Authentication Protocols

Authentication Protocols

Password Authentication

Concerns:

- An eavesdropper might see the password if sent in the clear
- An intruder might read the password file on the sever
- A password might be easy to guess by an attacker who makes several attempts to login
- A password may be crackable with an off-line guessing attack based on some recognizable item that is encrypted with it.
- Administrators tighten the rules for generating passwords to the point where it is too inconvenient to use a system

Authentication Protocols

Password Authentication

Concerns:

- An eavesdropper might see the password if sent in the clear
- An intruder might read the password file on the sever
- A password might be easy to guess by an attacker who makes several attempts to login
- A password may be crackable with an off-line guessing attack based on some recognizable item that is encrypted with it.
- Administrators tighten the rules for generating passwords to the point where it is too inconvenient to use a system

At UC you can use bearcat1 as a password but
9W+4-1Bfran may not be allowed

Authentication Protocols

Password Authentication

Possibilities:

- Transmit the password in the clear

- Establish a shared secret with Diffie-Hellman, encrypt the Password (but authentication is not mutual)

- Compute a hash of client password, use that to encrypt in challenge/response handshake

  Client <---------- *R* ---------- Server

  Client ----- *f*(password,*R*) ----> Server

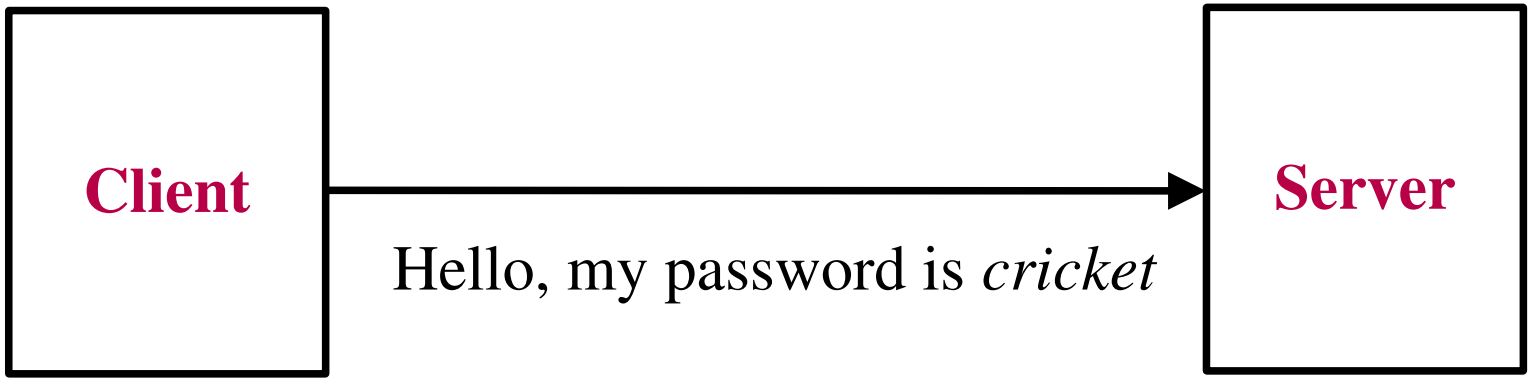
  (dictionary attack on password)

- Use something like Lamport's hash

- Use a strong password protocol

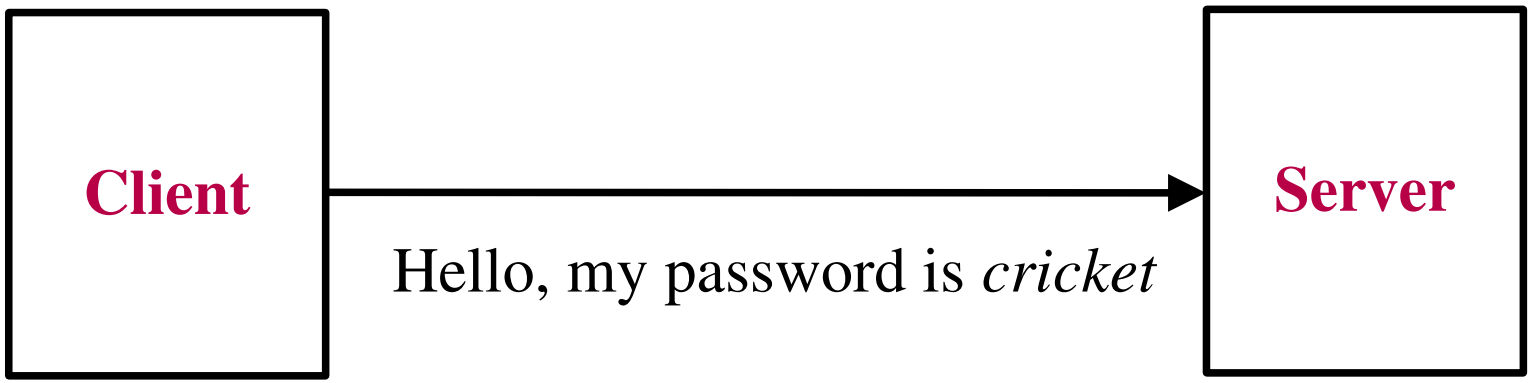
Authentication Protocols

Password Authentication



Authentication Protocols

Password Authentication



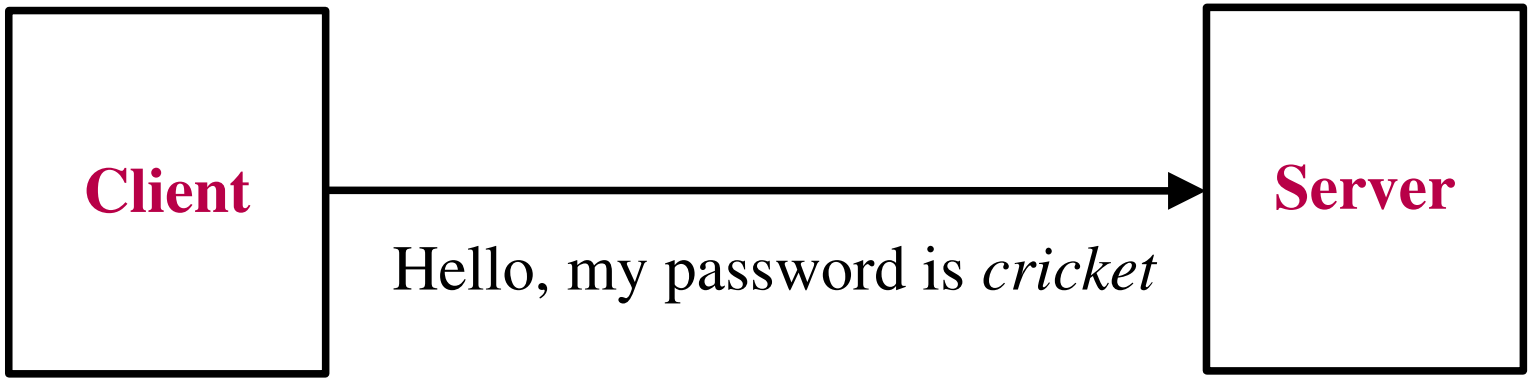
Passwords must be short - imagine a client trying to remember a password like this:

hgsYUW76ewtJHHDKDHJyyeu87362kdfhgodkfhj83hsd838gsb jjhjdj

Passwords must be encrypted - perhaps with RSA which can be used to sign

Authentication Protocols

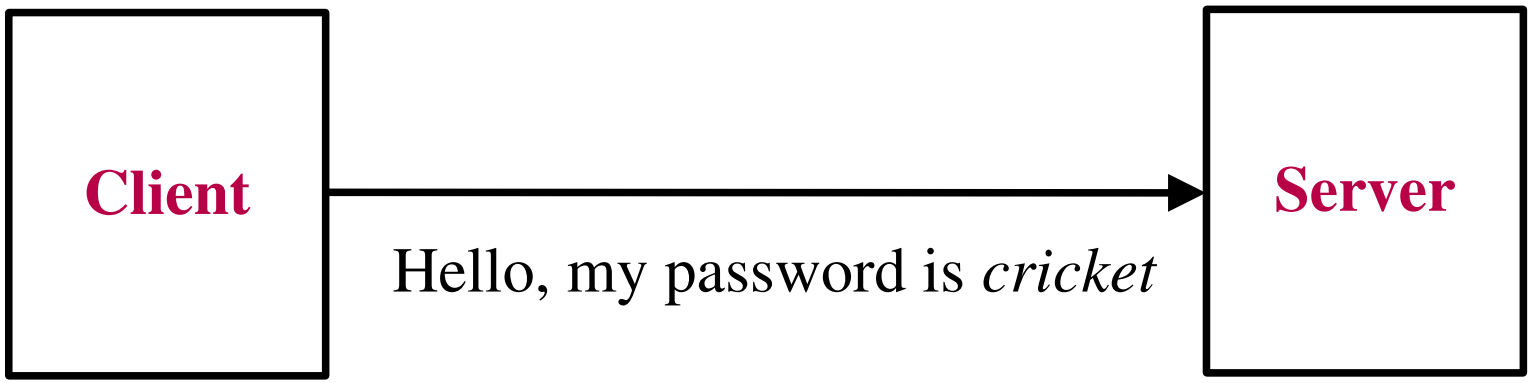
Password Authentication



Protection from on-line attack: Server disables client's account after too many password failures.

Authentication Protocols

Password Authentication

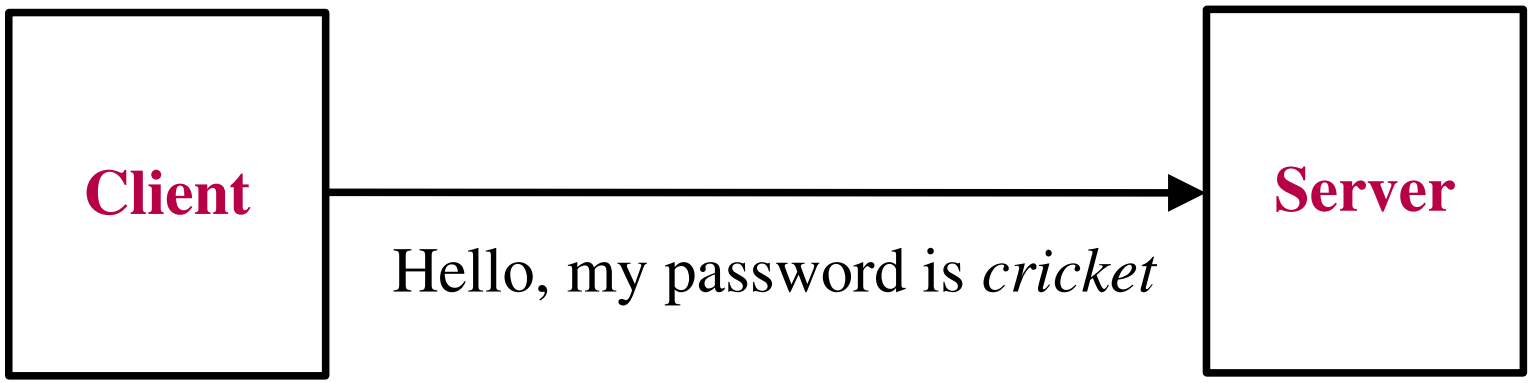


Protection from on-line attack: Server disables client's account after too many password failures.

Harder to protect from off-line attack: Attacker gets some important piece of information which can be used to check the password against. Attacker can repeatedly try passwords until agreement with info is reached.

Authentication Protocols

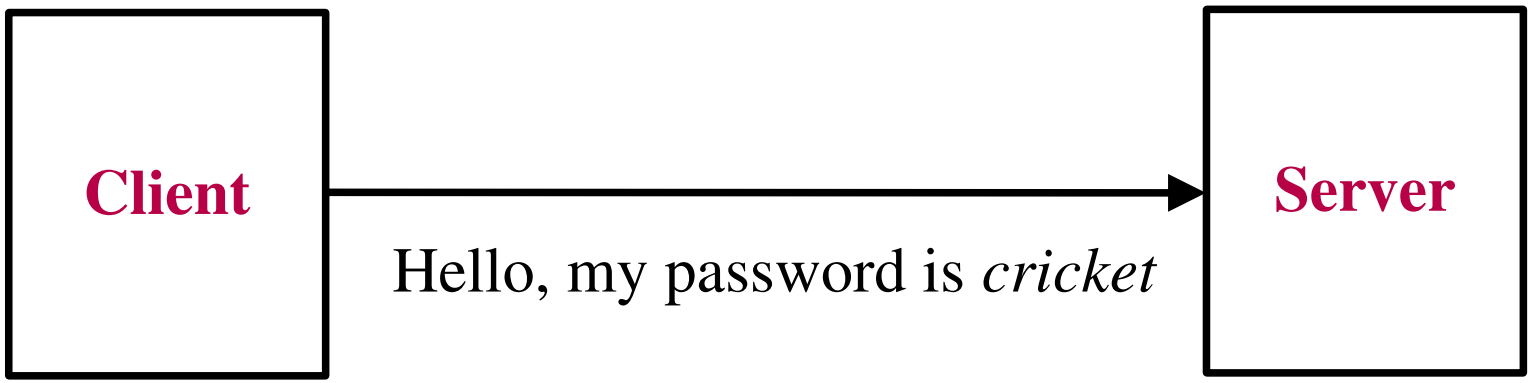
Password Authentication



How to store password information: Cannot just store the password as that is too vulnerable.

Authentication Protocols

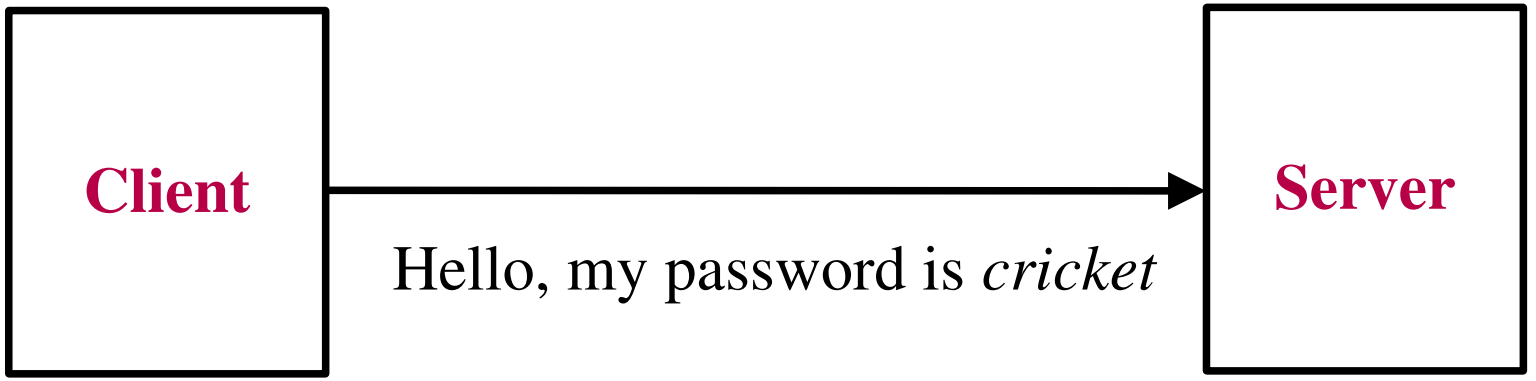
Password Authentication



How to store password information: Cannot just store the password as that is too vulnerable. Perhaps use a hash of the password or encrypt.

