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

Attack is based on frequency. $\{\forall x \in \mathtt{Letter}, \mathtt{freq}(x) = \mathtt{freq}(f(x))\}$. 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.

$\mathtt{crypto} + \mathtt{catcat} = \cdots$

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 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 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, $|K| \geqslant |M|$.

Note on bit sizes

It is possible that $|K| \geqslant |M|$, but $nbits(K) \leqslant nbits(M)$. 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 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.

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

3.6 Looking at the future (next lecture)

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

Chapter 4

Information Theory - Lecture 4

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

4.0.1 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.

4.0.2 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.

$\mathsf{P} \neq \mathsf{NP} \implies$ **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.

4.0.3 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:

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

This is equivalent to saying that $\mu \leqslant 2^k$ 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 $R[X]$?

4.0.4 Formal definition of encryption scheme

An encryption scheme is a 3-tuple $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$, such that:

- $\texttt{Gen} : (n : \mathbb{N}) \rightarrow \text{rand}\{0, 1\}^n$
- $\texttt{Enc}_k : \{0, 1\}^m \rightarrow \text{rand}\{0, 1\}^c$. Note that the encoding can be randomized.
- $\texttt{Dec}_k : \{0, 1\}^c \rightarrow \{0, 1\}^m$. Note that we assume that the decoding is deterministic. (NOTE: this is probably okay since we can pad the cipher with random bits that the decoder can access).
- $\forall m \in \{0, 1\}^m, \texttt{Dec}_k(\texttt{Enc}_k(m)) = m$.

4.0.5 Indistinguishability in the presence of an eavesdropper

- Fix message space with all messages of equal length.
- The adversary chooses two message of his choice, M_0, M_1.

- We geneate a key using $k \leftarrow \texttt{Gen}(n)$. A random bit $b \leftarrow \{0, 1\}$ is chosen. The ciphertext $c \leftarrow m_b$ is computed and is given to A.
- A outputs a bit b', which is A's guess of whether the ciphetext c corresponds to m_0 or m_1.
- The output of the experiment is $(b = b' \ ?\ 1 \ :\ -1)$.

We say that an encryption scheme $\Pi \equiv (\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ has Indistinguishability in the presence of an eavesdropper if:

$$Pr(experiment(A, \Pi, n) = 1] \leqslant \frac{1}{2} + \text{negl}(n)$$

TODO: write down definition 3.10 from textbook about the alternate definition.

Chapter 5

Information Theory, CPA security - Lecture 5

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

5.1 Our first example of circumventing an impossibility

5.2 PRNGs - Pseudo random number generators

This allows us to break $|K| \geqslant |M|$. 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,

$$|P[D(r) = 1] - P[D(G(s)) = 1]| \leqslant \text{negl}(|s|) \quad r \leftarrow \{0,1\}^{l(n)} \quad s \leftarrow \{0,1\}^{n}$$

r, s are chosen uniformly at random. The probability distribution is over the random coins used by D, along with the uniform distributions of r and s.

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

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.

5.2.1 Multi message Indistinguishability experiment

This is defined for an encryption scheme $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$.

- Adversary outputs a pair of vector of messages $(\vec{m}_0, \vec{m}_1)$. Each vector contains the same number of messages, and the ith messages have the same length. That is, $|m_0[i]| = |m_1[i]|$.
- A random key is created: $k \leftarrow \texttt{Gen}$, and a random bit $b \leftarrow \{0, 1\}$ is chosen. For all i, $c[i] \leftarrow \texttt{Enc}_k(m[i])$ is computed. $\vec{c}$ is given to the adversary A.
- The adversary A outputs a bit b'. The output of the experiment is 1 if $b = b'$, and 0 otherwise.

Security definition of the cryptosystem remains unchanged.

Weakness of one time pads under this threat model Note that one time pads will fall to this threat model, since repeatedly ciphering data with a one-time pad will allow us to extract data from the one-time pad. Indeed, any deterministic scheme can be attacked under this threat model. So, we now need probabilistic encryption schemes.

Attacking all deterministic cryptosystems under multi message threat model Let $m_0 \equiv (0^n, 0^n), m_1 \equiv (0^n, 1^n)$. Run this through the experiment. We will be given $c \equiv (c_0, c_1)$. If $c_0 = c_1$, then we know that the message was $(0^n, 0^n) = m_0$, and is m_1 otherwise. We know this since the encryption function is deterministic, and hence $\mathtt{Enc}_k(m_0) = \mathtt{Enc}_k(m_1) \implies m_0 = m_1$.

5.2.2 Secure multiple encryptions using a stream cipher

TODO: add rigorous definition of stream ciphers. Two modes of operation.

Synchronized mode Communicating parties make sure they use different parts of the stream cipher. This way, no one reuses bits from the stream cipher. This can be viewed as one giant plaintext, by concatenating the messages exchanged between the two parties. Security immediately follows.

This is difficult since we need to maintain state between encryptions, to make sure that we do not reuse randomness.

Unsynchronize mode We do not need to maintain state, but we need a more powerful definition of pseudorandom generator. We need a seed s and an initialization vector IV of length n. The requirement we will need is that $G(S, IV = iv)$ is pseudo-random (That is, knowing IV while s is secret keeps the output pseudorandom). Furthermore, for two randomly chosen initialization vectors IV_1, IV_2, $G(s, IV_1), G(s, IV_2)$ is pseudo random when $IV_{\{1,2\}}$ are known, but s is secret. Note that s is the same for both! We can create an encryption scheme as:

$$\mathtt{Enc}_k(m) \equiv (IV, G(k, IV) \oplus m) \qquad \mathtt{Dec}_k((IV, c)) = m \oplus G(k, IV)$$

CPA (Chosen plaintext attack) security: $PrivK^{cpa}_{A,\Pi}(n)$

This is defined for a cryptosystem $\Pi \equiv (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$.

