Principle of Information & Security

Siddharth Bhat

Contents

Chapter 1

Introduction

1.0.1 Impossiblity of Infosec problems

Common aspect across all infosec problems to date is that it is impossible to solve.

- Password schemes - It is impossible to design a good password scheme. The machine must know something about the password you need to give. Call it the password file (TODO: how to do monospace?)
- Password Length - everlasting is impossible. One can always brute force passwords. Infinite length passwords do not work.
- Secure communication over insecure channels
- Signing
- Digital cash

TODO: learn TIKZ

Secure communication over insecure channels

---

sender —¿ key —¿ receiver — v adversary

At time t0, everything that receiver knows, adversary knows (assuming no one-time pad). After that, everything the receiver receives, the adversary also knows as well. So, the adversary has all information that the receiver does.

It is impossible to do secure communication over insecure channels.

Signing

Digital signature is impossible - Unforgable digital signature should not exist.

1. Signature should be a function of the message for it to be useful as a signature. Otherwise, an attacker could intersect messages to find the signature. 2. Signature must be publically verifiable. 3. A trapdoor function can be reverse-engineered.

Digital cash

How do we detect counterfeit cash? Double spending is a problem. Cryptocurrencies used the exact same mathematical methods that are shared across crypto.

1.0.2 A Tom and Jerry analogy

Tom & Spike are both Jerry's opponents. So, Jerry is able to play Tom and spike against each other, and have them beat each other.

That is, pair adversaries against each other to have them screw with each other.

Password schemes, take 2

We needed infinite length passwords because an adversary will win if we have finite length password. However, there are other adversaries. For example, the adversary for algorithms is the person who provides inputs. Example, think of sorting networks or uses of bubble sort: Sorting networks are useful on small numbers of elements to sort. Bubble sort does not screw with cache coherence. However, these are both bad solutions *in general*.

When the worst case input giver is an adversary, and a person who is trying to crack our password is an adversary, we can have these two interfere.

To find out ‘y = f(x)‘, we wind up using the algorithmic adversary who provides hard problems for ‘f‘.

Structure of information matters. Example, linked list v/s balanced tree. The process of decryption can exploit structure of information.

eg: Natural number can be represented as a product of primes, and in the decimal notation.

Active adversary / noise

We cannot design error detection codes for any amount of error. Hence, if we think of adversary as error in the stream, we can think of secure communication on a channel with an active adversary as ECC.

So now, this problem is now an information theory problem.

The adversary must make a modification such that the bank cannot detect it. Coding theory tells us that such a modification is always possible. Infosec tells us that we can design schemes where this takes a long time.

Chapter 2

Lecture 2 - More philosophy - Amazing Advantages of Additional Adversity

Textbook is

- Introduction to Modern Cryptography

2.0.1 Ceasar Cipher

rotate letters by a certain amount.
crypto goes to FUBSWR.

2.1 Kerckhoff's Principle

Security of system depends on secrecy of the key and not on the obscurity of the algorithm.

2.1.1 Password Shadows

Password is $\{x\}$, we store $\{f(x)\}$.
It is possible to reverse-engineer f to discover x. So, we should not depend on f being secure.

2.1.2

2.1.3 Shift Cipher

We can brute force this, we can brute force keys.
Principles learnt from shift ciphers

- Key space needs to be large. for shift cipher, key space is 26.

p_i probability of letter in plaintext. q_i probability of letter in ciphertext.
$\exists delta, \forall xinLetter, p_i = q_{i+k}$
$pi \cdot p_{i+k} = p_i^2$ if we wind the right k. So, we need to find the right k.
So, large key space is not enough. We need to ensure that frequency is also fudged.

2.1.4 Monoalphabetic Substitution Cipher

Create a bijection $\{f : Letter \leftarrow Letter\}$. This has a large key space, $\{26!\}$.

Attack is based on frequency. $\{\forall x \in Letter, freq(x) = freq(f(x))\}$. So, one can match x with $f(x)$.

Again, we need to fudge frequency.

2.1.5 Polyalphabetic sustitution cipher

This needs a passphrase, for example, Cat.

Add passphrase to plaintext.

$crypto + catcat = \cdots$

Frequencies are not maintained, because different text is added each time to the same plaintext.

- Step 1 - Given length, we break the cipher.
- Step 2 - Length is susceptible to brute force attack.

Breaking given length

Assume the length of passphrase is known, say, k.

Let ciphertext be $c_0c_1c_2c_3c_4...$ Let us look at ciphertext at lengths of 3.

This will give us a *shift cipher*, since the text is all shifted by the *same* letter in the passphrase. Now, we can perform the frequency attack.

If we screw up the partition, then the frequency spectra will be gibberish. zsh:1: command not found: :w

What we learnt by breaking

This was also broken. So, they learnt that "security is hard, forget it!". Or, complication does not imply security.

2.1.6 What is an Unbreakable cipher? Or, shannon enters the scene

A preamble, the thought process

- We need to specify what it means to have a good cipher. Where do we stop? We need a formal spec. (Definition of security).
- Precise assumptions involved must be known. (Hardness assumption).
- The truth of security, and the trade-offs involved (Shannon's Proof)

Chapter 3

Information Theory

3.0.1 shannon's perfect secrecy (1949)

Shannon framed a secrecy theory based on information theory. If no information is revealed to the other person, it is secure.

Cipher $= < Gen, Enc, Dec, M >$.

M is the message space.

$Gen :: KeyLength- > Key$. Set of all keys Gen can output ($Image(Gen)$) is called the key space. Key space (K)is asked to be finite.

$Enc :: M- > K- > C$. $C = ciphertext$. $Image(Enc)$ is called the ciphertext space. Enc takes a message and a key, and returns a ciphertext.

