Interleaved Integrity Padding

Vorgeschlagenes Kürzel: “IIP” = Interleaved Integrity Padding.

Zweck und Motivation

Dieses Verfahren dient dazu, die Integrität von verschlüsselten Nachrichten sicherzustellen, ohne ein Merkmal der Nachricht (wie z.B. eine HMAC) zu verwenden.

Der Bedarf hierfür wurde durch patentrechtliche Gründe geschaffen. Es war strikt zu vermeiden, eine wie auch immer abgeleitete Eigenschaft der Nutzdaten (Plaintext) zu verwenden; daher durften die üblichen MAC-Verfahren Hash/HMAC/… nicht zur Anwendung kommen.

Grundidee

Kryptographische Verschlüsselungsverfahren weisen nach Shannon die Eigenschaften Confusion and diffusion auf.

Für unsere Zwecke interessant ist hierbei die Diffusion: jedes Bit des Plaintexts soll mit möglichst vielen Bits im Ciphertext verbunden sein, und umgekehrt. Verwandt ist der Avalanche Effect

Diese Erfindung basiert darauf, daß den eigentlichen Nutzdaten zusätzliche Schutzdaten mitgegeben werden. Diese können zufällig erzeugt, festgelegt, oder separat ausgehandelt sein.

Nach Entschlüsselung der empfangenen Nachricht werden die Schutzdaten auf ihre Konsistenz geprüft. Sollte die verschlüsselte Nachricht verändert worden sein, so sind durch Diffusion auch die Schutzdaten verändert worden. So kann man anhand der Schutzdaten die Integrität einer Nachricht prüfen, ohne die Nutzdaten zu kennen oder darüber irgendwelche Prüfsummen zu bilden.

Hinweis für Zitate: Erfunden wurde dieses Verfahren 2022 von Jo Wilkes, metabit. Anlass war ein Wunsch des SAFE e.V., für dessen Gebrauch die Referenzimplementierung im Auftrag der ABL GmbH angefertigt wurde.

Fortsetzung der Idee

Ein Blockzähler kann auf die Schutzdaten aufmoduliert werden, so daß diese pro Block variieren und die Reihenfolge der Blöcke geprüft werden kann. (auch: Schutzdaten als Pseudozufalls-sequenz)

Es können zusätzlich jeweils nicht genutzte Zufallsdaten hinzugefügt werden, in dem ersten und/oder folgenden Blöcken, damit die Nachrichteninhalte weniger vorhersehbar bzw. formbar (malleable) sind.

Der verwendete Blockcipher kann eine zusätzliche Verkettung der Blöcke bereitstellen.

Caveat

Werden mehrere Datenblöcke versendet, muß je Datenblock mindestens ein Satz Schutzdaten vorhanden sein.

Das eingesetzte Verschlüsselungsverfahren ist grundsätzlich auf seine Diffusion zu prüfen. Die Wirksamkeit des IIP ist von der Diffusion des Verschlüsselungsverfahrens abhängig, und der Avalanche Effect sollte möglichst umfassend sein.

Die Schutzwirkung hängt von Verschlüsselungsalgorithmus und Verhältnis Schutzdaten zu Nutzdaten ab. Insbesondere ist zu beachten: Die üblichen Streaming Ciphers, und die Operation Modes für symmetrische Ciphers welche streaming entsprechen, weisen *keine* Diffusion auf. Bei CFB, OFB, CTR und ähnlichen Modi kann man ein Bit des Ciphertexts ändern und exakt ein Bit des Plaintexts damit ändern. Das macht sie für Verwendung mit IIP untauglich. Dieses Verfahren ist also keineswegs unabhängig von der übrigen Kryptographie, sondern im Gegenteil mit dieser genau abzustimmen.

(Bei Verfahren mit substitution tables ist beispielsweise wünschenswert, daß die verwendete Verschlüsselungsfunktion das SAC (Strict Avalanche Criterion) erfüllt; zumindest aber sollte das nicht-strikte Avalanche Criterion erfüllt sein.)

Die vorliegende Implementierung ist in Schichten aufgebaut.

