Last Revised: 3/3/96.

ECS 227: Modern Cryptography
Lecture 15

Plaintext Awareness and Digital Signatures

Phillip Rogaway (Transcribed by John Black)
29 February 1996

In the last lecture Phil introduced the concept of "malleability." We saw that a desirable property of an encryption scheme is that the ciphertext could not be manipulated by an adversary to produce ciphertext of some related plaintext message. We then were briefly introduced to the concept of *Plaintext Awareness* and were given an intuitive sense of what this concept means. Today's lecture picked up with this topic and completed it. We were then introduced to *Digital Signatures*, which will be continued in Lecture 16.

10.8 Plaintext Awareness

As we saw last time, an encryption scheme is "plaintext-aware" if, for any ciphertext it produces, it must "know" the corresponding plaintext. This is a property which one might assume follows naturally in any encryption scheme, but—as it turns out—is a property which is rather rarely achieved by encryption schemes.

We're now going to develop the idea of making a "plaintext extractor." The idea is as follows: if we can figure out the plaintext by watching an encryption engine's interaction with its environment, but *without* looking inside the engine itself, then the engine is plaintext-aware.

So, let's assume there is some black-box B which encrypts x and produces y. If we were to observe B at work, just as a black box, what external interactions would we see?

- Calls to oracles G and H would be observed (along with their associated inputs and outputs).
- The security parameter 1^k might be given to B.
- The encryption function $\mathcal{E}$ would be given to B.

- A random string \$ would be given to B.
- The output y which B produced would emerge from B.

Suppose, just observing the above interactions, we could construct an algorithm (the extractor) which would determine x where $\mathcal{E}(x) = y$. If there's no x such that $\mathcal{E}(x) = y$, the algorithm would say so. The existence of such an algorithm is, intuitively, what it means for the encryption scheme to be plaintext-aware. The adversary could (if she wanted to) do whatever the extractor was doing since, undenyable, she has access to the same things which the extractor does. Doing this, the adversary would "know" the cleartext for any ciphertext she produces.

Now we develop more formally our concept of plaintext awareness:

Definition 10.8.1 *A (t, ϵ)-*plaintext extractor*, K, for adversary B and encryption scheme $\mathcal{G}(1^k)$ satisfies*

$$\Pr\Big[(\mathcal{E}, \mathcal{D}) \leftarrow \mathcal{G}(1^k);\ G, H \leftarrow \Omega;\ (y, \tau) \leftarrow \mathsf{Run}\ B^{G,H}(1^k, \epsilon);$$
$$x \leftarrow K(1^k, \mathcal{E}, y, \tau) :\ D(y) \neq x\Big] \ \leq\ \epsilon$$

where K runs in $\leq t$ steps, and where τ is a "history" of B's conversation with oracles G and H, encoded as a list of inputs and outputs:

$$\begin{array}{lcl} (g_1, G(g_1)), & \ldots & , (g_{q_G}, G(g_{q_G})) \\ (h_1, H(h_1)), & \ldots & , (h_{q_H}, H(h_{q_H})) \end{array}$$

to oracles G and H, respectively, together with B's output, y.

This definition can also be found in the Bellare-Rogaway paper, "Optimal Asymmetric Encryption—How to Encrypt with RSA," page 9.

In the above definition we didn't allow for y to be the encryption of "no valid string." You can either modify the definition to allow this, or simply define any such y to be an alternative encryption of some particular string, say the 0-string.

So, what does this definition mean? If we're given some encryption function $\mathcal{E}$ and a history of B's conversation with oracles G and H in the form of τ, then we can extract plaintext x given ciphertext y more than $1 - \epsilon$ of the time by running an algorithm K. Note that B's coins weren't given to K: the notion becomes stronger by their omission, and we wouldn't have been terribly happy had they been needed, since the coins may model randomness which is not, in truth, externally visible to the knowledge extractor. Also, the knowledge extractor wasn't given oracle access to G and H— knowledge extractor$K$ must find the plaintext using only the the "piece" of these oracles that appears in B's conversations with her oracles.

