Proseminar Kryptographie und Datensicherheit
Universität Potsdam, Wintersemester 2004/2005

Message Authentication Codes

Martin Schütte

3. Februar 2005

Dieser Text gibt einen Überblick über die wichtigsten Konzepte von Message Authentication Codes (MACs). Grundlegende Konstruktionsprinzipien und die gebräuchlichsten Verfahren werden vorgestellt und bewertet. Desweiteren gibt es einen Ausblick auf die aktuelle Entwicklung bedingungslos sicherer MACs.

1 Einleitung

1.1 Wozu Authentizität?

Authentizität bezeichnet die „Echtheit“ einer Nachricht. Dazu gehört, dass die Nachricht während der Übertragung nicht verändert wurde und dass die Nachricht auch wirklich vom erwarteten Sender stammt.

Die erste Bedingung, auch als Integrität bezeichnet, ist als Problem recht einleuchtend. Weil kein Übertragungskanal perfekt und ohne Rauschen arbeitet, gehören einfache Fehlererkennungscodes bei jeder Beschäftigung mit Datenübertragung zum Handwerkszeug. Der Schritt von Fehlererkennungscodes zu Hashfunktionen, mit denen auch gezielte Änderungen verhindert werden können, lag dann nahe.

Die zweite Bedingung, die Identität des Senders, wird nicht immer als Problem erkannt, sondern oft implizit vorausgesetzt. Bei elektronischer

Kommunikation ist dieses Problem aber durchaus schwierig, weil der Übertragungskanal generell als unsicher und alle darüber empfangenen Nachrichten als potentiell gefälscht anzusehen sind. Die Verbreitung symmetrischer Kryptographie erhöht hier auch nicht gerade das Problembewusstsein. Dass eine verschlüsselte Nachricht sich nur mit dem richtigen Schlüssel entschlüsseln lässt, führt oft zum Fehlschluss, dass jede Nachricht, die mit diesem Schlüssel entschlüsselt wird, auch authentisch ist. Dies stimmt vor allem in „Spezialfällen“, wenn die Nachricht sehr redundant ist (z. B. natürliche Sprache) und Änderungen dadurch leicht erkannt werden können. Im Allgemeinen darf man aber nicht davon ausgehen, dass verschlüsselte Nachrichten automatisch authentisch sind [Menezes97, S. 364f],[Goldwasser01, S. 138f].

Message Authentication Codes (MACs) sind Verfahren, die diese Authentizität von Nachrichten auch über unsichere Verbindungen garantieren können.

1.2 Ziel und Aufbau dieser Arbeit

Eine detaillierte Betrachtung der einzelnen MAC-Verfahren einschließlich ihrer Sicherheit und Implementation würde leicht Stoff für ein ganzes Buch bieten (oder eine Doktorarbeit wie [Black00] und [Rompay04], die beide eine gute Übersicht über das Gebiet beinhalten). Daher stelle ich in dieser Ausarbeitung nur die wichtigsten in der Praxis verbreiteten Verfahren und die zugrundeliegenden Konzepte vor. Kenntnisse über Verschlüsselungs- und Hash-Funktionen werden vorausgesetzt; Details der Implementierung bleiben weitestgehend unberücksichtigt.

Im Abschnitt 2 werde ich als theoretische Grundlage eine allgemeine Definition für Message Authentication Codes und Angriffe darauf angeben. Danach stelle ich in Abschnitt 3 die heute gebräuchlichsten Arten von MACs mit ihren Vor- und Nachteilen vor. Zuletzt skizziere ich in Abschnitt 4 noch die Grundlagen bedingungslos sicherer MACs.

2 Definitionen

2.1 Message Authentication Codes

Ein *Message Authentiation Code (MAC)* bezeichnet ein Verfahren mit dem Nachrichten authentifiziert werden können. Voraussetzung ist immer ein geheimer Schlüssel, den nur Sender und Empfänger der Nachricht kennen.

Der Sender kann mithilfe eines MAC-Verfahrens aus einer Nachricht x und dem Schlüssel k ein Prüfwert $y = \mathrm{MAC}(x, k)$ berechnen – dieser Wert y wird oft selbst MAC genannt; zur sprachlichen Klarheit benutze ich in diesem Text die Bezeichnung *Tag*. Er sendet dann das Paar (x, y) aus Nachricht und Tag an den Empfänger. Der Empfänger hat seinerseits ein Verfahren, mit dem er feststellen kann, ob die Nachricht authentisch ist. Üblicherweise berechnet er dazu aus der empfangenen Nachricht x' den Tag $y' = \mathrm{MAC}(x', k)$. Wenn $y = y'$, dann ist zum einen davon auszugehen, dass der Sender den geheimen Schlüssel k benutzt hat, und zum anderen dass $x = x'$, die Nachricht also authentisch ist.

