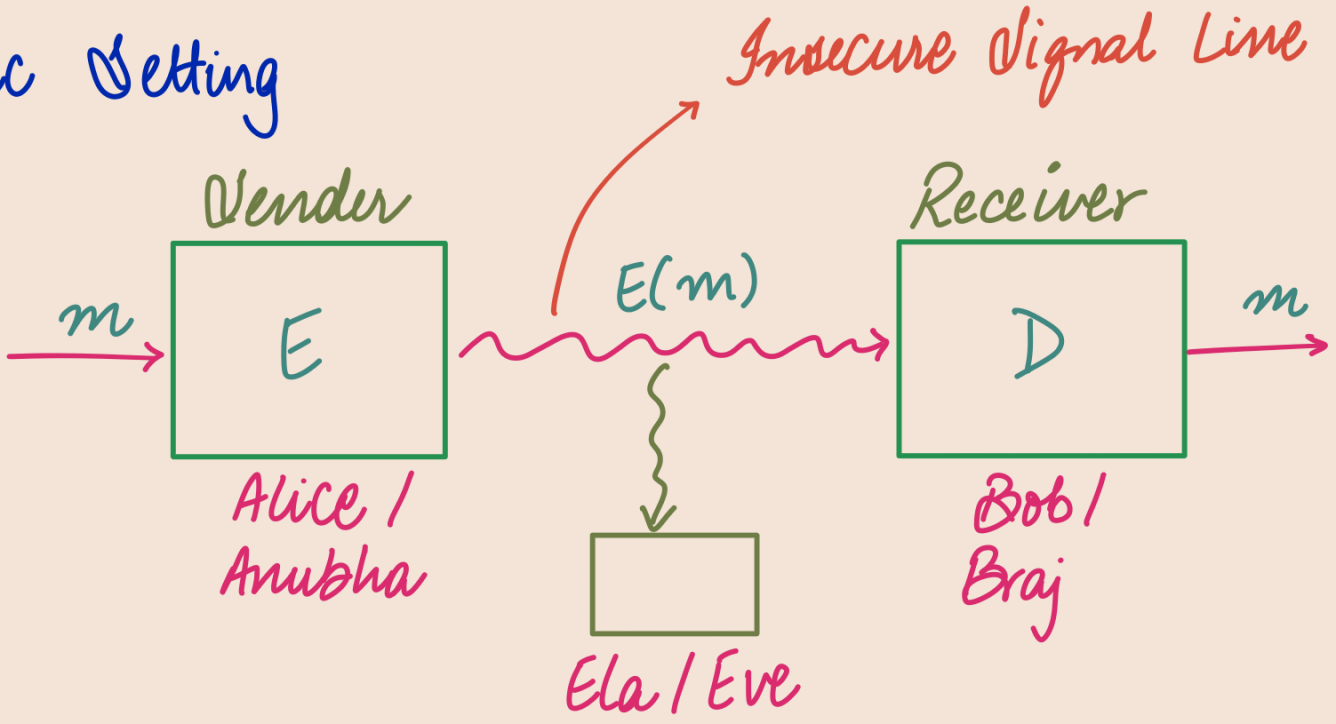
CS641

Modern Cryptology

- Cryptography → Encoding messages in a secure way
- Cryptanalysis → Decoding messages that one wasn't supposed to know

Basic Setting



- Anubha's job is that of cryptography
- Ela's job is that of cryptanalysis + eavesdropping
- E : Encryption
  D : Decryption s.t. $D(E(m)) = m$
- Ela can only get hold of $E(m)$

Caesar's Cipher : Replace each letter in m by a letter 3 places (cyclically) to the right.

Ex: This → Wklv

First Principle of CryptoGraphy (+ Assumption)

Everything that is fixed will eventually be known to everyone.

$\Leftrightarrow$ Everything fixed is known to everyone.

$\Rightarrow$ Need to have a variable element in D, say decryption key k_D. So that even if D is known, E(m) cannot be decrypted w/o key

$\Rightarrow$ k_D needs to keep changing.

But, if diff. keys can decrypt same message, then keys also fixed

$\Rightarrow$ E needs to have a variable element, k_E as well.

- Now, $\because$ E & D, the encryption methods, are fixed, assume that they are known to Ela. Though, k_E & k_D are still not known to Ela

Properties required

1) $D(E(m, k_E), k_D) = m$
2) W/o knowledge of k_D, Ela should not be able to decrypt.
3) D & E should be efficient.

128 bits enough to prevent bruteforce attacks

Substitution Cipher

- Generalization of Caesar's Cipher
- 1-1 mapping, $k_E : [a-z] \longrightarrow [a-z]$; $k_D = k_E^{-1}$

$$E(m) = E(a_1 a_2 \dots a_n) = k_E(a_1)\, k_E(a_2) \dots k_E(a_n)$$

- # of $k_E = 26! \approx (26/e)^{26} \approx 10^{26}$

Frequency Analysis

In the English Language, each of the 26 letters have typical frequencies. These frequencies can be used to "guess" substitution key letter-by-letter.

Some other peculiarities of the language can also be used. For example, the letter q is almost always followed by the letter u.

☆ Having a large number of keys does not guarantee security. However, they are a necessity for security, just not a sufficiency.

Permutation Cipher

Key is a permutation of $[1, n]$ {n: size of block}. To encrypt, break the plaintext into chunks of size n.

Ex: $m = a_1 a_2 a_3 \dots a_n \; a_{n+1} a_{n+2} \dots$

$\longrightarrow E(m) = a_{\pi(1)} a_{\pi(2)} a_{\pi(3)} \dots a_{\pi(n)} \; a_{n+\pi(1)} a_{n+\pi(2)} \dots$

π: a permutation of $[1, n]$

n: fixed; assume it to be known.

→ To find π, employ pairwise frequency analysis on blocks of size n.

More Ciphers

- Substitution – Permutation
- Multiple substitutions

Types of Attacks: Need to formalize the ways Ela can try to break the cryptographic methods used by Anubha-Braj.

1) Ciphertext only attacks: (weakest form of attack)
   Ex: Frequency analysis, BruteForce attack.
   Ela only gets $E(m)$
2) Known plaintext attack:
   Ela knows plaintexts corresponding to a few ciphertexts.
3) Chosen plaintext attack:
   Ela can get specific plaintexts encrypted.
4) Chosen ciphertext attack:
   Ela can get plaintext corresponding to some of the ciphertexts.
5) Chosen plaintext + ciphertext attack

↳ In the order of increasing strength of attacks.

Assumption

Ela can carry out both chosen plaintext and ciphertext attacks.

Linear Ciphers

- Break the message into blocks of size n and view the blocks as vectors in an n-dimensional space.
- View the key k_E as an $n \times n$ invertible matrix

$$E(v) = k_E \cdot v$$

$$k_D = k_E^{-1} \qquad D(E(v)) = k_D \cdot E(v) = k_D \cdot k_E \cdot v = k_E^{-1} \cdot k_E \cdot v = v$$

- So, permutation cipher is a linear cipher.

- Similarly, substitution ciphers is also a linear cipher. View the letters as a 26-dim. long one-hot vectors. Then, the key is a 26×26 matrix.
- **Breaking Linear Ciphers:**
  Known plain-text attack using $\begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix}$, etc.
  OR
  In general, known plain-text attack using plaintexts with n linearly independent vectors.

☆ Linear Ciphers can be broken using known plaintext methods. ∴ Our encryption methods cannot be linear ciphers.

☆ Need to use an invertible non-linear transformation for encryption.
But, a linear Transformation on the entire block helps mixing the information well.
∴ A combination of the two is desirable.

Thm. Let $u \in \mathbb{Z}^b$, $u \neq 0$, and $K \in \mathbb{Z}^{b \times b}$. Then, over random choices of K, $K \cdot u$ is a random vector in $\mathbb{Z}^b$

- Given $u \neq 0$ & c, let i^{th} entry of u, $u_i \neq 0$.
- Probability that $c = K \cdot u$ equals the probability that $K_i = 1/u_i \left(c - \sum_{j \neq i} u_j K_j\right)$
- ∴ K_i is a random vector, and c has a similar distribution, thus, $c = K \cdot u$ is a random vector

- But, we require K to be invertible.
- ∴ Encrypted text is required to be decrypted, all transformations done during encryption need to be invertible. Easy for LTs, but NLTs typically not invertible.

A generic way of doing it is given by **Feistel**.

Feistel Structure

Let f be any NLT with $f: \{0,1\}^n \to \{0,1\}^n$

- Define as: $g: \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^n \times \{0,1\}^n$
$$g(a, b) = (b, a \oplus f(b))$$ (⊕ → XOR)
- Then, g is clearly an NLT but, it is also invertible:
$$g^{-1}(b', a') = (a' \oplus f(b'), b')$$
$$\Rightarrow g^{-1}(g(a,b)) = g^{-1}(b, a \oplus f(b)) = (a \oplus f(b) \oplus f(b), b) = (a, b)$$
- ∴ Feistel structure can be used to ensure invertibility given any transformation f
- But, it only encrypts half of the input. A work-around is applying g twice to completely transform input.

Generally, greater number of rounds provide more security.

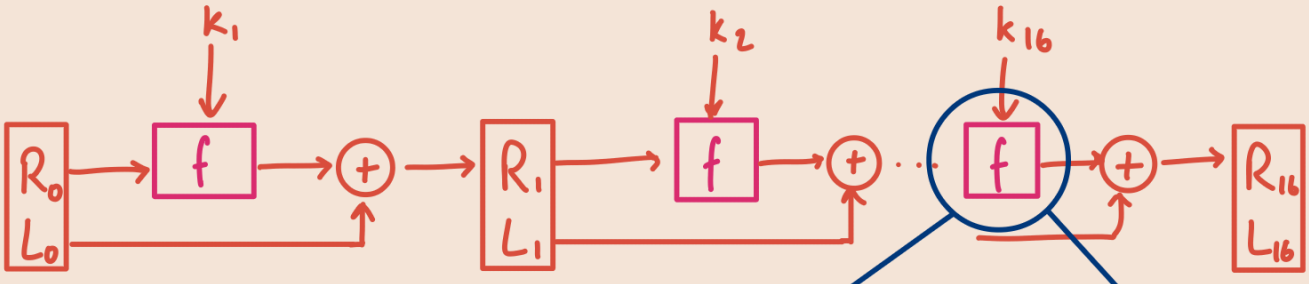
DES (Data Encryption Standard)

- Designed in 1974, used until 2001.
- Adopted by NBS (US) in 1976.

