Karlsruhe Institute of Technology

KASTEL

Titel

Bachelorarbeit

KIT – Karlsruher Institut für Technologie
ITI – Institut für Theoretische Informatik
Forschungsruppe Kryptographie und Sicherheit

Yuguan Zhao

07. Juli 2017

Verantwortlicher Betreuer:	Prof. Dr. rer. nat. Jörn Müller-Quade
Betreuender Mitarbeiter:	Jeremias Mechler, M.Sc

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung beachtet habe.

Karlsruhe, den 07. Juli 2017

(Yuguan Zhao)

Inhaltsverzeichnis

Notation

Spezielle Bezeichner

H	Kollisionsresistente Hashfunktion
CH	Chameleon Hashfunktion
$\mathcal{S}$	Unterzeichner
$\mathcal{Z}$	Zensor
$\mathcal{V}$	Verifizierer
$m_1, ..., m_t$	Zusammensetzung einer Nachricht m
ID_m	Identifikationsnummer einer Nachricht m
g	Erzeuger einer zyklischen Gruppe von Primzahlordnung
q	Primzahlordnung einer zyklischen Gruppe
e	String-Hashwert bestimmter Länge einer Nachricht
σ	Signatur
r	Zufallszahl
b	Ein Bit
pk	Öffentlicher Schlüssel
sk	Geheimschlüssel
vk	Verifikationsschlüssel

Einleitung

0.1 Motivation und Zielsetzung

E-Mail-Signaturen umfassen komplette Nachrichten, also insbesondere Body und Anhänge. Löscht man einen Anhang, beispielsweise um Speicherplatz zu sparen, kann die Gültigkeit der Signatur nicht mehr verifiziert werden. Ziel meiner Arbeit ist es, passende S/MIME Signaturverfahren, Standards für die Verschlüsselung und das Signieren von MIME-gekapselter E-Mail, durch ein hybrides Kryptosystem umzusetzen, bei dem die Integrität der verbleibenden Teile weiterhin verifiziert werden kann. Für die Umsetzung sind mehre Wege denkbar. Zwei Ansätze werden hier untersucht und gegenüber gestellt. Der erste Ansatz wäre die Verwendung von sogenannte Sanitizable Signatures. Ein weiterer Ansatz, der auch Rückwärtskompatibilität zu vorhandenen E-Mail-Lösungen verspricht, wäre das Hash-Tree-Verfahren. Die nichtabstreitbare Urheberschaft und Integrität der Nachrichten sollen beibehalten.

> „Integrität bezeichnet die „Korrektheit (Unversehrtheit) von Daten und der korrekten Funktionsweise von Systemen “
>
> — Glossar des Bundesamtes für Sicherheit in der Informationstechnik

0.2 Grundlagen

0.2.1 Vorbereitung

Die Operation $s \xleftarrow{R} S$ beschreibt, dass ein Element s zufällig aus der Menge S gezogen wird. $c \leftarrow F(a, b\ldots)$ zeigt, dass ein Algorithmus F mit den Eingaben $(a, b\ldots)$ c als Ergebnis liefert. Die Notation $F(a; r)$ bezeichnet einen auf Zufall basierten Algorithmus F mit Eingabe a und Zufallsvariable r.

0.2.2 Aufbau einer Email

Der Aufbau einer E-Mail ist im RFC 5322 festgelegt. E-Mails sind intern in zwei Teile geteilt: Die Header-Felder mit Kopfzeilen und den Body (Textkörper) mit dem eigentlichen Inhalt der

