CMSC 858K — Advanced Topics in Cryptography	February 5, 2004

Lecture 4

Lecturer: Jonathan Katz

Scribe(s): Chiu Yuen Koo, Nikolai Yakovenko, Jeffrey Blank

1 Summary

The focus of this lecture is efficient public-key encryption. In the previous lecture, we discussed a public-key encryption scheme for 1-bit messages. However, to encrypt an ℓ-bit message, we can simply encrypt ℓ one-bit messages and send these (and we proved last time that this remains secure in the case of public-key encryption). Here, we first describe (briefly) how to combine public and private key encryption to obtain a public-key encryption scheme with the efficiency of a private-key scheme (for long messages). Next, we describe an efficient public key encryption scheme called *El Gamal encryption* [2] which is based on a particular number-theoretic assumption rather than the general assumption of trapdoor permutations. In the course of introducing this scheme, we discuss how it relies on the Discrete Logarithm Problem and the Decisional Diffie-Hellman Assumption.

2 Hybrid Encryption

A hybrid encryption scheme uses public-key encryption to encrypt a random symmetric key, and then proceeds to encrypt the message with that symmetric key. The receiver decrypts the symmetric key using the public-key encryption scheme and then uses the recovered symmetric key to decrypt the message.

More formally, let (KeyGen, $\mathcal{E}$, $\mathcal{D}$) be a secure public- key encryption scheme and ($\mathcal{E}'$, $\mathcal{D}'$) be a secure private-key encryption scheme. We can construct a secure hybrid encryption scheme (KeyGen", $\mathcal{E}''$,$\mathcal{D}''$) as follows:

- KeyGen" is the same as *KeyGen*, generating a public key pk and a secret key sk.
- $\mathcal{E}''_{pk}(m)$:
  1. $sk' \leftarrow \{0,1\}^k$
  2. $C_1 \leftarrow \mathcal{E}_{pk}(sk')$
  3. $C_2 \leftarrow \mathcal{E}'_{sk'}(m)$
- $\mathcal{D}''_{sk}(C_1, C_2)$:
  1. $sk' = \mathcal{D}_{sk}(C_1)$
  2. $m = \mathcal{D}'_{sk'}(C_2)$

The above scheme can be proven semantically-secure, assuming the semantic security of the underlying public- and private-key schemes. However, we will not give a proof so here.

3 The El Gamal Encryption Scheme

3.1 Groups

Before describing the El Gamal encryption scheme, we give a very brief overview of the group theory necessary to understand the scheme. We will need to concept of a *finite, cyclic group*, so we first introduce the concept of a finite group (actually, we will introduce what is known as an *Abelian group*, since we will not use non-Abelian groups in this class).

Definition 1 An Abelian group $\mathcal{G}$ is a finite set of elements along with an operation $*$ (written multiplicatively, although not necessarily corresponding to integer multiplication!) such that:

Closure For all $a, b \in \mathcal{G}$ we have $a * b \in \mathcal{G}$. Since we are using multiplicative notation, we also write $a * b$ as ab when convenient.

Associativity For all $a, b, c \in \mathcal{G}$, we have $(ab)c = a(bc)$.

Commutativity For all $a, b \in \mathcal{G}$ we have $ab = ba$.

Existence of identity There exists an element $1 \in \mathcal{G}$ such that $1 * a = a$ for all $a \in \mathcal{G}$. This element is called the identity of $\mathcal{G}$.

Inverse For all $a \in \mathcal{G}$ there exists an element $a^{-1} \in \mathcal{G}$ such that $aa^{-1} = 1$.

