8 Block Cipher Modes of Operation

One of the drawbacks of the previous CPA-secure encryption scheme is that its ciphertexts are λ bits longer than its plaintexts. In the common case that we are using a block cipher with blocklength $blen = \lambda$, this means that ciphertexts are **twice as long** as plaintexts. Is there any way to encrypt data (especially lots of it) without requiring such a significant overhead?

A **block cipher mode** refers to a way to use a block cipher to efficiently encrypt a large amount of data (more than a single block). In this chapter, we will see the most common modes for CPA-secure encryption of long plaintexts.

8.1 A Tour of Common Modes

As usual, *blen* will denote the blocklength of a block cipher F. In our diagrams, we'll write F_k as shorthand for $F(k, \cdot)$. When m is the plaintext, we will write $m = m_1 \| m_2 \| \cdots \| m_\ell$, where each m_i is a single block (so ℓ is the length of the plaintext measured in blocks). For now, we will assume that m is an exact multiple of the block length.

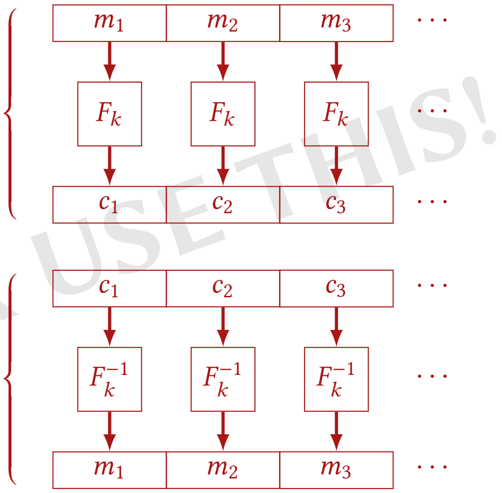
ECB: Electronic Codebook (NEVER NEVER USE THIS! NEVER‼)

The most obvious way to use a block cipher to encrypt a long message is to just apply the block cipher independently to each block. The only reason to know about this mode is to know never to use it (and to publicly shame anyone who does). It can't be said enough times: **never use ECB mode!** It does not provide security of encryption; can you see why?

Construction 8.1
(ECB Mode)

```
$\underline{\text{Enc}(k, m_1 \| \cdots \| m_\ell)}$:
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i)$
  return $c_1 \| \cdots \| c_\ell$
```

```
$\underline{\text{Dec}(k, c_1 \| \cdots \| c_\ell)}$:
  for $i = 1$ to $\ell$:
    $m_i := F^{-1}(k, c_i)$
  return $m_1 \| \cdots \| m_\ell$
```



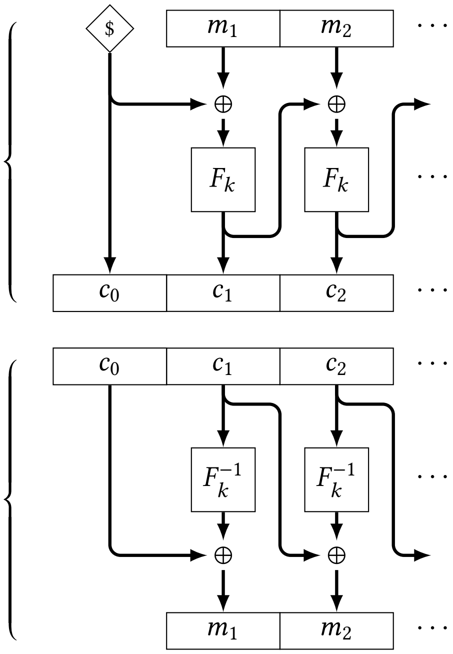
CBC: Cipher Block Chaining

CBC (which stands for cipher block chaining) is the most common mode in practice. The CBC encryption of an ℓ-block plaintext is $\ell + 1$ blocks long. The first ciphertext block is called an **initialization vector (IV).** Here we have described CBC mode as a *randomized* encryption, with the IV of each ciphertext being chosen uniformly. As you know, randomization is necessary (but not sufficient) for achieving CPA security, and indeed CBC mode provides CPA security.

Construction 8.2
(CBC Mode)

```
$\underline{\text{Enc}(k, m_1 \| \cdots \| m_\ell)}$:
  $c_0 \leftarrow \{0,1\}^{blen}$:
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1})$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

```
$\underline{\text{Dec}(k, c_0 \| \cdots \| c_\ell)}$:
  for $i = 1$ to $\ell$:
    $m_i := F^{-1}(k, c_i) \oplus c_{i-1}$
  return $m_1 \| \cdots \| m_\ell$
```



CTR: Counter

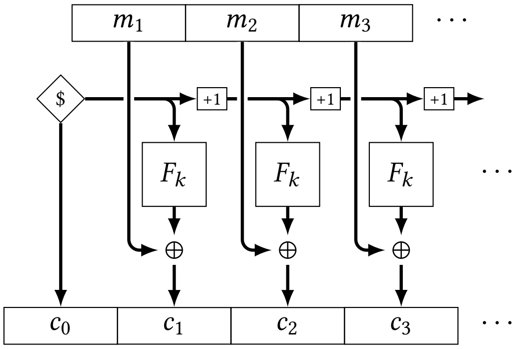
The next most common mode in practice is counter mode (usually abbreviated as CTR mode). Just like CBC mode, it involves an additional IV block r that is chosen uniformly. The idea is to then use the sequence

$$F(k,r); \quad F(k,r+1); \quad F(k,r+2); \quad \cdots$$

as a long one-time pad to mask the plaintext. Since r is a block of bits, the addition expressions like $r + 1$ refer to addition modulo 2^{blen} (this is the typical behavior of unsigned addition in a processor).

Construction 8.3 (CTR Mode)

