Reflection Attack :-

Reflection attack in an authentication Protocol :-

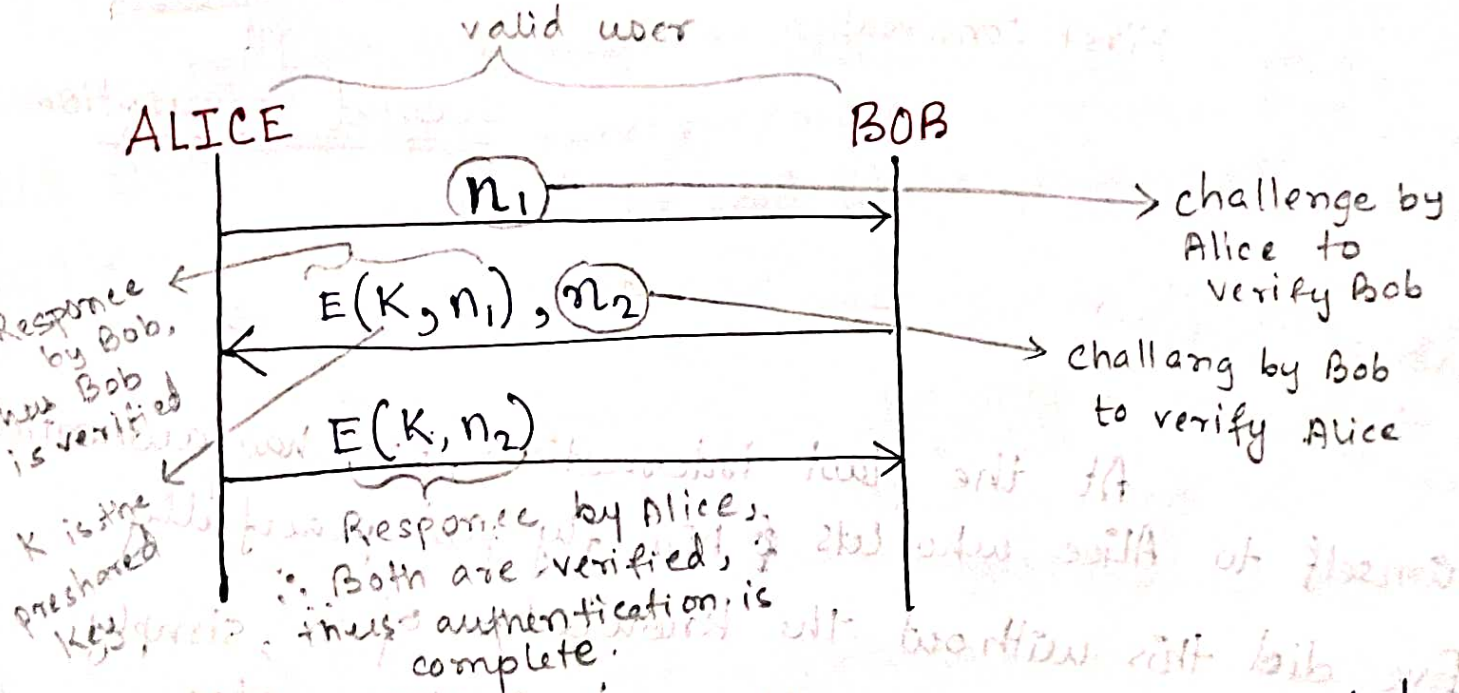
Simple authentication protocols are subjected to reflection attacks. A mutual authentication protocol requires each party to respond to a random challange by the other party by encrypting it with a pre-shared key. In a basic authentication schema, a secret key/value is known to both the valid user and the server. In a reflection attack, the attacker claims to be a valid user and requests the hash of a random value from the server. When the server returns this value and request its own value to be hashed, the attacker opens another connection to the server. This time the hash requested by the ~~user~~ attacker is the value the server requested in the first connection. When the server in the second connection returns this hash value, the attacker reflects this

value to the server in the first connection, thus authenticating the attacker successfully as the valid user.

Example :-

A military decides to implement a system that allows to tell immediately if an aircraft on their radar is a good or a bad guy.

