Fachhochschule Köln
University of Applied Sciences Cologne
Campus Gummersbach

Algorithmische Anwendungen

Symmetrische Verschlüsselung mit Blowfish-Algorithmus

von
Andrej Doumack
MatrikelNr: 11032929
Gruppe: C_rot

Inhaltsverzeichnis

Symmetrisch Verschlüsselungsalgorithmen

Die Verschlüsselungsverfahren sind symmetrisch, wenn diese zum Verschlüsseln und Entschlüsseln den gleichen Schlüssel benutzen. Bei den asymmetrischen wird ein Schlüsselpaar erstellt, meistens mit einem Public- und einem Private-Key, dabei wird für die Verschlüsselung ein Public-Key verwendet und für die Entschlüsselung der Private-Key.

Eines der Vorteil von einem symmetrischen Verschlüsselungsverfahren ist seine Geschwindigkeit, diese sind um einiges schneller als die asymmetrische Verschlüsselungsverfahren.

Die symmetrischen Verschlüsselungsverfahren werden auch Secret-Key-Verfahren genannt, da die Schlüssel mit denen Verschlüsselt und Entschlüsselt wird, müssen diese geheim gehalten werden. Beim Übertragen des Schlüssels an die Person, die den verschlüsselten Text entschlüsseln soll muss für die Verbindung eine weitere Verschlüsselung benutzt werden.

Ein weiterer Nachteil ist die Anzahl der Schlüssel die bei der Kommunikation von Personen benutzt wird. Bei der Verschlüsselung und Entschlüsselung braucht jedes Paar von Personen die miteinander kommunizieren wollen einen Schlüssel. Dadurch entsteht eine sehr große Anzahl an Schlüsseln. Bei n Personen sind es n*(n-1)/2.

Die bekanntesten symmetrische Verschlüsselungsalgorithmen sind: AES, Blowfish, CAST, DES, Rijndael, Twofish und 3DES.

Anforderungen an ein Verschlüsselungsalgorithmus

Um einen Verschlüsselungsalgorithmus zu entwickeln müssen einige Anforderungen an diesen gestellt werden. Ein Standardverschlüsselungsalgorithmus muss in mehreren Bereichen einsetzbar sein diese wären z.B.:

- Basisverschlüsselung. Der Algorithmus muss in der Lage sein Dateien oder Datenstreams zu verschlüsseln
- Zufallsbiterstellung. Der Algorithmus sollte effizient sein bei der Erstellung von Zufallsbits.
- Paketverschlüsselung. Der Algorithmus sollte effizient in Pakete aufgeteilte Daten verschlüsseln.

Einige Anforderungen an den Algorithmus bezüglich der Plattformen wurden gestellt.

- Große Prozessoren. Der Algorithmus sollte effizient auf einem 32-bit Prozessor arbeiten.
- Mittlere Prozessoren. Der Algorithmus sollte auf den Mikrokontrollern und anderen kleineren Prozessoren wie 68HC11 lauffähig sein.
- Kleine Prozessoren. Es soll möglich sein den Algorithmus auf Smartkarten zu implementieren.

Einige Zusatzanforderungen wurden gestellt. Die Zusatzanforderungen sollten wenn möglich mit einbezogen werden beim Design.

- Der Algorithmus muss einfach zu programmieren sein. Es gab zum Beispiel bei den ersten Implementierungen von DES zahlreiche Fehler, weil das Algorithmus zu kompliziert ist.
- Der Algorithmus sollte einen flachen Schlüsselabstand haben um einen Zufallsbitstring als Schlüssel zu benutzen. Es sollten auch keine schwachen Schlüssel geben.
- Der Algorithmus sollte einen einfache Schlüsselverwaltung haben um für einfache Softwareimplementation.
- Der Algorithmus sollte einfach zu modifizieren sein um verschiedene Sicherheitslevels einzustellten, minimum und maximum.
- Alle Operationen sollten Daten in Byte-Grossen Blöcken verarbeiten. Wenn möglich in 32-bit

Blöcken.

