9 Block Cipher Modes of Operation

Block ciphers / PRPs can only act on a single block (element of $\{0,1\}^{blen}$) of data at a time. In the previous section we showed at least one way to use a PRP (in fact, a PRF sufficed) to achieve CPA-secure encryption of a single block of data. This prompts the question, *can we use PRFs/PRPs to encrypt data that is longer than a single block?*

A **block cipher mode** specifies how to handle data that spans multiple blocks. Block cipher modes are where block ciphers really shine. There are modes for (CPA-secure) encryption, modes for data integrity, modes that achieve both privacy and integrity, modes for hard drive encryption, modes that gracefully recover from errors in transmission, modes that are designed to croak upon transmission errors, and so on. There is something of a cottage industry of clever block cipher modes, each with their own unique character and properties. Think of this chapter as a tour through the most common modes.

9.1 Common Modes

In this section, we will consider encryption modes for long plaintexts. As usual, *blen* will denote the blocklength of the 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.

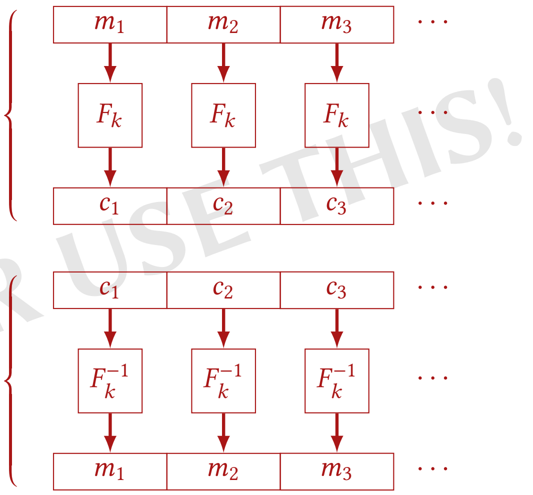
ECB: Electronic Codebook (NEVER NEVER USE THIS! NEVER‼)

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 scold 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 9.1 (ECB Mode)

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

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

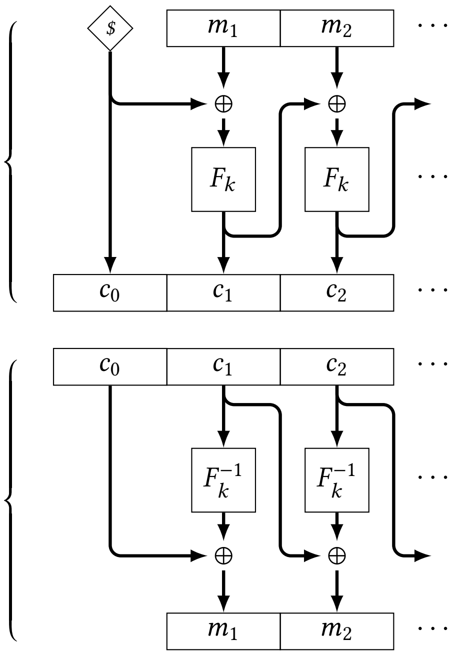
CBC is by far the most common block cipher mode in everyday use. It is used primarily to achieve CPA (CPA\$) security, but we will see later that a variant can also be used for data integrity properties.

Since CBC mode results in CPA-secure encryption, it's no surprise that its encryption algorithm is *randomized*. In particular, CBC mode specifies that a random block is chosen, which is called the **initialization vector (IV).** The IV is included in the output, and as a result, encrypting ℓ blocks under CBC mode results in $\ell + 1$ blocks of output.

