Authentication

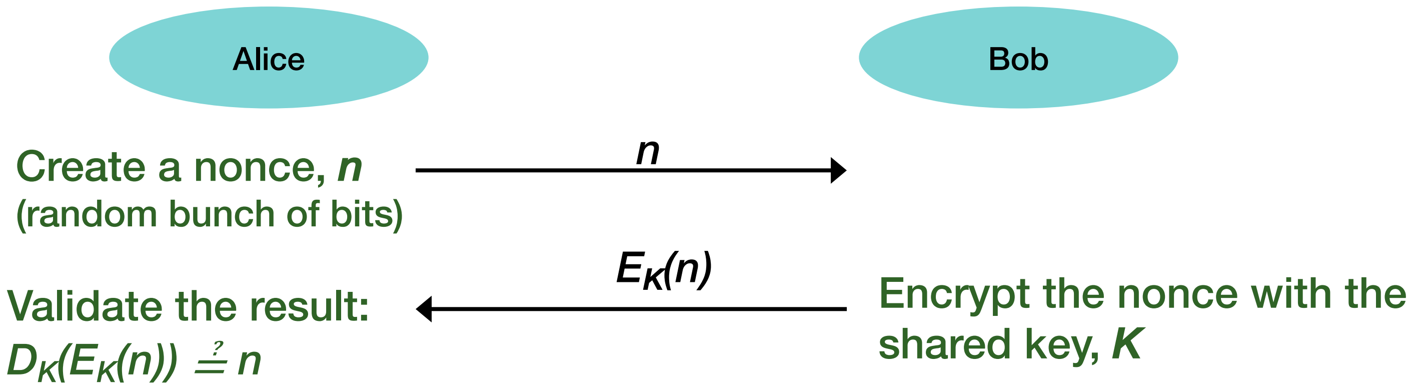
- **Identification**: *who are you?*
- **Authentication**: *prove it*
- **Authorization**: *you can do this*

Some protocols (or services) combine all three

Cryptographic Authentication

The concept: prove you have the key

Ask the other side to prove they can encrypt or decrypt a message with the key



- This assumes a **pre-shared key** and symmetric cryptography.
- After that, Alice can encrypt & send a **session key**.
- Minimize the use of the pre-shared key.

Mutual authentication

- **Alice had Bob prove he has the key**
- **Bob may want to validate Alice as well**

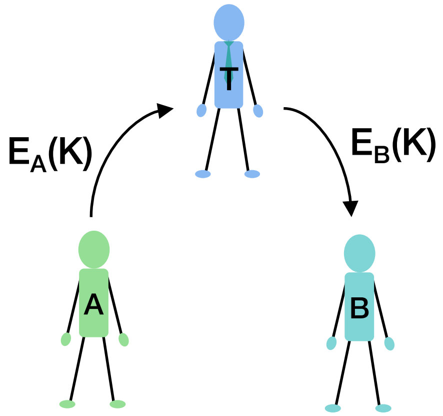
  ⇒ mutual authentication
  - Bob will do the same thing
    - Have Alice prove she has the key
- **Pre-shared key: Alice encrypts the nonce with the key**
- **Public key: Alice encrypts the nonce with her private key**

Combined authentication & key exchange

Basic idea with symmetric cryptography:

Use a trusted third party (Trent) that has all the keys

- **Alice wants to talk to Bob: she asks Trent**
  - Trent generates a session key encrypted for Alice
  - Trent encrypts the same key for Bob (ticket)
- **Authentication is implicit:**
  - If Alice can encrypt a message for Trent, she proved she knows her key
  - If Bob can decrypt the message from Trent, he proved he knows his key
- **Trent can also perform *authorization***
- **Weaknesses that we need to address:**
  - Replay attacks



Combined authentication & key exchange algorithms

Security Protocol Notation

$Z \parallel W$

- Z concatenated with W

$A \rightarrow B : \{ Z \parallel W \} k_{A,B}$

- A sends a message to B
- The message is the concatenation of Z & W and is encrypted by key $k_{A,B}$, which is shared by users A & B

$A \rightarrow B : \{ Z \} k_A \parallel \{ W \} k_{A,B}$

- X sends a message to Y
- The message is a concatenation of Z encrypted using A's key and W encrypted by a key shared by A and Y

r_1, r_2

- nonces – strings of random bits

Bootstrap problem

How can Alice & Bob communicate securely?

