Advanced Encryption Standard AES

Krypto II

VL24

Datum: 07.01.2019

Buch des Tages

Titel: Foundations of Cryptography, Volume 1: Basic Tools

Autor(en): Oded Goldreich

Verlag: Cambridge

Umfang: ca. 370 Seiten

Hinweise zum Inhalt:

Sehr formale und mathematische Herangehensweise. Wer sich für theoretische Sicherheit und/oder Komplexitätsabschätzungen interessiert, findet hier einen Klassiker von einem renommierten Kryptographen. Zum Einstieg in das Thema ist dieses Buch allerdings weniger geeignet. Der Titel ist eher irreführend.

Buch des Tages

Titel: Foundations of Cryptography, Volume 2: Basic Applications

Autor(en): Oded Goldreich

Verlag: Cambridge

Umfang: ca. 790 Seiten

Hinweise zum Inhalt:

Sehr formale und mathematische Herangehensweise. Wer sich für theoretische Sicherheit und/oder Komplexitätsabschätzungen interessiert, findet hier einen Klassiker von einem renommierten Kryptographen. Zum Einstieg in das Thema ist dieses Buch allerdings weniger geeignet. Der Titel ist eher irreführend.

Buch des Tages

Titel: Tutorials on the Foundations of Cryptography

Autor(en): diverse Autoren; Editor: Yehuda Lindell

Verlag: Springer

Umfang: ca. 450 Seiten

Hinweise zum Inhalt:

Das Buch verweist im Titel auf das klassische Kryptographie-Buch von Oded Goldreich (Foundations of Cryptography). Insofern könnte man meinen, es handele sich um ein begleitendes Buch mit Übungen und Erläuterungen. Dies ist jedoch nicht der Fall. Der Bezug zu Goldreich's Buch ist gering. Behandelt werden anspruchsvolle Themen, die an Kapitel aus Goldreich's Lehrbuch anknüpfen. Die Darstellung ist sehr formal mathematisch und für Einsteiger eher weniger geeignet.

Ideensammlung: Mögliche Operationen auf 8-Bit Bytes

Frage in die Runde: was könnte man alles anstellen mit Bitstrings der Länge 8, im Rahmen eines symmetrischen Verschlüsselungsalgorithmus?

Anmerkung: die Länge 8 dient hier nur als ein Beispiel, da sie im AES eine wichtige Rolle spielt)

➔ Ideensammlung an der Tafel

Operationen auf 8-Bit Bytes

Operationen auf Bitstrings erfolgen im AES zumeist auf Teilblocks der Länge 8 Bit. Eines der Designziele war effiziente Ausführbarkeit in Hard- und Software, und dies wird unterstützt durch Verwendung von Funktionen, die auf Bitvektoren dieser Länge operieren. AES wurde seinerzeit unter anderem für die Verwendung in Smartcards entwickelt, wo diese Bitlänge besonders wichtig war. Mittlerweile hat sich die Hardware (und mit ihr die Registerlängen etc.) weiterentwickelt, doch für die Ausführung von AES, auch in gewöhnlichen PCs, werden oftmals spezielle Hardwareimplementierungen verwendet.

Wir haben zahlreiche Möglichkeiten zur Ausführung von Operationen auf 8-Bit Vektoren. Zum Beispiel:

- wir können bitweise verknüpfen, beispielsweise mittels XOR
- wir können rotieren um eine oder mehrere Positionen
- wir können addieren modulo 256
- wir können auf unterschiedliche Weise multiplizieren

Lookup Tables

Jede Funktion f: $\{0,1\}^8 \rightarrow \{0,1\}^8$, die auf 8-Bit Vektoren operiert, könnten wir ganz einfach darstellen mithilfe einer Tabelle mit allen 256 möglichen Inputvektoren und dem zugehörigen Funktionswert. Warum machen wir uns also überhaupt die Mühe und definieren aufwendige Funktionen, statt einer simplen Substitutionstabelle?

