Block cipher modes of operation

Dr. Yee Wei Law ⟨yeewei.law@unisa.edu.au⟩

2023-10-17

Contents

List of acronyms

AEAD	Authenticated encryption with associated data
AES	Advanced Encryption Standard
AXU	Almost XOR universal
CBC	Cipher block chaining
CCA	Chosen-ciphertext attack
CCM	Counter with CBC-MAC
CFB	Counter feedback
CMAC	Cipher-based message authentication code
CPA	Chosen-plaintext attack
CTR	Counter
ECB	Electronic codebook
FIPS	Federal Information Processing Standard
FPE	Format-preserving encryption
GCM	Galois/counter mode
IETF	Internet Engineering Task Force
IND	Indistinguishability
IV	Initialisation vector
LFSR	Linear feedback shift register
LSB	Least significant bits
MAC	Message authentication code
MEE	MAC-encode-encrypt
MSB	Most significant bits
NIST	National Institute of Standards and Technology
NESSIE	New European Schemes for Signatures, Integrity and Encryption
PPT	Probabilistic polynomial-time
PRF	Pseudorandom function
PRP	Pseudorandom permutation
SSL	Secure Socket Layer
TLS	Transport Layer Security
XOR	Exclusive-or
XTS	XEX tweakable block cipher with ciphertext stealing

1 Introduction

Discussion of block ciphers in the previous lecture would not be complete without discussion of the modes of operation as well.

According to NIST SP 800-175B [Bar20],

Definition 1: Mode of operation [Bar20]

An algorithm that uses a block cipher algorithm as a cryptographic primitive to provide a cryptographic service, such as confidentiality or authentication.

A mode of operation of a block cipher 👉

- Specifies how to construct an encryption scheme using a block cipher as a building block.
- Partitions a plaintext message into a series of blocks which are then encrypted one block at a time with a block cipher [PBO+03, Sec. 2.1].

The distinct advantage of separating block cipher design from the design of mode of operation is that we can design block ciphers and modes of operation independently [Sma16, Sec. 13.1]:

- Whereas the design goal for a block cipher is that it is a strong pseudorandom permutation (PRP), the design goal of a mode of operation is one of indistinguishability or non-malleability goals, such as IND-CCA (see Lecture 2).
- A designer of a mode of operation tries to prove mathematically that the mode satisfies the required security definition on the assumption that the block cipher is a strong PRP.

NIST's recommendations for block cipher models of operation can be found in their SP 800-38 series of publications:

1. **SP 800-38A** [Dwo01] specifies the 1️⃣ *cipher block chaining* (CBC), 2️⃣ *cipher feedback* (CFB), 3️⃣ *output feedback* (OFB) and 4️⃣ *counter* (CTR) modes, besides the electronic codebook (ECB) mode (see previous lecture).
   - These modes are also standardised in ISO/IEC 10116:2017 [ISO17].
   - These modes are discussed in Secs. 2–5.
2. **SP 800-38B** [Dwo05] specifies the *cipher-based message authentication code* (CMAC) mode, which achieves authentication through a block cipher.
   - CMAC evolved from CBC-MAC.
   - CMAC is not covered in this course.
3. **SP 800-38C** [Dwo04] specifies the *counter with CBC-MAC* (CCM) mode for achieving authenticated encryption.
   - CCM combines the CTR mode for encryption with CBC-MAC for authentication.
   - CCM is not covered in this course.
4. **SP 800-38D** [NIS07] specifies the *Galois/counter mode* (GCM), for AEAD, and its specialisation, GMAC, for authentication of unencrypted data.
   - GCM is not covered in this course.
   - GMAC is not covered in this course.
5. **SP 800-38E** [Dwo10a] approves the specification of the XTS-AES mode of the AES in IEEE Std 1619-2007, subject to one additional requirement.
   - XTS = XOR-encrypt-XOR tweakable block cipher with ciphertext stealing.

- IEEE Std 1619-2007 has been revised to IEEE Std 1619-2018 [IEE19].
- Protects confidentiality of data at rest in block-oriented storage devices, when CBC and CTR modes cannot.
- XTS-AES is discussed in Sec. 6.

6. **SP 800-38F** [Dwo12] specifies two deterministic authenticated-encryption modes of operation based on the AES for key wrapping, i.e., the protection of the confidentiality and integrity of cryptographic keys: 1️⃣ the AES *key wrap* mode, and 2️⃣ the AES *key wrap with padding* mode.
   - AES key wrap is also standardised in RFC 3394 [SH02] and implemented in OpenSSL.
   - AES key wrap with padding is also standardised in RFC 5649 [HD09]. This eliminates the requirement that the length of the key to be wrapped be a multiple of 64 bits.
   - In principle, these modes provide more security than regular authenticated encryption schemes at the expense of throughput [Dwo12, Sec. 3.1].
   - These modes are not covered in this course.
7. **SP 800-38G** [Dwo19] specifies two format-preserving encryption (FPE) modes of the AES algorithm, namely FF1 and FF3-1.
   - NIST does not have a definition for FPE, but "format-preserving" can be understood as preserving the *alphabet* of the plaintext in the ciphertext [BRRS09], e.g., a 16-decimal-digit credit card number encrypted to a 16-decimal-digit number.
   - 🤔 Why? Classic scenario: integrating encryption into an existing database does not require changing the type and size of the data field, e.g., the "credit card number" field remains a string of 16 decimal digits.
   - These modes are not covered in this course.

2 CBC

In the CBC mode, ciphertext blocks are chained as illustrated in Figure 1.

- The IV can be public and provided along with the ciphertext blocks, but must be randomised/unpredictable.
- CBC encryption:
$$C_i = \begin{cases} \mathrm{CIPH}_K(P_i \oplus \mathrm{IV}) & \text{if } i = 1, \\ \mathrm{CIPH}_K(P_i \oplus C_{i-1}) & \text{otherwise,} \end{cases}$$
where C_i and P_i denote the ith ciphertext block and plaintext block respectively.
- CBC decryption:
$$P_i = \begin{cases} \mathrm{CIPH}_K^{-1}(C_i) \oplus \mathrm{IV} & \text{if } i = 1, \\ \mathrm{CIPH}_K^{-1}(C_i) \oplus C_{i-1} & \text{otherwise.} \end{cases}$$

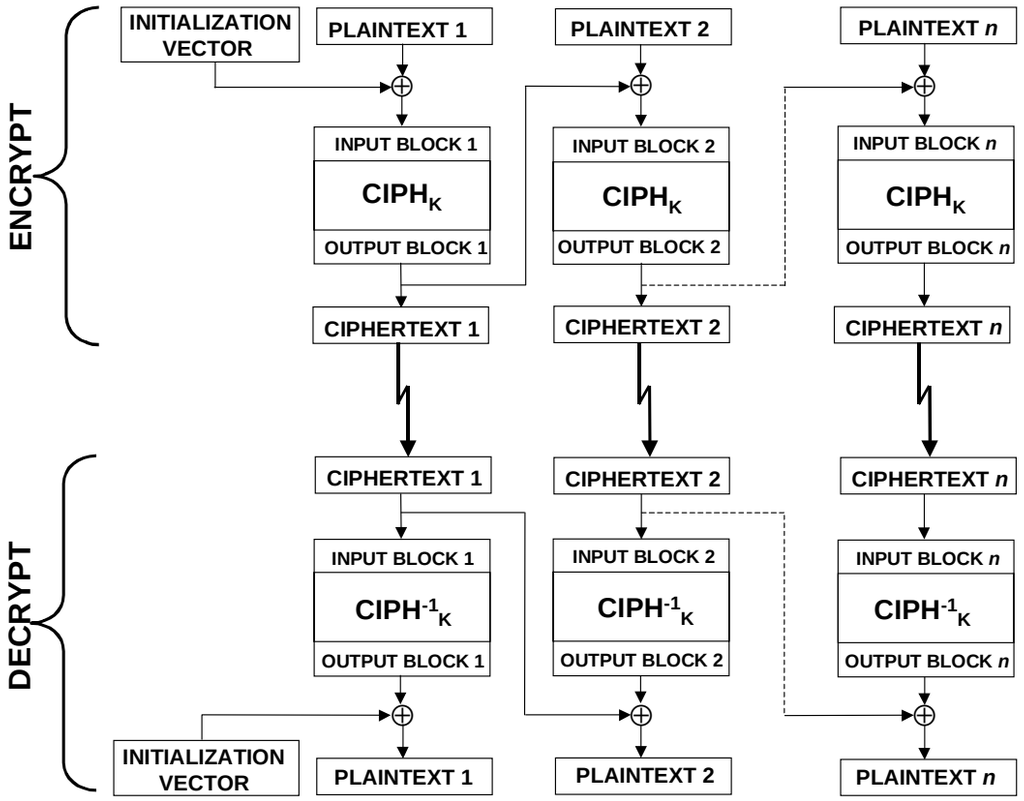


Figure 1: CBC mode [Dwo01, Figure 2].