- **Alice cannot send a key to Bob in the clear**
  - We assume an unsecure network
- **We looked at two mechanisms:**
  - Diffie-Hellman key exchange
  - Public key cryptography

Let's examine the problem some more … in the context of authentication & key exchange

Simple Protocol

Use a trusted third party – Trent – who has all the keys

Trent creates a session key for Alice and Bob

$$\begin{tikzcd}
\text{Alice} \arrow[r, "\text{Request session key to Bob}"] & \text{Trent} \\
\text{Alice} & \text{Trent} \arrow[l, "\{ k_{A,B} \} k_A \,\|\, \{ k_{A,B} \} k_B"'] \\
\text{Alice} \arrow[r, "\{ k_{A,B} \} k_B"] & \text{Bob} \\
\text{Alice} \arrow[r, leftrightarrow, "\{ m \} k_{A,B}"] & \text{Bob}
\end{tikzcd}$$

Problems

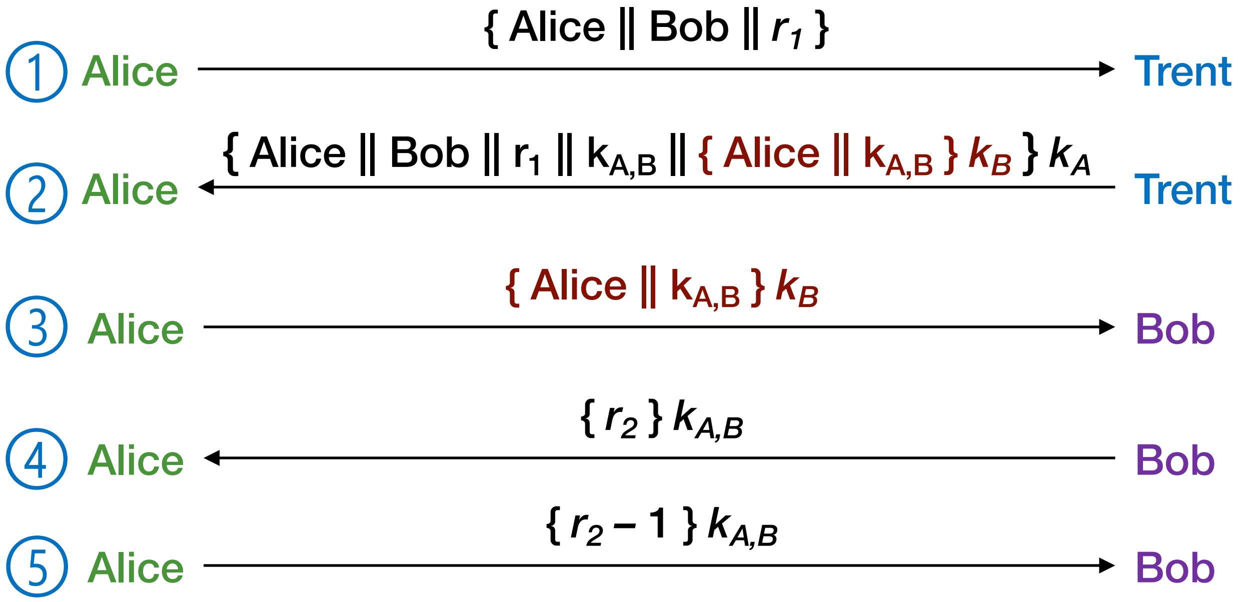
- **How does Bob know he is talking to Alice?**
  - Trusted third party, Trent, has all the keys
  - Trent knows the request came from Alice since only he and Alice can have the key
  - Trent can authorize Alice's request
  - Bob gets a session key encrypted with Bob's key, which only Trent could have created
    - But Bob doesn't know who requested the session – *is the request really from Alice?*
    - *Trent would need to add sender information to the message encrypted for Bob*

- **Vulnerable to replay attacks**
  - Eve records the message from Alice to Bob and later replays it
  - Bob will think he's talking to Alice and re-use the same session key