$Dec :: C- > K- > M$. such that $\forall(m : M)(k : K)Dec(k, Enc(m, k)) = m$.

A cipher scheme is secure iff: $\forall p =$ probability distributions over the message space (we don't know the exact probability distribution over plaintext). $\forall m \in M, \forall c \in C, P(C = c > 0) => P[M = m] = P[M = m|C = c]$.

What we know about the message before looking at the ciphertext is the same as what we know about the message ∋Ʊ≈∖ we know the ciphertext. We make sure that we do not take degenerate c (c that does not ever occur) to prevent nastiness in conditional probability.

3.1 Shannon and secure channel capacity of systems

Sender———* — — — — insercure channel secure channel(1Mbps) (1 Gbps) — v — Receiver¡— —*

If I have perfect security, what is the bandwidth of the ≾≃⋊⋖∼⌒∼≈⋗?

It must be between 1Mbps and 1Gpbs + 1Mbps. (minimum is 1Mbps).

If we have only an insecure channel, then set secure channel capacity to 0.

Shannon ı∖⋊≳ that the secure channel capacity of the ≾≃⋊⋖∼⌒∼≈⋗ is 1Mbps (that of the secure channel).

also, if the secure channel has bandwidth 0, then it is impossible to have security.

3.2 Proof

3.2.1 First equivalence

A cipher is perfectly secret iff $\forall m \in M, \forall c \in C, forallprobabilitydistributionsoverM, P[C = c|M = m] = P[C = c]$

Proof

TODO: how to get aligned text.

$P[C = c|M = m] = P[C = c]$ $P[C = c|M = m] * P[M = m]/P[C = c] = P[C = c] * P[M = m]/P[C = c]$

Reminder: $P[A|B] = \frac{P[B|A]*P[A]}{P[B]}$

$P[M = m|C = c] = P[M = m]$

Qed. (TODO: how to get box)

3.2.2 Second equivalence

A cipher is perfectly secret iff $\forall m_0, m_1 \in M, P[C = c|M = m_0] = P[C = c|M = m_1]$.

Proof (=¿ directiion)

If the cipher is perfectly secure, $P[C = c|M = m] = P[C = c]$ (from first equivalence).

$P[C = c|M = m_0] = P[C = c]$. $P[C = c|M = m_1] = P[C = c]$.

Hence, $P[C = c|M = m_0] = P[C = c|M = m_1]$ *Qed.*

Proof (¡= directiion)

Given $\forall m_0, m_1 \in M, P[C = c|M = m_0] = P[C = c|M = m_1] = p$ $P[C = c] = \sum_{m \in M} P[C = c|M = m] * P[M = m]$ $P[C = c] = \sum_{m \in M} p * P[M = m]$ $P[C = c] = p \sum_{m \in M} P[M = m]$ Since we are summing over probability space, $P[C = c] = p * 1$ $P[C = c] = P[C = c|M = m]$ for any m. Qed.

The reason it's all p is because of transitivity. $M_0 = M_1$, $M_1 = M_2$, hence everything is equal.

3.3 Is there a scheme that exists that is perfectly secure?

3.3.1 One time pad

$Gen : k = 0,1^n$ $P[K = k] = 1/2^n$. $Encrypt(m, k) = kXORm.m \in 0,1^n$. $Decrypt(c, k) = kXORc.c \in 0,1^n$.

Perfect security of one time pad: proof (Vernam cipher)

We will show this by using the phrasing: A cipher is perfectly secret iff $\forall m_0, m_1 \in M, P[C = c|M = m_0] = P[C = c|M = m_1]$.

$P[C = c|M = m_0] = P[C = kXORm_0] = P[K = cXORm_0] = \frac{1}{2^k}$ $P[C = c|M = m_1] = P[C = kXORm_1] = P[K = cXORm_1] = \frac{1}{2^k}$

We pick the key independent of the message, so it doesn't matter what the message is.

Limitations of one-time-pad

.

Since we need to send the key securely, we will need to send the key over the slow secure channel. We can send the message over the insecure channel. However, to decrypt the *nth* insecure bit, we need the *nth* secure bit. So, for this, we might as well send message over the secure channel.

However, if both the secure channel and the message are available at different times, then one-time-pad is useful. We can send the key over the secure channel, and use it later to decrypt a message sent over the insecure channel.

Next, if sender = receiver, then it makes sense to have one-time-pad. The channel of internal transfer should be very fast (eg. memory transfer is fast, versus network transfer is slow).

3.4 Every perfectly security scheme is isomorphic to one-time-pad.

3.4.1 Theorem

$\forall$perfectly secret cipher, $|K| \geq |M|$.

Note on bit sizes

It is possible that $|K| \geq |M|$, but $nbits(K) \leq nbits(M)$. That is, the number of bits needed to store the space can be smaller than the space (low entropy).

3.4.2 Proof

Suppose for contradiction $|K| < |M|$.

One ciphertext c can be decrypted into at most $|K|$ messages. In the message space, there must be one message $M*$ that is not part of the decryption of c (since $|K| < |M|$).

$P[M = m * |C = c] = 0$ since if $c = c$, $m*$ cannot occur. However $P[M = m*]$ is non-zero.

Hence, $P[M = m * |C = c] = P[M = m*]$.

Shannon further proves that the entropy of the key space must be greater than the entropy of our message space. Hence, we will need to send as much data over the secure channel as long as the key, usually.

3.5 Tangent: Entropy, Expectation, random kannan

3.5.1 Expectation

$E[x] = \sum_x x \cdot p(X = x)$

3.5.2 Entropy

There can be many indexing schemes to store data. $M = m_0, m_1, \cdots, m_n$. We can use $log(n)$ bits to store the index.

What is the expected number of bits to store a message space with n messages? Say m_i occurs with probability p_i.

