Allgemeine Informatik 3 (WS 2001/2002)

Abgabetermin 20.11. 2001 vor den Übungen

MD5 – Message Digest

Die Aufgabe dieses Programms besteht darin, die MD5 Prüfsumme einer Datei, die von der Standardeingabe gelesen wird, zu bestimmen. Dazu soll der vorbereitete Programmrumpf, den ihr unter http://www.mathematik.uni-ulm.de/sai/ws01/soft/blatt4/md5rumpf.c findet, ergänzt werden.

Hintergrund

Der MD5 Algorithmus liest eine beliebig lange Nachricht ein und berechnet in Abhängigkeit von dieser Nachricht eine 128 Bit Zahl (die Prüfsumme), die folgende Eigenschaften haben soll:

Sicherheit Es soll nur mit sehr großem Aufwand möglich sein, zwei Nachrichten zu finden, die die gleiche Prüfsumme haben.

Geschwindigkeit Die Berechnung der Prüfsumme soll schnell und einfach möglich sein.

Beim eigentlichen Entwurf gab es noch ein paar weitere Kriterien, die hier aber zu weit führen würden. Prüfsummen mit diesen Eigenschaften haben vielfältige Anwendungen. Unter anderem spielen solche Prüfsummen bei digitalen Signaturen eine große Rolle. Neben MD5 kommt dort auch häufig SHA (Secure Hash Algorithm) zum Einsatz.
Leider hat sich herausgestellt, daß es garnicht so einfach ist, einen Algorithmus zu entwerfen, der den beiden oben genannten Bedingungen genügt. MD4, der Vorgänger von MD5 muß inzwischen wohl als unsicher gelten.
Wer sich mit dem Thema Kryptographie weiter beschäftigen möchte, dem empfehle ich das Buch "Applied Cryptography" von Bruce Schneier. Weitere Informationen über die Schwächen von MD4 und bekannte Probleme mit MD5 sind z.B. auf http://www.rsa.com/rsalabs/faq/3-6-6.html zu finden. Eine vollständige Beschreibung des MD5 Algorithmus gibt es auch in rfc1321 (z.B. bei http://www.ietf.org/rfc/rfc1321.txt).

Hilfsfunktionen (7 Punkte)

Zunächst benötigen wir einige Hilfsfunktionen. Deklarationen für alle diese Funktionen befinden sich bereits im Programmrumpf, allerdings fehlt noch die Implementierung. Diese soll jetzt von

Euch ergänzt werden.

F, G, H und I

Die Funktionen F, G, H und I bekommen jeweils 3 Parameter (x, y und z), die durch Bitoperationen verknüpft werden. Der Rückgabewert ist das Ergebnis dieser Verknüpfung.

Die Funktion F: Wenn das i-te Bit im Parameter x gesetzt ist, dann wird das i-te Bit aus dem Parameter y in das i-te Bit des Ergebnisses übernommen, sonst wird das i-te Bit aus dem Parameter z übernommen. Interpretiert man ein gesetztes Bit als TRUE und ein nicht gesetztes Bit als FALSE, dann soll also für jedes Bit die Operation

$$\text{IF } x \text{ THEN ergebnis} = y \text{ ELSE ergebnis} = z \text{ END}$$

durchgeführt werden.

Die Funktion G: Analog zu F wird die folgende Operation bitweise durchgeführt:

$$\text{IF } z \text{ THEN ergebnis} = x \text{ ELSE ergebnis} = y \text{ END}$$

Die Funktion H: Es wird gezählt, in wievielen der drei Parameter das i-te Bit gesetzt ist. Das i-te Bit des Ergebnisses soll genau dann gesetzt sein, wenn diese Zahl ungerade ist.

Die Funktion I: Findet und implementiert eine Bitoperation, die zu folgender Wertetabelle paßt:

i-tes Bit in x	i-tes Bit in y	i-tes Bit in z	i-tes Bit im Ergebnis
0	0	0	1
0	0	1	0
1	0	0	1
1	0	1	1
0	1	0	0
0	1	1	1
1	1	0	0
1	1	1	0

Hinweis: Was fällt in der Ergebnisspalte auf, wenn man die ersten und die letzten vier Zeilen vergleicht?

