7 Security Against Chosen Plaintext Attacks

Our previous security definitions for encryption capture the case where a key is used to encrypt only one plaintext. Clearly it would be more useful to have an encryption scheme that allows many plaintexts to be encrypted under the same key.

Fortunately we have arranged things so that we get the "correct" security definition when we modify the earlier definition in a natural way. We simply let the libraries choose a secret key once and for all, which is used to encrypt all plaintexts. More formally:

Definition 7.1 (CPA security) *Let Σ be an encryption scheme. We say that Σ has* ***security against chosen-plaintext attacks (CPA security)*** *if $\mathcal{L}^{\Sigma}_{\text{cpa-L}} \approx \mathcal{L}^{\Sigma}_{\text{cpa-R}}$, where:*

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

EAVESDROP($m_L, m_R \in \Sigma.\mathcal{M}$):
    $c := \Sigma.\text{Enc}(k, m_L)$
    return $c$
```

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

EAVESDROP($m_L, m_R \in \Sigma.\mathcal{M}$):
    $c := \Sigma.\text{Enc}(k, m_R)$
    return $c$
```

Notice how the key k is chosen at initialization time and is static for all calls to Enc. CPA security is often called "IND-CPA" security, meaning "indistinguishability of ciphertexts under chosen-plaintext attack."

7.1 Limits of Deterministic Encryption

We have already seen block ciphers / PRPs, which seem to satisfy everything needed for a secure encryption scheme. For a block cipher, F corresponds to encryption, F^{-1} corresponds to decryption, and all outputs of F look pseudorandom. What more could you ask for in a good encryption scheme?

Example *We will see that a block cipher, when used "as-is," is* ***not*** *a CPA-secure encryption scheme. Let F denote the block cipher and suppose its block length is blen.*

Consider the following adversary $\mathcal{A}$, that tries to distinguish the $\mathcal{L}_{\text{cpa-}\star}$ libraries:

```
$\mathcal{A}$
$c_1 :=$ EAVESDROP($0^{blen}, 0^{blen}$)
$c_2 :=$ EAVESDROP($0^{blen}, 1^{blen}$)
return $c_1 \overset{?}{=} c_2$
```

```
$\mathcal{A}$
$c_1 := \textsc{eavesdrop}(\mathtt{0}^{blen}, \mathtt{0}^{blen})$
$c_2 := \textsc{eavesdrop}(\mathtt{0}^{blen}, \mathtt{1}^{blen})$
return $c_1 \overset{?}{=} c_2$
```

$\diamond$

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

$\textsc{eavesdrop}(m_L, m_R)$:
  $c := F(k, m_L)$
  return $c$
```

When $\mathcal{A}$ is linked to $\mathcal{L}_{\text{cpa-L}}$, the EAVESDROP *algorithm will encrypt its first argument. So, c_1 and c_2 will both be computed as $F(k, \mathtt{0}^{blen})$. Since F is a deterministic function, this results in identical outputs from* EAVESDROP*. In other words $c_1 = c_2$, and $\mathcal{A} \diamond \mathcal{L}_{\text{cpa-L}}$* ***always*** *outputs 1.*

```
$\mathcal{A}$
$c_1 := \textsc{eavesdrop}(\mathtt{0}^{blen}, \mathtt{0}^{blen})$
$c_2 := \textsc{eavesdrop}(\mathtt{0}^{blen}, \mathtt{1}^{blen})$
return $c_1 \overset{?}{=} c_2$
```

$\diamond$

```
$\mathcal{L}^{\Sigma}_{\text{cpa-R}}$
$k \leftarrow \{\mathtt{0},\mathtt{1}\}^\lambda$

$\textsc{eavesdrop}(m_L, m_R)$:
  $c := F(k, m_R)$
  return $c$
```

When $\mathcal{A}$ is linked to $\mathcal{L}_{\text{cpa-R}}$, the EAVESDROP *algorithm will encrypt its second argument. So, c_1 and c_2 are computed as $c_1 = F(k, \mathtt{0}^{blen})$ and $c_2 = F(k, \mathtt{1}^{blen})$. Since F is a permutation, $c_1 \neq c_2$, so $\mathcal{A} \diamond \mathcal{L}_{\text{cpa-R}}$* ***never*** *outputs 1.*

This adversary has advantage 1 in distinguishing the libraries, so the bare block cipher F is ***not*** *a CPA-secure encryption scheme.*

Impossibility of Deterministic Encryption

The reason a bare block cipher does not provide CPA security is that it is **deterministic**. Calling $\text{Enc}(k, m)$ twice — with the same key and same plaintext — leads to the same ciphertext. Even one-time pad is deterministic.[1] One of the first and most important aspects of CPA security is that it is incompatible with deterministic encryption. **Deterministic encryption can never be CPA-secure!** In other words, we can attack the CPA-security of any scheme Σ, knowing only that it has deterministic encryption. The attack is a simple generalization of our attack against a bare PRP:

