1. Given : Hash function is not collision resistant

⇒ Adversary can find two messages m and m' such that

$h(m) = h(m')$ in feasible time

Suppose adversary ~~chooses m~~ wants to forge a signature on m

so, adversary can simply send m' to Alice and get the signature s' for m'

$$s' = sig_K(h(m')) = (H(m'))^d = (H(m))^d$$
$$= sig_K(h(m))$$
$$= s$$

Thus, the adversary can forge the same signature s' on m and achieves existential forgery. for

2.

a) Digital signature scheme is secure if an adversary cannot make existential forgery ~~(most~~ (easiest forgery) under ~~the~~ chosen message attack (~~the strongest attack~~).

b) Full domain hash signature scheme

~~Sig(m) :~~

~~s = H(m~~

Key Generation :

Choose 2 large primes p, q

Compute $n = pq$

~~Com~~ Find e such that $GCD(e, \phi(n)) = 1$

i.e $\phi(n) = (p-1)(q-1)$

Compute d such that $d = e^{-1} \bmod \phi(n)$.

or $ed = 1 \bmod \phi(n)$.

Public key : (n, e) Private key : (p, q, d)

Sig(m) :

$$S = (H(m))^d \bmod n$$

sender sends (m, S) to verifier

Verify (m, S,

if ($S^e \bmod n == H(m)$)

accept

else

reject

Correctness :

$$S = (H(m))^d \bmod n$$

$$S^e = (H(m))^{ed} \bmod n$$

$$ed = 1 \bmod \phi(n)$$

$$\Rightarrow \phi(n) \mid ed - 1$$

$$ed - 1 = k.\phi(n)$$

$$ed = 1 + k.\phi(n)$$

$$S^e = (H(m))^{1+k\phi(n)} \bmod n$$

$$= H(m) \left[\underbrace{H(m)^{\phi(n)} \bmod n}_{=1 \text{ by Euler's theorem}} \right]^k \bmod n$$