Die Berechnung der Tags betrachtet man normalerweise als eine Funktion der Nachricht x und nimmt den Schlüssel k als festen Parameter an. Anstatt $\mathrm{MAC}(x, k)$ wird also von nun an $\mathrm{MAC}_k(x)$ als Bezeichnung einer *MAC-Familie* aus MAC-Funktionen über k notiert.

Die formale Definition von MAC-Familien erfolgt hier nach [Stinson02, S. 118] durch ein Viertupel $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$. Dieses enthält:

- $\mathcal{X}$, die Menge der möglichen Nachrichten,
- $\mathcal{Y}$, die Menge der möglichen Tags,
- $\mathcal{K}$, die Menge der Schlüssel, und
- $\mathcal{H}$, die Menge der Funktionen $h : K \times X \to Y$

Mit $|\mathcal{K}| = 1$ lassen sich auch Hash-Funktionen als Spezialfälle von MAC-Verfahren betrachten. Demzufolge sind in diesem Zusammenhang Hash-Familien und MAC-Familien gleichbedeutend.

Falls die einzelnen Mengen nicht genauer angegeben sind, ist davon auszugehen, dass $\mathcal{X}$ die Menge aller Zeichenketten ist, die Vielfaches der benutzten Blocklänge sind (Σ^{nb}), während $\mathcal{Y}$ und $\mathcal{K}$ jeweils Worte einer bestimmten – vom benutzten Verfahren und gegebenenfalls auch der Implementation abhängigen – Länge sind (Σ^{l}).

2.2 Angriffsmodell

Nachdem ein MAC nun als Verfahren zur Erzeugung und Prüfung eines Tags definiert wurde, muss noch geklärt werden was genau geschützt werden soll und was einen erfolgreichen Angriff ausmacht.

Nach [Preneel97] gibt es vier mögliche Arten einen MAC zu „brechen“:

existenzielle Fälschung: der Angreifer kann einen gültigen MAC zu mindestens einer Nachricht erzeugen

selektive Fälschung: der Angreifer kann den MAC einer a priori gewählten Nachricht erzeugen

universelle Fälschung: der Angreifer hat einen Algorithmus, der funktional äquivalent zum Verifikationsalgorithmus ist – damit kann er (genug Rechenleistung vorausgesetzt) durch Probieren MACs zu beliebigen Nachrichten erzeugen

totales Brechen des Systems: der Angreifer gelangt in Besitz des geheimen Schlüssels

Es wird generell vom *worst-case* ausgegangen: Für einen erfolgreichen Angriff genügt eine existenzielle Fälschung. Andererseits bekommt der Angreifer maximalen Zugriff auf den Kommunikationskanal – er soll legitime Nachrichten nicht nur abhören, sondern auch ändern können und außerdem die Möglichkeit haben sich Texte auszuwählen und vom Opfer authentifizieren zu lassen.

Theoretisch modelliert wird dieser Zugriff mit einem Orakel, das in Besitz des geheimen Schlüssels ist und zu beliebigen Nachrichten den zugehörigen Tag ausgibt. Ein Angriff läuft dann folgendermaßen ab:
Der Angreifer sendet q Nachrichten $x_1, x_2, \ldots, x_q$ an das Orakel und bekommt so die gültigen Nachricht/Tag-Paare $(x_1, y_1), (x_2, y_2), \ldots, (x_q, y_q)$.
Letztendlich muss der Angreifer ein neues Paar (x, y) erzeugen, das noch nicht vorkam (also $x \notin \{x_1, x_2, \ldots, x_q\}$).

In den meisten Fällen hat das ausgegebene Paar nur eine bestimmte Wahrscheinlichkeit ϵ mit der es ein gültiges Paar – d. h. eine *Fälschung* – ist. Weil q und ϵ die wichtigsten Größen sind, werden alle Angriffe durch (ϵ, q) klassifiziert.

Es liegt auf der Hand, dass Angriffe um so gefährlicher sind, je kleiner der Wert q und je größer die Erfolgswahrscheinlichkeit ϵ ist. Für besonders sichere Systeme kann man festlegen, dass Schlüssel nur eimal verwendet werden dürfen. Dadurch hat ein Angreifer nur noch die Möglichkeit ein Nachricht/Tag-Paar ohne Vorwissen zu erzeugen oder ein gültiges Paar abzufragen (bzw. real im Übertragungskanal abzufangen) und mithilfe dessen selbst ein neues Paar zu finden. Mit dieser Einschränkung bleiben nur $(\epsilon, 0)$- und $(\epsilon, 1)$-Angriffe übrig.