- **Protocols should provide authentication & defense against replay attacks**

Needham-Schroeder

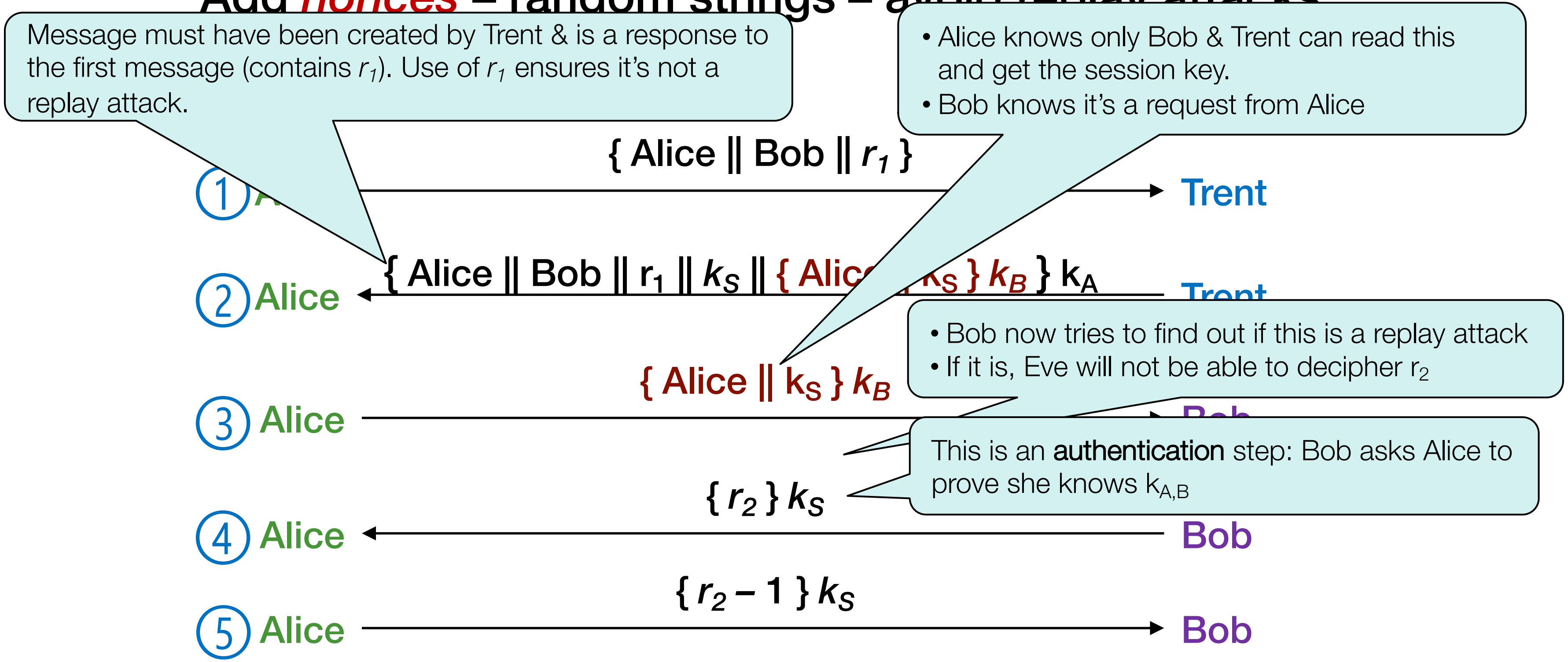
Add ***nonces*** – random strings (r_1, r_2) – to avoid replay attacks



① Alice $\xrightarrow{\{\ \text{Alice} \parallel \text{Bob} \parallel r_1\ \}}$ Trent
② Alice $\xleftarrow{\{\ \text{Alice} \parallel \text{Bob} \parallel r_1 \parallel k_{A,B} \parallel \{\ \text{Alice} \parallel k_{A,B}\ \} k_B\ \} k_A}$ Trent
③ Alice $\xrightarrow{\{\ \text{Alice} \parallel k_{A,B}\ \} k_B}$ Bob
④ Alice $\xleftarrow{\{\ r_2\ \} k_{A,B}}$ Bob
⑤ Alice $\xrightarrow{\{\ r_2 - 1\ \} k_{A,B}}$ Bob

