* Identity-Based Signature (IBS) :-

→ setup → System parameters, sk, vk (or) pk ← verification key

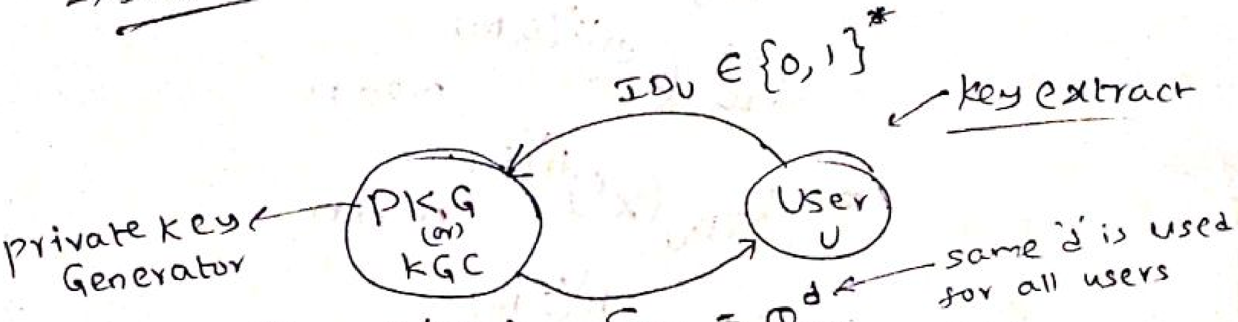
→ keyExtract → sk

→ sign → (sk, m) → σ → signature on 'm'
m ↓ message

→ verify → (vk, m, σ) → 1/0

* Shamir's IBS (Crypto 1984):-

→ setup → uses RSA setup.



$Q_{ID_U} = H(ID_U)$ $\quad S_{ID_U} = Q_{ID_U}^{d}$ ← RSA signature

Setup: ↙ run by PKG

p, q large primes

$n = pq$ (modulus)

$\phi(n) = (p-1)(q-1)$

choose $e \geq 3$ s.t. $\gcd(e, \phi(n)) = 1$

find 'd' $= e^{-1} \bmod \phi(n)$.

hash function,

$H: \{0,1\}^* \to Z_n$, $H': Z_n \times \{0,1\}^* \to Z_n$

→ publishes n, e, H ← params

→ $\underbrace{p, q, \phi(n), d}_{m,sk \text{ (master secret key)}}$ kept secret to PKG

→ **Sign** $(M, S_{ID_U}, \text{params}) \rightarrow (s, t)$

Choose $x \in_R Z_n$

$$s = x^e \bmod n$$

$$t = S_{ID_U} \cdot x^{H'(s,M)} \bmod n$$

→ **Verify** $(M, (s, t), ID_U, \text{Params})$

$$t^e = S_{ID_U}^e \cdot x^{H'(s,M)e} \bmod n$$

$$= Q_{ID_U}^{ed} (x^e)^{H'(s,M)} \bmod n$$

$$= Q_{ID_U} (x^e)^{H'(s,M)} \bmod n$$

$$= Q_{ID_U} \cdot s^{H'(s,M)} \bmod n.$$

check if $t^e = Q_{ID_U} \cdot s^{H'(s,M)} \bmod n.$

✱ **Security** :-

→ A forger can generate x, s, $H'(s,M)$

→ Generating the correct t is equivalent to knowing S_{ID_U}.

→ Getting S_{ID_U} from Q_{ID_U} is the RSA problem.

✱ Sakai-Ohgishi-Kasahara (SOK) IBS (2000)

→ uses : bilinear pairing

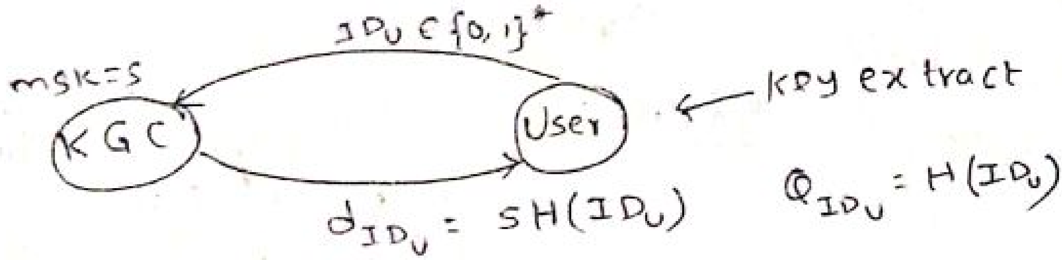
→ **Setup** → Bilinear setup

params = $\langle G_1, G_2, q, P, e, P_{Pub} = sP, H \rangle$

$G_1 = \langle P \rangle$ $\quad H : \{0,1\}^* \longrightarrow G_1$

$e : G_1^2 \longrightarrow G_2$ $\quad |G_1| = |G_2| = q$, so large that DLP in both G_1, G_2 hard

$G_1 \rightarrow$ additive $\quad G_2 \rightarrow$ multiplicative.