The system they implemented goes something like,

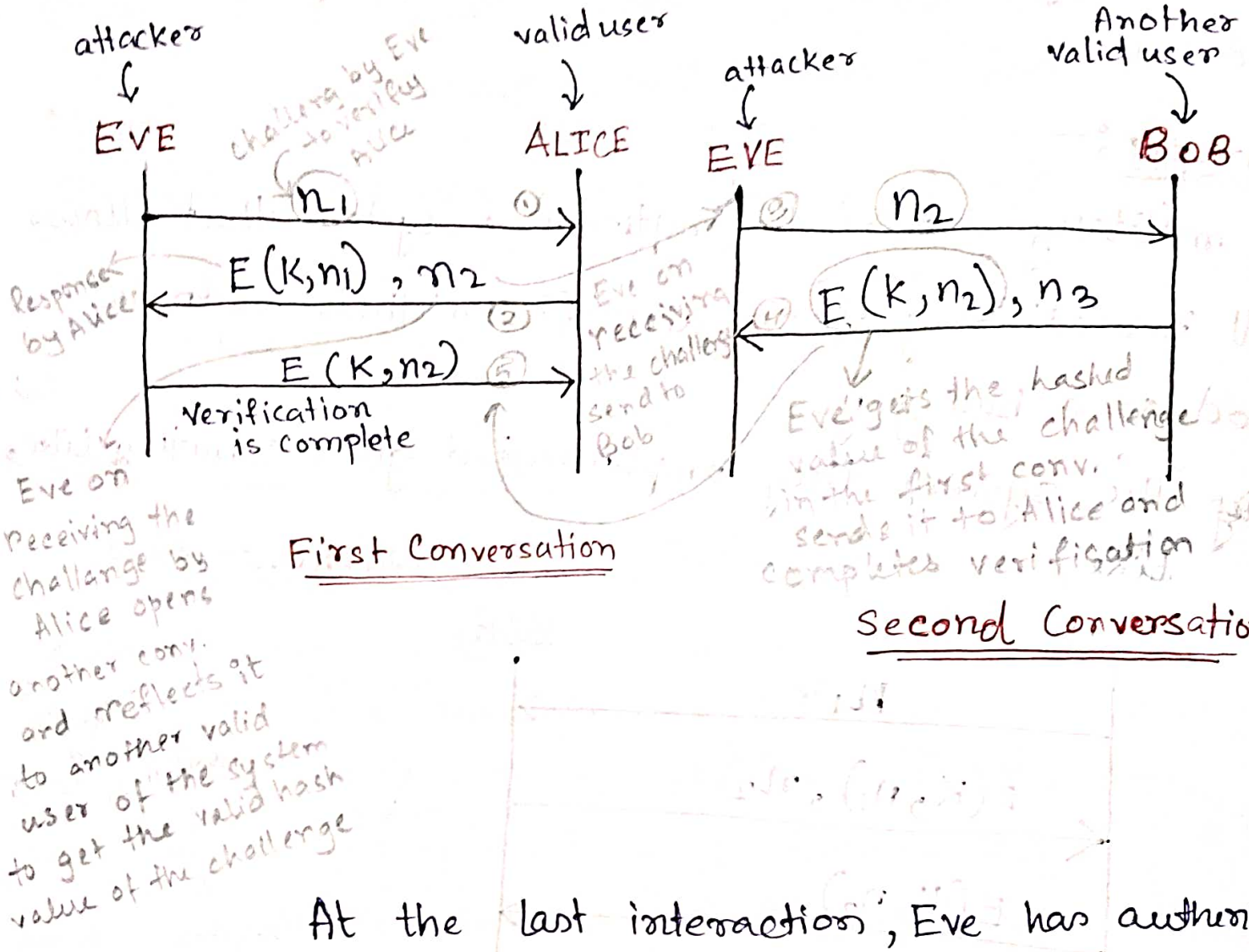


where n_1 and n_2 are unique random challenge generated by Alice and Bob respectively to verify each other. K is the pre shared key, and E is the suitable encryption or decryption function.

Now,

An enemy aircraft can defeat the system by reflecting the challenge back to some other member of the good guys:

shown below the system when reflection is used by an attacker (or intruder).



