Kryptographie: Integrität und Authentizität

3

Inhaltsverzeichnis

Die Integrität von Daten kann nur mithilfe von Hashfunktionen, Message Authentication Codes und digitalen Signaturen geschützt werden. Diese kryptographischen Mechanismen werden in diesem Kapitel eingeführt, zusammen mit der Kombination aus Verschlüsselung und Message Authentication Codes, die Authenticated Encryption ergibt.

3.1 Hashfunktionen

Notation Hashfunktionen sind wichtige Bausteine jedes komplexeren Mechanismus zum Integritäts- oder Authentizitätsschutz. Der von einer *Hashfunktion* $\mathsf{H}()$ aus einem beliebig langen Datensatz m berechnete Hashwert h, der eine feste Bitlänge λ – in der Praxis heute 160, 256 oder 512 Bit – besitzt, ist eine *kryptographische* Prüfsumme:

$$h \leftarrow \mathsf{H}(m), h \in \{0,1\}^{\lambda}, m \in \{0,1\}^{*}$$

Im Gegensatz zu den *zufälligen* Änderungen, die in der Codierungstheorie durch fehlererkennende oder fehlerkorrigierende Prüfsummen abgefangen werden, muss ein Hashwert sich auch bei gezielten Manipulationen der Daten immer ändern.

J. Schwenk, *Sicherheit und Kryptographie im Internet,*
https://doi.org/10.1007/978-3-658-29260-7_3

Mental Model Ein geeignetes mentales Modell für Hashfunktionen sind Fingerabdrücke von Personen. Es sollte praktisch unmöglich sein, zwei Personen zu finden, die exakt den gleichen Fingerabdruck haben, und aus einem Fingerabdruck kann man nicht die Person selbst rekonstruieren. Allerdings ist es einfach, einen Fingerabdruck zu bestimmen, wenn eine Person anwesend ist.

3.1.1 Hashfunktionen in der Praxis

Standardisierte Hashfunktionen Gute Hashfunktion sind schwer zu konstruieren. Daher gibt es nur eine knappe Handvoll von Familien von Hashfunktionen, die in der Praxis eingesetzt werden: Der veraltete, aber immer noch eingesetzte *Message Digest 5* (MD5) von Ron Rivest [Riv92] aus dem Jahr 1992, die aktuell meistverwendete Hashfunktion *SHA-1* [rJ01] aus dem Jahr 1995, die unter dem Begriff *SHA-2* zusammengefasste Familie von Hashfunktionen SHA-224, SHA-256, SHA-384 und SHA-512, die 2001 standardisiert wurden, und die neue *SHA-3*-Familie SHA3-224, SHA3-256, SHA3-384 und SHA3-512 [rH11] aus dem Jahr 2015. Bei SHA-2 und SHA-3 gibt die Zahl nach dem Bindestrich jeweils die Länge der Ausgabe der jeweiligen Hashfunktion in Bits an.

Aufbau einer Hashfunktion Da Hashfunktionen Eingaben beliebiger Länge auf einen Hashwert fester Länge komprimieren müssen, sind sie iterativ aufgebaut. In Abb. 3.1 ist dies für die Parameter von SHA-256 dargestellt.

Zunächst wird die zu hashende Nachricht m in Blöcke m_i fester Länge aufgeteilt, ähnlich wie bei einer Verschlüsselung mit einer Blockchiffre. Die einzelnen Blöcke sind hier aber wesentlich länger, der Wert 512 Bit ist für viele Hashfunktionen typisch. Der letzte Block muss mit einem Padding auf diese Länge gebracht werden, und die Gesamtlänge der ursprünglichen Nachricht wird in diesem Padding mit codiert.

Jeder Block wird jetzt in einer Kompressionsfunktion $SHA-C$ komprimiert, zusammen mit einem internen Zustand s_i aus der Verarbeitung der vorherigen Blöcke. Da dieser Zustand

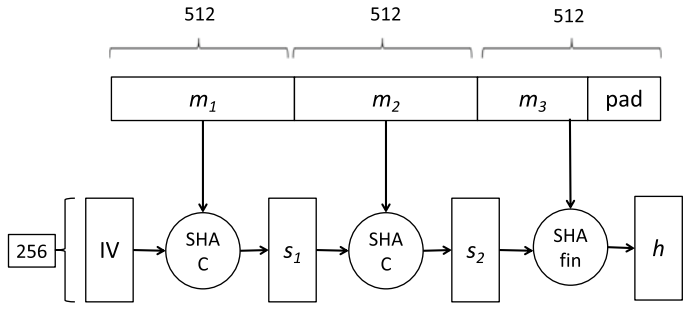


Abb. 3.1 Iterativer Aufbau einer Hashfunktion am Beispiel von SHA-256

für den ersten Block nicht zur Verfügung steht, wird er durch einen Initialisierungsvektor IV ersetzt, der aber hier konstant und im jeweiligen Standard festgelegt ist. Die Ausgabe h der letzten Kompressionsfunktion ist dann, ggf. nach einem zusätzlichen Bearbeitungsschritt, die Ausgabe der Hashfunktion.