$Q_{ID_U} = H(ID_U)$

→ Sign $(M, d_{ID_U}, \text{params}) \longrightarrow (U, V)$

Choose $r \in Z_q^*$

Set $U = rP$

$V = d_{ID_U} + rH(Q_{ID_U}, M, U)$

→ Verification $(M, (U,V), ID_U, \text{params}) \rightarrow 1/0$

$$e(V,P) = e(d_{ID_U} + rH(Q_{ID_U}, M, U), P)$$

$$= e(d_{ID_U}, P) \cdot e(rH(Q_{ID_U}, M, U), P)$$

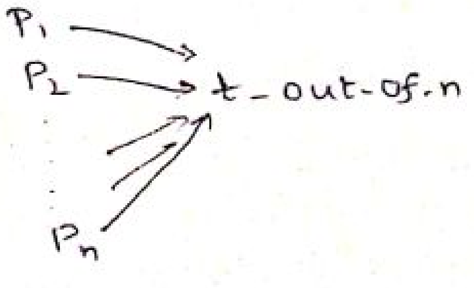
$$= e(sH(ID_U), P) \; e(rH(Q_{ID_U}, M, U), P)$$

$$= e(H(ID_U), sP) \cdot e(H(Q_{ID_U}, M, U), rP)$$

$$= e(H(ID_U), P_{Pub}) \cdot e(H(Q_{ID_U}, M, U), U)$$

* Shamir's Threshold secret sharing

K=s



Dealer → chooses a polynomial of degree $t-1$

s.

~~Let $f(x) = s +$~~

Let $f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{t-1} x^{t-1}$

$a_0, a_1, \dots, a_{t+1} \in F_q$

$a_0 = K = s.$

$\Rightarrow \boxed{f(0) = s}$

not necessarily i, it can be $(id_1, f(id_1))$

$P_1 \cdot (1, f(1))$

$P_2 \cdot (2, f(2))$

$\vdots \qquad \vdots$

$P_n \cdot (n, f(n))$

If we have parties $P_{i_1}, P_{i_2}, \dots, P_{i_t}$

$\Rightarrow (i_1, f(i_1)), (i_2, f(i_2)), \dots \dots (i_t, f(i_t))$

$\Rightarrow$ we have 't' equations & 't' unknowns, $a_0, a_1, \dots a_{t-1}$

$\Rightarrow$ we can solve to find out $a_0 = s$.

*Security

→ t parties will not get any information about s.

→ Sign $(M, S_{ID_U}, \text{params}) \rightarrow (s, t)$

choose $x \in_R Z_n$

$s = x^e \bmod n$

$t = S_{ID_U} \cdot x^{H'(s,M)} \bmod n$

→ Verify $(M, (s, t), ID_U, \text{params})$

$$t^e = S_{ID_U}^e \cdot x^{H'(S,M)e} \bmod n$$

$$= Q_{ID_U}^{ed} (x^e)^{H'(S,M)} \bmod n$$

$$= Q_{ID_U} (x^e)^{H'(S,M)} \bmod n$$

$$= Q_{ID_U} \cdot s^{H'(S,M)} \bmod n.$$

check if $t^e = Q_{ID_U} \cdot s^{H'(s,M)} \bmod n$.

* security :-

→ A forger can generate $x, s, H'(s,M)$

→ Generating the correct t is equivalent to knowing S_{ID_U}.

→ Getting S_{ID_U} from Q_{ID_U} is the RSA problem.

* Sakai-Ohgishi-Kasahara (SOK) IBS (2000)

→ uses bilinear pairing

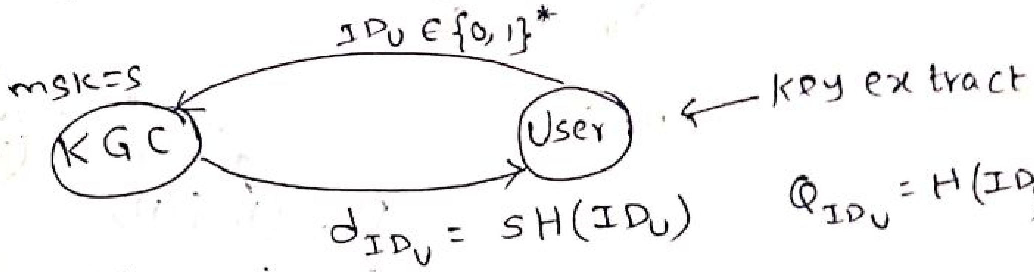
→ setup → Bilinear setup

params = $\langle G_1, G_2, q, P, e, P_{pub} = sP, H \rangle$

$G_1 = \langle P \rangle$ $\quad$ $H : \{0,1\}^* \rightarrow G_1$

$e : G_1^2 \rightarrow G_2$ $\quad$ $|G_1| = |G_2| = q$, so large that DLP in both G_1, G_2 hard

$G_1 \rightarrow$ additive $\quad$ $G_2 \rightarrow$ multiplicative.