Quiz 1

If CIPH in Figure 1 is the AES algorithm, how many bytes long should the IV be?

Security:

- Theorem 1 states, given the right conditions, the CBC mode is CPA-secure.

Theorem 1: [Sma16, Theorem 13.6], [KL21, Theorem 3.32]

Provided the underlying block cipher CIPH is a PRP, and the IV is random, the CBC mode satisfies IND-CPA.

In particular, suppose $\mathcal{A}$ denotes a PPT adversary against the n-bit CBC mode that makes at most T invocations of the block cipher through the encryption oracle, then there exists a PPT adversary $\mathcal{B}$ against CIPH such that

$$\mathrm{Adv}^{\text{ind-cpa}}_{\mathrm{CBC}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{prp}}_{\mathrm{CIPH}}(\mathcal{B}) + 3T^2/2^n.$$

To appreciate Theorem 1 numerically, set block length $n = 128$ bits and key length = 128 bits.

Assuming the AES behaves as a PRP to any PPT $\mathcal{B}$,

$$\mathrm{Adv}^{\mathrm{prp}}_{\mathrm{AES}}(\mathcal{B}) \leq 2^{-128},$$

and thus

$$\mathrm{Adv}^{\text{ind-cpa}}_{\mathrm{CBC}}(\mathcal{A}) \leq 2^{-128}(1 + 3T^2).$$

Bounding $\mathcal{A}$'s advantage at 2^{-64} requires $T \leq 2^{31}$, which is more than 2.14 billion.

Best practice: refresh encryption keys periodically.

- Some IND-CPA proofs [BR05, Theorem 5.8.1] model the underlying block cipher as a pseudorandom function (PRF) and provide a different advantage bound.
- Predictable IVs lead to attacks; see Example 1.

- Not CCA-secure [BR05, Sec. 5.10.2], and implementations must be guarded against timing attacks; see Example 2.
- Watch Dan Boneh's "Modes of Operation: Many Time Key (CBC)".

Example 1

TLS 1.0 and SSL 3.0 implemented a variant of the CBC mode called chained CBC [KL21, p. 91] or CBC-chain [BR05, Problem 33].

Figure 2 depicts the scenario where the first plaintext is encrypted into c_1, c_2, c_3, and the second plaintext is encrypted into c_4, c_5.

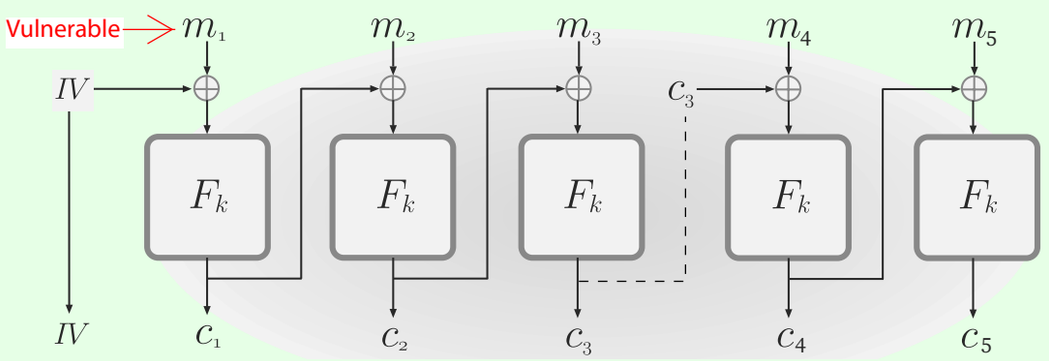


Figure 2: CBC mode using chained IVs [KL21, Figure 3.7].

IV unpredictability is destroyed when c_3 is used as the IV for m_4.

A sketch of a CPA:

1. Suppose the attacker $\mathcal{A}$ knows that m_1 is one of m_1^0 and m_1^1, e.g., "Yes" or "No" for criminal record.
2. $\mathcal{A}$ eavesdrops on IV, c_1, c_2, c_3.
3. Knowing c_3 will be used as the IV for m_4, $\mathcal{A}$ crafts $m_4 \leftarrow \text{IV} \oplus m_1^0 \oplus c_3$ and queries the encryption oracle with m_4.
4. The encryption oracle returns $c_4 \leftarrow F_k(c_3 \oplus \text{IV} \oplus m_1^0 \oplus c_3) = F_k(\text{IV} \oplus m_1^0)$.
5. If $c_1 == F_k(\text{IV} \oplus m_1^0)$, $\mathcal{A}$ knows c_1 is an encryption of m_1^0. Otherwise, $\mathcal{A}$ knows c_1 is an encryption of m_1^1.

The vulnerability above has been assigned the ID CVE-2011-3389 in the National Vulnerability Database and the associated attack is known as the BEAST attack.

Error characteristics [Sch15, pp. 195-196]:

- Lack of error tolerance due to chaining.
- Ciphertext error: One erroneous ciphertext bit affects the entire current plaintext block, and the corresponding bit of the next plaintext block.
  - » The phenomenon of *error extension* = Small ciphertext error causing large plaintext error.
  - » Nevertheless, *self-recovering* because blocks after the next are not affected.
- Synchronisation error: Lost ciphertext blocks need to be re-transmitted to decrypt the next ciphertext block.

Practical aspects:

- Plaintext that is not an integer multiple of blocks needs to be padded.

  Standard for padding: Cryptographic Message Syntax in RFC 5652 [Hou09] derived from PKCS #7 version 1.5 in RFC 2315 [Kal98].

  » Pad is appended, not prepended.

  » A one-byte pad is 0x01, a two-byte pad is 0x0202, etc.

  » Plaintext that is an integer multiple of blocks is padded with a whole block of 0x10 if the block length is 16 bytes.

  » Rationale: If the last byte of the last block happens to be 0x01, this might be mistaken as a 1-byte pad. A whole-block pad avoids the ambiguity.

  However, padding can be exploited for timing attacks; see Example 2.

- To avoid padding, use *ciphertext stealing*:

  » NIST SP 800-38A Addendum [Dwo10b] specifies three variants, namely CBC-CS1, CBC-CS2 and CBC-CS3, as illustrated in Figure 3.

  » CBC-CS2 originated in [Sch15] and is also described in RFC 2040 and RFC 3962.

  » IND-CPA security proof exists for alll three variants [RWZ12].

  » Not as popular [Aum21, p. 25] as the CTR mode for avoiding padding.

- Block chaining precludes parallelisation of encryption and decryption.

- Implementated in OpenSSL: `crypto/modes/cbc128.c`, `crypto/modes/cts128.c`.

- The Python cryptography library by default uses the AES in CBC mode.

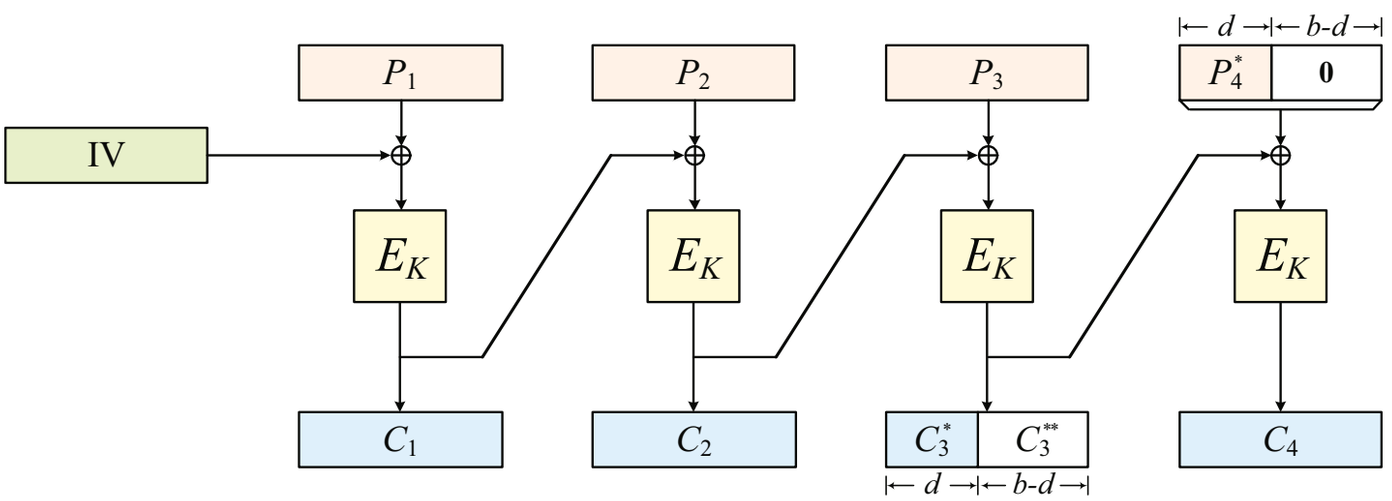


Figure 3: Ciphertext stealing for a multiple of b-bit blocks and one d-bit block [RWZ12, Fig. 1]. Ciphertext for
CBC-CS1: $C_1C_2C_3^*C_4$.
CBC-CS2: If $d = b$ then $C_1C_2C_3^*C_4$ else $C_1C_2C_4C_3^*$.
CBC-CS3: $C_1C_2C_4C_3^*$.