- Random key is generated $k \leftarrow \mathtt{Gen}(n)$.
- The adversary A is given oracle access to $\mathtt{Enc}_k(\cdot)$. The adversary generates two messages m_0, m_1.
- Random bit $b \leftarrow \{0, 1\}$ is chosen. A ciphertext $c \leftarrow \mathtt{Enc}_k(m_b)$ is computed, and is given to A.
- A continues to have oracle access to $\mathtt{Enc}_k(\cdot)$. Adversary outputs a bit b'.
- Output of experiment is 1 if $b = b'$, 0 otherwise.

A private key encryption scheme $\Pi \equiv (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ has Indistinguishable encryptions under CPA security if for all PPTM A, there exists a negligible function negl such that:

$$Pr[PrivK^{cpa}_{A,\Pi}(n) = 1] \leqslant \frac{1}{2} + \text{negl}(n)$$

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 $c \equiv \langle r, m \oplus \mathtt{Enc}_k(r) \rangle$ where $r \leftarrow \{0,1\}$ is a random string. Decryption will never fail. if we know r, we can $\oplus$ to find m. However, the problem is length doubling: For data of length n, we need $|r| = n$.

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 $g^x mod p$, 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. $s_1 s_2 ... s_n$ = stuff from PRNG is distinguishable from $r_1 r_2 r_3 ... r_n$ = Random info.

Construct $s_0 s_1 s_2 .. s_n$, $r_0 s_1 s_2 .. s_n$, $r_0 r_1 s_2 s_3 ... s_n$. $r_0 r_1 r_2 r_3 .. r_n$. 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 $r_i distr_{i+2}$, we need to have either $r_i distr_{i+1}$ 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.

Chapter 6

Information Theory - Lecture 5

6.1 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.

6.2 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?

6.2.1 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 $\mathrm{LSB}(x)$. 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—).

6.2.2 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.

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

$1.ifx == 0mod4- > 1$ $2.ifx == 1mod4- >?$ $3.ifx == 2mod4- >?$ $4.ifx == 3mod4- >?$

6.2.3 Hardness given ability to get MSB

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

We want to find $\mathrm{sqrt}(y)$. 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, $\mathrm{MSB} = 1$. 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)

6.2.4 One way function / Permutation

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

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

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

6.2.5 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 $0, 1$).

6.2.6 Convert one-way function to PNG

PNG(s) $h(s1)||h(s2)||h(s3)...||h(s_n)$
$|| = concatenation$
$s_i = f(s_{i-1})$ $s_0 = s$

6.2.7 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, $I \subset [1 \ldots n]$. $H(x) = \mathtt{XOR} x_i, i \in I$ will be a hardcore predicate.

6.2.8 Exercise

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

Chapter 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.

7.1 Truly random functions

Look at all functions from r to x. Pick one such function and use that. Number of such functions: $2^{n^{2^n}}$

Number of bits to index this set:

$$\log\left(2^{n^{2^n}}\right) = 2^n \log(2^n) = n \cdot 2^n$$

7.2 Pseudorandom Function (PRF)

We need distributions on functions. We define this by using keyed functions.

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

firstst string is key, second string is what to encode, output is encoded. In general, we fix a key, and then consider the function F_k. We assume that F_k is efficent. That is, there is a deterministic polynomial time algorithm that can compute $(F_k(x)\ \forall k, x)$.

Intuitively F is called a pseudorandom function if the function F_k for a randomly chosen k is indistinguishable from a random function chosen from the set of all functions having that domain and range.

Note that the space of all functions is $(2^n)^{2^n}$, while the space of keys is just (2^k).

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.

We formally define them as:

- Efficiency of computation: given x, computing $f_k(x)$ is easy.

- Pseudorandomness: for all PPTM A,

  $$|P[A^{f_k(\cdot)} = 1] - P[A^{f_n(\cdot)} = 1]| \leqslant \text{negl}(n)$$

  where $k \leftarrow \{0,1\}^k$, is a key that is chosen uniformly at random from the key space, and $f_n \leftarrow (\{0,1\}^n \to \{0,1\}^n)$ is chosen uniformly at random from the space of functions.

7.3 CPA security from pseudorandom functions

Let F be a pseudorandom function. Define an encryption scheme as follows:

- $\texttt{Gen} \equiv k \leftarrow \{0,1\}^n$. Choose a key at random
- $\texttt{Enc}(m) \equiv (r, F_k(r) \oplus m)$
- $\texttt{Dec}((r,c)) \equiv f_k(r) \oplus c$.

This as seen before is CPA secure, but is problematically length doubling.

7.4 Pseudorandom permutations

A pseudorandom permutation is much like a pseudorandom function, except it is bijective, and there is a polynomial time algorithm to compute both $F_k(\cdot)$ and $F_k^{-1}(\cdot)$.

