6.875J/18.425J Cryptography and Cryptanalysis September 10, 2001

Lecture 2

Instructor: Anna Lysyanskaya *Scribed by: Susan Hohenberger*

1 The Public-Key Setting

Recall the scenario (lecture 1, section 1) where there are two people, Alice and Bob, and Bob wants to send a love-letter to Alice. Unfortunately, there is an evesdropper, Eve, who also wants to read the love-letter. In lecture 1, we discussed using a public-key cryptosystem as a technique for Bob to send his love-letter to Alice without Eve being able to understand it. In this lecture we will discuss the definition of such a cryptosystem in more detail.

A public-key cryptosystem has the following setup:

1. Eve knows everything Bob knows, except the message m.[1]

2. Eve has limited computational power, more specifically Eve is a probabilistic, polynomial-time algorithm. Recall the discussion in lecture 1, section 4 for why this is a reasonable assumption.

3. Alice has a secret key (SK) and a corresponding public key (PK).

Therefore, the question is: *can a public-key cryptosystem be created that meets these requirements?* Unfortunately, no one knows for sure. If it is possible, then that would prove that P $\neq$ NP. Since the P vs. NP problem is still unsolved, we formulate our cryptosystem around assumptions about the difficulty of computing certain things in reasonable time.

2 Notation

A denotes an algorithm.

$A(\cdot)$ denotes an algorithm with an input.

$A(\cdot,\cdot)$ denotes an algorithm with two inputs.

$A(x)$ is well-defined if A is a deterministic algorithm.

$A(x)$ defines a probability distribution if A is a probabilistic algorithm. More precisely, let $A'(x,R)$ be the deterministic algorithm that represents the behavior of A on input x and random tape R. The uniform distribution on R induces a probability distribution on the output of $A'(x,R)$. We call that distribution $A(x)$.

$x \leftarrow A(y)$ denotes that x is a sample from the distribution $A(x)$; it also means that x was generated by running A on input y.

$x \leftarrow F$, for a set F, means that x was selected from F uniformly at random.

[1]Also, note that it is impossible for anyone besides Bob to know the randomness Bob used during encryption. For example, say Bob flips a coin during encryption and applies the result of the coin toss to the encryption algorithm. Such randomness is not visible to Eve since she is not allowed to watch Bob perform the encryption or the whole game is up.

We can now introduce notation that represents a sequence of experiments, for example

$$((x, y) \leftarrow A(3); z \leftarrow B(y))$$

denotes that first we ran algorithm A on input 3 to generate (x, y), and then we ran B on input y to generate z.

$(\{x_i \leftarrow A_i(y_i)\} 1 \leq i \leq n \ : \ \{x_i\}, 1 \leq i \leq n)$ denotes the distribution on the values $\{x_i\}$ that were obtained by running algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$.

Let p be a boolean function. $(\{x_i \leftarrow A_i(y_i)\} 1 \leq i \leq n \ : \ p(x_n))$ means that $p(x_n)$ is TRUE after the value x_n was obtained by running algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$.

Finally, the statement

$$\Pr[\{x_i \leftarrow A_i(y_i)\} 1 \leq i \leq n \ : \ p(x_n)] = \alpha$$

means that the probability that $p(x_n)$ is TRUE after the value x_n was obtained by running algorithms $A_1, \ldots, A_n$ on inputs $y_1, \ldots, y_n$, is α.

3 A Specification for a Cryptosystem

Let us first give specification of the algorithms that are part of the cryptosystem. It is important that these algorithms are *efficient*. As is usually the case in theoretical computer science, by "efficient," we mean polynomial-time. But polynomial-time in what? An important parameter of a cryptosystem is its *security parameter* k. All the algorithms that comprise the cryptosystem must have run-time polynomial in k. The adversaries against which we hope to achieve security, also have run-time polynomial in this security parameter.

Key generation The key generation algorithm, denoted by G, is a probabilistic polynomial-time algorithm that takes as its sole input the security parameter k. However, since traditionally we say that an algorithm is polynomial-time if its running time is polynomial in the size of its input, G's input k will be encoded in unary, denoted 1^k. The output of G is the key pair, (PK, SK). Using the notation introduced above, we write:

$$(PK, SK) \leftarrow G(1^k)$$

We say that the secret key SK *corresponds* to the public key PK if (PK, SK) can be produced by running $G(1^k)$.

Encryption The encryption algorithm E is a probabilistic polynomial-time algorithm that takes as input the message m from the message space $\mathcal{M}$, and the public key PK and produces as output the ciphertext c:

$$c \leftarrow E(m, PK)$$

Decryption The decryption algorithm D is a probabilistic polynomial-time algorithm that takes as input the ciphertext c and the secret key SK and produces as output the message m:

$$m \leftarrow D(c, SK)$$

There are two things that we require of a cryptosystem:

- The decryption should be correct, i.e., if SK corresponds to public key PK, then $D(E(b, PK), SK) = b$. (We can be more general and relax this condition to just asking that this happens with high probability.)
- The cryptosystem must be *secure*. The rest of this lecture, and several upcoming lectures, will deal with coming up with an appropriate formalization of this requirement.

4 Towards Defining Security

For today, let us focus on defining security for the one-bit message space: Bob can only send Alice one bit of information: "I love you" (encoded by a 0) or "I hate you" (encoded by a 1).