Blowfish

Blowfish wurde 1993 von Bruce Schneier entwickelt und als eine schnellere und freie Alternative zu den verfügbaren Algorithmen freigegeben. Dazu hat er einige Veröffentlichungen gemacht, wo er den Algorithmus beschreibt. Die Philosophie hinter Blowfish ist ein einfaches Design, welches es einfacher macht den Algorithmus zu verstehen und zu implementieren.

Blowfish ist ein 64-bit-Block-Cipher-Algorithmus, d.h. dieser benutzt 64-Bit-Blöcke von den zu Verschlüsselnden Daten. Der Algorithmus besteht aus zwei Teilen. Einem Schlüsselerweiterungsteil und einem Datenverschlüsselungsteil. Der Schlüsselerweiterungsteil konvertiert einen Schlüssel, der bis zu 448-Bit lang sein kann, in verschiedene Subschlüsselfelder insgesamt 4168 Byte.

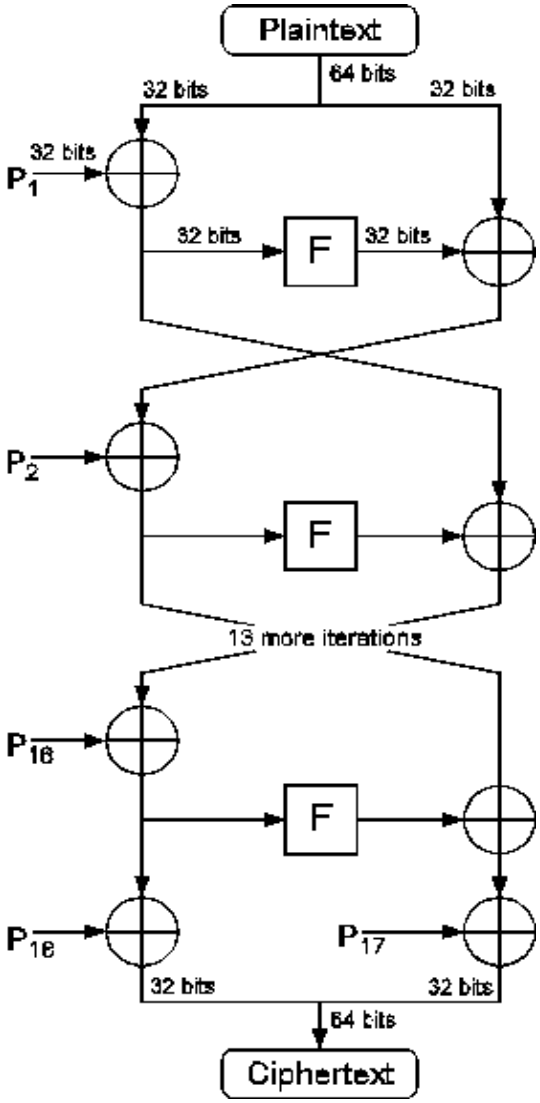
Die Verschlüsselung erfolgt in 16 Runden Feistelnetzwerk. Jeder Runde besteht aus einer Schlüssel abhängigen Permutation und Schlüssel- und Daten abhängigen Substitution. Alle Operationen sind XOR's und Additionen 32-Bit.

Subschlüssel

Blowfish benutzt große Anzahl von Subschlüsseln. Diese müssen erzeugt werden bevor irgendwelche Daten ver- oder entschlüsselt werden.

P-Array enthält 18 32-Bit Subschlüssel P1,P2...P18

Vier 32-Bit S-boxes mit jeweils 256 Einträgen:
S1,0 ,S1,1,....,S1,255;
S2,0 ,S2,1,....,S2,255;
S3,0 ,S3,1,....,S3,255;
S4,0 ,S4,1,....,S4,255;



Verschlüsselung

Blowfish arbeitet nach Feistelnetzwerk mit 16 Runden.
Als Eingang wird ein 64-Bit Datenelement verwendet.
Aufteilen des Datenelements in zwei 32-Bit Hälften: xL, xR.