Example 2

TLS 1.1 and 1.2 supported the MEE-TLS-CBC construction in Figure 4, where MEE = MAC-encode-encrypt.

The MEE-TLS-CBC construction provides *length-hiding authenticated encryption* security provided 1️⃣ MAC tags are adequately long, 2️⃣ decryption does not reveal the cause of any failure [PRS11].
☝ The second condition is violated by most implementations for practical purposes.

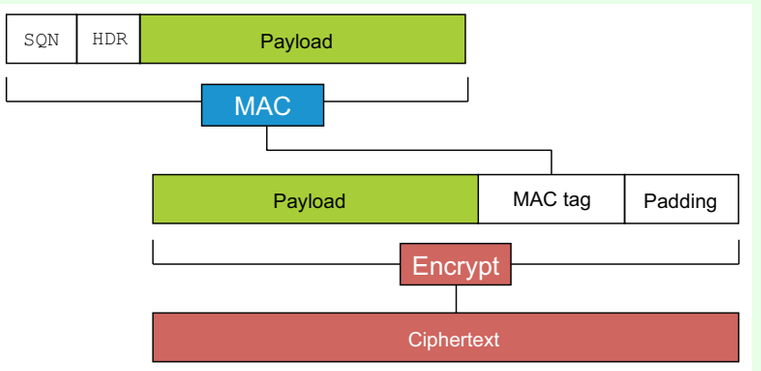


Figure 4: MEE-TLS-CBC performs the encryption part of MAC-encode-encrypt in the CBC mode, which requires padding, but padding is not authenticated [AP13b, Figure 1].

The Lucky Thirteen [AP13b] is a *distinguishing attack* that exploits the unauthenticated padding and the reporting of MAC verification errors by TLS implementations. 👈 "Thirteen" refers to the number of bytes in the message header [AP13a].

Sketch of distinguishing attack:

1. Attacker $\mathcal{A}$ prepares two 288-byte (18 blocks × 16 bytes/block) messages:
   - $m_0 \leftarrow$ 32 arbitrary bytes (2 blocks) followed by 256 copies of 0xFF.
   - $m_1 \leftarrow$ 287 arbitrary bytes (almost 18 blocks) followed by 0x00.
2. $\mathcal{A}$ sends m_0 and m_1 to the challenger.
3. Challenger returns $\mathrm{HDR} \| \mathrm{CBC}_k(m_b \| \mathrm{tag} \| \mathrm{pad})$, where b is either 0 or 1.
4. Since m_b is an integer number of blocks, $\mathcal{A}$ knows where to truncate the ciphertext to get the ciphertext blocks for m_b. 👈 Denote these ciphertext blocks by c_b.
5. $\mathcal{A}$ tricks the targeted TLS implementation into decrypting $\mathrm{HDR} \| c_b$.
6. Consider two cases:

Case 1: c_b is an encryption of m_0.
- Decrypting c_b reveals 256 copies of 0xFF at the trailing end, which the decryptor assumes to be a pad.
- Decryptor removes the 256-byte pad and interprets the remaining 32 bytes as a 12-byte message and a 20-byte MAC tag.
- Message authentication fails after 4 evaluations of the hash function.

Case 2: c_b is an encryption of m_1.
- Decrypting c_b reveals a single 0x00 the trailing end, which the decryptor assumes to be a pad.
- Decryptor removes the 1-byte pad and interprets the remaining 287 bytes as a 267-byte message and a 20-byte MAC tag.
- Message authentication fails after at least 8 evaluations of the hash function.

7. For either case, an error message is returned to $\mathcal{A}$, and the time difference between the two cases was in the order of μs on a typical processor (in 2013). 👈 This timing difference was enough to launch a timing attack; see Figure 5.

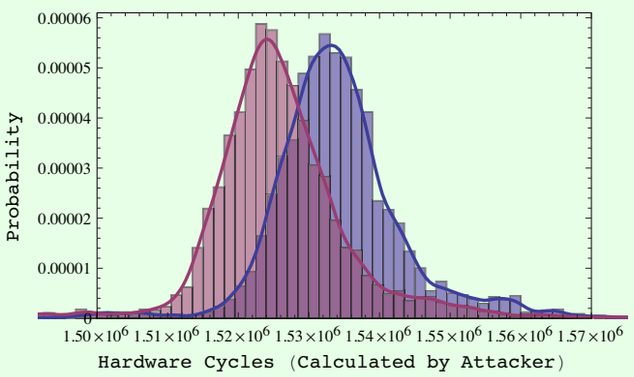


Figure 5: Distinguishable distributions of timing values for m_0 and m_1 [AP13b, Figure 2].

Following the disclosure of the attack, countermeasures including equalising running times and introducing random waiting periods were implemented in OpenSSL; see `ssl/record/tls_pad.c`.

Several years later, Lucky Thirteen was upgraded to Lucky Microseconds [AP16], defeating the countermeasures implemented in Amazon's implementation of TLS called s2n.

Starting with version 1.3, TLS no longer supports CBC [Res18].

⚠️ Attention: Best practice

Current best practice is replacing CBC-based encryption with authenticated encryption.

3 CFB

The CFB mode in Figure 6 enables a block cipher to operate as a self-synchronising stream cipher.

- The IV can be public and provided along with the ciphertext blocks, but must be randomised/unpredictable.
- Parameterised by s, where $1 \leq s \leq b$ and b is the block size of the block cipher. $s = 1$ gives us a stream cipher although an inefficient one.
- s-bit CFB encryption:

$$C_i^{\#} = P_i^{\#} \oplus \mathrm{MSB}_s\left(\mathrm{CIPH}_K(I_i)\right), \qquad I_i = \begin{cases} \mathrm{IV} & \text{if } i = 1, \\ \mathrm{LSB}_{b-s}(I_{i-1}) \| C_{i-1}^{\#} & \text{otherwise,} \end{cases}$$

  where $C_i^{\#}$ is the ith s-bit ciphertext segment, $P_i^{\#}$ is the ith s-bit plaintext segment, MSB$(\cdot)$ = most significant bits of $\cdot$, LSB$(\cdot)$ = least significant bits of $\cdot$.
  - » CIPH is used for keystream generation.
  - » The plaintext is XORed with the keystream and never fed to CIPH.
  - » If the latest plaintext block is less than s bits long, the resultant ciphertext is less than s bits long. No padding is required.
- s-bit CFB decryption:

$$P_i^{\#} = C_i^{\#} \oplus \mathrm{MSB}_s\left(\mathrm{CIPH}_K(I_i)\right), \qquad I_i = \begin{cases} \mathrm{IV} & \text{if } i = 1, \\ \mathrm{LSB}_{b-s}(I_{i-1}) \| C_{i-1}^{\#} & \text{otherwise.} \end{cases}$$

» Decryption uses CIPH ☝ rather than CIPH^{-1}.