```
$\underline{\text{Enc}(k, m_1\|\cdots\|m_\ell)}$:
  $r \leftarrow \{0,1\}^{blen}$
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $c_i := F(k,r) \oplus m_i$
    $r := r + 1 \% 2^{blen}$
  return $c_0\|\cdots\|c_\ell$
```



OFB: Output Feedback

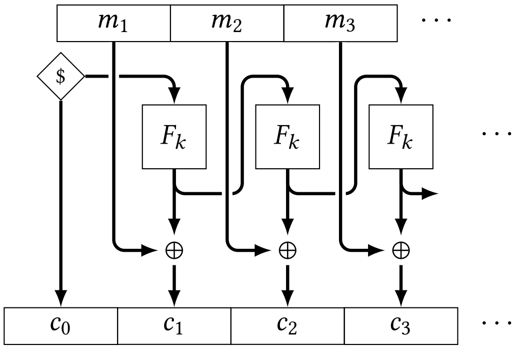
OFB (output feedback) mode is rarely used in practice. We'll include it in our discussion because it has the easiest security proof. As with CBC and CTR modes, OFB starts with a random IV r, and then uses the sequence:

$$F(k,r); \quad F(k,F(k,r)); \quad F(k,F(k,F(k,r))); \quad \cdots$$

as a one-time pad to mask the plaintext.

Construction 8.4 (OFB Mode)

```
$\underline{\text{Enc}(k, m_1\|\cdots\|m_\ell)}$:
  $r \leftarrow \{0,1\}^{blen}$
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r := F(k,r)$
    $c_i := r \oplus m_i$
  return $c_0\|\cdots\|c_\ell$
```



Compare & Contrast

CBC and CTR modes are essentially the only two modes that are ever considered in practice for CPA security. Both provide the same security guarantees, and so any comparison between the two must be based on factors outside of the CPA security definition. Here are a few properties that are often considered when choosing between these modes:

- Although we have not shown the decryption algorithm for CTR mode, it does not even use the block cipher's inverse F^{-1}. This is similar to our PRF-based encryption scheme from the previous chapter (in fact, CTR mode collapses to that construction when restricted to 1-block plaintexts). Strictly speaking, this means CTR mode can be instantiated from a PRF; it doesn't need a PRP. However, in practice it is rare to encounter an efficient PRF that is not a PRP.
- CTR mode encryption can be parallelized. Once the IV has been chosen, the ith block of ciphertext can be computed without first computing the previous $i-1$

blocks. CBC mode does not have this property, as it is inherently sequential. Both modes have a parallelizable *decryption* algorithm, though.

- If calls to the block cipher are expensive, it might be desirable to pre-compute and store them before the plaintext is known. CTR mode allows this, since only the IV affects the input given to the block cipher. In CBC mode, the plaintext influences the inputs to the block cipher, so these calls cannot be pre-computed before the plaintext is known.

- It is relatively easy to modify CTR to support plaintexts that are not an exact multiple of the blocklength. (This is left as an exercise.) We will see a way to make CBC mode support such plaintexts as well, but it is far from trivial.

- So far all of the comparisons have favored CTR mode, so here is one important property that favors CBC mode. It is common for implementers to misunderstand the security implications of the IV in these modes. Many careless implementations allow an IV to be reused. Technically speaking, reusing an IV (other than by accident, as the birthday bound allows) means that the scheme was not implemented correctly. But rather than dumping the blame on the developer, it is good design practice to anticipate likely misuses of a system and, when possible, try to make them non-catastrophic.

  The effects of IV-reuse in CTR mode are quite devastating to message privacy (see the exercises). In CBC mode, reusing an IV can actually be safe, if the two plaintexts have different first blocks!

8.2 CPA Security and Variable-Length Plaintexts

Here's a big surprise: none of these block cipher modes achieve CPA security, or at least CPA security as we have been defining it.

Example *Consider a block cipher with blen = λ, used in CBC mode. As you will see, there is nothing particularly specific to CBC mode, and the same observations apply to the other modes.*

In CBC mode, a plaintext consisting of ℓ blocks is encrypted into a ciphertext of $\ell + 1$ blocks. In other words, the ciphertext ***leaks the number of blocks in the plaintext.*** *We can leverage this observation into the following attack:*

```
$\mathcal{A}$:
$c :=$ EAVESDROP($0^\lambda, 0^{2\lambda}$)
return $|c| \overset{?}{=} 2\lambda$
```

The distinguisher chooses a 1-block plaintext and a 2-block plaintext. If this distinguisher is linked to $\mathcal{L}_{\text{cpa-L}}$, the 1-block plaintext is encrypted and the resulting ciphertext is 2 blocks (2λ bits) long. If the distinguisher is linked to $\mathcal{L}_{\text{cpa-R}}$, the 2-block plaintext is encrypted and the resulting ciphertext is 3 blocks (3λ bits) long. By simply checking the length of the ciphertext, this distinguisher can tell the difference and achieve advantage 1.

So, technically speaking, these block cipher modes do not provide CPA security, since ciphertexts leak the length (measured in blocks) of the plaintext. But suppose we don't really care about hiding the length of plaintexts.[1] Is there a way to make a security definition that says: **ciphertexts hide everything about the plaintext, except their length**?

It is clear from the previous example that a distinguisher can successfully distinguish the CPA libraries if it makes a query EAVESDROP(m_L, m_R) with $|m_L| \neq |m_R|$. A simple way to change the CPA security definition is to just disallow this kind of query. The libraries will give an error message if $|m_L| \neq |m_R|$. This would allow the adversary to make the challenge plaintexts differ in any way of his/her choice, *except in their length*. It doesn't really matter whether $|m|$ refers to the length of the plaintext in bits or in blocks — whichever makes the most sense for a particular scheme.

From now on, when discussing encryption schemes that support *variable-length* plaintexts, CPA security will refer to the following updated libraries:

```
$\mathcal{L}^{\Sigma}_{\text{cpa-L}}$
$k \leftarrow \Sigma.\text{KeyGen}$
CTXT($m_L, m_R \in \Sigma.\mathcal{M}$):
    if $|m_L| \neq |m_R|$ return err
    $c := \Sigma.\text{Enc}(k, m_L)$
    return $c$
```

