Grundpraktikum Netz- und Datensicherheit

Thema:

MD5-Kollisionen in PostScript

Lehrstuhl für Netz- und Datensicherheit
Ruhr-Universität Bochum

Versuchdurchführung: Raum ID 2/168

Betreuung: Florian Feldmann
Zusammengestellt von: Stefan Hoffmann & Hannes Oberender & Florian Bache & Patrick Meier & Endres Puschner & Dominik Noss
Stand: 3. Dezember 2018
Version: 1.3

1 Einleitung

Der folgende Praktikumsversuch beschäftigt sich mit der (Un-)Sicherheit von digitalen Signaturen, die auf dem Hash-Then-Sign-Paradigma basieren und deren zu Grunde liegende Hashfunktion nicht kollisionsresistent ist.

Hash-Then-Sign-Signaturen besitzen einige Vorteile: Die Signatur ist viel kürzer als die Nachricht, und der Sicherheitsbeweis ist sehr einfach, wenn die Hashfunktion als "random oracle" modelliert wird.

Auf der anderen Seite basiert die Sicherheit des Signaturschemas nun auf der Sicherheit eines zusätzlichen Primitivs.

Der Angriff in diesem Versuch erfolgt nicht auf die eigentliche Signaturfunktion, sondern auf die Hashfunktion. Da die Signatur auf den Hashwert angewendet wird, bedeutet ein gleicher Hashwert zwangsläufig auch eine gleiche Signatur. Das "Opfer" dieses Angriffs muss nur dazu bewogen werden, ein harmloses Dokument zu unterzeichnen. Die Signatur ist dann auch für das präparierte Dokument des Angreifers gültig.

Hashfunktionen zeichnen sich durch drei Eigenschaften aus: 1. "preimage resistance", 2. "second preimage resistance" und 3. Kollisionsresistenz. Die Kollisionsresistenz hat zwar von Natur aus eine deutliche geringere Komplexität als die Punkte 1 und 2 (Geburtstagsparadoxon), ist jedoch auf den ersten Blick wenig nützlich: anders als bei den beiden erstgenannten Punkten sind die Kollisionswerte quasi zufällig.

In [5] wurde ein Weg beschrieben, mit Hilfe von zufälligen Kollisionen die Sicherheit von digitalen Signaturen im elektronischen Geschäftsverkehr ernsthaft zu gefährden.

Die zentrale Idee ist, sich bedingter Darstellungsbefehle in Inhaltsbeschreibungssprachen wie PostScript, PDF, Word oder TIFF zu bedienen. Im Folgenden soll dieser Angriff anhand des Beispiels PostScript aufgezeigt werden.

Hierbei existieren Befehle, mit denen anhand des Vergleichs von Zeichenketten unterschiedliche Inhalte ausgegeben werden können. Da die verglichenen Zeichenketten beliebig sein dürfen, können hier die gefundenen Kollisionen eingesetzt werden. Sie dienen dabei als eine Art Schalter, mit dem zwischen den beiden anzuzeigenden Inhalten umgeschaltet werden kann.

Beim Angriff werden zwei fast identische Dokumente erzeugt, lediglich der Schalter ist jeweils unterschiedlich eingestellt.

Mit einem der beiden Dokumente muss das "Opfer" des Angriffs einverstanden sein und es signieren.

Der zweite Inhalt kann dann beliebig lauten. Sind die Kollisionen in den Dokumenten geschickt platziert, so haben beide den gleichen Hashwert und damit auch die gleiche Signatur. Eine Grundvoraussetzung für den Angriff ist natürlich, dass das Opfer sich das Dokument vor dem Signieren nicht im Quelltext anschaut, da es sonst den verborgenen, falschen Inhalt entdecken würde.

2 Hash-Funktionen

2.1 Allgemein

Eine Hashfunktion ist eine Abbildung, die einen Eingangswert beliebiger Länge (Nachricht m) auf einen Ausgangswert mit fester Länge (Hashwert h) abbildet (siehe Abbildung 1). Hashfunktionen sind Einwegfunktionen: Bei einer guten Hashfunktion ist es praktisch unmöglich, zu einem Ausgangswert einen "originalen" Eingangswert zu finden.

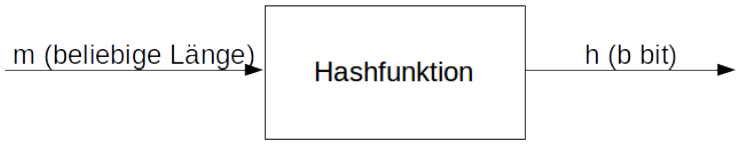

Abbildung 1: Hashfunktion

$$f : \{0,1\}^* \rightarrow \{0,1\}^n$$

Hashwerte werden auch als "digitale Fingerabdrücke" (engl. "fingerprint") bezeichnet, da sie eine Nachricht identifizieren können. Wenn sich nur ein Bit der Nachricht ändert, so sieht der Hashwert der veränderten Nachricht mit hoher Wahrscheinlichkeit völlig anders aus als der Hashwert der ursprünglichen Nachricht.