$$= H(m).$$

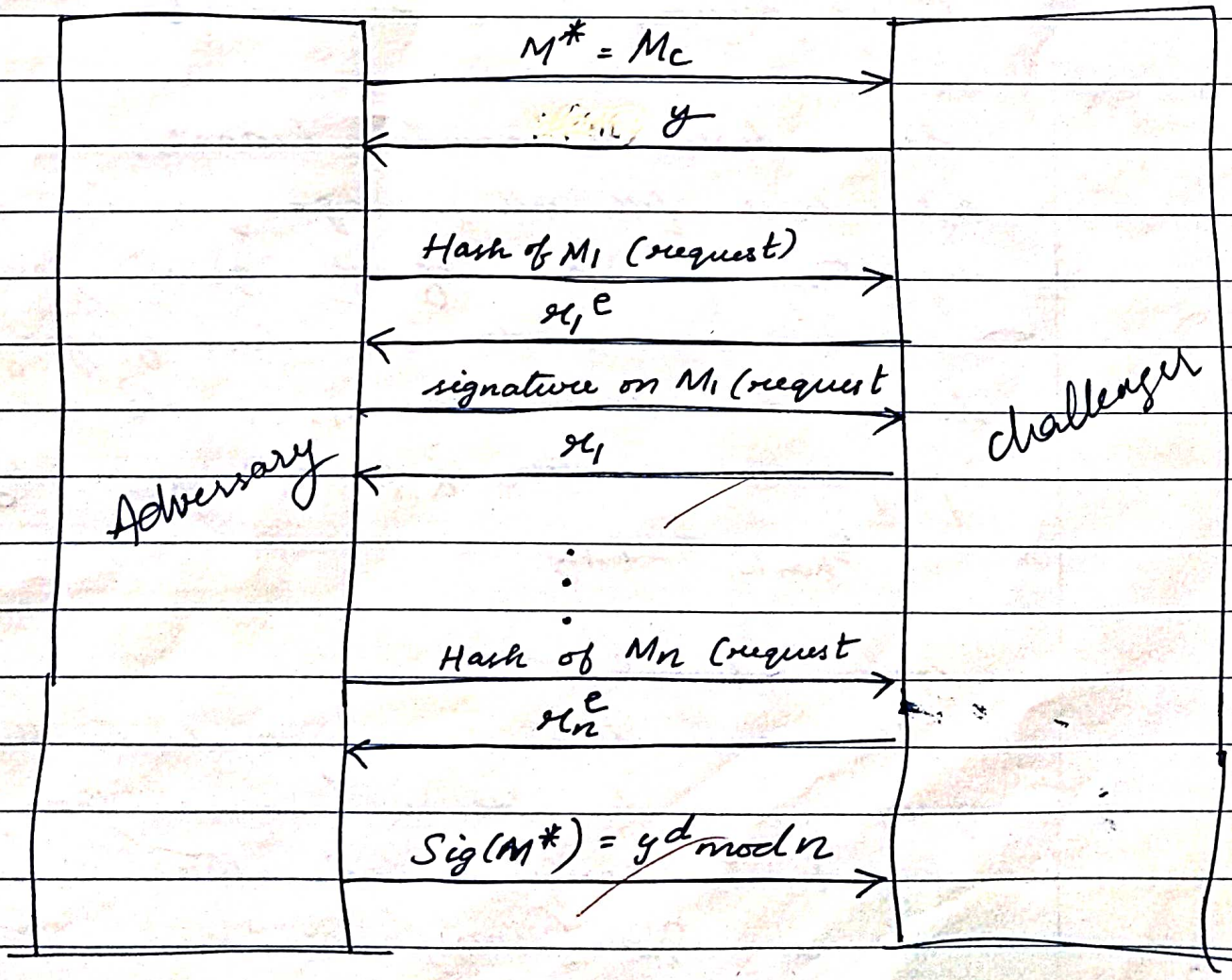
c) RSA is (t', ϵ')-secure means that RSA can be broken (or) RSA-hard problem can be solved in feasible time t' with probability less than ϵ'.

For security purposes

t' should be polynomial

ϵ' should be $\frac{1}{\text{exponential}}$

d)



Assume that adversary can break full domain hash signature

We now try to prove that if adversary can break full domain hash then challenger can solve RSA-hard problem

RSA-hard problem : Given $y = x^e \bmod n$ and e, n, it is hard to find x.

Consider the game shown above ~~Here~~ We use the random oracle model i.e we assume hash is ideal

Since hash is random (ideal), adversary asks hash for all message queries and signature for some of the queries only (as his aim is to do existential forgery to break Full domain hash). ⇒ He asks polynomial number of queries given by

$$q_{hash} + q_{sig}$$

~~Since adver~~

Challenger has to respond to these queries but he does not know the private key d to compute signature S_i on m_i

as $S_i = (H(m_i))^d \bmod n$

(If he knows d, then there is no point in him trying to break RSA as he can simply recompute it using d $\Rightarrow$ not a threat)

~~So.~~ But since $H(m_i)$ is ideal
$S_i = (H(m_i))^d$ is random

So challenger chooses a random number r and computes ~~r~~ $H(m) = r^e$

Now, signature on $m_i = r_i$
Hash of $m_i = r_i^e$

This is because $r_i = (H(m_i))^d \bmod n$
$r_i^e = H(m_i)^{ed} \bmod n$
$= H(m_i)$ ($\because ed \equiv 1 \bmod \phi(n)$)

Also, adversary knows that he has to forge a signature on M_c.
So he asks only the hash for M_c but not the signature

Since challenger knows adversary will forge a signature on M_c, he sends $y = x^e \bmod n$ as ~~the~~ $H(M_c)$.
Since challenger does not know d, he cannot compute $S_{M_c} = (H(M_c))^d \bmod n$

