Week - 9

- Universal Hashing
- Cryptographic Hash Function
- Secure Hash Algorithm (SHA)
- Digital Signature Standard (DSS)
- More on Key Exchange protocol.

Universal Hashing:

- A weakness of hashing:

For any hash function h, a set of keys exists that can cause the average access time of a hash table to skyrocket.

An adversary can pick all keys from $\{k \in U : h(k) = i\}$ for some slot i.

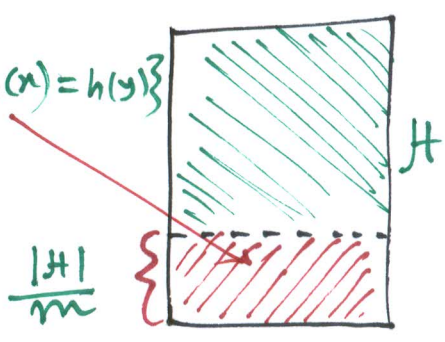
- Idea: Choose the hash function at random, independent of the keys.

  Even if the adversary can see your code, he or she cannot find a bad set of keys, since he or she doesn't know exactly which hash function will be chosen.

- Definition: Let U be a universe of keys, and let $\mathcal{H}$ be a finite collection of hash functions, each mapping U to $\{0, 1, \ldots, m-1\}$. We say $\mathcal{H}$ is universal if for all $x, y \in U$, where $x \neq y$, we have

$$|\{h \in \mathcal{H} : h(x) = h(y)\}| = \frac{|\mathcal{H}|}{m}$$

That is, the chance of a collision between x and y is $1/m$ if we choose h randomly from $\mathcal{H}$.



Universality is good :

Theorem: Let h be a hash function chosen (uniformly) at random from a universal set $\mathcal{H}$ of hash functions. Suppose h is used to hash n arbitrary keys into the m slots of a table T.

Then, for a given key x, we have

$$E[\#\text{collisions with } x] < n/m.$$

Proof: Let C_x be the random variable denoting the total number of collisions of keys in T with x, and let

$$C_{xy} = \begin{cases} 1, & \text{if } h(x) = h(y) \\ 0, & \text{otherwise} \end{cases}$$

Then $E[C_{xy}] = \frac{1}{m}$ and

$$C_x = \sum_{y \in T - \{x\}} C_{xy} \Rightarrow$$

$$E[C_x] = E\left[\sum_{y \in T - \{x\}} C_{xy}\right]$$

$$= \sum_{y \in T - \{x\}} E[C_{xy}]$$

$$= \sum_{y \in T - \{x\}} (1/m) = \frac{n-1}{m} < n/m .$$

■

- **Constructing a set of universal hash functions:**

Let m be prime. ~~Let~~ Decompose key k into $(r+1)$ digits, each with value in the set $\{0, 1, \ldots, m-1\}$. That is, let $k = \langle k_0, k_1, \ldots, k_r \rangle$, where $0 \leq k_i < m$.

Randomized strategy:

Pick $a = \langle a_0, a_1, \ldots, a_r \rangle$ where each a_i is chosen randomly from $\{0, 1, \ldots, m-1\}$.

Define $h_a(k) = \sum_{i=0}^{r} a_i k_i \bmod m$.

(Dot product modulo m)

See, $|H| = m^{r+1}$.

- **Theorem:** The set $H = \{h_a\}$ is universal.

Proof: Suppose that $x = \langle x_0, x_1, \cdots, x_r \rangle$ and $y = \langle y_0, y_1, \cdots, y_r \rangle$ be distinct keys. Thus, they differ in at least one digit position, wlog position 0.

For how many $h_a \in H$ do x and y collide?

we have $h_a(x) = h_a(y) \Rightarrow$

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \pmod m$$

$$\Rightarrow \sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \pmod m$$

$$\Rightarrow a_0 (x_0 - y_0) + \sum_{i=1}^{r} a_i (x_i - y_i) \equiv 0 \pmod m$$

$$\Rightarrow a_0 (x_0 - y_0) \equiv - \sum_{i=1}^{r} a_i (x_i - y_i) \pmod m$$

Since $x_0 \neq y_0$, an inverse $(x_0 - y_0)^{-1}$ must exist (provided $\gcd(x_0 - y_0, m) = 1$) and we have

$$a_0 \equiv \left(-\sum_{i=1}^{r} a_i (x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \pmod{m} \quad \text{— (1)}$$

Thus for any choices of $a_1, a_2, \ldots, a_r$ exactly one choice of a_0 causes x and y to collide.

There are m choices for each of $a_1, \ldots, a_r$, but once these are chosen, exactly one choice for a_0 causes x and y to collide, namely (1).

Thus, the number of h_a's that cause x and y to collide is

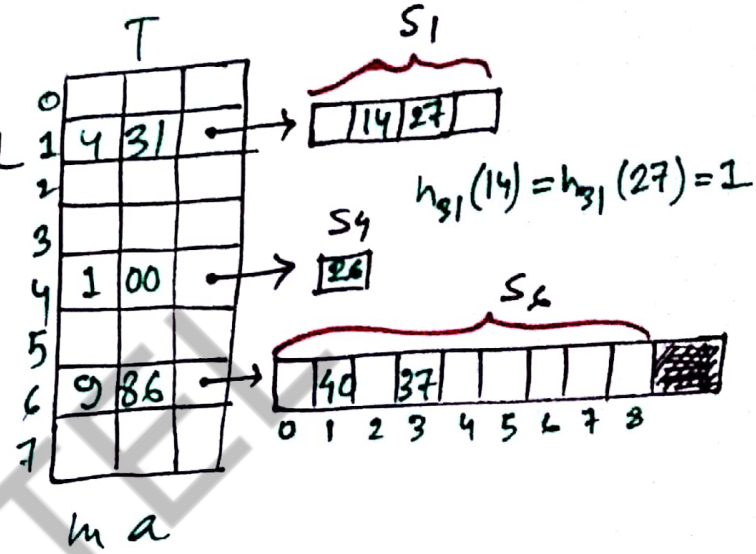
$$m^r \cdot 1 = m^r = \frac{|H|}{m}.$$

Hence H is a family of universal hash functions. ▨

- **Perfect hashing :**

Given a set of n keys, construct a static hash table of size $m = O(n)$ such that SEARCH takes $\Theta(1)$ time in the worst case.