Nachricht. Header-Felder sind Zeilen beginnend mit einem Feldnamen, gefolgt von einem Kolon („:“) und einem Feldkörper. Ein Header-Feld endet mit einem CRLF (Zeilenumbruch). Hier wird z.B. die Absenderangabe unter „From:“festgelegt. Innerhalb des Bodys werden weitere Untergliederungen definiert. Der Body einer E-Mail ist durch eine Leerzeile vom Header getrennt. Eine E-Mail darf gemäß RFC 5322 Abschnitt 2.3 nur Zeichen des 7-Bit-ASCII-Zeichensatzes enthalten. Sollen andere Zeichen, wie zum Beispiel deutsche Umlaute oder Daten wie zum Beispiel Bilder übertragen werden, müssen das Format im Header-Abschnitt deklariert und die Daten passend kodiert werden. Geregelt wird das durch RFC 2045. Außerdem müssen CR (Wagenrücklauf) und LF (Zeilenvorschub) zusammen als CRLF auftreten. Die Trennlinie der Untergliederung von Body wird in Header unter „Content-Type: “definiert.

0.2.3 Hashfunktion

Eine Hashfunktion ist eine Abbildung, die eine große Eingabemenge auf eine kleinere Zielmenge abbildet. Eine Hashfunktion ist daher im Allgemeinen nicht injektiv. Die Eingabemenge kann Elemente unterschiedlicher Längen enthalten, die Elemente der Zielmenge haben dagegen meist eine feste Länge. In meiner Arbeit wird SHA-256 verwendet, um Nachrichten beliebiger Länge auf eine fest Länge umzuwandeln.
Zur Erzeugung des Hash-Wertes bei SHA-256 werden die Quelldaten in 512-Bit-Blöcke bzw. sechzehn 32-Bit-Wörter aufgeteilt und iterativ mit 64 Konstanten unter Verwendung von vier logischen Funktionen verrechnet. Die Konstanten werden aus den binären Nachkommastellen der Kubikwurzeln der ersten 64 Primzahlen gebildet. Dabei wird mit einem Start-Hash aus acht 32-Bit-Wörtern begonnen, bestehend aus den ersten 32 Bits des Nachkommateils der Quadratwurzeln der ersten acht Primzahlen.

0.2.4 Signatur-Schema

Ein Signatur-Schema kann man mit Hilfe von einem beliebigen IND-CPA-sicheren Verschlüsselungsverfahren via Black-Box-Reduktion konstruieren[Rom90]. wir betrachten folgendes Signatur-Schema SS = (Sigkg, Sign, Verify) . Der Verifikationsschlüssel vk kann mit Sigkg generiert werden. vk besteht aus einer Folge von Bits. Die Länge von vk interpretieren wir als Zahlen in $\{1, ..., 2^k\}$ mit Sicherheitsparameter k [Buc+11] [Lam79].

- Sigkg(1^k) erhält als Eingabe einen Sicherheitsparameter k in unärer Notation und gibt ein Schlüsselpaar (vk, sk) aus.
- Sign($1^k, sk, m$) erhält als Eingabe den Sicherheitsparameter 1^k, den Geheimschlüssel sk eine Nachricht m und gibt eine Signatur σ aus.

- Verify$(1^k, vk, m, \sigma)$ erhält als Eingabe den Sicherheitsparameter 1^k, den öffentlichen Schlüssel pk, eine Nachricht m, eine Signatur σ und gibt entweder 1 oder 0 aus.
  1 bedeutet, dass σ als Signatur für Nachricht m und öffentlichen Schlüssel pk akzeptiert wird. Bei 0 wird die Signatur nicht akzeptiert.

0.2.5 Commitment-Schema

Das Commitment-Schema ist ein grundlegender Bestandteil vieler kryptographischer Protokolle. Sie ermöglichen einer Partei, sich gegenüber einer anderen Partei auf einen Wert festzulegen, ohne etwas über diesen Wert zu verraten. Später kann dieser Wert dann aufgedeckt werden. Ein Commitment-Schema ein kryptographisches Zwei-Parteien- und Zwei-Phasen-Protokoll, welches aus Commit- und Reveal-Phase besteht: Com = (Commit, Reveal) [CLP10]. Der Sender bestimmt einen festen Wert und übergibt diesen dem Empfänger (Commit-Phase). Zum Aufdecken (Reveal-Phase) bekommt der Empfänger von dem Sender den dazu nötigen Parameter. Damit das Verfahren korrekt ist, wird gefordert, dass der ursprüngliche Wert nach dem Aufdecken des Commitments wiederhergestellt sein muss. Die Algorithmen beider Parteien sollte man mit zwei PPT-Turingmaschinen umsetzen können. Dieses Protokoll muss außerdem folgende zwei Eigenschaften erfüllen:

