6.875J/18.425J Cryptography and Cryptanalysis	September 5, 2001

Lecture 1: Introduction

Instructor: Anna Lysyanskaya *Teaching Assistant: Moses Liskov*

Scribed by: Moses Liskov

1 The Problem

Suppose there are two people, Alice and Bob, and Bob wants to send a love-letter to Alice. Unfortunately, there is an evesdropper, Eve, who wants to read the love-letter. We want to find some way for Bob to send the letter to Alice without Eve being able to understand it. To formally describe the process,

1. Bob chooses a message m. Since we want Eve to be unsure of what message is being sent, the message must be chosen from a probability distribution. We use the symbol $\mathcal{M}$ to represent the *message space*, and write

$$m \leftarrow \mathcal{M}$$

2. Bob applies the encryption algorithm E to m, the *plaintext*, to get c, the *ciphertext*. What inputs does E take? Well,

   i. Certainly m will be an input.

   ii. Perhaps a shared secret key s will be an input.

   iii. Perhaps a piece of public info PK will be an input.

   iv. Perhaps E will be randomized, so a random string r may be another input.

   So Bob computes $c \leftarrow E(m, s, PK, r)$ and sends c to Alice.

3. Alice has a decryption algorithm D. What inputs does D take?

   i. Certainly, c will be an input.

   ii. Perhaps a shared secret key s will be an input.

   iii. Perhaps a piece of public info PK will be an input.

   iv. Perhaps Alice has a secret of her own, SK.

   Alice finally computes $m \leftarrow D(c, s, PK, SK)$.

Since Eve gets to observe the communication between Alice and Bob, she clearly gets to know c. However, we are sure she doesn't get to know m, because otherwise the whole exercise would be pointless. But what about the other information?

Observation 1: Alice must know *something* Eve doesn't know. If Eve knows everything Alice knows, Eve can just compute $m \leftarrow D(c, s, PK, SK)$. Thus, Eve either learns m or Alice can't decrypt c.

What about the rest of the information? Is it okay for Eve to know everything *Bob* knows? First of all, we have to define what Eve is. At the moment, let's assume nothing about Eve except that she is an algorithm. Specifically, we are not assuming any bounds on her computational power - she can take arbitrary amounts of time or use arbitrary amounts of space.

Observation 2: Bob must know *something* Eve doesn't know if encryption is to be achieved when Eve has no computational bound on her power. If Eve knows everything Bob knows, she can just try every possible message $m \in \mathcal{M}$, calculating $E(m, s, PK, r)$ every time until she gets c. Whatever m produced c would be the correct message (there cannot be any ambivalence if Alice can decrypt).

Note that for this method of attack to work, Eve really has to be a powerful adversary. Such a thing would not be possible if Eve were computationally bounded since there are probably exponentially many possible messages to try. Observation 1 and observation 2 lead us to conclude:

Observation 3: If Eve unbounded, Alice and Bob must both have secrets.

So the question is now, if Alice and Bob share a secret, can they achieve privacy? As we will see, the answer is a qualified "yes."

2 The One-Time Pad

Here is the one-time pad cryptosystem. We assume that a parameter k is known to everyone beforehand, and that $\mathcal{M}$ draws messages from the space $\{0,1\}^k$. Let $\mathcal{S}$ be the uniform distribution on $\{0,1\}^k$. Now, if we let $s \leftarrow \mathcal{S}$ be the shared secret key, we can define encryption and decryption as follows:

$$E(m, s) = m \oplus s,$$

and

$$D(c, s) = c \oplus s,$$

where $x \oplus y$ refers to the bitwise XOR operation. The correctness of the decryption should be obvious. We will now prove that this system is secure. Let m_P be the event that message m is the one chosen from $\mathcal{M}$ to send (P stands for "plaintext"). Let c_C be the event that ciphertext c is the one sent from Bob to Alice.

Theorem 1 (Vernon): $\forall m\, \forall c, Pr[m_P] = Pr[m_P | c_C]$. That is, the events m_P and c_C are independent events.