Idea: Two-level scheme with universal hashing at both levels.



- **Collisions at level 2 :**

- **Theorem:** Let $\mathcal{H}$ be a class of universal hash functions for a table of size $m = n^2$. Then if we use a random $h \in \mathcal{H}$ to hash n keys into the table, the expected number of collisions is at most $\frac{1}{2}$.

proof: By the definition of universality, the probability that 2 given keys in the table collide under h is

$1/m = \frac{1}{n^2}$. Since there are $\binom{n}{2}$ pairs of keys that can possibly collide, the expected number of collisions is

$$\binom{n}{2} \cdot \frac{1}{n^2} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} < \frac{1}{2}.$$

■

- **Analysis of Storage :**

For the level-1 hash table T, choose $m = n$, and let n be random variable for the number of keys that hash to slot i in T. By using n_i^2 slots for the level-2 hash table S_i, the expected total storage required for the two-level scheme is therefore

$$E\left[\sum_{i=1}^{m} \Theta(n_i^2)\right] = \Theta(n),$$

since the analysis is identical to the analysis of bucket sort.

Cryptographic Hash function :

- A hash value is generated by a function H of the form:

$$h = H(M)$$

where M is a variable-length message and H(M) is the fixed length hash value (also referred to as a message digest or hash code).

- Figure 1 and 2 shows the basic uses of the hash function.