Definition 1 *Let* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be an efficent keyed permutation. We call* F *a pseudorandom permutation if for all PPTM* D*, there exists a negligible function* negl *such that:*

$$\left| \Pr\left[D^{F_k(\cdot),F_k^{-1}(\cdot)} = 1\right] - \Pr\left[D^{f_n(\cdot),f_n^{-1}(\cdot)} = 1\right] \right| \leqslant \text{negl}(n)$$

Where $k \leftarrow \{0,1\}^n$ *is chosen uniformly at random, and* f_n *is chosen uniformly at random from the set of all permutations of n bit strings.*

7.5 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.

7.6 Output feedback mode

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

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

7.7 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 $< r, f_k(r) XORm >$, perhaps the adversary will XOR m with $f_k(r)$, to make the key scheme $< r, flipMSB(f_k(r) XORm) >$.

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

CCA secure := chosen cyphertext attack.

7.8 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 $\forall p \in R[x], \exists N_0, \forall n \geqslant N_0, f(n) < 1/p(n)$. ++ 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".

$f' :: ZxZ \to ZxZ$ $f' = (x, y) \to (y, (F_k(y)\ \texttt{xor}\ x))$

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.

7.9 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.

7.9.1 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.

7.9.2 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: $< p, q, d >$
- $Encryption(m) = m^e (modN)$
- $Decryption(c) = c^d (modN)$

Correctness of RSA

$$
\begin{aligned}
dec(enc(m)) &= & (7.1)\\
c^d(modN) &= & (7.2)\\
(m^e)^d(modN) &= & (7.3)\\
m^{ed}(modN) &= m(sinceed = (p-1)(q-1) = \phi(n)) & (7.4)
\end{aligned}
$$

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

$a^\phi(n) = 1(modn)$ consider the set $S' = i_1 a, i_2 a, i_3 a, i_\phi(n) a$, $S = i_1, i_2, ..., i_\phi(n)$. 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 $0 \leqslant m^3 \leqslant N_1 N_2 N_3$.

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 $x \in N_1 N_2 N_3 ... N_k$.

That is, there is a ring isomorphism:

$Z/N_1ZxZ/N_2Z, Z/N_3Zx...xZ/N_kZ = Z/(N_1 N_2 N_3 \cdots N_k)Z$

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, $x = q.N_2 N_3 N_4..N_k$. Let $q = (N_2 N_3..N_k)^{-1}(modN_1)$. Hence, $a_1 = 1(modN_1)$. (since $q.N_2 N_3 N_4..N_k = 1$ (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?)

7.9.3 PKCS v1.5 (Public Key standard)

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

- $Enc(m) = (0000\ 0000 \,||\, 00000010 \,||\, r$ (at least 8 bytes, $r \neq all - zeroes$) $||\ 0000\ 0000 || m)^e (modN)$ (—— = concatenation.)
- We lose *at least* 11 bytes of performance. $(1 + 1 + (\geqslant 8) + 1)$. 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 $x^e(modN)$. 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 $PKCS$.

7.9.4 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.

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

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

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

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 $m_1 * m_2$.

7.10 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.

7.11 Merkle Damgard Transform

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

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

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

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

7.12 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

7.13 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.

7.14 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.

7.15 Solving the universal problem

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

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.

7.16 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?

7.16.1 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, $\mathrm{Mem}(s) = \mathrm{Mem}(a)XOR\mathrm{Mem}(b)$

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, $x = x_A \mathtt{xor} x_b$. (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: $z_a < -\mathtt{random}0, 1$. 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.

7.17 Secure multipart communication

7.17.1 Synchrony

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

Or, equivalently, there exists a global clock.

7.17.2 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).

7.17.3 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.

$P(x) = \sum_{i=0}^{t} a_i x^i$. $a_0 = S$. $a_i, where i > 0 = randomelementfromF(Z_p inourcase)$.

The secret is $a_0 = S$.

The ith share is the polynomial evaluated at i. $Share_i = P(i), i = 1, 2, ...n$

Clearly, $t + 1$ or more shares reveals the secret, since we will have $t + 1$ points of a t degree polynomial.

However, given only t polynomials, we do not know anything about the constant term. We can look at this as a generalization of a one-time-pad for n terms.

Consider an example for t = 2

$$P(0) = S \tag{7.5}$$
$$P(1) = S + a1 + a2 \tag{7.6}$$
$$P(2) = S + 2a1 + 2a2 \tag{7.7}$$

Having $P(1)$, $P(2)$ is not enough to reconstruct $P(0)$, the secret.

Vandermonde matrix This construction can be seen as a vandermondle matrix:

$Share = A \cdot v$, where:

$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2^2 & 2^3 \\ 1 & 3 & 3^2 & 3^3 \\ 1 & 4 & 4^2 & 4^3 \end{bmatrix}$ $v = [S a_1 a_2 \cdots a_t]^T$. $A_{ij} = i^j$. A has full rank, A is invertible.

Hardness We know that $P[S = s] = \frac{1}{|F|}$ (since we pick each S randomly. We use finiteness of F here).

Compute $P(S = s | S_1 = s_1, S_2 = s_2) =? \frac{1}{|FF|}$. If it is equal to $\frac{1}{|FF|}$, then it's a one-time-pad.

We take this on faith, and say that this will work. Lookup proof.

7.18 Use machinery to solve secure multiparty communication

In this case, we use the key management scheme to share our memory data. We split our memory into n shares, and give the n shares to different parties.

We need $|FF| > n$, so that we can have a large field.

Now, we've built *secure memory*. We have yet to show that we can perform operations over this thing.

Suppose $n = 3, t = 1$ (that is, any one should not get the secret, out of 3 machines)

In this case, we have a 1-degree polynomial since $t = 1$. So, $P(x) = rx + s$. (r = random, s = secret).

