Aufgabenblatt 06

PS Einführung in die Kryptographie

Andreas Schlager

1. Mai 2025

Inhaltsverzeichnis

Aufgabe 21. *Fortsetzung Aufgabe 20.): Erklären sie, warum die Verwendung von Hashfunktionen im Kontext mit digitalen Signaturen die Protokollattacke gegen digitale Empfangsbestätigungen verunmöglicht. Bedenken sie dabei insbesondere, dass in diesem Fall NachrichtenHash und Nachricht übermittelt werden (die Fragen in Item 2 auf Skriptum S. 93 müssen bearbeitet werden).*

Sei m die Nachricht, die Alice an Bob übermitteln möchte. Alice verschlüsselt die Nachricht mit Bobs öffentlichem Schlüssel und sendet ihm $E_B(m)$. Zusätzlich erzeugt sie den Hashwert $h(m)$ mittels einer kryptografischen One-Way-Hashfunktion, signiert diesen mit ihrem privaten Schlüssel und verschlüsselt auch die Signatur mit Bobs öffentlichem Schlüssel:

$$\left.\begin{array}{l} E_B(m) \\ E_B(S_A(h(m))) \end{array}\right\} \rightarrow \text{Bob}$$

Bob entschlüsselt beide Nachrichtenkomponenten, extrahiert den Klartext m sowie die Signatur $S_A(h(m))$, und berechnet anschließend selbst den Hashwert $h(m)$. Anschließend überprüft er die Signatur mit Alices öffentlichem Schlüssel. Stimmen der berechnete Hash $h(m)$ und der signierte Hash überein, ist die Integrität der Nachricht sichergestellt.

Im Anschluss sendet Bob eine digitale Empfangsbestätigung zurück. Diese besteht nicht aus einer Signatur der vollständigen Nachricht, sondern aus der Signatur des Hashwerts:

$$E_A(S_B(h(m)))$$

Eine Protokollattacke, wie sie in Aufgabe 20 beschrieben wurde, ist in diesem Szenario nicht mehr möglich. Grund hierfür sind die Eigenschaften der verwendeten Hashfunktion:

- **Kollisionsresistenz:** Es ist praktisch nicht möglich, zwei verschiedene Nachrichten $m \neq m'$ zu finden, sodass $h(m) = h(m')$ gilt.
- **Pre-Image-Resistenz:** Aus einem gegebenen Hashwert $h(m)$ kann keine gültige Nachricht m rekonstruiert werden.

Selbst wenn Mallory die Nachrichtenkomponenten vom ursprünglichen Protokollangriff kennt und in der Lage wäre, die Signaturen und Verschlüsselungen zu trennen, fehlt ihm dennoch die Möglichkeit, aus dem Hashwert $h(m)$ die Originalnachricht m zu rekonstruieren oder eine alternative Nachricht m' mit demselben Hash zu erzeugen. Somit verhindert die Einbindung von One-Way-Hashfunktionen diese Art der Attacken gegen digitale Empfangsbestätigungen.

Aufgabe 22. *HÜ10 auf S. 86 der VO-Slides.*

Fall 1 Sei M eine beliebige, aber fixe Nachricht zu der eine Nachricht M' gefunden werden soll, sodass $h(M) = h(M')$. Da die Hashfunktion einen m-Bit langen Output produziert, gibt es 2^m mögliche Hashwerte. Die Wahrscheinlichkeit eine entsprechenden Nachricht M' zu generieren ist

$$P(h(M) = h(M')) = \frac{1}{2^m}.$$

Das zufällige erzeugen einer Nachricht von Nachricht $M_1, M_2, \dots$ bis der Hashwert übereinstimmt, entspricht einer geometrischen Verteilung.

$$\mathbb{E}[N] = \frac{1}{p} = \frac{1}{\frac{1}{2^m}} = 2^m$$

Fall 2 Man berechnet $M_1, M_2, \dots, M_n$ und will wissen wie viele Nachrichten man generieren muss, damit es zumindest eine Übereinstimmung der Hashwerte gibt. Dafür betrachtet man die Gegenwahrscheinlichkeit, dass alle Hashwerte unterschiedlich sind.

$$\begin{aligned}
\#\text{ günstige Fälle} &= 2^m \cdot (2^m - 1) \cdots (2^m - (n-1)) \\
\#\text{ mögliche Fälle} &= (2^m)^n \\
P(\text{alle unterschiedlich}) &= \frac{2^m}{2^m} \cdot \frac{2^m - 1}{2^m} \cdots \frac{2^m - (n-1)}{2^m} = \prod_{k=0}^{n-1} \left(1 - \frac{k}{2^m}\right)
\end{aligned}$$

Wir schreiben das Produkt durch den Logarithmus in eine Summe um.