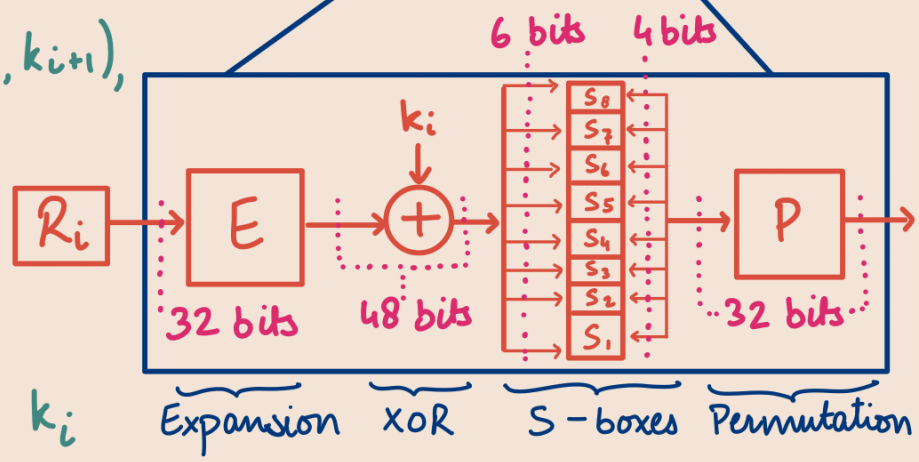
Parameters:
- Block Cipher w/ blocksize = 64 bits
- Key size = 56 bits, use 16 rounds of Feistel Structure.

- 1 byte = 8 bits : MSB stores parity of remaining 7 bits for robustness.

Structure



- $R_{i+1} = L_i \oplus f(R_i, k_{i+1})$,
  $L_{i+1} = R_i$,
  $\forall\ 0 \leq i < 16$
- Function f also depends on round key k_i



Round Keys:
- Key k_i is called round key for round i, $1 \leq i \leq 16$
- Each k_i is 48 bits long & is a fixed subset of 56 bits of the whole key.
- Plaintext block is $L_0 R_0$, $|L_0| = |R_0| = 32$ bits
- Input to round $i+1$ is $L_i R_i$ & its output is $L_{i+1} R_{i+1}$, with $|L_{i+1}| = |R_{i+1}| = 32$ bits.

f: Non-linear function. Input is right half of round input (32 bits), round key (48 bits) and produces a 32-bit output.
- Further divided into 4 operations, 3 of which are linear and 1 is non-linear (s-boxes)

Expansion E: Produces 48 bit output by replicating 16 bits of input.

Input: $b_0\ b_1 \dots b_{31}$

Output: $b_{31}\ b_0\ b_1\ b_2\ b_3\ b_4\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8\ b_7\ b_8 \dots b_{29}\ b_{30}\ b_{31}\ b_0$

- Basic idea is to pair adjacent bits & repeat alternate pairs (b_0 paired with b_{31}); For example, $b_1\ b_2$ is not repeated but $b_{31}\ b_0$, $b_3\ b_4$ are repeated.

Permutation P: Shuffles input bits as:

Input: $b_0\ b_1\ b_2 \dots b_{31}$

Output: $b_{15}\ b_7\ b_{19}\ b_{20}\ b_{28}\ b_{11}\ b_{27}\ b_{16} \dots b_{21}\ b_{10}\ b_3\ b_{24}$

- Primary aim is to shuffle bits so that all 4 bits in a block move to different blocks, for each of the 8 blocks
- For example, $b_0\ b_1\ b_2\ b_3$ are not in the same 4-bit block in the output.

XOR ⊕: The 48-bit expansion of the input half is bitwise XOR-ed with the round key k_i

S-boxes:
- Only nonlinear operation in entire algorithm
- There are 8 s-boxes, each is a mapping from 6 bits to 4 bits
- Each of the 8 boxes are distinct transformations

S-1:
- Columns indexed by middle 4 bits of input, and rows indexed by first & last bits of input.
- Numbers are b/w 0-15 representing 4 bit outputs.
- Every row has all 16 numbers occurring once.
- Small s-boxes for better speed. Same reason

for other choices of operation.

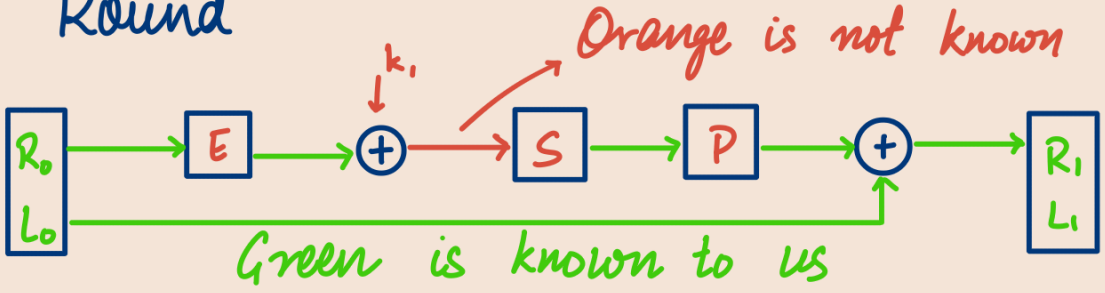
- The box itself:

S-1	0000	0001	0010 = 2	3	4	5	6	7	8	9	10	11	12	13	14	15
00	1110	0100	1101 = 13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	...	...												...	⋮
10	4	...	⋱												⋱	⋮
11	15	...	... ⋱												...	13

Cryptanalysis of DES

- Brute-force attack to find out the key requires $2^{56} \approx 10^{17}$ operations.
- Frequency analysis doesn't work ∵ variations in frequencies flattened out by multiple transformations
- Assume stronger forms of attacks: known-plaintext.
- Start w/ lesser number of rounds.

DES: One Round



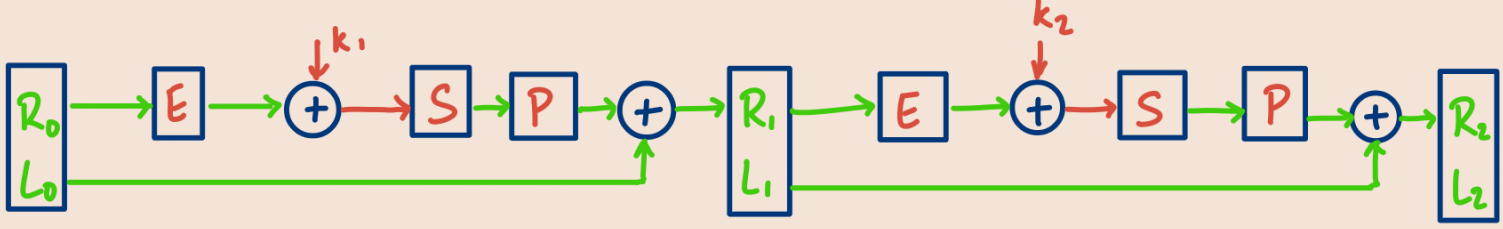
- We know the output of S-boxes & E.
- Let $E(R_0) = \alpha_1 \alpha_2 \ldots \alpha_8$ w/ $|\alpha_i| = 6$
- $E(R_0)$ gets XOR-ed with key $k_1 = k_{1,1}\, k_{1,2} \ldots k_{1,8}$ w/ $|k_{1,i}| = 6$ & $\beta_i = \alpha_i \oplus k_i$
- 6-bit string β_i is input to i^{th} S-box
- Let $\gamma_i = S_i(\beta_i)$ w/ $|\gamma_i| = 4$
- Each γ_i & α_i is known
- ∵ γ_i is known, we can look up the table for S_i to find out which inputs can

produce γ_i as output.

- S_i has exactly four occurrences of γ_i
- Let X_i be the set of inputs to S_i that produce γ_i as output. $|X_i| = 4$. $\beta_i \in X_i$. $K_i := \{\alpha_i \oplus \beta \mid \beta \in X_i\}$
- $\because k_{1,i} = \alpha_i \oplus \beta_i$, we have $k_{1,i} \in K_i$ & $|K_i| = 4$
- $\therefore \forall\ 1 \le i \le 8,\ k_{1,i} \in K_i,\ |K_i| = 4$
- Concatenating strings from K_i, we get $4^8 = 2^{16}$ strings, one of which is k_1. Significant improvement on brute-force attack already.

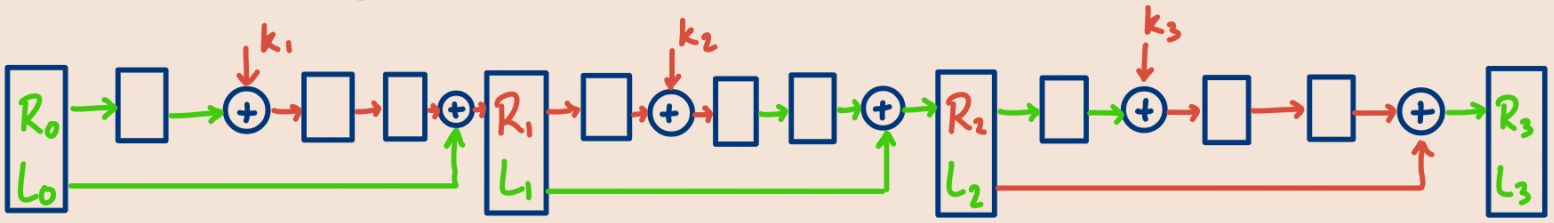
- Take another pair of plaintext & corresponding ciphertext block.
  Repeat the same analysis as above to get sets $K_i'|_{i=1}^{i=8}$, $|K_i'| = 4$. $k_{1,i} \in K_i' \Rightarrow k_{1,i} \in K_i' \cap K_i$
- Likely that $|K_i \cap K_i'| = 1$, uniquely identifying $k_{1,i}$ If it is not unique, repeat.
- Once all sets have size 1, the entire key k_1 is uniquely identified.

DES: Two Rounds

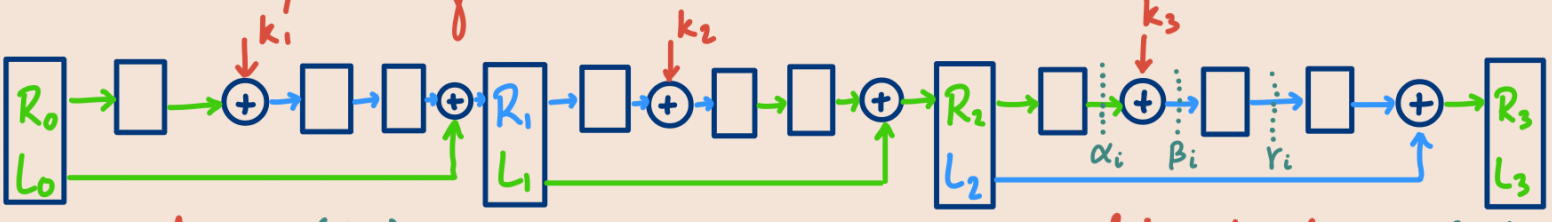


- We know the output of S-boxes as well as output of E for both rounds.
- Using the same strategy as before, we can extract k_1 as well as k_2 easily.