Construction 9.2 (CBC Mode)

```
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 $c_0 c_1 \cdots c_\ell$

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

Counter mode, like CBC mode, begins with the choice of a random IV block $r \leftarrow \{0,1\}^{blen}$. Then the sequence

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

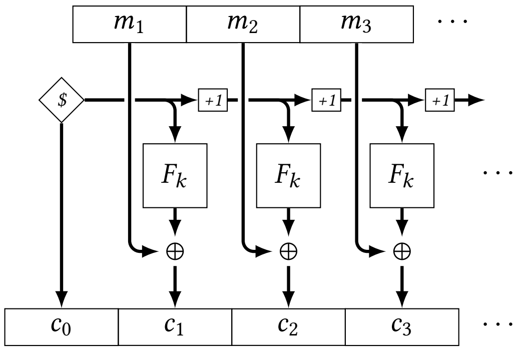
is used as a long one-time pad to mask the plaintext. Here, we interpret a block as an element of $\mathbb{Z}_{2^{blen}}$, so addition is performed modulo 2^{blen}.

Counter mode has a few nice features that lead many designers to favor it over CBC:

- Both encryption and decryption use F only in the forward direction (the decryption algorithm for CTR is left as an exercise). In particular, this means that F can be a PRF that is not a PRP. Modes with this property are sometimes called *stream cipher modes.* They can be easily adapted to support plaintexts that are not an exact multiple of the blocklength.
- It is one of the few modes for which encryption can be parallelized. That is, once the IV has been chosen, the ith block of ciphertext can be computed without first computing the previous $i-1$ blocks.

Construction 9.3 (CTR Mode)

```
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 mode is another stream cipher mode that uses F only in the forward direction. As with the other modes, a random IV block r is chosen at the beginning. Then the sequence

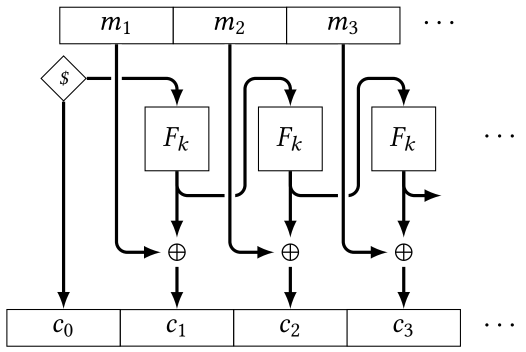
$$F(k,r); \quad F(k,F(k,r)); \quad F(k,F(k,F(k,r))); \quad \cdots$$

is used as a one-time pad to mask the plaintext. The name "output feedback" comes from this idea of repeatedly feeding the output of F into itself.

OFB is not nearly as widespread as CBC or CTR, but we include it here for self-serving reasons: it has the easiest security proof!

Construction 9.4
(OFB Mode)

```
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 $c_0 \cdots c_\ell$
```



9.2 CPA Security for Variable-Length Plaintexts

Block cipher modes allow us to encrypt a plaintext consisting of any number of blocks. In these modes, the length of the ciphertext depends on the length of the plaintext. So in that regard, these encryption schemes do not hide *all information* about the plaintext — in particular, they do not hide its length! We would run into problems trying to prove CPA security of these modes with respect to plaintext space $\mathcal{M} = (\{0,1\}^{blen})^*$. A successful distinguisher would break CPA security simply by chosing m_L and m_R of different lengths (measured in blocks) and then inspecting the length of the resulting ciphertext.

Leaking the length of the plaintext (measured in blocks) seems like a rather minor concession. But as we just described, it is necessary to modify our very conservative security definitions to allow any leakage at all.

For CPA security, we argued that if a distinguisher queries the CHALLENGE subroutine with plaintexts of different length in the CPA libraries, we cannot expect the libraries to be indistinguishable. The easiest way to avoid this situation is to simply insist that $|m_L| = |m_R|$. This would allow the adversary to make the challenge plaintexts differ in any way of his/her choice, *except in their length.* We can interpret the resulting security definition to mean that the scheme would hide all partial information about the plaintext apart from its length.