3 Konstruktion von MACs

3.1 Allgemeine Erwägungen

Einige Aspekte gelten für alle bisher vorgestellten Konstruktionen und sollen daher vorab erwähnt werden.

Ein „Replay“, also das Wiederholen von Nachrichten der Kommunikationspartner, ist im Angriffsmodell nicht berücksichtigt. Die hier vorgestellten Konstruktionen arbeiten alle deterministisch mit einzelnen Nachrichten; also ohne den Status von Sender oder Empfänger zu ändern. In der Praxis sollte nach Möglichkeit ein Zähler oder ähnliches in die Nutzdaten integriert werden um das einfache Wiederholen von Nachrichten als Angriff zu erkennen.

Alle hier vorgestellten Methoden arbeiten iterativ. Das heißt die Eingabe wird in Blöcke zerlegt, die nacheinander verarbeitet werden. Zwischen den Blöcken wird der Zustand des Algorithmus durch eine *Kettenvariable* bestimmt. Üblicherweise wird diese Variable nach dem letzten Block auch als Funktionswert ausgegeben. Dementsprechend hat die Länge dieser Kettenvariable großen Einfluss – sie ist zugleich die optimale Schlüssellänge und das Ziel von Kollisionsangriffen.

Bei „guten“ MAC Algorithmen – soll heißen bei Algorithmen ohne grobe Konstruktionsfehler – ist der Geburtstagsangriff die beste bekannte Angriffsmethode. Wenn die benutzte Kettenvariable eine Länge von l Bits hat, so werden für einen erfolgsversprechenden Geburtstagsangriff zum Erzeugen einer internen Kollision ungefähr $2^{l/2}$ Nachricht/Tag-Paare benötigt (es handelt sich also um einen $(1/2, 2^{l/2})$-Angriff).

Es besteht immer die Möglichkeit die Ausgabe der MAC-Funktionen zu kürzen und zum Beispiel von einer 160 Bit-langen SHA-1-Ausgabe nur die

ersten 128 Bit zu verwenden. Dies erhöht oft die Sicherheit, weil ein Angreifer dadurch weniger Rückschlüsse auf die Eingabe- oder internen Zwischenwerte ziehen kann. Andererseits darf die Länge des Tags natürlich nicht so kurz werden, dass kombinatorische Angriffe wie der Geburtstagsangriff oder gar einfaches Raten praktikabel werden. [Krawczyk97] empfiehlt mindestens die Hälfte des Funktionswertes, in jedem Fall aber 80 Bit als Tag zu benutzen.

Eine sehr praktische Eigenschaft von MACs ist die Tatsache, dass Angriffe immer nur in einem relativ kurzen Zeitfenster erfolgen können – nämlich solange die Kommunikationspartner ihren aktuellen Schlüssel benutzen. Bei einem Angriff auf Hash-Funktionen kann man lange Zeit Kollisionen suchen bevor man entsprechende Daten aufbereitet und zum Beispiel jemandem zur digitalen Unterschrift schickt. Bei einer wichtigen verschlüsselten Nachricht kann es auch Jahre nach dem Versand noch lohnenswert sein die Verschlüsselung zu brechen und die Nachricht zu lesen.
Für MACs gilt dies nicht: Erst nachdem eine Kommunikation begonnen hat, kann ein Angreifer sie abhören, um Hinweise auf den benutzten Schlüssel zu bekommen. Und um eine Fälschung zu generieren hat er gerade einmal solange Zeit, bis die Kommunikationspartner den Schlüssel wechseln – danach ist jede Fälschung wertlos.

Hinzu kommt die beschränkte Bandbreite eines Netzwerks: Selbst wenn ein Opfer für einen *chosen plaintext* Angriff gefunden wird, so muss man ihm die Nachrichten zukommen und sich zumindest die Tags zurückschicken lassen. Würde man eine 1 GBit/sec-Leitung nur für das Austauschen von Nachrichten und Tags verwenden (ohne Berücksichtigung von Protokoll-Overhead und verbrauchter Rechenzeit), so würde es immer noch Jahrtausende dauern, bis genug Daten für einen aussichtsreichen Geburtstagsangriff gesammelt wären[1].