Mit Hashfunktionen kann beispielsweise beim Versand einer Datei die Korrektheit dieser Datei auf Empfängerseite überprüft werden. Dies geschieht dadurch, dass neben der eigentlichen Datei auch der Hashwert derselbigen übertragen wird.

Der Empfänger kann nach Erhalt der Datei selbst den Hashwert der erhaltenen Datei ausrechnen und überprüfen, ob der errechnete Wert dem übertragenen entspricht. Ist dies der Fall, so ist kein Übertragungsfehler aufgetreten.

Eine weitere mögliche Anwendung liegt in der Kryptografie bei der digitalen Signatur. Hierbei wird von einem Dokument, welches signiert werden soll, zunächst der Hashwert bestimmt. Anschließend wird der Hashwert unter Anwendung des privaten Schlüssels signiert. Der Empfänger erhält das Originaldokument und die Signatur.

Um die Signatur zu überprüfen, kann der Empfänger zunächst den Hashwert bestimmen. Anschließend wendet er den öffentlichen Schlüssel auf die Signatur an. Den hierdurch erhaltenen Wert muss er anschließend auf Gleichheit mit dem berechneten Hashwert prüfen.

Das Unterzeichnen des Hashwertes anstelle der Originalnachricht bietet viele Vorteile. Zum einen ist es effizienter, da das asymmetrische Verfahren nur auf den kleinen Hashwert angewendet werden muss. Zum anderen erhält man eine kürzere Signatur.

2.2 Kollisionen

Eine sehr wichtige Eigenschaft von Hashfunktionen ist ihre Kollisionsresistenz.

Eine Kollision tritt genau dann auf, wenn für zwei unterschiedliche Eingaben m_1 und m_2 der gleiche Hashwert $h = f(m_1) = f(m_2)$ berechnet wurde (siehe Abbildung 2).

Eine völlige Vermeidung von Kollisionen ist bei den meisten Hashfunktionen nicht zu erreichen, da die Länge des Hashwertes und somit auch die Menge aller möglichen Hashwerte begrenzt ist. Die Menge aller Eingabewerte ist jedoch unbegrenzt. Kollisionen treten spätestens dann auf, wenn die Anzahl der

unterschiedlichen Nachrichten die maximale Anzahl der Hashwerte übersteigt (siehe "Schubfachprinzip").

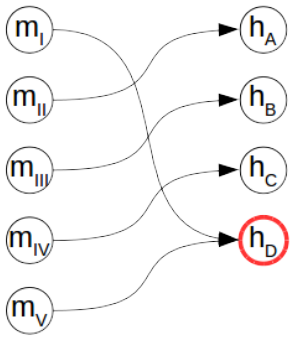

Abbildung 2: Kollision der Nachrichten m_I und m_V

2.3 Anforderungen

Eine gute Hashfunktion erfüllt drei besonders wichtige Eigenschaften.

1. Einwegfunktion ("preimage resistance")

   Es soll praktisch unmöglich sein, zu einem Hashwert h eine mögliche Ursprungsnachricht m zu berechnen.

   Komplexität: $C \approx 2^n$

2. Schwache Kollisionsresistenz ("second preimage resistance")

   Es soll ebenso schwer sein, zu einem gegebenen Eingangswert m einen Wert m' zu finden, der zwar verschieden zu m ist, aber auf den gleichen Hashwert h führt. Es handelt sich dabei um das Herbeiführen einer gezielten Kollision, da der gesuchte Hashwert durch die Originalnachricht vorgegeben ist.

   Komplexität: $C \approx 2^n$

3. Starke Kollisionsresistenz

   Es soll schwierig sein, zwei Eingangswerte m_1 und m_2 zu finden, die auf den gleichen Hashwert h führen.

   Der Unterschied zur schwachen Kollisionsresistenz besteht darin, dass der Hashwert beliebig sein darf und nicht dem einer vorgegebenen Nachricht entsprechen muss. Die Berechnungskomplexität ist dementsprechend deutlich niedriger. Ihre Ermittlung entspricht dem Geburtstagsproblem. Mit einer Wahrscheinlichkeit von 50% hat man bereits nach etwa $\sqrt{2^n} = 2^{\frac{n}{2}}$ Versuchen eine Kollision gefunden.

   Komplexität: $C \approx 2^{\frac{n}{2}}$

Fazit:

Die in der Praxis anfälligste Anforderung an Hashfunktionen ist offensichtlich die der starken Kollisionsresistenz. Hierauf konzentrieren sich die meisten wissenschaftlichen Arbeiten zu Angriffen auf Hashfunktionen. Für einige Hashfunktionen ist es in den letzten Jahren gelungen, effiziente Algorithmen anzugeben, die in Sekundenschnelle Kollisionen berechnen können.