So neither the challenger nor the adversary know the signature S_c on M_c

Now after all these queries, adversary forges signature S_c on M_c and sends it to challenger (since we assumed adversary can break Full domain hash)

$\Rightarrow$ Challenger receives $S_c = y^d \bmod n$

$= x^{ed} \bmod n$

$ed \equiv 1 \bmod \phi(n)$

$\Rightarrow ed = k\phi(n) + 1$

$S_c = x^{k\phi(n)+1} \bmod n$

$= x \cdot (x^{\phi(n)} \bmod n)^k \bmod n$

$= 1$ by Euler's theorem

$= x \bmod n$

$= x$

Thus challenger can break RSA - a hard problem as he found pre image message x from $y = x^e \bmod n$

Probability of challenger breaking the ~~mess~~ RSA = probability of choosing message M_c x probability of adversary breaking full domain hash

$$\epsilon' = P(M_c) \times \epsilon$$

Since M_c is already fixed, adversary knows that he has to forge a signature on M_c only

∴ Probability of adversary choosing M_c = $P(M_c) = 1$

$$\therefore \epsilon' = 1 \times \epsilon$$
$$\epsilon' = \epsilon.$$

Time to break RSA = Time to break Full domain + Time to compute all hashes and signatures by the challenger

$$t' = t + (q_{hash} + q_{sig} + 1) \times O(k^3)$$

$q_{hash} + q_{sig}$ → number of queries on $M \neq M_c$

1 is added because adversary also queries the hash for M_c to

The complexity of 1 round of hash, signature computation = $O(k^3)$
(k - size of key)

$\therefore t' = t + (q_{hash} + q_{sig} + 1) \cdot O(k^3)$

$\Rightarrow t = t' - (q_{hash} + q_{sig} + 1) \cdot O(k^3)$

3.

a) Double spending attack is possible because all nodes are distributed over * the entire globe and is difficult to keep track of the transactions since there is no account-based ledgers.

Blockchain prevents double spending by the following 2 measures

i) places all the blocks in 1 place - the blockchain itself

ii) this allows miners to keep track of all transactions and add a transaction to the chain only if it has unspent coins and valid signatures.

b). Proof of Work consensus is used to prevent people from adding transactions that are useless to the chain thereby ~~so~~ reducing resource wastage

(Eg): $P_1 \xrightarrow{5BTC} P_2$
$P_2 \xrightarrow{5BTC} P_1$
} these 2 transactions could have simply been avoided.

~~As~~ This was possible only because there is no transaction fee to add a block to the chain.

PoW prevents this by introducing a very hard challenge

⇒ Given a hash function H
the challenge is to find a nonce such that hash(nonce) is very small
This nonce will be the nonce of the new block created and added to the chain

(Eg): if hash is SHA-256,
the digest can be between $0, 2^{256}-1$
the challenge is then to find m such that $H(m) < 100$.

This challenge is made hard enough so that both intelligent and average person have the same fair chance of solving it.

To motivate people to solve the challenge, a reward of 6.25 BTC is given which will reduce by half once every 4 years.

1.5

It has the following 2 properties :

1. Random selection - miners are ~~to~~ selected randomly
2. Incentive to add to the longest chain
3. Penalty to add to their ~~ch~~ own chain

c) pragma solidity ^0.4.29

contract OnlineMarket {

address owner

address owner;

int item;

int the price;

int stock;

```
function price (uint
function OnlineMarket () {
    owner = msg.sender;
}
function price (uint newPrice) {


function OnlineMarket () {
    owner = msg.sender;
}
function addItem (uint newPrice,
  uint newItem, uint newStock) {
      if (msg.sender == owner) {
          item = newItem;
          price = newPrice;
          stock = newStock;
      }
}
function price (uint newPrice) {
      if (msg.sender == owner) {
          price = newPrice;
      }
}
}
```

(b) Possible attack : Mishandled exception

If updation of price fails, then the owner is not notified about this exception
⇒ Leads to ~~not~~ vulnerability in chain and fluctuation of other values which can be exploited by malicious people

4. Secret Given two graphs G_0, G_1

secret: G_0, G_1 are isomorphic to each other through the mapping I
i.e $I : G_0 \rightarrow G_1$.