[1] Minimale Nachrichtenlänge ist ein Block, der sei 512 Bit lang. Die Kapazität der Leitung ist dann 2^{30} Bit/512 Bit = 2^{21} Nachrichten pro Sekunde, das entspricht ungefähr 2^{46} Nachrichten pro Jahr.
Damit könnte man eine Kollision in einem 92-Bit langen MAC finden (mit der Wahrscheinlichkeit 1/2). Um die gleiche Erfolgsaussicht bei einem heute üblichen 128-Bit langen MAC zu haben, müsste man allerdings 2^{64} Nachrichten/Tag-Paare erzeugen. Das dauert dann $2^{18} \approx 250\,000$ Jahre.

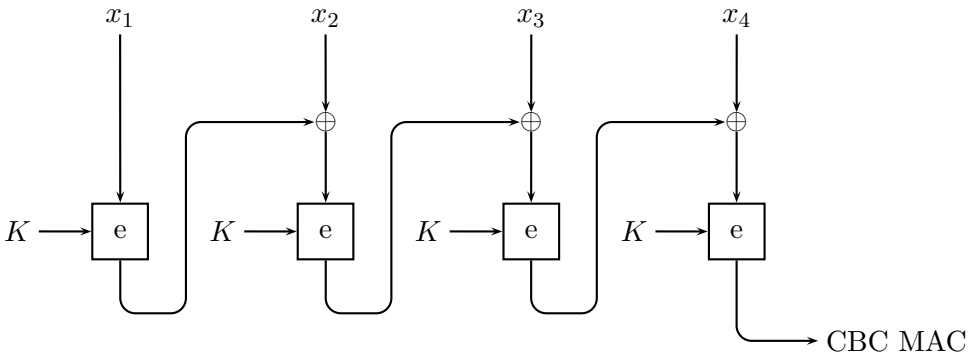

Abbildung 1: CBC MAC, mit einer 4 Block-langen Nachricht

3.2 CBC MACs

Aus allen Algorithmen zur symmetrischen Blockverschlüsselung lassen sich MACs konstruieren. Dazu wird der *Cipher Block Chaining* (CBC) Modus benutzt, bei dem jeder Eingabeblock $x_i (1 \leq i \leq m)$ erst mit dem Schlüsseltext des vorhergehenden Blocks y_{i-1} (mittels der XOR-Operation) verknüpft und dann verschlüsselt wird.

Beim CBC MAC benutzt man den letzten der verschlüsselten Blöcke y_m als Tag, also:

$$\text{CBC MAC}_k(x) = e_k(e_k(\ldots e_k(e_k(x_1) \oplus x_2) \oplus \ldots \oplus x_{m-1}) \oplus x_m)$$

Zur Sicherheit ist auf jeden Fall festzuhalten, dass CBC MACs nur für Nachrichten fester Länge benutzt werden sollten. Wird dies beachtet, dann ist das Verfahren sehr sicher; es wurde sogar gezeigt dass sich die Sicherheitseigenschaft der Pseudozufälligkeit (das heißt die Funktionswerte lassen sich nicht von zufälligen Ausgaben unterscheiden) von der Verschlüsselung auf den CBC MAC übertragen [Bellare00].

Für Nachrichten unterschiedlicher Länge lassen sich dagegen leicht existentielle Fälschungen herstellen. Ein Beispiel: Die Nachricht m sei einen Block lang und es sei $y = \text{MAC}_k(m) = e_k(m)$. Dann ist:
$\text{MAC}_k(m||m \oplus y) = e_k(e_k(m) \oplus (m \oplus y)) = e_k(y \oplus m \oplus y) = e_k(m) = y.$

Ein praktisches Ausnutzen dieser Schwäche dürfte sehr schwierig sein. Da es aber erweiterte Konstruktionen gibt, die CBC MAC mit nur we-

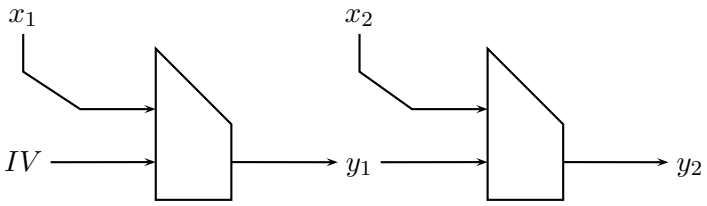

Abbildung 2: Schema einer iterierten Hashfunktion mit zwei Eingabeblöcken x_1, x_2 und den Kettenvariablen y_1, y_2

nig Mehraufwand beweisbar sicher machen (vgl. zum Beispiel [Black00, Kapitel 6]), gibt es keinen Grund einen reinen CBC MAC bei variablen Nachrichtenlängen einzusetzen.