$\diamondsuit$

If n is a positive integer and $a \in \mathcal{G}$, the notation a^n simply refers to the product of a with itself n times. Just as we are used to, we have $a^0 = 1$ and $a^1 = a$ for all $a \in \mathcal{G}$. Let $q = |\mathcal{G}|$ (i.e., the number of elements in $\mathcal{G}$); this is known as the *order* of $\mathcal{G}$. A useful result is the following theorem:

Theorem 1 *Let $\mathcal{G}$ be a finite Abelian group of order q. Then $a^q = 1$ for all $a \in \mathcal{G}$.*

Proof We may actually give a simple proof of this theorem. Let $a_1, \ldots, a_q$ be the elements of $\mathcal{G}$, and let $a \in \mathcal{G}$ be arbitrary. We may note that the sequence of elements $aa_1, aa_2, \ldots, aa_q$ also contains exactly the elements of $\mathcal{G}$ (clearly, this sequence contains at most q distinct elements; furthermore, if $aa_i = aa_j$ then we can multiply by a^{-1} on both sides to obtain $a_i = a_j$ which is not the case). So:

$$\begin{aligned} & a_1 \cdot a_2 \cdots a_q \\ = \; & (aa_1) \cdot (aa_2) \cdots (aa_q) \\ = \; & a^q (a_1 \cdot a_2 \cdots a_q). \end{aligned}$$

Multiplying each side by $(a_1 \cdots a_q)^{-1}$, we see that $a^q = 1$. ■

We mention the following easy corollary:

Corollary 2 *Let $\mathcal{G}$ be a finite Abelian group of order q, and let n be a positive integer. Then $g^n = g^{n \bmod q}$.*

Proof Let $n = n_q \bmod q$ so that n can be written as $n = aq + n_q$ for some integer a. We then have $g^n = g^{aq+n_q} = (g^a)^q g^{n_q} = g^{n_q}$. ■

A finite group $\mathcal{G}$ of order q is *cyclic* if there exists an element $g \in \mathcal{G}$ such that the set $\{g^0, g^1, \ldots, g^{q-1}\}$ is all of $\mathcal{G}$. (Note that $g^q = g^0 = 1$ by the above theorem, so that the sequence would simply "cycle" if continued.) If such a g exists, it is called a *generator* of $\mathcal{G}$. As an example, consider the group $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ under multiplication modulo 5. Since $4^2 = 1$, the element 4 is *not* a generator. However, since $2^1 = 2, 2^2 = 4$, and $2^3 = 3$, element 2 is a generator and $\mathbb{Z}_5^*$ is cyclic.

All the groups we are going to deal with here will be cyclic, and will additionally have prime order (i.e., $|\mathcal{G}| = q$ and q is prime.) The following is a simple fact in such groups:

Lemma 3 *If $\mathcal{G}$ is an Abelian group with prime order q, then (1) $\mathcal{G}$ is cyclic; furthermore, (2) every element of $\mathcal{G}$ (except the identity) is a generator.*

Proof We do not prove that $\mathcal{G}$ is cyclic, but instead refer the reader to any book on group theory. However, assuming $\mathcal{G}$ is cyclic, we may prove the second part. Let g be a generator of $\mathcal{G}$, and consider an element $h \in \mathcal{G} \setminus \{1\}$. We know that $h = g^i$ for some i between 1 and $q - 1$. Consider the set $\mathcal{G}' = \{1, h, h^2, \ldots, h^{q-1}\}$. By the closure property, $\mathcal{G}' \subseteq \mathcal{G}$. On the other hand, if $h^x = h^y$ for some $1 \leq x < y \leq q - 1$ then $g^{xi} = g^{yi}$. By the Corollary given above, this implies that $xi = yi \bmod q$, or, equivalently, that q divides $(y - x)i$. However, since q is prime and both $(y - x)$ and i are strictly less than q, this cannot occur. This implies that all elements in $\mathcal{G}'$ are unique, and hence h is a generator. ■

An important fact about a cyclic group $\mathcal{G}$ of order q is that given a generator g, every element of $h \in \mathcal{G}$ satisfies $h = g^s$ for exactly one s between 0 and $q - 1$ (this follows immediately from the definition of a generator). In this case, we say $\log_g h = s$ (the previous fact indicates that this is well-defined as long as g is a generator) and call s the discrete logarithm of h to the base g. Discrete logarithms satisfy many of the rules you are familiar with for logarithms over the reals; for example (always assuming g is a generator), we have $\log_g(h_1 h_2) = \log_g h_1 + \log_g h_2$ for all $h_1, h_2 \in \mathcal{G}$.

