Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

16. November 2018

GI DevCamp 2018 Hamburg

Häufige Fehler bei der Verwendung von Kryptografie und wie man diese vermeidet

Maximilian Blochberger

Tom Petersen

Arbeitsgruppe SVS, Fachbereich Informatik, Universität Hamburg

Zu uns

Yay, Datenschutz!

Maximilian Blochberger

Tom Petersen

Wissenschaftliche Mitarbeiter im Arbeitsbereich *Sicherheit in verteilten Systemen* an der Universität Hamburg.

Forschungsinteressen:

- Usable Security
- Mobile Security

Forschungsinteressen:

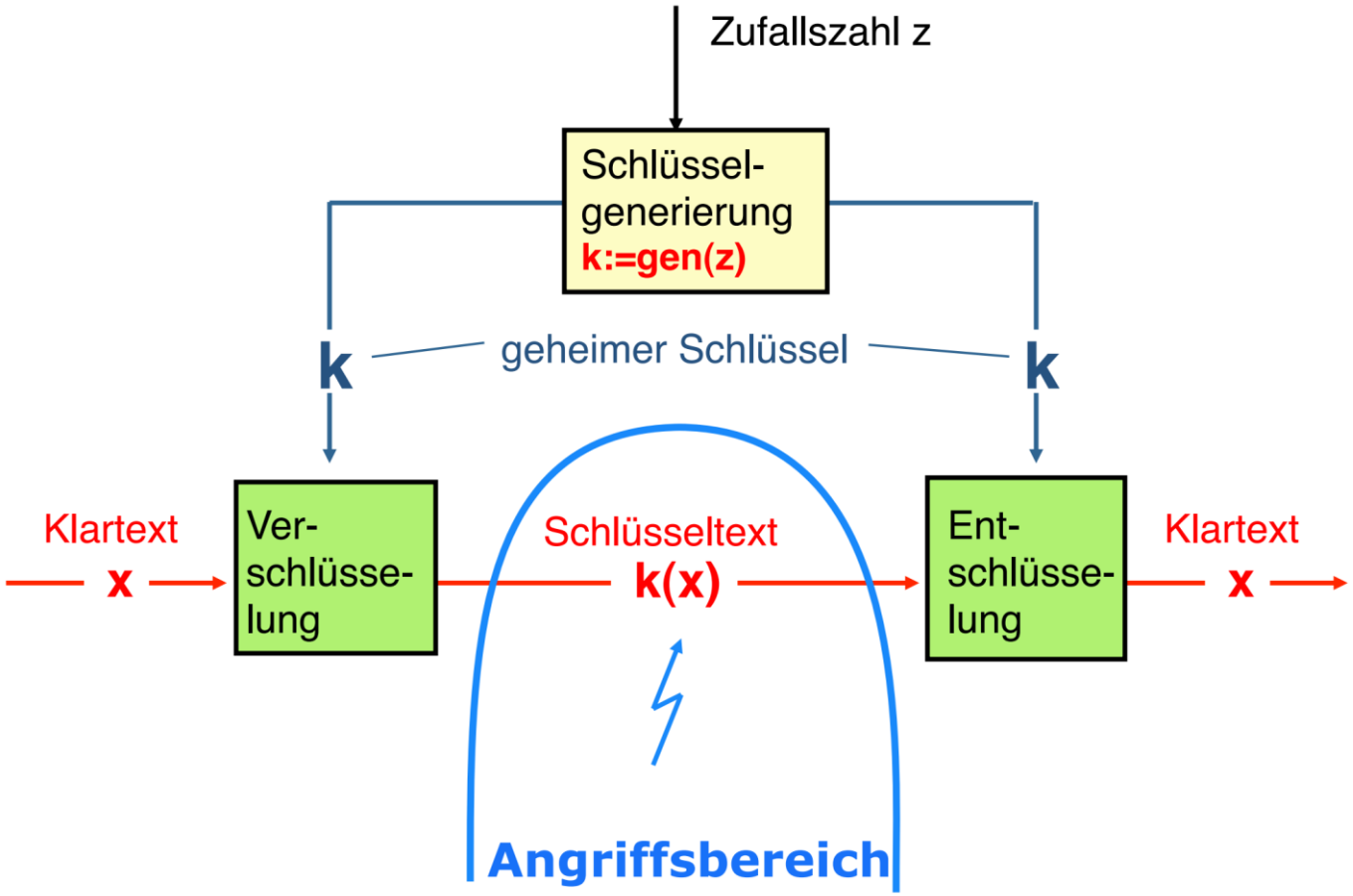
- Angewandte Kryptographie
- Datenschutz und PETs