Eine Möglichkeit, dies für einen praktischen Angriff auszunutzen, bietet die Seitenbeschreibungssprache "PostScript".

2.4 MD4-Familie

Die MD4-Familie (MD $\widehat{=}$ "message digest") ist eine Familie verbreiteter Hashalgorithmen. Sie sind in unterschiedlichen Sicherheitssystemen im Einsatz und sind Teil vieler internationaler Standards.

Die Algorithmen dieser Familie stimmen z. B. dahingehend überein, dass sie mit einem bestimmten Startwert initialisiert werden und den Eingangswert nach einem bestimmten Grundprinzip verarbeiten, welches im Unterpunkt zu MD5 näher beschrieben wird.

Zur MD4-Familie gehören die verbreiteten Hashfunktionen MD4 [6], MD5 [7], SHA-0, SHA- 1 [3], RIPEMD-160 [2] und die SHA-2-Gruppe.

2.4.1 MD5-Algorithmus

Nach einem geeigneten Padding verarbeitet der MD5-Algorithmus jeden einzelnen 512-Bit-Block der Nachricht iterativ mithilfe einer sogenannten Kompressionsfunktion f.

$$f : \{0,1\}^{128} \times \{0,1\}^{512} \to \{0,1\}^{128}$$

Als Eingangswert dient neben dem 512-Bit-Block ein Puffer C_i mit einer Größe von 128 Bit, der für den ersten Block mit einem Standardwert IV belegt ist (Initialisierungsvektor).

Die Kompressionsfunktion f wird iterativ auf jeden Block der Nachricht angewendet. Aufeinanderfolgende Berechnungen werden verknüpft, indem der Ausgangswert der vorigen Berechnung als Eingangswert für den nächsten Funktionsaufruf verwendet wird.

Wenn alle Blöcke abgearbeitet worden sind, wird der letzte Wert des Puffers C_k als Hashwert verwendet.

Abbildung 3 beschreibt die Verarbeitung einer Nachricht mit fünf Blöcken.

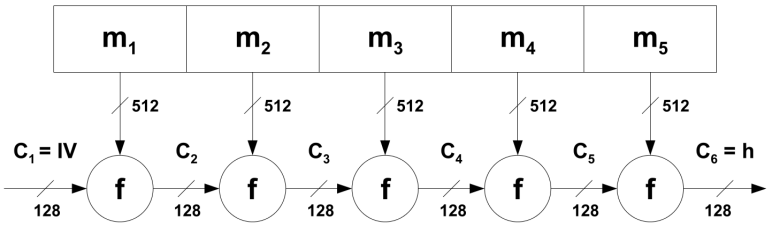

Abbildung 3: MD5-Prinzip

Der Kollisionsangriff, der in diesem Versuch nachvollzogen werden soll, wird mit dem MD5-Algorithmus durchgeführt. Der Grund dafür ist, dass für diese Funktion Algorithmen bekannt sind, die strukturelle Schwächen ausnutzen und die eine zufällige Kollision in wenigen Sekunden auf einem Desktop-PC finden. Die starke Kollisionsresistenz ist damit für den MD5-Algorithmus gebrochen, die Komplexität, Kollisionen zu finden, liegt unterhalb der Schranke des Geburtstagsproblems.

2.4.2 MD5-Padding

Bevor eine Nachricht vom MD5-Algorithmus verarbeitet wird, werden zuvor bestimmte Daten angehängt ("Padding"). Ihre Länge ist nach diesem Schritt ein Vielfaches von 512 Bit.

Als erstes wird ein Bit mit dem Wert 1 hinzugefügt. Danach werden genausoviele Null-Bits angehängt, dass noch acht Bytes bis zum nächsten Blockende fehlen. Abschließend wird in diese acht Bytes die Länge der ursprünglichen Nachricht (ohne Padding) eingefügt (siehe Abbildung 4).

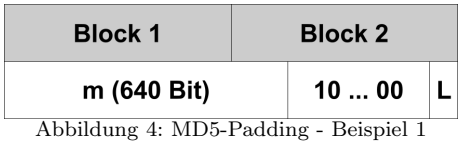

Abbildung 4: MD5-Padding - Beispiel 1

Unter Umständen muss beim Anhängen der Zusatzinformationen ein Block zusätzlich hinzugefügt werden, wie in Abbildung 5 zu sehen ist.