Solche Tabellen bzw. ***Lookup Tables*** werden tatsächlich verwendet, sind aber nicht in jedem Fall praktikabel.

Zum einen gibt es Operationen, deren Ausführung so effizient ist, dass eine Tabelle keinen Zusatznutzen darstellen würde, beispielsweise Rotation eines 8-Bit Registers.

Zum anderen sind viele Funktionen f(x) abhängig von weiteren Parametern, beispielsweise dem jeweiligen Verschlüsselungsschlüssel. Dann müssten wir für jeden Schlüssel eine eigene Tabelle hinterlegen, was völlig unmöglich ist. Eine Alternative sind schlüsselabhängige Vorberechnungen, doch auch diese sind selten praktikabel.

Wieviele verschiedene Funktionen f: $\{0,1\}^8 \rightarrow \{0,1\}^8$ lassen sich definieren?

- beliebiges f: $(2^8)^{256} = (2^8)^{2^8} = 2^{8x2^8} = 2^{2^{11}} = 2^{2048}$
- bijektives f: 256 x 255 x 254 x … x 2 x 1 = 256!

Beide Werte sind weit höher als die Anzahl der Atome im Weltall…

Arithmetik auf 8-Bit Bytes

Binäre Strings der Länge 8 beschreiben eine Menge mit 256 Elementen. Nützlich wäre es, wenn wir geeignete Operationen (Addition und Multiplikation) so definieren könnten, dass wir einen endlichen Körper auf dieser Menge erhalten:

- Betrachten wir allerdings bitweise Operationen und operieren auf $GF(2)^8$, so bleibt außer XOR nicht viel brauchbares übrig.
- Wenn wir stattdessen $\mathbb{Z}_{256}$ bzw. $\mathbb{Z}_{2^8}$ betrachten und modulo 256 rechnen, so ergibt dies keinen Körper: das Element 2 hat beispielsweise kein inverses Element b, so dass $2b \equiv 1 \bmod 256$ (denn 2b ist immer gerade).
- $\mathbb{Z}_n$ ist nur dann ein Körper, wenn n Primzahl. Die "naheste" Primzahl < 256 ist 251. Arithmetik modulo 251 wäre jedoch vergleichsweise ineffizient und $\mathbb{Z}_{251}$ daher keine brauchbare Alternative.
- Wir wissen indes, dass für jede Primzahl p und jedes $n \in \mathbb{N}$, $n > 1$, ein Körper $GF(p^n)$ existiert. Folglich auch für $GF(2^8)$. Dessen Elemente sind die Polynome
- $f(x) = \sum_{i=0}^{7} b_i x^i = b_7x^7 + b_6x^6 + \ldots + b_1x + b_0$, wobei $b_i \in \mathbb{Z}_2$.
- Auf diese Weise erhalten wir 256 Polynome, die wir einfach darstellen können mithilfe ihrer Koeffizienten $(b_7, b_6, \ldots, b_0)$, also als 8-Bit Vektor.

Addition und Multiplikation in GF(p^n)

- Die Addition zweier Polynome vom Grad < n ergibt wieder ein Polynom vom Grad < n. Hierzu werden die zugehörigen Koeffizienten aus $\mathbb{Z}_p$ zueinander addiert, was im Fall von GF(2^n) einer simplen XOR-Operation entspricht. Addition in GF(2^n) ist also einfach als bitweises XOR ausführbar.
- Angenommen, wir haben zwei Polynome f(x) und g(x) in GF(p^n). Seien hierbei

$$f(x) = \sum_{i=0}^{n-1} a_i x^i \text{ , } g(x) = \sum_{i=0}^{n-1} b_i x^i \text{ ,}$$

$$\text{dann gilt } f(x) + g(x) = \sum_{i=0}^{n-1} (a_i + b_i) x^i \text{, wobei } a_i + b_i \bmod p \in \mathbb{Z}_p.$$

- Das neutrale Element bezüglich Addition ist

  (0, 0, 0, 0, 0, 0, 0, 0)

