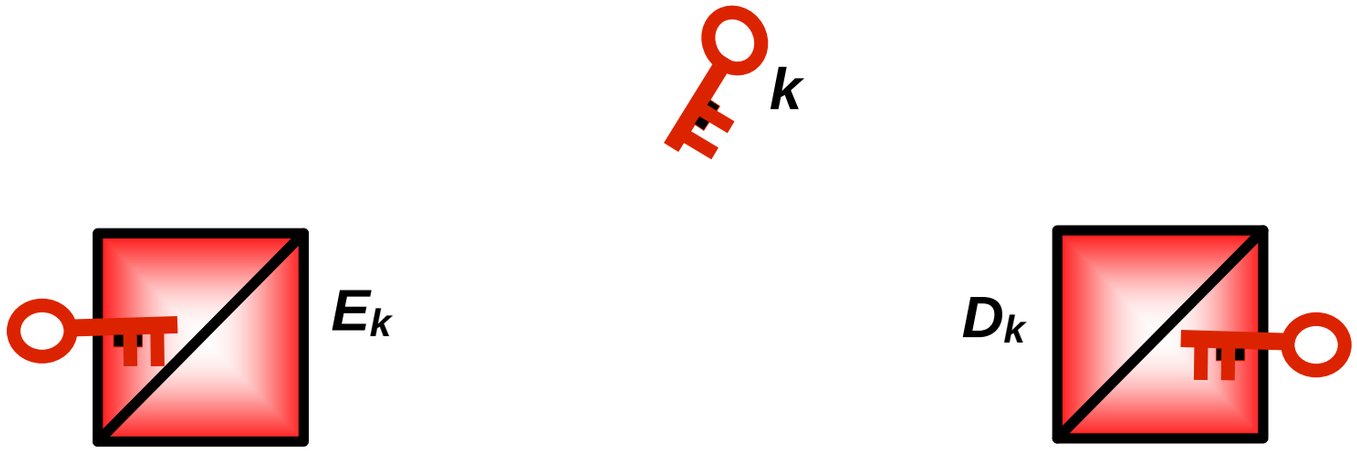
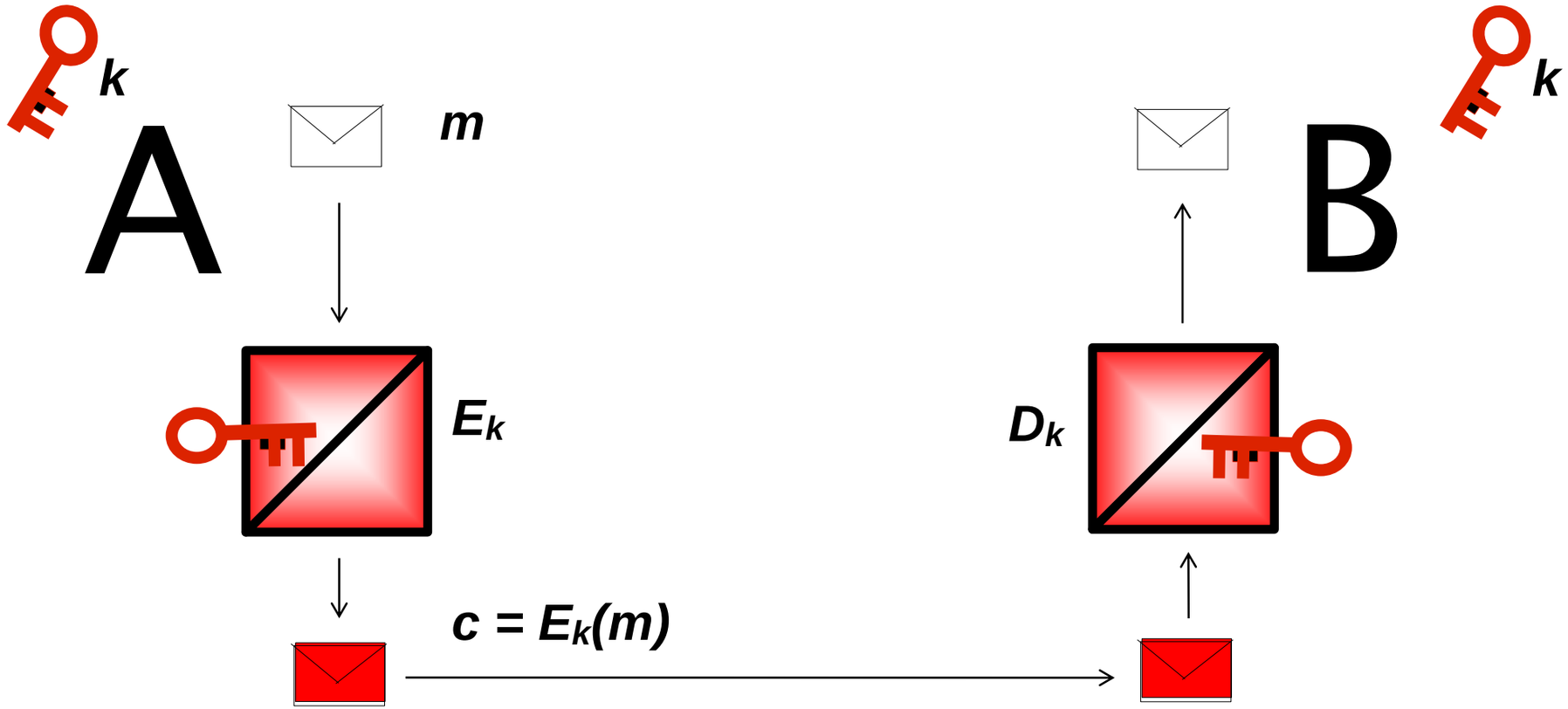
Symmetric Ciphers

- JCA implementations: Cipher




k
A
m
E_k
$c = E_k(m)$
D_k
B
k

- **Key exchange:**
  - Alice and Bob must share a secret key, which has to be exchanged over a secure channel, before it can be used to initialize an encryption algorithm to encrypt messages.

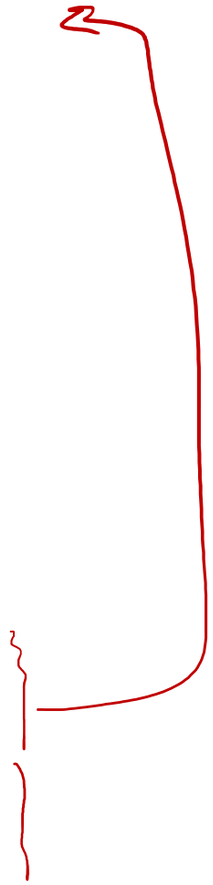
- **Key storage:**
  - Keys have to be securely managed and stored.

- **Aim of constructions of cipher algorithms:**
  - No attack has a better performance than a *Brute Force attack*.
  - This means: The size of the key space |*K*| (number of possible keys) is directly proportional to the security of the algorithm.

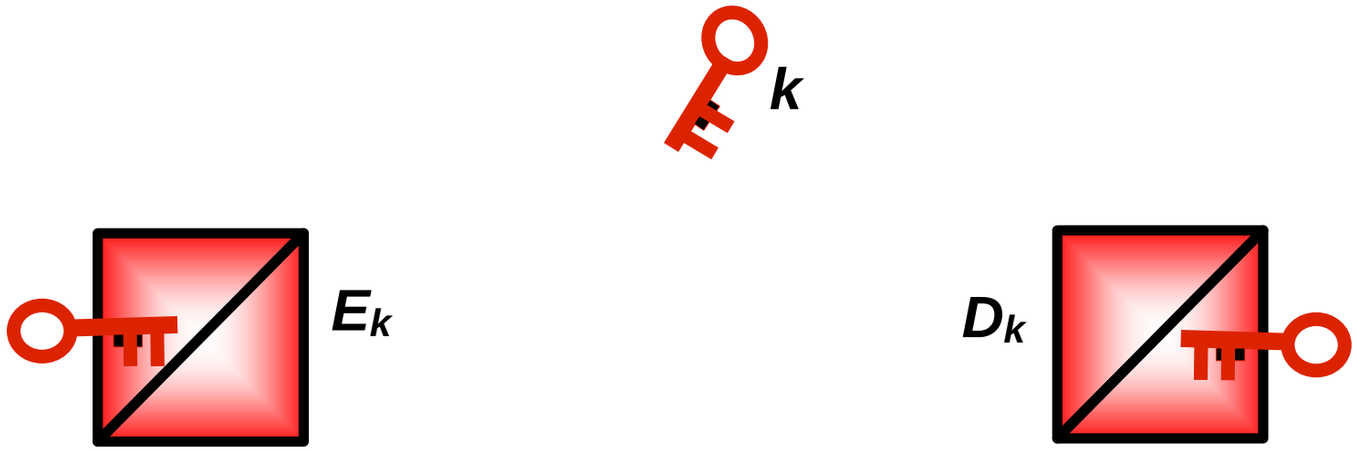
- **Stream ciphers**

- **Block ciphers**
- Modes of operation:
  - ECB (Electronic Codebook Modus)
  - CBC (Cipher Block Chaining Modus)
  - CFB (Cipher Feedback Modus)
  - OFB (Output Feedback Modus)
  - CTR (Counter)
  - ⋮