$E[\#bitstostoreM] = \sum p_i \cdot ixlength(m_i)$

The ones where p_i is high, we want $ixlnegth_i$ to be small for an efficient compression scheme. Index the thing that occurs most often with the least bits.

We can represent the $\lessdot \gtrdot \ltimes \approx m_i$ in terms of p_i (how often it occurs in the space). We can make messages that occur more frequently with strings of smaller length.

Eg: for a message with $p = \frac{1}{2}$, use 1 bit to represent ix. Eg: for a message with $p = \frac{1}{4}$, use 2 bits to represent ix. Eg: for a message with $p = \frac{1}{n}$, use n bits to represent ix. Eg: for a message with $p = k$, use $\frac{1}{k}$ bits to represent ix.

$E\#bitstostoreM = \sum_i p_i \log(\frac{1}{p_i})$. $E\#bitstostoreM = -\sum_i p_i \log(p_i)$. Entropy $= -\sum_i p_i \log(p_i)$.

3.6 Looking at the future (next lecture)

Can we actually fully utilise the insecure channel by relaxing our definitions of secure?

3.7 Information Theory - Lecture 4

- Two famous relaxations
- Modern approximations
- Modern definition of Security

3.7.1 Shannon's perfect secrecy

Perfect secrecy assumes secrecy required for infinite time. We can relax this to be some large number.

However, if we accept this, then the scheme *will be breakable* by brute force since our key space is finite. This naturally forces another relaxation:

We allow a small error term of probability in terms of failure of secrecy.

3.7.2 Representation Change

Pick two representations of the same information. Converting from one (say A) to the other (say B) is easy, but converting back from B to A is hard.

$P \neq NP \implies$ **existence of trapdoor function**

Verifying an NP complete problem will be in P. Computing it will be NP.

If we have a certificate, then a non-deterministic turing machine can solve in polynomial time by guessing the certificate.

Other direction, if there is a non-deterministic turing machine that can solve in polynomial time implies a certificate, because the "path" in the non-deterministic TM will be the polynomial time certificate vertification.

Using the relaxed definition

We have a secure channel and an insecure channel, how do we improve bandwidth?

the field divided into two: - Assume we have a slow secure channel and a fast insecure channel, how do I create a fast secure channel? (Slow secure + fast insecure =? Fast Secure) / Private key crypto.

- (No secure + Slow insecure =? Slow secure) / public key crypto.

3.7.3 Formalization

If the probability that any adversary can win the game is $\frac{1}{2}$.

$$\forall adversary, P[b' = b] = \frac{1}{2}$$

For all probabilistic polynomial time turing machines A, if A interacts with a protocol in the game, the prob. that the output of the game will be b, is bounded by $1/2 + \mu$, where μ is negligible, then the game is secure.

negligible

A function μ is said to be negligible if:

$\forall p \in polynomials, \exists n_0, \forall n \geq n_0, \mu(n) \leq \frac{1}{p(n)}$.

This is equivalent to saying that $\mu \leq 2^k$ because 2^k will always outgrow any polynomial p.

If the adversary has some non-negligible chance of doing better than 1/2, then he can repeatedly reapply the strategy some polynomial number of times to "blow up" the advantage (see: randomized algorithms).

How close to 1 we can get by re-running is how away from $\frac{1}{2}$ we are.

Roughly, by repeating M times, we can push it to $\frac{1}{2} + M\mu(n)$.

However, if a function is negligible, polynomial times multiplication with negligible will continue to be negligible. It's some weird ideal in $R[X]$?

3.7.4 The game to be played

1. Fix message space with all messages of equal length. 2. The adversary chooses two message of his choice, M0 and M1. 3. We encrypt one of the messages, call it C 4. The adversary has to find out which one (C = encrypt (M0) or C = encrypt(M1))

The adversary wins if the adversary does not satisfy the security criteria.

3.8 Information Theory - Lecture 5

Most theorems will read as: if X is true, then the protocol Π is secure.

3.9 Our first example of circumventing an impossibility

3.10 PRNGs - Pseudo random number generators

This allows us to break $|K| \geq |M|$. This is still a one-time pad, but it allows us to create $|K| << |M|$.

Deterministic program G. Takes as input n-bit string, returns $l(n)$ bit string. We have two assumptions.

- 1. $l(n) > n$. Expansion.
- 2. Pseudorandomess.

Pseudorandomness

for all PPTM(probabilistic polynomial turing machine) D, $|P[D(r)] - P[D(G(s))]| <= negligible(|s|).r = 0,1^{l(n)}.s = 0,1^n$

1. Strings of length $l(n)$. pick one at random. probability of picking one of them is

2. Strings of length n, and then we inject into $l(n)$ with G. Clearly, $|Im(G)| < 2^{l(n)}|$. So, we can sample all $|Im(G)|$. If we are in a pseudo-random world, it will repeat for sure (with $P = 1$). If we are in the non-PRNG world (true randomness), the chance that something repeats will be negligibly small.

we cannot distinguish with polynomial samples, however. So PPTM is a good choice for a distinguisher.

Given that we have to assume PRNGs exist, there are different ways to proceed

- Heuristics - Assume that the PRNG we write is a true PRNG, and then get to work.
- Specific mathematical assumptions - Assume that certain problems are hard. Build PRNGs from this mathematical assumption.
- Provable Security - If there exists even one hard problem P, then we can use that to build a PRNG.
- Proven security - prove PRNGs exist.

Assume PRNGs exist. We will build a secure encryption scheme

$M = 0,1^{l(m)}.K = 0,1^m$. $Gen : k = 0,1^n$ $Enc_k(m).|m| = l(n)$. $Ciphertext = mxorG(k)$. $Dec_k(c) = cxorG(k)$.

Note that this is just one time. If they attacker can see two ciphertexts, they can XOR the ciphertexts to get the XOR of the cleartexts.