Authentication Protocols

Password Authentication

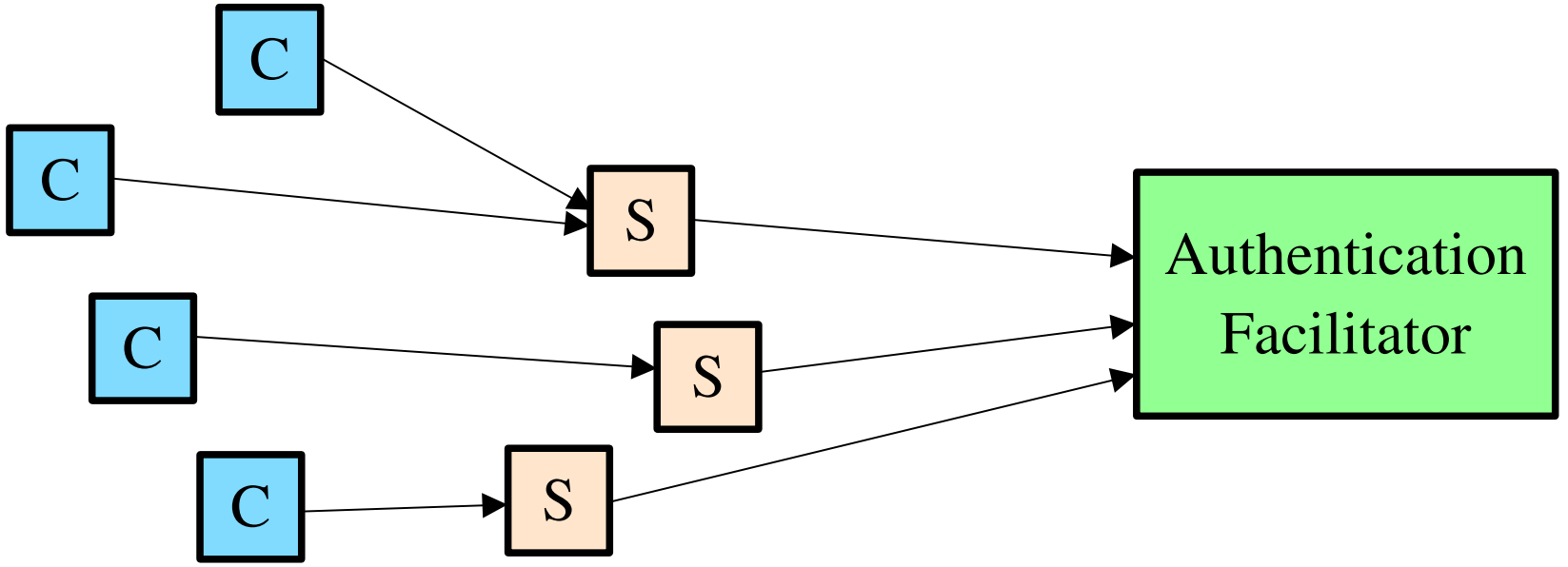


How to store password information: Cannot just store the password as that is too vulnerable. Perhaps use a hash of the password or encrypt.

Should the hash be on every machine client accesses or on one?

Authentication Protocols

Password Authentication



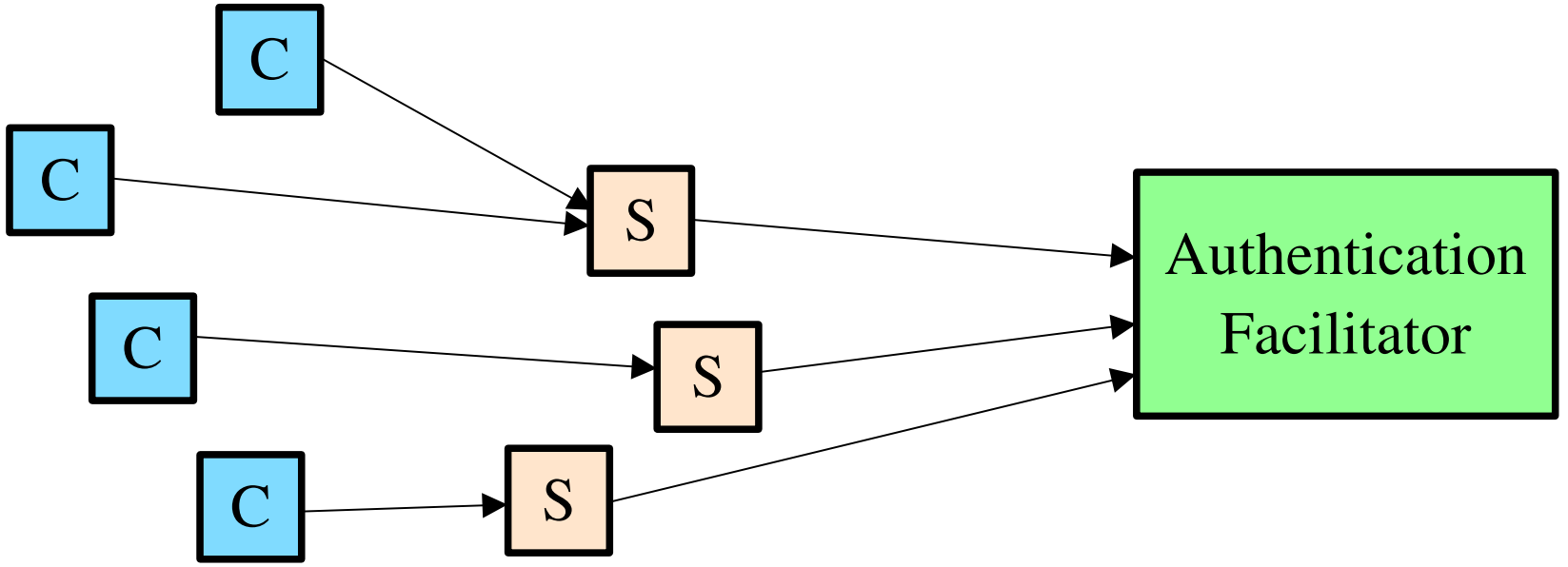
How to store password information: Cannot just store the password as that is too vulnerable. Perhaps use a hash of the password or encrypt.

Should the hash be on every machine client accesses or on one?

If on one machine, should servers retrieve that info when needed or should the authentication be passed onto the ***authentication facilitator***?

Authentication Protocols

Password Authentication



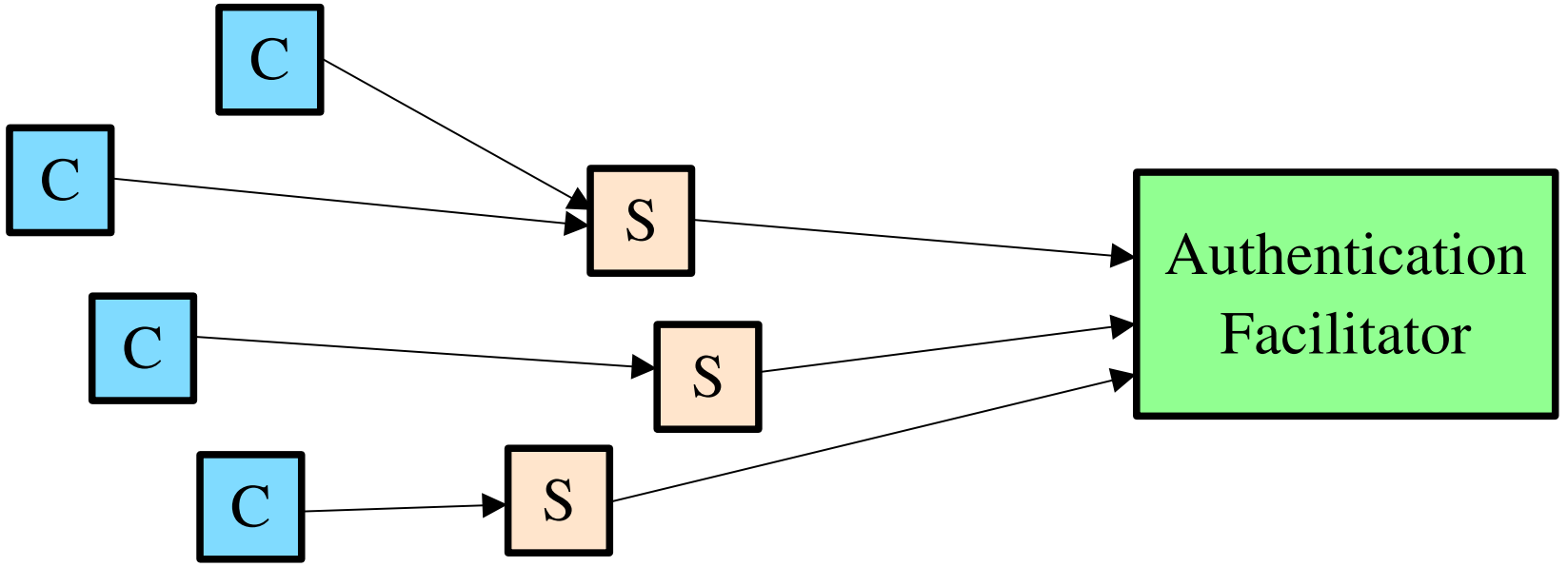
How to store password information: Cannot just store the password as that is too vulnerable. Perhaps use a hash of the password or encrypt.

Should the hash be on every machine client accesses or on one?

If on one, server authenticates the storage node or authentication facilitator

Authentication Protocols

Password Authentication



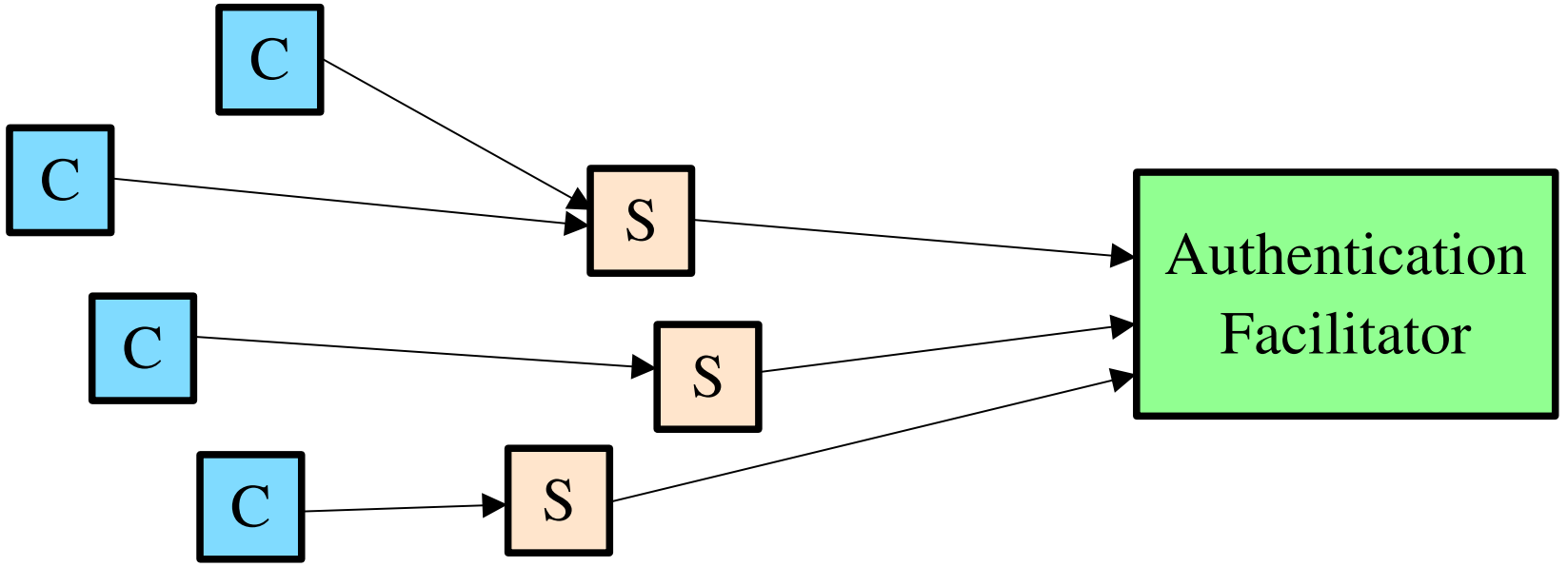
How to store password information: Cannot just store the password as that is too vulnerable. Perhaps use a hash of the password or encrypt.

Should the hash be on every machine client accesses or on one?

If on every server, if passwords are not encrypted then capturing a server's database enables impersonation of all users on that system - maybe on other systems as well if users have the same passwords there

Authentication Protocols

Password Authentication



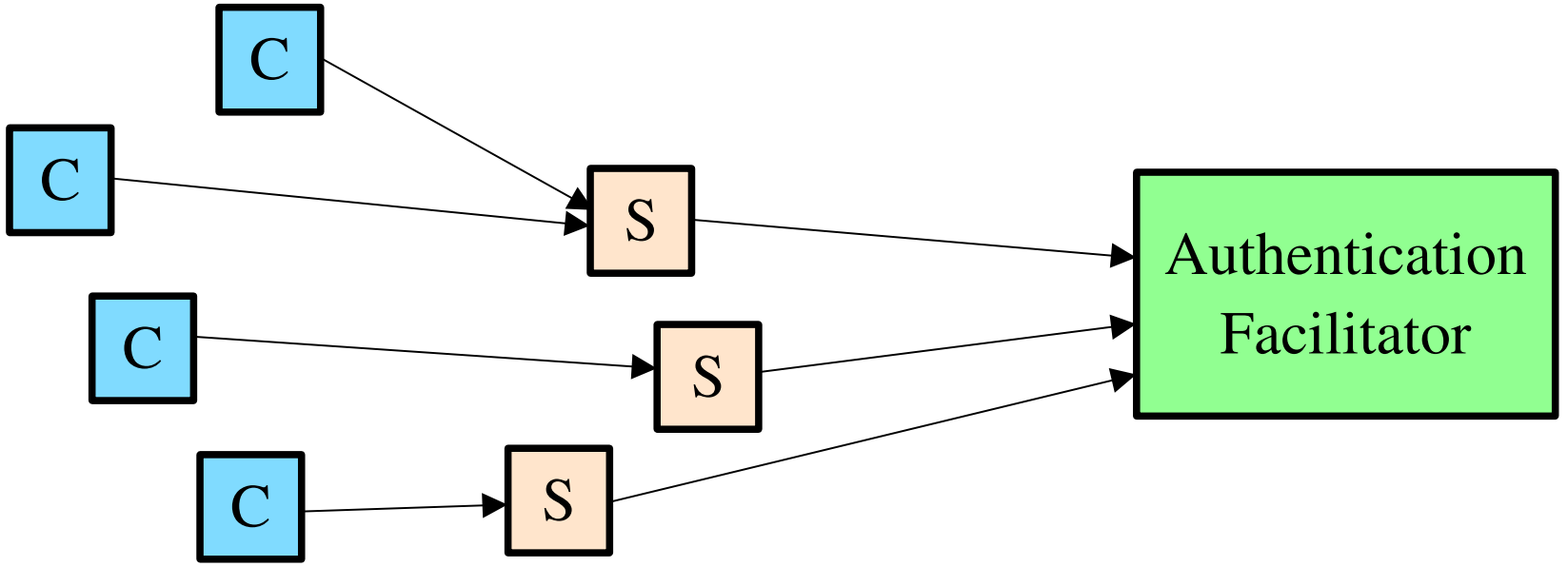
How to store password information: Cannot just store the password as that is too vulnerable. Perhaps use a hash of the password or encrypt.

Should the hash be on every machine client accesses or on one?

Same argument applies to authentication facilitator but it is much easier to protect one machine than all the servers

Authentication Protocols

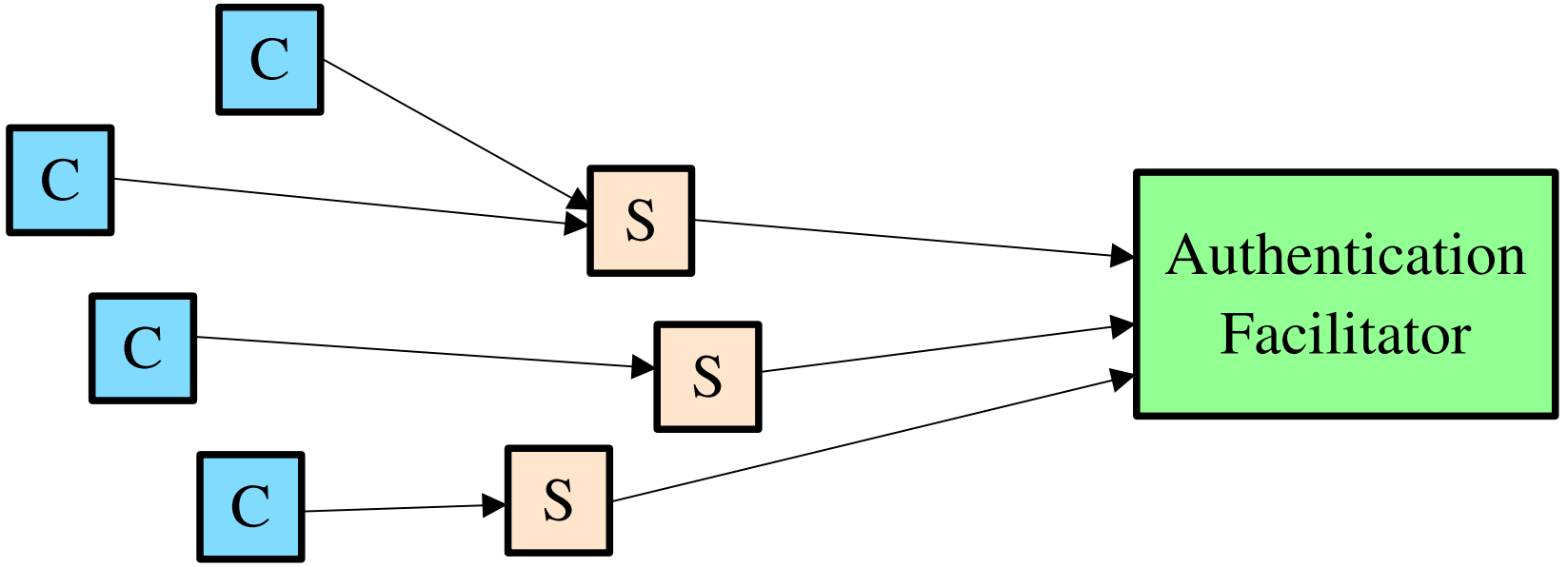
Password Authentication



Encrypt passwords or hash them?