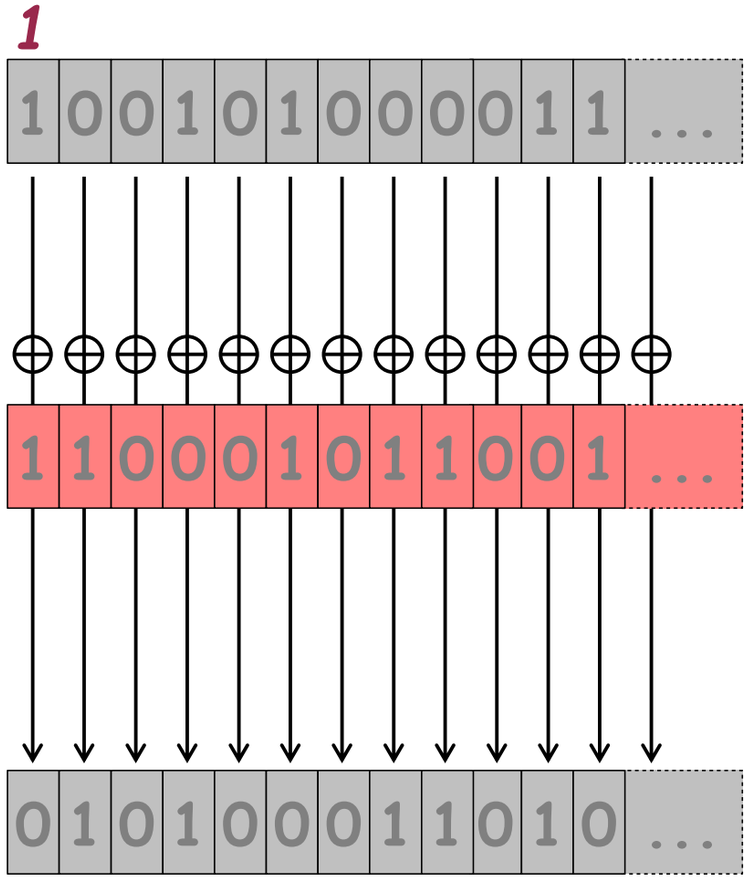
GCM (Galois Counter Mode) adds MAC



Symmetric Ciphers – Stream Ciphers



One Time Pad (OTP)



Plaintext (Bitstream)

One Time Pad *k* (generated by a real random process)

Ciphertext

One Time Pad (OTP) - Pros

- A truely randomly generated one time pad is the only cipher that guarantees absolut (provable) security.
- The only information that can be deduced from eavesdropping is the length of the plaintext.

One Time Pad (OTP) - Cons

- **Key establishment**

  The one time pad has to be exchanged over some other secure channel prior to its use.

- **Key length**

  The one time pad (key) has to be as long as the plaintext.

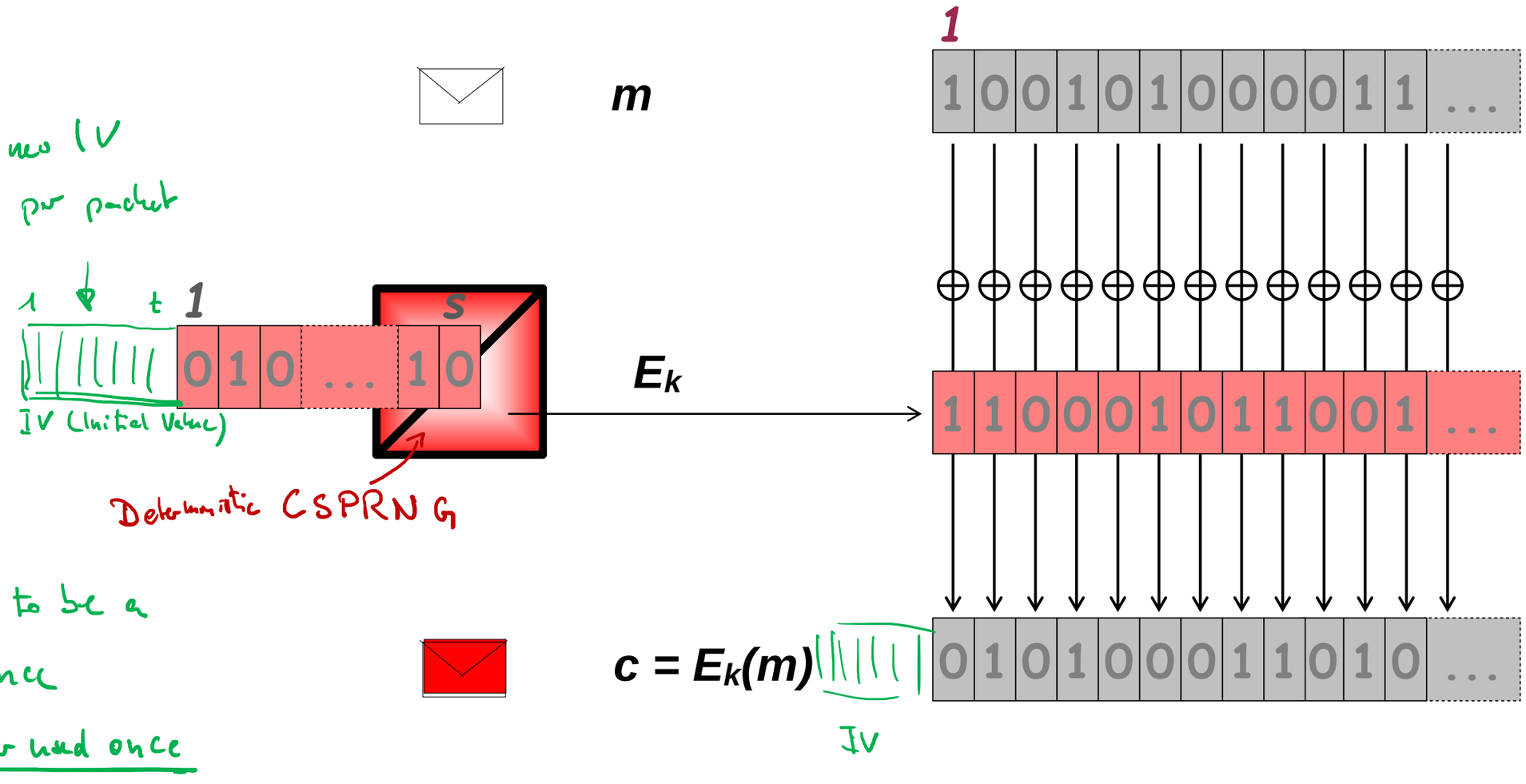
- **Reusability**

  Reusage of a one time pad is strictly prohibited, as it would allow an attack by statistical analysis.

- **Key generation**

  Costly, as a real physicaly random process has to be used.

Additive Synchronous Stream Ciphers



IV has to be a Nonce

Number used once

Additive Synchronous Stream Ciphers – Pros

- The keystream is independent of the plaintext. (Keystream can be precalculated.)
- Encryption is a simple (fast) XOR operation.
- Decryption = Encryption

- => often used in Networks implemented on small devices
- => mostly in Wireless Networks
- often AES is used in the CSPRNG

Additive Synchronous Stream Ciphers – Cons

- **Key establishment**

  The key has to be exchanged over some other secure channel prior to its use.

- **Reusability**

  Reusage of the same key is strictly prohibited, as it compromises the encryption scheme.

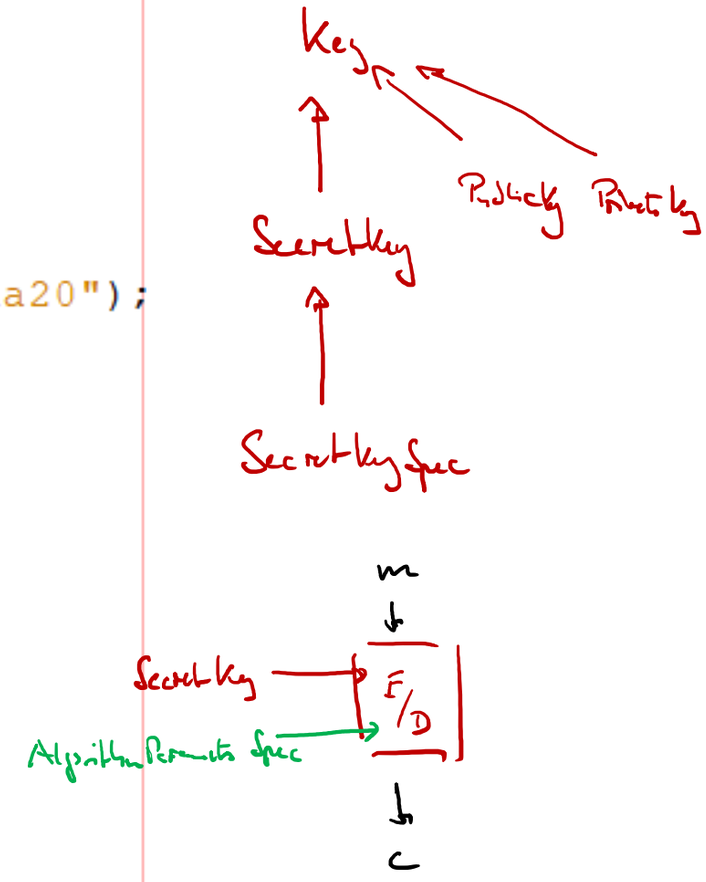
- **Integrity**

  Integrity is not protected: Single bits can be switched by an attacker.

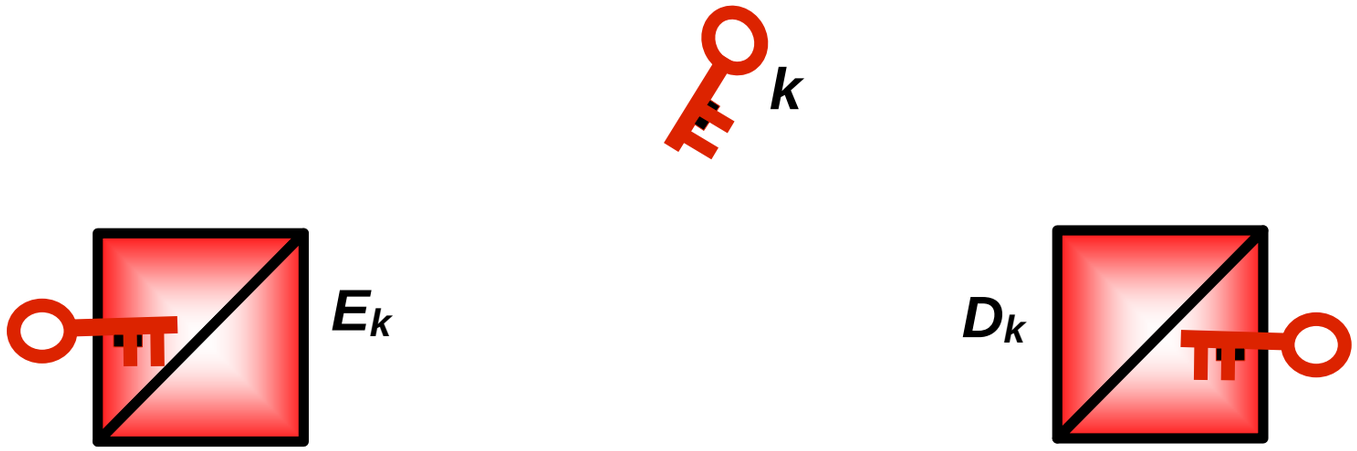
- Additive Synchronous Stream Cipher specified in:
  - **RFC 8439 - ChaCha20 and Poly1305 for IETF Protocols**
- Currently the only alternative to the AES cipher defined for record layer protocol encryption in TLS 1.3 (used in combination with the Poly1305 authenticator). See RFC 8446, B.4. Cipher Suites).