Definition 0.1 (Binding) *Es darf nicht möglich sein, ein Commitment nachträglich auf einen anderen Wert zu ändern. Nachdem die Commit-Phase beendet ist, gibt es nur einen Wert, den ein möglicherweise schummelnder Sender den Empfänger offenlegen kann, sodass dieser akzeptiert [Ber13].*

Definition 0.2 (Hiding) *Das Commitment darf keinen Rückschluss auf den Wert zulassen, auf den sich der Committer festgelegt hat. Das muss auch gelten, falls der Empfänger zu schummeln versucht.*

1 Verfahren mit Sanitizable Signatures

Sanitizable Signatures werden in den Situationen verwendet, in denen eine ordnungsgemäß bevollmächtigte dritte Partei (Zensur) möglicherweise das Dokument auf eine kontrollierte und begrenzte Weise modifizieren muss. Der autorisierte Zensor muss dabei eine gültige Signatur für das aktualisierte Dokument erstellen, ohne den ursprünglichen Unterzeichner zu kontaktieren. Es kann viele mögliche Gründe dafür geben, den ursprünglichen Unterzeichner nicht erneut unterzeichnen zu lassen:

1. Der Schlüssel des Unterzeichners ist abgelaufen.
2. Die Originalunterschrift wurde fest mit Zeit gestempelt.
3. Der Unterzeichner ist möglicherweise nicht erreichbar.
4. Jede neue Signatur würde zu viel kosten, entweder in Bezug auf die tatsächlichen Kosten oder in Bezug auf die Berechnung.

In diesem Kapitel stellen wir den Begriff der Sanitizable Signaturen vor. Ein Sanitizable Signature-Schema erlaubt einem vertrauenswürdigen Zensor, bestimmte Teile des Dokuments zu ändern und eine gültige Unterschrift auf das rechtlich geänderte Dokument ohne jegliche Hilfe des ursprünglichen Unterzeichners zu geben. Diese geänderte Teile des Dokuments sind explizite Segmente nach vorheriger Vereinbarung zwischen dem Unterzeichner und dem Zensor als veränderbar angegeben. Der Zensor kann nur dann eine gültige Signatur erzeugen, wenn er nur diese Teile modifiziert und keine andere Teile des Dokuments. Konkret muss eine Sanitizable Signature die folgenden Eigenschaften aufweisen:

1. Unveränderlichkeit: Der Zensor sollte nicht in der Lage sein, irgendeinen Teil des Dokuments zu modifizieren, der von dem ursprünglichen Unterzeichner nicht ausdrücklich als änderbar bestimmt wird.
2. Datenschutz: Bei einer Nachricht mit einer gültigen Sanitizable Signatur ist es für niemanden möglich (mit Ausnahme des Unterzeichners und des Zensors), Informationen über die Teile der Nachricht abzuleiten, die von dem Zensor bereinigt wurden. In anderen Worten sind alle sanizisierten Informationen nicht aufdeckbar.

3. Verantwortlichkeit: Im Falle eines Widerspruchs kann der Unterzeichner einem vertrauenswürdigen Dritten nachweisen, dass eine bestimmte Nachricht durch den Zensor geändert wurde.

4. Transparenz: Bei einer signierten Nachricht mit einer gültigen Unterschrift sind nur der Zensor und der Unterzeichner in der Lage, richtig zu erraten, ob die Nachricht geändert wurde.