Proof that this is sane . If the adversary can differentiate between M_0 M_1, we will use it to break the PRNG (as in, distinguish between PRNG and RNG). , Call the adversary A. It can generate 2 messages M_0 and M_1. When given $encryption(M_b) = G(k)xorM_b$, he can guess $b = 0b = 1$ with non-negligible probability.

Call the distinguisher D. D has to distinguish between truly random and pseudo random world for our proof. Given a string w and ask if w can be distinguished by A. We can pick w from the PRNG world or the RNG world.

If $A(wxorM0, wxorM1)$ gives us the correct value(can distinguish), then we are using the PRNG. Otherwise, it is the RNG.

CPA secure

Adversary gets to pick M_0 and M_1, we choose a bit b at random and give encryption of M_b. He has an oracle that has oracle access to the encryption algorithm. Even with this, he should not be able to guess b.

No determistic algorithm can be CPA secure The adversary will ask for encryption of M_0 and encryption of M_1. He gets back C_0 and C_1. Then, we can compare that to our result, and find the random bit b.

How to create CPA secure $C =< R, cxorenc(R) >$ where R is a random string. Decryption will never fail. if we know R, we can xor twice. Encryption will not fail because encryption of random data is still random.

We have a problem of length doubling: For one length of data, we need R as well.

Indexable PRNGS

A PRNG that we can index at a point, and it will start generating from that index. They are called "pseudorandom functions".

Consider Z/pZ^x. All numbers except 1 in Z/pZ are generators.

Discrete log: Given $g^x modp$, given g, given p, find x. (log in a group). We know that Discrete log is hard. Let us try and build a PRNG.

Step 1. Given a PRNG that expands 1 bit, we can use it to create a PRNG that expands any number of bits n $s = seed$. $G(s), G(G(s)), G(G(G(s))), G^n(s)$, take the extra bits from each $G^i(s)$. This is a PRNG.

This is a PRNG.

Assume we can break this PRNG. $s_1s_2...s_n$ = stuff from PRNG is distinguishable from $r_1r_2r_3...r_n$ = Random info.

Construct $s_0s_1s_2..s_n$, $r_0s_1s_2..s_n$, $r_0r_1s_2s_3...s_n$. $r_0r_1r_2r_3..r_n$. We know that we can distinguish first from last. Hence, there must be an adjacent set of strings that can be distinguished, since "distinguishable" is transitive (why?) so, if r_idistr_{i+2}, we need to have either r_idistr_{i+1} or $r_{i+1}distr_{i+2}$. However, between these strings, we have only edited s_i. So, we are able to distinguish one bit extra. This means we can actually distinguish the output of G.

Step 2. if we can find $MSB(x)$, we can find x in polynomial time. So, all we need to do is to break $MSB(x)$.

Step 3. Create PRNG that produces one bit output using discrete log.

Take seed s. output $MSB(s_1 = g^s mod p)$. So we now have a PRNG that can create one bit. Second output: $MSB(s_2 = g^{s_1} mod p)$ Third output: $MSB(s_3 = g^{s_3} mod p)$.

Hence, if discrete log is hard, we can get a PRNG.

3.11 Information Theory - Lecture 5

3.12 Our first example of circumventing an impossibility

One way function: hard one way, easy the other Trapdoor one way: One way function with a trapdoor that makes the hard way easy with the key.

3.13 Exploring discrete log problem

Take a seed, that is a member of Z_p^x.

Construct the following sequence of bits: $[MSB(x1)\ MSB(x2)\ \ldots\ MSB(x_i)]$

`||` = concatenation $x_j = g_{j-1}^x$ in Z_p^x

Given $(g^x mod p, p, g)$ what is the $MSB(x)$? Is this actually as tough as trying to find x?

3.13.1 Theorem: $LSB(x)$ is easy to get

Proof

Fermat's little theorem: $\forall x \in Z_p^x, x^{p-1} = 1$

Proof of fermat's little theorem (raw number theory)

Everything happens in x^{p-1}.

$S = a, 2a, 3a, ...(p-1)a$. We show that this is a permutation of $S' = 1, 2, 3, ..., (p-1)$

$a \neq 0$. So, $a \cdot x = 0 implies x = 0 since Z_p^x is integral domain$

Suppose two elements are not distinct in S. This means that $ai - aj = 0$ Hence, $p|a(i-j)$. But, $a < p$, $(i-j) < p$. Hence, their product cannot be divisible by p (product of two numbers less than a prime numbers.

Multiplying all numbers in S should be equal to multiplying all numbers in S'

$a^(p-1)(p-1)! = (congruent mod p) = (p-1)!$ Hence $a^(p-1) = (congruent mod p) = 1$

Continuing proof of LSB of discrete log is easy

$(g^x)^(p-1) = 1$ $(g^x)^{\frac{p-1}{2}} = +-1$

When x is even, this will be $+1$. When x is odd, this will be -1

In some sense, we are computing $(g^x|p)$ (legendre symbol)

Hence, we can find $LSB(x)$. Note that this will fail if $g^(p-1)/2 = 1$, but g is a generator of Z_p^x so it can't happen (g has order —p - 1—).

3.13.2 Can we not use this to "peel bits" off? We can peel more than just LSB

If $4|p-1$, then we can reapply the same method to get *two* bits.

$g^{\frac{p-1}{4}} =$

$1. if x == 0 mod 4 -> 1$ $2. if x == 1 mod 4 -> ?$ $3. if x == 2 mod 4 -> ?$ $4. if x == 3 mod 4 -> ?$

3.13.3 Hardness given ability to get MSB

Assume there is an algorithm to find $MSB(x)$ given $y = g^x$ (everything in Z_p)

We want to find $sqrt(y)$. That is, it finds z such that $z^2 = y$. Suppose sqrt(y) *does exist.*

Note: algorithm to find roots of polynomial in a FF efficiently (?) Look this up. If we have this, we can nuke this problem.