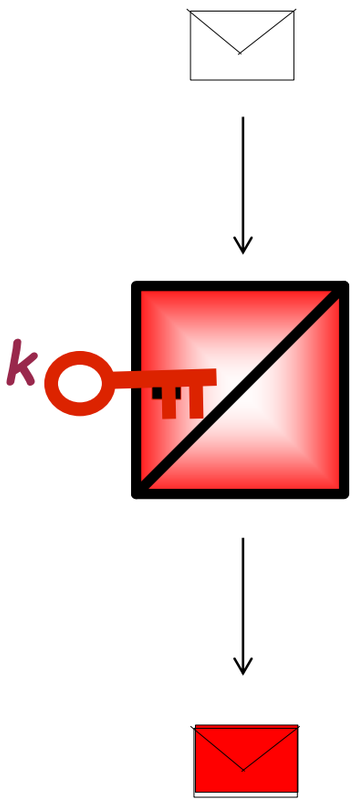
ChaCha20_Demo

```
31      Cipher cipher = Cipher.getInstance("ChaCha20");
32
33      byte[] key = new byte[32];
34      for( byte i = 0; i < 32; ++i ) {
35        key[i] = i;
36      }
37      SecretKeySpec keyChaCha20 = new SecretKeySpec(key, "ChaCha20");
38
39      byte[] nonce
40       = new byte[]{0, 0, 0, 0, 0, 0, 0, 0x4a, 0, 0, 0, 0};
41      AlgorithmParameterSpec params
42       = new ChaCha20ParameterSpec(nonce, 1);
43      cipher.init(Cipher.ENCRYPT_MODE, keyChaCha20, params);
44
45      byte[] m = ("Ladies and Gentlemen of the class of '99:"
46       + " If I could offer you only one tip for the future,"
47       + " sunscreen would be it.").getBytes();
48      byte[] c = cipher.doFinal(m);
```



Symmetric Ciphers – Block Ciphers





m

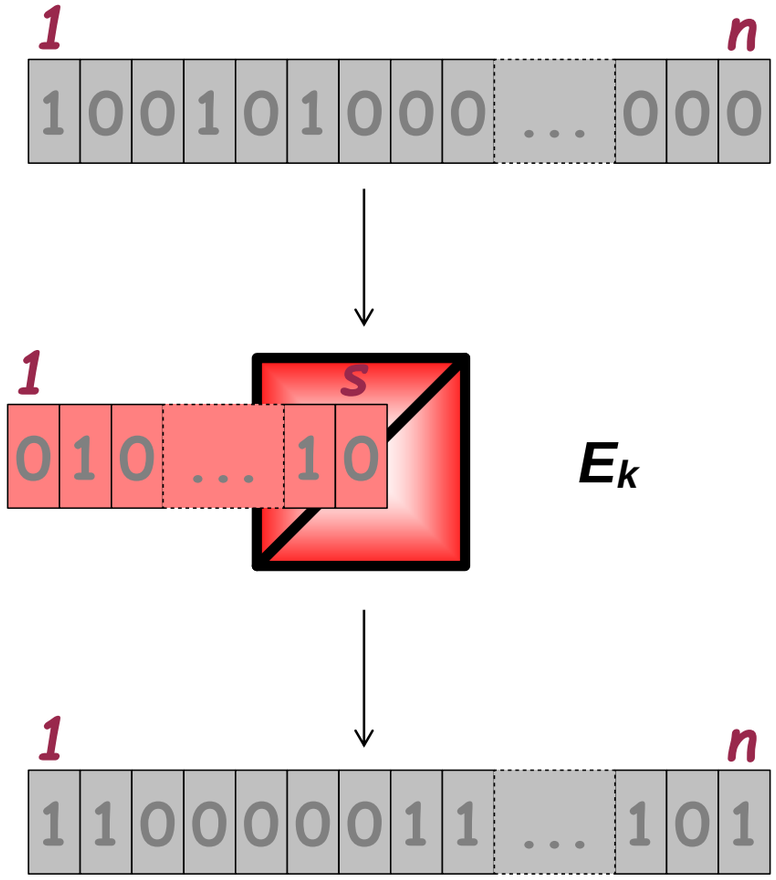
E_k

$c = E_k(m)$

n bit plaintext (block length n)

$k \in K$ (key space K)

n bit cipher text



$m \in (F_2)^n$ **(block)**

$k \in K = (F_2)^s$

$$E : K \times (F_2)^n \rightarrow (F_2)^n$$

$$E_k : (F_2)^n \rightarrow (F_2)^n$$

has to be a bijective mapping for every $k \in K$

$c \in (F_2)^n$ **(cipher block)**

- How many different block ciphers can be defined for the encryption of blocks of length n ?

- Why not use random permutations of the set of all 2^n blocks of length n for the construction of block ciphers?

- A block cipher shall exhibit the same statistical features as a random permutation of all 2^n blocks of length n.
- Encryption and Decryption shall be efficiently implementable in SW and HW (runtime performance, memory requirements).
- Block ciphers are usually organized in rounds, where the following types of basic operations are repeatedly executed:
  - Permutations of the bits of a block.
  - Substitutions (S-Boxes) of values in subblocks.

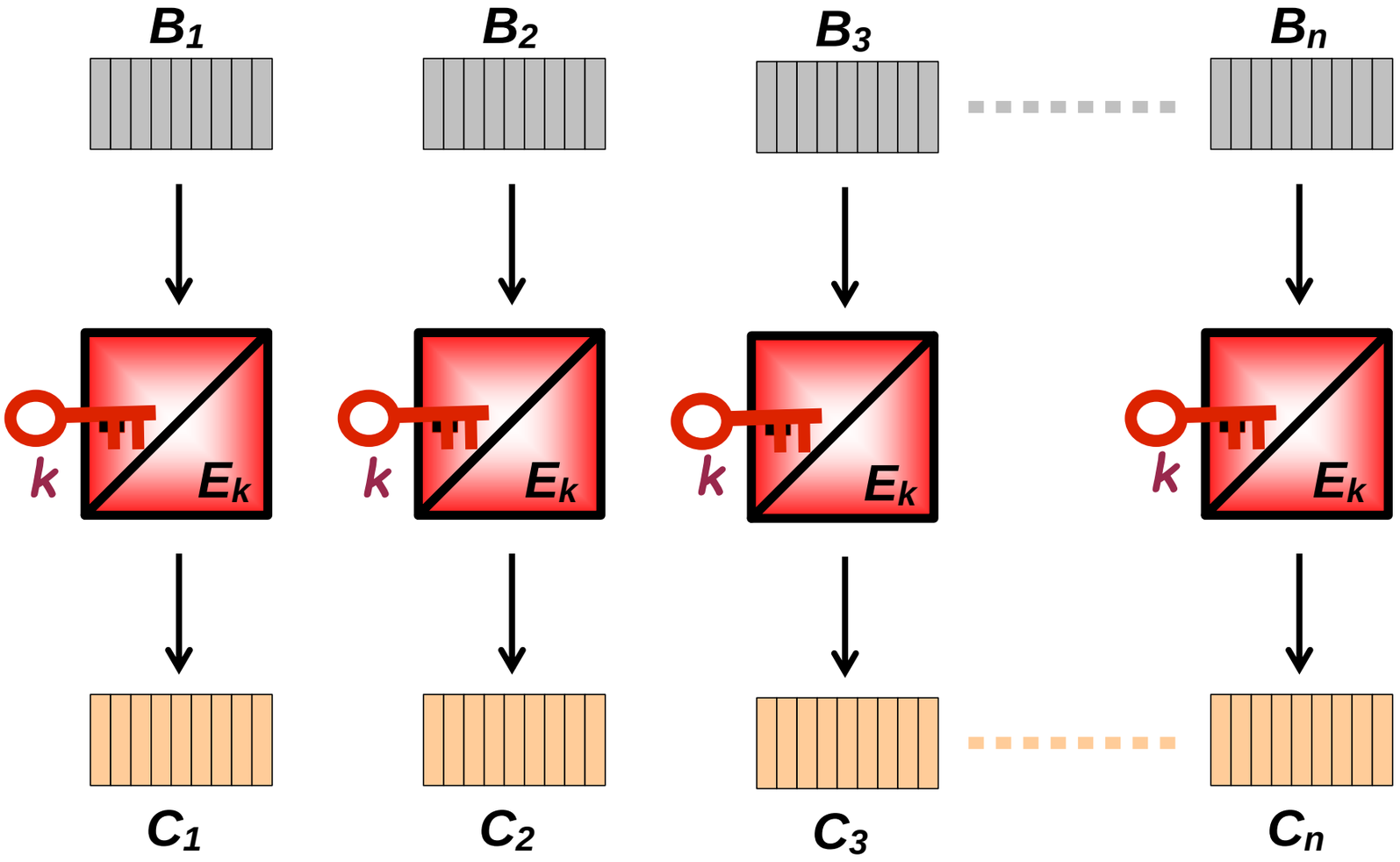
- **AES (Advanced Encryption Standard)**
  - Specified key lengths: 128, 192, 256 bit
  - Block length: 128 bit
  - Winning algorithm (Rijndal algorithm) from an international contest (organised by NIST).
  - US Federal Standard FIPS PUB 197, published 2001.
  - Nice animation available in Cryptool1
    - https://www.cryptool.org/en/ct1
    - CryptTool -> Indiv. Procedures -> Visualisation of Algorithms -> AES -> Rijndale Animation -> Steuerung -> Abspielen