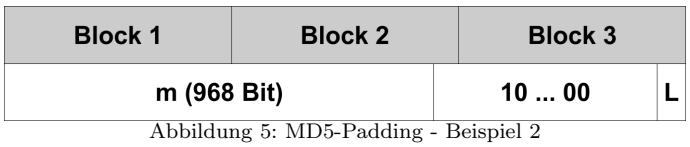

Abbildung 5: MD5-Padding - Beispiel 2

Das Padding wird auch dann durchgeführt, wenn die ursprüngliche Nachricht bereits ein ganzzahliges Vielfaches der Blockgröße ist. Ein Beispiel dafür ist in Abbildung 6 dargestellt.

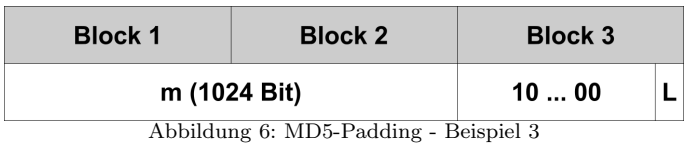

Abbildung 6: MD5-Padding - Beispiel 3

2.4.3 Kollisionsfindungsalgorithmen in MD5

Für MD5 sind Algorithmen veröffentlicht worden, die sehr effizient in wenigen Sekunden Kollisionen auf einem Desktop-PC berechnen können.

Der hier zu verwendende Algorithmus [4] benötigt einen "Init Vector" als Eingabeparameter. Dieser Wert entspricht dem Pufferwert C_i, der an der Stelle vorliegt, an der die Kollision eingesetzt werden soll. Wird kein Eingabeparameter verwendet, so wird der Standard-IV von MD5 benutzt.

Die gefundenen Kollisionen sind also abhängig vom Wert C_i. Dies bedeutet z. B., dass Kollisionen, die für den Start-IV gefunden wurden, nur am Anfang einer Nachricht einsetzbar sind. An einer anderen Stelle in der Nachricht und mit einem anderen C_i funktionieren sie im Allgemeinen nicht.

Falls Kollisionen gesucht werden, die an einer bestimmten Stelle in der Nachricht Wirkung finden, so muss man den dort vorliegenden Pufferwert C_i bestimmen, um ihn als Eingabewert in die Software eingeben zu können.

Um den an einer bestimmten Stelle vorliegenden Wert C_i zu bestimmen, muss der Hashwert der Nachricht bis zu dieser Stelle berechnet werden. Es ist allerdings zu beachten, dass es falsch wäre, den Hashwert dieses Nachrichtenteils mit dem Standard-MD5-Algorithmus zu berechnen, da hierbei automatisch das Padding durchgeführt wird. Da es sich jedoch nur um einen Nachrichtenauszug handelt und das Padding nicht schon mitten in der Nachricht ausgeführt werden soll, muss es in diesem Falle unterbunden werden. Andernfalls würde ein falscher Wert für C_i ermittelt.

Der Algorithmus muss für diese spezielle Berechnung also so abgeändert werden, dass das Padding nicht stattfindet.

2.4.4 Ausgabeformat von MD5

Der errechnete Hashwert wird in einem bestimmten Format angezeigt.

Es handelt sich um die "Little-Endian-First-Byte-Order", bei der das niederwertigste Byte zuerst und das höchstwertigste Byte als letztes ausgegeben wird.

Das Format für die Eingabe eines "Init Vector" als Eingabeparameter in das Kollisionsfindungsprogramm ist genau byteweise umgekehrt und folgt somit der Big-Endian-Darstellung über je 32-Bits.

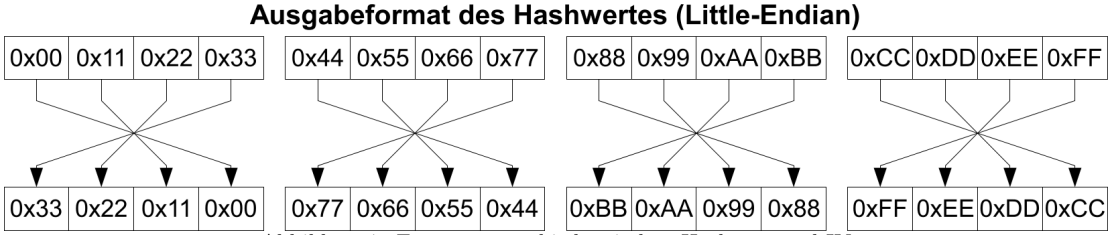

Abbildung 7: Formatunterschied zwischen Hashwert und IV

Diese Tatsache ist von Bedeutung, wenn der ausgegebene Wert als "Init Vector"- Parameter benutzt werden soll. Er muss zuvor in das entsprechende Format gebracht werden.

Die Umwandlung zwischen beiden Formaten kann Abbildung 7 entnommen werden.