3.1.2 Sicherheit von Hashfunktionen

Zu einem gegebenen Hashwert h existieren typischerweise viele Urbilder m, da die unendlich vielen Bitfolgen aus $\{0, 1\}^*$ von der Hashfunktion $\mathsf{H}()$ auf eine endliche Menge von Bitfolgen $\{0, 1\}^\lambda$ abgebildet werden. Die Sicherheitseigenschaften von Hashfunktionen, die im mentalen Modell angedeutet wurden, lassen sich formal wie folgt beschreiben:

- **One-Way Function:** Es ist praktisch unmöglich, aus einem gegebenen Hashwert h ein Urbild m' mit $\mathsf{H}(m') = h$ zu berechnen *(Einwegeigenschaft).*
- **Second Preimage Resistance:** Es ist praktisch unmöglich, zu einem gegebenen Datensatz m mit $\mathsf{H}(m) = h$ einen zweiten Datensatz m' mit $\mathsf{H}(m') = h$ zu berechnen *(schwache Kollisionsresistenz).*
- **Collision Resistance:** Es ist praktisch unmöglich, zwei Datensätze m und m' zu berechnen, die den gleichen Hashwert h besitzen *(starke Kollisionsresistenz).*

„Praktisch unmöglich" bedeutet dabei, dass es weder mit der heute noch mit der in der nahen Zukunft verfügbaren Rechenleistung möglich sein soll, dies in einem sinnvollen Zeitrahmen zu berechnen. Man kann diesen vagen Begriff in der Sprache der Komplexitätstheorie, eines Teilbereichs der theoretischen Informatik, formalisieren [BDG90a, BDG90b].

Angreifermodell Das Angreifermodell für Hashfunktionen besteht darin, dass der Angreifer einen „wertvollen" Hashwert kennt und diesen weiterverwenden möchte. Da der Angreifer selbst beliebig viele Hashwerte erzeugen kann, wird ein Hashwert erst dann „wertvoll", wenn eine weitere kryptographische Operation darauf aufbaut, z. B. eine digitale Signatur oder ein MAC.

Angriffe auf die starke Kollisionsresistenz Eine Hashfunktion gilt als gebrochen, wenn es möglich ist, die *collision resistance* zu brechen, d. h. zwei Urbilder x, x' mit gleichem Hashwert zu finden:

$$hash(x) = hash(x')$$

Dies ist z. B. für MD5 leicht möglich, und auch für SHA-1 wurde bereits eine Kollision gefunden. Dadurch hat er aber nach unserem Angreifermodell nur einen „wertlosen" Hashwert mit zwei Urbildern gefunden, also ist dieser Angriff nicht direkt kritisch. Aus einer gefundenen Kollision können unter bestimmten Umständen (das Padding des letzten Blocks muss passen) beliebig viele abhängige Kollisionen erzeugt werden, indem identische

Bytes an die beiden gefundenen Urbilder angefügt werden: Ist der Hashwert nach Abarbeitung der Kollision am Anfang einmal gleich, so bleibt er wegen der iterativen Struktur aller Hashfunktionen auch gleich, wenn danach auf beiden Seiten identische Werte „dazugehasht“ werden:

$$hash(x|y) = hash(x'|y)$$

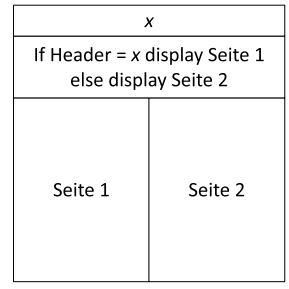
Auf einer solchen Hashkollision können aber weitere Angriffe aufbauen. Ein Klassiker ist hier der *If-Then-Else*-Angriff, bei dem z. B. zwei PDF-Seiten in einem PDF-Dokument gespeichert werden. Das Dokument beginnt mit einem der beiden Urbilder x und enthält ein kleines Programm das steuert, welche der beiden Seiten angezeigt wird (Abb. 3.2).

Beginnt das PDF-Dokument mit dem Wert x, so wird die harmlos aussehende S. 1 „Ich kaufe die Uhr für 10 Euro“ angezeigt. Das Opfer glaubt, den angezeigten harmlosen Text zu signieren, und erzeugt eine digitale Signatur über den Hashwert des gesamten PDF-Dokuments. Nun tauscht der Angreifer den Wert x im Dokument durch den Wert x' aus. Der Hashwert über das gesamte Dokument, und damit auch die digitale Signatur, bleibt dabei gleich, da x und x' ja den gleichen Hashwert haben und die nachfolgenden Bytes in beiden Versionen der Datei identisch sind. Der Angreifer kann nun ein gültig signiertes PDF-Dokument vorweisen, bei dem der Text der S. 2 „Ich kaufe die Uhr für 10.000 Euro“ angezeigt wird. Zum Glück für das Opfer kann der If-Then-Else-Angriff leicht durch eine Analyse des PDF-Dokuments erkannt werden.

Ein Angriff auf die *Second Preimage Resistance,* bei dem zu einem gegebenen Urbild x und einem „wertvollen“ Hashwert $hash(x)$ ein zweites Urbild x' berechnet wird, könnte dagegen direkt ausgenutzt werden, um z. B. eine gültige Signatur für x auf den Datensatz x' übertragen wird. Solche Angriffe sind jedoch bis jetzt für keine in der Praxis eingesetzten Hashfunktionen – einschließlich MD5 – bekannt. Das Gleiche gilt für Angriffe auf die *Einwegeigenschaft* von Hashfunktionen.

Chosen Prefix Collisions Für einen Angriff auf eine reale, Hash-basierte Sicherheitsanwendung reicht es nicht, irgendeine Kollision oder irgendein *Second Preimage* zu finden, da diese Urbilder der Hashfunktion in der Regel nur aus zufälligen Bits bestehen. Reale Angriffe sind deutlich komplexer.

