11.7 Literaturverzeichnis

1. P. S. L. M. Barreto and V. Rijmen. The Whirlpool Hashing Function, September 2000. (revised May 2003), http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html.
2. B. den Boer and A. Bosselaers. An attack on the last two rounds of MD4. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference, Advances in Cryptology*, LNCS, pages 194–203. Springer, 1992.
3. B. den Boer and A. Bosselaers. Collisions for the compression function of MD5. In *Advances in Cryptology - EUROCRYPT'93*, LNCS, pages 293–304. Springer, 1994.
4. Hans Dobbertin. Alf swindles Ann. *CRYPTOBYTES*, 3(1), 1995.
5. Hans Dobbertin. The status of MD5 after a recent attack. *CRYPTOBYTES*, 2(2), 1996.
6. Federal Information Processing Standards Publications — FIPS PUBS. http://www.itl.nist.gov/fipspubs/index.htm.
7. Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche. The Keccak SHA-3 submission, Version 3, 2011. http://keccak.noekeon.org/Keccak-submission-3.pdf.
8. Shoichi Hirose. Some plausible constructions of double-block-length hash functions. In *FSE: Fast Software Encryption*, volume 4047 of *LNCS*, pages 210–225. Springer, 2006.
9. International Organization for Standardization (ISO). ISO/IEC 10118-4, Information technology—Security techniques—Hash-functions—Part 4: Hash-functions using modular arithmetic, 1998. http://www.iso.org/iso/.
10. National Institute of Standards and Technology. Cryptographic Hash Algorithm Competition. http://csrc.nist.gov/groups/ST/hash/sha-3/index.html.
11. B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. *LNCS*, 773:368–378, 1994.
12. Bart Preneel. MDC-2 and MDC-4. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

Aufgaben

11.1. Berechnen Sie den Ausgangswert nach der ersten Runde von der ersten Stufe von SHA-1 für die beiden Eingangsblöcke der Länge 512 Bit:

1. $x = \{0...00\}$
2. $x = \{0...01\}$ (d. h. Bit 512 ist eine 1).

Für diese Aufgabe nehmen wir an, dass der Anfangswert H_0 nur aus Nullen besteht, d. h. $A_0 = B_0 = ... = 00000000_{16}$.

11.2. Eine frühe Anwendung für Hashfunktionen war das Abspeichern von Passwörtern, die für die Benutzerauthentisierung benutzt wurden. Hierbei wird ein Passwort gehasht, nachdem es eingegeben wurde, und der Wert wird mit dem Hashwert des gespeicherten Referenzpasswortes verglichen. Es wurde schon früh erkannt, dass es ausreicht, nur den Hashwert anstatt des eigentlichen Passwortes abzuspeichern.

1. Nehmen Sie an, Sie wären ein Hacker und Sie haben Zugang zu einer Liste mit gehashten Passwörtern bekommen. Sie würden natürlich gerne die Passwörter

rekonstruieren, um sich als legitimer Benutzer anmelden zu können. Diskutieren Sie, mit welchen der unten stehenden Angriffe dies möglich ist. Beschreiben Sie genau die Konsequenzen Ihrer Angriffe.

- Angriff A: Sie können die Einwegeigenschaft von h brechen.
- Angriff B: Sie können zweite Urbilder für h finden.
- Angriff C: Sie können Kollisionen konstruieren.

2. Warum wird beim Speichern von Passwörtern oft ein sogenanntes Salt benutzt? (Hierbei handelt es sich um einen Wert, der an das Passwort angehängt wird, bevor es gehasht wird. Der Wert des Salt wird zusammen mit dem Hashwert abgespeichert, d. h. ein Angreifer kennt das Salt auch.) Werden die oben stehenden Angriffe durch die Verwendung von Salt beeinflusst?
3. Ist eine Hashfunktion mit einer Ausgangsbreite von 80 Bit ausreichend für diese Anwendung?

11.3. Zeichnen Sie ein Blockdiagramm für die folgenden Hashfunktionen, die mit Blockchiffre $e()$ konstruiert wurden:

1. $e(H_{i-1}, x_i) \oplus x_i$
2. $e(H_{i-1}, x_i \oplus H_{i-1}) \oplus x_i \oplus H_{i-1}$
3. $e(H_{i-1}, x_i) \oplus x_i \oplus H_{i-1}$
4. $e(H_{i-1}, x_i \oplus H_{i-1}) \oplus x_i$
5. $e(x_i, H_{i-1}) \oplus H_{i-1}$
6. $e(x_i, x_i \oplus H_{i-1}) \oplus x_i \oplus H_{i-1}$
7. $e(x_i, H_{i-1}) \oplus x_i \oplus H_{i-1}$
8. $e(x_i, x_i \oplus H_{i-1}) \oplus H_{i-1}$
9. $e(x_i \oplus H_{i-1}, x_i) \oplus x_i$
10. $e(x_i \oplus H_{i-1}, H_{i-1}) \oplus H_{i-1}$
11. $e(x_i \oplus H_{i-1}, x_i) \oplus H_{i-1}$
12. $e(x_i \oplus H_{i-1}, H_{i-1}) \oplus x_i$