P QC - strength $\sqrt{2^{128}} = 2^{64}$
key lengths: 64, 96, 128

- The length of plaintext data must be a multiple of *n*.
  - Padding operations are needed.
- Simply encrypting data block by block (ECB modus) may allow dictionary attacks. To prevent such attacks, use:
  - CBC modus
  - Random IV values

ECB	Electronic Code Book
CBC	Cipher Block Chaining
IV	Initialization Vector

ECB mode (Electronic Code Book)



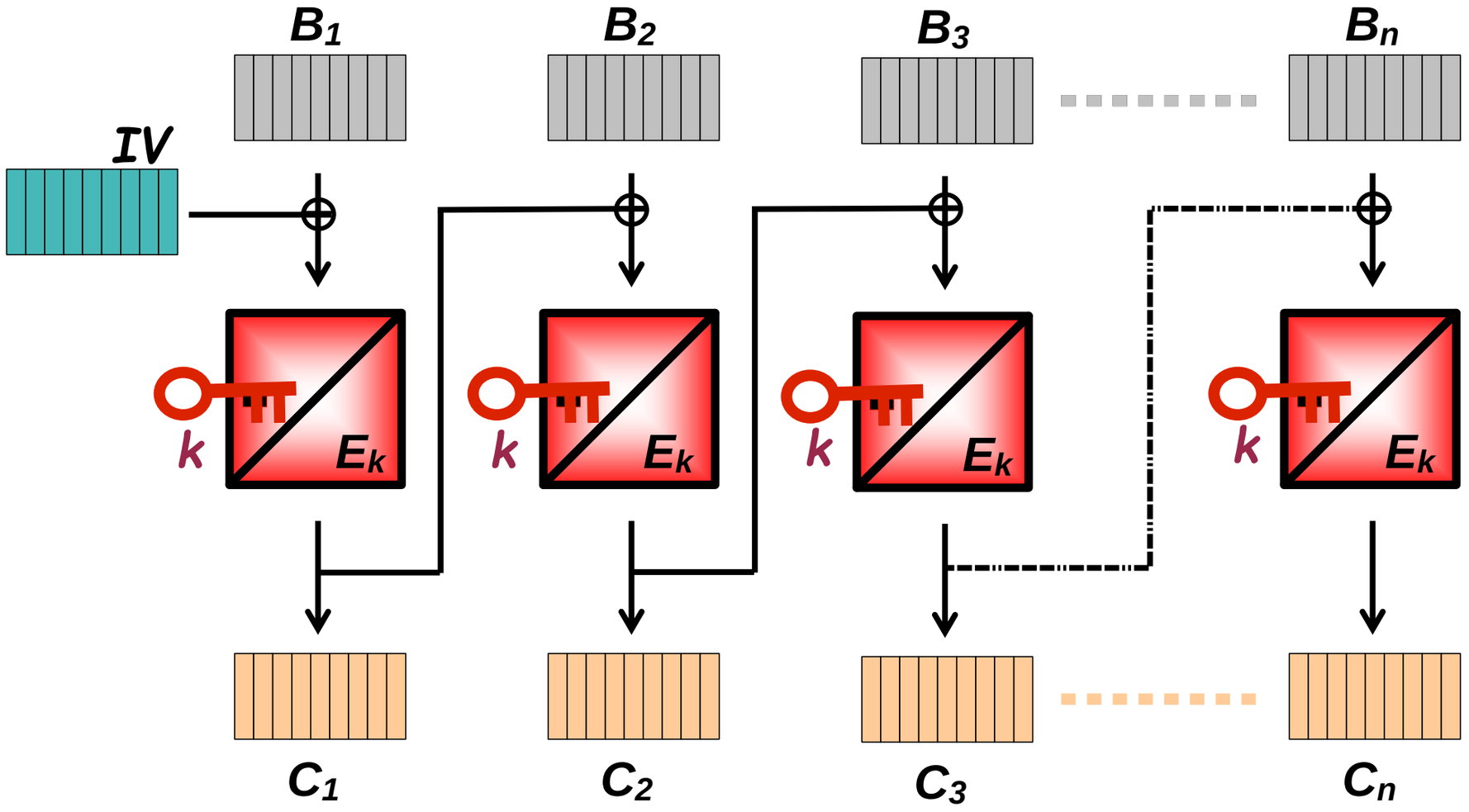
Cipher_ECB_Demo

```
18        byte[] key = Dump.hexString2byteArray(
19         "0102030405060708090A0B0C0D0E0F10");
20        SecretKey secretKey = new SecretKeySpec(key, "AES");
21
22        Cipher cipher
23         = Cipher.getInstance("AES/ECB/NOPADDING");
24
25        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
26
27        byte[] m = new byte[48];
```

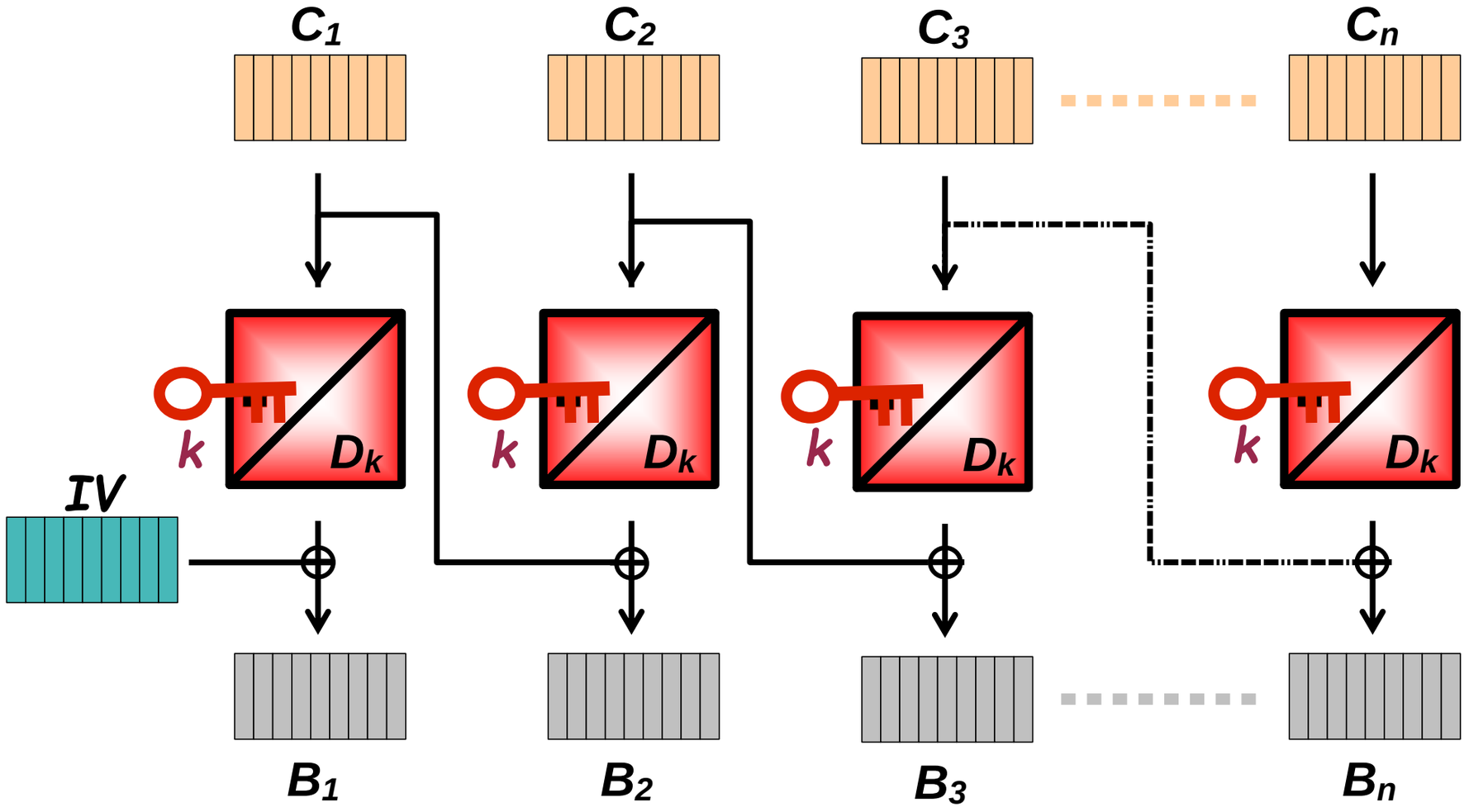
```
35        byte[] c = cipher.doFinal(m);
36
37        System.out.println("Ciphertext:");
38        System.out.println(Dump.dump(c));
```

```
40        cipher.init(Cipher.DECRYPT_MODE, secretKey);
41        byte[] m2 = cipher.doFinal(c);
42        System.out.println("Decrypted Ciphertext:");
43        System.out.println(Dump.dump(m2));
```