```
$\mathcal{A}$
arbitrarily choose *distinct* plaintexts $x, y \in \mathcal{M}$
$c_1 := \textsc{eavesdrop}(x, x)$
$c_2 := \textsc{eavesdrop}(x, y)$
return $c_1 \overset{?}{=} c_2$
```

A good way to think about what goes wrong with deterministic encryption is that it **leaks whether two ciphertexts encode the same plaintext,** and this is not allowed by CPA security. Think of sealed envelopes as an analogy for encryption. I shouldn't be able to tell whether two sealed envelopes contain the same text! We are only now seeing this issue because this is the first time our security definition allows an adversary to see multiple ciphertexts encrypted under the same key.

[1]Remember, we can always consider what will happen when running one-time pad encryption twice with the same key + plaintext. The one-time secrecy definition doesn't give us any security guarantees about using one-time pad in this way, but we can still consider it as a thought experiment.

Avoiding Deterministic Encryption

Is CPA security even possible? How exactly can we make a non-deterministic encryption scheme? This sounds challenging! We must design an Enc algorithm such that calling it twice with the same plaintext and key results in different ciphertexts (otherwise the attack $\mathcal{A}$ above violates CPA security). What's more, it must be possible to decrypt all of those different encryptions of the same plaintext to the correct value!

There are 3 general ways to design an encryption scheme that is not deterministic:

- Encryption/decryption can be **stateful**, meaning that every call to Enc or Dec will actually modify the value of k. The symmetric ratchet construction described in Section 5.5 could be thought of as such a stateful construction. The key is updated via the ratchet mechanism for every encryption. A significant drawback with stateful encryption is that synchronization between sender and receiver is fragile and can be broken if a ciphertext is lost in transit.
- Encryption can be **randomized**. Each time a plaintext is encrypted, the Enc algorithm chooses fresh, independent randomness specific to that encryption. The main challenge in designing a randomized encryption method is to incorporate randomness into each ciphertext in such a way that decryption is still possible. Although this sounds quite challenging, we have already seen such a method, and we will prove its CPA security in the next sections. In this book we will focus almost entirely on randomized encryption.
- Encryption can be **nonce-based**. A "nonce" stands for "number used only once," and it refers to an extra argument that is passed to the Enc and Dec algorithms. A nonce does not need to be chosen randomly; it does not need to be secret; it only needs to be **distinct** among all calls made to Enc. By guaranteeing that some input to Enc will be different every time (even when the key and plaintext are repeated), the Enc algorithm can be deterministic and still provide CPA security.

  Nonce-based encryption requires a change to the interface of encryption, and therefore a change to the correctness & security definitions as well. The encryption/decryption algorithms syntax is updated to $\text{Enc}(k, v, m)$ and $\text{Dec}(k, v, c)$, where v is a nonce. The correctness property is that $\text{Dec}(k, v, \text{Enc}(k, v, m)) = m$ for all k, v, m, so both encryption & decryption algorithms should use the same nonce. The security definition allows the adversary to choose the nonce, but gives an error if the adversary tries to encrypt multiple ciphertexts with the same nonce. In this way, the definition enforces that the nonces are distinct.

```
$k \leftarrow \Sigma.\text{KeyGen}$
$V := \emptyset$

EAVESDROP($v, m_L, m_R \in \Sigma.\mathcal{M}$):
    if $v \in V$: return err
    $V := V \cup \{v\}$
    $c := \Sigma.\text{Enc}(k, v, m_L)$
    return $c$
```

$\approx$