```
$\mathcal{L}^{\Sigma}_{\text{cpa-R}}$
$k \leftarrow \Sigma.\text{KeyGen}$
CTXT($m_L, m_R \in \Sigma.\mathcal{M}$):
    if $|m_L| \neq |m_R|$ return err
    $c := \Sigma.\text{Enc}(k, m_R)$
    return $c$
```

In the definition of CPA$ security (pseudorandom ciphertexts), the $\mathcal{L}_{\text{cpa\$-rand}}$ library responds to queries with uniform responses. Since these responses must look like ciphertexts, they must have the appropriate length. For example, for the modes discussed in this chapter, an ℓ-block plaintext is expected to be encrypted to an $(\ell + 1)$-block ciphertext. So, based on the length of the plaintext that is provided, the library must choose the appropriate ciphertext length. We are already using $\Sigma.\mathcal{C}$ to denote the set of possible ciphertexts of an encryption scheme Σ. So let's extend the notation slightly and write $\Sigma.\mathcal{C}(\ell)$ to denote the set of possible ciphertexts for plaintexts of length ℓ. Then when discussing encryption schemes supporting variable-length plaintexts, CPA$ security will refer to the following libraries:

```
$\mathcal{L}^{\Sigma}_{\text{cpa\$-real}}$
$k \leftarrow \Sigma.\text{KeyGen}$
CHALLENGE($m \in \Sigma.\mathcal{M}$):
    $c := \Sigma.\text{Enc}(k, m)$
    return $c$
```

```
$\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$
CHALLENGE($m \in \Sigma.\mathcal{M}$):
    $c \leftarrow \Sigma.\mathcal{C}(|m|)$
    return $c$
```

Note that the $\mathcal{L}_{\text{cpa\$-rand}}$ library does not use any information about m other than its length. This again reflects the idea that ciphertexts leak nothing about plaintexts other than their length.

[1] Indeed, hiding the length of communication (in the extreme, hiding the *existence* of communication) is a very hard problem.

In the exercises, you are asked to prove that, with respect to these updated security definitions, CPA$ security implies CPA security as before.

Don't Take Length-Leaking for Granted!

We have just gone from requiring encryption to leak *no partial information* to casually allowing some specific information to leak. Let us not be too hasty about this!

If we want to truly support plaintexts of *arbitrary* length, then leaking the length is in fact unavoidable. But "unavoidable" doesn't mean "free of consequences." By observing only the length of encrypted network traffic, many serious attacks are possible. Here are several examples:

- When accessing Google maps, your browser receives many image tiles that comprise the map that you see. Each image tile has the same pixel dimensions. However, they are compressed to save resources, and not all images compress as significantly as others. Every region of the world has its own rather unique "fingerprint" of image-tile lengths. So even though traffic to and from Google maps is encrypted, the sizes of the image tiles are leaked. This can indeed be used to determine the region for which a user is requesting a map.[2] The same idea applies to auto-complete suggestions in a search form.

- Variable-bit-rate (VBR) encoding is a common technique in audio/video encoding. When the data stream is carrying less information (*e.g.*, a scene with a fixed camera position, or a quiet section of audio), it is encoded at a lower bit rate, meaning that each unit of time is encoded in fewer bits. In an encrypted video stream, the changes in bit rate are reflected as changes in packet length. When a user is watching a movie on Netflix or a Youtube video (as opposed to a live event stream), the bit-rate changes are consistent and predictable. It is therefore rather straight-forward to determine which video is being watched, even on an encrypted connection, just by observing the packet lengths.

- VBR is also used in many encrypted voice chat programs. Attacks on these tools have been increasing in sophistication. The first attacks on encrypted voice programs showed how to identify who was speaking (from a set of candidates), just by observing the stream of ciphertext sizes. Later attacks could determine the language being spoken. Eventually, when combined with sophisticated linguistic models, it was shown possible to even identify individual words to some extent!

It's worth emphasizing again that none of these attacks involve any attempt to break the encryption. The attacks rely solely on the fact that encryption leaks the length of the plaintexts.

8.3 Security of OFB Mode

In this section we will prove that OFB mode has pseudorandom ciphertexts (when the blocklength is $\mathit{blen} = \lambda$ bits). OFB encryption and decryption both use the forward direc-

[2] http://blog.ioactive.com/2012/02/ssl-traffic-analysis-on-google-maps.html

tion of F, so OFB provides security even when F is not invertible. Therefore we will prove security assuming F is simply a PRF.

Claim 8.5 *OFB mode (Construction 8.4) has CPA\$ security, if its underlying block cipher F is a secure PRF with parameters $in = out = \lambda$.*

Proof The general structure of the proof is very similar to the proof used for the PRF-based encryption scheme in the previous chapter (Construction 7.4). This is no coincidence: if OFB mode is restricted to plaintexts of a single block, we obtain exactly Construction 7.4!

The idea is that each ciphertext block (apart from the IV) is computed as $c_i := r \oplus m_i$. By the one-time pad rule, it suffices to show that the r values are independently pseudorandom. Each r value is the result of a call to the PRF. These PRF outputs will be independently pseudorandom only if all of the *inputs* to the PRF are *distinct*. In OFB mode, we use the *output* r of a previous PRF call as *input* to the next, so it is highly unlikely that this PRF output r matches a past PRF-input value. To argue this more precisely, the proof includes hybrids in which r is chosen without replacement (before changing r back to uniform sampling).

The formal sequence of hybrid libraries is given below:

$\mathcal{L}^{\text{OFB}}_{\text{cpa\$-real}}$:

```
$\mathcal{L}^{\text{OFB}}_{\text{cpa\$-real}}$
$k \leftarrow \{0,1\}^\lambda$

$\textsc{ctxt}(m_1 \| \cdots \| m_\ell)$:
  $r \leftarrow \{0,1\}^\lambda$
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r := F(k, r)$
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

The starting point is $\mathcal{L}^{\text{OFB}}_{\text{cpa\$-real}}$, shown here with the details of OFB filled in.

```
$\textsc{ctxt}(m_1 \| \cdots \| m_\ell)$:
  $r \leftarrow \{0,1\}^\lambda$
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r := \textsc{lookup}(r)$
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

$\diamond$

```
$\mathcal{L}^{F}_{\text{prf-real}}$
$k \leftarrow \{0,1\}^\lambda$

$\textsc{lookup}(r)$:
  return $F(k, r)$
```

The statements pertaining to the PRF F have been factored out in terms of $\mathcal{L}^{F}_{\text{prf-real}}$.