Protocol :

(1) Prover computes $f : G_1 \rightarrow G_2$ ~~such tha~~ and finds graph G_2 such that G_2 is isomorphic to G_1
(this can be done easily by just permuting the vertices of G_1).

Prover sends G_2 to Verifier

(2) Verifier computes $i \in \{0,1\}$ randomly and asks the prover to prove the isomorphism between G_i and G_2 ✓

(3) Prover proves the isomorphism challenge always if he is honest to the verifier

Proof

1) Completeness :

Assume Prover is honest.

⇒ Prover knows $I: G_0 \to G_1$ and

$f: G_1 \to G_2$ ✓

Suppose verifier chooses $i=1$

Prove has to prove isomorphism between G_1 and G_2 ✓

since he computed G_2 from G_1 via $f: G_1 \to G_2$ he can simply return f to the verifier and solve the challenge

suppose verifier chooses $i=0$

⇒ Prover has to prove isomorphism between G_0 and G_2. ✓

Prover knows $I : G_0 \to G_1$ and
$f : G_1 \to G_2$

$\Rightarrow$ he can find $g : G_0 \to G_2$ by using transitivity property.

∴ Prover can return g to the verifier and solve the challenge

∴ Prover (honest) can convince the verifier always with overwhelming probability

2) Soundness :

Assume Prover is dishonest

$\Rightarrow$ If prover does not know $I : G_0 \to G_2$

Case 1 : prover follows the protocol

$\Rightarrow$ Prover knows $f : G_1 \to G_2$

If verifier asks for $i = 1$,
prover can return f and prove isomorphism between G_1 and G_2.

If verifier asks for $i = 0$
prover cannot prove $g : G_0 \to G_2$ because he knows only $f : G_1 \to G_2$ and not

I: $G_0 \to G_1$ and hence he will fail to convince the verifier

$\Rightarrow$ P(dishonest prover convincing verifier) = 1/2

Case 2 : Prover guesses that ~~the~~ verifier will ask for $i = 0$

$\Rightarrow$ he can modify the protocol and generate $f': G_0 \to G_2$ and send G_2 to verifier

So if verifier asks for $i = 0$, prover can convince him

However verifier randomly picks $i \in \{0, 1\}$

$\therefore$ P(verifier choosing $i = 0$) = 1/2

$\Rightarrow$ P(dishonest prover convincing verifier) = 1/2

By repeating this experiment for t times

P(dishonest prover convincing verifier) = $\frac{1}{2^t}$

which is negligible if $t = 100$

$\therefore$ Dishonest prover cannot convince verifier

3) Zero Knowledge : ~~I~~

If there exists a simulator that can simulate the above protocol without any real prover, then no information is revealed

Consider the following algorithm :

i) simulator chooses $i \in \{0,1\}$ randomly and computes $f : G_i \to G_2$ (sends G_2 isomorphic to G_i) and sends G_2 to verifier

ii) Verifier computes $i' \in \{0,1\}$ randomly and asks the simulator to prove isomorphism between G_i and G_2

iii) If $i == i'$

simulator can simply return f and solve the challenge

if $i \neq i'$,

simulator cannot solve it, so we repeat

P(simulator convincing verifier) $= P(i == i')$

$= 1/2$

Thus on average, simulator can simulate the real protocol in just an expected value of 2 trials

If time complexity of protocol is t_1
time complexity of simulation algo is $2t_1$
which is polynomial

Thus we can construct a polynomial time ~~good~~ simulation algo which can simulate the real protocol without even knowing the secret $I: G_0 \to G_1$

$\Rightarrow$ Zero knowledge property is preserved