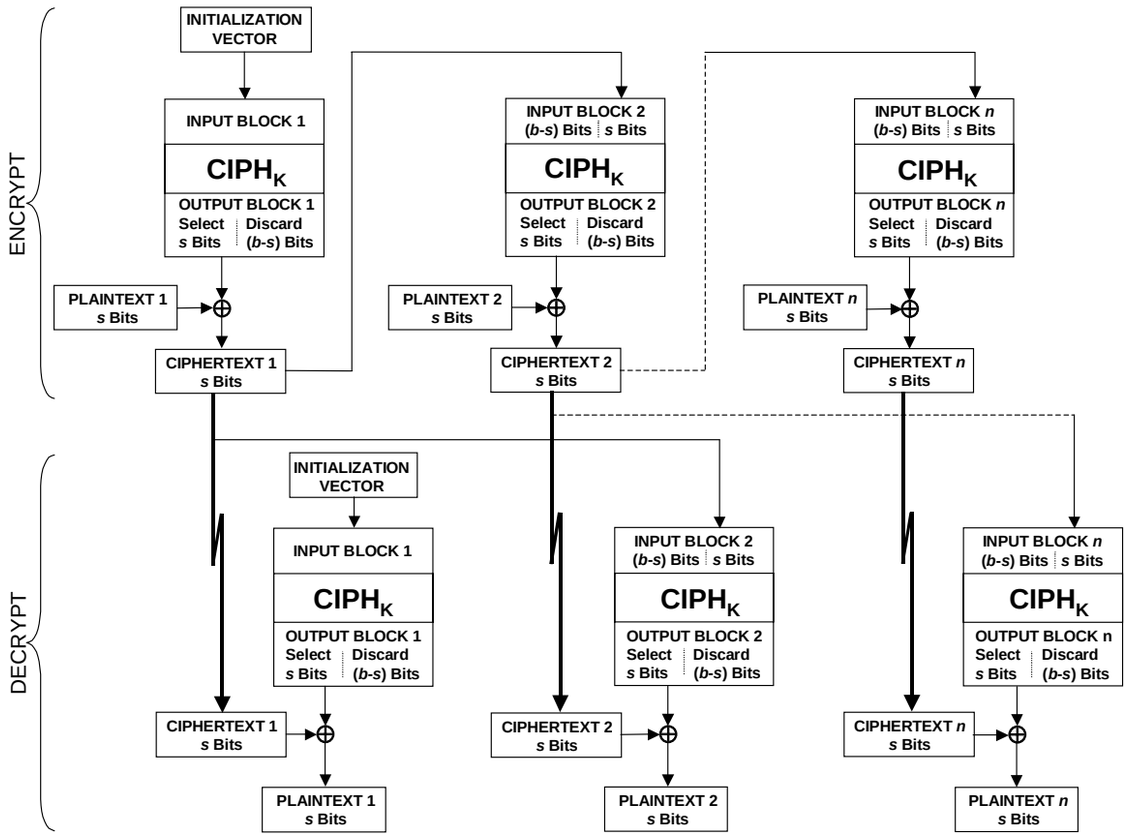


Figure 6: CFB mode [Dwo01, Figure 3].

Security:

- Believed to provide IND-CPA provided the underlying block cipher is indisinguishable from a random permutation; see [vJ11, p. 793] and [Mau91].
- Not CCA-secure [Sma16, Table 13.2].
- Best practice is to refresh the key after $2^{n/2-10}$ n-bit blocks.

Error characteristics [Sch15, p. 201]:

- Ciphertext error: One errorneous ciphertext bit affects the corresponding bit in the current plaintext block and the entire next plaintext block.
  Like CBC, suffers from error extension but is also self-recovering.
- Synchronisation error: Lost ciphertext blocks need to re-transmitted to decrypt the next ciphertext block.

Practical aspects:

- Encryption must be serial, while decryption can be parallelised since decryption of the current ciphertext block does not depend on the decryption of any preceding ciphertext block [vJ11, p. 791].
- Rarely used because it offers little advantage over the CTR mode (see Sec. 5).
- Implemented in OpenSSL: `crypto/modes/cfb128.c`.

4 OFB

The OFB mode in Figure 7 enables a block cipher to operate as a synchronous stream cipher.

- The IV can be public and provided along with the ciphertext blocks, but must be randomised/unpredictable [Sma16, Theorems 13.8-13.9].

> ⚠️ Attention: Nonce-based IV 🤦
>
> NIST's recommendation [Dwo01, Appendix C] that the IV can be a nonce – unique for each execution of the mode under a given key but not necessarily unpredictable — is not strong enough.

- OFB encryption:

$$C_i = \begin{cases} P_i \oplus \mathrm{CIPH}_K^{(i)}(\mathrm{IV}) & \text{if } i < n, \\ P_i \oplus \mathrm{MSB}_u\left(\mathrm{CIPH}_K^{(i)}(\mathrm{IV})\right) & \text{otherwise,} \end{cases} \tag{1}$$

  where $\mathrm{CIPH}_K^{(i)}(\mathrm{IV})$ means applying the block cipher to IV i number of times, u is the number of bits in the final plaintext block. 👈 The OFB mode does *not* require an integer multiple of blocks and hence padding.
- OFB decryption:

$$P_i = \begin{cases} C_i \oplus \mathrm{CIPH}_K^{(i)}(\mathrm{IV}) & \text{if } i < n, \\ C_i \oplus \mathrm{MSB}_u\left(\mathrm{CIPH}_K^{(i)}(\mathrm{IV})\right) & \text{otherwise.} \end{cases} \tag{2}$$

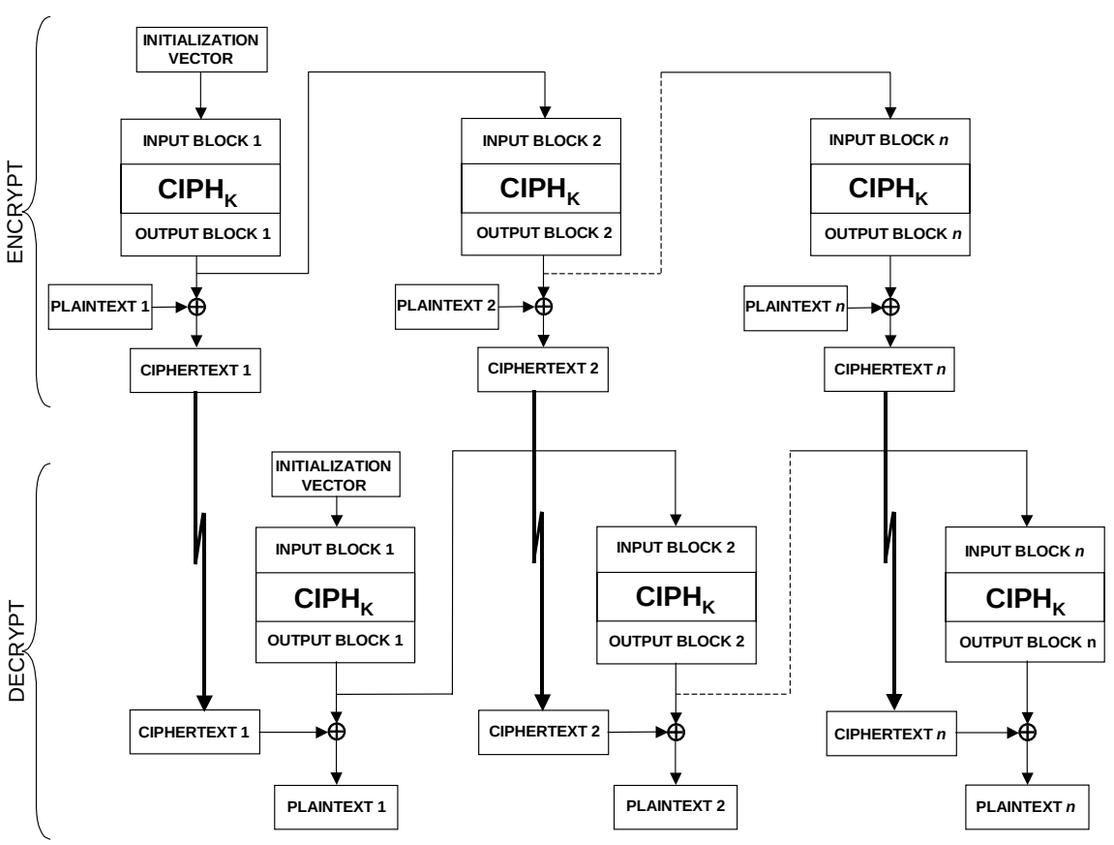


Figure 7: OFB mode [Dwo01, Figure 4]. The last plaintext block can be less than one-block long, say u-bit long.

