Study notes

Joshua Hwang (44302650)

June 11, 2019

Chapter 1

Preamble

Terminology

Avalanche effect is much like the butterfly effect. A small change in initial input causes radically different output. Without it a Cipher text only attack (see below) can be executed.

Completeness effect means that a small part of the output depends on a large amount of the input. This is good so attackers cannot find ways to simplify the problem and determine things like a quarter of the key (e.g. 2^{10}) they must solve for the whole key (e.g. 2^{40}).

Provable security proves that breaking the cipher is equivalent ot solving some difficult computation problem (barring P=NP).

Computational security focusses on how long it would take to break the cipher with a specific attack. This is not very useful since there may be alternative ways to crack the cipher that can reduce the required computation.

Unconditional security is unbreakable even with infinite computing power (read the fineprint at the end of notes).

Perfect secrecy states that the ciphertext gives no extra knowledge about the

plaintext. $P(x|y) = P(x)$ where x is plaintext and y is ciphertext.

Kerchoff's principle states: "A cryptosystem should be secure even if everything about the system, except **the key**, is public knowledge."

Private key (secret-key or symmetric-key) both parties share a hidden key. This is more efficient but we must find a way to share the key.

Public key is obvious from the previous definition. This form is less efficient. There exists a public key that is used to encrypt and verify (more on this later) and private key only known to the user and create signatures. A signature works by encrypting with the private key, then the public key can be used to decrypt. This works because only the person in possession of the private key could've constructed such a message, anyone can decrypt it though.

These cursive letters have important meanings.

$\mathscr{P}$ is the plaintext space

$\mathscr{C}$ is the cipher space

$\mathscr{K}$ is the key space

$\mathscr{L}$ is the keystream alphabet

$\mathscr{E}$ is the encryption function space

$\mathscr{D}$ is the decyrption function space

There are 4 types of attacks we will be looking at:

Cipher text only attack the opponent has a string of cipher text

Known plaintext attack the opponent has a string of plain text and its corresponding cipher text

Chosen plaintext attack the opponent has obtained temporary access to the **encryption** machinery

Chosen ciphertext attack the opponent has obatined temporary access to the **decryption** machinery

Algorithms

Important algorithms for the exam.

Chinese Remainder Theorem

Given a system of modulo equations our aim is to construct a solution of additions. Each term of our solution will be divisible by all other modulos (meaning they will disapear) except for a single modulo. In this way our problem is simplified in a way. An example from an assignment is provided below.

We first let $s = x^3$ thus our equations can become,

$$s = 42710 \bmod 43807$$
$$s = 15083 \bmod 48721$$
$$s = 40156 \bmod 44767$$

From here we can perform the Chinese Remainder Theorem.

$$\begin{aligned} s &= a_1(m_2 \times m_3)y_1 \\ &\quad + a_2(m_1 \times m_3)y_2 \\ &\quad + a_3(m_1 \times m_2)y_3 \bmod (m_1 \times m_2 \times m_3) \\ s &= 42710(48721 \times 44767)y_1 \\ &\quad + 15083(43807 \times 44767)y_2 \\ &\quad + 40156(43807 \times 48721)y_3 \bmod (43807 \times 48721 \times 44767) \end{aligned}$$

Where y_i is $(\prod_{n \neq i} m_n)^{-1} \bmod m_i$.

Extended Euler Algorithm

Several examples are provided from a previous assignment. It's important to note this is only the last half of the algorithm. The first part involves dividing

then taking the remainder and dividing the divisor but that (confusing I know sorry).

Finding inverse of 2181093007 mod 43807

$$1 = 1 \times 1 + 22 \times 0$$
$$1 = 22 \times -6 + 133 \times 1$$
$$1 = 133 \times 19 + 421 \times -6$$
$$1 = 421 \times -120 + 2659 \times 19$$
$$1 = 2659 \times 619 + 13716 \times -120$$
$$1 = 13716 \times -1358 + 30091 \times 619$$
$$1 = 30091 \times 1977 + 43807 \times -1358$$
$$1 = 43807 \times -98432234 + 2181093007 \times 1977$$
$$1 = 2181093007 \times 1977 + 43807 \times -98432234$$

Found inverse 1977

$$y_1 = 1977$$

Dividing polynomials

Hopefully you know what to do here because I don't want to write up an example.