Jede Blockverschlüsselung lässt sich für die Konstruktion von CBC MACs benutzen; je nach Algorithmus spricht man dann von DES-CBC MAC, AES-CBC MAC usw.. Genereller Nachteil dieser Methode ist der vergleichsweise hohe Bedarf an Rechenleistung (er ist genauso groß wie für die Verschlüsselung der Nachricht).

3.3 Einfache hash-basierte MACs

Anstatt eine Verschlüsselungsfunktion so anzupassen, dass die Ausgabe komprimiert wird, kann man das Problem auch von der anderen Seite angehen und eine Hash-Funktion um einen Schlüssel erweitern.

Ausgangspunkt ist im Folgenden immer eine iterierte Hash-Funktion F, die die Anforderungen der Urbild-, 2. Urbild- und Kollisions-Resistenz erfüllt. F arbeitet außerdem blockweise mit einer Kompressionsfunktion, die jeweils einen Eingabeblock x_i zusammen mit einer internen Kettenvariable y_{i-1} zur Kettenvariable der nächsten Runde y_i komprimiert; beim ersten Block wir die Kettenvariable auf einen Initialwert IV gesetzt, nachdem alle Blöcke durchlaufen sind ist die letzte Kettenvariable der Funktionswert: $F(x) = y_n$.

Die Beschränkung auf iterierte Hash-Funktionen ist deshalb sinnvoll, weil die verbreitetesten Hashfunktionen auf diese Weise arbeiten und sich daraus wichtige Sicherheitseigenschaften ergeben [Bellare96].

3.3.1 Modifikation einer Hash-Funktion

Eine bestehende Hash-Funktion kann so modifiziert werden, dass in den einzelnen Runden Teile des Schlüssels in die Berechnung einfließen. (So eine Modifikation für MD4-basierte Hash-Funktionen findet sich in [Preneel95].)

Dieses Vorgehen ist eher unüblich, weil eine Änderung ohne Beeinträchtigung der Sicherheit der Hash-Funktion schwierig ist. Außerdem ist damit nur eine bestimmte Funktion festgelegt, während alle folgenden Verfahren für beliebige Hash-Funktionen definiert sind. Allgemeine Verfahren sind auch deshalb vorzuziehen, weil es im Falle von Sicherheitsprobleme einfacher ist später nur die Hash-Funktion auszutauschen.

3.3.2 Initialwert oder Geheimes Präfix

Eine sehr einfache Möglichkeit einen Schlüssel in eine Hash-Funktion zu integrieren ist, die Funktion mit dem Schlüssel zu initialisieren ($IV = k$) oder den Schlüssel als Präfix der Nachricht voranzustellen: $\mathrm{MAC}_k(x) = F(k||x)$ Im Prinzip sind diese beiden Wege identisch (in der Praxis wird der Initialwert einer Funktion kürzer als die Blocklänge sein, so dass der Schlüssel durch Padding verlängert oder irgendwie verkürzt werden müsste).

Dieses Verfahren ist unsicher, weil jeder Tag genug Informationen über den Schlüssel erhält, um weitere Nachrichten zu authentifizieren. Weil der Tag nichts anderes als die Kettenvariable ist, kann ein Angreifer mit einer bekannten Nachricht x beliebig viele neue Nachrichten authentifizieren, solange diese x als Präfix haben (es sei $IV = 0$): $\mathrm{MAC}_k(x||x') = F(k||x||x') = F(\mathrm{MAC}_k(x)||x')$

3.3.3 Geheimes Suffix

Alternativ zum Voranstellen kann der Schlüssel angehängt werden mit: $\mathrm{MAC}_k(x) = F(x||k)$.

Hauptschwäche ist nun, dass ein Angreifer einen Geburtstagsangriff offline durchführen kann indem er versucht ein Nachrichtenpaar (x, x') mit $F(x) = F(x')$ zu finden. Dies ist natürlcih sehr schwierig, aber deutlich einfacher als ein Angriff online, bei dem das Opfer die Berechnungen durchführen müsste.

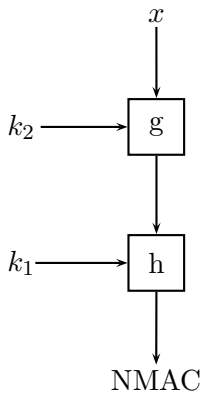

Abbildung 3: NMAC

3.3.4 Umschlag

Verbindet man die Präfix- und Suffix-Methode, gelangt man so zur sogenannten Umschlags- (engl. Envelope) Methode, bei der der Nachricht ein Schlüssel vorangestellt und ein zweiter (nicht notwendigerweise verschiedener) Schlüssel angehängt wird: $\mathrm{MAC}_{(k_1,k_2)}(x) = F(k_1||x||k_2)$

Hinsichtlich der Sicherheit ist zu beachten, dass ein Angreifer durch geschicktes Suchen einer internen Kollision erst k_1 finden und dann (wie beim Suffix) offline nach k_2 suchen kann. Ein Angreifer benötigt daher weniger Rechenleistung als es durch die Schlüssellänge von $|k_1| + |k_2|$ Bits zunächst den Anschein hat [Preneel95].