CBC mode (Cipher Block Chaining) - Encryption



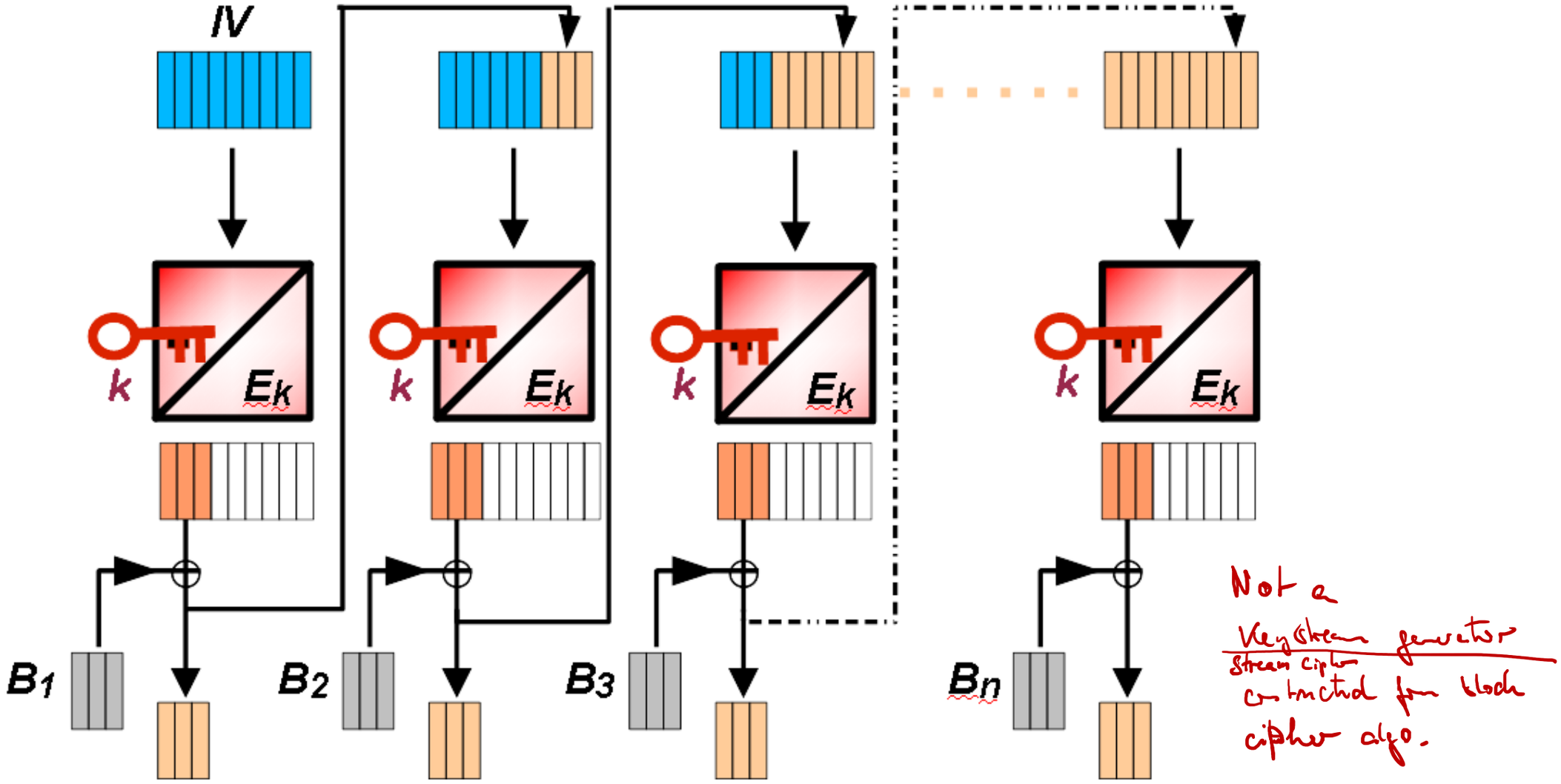
CBC mode (Cipher Block Chaining) - Decryption



Cipher_CBC_Demo

```
20          byte[] key = Dump.hexString2byteArray(
21           "0102030405060708090A0B0C0D0E0F10");
22          SecretKey secretKey = new SecretKeySpec(key, "AES");
23
24          Cipher cipher
25           = Cipher.getInstance("AES/CBC/PKCS5PADDING");
26
27          byte[] ivBytes = new byte[16];
28          new SecureRandom().nextBytes(ivBytes);
29          IvParameterSpec iv = new IvParameterSpec(ivBytes);
30          cipher.init(Cipher.ENCRYPT_MODE, secretKey, iv);
31
32          byte[] m = new byte[48];
```

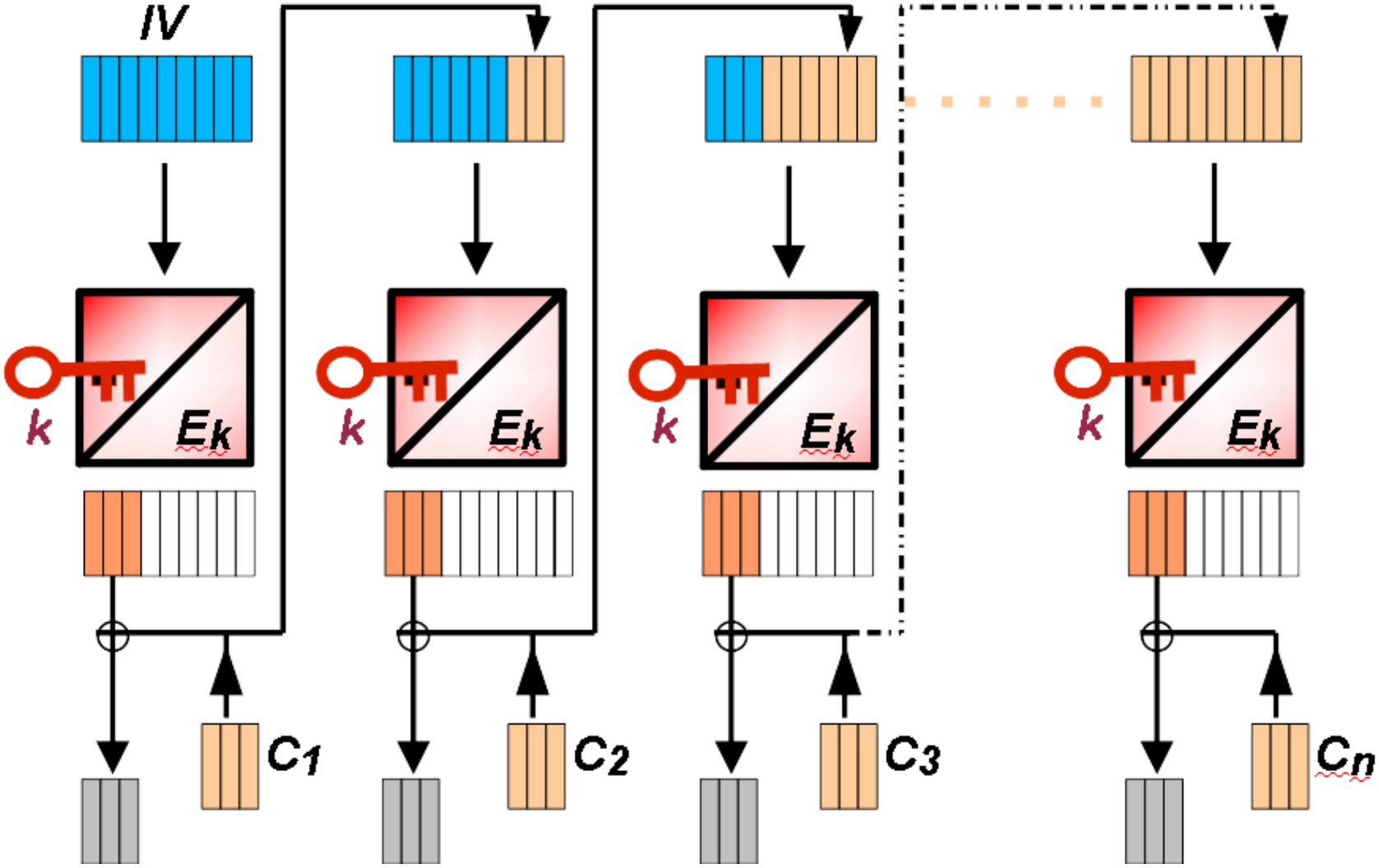
CFB mode (Cipher Feedback) - Encryption



Not a
Key stream generator
Stream cipher
construction from block
cipher algo.