Authentication Protocols

Password Authentication

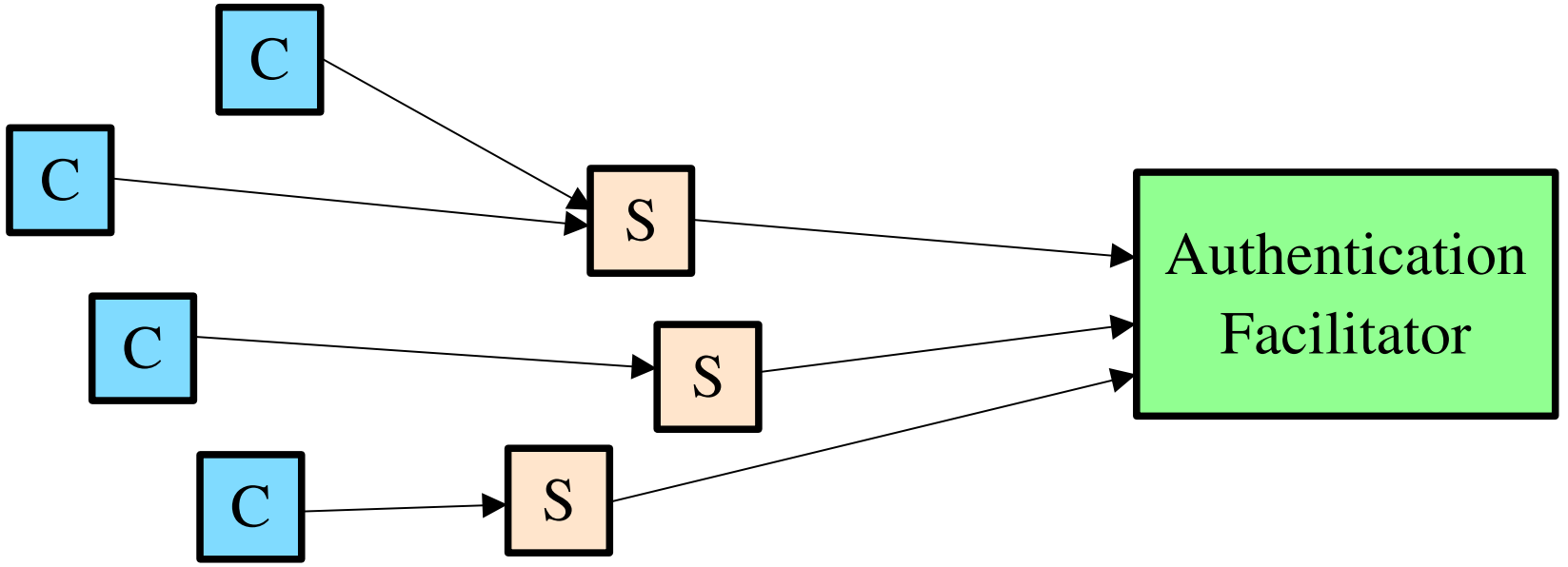


Encrypt passwords or hash them?

Hash subject to off-line attack because we can assume the attacker knows the hash function – check out the encrypt function on OpenBSD.

Authentication Protocols

Password Authentication



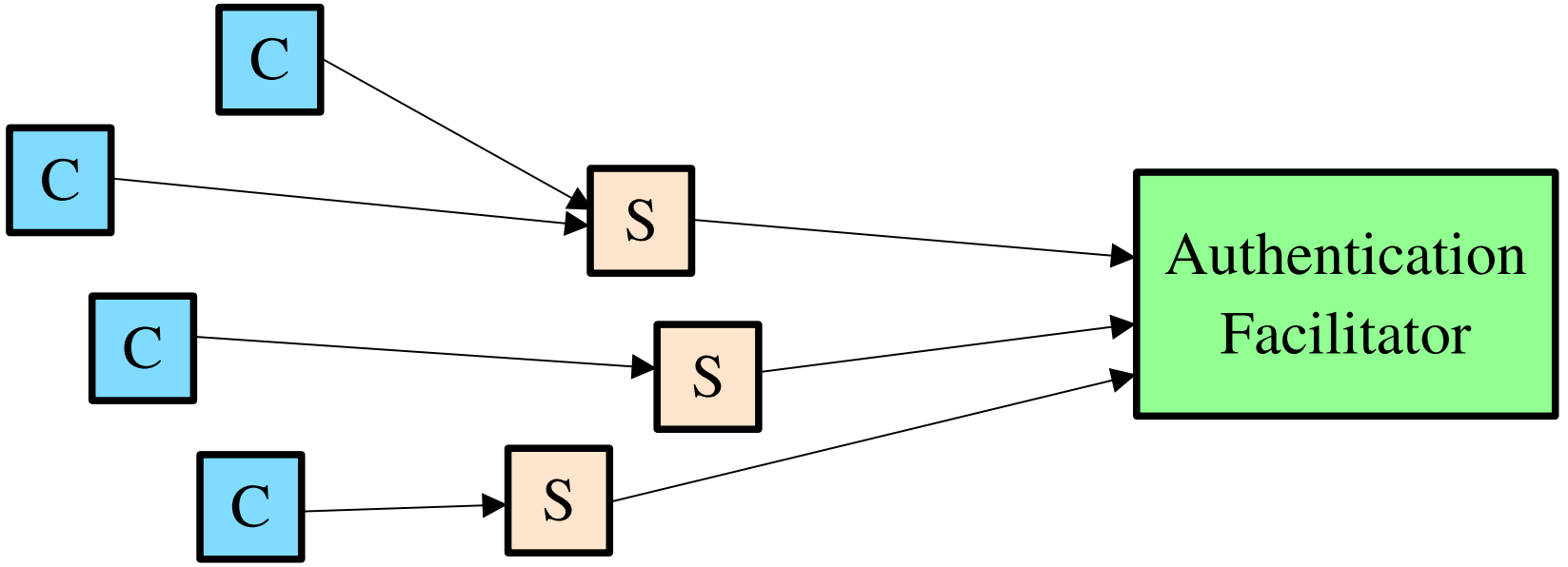
Encrypt passwords or hash them?

Hash subject to off-line attack because we can assume the attacker knows the hash function – check out the encrypt function on OpenBSD.

Encryption based on the authentication facilitator's key which can be huge (because it is a computer, not a person) - hence should be more secure.

Authentication Protocols

Password Authentication



Encrypt passwords or hash them?

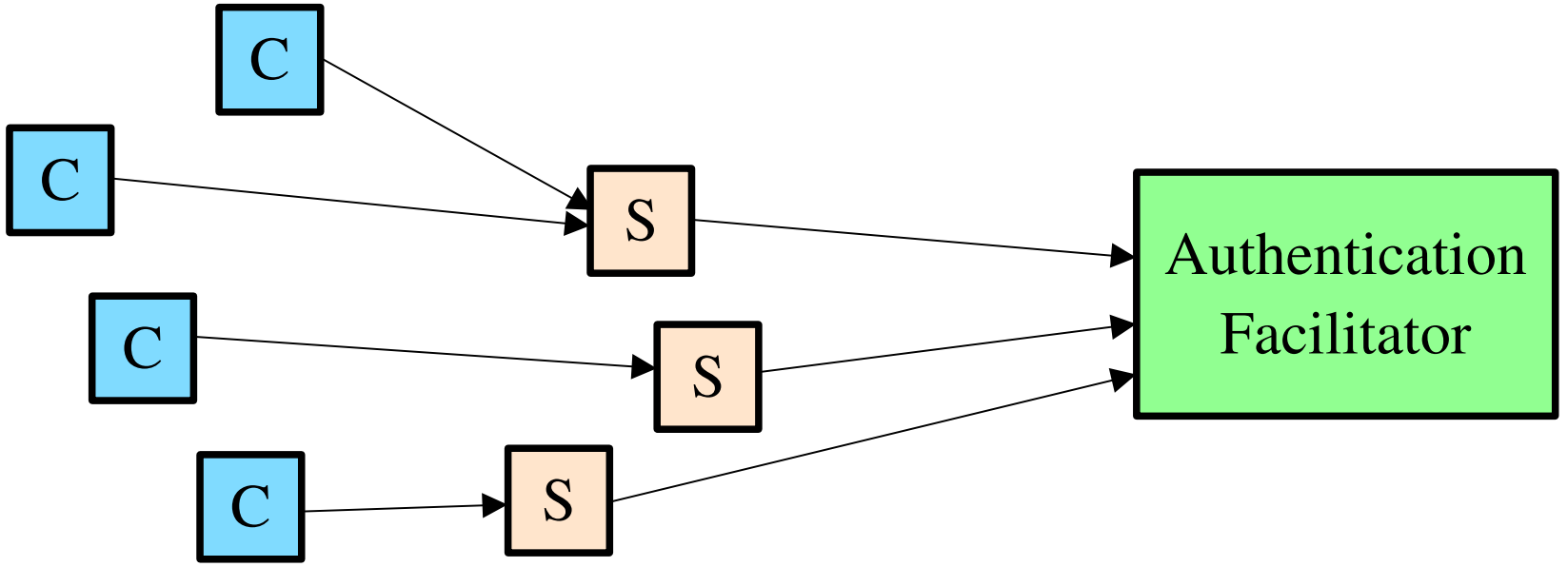
Hash subject to off-line attack because we can assume the attacker knows the hash function – check out the encrypt function on OpenBSD.

Encryption based on the authentication facilitator's key which can be huge (because it is a computer, not a person) - hence should be more secure.

But if the key is found by attacker, then everyone's account is at risk.
If the key is lost then clean-up could be really difficult.

Authentication Protocols

Password Authentication



Encrypt passwords or hash them?

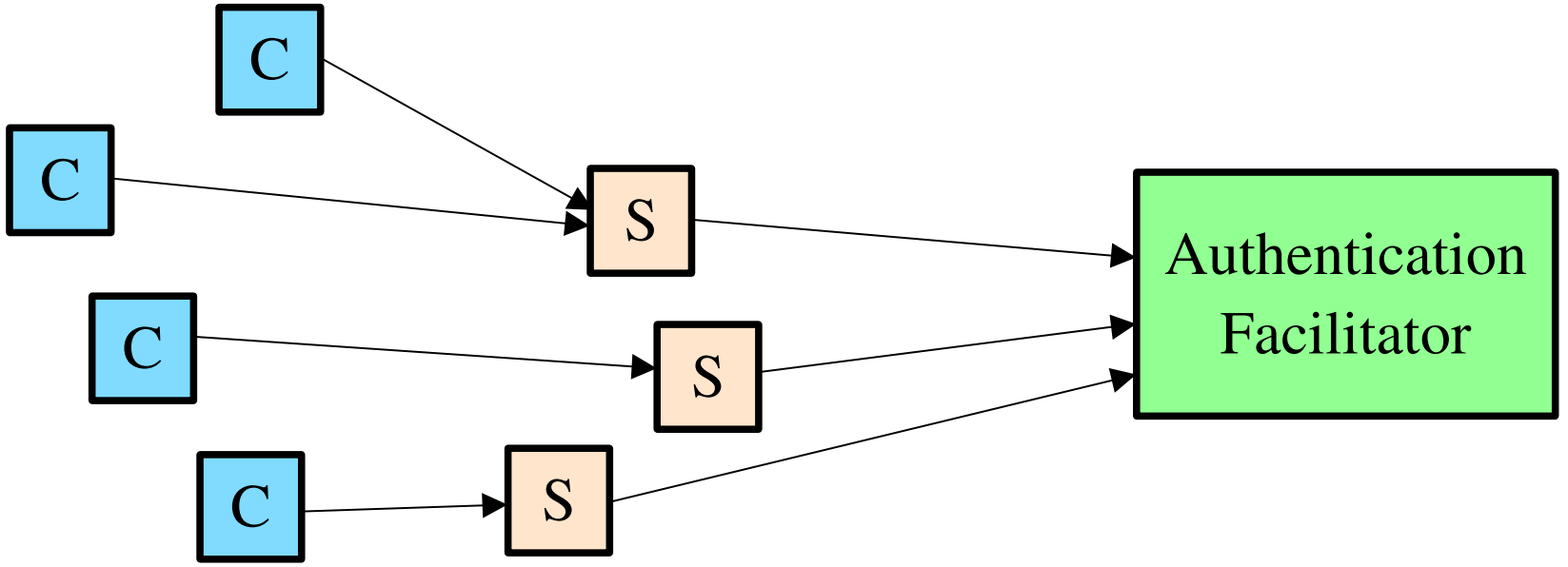
Hash subject to off-line attack because we can assume the attacker knows the hash function – check out the encrypt function on OpenBSD.

Encryption based on the authentication facilitator's key which can be huge (because it is a computer, not a person) - hence should be more secure.

Obvious solution: encrypt a database of hashed passwords.

Authentication Protocols

Password Authentication

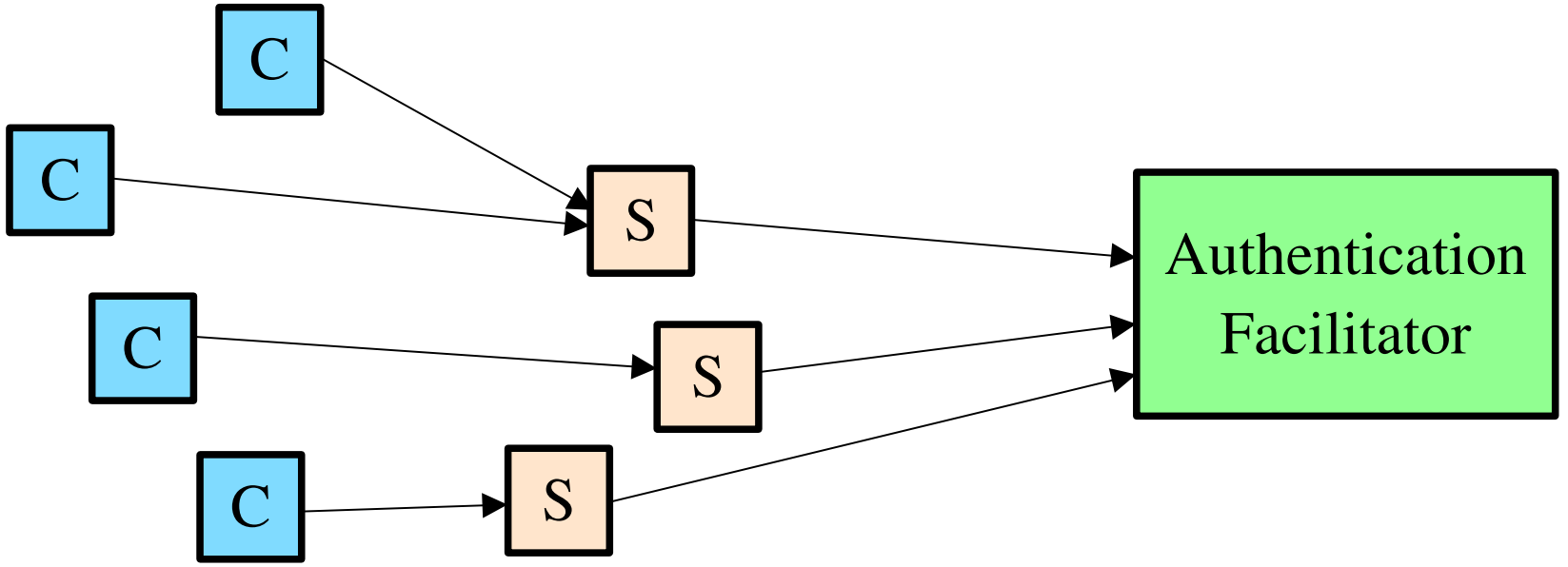


Address-based Authentication

Uses the network address of originator, not a password, to infer the identity.

