Keygarantie Verschlüsselungsverfahren

Christopher Besch und Katharina Libner

13. März 2021

Inhaltsverzeichnis

1 Abstract

Keygarantie ist ein symmetrisches 128-bit Stromverschlüsselungsverfahren, das stark an ChaCha angelehnt ist [J.08]. Damit basiert es auf XOR, was die Ver- und Entschlüsselung mit dem selben Algorithmus erlaubt. Für die Generation des mit XOR verwendeten Stromschlüssels wird ein komplexer und praktisch äußerst schwer umkehrbarer Mechanismus aus mehreren mathematischen Methoden verwendet, wodurch selbst bei kompromittiertem Klartext die verwendete Passphrase geheim bleibt.

2 Einleitung

Das Kernstück von Keygarantie bildet die XOR-Verschlüsselung, wobei ein Klartext mit einem Stromschlüssel bitweise verschlüsselt wird. Eine 1 in dem Stromschlüssel dreht den gegenüberliegenden Bit im Klartext um, eine 0 verändert ihn nicht. Wenn nun der so generierte Geheimtext das Verfahren mit dem selben Schlüssel erneut durchläuft, werden die umgedrehten Bits erneut verändert, sodass wieder der Klartext entsteht. Allerdings ist davon auszugehen, dass Teile des Klartextes kompromittiert werden. Ist dies der Fall, kann zusammen mit dem öffentlichen Geheimtext auf den verwendeten Stromschlüssel geschlossen werden. Dadurch wären alle zukünftigen Transmissionen ebenfalls kompromittiert. Dies wäre fatal.

Es muss also davon ausgegangen werden, dass unter Umständen der Stromschlüssel öffentlich bekannt ist. Daher ist sicherzustellen, dass ein bekannter Teil des Stromschlüssels praktisch nie erneut für die Verschlüsselung mit XOR verwendet wird. Um dies zu erreichen wird ein 128-bit Schlüssel über 30 Runden diffundiert. Dieses Verfahren generiert einen 256-bit Seed.

Der Seed wird anschließend in einem Pseudozufallsgenerator eingesetzt, um eine unendliche Menge an Bits für den Stromschlüssels zu generieren. Praktisch werden allerdings aus Sicherheitsgründen nur 512 Bits pro Seed verwendet, wodurch weniger Information für einen Angriff vorhanden sind. Daher muss der Klartext in 512-bit Blöcke unterteilt werden, die aufsteigend und eindeutig mit einer Blocknummer nummeriert und identifiziert werden. Hieraus folgt, dass für jeden einzelnen Block ein anderer Seed für den Pseudozufallsgenerator verwendet werden muss. Durch diesen Schritt, der praktisch mit modernen Methoden unumkehrbar ist, wird die geforderte Sicherheit garantiert.

3 Pseudozufallsgenerator

Ziel ist es einen „zufälligen“ Schlüssel zu erzeugen, der unendlich lang werden kann, um unbeschränkt große Bitanzahlen zu verschlüsseln.

3.1 Linearer Kongruenzgenerator

$$x_i = ((x_{i-1} \cdot a) + c) \bmod 2^n$$

- x_i ist ein neuer „Zufallswert“, der dem Schlüssel zugeordnet wird.

- x_{i-1} ist dementsprechend der vorherige Wert, der bestimmt wurde.
- Da jeder Wert aus der Menge aller x_i erst durch einen vorherigen Wert entsteht, muss es einen Startwert x_0 geben. Der Seed liefert diesen Startwert.
- n sei aus $\mathbb{N}$ und markiert die Bitintervallgrenze in der x_i liegt. Für das Intervall gilt also: $[0, n]$.
- c sei aus $\mathbb{N} < 2$
- n und dient als Summand.
- a sei in $\mathbb{N} < 2$
- n und dienst als Multiplikator.

Die „Zufallswerte“ des Kongruenzgenerators legen die Grundlage für den Blockschlüssel.

3.2 Enstehung des Schlüssel

Gehe man davon aus, dass alle Werte x_i den Schlüssel bilden. Sobald also ein Teil oder der ganze Schlüssel bekannt ist, sind ebenfalls die entsprechenden Werte aus der Menge der x_i bekannt. Mit mathematischen Mittel können alle Parameter und damit auch der *seed* gefunden werden. Um diesem Sicherheitsproblem zu entgehen, wird dem Schlüssel nur die rechte Bithälfte von jedem Wert x_i zugeordnet. Der Generator arbeitet jedoch mit dem ganzen Werten von x_i verborgen weiter. Beispiel:

$$B_{x_i} = 10011111$$

$$B_{x_s} = 1111$$

Der Schlüssel s besteht demnach aus allen x_s, die aneinandergereiht werden.

$$s = x_{s_1}, x_{s_2}, x_{s_3}, x_{s_4}, ...$$

3.3 Abscheinden der linken Bithälfte

Der Wert von x_s entsteht durch eine Art Maske oder Folie, die über x_i gelegt wird. Mit diesem Instrument lass sich nur noch die rechte Bithälfte von x_i auf x_s abbilden. Programmiertechnisch werden die Bits von der Maske mit den von x_i verglichen. Wenn beide an der gleichen Stelle den gleichen Bitwert haben, wird dieser übernommen, ist dies nicht der Fall gilt für x_s an der Stelle 0. Um nur die rechte Bitseite zu übernehmen, muss also eine gleichlange Bitmaske m erzeugt werden, die von links bis zur Mitte aus Nullen und von der Mitte bis rechts aus Einsen besteht. Beispiel:

$$B_{x_i} = 11111001SS$$

$$B_{x_i} = 00001111$$

$$B_{x_s} = 00001001$$

4 Matrix

Zur Erzeugung des Seeds wird eine 4×4 Matrix verwendet, die aus folgenden Teilen besteht:

- einer öffentlichen Konstante,
- dem geheimen Schlüssel,
- der öffentlichen Blocknummer und
- einer ebenfalls öffentlichen Nonce.

