Cryptographic Splitting

als einfache Methode zur Datensicherung und gesicherten Datenübertragung

FH-Prof. DDr. techn. Dr.-Ing. Gernot Kucera MA, FH Campus Wien, Department Technik

Abstract

In dieser Arbeit wird ein Algorithmus vorgestellt, der zur Verbesserung der Datensicherung und für die gesicherte Datenübertragung genutzt werden kann. Motivation für diese Untersuchung war die Suche nach einem Verfahren, das sich vergleichsweise einfach implementieren lässt und auf dem Prinzip des kryptographischen Teilens von Informationen beruht.

Untersucht werden Fragen, mit welchen Konzepten sich das Aufteilen von Daten möglichst einfach realisieren lässt. Der daraus entstandene auch mehrfach anwendbare Algorithmus für das Verschlüsseln bewirkt die Umordnung der Quellinformation auf Bit-Ebene, der Algorithmus für das Entschlüsseln die Rekonstruktion der Quellinformation.

Keywords

Cryptographic splitting, Datensicherung, Algorithmus Verschlüsseln, Algorithmus Entschlüsseln

1 EINLEITUNG

Datensicherung und gesicherte Verfahren zur Datenübertragung sowohl im privaten als auch im kommerziellen Umfeld und insbesondere für Daten, die beispielsweise zur Identifizierung von Personen dienen könnten, wird immer wichtiger. Zu Zielen kryptographischer Verfahren zählen unter anderem die Vertraulichkeit von Daten sicherzustellen.

Zu den fundamentalen Problemstellungen der modernen Kryptographie zählen unter anderem die Handhabung der Schlüssel, deren Länge und wie diese geheim gehalten werden können. Für kryptographische Verfahren sind im Laufe der letzten Jahre auch Standards entwickelt worden, die hier nicht weiter behandelt werden.

Motivation für diese Untersuchung war ein vergleichsweise einfach zu implementierendes Verfahren in der Form von Algorithmen vorzustellen, das auf dem Prinzip des Teilens von Informationen beruht. Gesucht werden dazu Konzepte einerseits zum Aufteilen von Daten und andererseits zu deren Rekombination sowie nach der Vorgehensweise für die Realisierung.

Cryptographic splitting

Cryptographic splitting, also das kryptografische Aufteilen, ist eine Technik zum Sichern von Daten, die als „Secure data parser method and system“ als US-Patent 7391865 patentiert worden ist. Ursprünglich war laut der Patentbeschreibung vorgesehen, Daten in mehrere kleinere Dateneinheiten aufzuteilen und diese an verschiedenen Speicherorten zu speichern. Die Idee dabei war, dass falls es einem Eindringling gelingen sollte eine Dateneinheit abzurufen und zu entschlüsseln, die Informationen solange unbrauchbar bleiben, bis sie mit den anderen ebenfalls entschlüsselten Dateneinheiten von weiteren Speicherorten rekombiniert worden sind (vgl. [1]). Bei der Konferenz der Speicherentwickler 2009 wurde diese Methode als kryptografisches Splitten zur Datensicherung diskutiert (vgl. [2]).

Beitrag im Rahmen des 16. Forschungsforums der österreichischen Fachhochschulen von 19.-20. April 2023 an der FH St. Pölten.

Verwandte Arbeiten

Die Arbeit S. Dey und A. Nath (vgl. [3]) nutzt eine ähnliche Methode zur Datensicherung in der Cloud, ist aber im Gegensatz zu dem hier verfolgten Ansatz vergleichsweise aufwändig. Lidia Ogiela behandelt in ihrer Arbeit (vgl. [4]) Informationsteilungsalgorithmen zum Austausch geheimer Informationen durch Aufspalten, um die Vertraulichkeit solcher Daten zu verbessern und zur Datensicherung. Auch Balasaraswathi V.R. und Manikandan S. konzentrieren sich in ihrer Arbeit (vgl. [5]) auf die Frage, wie Datensicherheit bei der Übertragung erreicht werden kann. In ihrem Verschlüsselungsstandardalgorithmus wird die ursprüngliche Information geteilt und getrennt mit zwei unterschiedlichen Algorithmen verschlüsselt, die beim Empfänger rekombiniert werden.