At the last interaction, Eve has authenticated himself to Alice who lets him fly past peacefully. Eve did this without the knowledge of K, simply by sending the same challenge to someone else. Eve reflected the challenge to Bob, who is another valid user of the system, knows the value of k and can generate the correct response. Bob answers Eve's challange as he normally would do to ~~verify~~ verify himself, thereby giving Eve the correct response to the original challenge.

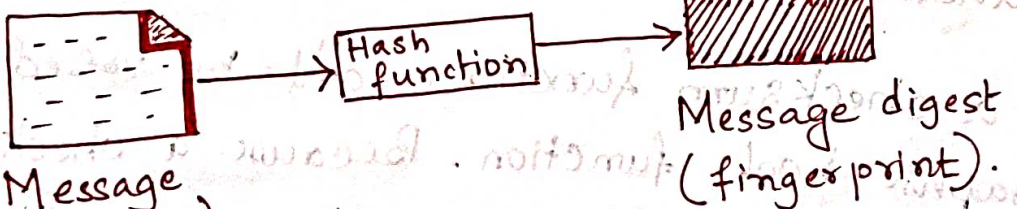
Message Integrity :-

Message integrity means checking the message's authenticity

Message integrity can be described as the concept of ensuring that the data has not been modified or tampered with during transmission. This is typically accomppolished with the use of Hashing algorithm.

Hash function :- (one way function)

To preserve the integrity of a message, the message is passed through an algorithm called Cryptographic hash function. This function creates a compressed image of the message that can be used like a fingerprint for authentication.



Cryptographic Hash function Criteria :-

A cryptographic hash function must satisfy three criteria, They are :-

i) Preimge Resistance.
ii) Second preimage resistance
iii) Collision resistance.

❑ Preimage Resistance :

A cryptographic hash function must be preimage resistance. That is given a hash function 'h' and message digest (or hashed message) $Y = h(M)$. It must be extremely difficult for Eve (intruder) to find any message M', such that $Y = h(M')$.

[If only the hash value is given, and not the message. It should be difficult to find M and M']

Note : If the hash function is not preimaged resistance, Eve can intercept the message digest $h(M)$ and Create a Message M'. Then Eve can send M' to Bob (valid user) pretending it is M. Thus message is changed.

◆ Can check sum function be used as a cryptographic hash function ?

→ No, a checksum function can't be used as a cryptographic hash function. Because a checksum function is not preimage resistant, as Eve (attacker) may find several messages whose checksum matches the given one.

for eg :-
check sum of "Hello Word!" will be same as "World! Hello" or any other combination of the original message. Thus it is easily guessable and not preimage resistant.

Second Preimage resistance:

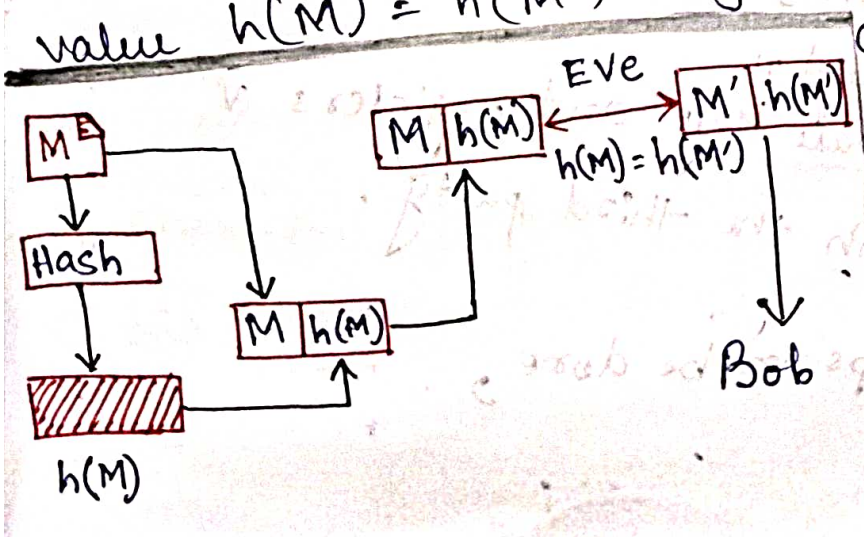
It ensures that a message cannot be easily forged. Alice creates a message (M) and a digest h(M) and sends both to Bob.