Abb. 3.2 If-Then-Else-Angriff



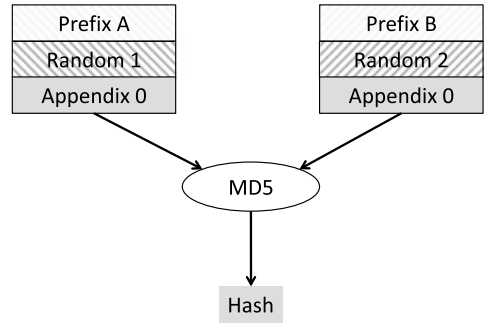


Abb. 3.3 Chosen-Prefix-Collision-Angriff auf MD5

Im Jahr 2007 zeigten Marc Stevens, Arjen K. Lenstra und Benne de Weger [SLdW07], wie man die Tatsache, dass MD5-Kollisionen leicht zu berechnen sind, für einen Angriff auf Public-Key-Infrastrukturen (Abschn. 4.6) ausnutzen kann. Dazu verbesserten sie zunächst die für MD5 bekannten Kollisionsangriffe zu dem in Abb. 3.3 beschriebenen *Chosen-Prefix-Collision*-Angriff. Bei diesem Angriff kann der Angreifer zwei Präfixe A und B beliebig vorgeben. Danach wird eine zufällige MD5-Kollision gesucht, sodass gilt:

$$\mathsf{MD5}(\mathsf{Prefix\ A}|\mathsf{Random\ 1}) = \mathsf{MD5}(\mathsf{Prefix\ B}|\mathsf{Random\ 2})$$

Diese Kollision bleibt natürlich bestehen, wenn an beide Datensätze der gleiche Appendix angehängt wird:

$$\mathsf{MD5}(\mathsf{Prefix\ A}|\mathsf{Random\ 1}|\mathsf{Appendix\ 0}) = \mathsf{MD5}(\mathsf{Prefix\ B}|\mathsf{Random\ 2}|\mathsf{Appendix\ 0})$$

Die Idee für die Fälschung eine X.509-Zertifikats bestand nun darin, dass der erste Datensatz Prefix A|Random 1|Appendix 0 „harmlos“ aussieht und daher ohne Probleme von einer Zertifizierungsstelle signiert wird. Der zweite Datensatz Prefix B|Random 2|Appendix 0 entspricht dann einem Zertifikatsinhalt, der nie signiert worden wäre – im Ergebnis des Feldversuchs aus [SLdW07] war dies ein Zertifikat, mit dem man beliebig viele weitere gültige Zertifikate hätte ausstellen können.

Sicherheitsempfehlungen Für Hashfunktionen verschiebt sich die Grenze dessen, was „praktisch möglich“ ist, natürlich ständig, sowohl durch Fortschritte in der Kryptoanalyse als auch durch die ständig wachsende Rechenleistung. Durch den Nachweis eklatanter Schwächen im MD4 durch Hans Dobbertin [Dob98] wurde der Kreis der einsetzbaren Hashfunktionen verkleinert. Auch MD5 weist deutliche Schwächen auf: Für MD5 kann man heute leicht zwei Urbilder mit dem gleichen Hashwert berechnen, er besitzt also nicht (mehr) die Eigenschaft der starken Kollisionsresistenz. Die schwache Kollisionsresistenz ist noch nicht gebrochen, aber irgendwo zwischen diesen beiden Eigenschaften ist der bislang stärkste bekannte Angriff auf MD5 anzusiedeln, der *Chosen-Prefix-Collision*-Angriff, mit dessen Hilfe der erste wirklich gravierende Angriff realisiert werden konnte, das Fälschen

eines X.509-Zertifikats [SSA+09]. Für SHA-1 ist bereits ein vereinzelter Angriff auf die starke Kollisionsresistenz bekannt (https://shattered.io), sodass man hier die weitere Entwicklung aufmerksam beobachten sollte. Für alle anderen Hashfunktionen sind bislang nur generische Angriffe bekannt, bei denen so lange zufällig gewählte Werte gehasht werden, bis man zufällig eine Kollision findet. Aufgrund des Geburtstagsparadoxons passiert das für eine Hashfunktion der Ausgabelänge $2n$ Bit schon nach 2^n Versuchen mit hoher Wahrscheinlichkeit. Daher sollte die Ausgabelänge jeder Hashfunktion heute 160 Bit nicht unterschreiten. MD5 muss als gebrochen bezeichnet werden und sollte aus allen Anwendungen entfernt werden. Die Sicherheit von SHA-1 wird angezweifelt, da eine einzelne Kollision gefunden wurde. Daher wird heute der Einsatz von SHA-224, SHA-256, SHA-386 oder SHA-512 empfohlen. Die neuen Hashfunktionen wurden im Rahmen eines SHA-3-Wettbewerbs ermittelt, in dem das amerikanische National Institute of Standards and Technology (NIST) am 02.10.2012 den Gewinner bekannt gegeben hat [oSN].

3.2 Message Authentication Codes und Pseudozufallsfunktionen

Ein *Message Authentication Code* (MAC) ist eine kryptographische Prüfsumme, in die neben dem Datensatz auch noch der geheime (symmetrische) Schlüssel von Sender und Empfänger einfließt. Ein MAC kann daher im Gegensatz zu einem Hashwert nur vom Sender oder Empfänger berechnet und auch nur von diesen beiden verifiziert werden. Mit einem MAC und einem symmetrischen Schlüssel k kann daher der Sender die Integrität einer Nachricht schützen, und genau ein Empfänger – derjenige, der ebenfalls den Schlüssel k kennt – kann die Integrität überprüfen.

Notation Ein Message Authentication Code mac zu einer Nachricht m wird mithilfe der MAC-Funktion $\mathsf{MAC}_{()}()$ und eines MAC-Schlüssels k berechnet. Die Nachricht m darf eine beliebige Länge haben, der MAC mac und der Schlüssel k haben eine feste Länge:

$$mac \leftarrow \mathsf{MAC}_k(m), m \in \{0,1\}^*, k \in \{0,1\}^\mu, mac \in \{0,1\}^\lambda$$

Mental Model In Ermangelung eines besseren mentalen Modells kann ein MAC als verschlüsselter Fingerabdruck einer Person imaginiert werden: Wenn von einer Person ein Fingerabdruck genommen wurde, so kann dieser nur dann überprüft werden, wenn der passende Schlüssel k vorliegt.

Standardkonstruktionen In der Literatur wurden verschiedene Arten der MAC-Berechnung vorgeschlagen, in der Praxis sind vor allen Dingen zwei Konstruktionen wichtig: die Berechnung eines MAC durch Iteration einer Blockchiffre (CBC-MAC [97911],

CMAC [SPLI06]) oder durch Anwendung einer Hashfunktion auf Schlüssel und Daten in einer festgelegten Reihenfolge (HMAC; RFC 2104 [KBC97]).

