aufruf der Signaturfunktion.

Notiz:
- DSA referenziert den Digital Signature Algorithm
- DSS referenziert den NIST Digital Signature Standard, DSS wird in der SSL-Terminologie verwendet

Für die genaue Funktionalität des DSS verweisen wir auf den NIST "Digital Signature Standard", welcher auch in RFC 5246 referenziert wird. Eine vereinfachte Darstellung der Signaturerstellung ist im Folgenden gegeben.

Parameter:
- Hashfunktion H (in TLS 1.2 wird SHA-1 benutzt)
- Schlüssellängen N und L
- Primzahl q der Länge N
- Primzahl Modulus p der Länge L, wobei: q = 0 mod (p-1)
- Generator g

Schlüsselerzeugung:
- Generiere eine zufällige Zahl x, wobei 0 < x < q
- Berechne y = g^x mod p
- Öffentlicher Schlüssel: (p, q, g, y)
- Privater Schlüssel: x

Signaturerstellung: Bei der Signaturerstellung für die Nachricht m wird wie folgt vorgegangen.
- Generiere einen zufälligen Wert k: 0 < k < q
- Berechne r = (g^k mod p) mod q
- Berechne s = k^(-1) (H(m)+xr) mod q
- Signatur: (r,s)

Kritische Überprüfungen während der Signaturerstellung:
- Der zufällige Wert k muss bei jeder Signaturerstellung neu generiert werden, sonst kann der Angreifer den privaten Schlüssel selber berechnen
- Wenn r=0 muss der randomisierte Wert k neu erstellt und der Prozess neu gestartet werden

- Wenn s=0 muss der randomisierte Wert k neu erstellt und der Prozess neu gestartet werden

Enkodierung:
Die Signatur (r,s) wird DER-enkodiert und im Protokoll direkt eingesetzt (siehe RFC 5246, Kapitel 4.7).

2.5 Explizit im Namen benannte Kryptofunktionen: Record Layer Encryption

Im Folgenden werden TLS-Verschlüsselungsalgorithmen und die relevanten Implementierungsdateien aufgeführt.

2.5.1 AES 128 / AES 256

Verzeichnis: **crypto/aes**

Relevante Dateien:

- **crypto/aes/aes_core.c**: AES-Implementierung.

AES wird gemäß Spezifikation ausgeführt.

2.6 Explizit im Namen benannte Kryptofunktionen: Record Layer Blockmode

2.6.1 CBC

Verzeichnis: **crypto/modes**

Relevante Dateien:

- **crypto/modes/cbc128.c**: CBC-Modus für AES und andere Blockchiffren.

CBC wird gemäß Spezifikation ausgeführt.

2.6.2 GCM

Verzeichnis: **crypto/modes**

Relevante Dateien:

- **crypto/modes/gcm128.c**: GCM mode für AES Blockschiffren.

GCM ist spezifiziert in:

Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Federal Information Processing Standards Publication 800- 38 D, 2007.

2.7 Explizit im Namen benannte Kryptofunktionen: Record Layer HMAC

Verzeichnis: **crypto/hmac**

Relevante Dateien:

- **crypto/hmac/hmac.c**: HMAC-Implementierung. Der zu verwendende Hash-Algorithmus wird über das EVP Interface definiert, vgl. Kapitel 1.4.10.
- **crypto/evp/m_sha1.c**: Enthält die Implementierungen für verschiedene SHA-Algorithmen.

OpenSSL bietet $HMAC_{SHA1}$, $HMAC_{SHA256}$ und $HMAC_{SHA384}$ an. Alle Varianten sind in der Datei **hmac.c** zu finden.

