University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 42

1 Public-Key Encryption in the RO Model

So far, we have seen examples of (very) efficient signature schemes in the so-called random oracle (RO) model. Can we use the RO model to construct efficient public-key encryption schemes as well?

As motivation, we remind the reader that the most efficient public-key encryption scheme based on trapdoor permutations (e.g., RSA) in the standard model works as follows: the receiver generates a key key and associated trapdoor td for a trapdoor permutation. The public key is key and the secret key is simply td. There is also a (publicly-known) hard-core bit $h : \mathcal{D}_{\mathsf{key}} \rightarrow \{0,1\}$ for the trapdoor permutation. To encrypt the single bit b the sender chooses a random r, computes $C_1 = f_{\mathsf{key}}(r)$ and $C_2 = h(r) \oplus b$, and sends C_1, C_2. To decrypt, the receiver first recovers r from C_1 (using the trapdoor) and then recovers the message by computing $b = h(r) \oplus C_2$. Note that this requires one evaluation of f for each bit of the original message. In class we also discussed encryption schemes which reduce the ciphertext length (for longer messages) but these schemes have the same *computational* efficiency as above (i.e., one evaluation of f per bit of original message).

Unfortunately, we can essentially do no better than this[1] using arbitrary trapdoor permutations in the standard model. It is also not known whether it is possible to do substantially better using, e.g., RSA in the standard model. This has motivated researchers to consider what sort of efficiency might be possible in the random oracle model.

We now present a simple and very efficient scheme. As before, let $(\mathcal{K}, f, \mathsf{Inv})$ be a trapdoor permutation family, and let $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ be a random oracle. The scheme works as follows:

- To generate keys, the receiver runs $\mathcal{K}$ to generate public key key and secret key the associated trapdoor td.
- To encrypt message $m \in \{0,1\}^\ell$, the sender chooses random $r \in \mathcal{D}_{\mathsf{key}}$ and computes $C_1 = f_{\mathsf{key}}(r)$. The sender also computes $C_2 = H(r) \oplus m$ and sends C_1, C_2.
- To decrypt ciphertext C_1, C_2, the receiver computes (using the trapdoor) $r = f_{\mathsf{key}}^{-1}(C_1)$ and then obtains the message as $C_2 = H(r) \oplus C_2$. (Recall that H is *publicly accessible* by the receiver, any possible senders, and the adversary as well.)

It should be clear that this scheme has correct decryption. The parallel with the encryption scheme in the standard model (given previously) should also be obvious. Finally, the scheme

[1] For those who are interested, I will point out that there are methods with slightly better efficiency but even these do not yield a truly practical scheme.

is fairly efficient: if $\ell = 160$ then up to 160 bits of the message can be encrypted using only a single evaluation of f. (Note also that encrypting 160 bits is enough to allow the use of hybrid encryption for messages longer than 160 bits, since the "private key" set up need only be 128 bits long anyway.)

Before giving the proof of security for the above scheme, we give some intuition and a caveat. For the intuition as to why the scheme is secure, note that when an adversary sees ciphertext C_1, C_2 (encrypted under public key key) the adversary has no information at all about the underlying message unless it knows $H(r)$ (where $r \stackrel{\text{def}}{=} f_{\mathsf{key}}^{-1}(C_1)$). And because we model H as a random oracle, the adversary has no information at all about $H(r)$ unless it has explicitly queried H at point r. However, to do so the adversary would have had to completely determine r from C_1 which is equivalent to inverting the trapdoor permutation. Since the occurs with only small probability, the scheme is hence secure (we give a more formal proof below).

The caveat of the construction (and the proof) is that *there is no "real world" hash function for which the above construction is provably secure.* The construction can *only* be proven secure when H is treated as a random oracle. In practice we may not worry about such things since no actual attack against the scheme is known when using common hash functions for H. But from a theoretical point of view this state of affairs is a little unsettling.