Die Berechnung eines CBC-MAC über eine Nachricht m mit einer Blockchiffre BC ist ähnlich zur CBC-Verschlüsselung von m und benötigt den gleichen Rechenaufwand (Abb. 3.4). Im Unterschied zur Verschlüsselung wird aber der Initialisierungsvektor auf den Wert NULL gesetzt, d. h., alle Bits haben den Wert 0. Der CBC-MAC mac ist dann der letzte Chiffretextblock, in Abb. 3.4 also c_3.

Die Sicherheit der HMAC-Konstruktion [KBC97] wurde kryptographisch nachgewiesen [Bel06]. Der Wert $\mathsf{HMAC\text{-}H}_k(m)$ – wobei H für eine bestimmte Hashfunktion steht, also z. B. HMAC-SHA1() oder HMAC-SHA256() – für den Datensatz m wird wie folgt berechnet (Abb. 3.5):

$$mac \leftarrow \mathsf{HMAC\text{-}H}_k(m) := \mathsf{H}(k|00...00 \oplus opad|\mathsf{H}(k|00...00 \oplus ipad|m))$$

Dabei wird zunächst der Schlüssel k durch Anfügen von Nullen am Ende auf die Input-Blocklänge der verwendeten Hashfunktion verlängert. Dann wird dieser Wert bitweise mit $ipad$ XOR-verknüpft, wobei $ipad$ aus hinreichend vielen Bytes `0x36` besteht. An das

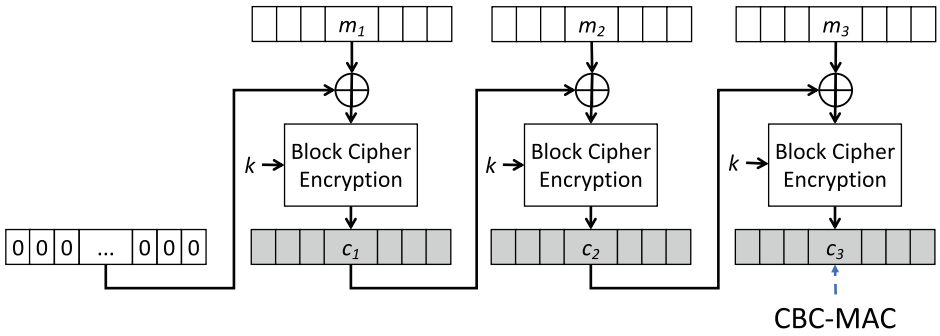


Abb. 3.4 CBC-MAC-Berechnung mithilfe einer Blockchiffre

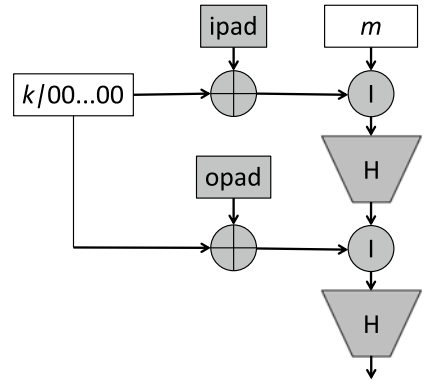


Abb. 3.5 Schematische Darstellung der HMAC-Berechnung

Ergebnis wird nun der Datensatz m angefügt und das Ganze wird gehasht. Anschließend wird der verlängerte Schlüssel bitweise XOR-verknüpft mit $opad$, einer hinreichend langen Wiederholung des Bytes `0x5C`. Das Ergebnis der ersten Anwendung der Hashfunktion wird angefügt, und beides zusammen ein zweites Mal gehasht. Das Ergebnis dieser zweiten Hashwertberechnung ist der gesuchte Message Authentication Code mac. Die HMAC-Konstruktion wird im Bereich der Internetsicherheit sehr häufig eingesetzt, z. B. auch zur Ableitung neuer Schlüssel.

Pseudozufallsfunktionen Die HMAC-Konstruktion wird auch zur Erzeugung von *Pseudozufallsfolgen* eingesetzt, also als *Pseudozufallsfunktion* (PRF). Dies ist z. B. in TLS (Abschn. 10.3.5) und in HKDF [KE10] der Fall. Eine PRF unterscheidet sich von einem MAC in zwei Aspekten:

- **Ausgabelänge:** Eine MAC-Funktion hat eine Ausgabe fester Länge, während eine PRF beliebig lange Pseudozufallsfolgen ausgeben kann.
- **Sicherheitsziel:** Von einem MAC fordert man, dass er nicht gefälscht werden kann, wenn der geheime MAC-Schlüssel unbekannt ist *(Computational Security)*. Von einer PRF verlangt man, dass ihre Ausgabe nicht von einem Zufallswert unterschieden werden kann, wenn der geheime PRF-Schlüssel zufällig gewählt wurde *(Decisional Security)*. Man kann also die HMAC-Konstruktion sowohl zur Berechnung eines MAC als auch zur Erzeugung von Pseudozufallswerten einsetzen, nur sind die kryptographischen Anforderungen an den HMAC im zweiten Fall höher.

Pseudozufallsfunktionen sind ein wichtiges theoretisches Konstrukt in der modernen Kryptographie; exakte Definitionen findet man z. B. in [KL14].

