1) Callision Resistant Hash Function

a) Yes, H_2 is more secure than H_0 and H_1.

Argument: Let us assume H_2 is not callision-resistant and H_0 and H_1 are callision resistant.
i.e. $H_2(x) = H_2(x')$; $x \neq x'$

$$\Rightarrow H_0(x) \,||\, H_1(x) = H_0(x') \,||\, H_1(x')$$

This means that;

$$H_0(x) = H_0(x') \;\&\; H_1(x) = H_1(x'),$$ i.e

H_0 and H_1 are not callision-resistant.

This is a contradiction to H_0 and H_1 being callision resistant. Hence H_2 is callision resistant if both H_0 and H_1 are callision resistant. Similarly, even if one of H_0 or H_1 is not callision resistant hash function, H_2 would still be callision resistant hash function. Hence, H_2 is more secure than H_0 and H_1 provided either H_0 or H_1 is callision resistant and also H_2's length being a combination of the lengths of H_0 and H_1, extra effort would be required to break H_2 and is not robust for pseudorandomness. Therefore, H_2's brute force attack & birthday attack is difficult. Whereas for H_2', H_2' may or may not be more secure.

(H_2') It is secure for pseudorandomness if instantiated with 2 independent hash functions. However, it is not robust for callision resistance, nor even callision resistance preserving. But, it(H_2') can be as resistant as H_0 and H_1 in certain cases.

Ref: Cryptographic's Short Combiners for Callision - Resistant Hash Function - Arno Mittelbach.

1) Collision Resistant Hash Function

b)

H_0 is collision resistant

Let's define $H_3(x) = H_0(H_0(x))$; → which is collision resistant.

In this case, the output length would be n in H_3 and H_0 and effort needed to perform a collision is $2^{n/2}$.

However, if even a bit a H_3 is truncated, the effor to perform collision reduces drastically to $2^{(n-1/2)}$, thereby making

$H_4(x) := H_3(x)[0 .. n-2]$ not collision resistant.

2) Authenticated Encryption

a) $m, MAC_{k2}(E_{k1}(m))$

INSECURE because the message is sent without encryption. Thereby allowing an attacker to intercept the message. However, forgery is not possible as integrity has be accounted for.

b) $E_{k1}(m, MAC_{k2}(m))$

SECURE because the message is encrypted. But, it does not have integrit ($MAC_{k2}(m)$ is not encrypted).

c) $MAC_{k2}(E_{k1}(m))$ - It is falsely designed.

INSECURE because MAC is sent and any alteration to MAC would not be identified by Bob (as the message would not be sent to Bob) provided MAC_{k2} is reversible. However, it is not forgeable.
which is usually not reversible.

d) $E_{k1}(m)$, ~~E~~ $(MAC_{k2}(m))$

INSECURE because $MAC_{k2}(m)$ is not encrypted and does not provide secrecy and hence may reveal parts of the plaintext m (no integrity). Yes, it is forgeable as well (as it has no cipher-text integrity).

e) $E_{k1}(m), E_{k1}(MAC_{k2}(m))$

SECURE : However, ciphertext integrity is not present. Hence, it is forgeable.

f) $E_{k_1}(m), MAC_{k_2}(E_{k_1}(m))$

<u>SECURE</u> as m is encrypted (providing secrecy) and ensures authentication. It is also not forgeable (has cipher text integrity)

3) Timing attack on MAC Verification

a) $t[i] \neq t'[i]$ is a byte-to-byte comparison operation that would help the attacker perform the timing attack.

<u>Case 1</u>: 1^{st} byte of m matches $\rightarrow t_1$

1^{st} byte of m does not match $\rightarrow t_1'$

$t_1' < t_1$; therefore attacker can comfortably presume that there was a match for the first byte while brute-forcing with 2^n ~~256~~ combinations for the 1^{st} byte

<u>Case 2</u>: Similarly,

1^{st} byte match + 2^{nd} byte match $\rightarrow t_2$

1^{st} byte match + 2^{nd} byte does not match $\rightarrow t_2'$

$t_2 > t_2'$; again, the attacker would be able to determine the correction match for byte 2.

Therefore, the attacker can repeat the process of timing the matching bytes and determine the message.

3) Timing Attacks on MAC Verification

b). The solution would be to replace $t[i] \neq t'[i]$ with $HMAC(key, mac) \neq HMAC(k, sig\text{-}bytes)$

$\approx HMAC(k, t[i]) \neq HMAC(k, t'[i])$

This comparison always takes the same time for each byte regardless of whether there is a match or not. Hence, attacker would not be able to determine the bytes of the messages even with brute force.

