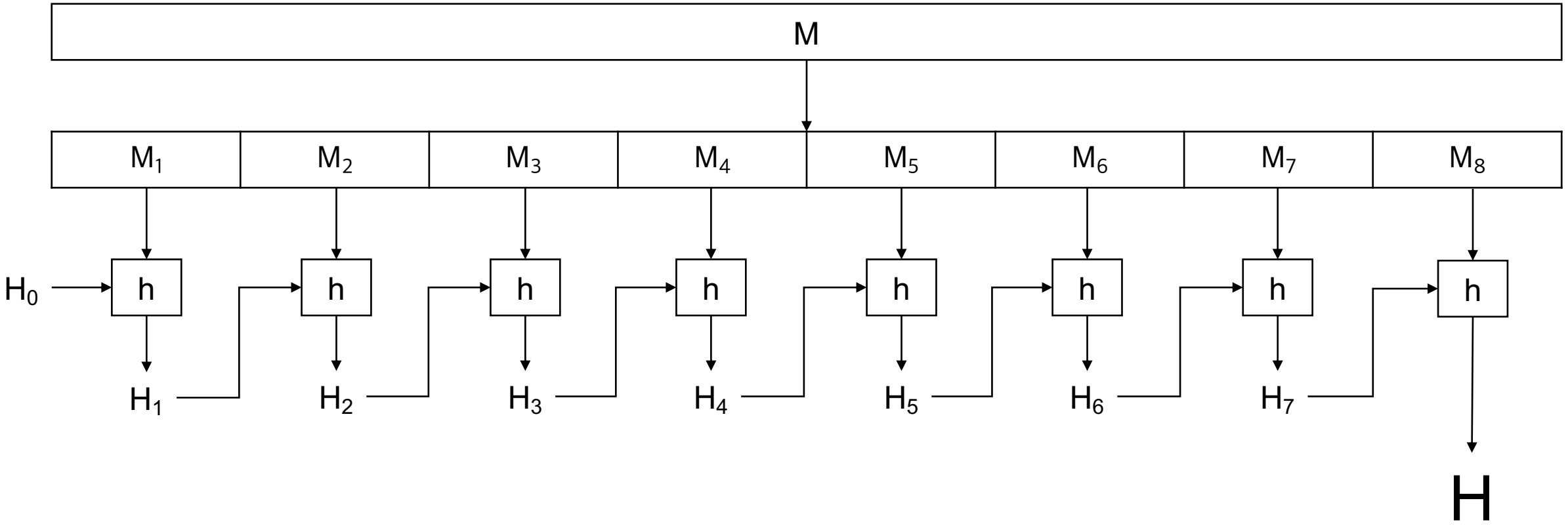
Hash Functions Based on Block Ciphers

https://youtu.be/iQ3I7-CSdn8

Iterated hash functions and attacks

$$H_i = h(H_{i-1}, M_i) \qquad i = 1, 2, ..n,$$



Iterated hash functions and attacks

1. **Target attack:** Given H_0 and M, find M' such that $M' \neq M$ but $\text{Hash}(H_0, M') = \text{Hash}(H_0, M)$.

2. **Free-start target attack:** Given H_0 and M, find H_0' and M' such that $(H_0', M') \neq (H_0, M)$ but $\text{Hash}(H_0', M') = \text{Hash}(H_0, M)$.

