Blockchiffren Teil 1

Krypto II

VL 11

08.11.2018

Buch der Woche

Titel: Cryptography Engineering

Autor(en): Niels Ferguson, Bruce Schneier, Tadayoshi Kohno

Verlag: Wiley

Umfang: ca. 350 Seiten

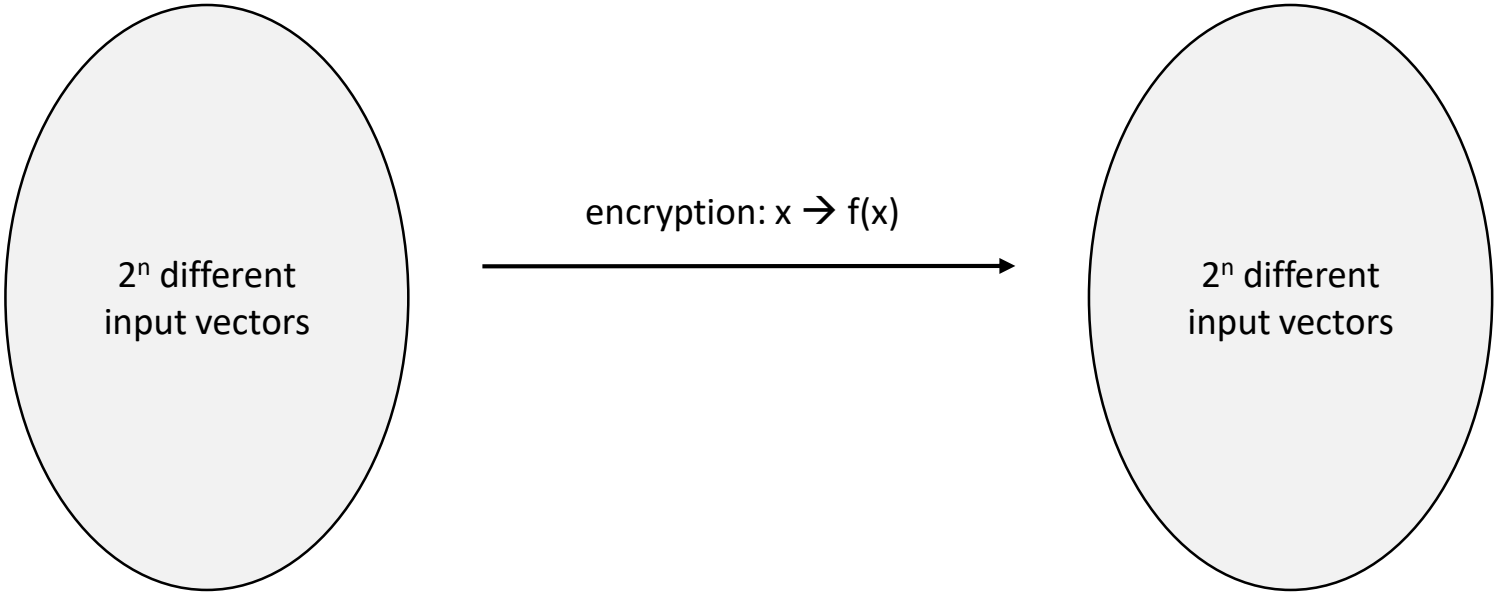
Hinweise zum Inhalt:

Das Buch wurde von ausgewiesenen Krypto-Experten geschrieben, die nicht nur die technischen Grundlagen vermitteln können, sondern zugleich auch ihre Erfahrungen in der praktischen Umsetzung einfliessen lassen. So finden sich zwar keine Code-Beispiele, aber gelegentliche Warnungen hinsichtlich häufig gemachter Anwendungsfehler. Folgerichtig schließt as Buch ab mit einem Kapitel namens “The Dream of PKI”, in dem die Umsetzungshürden bei der Implementierung einer Public Key Infrastruktur beschrieben werden. Insgesamt ein Buch, das in keiner Krypto-Library fehlen sollte. Gut verständlich geschrieben, wenig Mathematik und Formelbalast, und dennoch nicht zu oberflächlich.

Anmerkung: die erste Auflage des Buches hieß “Practical Cryptography”.