For our applications to cryptographic protocols, we will consider groups of very large order (on the order of $q \approx 2^{100}$ or more!). When dealing with such large numbers, it is important to verify that the desired arithmetic operations can be done *efficiently*. As usual, we associate "efficient" with "polynomial time". The only subtlety here is that the running time should be polynomial in the *lengths* of all inputs (and not their absolute value); so, for example, computing g^x for some generator g in some group $\mathcal{G}$ should require time polynomial in $|x|$ (equivalently, the number of bits needed to describe x) and $|q|$, rather than polynomial in x and q. This clearly makes a big difference — an algorithm running in 2^{100} steps is infeasible, while one running in 100 steps certainly is!

We will take it as a given that all the groups with which we will deal support efficient "base" operations such as multiplication, equality testing, and membership testing. (Yet, for some groups that are used in cryptography, verifying these properties is non-trivial!) However, we do *not* assume that exponentiation is efficient, but will instead show that it can be done efficiently (assuming that multiplication in the group can be done efficiently).

To see that this is not entirely trivial, consider the following naive algorithm to compute g^x for some element g in some group $\mathcal{G}$ (the exponent x is, of course, a positive integer):

```
exponentiate_naive( g , x ) {
    ans = 1;
    if x ?= 0 return 1;
    while(x ≥ 1){
        ans = ans * g;
        x = x − 1;
    }
    return ans; }
```

(In the above algorithm, the expression $\mathsf{ans} * g$ means multiplication in the *group* and not over the integers.) However, this algorithm requires time $O(x)$ to run (there are x iterations of the loop), which is unacceptable and not polynomial time! However, we can improve this by using *repeated squaring*. First, note that:

$$g^x = \begin{cases} (g^{\frac{x}{2}})^2 & \text{if x is even} \\ g(g^{\frac{x-1}{2}})^2 & \text{if x is odd} \end{cases}.$$

This leads us to the following algorithm for exponentiation:

```
exponentiate_efficient (g, x) {
        if (x ?= 0) return 1;
        tmp = 1, ans = g;
        // we maintain the invariant that tmp * ans^x is our answer
        while (x > 1) {
                if (x is odd) {
                        tmp = tmp * ans;
                        x = x − 1; }
                if (x > 1) {
                        ans = ans * ans;
                        x = x/2; }
                }
        return tmp * ans; }
```

(Again, the " $*$ " in the above code refers to multiplication in the group, not over the integers. However, expressions involving x *are* performed over the integers.) Note that in each execution of the loop the value of x is decreased by at least half, and thus the number of executions of this loop is $O(\log x) = O(|x|)$. Thus, the algorithm as a whole runs in polynomial time.

From the above, we see that given a generator g and an integer x, we can compute $h = g^x$ in polynomial time. What about the inverse problem of finding x given h and g? This is known as the *discrete logarithm problem* which we define next.

3.2 The Discrete Logarithm Problem

The discrete logarithm problem is as follows: given generator g and random element $h \in \mathcal{G}$, compute $\log_g h$. For many groups, this problem is conjectured to be "hard"; this is referred

to as the *discrete logarithm assumption* which we make precise now. In the following, we let $\mathsf{GroupGen}$ be a polynomial time algorithm which on input 1^k outputs a description of a cyclic group $\mathcal{G}$ of order q (with $|q| = k$ and q not necessarily prime), and also outputs q and a generator $g \in \mathcal{G}$. (Note that $\mathsf{GroupGen}$ may possibly be deterministic.) The *discrete logarithm assumption* is simply that the assumption that the discrete logarithm problem is hard for $\mathsf{GroupGen}$, where this is defined as follows:

Definition 2 The *discrete logarithm problem is hard for* $\mathsf{GroupGen}$ if the following is negligible for all PPT algorithms A:

$$\Pr[(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k); h \leftarrow \mathcal{G}; x \leftarrow A(\mathcal{G}, q, g, h) : g^x = h].$$

$\diamondsuit$

Sometimes, if the discrete logarithm problem is hard for $\mathsf{GroupGen}$ and $\mathcal{G}$ is a group output by $\mathsf{GroupGen}$, we will informally say that the discrete logarithm problem is hard in $\mathcal{G}$.