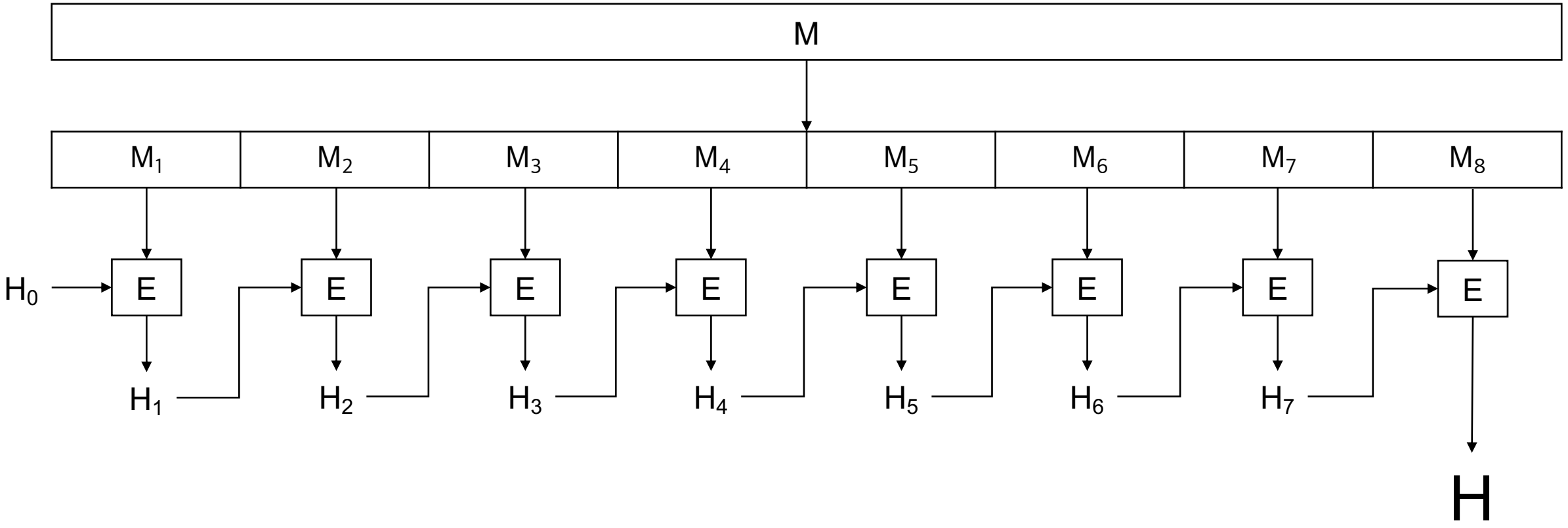
3. **Collision attack:** Given H_0, find M and M' such that $M' \neq M$ but $\text{Hash}(H_0, M') = \text{Hash}(H_0, M)$.

4. **Semi-free-start collision attack:** Find H_0, M and M' such that $M' \neq M$ but $\text{Hash}(H_0, M') = \text{Hash}(H_0, M)$.

5. **Free-start collision attack:** Find H_0, H_0', M and M' such that $(H_0', M') \neq (H_0, M)$ but $\text{Hash}(H_0', M') = \text{Hash}(H_0, M)$.

Hash round functions based on block ciphers

$$H_i = E(H_{i-1}, M_i) \qquad i = 1, 2, ..n,$$

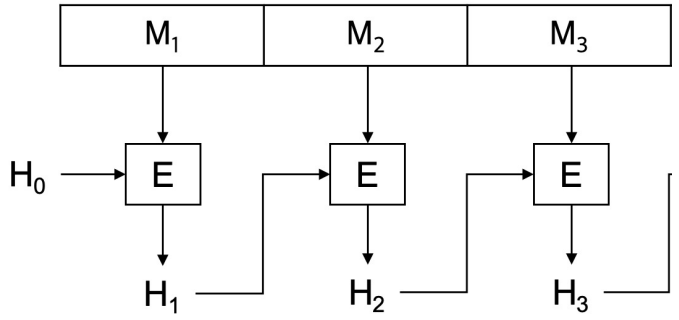


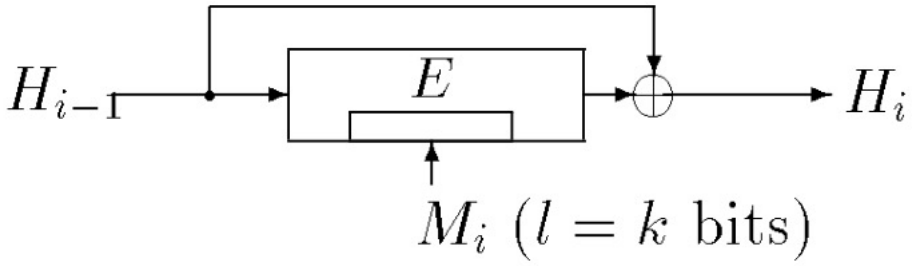
m-bit hash round functions

- Davies-Meyer(DM) scheme
  - 어느 블록 암호에나 적용 가능한 구조를 가지고 있음.
  - Hash size = block size(m-bit)
  - ℓ size = key size
    - ex) PIPO-64/128 : Hash size(64-bit) / ℓ size(128-bit)