Blockchiffren und Permutationsgruppen

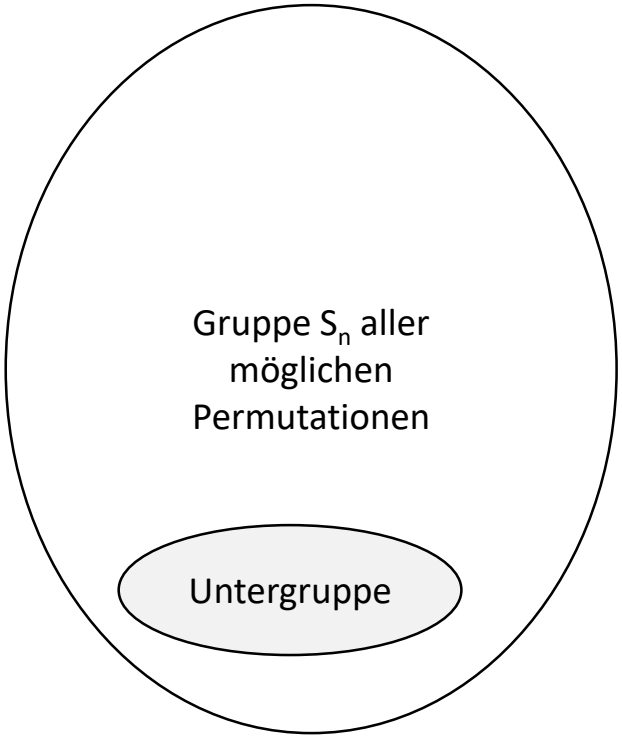
Blockchiffren als Permutation



- die Abbildung f auf der endlichen Menge der Inputvektoren ist bijektiv
- dies nennt man auch eine Permutation
- Die Menge aller Permutationen auf einer Menge bilden eine Gruppe
- jede Permutation kann dargestellt werden als Kombination “erzeugender” Permutationen
- jede gegebene Teilmenge von Permutationen erzeugt eine bestimmte (Unter-)Gruppe, die allerdings deutlich kleiner sein kann als die symmetrische Gruppe aller Permutationen

Wozu Gruppentheorie in Kryptographie? Wozu in Blockchiffren?

- endliche nichtabelsche Gruppen haben interessante algebraische Eigenschaften!
- was war nochmal “abelsch” bzw. “nichtabelsch”?
- Permutationsgruppen können ganz unterschiedlich große und unterschiedlich strukturierte Untergruppen besitzen
- Angenommen wir haben einen aus mehreren Permutationen zusammengesetzten Verschlüsselungs-Algorithmus. Falls diese Permutationen nur eine (kleine) Untergruppe erzeugen, dann können wir diese so oft hintereinander ausführen, wie wir wollen, doch wir bleiben stets gefangen in der Untergruppe!



Codebreaker haben stets zwei Ziele:

1. Einschränkung des Suchraums

Wenn wir einen riesige Suchraum durchkämmen müssen, um einen bestimmten Kryptoschlüssel oder ein Hash Pre-Image zu finden, dann benötigen wir clevere Strategien zur Verkleinerung des Suchraums, um einen praktikablen Angriff zu ermöglichen.

➔ In unserem Fall besteht der reduzierte Suchraum aus einer versteckten Untergruppe!

2. Einschränkung der Komplexität

Zum besseren Verständnis eines Krypto-Algorithmus versucht man, dessen Funktionalität in eine weniger komplexe Darstellungsform zu bringen.

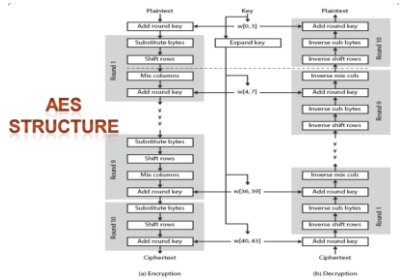
Die erreichte Vereinfachung kann sodann zu Einsichten und Angriffen führen, die zuvor nicht ersichtlich waren.

➔ In unserem Fall besteht die Komplexitäts-Reduktion aus der Identifizierung einfacher, die gesamte Gruppe erzeugender Permutationen!

