

User authentication

Beyond user authentication (reminder)

User authentication is not the full story.

- Computers have identities too.
- Computers can help.

User authentication

User authentication can be based on:

- Something the user knows,
for example a PIN or a password.
- Something the user has,
for example a smart-card.
- Some characteristic of the user,
for example typing pattern, voice, fingerprints.
- Where the user is located,
for example in a secure building.

User authentication (cont.)

Passwords are pervasive.

Other user authentication mechanisms are sometimes stronger, more convenient, or more appropriate.

Other user authentication mechanisms are often combined with passwords:

- Some authentication tokens require a PIN.
- Typing patterns can be used to enhance passwords.
- Location may reduce but not eliminate the role of other bases for authentication.

The trouble with users

User authentication is difficult.

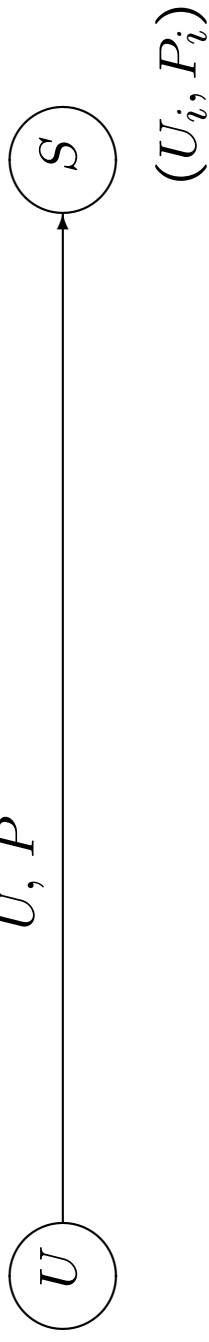
- Users cannot remember long secrets.
- Users cannot compute much,
so for example they cannot sign delegation certificates.
- Users are gullible, impatient, demanding, changing, . . .

User authentication is often the weakest link.

Naive passwords

The system has a table that maps users to passwords.

The user presents a password via a direct link.



Naive passwords: problems

The passwords must be hard to guess (on-line and off-line).

The passwords must be stored somewhere.

- We must rely on the password file remaining secure.
- The file could be encrypted, but then where would the key be kept?

We need to transmit passwords from user to system.

- With physical security?
- With encryption?

Improving passwords

Force users to choose better passwords.

- Test selected passwords against a known dictionary.
- Require numeric or non-alphanumeric characters.
- Require at least n -character passwords.

But users do not like rules:

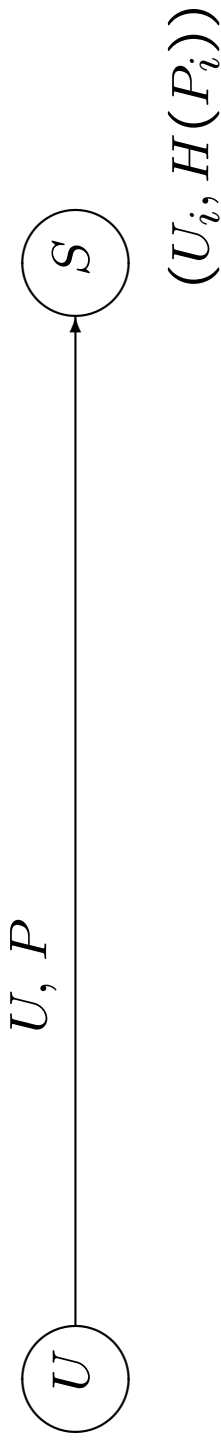
- They get annoyed.
- They may write down hard-to-remember passwords.
- They use the same passwords in many places.
- They do even worse when passwords are changed according to a schedule.

Encrypted password files (Needham)

The system has a table that maps users to the result of applying a one-way function to their passwords.

The user presents a password via a direct link.

The system checks by applying the function.