Eve (attacker) intecepts the message M and its digest h(M) and tries to create another message $M' \neq M$ such that $h(M) = h(M')$. Eve cannot create another message that hashes to the exact same digest.

In other word, given a specific message and its digest, it is impossible (or at least very difficult) to create another message with the same digest.

Collision Resistance:-

Collision resistance ensures that Eve (attacker) cannot find two messages that hashes to the same value. Collision of a hash function is the event when two messages M and M' such that $M \neq M'$ hashes to the same value $h(M) = h(M')$. A given hash function is said to be collision resistant when it is difficult for Eve (attacker) to find a collision.



Zero Knowledge Authentication :-

In zero knowledge Authentication, the claimant doesn't reveal anything that might endanger the confidentiality of the secret.

The claimant proves to the verifier that she knows a secret, without revealing it.

After exchanging message, the verifier only does or knows that the claiment doesnot have the secret but nothing more.

** In zero knowledge authentication, the claimant proves that she knows a secret without revealing it.

Fiat - Shamir Protocol :-

Here, a trusted 3rd party chooses two large-prime numbers say, p and q to calculate the value of $n = p \times q$. The value of n is announced to the public and the value of p and q are kept secret.

Alice (the claimant) chooses a secret number 's' between 1 and n-1 (exclusive). She calculates,

$$v = s^2 \bmod n$$

she keeps s as her private key and registers v as her public key with the third party.