```
$k \leftarrow \Sigma.\text{KeyGen}$
$V := \emptyset$

EAVESDROP($v, m_L, m_R \in \Sigma.\mathcal{M}$):
    if $v \in V$: return err
    $V := V \cup \{v\}$
    $c := \Sigma.\text{Enc}(k, v, m_R)$
    return $c$
```

Note that the calling program provides a single value v (not a v_L and v_R). Both libraries use the nonce v that is given, and this implies that the encryption scheme does not need to *hide* v. If something is the same between both libraries, then it is not necessary to hide it in order to make the libraries indistinguishable.

If an encryption scheme does not fall into one of these three categories, it cannot satisfy our definition of CPA-security. You can and should use deterministic encryption as a sanity check against any proposed encryption algorithm.

7.2 Pseudorandom Ciphertexts

When we introduced one-time security of encryption (in Section 2.2), we had two variants of the definition. The more general variant said, roughly, that encryptions of m_L should look like encryptions of m_R. The more specific variant said that encryptions of every m should look uniform.

We can do something similar for CPA security, by defining a security definition that says "encryptions of m look uniform." Note that it is not sufficient to use the same security libraries from the one-time security definition. It is important for the library to allow multiple encryptions under the same key. Just because a single encryption is pseudorandom, it doesn't mean that multiple encryptions appear *jointly* pseudorandom. In particular, they may not look *independent* (this was an issue we saw when discussing the difficulty of constructing a PRF from a PRG).

Definition 7.2 (CPA\$ security) *Let Σ be an encryption scheme. We say that Σ has **pseudorandom ciphertexts in the presence of chosen-plaintext attacks (CPA\$ security)** if $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}} \approx \mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$, where:*

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

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

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

This definition is also called "IND\$-CPA", meaning "indistinguishable from random under chosen plaintext attacks." This definition will be useful to use since:

- It is easier to prove CPA\$ security than to prove CPA security. Proofs for CPA security tend to be about twice as long and twice as repetitive, since they involve getting to a "half-way hybrid" and then performing the same sequence of hybrids steps in reverse. Taking the proof only to the same half-way point is generally enough to prove CPA\$ security

- CPA\$ security implies CPA security. We show this below, but the main idea is the same as in the case of one-time security. If encryptions of all plaintexts look uniform, then encryptions of m_L look like encryptions of m_R.

- Most of the schemes we will consider achieve CPA\$ anyway.

Still, most of our high-level discussion of security properties will be based on CPA security. It is the "minimal" (*i.e.*, least restrictive) definition that appears to capture our security intuitions.

Claim 7.3 *If an encryption scheme has CPA\$ security, then it also has CPA security.*

Proof We want to prove that $\mathcal{L}^{\Sigma}_{\text{cpa-L}} \approx \mathcal{L}^{\Sigma}_{\text{cpa-R}}$, *using* the assumption that $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}} \approx \mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$. The sequence of hybrids follows:

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

```
$\mathcal{L}^{\Sigma}_{\text{cpa-L}}$
$k \leftarrow \Sigma.\text{KeyGen}$
EAVESDROP$(m_L, m_R)$:
    $c := \Sigma.\text{Enc}(k, m_L)$
    return $c$
```

The starting point is $\mathcal{L}^{\Sigma}_{\text{cpa-L}}$, as expected.

```
EAVESDROP$(m_L, m_R)$:               $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}}$
    $c :=$ CTXT$(m_L)$          $\diamond$    $k \leftarrow \Sigma.\text{KeyGen}$
    return $c$                       CTXT$(m)$:
                                         $c := \Sigma.\text{Enc}(k, m)$
                                         return $c$
```

It may look strange, but we have further factored out the call to Enc into a subroutine. It looks like *everything* from $\mathcal{L}_{\text{cpa-L}}$ has been factored out, but actually the original library still "makes the choice" of which of m_L, m_R to encrypt.

```
EAVESDROP$(m_L, m_R)$:               $\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$
    $c :=$ CTXT$(m_L)$          $\diamond$    CTXT$(m)$:
    return $c$                           $c \leftarrow \Sigma.\mathcal{C}$
                                         return $c$
```

We have replaced $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}}$ with $\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$. By our assumption, the change is indistinguishable.

```
EAVESDROP$(m_L, m_R)$:               $\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$
    $c :=$ CTXT$(m_R)$          $\diamond$    CTXT$(m)$:
    return $c$                           $c \leftarrow \Sigma.\mathcal{C}$
                                         return $c$
```

We have changed the argument being passed to CTXT. This has no effect on the library's behavior since CTXT completely ignores its argument in these hybrids.

