14.1 XOR encryption step by step

Peter Rutschmann

12.11.2025

XOR-Verschlüsselung Schritt für Schritt

In diesem Jupyter Notebook wird die **XOR-Verschlüsselung** erklärt und angewendet:

- 1. XOR-Idee verstehen
- 2. **XOR von Hand** an einem einfachen Beispiel nachvollziehen
- 3. **XOR mit Python** programmieren
- 4. Klartext als **HEX** und **BIN** darstellen
- 5. Schlüssel als **BIN** zeigen
- 6. XOR anwenden und Ergebnis als **HEX** ausgeben

Was ist XOR?

Sie kennen die Addition von zwei Zahlen 1+2=3

XOR ("exclusive or") ist eine **logische Verknüpfung** auf Bit-Ebene. XOR wird nicht auf zwei dezimale Zahlen sonderen auf zwei Bits (zBps. Bit-A und Bit-B) angewandt Beide können 0 oder 1 sein.

Regeln für XOR:

Bit-A	Bit-B	Ergebnis A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Man kann XOR auch auf zwei Bitfolgen anwenden, wobei immer die beiden Bits an der gleichen Position verknüpft werden.

```
Beispiel:

    0100'0100  (Bitfolge A -> Buchstabe 'H')
XOR 0100'1011  (Bitfolge B -> Schlüssel Buchstabe 'K' )
--------
    0000'1111  (Ergebnis, das muss nicht einem Buchstaben
 ↪  entsprechen!)
```

XOR-Verschlüsselung

- **Klartext:** *INFORMATIK*
- **Schlüssel:** *BYTE*
- Kodierung UTF8 (ein Byte pro Buchstabe)

1. Klartext *ÌNFORMATIK* in Binär umwandeln

```
   I        N        F        O         R       M
 ↪  A        T        I        K
01001001 01001110 01000110 01001111 01010010 01001101
 ↪  01000001 01010100 01001001 01001011
```

2. Schlüssel *BYTE* in Binär umwandeln

```
   B        Y        T        E
01000010 01011001 01010100 01000101
```

3. Klartext und Schlüssel aufreihen, Schlüssel wird über die Länge des Klartexts wiederholt:

```
   I        N        F        O          R        M
 ↪  A        T        I        K
01001001 01001110 01000110 01001111 01010010 01001101
 ↪  01000001 01010100 01001001 01001011
   B        Y        T        E          B        Y
 ↪  T        E        B        Y
01000010 01011001 01010100 01000101 01000010 01011001
 ↪  01010100 01000101 01000010 01011001
```

4. Bei den beiden obigen Bit-Muster XOR bitweise anwenden

```
01001001 01001110 01000110 01001111 01010010 01001101
 ↪  01000001 01010100 01001001 01001011
01000010 01011001 01010100 01000101 01000010 01011001
 ↪  01010100 01000101 01000010 01011001
⌋
 ↪  ------------------------------------------------------------------
00001011 00010111 00010010 00001010 00010000 00010100
 ↪  00010101 00010001 00001011 00010010
```

```
   0B       17       12       0A       10       14
 ↪  15       11       0B       12
```

ACHTUNG: Die Bytes des Ergebnisses entsprechen nicht unbedingt druckbaren Zeichen! Man notiert das Ergebnis bin oder kompakter in HEX: ***0B 17 12 0A 10 14 15 11 0B 12***

XOR-Entschlüsselung

Prinzip der Entschlüsselung: - Man wendet auf den *verschlüsselten Text* den *gleichen Schlüssel* an. - Wenn man ein Bit des *verschlüsselten Texts* mit einem *Bit des Schlüssels* per *XOR verknüpft*, erhält man das Bit des Klartexts zurück. - Also: *(Klartext XOR Schlüssel) XOR Schlüssel = Klartext*

- **Verschlüsselt:** *0B 17 12 0A 10 14 15 11 0B 12*
- **Schlüssel:** *BYTE*
- Kodierung UTF8 (ein Byte pro Buchstabe)