$Q_{ID_U} = H(ID_U)$

→ **Sign** $(M, d_{ID_U}, \text{params}) \longrightarrow (U, V)$

Choose $r \in Z_q^*$

Set $U = rP$

$V = d_{ID_U} + rH(Q_{ID_U}, M, U)$

→ **verification** $(M, (U,V), ID_U, \text{params}) \rightarrow 1/0$

$$e(V, P) = e(d_{ID_U} + rH(Q_{ID_U}, M, U), P)$$

$$= e(d_{ID_U}, P) \cdot e(rH(Q_{ID_U}, M, U), P)$$

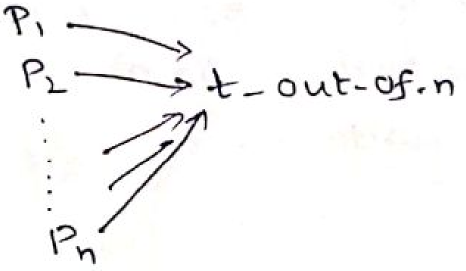
$$= e(sH(ID_U), P) \cdot e(rH(Q_{ID_U}, M, U), P)$$

$$= e(H(ID_U), sP) \cdot e(H(Q_{ID_U}, M, U), rP)$$

$$= e(H(ID_U), P_{pub}) \cdot e(H(Q_{ID_U}, M, U), U)$$

* Shamir's Threshold Secret sharing

K=s



Dealer → chooses a polynomial of degree t-1 ~~t+1~~

S.

~~Let $f(x) = s +$~~

Let $f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{t-1} x^{t-1}$

$a_0, a_1, \dots, a_{t+1} \in F_q$

$a_0 = K = s.$

$\Rightarrow \boxed{f(0) = s}$

not necessarily i, it can be $(id_1, f(id_1))$

$P_1 \cdot (1, f(1))$

$P_2 \cdot (2, f(2))$

$\vdots$

$P_n \cdot (n, f(n))$

If we have parties $P_{i_1}, P_{i_2}, \dots, P_{i_t}$

$\Rightarrow (i_1, f(i_1)), (i_2, f(i_2)), \dots \dots (i_t, f(i_t))$

⇒ we have 't' equations & 't' unknowns, $a_0, a_1, \dots a_{t-1}$

⇒ we can solve to find out $a_0 = s$.

*Security

→ t parties will not get any information about s.

*Cryptographic hash functions :-

→ provides assurance of data integrity

to detect modification of transmitted message (encrypted or not).

Dealer $\longrightarrow$ chooses a polynomial of degree $t-1$ ~~$t+1$~~
S.

~~Let $f(x) = s +$~~

Let $f(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{t-1} x^{t-1}$

$a_0, a_1, \dots, a_{t+1} \in F_q$.

$a_0 = K = s$.

$\Rightarrow \boxed{f(0) = s}$

not necessarily 1, it can be $(id_1, f(id_1))$

$P_1 \cdot (1, f(1))$

$P_2 \cdot (2, f(2))$

$\vdots$

$P_n \cdot (n, f(n))$

If we have parties $P_{i_1}, P_{i_2}, \dots, P_{i_t}$

$\Rightarrow (i_1, f(i_1)), (i_2, f(i_2)), \dots (i_t, f(i_t))$

$\Rightarrow$ we have 't' equations & 't' unknowns, $a_0, a_1, \dots a_{t-1}$

$\Rightarrow$ we can solve to find out $a_0 = s$.

*Security

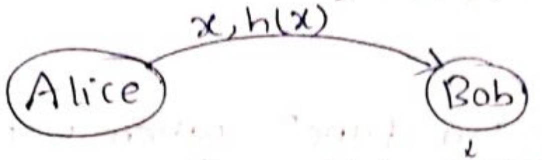
$\rightarrow$ t parties will not get any information about s.

*Cryptographic hash functions :-

$h : X \to Y$ (hash func^n)

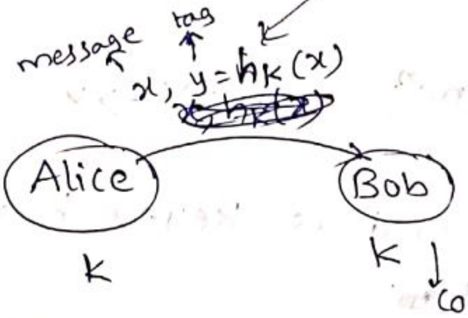
$h(x) = y$ message digest (stored in a secure place).