DES : Three Rounds



- $R_1 = L_2$ is not known.
- There is no round for which outputs of both E & S-boxes are known.
- Cannot use earlier strategy for this.
- Idea: Use two inputs & see how their XOR moves.
- Let $L_0 R_0$ & $L_0' R_0$ be two inputs. Then
  $$L_0 \oplus L_0' = R_1 \oplus R_1' = L_2 \oplus L_2'$$
- Let / denote values whose XOR's we know for two inputs of the above kind. Then:



- Let $E(R_2) = \alpha_1 \alpha_2 \ldots \alpha_8$, $|\alpha_i| = 6$. Similarly, $E(R_2')$.
  Let $\beta_i = \alpha_i \oplus k_{3,i}$, $|k_{3,i}| = 6$. Similarly, β_i'.
  Let $\gamma_i = S_i(\beta_i)$. Similarly, γ_i'. We know $\gamma_i \oplus \gamma_i'$

Q. How many pairs (δ_i, δ_i') exist s.t. $S_i(\delta_i) \oplus S_i(\delta_i') = \gamma_i \oplus \gamma_i'$

$$X_i := \{(\delta_i, \delta_i') \mid S_i(\delta_i) \oplus S_i(\delta_i') = \gamma_i \oplus \gamma_i'\}$$

Lemma: $|X_i| = 256$

Proof: $\exists$ 16 pairs $(\epsilon_i, \epsilon_i')$ s.t. $\epsilon_i \oplus \epsilon_i' = \gamma_i \oplus \gamma_i'$
For each ϵ_i $\exists$ 4 values of δ_i s.t. $S_i(\delta_i) = \epsilon_i$
Similarly for each ϵ_i'. $\therefore |X_i| = 16 \cdot 4 \cdot 4 = 256$

Q. How many XOR values are there for pairs in X_i?
Trivial upper bound = 64 $\because$ $|\delta_i| = 6 = |\delta_i'| \Rightarrow 2^6 = 64$

But, all pairs s.t. $\delta_i \oplus \delta_i' \neq \beta_i \oplus \beta_i'$ (known to us as well) can be removed from X_i.

Q. What is the size of X_i now?

If $|X_i| = 64$, then:

$\delta_i \oplus \delta_i' = \beta_i \oplus \beta_i' \Rightarrow \delta_i' = \delta_i \oplus \beta_i \oplus \beta_i'$

$S_i(\delta_i) \oplus S_i(\delta_i \oplus \beta_i \oplus \beta_i') = \gamma_i \oplus \gamma_i'$

$\Rightarrow S_i(\delta_i \oplus \beta_i \oplus \beta_i') = S_i(\delta_i) \oplus \gamma_i \oplus \gamma_i' \quad \forall \delta_i$

$\oplus \sim +,\ \delta_i \sim x,\ \beta_i \oplus \beta_i' \sim c,\ \gamma_i \oplus \gamma_i' \sim d,\ S_i \sim f$

$\rightarrow f(x + c) = f(x) + d$ ⇝ Similar to a Linear function

- This would make S_i vulnerable to linear attacks. But, in reality, due to the non-linearity of S_i, we often have $|X_i| \leq 16$

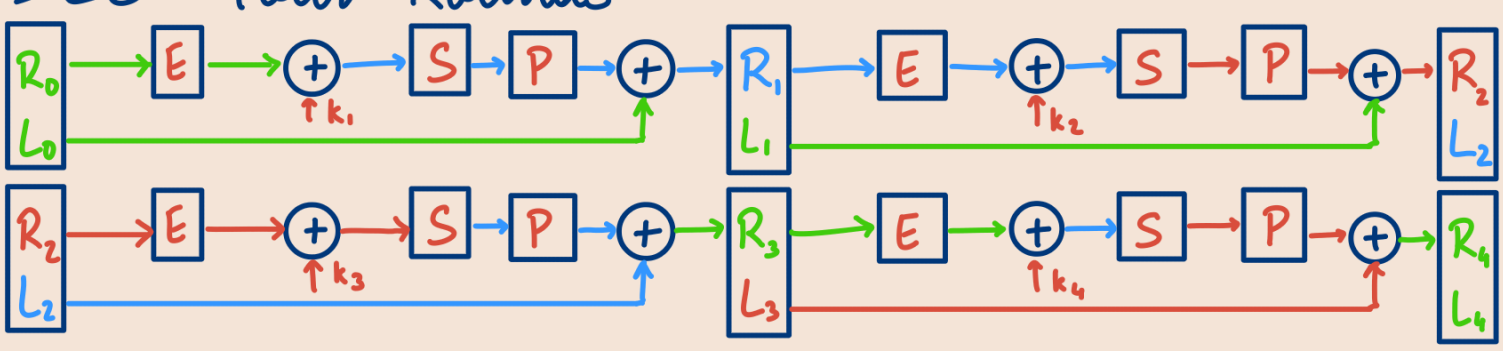
Let $K_i := \{\alpha_i \oplus \delta_i \mid (\delta_i, \delta_i') \in X_i\}$ & $|X_i| \leq 16$

$\Rightarrow |K_i| \leq 16$ w.l. $k_{3,i} \in K_i$

- A brute-force attack requires $16^8 = 2^{32}$ trials.
- Instead, repeat this analysis for different input pairs and keep taking intersections until we have a single candidate for k_3.
- Once we know k_3, we can work to know $L_2 = R_1$ which reduces the system to DES w/ 2 rounds which we know how to break.

☆ Chosen plaintext attack was required to break DES: 3 rounds.

DES: Four Rounds



- Just because we know the XOR of the inputs to an S-box does not mean that we can know the XOR of the output due to the design of S-boxes.
- Can work with finding the key upto some probability, ie, guessing the output XOR.
- For example,

S1-box: Input XOR = 001100 $\Rightarrow$ Output XOR = 1110
(with probability 14/64)

{64 pairs of input (2^6) w/ given input XOR. 14 are s.t. output XOR = 1110. One would expect 4 but S-boxes do not work like that}

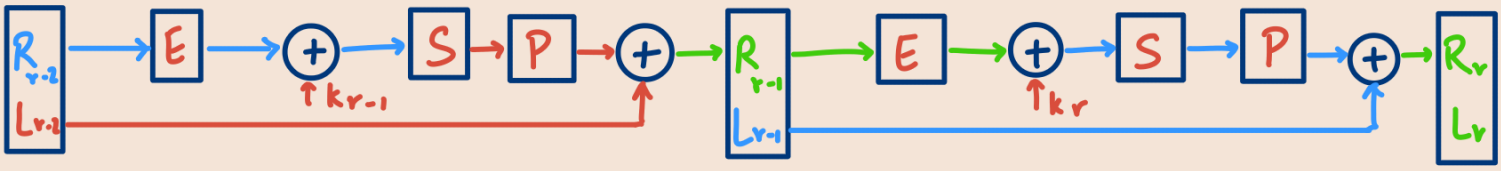
- 3 pairs w/ probability higher than 14/64 but this will suffice.
- $\therefore$ we would like for input XOR of S1-box of round to be 001100. Can feed 000000 to other S-boxes so that we know the output XOR (guaranteed to be 0000).
- From here on, represent numbers in hexadecimal
- Let $X_i = \{(\delta_i, \delta_i') : \delta_i \oplus \delta_i' = \beta_i \oplus \beta_i', S_i(\delta_i) \oplus S_i(\delta_i') = \gamma_i \oplus \gamma_i'\}$, $|X_i| \leq 16$. Let $K_i = \{\alpha_i \oplus \delta_i : (\delta_i, \delta_i') \in X_i\}$
$\Rightarrow |K_i| = |X_i| \leq 16$
- We have, with probability $\geq 14/64$, $k_{4,i} \in K_i$
- Can repeat the experiment multiple times & take the most frequently occurring element amongst these sets and let this element be our guess for $k_{4,i}$. Then, we would like to know $\mathbb{P}[k_{4,i} \neq m \in K_i]$. $|k_{4,i}| = 6 \Rightarrow 64$ candidates for m and $|K_i| = 16$
$\therefore \mathbb{P}[k_{4,i} \neq m \in K_i] = 16/64$
- Compute l sets $K_i^1, K_i^2, \ldots, K_i^l$. Expected no. of sets in which $m \neq k_{4,i}$ occurs: $\sum_{j=1:l} \frac{|K_i^j|}{64} \leq \frac{16l}{64}$ } Assumption

Similarly, expected no. of sets $k_{4,i}$ occurs in

$$\geq \underbrace{14\ell/64}_{\text{output XOR is the one we wanted}} + \sum_j |K_i^j|/64 \cdot \underbrace{50/64}_{\text{output XOR is wrong in } j^{th} \text{ group}}$$

- When $|K_i^j| = 16$ ↗ ↳ $\frac{14\ell}{64} + \frac{50\ell}{64} \cdot \frac{1}{4} = \frac{26.5\ell}{64}$
- Taking $\ell = 64$ gives a substantial difference of 10 frequencies.

DES: r Rounds



Characteristic: An s-round characteristic is a tuple $(x_0, y_0, p_1, x_1, y_1, p_2, x_2, y_2, \ldots, p_s, x_s, y_s)$ where $|x_i| = |y_i| = 32$, $0 \leq p_i \leq 1$, and when input XOR of round $i+1$ is $x_i y_i$, its output XOR is $x_{i+1} y_{i+1}$ with probability p_{i+1}

Observation: To break r-round DES, we need an $(r-2)$-round characteristic.

- A 2-round characteristic was used to break 4-round DES: $(6\bar{0}, \bar{0}, 1, \bar{0}, 6\bar{0}, {}^{14}/_{64}, 6\bar{0}, 00828000)$
  Trailing Hex-digits are all 0
- The probability of output XOR being $x_{i+1} y_{i+1}$ given that the input XOR was $x_0 y_0$ is π_{p_r} which will keep decreasing with increasing i.
- If probability is p, then the key will be present in $\approx p \cdot \ell + (1-p) \cdot \ell/4$ sets, while other values will be present in $\approx \ell/4$ sets.

- So, it is present in $\approx 3pl/4$ more sets. We would need $l \approx 20/p$ to be sure of a candidate.
- This technique of breaking DES using XOR values is known as **Differential Cryptanalysis**. Proposed by Biham & Shamir in 1990.

- 2-round characteristic: $(\bar{0}, 1\,6\,\bar{0}, 1/234, 1\,6\,\bar{0}, \bar{0}, 1, \bar{0}, 1\,6\,\bar{0})$ can be concatenated r times to create a $2r$-round characteristic.
- For 16 rounds, need a 14-round characteristic
  $p = (1/234)^7 \approx 1/2^{55} \Rightarrow l \approx 2^{59}$
  $\therefore$ 16 is the minimum number of rounds that makes DES fully resistant against differential cryptanalysis.

- **Linear Cryptanalysis**: Proposed by Matsui in 1994.