- Das neutrale Element bezüglich Multiplikation ist

  (0, 0, 0, 0, 0, 0, 0, 1)

AES-Polynom und Polynomreduktion

- Die Multiplikation zweier Polynome vom Grad < n kann ein Polynom mit einem Grad $\geq$ n ergeben. Dieses müssen wir dann modulo reduzieren, um wieder ein Element aus unserer Menge der Polynome vom Grad < n zu erhalten.
- Die Reduktion erfolgt modulo eines irreduziblen Polynoms m(x) vom Grad n. Erinnerung: ein Polynom heißt ***irreduzibel*** genau dann, wenn es nicht darstellbar ist als Produkt zweier Polynome, deren Grad kleiner ist als jener von m(x).
- In GF(2^8) finden wir insgesamt 30 irreduzible Polynome. Für AES haben dessen Entwickler das folgende Polynom, auch als ***AES-Polynom*** bezeichnet, gewählt:

$$\mathbf{m(x) = x^8 + x^4 + x^3 + x + 1.}$$

Polynommultiplikation in GF(2^n)

- Betrachten wir nochmals die Multiplikation zweier Polynome f(x) und g(x) in GF(2^n), mit $f(x) = \sum_{i=0}^{n-1} a_i x^i$ und $g(x) = \sum_{i=0}^{n-1} b_i x^i$
- Nehmen wir an, das Produkt f(x)g(x) habe den Grad n+i für ein $i \geq 0$.
- Die erforderliche Reduktion des Produkts auf ein Polynom vom Grad < n können wir wie folgt vornehmen: sei r(x) ein irreduzibles Polynom vom Grad n:

$$r(x) = \sum_{i=0}^{n} c_i x^i = x^n + c_{n-1} x^{n-1} + \ldots + c_1 x + c_0$$

- Bilden wir das Polynom $r(x)x^i$, so hat dies, wie f(x)g(x), den Grad n+i.
- Addieren wir die beiden Polynome $r(x)x^i$ und f(x)g(x), so fällt der höchste Term weg (der höchste Koeffizient ist in beiden Fällen 1, und $1 \oplus 1 = 0$) und das Ergebnis ist ein Polynom mit niedrigerem Grad als f(x)g(x). Zudem sind f(x)g(x) und das Ergebnis $r(x)x^i + f(x)g(x)$ in derselben Restklasse modulo r(x).
- Wiederholen wir diesen Prozess, so bekommen wir in jedem Schritt ein Polynom niedrigeren Grades als zuvor, bis wir schließlich ein Polynom vom Grad < n erhalten.

Effiziente Multiplikation in GF(2^n) (1)

- Multiplikation in GF(p^n) ist aufwendiger als Addition (nicht anders als in der "gewöhnlichen" Arithmetik auch). Im Falle von GF(2^n) gibt es jedoch eine effiziente Lösung. Wir betrachten exemplarisch den Fall GF(2^8), doch das Vorgehen für allgemeines GF(2^n) verläuft analog.
- Sei $f(x) = \sum_{i=0}^{7} b_i x^i = b_7x^7 + b_6x^6 + \ldots + b_1x + b_0$, wobei $b_i \in \mathbb{Z}_2$.

  Multiplizieren wir $f(x)$ mit dem Polynom $g(x) = x$ (bzw. 02 in hexadezimal), so bekommen wir:

  $f(x)x = \sum_{i=0}^{7} b_i x^{i+1} = b_7x^8 + b_6x^7 + \ldots + b_0x$

  - Falls $b_7 = 0$, so hat das Ergebnis einen Grad < 8 und es ist nichts mehr tun.
  - Falls $b_7 = 1$, addieren wir einfach das zugehörige irreduzible Polynom vom Grad 8, in unserem Fall das zu AES gehörende Polynom $m(x) = x^8 + x^4 + x^3 + x + 1$.
- Wir erhalten: $(x^8 + b_6x^7 + \ldots + b_0x) + (x^8 + x^4 + x^3 + x + 1) =$