3.4 NMAC

Deutlich größere Sicherheit bieten *Nested MACs* (NMACs). Dabei wird die Ausgabe einer MAC-Funktion noch einmal von einer anderen verarbeitet. Es werden also zwei hash-basierte MACs verknüpft mit:

$$\mathrm{NMAC}_{K_1,K_2}(x) = h_{K_1}(g_{K_2}(x))$$

Der entscheidende Vorteil dabei ist, dass ein Angreifer keine Informationen über die Ausgabe von g_{K_2} bzw. die Eingabe von h_{K_1} erhält. Das macht viele Angriffe auf besondere Eigenschaften der benutzten Funktion unmöglich.

Betrachtet man alle überhaupt möglichen Angriffe, so gibt es drei Möglichkeiten:

- Erzeugen einer Kollision in g_{K_2},
  d. h. ein Paar (x', x'') mit $g_{K_2}(x') = g_{K_2}(x'')$ finden.
- Fälschen von h_{K_1}, dem sogenannten *little MAC*,
  d. h. ein Paar (x', x'') mit $h_{K_1}(x') = h_{K_1}(x'')$ finden.
- Fälschen des ganzen NMAC_{K_1,K_2}, dem sogenannten *big MAC*,
  d. h. ein Paar (x', x'') mit $h_{K_1}(g_{K_2}(x')) = h_{K_1}(g_{K_2}(x''))$ finden.

Nun lässt sich beweisen, dass die Erfolgswahrscheinlichkeit ϵ eines Angriffs auf NMAC_{K_1,K_2} von den Erfolgswahrscheinlichkeiten ϵ_g, ϵ_h der Angriffe auf g und h abhängt und es gilt: $\epsilon \leq \epsilon_g + \epsilon_h$ [Stinson02, S. 138f], [Bellare96, S. 10f].

Damit ist die Sicherheit des NMAC sehr eng an die Sicherheit der zugrundeliegenden Funktionsfamilien g und h. Gibt es in g oder h eine Schwäche, so lässt diese sich ausnutzen, um einen NMAC zu fälschen. Umgekehrt zeigt jedes Verfahren, das NMAC fälschen kann, dass g oder h nicht sicher ist.

3.5 HMAC

Die Hashfunktionen des NMAC mit Schlüsseln zu versehen, kann immer eine Änderung ihrer Implementierung mit sich bringen. In vielen Fällen ist es wünschenswert ganz ohne Änderung auszukommen und eine gegebene Hash-Funktion zu benutzen (zum Beispiel wenn die Hash-Funktion in Hardware implementiert ist oder aus einer Programm-Bibliothek verwendet wird).

Für diesen Einsatz ist HMAC besonders gut geeignet:

$$\mathrm{HMAC}_K(x) = H(K \oplus \mathrm{opad} || H(K \oplus \mathrm{ipad} || x))$$

HMAC ist ein Spezialfall von NMAC, der eine gegebene Hash-Funktion H als „black box“ benutzt wird. Dadurch ist die Hash-Funktion beliebig austauschbar.

Außerdem ist in HMAC nur ein Schlüssel vorgesehen, aus dem dann zwei Schlüssel für die innere und äußere Hashfunktion abgeleitet wird. Dafür

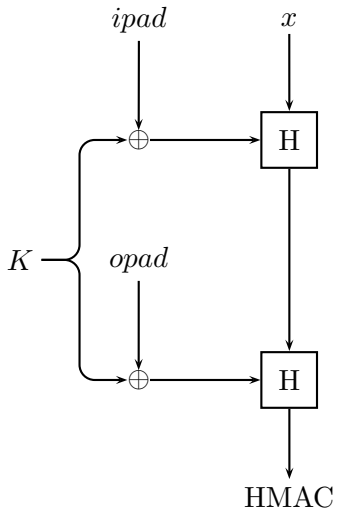

Abbildung 4: HMAC

sind die Werte ipad = 0x5c5c5c...5c5c und opad = 0x363636...3636 definiert (die Länge entspricht der benutzten Blocklänge). Das XOR aus dem Schlüssel K und ipad bzw. opad ergibt die Teilschlüssel $K \oplus$ ipad für die innere und $K \oplus$ opad für die äußere Anwendung der Hash-Funktion. Die Werte wurden gewählt, damit sich die Teilschlüssel in der Hälfte ihrer Bits unterscheiden (unter Ausnutzung des *Avalanche Effekts* sollen die Hash-Resultate dadurch besonders unvorhersagbar werden).