Vorweg

- Vorerfahrungen? Programmieren – Kryptographie?
- MacBook dabei?
- Xcode installiert?

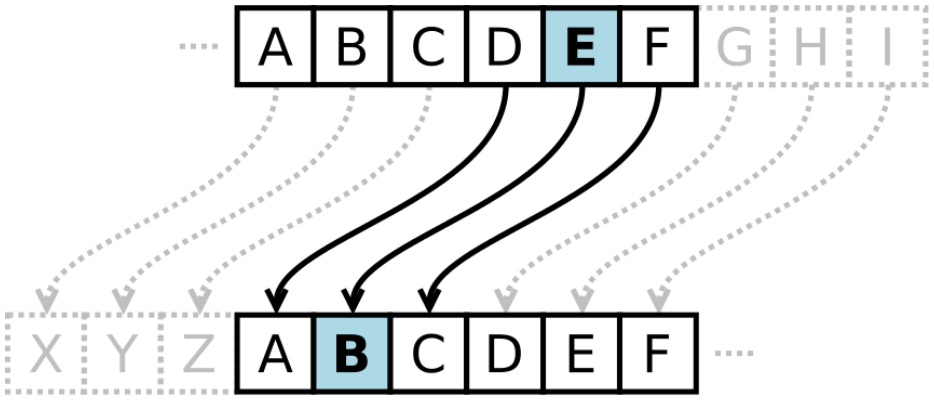
- Schutzziele
  - **Vertraulichkeit**
    Schutz vor unberechtigtem Zugriff auf Daten
  - **Integrität**
    Schutz vor unerkannter Veränderung von Daten
  - **Verfügbarkeit**
    Sicherstellung der Nutzbarkeit von Ressourcen
- Kryptographie
  - **Verschlüsselung**
    - **symmetrisch**
    - asymmetrisch
  - Signaturen
  - …