Chapter 2

Public key ciphers

Diffie-Hellman key exchange protocol

One of the topics on public key ciphers, before RSA. The protocol is rather simple.

Both parties publicly agree on a prime p and $\alpha \in \mathbb{Z}_p^\times$. Each person secretly picks an exponent a and b respectively. Each person sends α^a or α^b. The final key becomes α^{ab} which both parties can construct on their own. Please realise that an eavesdropper could not figure out what the final key is.

RSA

First we choose two distinct large primes p and q. We choose $n = pq$ also note $\phi(n) = (p-1)(q-1)$.

We then choose a random $b \in \mathbb{Z}_{\phi(n)}^\times$ and compute its inverse $a = b^{-1} \mod \phi(n)$.

The public key is now (n, b) and the private key is (a, p, q).

The plaintext and ciphertext space are both $\mathbb{Z}_n$. The encryption and decryption

rules are $e_K(x) = x^b \mod n$ and $d_k(y) = y^a \mod n$.

Some may note that exponentiation is not very efficient. But in context of modulo arithmetic we may simplify at each step allowing for $O(\log n)$ time.

The "toughness" of RSA comes from the difficulty to obtain prime factorisation.

Signatures with RSA are done by sending both a message and it's "decryption" using the private key. It is verified by encrypting the "decrypted" message and revealing the original message (Note how decryption happens **before** encryption we will see in the Cryptanalysis section why we can do this).

Cryptanalysis

RSA becomes weaker in the following scenarios.

If $p \ll q$ then the weakest link in the chain will be p which would then be used to find q and ruin everything.

If $p \simeq q$ then it is somehow easy to factorise n (personally have no idea how though). Try $n = x^2 - y^2$.

If $x \in \mathscr{P}$ is too small then it's possible that $x^b < n$ which means no fancy modulo arithmetic occurs. So we can just take b^{th} root in the reals.

If x is predictable then we can encrypt whatever we want (we have the encryption machinery).

If the bs are too small and the same $x \in \mathscr{P}$ is encrypted with different bs then we can use the Chinese Remainder thereom. (You do remember how to do that right?)

Note: that since we're doing exponentiation we actually have a commutative operation. Decrypting **before** encrypting works fine so our method of signature verification is all good.

ElGamal

First we choose a large prime p. Choose a generator α of $\mathbb{Z}_p^\times$ (this is not trivial). Choose a random $a \in \mathbb{Z}_p^\times$ ensuring it isn't $a \neq 1$ and $a \neq p-1$.

Compute $\beta = \alpha^a \mod p$.

Our public key is now (p, α, β) and our private key is a.

The plaintext and ciphertext are both $\mathbb{Z}_p$. To encrypt we must choose a new random $d \in \mathbb{Z}_p^\times$ for each plaintext message. Encryption is done by $e_K(x) = (\alpha^d, \beta^d x) \mod p$. Decryption is done with $d_K(c_1, c_2) = (c_1^a)^{-1} c_2 \mod p$.

The "toughness" of ElGamal is due to the difficulty of discrete logarithms (logarithms under modulo). If you're confused on how it works it's due to $\beta = \alpha^a \mod p$. Once you replace β with this it should become clear.

Signatures with ElGamal are a little different since commutativity is not available here. It won't be in the exam but for more information look up DSA (the Digital Signature Algorithm).

Cryptanalysis

I don't think we covered this stuff.

PKI (Public Key Infrastructure)

Infrastructure and protocols for ensuring the validity of public keys. For example, what if you wanted to communicate with someone that your computer doesn't have information on. It's not like your computer can just hold all 7 billion peoples public keys.

There are two popular methods: Certificate authorities (used by web browsers) and Web of trust (used by PGP).

Keysigning

The idea behind signing keys is to show that a third party can verify the key's validity.

Suppose Bernard knows Margot's public key is genuine (they met face-to-face). Bernard creates a digital signature for Margot's public key, 'I Bernard am able to verify that this signature does in fact belong to Margot'.

Margot publishes this digital signature alongside her public key.

In this way you only need to trust Bernard (or trust someone that trusts Bernard) to trust Margot. This is called a chain of trust.

The section below will explain why this isn't a simple lookup database server.

Man in the middle attacks

If an enemy, Eve, is in control of the network which Alice and Bob intend to use (at the very least control over one of the players messages). Then the following attack may occur.