$$H_i = h(H_{i-1}, M_i)$$

$$h(H_{i-1}, M_i) = E_{M_i}(H_{i-1}) \oplus H_{i-1}$$



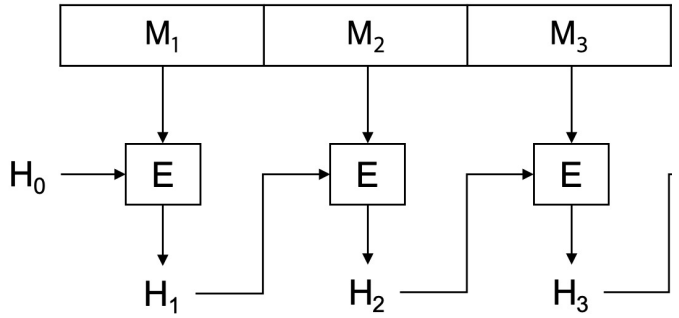


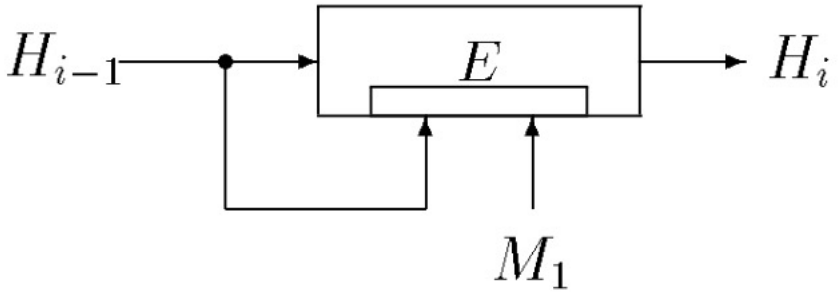
m-bit hash round functions

- A proposed m-bit hash round function using a block cipher with m-bit bl ock and 2m-bit key
  - Hash size = block size(m-bit)
  - ℓ size = block size(m-bit)
    - ex) PIPO-64/128 : Hash size(64-bit) / ℓ size(64-bit)

$$H_i = h(H_{i-1}, M_i)$$

$$h(H_{i-1}, M_i) = E_{H_{i-1}, M_i}(H_{i-1})$$





2m-bit hash round functions

- The Preneel-Bosselaers-Govaerts-Vandewalle(PBGV) scheme
  - block size : m-bit , key size : m-bit
  - Hash size = 2 * block size(m-bit)
  - ℓ size = key size(m-bit), M -> (L_1, N_1, ⋯ , L_n, N_n)
    - ex) LEA-128/128 : Hash size(256-bit) / ℓ size(128-bit)