- Property of S5: Input $= b_0 b_1 b_2 b_3 b_4 b_5$, Output $= c_0 c_1 c_2 c_3$
  Then $b_1 \oplus c_0 \oplus c_1 \oplus c_2 \oplus c_3 = 0$
  with probability $12/64$ → Linear Nature
  (Very Low. Would think 1/2)
- Using linearity, consider round i; Input $= L_{i-1} R_{i-1}$
  then we have, ($\oplus$ these four bits of L_{i-1} together)

$$R_{i-1}[15] \oplus k_i[22] \oplus L_{i-1}[7, 18, 24, 29] \oplus R_i[7, 18, 24, 29] = 0$$

  (15^{th} bit of R_{i-1})
  with probability $12/64$
- 3 round DES:

$$R_0[15] \oplus k_1[22] \oplus L_0[7, 18, 24, 29] = R_1[7, 18, 24, 29]$$
(= holds with probability 12/64)

$$= L_2[7, 18, 24, 29] = R_3[7, 18, 24, 29] \oplus f(R_2, k_3)[7, 18, 24, 29]$$
($f(R_2, k_3)$: Output of 3^{rd} round after P)

- Guess 6 bits of k_3. If guess is wrong, then eqn

holds for roughly half of the times. If guess is right, it only holds 12/64 of the times.

This way, we can get 6+1 bits of the key, 6 bits from k_3 & 1 bit from k_1.

- Doing the same for 3rd round would lead to 7 more bits. 1 from k_3 & 6 bits from k_1.

- Can form an equation for 16-round DES too which holds w/ probability $1/2^{22}$.

Then $\approx 2^{47}$ plaintext attacks required for 14 bits of the key.

The rest 42 bits can be found from BruteForce attacks. So, overall $\approx 2^{47}$ attacks required.

- However, this can be improved by drawing useful inferences (through frequency analysis) about the way English messages are encoded using ASCII codes (for example, the MSB is a 0 for alphanumeric characters).

Doing so could allow us to break DES using ciphertext-only attacks.

Finite Rings and Fields

- Rings: Collection of numbers with +, −, *
  Ex: Integers w/ known operations. (excepting division by zero)
- Fields: Collection of numbers with +, −, *, /
  Ex: Rationals, Reals, Complex Numbers
- Groups: A set of elements with one operation :

Let $(G, \cdot)$ be a group. Then:

1) $a \cdot b \in G \quad \forall \; a, b \in G \rightsquigarrow$ Closure
2) $a \cdot (b \cdot c) = (a \cdot b) \cdot c \quad \forall \; a, b, c \in G \rightsquigarrow$ Associativity
3) $\exists \; \underbrace{e \in G}_{\text{Identity element}} : \; a \cdot e = e \cdot a = a \quad \forall \; a \in G \rightsquigarrow$ Existence of "e"
4) $\forall \; a \in G \quad \exists! \; \underbrace{b \in G}_{\text{Existence of: Inverse of a}}$ s.t. $a \cdot b = e$

Examples: $(\mathbb{Z}, +)$, $(\mathbb{Q}, +)$, $(\mathbb{Q} - \{0\}, *)$, $(\mathbb{R}^{n \times n}, +)$ ($n \times n$ invertible matrices, $*$), $(\underbrace{S_n}_{\text{Permutation of } [1, n]}, \circ)$

* Commutative Groups: A Group with the additional property: $a \cdot b = b \cdot a \quad \forall \; a, b \in G$

Rings

A set of elements R with operators such that:

1) $(R, +)$ is a commutative group
2) $(R, *)$ satisfies properties 1-3 of groups (i.e., existence of inverse is not guaranteed)
3) $a * (b + c) = a * b + a * c \quad \forall \; a, b, c \in R$

Ex: $(\mathbb{Z}, +, *)$, $(\mathbb{Q}, +, *)$, $(\mathbb{R}^{n \times n}, +, *)$, $(\mathbb{Q}[x], +, *)$

Commutative Rings: Ring $(R, +, *)$ with additional property: $a * b = b * a \quad \forall \; a, b \in R$

Fields

A set of elements with operations such that:

1) $(F, +, *)$ is a commutative ring
2) $(F - \underbrace{\{0\}}_{\text{identity of addition}}, *)$ is a commutative group

Ex: $(\mathbb{Q}, +, \times)$ or replace $\mathbb{Q}$ with $\mathbb{R}, \mathbb{C}$.

* $F - \{0\} =: F^*$

Finite Groups / Rings / Fields

Finite number of elements.

Ex: $(\mathbb{Z}_n := \{0, 1, \ldots, n-1\} \equiv$ Set of integers modulo n,
$+ \pmod n$, $* \pmod n) \leftarrow$ Finite ring

$(\mathbb{F}_p := \{0, 1, \ldots, p-1\}, + \pmod p, * \pmod p)$

↳ Same as $\mathbb{Z}_{n=p}$ except p is a prime

Finite Field. Consider $0 \neq a \in \mathbb{F}_p$

$\Rightarrow \exists\ k, l$ s.t. $ka + lp = 1 \Rightarrow ka = 1 \pmod p$

$\Rightarrow k \pmod p \in \mathbb{F}_p$ & is inverse of a.

- Closure of arithmetic operations over finite fields guarantees that numbers do not grow arbitrarily large irrespective of the number of operations

Function Fields

- Let $F[x]$ be the set of all polynomials in x with coefficients from field F.
- Then, $(F[x], +, *)$ is a commutative ring where arithmetic is over polynomials.
- Let $F(x)$ be the set of rational functions in x, that is:

$$F(x) = \{ f(x)/g(x) \mid f(x), g(x) \in F[x],\ g(x) \neq 0 \}$$

- Then, $(F(x), +, *)$ is a field:
  - Multiplicative inverse of f/g, $f \neq 0$, is g/f

Prime Extension Fields

- Let $f(x) \in F_p[x]$ be an irreducible polynomial over F_p

→ $f(x)$ cannot be factored as $f_1(x) f_2(x)$ with $f_1, f_2 \in F_p[x]$, both of degree >0.

- Let degree of f be d
- Define F_{p^d} to be the set of all polynomials of degree $<d$ in $F_p[x]$
- Then, $(F_{p^d}, +, *)$ is a field with arithmetic modulo p and $f(x)$:
  - All co-efficients are reduced mod p and all powers of x of degree $\geq d$ are reduced mod $f(x)$
  - Mult. inv. of $g \in F_{p^d}^*$ is $h \in F_{p^d}$ such that $gh + rf = 1 \mod p$

- A finite field has size p^d where p is a prime and $d > 0$

Order: Let $(G, \cdot)$ be a finite commutative group. and $a \in G$. Order of a is the smallest non-zero number k such that $a^k = e$

- Order is well defined for each $a \in G$, G finite
  - Let $A := \{a^i \mid i > 0\} \subseteq G$. G finite $\Rightarrow A$ finite
  - Let a^k be the largest power of a in A Then $a^{k+1} = a^i$ for some $i \leq k \Rightarrow a^{k+1-i} = e$

Lemma: If order of a equals k, then for every l such that $a^l = e$: $k \mid l$

- Let $m = (l, k) \Rightarrow uk + lv \Rightarrow a^m = a^{uk} \cdot a^{lv} = e$
- $\because k$ = order $\therefore k \leq m$
- $\because m \mid k$, $k \leq m \Rightarrow k = m \Rightarrow k \mid l$

Theorem: Let $(G, \cdot)$ be a finite commutative group. Then $\forall\ a \in G$, $a^{|G|} = e$

· Hint: Use $A := \{ab \mid b \in G\}$, then look at $\prod_{i \in A} i$ & $\prod_{i \in G} i$

Corollary: Order of each element divides $|G|$

Cyclic Groups

Let $(G, \cdot)$ be a commutative group. G is cyclic if $\exists\ a \in G$ such that $G = \{a^i \mid i \in \mathbb{Z}\}$

Element a is called generator of the group. If G is finite, then order of a equals $|G|$

· Eg: $(\mathbb{Z}, +)$, $(F_p, +)$ are cyclic groups w/ generator 1. $(\mathbb{Q}, +)$, $(F_{p^d}, +)|_{d>1}$ are not cyclic groups.

Theorem: For a finite field F, $(F^*, *)$ is a cyclic group.

Proof: Let $\prod_i p_i^{r_i} = m = |F^*|$ and $S_i = \{a \mid a \in F^*, \text{ order of } a \text{ divides } p_i^{r_i}\}$ is a group.

· S_i is a cyclic group

· Let $a_i \in S_i$ be an element w/ max. order $p_i^{s_i}$, for some $s_i \leq r_i$

· Order of each element of S_i divides $p_i^{s_i}$

· $\therefore y^{p_i^{s_i}} = 1 \quad \forall\ y \in S_i$

At most $p_i^{s_i}$ roots $\Rightarrow |S_i| \leq p_i^{s_i}$

· But, a has exactly $p_i^{s_i}$ distinct powers, all in S_i, $\therefore a_i$ is generator of S_i

· By Structure Theorem of Finite Fields,

each $b \in F^*$ can be uniquely written as a product of one element of S_i for each i

- $\therefore m = \prod_i p_i^{s_i} \Rightarrow s_i = r_i \quad \forall i$
- $a := \prod_i a_i$
- a is a generator of F^*:
  - Let $a^{m'} = 1 \Rightarrow 1 = \prod_i a_i^{m'} \Rightarrow a_i^{m'} = 1$
    $\therefore p^{r_i} \mid m' \Rightarrow m \mid m' \Rightarrow m' = m$

Theorem: Let F be any finite field. Then polynomial $q(y) \in F[y]$ of degree d has at most d roots in F.

Advanced Encryption Scheme (AES)

- Designed in ~1997, adopted in 2001, originally called Rijndael.
- Developed by Vincent Rijmen and Joan Deamen.

Parameters: Blocksize = 128 bits
Keysize = 128, 192, or 256 bits
Rounds = 10, 12 or 14

Each round has four operations in a sequence: ByteSub, ShiftRow, MixColumn, AddRoundKey.

Viewing a Block

- A block is 128 bits, or 16 bytes long
- It is viewed as an element of $F_{256}^{4\times4}$

ByteSub

$$b_{ij} = \begin{cases} 1/a_{ij}, & \text{if } a_{ij} \neq 0 \\ 0, & \text{otherwise} \end{cases} \quad \left.\right\} \text{Non-linear part}$$

$$c_{ij} = T \cdot b_{ij} + c \quad \left.\right\} \longrightarrow \text{linear part}$$

- T is a fixed 8×8 invertible matrix over F_2 and c is a fixed column vector over F_2
- In F^*_{256}, $1/a_{ij} = a_{ij}^{254}$ $\because$ $a_{ij}^{255} = 1$
- The only non-linear operation. Easily seen to be invertible.