Hence, $S_a = r + s$, $S_b = 2r + s$, $S_c = 3r + s$.

We will construct an instruction set consisting of add, $multiply$ *over our FF* (kannan will be doing it over Z_p).

We will need $z < -x + y(inFF)$, $z < -x * y(inFF)$.

Constructing +

x is shared according to $P_x(t) = r.t + x$, y is shared according to $P_y(t) = r'.t + y$.

A has x_a, y_a. B has x_b, y_b. C has x_c, y_c. A should have z_a, B has z_b, C has z_c.

A computes: $z_a = x_a + y_a$ B computes: $z_b = x_b + y_b$ C computes: $z_c = x_c + y_c$

Note that:

$$P_x(1) = x_a.$$
$$P_y(1) = y_a.$$

$$P_x(2) = x_b.$$
$$P_y(2) = y_b.$$

$$P_x(3) = x_c.$$
$$P_y(3) = y_c.$$

We denote $P_{x+y} = (definedas) = P_x + P_y$.

$$x_a + y_a = P_x(1) + P_y(1) = (P_x + P_y)(1) = P_{x+y}(1)$$
$$x_b + y_b = P_x(2) + P_y(2) = (P_x + P_y)(2) = P_{x+y}(2)$$
$$x_c + y_c = P_x(3) + P_y(3) = (P_x + P_y)(3) = P_{x+y}(3)$$

Note that we have now computed a new secret sharing polynomial, P_{x+y}. $z = P_x(0) + P_y(0) = x + y$ So, z is encrypted with P_{x+y}.

Constructing $*$ / Rabin Matching

We denote $P_{x*y} = (definedas) = P_x * P_y$. Note that we have now computed a new secret sharing polynomial, P_{x*y}. $z = P_x(0) * P_y(0) = x * y$

```
z <- x * y
z_a <- x_a * y_a.
z_b <- x_b * y_b.
z_c <- x_c * y_c
```

Polynomials are also homomorphic according to $*$ Hence, $z_a < -P_x * P_y = P_{x*y}$

However, this is a 2 degree polynomial! So, we now do not have P_{x*y}.

There is another problem: Shannon secret sharing gives us secrecy if the polynomial is *random*. In this case, the polynomial is not random, since it is reducible ($P_{x*y} = P_x * P_y$). Reducible polynomials are a subset of all polynomials (Sid question: what's the size of the subset?).

Now, can we somehow "linearize" P_{x*y} such that the constant term remains the same, and it is uniformly chosen across polynomials of degree t? (chosen over: $\mathsf{FF}[x]/x^{t+1}$).

NOTE: I do not understand this final part properly!

We construct $z'_a = x_a * y_a$, $z'_b = x_b * y_B$, $z'_c = x_c * y_c$. Then, we *share* z'_a as z'_{aa}, z'_{ab},z'_{ac}. Repeat with z'_b,z'_c.

They lie on a polynomial whose constant term is z.So, $\lambda_a z'_a + \lambda_b z'_b + \lambda_c z'_c = z$.

We know that z'_a is a linear combination of z'_{aa}, z'_{ab}, $z_{ac'}$ so, $z'_a = \lambda'_a z'_{aa} + \lambda'_b z'_{ab} + \lambda'_c z'_{ac}$. Similarly for z'_b, z'_c.

So, z will be a linear combination of all z'_{pq} where $p, q \in a, b, c$

We can write this linear combination as:

$$z = (\lambda_a \lambda'_a z'_{aa} + \lambda_b \lambda'_a z'_{ba} + \lambda_c \lambda'_a z'_{ca}) + (sameforb) + (sameforc)$$

$z = \lambda_a znew_a + \lambda_b znew_b + \lambda_c znew_c$

Note that $znew_a, znew_b, znew_c$ are 1 degree.

```
P_z(t) = r.t + z.

z_a = r + z
z_b = 2r + z
z_c = 3r + z

Find some linear combination of z_a, z_b, z_c such that we get z
z = t_a z_a + t_b z_b + t_c z_c
```

OK, I don't know WTF happened. Read this tonight (21 feb 2018)

References for this: 1. BGW 1988 (in SoTC) 2. GRR 1998 (in ACM PoDC)

Kannan philosphy

Today is the first time we have brought in a different adversary. Usually, we computationally bound the machine. Today, we are bounding the "omnipresence factor" of the machine.

Suppose $n = 3, t = 2$ (that is, any two should not get the secret, out of 3 machines) This is equivalent to 1 out of 2 (what we did last class), by clubbing two machines together. So, for this, we will require the existence of trapdoor one-way functions (since this reduces to 1-of-2).

7.19 Generalized Secret Sharing

(a beginning towards a major unresolved problem/issue in Crypto/Algo)

7.20 Review of last class

- Shamir's secret sharing, and using it for secure multi-party communication.

7.21 Kannan philosophy

We will take generalized secret sharing and generalise it. This will crop up to have ramifications on crypto and algo.

7.22 Generalization

We have a secret S that we split into n shares $S_1 \cdots S_n$.

In Shamir's secret sharing, $\leqslant t$ learns nothing, $\geqslant t+1$ learns full information.

Implicit assumption: Network is homogeneous in trust. Adversary will attack one part of the network just as likely as any other part of the network.

However, there can be situations where we believe that some subset of the network is relatively more trustworthy than other sections of the network.