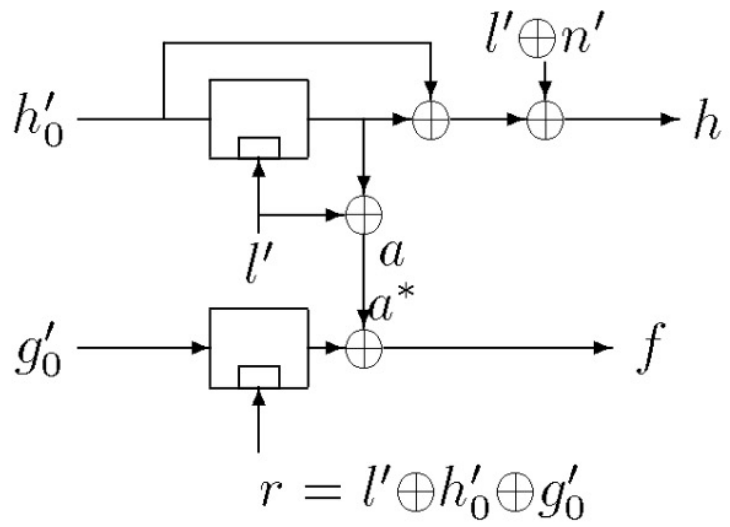
$$
\begin{aligned}
H_i &= E_{L_i \oplus N_i}(H_{i-1} \oplus G_{i-1}) \oplus L_i \oplus H_{i-1} \oplus G_{i-1} \\
G_i &= E_{L_i \oplus H_{i-1}}(N_i \oplus G_{i-1}) \oplus N_i \oplus H_{i-1} \oplus G_{i-1}
\end{aligned}
$$

$$
\begin{aligned}
h &= E_{l \oplus n}(h_0 \oplus g_0) \oplus l \oplus h_0 \oplus g_0 \\
g &= E_{l \oplus h_0}(n \oplus g_0) \oplus n \oplus h_0 \oplus g_0.
\end{aligned}
$$

$$
(h, g) \longrightarrow (h, f) = (h, h \oplus g)
$$

$$
(h_0, g_0, l, n) \longrightarrow (h'_0, g'_0, l', n') = (h_0 \oplus g_0, g_0 \oplus n, l \oplus n, n),
$$

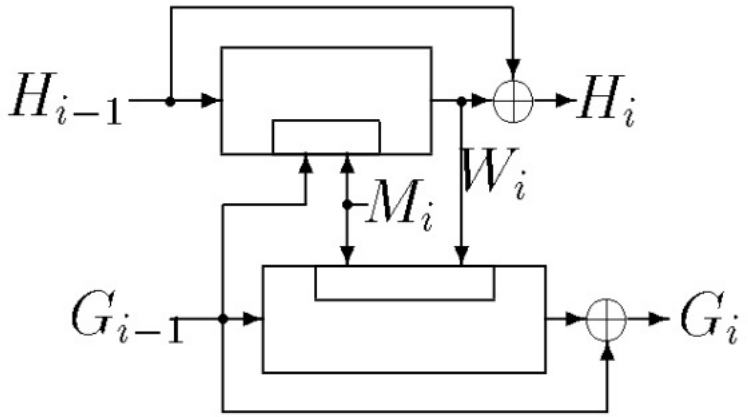
$$
\begin{aligned}
h &= E_{l'}(h'_0) \oplus l' \oplus n' \oplus h'_0 \\
f &= E_{l' \oplus h'_0 \oplus g'_0}(g'_0) \oplus E_{l'}(h'_0) \oplus l'.
\end{aligned}
$$



Proposed 2m-bit hash round functions

- Tandem DM
  - 두 개의 DM schemes 구조를 사용한 구조
  - Hash size = 2 * block size(m-bit)
  - ℓ size = block size(m-bit)
    - ex) PIPO-64/128 : Hash size(128-bit) / ℓ size(64-bit)

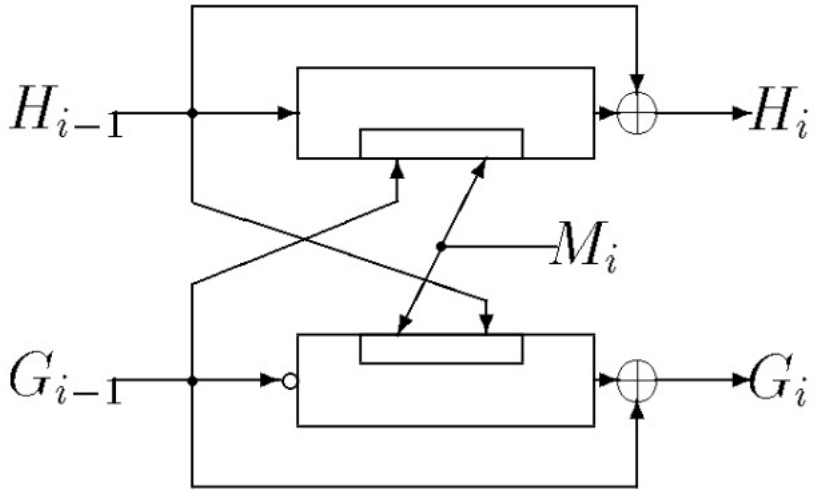
$$
\begin{aligned}
W_i &= E_{G_{i-1},M_i}(H_{i-1}) \\
H_i &= W_i \oplus H_{i-1} \\
G_i &= G_{i-1} \oplus E_{M_i,W_i}(G_{i-1}).
\end{aligned}
$$



