Crypto Workshop

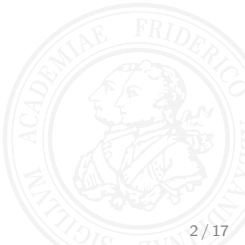
Symmetrische Verschlüsselung

Christoph Egger
Dominik Paulus

12. Mai 2018

Verschlüsseln

- Transformation der Eingabe
- Ohne Kenntnis des Schlüssels erfährt man “nichts” über die Eingabe
- Mit Schlüssel lässt sich die Eingabe wiederherstellen



IND-CPA

IND-CPA Game

$\mathsf{sk} \leftarrow_{\$} \{0,1\}^{\lambda}$

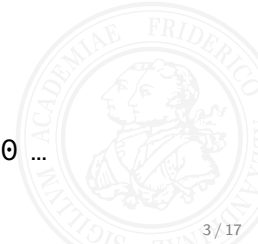
$m_1, m_2 \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot)}$

$b \leftarrow_{\$} \{0,1\}$

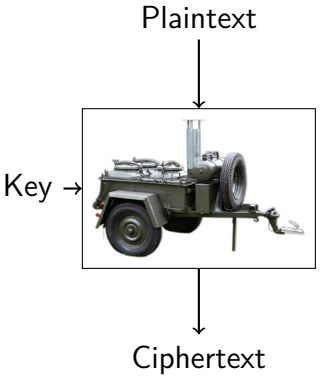
$b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{sk},\cdot)}(\mathsf{Enc}(\mathsf{sk}, m_b))$

$\mathit{return}(b = b')$

- "Schwache" Definition von Verschüsselung
- Wird regelmäßig verwendet: `AES-CBC`, `AES-CTR`, `ChaCha20` ...

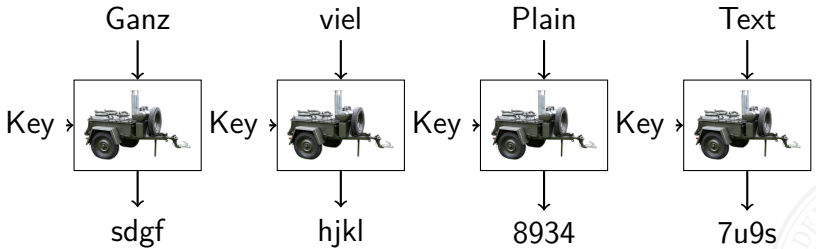


Blockchiffre

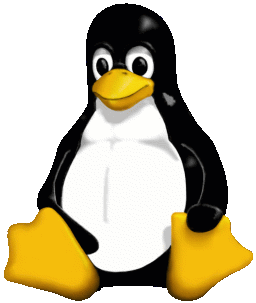


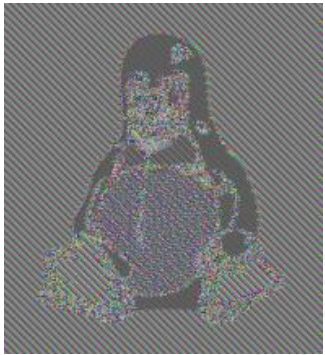
ECB Mode

"Electronic Code Book"



ECB Pinguin





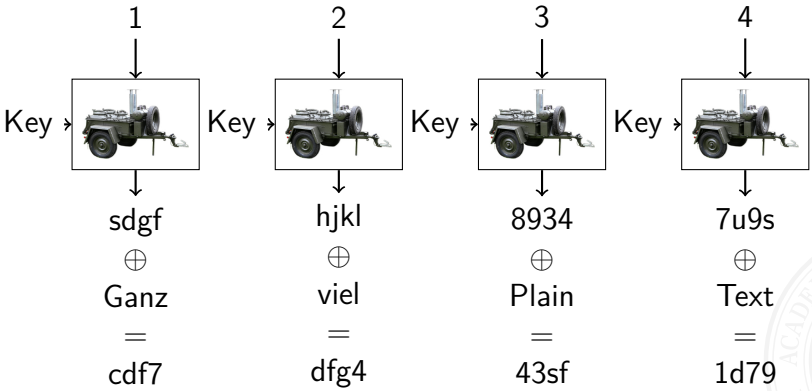
Mix & Match

"Ciphertexte zusammenbasteln"

- Ciphertext in Blöcke zerlegen (16 Zeichen/128 Bit für AES)
- Beliebig wieder zusammenfügen
  - SQL Injection trotz verschlüsselter Anfrage
  - Verbotene Strings umgehen
  - ...