SQRT-WHEN-SQRT-EXISTS(y): Compute $a = y^{\frac{p+1}{4}}$. $a^2 = y^{\frac{p+1}{4}})^2 = y^{(\frac{p+1}{2}}$).

We know that y is a quadratic residue, so $y = g^2k$ So, $a^2 = (g^2k^{\frac{p+1}{2}}) = g^{(}k * \frac{p+1}{)}$ ¡Lost, do the arithmetic yourself¿.

`x -> x/2` if x is even `x -> x - 1` if x is odd.

If we have the trace of the function fixpoint (0), then we can reconstruct x.

Going to $x/2$ is difficult because we have two square roots in Z_p^x.

Brilliant:

If we have an MSB algorithm, then this step can be *made unique*. If the number is between $0..(\frac{p-1}{2})$, then MSB = 0. Otherwise, if it is in the other portion, $MSB = 1$. So, we can use MSB because we know that the sqrt will be $g^{\frac{p-1}{2}} = -1$ multiplicative factor away from each other (the roots of x are $c + -k$)

So, given MSB, we can find discrete log. Therefore, MSB is just as hard as discrete log, because:

discrete log == MSB algorithm + LSB algorithm (WTF)

3.13.4 One way function / Permutation

$F : \{0,1\}^n \rightarrow \{0,1\}^n$

There exists PPTM such that $P[M(x) == F(x)] = 1 - negligible$.

For all PPTM A, forall x chosen at random from domain(f), $P[A(f(x)) \in f^{-1}(f(x))] = negligible$

3.13.5 Hard-core predicate of a one-way function f

$H : 0, 1^n - > 0, 1$ is a hard core predicate of f if

1. x -¿ h(x) is easy,

$\forall PPTMA, \forall randomxindom(f), P[A(f(x)) = h(x)] = negligible + \frac{1}{2}$

As in, should be negligible from random guess (since range is $0, 1$).

3.13.6 Convert one-way function to PNG

PNG(s) $h(s1)||h(s2)||h(s3)...||h(s_n)$

$|| = concatenation$

$s_i = f(s_{i-1})$ $s_0 = s$

3.13.7 General construction of hard-core predicates

For a one-way function f, the XOR of a random subset of bits will be a hardcore prediate.

Let I be the index set, $I \subset [1 \ldots n]$. $H(x) = \texttt{XOR}x_i, i \in I$ will be a hardcore predicate.

3.13.8 Exercise

if $p = s.2^r, maximumr$, LSB is 0th bit, rth bit is a hardcore predicate.

3.14 Information Theory - Lecture 7: Probabilistic encryption

Determinisim fucks over security. Since now-a-days, servers encrypt pretty much everything you send them, you can try to mount a chosen plaintext attack.

3.15 Pseudorandom Function (PRF)

$F : (k : \{1,0\}^n) \to (r : \{1,0\}^n) \to (x : \{1,0\}^n)$

1st string is key, second string is what to encode, output is encoded.

3.16 Truly random function

Look at all functions from r to x. Pick one such function and use that. Number of such functions:

Number of such functions: $2^{n^{2^n}}$ Number of bits to index this set: $log(2^{n^{2^n}}) = 2^n log(2^n) = n\dot{2}^n$.

If we have key size as $n \cdot 2^n$, then F_k (the kth function in the set of all TRFS from r to x) will be truly random.

3.17 Oracles

A thing that can solve problems. Example: $P^N P$ is a machine in P that has access to an NP oracle. (some NP problem. so it's existential)

3.18 PRF continued

A PRF is a function that uses n bits of key to index the TRF space. So, our of $2^{n^{2^n}}$, we can index only 2^n.

For a PRF:

1. given x, computing $f_k(x)$ is easy.

2. forall PPTM A, $|P[A_k^f = 1] - P[A^{TRF} = 1]|$ is neglibile. That is, A cannot differentiate where a key lies (from PRF or from TRF).

3.19 Cipher Block Chaining

Like the name says, chain blocks for messages. we perform $c_k = F_k(m_k XOR c_{k-1})$. This creates a chain of dependences.

3.20 Output feedback mode

$r_1 = public$ $r_k = f_k(r_{k-1})$ $c_k = m_k XOR r_k$

$G(x) = G_0(x)||G_1(x)$

3.21 Information Theory - Lecture 8: CPA security

Adversary has oracle access to the encryption machine, still can't decrypt it.

However, CPA secure is not really enough.

If we have a key scheme of the form $< r, f_k(r) XOR m >$, perhaps the adversary will XOR m with $f_k(r)$, to make the key scheme $< r, flipMSB(f_k(r) XOR m) >$.

Now, we allow the adversary access to the decryptor as well.

CCA secure := chosen cyphertext attack.

3.22 Data integrity

`Sender(Alice) ---> Receiver(Bob)`

Both have a secret key.

Alice has a message M which she sends to bob.

Bob will receive M' that could be tampered. Bob should be able to tell if $M' =? M$.

This is kind of impossible. If Bob could actually tell the difference, then there is no need to transmit the message.

We want a MAC algorithm (message authentication code)

$< Gen, MAC, Verify >$

Verify(M, tag) returns Valid or Invalid 4 when tag is generated by $MAC_key(m)$, verifying algorithm should generate true.

CBCMAC

- Historical ciphers: (breaking) + ceasar and shift + Monoalphabetic substitution cipher + Vigenere cipher

- 17th century: Kerchoff's Principle: (don't use obscurity) + Shannon's pessimistic theorem + One time pad is perfectly secure + —M— ¡= —K— (limitation of perfect security)

- Two relaxations + PPTM adversary + Negligible p of error + $f(n)$ is negligible iff $\forall p \in R[x], \exists N_0, \forall n \geq N_0, f(n) < 1/p(n)$. ++ examples: $f(n) = \frac{1}{2^n}$. $f(n) = \frac{1}{eps^n}$ where $eps > 0$.