```
CTXT($m_1 \| \cdots \| m_\ell$):
  $r \leftarrow \{0,1\}^\lambda$
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r :=$ LOOKUP($r$)
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

◇

```
$\mathcal{L}^F_{\text{prf-rand}}$

$T :=$ empty

LOOKUP($x$):
  if $T[x]$ undefined:
    $T[x] \leftarrow \{0,1\}^\lambda$
  return $T[x]$
```

$\mathcal{L}^F_{\text{prf-real}}$ has been replaced by $\mathcal{L}^F_{\text{prf-rand}}$. By the PRF security of F, the change is indistinguishable.

Next, all of the statements that involve sampling values for the variable r are factored out in terms of the $\mathcal{L}_{\text{samp-L}}$ library from Lemma 4.11:

```
CHALLENGE($m_1 \| \cdots \| m_\ell$):
  $r :=$ SAMP()
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r :=$ LOOKUP($r$)
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

◇

```
$T :=$ empty

LOOKUP($x$):
  if $T[x]$ undefined:
    $T[x] :=$ SAMP()
  return $T[x]$
```

◇

```
$\mathcal{L}_{\text{samp-L}}$

SAMP():
  $r \leftarrow \{0,1\}^\lambda$
  return $r$
```

$\mathcal{L}_{\text{samp-L}}$ is then replaced by $\mathcal{L}_{\text{samp-R}}$. By Lemma 4.11, this change is indistinguishable:

```
CTXT($m_1 \| \cdots \| m_\ell$):
  $r :=$ SAMP()
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r :=$ LOOKUP($r$)
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

◇

```
$T :=$ empty

LOOKUP($x$):
  if $T[x]$ undefined:
    $T[x] :=$ SAMP()
  return $T[x]$
```

◇

```
$\mathcal{L}_{\text{samp-R}}$

$R := \emptyset$

SAMP():
  $r \leftarrow \{0,1\}^\lambda \setminus R$
  $R := R \cup \{r\}$
  return $r$
```

Arguments to LOOKUP are never repeated in this hybrid, so the middle library can be significantly simplified:

```
CTXT($m_1 \| \cdots \| m_\ell$):
  $r :=$ SAMP()
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r :=$ LOOKUP($r$)
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

◇

```
LOOKUP($x$):
  $t :=$ SAMP()
  return $t$
```

◇

```
$\mathcal{L}_{\text{samp-R}}$

$R := \emptyset$

SAMP():
  $r \leftarrow \{0,1\}^\lambda \setminus R$
  $R := R \cup \{r\}$
  return $r$
```

Next, $\mathcal{L}_{\text{samp-R}}$ is replaced by $\mathcal{L}_{\text{samp-L}}$. By Lemma 4.11, this change is indistinguishable:

```
CTXT($m_1 \| \cdots \| m_\ell$):
  $r :=$ SAMP()
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r :=$ LOOKUP($r$)
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

$\diamond$

```
LOOKUP($x$):
  $t :=$ SAMP()
  return $t$
```

$\diamond$

```
$\mathcal{L}_{\text{samp-L}}$
SAMP():
  $r \leftarrow \{0,1\}^\lambda$
  return $r$
```

Subroutine calls to LOOKUP and SAMP are inlined:

```
CTXT($m_1 \| \cdots \| m_\ell$):
  $r \leftarrow \{0,1\}^\lambda$
  $c_0 := r$
  for $i = 1$ to $\ell$:
    $r \leftarrow \{0,1\}^\lambda$
    $c_i := r \oplus m_i$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

Finally, the one-time pad rule is applied within the for-loop (omitting some common steps). Note that in the previous hybrid, each value of r is used *only once* as a one-time pad. The $i = 0$ case has also been absorbed into the for-loop. The result is $\mathcal{L}^{\text{OFB}}_{\text{cpa\$-rand}}$, since OFB encrypts plaintexts of ℓ blocks into $\ell + 1$ blocks.

```
$\mathcal{L}^{\text{OFB}}_{\text{cpa\$-rand}}$
CTXT($m_1 \| \cdots \| m_\ell$):
  for $i = 0$ to $\ell$:
    $c_i \leftarrow \{0,1\}^\lambda$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

The sequence of hybrids shows that $\mathcal{L}^{\text{OFB}}_{\text{cpa\$-real}} \approx \mathcal{L}^{\text{OFB}}_{\text{cpa\$-rand}}$, and so OFB mode has pseudorandom ciphertexts. ■

We proved the claim assuming F is a PRF only, since OFB mode does not require F to be invertible. Since we assume a PRF with parameters $in = out = \lambda$, the PRP switching lemma (Lemma 6.7) shows that OFB is secure also when F is a PRP with blocklength $n = \lambda$.

8.4 Padding & Ciphertext Stealing

So far we have assumed that all plaintexts are exact multiples of the blocklength. But data in the real world is not always so accommodating. How are block ciphers used in practice with data that has arbitrary length?

Padding

Padding just refers to any approach to encode arbitrary-length data into data that is a multiple of the blocklength. The only requirement is that this encoding is reversible. More formally, a **padding scheme** should consist of two algorithms:

- pad: takes as input a string of any length, and outputs a string whose length is a multiple of the blocklength
- unpad: the inverse of pad. We require that $\text{unpad}(\text{pad}(x)) = x$ for all strings x.

The idea is that the sender can encrypt $\text{pad}(x)$, which is guaranteed to be a multiple of the blocklength; the receiver can decrypt and run unpad on the result to obtain x.

In the real world, data almost always comes in **bytes** and not bits, so that will be our assumption here. In this section we will write bytes in hex, for example `8f`. Typical blocklengths are 128 bits (16 bytes) or 256 bits (32 bytes).