Authentication Protocols

Password Authentication



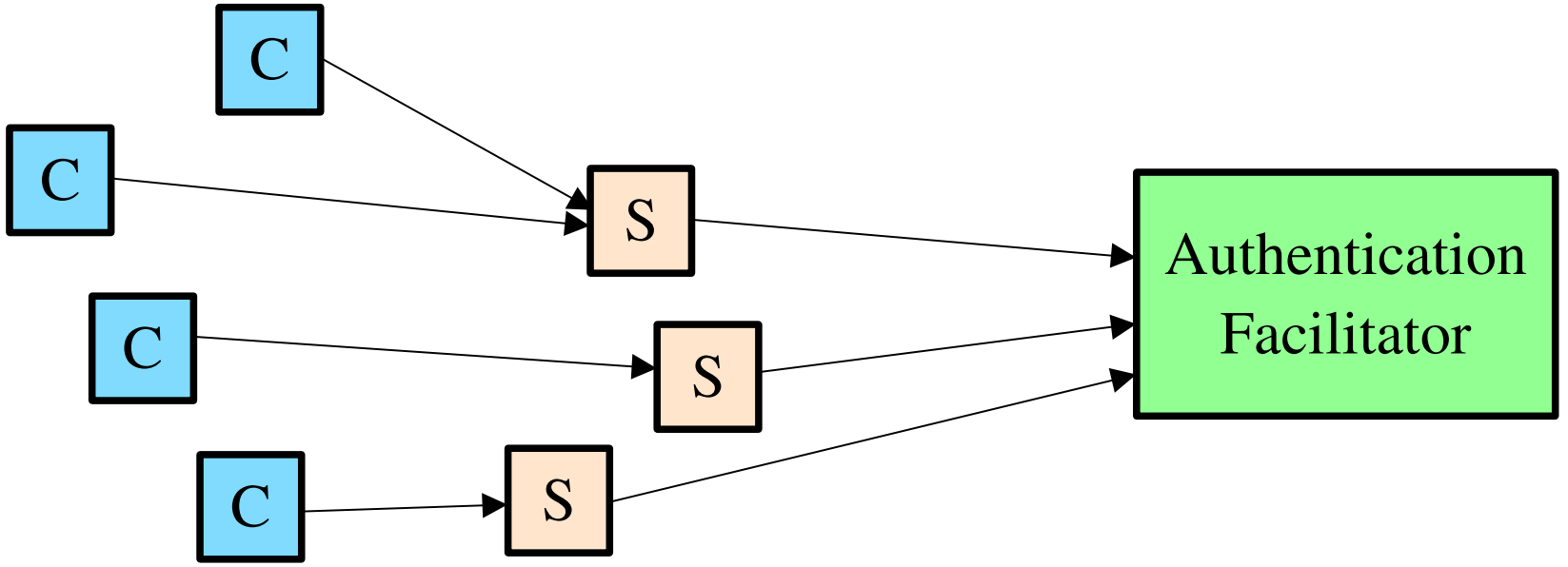
Address-based Authentication

Uses the network address of originator, not a password, to infer the identity.

1. Machine B lists addresses of "equivalent" machines. If machine A is listed then any account name on A is equivalent to one on B (but user must have same account names on both machines)

Authentication Protocols

Password Authentication



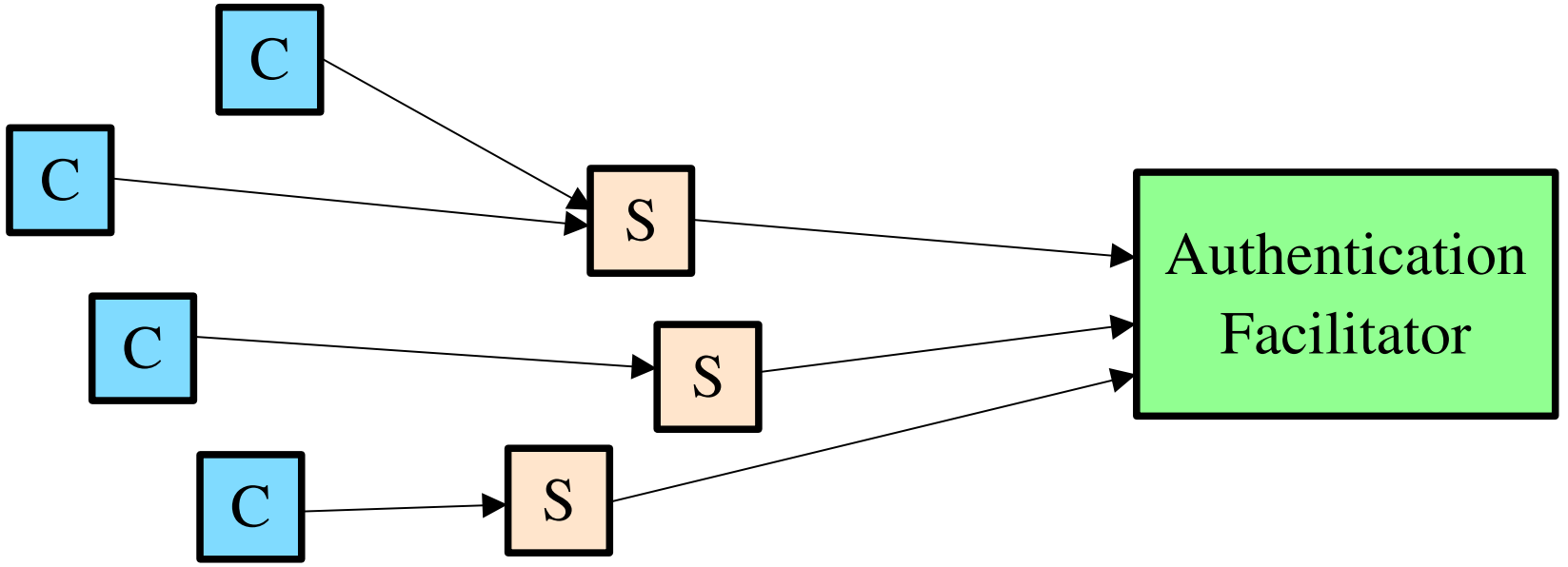
Address-based Authentication

Uses the network address of originator, not a password, to infer the identity.

1. Machine B lists addresses of "equivalent" machines. If machine A is listed then any account name on A is equivalent to one on B (but user must have same account names on both machines)
2. Machine B lists <address, remote-account, local-account> (local authorizes)

Authentication Protocols

Password Authentication



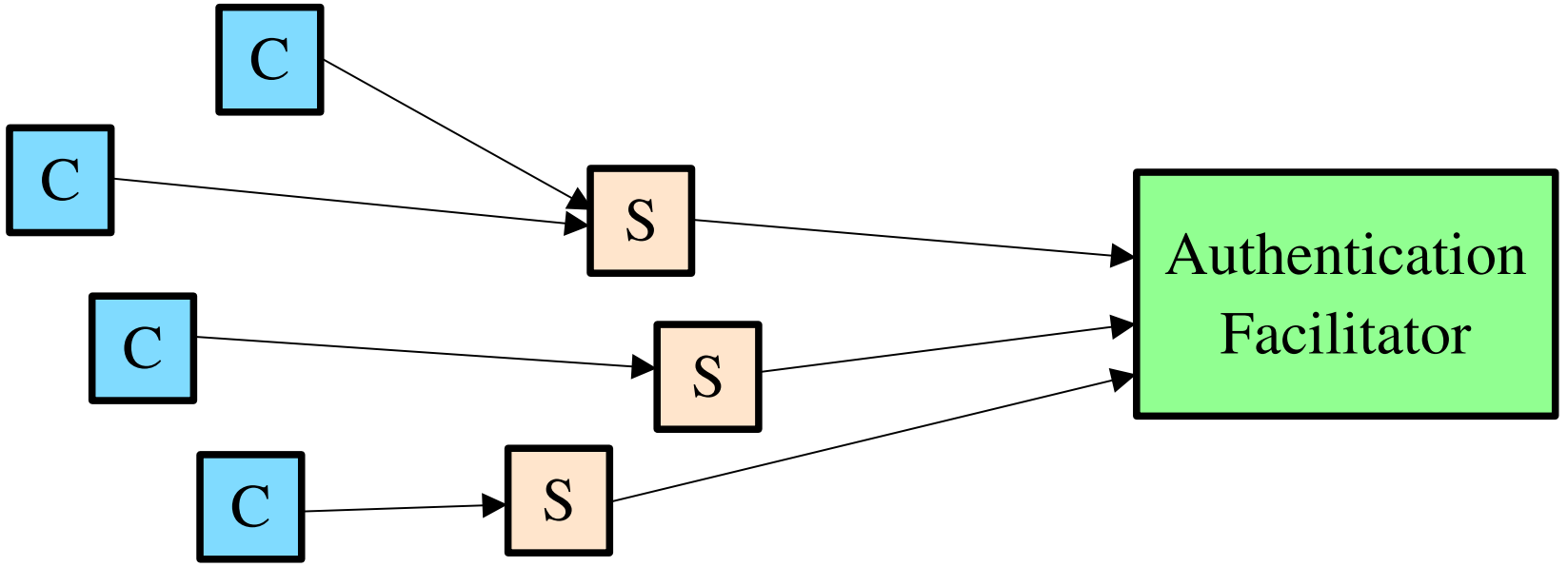
Address-based Authentication

Uses the network address of originator, not a password, to infer the identity.

1. Machine B lists addresses of "equivalent" machines. If machine A is listed then any account name on A is equivalent to one on B (but user must have same account names on both machines)
2. Machine B lists <address, remote-account, local-account> (local authorizes)
3. Let user decide - `.rhosts` file in user's account <address, remote-account>

Authentication Protocols

Password Authentication



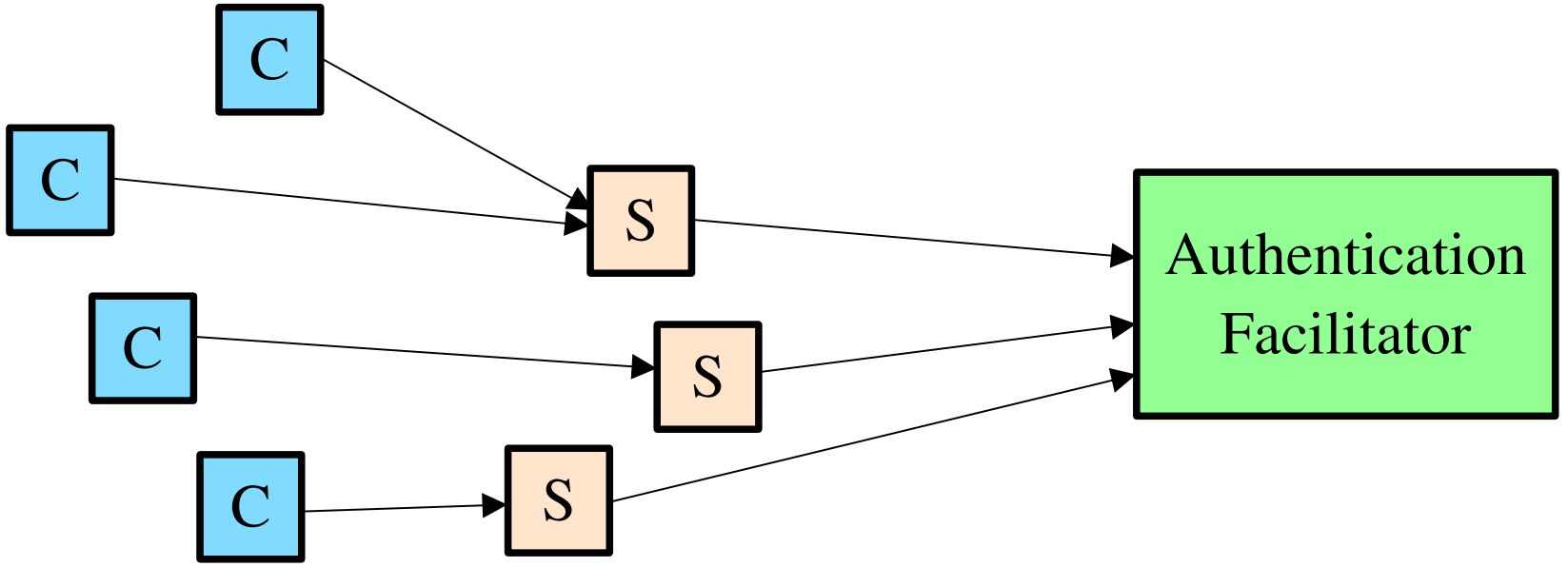
Address-based Authentication

Safe from eavesdropping but has the following problems:

1. If attacker gains superuser access on machine A, it can access network resources of any user with an account on A by getting A to claim the request is from that user.

Authentication Protocols

Password Authentication



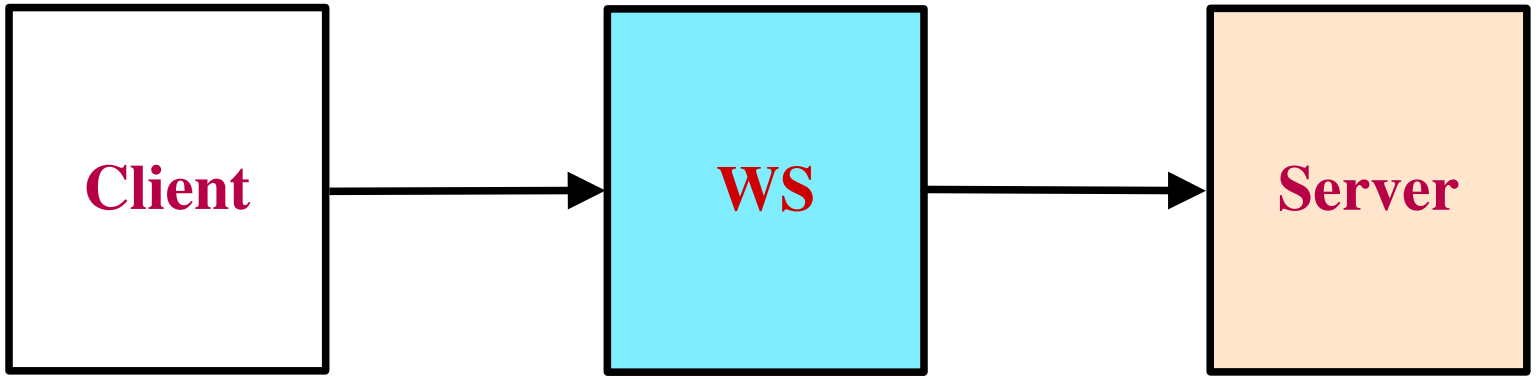
Address-based Authentication

Safe from eavesdropping but has the following problems:

1. If attacker gains superuser access on machine A, it can access network resources of any user with an account on A by getting A to claim the request is from that user.
2. If attacker can impersonate the network address of a machine then all network resources of all users with accounts on those machines are compromised.