  $= (b_6x^7 + \ldots + b_0x) + (x^4 + x^3 + x + 1) = \underbrace{b_6 \ldots b_1 b_0 0 + 00011011}_{\text{Binärdarstellung}}$

Effiziente Multiplikation in GF(2^n) (2)

- Wie wir sehen, ist f(x)x + m(x) ein Polynom vom Grad < 8.
- Da f(x)x + m(x) in derselben Restklasse (modulo m(x) betrachtet) ist wie f(x)x, hat das Polynom f(x)x also einen Grad <8 und somit die gewünschte Form.
- Betrachten wir die binäre Darstellung der Polynome f(x) = $b_7b_6…b_0$ und g(x)= x = 00000010, so sehen wir, dass deren Multiplikation mittels folgender Fallunterscheidung durchführbar ist:
  a) falls $b_7 = 0$, so erhalten wir $f(x)x = b_6b_5…b_00$. Dies entspricht einem Shift von f(x) um eine Position nach links.
  b) falls $b_7 = 1$, so führen wir obigen Linksshift aus und addieren anschließend das AES-Polynom $x^4 + x^3 + x + 1$ = 00011011:

$$\begin{array}{l} b_6b_5b_4b_3b_2b_1b_00 \\ \oplus\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \end{array}$$

- Wir sehen, dass die Multiplikation mit g(x) = x ausführbar ist durch ein simples Shift und ein optionales XOR. Die Multiplikation mit beliebigen anderen Polynomen erfolgt durch wiederholte Anwendung obiger Prozedur und Zusammensetzung der Teilergebnisse.

Ideensammlung: Wie kreiert man eine Blockchiffre mit Hintertür?

Frage in die Runde: angenommen, Sie möchten eine neue Blockchiffre im Markt etablieren. Ihr Auftraggeber benötigt eine Hintertür, die es ihm ermöglicht, verschlüsselte Daten Dritter wieder zu entschlüsseln. Was könnte man tun, um dieses Ziel zu erreichen?

➔ Ideensammlung an der Tafel

Ideensammlung: Wie kreiert man eine vertrauenswürdige (Block-)Chiffre

Frage in die Runde: angenommen, Sie möchten eine neue Blockchiffre als zukünftige Verschlüsselungsmethode etablieren. Welche organisatorischen, technischen etc. Maßnahmen halten Sie für notwendig, um potentielle Anwender davon zu überzeugen, dass Ihr Verfahren vertrauenswürdig und gut ist?