$$\begin{aligned}
P(\text{alle unterschiedlich}) &= \prod_{k=0}^{n-1} \left(1 - \frac{k}{2^m}\right) \\
\ln P(\dots) &= \sum_{k=0}^{n-1} \ln \left(1 - \frac{k}{2^m}\right)
\end{aligned}$$

Da $k \ll 2^m$ kann für $\ln(1-x)$ die Annäherung $\ln(1-x) \approx -x$ getroffen werden.

$$\sum_{k=0}^{n-1} \ln \left(1 - \frac{k}{2^m}\right) = -\sum_{k=0}^{n-1} \frac{k}{2^m} = -\frac{1}{2^m} \sum_{k=0}^{n-1} k = -\frac{1}{2^m} \cdot \frac{n(n-1)}{2}$$

Für die Gegenwahrscheinlichkeit bedeutet das

$$\begin{aligned}
\ln P(\text{alle unterschiedlich}) &\approx -\frac{1}{2^m} \cdot \frac{n(n-1)}{2} \\
P(\text{alle unterschiedlich}) &\approx \exp\left(-\frac{1}{2^m} \cdot \frac{n(n-1)}{2}\right)
\end{aligned}$$

Wir suchen wieder ein n für das $P(\text{alle unterschiedlich}) \approx 1/2$, d.h.

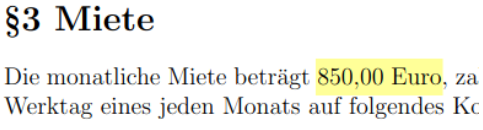
$$\begin{aligned}
\exp\left(-\frac{1}{2^m} \cdot \frac{n(n-1)}{2}\right) &= \frac{1}{2} \\
-\frac{1}{2^m} \cdot \frac{n(n-1)}{2} &= \ln\frac{1}{2} = -\ln 2 \\
\frac{n(n-1)}{2^m \cdot 2} &= \ln 2 \\
n(n-1) &= 2^m \cdot 2 \cdot \ln 2
\end{aligned}$$

Für große n ist $n(n-1) \approx n^2$

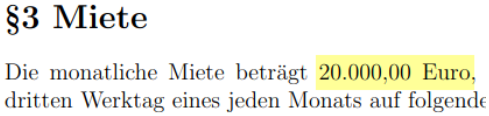
$$\begin{aligned}
n^2 &\approx 2^m \cdot 2 \cdot \ln 2 \\
n &\approx \sqrt{2^m \cdot 2 \cdot \ln 2} = \sqrt{2^m} \cdot \sqrt{2\ln 2} \\
n &\approx 2^{m/2} \cdot \sqrt{2 \cdot \ln 2} = 2^{m/2} \cdot 1.1774\ldots \\
n &\approx 2^{m/2}
\end{aligned}$$

Aufgabe 23. *Führen sie eine Geburtstagsattacke (beschrieben auf S. 87 der VO-Slides) durch: Erstellen sie zwei semantisch unterschiedliche Dokumente (wie im besprochenen Beispiel die beiden unterschiedlichen Mietverträge) im Format ihrer Wahl (Word, Postscript, etc.), und modifizieren sie beide Varianten Semantik-erhaltend automatisiert (beschreiben sie das detailliert, wie sie dabei vorgehen), bis sie auf zwei Varianten mit dem gleichen hash-Wert treffen (verwenden sie dafür die eigentlich obsolete Hashfunktion MD5).*

Die Idee einer Geburtstagsattacke besteht darin, zwei inhaltlich unterschiedliche Dokumente zu erzeugen, die denselben Hashwert besitzen, sodass eine Manipulation unentdeckt bleibt.Als Demonstrationsbeispiel dienen zwei Mietverträge im PDF-Format. Das erste Dokument ist ein regulärer Vertrag mit einer monatlichen Mietzahlung von 850,00 €. Das zweite Dokument unterscheidet sich lediglich in einem zentralen Punkt: Die monatliche Miete beträgt hier 20.000,00 €. Abgesehen von dieser Änderung sind beide Dokumente



(a) Originales Dokument



(b) Verändertes Dokument

Abbildung 1: Ausschnitte aus den Mietverträgen mit Mietzins

identisch aufgebaut. Ziel ist es, die beiden PDF-Dateien semantisch konsistent zu halten, während im Hintergrund so modifiziert wird, dass beide Dateien denselben MD5-Hashwert aufweisen.