Symmetrische Verschlüsselung



Quelle: Hannes Federrath: Vorlesung *Grundlagen der Systemsoftware.*

Beispiel für eine symmetrische Verschlüsselung: Cäsar-Chiffre



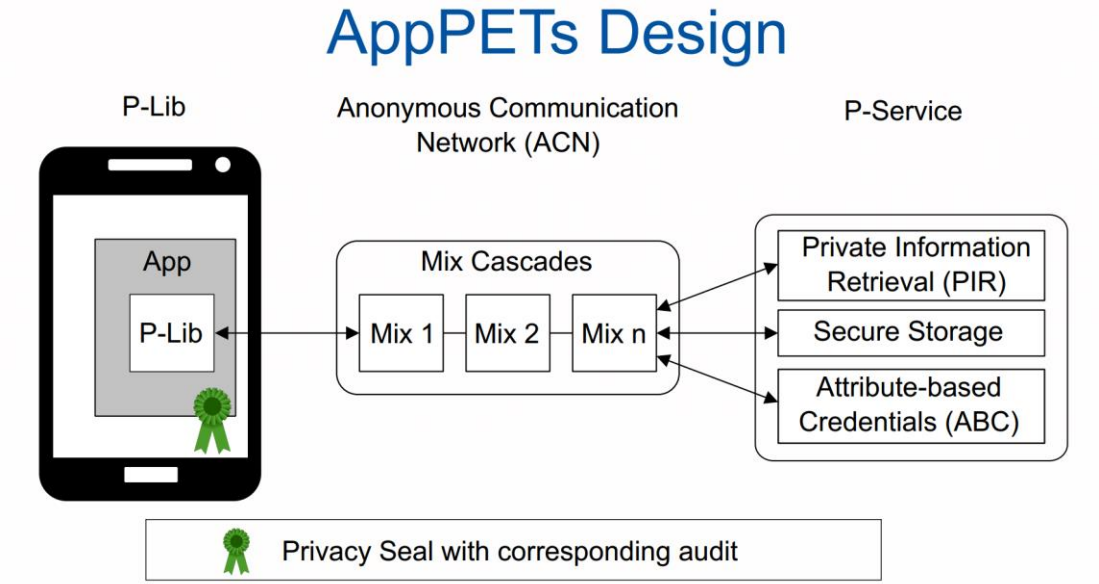

```
Plain:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:   XYZABCDEFGHIJKLMNOPQRSTUVW

Plain:    GI DEVCAMP
Cipher:   DF ABYZXJM
```


Quelle: Wikipedia, https://en.wikipedia.org/wiki/Caesar_cipher

Das AppPETS-Projekt

AppPETs
Datenschutzfreundliche Smartphone Anwendungen ohne Kompromisse

- Integration von datenschutzfreundlichen Technologien in Smartphone-Apps



- Kryptographische Bibliothek: **Tafelsalz**
- Für weitere Infos: http://app-pets.org

Quelle: Erik Sy: AppPETs. Poster, ACM SAC 2018.