```
EAVESDROP$(m_L, m_R)$:               $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}}$
    $c :=$ CTXT$(m_R)$          $\diamond$    $k \leftarrow \Sigma.\text{KeyGen}$
    return $c$                       CTXT$(m)$:
                                         $c := \Sigma.\text{Enc}(k, m)$
                                         return $c$
```

The mirror image of a previous step; we replace $\mathcal{L}_{\text{cpa\$-rand}}$ with $\mathcal{L}_{\text{cpa\$-real}}$.

$\mathcal{L}^{\Sigma}_{\text{cpa-R}}$:

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

EAVESDROP($m_L, m_R$):
    $c := \Sigma.\text{Enc}(k, m_R)$
    return $c$
```

The $\mathcal{L}_{\text{cpa\$-real}}$ library has been inlined, and the result is $\mathcal{L}^{\Sigma}_{\text{cpa-R}}$.

The sequence of hybrids shows that $\mathcal{L}^{\Sigma}_{\text{cpa-L}} \approx \mathcal{L}^{\Sigma}_{\text{cpa-R}}$, as desired. ■

7.3 CPA-Secure Encryption Based On PRFs

CPA security presents a significant challenge; its goals seem difficult to reconcile. On the one hand, we need an encryption method that is randomized, so that each plaintext m is mapped to a large number of potential ciphertexts. On the other hand, the decryption method must be able to recognize all of these various ciphertexts as being encryptions of m.

However, we have already seen a way to do this! In Chapter 6 we motivated the concept of a PRF with the following encryption technique. If Alice and Bob share a huge table T initialized with uniform data, then Alice can encrypt a plaintext m to Bob by saying something like “this is encrypted with one-time pad, using key #674696273” and sending $T[674696273] \oplus m$. Seeing the number 674696273 doesn’t help the eavesdropper know what $T[674696273]$ is. A PRF allows Alice & Bob to do the same encryption while sharing only a short key k. Instead of a the huge table T, they can instead use a PRF $F(k, \cdot)$ to derive a common pseudorandom value. Knowing a value r doesn’t help the adversary predict $F(k, r)$, when k is secret.

So, translated into more precise PRF notation, an encryption of m will look like $(r, F(k, r) \oplus m)$. Since Bob also has k, he can decrypt *any* ciphertext of this form by computing $F(k, r)$ and XOR’ing the second ciphertext component to recover m.

It remains to decide how exactly Alice will choose r values. We argued, informally, that as long as these r values don’t repeat, security is preserved. This is indeed true, and the distinctness of the r values is critical. Recall that there are 3 ways to avoid deterministic encryption, and all 3 of them would work here:

- In a **stateful** encryption, r could be used as a counter. Use $r = i$ to encrypt/decrypt the ith ciphertext.
- In a **randomized** encryption, choose r uniformly at random for each encryption. If the r values are long enough strings, then repeating an r value should be negligibly likely.
- In a **nonce-based** encryption, we can simply let r be the nonce. In the nonce-based setting, it is guaranteed that these values won’t repeat.

In this section we will show the security proof for the case of randomized encryption, since it is the most traditional setting and also somewhat more robust than the others.

The exercises explore how the nonce-based approach is more fragile when this scheme is extended in natural ways.

Construction 7.4 *Let F be a secure PRF with $in = \lambda$. Define the following encryption scheme based on F:*

```
$\mathcal{K} = \{0,1\}^\lambda$
$\mathcal{M} = \{0,1\}^{out}$
$\mathcal{C} = \{0,1\}^\lambda \times \{0,1\}^{out}$

KeyGen:
    $k \leftarrow \{0,1\}^\lambda$
    return $k$

Enc($k, m$):
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k,r) \oplus m$
    return $(r, x)$

Dec($k, (r,x)$):
    $m := F(k,r) \oplus x$
    return $m$
```

It is easy to check that the scheme satisfies the correctness property.

Claim 7.5 *Construction 7.4 has CPA\$ security (and therefore CPA security) if F is a secure PRF.*

The proof has more steps than other proofs we have seen before, and some steps are subtle. So let us use a Socratic dialogue to illustrate the strategy behind the proof:

Salviati: *The ciphertexts of Construction 7.4 are indistinguishable from uniform randomness.*

Simplicio: Salviati, you speak with such confidence! Do tell me why you say that these ciphertexts appear pseudorandom.

Salviati: *Simple! The ciphertexts have the form $(r, F(k,r) \oplus m)$. By its very definition, r is chosen uniformly, while $F(k,r) \oplus m$ is like a one-time pad ciphertext which is also uniformly distributed.*

Simplicio: Your statement about r is self-evident but $F(k,r) \oplus m$ confuses me. This does not look like the one-time pad that we have discussed. For one thing, the same k is used "every time," not "one-time."