Theorem 1 *If $(\mathcal{K}, f, \mathsf{Inv})$ is a (t, ϵ)-secure trapdoor permutation, then the above construction of a public-key encryption scheme is (t, ϵ)-secure in the sense of indistinguishability in the random oracle model.*

Proof Assume we have an adversary A and messages m_0, m_1 for which:

$$\begin{aligned}|\Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_1) : A(\mathsf{key}, C_1, C_2) = 1] \\ - \Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_0) : A(\mathsf{key}, C_1, C_2) = 1]| = \delta\end{aligned}$$

(where the probabilities are taken over randomly generated public key key). We show how to construct an adversary A' breaking the security of the trapdoor permutation. A' will be given a random key key and a random element y, and will try to find $f_{\mathsf{key}}^{-1}(y)$.

$A'(\mathsf{key}, y)$
Set $C_1 = y$
$C_2 \leftarrow \{0,1\}^\ell$
Run $A(\mathsf{key}, C_1, C_2)$, answering H-oracle queries of A as follows:
On the i^{th} query x_i, check whether $f_{\mathsf{key}}(x_i) = y$; if so, output x_i and stop
Otherwise, simply choose random $y_i \leftarrow \{0,1\}^\ell$ and return y_i

(We assume without loss of generality that A never asks the same H-oracle query twice; note that it gains nothing by doing so.) Clearly, if A' outputs anything then it succeeds in finding $f_{\mathsf{key}}^{-1}(y)$. And the probability that A' outputs anything is exactly the probability that A ever queries H on input $f_{\mathsf{key}}^{-1}(y)$. We may also note that A' provides a perfect simulation for A (up until the point — if any — that A' stops): the output of H is independently and uniformly distributed, and the ciphertext and public key have the required distribution (C_1

is equal to $f_{\mathsf{key}}(r)$ for some random [unknown] r and C_2 is uniformly distributed which is fine as long as A has not yet queried $H(r)$).

Let Find be the event that A ever explicitly queries $H(r)$, where as above we have $r \stackrel{\text{def}}{=} f_{\mathsf{key}}^{-1}(C_1)$. Note that, by what we said above, the probability that A' inverts y is exactly $\Pr[\mathsf{Find}]$; thus, we must have $\Pr[\mathsf{Find}] \leq \epsilon$ by the security of the trapdoor permutation. Conditioning on event Find gives:

$$\begin{aligned}
&\Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_1) : A(\mathsf{key}, C_1, C_2) = 1] \\
&\quad = \Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_1) : A(\mathsf{key}, C_1, C_2) = 1 | \mathsf{Find}] \cdot \Pr[\mathsf{Find}] \\
&\qquad + \Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_1) : A(\mathsf{key}, C_1, C_2) = 1 | \overline{\mathsf{Find}}] \cdot \Pr[\overline{\mathsf{Find}}],
\end{aligned}$$

with a similar expression for the case of encrypting m_0. The key point is:

$$\begin{aligned}
&\Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_1) : A(\mathsf{key}, C_1, C_2) = 1 | \overline{\mathsf{Find}}] \\
&\qquad = \Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_0) : A(\mathsf{key}, C_1, C_2) = 1 | \overline{\mathsf{Find}}],
\end{aligned}$$

where (as discussed in the paragraph preceding the proof) this is because H is a random oracle and so the adversary has no information about $H(r)$ as long as it has never explicitly queried $H(r)$ (and therefore $m_0 \oplus H(r)$ is equally distributed to $m_1 \oplus H(r)$ just as in the case of the one-time pad). Putting everything together gives:

$$\begin{aligned}
&\delta \\
&\quad = |\Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_1) : A(\mathsf{key}, C_1, C_2) = 1 | \mathsf{Find}] \Pr[\mathsf{Find}] \\
&\qquad - \Pr[(C_1, C_2) \leftarrow \mathcal{E}_{\mathsf{key}}(m_0) : A(\mathsf{key}, C_1, C_2) = 1 | \mathsf{Find}] \Pr[\mathsf{Find}]| \\
&\quad \leq \Pr[\mathsf{Find}] \\
&\quad \leq \epsilon.
\end{aligned}$$

This completes the proof. ■

2 Identification Protocols

We have now finished with our coverage of the essential cryptographic primitives: encryption and message authentication in the private-key setting, and encryption and digital signatures in the public-key setting. However, these are not the *only* uses and applications of cryptography. Often they form important building blocks for larger and more complex protocols. We briefly example one such application here.