Workshop-Aufgabe

https://github.com/AppPETs/DCrypt

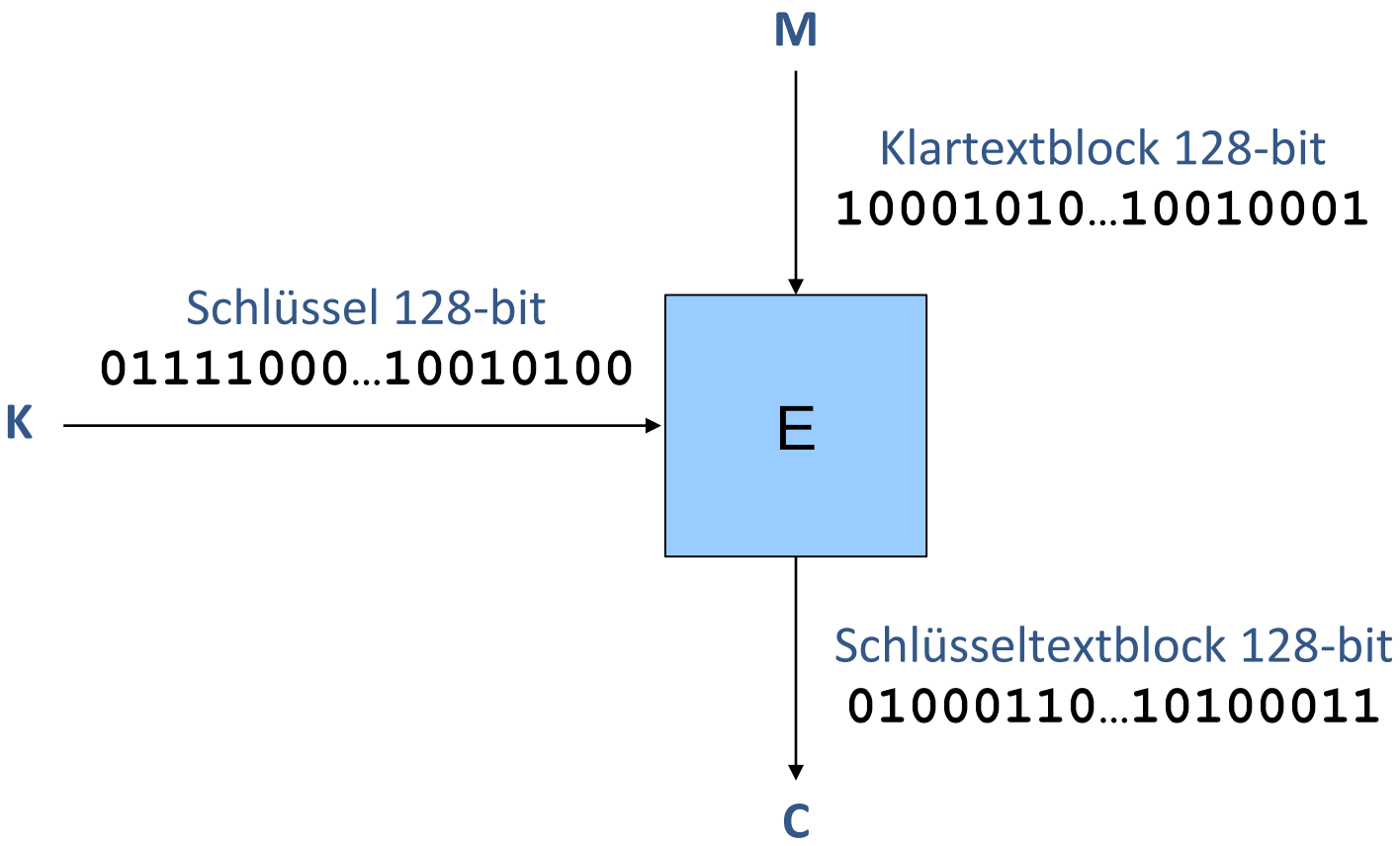
Musterlösung

(hoffentlich)

Kryptographie ist einfach, oder?

Symmetrische Verschlüsselung II

- Blockchiffren, beispielsweise AES-128



Häufige Fehler bei der Verwendung von Krypto-Libs I

8 Answers

active | oldest | votes

124

You could use functions like these:

```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.DECRYPT_MODE, skeySpec);
    byte[] decrypted = cipher.doFinal(encrypted);
    return decrypted;
}
```

Quelle: StackOverflow, https://stackoverflow.com/questions/6788018/android-encryption-decryption-with-aes