Salviati: *I did say it was merely "like" one-time pad. The one-time pad "key" is not k but $F(k,r)$. And since F is a pseudorandom function, all its outputs will appear independently uniform (not to mention uncorrelated with their respective r), even when the same seed is used every time. Is this not what we require from a one-time pad key?*

Simplicio: I see, but surely the outputs of F appear independent only when its *inputs are distinct?* I know that F is deterministic, and this may lead to the same "one-time pad key" being used on different occasions.

Salviati: *Your skepticism serves you well in this endeavor, Simplicio. Indeed, the heart of your concern is that Alice may choose r such that it repeats. I say that this is negligibly likely, so that we can safely ignore such a bothersome event.*

Simplicio: Bothersome indeed, but why do you say that r is unlikely to repeat?

Salviati: *Oh Simplicio, now you are becoming bothersome! This value r is λ bits long and chosen uniformly at random each time. Do you not recall our agonizingly long discussion about the birthday paradox?*

Simplicio: Oh yes, now I remember it well. Now I believe I understand all of your reasoning: Across all ciphertexts that are generated, r is unlikely to repeat because of the birthday paradox. Now, provided that r never repeats, Alice invokes the PRF on distinct inputs. A PRF invoked on distinct inputs provides outputs that are uniformly random for all intents and purposes. Hence, using these outputs as one-time pads completely hides the plaintext. Is that right, Salviati?

Salviati: *Excellent! Now we may return to discussing the motion of the Sun and Earth . . .*

Look for Simplicio's final summary to be reflected in the sequence of hybrids used in the formal proof:

Proof We prove that $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}} \approx \mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$ using the hybrid technique:

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

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

CTXT($m$):
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k,r) \oplus m$
    return $(r, x)$
```

The starting point is $\mathcal{L}^{\Sigma}_{\text{cpa\$-real}}$. The details of Σ have been filled in and highlighted.

```
CTXT($m$):
    $r \leftarrow \{0,1\}^\lambda$
    $z :=$ LOOKUP($r$)
    $x := z \oplus m$
    return $(r, x)$
```

$\diamond$

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

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

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

```
CTXT($m$):
    $r \leftarrow \{0,1\}^\lambda$
    $z :=$ LOOKUP($r$)
    $x := z \oplus m$
    return $(r, x)$
```

$\diamond$

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

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

We have replaced $\mathcal{L}^{F}_{\text{prf-real}}$ with $\mathcal{L}^{F}_{\text{prf-rand}}$. From the PRF security of F, these two hybrids are indistinguishable.

At this point in the proof, it is easy to imagine that we are done. Ciphertexts have the form (r, x), where r is chosen uniformly and x is the result of encrypting the plaintext with what appears to be a one-time pad. Looking more carefully, however, the "one-time pad key" is $T[r]$ — a value that could potentially be used more than once if r is ever repeated!

As Simplicio rightly pointed out, a PRF gives independently random(-looking) outputs when called on *distinct inputs*. But in our current hybrid there is no guarantee that PRF inputs are distinct! Our proof must explicitly contain reasoning about why PRF inputs are unlikely to be repeated. We do so by appealing to the sampling-with-replacement lemma of Lemma 4.11.

We first factor out the sampling of r values into a subroutine. The subroutine corresponds to the $\mathcal{L}_{\text{samp-L}}$ library of Lemma 4.11:

```
ctxt(m):
    $r \leftarrow$ samp()
    $z :=$ lookup($r$)
    $x := z \oplus m$
    return $(r, x)$
```
◇
```
$\mathcal{L}^F_{\text{prf-rand}}$
$T :=$ empty

lookup($r$):
    if $T[r]$ undefined:
        $T[r] \leftarrow \{0,1\}^{out}$
    return $T[r]$
```
◇
```
$\mathcal{L}_{\text{samp-L}}$
samp():
    $r \leftarrow \{0,1\}^\lambda$
    return $r$
```

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

```
ctxt(m):
    $r \leftarrow$ samp()
    $z :=$ lookup($r$)
    $x := z \oplus m$
    return $(r, x)$
```
◇
```
$\mathcal{L}^F_{\text{prf-rand}}$
$T :=$ empty

lookup($r$):
    if $T[r]$ undefined:
        $T[r] \leftarrow \{0,1\}^{out}$
    return $T[r]$
```
◇
```
$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$