HKDF Die *Hashed Key Derivation Function* (HKDF) [KE10] wurde von Hugo Krawczyk [Kra10] analysiert. Die HKDF-Funktion erzeugt aus drei Eingaben und einer Längenangabe L eine Pseudozufallsfolge der Länge L:

$$K(1)|K(2)|K(3)|... \leftarrow HKDF(XTS, SKM, CTXInfo, L)$$

Die Eingabe besteht aus einem (öffentlichen) Salt XTS, dem geheimen Schlüsselmaterial SKM und einer Kontextinformation $CTXinfo$. Die Werte $K(i)$ werden mithilfe von HMAC wie folgt berechnet:

$$\begin{aligned} PRK &\leftarrow \mathsf{HMAC\text{-}H}_{XTS}(SKM) \\ K(1) &\leftarrow \mathsf{HMAC\text{-}H}_{PRK}(CTXinfo|0) \\ K(i+1) &\leftarrow \mathsf{HMAC\text{-}H}_{PRK}(K(i)|CTXinfo|i) \end{aligned}$$

Dabei werden so viele Blöcke $K(i)$ berechnet, bis die Längenangabe L überschritten wird. Die Berechnung von PRK wird oft auch als *HKDF-Extract* bezeichnet, die Berechnung der Blöcke $K(i)$ als *HKDF-Expand.*

3.3 Authenticated Encryption

Authenticated Encryption (AE) Die in Abschn. 12.3 beschriebenen Angriffe auf das MAC-then-PAD-then-Encrypt-Paradigma von SSL/TLS machen klar, dass durch fehlende Integrität des Chiffretextes sogar die Vertraulichkeit des Klartextes gefährdet sein kann. Daher wird der Begriff *Authenticated Encryption* heute nicht mehr auf jede Kombination von Verschlüsselung und MAC angewandt, sondern bezeichnet nur noch die Modi, in denen der Chiffretext durch den MAC geschützt wird. Dies kann durch Encrypt-then-MAC-Konstruktionen – auch für Stromchiffren – oder durch spezielle Blockchiffrenmodi wie den *Galois/Counter Mode* (GCM) [GCM07] erreicht werden.

Beim GCM wird zunächst die verwendete 128-Bit-Blockchiffre in eine Stromchiffre umgewandelt. Ein zufällig gewählter Initialisierungsvektor IV wird für jeden neuen zu verschlüsselnden Block inkrementiert und mit der Blockchiffre verschlüsselt. Dadurch wird ein Schlüsselstrom der Länge 128 Bit erzeugt, und der Plaintext wird durch bitweise XOR-Verknüpfung mit diesem Schlüsselstromblock in den Chiffretext umgewandelt. Diese Vorgehensweise war als *Counter Mode* (CTR) schon länger bekannt. Neu ist bei GCM, dass parallel hierzu ein MAC (Abschn. 3.2) berechnet wird, der die Nachteile der Stromchiffre kompensiert und die Integrität des Chiffretextes garantiert. Hierzu wird eine neu entwickelte Funktion auf Basis der Multiplikation im endlichen Körper $GF(2^{128})$ verwendet, was den GCM nur für Blockchiffren der Blocklänge 128 Bit nutzbar macht.

Authenticated Encryption with Additional Data (AEAD) Die MAC-Berechnung bei Authenticated-Encryption-Verfahren kann parallel zur Verschlüsselung erfolgen, ist aber in der Regel unabhängig davon. Daher spricht nichts dagegen, neben dem Chiffretext noch weitere Klartextdaten in diese MAC-Berechnung einfließen zu lassen. Ist dies der Fall, so spricht man von *Authenticated Encryption with Additional Data* (AEAD).

3.4 Digitale Signaturen

Um eine *digitale Signatur* eines Datensatzes zu erstellen, wird zunächst sein Hashwert gebildet. Dieser Hashwert wird dann mit dem privaten Schlüssel signiert. Überprüft werden kann die digitale Signatur dann von quasi jedem mithilfe des öffentlichen Schlüssels.

Notation Um eine digitale Signatur erstellen zu können, muss der Signierer ein Signaturschlüsselpaar (sk, pk) besitzen. Der öffentliche Schlüssel muss veröffentlicht werden – in der Praxis geschieht dies mit X.509-Zertifikaten, die auch eine Identität des Signierers enthalten. Eine Signatur sig über die Nachricht m wird mithilfe des privaten Schlüssels sk erzeugt:

$$sig \leftarrow \mathsf{Sign}(sk, m)$$

Die Signaturerzeugung kann eine deterministische oder probabilistische Funktion sein. Zur Überprüfung einer digitalen Signatur sind eine digitale Signatur sig, die Nachricht m und der öffentliche Schlüssel pk des Signierers erforderlich und ergibt einen Booleschen Wert:

$$TRUE/FALSE \leftarrow \mathsf{Vrfy}(pk, sig, m)$$

Mental Model Eine digitale Signatur weist Ähnlichkeiten mit der handschriftlichen Unterschrift auf. Hierzu gehört z. B. die Tatsache, dass eine Unterschrift genau wie eine digitale Signatur nur von einer einzigen Person erzeugt, aber von vielen verifiziert werden kann. Es gibt aber auch Unterschiede: Während die handschriftliche Unterschrift in der Regel immer ungefähr gleich aussieht und nur dadurch mit einem Dokument verknüpft wird, dass sie auf dem gleichen Blatt Papier steht, hängt der Wert der digitalen Unterschrift direkt vom Wert des zu signierenden Dokuments ab.

Ein besseres mentales Modell ist in Abb. 3.6 visualisiert: Ein gläserner Tresor beinhaltet das signierte Dokument. Nur eine Entität besitzt den privaten Schlüssel, um diesen Tresor zu öffnen und ein Dokument hineinzulegen, aber jeder kann das Dokument lesen und verifizieren, dass es sich tatsächlich in dem gläsernen Tresor befindet. In der Praxis wird ein ähnliches Verfahren zur Authentifizierung von öffentlichen Aushängen angewandt: Jede Kommune besitzt einen Aushangkasten, in dem öffentliche Bekanntmachungen publiziert werden. Nur das Bürgermeisteramt hat einen (privaten) Schlüssel zu diesem Kasten.