no precalculation possible

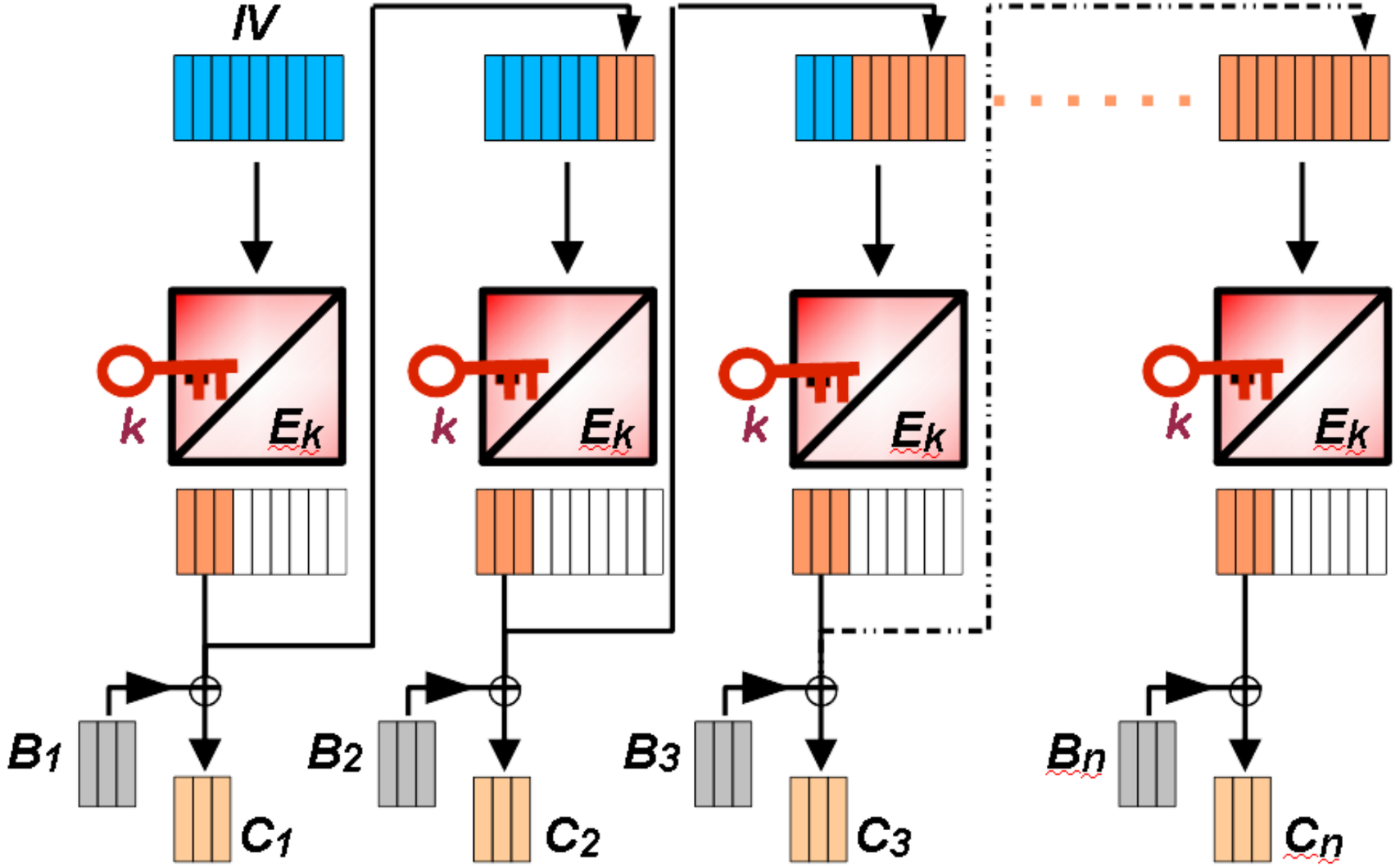
CFB mode (Cipher Feedback) - Decryption



Cipher_CFB_Demo

```
19      byte[] key = Dump.hexString2byteArray(
20       "0102030405060708090A0B0C0D0E0F10");
21      SecretKey secretKey = new SecretKeySpec(key, "AES");
22
23      Cipher cipher
24       = Cipher.getInstance("AES/CFB24/NOPADDING");
25
26      byte[] ivBytes = new byte[16];
27      (new Random()).nextBytes(ivBytes);
28      IvParameterSpec iv = new IvParameterSpec(ivBytes);
29      cipher.init(Cipher.ENCRYPT_MODE, secretKey, iv);
30
31      byte[] m = "Test".getBytes();
32
33      System.out.println("Plaintext:");
34      System.out.println(Dump.dump(m));
35      System.out.println();
36
37      byte[] c = cipher.doFinal(m);
38
```

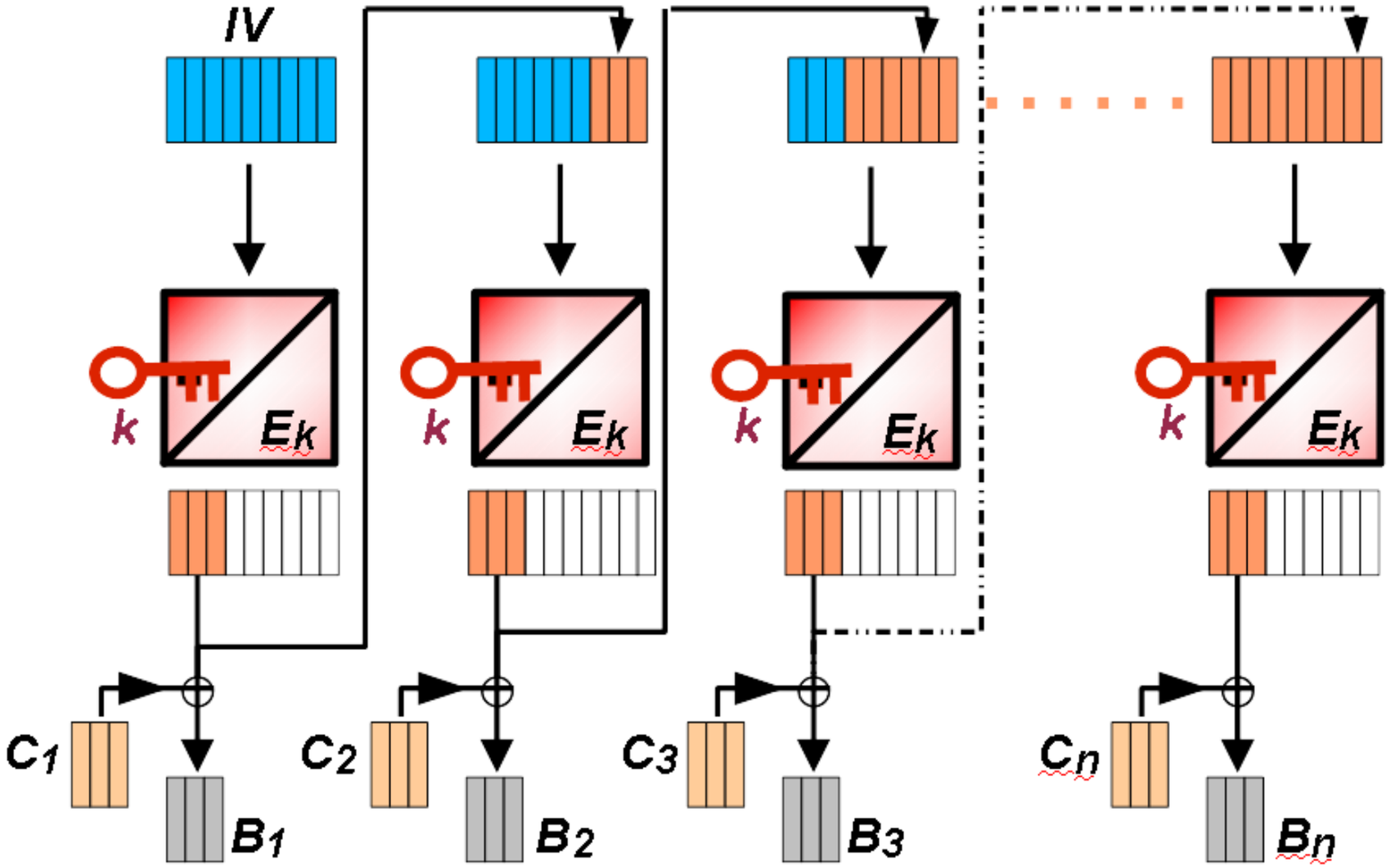
OFB mode (Output Feedback) - Encryption



Keystream generator

precalculation possible!