Die Nonce ist eine „**N**umber used **ONCE**“, die pro Verschlüsselung neu gewählt werden muss. Hierdurch werden Replay-Angriffe praktisch unmöglich, da ein so ein kompromittierter Stromschlüssel nie erneut eingesetzt werden kann.

Die öffentliche Konstante entspricht der ASCII-Repräsentation von

„ILoveYou“.

Die Elemente der Matrix sind 16-bit Wörter und sind im Ausgangszustand wie folgt angeordnet:

const	const	const	const
key	key	key	key
key	key	key	key
block_num	block_num	nonce	nonce

Diese Matrix durchläuft 30 Runden an Diffusion, wodurch eine geringfügige Änderung eines der Eingangswörter die gesamte Ausgangsmatrix beeinflusst. Hierdurch wird der Rückschluss auf die Eingangswörter mithilfe des generierten Seeds praktisch unmöglich.

4.1 Runden

Jede Runde besteht aus acht Rechenschritten. Für alle werden vier Wörter aus der Matrix ausgewählt und dem in Unterunterabschnitt 4.1.1 beschriebenen Verfahren übergeben. Die Ausgaben dieser Berechnung ersetzten die entsprechenden Eingangswerte.

Die Auswahl der vier Wörter ist vordefiniert, mit einer zeilenweisen Nummerierung der Wörter ergeben sich die folgenden Rechenschritte, die in zwei Teile unterteilt werden können und in der gelisteten Reihen folge ausgeführt werden:

1. Spalten:
   - Achtelrunde(00, 04, 08, 12)
   - Achtelrunde(01, 05, 09, 13)
   - Achtelrunde(02, 06, 10, 14)
   - Achtelrunde(03, 07, 11, 15)
2. Diagonalen:
   - Achtelrunde(00, 05, 10, 15)
   - Achtelrunde(01, 06, 11, 12)
   - Achtelrunde(02, 07, 08, 13)
   - Achtelrunde(03, 04, 09, 14)

4.1.1 Achtelrunden

Jede Achtelrunde nimmt vier 2-bit Wörter (a bis d) entgegen und produziert ebenfalls vier 2-bit Wörter. Das Ziel dieses Verfahren ist es, alle Eingangswörter derartig miteinander in Beziehung zu setzten, dass eine kleine Änderung eines Eingangswortes alle Ausganswörter stark beeinflusst. Hierzu werden die folgenden Berechnungen durchgeführt:

$$
\begin{aligned}
a &+= b \\
d &\wedge= b \\
d &<<<= 6
\end{aligned}
$$

$$
\begin{aligned}
c &+= c \\
b &\wedge= b \\
c &<<<= 6
\end{aligned}
$$

$$
\begin{aligned}
a &+= b \\
d &\wedge= b \\
d &<<<= 6
\end{aligned}
$$

$$
\begin{aligned}
c &+= d \\
b &\wedge= c \\
c &<<<= 6
\end{aligned}
$$

Es wird die übliche Schreibweise der C Programmiersprache verwendet, die ebenfalls in [J.08] eingesetzt wurde: $\wedge$ steht für XOR, $+$ für Addition modulo 2^{16} und $<<< b$ bitweise Rotation um b-bits nach links.

5 Hashfunktion

Zur Generation des Schlüssels wird eine Hashfunktion verwendet. Sie muss nicht besonders sicher und unumkehrbare sein, da die Matrix diese Aufgabe bereits übernimmt. Der Zweck dieser ist statt dessen die Reduzierung, beziehungsweise Erweiterung, einer Passphrase auf die von der Matrix geforderten 128 Bits. Daher wird das sehr simple Division-Rest-Verfahren angewendet.

Sei s_0 bis s_{n-1} die ASCII-Repräsentation einer Passphrase der Länge n, h der zu generierende Hashwert und $p = 2^{128}$ die Begrenzung des Hashwertes, die von der Matrix vorgegeben ist. Es gilt:

$$h = \left(s_0 \cdot 31^{n-1} + s_1 \cdot 31^{n-2} + ... + s_{n-2} \cdot 31 + s_{n-1}\right) \mod p$$

6 Implementation

Das genannte Verfahren wird in Java implementiert. Im Folgenden werden auf Implementationsdetails eingegangen.

6.1 Hashfunktion

6.2 Matrix

6.3 Pseudozufallsgenerator

Der Generator wird in einer Methode, die den Seed als Parameter bekommt generiert.. Es wird jeder letzte Wert von x_i, aus dem immer der nächste folgt, gespeichert. Dies ermöglicht es einen n langen Blockschlüssel zu erzeugen.

7 Analyse

8 Beispiele

9 Zusammenfassung

Keygarantie ist ein experimentelles, nicht empfohlenes, Verschlüsselungsverfahren, das viele Aspekte von ChaCha übernimmt und abwandelt. Für die praktische Verwendung im Secure Socket Layer ist es nicht empfohlen, da weitreichende Kryptoanalysen noch nicht durchgeführt aber für dieses Anwendungsgebiet notwendig wären.

Ausschließlich für die in RFC 1149 standardisierte Übertragung von IP Datagrams [Wai90] ist dieses Verfahren vertretbar und empfohlen.

10 Literatur

[J.08] Bernstein Daniel J. Chacha, a variant of salsa20. April 2008.

[Wai90] David Waitzman. Standard for the transmission of IP datagrams on avian carriers. RFC 1149, April 1990.