Komponenten

Reihenfolge bottom-up: * IntegrityPaddingWithNonceForSymmetricCiphers - padded Rohdaten; Kern des Verfahrens * SymmetricEncryptionWithIntegrityPadding - verschlüsselt gepaddete Rohdaten symmetrisch * RSAWithIntegrityPadding - verschlüsselt gepaddete Rohdaten mit RSA asymmetrisch

Diese Klassen sind der Kern der Implementation. Sie stellen Verschlüsselung mit Integritätssicherung ohne Verwendung einer MAC bereit.

Nächste Ebene: * ECHDHEWithIntegrityPadding - führt ECHDE durch, verschlüsselt dann gepaddete Rohdaten symmetrisch

Die dabei verwendeten/erzeugten Daten zu Einmalkennung für ECHDE, symmetrischem IV, verschlüsselten Daten usw. werden gebündelt:

- TransportFormatConverter - wandelt das Datentupel nach und von ASN.1-Transportformat
- SAFESealSealer - bündelt den gesamten Vorgang. Dies ist die Klasse, welche Anwender verwenden sollten.

Folgende Klassen haben unterstützende Funktion:

- SharedConstants enthält vor allem die Algorithmen-kennungen, aber auch feste Werte für Version, Nonce-Größe u.a.m.

- AlgorithmSpec - Parametrisierung eines Algorithmus, Abstraktion mit Größenangaben u.a.m.

- AlgorithmSpecCollection - Tabelle zum Nachschlagen unterstützter Algorithmen und ihrer Parameter.

- CryptoFactory - liefert zu AlgorithmSpec ausführbare Instanzen. Kapselt java.security.Provider.

- CryptoSettingsStruct - Struktur zum Bündeln der Kennungen der verwendeten Algorithmen

- InternalTransportTuple - Struktur zum Bündeln der verwendeten Daten

- SharedCode enthält kleine Codefragmente, welche wiederverwendet werden

- ByteIntegerConversion enthält weitere Codefragmente zur Wiederverwendung

Ablauf Prüfung und Entsiegelung

Für eine Top-Down-Analyse ist also der Ablauf der Prüfung und Entsiegelung wie folgt: 1. Aufsetzen der Umgebung (Crypto-Bibliothek, Algorithmen); Bereitstellung der Schlüssel 2. Zunächst ist das versiegelte Format mittels des TransportFormatConverter in ein InternalTransportTuple umzuwandeln, 3. Die dort angegebenen Algorithmen sind mit den verfügbaren/akzeptierten abgleichen und Instanzen bereitstellen 4. Dann ist mittels ECHDHEWithIntegrityPadding aus den asymmetrischen Schlüsseln und Diversifikationsdaten der ephemeral key errechnet. 5. Mit diesem wird dann in SymmetricEncryptionWithIntegrityPadding die Entschlüsselung durchgeführt. 6. Die entschlüsselten Rohdaten werden vom ECHDHEWithIntegrityPadding auf Integrität geprüft. 7. Wird diese Prüfung mit positivem Ergebnis abgeschlossen, werden die Nutzdaten an den Aufrufer übergeben; andernfalls wird das Verfahren mit Fehler abgebrochen.

Integration des IIP-Verfahrens für praktischen Gebrauch

Ausformung des Verfahrens für die Praxis

Erklärt man das Verfahren allgemein, kann man die Schutzdaten mit festen Werten wie z.B. “SAFE” darstellen. In der Praxis sind festgelegte Werte jedoch ungünstig, da für Angreifer vorhersehbar; man bevorzugt daher Zufallsdaten.

Es wird ein zufälliger Einmalwert (nonce) erzeugt. Dieser wird den Nutzdaten vorangestellt, und pro zu verschlüsselndem Datenblock mindestens je einmal ein davon abgeleiteter Wert hinzugefügt. Weitere Leitinformationen vor, und ggf. notwendiges padding auf blockgröße nach den so resultierenden Daten werden angefügt. Die Datenblöcke werden mit einem geeeigneten (!) Verschlüsselungsverfahren und Chaining Mode / Operation Mode verschlüsselt.

Bei der Verschlüsselung finden Diffusion und Confusion statt, so daß in verschlüsseltem Zustand die Schutzdaten mit den Nutzdaten verflochten sind.