Da diese Unwandlung per Hand sehr fehleranfällig ist, steht Ihnen das Programm "hash_zu_IV.sh" zur Verfügung, welches einen MD5-Hash als Parameter übernimmt und einen IV ausgibt:

```
$ ./hash_zu_IV.sh ed1ca3af1f27699933fbcb0a7d0edd94
0xafa31ced 0x9969271f 0x0acbfb33 0x94dd0e7d
```

3 PostScript-Format

3.1 Allgemein

PostScript ist eine Programmiersprache zur Seitenbeschreibung.

Mittels PostScript kann man auf unterschiedlichen Geräten Ausgaben in beliebiger Größe und Auflösung verlustfrei erzeugen. Elemente und Schriften werden dabei als skalierbare Vektorgrafik beschrieben.

PostScript ist stapelorientiert und arbeitet nach dem Prinzip der umgekehrten polnischen Notation. Letztere zeichnet sich dadurch aus, dass in der Regel zunächst die Parameter angegeben werden und erst danach die eigentliche Funktion, die auf die angegebenen Werte anzuwenden ist.

3.2 Einfache Ausgabe

Ein Beispiel zur Anzeige eines einfachen Textes im Dokument sieht folgendermaßen aus:

```
%!
/Courier findfont    % Schrift auswählen
20 scalefont         % auf Schriftgröße 20 setzen
setfont              % aktuellen Zeichensatz setzen
50 50 moveto         % Koordinaten (50,50) als Schreibposition setzen
(Hallo Welt!) show   % eingeklammerten Text anzeigen
showpage             % Seite ausgeben
```

Hierbei geht das Programm an die Stelle mit den Koordinaten (50,50) und gibt dort den Text "Hallo Welt!" aus.

Der **Koordinatenursprung liegt dabei an der linken unteren Ecke** der Seite.

3.3 If-Else-Anweisung

Will man eine Ausgabe erstellen, die abhängig von bestimmten Werten ist, so kann der folgende Befehl verwendet werden:

```
%!
/Courier findfont
20 scalefont
setfont
(V1) (V2) eq
{
  50 600 moveto
  (Hallo Welt!) show
  showpage
}
{
  200 200 moveto
  (Die Parameter sind UNGLEICH!) show
  showpage
}
ifelse
```

Hierbei werden zwei vom Programmablauf unabhängige Konstanten verglichen.

Sind die Konstanten gleich, so wird die gleiche Ausgabe angezeigt wie in Abschnitt 3.2.

Wurde der Quelltext dagegen so geändert, dass die Konstanten ungleich sind, so erscheint die Ausgabe "Die Parameter sind UNGLEICH!".

3.4 Angriff unter Verwendung von Hash-Kollisionen [5]

Im vorigen Beispiel ist zu sehen, dass eine geringe Manipulation an einer PostScript-Datei zur Ausgabe eines komplett anderen Inhalts führen kann.

Jedoch hat aufgrund der Anforderungen an Hash-Funktionen bereits eine geringe Änderung der Eingabenachricht einen völlig anderen Hashwert zur Folge. Dies trifft auch hier zu: Die Hashwerte der Originaldatei und der manipulierten Datei sind unterschiedlich.

Es ist aber wie gesehen effizient möglich, zwei beliebige unterschiedliche Nachrichten zu finden, die auf den gleichen Hashwert führen. Die Tatsache, dass die zu vergleichenden Konstanten im PostScript-Dokument beliebige Werte haben dürfen und keinen Sinn ergeben müssen, kann an dieser Stelle für einen Angriff ausgenutzt werden. Die beiden Konstanten wirken dann als eine Art "Schalter".

Hierzu wird die PostScript-Datei mit dem Schalter einmal kopiert. Die erste Variante enthält die zwei unterschiedlichen kollidierenden Eingabewerte als zu vergleichende Konstanten. Variante 2 enthält nur eine der beiden Kollisionen in doppelter Ausführung, siehe Abbildung 8. Damit unterscheiden sich beide Dateien nur im ersten Parameter der EQ-Funktion.

Da vor der eigentlichen Kollision eine Art "Dateiheader" steht, muss eine Kollision für den Wert C_i gefunden werden, der an dieser Stelle auftritt, im Beispiel C_4.

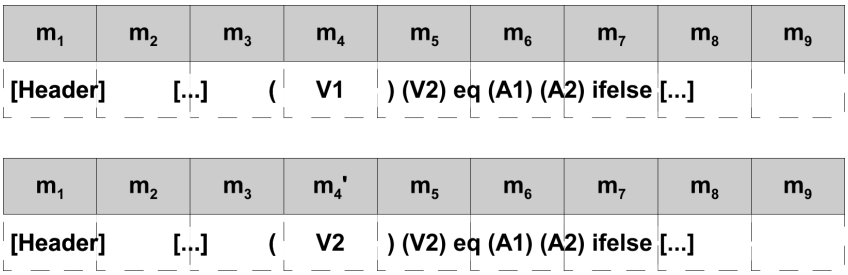

Abbildung 8: Kollisionsanwendung beim PostScript-Format

In Abbildung 8 sind die zwei Dateien in Eingabeblöcke der Hashfunktion unterteilt. Die Werte V_1 und V_2 entsprechen den gefundenen kollidierenden Nachrichten, die Werte A_1 und A_2 sind die Anweisungen zur Ausgabe der jeweiligen Inhalte.