ShiftRow

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \Longrightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{11} & a_{12} & a_{13} & a_{10} \\ a_{22} & a_{23} & a_{20} & a_{21} \\ a_{33} & a_{30} & a_{31} & a_{32} \end{bmatrix}$$

- A left rotation of i columns is applied on i^{th} row
- Each column in new matrix consists of one element from every column of old matrix

MixColumn

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \Longrightarrow \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- A fix matrix is multiplied to block matrix
- Viewing each column as a degree 3 polynomial in $F_{256}[x]$, the operation is same as multiplying column polynomial by fixed polynomial $3x^3 + x^2 + x + 2 \mod x^4 + 1$

AddRoundKey

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \Longrightarrow \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} k_{00} & k_{01} & k_{02} & k_{03} \\ k_{10} & k_{11} & k_{12} & k_{13} \\ k_{20} & k_{21} & k_{22} & k_{23} \\ k_{30} & k_{31} & k_{32} & k_{33} \end{bmatrix}$$

- Round key matrix is added to block matrix
- Addition is bitwise XOR in F_{256}
- Each round key matrix is derived from key using a fixed algorithm.

Rounds: 10, 12 or 14 depending on key size
- One additional ARK performed before first round.
- Last round does not have MC operation

Observations:
- MC only operation that 'mixes' elements,
- Together with SR, in two rounds, all elements mixed up.
- BS is the only non-linear operation — linear part chosen so that at least one operation not in F_{256}.

Decryption
- The rounds and ordering of round operations get reversed.
- ARK remains same, except use of round keys is reversed.
- SR and BS can be exchanged.
- ARK and MC can be exchanged.

$$M \cdot (B + K) = M \cdot B + M \cdot K = (M \cdot B) + (M \cdot K)$$

Some other key added

- ∴ MC not used in last round, MC & ARK of 2nd last round can be exchanged and viewed as operations in 1st encryption round.

AES: 1 round

- Sequence of operations: ARK → BS → SR → ARK
- No mixing ⇒ byte-by-byte encryption
- Can be broken w/ a 256 byte chosen-plaintext attack.

AES: 2 rounds

- ARK → BS → SR → MC → ARK → BS → SR → ARK
- Mixing only on four byte groups.
- Can be broken w/ a $2^{32} \approx 4$ GB chosen plaintext attacks

AES: 3 rounds

- Differential cryptanalysis does not work:

  x, $x + d_0$ — two plaintexts w/ fixed diff

  BS → $\frac{1}{x} + \frac{1}{x+d_0} = \frac{d_0}{x(x+d_0)}$ → Takes some specific value w/ prob. $\leq \frac{2}{256}$

  $d_0 \neq 0$ $\frac{d_0}{x(x+d_0)} = d_1 \Rightarrow$ 2-degree eq^n in x ⇒ At most 2 roots

  After MC, difference propagated to four bytes

  ⇒ After 2 rounds, prob. of diff. being a fixed value is $\leq 2^{-28}$

- Linear Cryptanalysis does not work due to similar reasons.

★ A modified form of differential cryptanalysis, called square attack, does break three round AES.

- In this, we consider a set of 256 plaintext blocks being encrypted simultaneously & trace patterns

AES: 3-rounds

Input block: 4×4 matrix of elements from F_{256}

- Patterns of 256 values from F_{256}.
  - P: all 256 values are distinct
  - C: all 256 values are same
  - Z: sum of 256 values is 0
  - X: everything
- P is also Z; C is also Z

AddRoundKey: Same one byte added (XOR-ed) to each byte.

$P \xrightarrow{ARK} P$ $\quad C \xrightarrow{ARK} C$ $\quad Z \xrightarrow{ARK} Z$ $\quad X \xrightarrow{ARK} X$

ByteSub: Non-linear operation, but one-to-one.

$P \xrightarrow{BS} P$ $\quad C \xrightarrow{BS} C$ $\quad Z \xrightarrow{BS} X$ $\quad X \xrightarrow{BS} X$

ShiftRow: Relocates the byte to some other location, but the bit-pattern of a byte remains the same. $P \xrightarrow{SR} P$ $\quad C \xrightarrow{SR} C$ $\quad Z \xrightarrow{SR} Z$ $\quad X \xrightarrow{SR} X$

MixColumn:

$$\begin{bmatrix} C \\ C \\ C \\ C \end{bmatrix} \rightarrow \begin{bmatrix} C \\ C \\ C \\ C \end{bmatrix}, \quad \begin{bmatrix} P \\ C \\ C \\ C \end{bmatrix} \rightarrow M \begin{bmatrix} \alpha_{i,1} \\ \alpha_{i,2} \\ \alpha_{i,3} \\ \alpha_{i,4} \end{bmatrix} \rightarrow \begin{bmatrix} \sum_k m_{1k}\alpha_{ik} \\ \sum_k m_{2k}\alpha_{ik} \\ \sum_k m_{3k}\alpha_{ik} \\ \sum_k m_{4k}\alpha_{ik} \end{bmatrix} \xrightarrow{\alpha_{jk} \neq 0 \Rightarrow P} \begin{bmatrix} P \\ P \\ P \\ P \end{bmatrix}$$

$$\begin{bmatrix} P \\ P \\ C \\ C \end{bmatrix} \xrightarrow{\text{Sum still remains zero } \because P \subseteq Z,\ C \subseteq Z} \begin{bmatrix} Z \\ Z \\ Z \\ Z \end{bmatrix}, \quad \begin{bmatrix} X \\ ? \\ ? \\ ? \end{bmatrix} \rightarrow \begin{bmatrix} X \\ X \\ X \\ X \end{bmatrix}$$

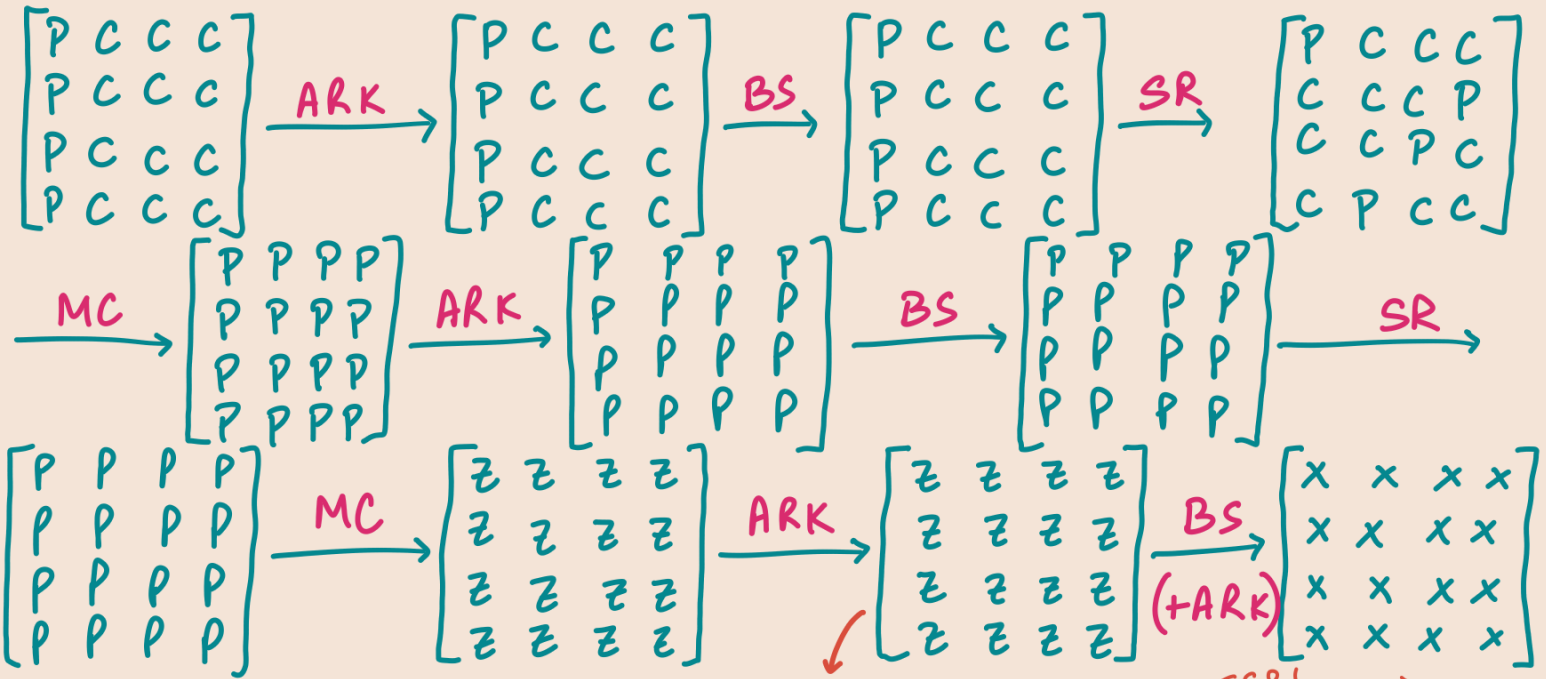
Takes all 256 values w/ i → Does not change w/ i

$(m_{11}\alpha_{i1}) + (m_{12}\alpha_{i2} + m_{13}\alpha_{i3} + m_{14}\alpha_{i4})$

⇒ Takes all 256 vals w/ changing i ⇒ P

- Now, we observe how these patterns of 256

values move with AES operations:



Call the output values $[\beta^i_{j,k}]$ ← Call the values $[\alpha^i_{j,k}]$

$\alpha^i_{j,k}$ —Byte Sub→ —Add Round Key ($k^3_{j,k}$)→ $\beta^i_{j,k}$

- Guess a value of $k^3_{j,k}$, compute $\alpha^i_{j,k}$ $\forall i$ & check if their sum is zero. If the guess is right, then the sum will be zero, other values of $k^3_{j,k}$ are unlikely to give zero sum.
- Obtain a set of candidate $k^3_{j,k}$'s. Repeat with another input & take intersection till only one candidate remains.
- 256 bruteforce attacks help yield one byte of k^3, i.e., $k^3_{j,k}$. To get the complete 128 bits of k^3, only need $256 \times 16 = 4096$ bruteforce attacks. ($16 = 128/8$)

- For AES: 4 rounds, use $\begin{bmatrix} p & c & c & c \\ c & c & c & c \\ c & c & c & c \\ c & c & c & c \end{bmatrix}$ as the inputs.

- This is called Square Attack.
  It works upto 4-rounds. It fails beyond that.
- For 6-round AES, ∃ an attack taking time $\approx 2^{126}$ operations, better than bruteforce attack by a factor of 4.

Key Exchange

[~1977] Diffie Hellman: Uses exponentiation (can be done over an insecure channel) → 1st method

Method: 1) Anubha and Braj agree on a large prime p & a generator g of F_p^*
2) Anubha picks a random number a, $0 < a < p-1$, computes and sends $g^a \pmod p$ to Braj
3) Braj picks a random number b, $0 < b < p-1$, and sends $g^b \pmod p$ to Anubha.
4) Anubha computes $(g^b)^a \pmod p$ ⤡ Equal
   Braj computes $(g^a)^b \pmod p$ ⤢ This is the key

What does Ela know? What does Ela want?
p, g, $g^a \pmod p$, $g^b \pmod p$, $g^{ab} \pmod p$

- Can be used to find a
  Keep multiplying with g^{-1} till we get 1.
  ↳ Requires a multiplications
- Want $a \sim 2^{128} \Rightarrow p \sim 2^{128}$
- ∴ Simply multiplying with g^{-1} does not work
  Need to find some other method.