Authentication Protocols

Password Authentication

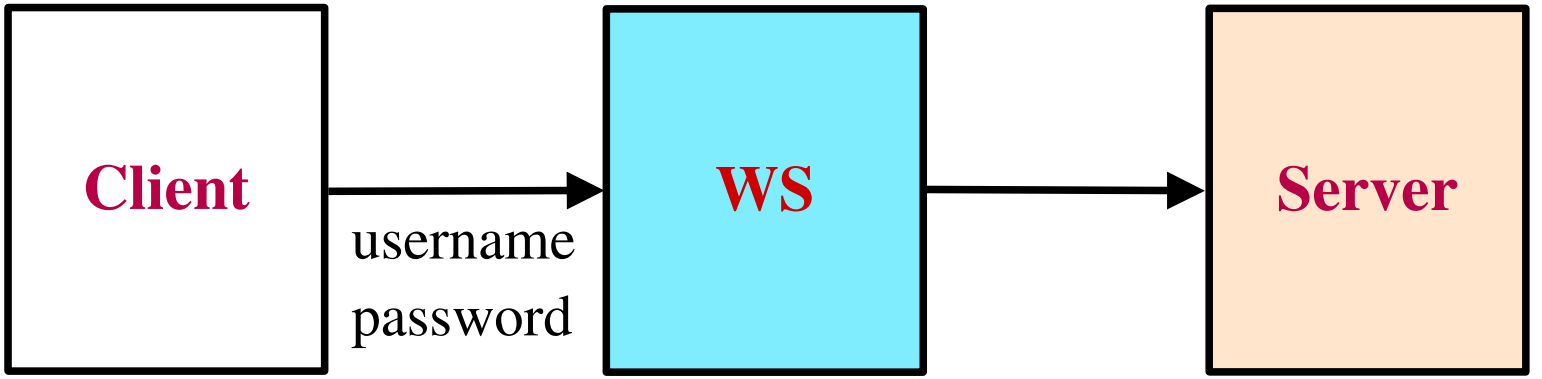


Lamport's Hash (safe from eavesdropping, server attack; no public keys)

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Authentication Protocols

Password Authentication



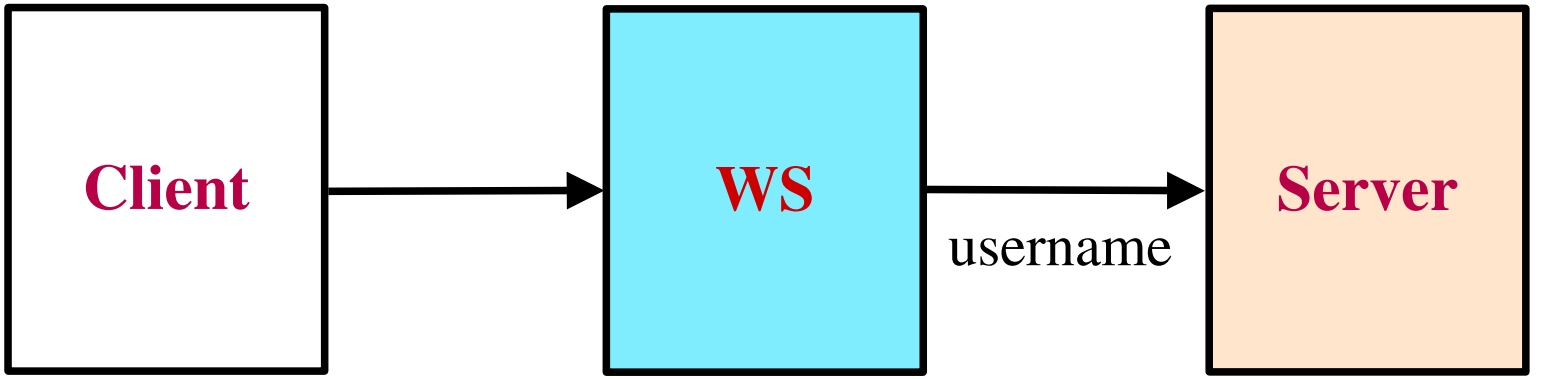
Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Client sends username, password to Workstation

Authentication Protocols

Password Authentication



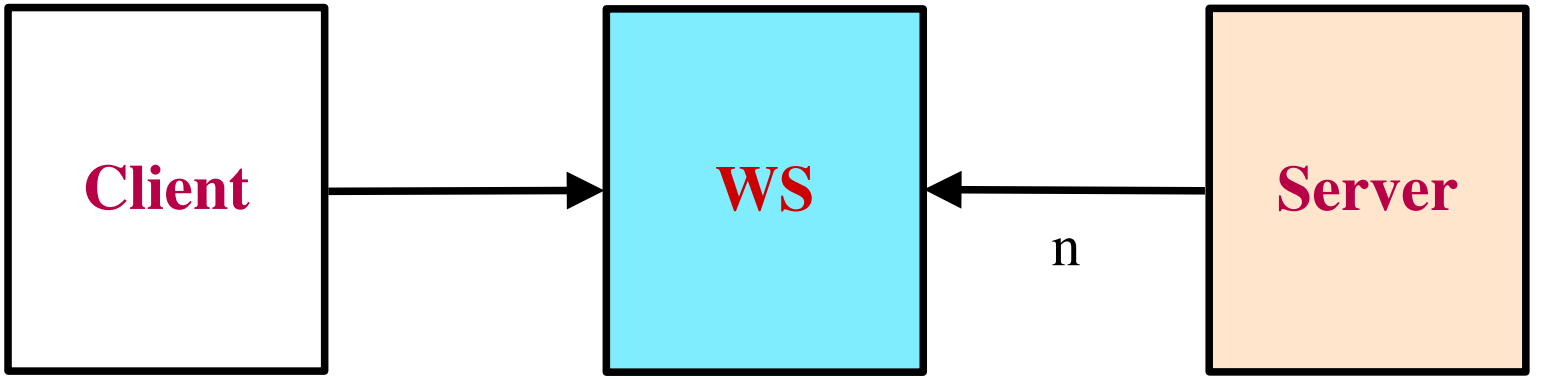
Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Workstation sends username to Server

Authentication Protocols

Password Authentication



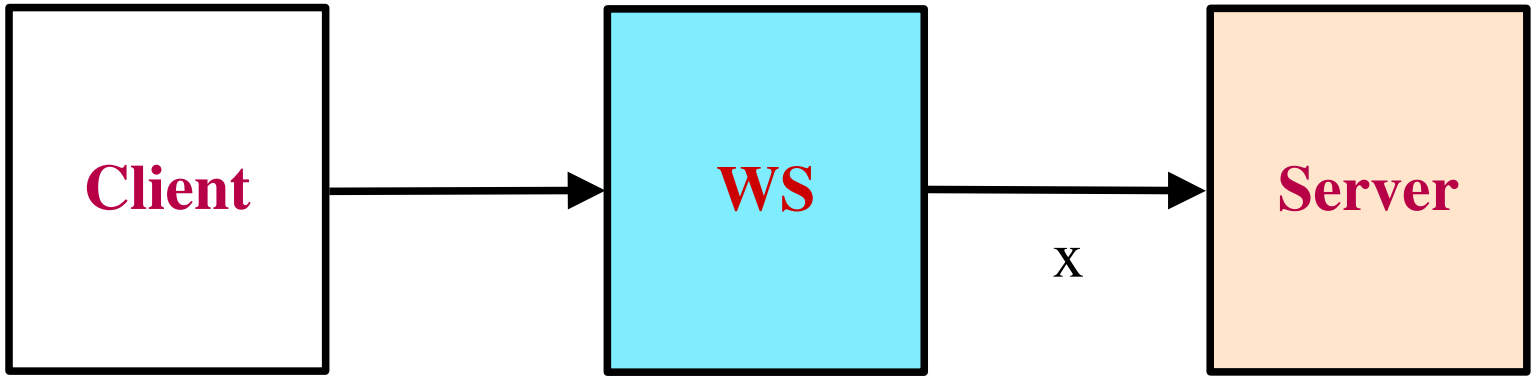
Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Server sends the number n to the Workstation

Authentication Protocols

Password Authentication



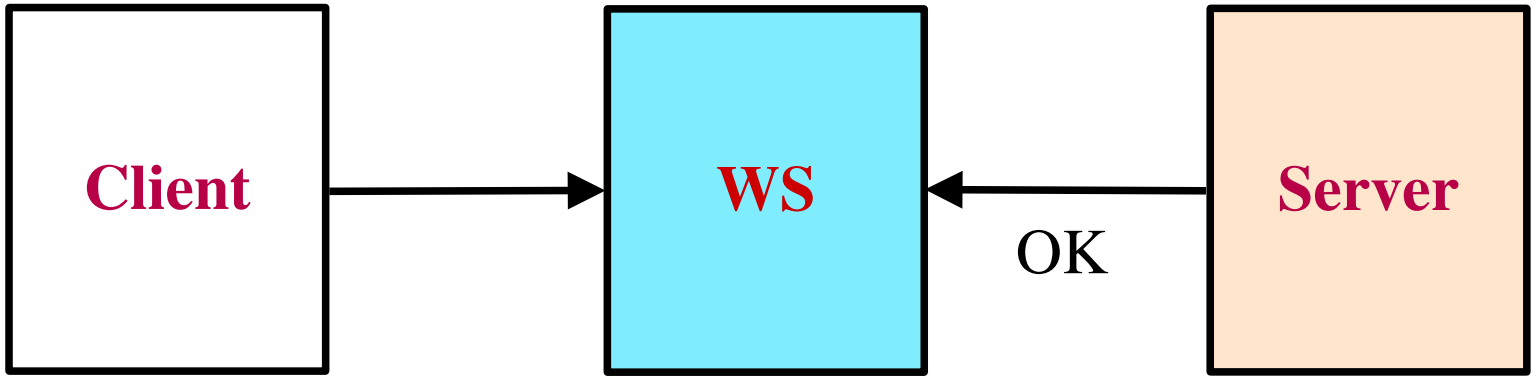
Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Workstation computes x = hash(hash(...n-1 times...(hash(password))...)), sends to Server

Authentication Protocols

Password Authentication



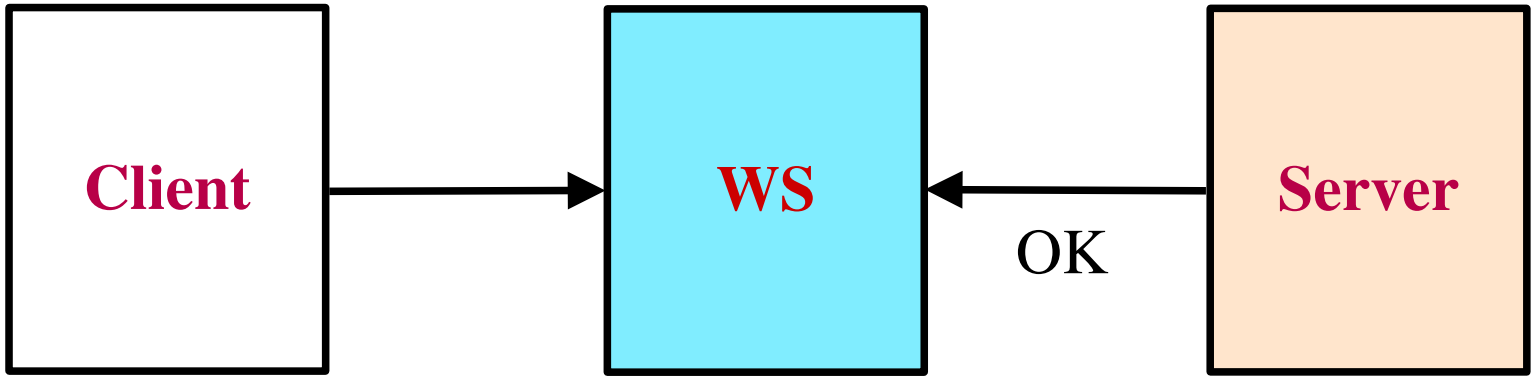
Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Server computes hash(x)...if it matches, authorizes Client to login to Server, replaces database item with x.

Authentication Protocols

Password Authentication



Lamport's Hash

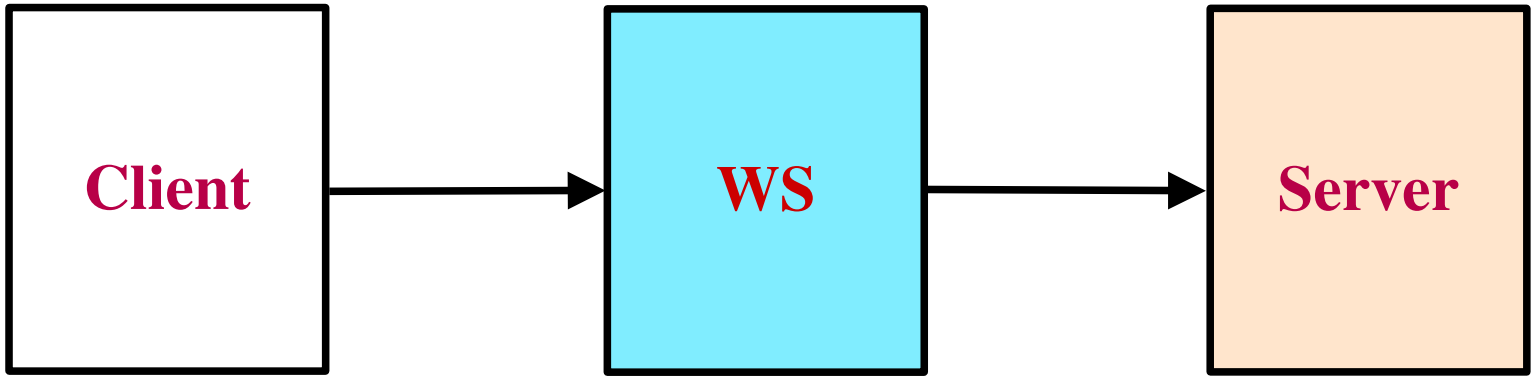
Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password))...))

Two questions: What happens when n goes to 0?

If client talks to several servers, there will be a different n for each – isn't this vulnerable to a replay attack?

Authentication Protocols

Password Authentication



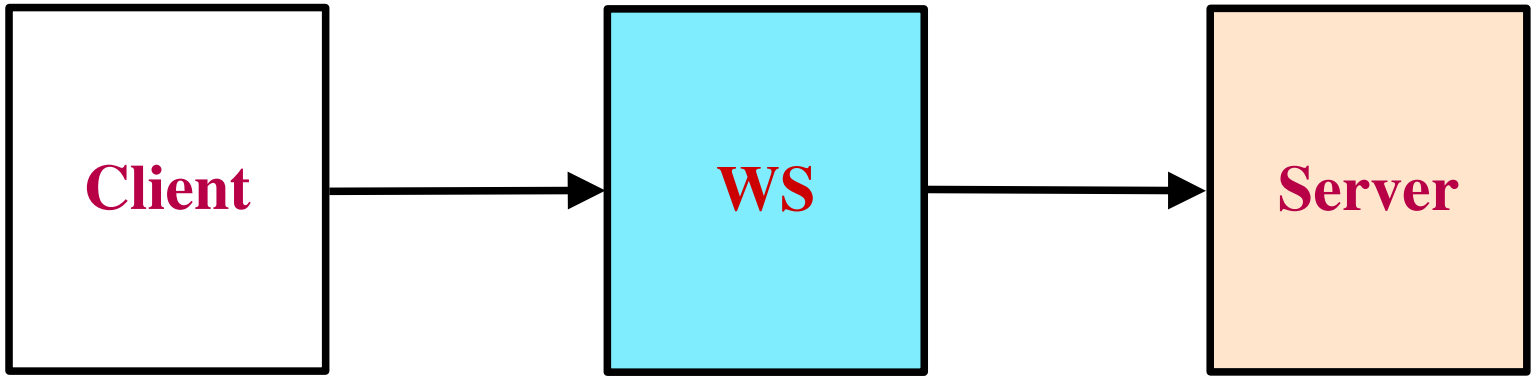
Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password | salt | server))...))

Salt: a number chosen at password installation to be unique to Client

Authentication Protocols

Password Authentication



Lamport's Hash

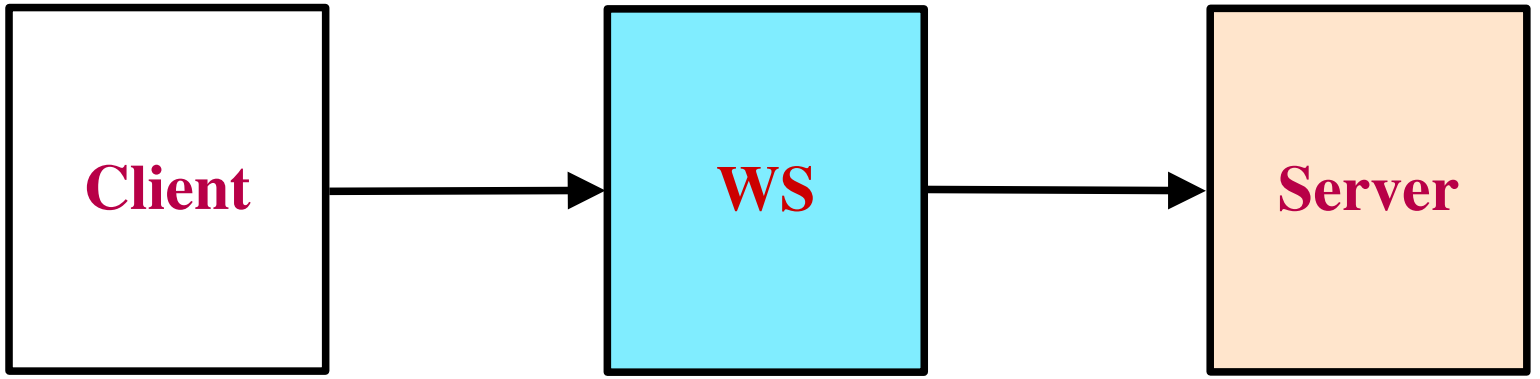
Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password | salt | server))...))