2.8 Implizit enthaltene Kryptofunktionen

2.8.1 Zufallszahlengeneratoren

Zufallszahlengeneratoren werden in TLS für verschiedene Aufgaben eingesetzt:

1. Erzeugung der Nonces in ClientHello und ServerHello (46 oder 48 Byte in TLS 1.2)
2. Erzeugung der Exponenten für TLS-DH (nur Client) und TLS-DHE (zufälliges Element aus [0,q-1])
3. Erzeugung des Non-Zero Padding in der PKCS#1-Encryption-Codierung (Client)
4. Erzeugung eines zufälligen Premaster Secrets im Fall einer inkorrekten PKCS#1-Codierung des RSA-Plaintextes.
5. Erzeugung von Initialisierungsvektoren, z.B. für den CBC- oder GCM-Modus.

2.8.2 TLS-PRF

Relevante Dateien:

- **ssl/t1_enc.c**: tls1_PRF() Funktion in dieser Datei implementiert die TLS-PRF Funktionalität.

TLS definiert und verwendet in allen gängigen Versionen eine Pseudozufallsfunktion, die Ausgaben beliebiger Länge machen kann. Diese Funktion nutzt eine HMAC-Konstruktion: Für TLS 1.0 und 1.1 sind dies $HMAC_{MD5}$ und $HMAC_{SHA1}$, für TLS 1.2 ist dies $HMAC_{SHA256}$. Darüber hinaus ist es ab TLS Version 1.2 möglich für jede neue Ciphersuite eine eigene PRF zu definieren.

Die TLS-PRF ist im Gegensatz zur kryptographischen Definition von Pseudozufallsfunktionen für 3 Argumente definiert:

TLS-PRF(key, label, data).

Die TLS-PRF wird genutzt um das Master Secret abzuleiten, um die beiden FINISHED-Nachrichten zu berechnen und um das Schlüsselmaterial abzuleiten:

master_secret = TLS-PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)[0..47];

client_finished = TLS-PRF(master_secret, "client finished", Hash(handshake_messages)) [0..verify_data_length-1]

server_finished = TLS-PRF(master_secret, "server finished", Hash(handshake_messages)) [0..verify_data_length-1]

key_block = PRF(SecurityParameters.master_secret, "key expansion", SecurityParameters.server_random + SecurityParameters.client_random)

= client_write_MAC_key + server_write_MAC_key + client_write_key + + server_write_key + client_write_IV + server_write_IV

2.8.2.1 Bestandteile TLS-PRF in TLS 1.2

Der Aufbau der TLS-PRF wurde für TLS 1.2 wesentlich geändert: „The MD5/SHA-1 combination in the pseudorandom function (PRF) has been replaced with cipher-suite-specified PRFs. All cipher suites in this document use P_SHA256.“ [RFC 5246, Abschnitt 1.2]. Dieser Aufbau soll hier dargestellt werden. Da alle in RFC 5246 genannten Ciphersuites eine TLS-PRF auf Basis von SHA_{256} verwenden, und da keine spezifischen TLS-PRF-Konstruktionen in den im Auftrag genannten Ciphersuites vorkommen, erfolgt die Beschreibung auf Basis von SHA_{256}.

2.8.2.2 Konstruktion der TLS-PRF aus P_SHA256

In diesem Schritt wird lediglich klar gestellt, dass das zweite und dritte Argument konkateniert werden. Dadurch entspricht die Syntax von P_SHA256 der Standardsyntax von Pseudozufallsfunktionen.

TLS-PRF(secret, label, seed) = P_SHA256(secret, label + seed)

2.8.2.3 Konstruktion von P_SHA256 aus $HMAC_{SHA256}$

Die Funktion P_SHA256 liefert bei Eingabe von secret und seed eine Konkatenation von $HMAC_{SHA256}$-Werten zurück. Der Schlüssel für alle HMAC-Aufrufe ist secret, es wird lediglich der zweite Eingabewert variiert: seed wird für jeden Aufruf mit einem Präfix versehen, der sich aus dem rekursiven Aufruf von HMAC_SHA256 mit Schlüssel secret ergibt, wobei der zweite Eingabewert das Ergebnis des vorangegangenen Aufrufs ist.