Prüfvorgang: Nach Entschlüsselung wird die Integrität der Schutzdaten überprüft. Hierzu nimmt man den vor den Daten stehenden Wert und vergleicht ihn je Datenblock mit dem korrespondierenden Daten im

entschlüsselten Datenblock. Diese müssen der spezifischen Erwartung gemäß zueinander passen; tun sie es nicht, liegt ein Integritätsfehler vor.

Als Verbesserung eines einfachen wiederholten Fest- oder Zufallswerts kann man den Zufallswert 1. diesen als Ganzzahl ansehen und hochzählen (analog zum CTR-mode); 2. diesen als Startwert einer komplexeren Erzeugungsfunktion ansehen und den Datenstrom auf Richtigkeit prüfen.

Mit dieser Verbesserung wird auch bei ECB und ähnlichen Chainings/operation modes erkannt, wenn Datenblöcke im Ciphertext vertauscht wurden. Eine Parallelisierung der Ver- und Entschlüsselung ist damit weiterhin möglich, aber eine Parallelisierung der Angriffe sollte erschwert sein.

Aufbau

kryptographische Darstellung

Die zu verschlüsselnde Nachricht wird zusammengesetzt aus:

```
( I | R0 | P | Mlen ) | ( P_i M_i )+ | (P_i M_i R1)* |
(P_i | R2)*
```

Zeichen	Bedeutung
P	padding nonce für dieses padding (“inner nonce”)
Mlen	message length, länge des eigentlichen Inhalts
I	“magic ID” - ggf. gekürzt, so daß stets ein R0 vorhanden ist.
R0	Zufallsdaten, um stets im ersten Block ungenutzen Zufall zu haben
P_i	abgeleitete padding werte an Position “i”
M_i	Abschnitt Nummer “i” der Nutzdaten
+	Wiederholung des Musters (mindestens ein Mal), Datenblöcke mit steigendem i
*	letzter Block wird abhängig vom “Rest” nach Aufteilung der Nutzdaten gebildet
R1	optionales padding im letzten Datenblock auf Blockgröße des Ciphers
R2	Zufallsdaten zum Auffüllen zur Blockgrenze, falls trailing block nötig

In obiger Notation sind jeweils die in () gestellten Teile in ihrer Länge auf die Blockgröße des Verschlüsselungsverfahrens abgestimmt.

Die Größe von P_i + M_i muß exakt der Blockgröße des Verschlüsselungsverfahrens entsprechen; in der Praxis wird die Größe von M_i aus dieser abzüglich der Größe von P_i berechnet.

(Im Rahmen dieses Dokumentes wird _i geschrieben; wo entsprechende Formatierung verfügbar ist, soll dies mit einem tiefergestellten i dargestellt werden.)

Wenn die Aufteilung der Nutzdaten nicht genau aufgeht, also für R1 ein Platz ist, wird als letzter Block (P_i M_i R1) geschrieben. Geht die Aufteilung der Nutzdaten jedoch genau auf und die Länge eines solchen R1 wäre 0 (anders gesagt: ist die Länge vom letzten M_i == 0), dann wird der Block (P_i | R2) angehangen. So endet die gepaddete Sequenz in jedem Fall auf Zufallsdaten.

In obiger Darstellung wird, wie bei Kryptographie üblich, das Zeichen "|" als Symbol für Verkettung verwendet. Daher steht es für die Darstellung der Alternative (entweder "(P_i M_i R1)" oder "(P_i | R2)") nicht zur Verfügung.

Darstellung für Implementation

Einer Nachricht **M** mit einer Länge von **M_len** bytes ist mit IIP zu verarbeiten. Zudem ist als Input notwendig, die Blockgröße des Verschlüsselungsverfahrens zu kennen, diese sei **CB_len**.

(Die Nachricht "M" wird hier auch als Payload oder "Plaintext" bezeichnet, im Kontrast zum "Ciphertext" nach Verschlüsselung.)

Die Nachricht muß in mehrere aufeinanderfolgende Datenblöcke aufgeteilt werden, welche dann jeweils verschlüsselt werden. Die Länge dieser Blöcke CB_len hängt von Algorithmus und Schlüsselgröße ab. Beispielsweise beträgt bei RSA 2048 bit diese Blockgröße 255 byte (2048/8=256; da das MSB stets gesetzt sein muß, wird von Implementationen das oberste Byte belegt/gesperrt).