Here are a few common approaches for padding:

Null padding: The simplest padding approach is to just fill the final block with null bytes (`00`). The problem with this approach is that it is not always reversible. For example, pad(`31 41 59`) and pad(`31 41 59 00`) will give the same result. It is not possible to distinguish between a null byte that was added for padding and one that was intentionally the last byte of the data.

ANSI X.923 standard: Data is padded with null bytes, except for the last byte of padding which indicates how many padding bytes there are. In essence, the last byte of the padded message tells the receiver how many bytes to remove to recover the original message.

Note that in this padding scheme (and indeed in all of them), if the original unpadded data is *already* a multiple of the block length, then **an entire extra block of padding** must be added. This is necessary because it is possible for the original data to end with some bytes that look like valid padding (*e.g.*, `00 00 03`), and we do not want these bytes to be removed erroneously.

Example *Below are some examples of valid and invalid X.923 padding (using 16-byte blocks):*

`01 34 11 d9 81 88 05 57 1d 73 c3 00 00 00 00 05` ⇒ *valid*

`95 51 05 4a d6 5a a3 44 af b3 85 00 00 00 00 03` ⇒ *valid*

`71 da 77 5a 5e 77 eb a8 73 c5 50 b5 81 d5 96 01` ⇒ *valid*

`5b 1c 01 41 5d 53 86 4e e4 94 13 e8 7a 89 c4 71` ⇒ *invalid*

`d4 0d d8 7b 53 24 c6 d1 af 5f d6 f6 00 c0 00 04` ⇒ *invalid*

PKCS#7 standard: If b bytes of padding are needed, then the data is padded not with null bytes but with b bytes. Again, the last byte of the padded message tells the receiver how many bytes to remove.

Example *Below are some examples of valid and invalid PKCS#7 padding (using 16-byte blocks):*

`01 34 11 d9 81 88 05 57 1d 73 c3` **`05 05 05 05 05`** ⇒ *valid*

`95 51 05 4a d6 5a a3 44 af b3 85 03 03` **`03 03 03`** ⇒ *valid*

`71 da 77 5a 5e 77 eb a8 73 c5 50 b5 81 d5 96` **`01`** ⇒ *valid*

`5b 1c 01 41 5d 53 86 4e e4 94 13 e8 7a 89 c4` **`71`** ⇒ *invalid*

`d4 0d d8 7b 53 24 c6 d1 af 5f d6 f6` **`04 c0 04 04`** ⇒ *invalid*

ISO/IEC 7816-4 standard: The data is padded with a `80` byte followed by null bytes. To remove the padding, remove all trailing null bytes and ensure that the last byte is `80` (and then remove it too).

The significance of `80` is clearer when you write it in binary as `10000000`. So another way to describe this padding scheme is: append a `1` bit, and then pad with `0` bits until reaching the blocklength. To remove the padding, remove all trailing `0` bits as well as the rightmost `1` bit. Hence, this approach generalizes easily to padding data that is not a multiple of a byte.

Example *Below are some examples of valid and invalid ISO/IEC 7816-4 padding (using 16-byte blocks):*

`01 34 11 d9 81 88 05 57 1d 73 c3` **`80 00 00 00 00`** ⇒ *valid*

`95 51 05 4a d6 5a a3 44 af b3 85 03 03` **`80 00 00`** ⇒ *valid*

`71 da 77 5a 5e 77 eb a8 73 c5 50 b5 81 d5 96` **`80`** ⇒ *valid*

`5b 1c 01 41 5d 53 86 4e e4 94 13 e8 7a 89 c4` **`71`** ⇒ *invalid*

`d4 0d d8 7b 53 24 c6 d1 af 5f d6 f6` **`c4 00 00 00`** ⇒ *invalid*

The choice of padding scheme is not terribly important, and any of these is generally fine. Just remember that **padding schemes are not a security feature!** Padding is a public method for encoding data, and it does not involve any secret keys. The only purpose of padding is to enable functionality — using block cipher modes like CBC with data that is not a multiple of the block length.

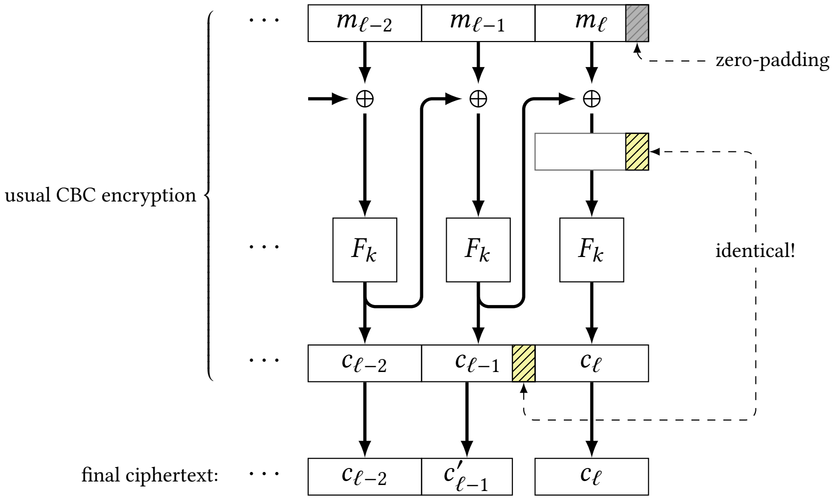
Furthermore, as we will see in the next chapter, padding is associated with certain attacks against improper use of encryption. Even though this is not really the fault of the padding (rather, it is the fault of using the wrong flavor of encryption), it is such a common pitfall that it is always worth considering in a discussion about padding.