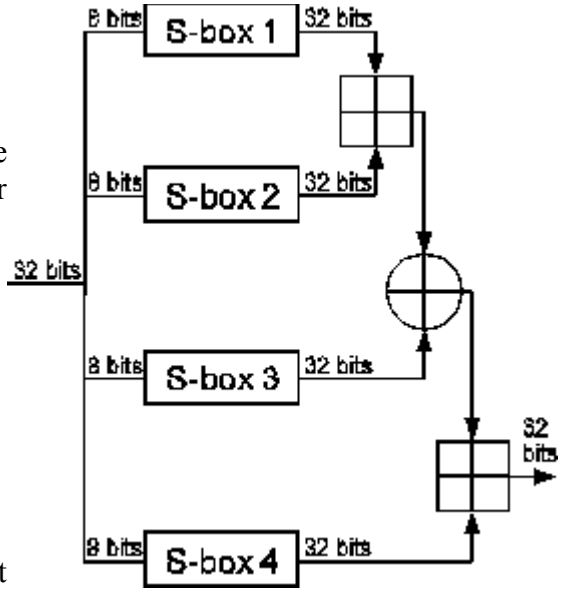
```
For i = 1 to 16:
xL = xL XOR Pi
xR = F(xL) XOR xR
Swap xL and xR
Next i

Swap xL and xR (Undo the last swap.)
xR = xR XOR P17
xL = xL XOR P18
Recombine xL and xR
```

Funktion F:
Das Element xL wird in vier 8-Bit Teile aufgeteilt: a, b, c und d

$F(xL) = ((S1,a + S2,b \bmod 2^{32}) \text{ XOR } S3,c) + S4,d \bmod 2^{32}$

Die Entschlüsselung funktioniert wie die Verschlüsselung, der P-Array wird dabei in der umgekehrten Reihenfolge verwendet.



Erstellung der Subschlüssel

1. Es werden P-Array und vier S-boxen initialisiert mit einem festen String, z.B. Hexadezimalwerte von pi.
2. Jetzt werden die ersten 32-Bit von unserem Schlüssel mit P1 gexort, danach die nächsten 32-Bit mit P2 usw. Dies wird solange ausgeführt bis P-Array komplett mit den Werten aus dem Schlüssel gexort wurde.
3. Dann wird mit dem Blowfish Algorithmus verschlüsselt dabei sind die beiden Werte xL und xR null und es werden die Subschlüssel, die in den ersten beiden Schritten erzeugt wurden benutzt.
4. Dann wird P1 und P2 mit dem Ergebnis aus Schritt 3 ersetzt.
5. Jetzt wird das Ergebnis aus Schritt 3 mit Blowfish verschlüsselt.
6. P3 und P4 werden jetzt mit dem Ergebnis aus Schritt 5 ersetzt.
7. Dies wird jetzt so lange durchgeführt bis alle P-Array und S-boxen mit den neuen Werten ersetzt wurden.

Kryptoanalyse von Blowfish

Einige der Cryptographen haben Blowfish auf Herz und Nieren getestet, dabei sind 2 Ergebnisse raus gekommen.
1. Sarge Vaudenay hat raus gefunden, dass es bei schwachen Schlüsseln - bei denen z. B. mehrere Werte gleichzeitig vorkommen - die Teile des Schlüssels ermittelt werden können, dies gilt aber nur für Verschlüsselung von weniger als mit 14 Runden.

2. Vincent Rijmen hat die Second-Order Differential Attacke erfolgreich auf 4 Runden-Blowfish durchgeführt, diese hat aber bei mehr Runden nicht mehr funktioniert.

Beim Release in Dr. Dobb's Journal hat Schneier die Commutity zu einem Kryptoanalyse-Contest aufgefordert. Damit auch der Rest der Welt die Möglichkeit hat den Algorithmus zu prüfen und gegebenenfalls zu knacken. Ein Jahr später wurde ein Artikel in Dr. Dobb's Journal veröffentlicht, wo Schneier die Ergebnisse des Contests vorgestellt hat. Es gab einige Fälle über, die es lohnte zu sprechen.