Security:

- Believed to provide IND-CPA provided the underlying block cipher is indistinguishable from a random permutation [vJ11, p. 791].
- Not CCA-secure [Sma16, Table 13.2].
- Best practice is to refresh the key after $2^{n/2-10}$ n-bit blocks.

Error characteristics [Sch15, p. 204]:

- Ciphertext error: One errorneous ciphertext bit affects the corresponding bit in the current plaintext block.
  Therefore, unlike CBC and CFB, OFB does *not* suffer from error extension 👍.
- Synchronisation error: Lost ciphertext blocks do not need to be re-transmitted but the encrypted IV in Eqs. (1)-(2) must be resynchronised to decrypt subsequent ciphertext blocks, e.g., by trying subsequent encrypted values of the IV until decryption produces a valid plaintext; see Example 3.

Example 3

For the ith ciphertext block, Figure 7 shows the encrypted IV is $\mathrm{IV}_i = \mathrm{CIPH}_K^{(i)}(\mathrm{IV})$.

Suppose the 2nd and 3rd ciphertext blocks are lost.

When the 4th ciphertext block C_4 arrives, decrypting C_4 with IV_2 or IV_3 produces an invalid plaintext, whereas IV_4 produces a valid plaintext. 👈 This achieves resynchronisation of the encrypted IV.

Practical aspects:

- Both encryption and decryption can be parallelised since the block cipher is *never* applied to the plaintext or ciphertext directly.
  🔎 The encrypted IVs must be generated in sequence, but they can be generated before encryption/decryption begins.
- Rarely used because it offers little advantage over the CTR mode (see Sec. 5).
- Implemented in OpenSSL: `crypto/modes/ofb128.c`.

5 CTR

The CTR mode in Figure 8 enables a block cipher to operate as a synchronous stream cipher.

- Instead of an IV, a typically increasing counter sequence is used. 👈 This sequence can be public and predictable, but must be *unique* for each execution of the mode under a given key [Dwo01, Appendix B].
  Example 4 provides two examples of how counters can be generated.
- CTR encryption:

$$C_i = \begin{cases} P_i \oplus \mathrm{CIPH}_K(T_i) & \text{if } i < n, \\ P_i \oplus \mathrm{MSB}_u(\mathrm{CIPH}_K(T_i)) & \text{otherwise,} \end{cases} \tag{3}$$

  where T_i is the ith counter, and u is the number of bits in the final plaintext block. Thus, like OFB, CTR does *not* require an integer multiple of blocks and hence padding.

- CTR decryption:

$$P_i = \begin{cases} C_i \oplus \mathrm{CIPH}_K(T_i) & \text{if } i < n, \\ C_i \oplus \mathrm{MSB}_u(\mathrm{CIPH}_K(T_i)) & \text{otherwise.} \end{cases} \tag{4}$$

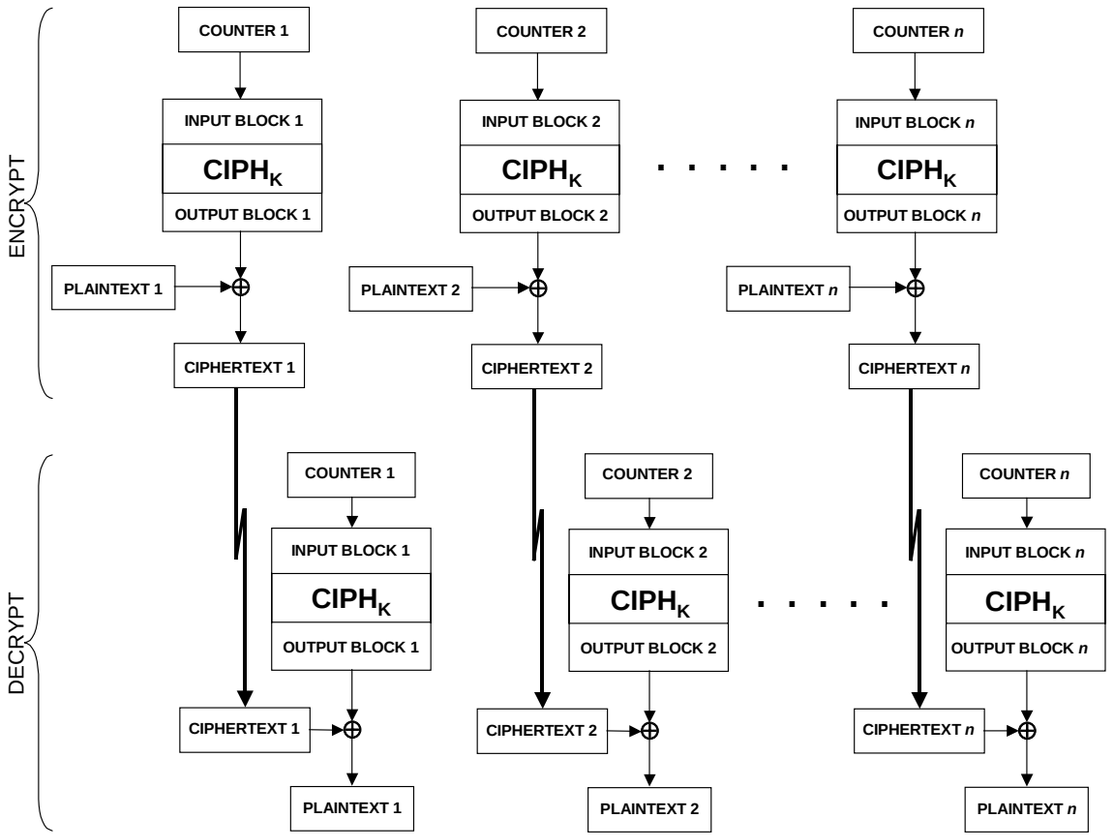


Figure 8: CTR mode [Dwo01, Figure 5].

Example 4

Two examples of counter generation methods can be found in [Dwo01, Sec. B.2]:

1. Generate a random string as wide as a block and reserve m number of bits for counting, e.g.,

1	1	0	1	**0**	0	**1**	**0**
1	1	0	1	**0**	0	**1**	**1**
1	1	0	1	**1**	0	**0**	**0**
1	1	0	1	**1**	0	**0**	**1**
1	1	0	1	**1**	0	**1**	**0**
1	1	0	1	**1**	0	**1**	**1**
1	1	0	1	**0**	0	**0**	**0**
1	1	0	1	**0**	0	**0**	**1**

👈 Sample initial counter.
$m = 3$.

2. Partition a counter block into a nonce segment and an m-bit counter segment, e.g.,

Nonce					Counter		
1	1	0	1	0	**0**	**0**	**1**
1	1	0	1	0	**0**	**1**	**0**
1	1	0	1	0	**0**	**1**	**1**
1	1	0	1	0	**1**	**0**	**0**
1	1	0	1	0	**1**	**0**	**1**
1	1	0	1	0	**1**	**1**	**0**
1	1	0	1	0	**1**	**1**	**1**

👈 Sample initial counter.
$m = 3$.
Instead of an incrementing function, an LFSR can also be used, but the counter bits cannot be all zero in the latter case.

Security:

- Theorem 2 states, given the right conditions, the CTR mode is CPA-secure.

Theorem 2: [Sma16, Theorem 13.11]

Provided the underlying block cipher CIPH is a PRP, and each counter is unique, the CTR mode satisfies IND-CPA.

In particular, suppose $\mathcal{A}$ denotes a PPT adversary against the n-bit CTR mode that makes at most T invocations of the block cipher through the encryption oracle, then there exists a PPT adversary $\mathcal{B}$ against CIPH such that

$$\mathrm{Adv}^{\text{ind-cpa}}_{\mathrm{CTR}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{prp}}_{\mathrm{CIPH}}(\mathcal{B}) + T^2/2^n.$$

- Some IND-CPA proofs such as [BR05, Theorem 5.7.1] and [KL21, Theorem 3.33] model the underlying block cipher as a PRF and provide a different advantage bound.
- Not CCA-secure [BR05, Sec. 5.10.1].
- Best practice is to refresh the key after $2^{n/2-10}$ n-bit blocks [vJ11, p. 791].
- Watch Dan Boneh's ▶ "Modes of Operation: Many Time Key (CTR)".

Error characteristics same as those of the OFB mode, except the resynchronisation of the counter for CTR mode is more efficient than the resynchronisation of the encrypted IV for the OFB mode.

Practical aspects:

- Both encryption and decryption can be parallelised since the block cipher is *never* applied to the plaintext or ciphertext directly.
  👍 Counters are *not* bound by serial generation, unlike the chain of encrypted IVs in the OFB mode.
- Implemented in OpenSSL: `crypto/modes/ctr128.c`.

