

# Denial of service attacks

*Pramod Adiddam 04/27*

## **Network denial of service (Contd.)**

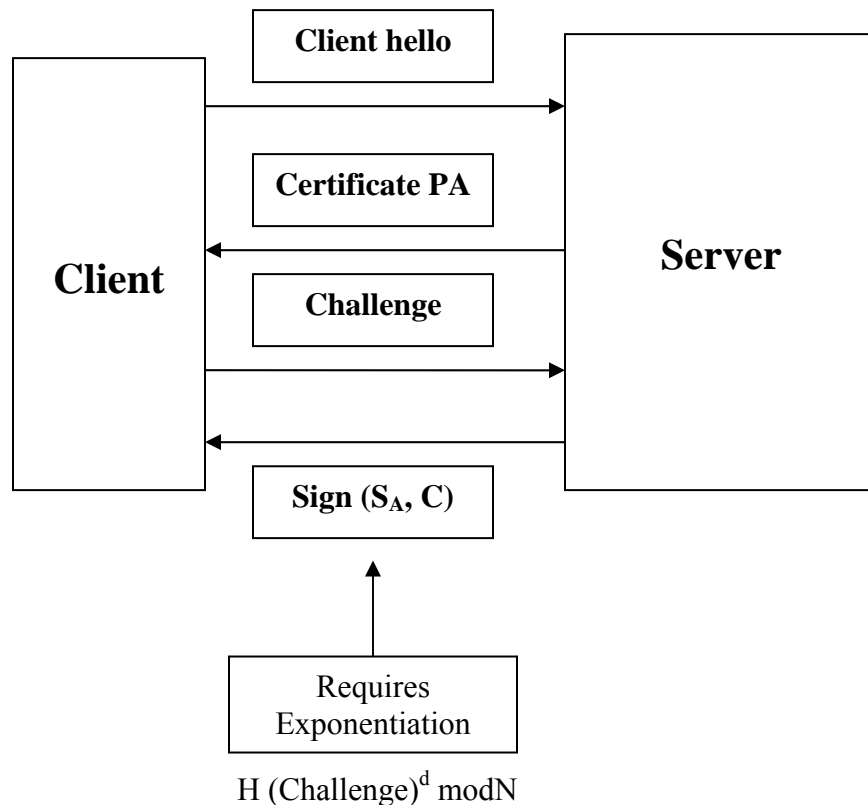
-> Smurf attack – third party broadcast

-> Zombie net - property for the attacker

- huge number of machines (100K zombies)
- attack traffic can be legitimate traffic
- no way to distinguish attackers from legitimate browsers
- CAPTCHAs (degree of sharing not possible otherwise) so other than CAPTCHAs we have very little to go on

## **Computational Denial of service**

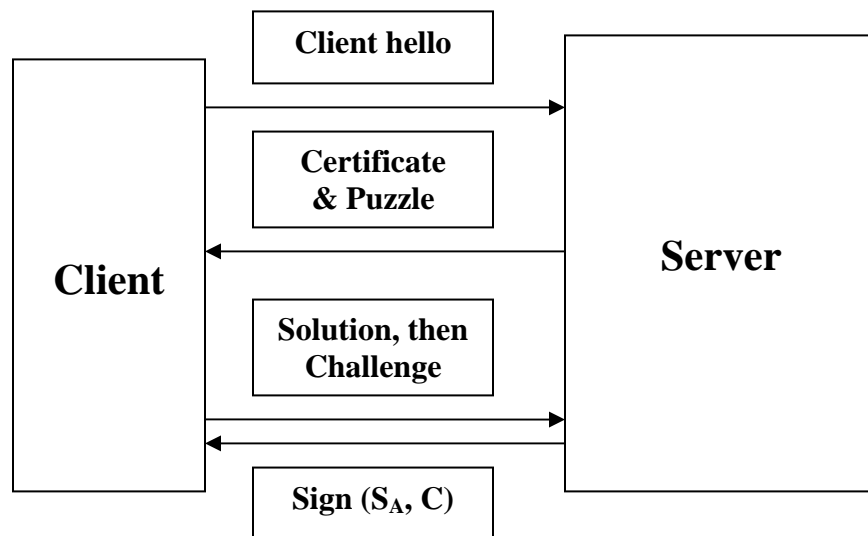
### **Client puzzles and TLS**



d is large and hence there is a huge computation cost on the server side.

Say if the computation cost on the server side is 10ms and the computation cost on the client side is 1ms then the difference in the computation costs between server and client are exploited to overload the server. If client is not computationally close to the server then we can send more requests from a single client or a network of clients more than the server can handle.

To even out the computational costs or fix the computational asymmetry we use client puzzles.

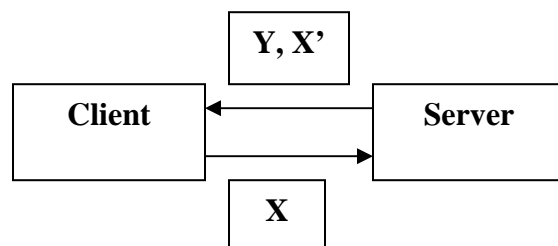


Puzzle should be

- hard to solve (for client)
- easy to generate (for server)
- easy to grade (for server)

Examples of puzzles are:

If  $h(X) = Y$



X' and X differ only in some small LSB bits.