- PRG - Secure encryption

- CPA - CPA secure : Adversary has free access to encryption oracle - So, we need probabilistic encryption to offer CPA security. - PRF - CPA secure encryption

- CBC - IFC - Random counter mode

- Convert Pseudo random function to pseudo random permutation. If both forward and backward are efficient, then it's a block cipher. We did this using a "Feistel structure".

$f' :: ZxZ \to ZxZ$ $f' = (x, y) \to (y, (F_k(y)$ `xor` $x))$

This function is invertible. Each application is a "fiestel round". Apply this as many times as wanted, at least 4 is recommended.

- 3 DES. 2 keys of 56 bits each.

- CCA secure (chosen ciphertext attack) Adversary does not know what the message is. He can actively modify the *ciphertext*.

- Semantic security

- MAC : message authentication code - solves problem of data integrity.

CPA secure + MAC =¿ CCA secure.

c -¿ cpa secure(c) + mac (c)

What is information?

- Shannon, Kolmogrov, Lenin - Randomness, Space, Time.

3.23 Public key crypto

Before Mid-1, we created a CCA-secure scheme. We assumed that the key K is pre-shared between sender, receiver.

Diffie Hellman key exchange.

Can the key be made public, such that converting from public (encryption) key to the private (decryption) key is hard?

So, we can publish an encryption key that is public, thereby allowing everyone to communicate with us.

3.23.1 Diffie Hellman SKE(Secret Key Exchange)

- There is a group G and a generator g of G. Eg: $Z_p^x =< g >$. These are *public*.
- Alice chooses a random element $a \in G$. Alice sends g^a to Bob.
- Eve is eavesdropping, all she can see is g^a.
- Bob chooses b, sends g^b to Alice.
- Eve sees g^b, cannot find b.
- Key is $g^{ab} = g^a \cdot g^b$
- key for Eve is g^{b^a} (g^b came from Bob).
- Key for Bob is g^{a^b} (g^a came from Alice).
- Now, they both have the key, while an eavesdropper cannot find the key.

This is insecure if it is possible to get g^{ab} from g^a, g^b. Even if discrete log is hard, there could be some way to use group structure to do this.

This assumption is called 'CDH assumption': given g^a, g^b, computing $g^a b$ is hard.

3.23.2 Does this satisfy our need? Or, RSA

This solves key exchange, but not the way we wanted to. We wanted to *publish* the encryption key.

RSA

- p and q are two *large* primes of nearly same length. (today, 512 bits). $n = p \cdot q$. $e \in [1..(p-1)]$, $(e, (p-1)(q-1)) = 1$ d such that $ed = 1 mod (p-1)(q-1)$

  Public key: $< N, e >$ Private key: $< p, q, d >$
- $Encryption(m) = m^e (mod N)$
- $Decryption(c) = c^d (mod N)$

Correctness of RSA

$$dec(enc(m)) = \tag{3.1}$$
$$c^d(modN) = \tag{3.2}$$
$$(m^e)^d(modN) = \tag{3.3}$$
$$m^{ed}(modN) = m(since ed = (p-1)(q-1) = \phi(n)) \tag{3.4}$$

$phi(N)$ $phi(N) = pq$(all numbers)$-q$(multiples of p)$-p$(multiples of q)+1(double subtraction of N)

$a^{\phi}(n) = 1(modn)$ consider the set $S' = i_1a, i_2a, i_3a, i_{\phi}(n)a$, $S = i_1, i_2, ..., i_{\phi}(n)$. Show that S and S' are permutations. QED. (TODO: how to box?)

However, here, we don't publish the key.

RSA assumption Given Encrypted message $m^e(modN)$, and the public key $< N, e >$ we cannot get m.

Textbook RSA does not work

- RSA is deterministic. Hence, we do not have CPA security.
- Small key, small N: If Key is small, then, for example, let $m^3 = N$. Now, we can compute cube root $m < \sqrt[3]{(}N)$.
- Small key: $c_1 = m^3 mod(N_1)$. $c_2 = m^3 mod(N_2)$. $c_3 = m^3 mod(N_3)$ We are multi-casting this message to two people, both of whom have chosen 3 as their exponent. Use chinese remainder theorem. Find m^3 in $0 \leq m^3 \leq N_1N_2N_3$.

Chinese Remainder Theorem Given a family of congruence equations $x = a_i(modN_i)$, all $N_i, N_j, i \neq j$ are pairwise coprime, Then we can find $x \in N_1N_2N_3...N_k$.

That is, there is a ring isomorphism:

$Z/N_1ZxZ/N_2Z, Z/N_3Zx...xZ/N_kZ = Z/(N_1N_2N_3 \cdots N_k)Z$

Backwards is obvious, just take modulo $N_1, N_2, N_3, \cdots, N_k$.

Simplest case:

Assume $a_1 = 1$, all other $a_k = 0$. In this case we must set, $x = q.N_2N_3N_4..N_k$. Let $q = (N_2N_3..N_k)^{-1}(modN_1)$. Hence, $a_1 = 1(modN_1)$. (since $q.N_2N_3N_4..N_k = 1$ (mod N_1)). Also, this number x modulo any *other* $N_k, k \neq 1$ will be 0 since x is a multiple of that N_k.

Similarly, the vector $[0, 1, 0, 0, ...0]$ = Let $Sol = Pi_{i=1}^{k} N_i/N_2$. Now, the number we need is $x = Sol * (Sol^{-1})(modN_2)$.