Salt: a number chosen at password installation to be unique to Client

Server: adding this makes sure the extra stuff is different for each server

Authentication Protocols

Password Authentication



Lamport's Hash

Server has client's username, counter (decr by 1 when client is authenticated), hash(hash(... n times...(hash(password | salt | server))...))

Salt: a number chosen at password installation to be unique to Client

Server: adding this makes sure the extra stuff is different for each server

Client can use same password on lots of Servers with different salt values, n is different on all the Servers. No need to change password when n=1, just choose a different salt.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

x = hash(hash(...n-p-1 times...(hash(password))...))

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

y = hash(hash(...p times...(hash(x))...))

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Protection: WS needs to tell the user what n it got from the server?

Authentication Protocols

Password Authentication

Lamport's Hash - Problems

1. No mutual authentication - client does not know Server is on other end hence vulnerable to man-in-the-middle attack
2. Small n attack - Attacker impersonates Server by spoofing network addr. Attacker sends small n (less than current n) and salt (from prior look) to Workstation. Client computes hash, n times. Sends result to Attacker who hashes a few more times to get hash(... current n times ...(hash(.))...) and sends result to Server who thinks Attacker is the Client and allows login.

Protection: better – use on top of something like ssl to encrypt session

Authentication Protocols

Password Authentication

Strong Password Protocols

Designed so Attacker cannot gain information from authentication that will assist in an off-line attack (password guesses).

Authentication Protocols

Password Authentication

Strong Password Protocols

Designed so Attacker cannot gain information from authentication that will assist in an off-line attack (password guesses).

Client and Server share a secret key W - the hash of the Client's password
Server stores the hash, Client computes it from the hash function.

Authentication Protocols

Password Authentication

Strong Password Protocols

Designed so Attacker cannot gain information from authentication that will assist in an off-line attack (password guesses).

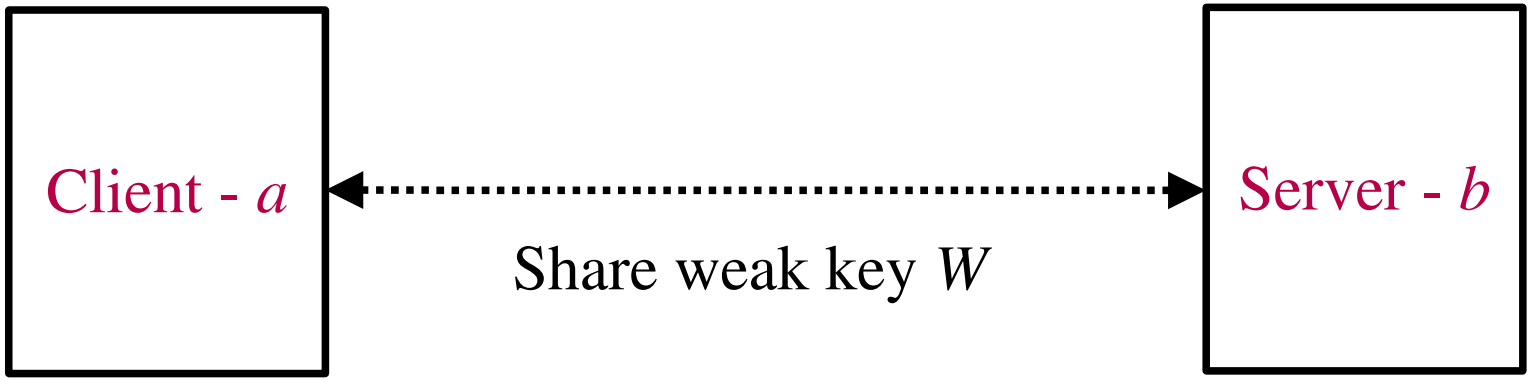
Client and Server share a secret key W - the hash of the Client's password
Server stores the hash, Client computes it from the hash function.

Do Diffie-Hellman exchange encrypting p,g with W and then mutual authentication based on the new Diffie-Hellman secret which is strong.

Authentication Protocols

Password Authentication

Strong Password Protocols

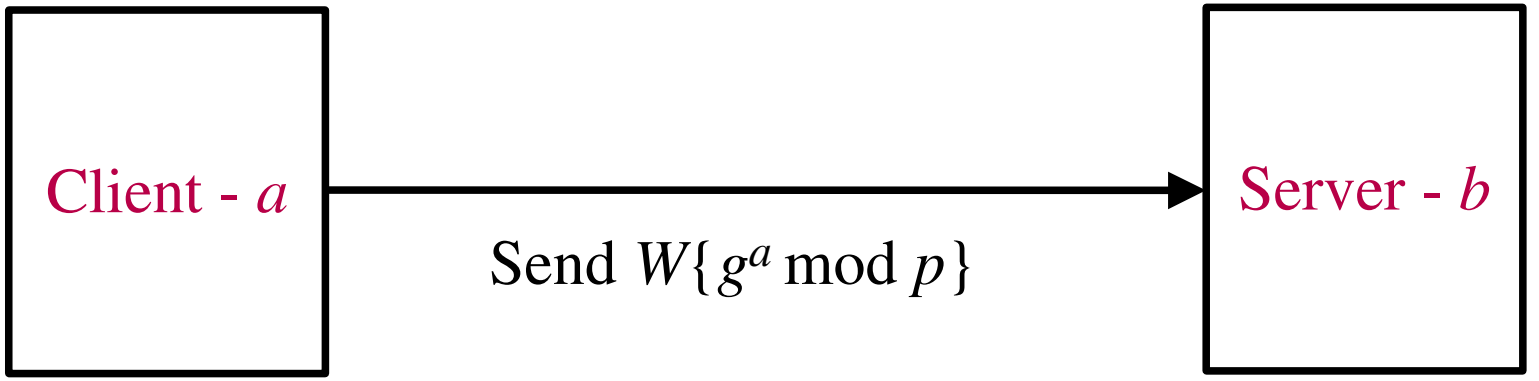


Start out with weak key *W* which is hash(password)

Authentication Protocols

Password Authentication

Strong Password Protocols



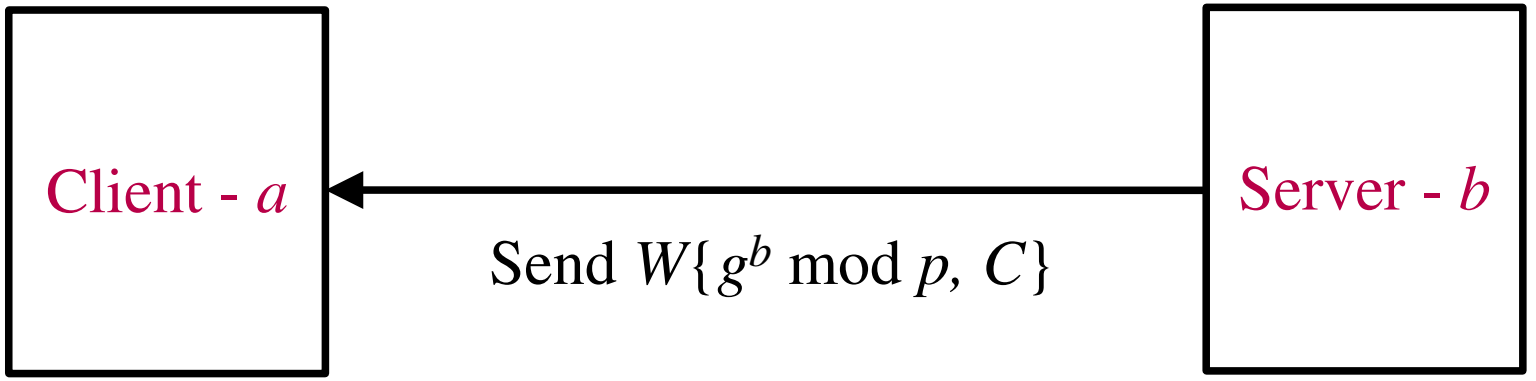
Start out with weak key W which is hash(password)

Client sends $g^a \bmod p$ to Server encrypted using W

Authentication Protocols

Password Authentication

Strong Password Protocols



Start out with weak key W which is hash(password)

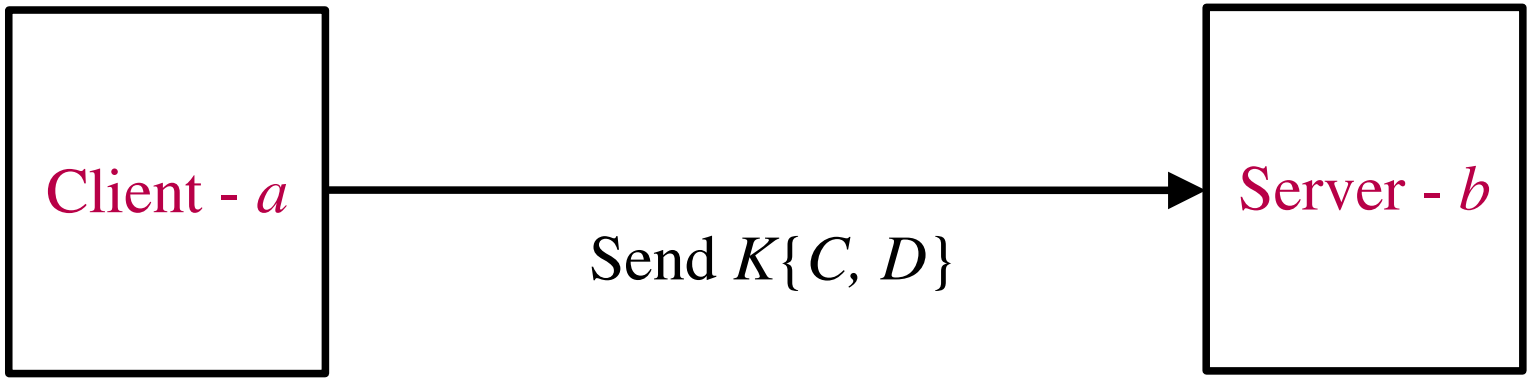
Client sends $g^a \bmod p$ to Server encrypted using W

Server sends $g^b \bmod p$ and a challenge to Client, encrypted with W

Authentication Protocols

Password Authentication

Strong Password Protocols



Start out with weak key W which is hash(password)

Client sends g^a mod p to Server encrypted using W

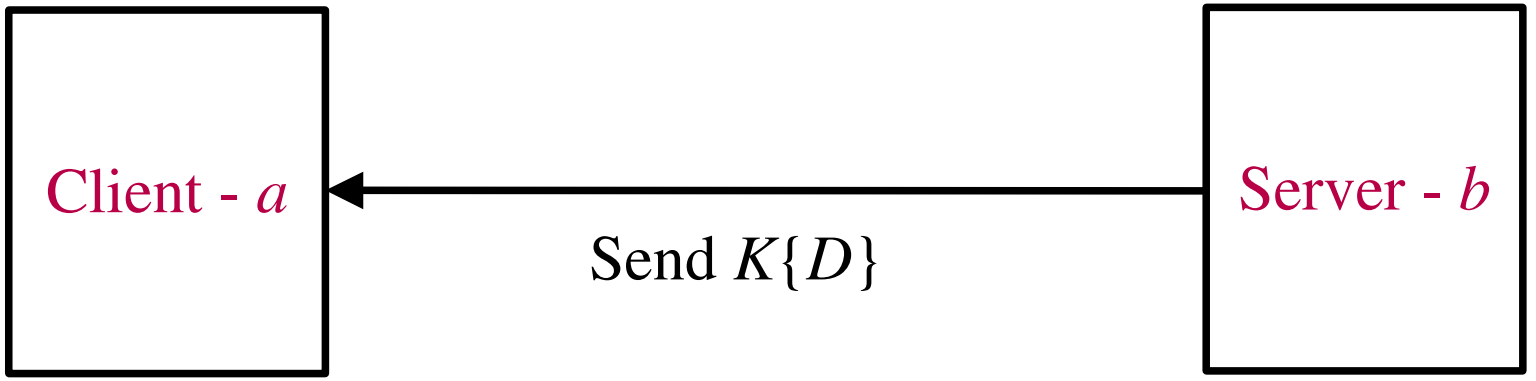
Server sends g^b mod p and a challenge to Client, encrypted with W

Client computes strong key $K = g^{ab}$ mod p, choose challenge, encrypts with K and sends to Server

Authentication Protocols

Password Authentication

Strong Password Protocols



Start out with weak key W which is hash(password)

Client sends g^a mod p to Server encrypted using W

Server sends g^b mod p and a challenge to Client, encrypted with W

Client computes strong key $K = g^{ab}$ mod p, choose challenge, encrypts with K and sends to Server

Server sends Client's challenge back to client encrypted with K

Authentication Protocols

Password Authentication

Strong Password Protocols - Why does this work?

g^a mod p looks random so $W\{g^a \text{ mod } p\}$ looks random too - cannot verify in an off-line attack that the password has been found - decryption with any password still gives a "random" looking number.

Subtleties:

Assume straightforward encryption of g^a mod p - then any off-line password guess that comes up with a number greater than p cannot be that password

Authentication Protocols

Password Authentication

Strong Password Protocols - Why does this work?

$g^a \bmod p$ looks random so $W\{g^a \bmod p\}$ looks random too - cannot verify in an off-line attack that the password has been found - decryption with any password still gives a "random" looking number.

Subtleties:

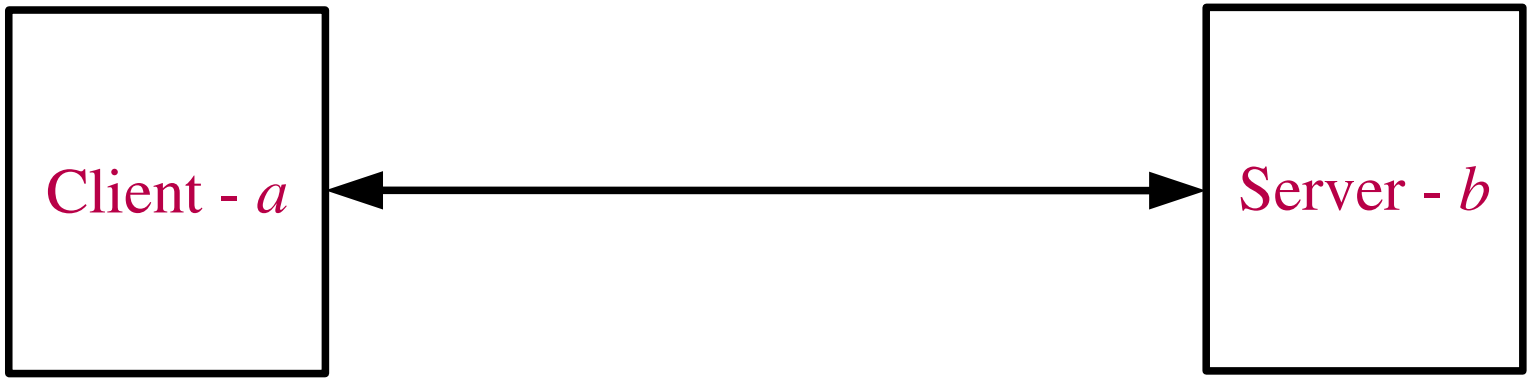
Assume straightforward encryption of $g^a \bmod p$ - then any off-line password guess that comes up with a number greater than p cannot be that password

If p is a little more than a power of 2 then an incorrect password has 50% chance of being eliminated. Wait for second message to get another 25% and so on.

Authentication Protocols

Password Authentication

Strong Password Protocols - SPEKE



Start out with weak key W which is hash(password)

Let $g = W^2 \bmod p$

Client sends $K_c = g^a \bmod p$ to Server

Server sends $K_s = g^b \bmod p$ to Client

Client and Server compute strong key $K = g^{ab} \bmod p$

Same K for Client and Server if and only if the same password was hashed

Abort if K_c or K_s is not between 2 and p-2

Authentication Protocols

Password Authentication

Two Factor Authentication

Requestor uses two independent means of evidence when claiming ID
Something one knows (PIN), has (dongle), is (fingerprint, DNA)
Seeks to decrease the probability that a requestor is presenting false ID
Common example: ATM card (has) + PIN (knows)
RSA SecureID

How might it work?