CTR Mode

"Counter mode" oder auch "One-Time-Pads waren doch gar nicht so schlecht!"



CTR vs OTP

- Sehen ähnlich aus?

CTR vs OTP

- Sehen ähnlich aus?
- Zufällige folge von bytes als “key”
- $\oplus$ mit dem Plaintext um die Verschlüsselung zu generieren

CTR vs OTP

- Sehen ähnlich aus?
- Zufällige folge von bytes als “key”
- $\oplus$ mit dem Plaintext um die Verschlüsselung zu generieren
- $\Rightarrow$ “key” nicht Wiederverwerten!

CTR vs OTP

- Sehen ähnlich aus?
- Zufällige folge von bytes als “key”
- $\oplus$ mit dem Plaintext um die Verschlüsselung zu generieren

- $\Rightarrow$ “key” nicht Wiederverwerten!
  - Für CTR reicht es einen anderen Zähler zu verwenden
  - … und man kann diesen sogar veröffentlichen

CTR vs OTP

- Sehen ähnlich aus?
- Zufällige folge von bytes als “key”
- $\oplus$ mit dem Plaintext um die Verschlüsselung zu generieren

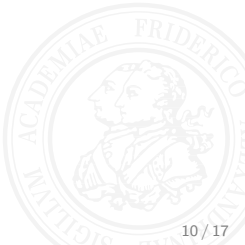
- $\Rightarrow$ “key” nicht Wiederverwerten!
  - Für CTR reicht es einen anderen Zähler zu verwenden
  - … und man kann diesen sogar veröffentlichen

- Stromchiffren setzen das Konzept “direkter” um

Malleability

Vermutung

$\Rightarrow$ Wenn der Angreifer den Inhalt nicht lesen kann, dann kann er ihn auch nicht (kontrolliert) verändern?



Demo

```
c = AES.new(b'testtesttesttest',
            mode=AES.MODE_CTR,
            counter=Counter.new(128))
       .encrypt(b"Administratob")
c[-1] = c[-1] ^ (1<<4)
AES.new(b'testtesttesttest',
        mode=AES.MODE_CTR,
        counter=Counter.new(128))
   .decrypt(c)

b'Administrator'
```

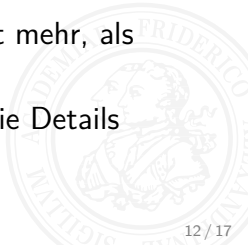
Malleability

Vermutung

$\Rightarrow$ Wenn der Angreifer den Inhalt nicht lesen kann, dann kann er ihn auch nicht (kontrolliert) verändern?

Ergebnis

- Stimmt nicht!
- Ist auch nicht spezifisch für den Counter-Mode, sondern fordert mehr, als unsere Sicherheitsdefinition bietet.
- Ähnliches lässt sich auch mit anderen Cipher-Modi machen – die Details sind verschieden aber das Problem bleibt.

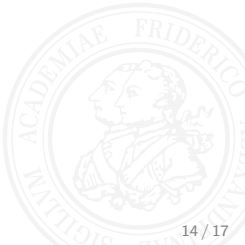


Fragen?

42

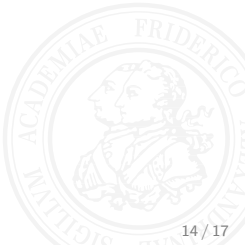
ECB Mode

- Strukturpattern finden
  - Gedichte werden zyklisch wiederholt
  - Fixes Padding $\rightarrow$ man kann kurze Zeilen erkennen!
  - Man kann die leeren Zeilen zwischen Absätzen erkennen



ECB Mode

- Strukturpattern finden
  - Gedichte werden zyklisch wiederholt
  - Fixes Padding $\rightarrow$ man kann kurze Zeilen erkennen!
  - Man kann die leeren Zeilen zwischen Absätzen erkennen
- Struktur verwenden um Gedicht zu identifizieren



Counter Mode

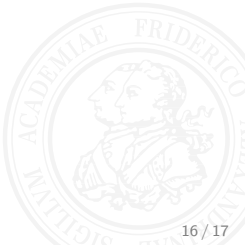
```
import nclib

nc = nclib.Netcat(('localhost', 4212))
print(nc.recv())
nc.sendall(b'R')
print(nc.recv())
nc.sendall(b'Administratob')
print(nc.recv())

# Flip the 5 to last bit and use that
# as your login token
```

Abschliessend

- Verschüsselung schützt die Integrität nicht!
- Verschüsselte Logintoken == Nutzlos



Fragen?

42