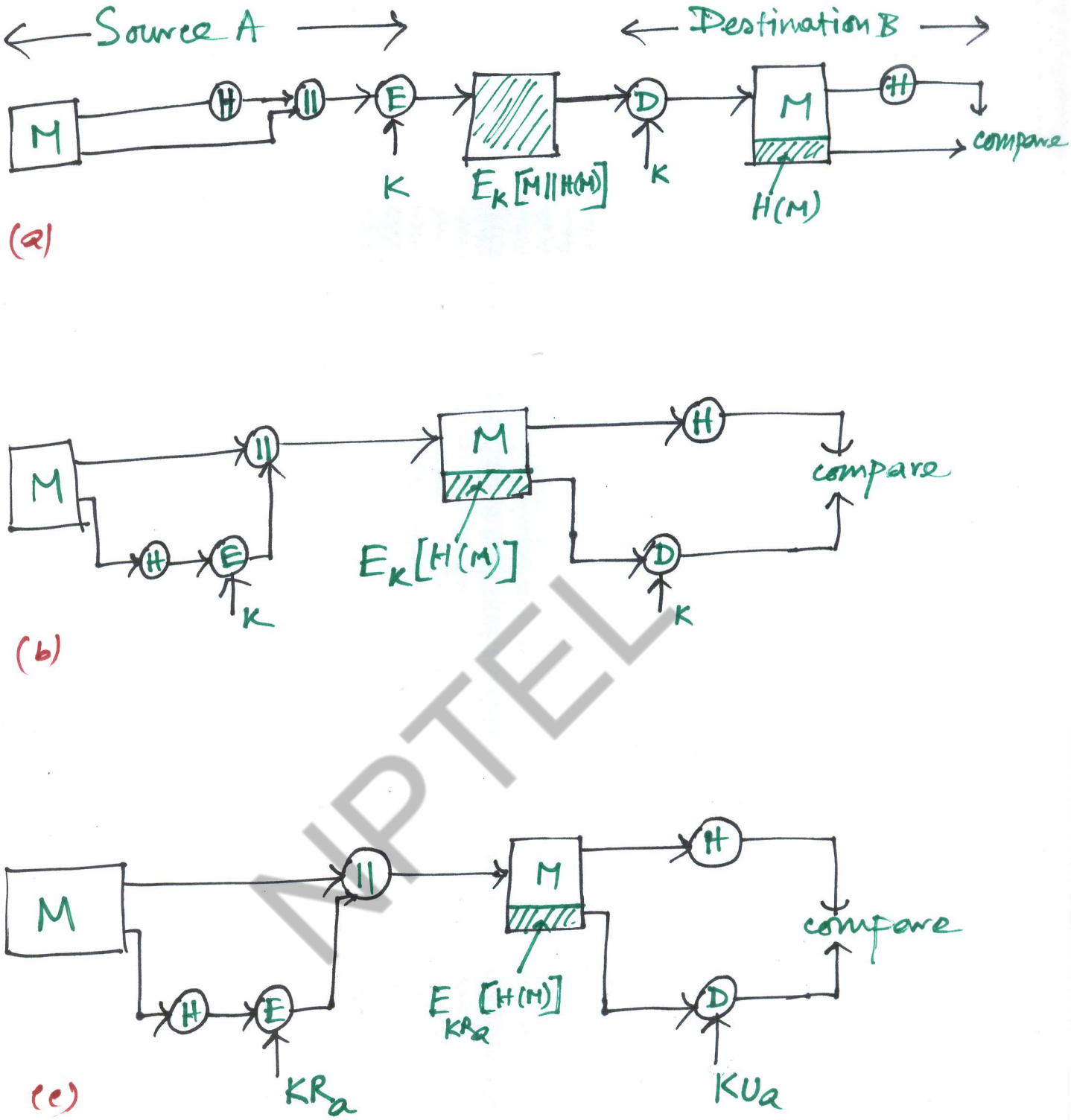
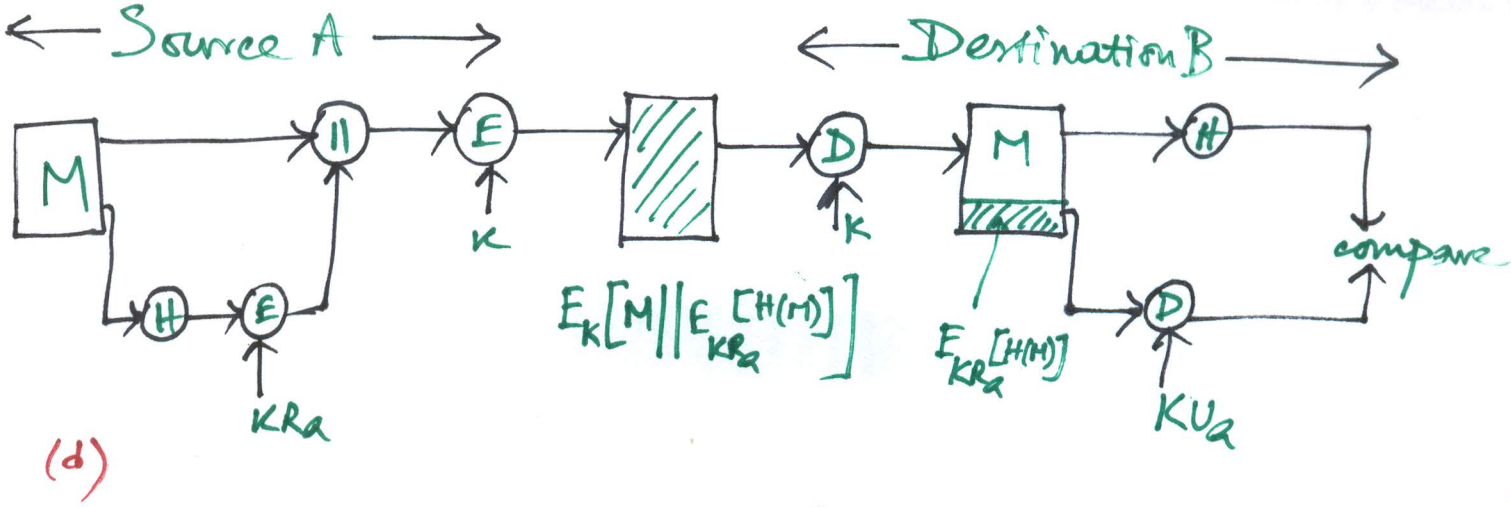
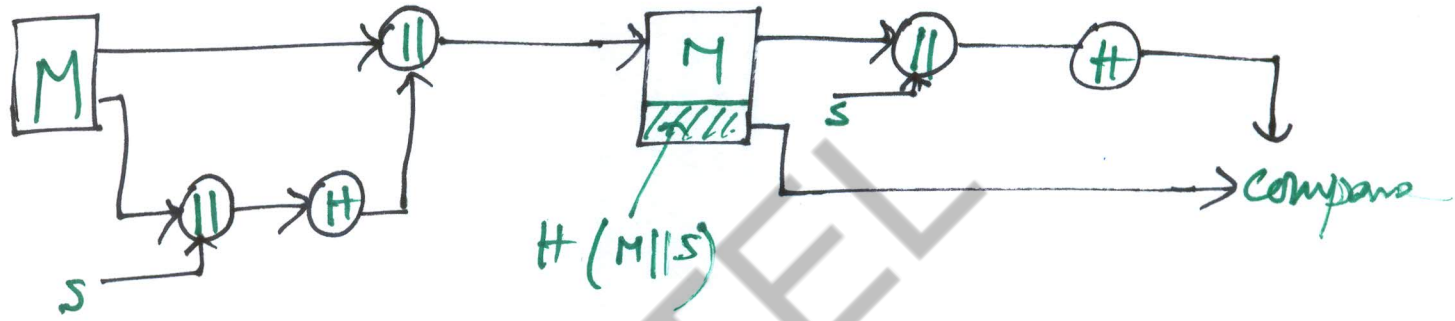


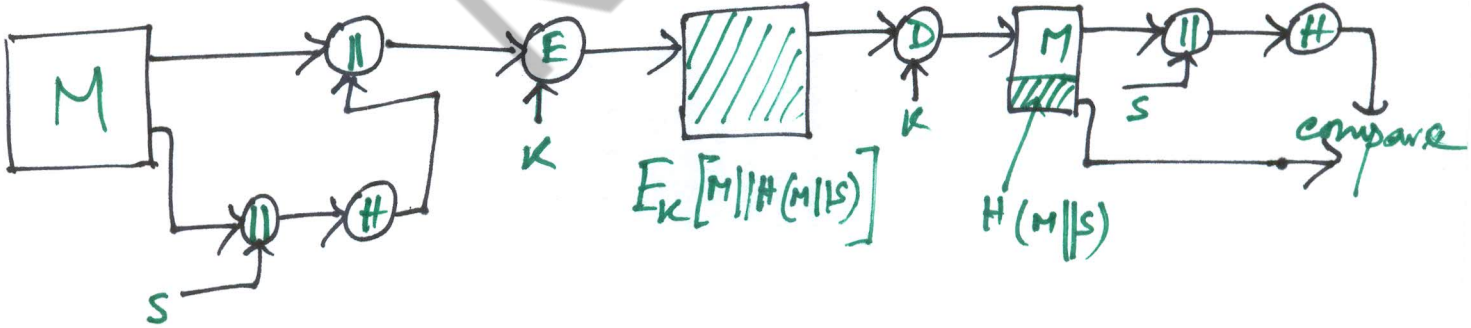
Figure 1: Basic uses of the hash function



(d)



(e)