1. John Kelsey hat einen Angriff entwickelt, der 3-Runden-Blowish knackt, konnte aber nicht

erweitert werden. Bei dem Angriff wurde die F Funktion ausgenutzt das Ergebnis war, dass mod 232 und Xor nicht ausgeführt wurden.

2. Vikramjit Singh Chhabra hat versucht eine effiziente Brute Force Suchmaschine zu implementieren.

3. Serge Vaudenay hat eine vereinfachte Variante von Blowfish geprüft, dabei waren die S-Boxes bekannt und nicht Schlüssel abhängig. Bei dieser Variante konnte man mit einem Differential Angriff den P-Array wiederherstellen. Dieser Angriff funktioniert aber nicht für 8-Runden-Blowfish oder höher.

Keiner hat es geschafft einen Angriff zu entwickeln, welcher Blowfish knackt. Es soll aber nicht heißen, dass der Algorithmus nicht knackbar wäre, es bedarf immer noch Analysen, bevor man sagen kann, dass dieser wirklich sicher sei.

Einsatz von Blowfish

Blowfish wird immer noch zahlreich in dem Kommerziellen und OSS-Bereich verwendet um Daten zu verschlüsseln. Obwohl der Algorithmus von 1993 ist, wird dieser immer noch eifrig von den Entwicklern eingesetzt, wie man es in der nachfolgenden Liste sieht.

OpenBSD nutzt eine Abwandlung von Blowfish, der einen langsamen Schlüssel-Scheduler verwendet. Die Idee dahinter ist der Schutz gegen die Dictionary Attacken.

Secure IT von Cypherix basiert auf Blowfishalgorithmus. Mit dem kommerziellen Tool kann man individuell Dateien oder Verzeichnisse verschlüsseln.

Blowfish Advanced CS, eine Open Source Software, die es ermöglicht Daten zu verschlüsseln. Die Software verwendet 8 Verschlüsselungsalgorithmen unter anderem Blowfish, Twofish, AES, RC4, Triple-DES.

ABI-Coder ein Sharewareprogramm, mit dem man Dateien verschlüsseln kann. Das Programm benutzt Blowfish, Tripple-Des oder AES um Daten zu verschlüsseln und um selbst entschlüsselnde Datei zu erstellen, die dann übers Internet oder Netzwerk an den Empfänger verschickt werden sollen.

ScramDisk ist ein Freewareprogramm für Windows 95/98 zur Erzeugung von virtuellen, verschlüsselten Partitionen. ScramDisk benutzt zahlreiche Verschlüsselungsalgorithmen wie z.B. Triple-DES, Blowfish, IDEA, TEA, DES usw.

TrueCrypt erstellt verschlüsselte Containerdaten oder verschlüsselt ganze Partitionen.
Das Freewaretool TrueCrypt ist für alle Windows Versionen und Linux erhältlich und benutzt einige Verschlüsselungsalgorithmen wie Blowfish, Twofish, Triple-DES, AES usw.

BestCrypt ist ein Shareware Tool für alle Windows Versionen und Linux erhältlich und verschlüsselt Daten in Containerdateien, die als virtuelle Partitionen gemountet werden können. Zur Verschlüsselung verwendet das Tool zahlreiche Algorithmen wie DES, CAST-128, GOST-256, Blowfish, Twofish usw. Die Verschlüsslungsalgorithmen liegen als Module vor und können entfernt oder hinzugefügt werden, dafür gibt es ein BestyCrypt Development Kit, mit dem man eigene Verschlüsselungsmodule schreiben kann.

Silver Key ist eine Shareware, die für alle Windows Versionen erhältlich ist. Damit kann man

Dateien und ganze Ordner verschlüsseln. Verwendet Verschlüsselungsalgorithmen sind : Blowfish, Twofish, AES, Triple-DES usw. Das Programm erstellt auch selbst entschlüsselnde Dateien, die übers Netzwerk oder Internet verschickt werden können.