Im Gegensatz zu den genannten Ansätzen bewirken die in der Folge entwickelten Algorithmen ausschließlich die Umordnung der Quellinformation auf Bit-Ebene und deren Rekonstruktion.

2 METHODEN

Der generelle Ansatz ist das Aufteilen der Quelldaten auf zwei Datensätze, die jeweils Teile der Information der Quelle enthalten. Das Prinzip der „Verschlüsselung“ beruht auf der Manipulation der darin enthaltenen Datenfragmente. Die Größe dieser Fragmente beschränkt sich auf ein Byte bzw. Vielfache davon.

Die Hauptfunktionen „split“ und „merge“ sowie „crunch“ und „expand“ bilden das Gerüst für die genutzten Algorithmen. Dabei wird die Funktion „split“ für das Aufteilen der Quelldaten genutzt, „merge“ für die Rekombination der Informationen und ergänzend „crunch“ sowie „expand“ für weitere Datenmanipulationen.

Die Funktion „split“ kann mit einem sich öffnenden Reißverschluss assoziiert werden (Fig. 1). Dargestellt ist, wie ein einfacher Text „Hello World“ durch die Teilung der in ASCII-Zeichen dargestellten Buchstaben des Textes in zwei Teildatenbereiche aufgetrennt werden kann. Es entstehen zwei Teildatenströme, die beim Empfänger der Nachricht mit „merge“ wie ein Puzzle wieder zusammengesetzt werden können (Fig. 2).

Fig. 1: Prinzip der Funktion „split“

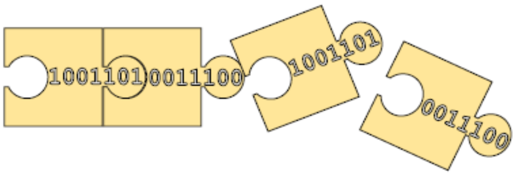
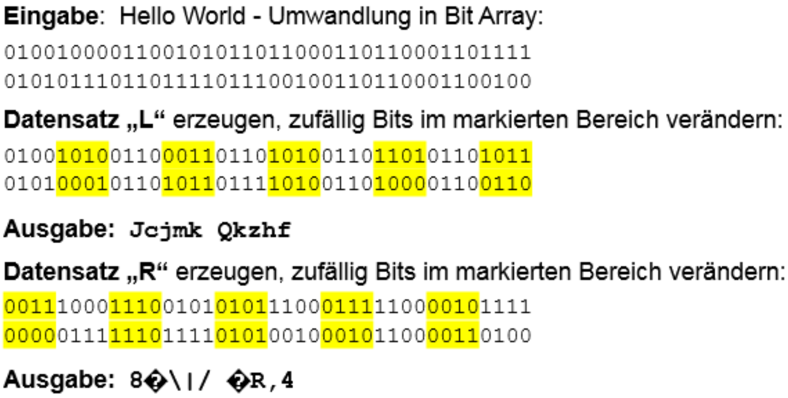


Fig. 2: Prinzip der Funktion „merge“

In der Folge wird die Funktion von „split“ anhand des Textes „Hello World“ graphisch verdeutlicht (Fig. 3). Zuerst wird der Text als Bitmuster seiner ASCII-Zeichen dargestellt (Eingabe). Dem Beispiel ist der Code 11110000 zugrunde gelegt, das bedeutet, dass die Aufteilung in Gruppen zu je 4 Bit erfolgt. Daraus entsteht der Datensatz „L“ (im Beispiel jeweils die ersten 4 Bit) und der Datensatz „R“ (die restlichen 4 Bit). Die Füllinformationen sind die farbig gekennzeichneten Bereiche und werden in diesem Beispiel mit zufällig erzeugten Bitmustern befüllt.