Ein erster Versuch

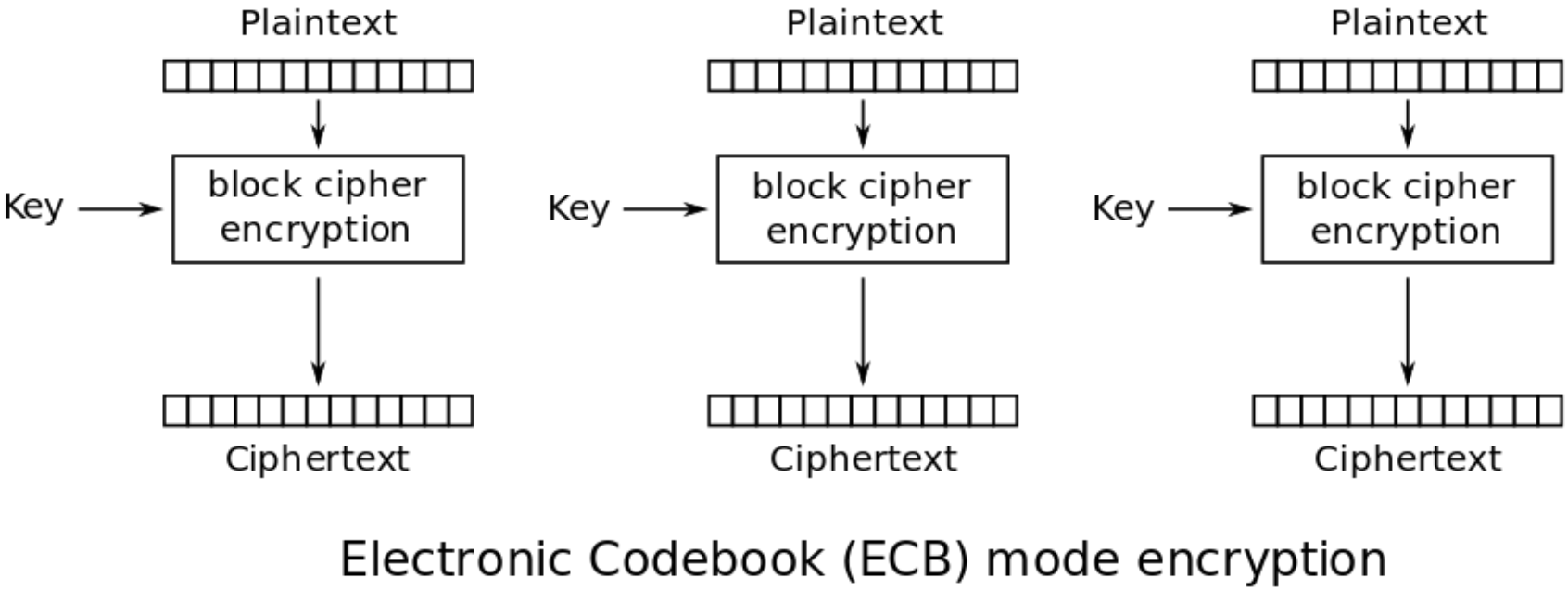
```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

public static void main(String[] args) throws Exception {
    ...
    byte[] plaintext = ...
    byte[] key = ...

    ciphertext = encrypt(plaintext, key);
    ...
}
```

Korrekte Parameterreihenfolge

```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

public static void main(String[] args) throws Exception {
    ...
    byte[] plaintext = ...
    byte[] key = ...

    ciphertext = encrypt(key, plaintext);
    ...
}
```

Symmetrische Verschlüsselung III

- Betriebsmodi: Wie werden Nachrichten länger als Blocklänge behandelt?