Needham-Schroeder

Add *nonces* – random strings – avoid replay attacks


Message must have been created by Trent & is a response to the first message (contains r_1). Use of r_1 ensures it's not a replay attack.
• Alice knows only Bob & Trent can read this and get the session key.
• Bob knows it's a request from Alice
{ Alice || Bob || r_1 }
1 A
Trent
2 Alice
{ Alice || Bob || r_1 || k_S || { Alic… k_S } k_B } k_A
Trent
• Bob now tries to find out if this is a replay attack
• If it is, Eve will not be able to decipher r_2
{ Alice || k_S } k_B
3 Alice
Bob
This is an **authentication** step: Bob asks Alice to prove she knows $k_{A,B}$
{ r_2 } k_S
4 Alice
Bob
{ r_2 – 1 } k_S
5 Alice
Bob

Needham-Schroeder

Add *nonces* – random strings – a

Message must have been created by Trent & is a response to the first message (contains r_1). Use of r_1 ensures it's not a replay attack.

- Alice knows only Bob & Trent can read this and get the session key.
- Bob knows it's a request from Alice

1. Alice $\xrightarrow{\{\,\text{Alice} \,\|\, \text{Bob} \,\|\, r_1\,\}}$ Trent
2. Alice $\xleftarrow{\{\,\text{Alice} \,\|\, \text{Bob} \,\|\, r_1 \,\|\, k_{A,B} \,\|\, \{\,\text{Alice} \,\|\, k_{A,B}\,\}\,k_B\,\}\,k_A}$ Trent
3. Alice $\xrightarrow{\{\,\text{Alice} \,\|\, k_{A,B}\,\}\,k_B}$ Bob
4. Alice $\xleftarrow{\{\,r_2\,\}\,k_{A,B}}$ Bob
5. Alice $\xrightarrow{\{\,r_2 - 1\,\}\,k_{A,B}}$ Bob

- Bob now tries to find out if this is a replay attack
- If it is, Eve will not be able to decipher r_2

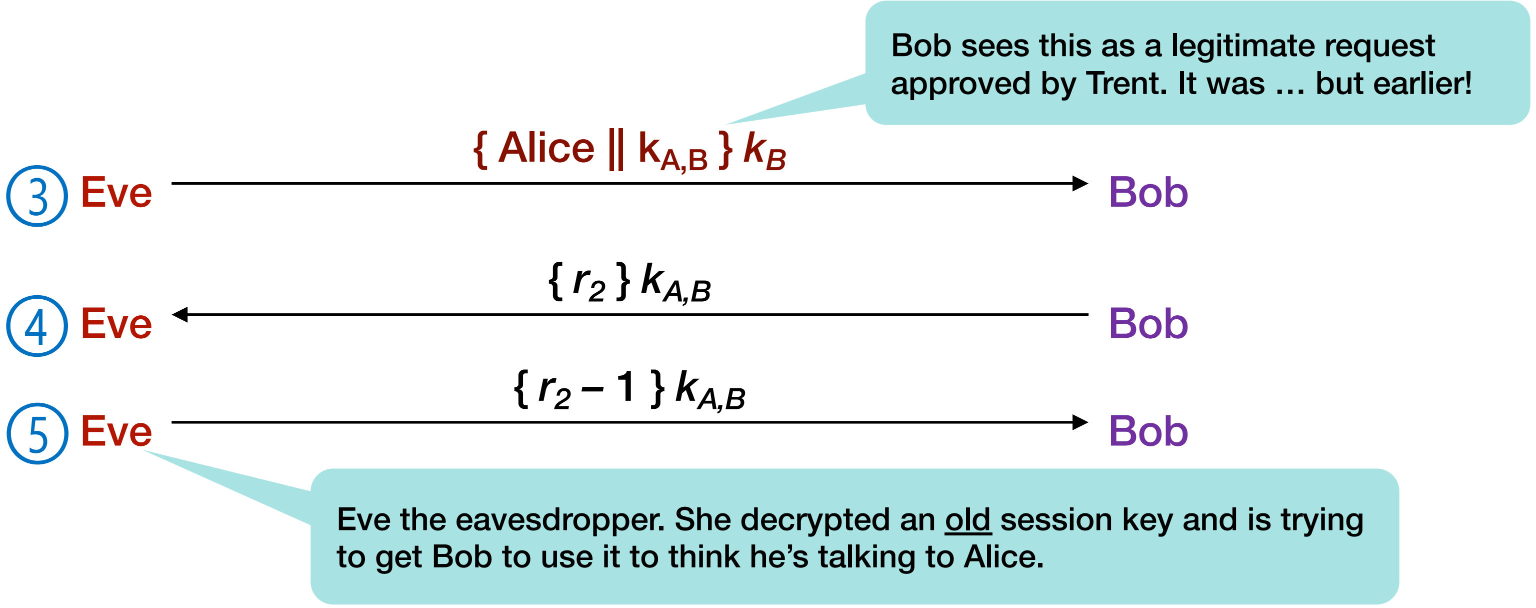
This is an **authentication** step: Bob asks Alice to prove she knows $k_{A,B}$