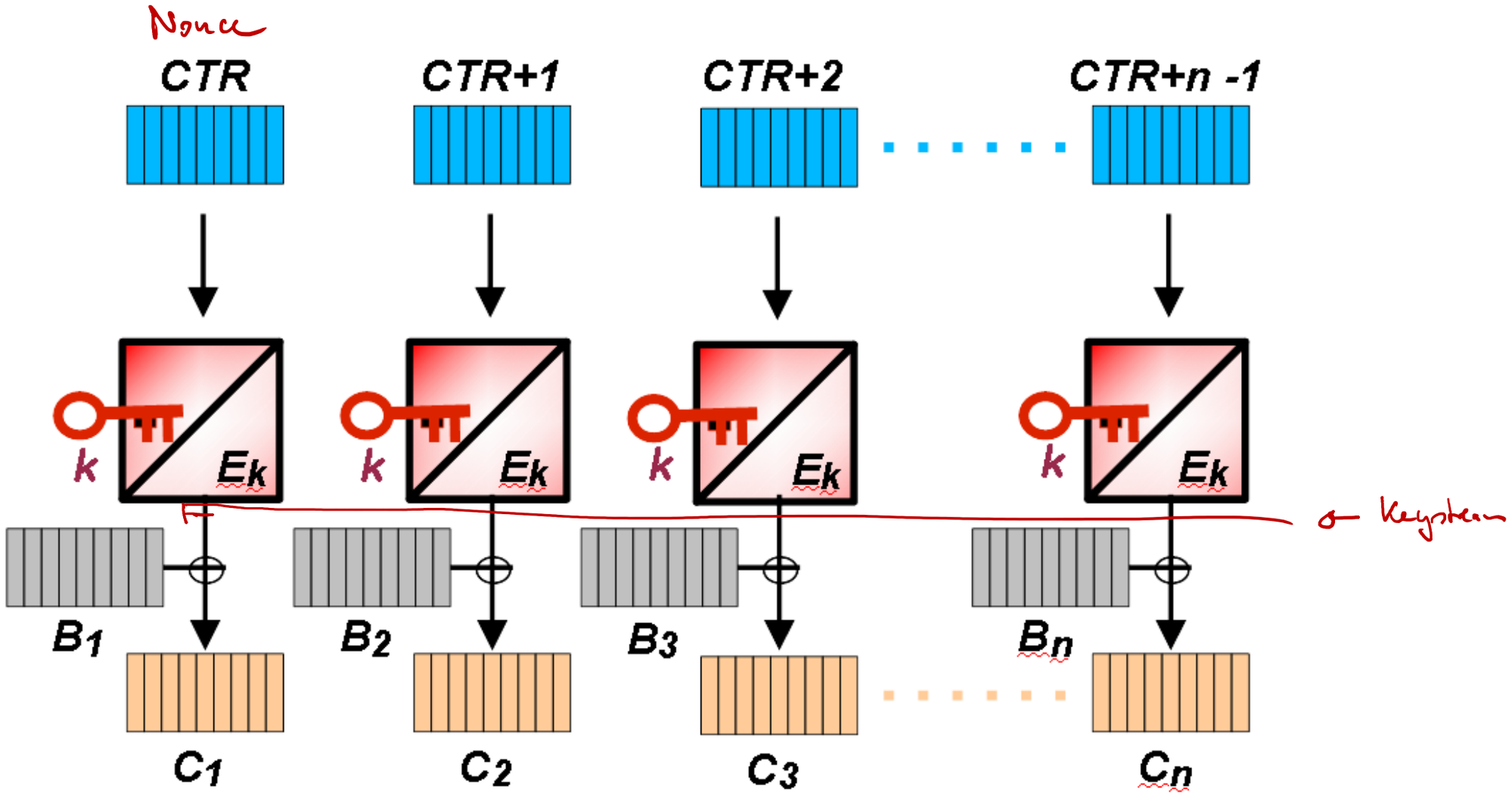
OFB mode (Output Feedback) - Decryption



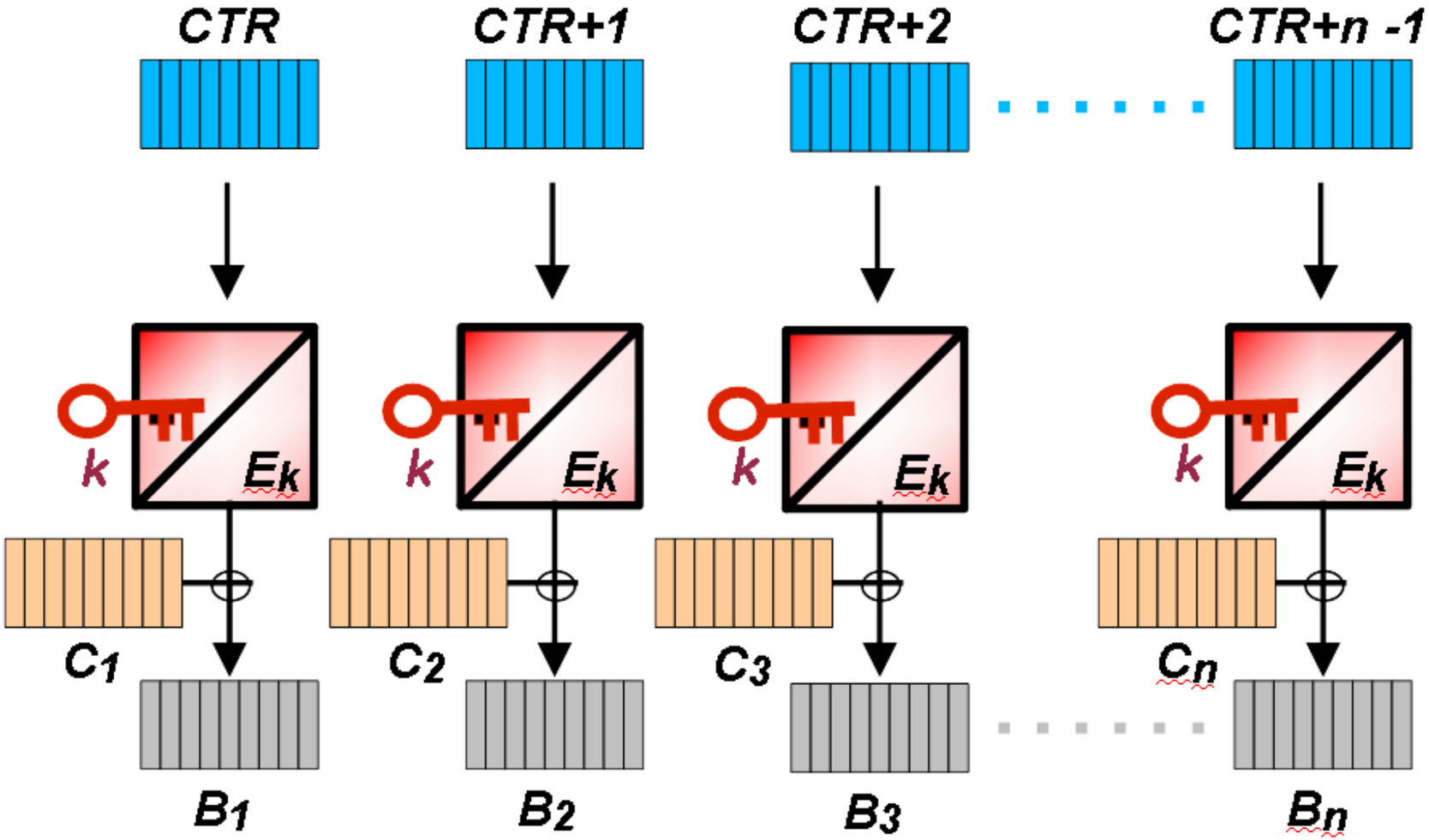
Cipher_OFB_Demo

```
19          byte[] key = Dump.hexString2byteArray(
20           "0102030405060708090A0B0C0D0E0F10");
21          SecretKey secretKey = new SecretKeySpec(key, "AES");
22
23          Cipher cipher
24      //      = Cipher.getInstance("AES/OFB24/NOPADDING");
25           = Cipher.getInstance("AES/OFB/NOPADDING");
26
27          byte[] ivBytes = new byte[16];
28          (new SecureRandom()).nextBytes(ivBytes);
29          IvParameterSpec iv = new IvParameterSpec(ivBytes);
30          cipher.init(Cipher.ENCRYPT_MODE, secretKey, iv);
31
32          byte[] m = "Test".getBytes();
33
34          System.out.println("Plaintext:");
35          System.out.println(Dump.dump(m));
36          System.out.println();
37
38          byte[] c = cipher.doFinal(m);
```

CTR mode (Counter) - Encryption



CTR mode (Counter) - Decryption



Cipher_CTR_Demo

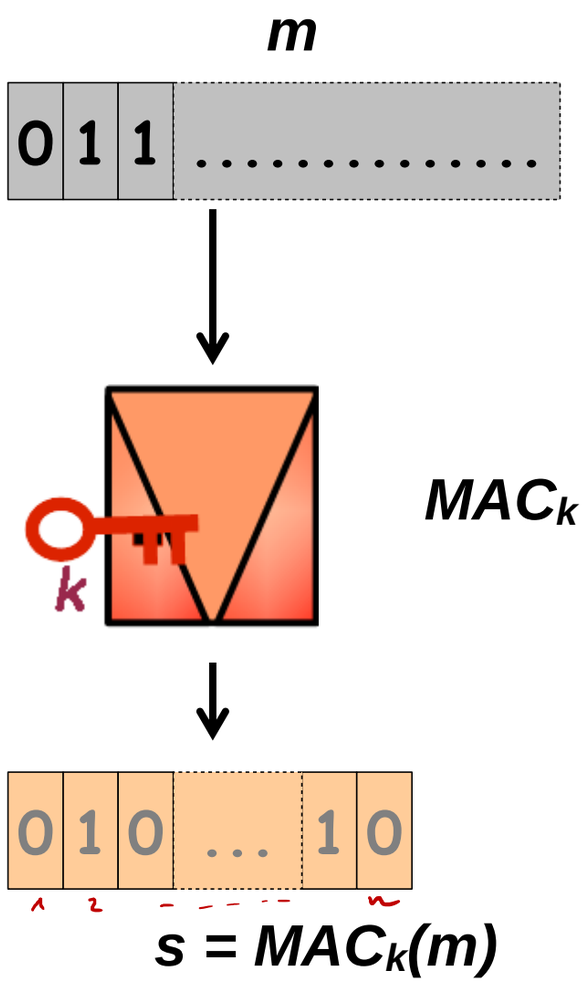
```
18      byte[] key = Dump.hexString2byteArray(
19       "0102030405060708090A0B0C0D0E0F10");
20      SecretKey secretKey = new SecretKeySpec(key, "AES");
21
22      Cipher cipher
23       = Cipher.getInstance("AES/CTR/NOPADDING");
24
25      byte[] ctr = Dump.hexString2byteArray(
26       "FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE");
27      IvParameterSpec iv = new IvParameterSpec(ctr);
28      cipher.init(Cipher.ENCRYPT_MODE, secretKey, iv);
29
30      byte[] m = new byte[35];
31
32      byte[] c = cipher.doFinal(m);
```

MAC‘s (Message Authentication Codes)

A Message Authentication is a hash function that depends on a key k.

Again the set of all binary sequences of finite length $m = (m_1,m_2,m_3,...)$ is mapped to the set of binary sequences of some fixed length n:

$Mac_k(m) = (h_1,h_2,...,h_n) \epsilon (F_2)^n$



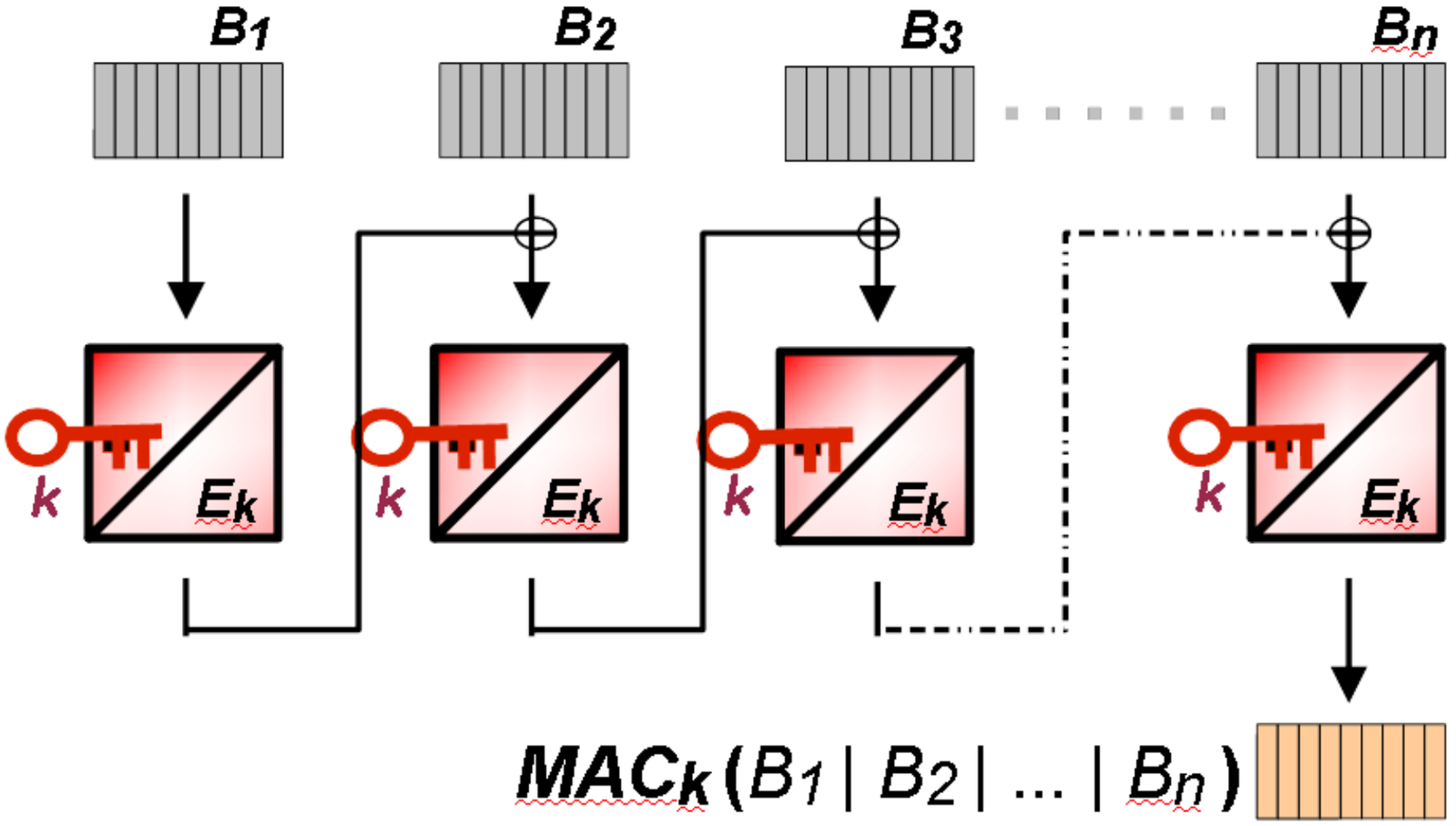
Applications:

Protection of message authenticity within a (closed) group of users with a common secret key.

