University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 28

1 Secure Public-Key Encryption

We begin our study of public-key encryption as we began our study of private-key encryption. Thus, we start by considering a notion of *perfect secrecy* and see whether any public-key encryption scheme can satisfy it. Our definition here will be analogous to the definition of perfect secrecy for private-key encryption; the only difference now is that we assume the adversary knows — in addition to the ciphertext — the recipient's public key.

Definition 1 *A public-key encryption scheme is* perfectly secret *if for all public keys PK, all messages m_0, m_1, all ciphertexts C, and all algorithms A we have:*

$$\Pr[A(PK, C) = 0 \mid C \leftarrow \mathcal{E}_{PK}(m_0)] = \Pr[A(PK, C) = 0 \mid C \leftarrow \mathcal{E}_{PK}(m_1)].$$

Recall that in the case of private-key encryption, perfect secrecy was achievable (although not very efficiently) using the one-time pad scheme. However, we now show that perfect secrecy is *impossible* in the public-key case.

Lemma 1 *No public-key encryption scheme is perfectly secret.*

Proof Assume $m_0 \neq m_1$ and consider the following algorithm A:

```
On input $(PK, C)$:
  For each possible choice of the random bits:
    Set $C_0 = \mathcal{E}_{PK}(m_0)$
    if $C = C_0$ output 0 and end
  output 1 and end
```

This algorithm certainly terminates; note that there are only a finite number of possible choices for the random bits used by the encryption algorithm (since the encryption algorithm must terminate — in fact, we will always assume that encryption can be done in polynomial time). Of course, the above algorithm will not be *efficient* (i.e., will not run in polynomial time) but that is ok because we are dealing with perfect secrecy.

We note the following behavior of the algorithm: (1) If C is an encryption of m_0, the algorithm always outputs 0 (since the algorithm eventually tries the same random bits that were used when computing C). (2) If C is an encryption of m_1, the algorithm always outputs 1. This is so as long as we assume that the scheme is *correct*; i.e., that the receiver always decrypts correctly (note that if C could be an encryption of both m_0 and m_1 then this would cause a decryption error). So, $\Pr[A(PK, C) = 0 \mid C \leftarrow \mathcal{E}_{PK}(m_0)] = 1$ and $\Pr[A(PK, C) = 0 \mid C \leftarrow \mathcal{E}_{PK}(m_1)] = 0$ and perfect secrecy is violated. ■

Since perfect secrecy is not achievable for public-key encryption schemes, we try to relax the definition to see what can be achieved. As in the case of private-key encryption, we consider a restricted class of adversaries; namely, we consider adversaries with some bound on their total running time. Before introducing our first definition of security, we give a precise definition of a public-key encryption scheme.

Definition 2 *A* public-key encryption scheme *is a triple of polynomial-time algorithms* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ *such that:*

- $\mathcal{K}$ *is a randomized* key generation algorithm. *It takes as input a security parameter*[1] 1^k *and outputs a pair of keys* (PK, SK)*. We call* PK *the* public key *and* SK *the* secret key.
- $\mathcal{E}$ *is a (possibly randomized)* encryption algorithm. *It takes as input a public key* PK *and a message* m *and outputs a ciphertext* C*. We write this as:* $C \leftarrow \mathcal{E}_{PK}(m)$.
- $\mathcal{D}$ *is a deterministic* decryption algorithm. *It takes as input a secret key* SK *and a ciphertext* C *and outputs a message* m*. (It is theoretically possible to have a randomized decryption algorithm; all the examples we will see, however, will be deterministic.)*

We assume the scheme is correct*; that is, for all* (PK, SK) *output by* $\mathcal{K}(1^k)$*, for all* m*, and for all* C *output by* $\mathcal{E}_{PK}(m)$ *we have:* $\mathcal{D}_{SK}(C) = m$.

Having defined an encryption scheme, we can now propose a definition of security.

Definition 3 *Fix a security parameter* 1^k *(so we no longer write it explicitly). A public-key encryption scheme is* (t, ϵ)-secure against ciphertext-only attacks *if, for all messages* m_0, m_1 *and all algorithms* A *running in time at most* t *we have:*

$$\begin{aligned}\epsilon \quad \geq \quad & |\Pr[(PK, SK) \leftarrow \mathcal{K}; C \leftarrow \mathcal{E}_{PK}(m_0) : A(PK, C) = 0] \\ & - \Pr[(PK, SK) \leftarrow \mathcal{K}; C \leftarrow \mathcal{E}_{PK}(m_1) : A(PK, C) = 0]| .\end{aligned}$$

In the private-key case, there exist deterministic encryption schemes secure against ciphertext-only attacks. We show that this is impossible in the public-key case.

Lemma 2 *(For any reasonable values of* t, ϵ*) no deterministic public-key encryption scheme can be* (t, ϵ)*-secure against ciphertext-only attacks.*

Proof Let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a deterministic encryption scheme; i.e., one for which $\mathcal{E}$ is a deterministic algorithm. Consider the following algorithm A:

```
On input $(PK, C)$:
  compute $C_0 = \mathcal{E}_{PK}(m_0)$
  if $C = C_0$ output 0 else output 1
```

[1] The security parameter k can be thought of as determining the key length of the scheme; i.e., $|PK| = k$.

Note that when $\mathcal{E}$ is deterministic, A always outputs 0 if C is an encryption of m_0 and always outputs 1 if C is an encryption of m_1. The running time of A is essentially the running time of the encryption algorithm $\mathcal{E}$. So, for any reasonable value of t (certainly, the allowed running time of A should be longer than the time needed to encrypt) and any $\epsilon < 1$, the scheme is not (t, ϵ)-secure against ciphertext-only attacks. ■

Because this result is so important, we state the following (easy) corollary.