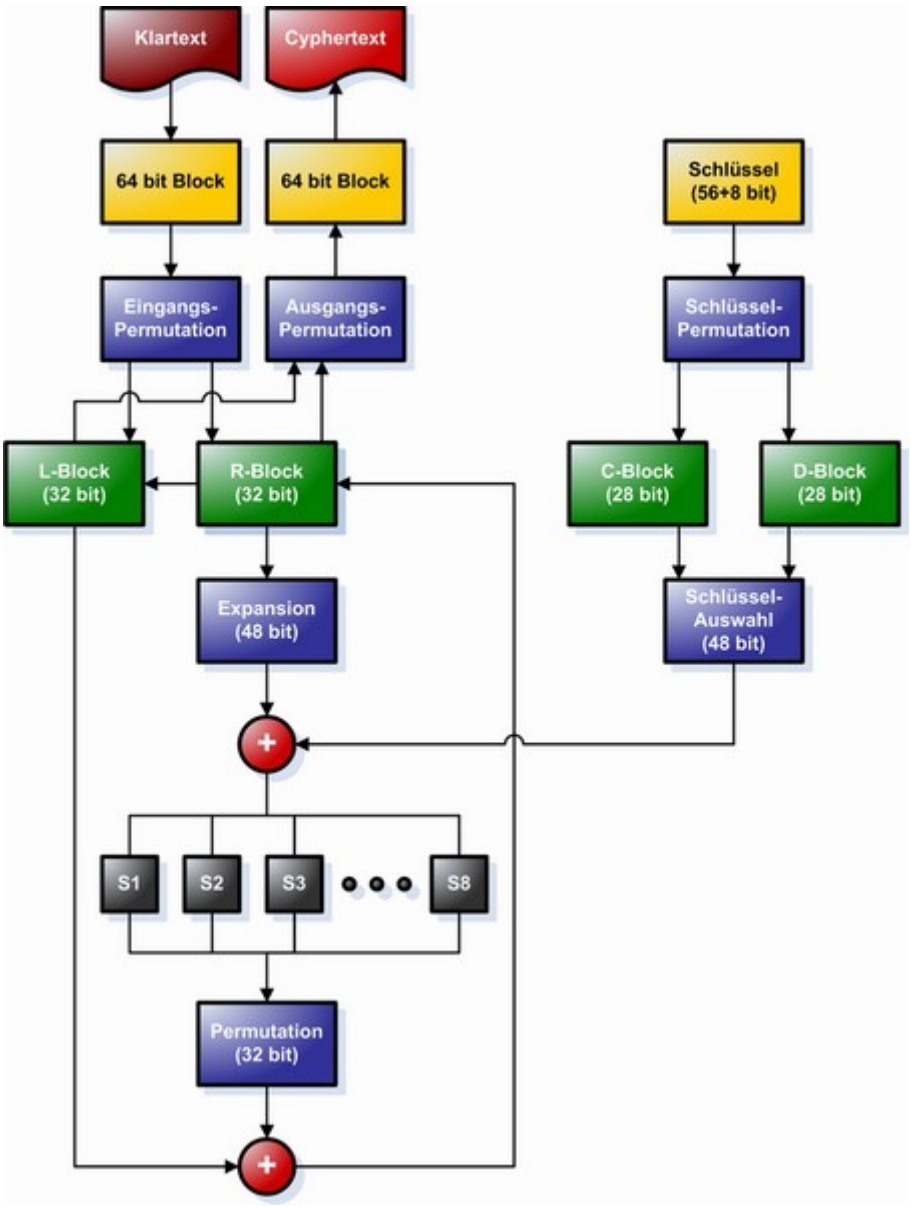
SecureTask ein Freeware Programm, das für alle Windows Versionen verfügbar ist. Das Programm verschlüsselt Dateien und erstellt auf Wunsch verschlüsselte Archive. Es werden folgende Verschlüsselungsalgorithmen verwendet: AES (256-Bit), Blowfish (576-Bit), Twofish (256-Bit), CAST-256, GOST (256-Bit), MARS (448-Bit), Square, RC-6 (2040-Bit), Serpent (256-Bit) und TripleDES (192-Bit) mit den Modi ECB, CBC,CFB oder OFB