Proof. First, we note that $Pr[m_P | c_C] = Pr[m_P \wedge c_C]/Pr[c_C]$. Now, we can derive

$$\begin{aligned}
Pr[m_P \wedge c_C] &= Pr[m_P]Pr[c_C|m_P] \\
&= Pr[m_P]Pr[s = c \oplus m|m_P] \\
&= Pr[m_P]2^{-k}
\end{aligned}$$

since s is chosen independently of m. However, we also have that

$$\begin{aligned}
Pr[c_C] &= \sum_{m' \in \{0,1\}^k} Pr[m'_P]Pr[c_C|m'_P] \\
&= \sum_{m' \in \{0,1\}^k} Pr[m'_P]2^{-k} \\
&= 2^{-k} \sum_{m' \in \{0,1\}^k} Pr[m'_P] \\
&= 2^{-k}
\end{aligned}$$

Thus, we get

$$\begin{aligned}
Pr[m_P|c_C] &= \frac{Pr[m_P \wedge c_C]}{Pr[c_C]} \\
&= \frac{2^{-k}Pr[m_P]}{2^{-k}} \\
&= Pr[m_P],
\end{aligned}$$

which completes the proof.

This is wonderful! Since the ciphertext is independent of the message, Eve cannot learn the message from observing the ciphertext. But there's a problem: *s can only be used once!* In our scenario, only one message is being sent so this is not a security problem, but rather an efficiency problem. If Alice and Bob must get together once for each message they are to send, why bother having cryptography? Now, Alice and Bob could get together once and establish *many* secret keys, but this is certainly a problem with this system. But maybe we can do better than this?

3 Shannon's Theorem

As it turns out, we can't. Shannon's Theorem is that if Alice and Bob share only a secret of length k, there is *no* method which can securely communicate messages of length $k+1$. We should be careful about saying "securely communicate" without knowing what it means. For this theorem, a system is "secure" if it meets the above definition, namely that for any message m and ciphertext c, $Pr[m_P] = Pr[m_P|c_C]$.

Theorem 2 (Shannon): If $|\mathcal{M}| > |\mathcal{S}|$ then $\exists m, c,\ Pr[m_P] \neq Pr[m_P|c_C]$.

Proof. Suppose not. That is, suppose that $\forall m, c,\ Pr[m_P] = Pr[m_P | c_C]$. Then, we know that

$$\forall m, m' \in \mathcal{M}, s \in \mathcal{S},\ \exists s' : E(m, s) = E(m', s') \wedge s' \neq s.$$

If this was not true, then there would be a contradiction, since $Pr[m'_P] > 0$ (since $m' \in \mathcal{M}$, it is a possible message), but $Pr[m'_P | (E(m, s))_C] = 0 \neq Pr[m'_P]$.

Clearly, $s' \neq s$ if they exist. Otherwise, $E(m, s) = E(m', s)$, and then Alice cannot compute $D(E(m, s), s)$. Now we can see there must be as many keys as there are plaintexts, thus contradicting that $|\mathcal{M}| > |\mathcal{S}|$. This completes the proof.

4 The Computationally Bounded Adversary Model

Shannon's theorem proves that we cannot do better than the one-time pad if we want security. However, the security the one-time pad provides allows the adversary to be very powerful indeed. So instead of simply going home, we ought to see if we can do better in the model where we only allow Eve to run for polynomial time. This is a more realistic model anyway - after all, the NSA may be very rich and powerful, but they are computationally bounded! However, to be fair, we should also assume that Alice and Bob are computationally bounded. It wouldn't be a very useful model if we assumed that the adversary was *less* powerful than the players.

Note that observation 1 still holds when the adversary is computationally bounded. If Eve knows everything that Alice knows, Eve can simply run the decryption algorithm (which is efficient, since Alice is also bounded), and recover the message. But, what about observation 2?

Observation 2 does not hold, since Eve can no longer try every message. It is very reasonable to assume that the message space is exponentially large (for example, if the possible messages are all in $\{0, 1\}^k$), so Eve cannot simply run this attack. However, just because observation 2 does not hold does *not* mean there are definitely secure encryption schemes! Unfortunately, there is a lot we don't know in complexity theory, so we have to make some assumptions. One assumption we will probably have to make is that $P \neq NP$. (For example, if Eve knows everything Bob knows except for m, it would be easy to verify that a string w contained all of Alice's private information, since decryption would work. Thus, if Eve can run an NP algorithm, she can break the cryptosystem). However, in practice we must make even more assumptions.