Ciphertext Stealing

Another approach with a provocative name is **ciphertext stealing** (CTS, if you are not yet tired of three-leter acronyms), which results in ciphertexts that are not a multiple of the blocklength. The main idea behind ciphertext stealing is to use a standard block-cipher mode that only supports full blocks (*e.g.*, CBC mode), and then **simply throw away some bits of the ciphertext**, in such a way that decryption is still possible. If the last plaintext blocks is j bits short of being a full block, it is generally possible to throw away j bits of the ciphertext. In this way, a plaintext of n bits will be encrypted to a ciphertext of $blen + n$ bits, where $blen$ is the length of the extra IV block.

As an example, let's see ciphertext stealing as applied to CBC mode. Suppose the blocklength is *blen* and the last plaintext block m_ℓ is j bits short of being a full block. We start by extending m_ℓ with j zeroes (*i.e.*, null-padding the plaintext) and performing CBC encryption as usual.

Now our goal is to identify j bits of the CBC ciphertext that can be thrown away while still making decryption possible. In this case, the appropriate bits to throw away are **the last j bits of $c_{\ell-1}$** (the next-to-last block of the CBC ciphertext). The reason is illustrated in the figure below:



Suppose the receiver obtains this CBC ciphertext but the last j bits of $c_{\ell-1}$ have been deleted. How can he/she decrypt? The important idea is that those missing j bits were redundant, because there is another way to compute them.

In CBC encryption, the last value given as input into the block cipher is $c_{\ell-1} \oplus m_\ell$. Let us give this value a name $x^* := c_{\ell-1} \oplus m_\ell$. Since the last j bits of m_ℓ are 0's,[3] the last j bits of x^* are the last j bits of $c_{\ell-1}$ — the missing bits. Even though these bits are missing from $c_{\ell-1}$, the receiver has a different way of computing them as $x^* := F^{-1}(k, c_\ell)$.

Putting it all together, the receiver does the following: First, it observes that the ciphertext is j bits short of a full block. It computes $F^{-1}(k, c_\ell)$ and takes the last j bits of this value to be the missing bits from $c_{\ell-1}$. With the missing bits recovered, the receiver does CBC decryption as usual. The result is a plaintext consisting of ℓ full blocks, but we know that the last j bits of that plaintext are 0 padding that the receiver can remove.

It is convenient in an implementation for the boundaries between blocks to be in predictable places. For that reason, it is slightly awkward to remove j bits from the *middle* of the ciphertext during encryption (or add them during decryption), as we have done here. So in practice, the last two blocks of the ciphertext are often interchanged. In the example above, the resulting ciphertext (after ciphertext stealing) would be:

$$c_0 \| c_1 \| c_2 \cdots c_{\ell-3} \| c_{\ell-2} \| \boxed{c_\ell \| c'_{\ell-1}}, \text{ where } c'_{\ell-1} \text{ is the first } \mathit{blen} - j \text{ bits of } c_{\ell-1}.$$

[3] The receiver knows this fact, because the ciphertext is j bits short of a full block. The length of the (shortened) ciphertext is a signal about how many 0-bits of padding were used during encryption.

That way, all ciphertext blocks except the last one are the full *blen* bits long, and the boundaries between blocks appear consistently every *blen* bits. This "optimization" does add some significant edge cases to any implementation. One must also decide what to do when the plaintext is already an exact multiple of the blocklength — should the final two ciphertext blocks be swapped even in this case? Below we present an implementation of ciphertext stealing (CTS) that does *not* swap the final two blocks in this case. This means that it collapses to plain CBC mode when the plaintext is an exact multiple of the block length.

Construction 8.6 (CBC-CTS)

```
$\text{Enc}(k, m_1 \| \cdots \| m_\ell)$:
  // each $m_i$ is blen bits,
  // except possibly $m_\ell$

  $j := blen - |m_\ell|$
  $m_\ell := m_\ell \| 0^j$
  $c_0 \leftarrow \{0,1\}^{blen}$:
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1})$
  if $j \neq 0$:
    remove final $j$ bits of $c_{\ell-1}$
    swap $c_{\ell-1}$ and $c_\ell$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```

```
$\text{Dec}(k, c_0 \| \cdots \| c_\ell)$:
  // each $c_i$ is blen bits,
  // except possibly $c_\ell$

  $j := blen - |c_\ell|$
  if $j \neq 0$:
    swap $c_{\ell-1}$ and $c_\ell$
    $x :=$ last $j$ bits of $F^{-1}(k, c_\ell)$
    $c_{\ell-1} := c_{\ell-1} \| x$
  for $i = 1$ to $\ell$:
    $m_i := F^{-1}(k, c_i) \oplus c_{i-1}$
  remove final $j$ bits of $m_\ell$
  return $m_1 \| \cdots \| m_\ell$
```

The marked lines correspond to plain CBC mode.

Exercises

8.1. Prove that a block cipher in ECB mode does not provide CPA security. Describe a distinguisher and compute its advantage.

8.2. Describe OFB decryption mode.

8.3. Describe CTR decryption mode.

8.4. CBC mode:

(a) In CBC-mode encryption, if a single bit of the plaintext is changed, which ciphertext blocks are affected (assume the same IV is used)?

(b) In CBC-mode decryption, if a single bit of the ciphertext is changed, which plaintext blocks are affected?

8.5. Prove that CPA\$ security for variable-length plaintexts implies CPA security for variable-length plaintexts. Where in the proof do you use the fact that $|m_L| = |m_R|$?

8.6. Suppose that instead of applying CBC mode to a block cipher, we apply it to one-time pad. In other words, we replace every occurrence of $F(k, \star)$ with $k \oplus \star$ in the code for CBC encryption. Show that the result does not have CPA security. Describe a distinguisher and compute its advantage.

8.7. Prove that there is an attacker that runs in time $O(2^{\lambda/2})$ and that can break CPA security of CBC mode encryption with constant probability.

8.8. Below are several block cipher modes for encryption, applied to a PRP F with blocklength $blen = \lambda$. For each of the modes:

- Describe the corresponding decryption procedure.
- Show that the mode does **not** have CPA-security. That means describe a distinguisher and compute its advantage.

(a)