Needham-Schroeder Protocol Vulnerability

- We assume all keys are secret

Needham-Schroeder is still vulnerable to a certain replay attack … if an old session key is known!

- But suppose Eve can obtain the session key from an <u>old</u> message (she worked hard, got lucky, and cracked an earlier message)


Bob sees this as a legitimate request approved by Trent. It was … but earlier!
$\{ \text{Alice} \parallel k_{A,B} \} k_B$
3 Eve
Bob
$\{ r_2 \} k_{A,B}$
4 Eve
Bob
$\{ r_2 - 1 \} k_{A,B}$
5 Eve
Bob
Eve the eavesdropper. She decrypted an old session key and is trying to get Bob to use it to think he's talking to Alice.

Denning-Sacco Solution

- **Problem: replay in the third step of the protocol**
  - Eve replays the message: $\{ \text{Alice} \,||\, k_{A,B} \} k_B$

- **Solution: use a timestamp *T* to detect replay attacks**
  - The trusted third party (Trent) places a timestamp in a message that is encrypted for Bob
  - The attacker has an old session key but not Alice's, Bob's or Trent's keys
  - Eve cannot spoof a valid message that is encrypted for Bob

Needham-Shroeder w/Denning-Sacco mods

Add nonces – random strings – AND a timestamp

$$\begin{tikzcd}[column sep=huge] \text{Alice} \arrow[r, "\{\ \text{Alice} \parallel \text{Bob} \parallel r_1\ \}"] & \text{Trent} \end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge] \text{Alice} & \text{Trent} \arrow[l, "\{\ \text{Alice} \parallel \text{Bob} \parallel r_1 \parallel k_{A,B} \parallel \{\ \text{Alice} \parallel T \parallel k_{A,B}\ \}\, k_B\ \}\, k_A"'] \end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge] \text{Alice} \arrow[r, "\{\ \text{Alice} \parallel T \parallel k_{A,B}\ \}\, k_B"] & \text{Bob} \end{tikzcd}$$

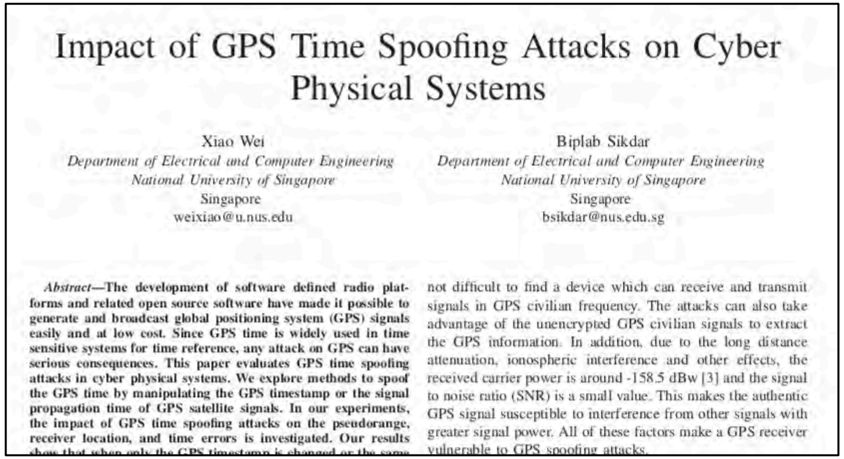
$$\begin{tikzcd}[column sep=huge] \text{Alice} & \text{Bob} \arrow[l, "\{\ r_2\ \}\, k_{A,B}"'] \end{tikzcd}$$

$$\begin{tikzcd}[column sep=huge] \text{Alice} \arrow[r, "\{\ r_2 - 1\ \}\, k_{A,B}"] & \text{Bob} \end{tikzcd}$$