Alice requests for Bob's encryption method over the channel. In the world without Eve, Alice receives Bob's encryption and sends Bob a message that is uncrackable (well... hidden sufficiently). From here on the communication is secure.

Now consider when Eve is present. Eve intercepts the request for Bob's encryption algorithm, e_B. Instead of passing e_B to Alice, Eve sends her own encryption method which she has the decryption for e_E. Alice believes this to be Bob's encryption method so encrypts her message with it and sends it. Eve then intercepts Alice's message and decrypts it. Eve is now in possession of the plaintext. Eve herself then re-encrypts the plaintext using Bob's real encryption method and forwards it to Bob all without either party knowing that their messages are being intercepted.

The only way Alice and Bob could know something was amiss is to find a way

to compare their encryption algorithms in person (or safe channel). Alice and Bob will realise that their algorithms don't match up, there is no way they could have been sending each other messages unless someone in the middle was translating the messages... **Eve**!

Certificate authorities (CA)

This is a centralised way of trust. Your web browser has a few root CAs that it trusts (e.g. Symantec, DigiCert). These root CAs trust intermediate CAs who might then go on to trust lower level intermediate CAs. At the bottom are the end users who need certificates from intermediate CAs.

The main issue with this is the main issue with all centralised methods. A single point of failure at root CAs.

Web of trust

All nodes in the chain are end users and the system is decentralised.

The issue with this system is that everyone needs to be proactive in their security. What if your encryption key gets leaked? You can no longer be trusted **yet** you are part of the web of trust. Not only but you need to engage in the web by signing other people's keys.

Chapter 3

Stream ciphers

Stream ciphers encrypt each letter/bit with a different encryption method. This stream of keys may only depend on the previous stream key and initial key (seed key) this is called **synchronous** or may also depend on the previous plaintext called **asynchronous**. Note: The seed key may come from a different space from the stream keys.

A stream cipher is considered **periodic** if the stream keys repeat.

Vigenère cipher

Caesar cipher but each letter uses a different key for a shift. You should know how this cipher works since there was an assignment on it.

This is a periodic synchronous stream cipher.

Cryptanalysis

There are three tests for cracking the Vigenère cipher; Kasiski's method and Friedman's methods 1 & 2. Each of these methods simply attempt to find the

length of the key stream. From there it's a simple brute force job.

Kasiski's test

This is what you used in the assignment. Find triples and their distances from each other.

The greatest common factor between the distances is the best guess for the length of the key.

Friedman's method 1

First guess the keylength m. This will give us the key $k_1 k_2 \cdots k_m$. Then group the cipher text into m buckets for each k_i that letter was encoded with. From here we calculate our ϕ.

$$\phi = \sum \frac{n_i(n_i - 1)}{n(n-1)}$$

Where n_i is the number of times the i^{th} letter comes up. A large ϕ indicates a less uniform distribution.

This ϕ is constant for every language thus we'd want a ϕ that is close to English's. We try different ms until we get close to our desired ϕ.

Friedman's method 2

We automate the above process with the following formula.

$$m \simeq \frac{n(\phi_L - \phi_0)}{(n-1)\phi(T) - n\phi_0 + \phi_L}$$

Where ϕ_l is the ϕ for the language. ϕ_0 is for random text ($^1/_{26}$). $\phi(T)$ is the ϕ for our entire ciphertext.

For very large n we get,

$$m \simeq \frac{\phi_L - \phi_0}{\phi(T) - \phi_0}$$

Autokey cipher

This is an asynchronous cipher. The seed key is an initial letter which encrypts the first letter. The first letter is then used as the stream key for the second letter and so on. In this way decrypting the first letter in the message gives you the key for the second letter.

In more mathematical terms, the plain space, cipher space, keyspace and keystream space are all $\mathbb{Z}_{26}$. It is non-periodic and asynchronous.

Cryptanalysis

There's only 26 possible seed keys so you may try all 26 and crack the cipher.

Linear feedback shift register (LFSR)

The plaintext space, ciphertext space and keystream space are all binary $\{0, 1\}$. Encryption is done by $e_l(x) = x + l$ where l is the stream key. Note: since it's binary decryption works the exact same way.