Erstellt für die ersten drei Funktionen (F, G und H) jeweils eine Wertetabelle, wie sie bei der Funktion I gegeben ist. Anschließend sollen die vier Funktionen natürlich auch implementiert werden.
Hinweis: Jede dieser vier Funktionen läßt sich mit Hilfe von einigen Bitoperationen in einer Zeile realisieren.

Rotieren

Die Funktion rotl soll die Bits im ersten Parameter nach links rotieren und zwar um so viele Positionen, wie der zweite Parameter angibt. Das Ergebnis ist dann der Rückgabewert der Funktion. Eine Linksrotation um eine Position bedeutet einfach, daß das oberste Bit weggenommen und hinten angehängt wird.

Bytes tauschen

Die Funktion swapbytes soll die vier Bytes des Parameters vertauschen. Dabei wird das erste Byte zum vierten, das zweite zum dritten und umgekehrt.

Bits schütteln – Zum Ersten (1 Punkt)

Jetzt werden die Bits richtig durcheinandergeschüttelt. Dazu soll die Funktion scramble implementiert werden. Haltet euch dabei bitte genau an diese Beschreibung:

- Als erstes wird die durch den Parameter func spezifizierte Funktion mit den Parametern p, q und r aufgerufen. Es soll also F aufgerufen werden, falls func den Wert USE_F hat, G falls func den Wert USE_G hat usw.
- Zu diesem Ergebnis wird der Parameter o, die "Konstante" constant und der Wert chunk[chunkidx] addiert. Bei letzterem handelt es sich um ein Stück der aktuellen Nachricht (chunkidx ist der sechste Parameter von scramble).
- Anschließend werden die Bits im Ergebnis um shift Positionen nach links rotiert.
- Der Rückgabewert ist die Summe aus dem Ergebnis des letzten Schritts und dem Parameter p.

Die Werte von constant und shift sind durch den Algorithmus vorgegeben. Die nötigen Funktionsaufrufe zur Berechnung dieser Werte befinden sich bereits im Programmrumpf. Darum müßt ihr euch also nicht mehr kümmern.

Bits schütteln – Zum Zweiten (2 Punkte)

Zum Schluß kommen wir jetzt zum Hauptprogramm. Im Hauptprogramm werden vier Variablen (A, B, C und D) zusammen mit der eingelesenen Nachricht in einen großen Topf geworfen, dann wird kräftig geschüttelt (dabei hilft die Funktion scramble) und am Ende enthalten die vier Variablen dann die gesuchte Prüfsumme. Bitte haltet euch auch hier wieder genau an die folgende Beschreibung.

1. Zunächst werden die vier Variablen A, B, C und D mit Anfangswerten initialisiert. Der Code hierfür befindet sich bereits im Programmrumpf.
2. Anschließend wird die Nachricht in Blöcken bestehend aus je 64 Byte eingelesen. Dazu gibt es im Programmrumpf bereits die Funktion getchunk. Diese Funktion liest 64 Bytes ein. Falls nicht mehr genügend Bytes gelesen werden können, erfindet die Funktion getchunk nach einem festen Schema genügend weitere Daten, um den Block zu füllen (padding). Die gelesenen Daten werden zu 16 unsigned int Werten verknüpft und in der globalen Variable chunk abgelegt. Der Rückgabewert der Funktion getchunk ist 1, wenn noch ein Block gelesen werden konnte und 0, wenn das Dateiende erreicht ist.
   Für jeden gelesenen Block sollen nun folgende Operationen durchgeführt werden:
   (a) Die aktuellen Werte von A, B, C und D werden in temporären Variablen zwischengespeichert.