So, we can in generate construct our "basis vectors" $[1, 0, 0, ...], [0, 1, 0, 0, ..], [0, 0, 1, ...]$. So, write any number as: $a_1[1, 0, 0, ...] + a_2[0, 1, 0, ...] + a_3[0, 0, 1, ...]$ (I am somewhat confused, how do we *find* a_i?)

3.23.3 PKCS v1.5 (Public Key standard)

We give a probabilistic version of RSA to give it CPA security. To encrypt a message m:

- $Enc(m) = (0000\ 0000 \mid\mid 00000010 \mid\mid r$ (at least 8 bytes, $r \neq all{-}zeroes) \mid\mid 0000\ 0000 \mid\mid m)^e (mod N)$ (—— = concatenation.)
- We lose *at least* 11 bytes of performance. $(1 + 1 + (\geq 8) + 1)$. If we fix length of r to be 8 (or some other constant), then the scheme is insecure! (Homework assignment).
- for RSA, LSB is the hard core predicate. That is, it is hard to get LSB(x) given $x^e (mod N)$. The first 16 bits are very easy to get (apparently). So, we standardise something in the first 16 bits. (WTF? Read the proof of this). The bits after that are right after (where r) sits is also weaker than LSB. So, we keep the randomness in the weaker bits, and the *actual message* in the stronger bits (remember, LSB is hard!).
-
- Dec(m)

Theoretical version of this is called $RSA - OAEP$. (OAEP = optimal asymmetric encryption padding). This has a proof in the random oracle model that $RSA - OAEP$ is secure. There is no such proof of $PKCS$.

3.23.4 El Gamal scheme

New public key scheme that is based on discrete log, but uses the public key template. Has a proof of CPA-security in the standard model.

Let G be a group. Let the message be an *element* of the group $m \in G$. Let $r \in G$, r random. Let the cipher text $c = m \cdot r$. $c \in G$.

We want r to look like g^{xy}. We know from diffie-hellman that $g^x y$ cannot be found from g^x, g^y.

Group G is published. Generator g is published ($G =< g >$), $|G|$ is public. There is a *secret element x*. We publish g^x.

$PK =< G, g, |G|, h = g^x > SK = x$

$Enc(m) = Choosey \in G. < g^y, h^y * m >$

$Dec(g^y, h^y * m) = (g^x)^y * m = (g^{y^x}) * m\ m = h^y * m/(g^y)^x$.

So, we can get m.

We get CPA security since it is probabilistic (choice of y is probabilistic).

Homomorphic Encryption with El-Gamal

Note that El-Gamal is homomorphic WRT group operation. $< g^y 1, h^y 1 * m1 >, < g^y 2, h^y 2 * m2 >$. Then multiply pointwise. $< g^{y_1+y_2}, h^{y_1+y_2} * m_1 * m_2 >$. Hence, we have encrypted $m_1 * m_2$.

3.24 Hashing - collision resistant hashing schemes

Key superscript: index Key subscript: secret.

Probability that we can find collision for H^s by a PPTM must be negligible.

3.25 Merkle Damgard Transform

Given a collision resistant, hash function of the form $(h : \{0,1\}^{2n} \rightarrow 0,1^n)$. (Fixed length)

Then we can construct a collision resistant hash function of the form $(H : \{0,1\})^* \rightarrow \{0,1\}^n)$.

$m = m1||m2||m3||..m_n$

m_i is n bits.

$z_1 = h(m_1, IV)$ $z_2 = h(m_2, z_1)$ $z_t = h(m_t, z_{t-1})$

Finally, $H(m) = h(z_t, |m|)(?)$

If h is collision resistant, then H is collision resistant

3.26 How to query a DB without revealing data / Oblivious Transfer

1. We shoul not reveal the query 2. Database reveals nothing except for the query answer to the query

3.27 Simplification of the problem

Consider an array of n bits, $Arr = b_0 b_1 \cdots b_n$. The query is an index query i. The task is that the querier B should get to know b_i, should not get to know $b_j, j \neq i$. A should not *know i*!

Intuitively, it seems like "information deadlock" should take place. Since neither party knows what to do, there is some sort of deadlock. Hence, impossibility. (Kannan is happy here, since now we can information theory this, as he puts it).

Supposedly, one-way-functions will work.

$x \to f(x) is easy.$ $f(x) \to x is hard.$ If we know trapdoor information, $f(x) \to x$ is easy.

3.28 Construction of the scheme

$f : [1, n] -> [1, n]$ is a trapdoor one-way *permutation* (unique decryption).

- 1. B chooses n bits at random - $r_1 r_2 \cdots r_n$.
- 2. B applies f only at r_i to obtain $Z = r_1 r_2 \cdots r_{i-1} f(r_i) r_{i+1} \cdots r_n$. f may not be applicable to single bits. In this case, we can use the hadcore predicate and $XOR\ r_i$ with the hardcore predicate.
- 3. B sends Z to A.
- 4. A decrypts (find inverse) of Z and obtains $Y = f^{-1}(r_1), f^{-1}(r_2), \cdots r_i, f^{-1}(r_{i+1}), f^{-1}(r_n)$
- 5. Let array be $Arr = b_0 b_1 \cdots b_n$.
- 6. Perform $ArrXORY = b_0 XOR f^{-1} r_1, \cdots, b_i XOR r_i, \cdots b_n XOR f^{-1}(r_n)$.
- 7. A sends $ArrXORY$ to B.
- 8. B obtain $b_i = (ArrXORY)[i] XOR r_i = (b_i XOR r_i) XOR r_i = b_i$.

Note that B cannot see b_j where $j \neq i$, since in some sense, we have "encoded" the b_j with f^{-1}.

A cannot know which index is the correct index, since there is no "marker" for the correct index.