Die Anweisungen A_1 beschreiben den Inhalt, den das Opfer des Angriffs akzeptieren und signieren wird. Die Anweisungen A_2 hingegen führen zum Inhalt, der dem Opfer untergeschoben wird.

4 Versuch

4.1 Durchführung

Zur Versuchsdurchführung stehen Ihnen die in Tabelle 1 genannten Programme zur Verfügung. Das Praktikumsverzeichnis lautet:

`/home/student/Praktikum/md5`

	Linux
PS-Viewer	**Document Viewer**
standardmäßig installiertes MD5 von Linux	**md5sum**
Hex-Viewer	**bless**
modifiziertes MD5	**md5.c**
MD5-Koll.-Finder	**md5_vk1_single.c**
PS-Überweisungsdatei	**ueberweisung.ps**
Konvertierung eines Hashs in einen IV	**hash_to_IV.sh**
Zusammensetzen der Teildateien inkl. Überprüfung auf Fehler	**collision_test.sh**

Tabelle 1: Verfügbare Programme

Ziel des Versuchs ist, die Vorgabe für jede Aufgabe der Versuchsdurchführung Schritt für Schritt zu erfüllen und jedes Zwischenergebnis zu dokumentieren. Anhand der Zwischenergebnisse sollte eine schriftliche Versuchsauswertung verfasst werden.

Lesen Sie zunächst die theoretische Einleitung dieses Dokuments durch und nutzen Sie die Kontrollfragen als Verständniskontrolle für den beschriebenen Inhalt.

4.2 Szenario

Oskar bereitet eine Bank-Überweisung von Bobs auf Alices Konto vor, doch plant er das Geld zu stehlen.

Überweisungen werden hier als PostScript-Dateien verfasst. Dort stehen Kontodaten der beiden Beteiligten und der Betrag im Klartext. Um die Datei vor Manipulation zu schützen, signiert Bob den MD5-Hash der Datei mittels digitaler Signaturen[1], um der Bank zu beweisen, dass der Überweisungs-Auftrag von ihm stammt. Die Bank überprüft die Signatur und führt die Überweisung bei Richtigkeit aus.

Der Wert der Signatur hängt nur vom Hash der Datei ab. Existieren zwei Dateien mit identischem Hash, so ist auch deren Signatur identisch.
Oskar plant nun, zwei PostScript-Überweisungs-Dateien mit gleichem MD5-Hash zu erstellen. Die erste Variante enthält Alice als Begünstigte und wird Bob vorgelegt. Dieser öffnet die Datei, vergewissert sich, dass die Daten stimmen, und signiert den Hash.
Die zweite Variante enthält Oskar als Begünstigten. Nachdem Bob die erste Variante signiert hat, tauscht Oskar die Dateien aus und schickt Variante Zwei samt Signatur zur Bank.
Da die Dateien den gleichen Hash haben, wird die Signatur für beide Varianten verifizierbar sein und deshalb ausgeführt werden.

[1] Das Signieren ist nicht teil des Versuches

4.3 Aufgaben

4.3.1 Anzeigen der Datei

Öffnen Sie die Datei "ueberweisung.ps" im Document Viewer und machen Sie einen Screenshot der Überweisung.

4.3.2 Erstellung der zweiten, versteckten Überweisung

Öffnen Sie "ueberweisung.ps" mit einem Text-Editor und füllen Sie die Definition von "dokument2" mit einer Kopie von "dokument1", in welcher Oskar[2] als Begünstigter (Empfänger des Geldes) steht. Ersetzen Sie dann den Aufruf von "dokument1" am Ende der Datei durch "dokument2". Beim Öffnen im Document Viewer sollte nun die Überweisung mit Oskars Daten angezeigt werden.

4.3.3 Einbau des Schalters zum Auswechseln der Überweisungen

Sie haben nun eine Überweisung, welche durch geringfügige Änderung (Aufruf von dokument1/2) unterschiedliche Empfänger anzeigt. Implementieren Sie eine IF-Verzweigung am Ende der Datei, die mittels zweier Konstanten zwischen zwei Dokumentanzeigen umschalten kann. Benutzen Sie als Konstanten zunächst die Bezeichner $V1$ und $V2$. Die beiden Konstanten werden später durch die Kollisionsblöcke ersetzt.