Damit werden die ursprünglichen Zeichen verfälscht. Die daraus resultierenden Ausgaben sind unbrauchbar, solange nicht bekannt ist, wie und wo Nutzinformationen lokalisiert sind.



Eingabe: Hello World - Umwandlung in Bit Array:

```
01001000011001010110110001101100011011110
01010111011011110111001001101100011001000
```

Datensatz „L“ erzeugen, zufällig Bits im markierten Bereich verändern:

```
0100101001100011011010100110110101101011
0101000101101011011110100110100001100110
```

Ausgabe: `Jcjmk Qkzhf`

Datensatz „R“ erzeugen, zufällig Bits im markierten Bereich verändern:

```
0011100011100101010111000111110000101111
0000011111101111010100100010110000110100
```

Ausgabe: `8�\|/ �R,4`

Fig. 3: Verschlüsselung der ASCII-Zeichen des Textes „Hello World“ als Bitmuster

2.1 Entwicklung der Algorithmen und Tests

Um die Wirkungsweise des Systems zu untersuchen wurden mehrere einfache Testprogramme entwickelt. Für das erste Testprogramm wurde als Teilungsparameter für die Funktion „split“ ein Byte (8 Bit) für das Auftrennen der Nutzinformation festgelegt. Daraus ergibt sich als Steuergröße für den „Schlüssel“ ebenfalls nur 8 Bit, wobei der einmal festgelegte „Schlüssel“ konstant bleibt und für jedes Byte der Quelle genutzt wird.

Eine Länge des Schlüssels von nur 8 Bit? Hier könnte sofort gefragt werden, ob eine derart einfache Verschlüsselung von Nachrichten als ausreichend betrachtet werden kann. Zu bedenken ist, dass dieser Schlüssel auf jedes Byte einer Nachricht angewendet wird. Die Schlüssellänge von 8 Bit oder genauer gesagt von x mal 8 Bit hat sich für die weiteren Untersuchungen als praktisch erwiesen und wurde deshalb beibehalten.

In den ersten Testversuchen wurde der Schlüssel entsprechend der Länge der Quelle dupliziert. In den weiteren Versuchen wurde die Schlüssellänge mehrfach erweitert, indem zuerst Schlüssel für Bytefolgen eingeführt und getestet worden sind. Im Prinzip kann deshalb jedem Byte seine individuelle Schlüsselinformation zugewiesen werden. Da jedoch eine Datei deutlich größer als 256 Bytes groß sein wird, existieren in der Regel mehrere Duplikate der Teil-Schlüssel mit der Größe eines Bytes.

Wie weiter oben ausgeführt entstehen durch die Funktion „split“ zwei Datenfragmente, die in der Folge als „linke“ bzw. „rechte“ Datei bezeichnet sind. Verknüpft man als Filterfunktion jedes Byte einer Quelle mit dem zugehörigen als Schlüssel festgelegtem Byte logisch UND, bedeutet das, dass die Bits an bestimmten Stellen unverändert übernommen werden. Die restlichen Stellen im Byte werden mit NULL überschrieben. Das aus der Zusammensetzung aller Bytes entstandene Bitmuster ist die „linke“ Datei. Die „rechte“ Datei wird mit dem invertierten Schlüssel erzeugt.

„Linke“ und „rechte“ Datei können auf verschiedenen Kanälen oder getrennt zu einem Empfänger übertragen werden und mit Kenntnis des Schlüssels mit logisch ODER zur ursprünglichen Datei zusammengesetzt werden („merge“).