In this case, we want to get n shares, such that $<= t_1$ in the first $\frac{n}{2}$, $<= t_2$ in the next $\frac{n}{2}$ does not get the secret. Other combinations can learn.

Example: let $n = 4$, $t = 2$, this means all subsets of size $>= 2+1$ can access the secret.

Example': we want to make a statement such as: This basis (aka "access structure") and all supersets of the basis should be allowed to access the secret: Eg, we can give a basis S1, S2, S1, S3, S4, and we want a secret sharing method that will let us share secrets among these sets and their supersets.

Note that this *is a generalization* of the original. A t access threshold is this basis scheme, where the basis is all subsets of size $t+1$.

7.23 Alternate view of access structure as boolean functions

We said that an access structure is *monotone* over over subsets of $1..n$ (MONOTONE: If a subset S is in the access structure, all supersets of S are in the access structure)

We can look at the access structure as $f : 0, 1^n \to 0, 1$. (There is clearly a bijection between subsets and $0, 1^n$, so represent subset as $0, 1^n$. We define $f(\mathtt{bitencode}(\mathtt{subset})) = 1$ if subset is in access structure, 0 otherwise.

Now, we need f to be a monotone boolean function. that is, $f(\mathtt{bitencode}(S)) = 1 \implies \mathtt{forall} S \subset S', f(\mathtt{bitencode}(S')) = 1$ (if S can access the secret, all supersets of S can access the secret).

So now, we have constructed a boolean function to represent our access structure. We will now invoke complexity theory.

7.23.1 Hardness

We can construct an access structure which will be lower bounded in exponential size of the secret?

Number of subsets of powerset of set n is $2^{(}2^{n})$.

The monotonicity does not reduce this by much (Sid: Proof?)

For every access structure, we will have a secret sharing scheme. So, we will have $O(2^{2^{n}})$ secret sharing scheme. So, the number of bits for some secret sharing scheme will be $O(log(2^{2^{n}})) = O(2^{n})$.

So, we will have a secret sharing scheme that is exponential. Note that the length is the number of instructions in the scheme (both the message itself and the instructions for the secret sharing) will be exponential. However, the *share* could be small.

So, we have a computation versus communication trade-off that we need to explore. That is, we can trade-off the sizes of the computation (instructions length) and the size of communication (that is, the share length).

Unresolved Problems (Open problems)

- Does there exist an access structure on n shares such that *every* secret sharing scheme for it has super-polynomial length? (that is, is there an access structure that does not allow for efficient encoding in terms of share length).

- Suppose we convert an access structure to a monotone boolean function. Suppose we only care about those functions that are in P (that is, polynomial circuit depth).

  Do all efficiently computable (in P) access structures have efficient secret sharing schemes?

  Given an access structure that is *efficiently computable* (in P), can we construct a secret sharing scheme that is in P?

  Relationship to algorithms:

  We have a notion of "input size" in algorithms. If we have a distributed algorithm, assume it is parametrised by number of nodes n, and say $<= t$ nodes that are allowed to be faulty.

  Assume we had captured the fault tolerance in terms of our boolean function, $f : 0, 1^{n} - > 0, 1$. When $f(S) = 0$, it is faulty, and failure here need to be tolerated. When $f(S) = 1$, it is not faulty, and failure here need not be tolerate.

  Note that these are equivalent to the old definition. Sets with $f(S) = 1$ are the "critical nodes", which need to be present. We can have $f(S) = 1 forall|S| >= t + 1$, and this gives us back our old definition based on number of nodes that are faulty.

  So, distributed systems can also say, we wish to tolerate some monotone f in general, so we have generalized distributed system tolerance to a general "critical nodes" notion.

  So now, the size of our description of our distributed systems algorithm depends on the **size of f** (Since f is now a parameter to our distriuted system algorithm)

Before, our distributed system algorithm was $\mathtt{dsalgo}(n, t)$, Now it is $\mathtt{dsalgo}'(n, f)$, so we need to give f as a parameter.

Consider the old problem with tolerance described in terms of the number t. Encoded as a function, it is: $f(S) = 1 if |S| > t, 0 when |S| <= t$. If we have to *describe* f, then if we decide to describe it in terms of the basis, we will have $nC(t+1)$ elements in the basis. What used to be a **number**(t) is now a set of size $nC(t+1)$, since we generalized t to f.

If I choose to view fault tolerance as an access structure problem always, then the basis size is in itself $nC(t+1)$. So, my input size is $nC(t+1)$. So, any ridiculous crap of $nC(t+1)$ will be "polynomial".

However, the old encoding will consider a polynomial in $nC(t+1)$ as exponential.

So, our choice of encoding is skewing our notion of what is "small".

So, we need some reasonable way to define "size of access structure", that does not allow blowup like this.

Why can't we argue that the input size is size of the function f? Size of f has two notions: one as the length of the description of f, and the other as the *time taken for f to execute*. That is, I can have a small description that takes exponential time, or I can have an exponential description that takes constant time. (list of tuples verus encoding the smallest program)

This leads us to: what is the size of a program? If two programs have the same length, but one is faster, then we would like to consider the one that is faster (maybe?)

Why should runtime play a role? Kannan argument: of what use is a program that will not give us output for billions of years. This is as good as not giving us a program at all.

So, we would like to optimise on both length and running time.

This fucks us over when the program that we choose is used to describe something about our runtime environment, like the access structure as given above.

This naturally leads us to the question, given that the access strucrure is efficiently computable, can we have an efficient secret sharing scheme for that?