Benutzen Sie dafür den unten stehenden PostScript-Code.

```
(V1)
(V2)
eq
{ dokument2 }
{ dokument1 }
ifelse
```

Im Document Viewer sollte nun die an Alice gerichtete Überweisung angezeigt werden. Wenn Sie "V1" durch "V2" ersetzen, sollte die an Oskar gerichtete Überweisung angezeigt werden. Überprüfen Sie dies.

4.3.4 Zerteilen der PostScript-Datei

Da es in diesem Versuch sehr auf Genauigkeit ankommt und einige Tätigkeiten wiederholt ausgeführt werden müssen, wurden die Kopier-Operationen, welche später die einzelnen Teilstücke zusammensetzen, automatisiert.

Dafür müssen Sie die Datei nun in mehrere Teile aufteilen und diese in den Ordner "components" speichern. **Benutzen Sie hierfür einen Hex-Editor, da Text-Editoren beim Speichern häufig unerwünscht Zeilenumbrüche an das Ende einer Datei anfügen.**

Am Ende sollen die Dateien wie folt aussehen:

- Der Anfang der Überweisung ist in "components/start" zu speichern. Der Anfang geht bis zu der Stelle, an welcher der Kollisionsblock eingefügt werden soll. Die Länge der Datei muss ein Vielfaches der MD5-Blockgröße sein und muss deshalb mit Leerzeilen aufgefüllt werden.

[2] Nachname und Bankdaten dürfen kreativ gewählt werden.

- Der Mittelteil ist der Datei-Inhalt zwischen V1 und V2. In der Regel enthält er die Zeichenfolge ")\n(", welche bereits in der Datei "components/center" eingetragen ist.[3]
- Der Endteil in "components/end" beginnt nach V2, dem zweiten Kollisions-Block. Er beginnt mit der schließenden Klammer und ist in der Länge ebenso unbeschränkt wie der Mittelteil.

Weiterhin existieren die Dateien "collision_block1" und "collision_block2" im Ordner "components". Sie werden später mit den Kollisions-Blöcken gefüllt.

Um dies zu erreichen müssen sie als Erstesden Anfangsteil extrahieren.
Verschieben Sie die Zeichen innerhalb der Datei so, dass der erste der beiden Bezeichner "V1" am Anfang eines 512-Bit-Blocks steht. In anderen Worten: Das "V" des "V1" soll das erste Byte eines 64 Byte (512 Bit) Blocks sein. Die öffnende Klammer "(" vor dem "V" soll das letzte Byte des vorhergehenden Blocks sein. Sie können dafür z.B. Leerzeichen oder Zeilenumbrüche an geeigneter Stelle einfügen. Leerzeichen haben sich in der Vergangenheit als problematisch erwiesen. Entfernen Sie nun die Daten beginnend am "V" von "V1" bis zum Ende und speichern Sie das Resultat in "components/start".

Vergewissern Sie sich nun, dass "components/center" die Daten zwischen V1 und V2 enthält. Falls nicht, berichtigen sie es. Benutzen Sie dafür weiterhin den Hex-Editor.

Nun gilt es den Endteil der Datei zu extrahieren. Löschen Sie dafür alle Daten der Überweisungs-Datei bis zur "2" von "V2", sodass das Resultat mit der schließenden Klammer beginnt. Speichern Sie dies in "components/end".

4.3.5 Kompilierung des IV-Finders

Programmieren Sie einen IV-Finder zur Berechnung des Wertes C_i: Verwenden Sie dazu den vorgegebenen Quelltext in MD5.c und entfernen Sie die Paddingfunktionalität durch geeignetes Auskommentieren. Das Padding wird in der C-Prozedur "MD5Final" durchgeführt.

Zum kompilieren der Dateien benutzen Sie die vorhandene Make-Datei: Öffnen Sie eine Shell und wechseln Sie in das Verzeichnis "Praktikum/md5". Tippen Sie nun "make" ein und drücken Sie Enter.

Es befinden sich nun neue Programmdateien im aktuellen Ordner. Denken Sie daran, dass lokale Programme immer mittels "./" ausgeführt werden müssen. So führen Sie mit "./md5" ihr gerade kompiliertes Programm aus und mit "md5sum" das standardmäßig installierte MD5-Programm.

4.3.6 Finden des IV

Finden Sie mit dem IV-Finder den passenden Wert C_i. Kopieren Sie dazu den Inhalt von "ueberweisung.ps" bis vor dem ersten Bezeichner "V1" in die Datei "Praktikum/md5/components/start". Da "V1" am Anfang eines Blocks steht, hat die Datei eine Größe, die einem ganzzahligen Vielfachen von 512 Bit entspricht.