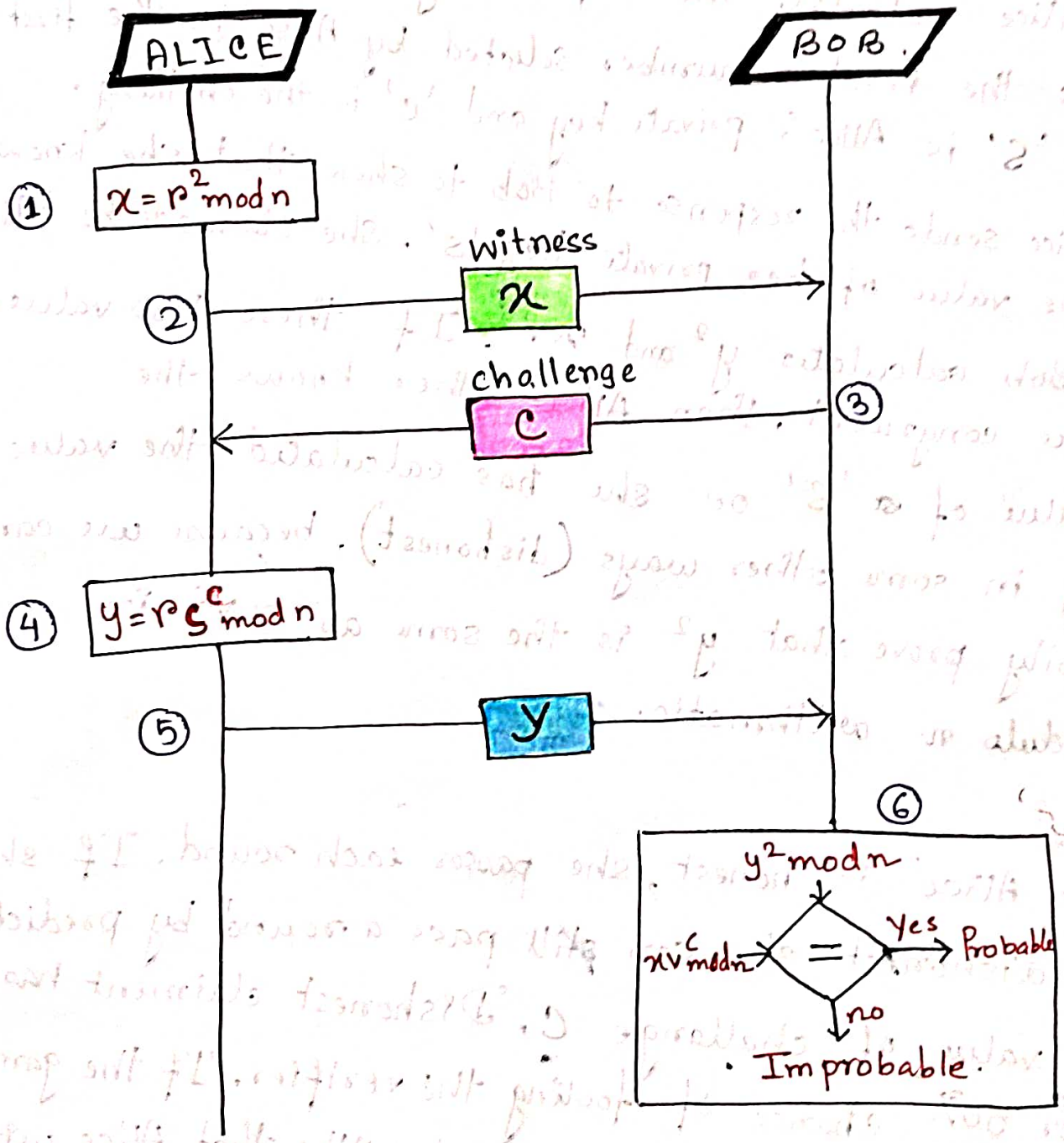
The following steps can be done,

s: Alice's Private key.

v: Alice's Public key.

r: Random number.

n: public



① Alice the claimant, chooses a random number 'r' between

0 and n-1. She then calculates

$$x = r^2 \bmod n$$

x is called the witness

(2) Alice sends x to Bob as the witness

(3) Bob, the verifier, sends the challange c to Alice. The value of c is either 0 or 1

(4) Alice calculate the response $y = rs^c$. Note that 'r' is the random number selected by Alice in the first step 's' is Alice's private key and 'c' is the challang.

(5) Alice sends the response to Bob to show that she knows the value of her private key 's'. She claims to be Alice.

(6) Bob calculates y^2 and xv^c. If these two values are congruent, then Alice either knows the value of 's' or she has calculated the value of y in some other ways (dishonest), because we can easily prove that y^2 is the same as xv^c in modulo n arithmatic.

Note:

If Alice is honest, she passes each round. If she is dishonest, she can still pass a round by predicting the value of challange C. Dishonest claiment has 1/2 or 50% chance of fooling the verifier. If the game is repeated many times. The probability that Alice wins the game is very low. In other words probability $P = (1/2)^N$ where P is the probability of wining without knowing the secret and N is the no. of times the test is run.

So, if the process is repeated say 20 times the overall probability will be $\left(\frac{1}{2}\right)^{20}$ times, which is highly improbable, that dishonest claiment can guess correctly 20 times.

Feige - Fiat - shamir Protocol :-

The Feige - Fiat - Shamir protocol is similar to fiat - shamir protocol except that it uses a vector of private key $[s_1, s_2, \ldots, s_k]$, a vector of public keys $[v_1, v_2, \ldots, v_k]$ and a vector of challenges $(e_1, e_2 \ldots e_k)$. The private key are chosen randomly but they must be relatively prime to n. The public keys are chosen such that, $v_i = (s_i^2)^{-1} \bmod n$.