P_SHA256(secret, seed) = HMAC_SHA256 (secret, A(1) + seed) +
HMAC_SHA256 (secret, A(2) + seed) +
HMAC_SHA256 (secret, A(3) + seed) + …

A(0) = seed

A(i) = HMAC_SHA256 (secret, A(i-1))

2.8.2.4 Konstruktion von $HMAC_{SHA256}$ aus SHA_{256}

Die HMAC-Konstruktion ist standardisiert, es gibt hier keine TLS-Besonderheiten.

Die Hash-Funktion wird als Pointer über das EVP Interface angegeben (Kapitel 1.4.10).

Somit liegt eine modulare HMAC-Konstruktion vor, die grundsätzlich beliebige Hashfunktionen benutzen kann. Die Implementierung wird in **crypto/md32_common.h** umgesetzt. Dies geschieht über Präprozessor-Direktiven die in der Datei **crypto/sha/sha_locl.h** gesetzt werden.

2.8.3 TLS-Padding (Record Layer)

Der TLS Record Layer implementiert ein MAC-then-PAD-then-Encrypt-Verfahren. Dieses ist theoretisch nur dann sicher, wenn das Padding eine hinreichende Länge hat [Paterson2011]. Daher ist im weiteren Verlauf (AP3) zu untersuchen ob OpenSSL es erlaubt, das Padding in der im obigen Artikel beschriebenen Form sicher zu implementieren.

2.8.4 TLS Sequenzzähler (Record Layer)

Der TLS Record Layer implementiert ein „Stateful Encyrption“-Verfahren. Dazu wird zur Berechnung des HMAC nicht nur die Nachricht selbst herangezogen, sondern es fließt außerdem noch ein Zählerwert mit ein. Im weiteren Verlauf (AP3) soll daher untersucht werden wie dieser Zähler initialisiert und wo er geführt wird.

Bibliographie

[FIPS197] NIST: "Announcing the ADVANCED ENCRYPTION STANDARD (AES)"
[SP800] NIST: "Recommendation for Block Cipher Modes of Operation"
[MAKE] The Open Group: "make - maintain, update and regenerate groups of programs"
[SSL-evp] OpenSSL Project: "bio", https://www.openssl.org/docs/crypto/evp.html
[SSL-bio] OpenSSL Project: "bio", https://www.openssl.org/docs/crypto/bio.html
[SSL-pem] OpenSSL Project: "pem", https://www.openssl.org/docs/crypto/pem.html
[SSL-x509] OpenSSL Project: "x509", https://www.openssl.org/docs/crypto/x509.html
[SSL-ec] OpenSSL Project: "ec", https://www.openssl.org/docs/crypto/ec.html
[SSL-bn] OpenSSL Project: "bn", https://www.openssl.org/docs/crypto/bn.html
[SSL-err] OpenSSL Project: "err", https://www.openssl.org/docs/crypto/err.html
[SSL-crypto] OpenSSL Project: "crypto", https://www.openssl.org/docs/crypto/crypto.html
[CVE-2014-0092] CVE-2014-0092: "GnuTLS goto fail", http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0092
[GTOF] Adam Langley: "Apple's SSL/TLS Bug", https://www.imperialviolet.org/2014/02/22/applebug.html
[GCM] David A. McGrew, John Viega: "The Galois/Counter Mode of Operation (GCM)"
[POD] Larry Wall, Sean M. Burke: "perlpod - the Plain Old Documentation format", http://perldoc.perl.org/perlpod.html
[GNUAM] GNU Project: "Automake", http://www.gnu.org/software/automake
[Brumley2003] David Brumley, Dan Boneh, „Remote Timing Attacks are Still Practical“, Cryptology ePrint Archive, Report 2011/232
[Paterson2011] Kenneth G. Paterson, Thomas Ristenpart, Thomas Shrimpton, „Tag Size Does Matter: Attacks and Proofs for the TLS Record Protocol“, In Proceedings of the ASIACRYPT 11 Conference, 2011