So, can we construct a cryptosystem where Alice and Bob share no secret?

5 Diffie-Hellman Key Exchange

So how can we construct a cryptosystem where Alice and Bob don't share any secrets? The first result to answer this question was "New Directions in Cryptography," by Whitfield Diffie and Martin Hellman, published in 1976. They do not precisely solve this problem, but solve a similar one with two differences:

1. Alice and Bob may exchange multiple messages (Eve gets to see all the messages), and

2. At the end of the protocol, Alice and Bob have a shared secret. There is no message communicated.

However, part of the idea is that once Alice and Bob share a secret, the problem of encrypting a message becomes much easier. This is called a *key exchange protocol* since the end result is that Alice and Bob have a shared secret key. So, how does the Diffie-Hellman protocol (hereafter referred to as "DH") work? Let k be a common "security parameter."

1. Alice picks a random k-bit prime p.

2. Alice picks a random generator $g \in Z_p^*$, that is, a number $g \in \{1, \ldots, p-1\}$ such that the two sets $\{1, \ldots, p-1\}$ and $\{g, g^2, \ldots, g^{p-1}\}$ are equal.

3. Alice picks a random number $x \in Z_p^*$.

4. Alice sends p, g, and $a = g^x \bmod p$ to Bob.

5. Bob picks a random number $y \in Z_p^*$.

6. Bob sends $b = g^y \bmod p$ to Alice.

7. Alice computes $s = b^x \bmod p$, and Bob computes $s = a^y \bmod p$.

Since $a = g^x$ and $b = g^y$ we see that $(g^x)^y = g^{xy} = (g^y)^x$ so Alice and Bob do agree on their values.

5.1 Efficiency of DH

Since we are in the computationally bounded model, we not only need to make sure that the protocol is correct, but that it is efficient. There are basically three things we need to verify can be done efficiently.

1. Generate a random k-bit prime. Alice does this in step 1. We can certainly generate k-bit numbers efficiently, and the prime number theorem tells us that the probability that a random k-bit number is prime is about $1/k$. However, the problem of determining whether a given number is prime is a difficult one. As it turns out, there are deterministic methods which solve this problem, but probabilistic methods are far simpler. It suffices for our purposes to state that there are algorithms which are RP, for generating a random k-bit prime number, that is, their running time is expected to be polynomial (in 1^k[1], and their output is always a k-bit prime number.

2. Generate a random generator for Z_p^*. This is also solvable efficiently, but the methods are complicated and we need not get into them now.

[1]We write 1^k instead of k because the polynomial in polynomial time is considered on the *length* of the inputs. The length of 1^k is k, so this gives us what we want; on the other hand, the length of k is $\log k$, which is *not* what we want.

3. Given x, y, n, compute $x^y \bmod n$. This is a bit tricky. First of all, it is not polynomial-time to compute x^y and then reduce this modulo n, since x^y is too long for a polynomial-time algorithm to compute.[2] So, we might try this simple algorithm:

ModExp(x, y, n):
1. Set $a := 1$.
2. For $b := 1$ to y do
   2a. $a := a * x$
   2b. $a := a \bmod n$
3. Output a

However, this is not efficient either. Step 2 will run y times, which is not polynomial in the length of y. So, we have to be a little more clever. Let y_i be the ith bit of y.

ModExp(x, y, n):
1. Set $a := 1$.
2. Set $s_1 := x$.
3. For $b := 1$ to $k - 1$ do
   3a. $s_{b+1} := s_b * s_b$
   3b. $s_{b+1} := s_{b+1} \bmod n$
4. For $b := 1$ to k do
   4a. If $y_b == 1$ then
      $a := a * s_b \bmod n$
5. Output a.

You can verify that this is correct if y is no more than k bits long, and that it is efficient.

5.2 Security of DH

The next question is, how secret is g^{xy}, the secret that Alice and Bob share at the end of the protocol? Diffie and Hellman weren't exactly sure, but they couldn't find an efficent way for Eve to break the scheme. So, the formulated the answer to the question of the security of their scheme like this:

"If Eve can efficiently recover the secret, she can solve the following problem: Given p, g, a, and b, compute c such that there is an x and a y where $a = g^x \bmod p$, $b = g^y \bmod p$, and $c = g^{xy} \bmod p$."