Problem with timestamps

- **Use of timestamps relies on synchronized clocks**
  - Messages may be falsely accepted or falsely rejected because of bad time

- **Time synchronization becomes an attack vector**
  - Create fake NTP responses
  - Generate fake GPS signals



Impact of GPS Time Spoofing Attacks on Cyber Physical Systems

Xiao Wei
Department of Electrical and Computer Engineering
National University of Singapore
Singapore
weixiao@u.nus.edu

Biplab Sikdar
Department of Electrical and Computer Engineering
National University of Singapore
Singapore
bsikdar@nus.edu.sg

***Abstract*—The development of software defined radio platforms and related open source software have made it possible to generate and broadcast global positioning system (GPS) signals easily and at low cost. Since GPS time is widely used in time sensitive systems for time reference, any attack on GPS can have serious consequences. This paper evaluates GPS time spoofing attacks in cyber physical systems. We explore methods to spoof the GPS time by manipulating the GPS timestamp or the signal propagation time of GPS satellite signals. In our experiments, the impact of GPS time spoofing attacks on the pseudorange, receiver location, and time errors is investigated. Our results show that when only the GPS timestamp is changed or the same**

not difficult to find a device which can receive and transmit signals in GPS civilian frequency. The attacks can also take advantage of the unencrypted GPS civilian signals to extract the GPS information. In addition, due to the long distance attenuation, ionospheric interference and other effects, the received carrier power is around -158.5 dBw [3] and the signal to noise ratio (SNR) is a small value. This makes the authentic GPS signal susceptible to interference from other signals with greater signal power. All of these factors make a GPS receiver vulnerable to GPS spoofing attacks.

Otway-Rees Protocol: Session IDs

Another way to correct the *third message replay* problem

- **Instead of using timestamps**
  - Use a random integer, *n*, that is associated with all messages in the key exchange
- **The protocol is altered slightly**
  - Alice first sends a message to Bob
    - The message contains the session ID & nonce encrypted with Alice's secret key
  - Bob forwards the message to Trent
    - And creates a message containing a nonce & the same session ID encrypted with Bob's secret key
  - Trent creates a session key & encrypts it for both Alice and for Bob

Otway-Rees Protocol

Use nonces (r_1, r_2) & session IDs (n)

Alice sends the communication request to Bob – with the session ID

$$\text{Alice} \xrightarrow{\;n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob}\}\, k_A\;} \text{Bob}$$

$$\text{Trent} \xleftarrow[\;\{r_2 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob}\}\, k_B\;]{\;n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob}\}\, k_A\;} \text{Bob}$$

Bob authenticates himself & forwards request to Trent

$$\text{Trent} \xrightarrow{\;n \,\|\, \{r_1 \,\|\, k_{A,B}\}\, k_A \,\|\, \{r_2 \,\|\, k_{A,B}\}\, k_B\;} \text{Bob}$$

$$\text{Alice} \xleftarrow{\;n \,\|\, \{r_1 \,\|\, k_{A,B}\}\, k_A\;} \text{Bob}$$

Kerberos

Kerberos

- **Authentication service developed by MIT**
  - project Athena 1983-1988
- **Uses a trusted third party & symmetric cryptography**
- **Based on Needham Schroeder with the Denning Sacco modification**
- **Passwords not sent in clear text**
  - assumes only the network can be compromised
- **Supported in most all popular operating systems**
  - Default network authentication used in Microsoft Windows
  - Supported in macOS, Linux, FreeBSD, z/OS, …
  - Used by Rutgers to store NetIDs via the Central Authentication Service (CAS)

Kerberos

Users and services authenticate themselves to each other

To access a service:

- User presents a ticket issued by the Kerberos authentication server
- Service uses the ticket to verify the identity of the user

Kerberos is a trusted third party

- Knows all (users and services) passwords
- Responsible for
  - Authentication: validating an identity
  - Authorization: deciding whether someone can access a service
  - Key distribution: giving both parties an encryption key (securely)