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

```
$\mathcal{L}^{\Sigma}_{\text{cpa-L}}$
$k \leftarrow \Sigma$.KeyGen

CHALLENGE($m_L, m_R \in \Sigma.\mathcal{M}$):
  if $|m_L| \neq |m_R|$ return null
  $c := \Sigma$.Enc($k, m_L$)
  return $c$
```

```
$\mathcal{L}^{\Sigma}_{\text{cpa-R}}$
$k \leftarrow \Sigma$.KeyGen

CHALLENGE($m_L, m_R \in \Sigma.\mathcal{M}$):
  if $|m_L| \neq |m_R|$ return null
  $c := \Sigma$.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 $\Sigma.\mathcal{C}(\ell)$ 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:

```
$\mathcal{L}^{\Sigma}_{\text{cpa\$-real}}$
$k \leftarrow \Sigma.\mathsf{KeyGen}$

CHALLENGE($m \in \Sigma.\mathcal{M}$):
    $c := \Sigma.\mathsf{Enc}(k, m)$
    return $c$
```

```
$\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$

CHALLENGE($m \in \Sigma.\mathcal{M}$):
    $c \leftarrow \Sigma.\mathcal{C}(|m|)$
    return $c$
```

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 that doesn't mean it is consequence-free. 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.[1] 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 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.

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

- ► 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.

9.3 Security of OFB Mode

In this section we will prove that OFB mode has pseudorandom ciphertexts (when the blocklength is $blen = \lambda$ bits). OFB encryption and decryption both use the forward direction of F, so OFB provides security even when F is not invertible. Therefore we will prove security assuming F is simply a PRF.

Claim 9.5 *OFB mode (Construction 9.4) 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 8.4). This is no coincidence: if OFB mode is restricted to plaintexts of a single block, we obtain exactly Construction 8.4!

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:

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

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

CHALLENGE($m_1 \cdots m_\ell$):
  $r \leftarrow \{0,1\}^\lambda$
  $c_0 := r$
  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.

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

$\diamond$

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

QUERY($r$):
  return $F(k, r)$
```

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

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

$\diamond$

```
$\mathcal{L}^F_{\text{prf-rand}}$
$T :=$ empty

QUERY($x$):
  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 4.10:

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

$\diamond$

```
$T :=$ empty

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

$\diamond$

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

$\mathcal{L}_{\text{samp-L}}$ is then replaced by $\mathcal{L}_{\text{samp-R}}$. By Lemma 4.10, this change is indistinguishable:

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

$\diamond$

```
$T :=$ empty

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

$\diamond$

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

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

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

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

$\diamond$

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

$\diamond$

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

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 4.10, this change is indistinguishable:

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

$\diamond$

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

$\diamond$

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

Subroutine calls to QUERY and SAMP are inlined:

```
CHALLENGE($m_1 \cdots m_\ell$):
    $r \leftarrow \{0,1\}^\lambda$
    $c_0$ := $r$
    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.

```
$\mathcal{L}^{\text{OFB}}_{\text{cpa\$-rand}}$
CHALLENGE($m_1 \cdots m_\ell$):
    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 7.3) shows that OFB is secure also when F is a PRP with blocklength $n = \lambda$.

9.4 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

While real-world data does not always come in multiples of the blocklength (typically 128 or 256 bits), it does almost always come in multiples of *bytes* (8 bits). So for now let us assume that the data is represented as bytes. We will write bytes in hex, for example `8f`.

The most natural way to deal with plaintexts of odd lengths is to add some null bytes (byte `00`, consisting of all zeroes) to fill out the last block. However, this leads to problems when decrypting: how can one distinguish between null bytes added for padding and null bytes that were part of the original plaintext?

A **padding scheme** is a method to encode arbitrary-length data into data that is a multiple of the blocklength. Importantly, the padding scheme must be reversible! Many padding schemes are in use, and we show just one below.

The ANSI X.923 standard defines a padding scheme, for block length 128 bits = 16 bytes. In the algorithms below, $|x|$ refers to the length of string x, measured in *bytes*, not bits.

Construction 9.6 (ANSI X.923)