$\downarrow$ a short fingerprint of data x



(incase 'x' is changed to 'x'') $\to$ compute $y' = h(x')$ and checks if $y' \neq y$

$\to$ hash func^n
- $\to$ unkeyed hash func^n (eg: MDC) (y is stored in a secure place, x is not)
- $\to$ keyed hash func^n (eg: MAC) (Both x & y can be transmitted)



$h_K : X \to Y$

computes $h_K(x)$ & checks whether $y \overset{?}{=} h_K(x)$ $\leftarrow$ Assuming h_K is secure

* MDC $\to$ Modification Detection Code
(or)
Manipulation Detection Code
(or)
Message Integrity Code (MIC)

* Keyed hash families ($h_K : X \to Y$)

$\to (X, Y, K, H)$

$X \to$ a set of possible messages (may be finite or infinite)

$Y \to$ finite set of possible message digests/ tags. /authentication tags.

$K \rightarrow$ keyspace, a finite set of possible keys.
for each $k \in K$, $\exists$ a hash func^n, $h_k \in \underset{\text{Hash family}}{H}$,

$$h_K : X \rightarrow Y$$

* $h_K \rightarrow$ compression func^n when both X & Y are finite and $|X| \geq |Y|$

* $(x, y) \in X \times Y$ is called a valid pair if $h_K(x) = y$.

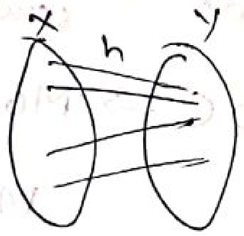
* prevent adversary to construct such valid pairs.

* <u>Unkeyed Hash func^n</u> :- $h : X \rightarrow Y$

~~→ keyed has~~
→ can be viewed as keyed hash functions with $|K| = 1$
i.e only one possible key.

$|X| = N$, $|Y| = M$.

$$\underset{\substack{\downarrow \\ \text{Total No. of functions from } X \rightarrow Y.}}{|F^{X \times Y}|} = M^N$$



$h \in F^{X \times Y}$ is called (M, N) hash family.

* <u>Security of Hash func^{n}s (unkeyed)</u> :-

→ Difficult to solve the following three problems for a cryptographically secure hash functions.

- pre image
- second pre image
- collision.

(P1) → Preimage problem:

Given $h: X \to Y$ and element $y \in Y$, find $x \in X$ such that $h(x) = y$.

→ This should be difficult.

→ Hash func^n for which preimage cannot be found efficiently is called preimage resistant / one-way hash func^n.

(P2) → Second Preimage problem:-

Instance:- A hash func^n, $h: X \to Y$ and an element $x \in X$

Find :- another $x' \in X$, $x' \neq x$ ~~such~~ and $h(x') = h(x)$.

→ Hash func^n for which second preimage cannot be efficiently solved is called second preimage resistant (or) weak collision resistant hash func^n.

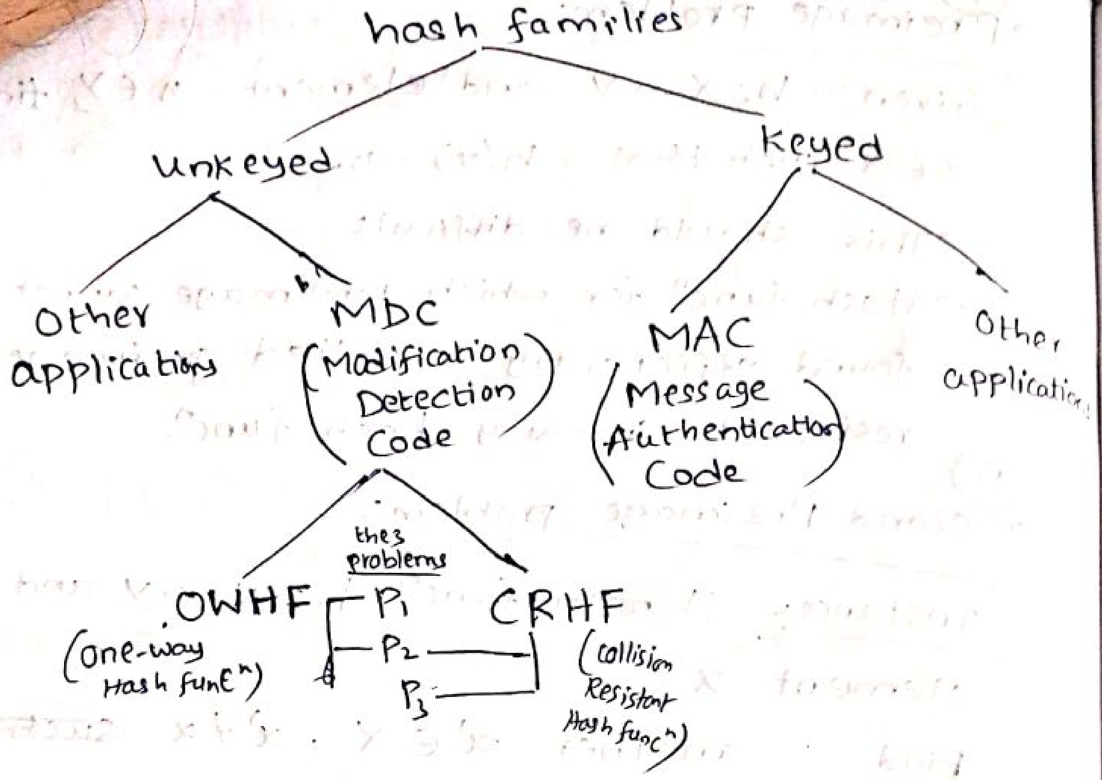
→ Second preimage shouldn't be there for a good hash func^n.