11.4. Die Durchsatzrate einer Hashfunktion, die auf Blockchiffren basiert, ist wie folgt definiert: Wenn die Hashfunktion u Eingangsbits gleichzeitig verarbeitet, v Ausgangsbits produziert und w Verschlüsselungen mit der Blockchiffre durchführt, beträgt die Rate

$$v/(u \cdot w).$$

Was ist die Durchsatzrate von den vier Hashfunktionen, die in Abschnitt 11.3.2 eingeführt wurden?

11.5. Wir betrachten drei verschiedene Hashfunktionen, die Ausgangsbreiten von jeweils 64, 128 und 160 Bit haben. Wie viele zufällig gewählte Eingangswerte werden benötigt, um eine Kollisionswahrscheinlichkeit von $\varepsilon = 0,5$ zu erreichen? Wie viele Eingangswerte benötigt man für eine Wahrscheinlichkeit von $\varepsilon = 0,1$?

11.6. Beschreiben Sie genau, wie Sie eine Kollisionssuche nach einem Paar x_1 und x_2 durchführen würden, so dass die Bedingung $h(x_1) = h(x_2)$ für eine gegebene

Hashfunktion h erfüllt ist. Wie viel Speicherplatz benötigen Sie für die Suche, wenn die Ausgangsbreite der Hashfunktion n Bit beträgt?

11.7. Wir betrachten die Hashfunktionskonstruktion nach Hirose. Sie wird mit der Blockchiffre PRESENT (Blockbreite 64 Bit und ein Schlüssel von 128 Bit) realisiert. Mit der Funktion werden gehashte Passwörter in einem Computersystem abgespeichert. In dem System werden für jeden Benutzer i mit Passwort PW_i die folgenden Werte gespeichert:

$$h(PW_i) = y_i,$$

wobei die Passwörter (oder Passphrasen) eine beliebige Größe haben können. In den Computersystemen werden nur die Werte y_i benutzt, um Benutzer zu identifizieren und ihnen Zugang zu Ressourcen zu geben.

"Erfreulicherweise" bekommen Sie Zugang zu der Datei, die alle Hashwerte enthält, und ein Ruf als gefährlicher Hacker eilt Ihnen voraus. Dies sollte erst einmal keine Gefahr darstellen, da es aufgrund der Einwegeigenschaft der Hashfunktion unmöglich sein sollte, Passwörter aus den Hashwerten zu berechnen. Allerdings finden Sie einen kleinen, aber schwerwiegenden Fehler in der Implementierung: Die Konstante c in der Hashfunktion hat den Wert $c = 0$. Wir nehmen an, Sie kennen auch die Anfangswerte ($H_{0,L}$ und $H_{0,R}$).

1. Wie viele Bit besitzt jeder Eintrag y_i?
2. Ihr Ziel ist es nun, sich als Benutzer U einzuloggen (wobei U beispielsweise der Geschäftsführer der Organisation sein kann). Beschreiben Sie genau, warum nur etwa 2^{64} Schritte notwendig sind, um einen Wert PW_{hack} zu finden, für den gilt:

$$PW_{\text{hack}} = y_U$$

3. Welcher der drei Angriffe gegen Hashfunktionen wird hier durchgeführt?
4. Warum ist der Angriff nicht möglich, wenn $c \neq 0$?

11.8. In dieser Aufgabe untersuchen wir, warum fehlerkorrigierende Codes nicht als Hashfunktionen benutzt werden können. Wir betrachten eine Funktion, die für jedes 8-Bit-ASCII-Zeichen $b_{i1}, \ldots, b_{i8}$ einen 1-Bit-Hashwert wie folgt berechnet:

$$C_i = b_{i1} \oplus b_{i2} \oplus b_{i3} \oplus b_{i4} \oplus b_{i5} \oplus b_{i6} \oplus b_{i7} \oplus b_{i8}$$

1. Stellen Sie das Wort `CRYPTO` als Binär- oder Hexadezimalwert dar.
2. Berechnen Sie die sechs Bits des Hashwerts mittels der oben stehenden Gleichung.
3. Zeigen Sie, dass die Hashfunktion unsicher ist, indem Sie beschreiben, wie man eine Zeichenkette mit dem gleichen Hashwert konstruieren kann. Konstruieren Sie ein Beispiel hierfür.
4. Welche fundamentale Eigenschaft von Hashfunktionen ist hier nicht gegeben?