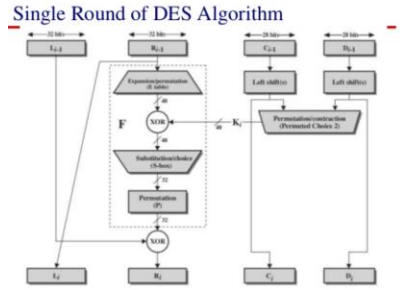
Blockchiffren sind komplex…

- Frage: wer könnte RSA oder Diffie-Hellman aufschreiben? OK. Aber: wer könnte AES aufschreiben?
- Blockchiffren können bestehen aus Substitution-Permutation Networks, Feistel, Modularer Multiplikation, Replacement Tables, diversen Rundenschlüsseln, und so weiter...
- Folglich ist es schwierig, deren Sicherheit zu bewerten, Robustheit gegen unterschiedliche Angriffe, und eine effiziente Implementierung kann ebenfalls komplex sein.
- Aber: jede Blockchiffre ist eine bijektive Abbildung auf einer endlichen Menge, also nichts anderes als eine Permutation der Input-Vektoren.
- Dies führt zur folgenden Frage:

Kann man mit einfachen, wenigen Abbildungen **jede** denkbare bijektive Abbildung (und somit Blockchiffre) realisieren?



ist das nicht schon Security by obscurity?



Frage: was sind Beispiele für **sehr einfache** Funktionen in Blockchiffren?

Unser Toolset

- Betrachten wir die folgenden wichtigen Basiskomponenten:
  - Addition mod 2^n (ADD)
  - Bitwise Rotation (ROT)
  - Multiplication mod 2^n (MULT)
  - XOR
- Sind wir in der Lage, **jede mögliche bijektive Abbildung** zu realisieren als Kombination der obigen vier Basiskomponenten?
- Wieviele und welche davon – falls überhaupt – könnten ausreichen, um die gesamte symmetrische Gruppe aller Permutationen bzw. Bijektionen zu erzeugen?
- Dann könnten wir jede Blockchiffre als Kombination solch einfacher Basiskomponenten darstellen!

Erzeugung der Symmetrischen Gruppe

- **Satz**: Sei S_n die symmetrische Gruppe mit n Elementen und sei $x \in S_n$ ein beliebiges Element ungleich dem neutralen Element. Dann existiert, sofern $n \neq 4$, stets ein $y \in S_n$, so dass x und y die gesamte symmetrische Gruppe S_n erzeugen.
  ➔ Zwei Elemente reichen also bereits aus, um jede mögliche Permutation einer gegebenen endlichen Menge zu erzeugen!
- Betrachten wir zum Beispiel eine einfache Rotation ROT. Nach obigem Satz wissen wir, dass eine Permutation $p \in S_n$ existiert, so dass <p, ROT> die gesamte symmetrische Gruppe aller Permutationen erzeugen

  Aber: diese Abbildung p muss nicht unbedingt die Form einer unserer simplen Basiskomponenten ADD, ROT, XOR, MULT aufweisen.
  Insofern stellt sich die interessante Frage, ob es evtl. ein p aus obiger Menge gibt, so dass jede bijektive Abbildung mittels <p, ROT> erzeugt werden kann?
  a) wie wäre es mit XOR?
  b) wie wäre es mit ADD?

Nicht jedes Permutations-Paar taugt

- ROT und **ADD** sind ausreichend, um **jede** mögliche Bijektion (also auch Blockverschlüsselung) zu implementieren!
- Rot und XOR hingegen können nur eine kleine Untergruppe aller Permutationen erzeugen. Es zeigt sich, dass das Carry Bit bei der Addition den maßgebliche Unterschied ausmacht!
- Dieses “Hidden Subgroup Feature” (i.e. gefangen bleiben in einer kleinen Untergruppe statt Erzeugung der gesamten Symmetrischen Gruppe) bedeutet in der Praxis, dass wir ggf. keinesfalls alle möglichen Bijektionen untersuchen müssen.

Mix-2: Ein einfaches Beispiel

Wir betrachten eine sehr einfache Blockchiffre: sei m ein Inputvektor der Länge n, r bezeichne die Rundenanzahl, und $K = (k_1, k_2, ..., k_r)$ ein Schlüssel der Länge r. Dann definieren wir den Verschlüsselungsalgorithmus MIX-2 wie folgt:

```
FOR (i = 0; i < r; ++i)
  IF   k_i      ADD_a(m);
  ELSE          ROT_b(m);
```

wobei a und b zwei ungerade Konstanten sind, $0 < a,b < 2^n$.