Bei dieser Methode entstehen zwei gleich lange Dateien, die jeweils aus einer Durchmischung von Nutzinformationen und Füllinformationen bestehen. In den Funktionstests wurde untersucht, wie Füllinformationen manipuliert werden können, um die Verschlüsselung zu verbessern. Da die Füllinformationen zur Übertragung der Nutzinformationen keinen Beitrag leisten, wurden die Funktionen „crunch“ und „expand“ eingeführt. „Crunch“ verwirft Füllinformationen und schließt die entstehenden Lücken, deren Positionen durch den Schlüssel bestimmbar sind. „Expand“ ist die inverse „crunch“ Funktion und stellt die entfernten Lücken durch das Auffüllen mit NULL wieder her. Damit bleibt „merge“ eine einfache logische ODER Funktion, die bitweise aus den rekonstruierten „linken“ und „rechten“ Dateien die Quelldateien nach deren Übertragung rekonstruieren kann.

2.2 Algorithmen des Testprogramms

Für die Tests wurden mehrere Varianten von Testprogrammen genutzt. Das zuletzt genutzte Testprogramm folgt schrittweise folgenden Algorithmen:

Algorithmus Verschlüsseln

(1) Einlesen der Eingabe (Quelldatei) und des Codes (Code-Snippet) als Bitarray

(2) Erzeugen und Anpassen des Codes auf die Länge der Eingabe (durch Wiederholen und Anhängen des Code-Snippets). Daraus entsteht der „working code“ und seine Invertierung.

(3) Filterfunktion 1: Eingabe und Code werden logisch UND mit dem „working code“ sowie „invertiertem working code“ verknüpft. Daraus ergibt sich die Funktion „split“ und als Ergebnisse das „left array“ bzw. „right array“.

(4) Filterfunktion 2: Aussortieren der durch Filter 1 ausgesonderten nicht genutzten Bereiche im „left array“ und „right array“ und zusammenschieben der entstandenen Lücken. Ergebnisse sind die Elemente „crunch left“ bzw. „crunch right“

(5) Konkatenation von „crunch left“ mit „crunch right“ mit Bestimmen der Verbindungsstelle. Die Verbindungstelle muss als Zahlenwert mitübertragen werden, da je nach Codewort auch eine unsymmetrische Aufteilung erfolgt sein kann und damit die Arrays „crunch left“ bzw. „crunch right“ verschieden lang sein können.

Anmerkung: Um den Typ der zu verschlüsselnden Information (z.B. für die Datei „Text.txt“) für die Entschlüsselung bereitzustellen, wird der ursprüngliche Dateiname (hier „Text“) und der Dateianhang (hier „txt“) zusammen mit einem Zahlenwert (als die Verbindungsstelle von „crunch_left“ mit „crunch_right“) an die gesplitteten Teile „left array“ und „right array“ jeweils als Dateianhang derart angehängt, sodass als Ergebnis der Verschlüsselung die Dateien „crunch_left.Text“ bzw. „crunch_right.txt_Zahl“ entsteht.

Zuletzt werden als Ergebnis für die Ausgabe das Bitarray „crunch_left_right“ durch Aneinanderreihen von „crunch left“ und „crunch right“ gebildet.

Algorithmus Entschlüssen

(1) Einlesen der Eingabe (als Bitarray „crunch_left_right“). Der Code muss dem Empfänger bekannt sein und wird als Bitarray (Code-Snippet) eingelesen.

Anmerkung: Die Länge von „crunch_left_right“ stimmt mit der Länge der Eingabedatei überein, da nur eine Umordnung der Bits erfolgt ist.

(2) Aufteilen der Eingabe in „crunch left“ und „crunch right“ an der Verbindungsstelle.

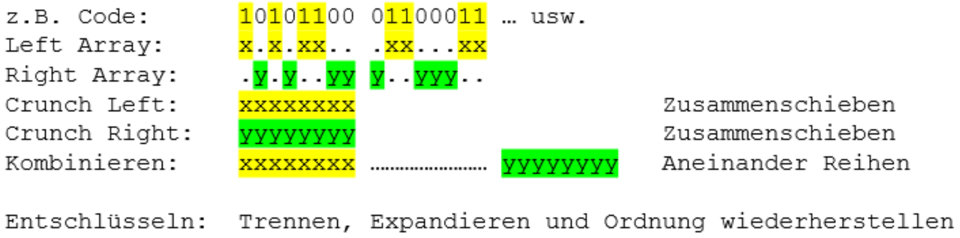
(3) Mit Hilfe der Länge der Eingabedatei wird der Code auf die Länge der Eingabe expandiert (Wiederholen und Anhängen des Code-Snippets). Als Ergebnis stehen der „working code“ und seine Invertierung zur Verfügung.

