CS 290G (Fall 2014) – Introduction to Cryptography Oct 2nd, 2014

Lecture 1: Perfect Security

Instructor: Rachel Lin *Scribe: John Retterer-Moore*

1 Recap

Last class, we introduced modern cryptography and gave a broad overview of a lot of the ideas and concepts used. Today, we took a closer look at the definition of what "security" really means. While the exact meaning may vary depending on context (for example, preserving message privacy vs. message authentication), there are 3 common traits that we look for in a definition of security:

- The definition provides strong guarantees about the security of our information
- The definition is achievable (there is some way to implement it)
- The definition is achievable in a practical, efficient way

The first definition of security we saw was Shannon's "perfect security": when transmitting a message, any 3rd party eavesdropper will learn nothing about the message (except its length) just from intercepting the cipher text. Clearly, this provides some strong guarantees about our information's security and satisfies the first bullet point. Today, we'll take a more formal look at perfect security and see that while it is achievable, it is not achievable in any efficient way.

2 Formal Definitions

We consider there to be three parties: Alice, Bob, and Eve. Alice starts with a message m (the *plain text*), encrypts it using some encryption algorithm E into a different message c (the *cipher text*), and sends it to Bob (Eve, the third party attacker, will be able to see c and attempt to figure out things about the original plain text from that). When Bob receives the cipher text, he'll use a decryption algorithm D to recover the original message.

One of the main principles underlying cryptography mentioned last class was Kerckhoff's Principle: a cryptographic system shouldn't rely on the attacker not knowing the system to be secure. Because of this, we need to use some form of private information keys when encrypting our messages and not simply rely on the attacker not discovering which encryption and decryption algorithms we've chosen. There are two ways we can use keys; in *private key encryption*, both our encryption algorithm and decryption algorithm take a secret key as an additional argument, randomly chosen and known only to Alice and

Bob. *Public key encryption* is similar, but we use different keys to encrypt and decrypt, and the encryption key is made public (for example, RSA). We're primarily going to be considering private key encryption in this course.

Formally, we refer to the set of algorithms Alice and Bob use to communicate as an *encryption scheme*, and define it like this:

An encryption scheme π(Gen, Enc, Dec) with message space M and key space K, consists of 3 algorithms:

- Gen, the key generation algorithm, takes no input and produces a key $k \in K$
- Enc, the encryption algorithm, takes a key $k \in K$ and a message $m \in M$ as input and produces a cipher text c
- Dec, the decryption algorithm, takes a key $k \in K$ and a cipher text c as input and produces a plain text $m \in M$

3 More on Encryption Schemes

One question that arose in class was: if the key generation algorithm takes no input and Alice and Bob run it independently, how can they ensure that they are encrypting/decrypting the message with the same key? The answer was that Gen doesn't really take no input, but rather takes a small input and produces a much larger key as an output. Alice and Bob have to meet each other once to communicate this shared small input, but their goal is to get a much larger key from that shared small input and to be able to communicate securely without meeting for many messages in the future.

Which of the algorithms in an encryption scheme are randomized, and which are deterministic?

- Gen, the key generation algorithm, must be randomized. Otherwise our encryption scheme relies on security by obscurity.
- Dec, our decryption algorithm, must be deterministic. If the same cipher text c and the same key k could be decrypted to two distinct values, one of those would not be the original message Alice encrypted and our decryption algorithm could give us the wrong message.

Enc, the encryption algorithm, is a little more complicated. It could be deterministic - for example, the Caesar cipher is fully deterministic given a message and a key (the key being how many letters to shift by). However, a deterministic encryption algorithm will cause certain problems. If we encrypt two identical messages deterministically, they

will get mapped to the same cipher text. This means that if an attacker is watching and knows our encryption scheme, she will know that we sent the same message twice. This could be a huge problem (imagine you're a military general, and you know your enemy is planning to attack either Cape Horn or Bora Bora - knowing that the two words they sent in a message are the same tells you exactly which destination the enemy will attack!) So deterministic algorithms will need a new key for every message they send, causing practical concerns, and most encryption algorithms we see in this class will not be deterministic. Instead, the same message-key pair (m, k) may get encrypted as any number of different cipher texts c_1, c_2, ... c_n (but all of those will decrypt to m)

4 Goals of an Encryption Scheme

We've talked in general about what we want our encryption scheme to do, but how do we formally define these goals? Using perfect security as our definition of security, there are two properties we want from our encryption scheme.