Each stream key, l, is calculated from a sum of the m previous bits,

$$l_{i+m} = c_0 l_i + c_1 l_{i+1} + c_2 l_{i+2} + c_3 l_{i+3} + \cdots + c_{m-1} l_{i+m-1}$$

With good choices of the c_is we can get a maximum period of $2^m - 1$.

Cryptanalysis

Using a **known plaintext attack** and having $2m$ consecutive plaintext-ciphertext pairs. We can calculate the keystream bits by XORing the plaintext and ciphertext.

We need $2m$ bits because we attempt to deduce the c_is in our equation with m linear equations.

ENIGMA

Enigma uses three permutation operations; the plugboard (S), the rotors (N, M, L) and the reversing drum R.

The plugboard allows for up to 6 pairs of letters to be swapped with the use of 6 wires. Note: if you applied the plugboard to itself twice it would result back to the plaintext, $S^2 = 1$.

The reversing drum created 13 pairs (no loops or cycles greater than 2). Just like the plugboard $R^2 = 1$. Since 13 pairs are created, $R(x) \neq x$. The reversing drum cannot be changed and is not part of they key.

The rotors are arbitrary permutations. Though the choice of rotors is available the internals of each rotor is fixed. The initial rotations of the machine are part of the initial key.

Each keypress the rotors N, M and L will rotate. N rotates every keypress. M rotates once when N lands on position 0. L rotates once when M lands on position 0. Thus the period is $26^3 = 17576$.

A neat thing about ENIGMA is encryption is decryption due to both $R^2 = S^2 = 1$.

ENIGMA is a synchronous periodic stream cipher. This is because the rotor rotations are not governed by which key was pressed (just that **a** key was pressed).

Cryptanalysis

We first define the rotation permutation as $\phi = (a\ b\ c\ d \ldots y\ z)$ where $a \to b$, $b \to c$ etc.

Note that the permutation of a rotor, L, after a single rotation becomes $\phi L \phi^{-1}$. This is better explained with the image on page 13 of the 30 April slides but I will try to explain here. Suppose L has a permutation from $d \to f$. That pair will be rotated to $c \to e$. To achieve this effect enter c into ϕ to get d then pass it through L to get f then through ϕ^{-1} to get to e.

If we are not in possession of the rotors we can deduce them from a chosen plaintext attack by sending 3 letters twice.

Let A, B, C, D, E, F be the first 6 states of the ENIGMA machine for that day.

We know AD turns the first ciphertext into the fourth. This is because the ciphertext will be decrypted by A then re-encrypted by D to give the fourth cipher text.

From here we do what we did in the assignment. Get enough pairs to produce a permutation diagram for AD, BE and CF. Find groups in AD for example that have the same period. We attempt to "line up" these two groups such that it satisfies some ciphertext. Don't forget that when the solution for A is found, the remaining part should be reversed before calling it B. (See 2 May slides)

Chapter 4

Random numbers

There's the difference between pseudo-random and random number generation. Most of the details are present from page 15 on the 7 May slides. I will assume the reader is comfortable with the concepts.

From here on we will be talking about a method to generate pseudo random numbers that cannot be predicted in polynomial time.

Blum-Blum-Shub

First choose large primes p and q where $p, q \equiv 3 \mod 4$.

Compute $n = pq$ and choose an $0 < s < n$ such that $\gcd(s, n) = 1$.

The first random bit is generated by $x_0 = s^2 \mod n$ and whether x_0 is even (0) or odd (1). Afterwards, each new random bit is generated by $x_i = x_{i-1}^2 \mod n$ and whether x_i is even (0) or odd (1)

This method is **provably unpredictable** assuming integer factorisation is hard.

Chapter 5

Block ciphers

These ciphers work on **blocks** of n bits.

Hill cipher

This cipher works over the alphabet unlike the ones we've been seeing so far which work on binary.

The key is an $m \times m$ matrix which is used in the following way.

$$(y_1, y_2) = (x_1, x_2) \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Where x_i are the plaintext which get converted into y_i respectively.

Decoding should be obvious.

Substitution permuation networks

These types of ciphers are a subset of block ciphers. They use two types of boxes (like the ENIGMA rotors). The following ciphers below are symmetric

key ciphers where the same key must be known on both sides of the channel.

S-box performs substitutions with $(2^n)!$ bijections possible

P-box swaps bit positions (there must be the same number of 1s and 0s as started). This has $n!$ possibilities.