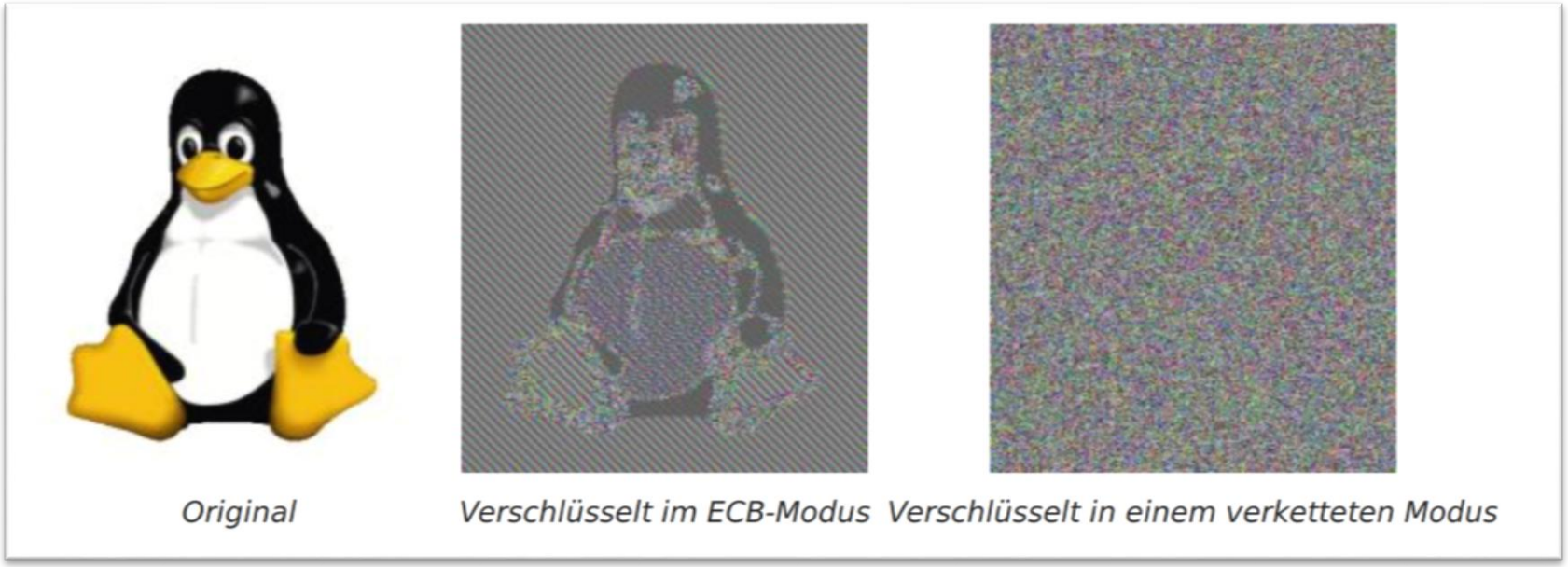


Electronic Codebook (ECB) mode encryption

Quelle: Wikipedia, https://de.wikipedia.org/wiki/Datei:ECB_encryption.svg

Symmetrische Verschlüsselung III

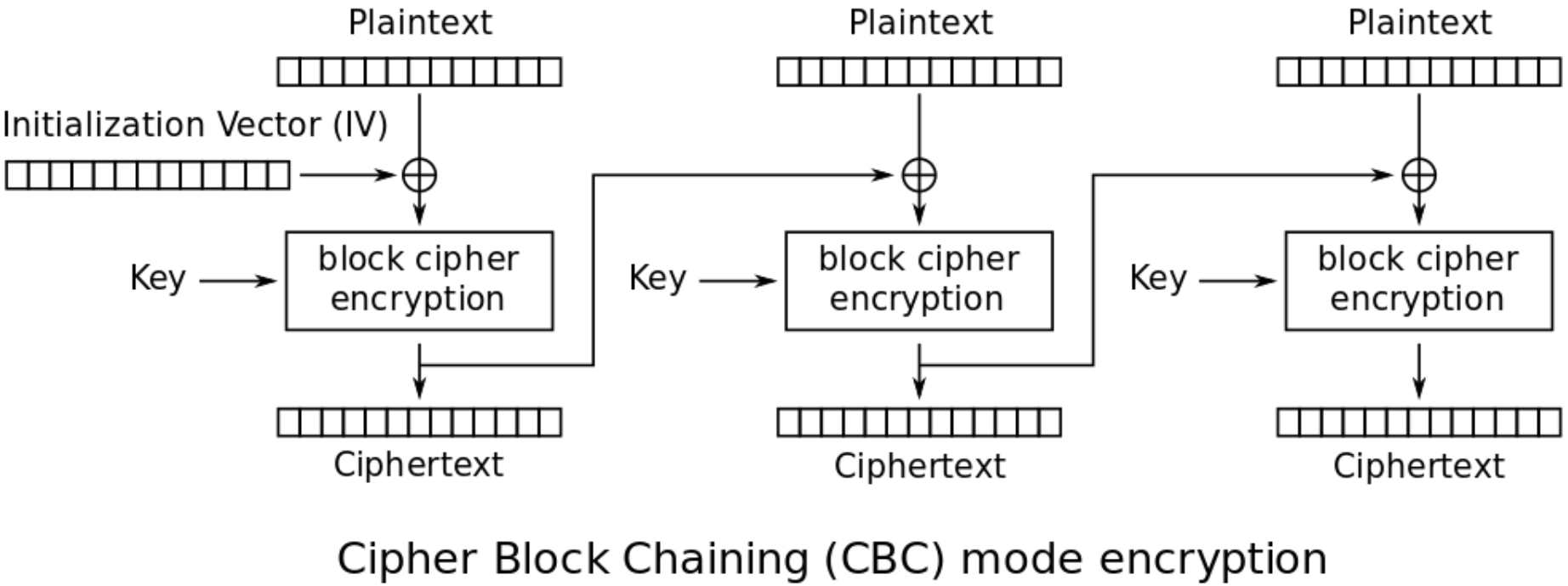
- Betriebsmodi: Wie werden Nachrichten länger als Blocklänge behandelt?