(4) Filterfunktion 3: Die Funktion „recover“ ergänzt entsprechend des „working code“ und invertiertem „working code” die Teile „crunch left“ mit „crunch right“ zur ursprünglichen Länge der Bitarrays zum „left array“ bzw. zum „right array“

(5) Filterfunktion 4: Die Funktion „merge“ verschmilzt „left array“ bzw. „right array“ durch logisch ODER. Die beiden Bitarrays werden zur ursprünglich übermittelten Quelldatei ergänzt und stellt damit die Quelle wieder her (unter Berücksichtigung des Dateinamens und des Dateianhangs).

3 ERGEBNISSE

Die Wirkungsweise des Algorithmus wird in Fig. 4 verdeutlicht. Als Beispiel wird eine Bitfolge für den Code angenommen, in dem die Bit mit dem Wert „1“ farbig hervorgehoben sind. Die Funktion „split“ teilt die zu verschlüsselnde Datei derart, dass im „left array“ nur jene Werte aufscheinen, bei denen das korrespondierende Bit im Code den Wert „1“ zeigt. Diese Bits werden im „left array“ mit „x“ bezeichnet. Analog wird das „right array“ gebildet, worin alle Werte, die im Code mit „0“ gekennzeichnet sind, gesammelt werden. Diese sind in Fig. 4 mit „y“ bezeichnet. Mit der Funktion „crunch“ werden die Werte zusammengeschoben. Damit entstehen zwei neue Ordnungen von Bitketten, die mit „merge“ zur verschlüsselten Datei aneinandergereiht werden.



```
z.B. Code:      10101100 01100011 … usw.
Left Array:     x.x.xx.. .xx...xx
Right Array:    .y.y..yy y..yyy..
Crunch Left:    xxxxxxxx                     Zusammenschieben
Crunch Right:   yyyyyyyy                     Zusammenschieben
Kombinieren:    xxxxxxxx ………………… yyyyyyyy   Aneinander Reihen

Entschlüsseln:  Trennen, Expandieren und Ordnung wiederherstellen
```

Fig. 4: Wirkungsweise des Algorithmus

Anmerkung zum Code: die Kenntnis, wie viele Bit im Code den Wert „1“ haben und wie viele mit „0“ gekennzeichnet sind, ist nicht beschränkt. Es ist aber klar, dass eine gute „Durchmischung“ der Informationen aus der Quelldatei dann gegeben sein wird, wenn die Summen von „1-Bits“ und „0-Bits“ die gleiche Größenordnung aufweisen. Sind diese Summen gleich, wird damit die Trennstelle von „crunch left“ und „crunch right“ genau in die Mitte der verschlüsselten Datei gelegt.

Anmerkung zur Verschlüsselung: mehrfaches Verschlüsseln, sogar unter Verwendung desselben Codes ist ebenfalls möglich. Durch das nochmalige Verschlüsseln einer bereits verschlüsselten Datei erfolgt eine weitere Umordnung der Informationen der Quelle.

Die Tests wurden auf einem älteren Laptop mit vergleichsweiser geringer Performance durchgeführt. Laut Systemklassifikation beträgt der angezeigte „Windows-Leistungsindex“ nur den Wert 3,4. Der Prozessor ist ein Intel® Core™ 2 Duo P8600 2,40 GHz 2,40 GHz und arbeitet unter Windows 7 als Betriebssystem. Das zuletzt genutzte Testprogramm wurde mit Python entwickelt.

Um einen Eindruck zur Performance zu gewinnen, wurde die Zeitdauer für das Verschlüsseln von verschiedenen Dateien gemessen.