Blowfish vs DES

DES ist genau wie Blowfish ein symmetrischer Verschlüsselungsalgorithmus. DES Funktioniert wie Blowfish als Blockchiffre und genau wie bei Blowfish werden 16 Runden nach dem Feistel Netzwerk durchlaufen.

Die Blockgrösse bei beiden ist 64 Bit, d.h. es werden immer 64 Bit des Eingabetextes in Chiffretext transformiert.

Der Schlüssel bei DES hat eine Länge von 64 Bit dem Benutzer stehen aber nur 56 Bit zur Verfügung die restlichen 8 Bit werden zum Paritäts-Check verwendet. Bei Blowfish kann dieser bis 448 Bit sein.



Auf dem Bild oben sieht man die Funktionsweise des DES-Algorithmus. Der Eingangstext von 64 Bit wird wie beim Blowfish in zwei 32 Bit Blöcke unterteilt.

Beim ver- und entschlüsseln arbeitet DES etwas anders als Blowfish. DES benutzt die sogenanten Permutationen bevor die Blöcke ins FeistelNetzwerk reingehen.

Da die Schlüssellänge nur 56 Bit lang ist beim DES, wurde dieser innerhalb von 3,5 Stunden auf einer $1M Maschine entschlüsselt. Die kurze Schlüssellänge ist für heutige Standards einfach nicht mehr zu gebrauchen, da es zu unsicher ist und viel zu schnell entschlüsselt werden kann.

Asymptotische Laufzeitanalyse

Schlüsselgenerierung:

```
long data = 0x00000000L;
int j = 0;

//Schleife 1
for (int i = 0; i < 18; i++) {
```

```
        data = 0x00000000L;
        //innere Schleife
        for (int k = 0; k < 4; k++) {
                data = (data << 8) | (key[j] & 0xff);
                j++;
                if (j == key.length) {
                        j = 0;
                }
        }
        this.parray[i] = this.parray[i] ^ data;
}

this.datal = 0x00000000L;
this.datar = 0x00000000L;

//Schleife 2
for (int i = 0; i < 18; i += 2) {
        encode(this.datal, this.datar);
        this.parray[i] = this.datal;
        this.parray[i + 1] = this.datar;
}

//Schleife 3
for (int j = 0; j < 4; j++) {
        //innere Schleife
        for (int i = 0; i < 256; i += 2) {
                encode(this.datal, this.datar);
                this.sboxes[j][i] = this.datal;
                this.sboxes[j][i + 1] = this.datar;
        }
}

private void encode(long Xl, long Xr) {
        long temp;
        for (int i = 0; i < 16; i++) {
                Xl = Xl ^ parray[i];
                Xr = functionF(Xl) ^ Xr;
                temp = Xl;
                Xl = Xr;
                Xr = temp;
        }
        temp = Xl;
        Xl = Xr;
        Xr = temp;
        Xr = Xr ^ this.parray[N];
        Xl = Xl ^ this.parray[N + 1];
        this.datal = Xl;
        this.datar = Xr;
}
```

Bei der Schlüsselgenerierung werden einige For-Schleifen benutzt, diese haben eine feste Laufkonstante. Die erste For-Schleife läuft 18 Mal und die innere Schleife 4 Mal, d.h. Die erste

Schleife läuft dann 18*4 Mal.

Die For-Schleife 2 läuft 18 Mal, in dieser wird aber die Methode encode aufgerufen, in der encode-Methode gibt's es eine Schleife die konstant immer 16 mal durchläuft. Also ist die Laufzeit der Schleife 2 18*16.

Die 3. Schleife hat noch eine innere Schleife. Die innere Schleife ruft noch die encode-Methode auf. Die Laufzeit der 3. Schleife ist also 4*256*18.