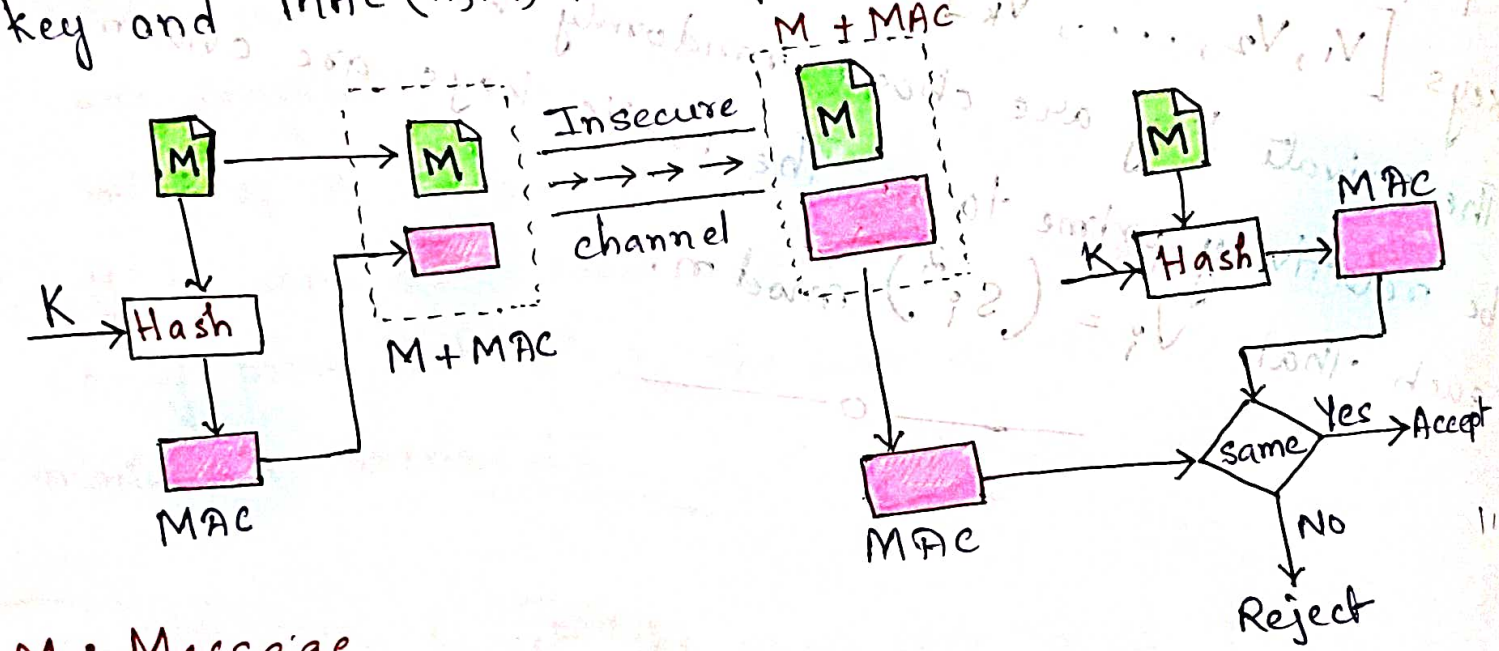
———— o ————

Message Authentication Code (MAC):

To ensure the integrity of the message and the data, that Alice is the originator of the message authentication code (MAC). A MAC also known as a cryptographic checksum, is generated by the function C of the form,

$$T = MAC(K, M)$$

where M is a variable length message, K is a secret key and MAC(K,M) is the fixed length authenticator.



M : Message
K : A shared Secret key
MAC : Message Authentication Code.

Alice uses a hash function to creati MAC with the key and the message she sends the message and the MAC to Bob. Then Bob seperatis the message from the MAC. Bob then makes a new MAC from the shared secret key and message. Bob now compared the newly created MAC with the one received. If the two MAC's matches, the

message is authentic and has not been modified by and adversary.

Security of MAC :- (Collision attack)

Suppose Eve (the adversary) has intercepted the message M and MAC. How can Eve forge a new message without knowing the secret key?