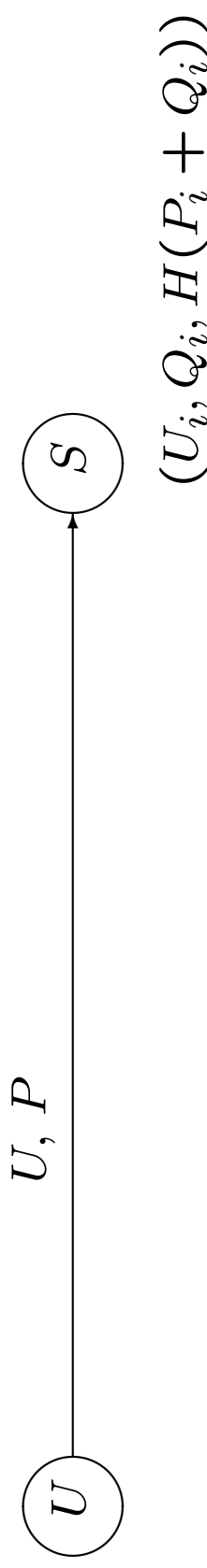


Protecting the password file

The password file is still subject to brute-force attacks, including dictionary attacks.

\Rightarrow

- The password file should itself be protected.
- The system may create a **salt** for each user, and store the salt and the result of applying a one-way function to the password and the salt.



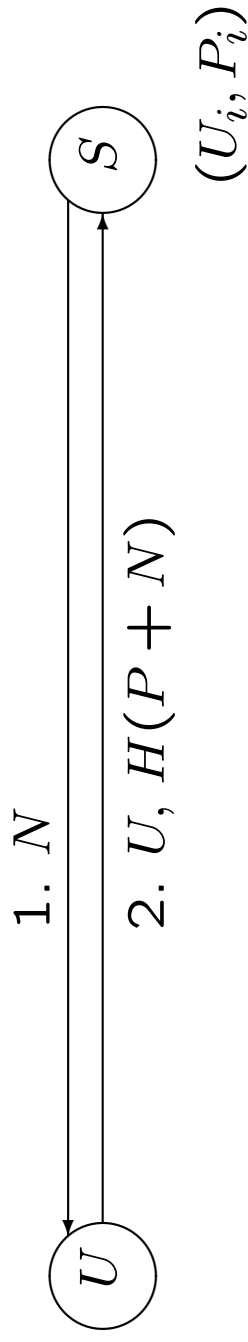
Another approach: challenge-response

The system has a table that maps users to passwords.

The system creates a nonce challenge.

The user replies with the result of applying a one-way hash function to the nonce and the password.

The system checks by applying the function too.



Combination

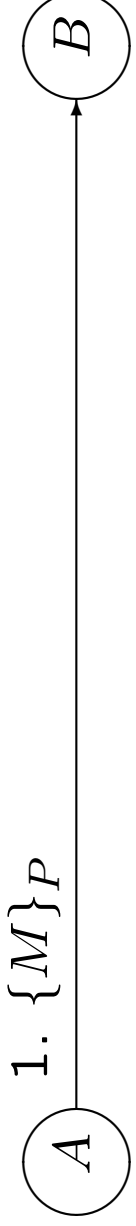
Use two passwords:

- one for challenge-response,
- the other stored encrypted.

The two can be derived from a master password.

Secure communication with
passwords

Simple encrypted communication



- A and B are two computers.
- P is a secret shared by A and B .
- M is a message (with timestamps, etc.).
- $\{M\}_P$ is the message encrypted under P .