Benutzen Sie diese Datei in Ihrem IV-Finder-Algorithmus. Der Ausgabewert ist der gesuchte Wert C_i. Beachten Sie das Ausgabeformat **"Little-Endian"**. Hierbei steht ganz links das niederwertigste Byte und ganz rechts das höchstwertigste Byte. Zur Umwandlung können Sie das Programm "./hash_to_IV.sh" benutzen.

Notieren Sie den Wert C_i vor und nach der Endianness-Konvertierung.

[3] mit \n ist das newline-Symbol (0x0A) gemeint.

4.3.7 Finden der Kollision

Nutzen Sie die vorgegebene Software, um mit dem berechneten C_i eine Kollision zu bestimmen. Der Wert C_i dient hierbei als Eingabeparameter "Init Vector" des Programms. Die beiden kollidierenden Nachrichten erhalten Sie in Form von zwei Binärdateien.

Speichern Sie die Kollisions-Blöcke jeweils in "components/collision_block1" und "components/collision_block2".

4.3.8 Zusammenbau

Das Programm "./collision_test.sh" überprüft die formale Korrektheit der einzelnen Dateien im Ordner "components". Bei Richtigkeit erstellt es daraus zwei Überweisungen und überprüft deren MD5-Hash. Finden Sie die 2 Zeilen in "collision_test.sh", welche für die Erstellung der beiden Dateien verantwortlich ist.
Führen Sie das Programm nun aus. Sollte es nicht erfolgreich beenden, beheben Sie die angezeigten Fehler.

Leider bedeutet das erfolgreiche Finden einer Kollision nicht, dass die resultierenden Dateien valides PostScript enthalten. Die Kollisions-Blöcke dürfen keine runden Klammern enthalten und Zeilenumbrüche darin führen oft zu invaliden IF/ELSE Konstrukten.
Prüfen Sie weiterhin mittels eines PostScript-Viewers, ob die beiden Dateien "Version11.ps" und "Version21.ps" im Ordner "components" angezeigt werden und ob sie unterschiedliche Ausgaben verursachen.

Sollte das Anzeigen fehlschlagen (z.B. endloses Laden), müssen Sie neue Kollisionsblöcke erzeugen und den Vorgang wiederholen. Da das Ergebnis vom Zufall abhängt, ist schwierig vorauszusehen, wie häufig Sie es wiederholen müssen. Versuchen Sie es maximal zehn Mal und wenden Sie sich bei Misserfolg an ihren Betreuer.

Hinweis: Über die Pfeiltasten nach oben und unten können Sie in der Konsole vergangene Befehle durchblättern und wieder verwerten. Damit sollte das Finden geeigneter Blöcke mit wenig Aufwand möglich sein.

5 Kontrollfragen

- Was ist eine Hashfunktion? Wobei findet sie Anwendung?
- Erklären Sie, weshalb es praktisch deutlich einfacher ist, Kollisionen für beliebige Hashwerte zu finden, als für vorgegebene.
- Wieso sind Angriffe auf Hashfunktionen für die Sicherheit von Signaturverfahren von Bedeutung?
- Welche Algorithmen gehören zur MD4-Familie?
- Erklären Sie den Aufbau des MD5-Algorithmus.
- Erklären Sie das Padding beim MD5-Algorithmus. Weshalb ist Padding notwendig?
- Was ist PostScript?
- Weshalb bietet sich PostScript für einen Angriff mittels Kollisionsnachrichten an, die inhaltlich keinen Sinn ergeben?
- Wieso ist es notwendig, die gefundenen Kollisionen genau an einer Blockgrenze einzufügen?
- Wie können präparierte Dokumente erkannt werden?
- Bisher wurden nur Inhaltsbeschreibungssprachen betrachtet. Lassen sich die beschriebenen Techniken auf kompilierte Programme erweitern?

Literatur

[1] Adobe. PS language specifications. http://partners.adobe.com/public/developer/ps/index_specs.html.

[2] Dobbertin, Bosselaers, and Preneel. The hash function RIPEMD-160. http://homes.esat.kuleuven.be/ bosselae/ripemd160.html.

[3] Eastlake. US Secure Hash Algorithm 1, 2001. http://tools.ietf.org/html/rfc3174.

[4] Klima and Dufek. MD5-Collisions - Algorithm from Pavel Dufek: Modification of Klima´s program, 2008. http://cryptography.hyperlink.cz/MD5_collisions.html.

[5] Lucks and Daum. The Story of Alice and her Boss, 2005. http://www.cits.rub.de/imperia/md/content/magnus/rump_ec05.pdf.

[6] Rivest. The MD4 Message-Digest Algorithm, 1992. http://tools.ietf.org/html/rfc1320.

[7] Rivest. The MD5 Message-Digest Algorithm, 1992. http://tools.ietf.org/html/rfc1321.

[8] Wang and Yu. How to break MD5 and other hash functions. *Advances in Cryptology - EUROCRYPT 2005*, pages 19–35, 2005.