Discrete Log Problem

Given a finite group G, and $g, h \in G$ find a such that $h = g^a$.

→ Believed to be hard for group F_p^* when p is large.

→ Fastest known algorithm time $2^{O(\log^{1/3} p \log\log^{2/3} p)}$

- For a p 1024 bits long, the best algorithm for discrete logarithm problem takes $2^{4 \cdot (1024)^{1/3} (10)^{2/3}} \approx 2^{180}$ operations

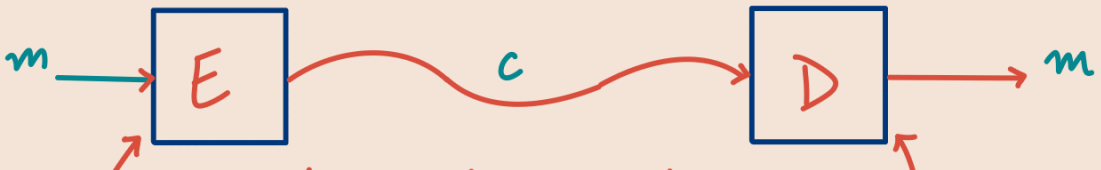
Efforts required by Anubha & Braj:

1) Find a large prime p (1024 bit) [~1024 expected picks]
2) Find a generator g of F_p^*
   Choose (P) such that $p-1 = 2q$, q prime
   → Sophie-Germaine primes.

Conjecture: In $[1, n]$, there are $\sim \frac{n}{\log^2 n}$ Sophie-Germaine primes.

- For Sophie-Germaine prime $p = 2q+1$, $F_p^* = G_2 \times G_q$ → Either $-a$ or $+a$, is a generator of order $2q$ for $a \neq \pm 1$

Public-Key Encryption



k_E → Needs to be secret

k_D → Must be secret

in private key encryption, can be public in public key encryption

- Possibility discussed by Diffie-Hellmann in 1976
- Public-key encryption solves key-exchange problem
- Chosen plaintext attack can be carried out easily.

* Two distinct map requirements: One for the keys, one for texts. Finding k_D from k_E should be hard.

One-way maps:

Discrete-Log: $e \longrightarrow a^e \pmod{n}$

Factoring: $(p, q) \longrightarrow pq$

Computing the map ($\longrightarrow$) is easy but, inverting the map ($\longleftarrow$) is difficult.

RSA

* Rivest-Shamir-Adleman, 1977
- First public-key encryption algorithm
- Uses the previous two one-way maps.

Key Generation

1.) Choose two large primes p & q

2.) Compute $n := pq$

3.) Pick a random e, $1 < e < (p-1)(q-1)$, such that $\gcd(e, (p-1)(q-1)) = 1$

4.) Compute d such that $de \equiv 1 \pmod{(p-1)(q-1)}$

Public Key $(k_E) = (n, e)$

Private Key $(k_D) = d$

Encryption

1.) Break plaintext into blocks such that each block is a number between 1 & n

2.) Let m be a number corresponding to a block. Compute ciphertext block

$$c = m^e \pmod n$$

Decryption

1.) Let c be the number corresponding to a ciphertext block. Compute

$$m = c^d \pmod n$$

Analysis (E + D)

$$c^d \bmod n = (m^e)^d = m^{ed}$$

$$= m^{1 + k(p-1)(q-1)} \quad (= |\mathbb{Z}_n^*|)$$

$$= m * (m^{(p-1)(q-1)})^k$$

$$= m \pmod n$$

provided $m \in \mathbb{Z}_n^*$ (Set of nos. $< n$ that are co-prime to n)

$$|\mathbb{Z}_n^*| = (n-1) - \overbrace{(p-1)}^{\text{Factors of } q} - \overbrace{(q-1)}^{\text{Factors of } p} = n - p - q + 1 = (p-1)(q-1)$$

- So, encryption + decryption works if $m \in \mathbb{Z}_n^*$, but what if $m \notin \mathbb{Z}_n^*$?

Then, $\gcd(m, n)$ is either p or q.

Then, $(p-1)(q-1)$ can be computed easily.

Then, d can be computed easily.

→ Numbers not in $\mathbb{Z}_n^* = p - 1 + q - 1$

$$\Rightarrow \mathbb{P}[m \notin \mathbb{Z}_n^*] = \frac{p-1+q-1}{n-1} \approx \frac{p+q}{n} = \frac{1}{p} + \frac{1}{q}$$

Very small if p & q are large

Security of k_D

Encryption Key : (n, e)
Decryption Key : d

n can be factored $\Leftrightarrow$ $(p-1)(q-1)$ can be computed

$\Rightarrow$ is trivial. $\Leftarrow$ $n=pq$ known, can find $p+q$, then finding p, q same as solving a quadratic equation.

$(p-1)(q-1)$ can be computed $\Leftrightarrow$ d can be computed

Factoring n

1.) School Method
2.) Quadratic Sieve method $2^{O(\log n \log\log n)^{1/2}\sqrt{n}}$
3.) Number Field Sieve method $2^{O(\log^{1/3} n \log\log^{2/3} n)}$

Taking $n \geq 1024$ bit is recommended as a safeguard.

Time Complexity

- Key Generation
  - p, q : Done by randomly choosing a number of size l & testing if it is prime.
  - d : Computing from e can be done via extended GCD.

- Encryption
  - $m^e \pmod n$: $O(\log e) = O(\log n)$ multiplications $= O(\log^2 \log\log n)$

→ Encrypting a plaintext, even though efficient, takes non-trivial time.

RSA is used primarily for exchanging small amount of information, like AES keys, and AES used for bulk encryption.

- Decryption
  → Same as Encryption

★ If d is not known, computing m from c has no method in general.

- This makes RSA encryption very secure in general. However, one needs to avoid several special cases when choosing the key, where the system becomes insecure.

Cryptanalysis: Prime Factors

$n = pq$

If p is small, then only $O(p)$ operations required: $1 \longrightarrow p$

If $p-q$ is small, then only $O(p-q)$ operations required: $-(p-q)/2 \leftarrow \sqrt{n} \longrightarrow +(p-q)/2$

★ In summary, both p & q should be large but not close to each other.

Smooth Primes

Composite number m is k-smooth if all prime factors of m are $\leq k$. Prime number p is k-smooth if $p-1$ is k-smooth.

- Let $n = pq$ & suppose p is k-smooth but q is not.

Let $T = (k!)^{\log n}$ — T is k-smooth → $(p-1) \mid T$ → Every prime divisor of $p-1$ is $\leq k$ w/ index at most $\log p \leq \log n$; $(q-1) \nmid T$ → some factor of $q-1$ is $> k$

Lemma: $\gcd(a^T - 1, n) = p$ for most a's

Proof: Suppose $a \in \mathbb{Z}_n^*$. Then, $a^T = a^{r(p-1)} = 1 \bmod p$

- But, $a^T \pmod q \neq 1$ whenever a is a generator of F_q^* $\because |F_q^*| = q-1 \nmid T$
- $\because$ Most a's are generators of F_q^*, it follows that $a^T - 1$ is divisible by p but not q.

- Hence, both p & q should not be k-smooth for small k.
- Best strategy is to choose p & q such that both $(p-1)/2$ and $(q-1)/2$ are primes.
  - choose Sophie Germain primes. → prob. is $\approx \frac{1}{(\ln n)^2}$

Integer Lattice

Given a set of LI vectors $v_1 \dots v_D \in \mathbb{R}^D$, integer lattice generated by them is:

$$\mathcal{L} = \left\{ \sum_{i=1}^{D} a_i v_i \mid a_i \in \mathbb{Z} \right\}$$

- $\mathcal{L}$ is a vector space consisting of integer linear combinations of vectors $v_1 \dots v_D$.

These vectors form a basis of $\mathcal{L}$

Ex: $\mathcal{L}$ of $(1\ 0)$ $(1\ 1)$ is $\mathbb{Z}^2$

Volume of a Lattice: Denoted by $v(\mathcal{L})$, is

defined as $|\det(V)|$ where $V = [v_1, v_2, \ldots, v_D]$

Lemma: $v(\mathcal{L})$ is independent of the basis

Proof: Let $u_1 \ldots u_D$ be another basis.

Then $U = [u_1 \ldots u_D] = AV$ and $V = BU$ with $A, B \in \mathbb{Z}^{D \times D}$. $\because \{u_i\}_{i=1}^{n}$ are LI

$\therefore \det(U) \neq 0 \Rightarrow U^{-1}$ exists, do,

$U = AV = ABU \Rightarrow I = AB \Rightarrow \det(A)\det(B) = 1$

$\Rightarrow \det(A) = \det(B) = \pm 1$

$\Rightarrow |\det(U)| = |\det(V)|$

Shortest Vector of lattice $\mathcal{L}$ is the minimum length non-zero vector in $\mathcal{L}$.

Length of shortest vector is denoted as $\lambda_1(\mathcal{L})$.

- Finding shortest vector of a lattice is known to be a **hard-to-solve** problem.
- Even finding a vector of length $\leq \sqrt{2}\,\lambda_1(\mathcal{L})$ is known to be hard [Ajtai-Micciancio]
- However, it is possible to find a vector of length $\leq 2^{(D-1)/2}\lambda_1(\mathcal{L})$.

Minkowski's Theorem

For any lattice $\mathcal{L}$, $\lambda_1(\mathcal{L}) \leq \sqrt{D}\, v(\mathcal{L})^{1/D}$

Lenstra-Lenstra-Lovasz (L^3) Algorithm

Given a lattice $\mathcal{L}$, it computes a vector v, in time polynomial in D, such that,

$|v| \leq 2^{(D-1)/2}\lambda_1(\mathcal{L})$ $\left(\leq 2^{(D-1)/2}\sqrt{D}\, v(\mathcal{L})^{1/D}\right)$

Small e

- In order to save time during encryption, e may be chosen small.
  - $e = 1$ ✗ Trivial
  - $e = 2$ ✗ $\gcd(e, \phi(n) = (p-1)(q-1)) \stackrel{?}{=} 1$
  - $e = 3$ ✓ Smallest case possible
- But, for small e, the system can be broken without d.