X: 12 32 16 104 16 35 7

H(X):                    X': 12 37 16 104 16 \_\_\_ \_\_

We don't

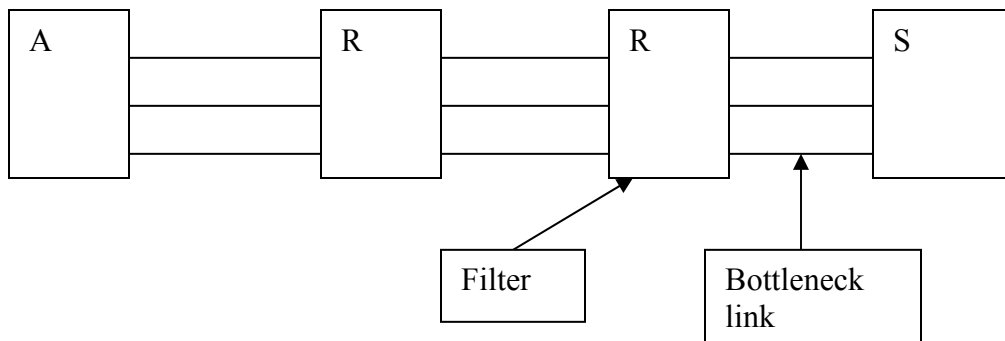
- 1) give the whole set of bits for solving (too long for the client)
- 2) give L bits only i.e., a small X (too small and all possible X and Y pairs can be stored in a lookup table)
- 3) keep the client puzzles mandatory all the time (too expensive for the clients so only turn it on when under attack)

Now if client computation cost is 1000ms and server computation cost is say still 10ms, then to overload the server

- you must force it to perform 100 signs/sec
- attacker must solve 100 puzzles/sec
- attack can be from 100 zombies

## In-network solutions

IP traceback



Flood of traffic

Without faking source address we have

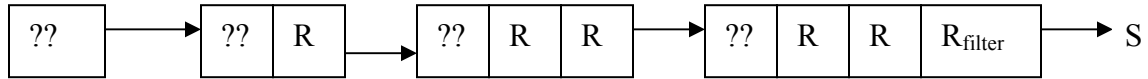
- A-R – not out problem
- R-R – links very high speed
- R-S – overloaded links

where A is attacker, Rs are routers and S is the server.

If A is not faking the address, then if the filter is in R, the traffic will stop at the last R.

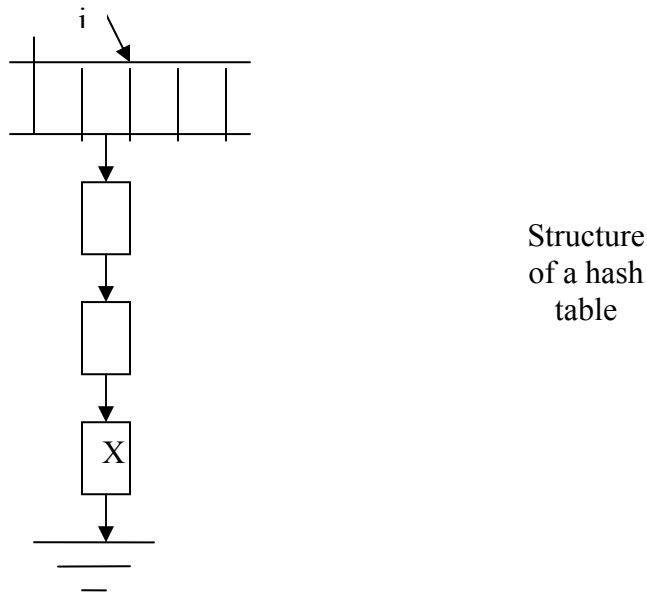
If A is faking the source address then R know what IPs can come, then R stops them (egress filtering) But there is little or no incentive for egress filtering to be implemented at any router since it has to maintain a list of possible addresses

Packet will have the list of routers they are traveling through and hence can be verified for source



Hence an attack can be traced for DOS.

### Hash table attack



```
Insert (X)
{
    i=h(X);
    LinkedListAdd (T[i], X);
}
```

If T contains n cells and we insert n objects the insertion/ lookup costs are O (1) with high probability – this is the hash table theorem

If  $h$  is fixed and known to the attacker, then he can force a lot of collision and send lookups for the last inserted  $X$  continuously to overload the server. Solution would be to use random  $h$  at some point like startup of the program from a family of hash functions  $H$ .

A modified theorem for hash functions is - if  $T$  contains  $n$  cells and insert  $n$  objects and  $h \in H$  over  $R$  and  $H$  is 2-universal then insertion/ lookup costs are  $O(1)$  with high probability. A family of hash function  $H$  is 2-universal if given  $(X, h_i(X))$  for some  $X$ , you have no information about which hash function  $h_i$  is. E.g.  $h(X) = aX + b$