(b) Es wird 64 Mal die Funktion scramble aufgerufen und zwar mit folgenden Parametern:

- Die ersten vier Parameter sind immer die Variablen A, B, C und D. Allerdings werden diese bei jedem weiteren Aufruf um eine Position nach rechts rotiert. Der zweite Aufruf hat also die ersten vier Parameter D, A, B und C; der dritte C, D, A und B usw.
- Wie der fünfte und sechste Parameter bestimmt wird ändert sich nach je 16 Aufrufen gemäß folgender Tabelle:

Aufruf Nr.	5. Parameter	6. Parameter
1-16	USE_F	i-1
17-32	USE_G	(1+5*(i-17)) MODULO 16
33-48	USE_H	(5+3*(i-33)) MODULO 16
49-64	USE_I	(7*(i-49)) MODULO 16

  Dabei ist i die Nummer des Aufrufs. Der fünfte Parameter wird von scramble verwendet, um auf einen der 16 Integer des aktuellen Blocks zuzugreifen. Die Formeln sind dabei so geschickt gewählt, daß jeder der 16 Integer eines Block in jeder 16er Gruppe von scramble-Aufrufen genau einmal verwendet wird. USE_F, USE_G, USE_H und USE_I sind Konstanten, die im Programmrumpf bereits definiert sind.
- Der Rückgabewert jedes scramble-Aufrufs wird sofort nach dem Aufruf an die Variable zugewiesen, die als erster Parameter übergeben wurde.

(c) Schließlich werden die in 2a gesicherten Werte von A, B, C und D zu den aktuellen Werten addiert. Dann kommt der nächste Block an die Reihe.

3. Nachdem alle Blöcke der Nachricht bearbeitet wurden wird als letzte Rechenoperation noch die Reihenfolge der Bytes in A, B, C und D vertauscht (swapbytes).

4. Zum Schluß werden die Variablen A, B, C und D noch in Hex-Darstellung (16er-System, printf Formatierung %x oder %X) ausgegeben.

Tests

Auf fast allen unseren Suns ist bereits das Programm md5 installiert. Wenn alles geklappt hat, dann sollten euer Programm und md5 bei der gleichen Eingabedatei auch die gleiche Prüfsumme berechnen.

Hinweise

Da eine der vorgefertigten Funktionen die Mathematikbibliothek verwendet muß beim Übersetzen des Programms die Option "-lm" mit angegeben werden.
Wenn euer Programm nicht die richtigen Ergebnisse liefert, dann testet es mit einer leeren Eingabedatei (Länge 0 Byte! Sonst klappt es nicht). In diesem Fall sollten die 64 Aufrufe von scramble die folgenden Rückgabewerte haben:

0xA5202774	0xF59592DD	0xE7F06B23	0x1B163203
0x32033344	0x2F35D494	0xF5B158DB	0x9BC13CE9
0x3893B991	0xFCE4A312	0xE1EF0576	0x70768A29
0xF56C7CF1	0x374943A7	0x5AA53F75	0xD6819C6A
0x1C7D7513	0x7BD57A3A	0xC095F13A	0xBD782E17
0x3D1E3E6C	0x68B7B3E3	0xEB41643E	0xE422531A
0x306EC122	0xD28C77C2	0xA3C663DA	0xA0572807
0x13707036	0xAE7813DB	0x1C31C384	0xA2205F1F
0xDF63EAA1	0xC3689F5B	0x12F3E755	0x004B6669
0x5F7A9B2E	0xABC34E16	0x91CA4CB7	0xC5DC8C15
0x4497169D	0x76FD93D4	0xFD95F243	0x0FE32453
0x3F55EDFD	0x22A31F54	0x68D84EA2	0xCA7D2DBD
0x93AA2577	0x1688DC85	0xCD85B8CB	0x561E0689
0x5625A114	0x3450F42B	0x392AD0D0	0x1E77FA61
0x474A9C8C	0xDFCE00BC	0x36594B14	0x30130182
0x7246FAD3	0x6E10A476	0xFF4EA3EB	0x14E45506

Um die Hilfsfunktionen testen zu können, sind hier für ein paar Fälle die richtigen Ergebnisse aufgelistet. Typische Fehler in den Hilfsfunktionen sollten bei diesen Testfällen eigentlich auffallen:

$$\begin{aligned}
F(204, 240, 170) &= 226 \\
G(204, 240, 170) &= 216 \\
H(204, 240, 170) &= 150 \\
I(204, 240, 170) &= 4294967085 \text{ (entspricht } -211) \\
\mathrm{rotl}(1073741824, 2) &= 1 \\
\mathrm{rotl}(10, 4) &= 160 \\
\mathrm{swapbytes}(19088743) &= 1732584193
\end{aligned}$$