```
PAD(x):
    $\ell := 16 - (|x| \% 16)$
    return $x \| \text{PADSTR}(\ell)$

PADSTR(ℓ):
    return $\underbrace{\texttt{00} \cdots \texttt{00}}_{\ell-1}\ \ell$

UNPAD(y):
    if not VALIDPAD(y) then return err
    $\ell :=$ last byte of $y$ (as integer)
    return first $|y| - \ell$ bytes of $y$

VALIDPAD(y):
    if $|y| \% 16 \neq 0$ then return false
    $\ell :=$ last byte of $y$ (as integer)
    if $\ell \notin \{1, \ldots, 16\}$ then return false
    return $\text{PADSTR}(\ell) \overset{?}{=}$ last $\ell$ bytes of $y$
```

Example *Below are some blocks (16 bytes) with valid and invalid X.923 padding:*

```
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
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
```

Note what happens in PAD(x) when x is already a multiple of 16 bytes in length. In that case, an additional 16 bytes are added — `00` ··· `00` `10`, since `10` is the hex encoding of 16. This is necessary because the unpadded x might already end with something that looks like valid ANSI X.923 padding!

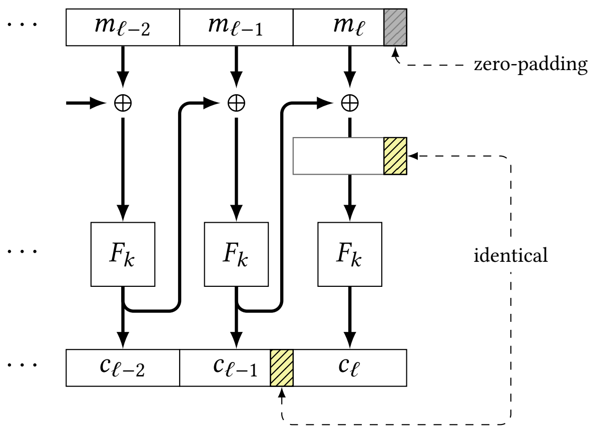
With a padding scheme available, one simply pads a string before encrypting and unpads the result after decrypting.

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. A plaintext of s bits will be encrypted to a ciphertext of $blen + s$ bits, where $blen$ is the length of the extra IV block.

As an example, we will show how ciphertext stealing applies to CBC mode. Suppose the blocklength is $blen$ and the last plaintext block m_ℓ has $blen - j$ bits. We extend m_ℓ with j zeroes and perform CBC encryption as usual.

Focus on the last j bits of block $c_{\ell-1}$. Because m_ℓ has been padded with zeroes, these final j bits of $c_{\ell-1}$ are also the last j bits of $F^{-1}(k, c_\ell)$, as shown in the diagram below.



Now imagine if we simply threw away the last j bits of $c_{\ell-1}$. These bits are somewhat redundant, so the decryption procedure would still have enough information to proceed. In particular, the decryption procedure could reconstruct the "missing" j bits via $F^{-1}(k, c_\ell)$. Furthermore, the fact that the ciphertext would be j bits shorter than a full block is a signal about how many bits should be treated in this special way (and how long the plaintext should be).

In general, the ciphertext stealing technique is to simply remove a carefully selected portion of the ciphertext, in a way that signals the original length of the plaintext and allows for correct decryption. For CBC mode, the appropriate bits to remove are the last j bits of the *next-to-last* ciphertext block. When decrypting, the last j bits of $F^{-1}(k, c_\ell)$ can be appended to $c_{\ell-1}$ and decryption can proceed as usual. The exercises ask you to prove the security of CBC mode with ciphertext stealing.

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, as we have done here. So in practice, the last two blocks of the ciphertext are typically interchanged. For the example above, the resulting ciphertext (after ciphertext stealing) would be:

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

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* 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 9.7 (CBC-CTS)

```
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 $c_0 c_1 \cdots c_\ell$
```

```
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

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

9.2. Describe OFB decryption mode.

9.3. Describe CTR decryption mode.

9.4. 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?