- Assume that it is *impossible* to have efficient secret sharing schemes for efficient access structures, so we wish to crypto it and solve-the-impossibility.

  We do not even know if we can solve this assuming the whole crypto-model. That is, adversary is PPTM, negligible prob. of error, one way functions exist, do all efficient access structures have efficient secret sharing schemes? Unknown.

- Supposedly, similar problems crop in other areas. To quote kannan, "messy waters", "murky", etc.

  Coding theory:

  We have a noisy channel, and we have to model noise. Noise is modelled as: if I send n bits/symbols, $<= t$ symbols are in error. Can we give an error correction code that can tolerate such noise?

Again, do the same thing, replace $<= t$ with an access structure, as we did in distributed systems. We land in the same problem of describing input size.

Why would we need access structures for ECC ever? For example, consider two channels between $S --> R$. One channel has $1/5$ chance of error, other channel has $1/4$ chance of error. Say I send the first $n/2$ bits over the first channel and the next $n/2$ bits over the next channel.

This can be modeled using an access structure over the full n bit string. So, knowing details about where this "toggling" will happen should let us design better ECC schemes.

For example, if our channel sent first $n/2$ bits correctly and next $n/2$ bits fully wrong, we have an ECC: only send data in the first $n/2$ bits. This is *not* the same as stating that "50% of data is corrupted" In this case, there is no ECC that can solve this.

So, having data regarding where the toggles happen is *useful information* to have. So, the access function is a *useful abstraction* in ECC, it is not frivolous.

Thus, we are not allowed to say "access structures are useless in ECC". So now, our noise is a program / monotone boolean function. This now leads us to the thorny problem of defining size of monotone boolean function.

For the past half-century, ECC has only worked on the threshold kind of f. We have not worked on ECC of the general access function kind.

Rounding back, we need to define input size of f, which is the noise in the channel.

If f is very short in length but it takes exponential time to run, then we cannot "observe" the error since it takes exp time to run.

Suppose we get a noisy channel in real life whose f is not in P. Then the real world Channel is computing f which is in NP.

So, we can sample the channel (which is computing f) to solve NP problems. But such channels should not exist, since nature cannot solve NP problems in P time. So, we need f which is in P.

Q) does there exist efficient ECC for all realistic channel $f \in P$?

Cute Scott Aaronson quote: "If P != NP were a question in physics, then it would have been a law by now (since we have *never* observed nature solve NP problems in P time".

Physics is ridiculous, since we expect reality == math prediction.

We can weaken it, by saying people should be able to distinguish reality and math prediction in polynomial time!

7.24 Agreement in a distributed system

7.25 Introduction: Philosophy of today's class

This is the beginning of "clash of philosophies". Agreement in a distributed system can be impossible! (which is why it is part of this course). If one-way-functions exist, then much more is possible.

Different adversaries:

- Computational Adversary - All modern crypto
- Practical Adversary - Noise based crypto (last class)
- Natura Adversary - Quantum crypto (after mid-2)
- Philosophical Adversary - ??? (So kannan, man)

  Our problem is a fundamental problem in distributed systems. Problems of agreement abound. There are instances where distrubuted systems textbooks give up, and give proofs of no-solutions. However, this is POIS class, so impossibility proofs are where we start our trade.

  Djikstra's paper in 1980 where consensus is not possible: three nodes connected in a synchronous network. We want to simulate a broadcast channel over P2P. That is, we wish to broadcast information over P2P.

  The problem is called "byzantine agreement problem".

  Each party has a single bit of input x_i. Each will give a single bit of output b_i ($i \in \{1, 2, 3\}$). Agreement requirement - All non-faulty nodes have the same output (all b_i are equal).

  If this was the *only goal*, then this is trivial. We just have all of them output a constant. Validity requirement - The output of all non-faulty nodes is equal to the input of *some* non-faulty node.

  We can simulate broadcast. Run the protocol that has agreement and valitiy. We want to simulate broadcast. We send 0 to everybody, and then simualate the protocol that we have crafted. Since everyone has started with the same input, everyone will end with the same output.

  Is this actually such a tough problem? Why can't we just say "send your input to everybody"? Why is this not equivalent to simulating a broadcast channel?

  Let's see what happens. Let A be a source. Say A sends message m to B and C. this looks like A has broadcasted message m. If this were happening over a *broadcast channel*, then m would have been received by B and C.

  However, on a unicast channel, it is possible that A fails between B receiving the message and C receiving the message. Now, this is *not broadcast*.

  We need broadcast to be atomic - either everyone gets the message, or no one gets the message!

assume A is adverserial: that is, A is able to send m_1 to B, m_2 to C, $m_1 \neq m_2$. So, we need to make sure that people can't fuck up broadcast over unicast. However, if we had a physical broadcat channel, this fuck up can't happen.

So, the problem *is complicated*. We wish to simulate an atomic broadcast channel over a unicast channel, given adverserial nodes.

7.26 Three nodes, one of them is byzantine faulty, no way to have both agreement and validity

7.26.1 Intutition

Say A, which is byzantine faulty sends 0 to B and 1 to C. B and C now exchange values, they realise agreement has not actually happened.

If they *know* that A was faulty, then they could agree that A was faulty.

However, B does not know whether C is faulty or not. It could be that A had sent 0 to C, but C was faulty and thus chose to broadcast 1 to B.

So, B only knows that *one of* A and C is faulty. Similarly, C only knows that *one of* A and B is faulty.

7.26.2 Proof

Let Π be a byzantine agreement protocol for this case.