Now let's look at our friend OAEP. The picture drawn many times on the board is too difficult to reproduce here, but we all know it by now: given plaintext x with length n, we generate a random r with length k_0. Then we compute $s = x \oplus G(r)$ and $t = r \oplus H(s)$. Finally, our ciphertext is $f(s\ .\ t)$.

But this scheme is clearly *not* plaintext-aware: any bit string is a valid possible output, which indicates that B could trivially generate ciphertext for which it does *not* know the corresponding plaintext. To do this B could just output a random string.

10.9 Making OAEP Plaintext-Aware

Fortunately, all is not lost. We can make OAEP plaintext-aware by a simple change in how we encrypt. Let's simply pad x with k_1 0-bits before running the rest of the OAEP encryption algorithm (call this added string ρ, for "redundancy"). We dub this augmented OAEP algorithm "OAEP/redundancy."

Now the corresponding decrypt algorithm will change slightly: get s and t, then compute $r = H(s) \oplus t$ and $x \parallel \rho$ is defined by $G(r) \oplus s$. If ρ is all 0's, we return x as the decryption, otherwise we return 0 as the decryption, to match the convention mentioned above. (Alternatively, we could have modified the decryption routine to return an indication, $*$, that we don't regard y as the encryption of any string at all.)

Let's again consider the adversary B which generates random strings. Does B know the plaintext for any ciphertext she generates? Yes; almost every random string will have plaintext of 0 (or $*$) because most random strings will not yield a value of ρ which is all zeroes. (This, of course, depends on the size of k_1, but we can make k_1 as large as we wish.)

Since the scheme seems to be plaintext-aware, let's see if we can't develop a plaintext-extractor K which satisfies the definition given above. We'll outline the strategy here; for those interested in a proof of the strategy's correctness, see the Bellare-Rogaway paper.

10.10 A Plaintext-Extractor for OAEP/redundancy

K is fed $(1^k, \mathcal{E}, y, \tau)$ (here τ is the conversation history with oracles G and H). We can simply try every pair of inputs and outputs seen for oracles G and H. In other words, try all $(g, G(g))$ and $(h, H(h))$ pairs. (This is clearly going to cost some time: there are q_G calls to G and q_H calls to H, so there will be $q_G q_H$ pairs to try.)

Now, for any such $(g, G(g))$, $(h, H(h))$, let x be the first n bits of $G(g) \oplus h$ and let ρ be the last k_1 bits of $G(g) \oplus h$. Let r simply be g and let w be $h \parallel H(h) \oplus y$. (Drawing the picture of OAEP/redundancy and writing in where $G(g)$ would replace $G(r)$ and where h replaces s helps quite a bit to see that w in fact produces a candidate output here.)

Now if $f(w) = y$ and ρ is all 0's, then we've found our pre-image and we output x. If we try all $(g, G(g))$ pairs and all $(h, H(h))$ pairs and do not produce a pre-image, we output 0. The following theorem formalizes that this works (it can also be found in Bellare-Rogaway along with a proof which will not be reproduced here).

Theorem 10.10.1 *Let $\mathcal{G}$ be OAEP/redundancy for $[\mathcal{F}, k, k_0, k_1, n]$. Let B be a (t, q_G, q_H)-adversary. Then there is a plaintext-extractor K that (t, ϵ)-recovers the plaintext, where*

$$\begin{aligned} t' &= t + q_G q_H (\mathsf{Time}_{\mathcal{F}}(1^k) + O(k)) \\ \epsilon' &= q_G \cdot 2^{-k_0} + 2^{-k_1} \end{aligned}$$

11 Digital Signatures

In the last section we saw an assumption one might make about encryption schemes which may not necessarily hold. In this section we introduce another idea which is tempting to assume, but may not necessarily hold for a given encryption scheme: that an encrypted message (especially one that "makes sense") necessarily originates from the party who claims to have sent it.