Die Laufzeit ist also bei der Schlüsselgenerierung immer konstant, d.h. unabhängig von der Länge des Schlüssels. Die Gesamtlaufzeit ist dann (18*4+18*16+4*256*18).

```
public String encodeString(String text) {
	byte[] textByte = text.getBytes();
	byte[] ergeb = new byte[(((text.length() - 1) & ~7) + 8)];
	int out = 0;
	int j = 0;
	for (int i = 0; i < textByte.length; i += 8) {
		this.datal = 0;
		this.datar = 0;
		for (int k = 0; k < 4; k++) {
			this.datal = (this.datal << 8)
			| ((j < textByte.length)
			? (textByte[j] & 0x0ff) : 00);
				j++;
		}
		for (int k = 0; k < 4; k++) {
			this.datar = (this.datar << 8)
			| ((j < textByte.length)
			? (textByte[j] & 0x0ff) : 00);
				j++;
		}
		encode(this.datal, this.datar);
		ergeb[out++] = (byte) ((this.datal >>> 24) & 0x0ff);
		ergeb[out++] = (byte) ((this.datal >>> 16) & 0x0ff);
		ergeb[out++] = (byte) ((this.datal >>> 8) & 0x0ff);
		ergeb[out++] = (byte) (this.datal & 0x0ff);
		ergeb[out++] = (byte) ((this.datar >>> 24) & 0x0ff);
		ergeb[out++] = (byte) ((this.datar >>> 16) & 0x0ff);
		ergeb[out++] = (byte) ((this.datar >>> 8) & 0x0ff);
		ergeb[out++] = (byte) (this.datar & 0x0ff);
	}
	return Base64.encodeBytes(ergeb);
}
```

Beim Verschlüsseln von den Strings ist die Laufzeit nicht mehr konstant und ist von der Länge des Strings abhängig. Da die Methode encode aufgerufen wird ist also die Laufzeit (n/8 * 16). n wird durch 8 geteilt da man von dem String die 64 bit-Blöcke braucht.

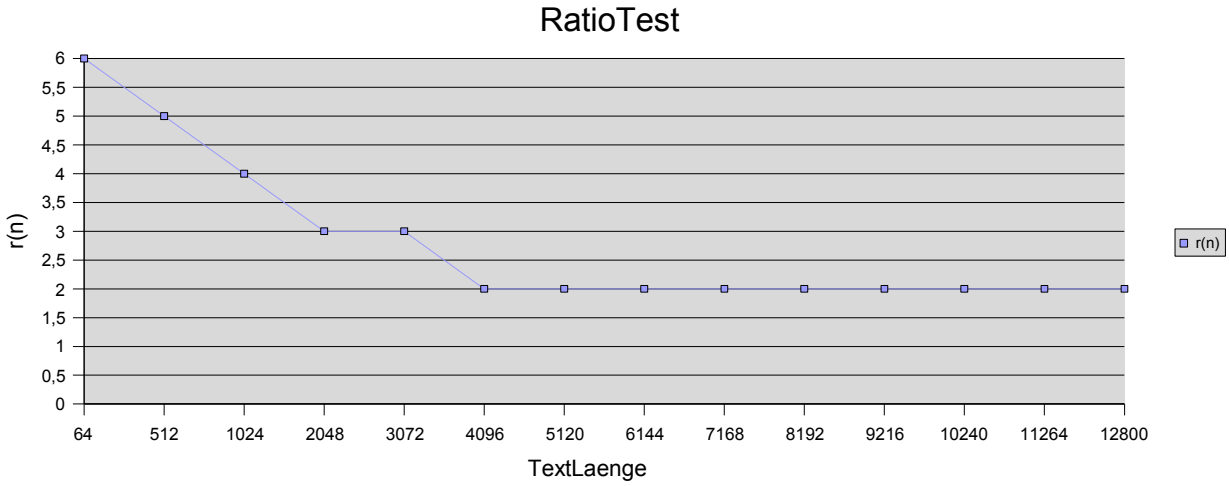
Die Gesamtlaufzeit mit Schlüsselgenerierung und Verschlüsselung ist:
(18*4+18*16+4*256*18)+(n/8 * 16)

Experimentelle Analyse

Ratio Test

Bei der experimentellen Analyse wurden die Operationen gezählt und ausgewertet.