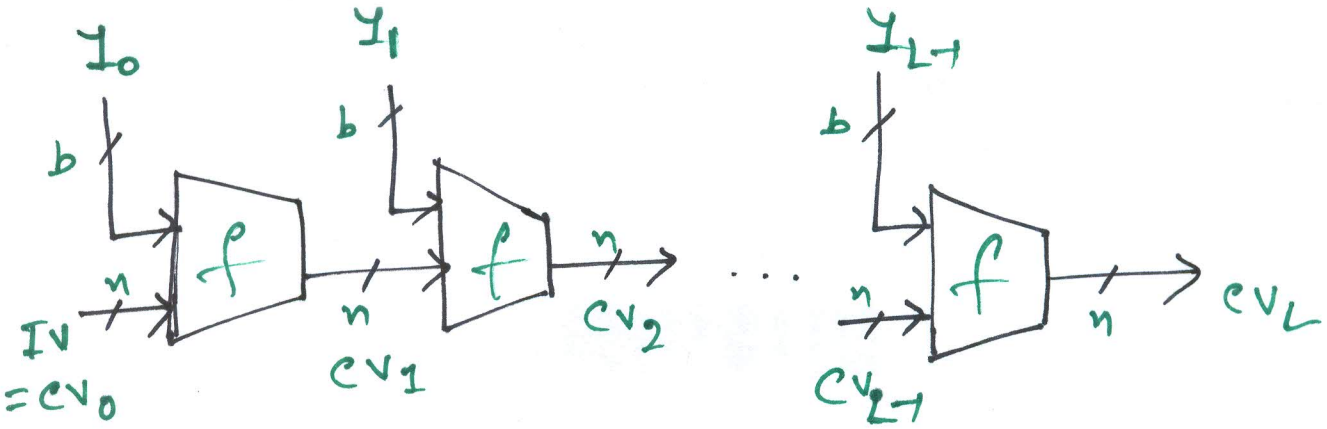


(f)

Figure 2: Basic uses of the hash function (cont.)

- A hash function H must have the following properties:

1. H can be applied to a block of data of any size.
2. H produces a fixed length output
3. $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given code h, it is computationally infeasible to find x such that $H(x) = h$.
5. For any given block x, it is computationally infeasible to find $y \neq x$ with $H(x) = H(y)$ (weak collision property).
6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$ (strong collision property).



IV = Initian value

CV = chaining variable

Y_i = i^{th} input block

f = compression algorithm

L = number of input blocks

n = length of hash code

b = length of input block

Figure 3 : General Structure of Secure Hash Code

The Secure Hash Algorithm :

- The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-I.

- SHA is based on the MD4 algorithm which is a message digest algorithm that was developed by Ron Rivest at MIT.

- MD4 was later replaced by the popular MD5 algorithm also by Ron Rivest.

- Both MD4 and MD5 produce a 128-bit message digest whereas SHA-1 produces 160-bit.

- SHA-1 takes as input a message with a maximum length of less than 2^{64} bits and produces output

a 160-bit message digest.

- The input is processed in 512-bit blocks.

- Figure 4 represents MD5 as the hash function which exactly same as SHA-1 with the exception that the message length is limited in size (its is not for MD5) and the hash value (and intermediate values CV_i) are 160 bits and not 128 as shown (which is the case for MD5)

