

8.1 CPA-secure encryption

8.1.a Σ' CPA\$-secure?

We consider the following distinguisher:

Distinguisher A
pick $k \in \{0, 1\}^\lambda$
pick $m \in \{0, 1\}^\lambda$
 $y = \Sigma'(k, m)$
return $\text{first two bits of } y \stackrel{?}{=} 00$

First we will pick k and m at random. Then the return value of Σ' is checked whether the two starting bits are both 0 or not.

It is obvious that for $L_{CPA-real}^{\Sigma'}$ the distinguisher will always output 1. For $L_{CPA-rand}^{\Sigma'}$ the distinguisher will only output 1 if the random algorithm outputs a bit-string with the first two bits being 0. The probability for this will be $\frac{1}{4}$.

For the advantage we will then get:

$$\begin{aligned} \text{Bias}(A) &= | \Pr(A \diamond L_{CPA-real}^{\Sigma'}) - \Pr(A \diamond L_{CPA-rand}^{\Sigma'}) | \\ &= | 1 - \frac{1}{4} | \\ &= \frac{3}{4} \neq 0 \end{aligned}$$

Therefore Σ' is not CPA\$-secure.

8.1.b Σ' CPA-secure?

1. We will show that $\Sigma' \diamond \Sigma(k, m_L)$ is indistinguishable from $\Sigma' \diamond \Sigma(k, m_R)$:

$L_{CPA-L}^{\Sigma'}$
pick $k \in \{0, 1\}^\lambda$

EAVESDROP(m_L, m_R)
 if $|m_L| \neq |m_R|$
 then return ERROR
 $c := 00 \parallel \Sigma.Enc(k, m_L)$
return c

This is our starting library $L_{CPA-L}^{\Sigma'}$.

$L_{CPA}^{\Sigma'}$
pick $k \in \{0, 1\}^\lambda$

EAVESDROP(m_L, m_R)
 if $|m_L| \neq |m_R|$
 then return ERROR
 $c := 00 \parallel \Sigma.Enc(k, (m_L, m_R))$
return c

\diamond

L_{CPA-L}^{Σ}
 $\Sigma.Enc(k, (m_L, m_R))$
 if $|m_L| \neq |m_R|$
 then return ERROR
 $c := \Sigma.Enc(k, m_L)$
return c

With adding L_{CPA-L}^{Σ} we create a hybrid-library.

$L_{CPA}^{\Sigma'}$
pick $k \in \{0, 1\}^\lambda$
EAVESDROP(m_L, m_R)
 if $|m_L| \neq |m_R|$
 then return ERROR
 $c := 00 \parallel \Sigma.Enc(k, (m_L, m_R))$
return c

◇

L_{CPA-R}^{Σ}
Enc($k, (m_L, m_R)$)
 if $|m_L| \neq |m_R|$
 then return ERROR
 $c := \Sigma.Enc(k, m_R)$
return c

Because we know that Σ has CPA\$-security (especially also CPA-security) $L_{CPA-L}^{\Sigma} \approx L_{CPA-R}^{\Sigma}$

$L_{CPA-R}^{\Sigma'}$
pick $k \in \{0, 1\}^\lambda$
EAVESDROP(m_L, m_R)
 if $|m_L| \neq |m_R|$
 then return ERROR
 $c := 00 \parallel \Sigma.Enc(k, m_R)$
return c

Therefore we can inline this subroutine to end with the library $L_{CPA-R}^{\Sigma'}$. Therefore we showed that $L_{CPA-L}^{\Sigma'} \approx L_{CPA-R}^{\Sigma'}$.

8.2 From a PRP to CPA-secure encryption

8.2.1 Corresponding Decoder

8.2.1.a

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

$Dec(r, z)$:
 $c = z \oplus r$
 $m' = F^{-1}(c)$
return m'

8.2.1.b

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

$Dec(r, z)$:
 $r = F^{-1}(x)$
 $m' = r \oplus s$
return m'

8.2.1.c

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

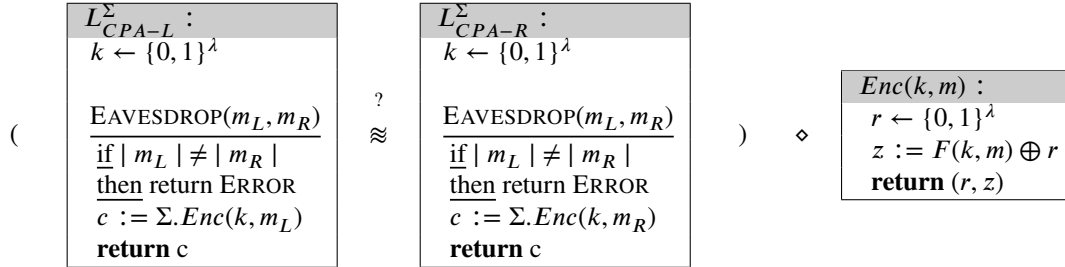
$Dec(r, z)$:
 $s_1 = F^{-1}(x)$
 $s_2 = F^{-1}(y)$
 $m' = s_2 \oplus s_1$
return m'