Feistel and DES

Feistel ciphers are a class of Substitution permuation networks (including DES) that are easy to reverse. Feistel ciphers have encryption and decryptions that are almost identical (See page 14 of 14 May slides). Should be noted that when decrypting the keys should use the subkeys in reverse order.

In a general form our plaintext is split into two part L_0R_0. For each round $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$. The aim is to find a function for f that is **highly non-linear**. This f is where the security comes from.

DES algorithm

DES uses 64 bit blocks, 32 for the left and 32 for the right. The master key is 56 bits and each subkey is 48 bits.

There is an inital fixed permutation IP followed by 16 round Feistel cipher followed by IP^{-1}. Note: IP does not provide any additional security to the cipher, it is simply there from legacy hardware implementations.

Each subkey is a permutation of 48 bits from the master key.

Below is the definition of f in DES. For context, $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$

1. Expand R_{i-1} using a predefined "expansion permuation" that repeats 16 elements, making it 48 bits long
2. XOR the subkey K_i
3. Split our result into 6 bit pieces and send them through S-boxes

4. Apply a predefined permutation and return this result

How to read a DES S-box It's annoying because the **first and last bit** determine the row and the middle 4 bits determine the column.

Cryptanalysis

This subsection will consider a simple substitution permutation network, no f function present.

Linear cryptanalysis

This method requires a known plaintext attack to deduce pieces of the final subkey.

The aim of linear cryptanalysis is to note how frequently bits of plaintext are mapped to bits in the final Feistel round. In a perfect cipher we should see a completely homogenous mess of binary digits but in reality there's a **bias**. We then aim to brute force all possible bits in the subkeys until we find one that gives the same bias as we find in the known plaintext.

The value of the bias is how much over or under you are from $^1/_2$. e.g. a bias of $+^1/_4$ if we find a frequency of $^3/_4$.

We assume our bits are independent of each other (in reality this isn't true but our simplification allows good mathematical work).

We analyse the S-box to find it's biases much like on page 3 of the 16 May slides. We consider input bits and their corresponding output bits. We then consider some relationship between the input bits and another with the output bits and see if they match up.

From here we construct a massive table from the different relationships we could find from the steps above (see page 6 of the 16 May slides). The values b and 6 represent which bits are being considered in our relationship.

We perform what is shown on page 10 of the 16 May slides. Note the 2^3 which is justified by the piling up lemma.

The piling up lemma is proved like in `https://en.wikipedia.org/wiki/Piling-up_lemma`.

I could explain the rest but instead just look at the slides for this part the diagrams are way more helpful.

Differential cryptanalysis

I hope you remember how to do this from the assignment.

Triple DES (why not double DES)

It isn't on the slides so I'm not going to be able to really explain it (See 22:33 9 May recording). With double DES a "meet-in-the-middle-attack" can be executed. So instead of $2^{56*2} = 2^{112}$ we get $2^{56} + 2^{56} = 2^{57}$.

AES algorithm

We split our plaintext into blocks of 128 bits or 16 bytes. In this way we produce a 4×4 state array. This state array is read column by column top to bottom left to right (See page 5 of 9 May slides).

In AES after each round of encryption the key used in that round is called a round key. It should be noted each round key K_i is generated from the master key.

1. Add round key K_0
2. For each round (excluding the final round)
   (a) Substitute bytes
   (b) Shift rows

(c) Mix columns

(d) Add round key K_i

3. For the last round

   (a) Substitute bytes

   (b) Shift rows

   (c) Add round key K_i

Substitute bytes using the S-box

We use the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$ as our *magic* polynomial we perform modulos with.

Our S-box with the following algorithm

```
// x is the initial input
// inverse_wit_magic_polynomial(0) = 0
x = inverse_with_magic_polynomial(x)
// c(i,x) shifts the bits i places
x = c(0,x) + c(1,x) + c(2,x) + c(3,x) + c(4,x) + binary(01100011)
```

Shift rows

This is a very simple operation which could be explained by the diagram on page 7 of the 9 May slides. But I will endeavour to explain it in words. Consider the state array. The first row of the state array will not be shifted at all. The second row will shift "left" by one. The third column shifted left by two and so on.