Technische Umsetzung Die MD5-Hashfunktion (Message-Digest Algorithm 5) berechnet aus einer beliebigen Eingabe einen 128-Bit langen Hashwert. Aufgrund des enormen Wertebereichs von 2^{128} wäre eine Kollision durch bloßes Probieren sehr aufwendig. Daher wird in dieser Demonstration auf eine vereinfachte Variante zurückgegriffen (Short-MD5), bei der lediglich ein 4 Byte (32 Bit) Präfix der echten MD5-Hashwerte verglichen wird. Zur Manipulation der Dateien wird deren Binärstruktur direkt verändert. PDF-Dateien beginnen mit einem Header, dem in der Regel Zeilenkommentare folgen können, die keine Auswirkungen auf den Inhalt des Dokuments haben. Diese Kommentare werden genutzt, um gezielt die Bytefolge zu beeinflussen:

```
%PDF-1.7
% XXXX <- Zeilenkommentar
...
```

An dieser Stelle wird in beiden Dokumenten ein identischer Kommentarblock eingefügt. Durch automatisierte, systematische Änderungen dieses Kommentarinhalts werden unter-

schiedliche Bytefolgen erzeugt. Nach jeder Änderung wird der Hashwert berechnet und mit den bisherigen Einträgen verglichen. Sobald eine identische 32-Bit-Kennung bei beiden Dokumenten festgestellt wird, wurde eine Kollision entdeckt.

Code Für die Automatisierung wurde die Programmiersprache Rust verwendet. Zur Berechnung des verkürzten MD5-Hashs dient die Funktion `short_md5`. Dabei wird lediglich der vollständige MD5-Hash der Daten errechnet und auf die vordersten vier Bytes reduziert.

```
fn short_md5<T: AsRef<[u8]>>(data: T) -> [u8; 4] {
    let digest = md5::compute(data).0; // 16 Bytes
    digest[..4].try_into().unwrap()    // 4 Bytes
}
```

Für die Kollisionssuche werden die zwei Dokumente A und B zuerst aus dem Speicher geladen und als Array von Bytes gespeichert.

```
let mut a = fs::read("./documents/mietvertrag_original.pdf")?;
let mut b = fs::read("./documents/mietvertrag_changed.pdf")?;
```

Zur Speicherung der bereits entdeckten Hashwerte wird eine HashMap verwendet, mit den 4 Byte Hashes als Key und den dazugehörigen Änderungen des Dokuments als Values.

```
let mut a_map: HashMap<[u8; 4], u32> = HashMap::new();
let mut b_map: HashMap<[u8; 4], u32> = HashMap::new();
```

Innerhalb der PDF-Dokumente wurde wie erwähnt ein Zeilenkommentar mit 'XXXX' als Platzhalter eingefügt. Die vier Zeichen entsprechen genau vier Bytes und somit einem `unsigned integer`. Das erste 'X' ist in diesem Fall das elfte Byte der beiden Dokumente. Um alle möglichen Varianten systematisch durchzugehen, wird von $0 \dots 2^{32} - 1$ iteriert und die binäre Darstellung der Dezimalzahl statt dem Platzhalter in beide Dokumente eingefügt.

```
for i in 0..u32::MAX {
    a[11..11 + 4].copy_from_slice(&i.to_be_bytes());
    b[11..11 + 4].copy_from_slice(&i.to_be_bytes());
    // ...
}
```

Nach der Änderung des Platzhalters wird der Short-MD5 Hash der beiden Dokumente berechnet.

```
let a_digest = short_md5(&a);
let b_digest = short_md5(&b);
```

Jetzt wird in den HashMaps nach den neuen Hashwerten gesucht. Falls einer der Werte bereits in der HashMap des anderen Dokuments ist, so wurde eine Kollision entdeckt. Dann kann das Dokument mit dem Wert, der den gleichen Hash erzeugt hat, aus der HashMap verändert werden. Sei beispielweise der neue Hashwert von Dokument B bereits in der HashMap von Dokument A (selbiges in die andere Richtung):

```
if let Some(modification) = a_map.get(&b_digest) {
    println!("Found Collision! {:?}", modification);
    a[11..11 + 4].copy_from_slice(&modification.to_be_bytes());
    break;
}
```

Ergebnis Durch den Algorithmus konnten die beiden Dokumente so angepasst werden, dass der gleiche Short-MD5 Hashwert nach ungefähr 87000 Iterationen ($\approx 20,5$ ms) entsteht.

	(Short-MD5)	(MD5-Rest)
Original:	7bd80b59	fa50ae16acf3770f74bd6a69
Changed:	7bd80b59	db7b3913e7de464f993a2db9

(a) Originales Dokument mit angepassten Platzhalter

(b) Verändertes Dokument mit angepassten Platzhalter

Abbildung 2: Ausschnitte aus der Textrepräsentation der beiden veränderten Dokumente

Fazit Aufgrund des Geburtstagsparadoxon lassen sich bei der vereinfachten MD5 Variante schnell zwei gleiche Hashwerte für die Dokumente finden. Für die volle Länge von MD5 Hashwerten würde die Suche allerdings beträchtlich länger dauern.