1. Verschlüsselten Text in Binär umwandeln

```
   0B       17       12       0A       10       14
 ↪  15       11       0B       12
00001011 00010111 00010010 00001010 00010000 00010100
 ↪  00010101 00010001 00001011 00010010
```

2. Schlüssel in Binär umwandeln

```
    B        Y        T        E
 01000010 01011001 01010100 01000101
```

3. Verschlüsselten Text und Schlüssel aufreihen. Schlüssel wird über die Länge des verschlüsselten Texts wiederholt:

```
   0B       17       12       0A       10       14
 ↪  15       11       0B       12
00001011 00010111 00010010 00001010 00010000 00010100
 ↪  00010101 00010001 00001011 00010010
   B        Y        T        E        B        Y
 ↪  T        E        B        Y
01000010 01011001 01010100 01000101 01000010 01011001
 ↪  01010100 01000101 01000010 01011001
```

4. Bei den beiden obigen Bit-Muster bitweise XOR anwenden

```
00001011 00010111 00010010 00001010 00010000 00010100
 ↪  00010101 00010001 00001011 00010010
01000010 01011001 01010100 01000101 01000010 01011001
 ↪  01010100 01000101 01000010 01011001
⌋
 ↪  ----------------------------------------------------------------
```

```
01001001 01001110 01000110 01001111 01010010 01001101
 ↪  01000001 01010100 01001001 01001011
```

5. Binär in Klartext umwandeln

```
01001001 01001110 01000110 01001111 01010010 01001101
 ↪  01000001 01010100 01001001 01001011
  I        N        F        O        R        M
 ↪  A        T        I        K
```

Aufgabe: Entschlüsseln Sie die Nachricht von Hand

- Verschlüsselt: *01 1B 04 0A 0D 0C*
- Schlüssel: *EI*
- Wie lautet der Klartext?

Lösung.. nicht spicken :-)

```
    01       1B       04       0A       0D       0C
00000001 00011011 00000100 00001010 00001101 00001100
    E        I        E        I        E        I
01000101 01001001 01000101 01001001 01000101 01001001
--------------------------------------------------------
01000100 01010010 01000001 01000011 01001000 01000101
   D        R        A        C        H        E
```

Aufgabe: Verschlüsselte Nachricht austauschen

- Verschlüsseln Sie ein Wort mit 6 Buchstaben und einem Schlüssel mit 3 Buchstaben.
- Notieren Sie das Ergebnis in HEX.
- Geben Sie Ihrem Lernpartner den verschlüsselten Text und den Schlüssel weiter.
- Kann er den Klartext wiederherstellen?

XOR-Verschlüsselung mit Python

- Vorgegeben ist die **Hilfsfunktion** *xor_bytes*, die zwei Byte-Arrays XOR-verknüpft.
- Weiter zeigt das Beispiel, wie man eine Text zerlegt und in Hex- und Binärdarstellungen ausgibt.

```
# Starten Sie den Block einmal und schauen Sie die Ausgaben an

# XOR-Funktion fuer zwei bytes-arrays definieren
def xor_bytes(data: bytes, key: bytes) -> bytes:
    """XORt eine Daten-Bytefolge mit einem Schlüssel (der ggf. wiederholt wird)."""
    if not key:
        raise ValueError('Schlüssel darf nicht leer sein')
    key_len = len(key)
    return bytes(b ^ key[i % key_len] for i, b in enumerate(data))

text = "HALLO"
print("text " + text)

# text als einzelne Buchstaben ausgeben
print("text als Buchstaben:", " ".join(text))

# text in einzelne Bytes zerlegen
textInBytes = text.encode("utf-8")

# textInBytes in HEX ausgeben
print("textInBytes in HEX:", " ".join(f"{x:02X}" for x in textInBytes))

# textInBytes in BIN ausgeben
print("textInBytes in BIN:", " ".join(f"{x:08b}" for x in textInBytes))

# textInBytes wieder zu Buchstaben zusammensetzen
print("textInBytes wieder zu Text:", textInBytes.decode("utf-8"))

print()
#schluessel als Buchstaben oder direkt als Bytes definieren
schluessel = "ADE"
schluesselInBytes = bytes([0x41, 0x44, 0x45])
print("schluessel als Buchstaben: " + schluessel)