1.1 Konstruktion basierend auf Chameleon-Hash-Verfahren

In diesem Abschnitt stelle ich eine Konstruktion von Sanitizable Signature basierend auf Chameleon-Hash vor. Eine wichtige Eigenschaft eines Signaturverfahrens ist die Nichtabstreitbarkeit der Signatur. Wenn eine Signatur mit einem öffentlichen Schlüssel pk verifiziert wurde, sollte damit auch bewiesen sein, dass die Signatur mit dem zugehörigen Geheimschlüssel sk erzeugt wurde. Wir konstruieren hier ein einfaches Signaturschema mit RSA.

1.1.1 Setup

- Ein Unterzeichner $\mathcal{S}$ mit $(pk_{\text{sign}}, sk_{\text{sign}})$ von Signatur-Schema, ein Zensor $\mathcal{Z}$ mit $(pk_{\text{sanit}}, sk_{\text{sanit}})$ von Sanitizable-Signatures-Schema und ein Verifizierer $\mathcal{V}$ sind beteiligt.

- Die Berechnung von Chameleon-Hash-Verfahren mit einer Nachricht m einer Zufallszahl r und einem pk wird als $CH_{pk}(m, r)$ bezeichnet. Eine Chameleon-Hash-Funktion (aka Trapdoor-Commitment) erlaubt es, gegeben die Falltür (der zu pk gehörige sk), Paare (m, r), $(\tilde{m}, \tilde{r})$ mit $m \neq \tilde{m}$ zu finden, so dass $CH_{pk}(m, r) = CH_{pk}(\tilde{m}, \tilde{r})$. Das bedeutet, dass man mit sk solche Paare finden kann und es andernfalls rechentechnisch unmöglich sein sollte. Besonders praktisch ist es bei den Sanitizable Signatures, bei denen das Wissen über sk es einem ermöglicht, eine bereits signierte Nachricht zu verändern, ohne dabei die alte Signatur zu invalidieren. Per Definition sind Chameleon-Hash-Verfahren probabilistische Algorithmen. Um die Korrektheit eines berechneten Chameleon-Hash-Wertes zu bestimmen, ist es notwendig, sowohl die ursprüngliche Nachricht m als auch die Zufallszahl r anzugeben.

1.1.2 Konstruktion

Angenommen möchten wir ein Dokument $m = (m_1, ..., m_t)$ signieren, welches in t Blöcke aufgeteilt ist. Zunächst wählt der $\mathcal{S}$ eine zufällige eindeutige Dokument-Identifikationsnummer ID_m und bestimmt änderbare Teile. Angenommen $m_{i_1}, ..., m_{i_k}$ des Dokuments kann $\mathcal{Z}$ mit dem

öffentlichen Schlüssel pk_{sanit} modifizieren. Dazu berechnet $\mathcal{S}$ zu jedem m_i einen Chameleon-Hash mit $\bar{m}_i = CH_{pk_{\text{sanit}}}(ID_m \| i \| m_i, r_i)$ für $i \in \{i_1, i_2, \ldots, i_k\}$. Andernfalls besteht $\bar{m}_i$ aus $m_i \| i$. Der Wert r sollte als Verkettung aller Zufallszahlen r_{i_k} mit $i = 1,\ldots,k$ interpretiert werden. Zum Schluss kann $\mathcal{S}$ signieren:

$$\sigma = SIGN(m, r; sk_{\text{sign}}, pk_{\text{sanit}}) := \mathcal{S}_{sk_{\text{sign}}}(ID_m \| t \| pk_{\text{sign}} \| \bar{m}_1 \| \ldots \| \bar{m}_t)$$

Um die obige Signatur zu überprüfen, benötigt man σ, m, r und Hilfsinformation, um eine Formatierung von m in Blöcke zu ermöglichen. Die Länge der sanitizable Signatur ist nur proportional zur Anzahl der veränderbaren Nachrichtenblöcke.
Weil nur der Zensor den sk_{sanit} kennt, der zu pk_{sanit} gehört, kann er nun Kollisionen der Chameleon-Hashs mit beliebigen Nachrichten produzieren:

$$CH_{pk_{\text{sanit}}}(ID_m \| i \| m_i, r_i) = CH_{pk_{\text{sanit}}}(ID_m \| i \| \tilde{m}_i, \tilde{r}_i)$$

Die Verwendung der ID und eines Blockindex ist notwendig, um die Wiederverwendung von veränderbaren Blöcken innerhalb einer Nachricht zu verhindern.