We provide an example of a $\mathsf{GroupGen}$ for which the discrete logarithm assumption is believed to hold: Let $\mathsf{GroupGen}$ be an algorithm which, on input 1^k, generates a random prime q of length k (note that this can be done efficiently via a randomized algorithm), and let $\mathcal{G} = \mathbb{Z}_q^*$. It is known that this forms a cyclic group of order $q - 1$ (not a prime). It is also known how to efficiently find a generator g of $\mathcal{G}$ via a randomized algorithm which we do not describe here. Let $(\mathcal{G}, q, g)$ be the output of $\mathsf{GroupGen}$.

We now describe the El Gamal encryption scheme whose security is related to (but does *not* follow from) the discrete logarithm assumption:

Key generation $\mathsf{Gen}(1^k)$:
$(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k)$
Choose $x \leftarrow \mathbb{Z}_q$; set $y = g^x$
Output $PK = (\mathcal{G}, q, g, y)$ and $SK = x$

Encryption $\mathcal{E}_{pk}(m)$ (where $m \in \mathcal{G}$):
Pick $r \leftarrow \mathbb{Z}_q$
Output $\langle g^r, y^r m \rangle$

Decryption $\mathcal{D}_{sk}(A, B)$:
Compute $m = \frac{B}{A^x}$

Correctness of decryption follows from $\frac{y^r m}{(g^r)^x} = \frac{y^r m}{(g^x)^r} = \frac{y^r m}{y^r} = m$.

The discrete logarithm problem implies that no adversary can determine the secret key given the public key (can you prove this?). However, this alone is *not* enough to guarantee semantic security! In fact, we can show a particular group for which the discrete logarithm assumption (DLA) is believed to hold, yet the El Gamal encryption scheme is *not* semantically secure. Namely, consider groups $\mathbb{Z}_p^*$ for p prime, as discussed earlier. We noted that the DLA is believed to hold in groups of this form. However, it is also known how to determine in polynomial time whether a given element of $\mathbb{Z}_p^*$ is a *quadratic residue* or not (an element $y \in \mathbb{Z}_p^*$ is a quadratic residue if there exists an $x \in \mathbb{Z}_p^*$ such that $x^2 = y$). Furthermore, a generator g of $\mathbb{Z}_p^*$ cannot be a quadratic residue. These observations leads to a direct attack on the El Gamal scheme (we sketch the attack here, but let the reader

fill in the details or refer to [1, Section 9.5.2]): output (m_0, m_1) such that m_0 is a quadratic residue but m_1 is not. Given a ciphertext $\langle A, B\rangle$, where $A = g^r$ and $B = y^r m_b$ for some r, we can determine in polynomial time whether y^r is a quadratic residue or not (for example, if A or y are quadratic residues then at least one of r, x is even and thus $y^r = y^{xr}$ is also a quadratic residue). But then by looking at B we can determine whether m_b is a quadratic residue or not (e.g., if y^r is a non-residue and B is a residue, then it must be the case that m_b was not a quadratic residue), and hence determine which message was encrypted.

Evidently, then, we need need a stronger assumption about $\mathsf{GroupGen}$ in order to prove that El Gamal encryption is semantically secure.