- Set $e = 3$.
- Suppose that m is a 128 bit key of AES, and n is 1024 bits long.
  Then $c = m^e = m^3 \pmod n = m^3$ over integers and can be computed easily.
  ∴ Assume that $m = k + x$ where $x < 2^l$ and k is known. Then,

$$c = m^3 = (k+x)^3 = x^3 + 3kx^2 + 3k^2x + k^3 \pmod n$$

gives a degree 3 polynomial whose root $(< 2^l)$ mod n needs to be computed.

$$P(x) = x^3 + 3kx^2 + 3k^2x + k^3 - c$$

- Define lattice $\mathcal{L} \subseteq \mathbb{R}^6$:

$$\mathcal{L} = \begin{bmatrix} 1 & 3k & 3k^2 & k^3-c & 0 & 0 \\ 0 & 1 & 3k & 3k^2 & k^3-c & 0 \\ 0 & 0 & 1 & 3k & 3k^2 & k^3-c \\ 0 & 0 & 0 & n & 0 & 0 \\ 0 & 0 & 0 & 0 & n & 0 \\ 0 & 0 & 0 & 0 & 0 & n \end{bmatrix} \begin{matrix} \to x^2P(x) \\ \to xP(x) \\ \to P(x) \\ \to nx^2 \\ \to nx \\ \to n \end{matrix}$$

6 basic vectors

construct $\delta(x)$

$$v(\mathcal{L}) = n^3$$

- Use L^3-algorithm to find a vector in $\mathcal{L}$ of length,

$$\leq 2^{(6-1)/2} \sqrt{6}\, n^{3/6} = \sqrt{192n}$$

- Construct $\delta(x)$ as such:

$r_1 \cdot x^2 P(x) + r_2 \cdot x P(x) + r_3 \cdot P(x) + r_4 \cdot n x^2 + r_5 n x + r_6 n$

$\hookrightarrow Q(m) \equiv 0 \pmod n$

Also, using L^3-algorithm, can find a vector r s.t. $|r| \leq \sqrt{192n} \Rightarrow |r_i| \leq \sqrt{192n}$.

Also note that $\because r \in L$, it represents a polynomial $Q(x)$ s.t. each co-efficient is $\leq \sqrt{192n}$, and so, for any $x < 2^l$, we have,

$$|Q(x)| \leq \sqrt{192n}\,(x^5 + x^4 + x^3 + x^2 + x^1 + 1)$$
$$\leq \sqrt{192n} \cdot 2^{6l} \qquad (\because x < 2^l)$$
$$\leq n \quad \text{for } l < \frac{1}{12}(\log n - \log 192)$$
$$\Rightarrow x < n^{1/12}$$

$\forall\ x < 2^l$ So, $Q(x)$ can be treated as a normal polynomial and the problem reduces to finding one of its root.

$\therefore$ If unknown part is $< n^{1/12}$, the complete message can be recovered.

But, this bound of $n^{1/12}$ can be improved to $n^{1/2}$ with the choice of more sophisticated polynomials.

- So, a small e cannot be chosen without compromising security.

Small d

- Let $d = O(n^\epsilon)$, $\epsilon > 0$. Typically, $e = \Omega(n)$ in such a case

$$de = 1 \mod \phi(n) = (p-1)(q-1)$$
(de unknown)
$$= 1 + r(p-1)(q-1)$$
$$= 1 + r(n+1) - r(p+q)$$ ($p+q$ unknown but $O(n^{1/2})$)

$\rightarrow$ $r = \dfrac{de-1}{(p-1)(q-1)} = O(n^{1+\epsilon-1}) = O(n^\epsilon)$

$\Rightarrow r = O(n^\epsilon)$

$\rightarrow$ $1 + \underbrace{r(n+1)}_{r = O(n^\epsilon) \Rightarrow r \leq K} - \underbrace{r(p+q)}_{= O(n^{1/2+\epsilon}) \Rightarrow \leq L} = 0 \pmod e$

Let $R(x,y) = Ly - (n+1)Kx - 1$. Goal is to find (r,s) s.t. $R(r/K, s/L) = 0$ (mod e), $r/K \le 1$, $s/L \le 1$

We use $K^i x^i P(x,y)$ for additional vectors: there will be t such vectors for $0 \le i < t$ with terms $x^i y$ & x^j for $0 \le i < t$, $0 \le j \le t$. ∴ There are a total of $2t+1$ terms.

We can get the same number of vectors by taking additional polynomials $K^i x^i e$, $0 \le i \le t$

All the additional polynomials are 0 mode

These polynomials give rise to a lattice $\mathcal{L}$ in $\mathbb{Z}^{2t+1}$.

Volume of $\mathcal{L}$ equals

$$\Pi_{i=0}^{t-1}(K^i L)\Pi_{i=0}^{t}(K^i e) = L^t K^{t^2} e^{t+1}$$

Then, the vector computed by L^3-algorithm has length at most

$$l = 2^t \sqrt{2t+1}\; L^{t/(2t+1)} K^{t^2/(2t+1)} e^{t+1/2t+1}$$

Using $L = c_L n^{1/2+\epsilon}$, $K = n^\epsilon$, $e < n$, we get

$$l < 2^t \sqrt{(2t+1)c_L}\; n^{t/2(2t+1) + \epsilon t/(2t+1) + \epsilon t^2/(2t+1) + \frac{t+1}{2t+1}}$$

$$= 2^t \sqrt{(2t+1)c_L}\; n^{(2\epsilon t^2 + 2\epsilon t + 3t + 2)/2(2t+1)}$$

If $(2t+1)l < e$, then we have the desired property: polynomial defined by the L^3 vector is zero over integers for $x = r/K$, $y = s/L$

This requires $n^{(\cdot)} < n \Rightarrow \epsilon < 1/2(t+1)$

But, a bivariate polynomial has infinitely many roots → Identify another polynomial w/ same property using 2nd smallest vector computed by L^3 algorithm: this

vector is linearly independent from the 1st one & has length bounded by $2^{(D-1)/2}\sqrt{D}\,v(L)^{1/(D-1)}$
$\rightsquigarrow \epsilon < 1/2(t+1) - 1/t(t+1)$

- We get two bivariate polynomials R_1 & R_2 with $(r/K, s/L)$ as common root.
- GCD likely to give a univariate polynomial in x w/ root r/K.
- Can find r & s. Can find d.
- ∴ If $d < n^{1/20}$, it can be computed from e and n.
- With choice of more involved polynomials, one can show the same result for $d < n^{0.292}$
- Hence, neither e nor d should be chosen small.

1.) p & q close but not too close
2.) p & q both Sophie-Germain primes
3.) $d, e = \lambda(n)$

El-Gamal Cryptosystem

Key Generation

1.) Let $(G, \cdot)$ be a commutative finite group, $|G| = n$.
2.) Let $g \in G$. Let order$(g) = k$
3.) Choose a number e, $1 < e < k$

Public Key : (g, g^e, k)
Private Key : e

Encryption

Let plaintext $m \in G$

1.) Choose a random r, $1 < r < k$

2.) Output $(g^r, g^{er} \cdot m)$

Decryption

Let ciphertext (h_1, h_2)

1.) Output h_2 / h_1^e $(= g^{er} \cdot m / (g^r)^e = m)$

Security

Eda knows $(g, g^e, k, g^r, m \cdot g^{er})$

- She needs to compute g^{er}
- She essentially needs to solve discrete-log problem in the group G.

Groups for which discrete log is hard to solve provide secure public-key encryption.

Eg: $G = (F_p^*, *)$, $G = (F_p, +)$

→ Good → Bad

→ $2^{O(\log^{1/3} p \log\log^{2/3} p)}$

Elliptic Curves

$$y^2 = x^3 + Ax + B, \quad 4A^3 + 27B^2 \neq 0$$

Consider $y^2 = x^3 - x$

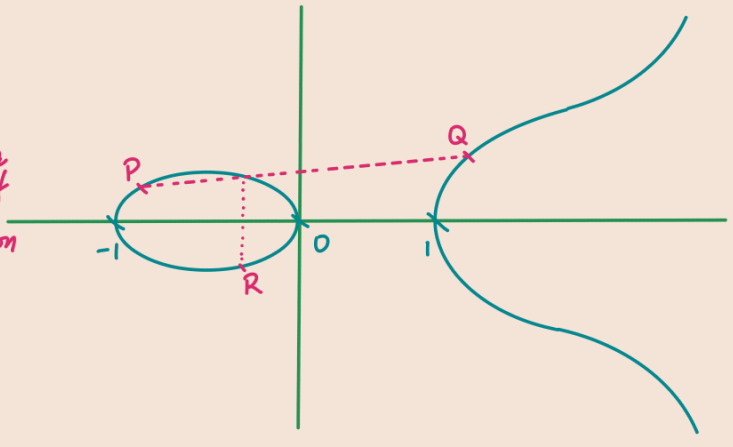
$P + Q := R$ } Connect $P - Q$ using a line, the reflection of the third point of intersection

$P + Q := \infty$ } P & Q reflection of each other

$P + P := S$ } Reflection of pt of tangent at P

$P + \infty := P$

$\infty + \infty := \infty$



$E(\mathbb{R})$:= Set of points on the curve over real numbers $\cup \{\infty\}$

$E(\mathbb{Q})$:= Set of rational points on curve $\cup \{\infty\}$

$E(F_p)$:= Set of points satisfying

$y^2 = x^3 + Ax^2 + B \pmod p$,

$4A^3 + 27B^2 \neq 0 \pmod p$, $\cup \{\infty\}$

* $E(\mathbb{Q})$ was used in proving Fermat's Last Theorem

$E(\mathbb{C})$ is donut-shaped.

Elliptic Curves over F_p

Theorem (Hasse): $p+1-2\sqrt{p} \leq |E(F_p)| \leq p+1+2\sqrt{p}$

Theorem: $E(F_p)$ is either cyclic, or a product of two cyclic groups.

· Until now, there is no algorithm that can solve discrete log problem in $E(F_p)$ in time $2^{O(\log |E(F_p)|)}$.

* El Gamal system based on elliptic curves is called Elliptic Curve Cryptography (ECC).

$\Rightarrow$ 160 bit ECC ~ 2048 bit RSA

Quantum Computers

· Store information in a sequence of qbits.

· Each qbit is a superposition of values 0 & 1, until measured.

· Unitary transformations can be applied on

these without measurement.

Theorem (Grover): Given a list of n unsorted numbers, a quantum computer can search for a specific number in the list in time $\sqrt{n}$

Theorem (Shore): Integer factoring & Discrete log, both can be solved efficiently by quantum computers.

Learning with Errors (LWE)

Consider $\mathbb{R}^n$, n-dimensional real space. and suppose one is given $u_1 \cdot s, u_2 \cdot s, \dots, u_n \cdot s$ for $u_1, u_2, \dots, u_n \in \mathbb{R}^n$ & $s \in \mathbb{Z}^n$.

P1: Compute s from $\{(u_i, u_i \cdot s)\}_{i=1}^{i=n}$
Easy if u_i's are LI

P2: Compute s from $\{(u_i, u_i \cdot s + e_i) : 1 \le i \le n\}$
where e_i is a random "small" error.
$\equiv$ Find closest vector in the lattice $[u_1 \dots u_n]$ to v
$\longrightarrow$ Closest Vector Problem (CVP)

Theorem: Finding closest lattice vector is NP-hard.

