COM S 6830 – Cryptography Sep 1, 2009

Lecture 2: Information Theoretic Security

Instructor: Rafael Pass *Scribe: Stefano Ermon*

1 Review

Alice wants to communicate privately with Bob over an insecure channel. There is a passive adversary, called Eve, that can listen to the messages on this channel but cannot interfere. To keep the communication private Alice and Bob use a private key encryption scheme:

Definition 1 (Private Key Encryption) *A triplet of algorithms (Gen,Enc,Dec) is called a private key encryption scheme over the message space M and keyspace K if the following conditions are met:*

1. *Gen is a randomized algorithm $k \leftarrow Gen$ s.t. $k \in K$*
2. *Enc is a (randomized) algorithm that takes a key $k \in K$ and a message $m \in M$ and outputs a ciphertext $c = Enc_k(m)$*
3. *Dec is an algorithm $m' = Dec_k(c) \in M$ such that for all $m \in M$ and $k \in K$, $Pr[Dec_k(Enc_k(m)) = m] = 1$*

2 Definition of secure communication

What does it mean for an encryption scheme to be secure?

2.1 Intuitive definition

Some flawed candidate properties are

1. an adversary cannot recover the key k. (*The message should be protected instead. An identity encryption algorithm satisfies this property but is not secure.*)
2. an adversary cannot recover the message m. (*Eve might still be able to recover a part of it*)

3. an adversary cannot recover any individual bit of the message m. (*Eve might still be able to recover information about a bit's probability distribution*)

4. an adversary cannot recover any partial information about the message m. (*Too strong. Eve might have a priori knowledge about* m)

A correct intuitive definition of security is

An adversary should not be able to recover any probabilistic information on m *besides what she already knew a priori.*

2.2 Shannon's definition

Definition 2 *A private key encryption scheme (Gen,Enc,Dec) over the message space* M *and keyspace* K *is* perfectly secure *if for all* $m_1, m_2 \in M$*, for all ciphertexts* c

$$Pr[k \leftarrow Gen : Enc_k(m_1) = c] = Pr[k \leftarrow Gen : Enc_k(m_2) = c]$$

This definition requires that the encrypted ciphertexts obtained by encrypting any two messages must have the same probability distribution.

Caesar and *Substitution* ciphers do not satisfy this definition unless the messages are one letter long.

$$\begin{aligned} AA &\rightarrow XX \\ AB &\rightarrow XY \quad , X \neq Y \end{aligned}$$

2.3 Vernam's OTP

Definition 3 *The One-time pad (Vernam's OTP) encryption scheme is defined as follows:*

$$\begin{aligned} M &= \{0,1\}^n \\ K &= \{0,1\}^n \\ Gen : k &= k_1 \dots k_n \leftarrow \{0,1\}^n \textit{ uniformly at random} \\ Enc_k(m_1 \dots m_n) &= c_1 \dots c_n \quad \textit{where } c_i = m_i \oplus k_i \\ Dec_k(c_1 \dots c_n) &= m_1 \dots m_n \textit{ where } m_i = c_i \oplus k_i \end{aligned}$$

Proposition 1 *OTP is perfectly secure.*

Proof. Consider any message $m \in \{0,1\}^n$, string $c \in \{0,1\}^*$.

Claim 1 *If* $c \in \{0,1\}^n$ *then* $Pr[Enc_k(m) = c] = 2^{-n}$

Claim 2 *If* $c \notin \{0,1\}^n$ *then* $Pr[Enc_k(m) = c] = 0$

Given any $m \in \{0,1\}^n$ and $c \in \{0,1\}^n$, there exists an unique k s.t. $c = m \oplus k = Enc_k(m)$, therefore $Pr[Enc_k(m) = c] = 2^{-n}$.

If $c \notin \{0,1\}^n$, then there is no key $k \in K$ such that $Enc_k(m)$ will generate c.

Given any two messages $m_1, m_2 \in M$, $|m_1| = |m_2$ and therefore

$$Pr[k \leftarrow Gen : Enc_k(m_1) = c] = Pr[k \leftarrow Gen : Enc_k(m_2) = c]$$

because either they are both 0 or both equal to 2^{-n}.

The practical drawback of this approach is that Alice and Bob need to share an extremely large key, with as many bits as the message they intend to exchange. However the following theorem shows that it is not possible to achieve perfect security with a shorter key length.

2.4 Shannon's Theorem

Theorem 1 *If an encryption scheme* (Gen, Enc, Dec) *over the message space* M *and keyspace* K *is perfectly secure, then* $|K| \geq |M|$.