Proposed 2m-bit hash round functions

- Abreast DM
  - 두 개의 DM schemes 구조를 사용한 구조
  - Hash size = 2 * block size(m-bit)
  - ℓ size = block size(m-bit)
    - ex) PIPO-64/128 : Hash size(128-bit) / ℓ size(64-bit)

$$
\begin{aligned}
H_i &= H_{i-1} \oplus E_{G_{i-1}, M_i}(H_{i-1}) \\
G_i &= G_{i-1} \oplus E_{M_i, H_{i-1}}(\overline{G}_{i-1})
\end{aligned}
$$



Tandem DM (PIPO, LEA, CHAM)

- PIPO(**64/128**, 64/256)
  - PIPO-64/128
  - Hash size : 128-bit

- LEA(128/128, 128/192, **128/256**)
  - LEA-128/256
  - Hash size : 256-bit

- CHAM(64/128, 128/128, **128/256**)
  - CHAM-128/256
  - Hash size : 256-bit

Tandem DM (PIPO, LEA, CHAM)

```
void sbox(u8 *X);
void keyadd(u8* val, u8* rk);
void pbox(u8* X);
void ROUND_KEY_GEN(u32* MASTER_KEY, u32* ROUND_KEY);
void ENC(u32* INPUT, u32* ROUND_KEY, u32* OUTPUT);

void tandm_init_key(u32 *Front, u32 *back, u32 *key);
void tandm_pipo(char *msg, u32 *output_HASH);
```

PIPO

```
void ROUND_KEY_GEN_128(u32 *mk, u32 *rk);
void ROUND_KEY_GEN_256(u32 *mk, u32 *rk);
void ENC(u32 *INPUT, u32 *rk, u32 *OUTPUT);

void tandm_init_key(u32 *Front, u32 *back, u32 *key);
void tandm_lea(char *msg, u32 *output_HASH);
```

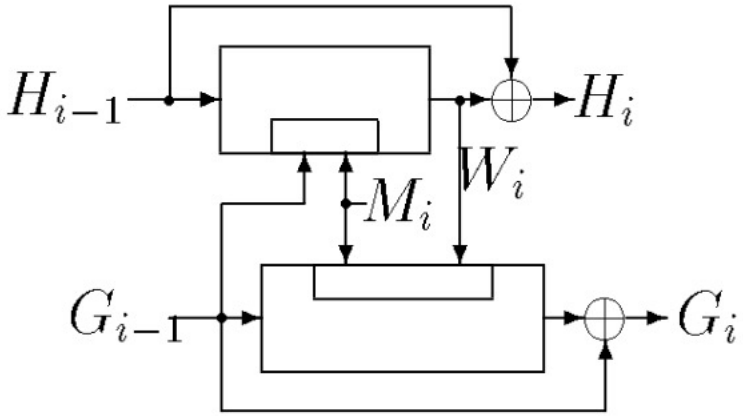
LEA

```
void ROUND_KEY_GEN(u32 *mk, u32 *rk);
void ENC(u32 *INPUT, u32 *roundkey, u32 *OUTPUT);

void tandm_init_key(u32 *Front, u32 *back, u32 *key);
void tandm_cham(char *msg, u32 *output_HASH);
```