➔ Ideensammlung an der Tafel

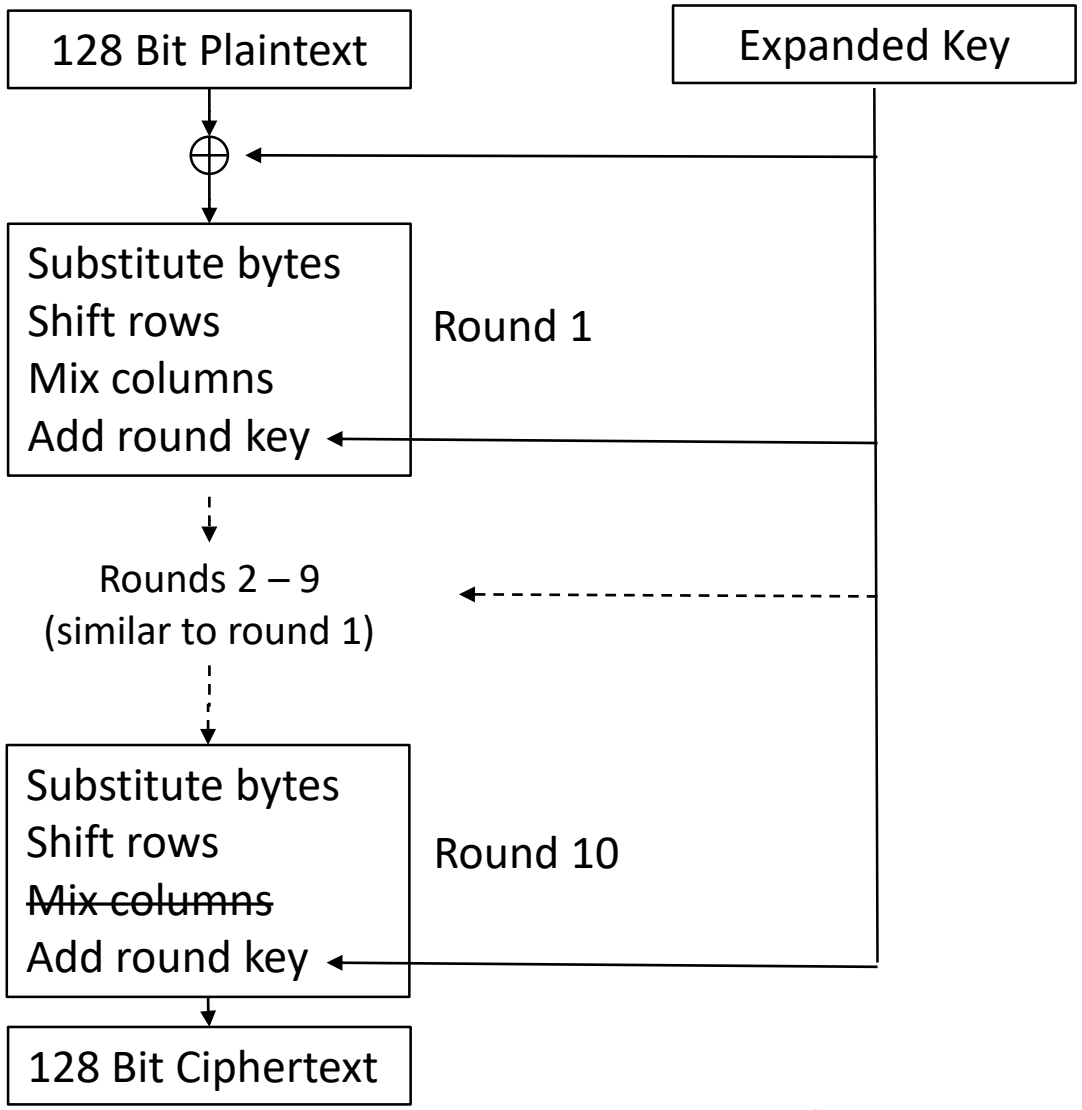
Fakten zu AES

- Der ***Advanced Encryption Standard AES*** wurde 2001 vom U.S. NIST als Verschlüsselungsstandard etabliert. Die Auswahl erfolgte im Rahmen eines öffentlichen Wettbewerbs. Gewinner waren zwei belgische Kryptographen, Vincent Rijmen und Joan Daemen, und deren Algorithmus namens Rijndael.
- Rijndael erlaubte ursprünglich zahlreiche Kombinationen unterschiedlicher Schlüssellängen und Blocklängen. Im Standard wurden diese Optionen eingeschränkt auf eine feste Blocklänge von 128 Bit und drei mögliche Schlüssellängen: 128 Bit, 192 Bit, 256 Bit.
- Die Spezifikation von AES findet sich in U.S. FIPS PUB 197. Ferner ist AES genannt (neben mehreren anderen, weniger bekannten Verfahren) in ISO/IEC 18033-3 Information technology - Security techniques - Encryption algorithms - Part 3: Block ciphers
- AES ist eine symmetrische Blockchiffre und der einzige öffentlich bekannte Algorithmus, der von der NSA für Verschlusssachen bis einschließlich top secret zugelassen wurde.
- AES ist in Industrie und Behörden weltweit die am häufigsten verwendete symmetrische Verschlüsselung.