Folgende Typen von Datenblöcken werden erzeugt: 1. der Headerblock, der stets als erstes gesendet wird; 2. "mittlere" Datenblöcke, wenn noch weitere Datenblöcke folgen. 3. ein letzter Datenblock in der Variante, wenn nach den Nutzdaten noch Platz übrig war; 4. ein letzter Datenblock in der Variante, wenn die Nutzdaten ohne Rest in den vorigen Block gepasst haben.

Es gibt nur einen letzten Datenblock, wobei abhängig von der Nachrichtenlänge entweder Typ 3 oder Typ 4 verwendet wird.

Der erste Block setzt sich wie folgt zusammen: 1. eine feste Bytefolge **I** 2. Zufallsdaten **R0** 3. die Länge der verpackten Nutzdaten **M_len** 4. der initiale Padding Nonce **P**

Die feste Bytefolge **I** dient als Magic ID mit Versionsnummer, auch um vorweg die zu verwendenden Konstanten bestimmen zu können. Diese Konstanten sind insbesondere Repräsentationsgrößen: wieviele Bytes **M_len** und **P** groß sind. In Version 1 gelten folgende Werte:

Kennung	Länge	Wert
I	8 (-x)	3e 7a b1 7C 5A FE E4 10
M_len	4	Payload=Plaintext-Länge, in byte
P	4	zufällig erzeugter Startwert

Zwischen I und M_len wird mit Zufallswerten R0 aufgefüllt, so daß insgesamt die Blockgröße CB_len entsteht. Für den Fall, daß die Blockgröße so klein sein sollte, daß zuwenig Platz ist, wird I gekürzt, jedoch auf nicht weniger als 3 byte. Mindestens ein Zufallsbyte R0 wird dem ersten Block immer mitgegeben.

Im ersten Block werden keine Plaintext-Daten untergebracht, da P hier zum ersten Mal erscheint - es gibt noch keinen Vergleichswert, zu dem eine Abweichung auffallen könnte. Nutzdaten im ersten Block wären ungeschützt.

In jedem nachfolgenden Datenblock werden Schutzdaten Pi und Nutzdaten kombiniert; im letzten zusätzlich noch Zufallswerte. Da für die Schutzdaten jeweils länge(P) bytes benötigt werden, bleiben jeweils CB_len - sizeof(P) nutzbare Bytes pro Block.

Daraus läßt sich die Anzahl benötigter Blöcke berechnen: Anzahl Plaintextbytes dividiert durch Anzahl nutzbare Bytes, aufgerundet; und falls die Division mit Rest 0 aufgehen sollte, noch ein Block extra.

Hieraus wiederum kann die Anzahl Bytes im Zielpuffer berechnet, und dessen Größe vorab geprüft - oder der Puffer passend bereitgestellt werden.

Die Nutzdaten werden nun aufgeteilt in Abschnitte der Länge “nutzbare bytes”; die Blöcke werden befüllt mit dem jeweils aktuellen Padding Pi und dem jeweiligen Abschnitt der Nutzdaten. (Blocktyp 2)

Das Padding Pi wird in Version 1 nicht einfach wiederholt, sondern als vorzeichenloser Integer mit Wraparound betrachtet und pro Block hochgezählt. Auf diese Weise ist ein “inneres Chaining” vorhanden, durch welches Vertauschung von Blöcken erkannt werden würde. (Spätere Versionen können Pseudozufalls-Datenströme verwenden, welche kryptanalytisch schwerer vorherzusagen sind.)

Wenn die Aufteilung der Nutzdaten mit den Blöcken nicht komplett aufging, so sind im letzten Block noch freie Bytes vorhanden; diese werden mit Zufallsdaten aufgefüllt (Blocktyp 3).

Sollte die Aufteilung jedoch glatt aufgegangen sein, wird ein weiterer Block angefügt, welcher außer dem nächsten Pi nur Zufallsdaten enthält (Blocktyp 4). So endet die zu verschlüsselnde Nachricht stets auf nicht genutzte Zufallsdaten.