3.3 The Decisional Diffie-Hellman (DDH) Assumption

Informally, the DDH assumption is that it is hard to distinguish between tuples of the form (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z), where g is a generator and x, y, z are random. More formally, $\mathsf{GroupGen}$ satisfies the DDH assumption if the DDH problem is hard for $\mathsf{GroupGen}$, where this is defined as follows:

Definition 3 The *DDH problem is hard for* $\mathsf{GroupGen}$ if the following distributions are computationally indistinguishable (cf. the definition from Lecture 3):

$$\{(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k); x, y, z \leftarrow \mathbb{Z}_q : (\mathcal{G}, q, g, g^x, g^y, g^z)\}$$

and

$$\{(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k); x, y \leftarrow \mathbb{Z}_q : (\mathcal{G}, q, g, g^x, g^y, g^{xy})\}.$$

♢

We call tuples chosen from the first distribution "random tuples" and tuples chosen from the second distribution "DH tuples". (Note that this is an abuse of terminology, since there exist tuples in the support of both distributions. But when we say that $\vec{g}$ is a random tuple we simply mean that $\vec{g}$ was drawn from the first distribution above.) Also, if the DDH assumption holds for $\mathsf{GroupGen}$ and $\mathcal{G}$ is a particular group output by $\mathsf{GroupGen}$ then we informally say that the DDH assumption holds in $\mathcal{G}$.

Security of the El Gamal encryption scheme turns out to be equivalent to the DDH assumption. We prove the more interesting direction in the following theorem.

Theorem 4 *Under the DDH assumption, the El Gamal encryption scheme is secure in the sense of indistinguishability.*

Proof (Note: what we really mean here is that if the DDH assumption holds for $\mathsf{GroupGen}$, and this algorithm is used in the key generation phase of El Gamal encryption as described above, then that particular instantiation of El Gamal encryption is secure.)

Assume a PPT adversary A attacking the El Gamal encryption scheme in the sense of indistinguishability. Recall this means that A outputs messages (m_0, m_1), is given a random encryption of m_b for random b, and outputs guess b'. We will say that A succeeds if $b' = b$ (and denote this event by SUCC), and we are ultimately interested in $\Pr_A[\mathsf{Succ}]$.

We construct an adversary A' as follows:

$\underline{A'(\mathcal{G}, q, g_1, g_2, g_3, g_4)}$
$PK = (g_1, g_2)$
run $A(PK)$ and get messages (m_0, m_1)
$b \leftarrow \{0,1\}$
$C = \langle g_3, g_4 m_b \rangle$
run $A(PK, C)$ to obtain b'
output 1 iff $b' = b$

Let Rand be the event that (g_1, g_2, g_3, g_4) are chosen from the distribution of random tuples, and let DH be the event that they were chosen from the distribution on DH tuples. Since the DDH assumption holds in $\mathcal{G}$ and A' is a PPT algorithm we know that the following is negligible:

$$\left|\Pr[A' = 1|\mathsf{DH}] - \Pr[A' = 1|\mathsf{Rand}]\right|.$$

Next, we claim that $\Pr[A' = 1|\mathsf{DH}] = \Pr_A[\mathsf{Succ}]$. To see this, note that when DH occurs we have $g_2 = g_1^x$, $g_3 = g_1^r$, and $g_4 = g_1^{xr} = g_2^r$ for some x and r chosen at random. But then the public key and the ciphertext are distributed exactly as they would be in a real execution of the El Gamal encryption scheme, and since A' outputs 1 iff A succeeds, the claim follows.

To complete the proof, we show that $\Pr[A' = 1|\mathsf{Rand}] = 1/2$ (do you see why this completes the proof?). Here, we know that g_4 is uniformly distributed in $\mathcal{G}$ independent of g_1, g_2, or g_3. In particular, then, the second component of the ciphertext given to A is uniformly distributed in $\mathcal{G}$ independent of the message being encrypted (and, in particular, independent of b). Thus, A' has *no information* about b — even an all-powerful A cannot predict b in this case with probability different from $1/2$ (we assume that A must always output some guess $b' \in \{0,1\}$). Since A' outputs 1 iff A succeeds, we may conclude that $\Pr[A' = 1|\mathsf{Rand}] = 1/2$. ■

References

[1] M. Bellare and P. Rogaway. Introduction to Modern Cryptography. Notes available from http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html.

[2] T. El Gamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* 31(4): 469–472 (1985).