(P3) → Collision problem:-

Instance, Given a hash func^n, $h: X \to Y$

Find :- $x, x' \in X$ s.t. $x \neq x'$ and
$h(x) = h(x')$.

→ Hash func^n for which ~~x~~ such x & x' can't exist is called collision resistant Hash func^n.



* Iterated hash funcn:-

Compression function called 'Compress'.

$\downarrow$ construct

a hash funcn with an infinite domain (using 'Compress')

suppose, Compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^{m}$ $(t \geq 1)$

⇒ construct an iterated hash funcn

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^{i} \longrightarrow \{0,1\}^{l}$$

based on the compression funcn 'Compress'

→ Preprocessing step

Given an input x, $|x| \geq m+t+1$, construct a string y, using a public algorithm such that $|y| = 0 \pmod t$.

Let $y = y_1 || y_2 || \cdots\cdots || y_r$, $|y_i| = t$, $1 \leq i \leq r$

↑ Concatenation.

→ Processing step

Let IV be a public value $\in \{0,1\}^m$

$$Z_0 \leftarrow IV$$

$$\downarrow$$

$$z_1 \leftarrow \text{Compress}(\overset{\text{m-bit}}{Z_0} \,||\, \overset{\text{t-bit}}{y_1})$$

output is of m-bits

$$\downarrow$$

$$Z_2 \leftarrow \text{Compress}(Z_1 || y_2)$$

$$\downarrow$$

$$\vdots$$

$$Z_r \leftarrow \text{Compress}(Z_{r-1} || y_r)$$

→ Output :(optional).

$g: \{0,1\}^m \rightarrow \{0,1\}^l$ a public function.

define $h(x) = g(z_r)$.

* The Markle-Damgård Construction (Iterated hash func^n)

→ Compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$

→ To compute a collision resistant hash func^n,

$$h: X \rightarrow \{0,1\}^m, \quad X = \bigcup_{i \geq m+t+1} \{0,1\}^i$$

→ Preprocessing step

input → $x \in X$ $\quad x || pad(x)$

$|x| = n \geq m+t+1$

$$x = \underset{(t-1)\text{ bit}}{x_1} || \underset{(t-1)\text{ bit}}{x_2} || \underset{(t-1)\text{ bit}}{x_3} \cdots\cdots || \underset{(t-1)\text{ bit}}{x_{k-1}} || \underset{(t-1-d)\text{ bit}}{x_k}$$

$$k = \left\lfloor \frac{n}{t-1} \right\rfloor, \quad d = (t-1)k - n$$

output → $y = y_1 || y_2 || y_3 || \text{-----} y_k || y_{k+1}$

$y_i = x_i$, $1 \le i \le k-1$

$y_k = x_k || 0^d$

y_{k+1} = the binary representation of 'd'

~~$x||Pad(x)$~~

$\Downarrow$

~~$x_1||x_2||\cdots||x_{k-1}||x_k||Pad(x)$~~

$x||Pad(x)$

$\Downarrow$

$$x_1||x_2||\cdots\cdots||x_{k-1}||\underbrace{x_k||0^d}||y_{k+1}$$

$\downarrow \quad \downarrow \quad \Downarrow \qquad \downarrow \qquad \downarrow$

$$y_1||y_2||\cdots\cdots||y_{k-1}||y_k||\underbrace{y_{k+1}}_{t-1 \text{ bit}}$$

if necessary, Pad on the left with zeroes to make $|y_{k+1}| = t-1$

→ Processing

$\underbrace{z_1}_{(m+t)\text{ bit}} = 0^{m+1}||\overset{(t-1)\text{ bit}}{y_1}$

$\underbrace{g_1}_{m \text{ bit}} = \text{Compress}(z_1)$

$$\left.\begin{aligned} \underbrace{z_{i+1}}_{(m+t)\text{ bits}} &= g_i||1||y_{i+1} \\ g_{i+1} &= \text{Compress}(z_{i+1}) \end{aligned}\right] \text{ for } 1 \le i \le k$$

Finally, set $h(x) = g_{k+1}$ (m-bit)

If funcⁿ compress is collision-resistant, then this hash funcⁿ is secure.