Theoretisch wird HMAC dadurch schwächer als NMAC mit zwei unabhängigen und zufälligen Schlüsseln. Praktisch kann dies ignoriert werden, weil ein Angreifer ohnehin nie Zugriff auf den bzw. die Schlüssel bekommen darf und die Unvorhersagbarkeit der Schlüssel über mehrere Nachrichten hinweg weitaus wichtiger ist.

Vorgeschlagen und analysiert wurde HMAC in [Bellare96], eine genaue Spezifikation findet sich darüberhinaus in RFC 2104 [Krawczyk97].

4 Bedingungslos sichere MACs

4.1 Begriffsbestimmung

Wie im vorhergehenden Abschnitt gezeigt, sind die derzeit gebräuchlichen MAC-Verfahren insofern sicher, als dass sie nach aktuellem Wissensstand nicht mit heute einzusetzender Rechenleistung zu brechen sind.

Neben diesem herkömmlichen Ansatz, der die Probleme und Verfahren nach ihrer Komplexität beurteilt, gibt es aber auch noch einen zweiten informationstheoretischen. Dieser verspricht eine *bedingungslose Sicherheit*, unabhängig von Ressourcen wie Zeit und Rechenleistung.

Wichtigste Bausteine mit denen diese bedingungslose Sicherheit für MACs zu erreichen ist, sind universelle Hash-Familien.

4.2 Universelle Hash-Familien

In der Literatur sind diese Hash-Familien auch als Wegman-Carter-Hashs bzw. -MACs zu finden (nach den Autoren, die sie zuerst beschrieben). In welchem Maße eine Hash-Familie universell ist, hängt von der Gleichverteilung der einzelnen Funktionen und ihrer Werte ab.

Sei $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ eine Hash-Familie und ϵ eine Konstante mit $1/|\mathcal{Y}| \leq \epsilon \leq 1$. Wir haben dann eine

- *universelle Hash-Familie*,
  wenn $\forall x, y \in \mathcal{X}(x \neq y) : \Pr_{h \in \mathcal{H}}[h(x) = h(y)] = 1/|\mathcal{Y}|$
- *stark-universelle Hash-Familie*,
  wenn $\forall x, y \in \mathcal{X}(x \neq y); a, b \in \mathcal{Y} : \Pr_{h \in \mathcal{H}}[h(x) = a, h(y) = b] \leq 1/|\mathcal{Y}|^2$
- *ϵ-fast-universelle Hash-Familie*,
  wenn $\forall x, y \in \mathcal{X}(x \neq y) : \Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq \epsilon$
- *ϵ-fast-stark-universelle Hash-Familie*,
  wenn $\forall x, y \in \mathcal{X}(x \neq y); a, b \in \mathcal{Y} : \Pr_{h \in \mathcal{H}}[h(x) = a, h(y) = b] \leq \epsilon/|\mathcal{Y}|$

Eine ϵ-fast-stark-universelle Hash-Familie ist also genau dann eine stark-universelle Hash-Familie, wenn $\epsilon = 1/|\mathcal{Y}|$.

4.3 Sicherheit universeller MACs

Es wird vorausgesetzt, dass jeder Schlüssel nur einmal verwendet werden darf (vgl. Abschnitt 2.2). Es seien nun P_0 und P_1 die maximale Erfolgswahrscheinlichkeiten ϵ, für $(\epsilon, 0)$- bzw. $(\epsilon, 1)$-Angriffe auf eine Hash-Familie, d. h. über alle möglichen Schlüssel hinweg. Es ist leicht zu sehen, dass $1/|\mathcal{Y}|$ immer untere Grenze ist: $P_0 \geq 1/|\mathcal{Y}|$ und $P_1 \geq 1/|\mathcal{Y}|$ (diese Erfolgswahrscheinlichkeit hat man durch zufälliges Raten irgendeines Tags).

Die Gleichheit $P_0 = 1/|\mathcal{Y}|$ liegt genau dann vor, wenn alle Tags mit gleicher Wahrscheinlichkeit (über alle Nachrichten und Schlüssel hinweg) vorkommen können. Genau dann wenn eine ϵ-fast-stark-universelle Hash-Familie vorliegt, ist $P_0 = 1/|\mathcal{Y}|$ und $P_1 \leq \epsilon$. Genau dann wenn eine stark-universelle Hash-Familie vorliegt, ist $P_0 = P_1 = 1/|\mathcal{Y}|$ [Stinson02].