Constructions of MAC‘s:

- CBC MAC based on a symmetric block cipher
- HMAC (Hash MAC) based on a hash function

CBC MAC (principle idea, for better solution see spec.)

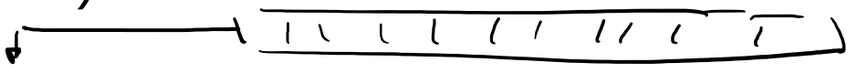


RFC 2104 - Keyed-Hashing for Message Authentication

- Hash function H, using a compression function which compresses b bytes from the input per round.
- (Example: b = 64 for SHA-1)
- Hash output length h (Example: h = 20 für SHA-1)
- HMAC(m) = H(k XOR opad | H(k XOR ipad | m))

  ipad = 0x36 | 0x36 | ... | 0x36 (b Bytes)

  opad = 0x5c | 0x5c | ... | 0x5c (b Bytes)



← At least 4 operations of the internal compression function of H have to be performed !

HMAC_Demo

```
25  public static void main(String[] args) throws Exception {
26      byte[] key = Dump.hexString2byteArray("17b52858e3e135be4440d7c
27      byte[] msg = Dump.hexString2byteArray("e0eff00f3c46e96c8d5bd18
28
29      SecretKey secretKey = new SecretKeySpec(key, "HmacSHA256");
30
31      Mac mac = Mac.getInstance("HmacSHA256");
32      mac.init(secretKey);
33      byte[] macValue = mac.doFinal(msg);
34
35      System.out.println("MAC value:");
36      System.out.println(Dump.dump(macValue));
37  }
```

RFC 5869 - HMAC-based Extract-and-Expand Key Derivation Function (HKDF)

- HKDF-Extract(salt, IKM) -> PRK
- HKDF-Expand(PRK, info, L) -> OKM
- Used in TLS 1.3 for key derivations, see RFC 8446, 7.1.

7.1. Key Schedule

The key derivation process makes use of the HKDF-Extract and HKDF-Expand functions as defined for HKDF [RFC5869], as well as the functions defined below:

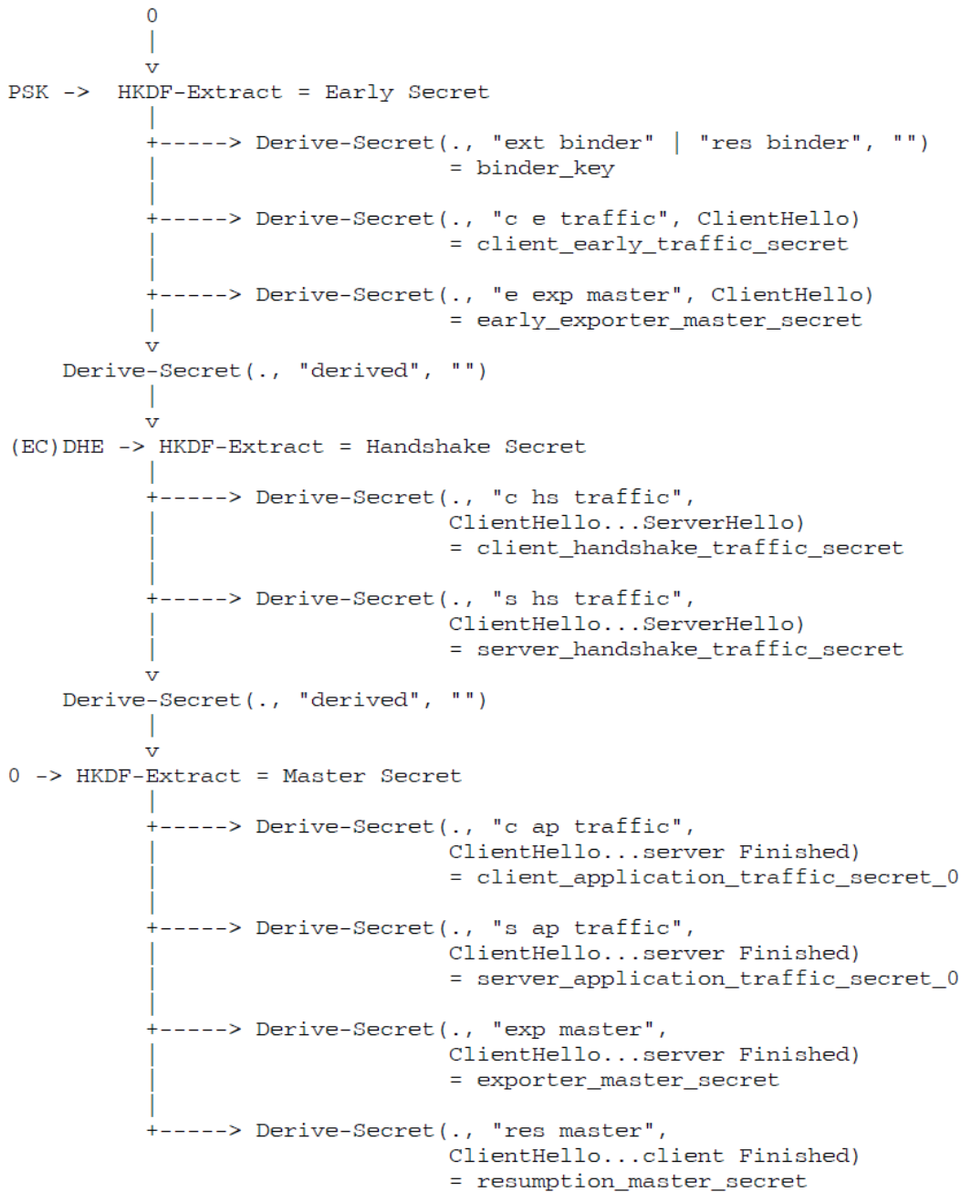
```
HKDF-Expand-Label(Secret, Label, Context, Length) =
     HKDF-Expand(Secret, HkdfLabel, Length)

Where HkdfLabel is specified as:

struct {
    uint16 length = Length;
    opaque label<7..255> = "tls13 " + Label;
    opaque context<0..255> = Context;
} HkdfLabel;

Derive-Secret(Secret, Label, Messages) =
     HKDF-Expand-Label(Secret, Label,
                       Transcript-Hash(Messages), Hash.length)
```

TLS 1.3 – Key Schedule



(R.8) Definition. If r is an element of a Ring R and there exists some $n \in \mathbb{N}$ with $r^n = 1$, than the smallest such integer is called the order of r, denoted by

$$o(r) .$$

(R.9) Remark. Assume R is a ring and $r \in R$ has finite order $o(r) \in \mathbb{N}$. Than

(i) r is invertibel with inverse $r^{o(r)-1}$, and

(ii) the elements $1, r, r^2, ..., r^{o(r)-1}$ are pairwise distinct.

Ex.: $\mathbb{Z}_{12}$: $o(5)$: $5, 5^2 = 1$ $o(5) = 2$

$3, 3^2 = 9, 3^3 = 3$ → $3 \in \mathbb{Z}_{12}$ has no order !

$\mathbb{Z}_7$: $o(1) = 1$

$2, 2^2 = 4, 2^3 = 1$ $o(2) = 3$, $2^{-1} = 4$ $o(4) = 3$, $4^{-1} = 2$

$3, 2, 6, 4, 5, 1$ $o(3) = 6$, $3^{-1} = 5$ $o(5) = 6$, $5^{-1} = 3$

$-1 = 6, 1$ $o(6) = 2$, $6^{-1} = 6$

(P.8) Construction of $\mathbb{F}_{p^n} = GF(p^n)$.

Let p be a prime number. Given an irreducible polynomial $m(x)$ over $GF(p)$ of degree n, the set of all polynomials in $GF(p)[x]$ of degree $< n$

$$F = \{a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_{n-1} \cdot x^{n-1} \mid a_i \in GF(p)\}$$

is a field with respect to the usual addition of polynomials and multiplication defined by reduction modulo $m(x)$:

$$f_1(x) \cdot_F f_2(x) = (f_1(x) \cdot f_2(x)) \mod m(x)$$

(P.9) Example. $\mathbb{F}_{2^8}$

The elements of the field $\mathbb{F}_{2^8}$ are the polynomials over $\mathbb{F}_2$ of degree ≤ 7, where addition is the ordinary addition of polynomials and multiplication can be defined to be the ordinary multiplication of polynomials followed by a reduction modulo the irreducible polynomial

$$m(x) \;:=\; x^8 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]$$

Moreover, identifying the elements of $\mathbb{F}_{2^8}$ with their bit sequences of coefficients and the corresponding byte values, we may multiply byte values according to this field structure. E.g.:

$$\begin{aligned} \texttt{0x5D} \cdot \texttt{0x14} \quad &\longleftrightarrow \quad \left((x^6 + x^4 + x^3 + x^2 + 1) \cdot (x^4 + x^2)\right) \bmod m(x) \\ &= \quad x^{10} + x^7 + x^5 + x^2 \bmod m(x) \\ \texttt{0xC8} \quad &\longleftrightarrow \quad = \quad x^7 + x^6 + x^3 \end{aligned}$$

01011101