Die so gepaddete Nachricht ist nun mit dem gewählten Verschlüsselungsalgorithmus zu verschlüsseln. Dabei ist darauf zu achten, daß dessen korrekte Variante gewählt wird. Ein weiteres Padding ist nicht nötig; bei den operation modes (chaining) ist sehr darauf zu achten, daß diese die Sicherheit des Verfahrens nicht gefährden.

RSA

Die Verwendung mit RSA findet wie zuvor beschrieben statt; die gepaddeten blocks werden mit “plain” RSA verschlüsselt.

NB: Die gängigen Implementierungen verwenden auch bei RSA/ECB/NoPadding mindestens 1 byte für eigene Zwecke; bei keysize 2048 bit = 256 byte sind also nur 255 byte pro Block verfügbar. Ein äußeres Chaining ist möglich, jedoch nicht notwendig.

Bei RSA sind RSA/ECB/NoPadding und RSA/CBC/NoPadding zulässig; für die Referenzimplementierung wird RSA/ECB/NoPadding verwendet.

EC

Eine Verwendung mit ECC direkt scheint möglich, aber für die praktische Verwendung durch Integratoren nicht angeraten. Günstiger für Integration und Prüfung ist es, von etablierten Verfahren auszugehen. Daher wurde ECDHE gewählt, um mittels EC auf ephemeral keys für symmetrische Verschlüsselung zu kommen.

symmetrische Kryptographie

Bei symmetrischer Verschlüsselung werden üblicherweise die Eingangsdaten zusammen mit dem Schlüssel nach bestimmten Mustern “verwirbelt” und verflochten, so daß man diese Muster rückgängig machen kann, aber möglichst schlecht analysieren.

Der Unterschied zu EC, RSA u.ä. besteht darin, daß letzere die Daten als Zahlen, Punkte auf einer Kurve o.ä. interpretieren und damit Rechenoperationen durchführen, wohingegen symmetrische Verfahren die Daten einfach nur als Bit-Sammlungen ansehen, welche sie ohne Interpretation verarbeiten.

Dieser Unterschied ist hier insbesondere wichtig, wenn es um die Verbindung mehrerer Blöcke geht, und Angriffsmöglichkeiten daraus.

Chaining / Operation Mode

Nochmals der Hinweis: Für symmetrische Ciphers sind hier die Operation Modes CFB, OFB, CTR, GCM nicht zu verwenden! Generell sind für dieses Verfahren alle Operation Modes strikt zu vermeiden, welche einen Datenstrom erzeugen und diesen XOR mit den Nutzdaten überlagern. Die für dieses Verfahren benötigte Diffusion ist bei diesen Operation Modes nicht vorhanden!

Bei RSA ist das Chaining weniger relevant, obiger Angriffsvektor besteht so nicht, da die Bits als große Ganzzahl interpretiert werden; jedes Bit Änderung hat große Folgen in der Multiplikation. Ähnlich EC selbst.

Benennung

Der vorgeschlagene Name für dieses Verfahren im Kontext kryptographischer Algortihmen lautet “IIP” = Interleaved Integrity Padding.

Es handelt sich dabei sowohl um ein Padding als auch ein Chaining. “AES/ECB/IIP” und “AES/CBC/IIP” sind mögliche Benennungen von Algorithmen; bei diesen wird aber nicht ausgedrückt, daß auch IIP ein Chaining vornimmt.

Verwendung in OCMF

Es wird der eichrechtlich relevante Teil der Nachricht gekapselt und mit diesem Verfahren verschlüsselt.

Im unverschlüsselten Teil der Nachricht finden sich Informationen über den Absender, aus welchen dessen Public Key gefolgert werden kann (lookup), und ein nonce für die key diversification.

Absender (Ladestation mit Zähler) und Empfänger (Transparenz-Software oder Server-Backend) haben vor, und außerhalb der Anwendung dieses Verfahrens ihre asymmetrischen Schlüsselpaare generiert und deren public keys ausgetauscht.

Für die Schlüsselableitung des symmetrischen Schlüssels wird ein möglichst einmaliger Zugabewert benötigt; dieser kann aus der Paginierung, oder aus einem beliebigen anderen monoton steigendem Zähler stammen. Zufallswerte mit breiter Streuung sind möglich, tragen aber das Risiko einer Kollision/Wiederverwendung desselben symmetrischen ephemeral keys.