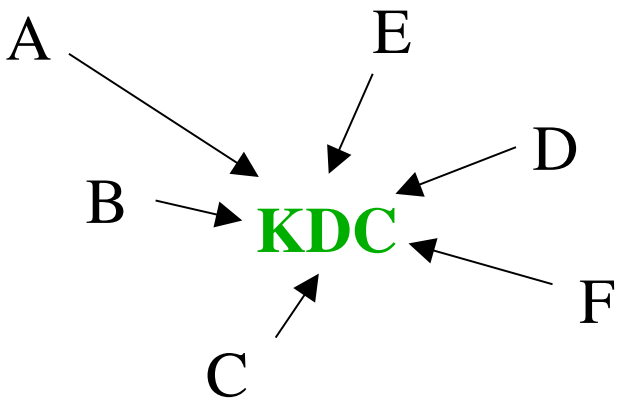
- Server sends some signal to the dongle
- The dongle interprets the signal and displays a number on the screen
- The requestor enters the number – this convinces the server that the requestor has the dongle

- The dongle may have a clock – the server gets the count to verify

Authentication

Key Distribution Centers

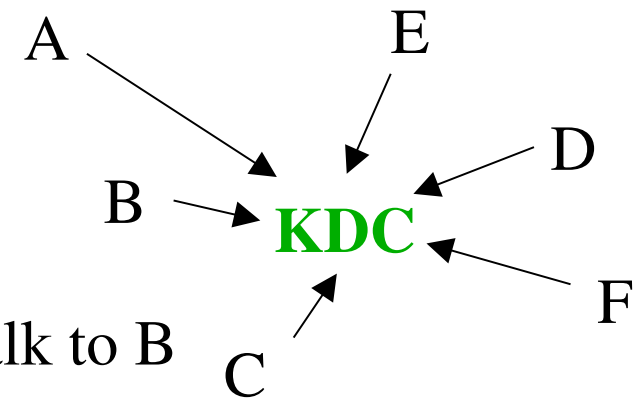
If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.



Authentication

Key Distribution Centers

If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.

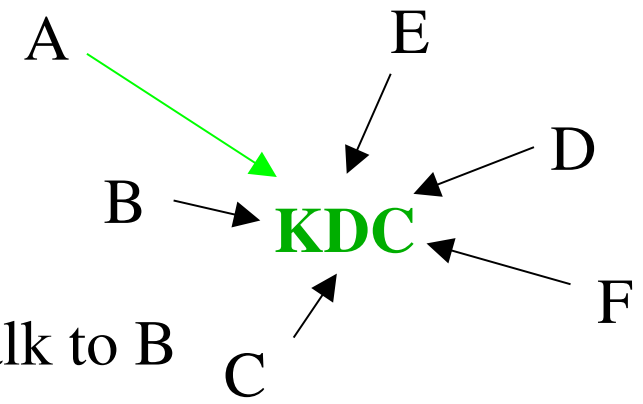


Suppose A wants to talk to B

Authentication

Key Distribution Centers

If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.

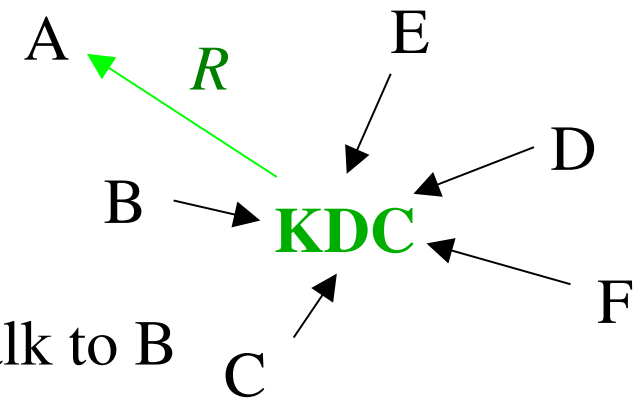


Suppose A wants to talk to B
A gets B's key from the KDC

Authentication

Key Distribution Centers

If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.

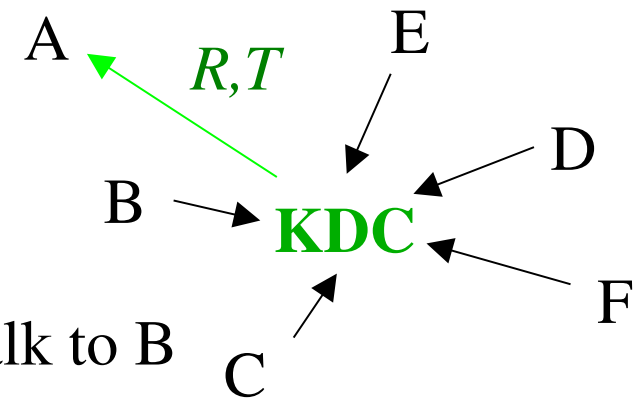


Suppose A wants to talk to B
A gets B's key from the KDC
KDC authenticates A, chooses random number *R* and sends *R* to A encrypted

Authentication

Key Distribution Centers

If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.



Suppose A wants to talk to B

A gets B's key from the KDC

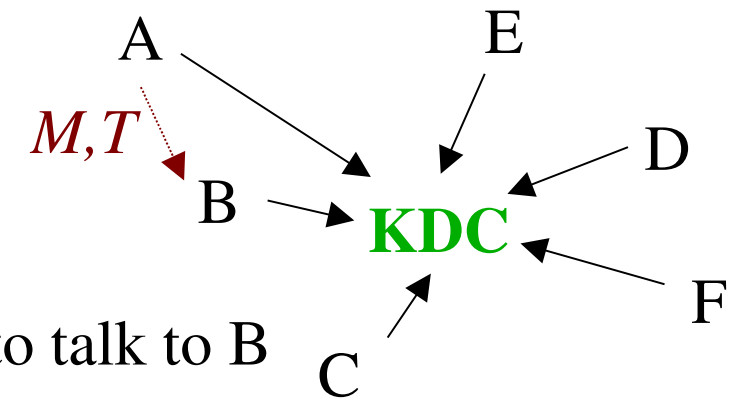
KDC authenticates A, chooses random number *R* and sends *R* to A encrypted

KDC encrypts *R* with B's key, call it *T*, and sends that to A to be forwarded

Authentication

Key Distribution Centers

If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.

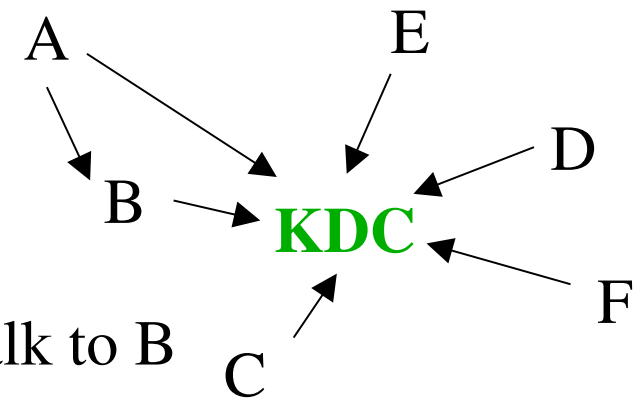


Suppose A wants to talk to B
A gets B's key from the KDC
KDC authenticates A, chooses random number *R* and sends *R* to A encrypted
KDC encrypts *R* with B's key, call it *T*, and sends that to A to be forwarded
A encrypts message using *R* as secret, sends enc message *M* plus *T* to B

Authentication

Key Distribution Centers

If the keys are in a few places, the explosion of needed keys is eliminated. The KDC knows keys for all machines. A new machine in the network causes update between that machine and KDC, not all other machines.



Suppose A wants to talk to B
A gets B's key from the KDC
KDC authenticates A, chooses random number *R* and sends *R* to A encrypted
KDC encrypts *R* with B's key, call it *T*, and sends that to A to be forwarded
A encrypts message using *R* as secret, sends enc message *M* plus *T* to B
B decrypts *T* to get the secret, then uses the secret on *M* to reveal message

Authentication

Key Distribution Centers

Problems:

Authentication

Key Distribution Centers

Problems:

1. KDC has enough information to impersonate anyone to anyone.
   If it is compromised the ballgame is over.

Authentication

Key Distribution Centers

Problems:

1. KDC has enough information to impersonate anyone to anyone. If it is compromised the ballgame is over.
2. KDC is a single point of failure. If it goes down the ballgame is suspended. Additional KDCs for fault tolerance results in more vulnerability.

Authentication

Key Distribution Centers

Problems:

1. KDC has enough information to impersonate anyone to anyone. If it is compromised the ballgame is over.
2. KDC is a single point of failure. If it goes down the ballgame is suspended. Additional KDCs for fault tolerance results in more vulnerability.
3. KDC could be a performance bottleneck

Authentication

Certification Authorities

For public keys:

Authentication

Certification Authorities

For public keys:

A trusted machine containing all public keys for public key LANs.

Generates certificates - signed message with name + public key.

Certificates have a termination date.

Certificates may be stored at each machine

- user presents certificate to partner when initiating transaction

Authentication

Certification Authorities

For public keys:

A trusted machine containing all public keys for public key LANs.

Generates certificates - signed message with name + public key.

Certificates have a termination date.

Certificates may be stored at each machine

- user presents certificate to partner when initiating transaction

Advantages:

1. Need not be on-line
2. Hence should be more secure and simpler
3. If CA crashes, network still functions, but no new users accommodated
4. Attacker (probably) cannot issue certificates signed by CA
5. A compromised CA cannot decrypt conversations – does not know private keys of clients.

Authentication

Certification Authorities

For public keys:

A trusted machine containing all public keys for public key LANs.

Generates certificates - signed message with name + public key.

Certificates have a termination date.

Certificates may be stored at each machine

- user presents certificate to partner when initiating transaction

Advantages:

1. Need not be on-line
2. Hence should be more secure and simpler
3. If CA crashes, network still functions, but no new users accommodated
4. Attacker (probably) cannot issue certificates signed by CA
5. A compromised CA cannot decrypt conversations – does not know private keys of clients.

Problem: what if someone should have certificate revoked?

- say a disgruntled ex-employee

Authentication

Certification Authorities

For public keys:

A trusted machine containing all public keys for public key LANs.

Generates certificates - signed message with name + public key.

Certificates have a termination date.

Certificates may be stored at each machine

- user presents certificate to partner when initiating transaction

Advantages:

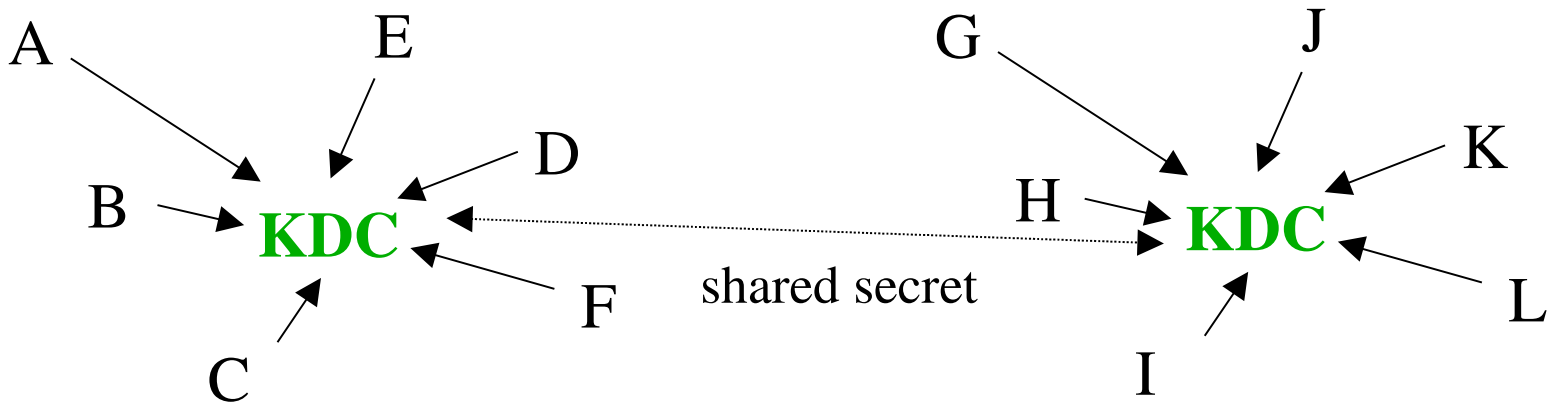
1. Need not be on-line
2. Hence should be more secure and simpler
3. If CA crashes, network still functions, but no new users accommodated
4. Attacker (probably) cannot issue certificates signed by CA
5. A compromised CA cannot decrypt conversations – does not know private keys of clients.