Proof. Suppose there exists a perfectly secure encryption scheme with $|K| < |M|$. Pick any message $m_1 \in M$ and let $c = Enc_k(m_1)$ for some key $k \in K$ that can be generated by Gen with nonzero probability. Since $|\cup_k Dec_k(c)| \leq K$, there exists a message $m_2 \in M$ such that $m_2 \neq Dec_{k'}(c)$ for any $k' \in K$. By definition of an encryption scheme this implies that

$$Enc_{k'}(m_2) \neq c$$

for any $k' \in K$. Therefore

$$Pr[k \leftarrow Gen : Enc_k(m_2) = c] = 0$$

By construction we have that

$$Pr[k \leftarrow Gen : Enc_k(m_1) = c] > 0$$

and therefore by definition such an encryption scheme cannot be perfectly secure.

This proof technique also suggests a possible way of attacking any encryption scheme with $|K| < |M|$. In fact given any ciphertext c, an adversary could decrypt c with

all possible keys, and then rule out all messages that could not have generated c. The attacker should then choose one of the remaining messages, possibly according to an a priori belief. This strategy can lead to very efficient attacks.

For example, consider a scenario where Alice picks a message m from $\{m_1, m_2\} = \{YES, NO\}$ uniformly at random and sends the corresponding ciphertext c to Bob. Eve, upon receiving c decrypts it with all possible keys to build $Dec(c)$ and then checks if $m_2 \in Dec(c)$. If $m_2 \notin Dec(c)$, Eve guesses that $m = m_1$, otherwise she makes a random guess. With this strategy Eve's success probability is

$$\frac{1}{2} + \frac{\epsilon}{2}$$

If ϵ is very small the attack is not very useful. However it can be shown that if the keys used are one bit shorter than the messages then $\epsilon = 1/2$.

This kind of attack requires a lot of computational time, but in a realistic setting adversaries can be assumed to be computationally bounded. For this reason we will restrict ourselves to efficient adversaries, that cannot use this kind of brute force attacks.

3 Computationally efficient adversaries

3.1 Deterministic

Algorithm = *Turing Machine* (TM) or *Random Access Machine* (RAM) where Inputs and Outputs are strings over a binary alphabet $\Sigma = \{0, 1\}$.

An algorithm $\mathcal{A}$ is said to **compute** a function $f : \{0,1\}^* \to \{0,1\}^*$ if $\mathcal{A}$ outputs $f(x)$ when the input is x.

An algorithm $\mathcal{A}$ **runs** in time $t(n)$ if $\forall x \in \{0,1\}^*$, $\mathcal{A}(x)$ halts within $t(|x|)$ steps.

An algorithm $\mathcal{A}$ runs in **polynomial time** if there exists a *constant* c such that $\mathcal{A}$ runs in time $t(n) = n^c$.

An algorithm is said to be **efficient** if it runs in polynomial time. Why? Philosophical question. Possible answers:

1. Definition is machine independent (RAM, TM, etc are polynomial time reducible)
2. Usually polynomial time turns out to be efficient in practice
3. Closed under composition
4. Natural functions that are known to be not computable in polynomial time require much more time to compute (close to be exponential)

In the course we will deal with **asymptotic security** (constant factors are ignored), but usually they can be turned into concrete security schemes.

Examples of polynomial time computable functions:

- multiplication $MULT(x, y)$
- division $DIV(x, y) = (q, r)$ s.t. $x = qy + r$
- greatest common divisor GCD
- $MODEXP(x, y, z) = x^y \ mod \ z$

Problems known or believed to be hard:

- halting problem (*uncomputable*)
- time hierarchy theorem (computable, not in polynomial time)
- SAT (belief, as other NP-complete problems)
- factorization (belief)

3.2 Randomized

It is easy to generate **random bits** in practice (flipping a coin), and randomization is necessary in the key generating algorithm Gen. We want to model this extra freedom.

A **randomized algorithm** is a Turing Machine with access to a tape with truly random bits, chosen uniformly and independently.

A randomized algorithm $\mathcal{A}$ **runs** in time $t(n)$ if $A(x)$ halts within $t(|x|)$ steps, independently of the random tape.

$\mathcal{A}$ is said to be **efficient** if it runs in polynomial time (*probabilistic polynomial time*, PPT).

A randomized algorithm $\mathcal{A}$ is said to **compute** $f : \{0,1\}^* \to \{0,1\}^*$ with probability $p(\cdot)$ if for all x, $Pr[\mathcal{A}(x) = f(x)] \geq p(|x|)$.

Proposition 2 *If a PPT algorithm $\mathcal{A}$ computes f with probability $p > 1/2$, then there exists a PPT algorithm $\mathcal{A}'$ such that $\mathcal{A}'$ computes f with probability $1 - 2^n$.*

$\mathcal{A}'$ is constructed by iterating the execution of $\mathcal{A}$ and by taking a majority output. The proof relies on the use of the Chernoff bound.

Hard problems for randomized algorithms:

- halting problem
- time hierarchy theorem (not known if still holds)
- SAT ($NP \subseteq BPP$)
- factorization (belief)