* Design principle of collision-resistant iterated Hash func^n:-

input ↑ output of preprocessing step

→ mapping $x \to y$ must be injective

→ Compress func^n must be collision resistant

* Ex^t of Compress func^n

Compress: $\{0,1\}^{m+t} \to \{0,1\}^m$.

→ SHA-1(x) Secure Hash Function

Input:- x, $|x| \le 2^{64}-1$

Output:- 160-bit message digest

$H_0||H_1||H_2||H_3||H_4$ → $32 \times 5 = 160$

32-bit each

→ preprocessing

$y = SHA\text{-}1\text{-}PAD(x)$

$= M_1||M_2||\cdots\cdots||M_n = x||pad(x)$

$|M_i| = 512$-bit

$= x||1||0^d||l$ → binary representation of $|x|$ in 64-bit

func^n:-

SHA_1_PAD(x)

Input: x with $|x| \le 2^{64}-1$

Output: y with $512 \mid |y|$

set $y = x||1||0^d||l$ → size of (x) in 64-bit binary.

Since $512 \mid |y|$

$\Rightarrow |y| = 512(k+1)$

$d = 512(k+1) - |x| - 64 - 1$

$d = 512k + (447 - |x|)$, k is an integer

Compress: $\{0,1\}^{512+160} \rightarrow \{0,1\}^{160}$

$$Z_1 = \text{Compress}(\overset{\nearrow 160}{IV} \,||\, \overset{\nearrow 512}{M_1})$$

$$Z_2 = \text{compress}(Z_1 \,||\, M_2)$$

$$Z_3 = \text{compress}(Z_2 \,||\, M_3)$$

$$\vdots \qquad \vdots$$

$$Z_n = \text{compress}(Z_{n-1} \,||\, M_n)$$

32-bit

$$IV = H_0 \,||\, H_1 \,||\, H_2 \,||\, H_3 \,||\, H_4$$

$$y = M_1 \,||\, M_2 \,||\, M_3 \,||\, M_4 \,||\, M_5 \,||\, \ldots$$

Initially, $H_0 = 0x67452301$

$H_i = - - -$

$\vdots$

```
for i=1 to n do
    let $M_i = W_0 || W_1 || \ldots || W_{15}$ → $32 \times 16 = 512$ bit
        each $W_i$ is 32-bit
    for j=16 to 79 do
        $W_j = \underbrace{ROTL^1}_{\text{Left-circular shift by 1-bit}}(W_{j-3} \oplus W_{j-8} \oplus W_{j-14} \oplus W_{j-16})$
    end do.
    $(A, B, C, D, E) \leftarrow (H_0, H_1, H_2, H_3, H_4)$.
    for j=0 to 79 do
        temp := $ROTL^5(A) + f_j(B,C,D) + E + W_j + K_j$
            E := D
            D := C
            C := B
            B := A
            A := temp
    end do
    $(H_0, H_1, H_2, H_3, H_4) \leftarrow (H_0+A, H_1+B, H_2+C, H_3+D, H_4+E)$
end do
```

32-bit
↑
given constants
↑
K_j

$$\rightarrow f_j(B,C,D) = \begin{cases} (B\wedge C)\vee(\neg B\wedge D), & 0\le j\le 19 \\ B\oplus C\oplus D, & 20\le j\le 39 \\ (B\wedge C)\vee(B\wedge D)\vee(C\wedge D), & 40\le j\le 59 \\ B\oplus C\oplus D, & 60\le j\le 79 \end{cases}$$

* SHA-3 Competition (2007-2012)

- 64-submissions by Oct 2008.
  - 1st round → 51 survived
  - 2nd " → 14 "
  - 3rd " → 5 "
- 5 finalists in Dec 2010.
  - BLAKE
  - GrØstl
  - JH
  - Keccak → known as SHA-3 ↓ in Oct 2012
  - Skein

→ Keyed Hash funcⁿ:-

CBC-MAC (x, K)

$$x = \underbrace{x_1}_{\text{m-bit}} \,||\, \underset{\text{m-bit}}{\underset{\downarrow}{x_2}} \,||\, \cdots \,||\, \underset{\text{m-bit}}{\underset{\downarrow}{x_n}}$$

```
IV = 000 ..... 0 (m-bit)
$y_0 \leftarrow IV$
for i=1 to n do
    $y_i = \mathcal{E}_K(y_{i-1} \oplus x_i)$
end do
return $y_n$.
```

* Burmester-Desmedt: Group key Agreement:

→ 2 rounds for any number of parties.

$$\begin{tikzcd}
U_1 & U_2 & U_3 & \cdots & U_n \\
x_1 \arrow[d, "\text{send}"'] & x_2 \arrow[d] & x_3 \arrow[d] & \cdots & x_n \arrow[d] \\
\text{round-1} \rightarrow g^{x_1} & g^{x_2} & g^{x_3} & \cdots & g^{x_n}
\end{tikzcd}$$

$G = \langle g \rangle$
$|G| = q$
$x_1, \ldots, x_n$ chosen by each user.