Corollary 1 *"Textbook" RSA encryption is* not *secure.*

(By "textbook" RSA encryption we mean the scheme in which the public key consists of a modulus N and a public exponent e, and a messages $m \in \mathbb{Z}_N^*$ is encrypted by computing $C = m^e \bmod N$. Since this scheme is *deterministic* it cannot be secure.)

In fact, it takes some work to make RSA secure, and we defer this until later in the semester. Instead, we present a simpler (yet secure) encryption scheme first.

2 A Secure Public-Key Encryption Scheme

Before presenting our first scheme, we review a number of facts from number theory (that we have seen already in this class). Throughout this section we let $N = pq$ be a product of two distinct, odd primes.

- Exactly half the elements of $\mathbb{Z}_N^*$ have Jacobi symbol $+1$, and exactly half have Jacobi symbol -1. There is a polynomial-time algorithm for computing the Jacobi symbol of an element $x \in \mathbb{Z}_N^*$ *even without knowledge of the factorization of* N (we did not cover this algorithm in class).

- Of the elements in $\mathbb{Z}_N^*$ with Jacobi symbol $+1$, exactly half of these are quadratic residues and half are quadratic non-residues. Every quadratic residue has four distinct square roots in $\mathbb{Z}_N^*$.

- Given an element $x \in \mathbb{Z}_N^*$ with Jacobi symbol $+1$, it is conjectured to be "hard" to decide whether or not x is a quadratic residue if the factorization of N is unknown. We refer to this as the *hardness of deciding quadratic residuosity*. Later, we give a precise definition of what this means.

- When the factorization of N *is* known, there is a polynomial-time algorithm for determining whether an element $x \in \mathbb{Z}_N^*$ is a quadratic residue or not. (As discussed in class, this algorithm uses the "Chinese remaindering" representation of x with respect to p and q; deciding quadratic residuosity modulo prime numbers is easy [as you show on homework 4].)

The hardness of deciding quadratic residuosity suggests the following encryption scheme for encryption of 1-bit messages:

1. $\mathcal{K}(1^k)$ generates (using the polynomial-time prime generation algorithms discussed in class) two random primes p, q with $|p| = |q| = k$ and computes $N = pq$. The public key is N and the secret key is (p, q).

2. To encrypt "0", the sender chooses C as a random quadratic residue in $\mathbb{Z}_N^*$; to encrypt "1", the sender chooses C as a random quadratic non-residue in $\mathbb{Z}_N^*$. The ciphertext is C. (As noted above, it is crucial that encryption is *randomized.*)

3. Upon receiving ciphertext C, the receiver (using p, q) determines whether C is a quadratic residue or not, and decrypts accordingly.

Informally, an eavesdropper sees only N and an element $C \in \mathbb{Z}_N^*$. Since it is "hard" to determine whether C is a quadratic residue or not, the scheme seems secure.

We need to be careful, though. First, note that it is hard to decide quadratic residuosity *only* for elements C with Jacobi symbol $+1$. But if the sender picks a random quadratic non-residue C when encrypting a "1", then with probability $2/3$ it will be the case that C has Jacobi symbol -1 (why?). Since the Jacobi symbol can be computed in polynomial time, an eavesdropping adversary can tell when this occurs and the scheme will not be secure. So, the scheme needs to be modified as follows:

2′. To encrypt "0", the sender chooses C as a random quadratic residue in $\mathbb{Z}_N^*$; to encrypt "1", the sender chooses C as a random quadratic non-residue in $\mathbb{Z}_N^*$ *with Jacobi symbol* $+1$. The ciphertext is C.

Now, the ciphertext C always has Jacobi symbol $+1$ and hence — appealing to the hardness of deciding quadratic residuosity for elements with Jacobi symbol $+1$ — the scheme seems secure. (Of course, we will still need to provide a rigorous proof of security to verify our intuition. We do this in the next class.)

Next, we need to verify that all algorithms in the encryption scheme can be executed in polynomial time. The key generation algorithm $\mathcal{K}$ certainly runs in polynomial time given the algorithms for generating primes that we have already seen. The decryption algorithm runs in polynomial time, since we have stated above that deciding quadratic residuosity is easy when the factorization of N is known. But what about the encryption algorithm? How can the sender — who does not know the factorization of N — choose random quadratic residues and non-residues?

We note the following (in all cases, we assume the factorization of N is not known):

- Choosing a random $x \in \mathbb{Z}_N^*$ with Jacobi symbol $+1$ is "easy". Simply pick a random $x \in \mathbb{Z}_N^*$ and test whether it has Jacobi symbol $+1$. If it does, then output x. If not, repeat. On average (since half the elements of $\mathbb{Z}_N^*$ have Jacobi symbol $+1$) the algorithm tests 2 elements before finding one with Jacobi symbol $+1$.

- Similarly, choosing a random $x \in \mathbb{Z}_N^*$ with Jacobi symbol -1 is "easy" (via the obvious modification of the previous algorithm).

- Choosing a random quadratic residue x is even easier: simply pick a random $y \in \mathbb{Z}_N^*$ and set $x = y^2 \bmod N$. Clearly, x is a quadratic residue. Furthermore, it is not hard to show (since every quadratic residue has four distinct square roots) that x is a *random* quadratic residue.

But how can the sender choose a random quadratic *non*-residue? Try to think about this until next time...