TextLaenge	r(n)
64	6
512	5
1024	4
2048	3
3072	3
4096	2
5120	2
6144	2
7168	2
8192	2
9216	2
10240	2
11264	2
12800	2



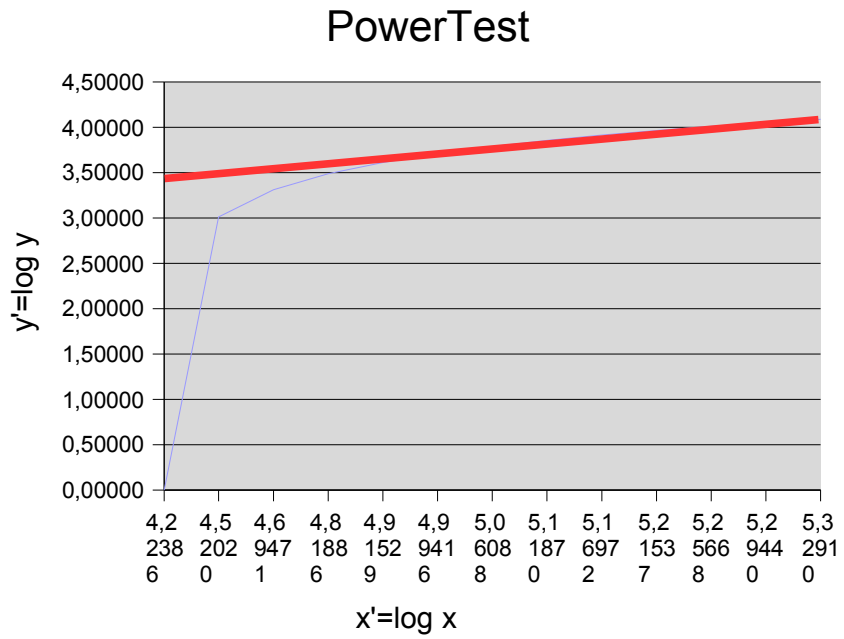
R(n) setzt sich wie folgt zusammen. R(n)=t(n)/f(n) dabei ist f(n) die geschätzte Funktion. Die geschätzte Funktion hierbei ist f(n)=16744+16*n.

Die Grafik fängt oben an, weil die Schlüsselgenerierung hierbei teilweise überwiegt und die geschätzte Funktion von der Länge des Textes abhängig ist. Ab der Textlänge 4096 pendelt sich das ganze auf 2 ein.

Power Test

Beim Powertest werden die Eingabedaten nach log überführt, d.h. (x,y) -> (x',y') x'=log x und y'=log y.

X=log x	Y'=log y
4,22386	0,00000
4,52020	3,01030
4,69471	3,31133
4,81886	3,48742
4,91529	3,61236
4,99416	3,70927
5,06088	3,78845
5,11870	3,85540
5,16972	3,91339
5,21537	3,96454
5,25668	4,01030
5,29440	4,05169
5,32910	4,08948



Gesucht: y=t(x)=bx^c
log-log-Transformation: y'=cx'+b
b=3,4 (y-Schnittpunkt)
c=1,18 (Steigung)
also: y' = 1,18x'+3,4
daraus folgt: t(n)=3,4n^1,18

Literatur

- http://www.schneier.com
- B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993), Springer-Verlag, 1994, pp. 191-204.
- B. Schneier,"The Blowfish Encryption Algorithm" Dr. Dobb's Journal, v. 19, n. 4, April 1994, pp. 38-40.
- B. Schneier, „Blowfish-One Year Later“ Dr. Dobb's Journal, 1995