```

```
# schluesselInBytes in HEX ausgeben
print("schluessel in HEX:", " ".join(f"{x:02X}" for x in
 ↪  schluesselInBytes))

# schluesselInBytes in BIN ausgeben
print("schluessel in BIN:", " ".join(f"{x:08b}" for x in
 ↪  schluesselInBytes))

print()
# XOR-Operation durchfuehren
xorResultAlsBytes = xor_bytes(textInBytes, schluesselInBytes)

# xorResultAlsBytes in BIN ausgeben
print("xorResultAlsBytes in BIN:", " ".join(f"{x:08b}" for x
 ↪  in xorResultAlsBytes))

# xorResultAlsBytes in HEX ausgeben
print("xorResultAlsBytes in HEX:", " ".join(f"{x:02X}" for x
 ↪  in xorResultAlsBytes))
```

```
text HALLO
text als Buchstaben: H A L L O
textInBytes in HEX: 48 41 4C 4C 4F
textInBytes in BIN: 01001000 01000001 01001100 01001100 01001111
textInBytes wieder zu Text: HALLO

schluessel als Buchstaben: ADE
schluessel in HEX: 41 44 45
schluessel in BIN: 01000001 01000100 01000101

xorResultAlsBytes in BIN: 00001001 00000101 00001001 00001101 00001011
xorResultAlsBytes in HEX: 09 05 09 0D 0B
```

Aufgabe: XOR-Verschlüsselung mit Python selber anwenden

Sie fangen mit einem einfachen Beispiel an: Klartext: ‘ERAGON’ Schlüssel: ‘SIR’ Wie lautet der verschlüsselte Text in HEX?

Vorgehen für Umsetzung in Python:

- Klartext als Variable definieren
- Klartext als einzelne Buchstaben ausgeben

- Klartext in Bytes umwandeln und ausgeben
- Klarext_In_Bytes in HEX umwandeln und ausgeben
- Klarext_In_Bytes in Binär umwandeln und ausgeben
- Schlüssel als Variable definieren
- Schlüssel als einzelne Buchstaben ausgeben
- Schlüssel in Bytes umwandeln und ausgeben
- Schlüssel_In_Bytes in HEX umwandeln und ausgeben
- Schlüssel_In_Bytes in Binär umwandeln und ausgeben
- Klartext und Schlüssel per XOR verknüpfen
- XOR-Ergebnis in Binär ausgeben
- XOR-Ergebnis in HEX ausgeben

```
# Programmieren Sie das Beschriebene in Python.
print("Meine Verschüsselung")
```

```
Meine Verschüsselung
```

Lösung.. nicht spicken :-)

```
text = "ERAGON"
print("text " + text)
print("text als Buchstaben:", " ".join(text))
textInBytes = text.encode("utf-8")
print("textInBytes in HEX:", " ".join(f"{x:02X}" for x in
 ↪  textInBytes))
print("textInBytes in BIN:", " ".join(f"{x:08b}" for x in
 ↪  textInBytes))

print()
schluessel = "SIR"
print("schluessel " + schluessel)
print("schluessel als Buchstaben:", " ".join(schluessel))
schluesselInBytes = schluessel.encode("utf-8")
print("schluesselInBytes in HEX:", " ".join(f"{x:02X}" for x
 ↪  in schluesselInBytes))
print("schluesselInBytes in BIN:", " ".join(f"{x:08b}" for x
 ↪  in schluesselInBytes))

print()
```

```
# XOR-Operation durchfuehren
xorResultAlsBytes = xor_bytes(textInBytes, schluesselInBytes)
print("xorResultAlsBytes in BIN:", " ".join(f"{x:08b}" for x
 ↪  in xorResultAlsBytes))
print("xorResultAlsBytes in HEX:", " ".join(f"{x:02X}" for x
 ↪  in xorResultAlsBytes))
```