Running with inputs x_1, x_2, x_3, we get outputs b_1, b_2, b_3, we have the guarantee that 1. all b_i are equal 2. b_i is equal to some x_i

We now show that such a Π cannot exist.

The original proof is difficult. Then, at MIT, a new proof technique came out in distributed systems which proved this. (Proof is called "hexagon proof").

Proof strategy:

- Given that Π is a byzantine agreement protocol.
- We create some other problem with some other network, and show that if Π is a byzantine agreement protocol in this network, then that protocol should do *something* in the other network.
- We show that the protocol does not do anything (is not consistent)

Consider a network of six nodes in a ring topology (hexagon) (Call this network Hex) Call the original network Tri.

The code delegated to three parties in Tri is $\Pi =< \Pi_1, \Pi_2, \Pi_3 >$.

Π' is the program for Hex. $\Pi' =< \Pi_1, \Pi_2, \Pi_3, \Pi_1, \Pi_2, \Pi_3 >$ (That is, node i gets program Π'_i).

In Tri, Π_1 got messages from Π_2, Π_3. This is the same structure in Hex. So, structurally, the networks are the same. Syntactically, the code will work in Hex - because the "local" network topology is the same. So, $\Pi_{whatisthis?}$ running at node 1 cannot distinguish if it is at Tri or Hex.

Proof by induction on rounds. At round 0, we cannot distinguish since structurally, the locales are the same. Then, induction on rounds.

We now show that Hex is an inconsistent protocol.

Input 0 for nodes $1, 2, 3$, 1 for nodes $4, 5, 6$.

Consider $(2, 3)$ in $\mathsf{Tri}, \mathsf{Hex}$. $(2, 3)$ in Tri will be talking to 1.

Say 1 is faulty in Tri (we can do this since Pi is a BA protocol). 1_tri will send what goes from 4_hex to 3_hex to 3_tri. 1_tri will send what goes from $1hex$ to 2_hex to 2_tri.

So, $2, 3$ do not know if there are working in Tri or in Hex, because the messages are the same!

We know that in Tri, there is a BA protocol, so it should terminate. Moreover, it should terminate with 0 since we assumed that the protocol is a BA protocol.

Also, this means that output will be 0 in Hex for $2, 3$ as well, since they are structurally isomorphic.

Now, let us look at $4, 5$ in Hex (which has protocols Π_1, Π_2). Let us say that $3'$sends α to $2'$ and β to $1'$.

This is equivalent to a faulty 3 in Tri.

We know that the output must be 1 for 1′, 2′ from our starting conditions.

Now, looking at $(1, 3)$, once again, if 2 is faulty, make outputs the same way such that they can't distinguish if they are running in Tri or in Hex.

In this case, 1′ will have to output 0, since 3 outputs 0. However, in the last case, we had agreed that 1′ would have to output 1.

Conditions: - 3 outputs 0 - 1′ outputs 1 - output(3) = output(1′).

Hence, such a Π' cannot exist. Hence, Π cannot exist.

7.27 Clash of philosophies

7.27.1 Distributed systems spirit

We use the term "non faulty" when defining Agreement, Validity. Originally, the definition was, "if I gave a node Π, then it should execute Π, not something else (Π')".

Kannan - anyone who contributes to the execution deserves the output

7.27.2 Clash

Assume there is a player who is participating in two programs, and is non-faulty.

Suppose it is a byzantine agreement program (1 out of 3), which is impossible.

We force everyone to digitally sign their messages. Now, impossibility becomes possible.

Now, if I was supposed to run Pi (which says sign your messages), so I run it.

In the background, my friend asks me to share my secret key with him, which I do. This person is also part of the BA network, which causes my key to be revealed, and thus the adversary can forge my signature, thus the protocol breaks.

Should the node in question be called faulty or non-faulty?

Assume we call this node faulty. This makes sense, because he is breaking our crypto assumptions of key-privateness.

However, distributed systems is of the opinion that anyone who shares their resources must reap the rewards. So, from a distributed systems viewpoint, the node is non-faulty.

Now, the notion of faulty and non faulty depends on the implementation detail!

7.27.3 Non modular attacks

Assume a byzantine agreement protocol between 3 people. It solves the problem using digital sinatures. Call this protocol Π.

They're running two byzantine agreement protocols in parallel.

Call the parties "1 = 1′, 2 = 2′, 3 = 3′", so we have Π running between $1, 2, 3$, and another instance of Π running between $1', 2', 3'$. (makes sense, two agreement protocols are running in parallel).

Let the input be 0 for (1, 2), and 3 is corrupt in Π.

Let the input be 1 for (1′, 3′) and $2'$ is corrupt in Π'.

Adversary can fail *both* protocols (Why?)

The adversary has access to the private key of 2′ (which is the private key of 2 as well). Since all processes receive the same private key, the adversary in 2′ has access to the private key of 2.

For example, 2′ creates (1, signed 1). The adversary in 2′ will drop the packet that 2 tried to send to 1 (which was supposed to be (0, signed 0)), and sends the (1, signed 1).

7.27.4 Processes

We assume that 2 processes do not have the same PID.

Consider three distributed processes $1, 2, 3$. We have no global PID assuming one of these are faulty, since we showed that it is impossible to agree on even a single bit. So now, we have three PIDs, PID_1, PID_2, PID_3.

Let us say server, client are connected across two ports P, P′. In theory, an adversary can send messages from P to P′, and P′ to P (flip P and P′).