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

Inspecting the previous hybrid, we can reason that the arguments to lookup are *guaranteed* to never repeat. Therefore the $\mathcal{L}_{\text{prf-rand}}$ library can be greatly simplified. In particular, the if-condition in $\mathcal{L}_{\text{prf-rand}}$ is always true. Simplifying has no effect on the library's output behavior:

```
ctxt(m):
    $r \leftarrow$ samp()
    $z :=$ lookup($r$)
    $x := z \oplus m$
    return $(r, x)$
```
◇
```
lookup($r$):
    $t \leftarrow \{0,1\}^{out}$
    return $t$
```
◇
```
$\mathcal{L}_{\text{samp-R}}$
$R := \emptyset$

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

Now we are indeed using unique one-time pads to mask the plaintext. We are in much better shape than before. Recall that our goal is to arrive at a hybrid in which the outputs of ctxt are chosen uniformly. These outputs include the value r, but now r is *no longer being chosen uniformly!* We must revert r back to being sampled uniformly, and then we are nearly to the finish line.

```
CTXT(m):
    $r \leftarrow$ SAMP()
    $z :=$ LOOKUP($r$)
    $x := z \oplus m$
    return $(r, x)$
```
$\diamond$
```
LOOKUP(r):
    $t \leftarrow \{0,1\}^{out}$
    return $t$
```
$\diamond$
```
$\mathcal{L}_{\text{samp-L}}$
SAMP():
    $r \leftarrow \{0,1\}^{\lambda}$
    return $r$
```

As promised, $\mathcal{L}_{\text{samp-R}}$ has been replaced by $\mathcal{L}_{\text{samp-L}}$. The difference is indistinguishable due to Lemma 4.11.

```
CTXT(m):
    $r \leftarrow \{0,1\}^{\lambda}$
    $z \leftarrow \{0,1\}^{out}$
    $x := z \oplus m$
    return $(r, x)$
```

All of the subroutine calls have been inlined; no effect on the library's output behavior.

$\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$:
```
$\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$
CTXT(m):
    $r \leftarrow \{0,1\}^{\lambda}$
    $x \leftarrow \{0,1\}^{out}$
    return $(r, x)$
```

We have applied the one-time pad rule with respect to variables z and x, but omitted the very familiar steps (factor out, replace library, inline) that we have seen several times before. The resulting library is precisely $\mathcal{L}^{\Sigma}_{\text{cpa\$-rand}}$ since it samples uniformly from $\Sigma.\mathcal{C} = \{0,1\}^{\lambda} \times \{0,1\}^{out}$.

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

Exercises

7.1. Let Σ be an encryption scheme, and suppose there is a program $\mathcal{A}$ that recovers the key from a chosen plaintext attack. More precisely, $\Pr[\mathcal{A} \diamond \mathcal{L} \text{ outputs } k]$ is non-negligible, where $\mathcal{L}$ is defined as:

```
$\mathcal{L}$
$k \leftarrow \Sigma$.KeyGen

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

Prove that if such an $\mathcal{A}$ exists, then Σ does not have CPA security. Use $\mathcal{A}$ as a subroutine in a distinguisher that violates the CPA security definition.

In other words, CPA security implies that it should be hard to determine the key from seeing encryptions of chosen plaintexts.

7.2. Let Σ be an encryption scheme with CPA\$ security. Let Σ' be the encryption scheme defined by:

$$\Sigma'.\text{Enc}(k, m) = 00 \| \Sigma.\text{Enc}(k, m)$$

The decryption algorithm in Σ' simply throws away the first two bits of the ciphertext and then calls Σ.Dec.

(a) Does Σ' have CPA\$ security? Prove or disprove (if disproving, show a distinguisher and calculate its advantage).

(b) Does Σ' have CPA security? Prove or disprove (if disproving, show a distinguisher and calculate its advantage).

7.3. Suppose a user is using Construction 7.4 and an adversary observes two ciphertexts that have the same r value.

(a) What exactly does the adversary learn about the plaintexts in this case?

(b) How do you reconcile this with the fact that in the proof of Claim 7.5 there is a hybrid where r values are *never* repeated?