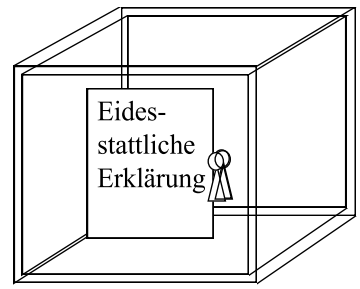


Abb. 3.6 Visualisierung der digitalen Signatur als gläserner Tresor. Nur der Besitzer des privaten Schlüssels kann eine Nachricht hineinlegen und damit signieren

Wichtige Algorithmen Wichtige Signaturalgorithmen sind RSA-PKCS#1, ElGamal und DSS/DSA, die im Folgenden eingeführt werden. Viele weitere Signaturverfahren sind in der kryptographischen Literatur beschrieben.

3.4.1 RSA-Signatur

Textbook RSA

Mithilfe des RSA-Verfahrens kann man auch leicht eine digitale Signatur realisieren. Das Signaturschlüsselpaar besteht, wie bei der RSA Verschlüsselung, aus einem öffentlichen Schlüssel $pk = (e, n)$ und einem privaten Schlüssel $sk = (d, n)$.

Nachrichten m, die kürzer sind als der Modulus n des öffentlichen Schlüssels, können direkt signiert werden:

$$sig \leftarrow \mathsf{Sign}((d, n), m) := m^d \bmod n$$

Für längere Nachrichten wird zunächst der Hashwert $h = \mathsf{H}(m)$ der zu signierenden Nachricht m gebildet und dann die Signatur analog berechnet:

$$sig \leftarrow \mathsf{Sign}((d, n), h) := h^d \bmod n$$

Dies ist der Standardfall.

Die Signatur wird überprüft, indem zunächst erneut der Hashwert des Dokuments gebildet und dann der Wert sig durch Potenzierung mit dem öffentlichen Schlüssel e „verschlüsselt“ wird. Die Signatur ist gültig, wenn diese beiden Werte übereinstimmen:

$$TRUE/FALSE \leftarrow \mathsf{Vrfy}((e, n), sig, h) := (h = sig^e \bmod n)$$

Der Beweis der Korrektheit dieser Vorgehensweise erfolgt analog zu Abb. 2.9, nur mit vertauschten Rollen von e und d.

An dieser Stelle sei direkt darauf hingewiesen, dass bei anderen Signaturverfahren das Signieren keineswegs als „Entschlüsseln mit dem privaten Schlüssel“ erklärt werden kann. Wegen der enormen Popularität von RSA ist dieses Missverständnis aber immer noch weit verbreitet.

RSA-PKCS#1

Um Angriffe wie die in Abschn. 3.5 zu verhindern, ist es sinnvoll, auch zu signierende Nachrichten vorher zu codieren. Auch hier wird häufig eine PKCS#1-Codierung eingesetzt, die sich in kleinen, aber wichtigen Details von der entsprechenden Codierung für Verschlüsselung unterscheidet (Abb. 3.7):

0x00	0x01	0xFF	0xFF	....	0xFF	0x00	h(message)

Abb. 3.7 PKCS#1-Padding vor dem Signieren einer Nachricht

- Das zweite Byte wird auf den Wert `0x01` gesetzt.
- Das Padding ist nicht mehr zufällig gewählt, sondern besteht aus Bytes mit dem konstanten Wert 255 (hexadezimal `0XFF`).

Die Sicherheit dieses Signaturverfahrens wurde in [JKM18] nachgewiesen.

3.4.2 ElGamal-Signatur

Im ElGamal-Signaturverfahren [Gam85] wird eine Nachricht nicht, wie beim RSA-Verfahren, durch Potenzierung mit dem privaten Exponenten unterschrieben, sondern durch eine deutlich komplexere Operation. Zur Erzeugung und Verifikation einer digitalen Signatur werden der gleiche private Schlüssel $sk = x$ und der gleiche öffentliche Schlüssel $pk = X = g^x \bmod p$ verwendet wie beim ElGamal-Verschlüsselungsverfahren. Die öffentlichen Systemparameter (p, g) sind ebenfalls identisch, wobei $g \in \mathbb{Z}_p^*$ ist.

Erzeugen einer digitalen Signatur Zur Erzeugung einer digitalen Signatur für eine Nachricht m geht ein Teilnehmer T dabei wie folgt vor: Zunächst bildet er $h(m)$, dann wählt er $r \in \mathbb{Z}_{p-1}^*$ zufällig und bildet

$$k \leftarrow g^r \bmod p.$$

Er berechnet $r^{-1} \pmod{p-1}$ mithilfe des erweiterten euklidischen Algorithmus und danach

$$s \leftarrow r^{-1}(h(m) - xk) \bmod (p-1).$$

Die digitale Unterschrift der Nachricht m besteht aus dem Paar (k, s), und es gilt

$$h(m) = xk + rs \bmod (p-1).$$

Die Länge einer Signatur beträgt $2|p|$ Bit.

Überprüfung der digitalen Signatur Der Empfänger der signierten Nachricht $(m, (k, s))$ kann die Unterschrift prüfen, indem er die beiden Werte $g^{h(m)} \bmod p$ und $y^k \cdot k^s \bmod p$ bildet und vergleicht, ob diese Zahlen identisch sind. Bei einer gültigen Signatur funktioniert das, weil

$$y^k \cdot k^s = g^{xk} \cdot g^{rs} = g^{xk+rs} = g^{h(m)} \pmod{p}$$

gilt. Zum Verständnis des letzten Schrittes muss man sich in Erinnerung rufen, dass eine Reduktion der Basis modulo p im Exponenten einer Reduktion modulo $p-1$ entspricht.

Die Idee bei diesem Signaturverfahren besteht darin, dass nur derjenige die Zahl s berechnen kann, der den privaten Schlüssel x kennt. Um die Sicherheit dieses privaten Schlüssels auch bei mehrmaliger Verwendung zu garantieren, muss zusätzlich noch eine Zufallszahl r, die jedes Mal verschieden sein muss, in die Berechnung von s mit einfließen. So wird verhindert, dass aus zwei unterschiedlichen Signaturwerten s und s' der private Schlüssel x berechnet werden kann.