Quiz 2

Referring to the OpenSSL code for the CTR mode summarised in Listing 1,

1. In the function `ctr128_inc`, what is the variable `c` mainly used for?
2. In the function `CRYPTO_ctr128_encrypt`, what does the argument `block` point to?
3. What is the effect of the check for `n==0`?

Listing 1: CRYPTO_ctr128_encrypt.

```
static void ctr128_inc(unsigned char *counter)
{
  u32 n = 16, c = 1;

  do {
    --n;
    c += counter[n];
    counter[n] = (u8)c;
    c >>= 8;
  } while (n);
}

/*
 * The input encrypted as though 128bit counter mode is being used.  The
 * extra state information to record how much of the 128bit block we have
 * used is contained in *num, and the encrypted counter is kept in
 * ecount_buf.  Both *num and ecount_buf must be initialised with zeros
 * before the first call to CRYPTO_ctr128_encrypt(). This algorithm assumes
 * that the counter is in the x lower bits of the IV (ivec), and that the
 * application has full control over overflow and the rest of the IV.  This
 * implementation takes NO responsibility for checking that the counter
 * doesn't overflow into the rest of the IV when incremented.
 */
void CRYPTO_ctr128_encrypt(const unsigned char *in, unsigned char *out,
                           size_t len, const void *key,
                           unsigned char ivec[16],
                           unsigned char ecount_buf[16], unsigned int *num,
                           block128_f block)
{
  unsigned int n; size_t l = 0; n = *num;

  while (l < len) {
    if (n == 0) {
      (*block) (ivec, ecount_buf, key);
      ctr128_inc(ivec);
    }
    out[l] = in[l] ^ ecount_buf[n];
    ++l;
    n = (n + 1) % 16;
  }

  *num = n;
}
```

6 XTS-AES

As introduced in Sec. 1, XTS-AES is specifically designed for protecting the confidentiality of *data at rest* [Dwo10a, BGH+12].

- Designed for data stored on hard disks where there is no additional space for integrity or authentication tags.

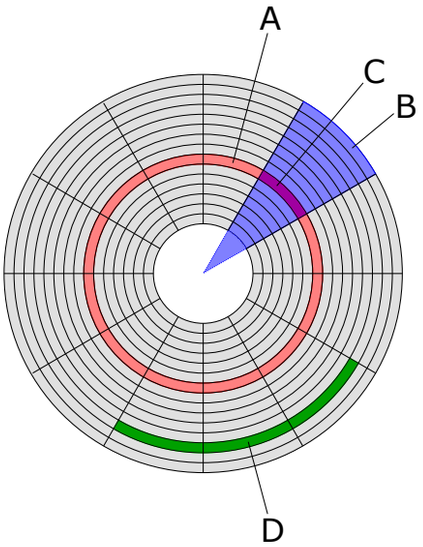


Figure 9: A = track, B = geometrical sector, C = disk sector, D = cluster. Image from Wikipedia.

Detail: Track, sector, block

As shown in Figure 9, a hard disk has *tracks*.

The smallest accessible subdivision of a track is a *sector*.

A sector is further divided into configurable *logical blocks*.

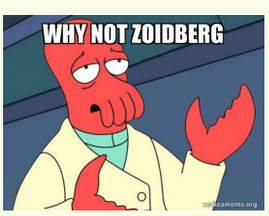
On a Linux system, get block size by running command:

```
sudo blockdev --getbsz -v /dev/sda
```

Block size can be configured to be 16 bytes, the block size of the AES.

- Not designed for data in transit.
- No authentication mechanism, but provides more protection than the other approved confidentiality-only modes against unauthorised manipulation of the encrypted data.
- Where storage blocks can have variable sizes, such as with tape drives, an AEAD scheme should be used instead.

Detail: Why not CBC or CTR? [IEE19, Sec. D.2]



🤔 Why not CBC?

- Using CBC, an adversary can flip any bit of the plaintext by flipping the corresponding ciphertext bit of the previous block, with the side-effect of "randomizing" the previous block. See Figure 1.
- Using CBC, an adversary with read/write access to the encrypted disk can copy a ciphertext sector from one position to another, and an application reading the sector off the new location will still get the same plaintext sector (except perhaps the first 16 bytes).

🤔 Why not CTR?

- Using CTR without authentication tags is trivially malleable, and an adversary with write access to the encrypted media can flip any bit of the plaintext simply by flipping the corresponding ciphertext bit. See Figure 8.

To see how XTS-AES works, we first explain how a tweakable block cipher works:

- A *tweakable block cipher* is a block cipher that encrypts a plaintext using a key *and* a tweak; and a *tweak* plays the role of an IV or nonce [LRW02], i.e., a fixed-length string that makes a block cipher less deterministic.

Liskov et al. [LRW02] used the name "tweak" to differentiate it from IV and nonce:

» Unlike an IV, a tweak does not have to be random [Mar10].

» Unlike a nonce, reuse of a tweak is not fatal.

Formally, a tweakable block cipher is a map $\widetilde{\mathsf{Enc}} : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$, where each $\widetilde{\mathsf{Enc}}_K^T(\cdot) = \mathsf{Enc}_K(T, \cdot)$ is a permutation, $\mathcal{T}$ is the set of tweaks, and n is the block length [Rog04].

🤔 Why tweakable block ciphers? Modes of operation can be easier to design and prove for tweakable block ciphers than for conventional block ciphers [LRW02].

- Liskov et al. [LRW02] proposed a tweakable block cipher construction called the *LRW construction*:

$$\widetilde{\mathsf{Enc}}_K(T, M) = \mathsf{Enc}_K(M \oplus h(T)) \oplus h(T), \tag{5}$$

where K is a key, T is a tweak, M is a plaintext, h is a hash function.

The LRW construction is CCA-secure [LRW02, Theorem 2] if h is from an ϵ-almost 2-xor-universal (ϵ-AXU$_2$ for short) family of hash functions.

A hash function family $\mathcal{H}$ is ϵ-AXU$_2$ if $\Pr\{h(x) \oplus h(y) = z\} \leq \epsilon$, $\forall x, y, z$, and for any h chosen uniformly at random (through a key) from $\mathcal{H}$.

Detail: AXU

AXU is a historical security definition traceable back to [Kra94] proposed for the purpose of hashing and authentication.

Suppose for message M, the authentication tag is $t = h(M) \oplus r$, where r is a nonce. An adversary that sees M and t but not h or r succeeds in breaking the authentication if it finds M' ($\neq M$) and $t' = h(M') \oplus r$.

How? One way: if the adversary can find $z = h(M) \oplus h(M')$, then it can forge the authentication tag

$$t' = t \oplus z = h(M) \oplus r \oplus h(M) \oplus h(M') = h(M') \oplus r.$$

If h is from an AXU hash function family, then the probability of finding z is negligible.

- Rogaway [Rog04] proposed the XOR-encrypt-XOR (XEX) tweakable block cipher construction:

$$\widetilde{\mathsf{Enc}}_K(T, M) = \mathsf{Enc}_K(M \oplus T) \oplus T, \tag{6}$$

where $T = \mathsf{Enc}_K(N)\alpha_1^{j_1} \cdots \alpha_j^{j_k}$, N is an n-bit nonce, $\alpha_1, \ldots, \alpha_k$ are primitive elements of the Galois field GF(2^n), $j_1, \ldots, j_k$ are integers. 👈 There are $k+1$ tweaks here, as opposed to one tweak in the LRW construction in Eq. (5).

The XEX construction is also CCA-secure.

XTS-AES is a specialisation of the XEX construction that

- **Uses two keys**, namely K_1 and K_2 in Eqs. (7)–(8).
  This is not for boosting security but rather to appeal to the perception of commercial users that two keys provide more security than one.

Quiz 3

According to [IEE19], which two key lengths are supported? These two key lengths determine the two variants of the AES that are supported – which are these variants?

- **Limits the number of tweaks to two**:
  1. the 128-bit Data Unit Sequence Number (i.e., sector number) denoted by i in (7)–(8); and
  2. the Block Sequence Number (i.e., block number) denoted by j in (7)–(8).

  i is nonnegative and is assigned consecutively, starting from an arbitrary nonnegative integer [IEE19, Sec. 5.1].

  j starts at zero and is at most 20-bit long because the maximum number of blocks with a data unit is 2^{20} [IEE19, Sec. 5.1].