1. Correctness

$\Pr[\ k \leftarrow \text{Gen} \quad c = \text{Enc}(k,m) \ \ \text{Dec}(k,c) = m\ ] = 1$
This means that if we encrypt any message with a random key, our decryption algorithm must be guaranteed to give us back the original message.

2. Perfect Security

An encryption scheme π is perfectly secure if $\forall m_1, m_2 \in M$ $\{k \leftarrow \text{Gen} \quad c_1 = \text{Enc}(k,m_1) : c_1\} = \{k \leftarrow \text{Gen} \quad c_2 = \text{Enc}(k,m) : c_2\}$ (The first formula refers to the distribution of all cipher texts c_1 reached by randomly choosing k and encrypting message m_1, the second is the same but for m_2)

So any two messages need to lead to the same distribution of cipher texts when encrypted, or in other words $\forall$ cipher texts c, $\Pr(c_1 = c) = \Pr(c_2 = c)$.

Another way of looking at this is that if you're given a cipher text c and told that it was either encrypted from m_1 or m_2, you shouldn't be able to tell which one.

5 Perfectly Secure Encryption Schemes

Two of the early encryption schemes we learned about, the Caesar cipher and the substitution cipher, clearly fail at being perfectly secure. If the first two letters of m are the same, the first two letters of c will be as well, and if they're different, the first two letters of c will be different, so for example, aa and ab will have completely different

distributions of cipher texts, violating perfect security. However, if we only encrypt one letter with a Caesar cipher, that's completely secure. Whatever letter we encrypt, there's a $\frac{1}{26}$ chance of it turning into each of the letters of the alphabet. Similarly, if we encrypt each letter with a different Caesar cipher, it will be perfectly secure. This leads us to a perfectly secure encryption scheme:

The One Time Pad

Definition:

- $M = K = \{0,1\}^n$
- Gen is chosen uniformly at random from the key space
- $E(m,k) = m \oplus k$
- $D(c,k) = c \oplus k$

The one time pad is correct because $c \oplus k = m \oplus k \oplus k = m$.

Proof that the one time pad is secure:

Take any two messages m_1, m_2. For any cipher text which is not an n-bit binary string, both m_1 and m_2 have no chance of being encrypted as that string. For each cipher text $c \in \{0,1\}^n$, $E(m_1,k) = c$ iff $k = m_1 \oplus c$, which has a $\frac{1}{2^n}$ chance. The exact same result holds for m_2 as $k = m_2 \oplus c$ also has a $\frac{1}{2^n}$ chance. Therefore, both m_1 and m_2 will have a uniform distribution of their cipher texts and the one time pad is perfectly secure.

6 Bad News

Unfortunately, one time pad requires a key of length equal to the message, which defeats the purpose to some extent; if Alice and Bob could securely exchange an n-bit key, why not just securely exchange an n-bit message? Even more unfortunately, this is not unique to one time pad.

Theorem: $|K| \geq |M|$ in any perfectly secure encryption scheme.

Proof: Assume not. Then take any message m_1 and key k and let $c = E(m_1,k)$. Now, let $S_D(c)$ be the set of all messages that can be decrypted from cipher text c. Since D is deterministic, $|S_D(c)| \leq |K| < |M|$, meaning that some message m_2 is never encrypted as c for any choice of key. But this means that m_1 and m_2 have different distributions of cipher texts, since m_1 has some positive probability of being encrypted as c and m_2

has no chance of being encrypted as c. Thus, if $|K| < |M|$, our encryption scheme is not perfectly secure, violating our assumption.

For this reason, perfect security is an unrealistic goal. Instead of information theoretic absolute security, we'll be looking more at schemes which are hard to break because of computational difficulty. We'll take some known hard problem such as factoring and transform it into a primitive such as an encryption scheme so that if the encryption scheme were broken, the attacker would also be able to solve the hard problem, which is not believed possible.