Consider the problem of secure identification. Here, we will assume a prover $\mathcal{P}$ who wants to prove his identity to some verifier $\mathcal{V}$. We imagine that $\mathcal{P}$ and $\mathcal{V}$ at some point securely establish a common, secret key k. At some later point in time, $\mathcal{P}$ (who is communicating over a network) wants to convince $\mathcal{V}$ that it is indeed him. (In fact, this sort of thing occurs all the time: consider what you do when you log on to your email account.) What sort of security might we need and how might we design a secure protocol for this task?

One immediate requirement is that an adversary (who does not know the secret key shared by $\mathcal{P}$ and $\mathcal{V}$) should be unable to impersonate $\mathcal{P}$ (that is, to falsely convince $\mathcal{V}$ that it is speaking with $\mathcal{P}$). This property should at a minimum hold when the adversary has never observed any interactions between $\mathcal{P}$ and $\mathcal{V}$. It is very easy to come up with a protocol meeting this definition of security: the protocol consists simply of $\mathcal{P}$ sending the key k to $\mathcal{V}$; $\mathcal{V}$ then checks whether this key matches the one it has previously shared with $\mathcal{P}$. Of course, our gut feeling is that such a protocol should not be considered secure! Indeed, if an adversary observes a single interaction between $\mathcal{P}$ and $\mathcal{V}$, then the adversary learns k and can later impersonate $\mathcal{P}$ at any later point in time! Thus, although the level of security mentioned above is certainly necessary, it is definitely *not* sufficient.

Instead, we need to additionally require that an adversary should be unable to impersonate $\mathcal{P}$ *even if the adversary can observe (eavesdrop on) as many executions of the protocol between* $\mathcal{P}$ *and* $\mathcal{V}$ *as it likes.* It should be clear from the discussion above that the preceding protocol does *not* satisfy this notion of security.

How can we now design a protocol secure under this definition? One suggestion is the following: let $(\mathcal{E}, \mathcal{D})$ be a private-key encryption scheme. To identify $\mathcal{P}$, the verifier $\mathcal{V}$ chooses a random value v and sends it to $\mathcal{P}$. The prover $\mathcal{P}$ encrypts v using k to obtain ciphertext C (i.e., computes $C \leftarrow \mathcal{E}_k(v)$) and sends C as its reply. The verifier decrypts C and checks whether the resulting decryption is equal to v; only then does it accept. Is such a protocol secure? More precisely, can we give natural conditions on the encryption scheme such that *any* encryption scheme satisfying those conditions will make this a secure identification protocol?

We show here that although the protocol may seem secure, such appearances are deceiving. In particular, even an encryption scheme secure in the sense of indistinguishability is *not* enough to make the above a secure identification protocol. This shows the importance of rigorous proofs of security and of not letting "intuition" be our guide for accepting constructions as secure. Recall the encryption scheme in which F is a PRF and a message m is encrypted by choosing a random r and sending $\langle r, F_k(r) \oplus m \rangle$. Recall further that we proved this scheme secure in the sense of indistinguishability. But consider now what happens when we use this scheme as part of the identification scheme sketched above.

When the adversary eavesdrops on a single execution of the protocol, the adversary learns v and the response $\langle r, C \rangle$. We claim this is enough to allow the adversary to impersonate $\mathcal{P}$ in the future via the following attack: When $\mathcal{V}$ later sends v' to the adversary, the adversary responds with $\langle r, C \oplus v \oplus v' \rangle$. You can check that $\mathcal{V}$ will always accept. Thus, the identification scheme given above is *not* secure when using an encryption scheme satisfying any definition of security we have seen thus far.

In fact, we might notice that there is really no reason to be using encryption at all. We do not care about secrecy here; all we care about is authentication. Thus, it is more natural to consider using a message authentication scheme as our building block for constructing a secure identification protocol; we do this as follows: Let $(\mathsf{MAC}, \mathsf{Vrfy})$ be a secure message authentication scheme (for polynomially-many messages). The protocol starts by having $\mathcal{V}$ choose a random value v and send it to $\mathcal{P}$. The prover $\mathcal{P}$ than computes $\mathsf{tag} = \mathsf{MAC}_k(v)$ and sends tag to $\mathcal{V}$. Finally, the verifier accepts if and only if $\mathsf{Vrfy}_k(v, \mathsf{tag}) = 1$.

We leave it to the reader to verify that as long as v is chosen from a large enough space, this identification protocol is secure.