7.4. Construction 7.4 is a randomized encryption scheme, but we could also consider defining it as a **nonce-based** scheme, interpreting r as the nonce: $\text{Enc}(k, r, m) = (r, F(k, r) \oplus m)$. Formally prove that it is secure as a deterministic, nonce-based scheme. In other words, show that the following two libraries are indistinguishable, where Σ refers to Construction 7.4.

```
$k \leftarrow \Sigma.\text{KeyGen}$
$V := \emptyset$

EAVESDROP($v, m_L, m_R \in \Sigma.\mathcal{M}$):
    if $v \in V$: return err
    $V := V \cup \{v\}$
    $c := \Sigma.\text{Enc}(k, v, m_L)$
    return $c$
```

```
$k \leftarrow \Sigma.\text{KeyGen}$
$V := \emptyset$

EAVESDROP($v, m_L, m_R \in \Sigma.\mathcal{M}$):
    if $v \in V$: return err
    $V := V \cup \{v\}$
    $c := \Sigma.\text{Enc}(k, v, m_R)$
    return $c$
```

7.5. Let F be a secure PRP with blocklength $blen = \lambda$. Consider the following randomized encryption scheme:

```
$\mathcal{K} = \{0,1\}^\lambda$
$\mathcal{M} = \{0,1\}^\lambda$
$\mathcal{C} = (\{0,1\}^\lambda)^2$

KeyGen:
    $k \leftarrow \{0,1\}^\lambda$
    return $k$

Enc($k, m$):
    $v \leftarrow \{0,1\}^\lambda$
    $x := F(k, v \oplus m)$
    return $(v, x)$
```

(a) Give the decryption algorithm for this scheme.

(b) Prove that the scheme has CPA\$ security.

(c) Suppose that we interpret this scheme as a nonce-based scheme, where v is the nonce. Show that the scheme does **not** have nonce-based CPA security. The libraries for this definition are given in the previous problem.

Note: Even in the standard CPA libraries, v is given to the adversary and it is unlikely to repeat. However, in the nonce-based libraries the adversary can *choose* v, and this is what leads to problems.

7.6. Let F be a secure PRP with blocklength $blen = \lambda$. Show the the following scheme has pseudorandom ciphertexts:

```
$\mathcal{K} = \{0,1\}^\lambda$
$\mathcal{M} = \{0,1\}^\lambda$
$\mathcal{C} = (\{0,1\}^\lambda)^2$

KeyGen:
    $k \leftarrow \{0,1\}^\lambda$
    return $k$

Enc$(k, m)$:
    $s \leftarrow \{0,1\}^\lambda$
    $z := F(k, s \oplus m) \oplus m$
    return $(s \oplus m, z)$

Dec$(k, (r, z))$:
    return $F(k, r) \oplus z$
```

Hint: Rewrite Enc to include a new variable $r := s \oplus m$ and write the output in terms of r instead of s. You might then recognize a familiar face.

7.7. Let F be a secure PRP with blocklength $blen = \lambda$. Below are several encryption schemes, each with $\mathcal{K} = \mathcal{M} = \{0,1\}^\lambda$ and $\mathcal{C} = (\{0,1\}^\lambda)^2$. For each one:

- ▶ Give the corresponding Dec algorithm.
- ▶ State whether the scheme has CPA security. (Assume KeyGen samples the key uniformly from $\{0,1\}^\lambda$.) If so, then give a security proof. If not, then describe a successful adversary and compute its distinguishing bias.

(a)
```
Enc$(k, m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $z := F(k, m) \oplus r$
    return $(r, z)$
```

(b)
```
Enc$(k, m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $s := r \oplus m$
    $x := F(k, r)$
    return $(s, x)$
```

(c)
```
Enc$(k, m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k, r)$
    $y := r \oplus m$
    return $(x, y)$
```

(d)
```
Enc$(k, m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k, r)$
    $y := F(k, r) \oplus m$
    return $(x, y)$
```

(e)
```
Enc$(k, m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k, r)$
    $y := r \oplus F(k, m)$
    return $(x, y)$
```

(f)
```
Enc$(k, m)$:
    $s_1 \leftarrow \{0,1\}^\lambda$
    $s_2 := s_1 \oplus m$
    $x := F(k, s_1)$
    $y := F(k, s_2)$
    return $(x, y)$
```

★ (g)
```
Enc$(k, m)$:
    $r \leftarrow \{0,1\}^\lambda$
    $x := F(k, m \oplus r) \oplus r$
    return $(r, x)$
```

Hint: In all security proofs, you can use the PRP switching lemma (Lemma 6.7) to start with the assumption that F is a PRF.