Der Vorteil einer stark-universelle Hash-Familie ist offensichtlich: Ein damit konstruierter MAC ist bedingungslos sicher. Erkauft wird diese Sicherheit allerdings mit sehr großen Schlüssellängen [Stinson94], [Preneel97].

Ein Ausweg aus diesem Dilemma ist die Verwendung von ϵ-fast-stark-universellen Hash-Familien. Schon minimale Erhöhungen von ϵ (also potentielle Verringerungen der Sicherheit) erlauben sehr viel kleinere Schlüssel.

Es gibt bereits mehrere MAC-Verfahren, die auf solchen fast-universellen Hash-Familien basieren und nicht nur theoretisch sicherer, sondern auch deutlich schneller als bisherige Verfahren sind. Eines davon – UMAC – steht kurz vor einer Veröffentlichung als RFC [Krovetz04]. Es ist zu erwarten, dass diese Art von MAC dadurch zu allgemeiner Verbreitung kommt.

Literatur

[Bellare96] Mihir Bellare, Ran Canetti and Hugo Krawczyk. *Keying Hash Functions for Message Authentication*. In: N. Koblitz, ed., *Advances in Cryptology – Crypto 96 Proceedings*. Springer-Verlag (1996). URL: `http://www.cse.ucsd.edu/users/mihir/papers/hmac.html`

[Bellare00] Mihir Bellare, Joe Kilian and Phillip Rogaway. *The Security of Cipher Block Chaining*. In: Journal of Computer and System Sciences, vol. 61, no. 3:pp. 362–399 (2000). URL: `http://www.cs.ucdavis.edu/~rogaway/papers/cbc-abstract.html` Eine kürzere Version erschien schon vorher in: *Advances in Cryptology – Crypto 94 Proceedings (LNCS 839)*, Springer-Verlag, Berlin, Heidelberg (1994)

[Black00] John Black. *Message Authentication Codes*. Ph.D. thesis, University of California Davis (2000). URL: `http://www.cs.colorado.edu/~jrblack/papers/thesis.pdf`

[Goldwasser01] Shafi Goldwasser and Mihir Bellare. *Lecture Notes on Cryptography* (2001). URL: `http://www.cse.ucsd.edu/users/mihir/papers/gb.html`

[Krawczyk97] Hugo Krawczyk, Mihir Bellare and Ran Canetti. *RFC 2104, HMAC: Keyed-Hashing for Message Authentication* (1997). URL: `http://www.rfc-editor.org/rfc/rfc2104.txt`

[Krovetz04] Ted Krovetz. *UMAC: Fast and Provably Secure Message Authentication* (2004). URL: `http://www.cs.ucdavis.edu/~rogaway/umac/`

[Menezes97] Alfred Menezes, Paul van Oorschot and Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton (1997). URL: `http://www.cacr.math.uwaterloo.ca/hac/`

[Preneel95] Bart Preneel and Paul van Oorschot. *MDx-MAC and Building Fast MACs from Hash Functions.* In: D. Coppersmith, ed., *Advances in Cryptology – Crypto 95 Proceedings (LNCS 963)*, pp. 1–14. Springer-Verlag, Berlin, Heidelberg (1995)

[Preneel97] Bart Preneel. *Cryptographic primitives for information authentication – state of the art (LNCS 1528).* In: State of the Art and Evolution of Computer Security and Industrial Cryptography (1997). URL: `http://www.cosic.esat.kuleuven.ac.be/publications/article-346.pdf`

[Rompay04] Bart van Rompay. *Analysis and Design of Cryptographic Hash Functions, MAC Algorithms and Block Ciphers.* Ph.D. thesis, Katholieke Universiteit Leuven, Leuven (2004). URL: `http://www.cosic.esat.kuleuven.ac.be/publications/thesis-16.pdf`

[Stinson94] Douglas R. Stinson. *Universal hashing and authentication codes.* In: Designs, Codes and Cryptography, , no. 4:pp. 369–380 (1994). URL: `http://www.cacr.math.uwaterloo.ca/~dstinson/papers/hashingdcc.ps` Eine kürzere Version erschien schon vorher in: *Advances in Cryptology – Crypto 91 Proceedings (LNCS 576)*, Springer-Verlag, Berlin, Heidelberg (1991)

[Stinson02] Douglas Stinson. *Cryptography – Theory and Practice.* Chapman & Hall/CRC, second edn. (2002)