2 Verfahren mit Hash Tree

2.1 Toeplitz-Matrix

Nach Otto Toeplitz bekannte Toeplitz-Matrizen sind (endliche oder unendliche) Matrizen mit einer speziellen Struktur. In unserer Arbeit wird die endliche quadratische Version verwendet. Die Länge der Zeilen und Spalten bestimmt die Bitlänge des gehashten einzelnen Nachrichtenteils.

Definition 2.1 (Toeplitz-Matrix) *Eine Matrix* $\mathsf{A} = (\mathsf{a}_{ij})$ *wird Toeplitz-Matrix genannt, wenn die Einträge* a_{ij} *nur von der Differenz* $i - j$ *der Indizes abhängen. Die Haupt- und Nebendiagonalen der Matrix sind also konstant. Eine endliche Toeplitz-Matrix mit* u *Zeilen und* v *Spalten ist somit durch die* $u + v - 1$ *Einträge am linken und oberen Rand (also die erste Zeile und erste Spalte) vollständig bestimmt.*

Quadratische Toeplitz-Matrizen sind persymmetrisch, das heißt ihre Einträge ändern sich nicht, wenn sie an der Gegendiagonale der Matrix gespiegelt werden.
Für große lineare Gleichungssysteme $\mathsf{Ax} = \mathsf{b}$, bei denen A eine Toeplitz-Matrix ist, gibt es besonders effiziente Lösungsverfahren. Dabei werden häufig unendlich große Toeplitz-Matrizen durch ihre Erzeugungsfunktion beschrieben. Sofern diese Fourier-transformierbar sind, können die Operationen Matrizenmultiplikation und Matrixinversion auf einfache Multiplikationen bzw. Divisionen zurückgeführt werden.

2.2 Commitment mit Toeplitz-Matrix

Ein Nachrichtenteil m_i wird mit einer sicheren Hashfunktion $\mathsf{H1}$ auf bestimmte Bitlänge gehasht und anschließend in einen Bitstring $m^*_{b_i}$ umgewandelt. Mit der Bitlänge l von $m^*_{b_i}$ erzeugt man eine zufällige Toeplitz-Matrix $A \in \{0,1\}^{l \times l}$. Zudem wird ein Randomvektor $r \in \{0,1\}^l$ generiert. Eine neue Hashfunktion $m^*_{b_i} \leftarrow \mathsf{H2(r)}$ mit $\mathsf{H2(r)} \coloneqq \mathsf{A} \cdot \mathsf{r} + \mathsf{vec}$ legt man fest. Der Vektor vec ergibt sich durch $m^*_{b_i} - A \cdot r$. Abschließend wird die Hashfunktion $\mathsf{H1}$ auf r angewandt und

man erhält den Wert z. Der Empfänger bekommt als Commitment z und H2 zugeschickt.

$$
\begin{aligned}
&\mathsf{Commit}(\mathsf{m_i}) :=\\
&m^*_{b_i} \leftarrow \mathsf{H1}(\mathsf{m_i})\\
&l = |m^*_{b_i}|\\
&A, r \leftarrow F(l)\\
&\mathsf{H2}(\mathsf{r}) := \mathsf{A} \cdot \mathsf{r} + \mathsf{vec}\\
&vec = m^*_{b_i} - A \cdot r\\
&z \leftarrow \mathsf{H1}(\mathsf{r})\\
&\text{return}\{z, \mathsf{H2}\}
\end{aligned}
$$

2.3 Schema HashTree

3 Realisierung in Java

3.1 Email Parser in Java

3.2 Sanitizable Signatures in Java

3.3 Schema HashTree in Java

4 Vergleich von Ergebnissen

5 Zusammenfassung und Ausblick