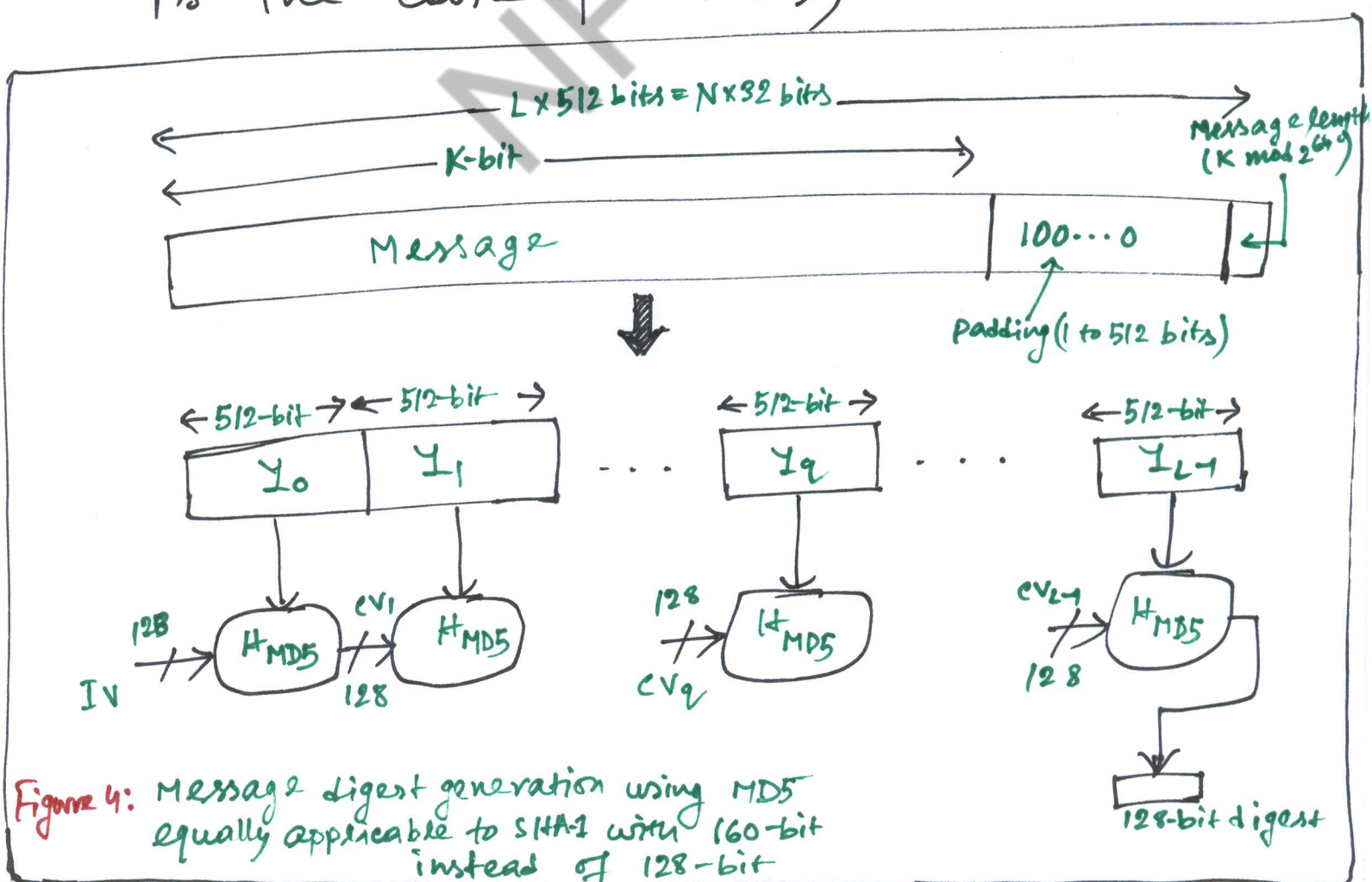


Figure 4: Message digest generation using MD5 equally applicable to SHA-1 with 160-bit instead of 128-bit

5-Steps :

1) Append padding bits :

- The message is padded so that its length is congruent to 448 modulo 512 (length $\equiv$ 448 (mod 512)).
- That is the length of the padded message is 64 bits less than an integer multiple of 512-bits.
- Padding is always added, even if the message is already of the desired size.
- Thus the number of padding bits is in the range of 1 to 512 bits.
- The padding consists of a single 1-bit followed by the necessary number of 0-bits.

2) Append length :

- A block of 64-bit is appended to the message.
- This block is treated as an unsigned 64-bit integer (most significant byte first) and contains the length of the original message (before padding).

3) Initialize MD buffer :

- A 160-bit buffer is used to hold intermediate values and final results of the hash function represented as 5, 32-bit registers (A, B, C, D, E) initialized as follws:

A = 67452301

B = EFCDAB89

C = 98BADCFE

D = 10325476

E = C3D2E1F0

4) Process message in 512 bit (16 word) blocks :

- The heart of the algorithm is a module which consits of four rounds of processing of 20 steps each (see figure 5)
- Each round has similar structure but uses a different primitive logical function (f_1, f_2, f_3 and f_4).
- Each round takes as input the current 512-bit block being processed (Y_q) and the 160-bit buffer value ABCDE and updates the contents of the buffer.
- Each round also makes use of an additive constant K_t where $0 \leq t \leq 79$ indicates one of the 80 steps across four rounds.
- In fact only four distinct constants are used (one for $0 \leq t \leq 19$, $20 \leq t \leq 39$, $40 \leq t \leq 59$ and $60 \leq t \leq 79$).

- The output of the fourth round is added (modulo 2^{32}) to the input to the first round (CV_q) to produce CV_{q+1}.

5) Output :

- After all L 512-bit blocks have been processed the output from the L^{th} stage is the 160-bit digest.