Quelle: Wikipedia, https://de.wikipedia.org/wiki/Electronic_Code_Book_Mode

Symmetrische Verschlüsselung III

- Betriebsmodi: Wie werden Nachrichten länger als Blocklänge behandelt?



Cipher Block Chaining (CBC) mode encryption

Quelle: Wikipedia, https://de.wikipedia.org/wiki/Datei:CBC_encryption.svg

Cipher Block Chaining Mode

```
private static byte[] encrypt(byte[] raw, byte[] clear, byte[] iv) throws Exception {
    IvParameterSpec ivSpec = new IvParameterSpec(iv);
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivSpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

public static void main(String[] args) throws Exception {
    ...
    byte[] plaintext = ...
    byte[] key = ...
    byte[] iv = new byte [IV_SIZE];
    new Random().nextBytes(iv);

    ciphertext = encrypt(key, plaintext, iv);
    ...
}
```

SecureRandom ftw

```
private static byte[] encrypt(byte[] raw, byte[] clear, byte[] iv) throws Exception {
    IvParameterSpec ivSpec = new IvParameterSpec (iv);
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivSpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

public static void main(String[] args) throws Exception {
    ...
    byte[] plaintext = ...
    byte[] key = ...
    byte[] iv = new byte [IV_SIZE];
    new SecureRandom().nextBytes(iv);

    ciphertext = encrypt(key, plaintext, iv);
    ...
}
```

Passwort als Grundlage für Schlüssel

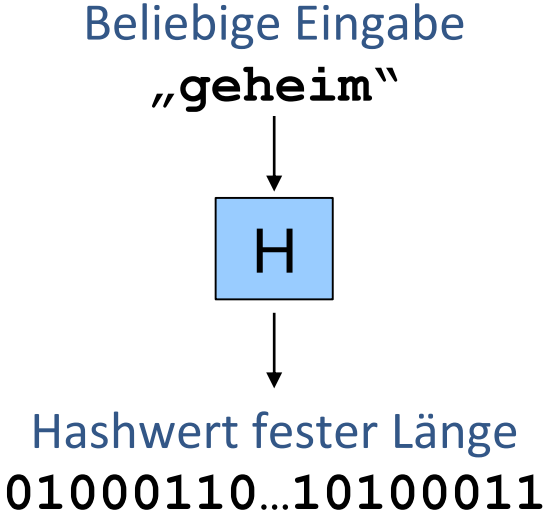
```
private static byte[] encrypt(byte[] raw, byte[] clear, byte[] iv) throws Exception {
    IvParameterSpec ivSpec = new IvParameterSpec (iv);
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivSpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}
```

```
private static byte[] deriveKey(String password) {
    byte[] keyBytes = new byte[KEY_SIZE];
    Arrays.fill(keyBytes, (byte) 0x0);
    byte[] passwordBytes = password.getBytes("UTF-8");
    int length = Math.min(passwordBytes.length, keyBytes.length);
    System.arraycopy(passwordBytes, 0, keyBytes, 0, length);
    return key;
}
```

```
public static void main(String[] args) throws Exception {
    ...
    String password = ...
    byte[] plaintext = ...
    byte[] key = deriveKey(password);
    byte[] iv = new byte [IV_SIZE];
    new SecureRandom().nextBytes(iv);

    ciphertext = encrypt(key, plaintext, iv);
    ...
}
```