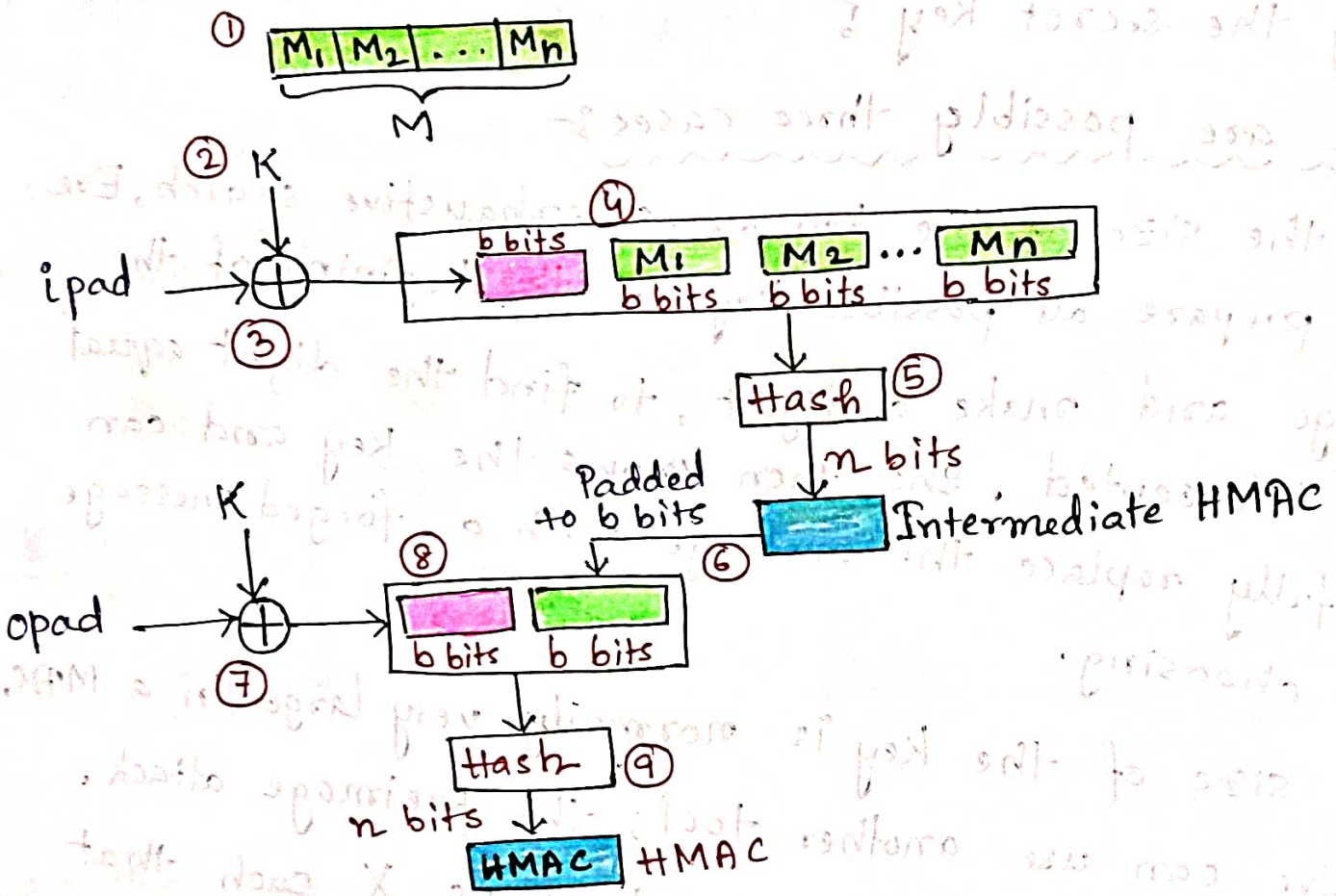
There are possibly three cases :-

- If the size of the key allows exhaustive search, Eve may prepare all possible keys at the begining of the message and make a digest, to find the digest equal to one intereepted. She then knows the key and can successfully replace the message with a forged message of her choosing.
- The size of the key is normally very large in a MAC, but Eve can use another tool; the preimage attack. She uses the algorithm until she finds X such that h(X) is equal to the MAC she has intercepted. She now can find the key and successfully replace the message with a forged one.
- Given some pairs of message and their MAC's, Eve can manipulate them to come up with a new message and its MAC.

Note: The security of MAC depends on the security of the underlying hash algorithm.

Hash-based message authentication code (HMAC).

HMAC is a type of message authentication code (MAC) that is acquired by executing an crytographic hash function on the data that is to be authenticated and a secret key HMAC is used for both the data intigrity and authentication.



1. The message is divided into N-blocks, each of b bits
2. The secret key is left padded with zero to create a b-bit key.
3. The result of step 2 is XORed with a constant called ipad (input pad) to create a b-bits block.
4. The resultant block is prepended (added in the front) to the N-block message.

5. The result of step 4 is hashed to create an n-bit digest.

6. The intermidiate n-bit HMAC is left padded with 0 to make b-bits block.

7. Step 2 and step 3 are repeated by a different constant Opad.

8. the result of step 7 is prepended to the block of step 6.

9. The result of step 8 is hashed with the same hashing algorithm to create the final n-bit HMAC.