In a *digital signature scheme*, each *signer* S has a pair of keys: a *public key* pk and a secret key sk. Party S wants to "sign" messages in such a way that any recipient can verify that each message was in fact generated by S. To accomplish this, S signs the message by producing a *signature* $\sigma = Sign_{sk}(M)$, and then transmitting (M, σ) to some *verifier.* The verifier wishes to establish whether the received message truly originated with S. To do this he runs $Verify_{pk}(M, \sigma)$. If this computes 1, then the message is deemed authentic and the verifier accepts it; otherwise the message is considered to be a forgery, and the verifier rejects it.

Because this scheme uses a public key and a private key, it's common to think of it as being related to an encryption scheme, such as $\mathcal{RSA}$. But this isn't really correct. Digital signatures are a completely different idea. To illustrate this we now show how to sign a bit with DES, and we show an example of a signature scheme directly based on $\mathcal{RSA}$, but which turns out to be bogus.

11.1 Signing a Bit with DES

We, the sender, want to transmit a bit to some verifier, and we want the verifier to be able to ensure that the bit really did originate from us. Here is a scheme which uses DES to accomplish this: randomly choose two DES keys k_0 and k_1 from $\{0,1\}^{56}$. Now set $sk = (k_0, k_1)$ and $pk = \langle \mathrm{DES}_{k_0}(0^{64}), \mathrm{DES}_{k_1}(0^{64}) \rangle$. Now if we wish to sign 0, we use a signature of $\sigma = k_0$, and if we wish to sign 1, we use a signature of $\sigma = k_1$. The verifier receives bit b and signature $sigma$ and decrypts the first half of the signers' public key if $b = 0$, and the second half of the signer's public key if $b = 1$. If this yields a plaintext of 0^{64}, the verifier accepts the signature. If it yields anything else, the verifier rejects the signature.

Of course this scheme is somewhat inefficient: we use a 128-bit public key to transmit a single bit! And it is not so clear how to adapt this scheme to allow the signing of multiple, possibly long messages.

11.2 Using RSA for Digital Signatures

Let's consider another attempt at a signature scheme, where we try to directly us $\mathcal{RSA}$ to implement signatures. Let $pk = (N, e)$ and $sk = (N, d)$ be the public and secret keys, respectively, as determined by $\mathcal{RSA}$. Now we sign and verify as follows:

$$\begin{aligned} \mathit{Sign}_{N,d}(M) &= M^d \bmod N \\ \mathit{Verify}_{N,e}(M, \sigma) &= \text{if } \sigma^e \bmod N = M \text{ then } \mathsf{valid} \text{ else } \mathsf{invalid} \end{aligned}$$

Here we assume that the messages we want to sign, M, satisfy $M \in \mathsf{Z}_N^*$.

Now we can generate signed messages $M_1, M_2, \ldots$ by applying the signature function:

$$\begin{aligned} M_1 &\rightarrow M_1^d \bmod N \\ M_2 &\rightarrow M_2^d \bmod N \\ &\vdots \end{aligned}$$

But notice here that we can create a *new* authenticated message which is different from all messages previously seen! We know that $M_1 M_2 \bmod N \rightarrow \sigma_1 \sigma_2 \bmod N$. This is clearly an undesirable property for a signature scheme to have: an adversary can produce new signed messages using a collection of previous signed messages. This amounts to a form of forgery.

In general, a forger's goal is to find (M^*, σ^*) such that:

$$\begin{aligned} \mathsf{Verify}_{pk}(M^*, \sigma^*) = 1 \\ M^* \notin \{M_1, M_2, \cdots\} \end{aligned}$$

where the M_is are the previously signed messages. In other words, the forger must find a message and a signature such that the verify function validates it and the message is different from all previously seen messages.

The lecture ended here; we will probably continue talking about signatures in the next lecture.