3.29 Solving the universal problem

A has input X. B has input Y. we wish to compute $Comp(X, Y)$. Either both of them want $Comp(x, y)$, or one of them want $Comp(x, y)$.

A and B are unwilling to reveal their information to each other.

So, how does one solve this? Generalize our specialized construction.

Andrew Yao was awarded the Turing award in 2000 for posing the general problem and solving it.

3.30 Yao's millionaire problem

There are two millionares (Kannan quip: let me make them billionaires, because inflation). They wish to find out who is richer, without revealing their bank balance to each other. Can we solve this?

3.30.1 Weird kannan style generalization

Given two machines A and B, can we construct a virtual machine S, such that A, B do not know what S is computing, but they virtually simulate S?

That is, $Mem(s) = Mem(a)XORMem(b)$

Can a cluster of insecure machines simulate a secure machine? (Neat!)

Sid question: Can we not construct FHE by keeping some data on the client as well? It could be redundant data, but it would still be part of the algorithm? I guess this does not really give you FHE, because the full data is not owned by one party.

Kannan tangent - Teaching ethics instead of teaching crypto It is better to teach ethics and forego the area of crypto, rather than teach crypto and allow people to forget ethics.

I'm not keen on crypto solving dishonesty problems.

- 1. There will be dishonest people in the world
- 2. Software will have bugs for dishonest people to exploit.

Outside of Earth, it has already caught on.

The first major implementation of our solution was performed by satellites (?) (what in the hell, TIL). They don't want to collide in mid-space, but they **do not want to reveal where they are**.

As satellite traffic increased, there was a genuine chance that collision would take place. They ran this protocol between satellites so they can prevent collisions.

Finally, the solution

We wish to perform an instruction $z < -x + y$ on S, where x, y are also stored in S.

Constructing XOR x is stored in S means that x_a is in A, x_b is in B, $x = x_A \texttt{xor} x_b$. (note that in our scheme, x_a and x_b are stored *at the same memory loc at A and B. That is $A_{ram}[i] = x_a, B_{ram}[i] = x_b$.

Party A performs: $z_a = x_a XOR y_a$

Party B performs: $z_b = x_b XOR y_b$

Construct AND We know what the output should look like: $z_a XOR z_b = (x_a XOR x_b)/\ (y_a XOR y_b)$

Code for A: $z_a < -random0, 1$. A creates an array of length 4, which stores values of z_b corresponding to values of x_b, y_b. (Look at $z_A XOR z_b$, and see what the value should look like). So, consider all 4 cases, corresponding to the indeces of A_i.

0. If $x_b = 0$, $y_b = 0$, $z_b = (x_a/\ y_a) XOR z_a. = Arr_0$
1. If $x_b = 0, y_b = 1$, $z_b = (x_a/\ (NOT y_a)) XOR z_a = Arr_1$.
2. If $x_b = 1, y_b = 0$, $z_b = ((NOT x_a)/\ y_a)) XOR z_a = Arr_2$.
3. If $x_b = 1, y_b = 1$, $z_b = ((NOT x_a)/\ (NOT y_a)) XOR z_A = Arr_3$.

Code for B: Run oblivious transfer with $n = 4$. A has a linear database of size 4. B has the index. Hence, B gets z_b.

Note: why pick z_a randomly? z_a acts as one-time-pad in the array table construction. This obscures what B can see about x_a, y_a.

Exploiting multiple machines - Byzantine situations With many machines, we can XOR the data between many machines. This gives us much higher security. However, we lose out on fault-tolerance. People have explored this fully, and we can ask for arbitrary fault tolerance and security, and we can then recieve a protol to be used for that setting.

We can have at most $n/3$ parties that were colluding and disrupting among a cluster of n machines, and still have fault tolerance and security.

3.31 Secure multipart communication

3.31.1 Synchrony

Existence of rounds. Send messages per round. Messages are recieved by all per round.

Or, equivalently, there exists a global clock.

3.31.2 Problem statement

A, B, C have x_a, x_b, x_c inputs.

We wish to compute $f(x_a, x_b, x_c)$ without revealing x_a, x_b, x_c.

For this, we **do not** need trapdoor one-way permutations! Somehow, the existence of 3 people allows us to sidestep the requirement of trapdoor one-way permutations.

We can simulate a trusted virtual server that is not under the control of A, B, C, that can compute $f(x_1, x_2, x_3)$ without revealing.

Adversary can only eavesdrop on **one of three parties** at any given time. We do not know which party adversary is spying. Adversary has access to NP-oracle.

Kannan philosophy

The adversary is *omnipotent* (can solve NP complete problems in P). BUT, he is not *omnipresent* (can only eavesdrop on one of A, B, C at a time).

3.31.3 Machinery: Key management

Can the key be stored in a network of n memory spaces (called n "shares" of the key) such that upto t shares reveals nothing about the secret. all $t+1$ or more shares reveals the secret.

Kannan philosophy: Rate of papers being published?

Supposedly, when he had last seen this area and surveyed it a couple decades back, there were 10,000 papers. However, for some reason, we are unable to actually *see* this in our research life.

That is, per year, the course does not change by so much. why is it that the rate of increase of knowledge is uncorrelated with the rate of papers being published?

He argues that most papers are trash, indirectly. "We have learnt something about the art of generating problems and solutions which is useful for training our mind, but not a contribution to the world."

Maybe by the end of today we can see why that happens? (what? Does the crypto scheme shed some light on this?)

Proof: Shamir's Secret Sharing scheme

Consider FF, finite field, characteristic p. He picks Z/pZ. (Note: I will continue to use FF for finite field).

Pick a polynomial of degree t, with $t+1$ coefficients.

$P(x) = \sum_{i=0}^{t} a_i x^i$. $a_0 = S$. $a_i, \text{where} i > 0 = randomelement from F(Z_p in our case)$.

The secret is $a_0 = S$.