Eigenschaften von AES

- AES ist ein sogenanntes Substitution-Permutation Network.
- AES ist sowohl in Hardware als auch Software sehr effizient implementierbar.
- Die Anzahl der Runden ist abhängig von der gewählten Schlüssellänge:
  - 128 Bit Schlüssellänge: 10 Runden
  - 192 Bit Schlüssellänge: 12 Runden
  - 256 Bit Schlüssellänge: 14 Runden
- Jede Runde besteht aus mehreren Teilschritten. Jeder dieser Teilschritte hat eine andere Funktionsweise und Zielsetzung.
- Der Verschlüsselungsschlüssel wird auf eine größere Länge expandiert und in jeder Runde werden 128 Bit des expandierten Schlüssels auf das Zwischenergebnis per XOR aufaddiert.

High-Level Darstellung des AES

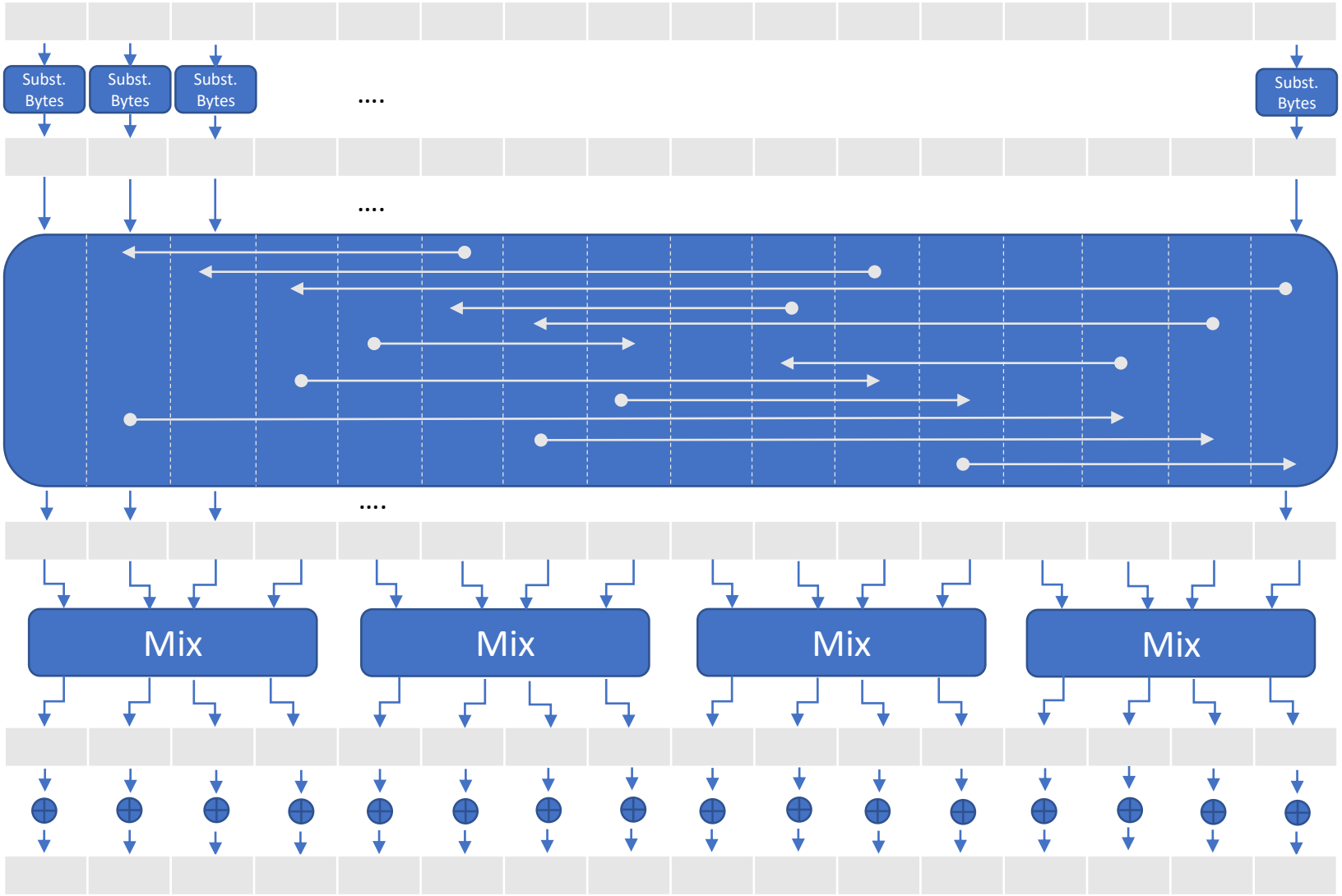


- vor Beginn der ersten Runde werden 128 Bit Expanded key addiert mittels XOR
- Runde 1 bis 9 besteht aus je 4 Teilfunktionen
- zum Schluss jeder Runde bzw. als vierte Teilfunktion werden 128 Bit Expanded key addiert mittels XOR
- in der letzten Runde wird die Teilfunktion Mix columns ausgelassen

Rundenstruktur des AES

- Eine AES-Runde besteht aus vier Teilkomponenten:
  - ***Substitute bytes***: ersetzt jedes Byte mittels Table lookup
  - ***Shift rows***: verschiebt Bytes innerhalb des Inputvektors
  - ***Mix columns***: wendet auf je 4 Bytes eine bijektive Funktion an
  - ***Add round key***: verknüpt einen 128-Bit Input per XOR mit dem Rundenschlüssel
- Frage in die Runde: Welche dieser Teilkomponenten sind abhängig vom Verschlüsselungsschlüssel?
- Nur die Funktion Add round key ist abhängig vom Schlüssel. Alle anderen Komponenten arbeiten schlüsselunabhängig.
- Daher beginnt und endet der Algorithmus mit einem Add Round Key, da andere (schlüssel-unabhängige) Operationen zu Beginn oder Ende einfach eliminiert werden könnten.