Now, if we assume that this problem is hard to solve, there can be no successful adversary. This amounts to Diffie and Hellman saying "Diffie-Hellman is secure if Diffie-Hellman is secure" but they can get away with that since they were the first ones to come up with this kind of scheme.[3]

However, there are two other main shortcomings of DH. First of all, it requires many messages to be exchanged, not just the single one. Secondly, it merely provides a shared secret, but does not allow Bob to send Alice a secret message.

[2] The length of x^y is y times the length of x, which is not polynomial in the lengths of x and y.

[3] This problem is called the Diffie-Hellman problem; the assumption that it is hard is called the Diffie-Hellman assumption, or the DHA.

6 The RSA Cryptosystem

The first improvement on DH was made in 1977 by Rivest, Shamir and Adleman ("RSA" comes from the first letters of their names). In their system, Alice and Bob need not send multiple messages, and Bob has control over their shared secret.[4] However, it is not quite a simple Bob-sends-Alice-one-message system. Instead, Alice comes up with a "public key" which Bob uses to send her message to Bob. However, it is fine if Eve knows Alice's public key. The setup (*key generation*) works as follows:

1. Alice picks two random k-bit prime numbers, p and q. Let $n = pq$.
2. Alice computes $\varphi(n) = (p-1)(q-1)$, the Euler Phi function.[5]
3. Alice picks a random e such that $gcd(e, \varphi(n)) = 1$ (that is, e and $\varphi(n)$ are relatively prime.
4. Alice computes d such that $ed = 1 \bmod \varphi(n)$.
5. Alice publishes n, e as her public key, and keeps d as her secret key.

Now, for Bob to encrypt a secret s, he simply computes $c = s^e \bmod n$. Alice decrypts by computing $s = c^d \bmod n$. To see that the s Alice gets is the same as the s Bob used, we call on a fact from number theory:

Fact 1. For any s such that $gcd(s, n) = 1$[6], $s^{\varphi(n)} = 1 \bmod n$. This is true for any n, actually, not just numbers which are the products of two primes.

Now, since $ed = 1 \bmod \varphi(n)$, there is an integer l such that $ed = l\varphi(n) + 1$, so

$$\begin{aligned}
(s^e)^d &= s^{ed} \\
&= s^{l\varphi(n)+1} \\
&= s \cdot s^{l\varphi(n)} \\
&= s \cdot (s^{\varphi(n)})^l \\
&= s \cdot 1^l \\
&= s
\end{aligned}$$

Encryption, decryption, and key generation are all efficient. The only operation which we haven't examined is the problem that given $e, \varphi(n)$, we must find a d such that $ed = 1 \bmod \varphi(n)$. However, this merely amounts to computing an inverse modulo $\varphi(n)$, which is an easy operation.

[4] In a sense, Bob can communicate the message this way, but for now we assume Bob draws the "message" from a uniform distribution.

[5] $\varphi(n)$ is the number of integers in $\{1, \ldots, n-1\}$ which are relatively prime to n. For $n = pq$, the formula $\varphi(n) = (p-1)(q-1)$ is correct.

[6] This condition does in fact give us a problem if someone tries to encrypt an s which is not relatively prime to n. However, this is very unlikely, so we won't worry about it.

6.1 Security of RSA

How hard is it for Eve to learn s when she sees only c, n, and e? Well, since RSA was one of the very early papers, they too make an analysis of their security similar to that of Diffie-Hellman. Namely, they say:

"If Eve can recover s then there is an efficient solution to this problem: Given n, e, and u, compute v such that $v^e = u \bmod n$."

This problem is called the RSA problem, and this assumption is called the RSA assumption. However, RSA did a bit more than just give this "RSA is secure if RSA is secure" as their security analysis. What they noted was that if Eve can factor n, she can recover the secret. Since factoring large numbers is a well-studied problem which has not been solved, it seems reasonable to assume that factoring is hard. However, there is room for the possibility that factoring is hard and yet the RSA problem is not. (If you were to prove that the RSA problem was equivalent to factoring, you would have an instant PhD!)