Kyber Public Key Encryption

- Based on LWE
- During decryption, if the result is $> q/2$ then $m=1$ otherwise if $< q/2$, $m=0$

Digital Signatures

Properties required: (1) Hard to forge
(2) Easy to verify
(3) Easy to sign

RSA Signature:

Public Key: (n, e)
Private Key: d

Suppose document m is to be signed, assume $m < n$

Signing: $s = m^d \pmod n$

Verification: Check if $m = s^e \pmod n$

Signed document: (m, s)

▷ What if message is larger?

$$m \stackrel{?}{=} m_1 m_2 \dots m_k, \quad m_i < n$$

$$s_i = m_i^d \bmod n$$

Signature: $(m, s_1 s_2 \dots s_k)$ ✗

If $m_1, m_2 \dots$ removed, ↙ then s does not change much.

Secure Hash Functions

Function $h: \{0,1\}^* \longrightarrow \{0,1\}^l$ is a secure hash function if:

1) Computing $h(x)$ is easy
2) Given x, finding $y \neq x$ with $h(y) = h(y)$ is hard
3) Given $h(x)$, finding x is hard
4) Finding $x \neq y$ such that $h(x) = h(y)$

is hard.

RSA Signature

Signing: $s = h(m)^d \pmod n$

Choose l such that $2^l < n$

Signed document: (m, s)

Verification: Check if $h(m) = s^e \pmod n$

☆ Other Public Encryption methods like ECC [Elliptic curve cryptography] can also be used for Digital Signatures.

Identifying Secure Hash Functions

In 1991, Ron Rivest proposed MD5

☆ Subsequently, SHA-1 & SHA-2 were proposed.

- MD5 was broken in 2005; So was SHA-1 within a few years.
- In 2006, NIST started a contest to identify a new secure hashing algorithm
- In 2012, SHA-3 came into being

SHA-3

Key parameters: $r, b, d \in \mathbb{Z}$, $b > r$ and an auxiliary parameter $c := b - r$

Key component: Function f

$f: \{0,1\}^b \to \{0,1\}^b$

The Algorithm:

1.) Break message m into blocks of r

bits, $m = m_1 m_2 \dots m_k$

2.) Let $s_0 := 0^b$

3.) Let $s_i := f(s_{i-1} \oplus m_i 0^c)$, $1 \le i \le k$

4.) Let $s_{k+i+1} := f(s_{k+i})$, $0 \le i \le \lceil d/r \rceil$

5.) Let z_i be first r bits of s_{k+i}

6.) Output $z_0 z_1 \dots$ truncated to d bits

Function f

- Typically, $b = 1600$
- View input s_i as 5×5 matrix of 64-bit strings.
- Let $a[i][j][k]$ denote k^{th} bit of string at the $(i,j)^{th}$ cell of the matrix
- Do following operations 24 times:

θ: $a[i][j][k] \leftarrow a[i][j][k] \oplus \sum_{u=0}^{u=4} \{ a[u][j-1][k] \oplus a[u][j+1][k] \}$

ρ: Bitwise rotate each $a[i][j]$ by a different triangular number $0, 1, 3, 6, 10, \dots$

π: $a[3i+2j][i] \leftarrow a[i][j]$

χ: $a[i][j][k] \leftarrow a[i][j][k] \oplus \{ \neg a[i][j+1][k] \wedge a[i][j+2][k] \}$

ι (iota): $a[0][0] \leftarrow a[0][0] \oplus$ Round constant

- θ: Bringing effect of other columns
- ρ: Change "depth" of each entry
- π: Shuffle
- χ: Only non-linear operation

2: Break the symmetry

Typical parameter values

1) d = 224, r = 1152, c = 448
2) d = 256, r = 1088, c = 512
3) d = 384, r = 832, c = 768
4) d = 512, r = 576, c = 1024

Public Key Infrastructure (PKI)

Certification Authorities

- Root CAs: Symantec, DigiCert, Comodo, ...
  - National / Intermediate CAs: NIC, Armed Forces, ... (Indian Examples)
    - CAs: Banks, Financial Institutions, ...

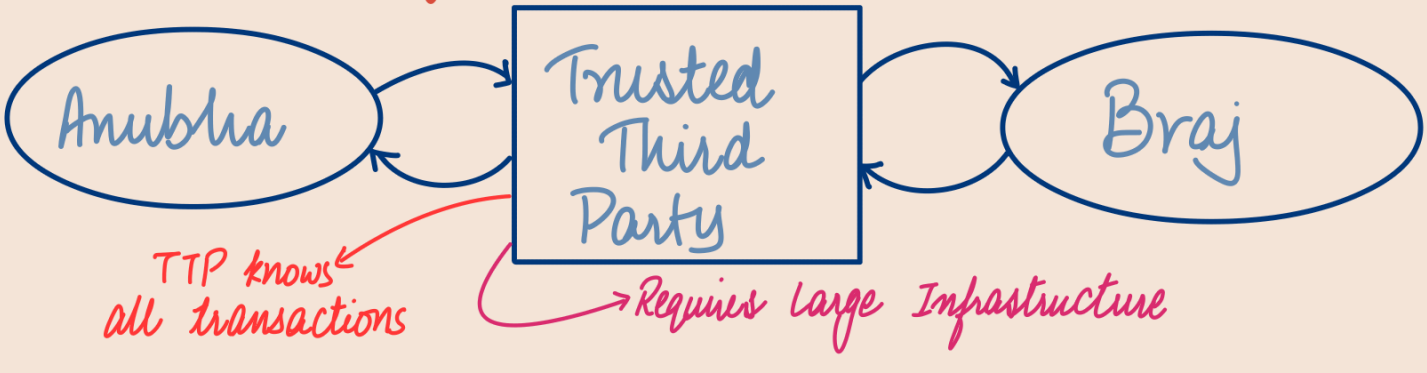
Digital Money

Requirements:

1) Easily transferrable
2) Difficult to forge
3) Easy to verify
4) Double spending not possible

Use-Case for CryptoCurrencies?

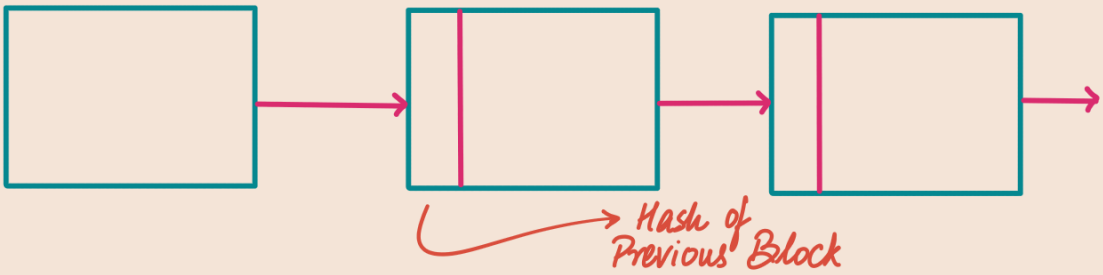
Typical digital money transactions:



Bitcoin

- Published by Satoshi Nakomoto in 2008.
- Based on BlockChains

BlockChain

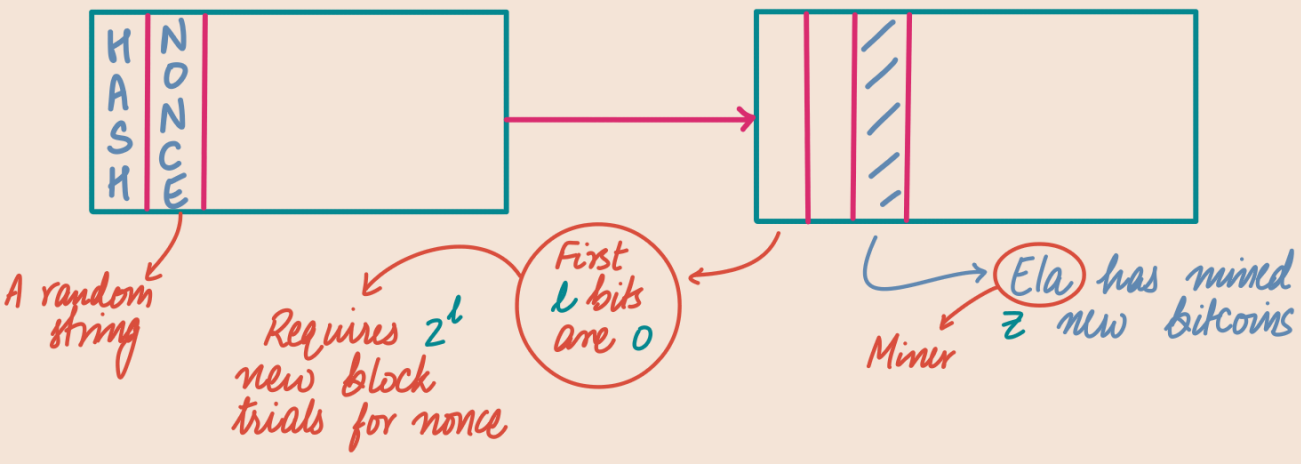


- Difficult to change order of the blocks.
- Difficult to change contents of the blocks while keeping the hash same.
- A transaction is a string of the form:
  ∴ Anubha pays x bitcoins to Braj. ∴
  These transactions are added to the blockchain.

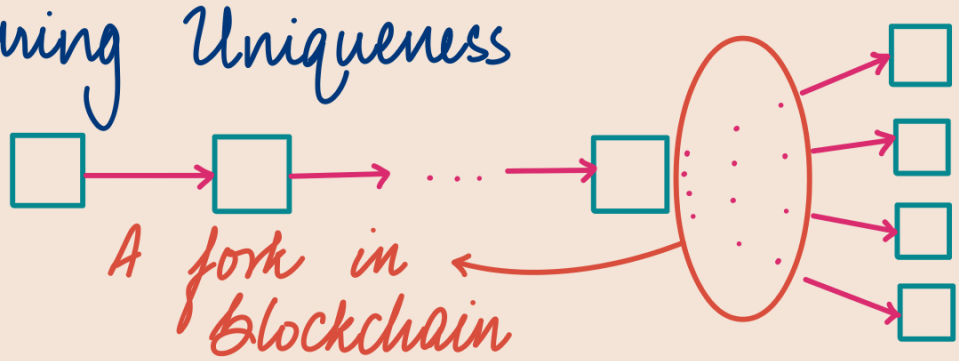
But, how is a unique blockchain maintained if it is managed by peers and not a centralized server / agency?

Adding a Block

- Anyone can add a block by collecting new transactions on network.
- But before adding a new block, a puzzle needs to be solved.



Ensuring Uniqueness



A fork in blockchain

Resolving Forks

- When one branch becomes 3+ blocks longer than others, remaining branches are removed.

Zero-Knowledge Proofs

Anubha knows s. She wants to convince Braj that she knows s without revealing anything about s.

A protocol that achieves this is called Zero-Knowledge-Proof or Protocol

Theorem: All problems in NP have Zero-Knowledge Proofs.

Proof: NP has NP-complete problems as a subclass & 3-coloring can be shown to have a ZKP using cryptographically secure hash functions.