$ADD_a(m)$ bedeutet Addition des Vektors m mit der Konstanten a, und $ROT_b(m)$ bedeutet Rotation (d.h. zyklischer Shift) des Vektors m um b Positionen.

Man kann beweisen, dass die Rundenfunktion von Mix-2 die gesamte symmetrische Gruppe von Grad 2^n erzeugt. Dasselbe gilt dann auch für Mix-2.

Typen von Blockchiffren

Welche Typen von Blockchiffren

Frage in die Runde:

Welche Typen von Blockchiffren kennen Sie?

Ist ein Typ besser als der andere?

Warum hat man unterschiedliche Typen?

Substitution Permutation Networks

Diese wurden bereits in Krypto 1 behandelt. Daher an dieser Stelle nur eine kurze Zusammenfassung.

Substitution-Permutation Networks (auch ***SPN*** abgekürzt) stehen für symmetrische Blockchiffren, die einem bestimmten Designprinzip folgen.

Hierbei unterscheidet man im allgemeinen zwei Ziele bzw. Kategorien, in die sich die an der Verschlüsselung beteiligten Operationen grob aufteilen lassen:

- ***Diffusion***: diese Art Verschlüsselungsoperation soll bewirken, dass sich der Einfluss individueller Input Bits möglichst breitflächig auf viele oder besser alle Outputbits erstreckt. Diffusion erreicht man beispielsweise durch Permutation der Inputbits, durch lineare Transformationen, Shifts und vieles mehr.
- ***Konfusion***: diese Art Verschlüsselungsoperation soll einen möglichst komplexen Zusammenhang zwischen Inputvektoren, Schlüssel, und Outputvektoren bewirken. Häufiges Beispiel ist die Realisierung als sogenannte S-Box. Solche S-Boxes operieren oft auf kürzeren Teilblocks (z.B. 4 oder 8 Bit) und werden mittels Lookup Tables realisiert. Beim Design von S-Boxes achtet man auf möglichst hohe Nichtlinearität und auf die Vermeidung unerwünschter statistischer Zusammenhänge.

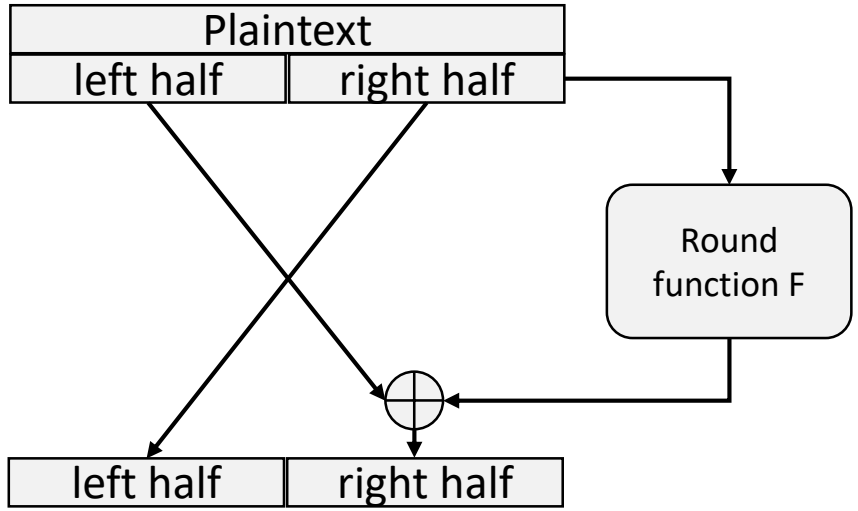
Feistel Schema

Feistel Chiffren wurden ebenfalls bereits behandelt. Daher auch hierzu wieder nur eine kurze Zusammenfassung.

Eine Feistelrunde sieht aus wie links dargestellt. Die Rundenfunktion F muss hierbei nicht bijektiv sein, die Chiffre bleibt dennoch invertierbar.

Mindestens vier Runden sind erforderlich, falls die Funktion F keine Schwächen aufweist. In der Praxis verwendet man mehr Runden.