Mix columns

Mixing is used to induce the avalanche effect. We mix the columns of our state array using the equation below, consider the column with elements b_i.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}_{new} \rightarrow \begin{bmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}_{old}$$

Generate the next round key

Not explained on slides so I'm guessing we don't worry about that.

Decrypting AES

Decrypting is much like encrypting. The only differences are:

- the use of the inverse S-box
- using an inverse matrix for the mix columns
- cycle in the other direction for shift rows
- use round keys in reverse

AES modes

DES had four modes of operation, AES extends this to six. The first four operations were those supported by DES.

electronic codebook mode (ECB) Simple use case of the cipher. Each block is encrypted with the same master key. This is weak since identical plaintext blocks yield identical ciphertext blocks.

cipher feedback mode (CFB) The next plaintext block to be encrypted x_{i+1} is XORed with the previously constructed ciphertext y_i before getting encrypted. This prevents the weakness in the previous mode, identical plaintext blocks do not have identical ciphertext.

To decrypt, decrypt the first block then XOR it to the second decrypted block. Rinse and repeat.

output feedback mode (OFB) We use AES to generate keys which we XOR onto our plaintext (the plaintext does not get encrypted with AES). We start with some initial vector (the receiver may not know it but that's okay since we send an encrypted version as part of the message) which we successively encrypt and XOR onto our plaintext.

cipher block chaining mode (CBC) Kind of like OFB. We start with some initial vector and encrypt it then use this to XOR our plaintext into ciphertext. From here we encrypt our new ciphertext to generate our new XORing term.

counter mode Not covered

counter with cipher-block chaining mode (CCM) Not covered

Chapter 6

Unconditional security

Perfect secrecy

It's not hard to show that $|\mathscr{K}| \geq |\mathscr{C}| \geq |\mathscr{P}|$.

If $\mathscr{K}$ was a smaller space than $\mathscr{C}$ then there would be places in $\mathscr{C}$ that don't get mapped from $\mathscr{P}$ since there isn't a key that can get you there. Therefore, $P(x|y) = 0 \neq P(x)$ (there exists a plaintext that knowing the ciphertext means its impossible, no key can get you to y from x).

Shannon's Theorem

Shannon's Theorem states: If $|\mathscr{K}| = |\mathscr{C}| = |\mathscr{P}|$, we get perfect secrecy if and only if each key (including identity key) is chosen with equal probability and for all possible plaintext and ciphertext pairs there exists a **unique** key that maps the two.

Shift cipher (again!)

An example of perfect secrecy is actually the shift cipher! The calculations are ignored here but the shift cipher offers perfect secrecy **if** the plaintext is a single letter.

Also note we must allow for $K = 0$ (the key where nothing gets shifted). Otherwise knowing the plaintext rules out a possibility.

One time pad

Hopefully won't be brought up. But a quick skim of pg 53 of Cryptography Theory and Practice should be sufficient.

The issue with a one time pad is that the key is as long as the plaintext.

Mixed alphabet cipher

A mixed alphabet cipher permutes the alphabet in the obvious way. The identity key (where everything is mapped to itself) should also be possible.

Chapter 7

Secret sharing schemes

What if you want a number of people to hold unique keys (shares) that when combined allow us to construct the key and unlock the encryption. But if a single person (or even a few) cannot decrypt on their own.

Simple example

Once you encrypt something you split the key into many parts. Only when everyone is present can we reconstruct the key and unlock the encryption.

A problem with this method is that as the number of people begin adding their keys the ability to brute-force becomes easier.

Another problem is if someone on the team dies/loses their key. Though that is failure by design.

An additional aim

Our simple example has shown two flaws in our design.

We would like it to be a threshold scheme instead of a 100% scheme. We shall label this threshold number $t < n$.

We would also like that if our threshold for unlocking is t then all possible subsets of people $< t$ will have equally little information as someone on the street. Having a part of the key does not mean you have some advantage to unlocking.

Shamir's scheme

We construct a secret polynomial with degree $t - 1$. Each share is a random point on the polynomial.

This polynomial is unique and we prove it here. Assume $h(x) \neq g(x)$ yet they both satisfy the t points. By definition of lying on the polynomial we have $g(x_0) = y_0 = h(x_0)$. By subtracting them, $g(x) - h(x)$, we would get a polynomial with degree $\leq t - 1$ but with t roots

From highschool math it should be obvious we can construct such a $t-1$ degree polynomial with t points.