Quiz 4

When encrypting a tweak value using AES, the tweak is first converted into a *little-endian* byte array [IEE19, Sec. 5.1].

If a tweak value is 0xCCBBAA, how should the byte array be represented in Python?

- **Uses GF(2^{128})** with irreducible polynomial: $x^{128} + x^7 + x^2 + x + 1$.

  α in (7)–(8) is 2 or equivalently polynomial x in GF(2^{128}).

  In Listing 2, look for the code commented with `alpha^j` for the Galois-field multiplication.
- **Uses ciphertext stealing** to cater for the situations where a sector size is not an integer multiple of the AES block size, because some hard disk implementations include an 8-byte non-cryptographic checksum at the end of a sector, which like normal content also needs to be encrypted [BGH+12, Sec. 2].

  See Figures 12–13.

XTS-AES encryption and decryption:

To encrypt plaintext block P in data unit i, block j into ciphertext block C:

$$C = \mathsf{Enc}(K_1, P \oplus T) \oplus T, \text{ where } T = \mathsf{Enc}(K_2, i) \otimes \alpha^j. \tag{7}$$

See Figure 10.

To decrypt ciphertext block C in data unit i, block j into plaintext block P:

$$P = \mathsf{Dec}(K_1, C \oplus T) \oplus T, \text{ where } T = \mathsf{Enc}(K_2, i) \otimes \alpha^j. \tag{8}$$

See Figure 11.

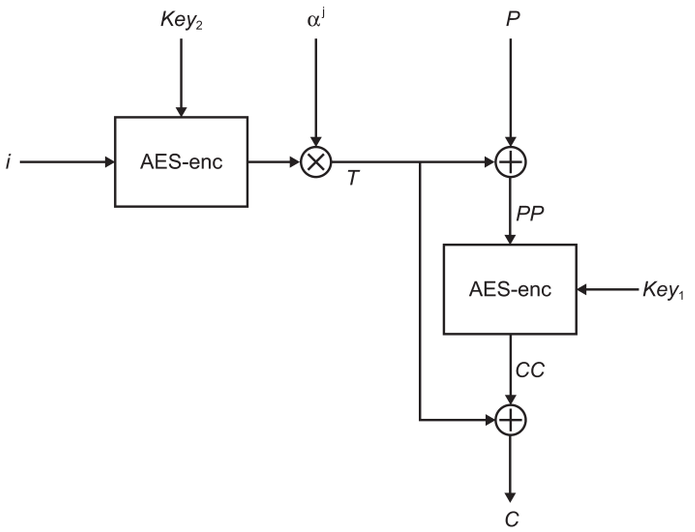


Figure 10: Encrypting a plaintext block P using XTS-AES [IEE19, Figure 1].

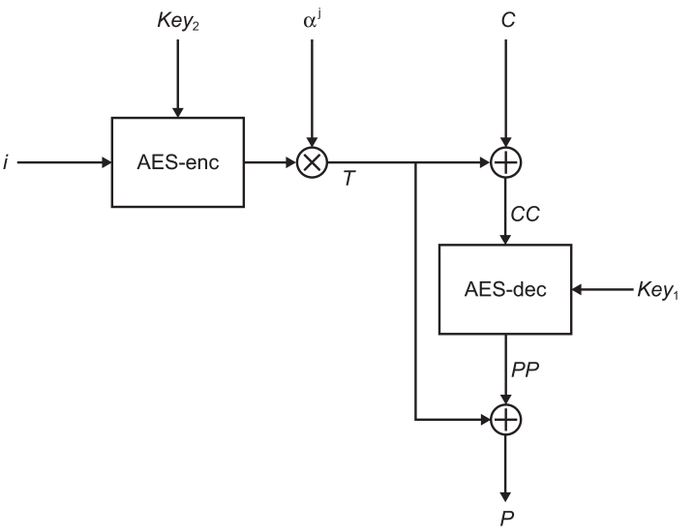


Figure 11: Decrypting a ciphertext block C using XTS-AES [IEE19, Figure 3].

Listing 2: OpenSSL implementation of XEX in `crypto/modes/xts128.c`. This is a sanitised, little-endian, encryption-only version of function `CRYPTO_xts128_encrypt`. Ciphertext stealing is implemented in `crypto/modes/cts128.c`.

```
int CRYPTO_xts128_encrypt(const XTS128_CONTEXT *ctx,
                          const unsigned char iv[16],
                          const unsigned char *inp, unsigned char *out,
                          size_t len, int enc)
{
  DECLARE_IS_ENDIAN;
  union {
    u64 u[2];
    u32 d[4];
    u8 c[16];
  } tweak, scratch;
  unsigned int i;

  if (len < 16) return -1;

  memcpy(tweak.c, iv, 16);
  (*ctx→block2) (tweak.c, tweak.c, ctx→key2);

  if (!enc && (len % 16)) len -= 16;

  while (len ⩾ 16) {
    scratch.u[0] = ((u64_a1 *)inp)[0] ^ tweak.u[0];
    scratch.u[1] = ((u64_a1 *)inp)[1] ^ tweak.u[1];
    (*ctx→block1) (scratch.c, scratch.c, ctx→key1);
    ((u64_a1 *)out)[0] = scratch.u[0] ^≔ tweak.u[0];
    ((u64_a1 *)out)[1] = scratch.u[1] ^≔ tweak.u[1];

    inp += 16; out += 16; len -= 16;

    if (len == 0) return 0;

    unsigned int carry, res;

    /* alpha^j */
    res = 0x87 & (((int)tweak.d[3]) >> 31); /* 0x87 = 135 = x^7+x^2+x+1 */
    carry = (unsigned int)(tweak.u[0] >> 63);
    tweak.u[0] = (tweak.u[0] << 1) ^ res;
    tweak.u[1] = (tweak.u[1] << 1) | carry;
  }
```

```
  if (enc) {
    for (i = 0; i < len; ++i) {
      u8 c = inp[i];
      out[i] = scratch.c[i];
      scratch.c[i] = c;
    }
    scratch.u[0] ^= tweak.u[0];
    scratch.u[1] ^= tweak.u[1];
    (*ctx->block1) (scratch.c, scratch.c, ctx->key1);
    scratch.u[0] ^= tweak.u[0];
    scratch.u[1] ^= tweak.u[1];
    memcpy(out - 16, scratch.c, 16);
  } else {
    /* decryption code omitted */
  }

  return 0;
}
```

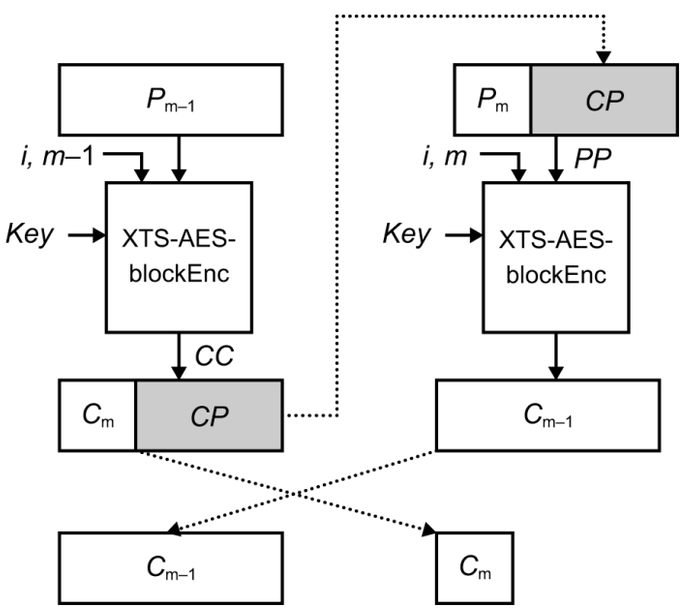


Figure 12: Ciphertext stealing during encryption [IEE19, Figure 2].