CHAM

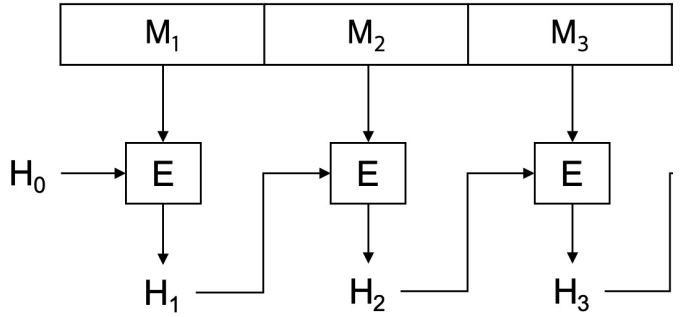
Tandem DM (PIPO, LEA, CHAM)

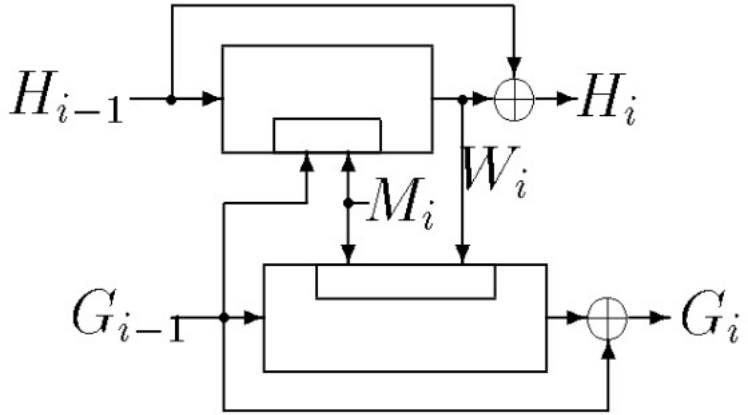
```
void tandm_init_key(u32 *Front, u32 *back, u32 *key){
    key[0] = Front[0];
    key[1] = Front[1];
    key[2] = back[0];
    key[3] = back[1];
}
```



Tandem DM (PIPO, LEA, CHAM)

```c
void tandm_pipo(char *msg, u32 *output_HASH){
    u32 msgLen = (u32)strlen(msg); //msg len byte
    printf("\nmsgLen = %d\n", msgLen);

    u32 loop_len = msgLen / 8;
    printf("loop_len = %d\n", loop_len);

    u32 key[4] = {0x0,};
    u32 roundkey[28] = {0x0,};
    u32 H[2] = {0x0,};
    u32 W[2] = {0x0,};
    u32 G[2] = {0x0,};
    u32 temp[2] = {0x0, };

    temp[0] = (u32)msg[0]|((u32)msg[1]<<8)|((u32)msg[2]<<16)|((u32)msg[3]<<24);
    temp[1] = (u32)msg[4]|((u32)msg[5]<<8)|((u32)msg[6]<<16)|((u32)msg[7]<<24);

    for(int loop=1; loop<loop_len+1; loop++){
        tandm_init_key(G, temp, key);
        ROUND_KEY_GEN(key, roundkey);
        ENC(H, roundkey, W);
        tandm_init_key(temp, W, key);
        ROUND_KEY_GEN(key, roundkey);
        ENC(G, roundkey, temp);
        H[0] ^= W[0]; H[1] ^= W[1];
        G[0] ^= temp[0]; G[1] ^= temp[1];

        if(loop!=loop_len){
            temp[0] = (u32)msg[loop*8]|((u32)msg[loop*8+1]<<8)|((u32)msg[loop*8+2]<<16)|((u32)msg[loop*8+3]<<24);
            temp[1] = (u32)msg[loop*8+4]|((u32)msg[loop*8+5]<<8)|((u32)msg[loop*8+6]<<16)|((u32)msg[loop*8+7]<<24);
        }
    }

    output_HASH[0] = H[0];
    output_HASH[1] = H[1];
    output_HASH[2] = G[0];
    output_HASH[3] = G[1];

}
```





Q & A