Erfinder des Schemas war Horst Feistel. Dessen Chiffre Luzifer diente als Vorbild für den Date Encryption Standard DES.



ARX Ciphers

- Beim Design von Blockchiffren und Stromchiffren legt man Wert auf hohe Geschwindigkeit der Verschlüsselung und Entschlüsselung. Ein Designansatz besteht darin, komplette Chiffren nur aus den drei Komponenten Addition (modulo Blocklänge), Rotation und XOR zu konstruieren.
- Solche Chiffren bezeichnet man auch als ARX Ciphers, nach den drei Anfangsbuchstaben von ADD, Rotate und XOR.
- Eine eigene Unterdisziplin der Kryptographie ist die sogenannte Lightweigt cryptography für den Einsatz auf Ressourcen-schwacher Hardware (z.B. im Internet of things). Hier finden sich zahlreiche Vorschläge für ARX Chiffren.
- Das Fehlen von S-Boxen im ARX Design erfordert besondere Sorgfalt, um lineare und differentielle Attacken abzuwehren.
- Grundsätzlich kann man mittels ARX jedoch beliebig gute Chiffren konstruieren (siehe Folgeabschnitt).

Blockchiffren und Modes of Operation

Welche Betriebsmodi?

Frage in die Runde:

Welche Betriebsmodi für Blockchiffren kennen Sie?

Ist ein Typ besser als der andere?

Warum hat man unterschiedliche Typen?

Betriebsmodi - Übersicht

Blockchiffren können in verschiedenen Betriebsmodi verwendet werden. Ein paar davon haben Sie bereits kennengelernt. An dieser Stelle möchten wir nochmals detaillierter auf das Thema eingehen.

Ziel eines Betriebsmodus bzw. Cipher Block Mode ist es, die Vertraulichkeit einer Nachricht zu gewährleisten. Je nach Modus gelingt dies besser oder schlechter. Die Integrität einer Nachricht ist ein weiteres Ziel dieser Modi, wird jedoch nur teilweise erzielt.

Zur Gewährleistung der Authentizität sind zusätzliche Maßnahmen erforderlich.

NIST Special Publication 800-38A
2001 Edition

Recommendation for Block Cipher Modes of Operation

Methods and Techniques

Morris Dworkin

NIST
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

COMPUTER SECURITY

Table of Contents

ECB Electronic Codebook Mode

ECB Electronic Codebook Mode entspricht der naiven Vorgehensweise, wenn man mehr als einen Block zu verschlüsseln hat. Im Zusammenhang mit Blockchiffren haben wir diesen Mode bereits kennengelernt mit dem Hinweis, dieser sei unsicher.

Warum: “weil man den Pinguin sieht”...

Genauer: identische Chiffretexte $\Leftrightarrow$ identische Klartexte

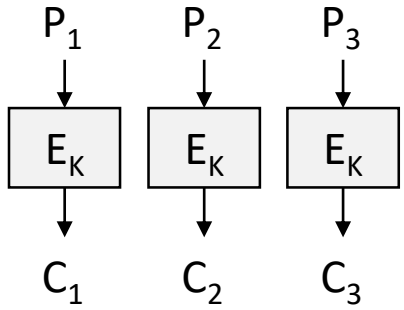
ECB scheint also höchstens dann angebracht, wenn nur ein einziger Block verschlüsselt werden soll (eher selten).

Seien P_1, P_2, P_3,... Plaintexts, ferner C_1, C_2, C_3,... Ciphertexts, K der verwendete Cipher key und E_K die zugehörige Verschlüsselungsfunktion.

Dann gilt für beliebiges i:

Verschlüsselung: $C_i = E_K(P_i)$

Entschlüsselung: $P_i = E^{-1}{}_K(C_i)$



CBC Cipher Block Chaining

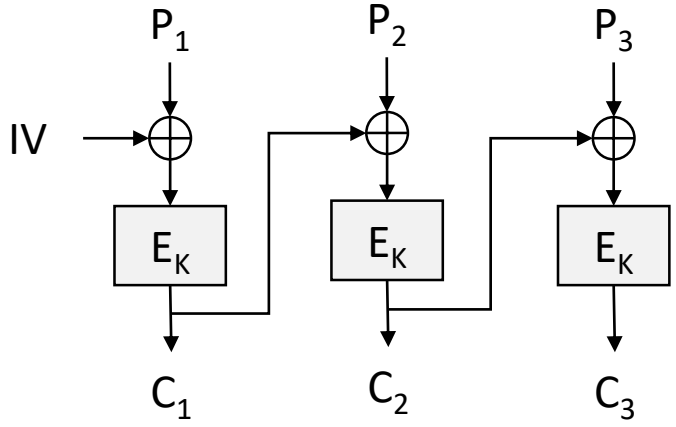
CBC Cipher Block Chaining ist nach ECB der erste und bekannteste Modus, den wir bei den Betriebsarten kennenlernen.

Verschlüsselung: $C_i = E_K(P_i \oplus C_{i-1})$

Für den ersten Block benötigen wir noch einen Initialisierungsvektor IV:

$C_1 = E_K(P_1 \oplus IV)$

Identische Klartextblöcke innerhalb des Plaintexts werden im Gegensatz zum ECB Mode nicht mehr auf identische Chiffretexte abgebildet.



CBC Entschlüsselung

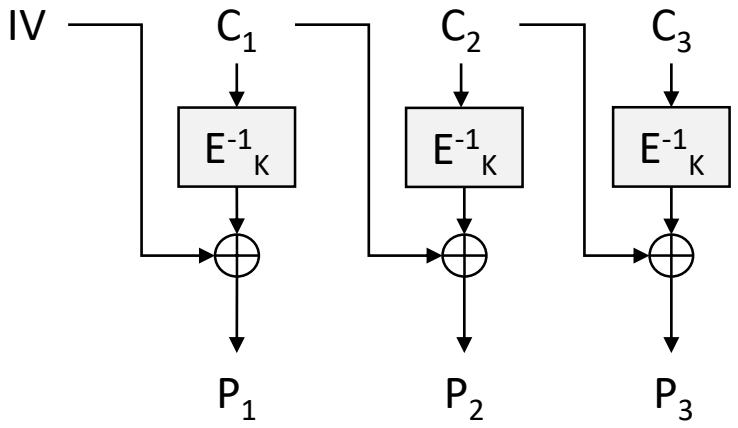
Entschlüsselung: $E^{-1}_K(C_i) = E^{-1}_K\, E_K(P_i \oplus C_{i-1}) = P_i \oplus C_{i-1} \Rightarrow$

$\Rightarrow E^{-1}_K(C_i) \oplus C_{i-1} = P_i \oplus C_{i-1} \oplus C_{i-1} = P_i$

Also erhalten wir: $P_i = E^{-1}_K(C_i) \oplus C_{i-1}$

Hieraus folgt insbesondere, dass wir an beliebigen Stellen mit der Entschlüsselung beginnen und/oder die Entschlüsselung parallelisieren können!

Bei der Verschlüsselung geht dies nicht.



CBC und Initialisierungsvektor

Der Initialisierungsvektor sollte nach jeder CBC-Anwendung gewechselt werden. Anderenfalls wird ein zweiter Plaintext, der ebenfalls (mit gleichem Cipher key K) CBC-verschlüsselt wird, auf denselben Ciphertext wie zuvor abgebildet, solange, bis sich die beiden Plaintexts zu unterscheiden beginnen.

Der Initialisierungsvektor IV muss beiden Parteien bekannt sein. Er sollte jedoch geschützt übertragen werden. Dadurch verhindert man beispielsweise, dass ein Angreifer einen veränderten IV an den Empfänger sendet. Entschlüsselt der Empfänger den ersten Block C_1, so berechnet er

$P_1 = E^{-1}_K(C_1) \oplus IV$

Ein Angreifer könnte also gezielt einzelne Bits kippen im ersten Klartextblock P_1.

Wahl des Initialisierungsvektors (1)

Wir haben gesehen, dass der IV nach jedem CBC-Einsatz geändert werden muss. Anderenfalls haben wir ein ähnliches Problem wie bei ECB. Doch wie wählen wir den IV? Und wie übertragen wir diesen zum Empfänger?

Man könnte den IV erzeugen, indem beide Kommunikationspartner den Verschlüsselungsalgorithmus im ECB-Mode anwenden auf einen zuvor verabredeten Wert (z.B. Counter, Zufallswert, etc.).

