Was wir gemacht haben

§0 Einführende Worte (✓)

§1 Geschichtlicher Überblick (✓)

§2 Zufall (✓)

§3 Perfekte Sicherheit und ihre Grenzen (✓)

§4 Angriffsszenarien (✓)

§5 Der komplexitätstheoretische Ansatz (✓)

§6 Pseudozufallsgeneratoren und Stromchiffren (✓)

§7 Pseudozufallsfunktionen und Blockchiffren (✓)

§8 Message Authentication Codes und Hash-Funktionen (✓)

§9 Kryptographie mit öffentlichen Schlüsseln (✓)

Lose Enden

(Ja, das ist eine wörtliche Übersetzung aus dem Englischen, aber “loose ends” kommt sicher – sag ich – aus der niederdeutschen Segelsprache. Ein Ende ist ein Tau.)

Verschlüsseln und Signieren

Beim Verschlüsseln und Signieren (oder Verschlüsseln und mit Nachrichtenauthentisierungscode-Versehen) ist es wirklich wichtig, verschiedene Schlüssel(-paare) zu benutzen.

Zum Zufallsorakelmodell

In der Vorlesung wurde immer Zufallsorakelmodell gesagt, wenn eine Schar von Funktionen durch eine "echt zufällige" Funktion ersetzt wurde.

D.h. $(f_{k_0})_{k_0}$ mit $f : \ldots \longrightarrow \{0,1\}^n$ wird ersetzt durch das Orkakel.

Dieses geht so:

Wenn ein String x eingegeben wird, wird geschaut, ob er schon mal eingegeben worden ist:

- Wenn nein: Es wird eine "echt zufälliges" (uniform zufällig verteiltes) Element in $\{0,1\}^n$ augegeben.
- Wenn ja und die Ausgabe war y: Es wird y ausgegeben.

Anwendung bei

- Beweisen zu Blockchiffren (zweistufige Beweise)
- und bei Hashfunktionen (Beweise teilweise nur im Zufallsorakelmodell)

Zum Zufallsorakelmodell

Normaler Sprachgebrauch. Nur bei der Modellierung von Hashfunktionen wird von Zufallsorakel (random oracle) gesprochen.

Frage. Warum erlaubt die Methode so elegante Argumente und "Resultate", die auf wirklich unsicheren Konstruktionen beruhen?

Beispiel. $\mathcal{M}ac_k(m) := H(k|m)$ ist sicher im Zufallsorakelmodell, aber vollkommen unsicher mit H basierend auf Merkle-Damgard-Konstruktion.

Das Spiel zum CPA

Gegeben: $\mathcal{G}en, \mathcal{E}nc, \mathcal{D}ec$.

Eingabe: Sicherheitsparameter 1^n.

1. Angreifer und Herausforderer erhalten 1^n.
2. Der Herausforderer erzeugt einen Schlüssel $k := \mathcal{G}en(1^n)$.
3. Der Angreifer kann beliebige Nachrichten m mit dem Schlüssel k verschlüsseln lassen (d.h. $\mathcal{E}nc_k(m)$ berechnen lassen).
4. Der Angreifer wählt zwei verschiedene Nachrichten m_1, m_2 gleicher Länge. Er schickt diese an den Herausforderer.
5. Der Herausforderer wählt $i \in \{0, 1\}$ uniform. Er berechnet $c := \mathcal{E}nc_k(m_i)$ und schickt c an den Angreifer.
6. Der Angreifer kann beliebige (!) Nachrichten mit dem Schlüssel k verschlüsseln lassen.
7. Der Angreifer entscheidet sich für $j \in \{0, 1\}$.

Der Angreifer hat gewonnen, wenn $i = j$ ist.

Definition von Hash-Funktionen

Man kann nun eine Hash-Funktion als ein Paar von Algorithmen $\mathcal{G}en, \mathcal{H}$ definieren.

Man kann dann unterscheiden zwischen Hash-Funktionen mit fester Eingabelänge und Hash-Funktionen mit variabler Eingabelänge.

Wenn $\mathcal{G}en$ als kanonisch angenommen wird, kann man es auch weglassen.

Dann kann man auch sagen:

Eine Hash-Funktion ist eine Funktionsfamilie $(H_{k^{(0)}})_{k^{(0)} \in \{0,1\}^*}$ mit:

Für $n := |k^{(0)}|$ ist $H_{k^{(0)}} : ... \longrightarrow \{0,1\}^{\ell(n)}$

und $H_{k^{(0)}}(s^{(0)})$ ist in Polynomzeit berechenbar.

Definition von Hash-Funktionen

Spiel

Eingabe: Sicherheitsparameter 1^n.

1. Angreifer und Herausforderer erhalten 1^n.
2. Der Herausforderer erzeugt einen Schlüssel $k := \mathcal{G}en(1^n)$ und sendet ihn dem Angreifer (!).
3. Der Angreifer gibt zwei Strings $x \neq x'$ aus.

Der Angreifer hat gewonnen, wenn $\mathcal{H}_k(x) = \mathcal{H}_k(x')$ ist. ("Er hat eine Kollision gefunden.")

RSA-Signatur

Für Nachrichten m_1, m_2 gilt

$$\mathcal{S}ign_{sk}(m_1)\,\mathcal{S}ign_{sk}(m_2) = \mathcal{S}ign_{sk}(m_1 m_2)$$

Somit ist das Verfahren auch vollkommen unsicher.

RSA-Verschlüsselung

Neues Signaturverfahren

Zuerst haschen, dann RSA-signieren

Satz. Es sei ein Schlüsselerzeugungsalgorithmus fixiert.

Wenn jeder PPT-Algorithmus für das entsprechende RSA-Problem nur vernachlässigbaren Erfolg hat, dann ist "zuerst-hashen-dann-RSA-signieren" sicher im Zufallsorakelmodell.

Aber ... Falls für die eingesetzte Hash-Funktion $a, b, c \in \mathbf{N}$ mit $a \cdot b \equiv c \mod N$ und

$$H(a) \cdot H(b) = H(c)$$

bekannt sind, ist das Verfahren wieder ganz unsicher.

Diskussion

"Warum" liefert die Zufallsorkelmethode

für Hash-Funktionen so viel elegantere Argumente aber vielleicht praxisuntaugliche Resultate

während sie im Kontext von Pseudozufallsfunktionen nur ein Hilfsmittel ist?

Zum Zweiten: Das spiegelt die Definition von Pseudozufallsfunktion wieder.

Zum Ersten: Nur im Zufallsorakelmodell muss ein Angreifer mit dem Herausforderer interagieren.