An attack

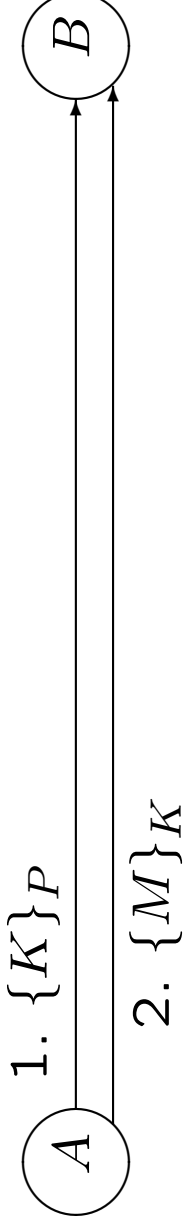
An attacker that guesses P can confirm the guess, by:

- eavesdropping,
- decrypting $\{M\}_P$ and
- seeing if the result makes sense.

The attacker can guess many times without detection.

⇒ This protocol is not adequate when P is weak.

Encrypted communication (Take 2)



- A creates a random key K .
- A sends K to B encrypted under P .

The previous attack no longer works.

But a variant of the attack still works.

An attacker who intercepts $\{K\}_P$ and $\{M\}_K$ can confirm a guess of P .

(Cf. Kerberos.)

Encrypted communication (Take 3)

There are several clever protocols for “encrypted key exchange”. (See Gong, Lomas, Needham, and Saltzer; Bellare and Merritt.)

These protocols allow the exchange of a key despite the weakness of a password.

But:

- They are all rather complex.
- Some of them have flaws. (See Patel.)

EKE (sketch of one basic version)

A generates a public-key pair and sends the public key encrypted under the password P to B .

B obtains the public key, encrypts a fresh secret R with it, and sends the whole thing to A encrypted under P .

Afterwards A and B both know R .

Assuming that the public key is random, an attacker does not get much.

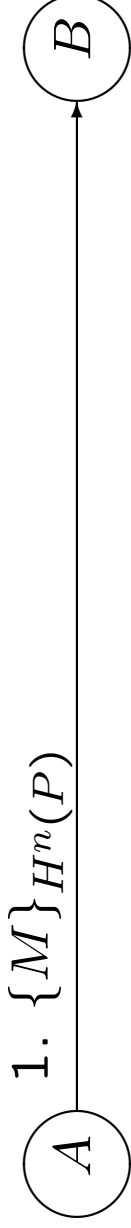
EKE variants and alternatives

There are by now several variants of EKE,

- with different cryptosystems,
 - so that B does not keep the password in clear,
- ⋮

There are also quite a few alternative protocols (e.g., SRP), and some proofs.

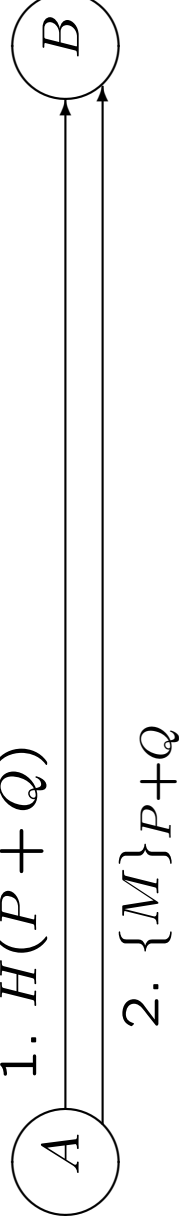
Stretching passwords for
encrypted communication



n is a number like 2^{20} .

Brute-force attacks are slowed down.

Strengthening passwords for encrypted communication



- A creates a random password supplement Q .
- H is a one-way function (like MD5 or SHA).
- $P + Q$ is the concatenation of P and Q , the full password.
- B reconstructs Q by brute force.

Password stretching and strengthening both illustrate a trade-off in three dimensions:

- security,
- user memory,
- access time.

Their assumptions and specifics differ.

Passwords for laptops

A password P may protect a laptop and be used as key for encrypting files on the laptop.

The laptop should not keep P in clear.

Instead, it should keep a function of P , for example $H(P)$.

Suppose an attacker gets physical control of the laptop, and direct access to its disk.

It may guess P and check against $H(P)$ or $\{\text{files}\}_P$.

Password strengthening/stretching can be applied.

Alternatively, a third party may help (see MacKenzie and Reiter).