15.4: Hybrid Encryption

As a rule, public-key encryption schemes are much more computationally expensive than symmetric-key schemes. Taking ElGamal as a representative example, computing $g^b$ in a cryptographically secure cyclic group is considerably more expensive than one evaluation of AES. As the plaintext data increases in length, the difference in cost between public-key and symmetric-key techniques only gets worse.

A clever way to minimize the cost of public-key cryptography is to use a method called hybrid encryption. The idea is to use the expensive public-key scheme to encrypt a temporary key for a symmetric-key scheme. Then use the temporary key to (cheaply) encrypt the large plaintext data.

To decrypt, one can use the decryption key of the public-key scheme to obtain the temporary key. Then the temporary key can be used to decrypt the main payload.

Construction 15.8: Hybrid Enc

Let $\Sigma_{pub}$ be a public-key encryption scheme, and let $\Sigma_{sym}$ be a symmetric-key encryption scheme, where $\Sigma_{sym} \subseteq \Sigma_{pub}$ — that is, the public-key scheme is capable of encrypting keys of the symmetric-key scheme.

Then we define $\Sigma_{hyb}$ to be the following construction:
Importantly, the message space of the hybrid encryption scheme is the message space of the symmetric-key scheme (think of this as involving very long plaintexts), but encryption and decryption involves expensive public-key operations only on a small temporary key (think of this as a very short string).

The correctness of the scheme can be verified via:

\[
\text{Dec}(sk, \text{Enc}(pk, m)) = \text{Dec}(sk, \alpha(\text{Enc}(pk, tk), \text{Enc}(tk, m))) \\
= \text{Dec}(sk, \text{Dec}(sk, \text{Enc}(pk, tk)), \text{Enc}(tk, m)) \\
= \text{Dec}(tk, \text{Enc}(tk, m)) \\
= m.
\]

To show that hybrid encryption is a valid way to encrypt data, we prove that it provides CPA security, when its two components have appropriate security properties:

**Claim 15.9**

If \( \Sigma_{\text{sym}} \) is a one-time-secret symmetric-key encryption scheme and \( \Sigma_{\text{pub}} \) is a CPA-secure public-key encryption scheme, then the hybrid scheme \( \Sigma_{\text{hyb}} \) ([Construction 15.8](https://eng.libretexts.org/Under_Construction/Book%3A_The_Joy_of_Cryptography_(Rosulek)/Chapter_15%3A_Public-Key_…)) is also a CPA-secure public-key encryption scheme.

Note that \( \Sigma_{\text{sym}} \) does not even need to be CPA-secure. Intuitively, one-time secrecy suffices because each temporary key \( tk \) is used only once to encrypt just a single plaintext.

**Proof**

As usual, our goal is to show that \( \mathcal{L}_{\Sigma_{\text{hyb}}}^{\text{pk-cpa-L}} \equiv \mathcal{L}_{\Sigma_{\text{hyb}}}^{\text{hybpk-cpa-R}} \), which we do in a standard sequence of hybrids:

The starting point is \( \mathcal{L}_{\text{pk-cpa-L}} \), shown here with the details of \( \Sigma_{\text{hyb}} \) filled in.

Our only goal is to somehow replace \( m_L \) with \( m_R \). Since \( m_L \) is only used as a plaintext for \( \Sigma_{\text{sym}} \), it is tempting to simply apply the one-time-secrecy property of \( \Sigma_{\text{sym}} \) to argue that \( m_L \) can be replaced with \( m_R \). Unfortunately, this cannot work because the key used for that ciphertext is \( tk \), which is used elsewhere. In particular, it is used as an argument to \( \Sigma_{\text{pub. Enc}} \).
However, using \( tk \) as the plaintext argument to \( \Sigma_{\text{pub}}.\text{Enc} \) should hide \( tk \) to the calling program, if \( \Sigma_{\text{pub}} \) is CPA-secure. That is, the \( \Sigma_{\text{pub}} \) encryption of \( tk \) should look like a \( \Sigma_{\text{pub}} \) encryption of some unrelated dummy value. More formally, we can factor out the call to \( \Sigma_{\text{pub}}.\text{Enc} \) in terms of the \( \mathcal{L}_{\text{pk-cpa-L}} \) library, as follows:

Here we have changed the variable names of the arguments of \( \text{CHALLENGE}' \) to avoid unnecessary confusion. Note also that \( \text{CHALLENGE} \) now chooses two temporary keys — one which is actually used to encrypt \( m_L \) and one which is not used anywhere. This is because syntactically we must have two arguments to pass into \( \text{CHALLENGE}' \).

Now imagine replacing \( \mathcal{L}_{\text{pk-cpa-L}} \) with \( \mathcal{L}_{\text{k-cpa-R}} \) and then inlining subroutine calls. The result is:

At this point, it does now work to factor out the call to \( \Sigma_{\text{sym}}.\text{Enc} \) in terms of the \( \mathcal{L}_{\text{ots-L}} \) library. This is because the key \( tk \) is not used anywhere else in the library. The result of factoring out in this way is:

At this point, we can replace \( \mathcal{L}_{\text{ots-L}} \) with \( \mathcal{L}_{\text{ots-R}} \). After this change the \( \Sigma_{\text{sym}} \)-ciphertext encrypts \( m_R \) instead of \( m_L \). This is the “half-way point” of the proof, and the rest of the steps are a mirror image of what has come before. In summary: we inline \( \mathcal{L}_{\text{ots-R}} \), then we apply CPA security to replace the \( \Sigma_{\text{pub}} \)-encryption of \( tk' \) with \( tk \). The result is exactly \( \mathcal{L}_{\text{pk-cpa-R}} \), as
desired.