- Die Verschlüsselung beginnt mit einem Add round key, erst danach beginnt die erste Runde.
- Die letzte Runde besteht aus nur drei Teilkomponenten und verzichtet auf Mix columns. Letzte ausgeführte Teilkomponente ist wiederum Add round key.
- Bei n Runden benötigen wir folglich n+1 Rundenschlüssel, so dass beispielsweise bei AES mit 128-Bit Schlüssellänge (und 10 Runden) ein Expanded Key mit 11 x 128 = 1408 Bit erzeugt werden muss.

High-Level Darstellung einer Runde



Entschlüsselung vs. Verschlüsselung

- In manchen Algorithmen, beispielsweise DES oder One-Time Pad, erfolgt die Entschlüsselung identisch zur Verschlüsselung. Für eine Implementierung ist dies sehr vorteilhaft.
- Schauen wir den Aufbau des AES an so sehen wir, dass wir hier zur Entschlüsselung nicht einfach den Verschlüsselungsalgorithmus nutzen können. Wir können auch nicht einfach den Algorithmus spiegelbildlich “von unten nach oben” ausführen.
- Betrachten wir eine einzelne Runde, so müssen wir die dort enthaltenen vier Teilfunktionen invertieren (für drei der vier Rundenoperationen ist die Invertierbarkeit offensichtlich, und für Mix columns werden wir diese noch zeigen) und in der Reihenfolge umkehren. Die Umkehrung lässt sich in der Praxis jedoch durch ein paar Kniffe vereinfachen.
- Hinzu kommt, dass die erste und letzte Verschlüsselungsrunde vom sonstigen Rundenschema abweichen: vor der ersten Runde erfolgt ein Add round key, und die letzte Verschlüsselungsrunde wurde um eine Funktion gekürzt. Bei der Umkehrung müssen wir all dies berücksichtigen.
- Wir werden sehen, dass wir mit einigen Anpassungen tatsächlich die für die Verschlüsselung verwendeten Rundenoperationen auch bei der Entschlüsselung verwenden können.

Entschlüsselung und Verschlüsselung