3.4.3 DSS und DSA

Der *Digital Signature Standard* (DSS) [Gal13] enthält vier verschiedene Signaturverfahren: Zwei RSA-Varianten – RSA-PKCS#1 v1.5 (Abschn. 2.4.2) und RSA-PSS [BR96b] – und zwei ElGamal-Varianten – DSA und dessen Variante auf elliptischen Kurven ECDSA.

Der *Digital Signature Algorithm* (DSA) ist eine besonders effiziente Variante des ElGamal-Signaturverfahrens, die auf eine Idee von Claus Schnorr [Sch90] zurückgeht. Dahinter steckt die Beobachtung, dass für ein fest gewähltes Element $g \in \mathbb{Z}_p^*$ alle Berechnungen nur in einer deutlich kleineren Untergruppe $G =< g >$ der Ordnung q stattfinden. Daraus folgt, dass die im ElGamal-Signaturverfahren übertragenen Signaturwerte k und s viel zu groß sind und durch kleinere Werte ersetzt werden können, was die Signatur kürzer und das Verifizieren von Signaturen deutlich effizienter macht.

Öffentlicher Schlüssel Der öffentliche Schlüssel bei DSA besteht aus den vier Werten g, p, q, X; neu ist hier der Wert q:

- p ist eine große Primzahl, $|p| \geq 1024$.
- q ist eine weitere Primzahl $|q| = 160$, mit der zusätzlichen Eigenschaft, dass sie ein Teiler von $p - 1$ ist.
- g ist ein Element der Ordnung q in $\mathbb{Z}_p^*$, d. h., es gilt $g^q = 1 \pmod p$ (und $g^c \neq 1 \pmod p$ für alle $c \leq q$).
- Der private Schlüssel x ist eine zufällig gewählte Zahl aus $\mathbb{Z}_q$, und $X \leftarrow g^x \bmod p$ ist der öffentliche Schlüssel.

Beim DSA spielen also zwei Primzahlen eine Rolle: eine „mittelgroße“ Primzahl $q \approx 2^{160}$ und eine „große“ Primzahl $p \geq 2^{1024}$. Die *Verifikation* der digitalen Signatur erfolgt in der multiplikativen Gruppe $\mathbb{Z}_p^*$, die $p - 1$ Elemente enthält. Das ausgewählte Element g erzeugt darin eine Untergruppe, die genau q Elemente enthält. Daher können alle Berechnungen zur *Erzeugung* einer digitalen Signatur ausschließlich modulo q durchgeführt werden, und dies ist letztendlich der Grund für die gegenüber RSA oder ElGamal deutlich kleineren Signaturen. Die Länge von q ist nicht zufällig gewählt, sondern sie entspricht genau der Länge der Ausgabe eines anderen Standards, nämlich der Hashfunktion SHA-1.

Erzeugung einer Signatur Die Erzeugung der Signatur ist ähnlich zum ElGamal-Signaturverfahren, nur werden alle Berechnungen modulo q durchgeführt, und eine Subtraktion wird durch eine Addition ersetzt:

1. Wähle eine geheime, zufällige Zahl $r \in \mathbb{Z}_q$.
2. Berechne $k \leftarrow (g^r \bmod p) \bmod q$.
3. Berechne $r^{-1} \pmod q$ mit dem erweiterten euklidischen Algorithmus.
4. Berechne $s \leftarrow r^{-1}(h(m) + x \cdot k) \bmod q$.
5. Die Signatur der Nachricht m ist das Paar (k, s). Die Länge dieser Signatur beträgt $2|q| \approx 320$ Bit, sie ist also wesentlich kürzer als eine RSA- oder ElGamal-Signatur.

Überprüfung einer Signatur Bei der Überprüfung einer Signatur muss auf die Reihenfolge der Reduktionen modulo p oder q geachtet werden:

1. Überprüfe, ob $0 \leq k < q$ und $0 \leq s < q$; wenn nicht, ist die Signatur ungültig.
2. Berechne $w := s^{-1} \pmod q$ und $h(m)$.
3. Berechne $u_1 \leftarrow w \cdot h(m) \bmod q$ und $u_2 \leftarrow k \cdot w \bmod q$.
4. Berechne $v \leftarrow (g^{u_1} y^{u_2} \bmod p) \bmod q$.
5. Die Signatur ist genau dann gültig, wenn $v = k$.

Für alle hier aufgeführten Berechnungen gibt es effiziente Algorithmen (z. B. [MvOV96]). Der große Vorteil des DSA liegt in der Kürze der erzeugten Signaturen. Das Paar (k, s) ist, bei einem beliebig vergrößerbaren Sicherheitsparameter p, der nur in den öffentlichen Schlüssel einfließt, nur 320 Bit groß. Zum Vergleich: Bei einem 1024-Bit-Modulus wären RSA-Signaturen 1024 Bit und ElGamal-Signaturen 2048 Bit lang.

3.5 Sicherheitsziel Integrität

Die Integrität von Nachrichten wird durch Message Authentication Codes oder durch digitale Signaturen geschützt. Mit diesen Verfahren kann sichergestellt werden, dass eine Nachricht, die beim Empfänger ankommt, auf ihrem Weg dorthin nicht verändert wurde. Gleichzeitig wird auch der Sender der Nachricht authentifiziert, da in beiden Fällen außer dem Empfänger nur eine einzige Partei als Sender infrage kommt.

Angreifermodelle In der Praxis stehen einem Angreifer viele Paare (m, sig) zur Verfügung, wobei m eine Nachricht ist und sig ein MAC oder eine digitale Signatur. In den theoretischen Modellen darf der Angreifer sogar mehrere Nachrichten m wählen und zu diesen eine digitale Signatur oder einen MAC berechnen lassen.