7.27.5 Correctness versus Security

Correct HW on top of a correct OS on top of a correct compiler on top of a correct program will give us a correct program.

In seurity, Secure HW on top of a secure OS on top of a secure compiler on top a secure program will give us a secure program (?)

This is indeed untrue. Consider the distributed process with same PIDs case. the attack is neither fully OS based nor network based. the router can arbitrarily swap PIDs acting as an adversary. We cannot construct consensus due to the byzantine agreement theorem we have.

We can have cross-model attacks.

7.28 Crypto, enter the stage

The idea is that the adversary needs to simulate the hexagon protocol to construct a contradiction. However, if we can ensure that this process is computationally intensive, then life is chill, because we cannot have such an adversary.

So, we force people to sign their messages, so that the adversary can't simulate.

Chapter 8

Quantum Secret Key Establishment

8.1 Three Polarizers Experiment

Photon is a single Qubit system: Vector in $\mathbb{C}^2$

$|\psi >= a|0 > +b|1 >, a, b \in C$

Normalized, such that $|a|^2 + |b|^2 = 1$.

Postulates of QM:

- For any system, there is a state vector in Hilbert space.
- Measurement postulate: If we measure a qubit, then with probablibity of $|a|^2$, we get $|0 >$, with probablibity of $|b|^2$, we get $|1 >$. After measurement, things collapse to one of the measured values
- Evolution postulate: Will evolve according to schrodinger / unitary transformation

8.2 Q

uantum Key exchange

A, B. Have two channels, classical and quantum. $E_{(ve)}$ is eavesdropping on both channels.

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt(2)} \, |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt(2)}$$

Step 1: A chooses to encode 0 as either $|0\rangle$ or as $|+\rangle$ A chooses to encode 1 as either $|0\rangle$ or as $|-\rangle$

Consider a stream of random bits: $r_0, r_1, \ldots, r_n, s_0, s_1, \ldots s_n$

r_i is the value we encode. s_i dictates how we encode r_i.

$r_i = 0, s_i = 0$, then we sent $|0\rangle$. $r_i = 0, s_i = 1$, then we sent $|+\rangle$. Et cetra.

A encodes r_i with respect to s_i and sends the qubits to bob. So, n qubits have been sent. All the qubits can be eavesdropped by E.

Step 2: B receives these qubits and measures them in one of the two bases *at random*. (either 01 or plus-minus). B records the answers.

B's choice of basis will be governed by random bits $s'_1, s'_2, \dots s'_n$ Let the answers on measuring in the s_i basis be $r'_1, r'_2, \dots r'_n$.

Note that if $s_i = s'_i$, then $r_i = r'_i$. If $s_i \neq s'_i$, then $r_i = r'_i$ with 0.5 probability.

Step 3: A and B publish the s_i, s'_i. (publish measurement basis info) Ignore all indeces where $s_i \neq s'_i$. So, from now on, we will consider the index set $I_{eq} = i \in [1 \dots n] | s_i = s'_i$

Analyzing Eve: Eve has a s''_i that is used to measure the channel. When $s_i = s'_i$, but $s''_i \neq s_i$ (that is, Eve is measuring using a different basis from A and B).

Assume WLOG that A and B are using the 01 basis. Now, if Eve measures using +- basis, then the original $|0\rangle$ or $|1\rangle$ will now become $|+\rangle$ or $|-\rangle$.

So now, when Bob measures, he will only *get the original with 0.5 probability!*

We can detect the presence of someone who is intercepting the channel with this principle.

So, there will be 25 percent chance that $s_i \neq s'_i$ when $r_i = r'_i$, if Eve is eavesdropping. If not, then $s_i \neq s'_i \implies r_i \neq r'_i$.

Step 4: Randomly sample some subset of the r_j for those indeces with $s_j = s'_j$. If eavesdropper did not exist, then all the r_j will match. If eavesdropper exists, then some values will be mismatched.

This will let us *detect* the presence of an eavesdropper.

So now, if the eavesdropper is not there, the rest of the *unpublished* r_k*'s* where $s_k = s'_k$ become our secret key. This is shared, since we know that this will not be corrupted, due to the lack of an eavesdropped. We also know that the values will be equal since we use the measurement basis ($s_k = s'_k$)

8.2.1 Why can't eve keep a copy of the qubits?

In theory, Eve could have tried to keep a copy of the qubits, and then measured once the s_i have been published. However, no-cloning prevents this from happening.

8.3 Dealing with noise

On a noisy quantum channel, we will need to deal with the case that possibly $s_i = s'_i$, but $r_i \neq r'_i$.

8.4 Dealing with noise

On a noisy quantum channel, we will need to deal with the case that possibly $s_i = s'_i$, but $r_i \neq r'_i$. We can use classical error correction on the r_i's to deal with this stuff. We do not go this analysis in the lecture.

8.5 Two qubit systems

four classical possible outcomes: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$. Superposition of all four is allowed.

$|\psi\rangle = a_{00} |00\rangle + a_{01} |01\rangle + a_{10} |10\rangle + a_{11} |11\rangle$.

8.6 n qubit system

n qubit system will have 2^n classical outcomes - we will need 2^n complex amplitudes. On the other hand, for am n bit system, we will need, well, n bits.

8.7 Why quantum computer?

Note that a quantum computer runs 2^n computations in parallel, samples one of them (by nature's choice), and deletes all $2^n - 1$ values.

the argument is that, for a classical computer, even *deleting* $2^n - 1$ values will take exponential time. However, we can at least exploit this "deleting" property.