Left $\leftarrow$ $K_1^L = (g^{x_n})^{x_1}$

Right $\leftarrow$ $K_1^R = (g^{x_n})^{x_2}$

round-1 User $U_i(x_i)$ $\rightarrow$ sends g^{x_i}

$\rightarrow$ computes $K_i^L = \left(g^{x_{i-1}}\right)^{x_i}$

$K_i^R = \left(g^{x_{i+1}}\right)^{x_i}$

Round-2 $\quad y_1 = \frac{K_1^R}{K_1^L} \quad y_2 = \frac{K_2^R}{K_2^L} \quad \cdots \quad y_n = \frac{K_n^R}{K_n^L}$

round-2 User U_i sends $\frac{K_i^R}{K_i^L} (= y_i)$

→ key computation

$$U_j (x_j) \begin{cases} K_i^L \\ K_i^R \\ y_i \end{cases} \qquad y_{i+1}, y_{i+2}, \ldots, y_1, y_2 \ldots, y_{i-1}$$

U_i computes $K_{i+1}^R = y_{i+1} K_i^R = \frac{K_{i+1}^R}{\cancel{K_{i+1}^L}} \cancel{K_i^R}$

$[\because K_i^R = K_{i+1}^L]$

$K_{i+2}^R = y_{i+2} K_{i+1}^R$

$\vdots$

$K_{i+(n-1)}^R = y_{i+(n-1)} K_{i+(n-2)}^R$

Verifies whether $K_{i+(n-1)}^R = K_i^L$

If not equal, abort

else compute the session key

$G = \langle g \rangle$
$|G| = q$, DLP hard
$x_1, \dots, x_n \in Z_q^*$
chosen by each user

U_1 U_2 U_3 $\dots$ U_n

x_1 x_2 x_3 $\dots$ x_n

send $\rightarrow \downarrow$

round-1 $\rightarrow g^{x_1}$ g^{x_2} g^{x_3} - - - - - - g^{x_n}

Left $\leftarrow$ $K_1^L = (g^{x_n})^{x_1}$

Right $\leftarrow$ $K_1^R = (g^{x_n})^{x_2}$

d-1 User $U_i(x_i)$ $\rightarrow$ sends g^{x_i}

$\rightarrow$ computes $K_i^L = (g^{x_{i-1}})^{x_i}$

$K_i^R = (g^{x_{i+1}})^{x_i}$

ound-2

$Y_1 = \frac{K_1^R}{K_1^L}$ $Y_2 = \frac{K_2^R}{K_2^L}$ - - - - $Y_n = \frac{K_n^R}{K_n^L}$

d-2 User U_i sends K_i^R $(= Y_i)$

$$SK = K_1^R K_2^R \cdots\cdots K_n^R$$

$$= g^{x_1 x_2 + x_2 x_3 + \cdots\cdots + x_n x_1}$$

security → DDH assumption.

* Post-quantum Cryptography

Classical computer — 0's & 1's

quantum computer — Qubit — 0's, 1's & superposition.

→ quantum computer → can be used to solve DLP.

Shor's algorithm (1994)

→ Integers can be factored quickly using a quantum computer.

$$O((\log n)^2 (\log \log n)(\log \log \log n))$$

→ practical quantum computers yet to come.

* post quantum Cryptography

→ Study potential ways to construct PKC based on different computational problems, not susceptible to attacks carried out by quantum computers.

→ approaches

- → lattice-based Crypto → Ex: NTRU
- → Code-based Crypto
- → Multi-variate crypto
- → Hash-based crypto
- → Isogeny-based crypto

NTRU: Nth degree TRuncated poly ring

↓ closest vector problem

shortest vector problem

Isogeny-based crypto ↓ gives lesser size keys

↓ more sophisticated

* Quantum Cryptography

→ refers to algorithms/primitives that rely on quantum mechanical techniques for their implementation.

* NTRU ~~poly ring~~

$R = Z[x]/(x^N - 1)$

working poly Ring

$f \in R$, $f(x) = a_0 + a_1 x + \cdots + a_{N-1} x^{N-1}$

$a_0, a_1, \ldots, a_{N-1} \in Z$

Setup :- p, q, N integers,

$q >> p$, $p > N$; $\gcd(p, q) = 1$

$p \to$ odd, $N \to$ prime.

$\mathcal{P} = \{-1, 0, 1\}^N$, $\mathcal{C} = \mathbb{Z}_J^N$
set of integers

choose $F, G \in \{-1, 0, 1\}^N$

set $f = 1 + pF$, $g = pG$.
number 1, not vector

define $h = f^{-1} * g \mod q$
convolution.