Aufgabe: XOR-Entschlüsselung mit Python selber anwenden

Schaffen Sie es den Ablauf für die Entsschlüsselung zu definieren und umzusetzen? **Geben Sie alle Zwischenergebnisse, HEX, BIN, Text … aus.**

```
# Programmieren Sie die zur vorangehenden Aufgabe passend XOR
 ↪  Entschlüsselung in Python.
print("Meine Entschlüsselung")

encryptionInBytes = bytes([0x16, 0x1B, 0x13, 0x14, 0x06,
 ↪  0x1C])
```

```
Meine Entschlüsselung
```

Lösung.. nicht spicken :-)

```
encryptionInBytes = bytes([0x16, 0x1B, 0x13, 0x14, 0x06,
 ↪  0x1C])

print("encryptionInBytes in BIN:", " ".join(f"{x:08b}" for x
 ↪  in encryptionInBytes))

schluessel = "SIR"
schluesselInBytes = schluessel.encode("utf-8")
print("schluesselInBytes in BIN:", " ".join(f"{x:08b}" for x
 ↪  in schluesselInBytes))

decryptionInBytes = xor_bytes(encryptionInBytes,
 ↪  schluesselInBytes)
print("decryptionInBytes in BIN:", " ".join(f"{x:08b}" for x
 ↪  in decryptionInBytes))
print(decryptionInBytes.decode("utf-8"))
```

Aufgabe: Eigenes Beispiel mit einem Schlüsselwort

Implementieren Sie eine eigenes Beispiel mit einem KEY aus mehreren Buchstaben. **Geben Sie alle Zwischenergebnisse, HEX, BIN, Text … aus.**

```
## XOR-Verschlüsselung mit Python
print("Eigenes Beispiel")
```

```
Eigenes Beispiel
```

Aufgabe: Gegenseitig Verschlüsseln und Entschlüsseln

Tauschen Sie mit einem Partner einen Schlüssel aus. Verschlüsseln Sie einen Text und geben Sie Ihrem Partner das verschlüsselte Ergebniss als HEX-Bytes encryptionInBytes = bytes([0x16, 0x1B, …. Der Partner soll den Text mit dem Schlüssel wieder entschlüsseln. Geben Sie alle Zwischenergebnisse, HEX, BIN, Text … aus.

```
# Partenerarbeit
print("Partenerarbeit")
```

```
Partenerarbeit
```

Aufgabe: Unvollständiger Schlüssel

Idee:

- Bob und Anne haben eine mit XOR verschlüsselte Nachricht ausgetauscht.
- Eve hat die ganze verschlüsselte Nachricht abgefangen.
- Zudem hat Eve auf verbotenem Weg vom Schlüssel der gesammt Länge 5, die ersten 4 Bytes stehlen können.

Schaffen Sie es Eve zu unterstützen und die ganze Nachricht zu entschlüsseln?

- Verschlüsselte Nachricht (HEX): 1f040215000d0b16041c1d03030c1b1c1d11141e09001d08061c1a1f021a0103
- Bekannter Teil des Schlüssels (UTF8), 4 von 5 Buchstaben, der letzte Buchstabe fehlt: **hmpa**

```
# hack the code
encryptionInBytes = bytes.fromhex(
 ↪  "1f040215000d0b16041c1d03030c1b1c1d11141e09001d08061c1a1f021a01031404
 ↪  )
print("encryptionInBytes in HEX:", " ".join(f"{x:02X}" for x
 ↪  in encryptionInBytes))
```

```
encryptionInBytes in HEX: 1F 04 02 15 00 0D 0B 16 04 1C 1D 03 03 0C 1B 1C
```