Betrachten wir die Alternativen im Einzelnen:

Angenommen wir verwenden einen Counter als IV.

Nehmen wir weiter an, wir verschlüsseln Plaintext P mit Counter 000...01 und Plaintext P’ mit Counter 000...10. Dann haben wir

$C = E_K(P \oplus 000...01)$

$C' = E_K(P' \oplus 000...10)$

Die verwendeten Counter unterscheiden sich also evtl. nur in einem Bit. Sollten sich die beiden Plaintexts zufällig auch nur in einem Bit (an derselben Position) unterscheiden (das ist nicht völlig unwahrscheinlich), so wären beide Ciphertexts identisch. Dies liefert überflüssige Information an potentielle Angreifer.

Wahl des Initialisierungsvektors (2)

Nehmen wir nun an, wir wählen unseren IV stattdessen als (Pseudo-)Zufallswert. Dann muss der Sender diesen Wert (geschützt) an den Empfänger übermitteln. Hierzu kann man diesen beispielsweise im ECB-Mode verschlüsselt übertragen.

Danach nutzen wir zur Verschlüsselung von P_1, P_2, P_3, ...:

$C_0 = IV$

$C_i = E_K(P_i \oplus C_{i-1})$ für i = 1, 2, 3, ...

Der Chiffretext ist also einen Block länger als der Klartext. Im Einzelfall kann dies zu Problemen führen.

Als dritte Möglichkeit wählen wir den IV als sogenannte Nonce (s.u.).

Nonce

In kryptographischen Protokollen und Algorithmen taucht häufig der Begriff ***Nonce*** auf. Dies ist ein Kunstwort und setzt sich zusammen aus den beiden Wörter *Number* und *once*. Eine Nonce ist also eine “number used (just) once”.

Für die Sicherheit des Verfahrens wird also ein Parameter benötigt, der sich bei nochmaliger Anwendung nicht wiederholt. Anderenfalls wird das Verfahren unsicher.

Oftmals wird der Begriff Nonce gleichbedeutend verwendet (oder einfach nur so interpretiert) mit einer Zufallszahl, beziehungsweise, in der Praxis, mit einer kryptographisch sicheren Pseudozufallszahl.

Der Begriff Nonce wurde jedoch gerade für Fälle geschaffen, in denen nicht entscheidend ist, ob ein Angreifer die Nonce raten kann bzw. kennt, oder nicht. Es ist ausreichend, wenn jeder Wert stets nur einmal verwendet wird.

In der Fachliteratur sollte man stets auf den Kontext achten, da manche Autoren “Nonce” schreiben, wenn sie “kryptographisch sichere (Pseudo)zufallszahl” meinen.

Wahl des Initialisierungsvektors (3)

Zuletzt betrachten wir den Fall, in dem unser Initialisierungsvektor als Nonce gewählt wird.

Wenn wir Nachrichten versenden, steht uns häufig eine zugehörige Message number zur Verfügung. Dies wäre ein möglicher Kandidat für eine Nonce. Wir haben weiter oben allerdings schon festgestellt, dass ein simpler Counter als IV nicht ideal ist. Wie könnten wir das Problem lösen? Indem beide Seiten den ihnen bekannten Wert (Counter, Message number etc.) zunächst (z.B. mittels ECB) verschlüsseln und das Ergebnis dann als Nonce verwenden.

Wichtig ist in diesem Zusammenhang, dass sich die Nonce während der Lebenszeit eines Cipher keys nirgends wiederholt, also auch nicht in der Kommunikation mit anderen, weiteren Partnern.

Beiden Seiten muss bekannt sein, wie und woraus die Nonce berechnet wird.

Ideen für weitere Betriebsmodi?

Frage in die Runde:

Wenn wir uns anschauen, wie man Blöcke miteinander verbinden kann, unter Einbeziehung verschiedener Kombinationen von Plaintext, Ciphertext, Verschlüsselungsfunktion, XOR, etc. – welche Möglichkeiten fallen Ihnen noch ein? Wieviele gibt es, wenn wir alles miteinander kombinieren, was uns einfällt?

(Ideensammlung an der Tafel und Diskussion)

Anmerkung: in der nächsten VL geht es weiter mit Betriebsmodi...