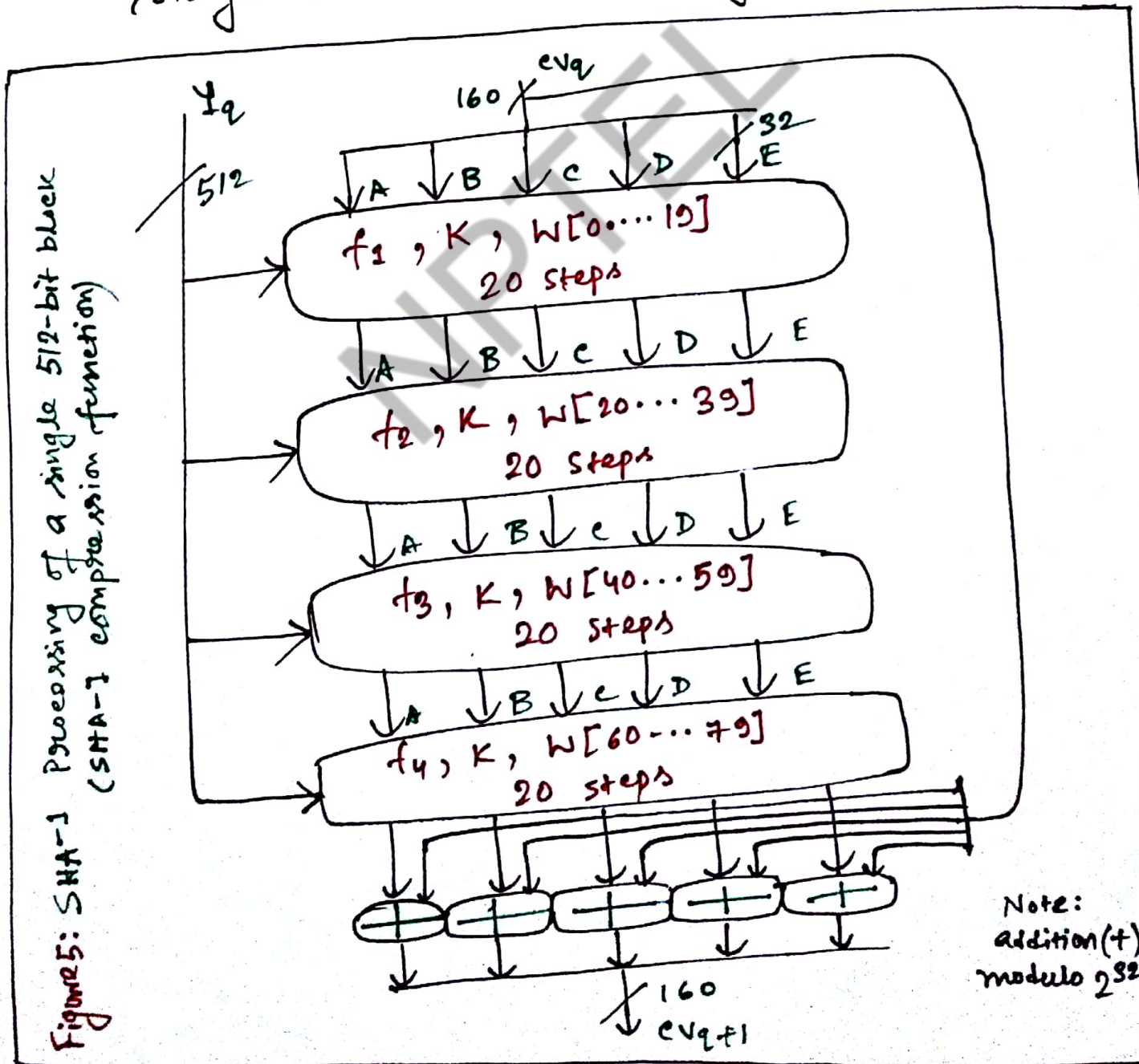


Figure 5: SHA-1 Processing of a single 512-bit block (SHA-1 compression function)

- We can summarise the behavior SHA-1 as follows:

$$CV_0 = IV$$
$$CV_{q+1} = SUM_{32}(CV_q, ABCDE_q)$$
$$MD = CV_L$$

IV = initial value of the ABCDE buffer defined in step3

$ABCDE_q$ = the output of the last round of processing of the q^{th} message block

L = the number of blocks in the message (including padding and length fields)

SUM_{32} = Addition modulo 2^{32} performed separately on each word of the pair of inputs.

MD = final message digest value.

Birthday attack :

- The birthday paradox can be stated as follows : what is the minimum value of k such that the probability is greater than 0.5 that at least two people in a group of k people have the same birthday?

- If there is 100 people, i.e, $k=100$, then the probability is .9999997, i.e, you are almost guaranteed that there will be a duplicate.

- Given a random variable that is an integer with uniform distribution between 1 and n and a selection of k instances ($k \leq n$) of the random variable, what is the probability $P(n,k)$, that there is at least one duplicate ?

- It turns out the value is

$$P(n,k) = 1 - \frac{n!}{(n-k)!\, n^k}$$

$$= 1 - \left[\left(1-\frac{1}{n}\right) \times \left(1-\frac{2}{n}\right) \times \cdots \times \left(1-\frac{k-1}{n}\right)\right]$$

$$> 1 - \left[e^{-1/n} \cdot e^{-2/n} \cdots e^{-(k-1)/n}\right]$$

[Using $(1-x) \leq e^{-x}$]

$$> 1 - e^{-\left(\frac{1}{n}+\frac{2}{n}+\cdots+\frac{k-1}{n}\right)}$$

$$> 1 - e^{-(k \times (k-1))/2n}$$

Now $P(n,k) > 0.5 \Rightarrow$ we set

$$0.5 = 1 - e^{-(k \times (k-1))/2n}$$

$$\Rightarrow \ln 2 = \frac{(k-1) \times k}{2n} \quad \text{—①}$$

For large values of k we can replace $k \times (k-1)$ by k^2, so ① gives

$$k = \sqrt{2 (\ln 2) n} = 1.18\sqrt{n} \approx \sqrt{n}$$

For $n = 365$ we see $k = 23$.

- Now assume that length of the digest is m, then there are 2^m possible message digests in some hash code.

- If we apply k random messages to our hash code what must the value of k be so that there is the probability of 0.5 that at least one duplicate (i.e. $H(x) = H(y)$ will occur for some inputs x, y)?

- Using the above discussion about birthday paradox we have

$$k = \sqrt{2^m} = 2^{m/2}$$

- The idea of birthday attack as follows:
  - The source A is prepared to sign a message by appending

the appropriate m-bit hash code and encrypting that hash code with A's private key.

. The opponent generates $2^{m/2}$ variations on the message, all of which convey essentially the same meaning. The opponent prepares an equal number of messages, all of which are variations of the fraudulent message to be substituted for the real one.

. Two set of messages are compared to find a pair of messages that produce the same hash code.

. The opponent offers the valid variation to A for signature. This signature can then be attached to the fraudulent variation for transmission to the intended recipient.

- Because the two variations have the same hash code, they will produce the same signature; the opponent is assured of success even though the encryption key is not known.

- If we use a 64-bit hash code then the level of effort required is only on the order of $2^{m/2} = 2^{64/2} = 2^{32}$ which is clearly not sufficient to withstand today's computational systems.

Digital Signature Standard (DSS):

- NIST has published FIPS 186 known as digital signature Standard (DSS).

Global Public key components

p → prime number where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64, i.e., bit length of between 512 and 1024 bits in increaments of 64 bits.

q → prime divisor of (p-1), where $2^{159} < q < 2^{160}$ i.e., bit length of 160 bits.

$g = h^{(p-1)/q} \bmod p$

where h is any integer with $1 < h < (p-1)$ such that $h^{(p-1)/q} \bmod p > 1$.