7.8. Suppose F is a secure PRP with blocklength $n + \lambda$. Below is the encryption algorithm for a scheme that supports plaintext space $\mathcal{M} = \{0,1\}^n$:

```
Enc(k, m):
  $r \leftarrow \{0,1\}^\lambda$
  return $F(k, m \| r)$
```

(a) Describe the corresponding decryption algorithm.

(b) Prove that the scheme satisfies CPA$ security.

★ 7.9. Suppose F is a secure PRP with blocklength λ. Give the decryption algorithm for the following scheme and prove that it does **not** have CPA security:

```
$\mathcal{K} = \{0,1\}^\lambda$
$\mathcal{M} = \{0,1\}^{2\lambda}$
$\mathcal{C} = (\{0,1\}^\lambda)^3$

KeyGen:
  $k \leftarrow \{0,1\}^\lambda$
  return $k$

Enc($k, m_1 \| m_2$):
  $r \leftarrow \{0,1\}^\lambda$
  $s := F(k, r \oplus m_1)$
  $t := F\big(k, r \oplus m_1 \oplus F(k, m_1) \oplus m_2\big)$
  return $(r, s, t)$
```

★ 7.10. Suppose F is a secure PRP with blocklength λ. Give the decryption algorithm for the following scheme and prove that it satisfies CPA$ security:

```
$\mathcal{K} = (\{0,1\}^\lambda)^2$
$\mathcal{M} = \{0,1\}^\lambda$
$\mathcal{C} = (\{0,1\}^\lambda)^2$

KeyGen:
  $k \leftarrow \{0,1\}^\lambda$
  $r \leftarrow \{0,1\}^\lambda$
  return $(k, r)$

Enc($(k, r), m$):
  $s \leftarrow \{0,1\}^\lambda$
  $x := F(k, s)$
  $y := F(k, s \oplus m \oplus r)$
  return $(x, y)$
```

Hint: You may find it useful to divide the Enc algorithm into two cases by introducing an "if $m = r$" statement.

Note: If $r = 0^\lambda$ then the scheme reduces to Exercise 7.7 (f). So it is important that r is secret and random.

7.11. Let Σ be an encryption scheme with plaintext space $\mathcal{M} = \{0,1\}^n$ and ciphertext space $\mathcal{C} = \{0,1\}^n$. Prove that Σ cannot have CPA security.

Conclude that direct application of a PRP to the plaintext is not a good choice for an encryption scheme.

★ 7.12. In all of the CPA-secure encryption schemes that we'll ever see, ciphertexts are at least λ bits longer than plaintexts. This problem shows that such **ciphertext expansion** is essentially unavoidable for CPA security.

Let Σ be an encryption scheme with plaintext space $\mathcal{M} = \{0,1\}^n$ and ciphertext space $\mathcal{C} = \{0,1\}^{n+\ell}$. Show that there exists a distinguisher that distinguishes the two CPA libraries with advantage $\Omega(1/2^\ell)$.

Hint: As a warmup, consider the case where each plaintext has *exactly* 2^ℓ possible ciphertexts. However, this need not be true in general. For the general case, choose a random plaintext m and argue that with "good probability" (that you should precisely quantify) m has at most $2^{\ell+1}$ possible ciphertexts.

7.13. Show that an encryption scheme Σ has CPA security **if and only if** the following two libraries are indistinguishable:

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

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

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

CHALLENGE($m \in \Sigma.\mathcal{M}$):
  $m' \leftarrow \Sigma.\mathcal{M}$
  return $\Sigma.\text{Enc}(k, m')$
```

In plain language: if these libraries are indistinguishable, then encryptions of chosen plaintexts are indistinguishable from encryptions of random plaintexts. You must prove both directions!

7.14. Let Σ_1 and Σ_2 be encryption schemes with $\Sigma_1.\mathcal{M} = \Sigma_2.\mathcal{M} = \{0,1\}^n$.

Consider the following approach for encrypting plaintext $m \in \{0,1\}^n$: First, secret-share m using any 2-out-of-2 secret-sharing scheme. Then encrypt one share under Σ_1 and the other share under Σ_2. Release both ciphertexts.

(a) Formally describe the algorithms of this encryption method.

(b) Prove that the scheme has CPA security if **at least one of** $\{\Sigma_1, \Sigma_2\}$ has CPA security. In other words, it is not necessary that *both* Σ_1 and Σ_2 are secure. This involves proving two cases (assuming Σ_1 is secure, and assuming Σ_2 is secure).