4.1 First Try

In lecture 1, we explored the possibility of Eve being computationally unbounded. We also assumed that the ciphertext, c, does not help in determining the message, m, i.e.,

$$\forall m, \forall c, \Pr[m \text{ was sent} \mid c \text{ was received}] = \Pr[m \text{ was sent}]$$

Which in our case with just one-bit messages chosen uniformly at random, corresponds to

$$\forall b, \forall c, \Pr[b \text{ was sent} \mid c \text{ was received}] = 1/2$$

As we saw in the previous lecture, this definition is too strong. In the public-key setting it is actually unattainable. The reason for this is that it implies that any given ciphertext c can be an encoding of a 1 as well as a 0. Since Alice and Bob have no shared secrets, this would be in contradiction to the correctness of decryption requirement! What happens is that if c could be obtained by running $E(0, PK)$ as well as by running $E(1, PK)$, then $0 = D(E(0, PK)) = c = D(E(1, PK)) = 1$ which is contradiction.

What is the lesson? The main lesson is that *c uniquely determines b*. Therefore, we cannot get a satisfactory definition insisting that the probability of a bit having been sent remain the same even when conditioning on a given ciphertext. Instead, we have to focus on the probability that Eve will figure out which bit was sent.

4.2 Second Try

Let us try another definition, which incorporates the lesson learned from our first try:

$$\Pr[b \leftarrow \{0, 1\}; c \leftarrow E(b, PK); b' \leftarrow Eve(c) : b = b'] = 1/2$$

The first thing that is funny about this definition is: what is PK? Should this hold for all PK? Or for some PK? The most natural way of getting the public key accounted for, is to take the probability over the choice of PK, as follows:

$$\Pr[(PK, SK) \leftarrow G(1^k); b \leftarrow \{0,1\}; c \leftarrow E(b, PK); b' \leftarrow Eve(c) : b = b'] = 1/2$$

It turns out that this definition of security is too strong. Suppose Eve is using the following algorithm:

Eve's Algorithm

1. Run G and obtain (PK', SK').
2. If $PK = PK'$, then obtain b' by running $D(c, SK)$.
3. Otherwise, output a random guess b'.

Let us estimate Eve's success in outputting the correct bit. Since G is a polynomial-time algorithm, it needs a polynomial number of random bits. Let us denote this polynomial bound by $r(k)$. With probability $2^{-r(k)}$, the random bits Eve used when running G coincide with the random bits used by Alice when generating her key pair.

Suppose $PK = PK'$. Then Eve's b' will always correspond to the bit b that was being sent: otherwise, the decryption would not be faithful!

Let p denote the probability that Eve's PK' is the same as PK. We just saw that $p \geq 2^{-r(k)}$. Now let's estimate Eve's success probability q:

$$\begin{aligned} q &= \Pr[PK = PK'] \Pr[\text{Eve is correct} | PK = PK'] + \Pr[PK \neq PK'] \Pr[\text{Eve is correct} | PK \neq PK'] \\ &= p * 1 + (1-p) * 1/2 = 1/2 + p/2 \geq 1/2 + 2^{-(r(k)+1)} \end{aligned}$$

What is the lesson? The lesson is that we cannot hope to prevent the adversary from having *any* advantage over randomly guessing the message sent. However, notice that in the above example, the adversary's advantage is so small that we don't care: if we only have polynomial time to live, we will never even notice this advantage.

4.3 Third Try

Let us weaken the definition to allow the adversary to have an advantage over random guessing, but only a very small advantage. A small advantage is an advantage that vanishes faster than any inverse polynomial – so it cannot even be observed in polynomial time!

A function f is called *negligible* if for all integers c, there exists an integer K such that for all $k \geq K$, $f(k) < 1/k^c$. Let negl denote the set of negligible functions.

So now the new definition is:

$$\begin{aligned} &\exists f(k) \in \text{negl} \\ &\Pr[(PK, SK) \leftarrow G(1^k); b \leftarrow \{0,1\}; \\ &c \leftarrow E(b, PK); b' \leftarrow Eve(c) : b = b'] \quad \leq \quad 1/2 + f(k) \end{aligned}$$

This definition is almost satisfactory, except that we don't know Eve's algorithm at the time when we design the cryptosystem. This needs to be accounted for.

5 Definition of Security (Indistinguishability)

The only thing that remains to be defined in further detail, is the notion of the algorithm Eve. We want it to capture all computation that is doable by a realistic device. For example, a probabilistic polynomial time Turing machine is a realistic device, since it has constant size and runs in polynomial time. But Turing machines don't capture all realistic devices: a device whose size is polynomial in k is still realistic for inputs of size k.

Let A be a Turing machine. Let $|A|$ denote the size of the description of Eve. By $\{A_k\}$ we denote a set of Turing machines, such that there exists a polynomial $p(k)$ such that $\forall k |A_k| < p(k)$. We also call such $\{A_k\}$ a *non-uniform* family of Turing machines. Let $PPTF$ stand for "probabilistic polynomial time family" of Turing machines.

Since we don't know Eve's algorithm ahead of time, we have to quantify over all possible probabilitic polynomial-time families.

The final definition is as follows:

$$\begin{aligned} &\forall PPTF\{Eve_k\}\exists f(k) \in \text{negl} \\ &\Pr[(PK, SK) \leftarrow G(1^k); b \leftarrow \{0,1\}; \\ &c \leftarrow E(b, PK); b' \leftarrow Eve(c) : b = b'] \quad \leq \quad 1/2 + f(k) \end{aligned}$$

This definition was first given by Shafi Goldwasser and Silvio Micali in their seminal work entitled "Probabilistic Encryption," which you will receive as a handout in the next lecture. In the next lecture, we will also see the Goldwasser-Micali cryptosystem which satisfies this definition.