9.5. 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 $|m_L| = |m_R|$?

9.6. 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.

9.7. 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.

9.8. 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.

9.9. 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.*

9.10. The CPA\$-security proof for CBC encryption has a slight complication compared to the proof of OFB encryption. Recall that the 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 $r$
```

(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).

★ 9.11. Prove that CTR mode achieves CPA\$ security.

Hint: Try to characterize the event that an IV is chosen so that the block cipher is called on the same value twice. Argue that this event happens with negligible probability, by defining an appropriate pair of libraries and showing they are indistinguishable.

9.12. 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?

9.13. In this question we investigate the subtleties of initialization vectors and how their choice contributes to CPA security. In CBC mode (indeed, in every mode discussed here), the IV is included in the ciphertext, so it is something that the adversary eventually sees anyway.

(a) Describe a modification to the CPA or CPA\$ security definition that allows the adversary to *choose* the IV that is used. Then show that no such security is provided by CBC mode. In other words, describe a distinguisher for the new libraries you defined, and compute its advantage.

(b) Describe a modification to the CPA or CPA\$ security definition that allows the adversary to choose the IV that is used, *but is not allowed to re-use an IV.* Does CBC mode satisfy this kind of security? If so, give a proof for your new kind of security notion; if not, then describe a distinguisher and compute its advantage.

(c) Describe a modification to the CPA or CPA\$ security definition in which the library chooses IVs uniformly (as in the standard security definition) but allows the adversary to *see in advance what the next IV will be,* before choosing the plaintext. Does CBC mode satisfy this kind of security? If so, give a proof for your new kind of security notion; if not, then describe a distinguisher and compute its advantage.

In these problems, you are being asked to come up with a new security definition, so you must modify the libraries that comprise the security definition. There could be many "correct" ways to do this. All that matters is that the new interface of the libraries gives the calling program a way to do the things that are specified. In part (c) you might even want to add another subroutine to the interface.

9.14. The previous problem shows that the IV in CBC mode encryption is somewhat fragile. Suppose that in order to protect the IV, we send it through the block cipher before including it in the ciphertext. In other words, we modify CBC encryption as follows:

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

To decrypt, we first compute $c_0 := F^{-1}(k, c_0')$ and proceed as in usual CBC decryption.

(a) Show that the resulting scheme no longer satisfies CPA security. Describe a successful distinguisher and compute its advantage.

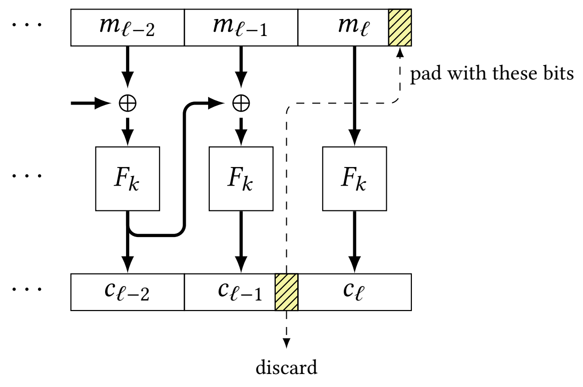
(b) Show that if we encrypt the IV c_0 with an independently chosen key k', the resulting scheme does in fact still achieve CPA\$ security. Your security proof can use the fact that standard CBC encryption has CPA\$ security.

9.15. 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?

9.16. 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.

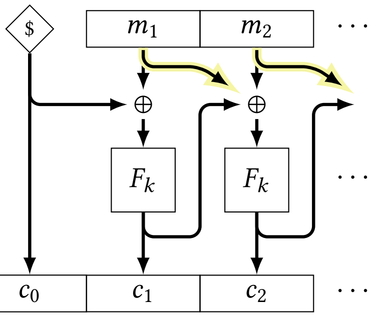
9.17. Prove that *any* CPA-secure encryption remains CPA-secure when augmented by padding the input.

9.18. 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.

9.19. 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 9.18.

(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.