Implementation

Ablauf

Um einen Datensatz zu prüfen, müssen zunächst die verschüsselten Daten entschlüsselt werden. Die entschlüsselten Daten werden dann mittels der Prüfung dieses Verfahrens auf Integrität geprüft.

Padding und Verschlüsselung

Verschlüsselungsfunktion auf Absenderseite wird mit den zu verschlüsselnden Daten aufgerufen.

Der Absender erzeugt folgende Zufallswerte: * einen “inneren nonce” (padding-nonce). Im aktuellen Verfahren ist dieser als 32 bit (4 Byte) groß definiert. * falls vom Algorithmus benötigt, einen IV (“äußerer nonce”) * falls key agreement genutzt, einen Zähler- oder Zufallswert für die Schlüsselableitung.

Die Blockgröße des Verschlüsselungsverfahrens muß bekannt sein oder aus dessen Schlüsselgröße gefolgert werden.

RSA: Der Absender-Code erzeugt aus Nutzdaten und “innerem nonce” die zu verschlüsselnde Nachricht. (Details in eigenem Abschnitt.)

Der Private Key des Absenders wird sodann zur Verschlüsselung verwendet. (Mit dem zugehörigen Public Key kann auf Empfängerseite dann die Identität des Absenders geprüft werden.) Zusätzliches Chaining oder Padding ist nicht nötig, da IIP beides bereitstellt. Daher ist RSA/ECB/NoPadding die passende hier verwendete Form des Algorithmus’.

RSA/CBC ist möglich, der hierfür benötigte IV auch in Code und Format vorgesehen.

Von anderen Operation modes/chainings ist nachdrücklich abzuraten.

ECDHE+AES: Der Absender berechnet aus Public key des Empfängers, eigenem Private key, und dem Zufallswert für die Schlüsselableitung eine Anzahl Bytes, welche als “ephemeral key” / Einmalschlüssel für das symmetrische Verschlüsselungsverfahren dienen. Hierbei wird bereits das symmetrische Verschlüsselungsverfahren als Bitstrom-Generator eingesetzt, um gute Schlüsselqualität und flexible Schlüssellänge zu erreichen; andernfalls wäre die Größe des symmetrischen Schlüssels an die Größe des asymmetrischen Schlüssels gekoppelt.

Dann erzeugt der Absender aus Nutzdaten und “innerem nonce” die zu verschlüsselnde Nachricht. (Details in eigenem Abschnitt.)

Die symmetrische Verschlüsselung wird mit dem ephemeral key, dem generierten IV, und den gepaddeten Daten aufgerufen.

Auf Senderseite werden die verschlüsselten Daten nun zusammen mit dem IV und dem Wert für die Schlüsselableitung dem allgemeineren Code für Formatierung in einer OCMF-Nachricht übergeben.

Nota Bene: der “innere nonce” verlässt die Verschlüsselungsfunktion nicht.

Entschlüsselung und Prüfung

Allgemein: Die Blockgröße des Verschlüsselungsverfahrens muß bekannt sein oder aus dessen Schlüsselgröße gefolgert werden.

RSA: Auf Empfängerseite wird zunächst der public key des Absenders ermittelt, sowie ein ggf. benötiger IV extrahiert. Dann wird mit dem public key der verschlüsselte ciphertext entschlüsselt.

ECDHE+AES: Auf Empfängerseite wird zunächst der public key des Absenders ermittelt, sowie IV und Wert für Schlüsselableitung extrahiert. Dann werden diese drei zusammen mit den verschlüsselten daten (ciphertext) und dem private key des Empfängers zur Verarbeitung übergeben. Zunächst wird mittels ECDHE aus public key des Absenders, private Key des Empfängers, und Wert für Schlüsselableitung der verwendete symmetrische Schlüssel abgeleitet. Dieser wird sodann verwendet, um zusammen mit dem IV die verschlüsselten Daten zu entschlüsseln.