Bei Dateigrößen und Dateitypen (Adobe Acrobat Document (.pdf), Größe: 1,34 MB, JPEG-Bild (.jpg), Größe: 52,4 KB und Microsoft Word-Dokument (.docx), Größe: 79,3 KB) wurden für das Verschlüsseln rund 13 µs/Byte benötigt, für das Entschlüsseln rund 24µs/Byte.

Alle getesteten Dateien wurden nicht auf einen anderen Rechner übertragen, sondern nur zwischengespeichert, und konnten problemlos wiederhergestellt werden.

In einer weiteren Testreihe wurden bereits verschlüsselte Dateien ein weiteres Mal verschlüsselt und wieder rekonstruiert. Die Zeitdauer verlängert sich entsprechend. Ob eine mehrfache Verschlüsslung die Sicherheit erhöhen kann, müsste noch untersucht werden.

4 CONCLUSIO

Zuletzt drei Fragen und Antworten als „frequently asked questions“:

Was müssen Empfänger über verschlüsselte Nachrichten wissen, um diese in Klartext nutzen zu können? Im hier gegebenen Fall sind das die Nachricht selbst und der zugehörige Schlüssel.

Was sind im gegebenen Fall Schlüsselinformationen?
Gemeinsam vereinbarte und damit festgelegte Bitmuster. Anzumerken ist, dass es keine Einschränkungen für solche Bitmuster gibt.

Gibt es Einschränkungen bezüglich der Nachrichten oder zu den Schlüsselinformationen?
Abgesehen von der Art der vorgesehenen Verschlüsselung und dem Umstand, dass Schlüsselinformationen geheim gehalten werden müssen, gibt es keine Einschränkungen.

Daraus kann gefolgert werden, dass, wenn die Art der Verschlüsslung nach obiger Methode erfolgt ist, auch etwa ein Text aus der klassischen Literatur oder ein Foto als Bilddatei in der Form ihrer Bitmuster als Schlüssel genutzt werden können.

Die hier beschriebenen Algorithmen bewirken eine auch mehrfache Umordnung auf Bit-Ebene der Quellinformationen. Festzuhalten ist, dass die Bits der Quelldatei bei dieser Methode nach dem Verschlüsseln in der verschlüsselten Datei unverändert enthalten bleiben, diese jedoch in ihrer Reihenfolge in verschiedenen Bereichen „verstreut“ angeordnet werden. Mehrfaches Verschlüsseln bedeutet in diesem Zusammenhang den Grad der „Verstreuung“ der Bits zu erhöhen.

5 REFERENZEN

[1] Orsini, Rick L., Vanzandt, John, O'Hare, Mark S., Davenport, Roger S.: “Secure data parser method and system”, US patent: 7391865, 24. Juni 2008, https://www.freepatentsonline.com/7391865.html, Abgerufen 14.03.2023

[2] Dodgson, David: “Storage Security Using Cryptographic Splitting”, Storage Developer Conference 2009, Santa Clara, 2009, https://www.snia.org/sites/default/orig/sdc_archives/2009_presentations/wednesday/DavidDodgson_StorageSecurityUsingCryptographicSplitting-BU.pdf , Abgerufen 14.03.2023

[3] Dey, S. and Nath, A.: "Modern Encryption Standard (MES) version-I: An advanced cryptographic method," 2012 World Congress on Information and Communication Technologies, Trivandrum, 2012, pp. 242-247, doi: 10.1109/WICT.2012.6409082.

[4] Ogiela, Lidia, “Advanced techniques for knowledge management and access to strategic information”, International Journal of Information Management, Volume 35, Issue 2, 2015, Pages 154-159, ISSN 0268-4012, https://doi.org/10.1016/j.ijinfomgt.2014.11.006.

[5] Balasaraswathi V.R. and Manikandan S.: "Enhanced security for multi-cloud storage using cryptographic data splitting with dynamic approach," 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, Ramanathapuram, 2014, pp. 1190-1194, doi: 10.1109/ICACCCT.2014.7019286.