Final V would be:

for $i \leftarrow 0$ to $n-1$ do
  if $HMAC(k, t[i]) \neq HMAC(k, t'[i])$ then
    output reject and exit
  end if
end for
output reject.

4) Key Exchange.

a) The attack performed in the paper is called Logjam. It is an attack on TLS that allows a man-in-the middle to downgrade vulnerable TLS connection to 512-bit export grade cryptography. This attack has been made possible because of a TLS protocol flaw. In, the paper, it is shown ~~the attacker has~~ that it is possible to perform NFS pre-computation for the popular 512 bit primes and the discrete log for any key exchange message using the prime can be computed. Eventually, it has been shown how ~~the~~ a man-in-the-middle attacker can recover the session key due to the flaw.

Two primary key problems are:

1) The generated groups are not proper in practice and hence as susceptible to attack.

2) Multiple servers use the same prime numbers for key exchange (Diffie Hellman). A brute force of the prime number and an an efficient algorithm can be used to break Diffie-Hellman.

4) Key Exchange

b) Two solutions are:

1) TLS protocol should be altered to include generating and validating parameters. Also, generating new and fresh groups will help mitigate the attack in implementations that use 1024 bit groups. To add, the use of safe prime ~~should~~ by the servers should be validated by the clients

2) Usage of Elliptical curve Diffie-Hellman instead of finite field Diffie Hellman would help mitigate the issue. Pre-computation would be hard and hence the attack would not be feasible

5) Factoring RSA Modulus

c) Let $X = \frac{3p+2q}{2}$

To prove: $X - \sqrt{6N} \leq 1$; i.e they are almost equivalent or close to each other.

$$X^2 - (\sqrt{6N})^2 = \left[\frac{3p+2q}{2}\right]^2 - 6N.$$

$$= \frac{9p^2 + 4q^2 + 12pq - 24N}{2\times 2} \quad ; \quad N \approx pq$$

$$= \frac{9p^2 + 4q^2 - 12pq}{4}$$

$\Rightarrow$

$$X^2 - (6N) = \left(\frac{3p-2q}{2}\right)^2$$

$$X - \sqrt{6N} \Rightarrow \frac{(X - \sqrt{6N})(X + \sqrt{6N})}{(X + \sqrt{6N})}$$

$$= \frac{X^2 - 6N}{(X + \sqrt{6N})} = \frac{3p-2q}{4(X + \sqrt{6N})}$$

Given. $\sqrt{6N} \leq X$

$$X - \sqrt{6N} = \frac{(3p-2q)^2}{4(X + \sqrt{6N})}$$

$$X - \sqrt{6N} \leq \frac{(3p-2q)^2}{8\sqrt{6N}}$$

Given: $|3p - 2q| < N^{1/4}$

$$\Rightarrow |3p-2q|^2 < N^{1/2}$$

Hence, $X - \sqrt{6N} \leq \dfrac{\cancel{\sqrt{N}}}{8\sqrt{6} \times \cancel{\sqrt{N}}}$

$$X - \sqrt{6N} \leq \frac{1}{8\sqrt{6}} \leq 1$$

Thus, $\dfrac{3p+2q}{2}$ and $\sqrt{6N}$ are very close to each other.

Scanned by CamScanner

6 ~~a)~~. Strong Second-Preimage Resistance

a) Take a function $H: S \rightarrow \{0,1\}^k$ where S is a large finite subset of $\{0,1\}^*$, such that H is collision resistant and first-pre-image resistant. Eg: SHA-512 for $k = 512$. Let 0 and 1 be 2 public distinct elements of S. Define $H': S \rightarrow \{0,1\}^k$ by

$$H'(M) = \begin{cases} H(1) & \text{if } M \text{ is } 0 \\ H(M) & \text{otherwise} \end{cases}$$

This H' is compressing, and not collision-resistant (since, ~~E~~ $H'(0) = H'(1)$). However, H' remains pre-image resistant, both first and second, but not collision resistant.

Real Life Example is where strong second-preimage ~~(5)~~ resistance is required, but not collision resistance is hashing of passwords for storing them.

b)

$$H'(x,y) = H^1(x_1,y_1) \| H^2(x_2,y_2) \| H^3(x_3,y_3) \| \ldots H^n(x_n y_n)$$

H being a strong SPR, a concatenation of n H's ~~and n(~~ results in a hash function H' that is a strong SPR and collision resistant.