Allgemein weiter:

Die entschlüsselten Daten werden dann wie folgt auf Integrität geprüft:

Die Daten werden von Anfang bis Ende gelesen; dabei wird geprüft 1. Die einleitende Kennung ("magic ID") muß einer der erwarteten Werte sein. (u.a. Versionsunterscheidung) 2. Der "innere nonce"/padding-nonce wird übernommen und temporär für die Dauer des Prüfungsvorgangs lokal gespeichert. 3. Die Länge der nachfolgenden Nutzdaten wird gelesen. 4. Die Anzahl der erwarteten weiteren Blöcke wird berechnet, rechnerische Plausiblitätsprüfungen vorgenommen. 5. Die Datenblöcke werden der Reihe nach durchgegangen, das jeweilige padding-nonce-exemplar mit dem erwarteten Wert verglichen. 6. Im letzten Block, welcher auf die Datenblöcke folgt, wird noch einmal der ursprüngliche padding-nonce erwartet. 7. Gegebenenfalls den letzten Nutzdaten nachfolgende Zufallsdaten werden ignoriert.

Wird bei Schritten 1,4,5, oder 6 eine Abweichung festgestellt, so ist die Integrität der Nachricht verletzt worden und es wird ein Fehler zurückgemeldet. Sind alle Prüfungen erfolgreich verlaufen, gilt die Integrität der Nachricht als gesichert und die rekombinierten Nutzdaten aus den Datenblöcken können für die weitere Verarbeitung weitergegeben werden.

Authentizität

Das IIP-Verfahren dient der Integritätsprüfung. Die Authentizität, also die Identität des Absenders, muß auf anderem Wege ermittelt werden.

Bei direkter Verschlüsselung kann die Kenntnis des jeweiligen Private Key/Secret Key verwendet werden, um die Authentizität zu prüfen: wird bei Entschlüsselung nicht der korrespondierende Schlüssel verwendet, schlagen Entschlüsselung und/oder Prüfung fehl.

Wird hierbei asymmetrische Kryptographie verwendet, so ist auch unwesentlich, ob der Private Key des Empfängers korrekt geheimgehalten wurde, oder einem Angreifer bekannt ist: Bei unidirektionaler

Kommunikation geht es um die Identität des Absenders, und dieser hält seinen Private Key geheim.

Werden jedoch symmetrische Schlüssel verwendet, so bedeutet deren Kompromittierung, daß die Authentizität nicht mehr geprüft werden kann. Hierauf ist insbesondere zu achten, wenn (z.B. mittels ECDHE) ein Key Agreement durchgeführt wird: der Ephemeral Key wird von beiden Seiten identisch konstruiert, so daß Inhaber des einen oder anderen Private Keys selbst gültige Nachrichten verfassen können. Bei diesem Szenario hängt also die Authentizität von der Geheimhaltung aller Private keys ab.

Kompression

Eine Inhaltskompression ist vorgesehen; diese dürfte insbesondere bei Textdaten als Plaintext nützlich sein.

Implementationsdetails generell

- Alle Zahlwerte werden Big-Endian gespeichert: das Wichtigste zuerst. (Network Byte Order, konsistent mit kryptographischen Funktionen, ASN.1-Repräsentation usw.)
- Es wird immer mindestens ein ungenutztes Zufallsbyte in den ersten Block gelegt; falls nötig, wird hierfür die MagicID gekürzt.
- Die MagicID lautet 0x3e7ab1705AFEE410.

Für alle bislang definierten Versionen gilt weiter: - Als Größe des “inneren nonce” wird 32 bit (4 byte) gewählt. Damit bleiben bei üblicher symmetrischer Verschlüsselung pro Datenblock 12 byte für Nutzdaten; das Padding vergrößert dann seine Eingabedaten um etwas mehr als 25%. - Als Größenangabe im Block wird 32 bit verwendet; damit ist eine Nutzdatengröße bis knapp 4 GB möglich.

Version 1

Version 1 verwendet RSA. - Wird ohne Key Agreement verschlüsselt, es kommt RSA zum Einsatz.

Version 2

Version 2 ermöglicht ECDHE und AES. - Wird mit Key Agreement verschlüsselt, es kommt ECHDE zum Einsatz. Die verwendete Kurve ist dabei flexibel; secp256r1 sollte jedoch stets möglich sein. - Als

symmetrisches Verschlüsselungsverfahren wird derzeit AES verwendet, mit AES-256 als default.