8.2.2 CPA-secure?

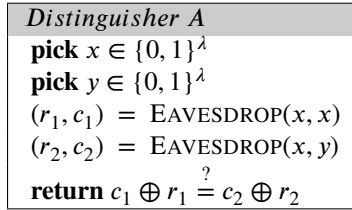
8.2.2.a

No, it is not *CPA-secure*, because:

Proof:



We will consider the following distinguisher:



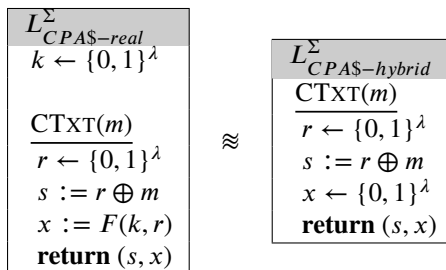
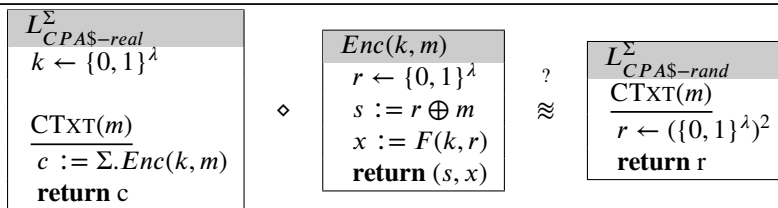
First we will pick two messages x and y at random. Then the return values of $\text{EAVESDROP}(m_L, m_R)$ are fetched. In the end the XOR-step which the encoder has done, will be reversed by using the return values. The distinguisher will therefore output whether both $F(k, m)$ have returned the same value.

It is obvious that if the L_{CPA-L}^{Σ} is used the adversary will output 1 because in both cases $F(k, x)$ is returned. If the L_{CPA-R}^{Σ} is used one will output $F(k, x)$ and the other one will output $F(k, y)$ which will not be the same and therefore the output value of A will be 0. So the adversary can distinguish between L_{CPA-L}^{Σ} and L_{CPA-R}^{Σ} and therefore this encoder is not *CPA-secure*.

8.2.2.b

Yes, it is *CPA-secure*, because:

Proof (We will show that the encoder has *CPA\$-security* and is therefore also *CPA-secure*):



Because we know that F is a secure PRP/PRF we can inline the subrou-tine of a random PRF.

$L_{CPA\$-hybrid}^\Sigma$
$CTXT(m)$
$r \leftarrow \{0, 1\}^\lambda$
$s := r \oplus m$
$x \leftarrow \{0, 1\}^\lambda$
return (s, x)

\approx

$L_{CPA\$-hybrid}^\Sigma$
$CTXT(m)$
$s \leftarrow \{0, 1\}^\lambda$
$x \leftarrow \{0, 1\}^\lambda$
return (s, x)

Because "XORing" a random bitstring with the message bitstring is also the same as generating a random bitstring, we can change this in the hybrid.

$L_{CPA\$-hybrid}^\Sigma$
$CTXT(m)$
$s \leftarrow \{0, 1\}^\lambda$
$x \leftarrow \{0, 1\}^\lambda$
return (s, x)

\approx

$L_{CPA\$-rand}^\Sigma$
$CTXT(m)$
$r \leftarrow (\{0, 1\}^\lambda)^2$
return r

Because s and x are making up a pair which is in C we can inline such a subroutine to end up with $L_{CPA\$-rand}^\Sigma$.

8.2.2.c

No, it is not *CPA-secure*, because if whether one of m_L and m_R is the 0-bit-string, the s_2 bit string will be the same as the s_1 bit string and therefore the return value x and y will be bitwise equal.

8.3 Modes of operation

8.3.1 CBC

It will affect the block which is corresponding with the encrypted block with the error as well as the following block.

8.3.2 OFB

It will only affect the block which is corresponding with the encrypted block with the error.

8.3.3 CTR

It will only affect the block which is corresponding with the encrypted block with the error.