Point to polynomial

We may either use Lagrange interpolation (not shown here, hopefully not assessed) or simultaneous equations (also not shown here because you should know how to do that).

Chinese remainder theorem

Apparently there's a way to construct a threshold scheme with the Chinese remainder theorem. I have no idea how.

Cheating

Suppose t people get together to construct the key. One of these people does not intend to give their real key. When everyone adds their share to construct the key, this devious person adds a false share. The key construction fails **but** the cheater is now in possession of a broken key with the advantage of knowing *how* it is broken.

It's possible that this devious person can use this broken key to construct the actual key on their own!

Shamir's scheme

Apparently you can cheat in this scheme. It was never stated how though.

Chapter 8

Zero knowledge proofs

The explanation at 32:20 on the May 28 recording does a far better job than what I could ever do.

A strange but important feature is for third parties to be unconvinced of the entire process. Think of a magic trick between two people on stage, the magician and a brave audience member. To the audience member on stage the magic seems very impressive and believable. But you're not convinced since that audience member may be an actor that was planted in the audience for this very trick.

Graph isomorphisms

There are two graphs that are isomorphic (they're structurally equivalent but relabelled). To find the relabelling is actually hard for a computer to do (**provably secure**).

To prove I know the relabelling a pick one of the graphs and construct a new labelling from it. That is different from the original two. The "sceptic" (a term I made up) asks how to relabel my new graph with **one** of the original graphs.

If it's the one I constructed it from then I just reveal my method. Note, anyone could do this step without knowing the secret relabelling.

On the other hand, if I'm asked for the relabelling from the graph I did not construct it from then I use my secret relabelling to transform my method for the second graph (this is computationally easy). Note, this is the part that only someone with the secret could do.

The sceptic receives information from the constructed graph to either of the original graphs however he cannot create the link between the graphs since he only has one half of the puzzle. The sceptic **may not** ask for the relabelling to **both** graphs.

Weakness

The whole "graph relabelling is hard" is actually no longer true after 2015. But that's okay since many other difficult problems could be used as replacements (e.g. discrete logs, Hamiltonian circuits, etc.).

Elliptic Curve Cryptography

Recall ElGamal, it did not use the fact that $\mathbb{Z}_n^\times$ was a field (no additions present) it only made use of the fact that $\mathbb{Z}_n^\times$ is a group.

Elliptic Curve Cryptography simply replaces our modulo group with another group where discrete logarithms are still hard. In this the elliptic curve groups.

$$y^2 = x^3 + ax + b$$

The operation (addition analogue) is determined by taking two points on this curve and finding where this line intersects the curve. We then flip along the horizontal axis.

If it never intersects the curve then it's considered ∞.

∞ is the identity.

The derivation is not presented here but the formulas should easily give you the points.

$$\lambda = \frac{y_0 - y_1}{x_0 - x_1}$$
$$x_2 = \lambda^2 - x_0 - x_1$$
$$y_2 = \lambda(x_0 - x_2) - y_0$$

Advantages and weaknesses

Comparable security to RSA but with smaller keys (great for hardward implementations).

The problem is that points on a curve are hard to work with. There is a workaround called ECIES that we did not discuss.

Also it's vulnerable to quantum computing.

Chapter 9

Quantum cryptogrpahy

Quantum cryptography allows us to detect eavesdropping.

BB84 key exchange protocol

This protocol is used to agree on a secret key consisting of random bits. It won't be able to encrypt a desired plaintext since this protocol destroys 50% of the message.

The protocol

Consider Alice and Bob. Alice and Bob independently chooses a basis. Alice then sends Bob a photon with a specific polarisation. Alice and Bob then compare the bases they each chose. If it turns out they had the same basis then this bit will be part of the shared key. If it turns out they don't have shared bases then this photon is discarded. They then publicly compare a few of their key bits to ensure correctness. If all bits match up the remaining hidden bits are used as the secret key.

The security is revealed from considering an enemy, Eve. If she chooses the

same basis as Alice then she can properly read the bit and send it to Bob much like a man in the middle attack. If her basis is wrong then Alice destroys the photon, Eve gives Bob random noise.

When Eve is listening Bob will receive a quarter of the bits incorrectly. Thus if Alice and Bob were to compare segments of their key bits in public they would notice key bits are incorrect and conclude someone was listening.