User's private key

x → random or pseudorandom integer with $0 < x < q$

User's Public key

$$y = g^x \bmod p$$

User's Per-Message Secret Number

k = random or pseudorandom integer with $0 < k < q$

Signing

$$r = (g^k \bmod p) \bmod q$$

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

Signature = (r, s)

Verifying

$$w = s^{-1} \bmod q$$

$$u_1 = H(M)\, w \bmod q$$

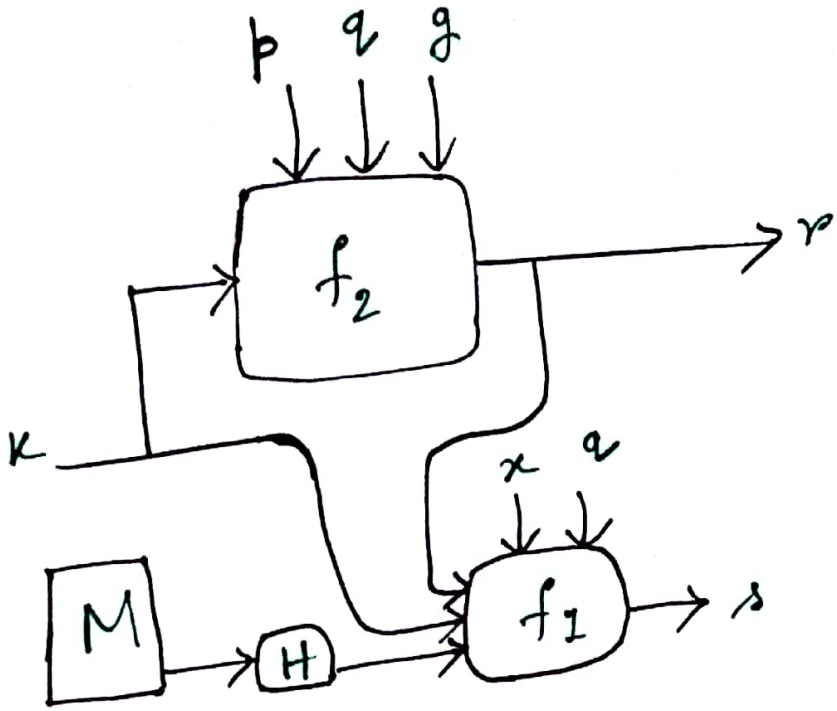
$$u_2 = r\, w \bmod q$$

$$v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$$

TEST: $v = r$

M = message to be signed

$H(M)$ = hash of M using SHA-1



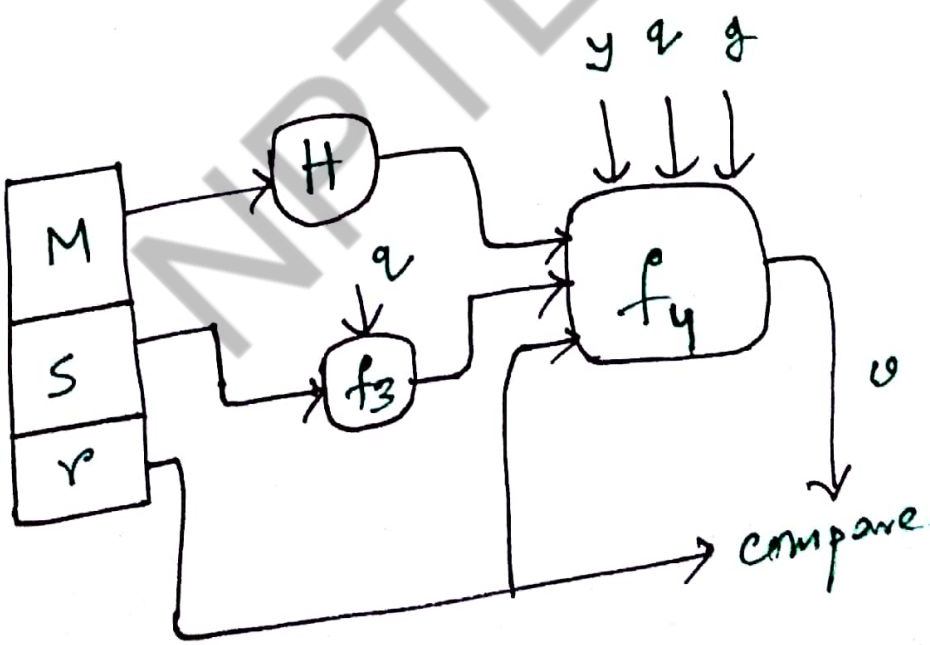
$s = f_1(H(M), K, x, r, q)$

$= (K^{-1}(H(M) + xr)) \bmod q$

$r = f_2(K, p, q, g)$

$= (g^K \bmod p) \bmod q$

(a) Signing



(b) Verifying

$w = f_3(s, q)$

$= s^{-1} \bmod q$

$v = f_4(y, q, g, H(M), w, r)$

$= (g^{(H(M)w) \bmod q} \cdot y^{rw \bmod q} \pmod p) \bmod q$

Example: $q = 101$, $p = 1248563$

$p - 1 \bmod q = 0$

$g = 2^{(p-1)/q} \bmod p = 588634$

$g^q \bmod p = 1$, $y = g^x \bmod p = 702644$

private key: $x = 23$

$m = 53$, $k = 42$

$r = (g^k \bmod p) \bmod q = 52$

$s = k^{-1}(m + xr) \bmod q = 61$

$u_1 = m s^{-1} \pmod q = 82$

$u_2 = r s^{-1} \pmod q = 29$

$v = (g^{u_1} y^{u_2} \bmod p) \bmod q$

$= 52$

(Here we take $H(M) = M$)

Hence $v = r$

3-parties key exchange scheme:

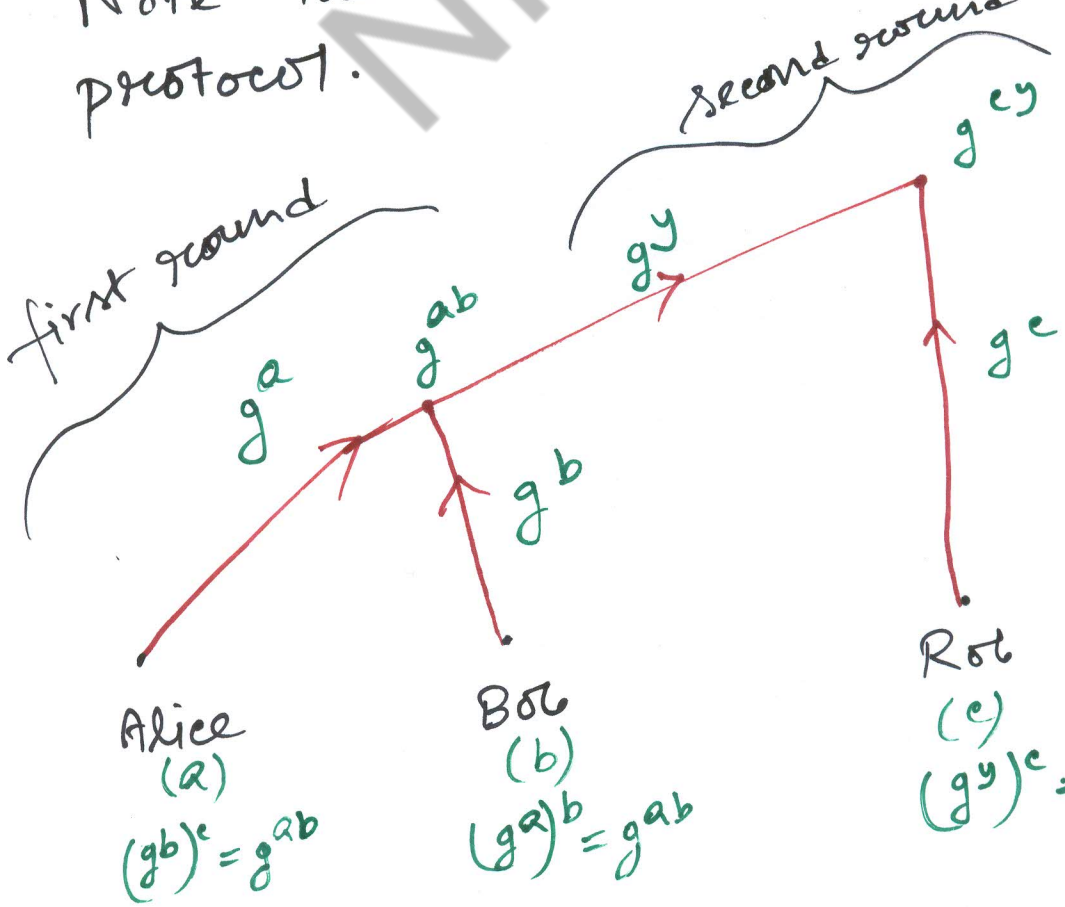
- Let Alice, Bob and Rob want to compute a secret (common) key over a public channel.
- Let G_1 be a group of prime order, say p, and let $H: G_1 \to \{0,1,\ldots, p-1\}$ be a hash function, $G_1 = \langle g \rangle$.
- Let secret keys of Alice, Bob and Rob are a, b, c respectively where $a, b, c \in \mathbb{Z}_p^*$.
- In first round Alice computes g^a and ~~Rob~~ Bob computes g^b and therefore the common key shared between Alice and Bob is g^{ab}.
- Now both Alice and Bob apply $H(g^{ab}) = y \in \mathbb{Z}_p$ and send g^y over public channel.

- Again Rob can compute g^c and send it as public key.
- Now ultimately, Alice and Bob both have g^y, y, g^c; Rob has c, g^c, g^y.

Therefore the common key K computed by both Alice and Bob is $(g^c)^y = g^{cy} = K$

Also, Rob computes the common key as $(g^y)^c = g^{yc} = K$

- Note that this is a two round protocol.



- Both Alice & Bob compute $H(g^{ab}) = y$
$K = (g^c)^y = g^{cy}$

Bilinear Pairing:

- Let G_1 be a additive group (Elliptic curve group) which is cyclic, $(G_1, +)$
- P be a primitive element of G_1.
- Let G_2 be a multiplicative group, $(G_2, \cdot)$.
- cryptographic bilinear pairing $\hat{e} : G_1 \times G_1 \to G_2$

1. Bilinearity: for all $R, S, T \in G_1$,

$$\hat{e}(S+R, T) = \hat{e}(S,T) \cdot \hat{e}(R,T)$$

$$\hat{e}(S, R+T) = \hat{e}(S,R) \cdot \hat{e}(S,T)$$

In other words, $\hat{e}(aS, bT) = \hat{e}(S,T)^{ab}$ for all $a, b \in \mathbb{Z}_n^*$.

2. Non-degeneracy: $\hat{e}(P,P) \neq 1$ where 1 is the identity element of G_2.

3. computability: $\hat{e}$ can be efficiently computed.

4. Symmetry: $\hat{e}(S,T) = \hat{e}(T,S)$

- Bilinear pairing can be constructed from

Weil, Tate pairing.

Joux key exchange scheme:

- Let G_1 be an additive group with generator P and $|G_1| = p$, p = prime.
- Let Alice, Bob and Rob want to compute a common key K.
- Let a, b, c are secret keys of Alice, Bob, Rob respectively.
- Alice, Bob, Rob sends aP, bP, cP as a public key respectively.
- Then the common key is computed as

$$\text{Alice}: K = e(bP, cP)^a = \left(e(P,P)^{bc}\right)^a = e(P,P)^{abc}$$

$$\text{Bob}: K = e(aP, cP)^b = \left(e(P,P)^{ac}\right)^b = e(P,P)^{abc}$$

$$\text{Rob}: K = e(bP, aP) = \left(e(P,P)^{ab}\right)^c = e(P,P)^{abc}$$

- This is one round 3-party protocol to compute a common key.