Problem: what if someone should have certificate revoked? Cert. Rev. List

- say a disgruntled ex-employee

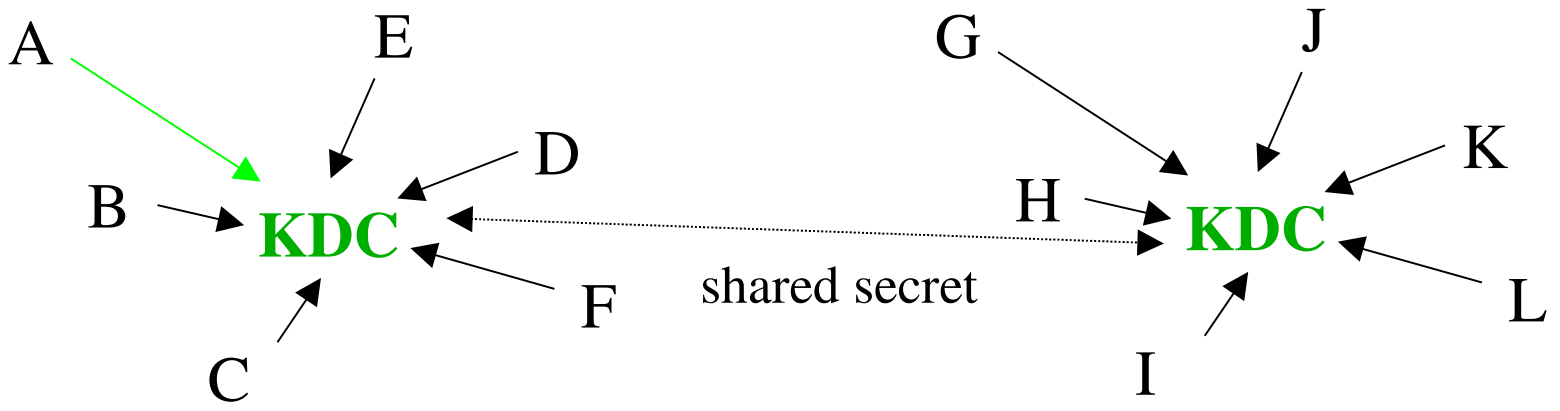
Authentication

Multiple Trusted Authorities - KDCs



Authentication

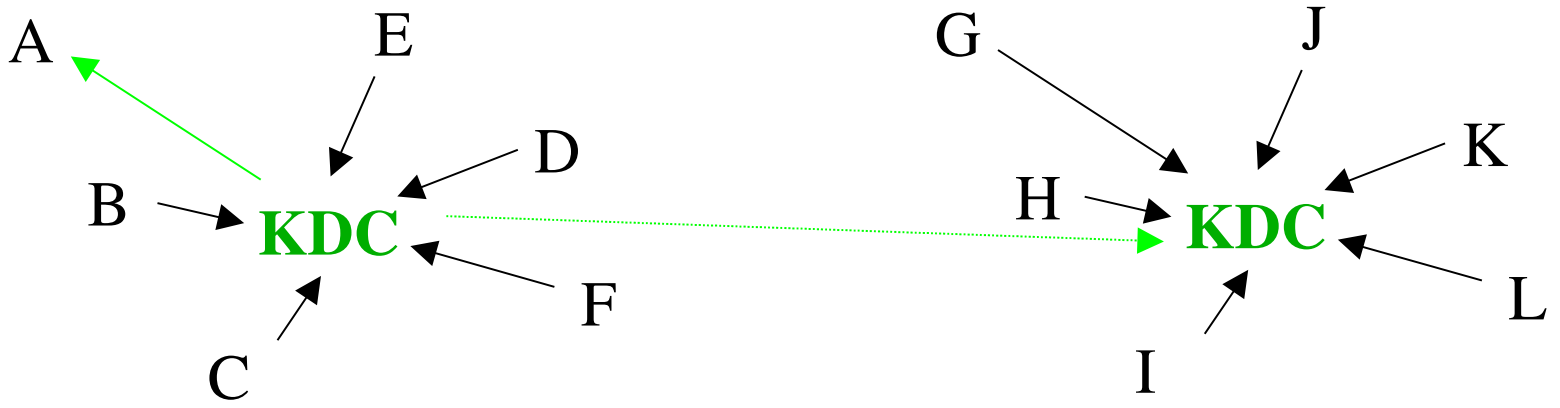
Multiple Trusted Authorities - KDCs



A requests to communicate with L

Authentication

Multiple Trusted Authorities - KDCs



A requests to communicate with L

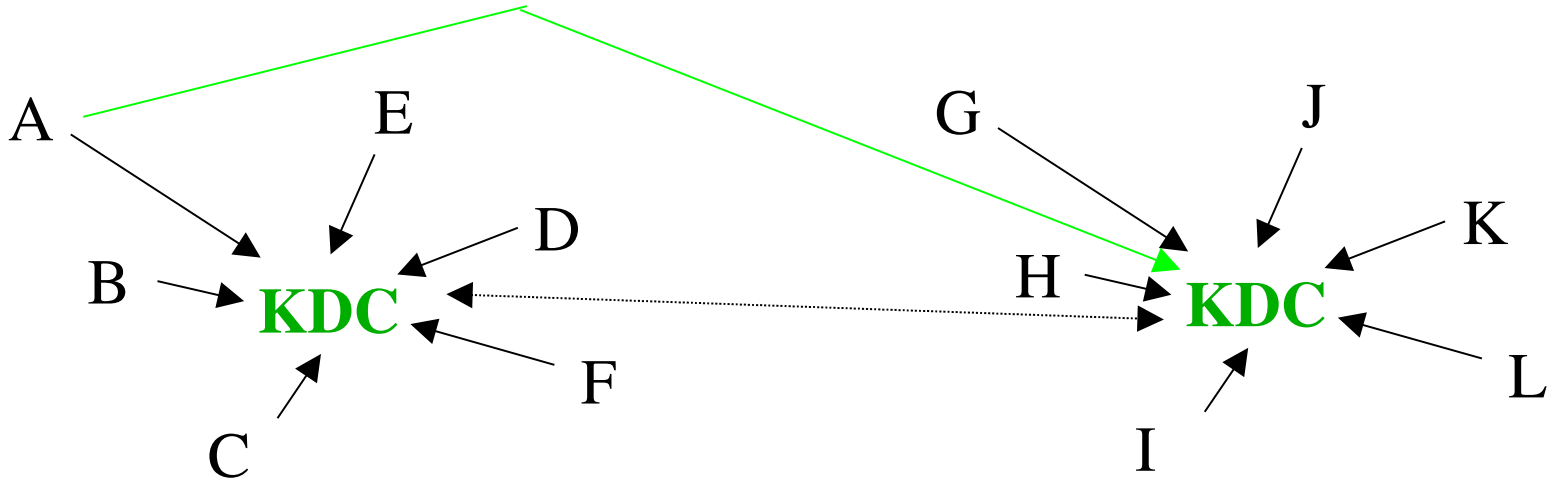
A's KDC generates key K_A for the transaction

A's KDC encrypts K_A with A's key and sends it to A

A's KDC encrypts K_A with shared secret and sends it to L's KDC

Authentication

Multiple Trusted Authorities - KDCs



A requests to communicate with L

A's KDC generates key K_A for the transaction

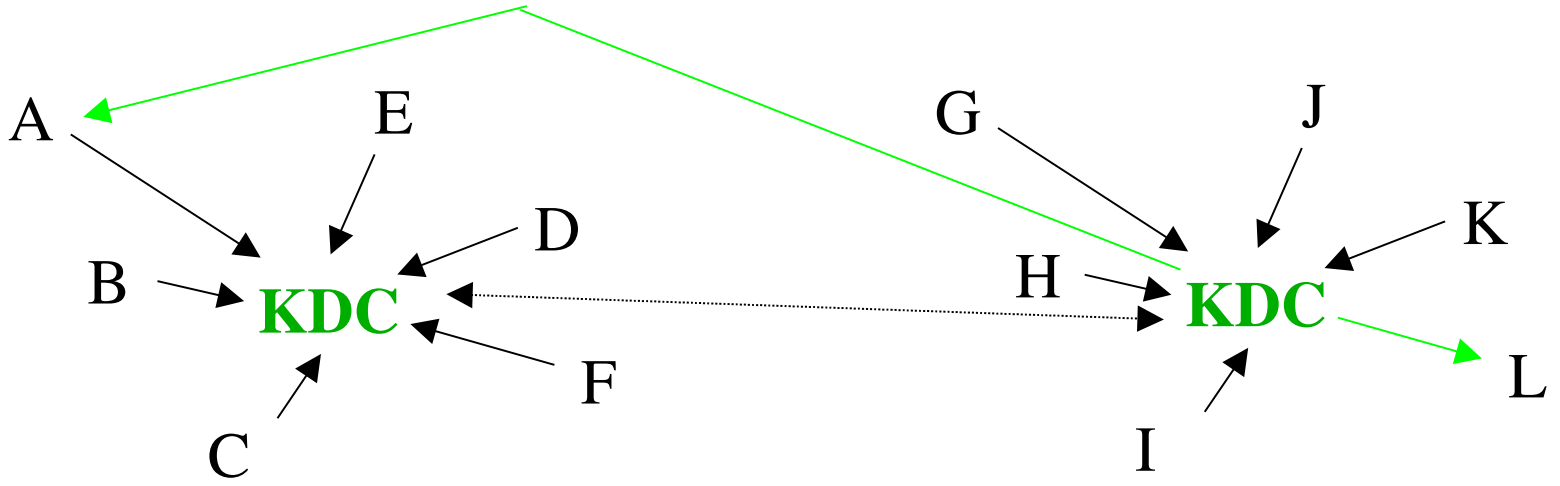
A's KDC encrypts K_A with A's key and sends it to A

A's KDC encrypts K_A with shared secret and sends it to L's KDC

A sends request to communicate with L to L's KDC

Authentication

Multiple Trusted Authorities - KDCs



A requests to communicate with L

A's KDC generates key K_A for the transaction

A's KDC encrypts K_A with A's key and sends it to A

A's KDC encrypts K_A with shared secret and sends it to L's KDC

A sends request to communicate with L to L's KDC

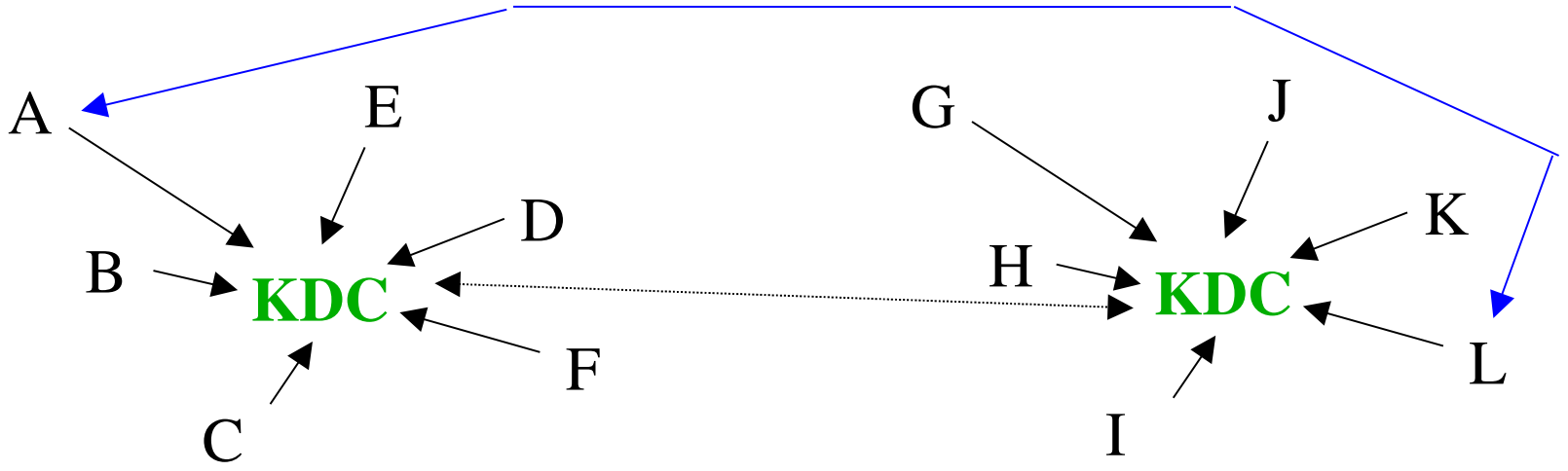
L's KDC generates key K_L for the transaction

L's KDC encrypts K_L with K_A and sends it to A

L's KDC encrypts K_L with L's key and sends it to L

Authentication

Multiple Trusted Authorities - KDCs



A requests to communicate with L

A's KDC generates key K_A for the transaction

A's KDC encrypts K_A with A's key and sends it to A

A's KDC encrypts K_A with shared secret and sends it to L's KDC

A sends request to communicate with L to L's KDC

L's KDC generates key K_L for the transaction

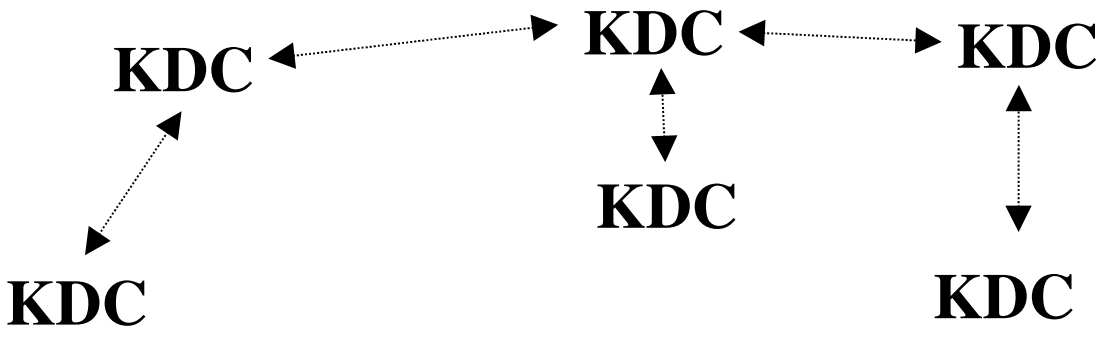
L's KDC encrypts K_L with K_A and sends it to A

L's KDC encrypts K_L with L's key and sends it to L

A and L communicate using key K_L

Authentication

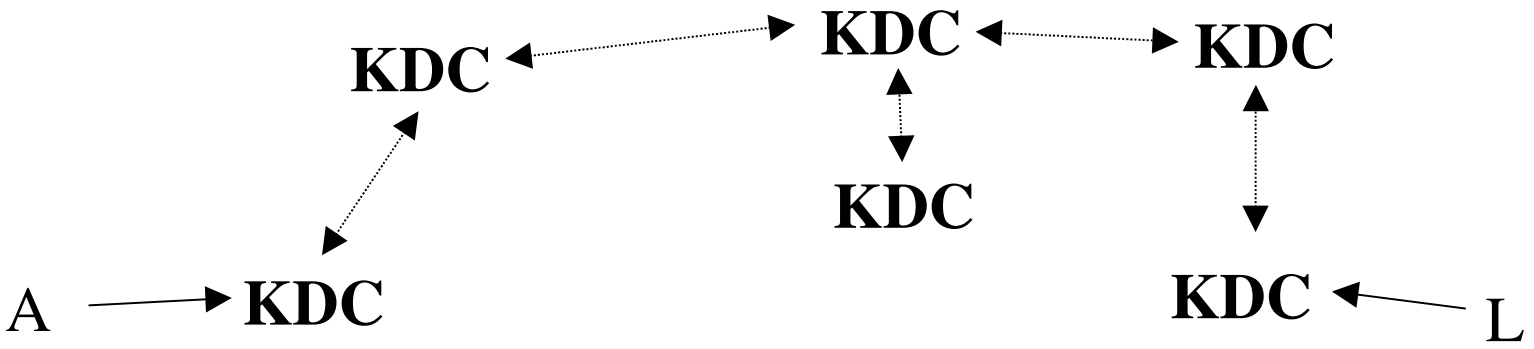
Multiple Trusted Authorities - KDCs



Unworkable due to explosion of shared keys unless there is some tree-like hierarchy of KDCs.

Authentication

Multiple Trusted Authorities - KDCs

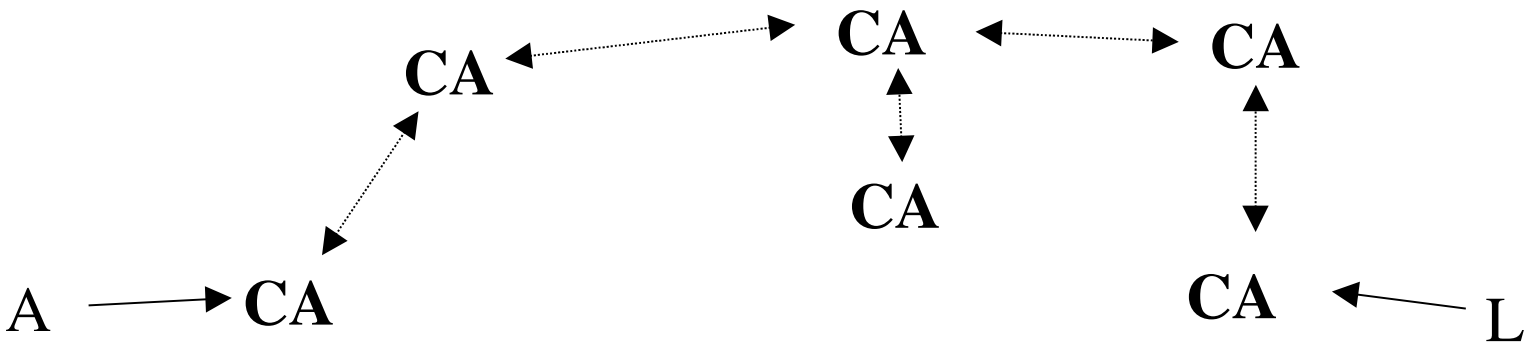


Unworkable due to explosion of shared keys unless there is some tree-like hierarchy of KDCs.

A must negotiate through a chain of KDCs before communicating with L each time getting a new key for the next transaction until finally getting a transaction key for A-L.

Authentication

Multiple Trusted Authorities - CAs

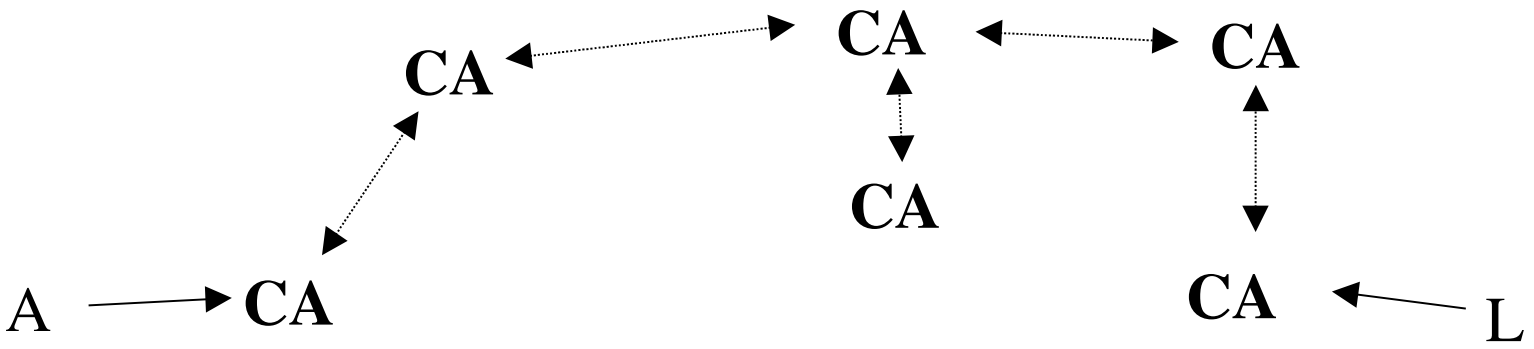


Each CA issues certificates for its own domain

All users in a domain can verify their certificates because they know the public key of their own CA

Authentication

Multiple Trusted Authorities - CAs



Each CA issues certificates for its own domain

All users in a domain can verify their certificates because they know the public key of their own CA

If A wants to communicate with L verification proceeds in a chain, as before, with each CA in the chain having certificates for the other

Example:

A gets L's CA's certificate stating its public key is PC signed by A's CA

A gets L's certificate stating its public key is PL signed with key PC

Hence A is sure it has L's public key