```
Enc($k, m_1 \| \cdots \| m_\ell$):
  $r_0 \leftarrow \{0,1\}^\lambda$
  $c_0 := r_0$
  for $i = 1$ to $\ell$:
    $r_i := F(k, m_i)$
    $c_i := r_i \oplus r_{i-1}$
  return $c_0 \| \cdots \| c_\ell$
```

(b)

```
Enc($k, m_1 \| \cdots \| m_\ell$):
  $c_0 \leftarrow \{0,1\}^\lambda$
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i) \oplus c_{i-1}$
  return $c_0 \| \cdots \| c_\ell$
```

(c)

```
Enc($k, m_1 \| \cdots \| m_\ell$):
  $c_0 \leftarrow \{0,1\}^\lambda$
  $m_0 := c_0$
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i) \oplus m_{i-1}$
  return $c_0 \| \cdots \| c_\ell$
```

(d)

```
Enc($k, m_1 \| \cdots \| m_\ell$):
  $c_0 \leftarrow \{0,1\}^\lambda$
  $r_0 := c_0$
  for $i = 1$ to $\ell$:
    $r_i := r_{i-1} \oplus m_i$
    $c_i := F(k, r_i)$
  return $c_0 \| \cdots \| c_\ell$
```

Mode (a) is similar to CBC, except the xor happens after, rather than before, the block cipher application. Mode (c) is essentially the same as CBC decryption.

8.9. Suppose you observe a CBC ciphertext and two of its blocks happen to be identical. What can you deduce about the plaintext? State some non-trivial property of the plaintext *that doesn't depend on the encryption key.*

8.10. The CPA\$-security proof for CBC encryption has a slight complication compared to the proof of OFB encryption. Recall that an important part of the proof is arguing that all inputs to the PRF are distinct.

In OFB, outputs of the PRF were fed directly into the PRF as inputs. The adversary had no influence over this process, so it wasn't so bad to argue that all PRF inputs were distinct (with probability negligibly close to 1).

By contrast, CBC mode takes an output block from the PRF, xor's it with a plaintext block (which is after all *chosen by the adversary*), and uses the result as input to the next PRF call. This means we have to be a little more careful when arguing that CBC mode gives distinct inputs to all PRF calls (with probability negligibly close to 1).

(a) Prove that the following two libraries are indistinguishable:

```
$\mathcal{L}_{\text{left}}$
samp($m \in \{0,1\}^\lambda$):
    $r \leftarrow \{0,1\}^\lambda$
    return $r$
```

```
$\mathcal{L}_{\text{right}}$
$R := \emptyset$

samp($m \in \{0,1\}^\lambda$):
    $r \leftarrow \{r' \in \{0,1\}^\lambda \mid r' \oplus m \notin R\}$
    $R := R \cup \{r \oplus m\}$
    return $r$
```

Hint: Use Lemma 4.12.

(b) Using part (a), and the security of the underlying PRF, prove the CPA$-security of CBC mode.

Hint: In $\mathcal{L}_{\text{right}}$, let R correspond to the set of all inputs sent to the PRF. Let m correspond to the next plaintext block. Instead of sampling r (the output of the PRF) uniformly as in $\mathcal{L}_{\text{left}}$, we sample r so that $r \oplus m$ has never been used as a PRF-input before. This guarantees that the next PRF call will be on a "fresh" input.

Note: Appreciate how important it is that the adversary chooses plaintext block m before "seeing" the output r from the PRF (which is included in the ciphertext).

★ 8.11. Prove that CTR mode achieves CPA$ security.

Hint: Use Lemma 4.12 to show that there is only negligible probability of chosing the IV so that the block cipher gets called on the same value twice.

8.12. Let F be a secure PRF with $out = in = \lambda$ and let $F^{(2)}$ denote the function $F^{(2)}(k, r) = F(k, F(k, r))$.

(a) Prove that $F^{(2)}$ is also a secure PRF.

(b) What if F is a secure PRP with blocklength *blen*? Is $F^{(2)}$ also a secure PRP?

8.13. This question refers to the nonce-based notion of CPA security.

(a) Show a definition for CPA$ security that incorporates both the nonce-based syntax of Section 7.1 and the variable-length plaintexts of Section 8.2.

(b) Show that CBC mode **not** secure as a nonce-based scheme (where the IV is used as a nonce).

(c) Show that CTR mode is **not** secure as a nonce-based scheme (where the IV is used as a nonce). Note that if we restrict (randomized) CTR mode to a single plaintext block, we get the CPA-secure scheme of Construction 7.4, which **is** is secure as a nonce-based scheme. The attack must therefore use the fact that plaintexts can be longer than one block. (Does the attack in part (b) work with single-block plaintexts?)

8.14. One way to convert a randomized-IV-based construction into a nonce-based construction is called the **synthetic IV** approach.

(a) The synthetic-IV (SIV) approach applied to CBC mode is shown below. Prove that it is CPA/CPA\$ secure as a nonce-based scheme (refer to the security definitions from the previous problem):

```
$\textsf{SIV-CBC.Enc}\Big((k_1,k_2), v, m_1\|\cdots\|m_\ell\Big)$:
  $c_0 := F(k_1, v)$
  for $i = 1$ to $\ell$:
    $c_i := F(k_2, m_i \oplus c_{i-1})$
  return $c_0\|c_1\|\cdots\|c_\ell$
```

Instead of chosing a random IV c_0, it is generated deterministically from the nonce v using the block cipher F. In your proof, you can use the fact that randomized CBC mode has CPA\$ security, and that F is also a secure PRF.

(b) It is important that the SIV construction uses two keys for different purposes. Suppose that we instead used the same key throughout:

```
$\textsf{BadSIV-CBC.Enc}(k, v, m_1\|\cdots\|m_\ell)$:
  $c_0 := F(k, v)$
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1})$
  return $c_0\|c_1\|\cdots\|c_\ell$
```

Show that the resulting scheme does **not** have CPA\$ security (in the nonce-based sense). Ignore the complication of padding, and only consider plaintexts that are a multiple of the blocklength. Describe a successful distinguisher and compute its advantage.

(c) For randomized encryption, it is necessary to include the IV in the ciphertext; otherwise the receiver cannot decrypt. In the nonce-based setting we assume that the receiver knows the correct nonce (*e.g.*, from some out-of-band communication). With that in mind, we could modify the scheme from part (b) to remove c_0, since the receiver could reconstruct it anyway from v.

Show that even with this modification, the scheme still fails to be CPA-secure under the nonce-based definition.

8.15. Implementers are sometimes cautious about IVs in block cipher modes and may attempt to "protect" them. One idea for protecting an IV is to prevent it from directly appearing in the ciphertext. The modified CBC encryption below sends the IV through the block cipher before including it in the ciphertext:

```
$\textsf{Enc}(k, m_1\|\cdots\|m_\ell)$:
  $c_0 \leftarrow \{0,1\}^{blen}$
  $c'_0 := F(k, c_0)$
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1})$
  return $c'_0\|c_1\|\cdots\|c_\ell$
```

This ciphertext can be decrypted by first computing $c_0 := F^{-1}(k, c_0')$ and then doing usual CBC decryption on $c_0 \| \cdots \| c_\ell$.

Show that this new scheme is **not** CPA-secure (under the traditional definitions for randomized encryption).

8.16. Suppose a bad implementation leads to two ciphertexts being encrypted with the same IV, rather than a random IV each time.

(a) Characterize as thoroughly as you can what information is leaked about the plaintexts when CBC mode was used and an IV is repeated.

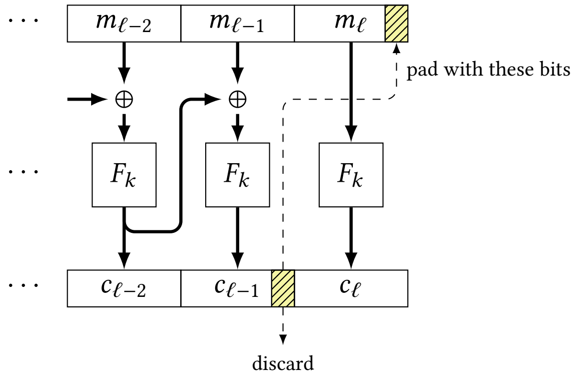
(b) Characterize as thoroughly as you can what information is leaked about the plaintexts when CTR mode was used and an IV is repeated.

8.17. Describe how to extend CTR and OFB modes to deal with plaintexts of arbitrary length (without using padding). Why is it so much simpler than CBC ciphertext stealing?

8.18. The following technique for ciphertext stealing in CBC was proposed in the 1980s and was even adopted into commercial products. Unfortunately, it's insecure.

Suppose the final plaintext block m_ℓ is *blen* $- j$ bits long. Rather than padding the final block with zeroes, it is padded with *the last j bits of ciphertext block* $c_{\ell-1}$. Then the padded block m_ℓ is sent through the PRP to produce the final ciphertext block c_ℓ. Since the final j bits of $c_{\ell-1}$ are recoverable from c_ℓ, they can be discarded.

If the final block of plaintext is already *blen* bits long, then standard CBC mode is used.



Show that the scheme does **not** satisfy CPA\$ security. Describe a distinguisher and compute its advantage.

Hint: ask for several encryptions of plaintexts whose last block is *blen* $- 1$ bits long.

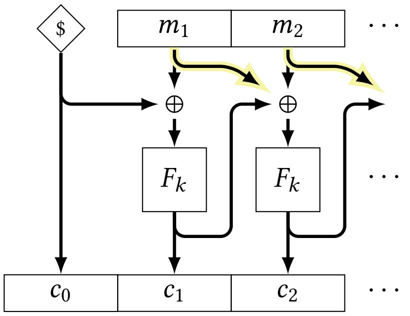
8.19. Prove that *any* CPA-secure encryption remains CPA-secure when augmented by padding the input.

8.20. Prove that CBC with ciphertext stealing has CPA\$ security. You may use the fact that CBC mode has CPA\$ security when restricted to plaintexts whose length is an exact multiple of the blocklength (*i.e.*, CBC mode without padding or ciphertext stealing).

Hint: Let CBC denote standard CBC mode restricted to plaintext space $\mathcal{M} = (\{0,1\}^{blen})^*$, and let CBC-CTS denote CBC mode with ciphertext stealing (so $\mathcal{M} = \{0,1\}^*$). Observe that it is easy to implement a call to $\mathcal{L}^{\text{CBC-CTS}}_{\text{cpa\$-real}}$ by a related call to $\mathcal{L}^{\text{CBC}}_{\text{cpa\$-real}}$ plus a small amount of additional processing.

8.21. Propagating CBC (PCBC) mode refers to the following variant of CBC mode:

```
Enc(k, $m_1 \| \cdots \| m_\ell$):
  $c_0 \leftarrow \{0,1\}^{blen}$;
  $m_0 := 0^{blen}$
  for $i = 1$ to $\ell$:
    $c_i := F(k, m_i \oplus c_{i-1} \oplus m_{i-1})$
  return $c_0 \| c_1 \| \cdots \| c_\ell$
```



(a) Describe PCBC decryption.

(b) Assuming that standard CBC mode has CPA\$-security (for plaintexts that are exact multiple of the block length), prove that PCBC mode also has CPA\$-security (for the same plaintext space).

Hint: Write PCBC encryption using plain CBC encryption as a subroutine.

(c) Consider the problem of adapting CBC ciphertext stealing to PCBC mode. Suppose the final plaintext block m_ℓ has $blen - j$ bits, and we pad it with the final j bits of the previous plaintext block $m_{\ell-1}$. Show that discarding the last j bits of $c_{\ell-1}$ still allows for correct decryption and results in CPA\$ security.

Hint: See Exercise 8.20.

(d) Suppose the final plaintext block is padded using the final j bits of the previous *ciphertext* block $c_{\ell-1}$. Although correct decryption is still possible, the construction is no longer secure. Show an attack violating the CPA\$-security of this construction. Why doesn't the proof approach from part (c) work?

Hint: Ask for several encryptions of plaintexts whose last block is 1 bit long.