Die wichtigste Sicherheitseigenschaft für digitale Signaturen und MACs ist *Existential Unforgeability under Chosen Message Attacks* (EUF-CMA). Diese Eigenschaft besagt, dass

ein Angreifer für eine beliebige Nachricht m^*, zu der er keine Signatur angefordert hat, auch keine berechnen kann. Umgekehrt formuliert: Gelingt es einem Angreifer, eine gültige Signatur s^* zu einer neuen Nachricht zu berechnen, so hat er die Sicherheitseigenschaft EUF-CMA gebrochen, und das betrachtete Signatur- bzw. MAC-Verfahren gilt als unsicher.

Angriffe Das klassische Beispiel für einen Angriff zur Erzeugung eines neuen Paares (m^*, sig^*) ist die Textbook RSA-Signatur. Hier kombiniert der Angreifer einfach zwei ihm bekannte Signaturen (m_1, s_1) und (m_2, s_2) wie folgt:

$$m^* \leftarrow m_1 \cdot m_2$$

$$sig^* \leftarrow s_1 \cdot s_2 \bmod n = (m_1^d \bmod n) \cdot (m_2^d \bmod n) = (m_1 \cdot m_2)^d \bmod n$$

Dieser Angriff ist einer der Gründe, warum Nachrichten vor Erstellung einer digitalen Signatur mittels PKCS#1 oder OAEP codiert werden sollten.

Ein zweiter klassischer Angriff ist der auf eine einfache MAC-Konstruktion, die wie folgt definiert ist:

$$MAC(k, m) := hash(k|m)$$

Bei dieser MAC-Konstruktion kann ebenfalls die EUF-CMA-Eigenschaft gebrochen werden, indem die Nachricht einfach verlängert wird. Ist ein Paar (m, mac) bekannt, so kann wegen des iterativen Aufbaus von Hashfunktionen leicht ein MAC zur Nachricht $m^* = m|m'$ berechnet werden:

$$mac^* \leftarrow hash(mac|m') = hash(k|m|m') = MAC(k, m|m')$$

Damit dieser Angriff in der Praxis funktioniert, müssen noch ein paar Details beachtet werden. Zum Beispiel wird eine Nachricht vor dem Hashen auf ein Vielfaches der Blocklänge der Hashfunktion (für SHA-256 sind das 512 Bit) gepaddet. Die Länge der Nachricht ohne dieses Padding wird in die letzten Bytes dieses Padding geschrieben. Dieses Padding wird normalerweise in der letzten Runde der Hashfunktion automatisch hinzugefügt – bei dem beschriebenen Angriff muss dieser Wert aber als Zwischenzustand in die Hashfunktion eingeschleust werden. Ein praktischer Angriff auf den Zugriffsschutz von Flickr wurde 2009 von Duong und Rizzo [DR09] beschrieben.

3.6 Sicherheitsziel Vertraulichkeit und Integrität

Eines der großen Missverständnisse in der Kryptographie ist die Annahme, dass man durch Verschlüsselung die Integrität einer Nachricht sicherstellen könnte, nach dem Motto „Wenn der Angreifer die Nachricht nicht kennt, kann er sie auch nicht ändern“. Seit der Jahrtausendwende wurden aber immer mehr auch praktisch relevante Angriffe beschrieben, die zeigen, dass dies nicht stimmt; als Beispiele seien hier nur die ersten Angriffe auf die WLAN-

Verschlüsselung WEP (Abschn. 6.3.3, [BGW01]) und die verschiedenen Padding-Oracle-Angriffe auf den TLS Record Layer (Abschn. 12.3) genannt. Die letztgenannte Angriffsklasse hat darüber hinaus eindrucksvoll gezeigt, dass eine fehlende Integrität sogar die Vertraulichkeit von Verschlüsselung unterminieren kann.

Als Reaktion auf diese Angriffe haben Verschlüsselungsmodi wie *Galois/Counter Mode* (GCM), die Verschlüsselung und Integritätsschutz kombinieren, in den vergangenen Jahren erheblich an Bedeutung gewonnen. Diese *authentische Verschlüsselung* wurde in [BN00, BN08] grundlegend untersucht, und verschiedene Sicherheitsziele wurden explizit formuliert. Zwei dieser Sicherheitsziele sollen hier kurz vorgestellt werden.

INT-PTXT Dieses Kürzel steht für *Integrity of Plaintext* – ein Angreifer kann hier möglicherweise den Chiffretext ändern, aber jede Änderung am Klartext würde erkannt. Wichtigstes Beispiel für Verschlüsselungsverfahren mit dieser Sicherheitseigenschaft ist das MAC-then-PAD-then-ENCRYPT-Verfahren des Record Layer von SSL 3.0, bei dem der MAC über den Klartext gebildet wird und es daher z. B. möglich ist, das verschlüsselte Padding auszutauschen und damit den Chiffretext zu verändern. Dies wurde z. B. beim POODLE-Angriff (Abschn. 12.3.3) ausgenutzt.

INT-CTXT Empfohlen wird heute die ausschließliche Verwendung von Modi wie GCM, die die Integrität des Chiffretextes (INT-CTXT) garantieren, da nur diese unter Berücksichtigung der neuesten Angriffe als sicher gelten können.

Diese Sicherheitsziele können durch verschiedene Kombinationen von Verschlüsselung und MAC-Berechnung erreicht werden [BN00, BN08]:

- **MAC-then-Encrypt:** Hier wird zunächst über den Klartext ein MAC berechnet, und anschließend werden Klartext und MAC verschlüsselt.
- **Encrypt-and-MAC:** Hier wird der MAC ebenfalls über den Klartext gebildet, es wird aber nur der Klartext verschlüsselt.
- **Encrypt-then-MAC:** Hier wird der Klartext zunächst verschlüsselt und der MAC dann über den Chiffretext berechnet.