Transportformat

Für die Verwendung im OCMF-Kontext werden verschlüsselte Daten, Algorithmenkennungen, IV und dergleichen in eine ASN.1-Struktur gepackt. Diese ist wie folgt aufgebaut:

Übersicht

```
SEQUENCE
  OID                   Kennung des Formates
  INTEGER               Version des Formates
  CONTEXT_SPECIFIC[0]   Verschluesselungs-Ebene
  CONTEXT_SPECIFIC[1]   Schluesselaustausch-Ebene
  CONTEXT_SPECIFIC[2]   Authentisierungs-Daten
  OCTET STRING          Verschluesselte Daten
```

Verschlüsselungs-Ebene

Diese Ebene ist notwendig. Je nach Verschlüsselungsverfahren (RSA oder AES) sind weitere Daten wie IV notwendig bzw. überflüssig.

```
CONTEXT_SPECIFIC[0]  Verschluesselungs-Ebene enthält in
Version 1:
    SEQUENCE
        OID             Kennung des Verfahrens
        CONTEXT[0]      informationen über die
Verschlüsselung
            OID         Kennung des
Verschlüsselungsverfahrens
        CONTEXT[1]      Informationen über optionale
Kompression
            OID         Kennung des Kompressionsverfahrens
(default: NONE)
        CONTEXT[2]      optionale Angabe der
Schlüsselgröße
            INTEGER     Schlüsselgröße, in bit
```

```
        CONTEXT[3]      optionale Angabe der Nonce-Größe
(in Version 1 nicht genutzt)
            INTEGER     Nonce-Größe, in bit
        OCTET STRING    IV des symmetrischen
Verschlüsselungsverfahrens (kann z.B. bei RSA und AES/ECB
weggelassen werden)
```

Schlüsselaustausch-Ebene

Diese ist optional; ist sie nicht vorhanden, so muß für die Verschlüsselungsebene das Schlüsselmaterial bereits vorhanden sein.

Diese Ebene wird bislang nur für ECDHE verwendet.

```
CONTEXT_SPECIFIC[1]  Schluesselabgleich-Ebene enthält in
Version 1:
    SEQUENCE
        OID             Kennung des Keyagreement-
Verfahrens
        OCTET STRING    Schlüsseldiversifikationsdaten
        CONTEXT[0]
            OID
Schlüsseldiversifikationsalgorithmus
        CONTEXT[1]      EC-Verfahren (optional, mit
default )
            OID         Kennung der EC
        CONTEXT[2]      EC-Kurve (optional, kein default);
siehe IETF RFC 3279
        CONTEXT[3]      Schlüsselreferenzen (optional,
kein default) ; siehe IETF RFC 5480
```

Authentizitäts-Ebene

Ebene für spätere Verwendung

```
CONTEXT_SPECIFIC[2]  Absender-Authentizitäts-Ebene; in
Version 1 ungenutzt.
    SEQUENCE
        OID             Kennung des Verfahrens
```

Referenzen

Standards

- RFC 8017 betreffs RSA
- RFC 6637 betreffs ECDH und ECC
- RFC 5480 clause 2.1.2 wegen subject key formats
- Ergänzende Erläuterung dazu RFC 8813
- RFC 3279 ECC parameter Format
- RFC 1950 ZLIB Kompressionsformat (optional)

Hintergrund

- Claude E. Shannon, “A Mathematical Theory of Cryptography”, Bell System Technical Memo MM 45-110-02, September 1, 1945.
- Claude E. Shannon, “Communication Theory of Secrecy Systems”, Bell System Technical Journal, vol. 28-4, pages 656–715, 1949.
- “Information Theory and Entropy”. Model Based Inference in the Life Sciences: A Primer on Evidence. Springer New York. 2008-01-01. pp. 51–82. doi:10.1007/978-0-387-74075-1_3. ISBN 9780387740737.

image cited after

Hussain, Dr & Jamal, Sajjad Shaukat & Shah, Tariq & Hussain, Iqtadar. (2020). A Power Associative Loop Structure For The Construction Of Non-Linear Components Of Block Cipher. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3005087.

chapter III