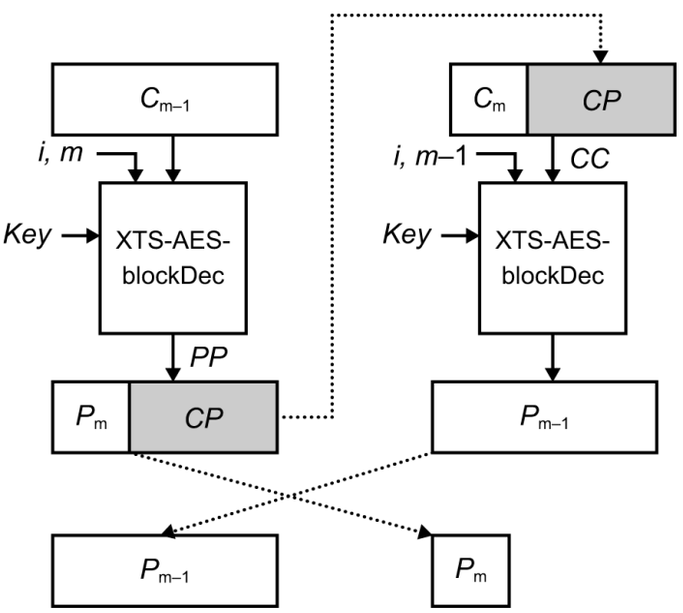


Figure 13: Ciphertext stealing during decryption [IEE19, Figure 4].

7 References

[AP16] M. R. Albrecht and K. G. Paterson, Lucky Microseconds: A Timing Attack on Amazon's s2n Implementation of TLS, in *Advances in Cryptology – EUROCRYPT 2016* (M. Fischlin and J.-S. Coron, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2016, pp. 622–643.

[AP13a] N. AlFardan and K. Paterson, Lucky Thirteen: Breaking the TLS and DTLS Record Protocols, web site, February 2013. Available at http://www.isg.rhul.ac.uk/tls/Lucky13.html.

[AP13b] N. J. AlFardan and K. G. Paterson, Lucky thirteen: Breaking the TLS and DTLS record protocols, in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 526–540. https://doi.org/10.1109/SP.2013.42.

[Aum21] J.-P. Aumasson, *Crypto Dictionary: 500 Tasty Tidbits for the Curious Cryptographer*, No Starch Press, 2021.

[BGH+12] M. V. Ball, C. Guyot, J. P. Hughes, L. Martin, and L. C. Noll, The XTS-AES disk encryption algorithm and the security of ciphertext stealing, *Cryptologia* **36** no. 1 (2012), 70–79. https://doi.org/10.1080/01611194.2012.635115.

[Bar20] E. Barker, *Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms*, Special Publication 800-175B Revision 1, NIST, 2020. https://doi.org/10.6028/NIST.SP.800-175Br1.

[BRRS09] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers, Format-preserving encryption, in *Selected Areas in Cryptography* (M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 295–312.

[BR05] M. Bellare and P. Rogaway, Introduction to modern cryptography, 2005. Available at https://cseweb.ucsd.edu/~mihir/papers/br-book.pdf.

[Dwo01] M. Dworkin, Recommendation for block cipher modes of operation: Methods and techniques, NIST Special Publication 800-38A, 2001. Available at https://csrc.nist.gov/publications/detail/sp/800-38a/final.

[Dwo04] M. Dworkin, Recommendation for block cipher modes of operation: The CCM mode for authentication and confidentiality, NIST Special Publication 800-38C, May 2004, updated in 2007. Available at https://csrc.nist.gov/publications/detail/sp/800-38c/final.

[Dwo05] M. Dworkin, Recommendation for block cipher modes of operation: The CMAC mode for authentication, NIST Special Publication 800-38B, May 2005. https://doi.org/10.6028/NIST.SP.800-38B.

[Dwo10a] M. Dworkin, Recommendation for block cipher modes of operation: The XTS-AES mode for confidentiality on storage devices, NIST Special Publication 800-38E, 2010. Available at https://csrc.nist.gov/publications/detail/sp/800-38e/final.

[Dwo10b] M. Dworkin, Recommendation for block cipher modes of operation: Three variants of ciphertext stealing for CBC mode, Addendum to NIST Special Publication 800-38A, October 2010, under revision by NIST as of 13 Sep 2022.

[Dwo12] M. Dworkin, Recommendation for block cipher modes of operation: Methods for key wrapping, NIST Special Publication 800-38F, 2012. Available at https://csrc.nist.gov/publications/detail/sp/800-38f/final.

[Dwo19] M. Dworkin, Recommendation for block cipher modes of operation: Methods for format-preserving encryption, Draft NIST Special Publication 800-38G Revision 1, February 2019. https://doi.org/https://doi.org/10.6028/NIST.SP.800-38Gr1-draft.

[Hou09] R. Housley, Cryptographic Message Syntax (CMS), IETF RFC 5652, 2009, derived from PKCS #7 version 1.5 in RFC 2315. Available at https://www.rfc-editor.org/rfc/rfc5652.

[HD09] R. Housley and M. Dworkin, Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm, IETF RFC 5649, August 2009.

[IEE19] IEEE, IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices, IEEE Std 1619-2018 (Revision of IEEE Std 1619-2007), 2019. https://doi.org/10.1109/IEEESTD.2019.8637988.

[ISO17] ISO/IEC, Information technology — security techniques — modes of operation for an n-bit block cipher, ISO/IEC 10116:2017, 2017. Available at https://www.iso.org/standard/64575.html.

[Kal98] B. Kaliski, PKCS #7: Cryptographic Message Syntax Version 1.5, IETF RFC 2315, 1998. Available at https://www.ietf.org/rfc/rfc2315.txt.

[KL21] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 3rd ed., CRC Press, 2021. Available at https://ebookcentral.proquest.com/lib/unisa/detail.action?docID=6425020.

[Kra94] H. Krawczyk, LFSR-based hashing and authentication, in *Advances in Cryptology — CRYPTO '94* (Y. G. Desmedt, ed.), Springer Berlin Heidelberg, Berlin, Heidelberg, 1994, pp. 129–139.

[LRW02] M. Liskov, R. L. Rivest, and D. Wagner, Tweakable block ciphers, in *Advances in Cryptology — CRYPTO 2002* (M. Yung, ed.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 31–46.

[Mar10] L. Martin, XTS: A mode of AES for encrypting hard disks, *IEEE Security & Privacy* **8** no. 3 (2010), 68–69. https://doi.org/10.1109/MSP.2010.111.

[Mau91] U. M. Maurer, New approaches to the design of self-synchronizing stream ciphers, in *Advances in Cryptology — EUROCRYPT '91* (D. W. Davies, ed.), Springer Berlin Heidelberg, Berlin, Heidelberg, 1991, pp. 458–471.

[NIS07] NIST, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST Special Publication 800-38D, 2007, under review as of 6 Aug 2021. Available at https://csrc.nist.gov/publications/detail/sp/800-38d/final.

[PRS11] K. G. Paterson, T. Ristenpart, and T. Shrimpton, Tag size does matter: Attacks and proofs for the TLS record protocol, in *Advances in Cryptology – ASIACRYPT 2011* (D. H. Lee and X. Wang, eds.), *LNCS* **7073**, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 372–389. https://doi.org/10.1007/978-3-642-25385-0_20.

[PBO+03] B. Preneel, A. Biryukov, E. Oswald, B. V. Rompay, L. Granboulan, E. Dottax, S. Murphy, A. Dent, J. White, M. Dichtl, S. Pyka, M. Schafheutle, P. Serf, E. Biham, E. Barkan, O. Dunkelman, J.-J. Quisquater, M. Ciet, F. Sica, L. Knudsen, M. Parker, and H. Raddum, *NESSIE Security Report*, Deliverable D20, NESSIE Consortium, February 2003, Version 2.0.

[Res18] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, IETF RFC 8446, August 2018. Available at https://www.rfc-editor.org/rfc/rfc8446.

[Rog04] P. Rogaway, Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac, in *Advances in Cryptology - ASIACRYPT 2004* (P. J. Lee, ed.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 16–31.

[RWZ12] P. Rogaway, M. Wooding, and H. Zhang, The security of ciphertext stealing, in *Fast Software Encryption* (A. Canteaut, ed.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 180–195.

[SH02] J. Schaad and R. Housley, Advanced Encryption Standard (AES) Key Wrap Algorithm, IETF RFC 3394, 2002. Available at https://www.rfc-editor.org/rfc/rfc3394.

[Sch15] B. Schneier, *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*, 20th anniversary edition ed., Wiley, 2015. Available at https://learning.oreilly.com/library/view/applied-cryptography-protocols/9781119096726.

[Sma16] N. P. Smart, *Cryptography Made Simple*, *Information Security and Cryptography*, Springer International Publishing Switzerland, 2016. https://doi.org/10.1007/978-3-319-21936-3.

[vJ11] H. C. van Tilborg and S. Jajodia (eds.), *Encyclopedia of Cryptography and Security*, Springer, Boston, MA, 2011. https://doi.org/10.1007/978-1-4419-5906-5.