$pk = h$, $sk = f$

$\sum_{i=0}^{N-1} a_i x^i$
$=$
$a(x) \leftarrow a = (a_0, a_1, \ldots, a_{N-1}) \in R$ ↗ vector
|||
polynomial

$b(x) \leftarrow b = (b_0, b_1, \ldots, b_{N-1}) \in R$

$c(x) = a(x)\, b(x) = \sum_{i=0}^{N-1} c_i x^i$

$\Rightarrow c_i = \sum_{j=0}^{i} a_j b_{i-j}$

Then $c = a * b$ [convolution operator]

→ We define

a mods n = b, if a ≡ b (mod n)

[illegible] $-\frac{n}{2} < b \le \frac{n}{2}$

Eg: a mods 5 ∈ {-2, -1, 0, 1, 2}

a mod 5 ∈ {0, 1, 2, 3, 4}

in mods, 4 = -1

in mods, 3 = -2

→ coefficients of $h(x)$ are in the rage,

$\left[-\frac{q-1}{2}, \frac{q}{2}\right]$

~~coefficients of f → {-p, 0, p}~~

coefficients of g → $\{-p, 0, p\}$

Encrypt $(m, pk = h)$

$m \in \mathcal{P} = \{-1, 0, 1\}^N$

Choose randomly $r \in \{-1, 0, 1\}^N$

& set the ciphertext,

$y = (r * h + m) \text{ mods } q$

Decrypt $(y, sk = f)$

$(f * y \text{ mods } q) \text{ mods } p$

Correctness

1st Consider mod q

$f * y = f * ((r * h + m) \text{ mods } q)$

$= (f * r * f^{-1} * g + f * m)$

$f * y \equiv (r * g + f * m) \bmod q$

suppose this congruence is actually equality in R → holds with high probability if the parameters are chosen in a suitable way.

* Quantum Cryptography

→ refers to algorithms/primitives that rely on quantum mechanical techniques for their implementation.

* NTRU ~~poly ring~~

$R = Z[x]/(x^N - 1)$

working poly Ring

$f \in R$, $f(x) = a_0 + a_1 x + \dots + a_{N-1} x^{N-1}$

$a_0, a_1, \dots, a_{N-1} \in Z$

Setup :- p, q, N integers,

$q >> p$, $p > N$; $\gcd(p, q) = 1$

$p \to$ odd, $N \to$ prime.

$\mathcal{P} = \{-1, 0, 1\}^N$, $\mathcal{C} = \underset{\text{set of integers}}{\mathbb{Z}^N}$

choose $F, G \in \{-1, 0, 1\}^N$

set $f = \underset{\text{number 1, not vector}}{1} + pF$, $g = pG$.

define $h = f^{-1} \underset{\text{convolution}}{*} g$ mods q

$pk = h$, $sk = f$

$\sum a_i x^i$

$a(x) \leftarrow a = (a_0, a_1, \dots, a_{N-1}) \in R$ ↗ vector ||| polynomial

$b(x) \leftarrow b = (b_0, b_1, \dots, b_{N-1}) \in R$

$c(x) = a(x) b(x) = \sum_{i=0}^{N-1} c_i x^i$

$\Rightarrow c_i = \sum_{j=0}^{i} a_j b_{i-j}$

Then $c = a * b$ [convolution operator].

We define

a mods n = b if $a \equiv b \pmod{n}$ &

$-\frac{n-1}{2} \leq b \leq \frac{n}{2}$

Eg: a mods $5 \in \{-2, -1, 0, 1, 2\}$

a mod $5 \in \{0, 1, 2, 3, \textcircled{4}\}$

in mods, $\equiv -1$

in mods, $\equiv -2$

→ coefficients of $h(x)$ are in the rage,

$\left[-\frac{q-1}{2}, \frac{q}{2}\right]$

~~coefficients of $f \to \{-p, 0, p\}$~~

coefficients of $g \to \{-p, 0, p\}$

Encrypt (m, pk = h)

$m \in \mathcal{P} = \{-1, 0, 1\}^N$

Choose randomly $r \in \{-1, 0, 1\}^N$

& set the ciphertext,

$y = (r * h + m)$ mods q

Decrypt (y, sk = f)

$(f * y$ mods $q)$ mods p

Correctness

$f * y = f * ((r * h + m)$ mods $q)$ ↙ 1st consider mod q

$= (f * r * f^{-1} * g + f * m)$

$f * y \equiv (r * g + f * m) \bmod q$

suppose this congruence is actually equality in R → holds with high probability if the parameters are chosen in a suitable way.

The above becomes equality if mods q is taken instead of mod q.

[∵ f & g has coefficents in that particular way]

$\Rightarrow f * y = r * g + f * m$

$(f * y) \bmod p = (r * g$ ~~mod p~~ $+ f * m) \bmod$

$= (r * pG + (1 + pF) m) \bmod p$

$= m \bmod p$ → These are modulo p.

$\Rightarrow m = f * y \text{ mods } p.$