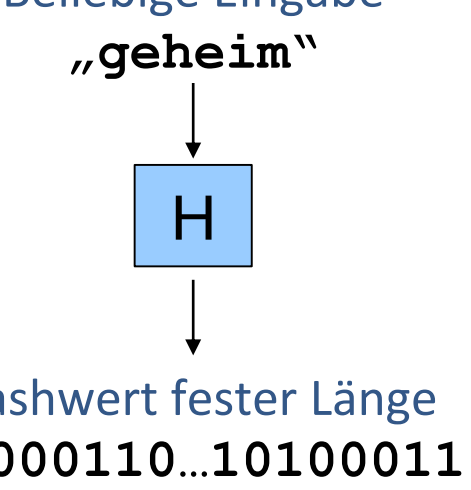
Schlüsselableitung (Key Stretching)

- Leitet aus dem Passwort einen kryptographischen Schlüssel ab
- Alle Bits des Schlüssels hängen vom Passwort ab
- Nicht umkehrbar



Schlüsselableitung (Key Stretching)

- Leitet aus dem Passwort einen kryptographischen Schlüssel ab
- Alle Bits des Schlüssels hängen vom Passwort ab
- Nicht umkehrbar
- Erschwert Brute-Force-Angriffe durch rechen- und speicherintensive Funktionen sowie Verwendung von Salt-Wert



Quelle: Mark Burnett, xato.net

Key stretching mit PBKDF2

```
private static byte[] encrypt(SecretKey skeySpec, byte[] clear, byte[] iv) throws Exception {
    IvParameterSpec ivSpec = new IvParameterSpec (iv);
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivSpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

private static SecretKey deriveKey(char[] password) throws Exception {
    byte[] salt = new byte[SALT_SIZE];
    new SecureRandom().nextBytes(salt);
    SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
    KeySpec spec = new PBEKeySpec(password, salt, ITERATIONS, KEY_SIZE);
    SecretKey tmp = factory.generateSecret(spec);
    SecretKey secret = new SecretKeySpec(tmp.getEncoded(), "AES");
    return secret;
}

public static void main(String[] args) throws Exception {
    ...
    char[] password = ...
    byte[] plaintext = ...
    SecretKey genKey = generateKey(password);
    byte[] iv = new byte [IV_SIZE];
    new SecureRandom().nextBytes(iv);

    ciphertext = encrypt(genKey, plaintext, iv);
    ...
}
```

Verbleibende Probleme

```
private static byte[] encrypt(SecretKey skeySpec, byte[] clear, byte[] iv) throws Exception {
    IvParameterSpec ivSpec = new IvParameterSpec (iv);
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivSpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;
}

private static SecretKey deriveKey(char[] password) throws Exception {
    byte[] salt = new byte[SALT_SIZE];
    new SecureRandom().nextBytes(salt);
    SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
    KeySpec spec = new PBEKeySpec(password, salt, ITERATIONS, KEY_SIZE);
    SecretKey tmp = factory.generateSecret(spec);
    SecretKey secret = new SecretKeySpec(tmp.getEncoded(), "AES");
    return secret;
}

public static void main(String[] args) throws Exception {
    ...
    char[] password = ...
    byte[] plaintext = ...
    SecretKey genKey = generateKey(password);
    byte[] iv = new byte [IV_SIZE];
    new SecureRandom().nextBytes(iv);

    ciphertext = encrypt(genKey, plaintext, iv);
    ...
}
```

Fazit

- Dont‘t invent your own crypto!
- Nutzt High-Level-Bibliotheken
  - libSodium
  - Tafelsalz ☺
- Fragt Menschen, die wissen was sie tun!
- Betrachtet Copy&Paste-Lösungen mit Vorsicht!