Kerberos – General Flow

User *Alice* wants to communicate with a service *Bob*

Both Alice and Bob have keys – Kerberos has copies

- key = *hash*(password)

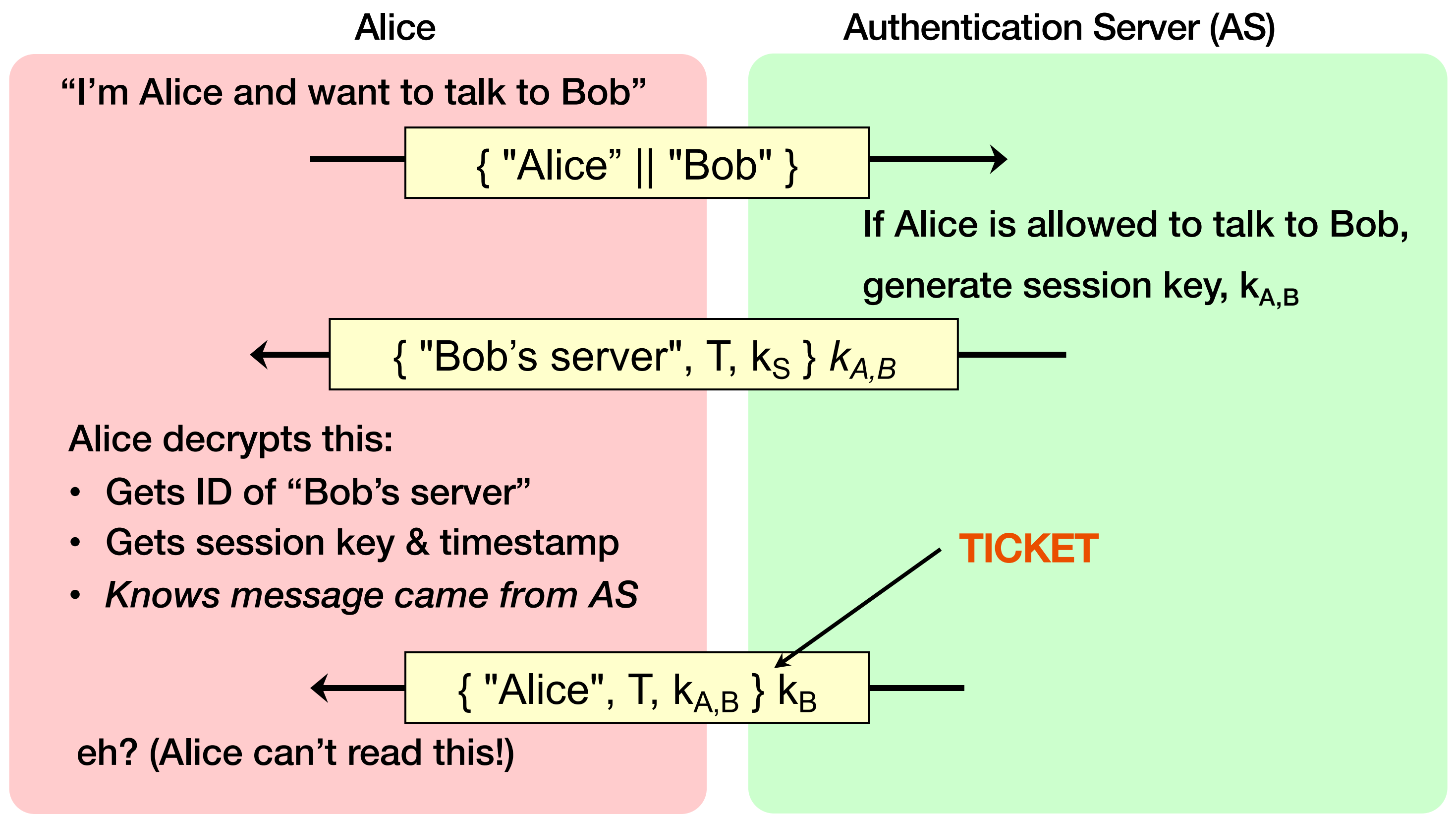
Step 1:

- Alice authenticates with Kerberos server
  - Gets *session key* and *ticket*

Step 2:

- Alice gives Bob the ticket, which contains the session key
- Convinces Bob that she got the session key from Kerberos

Kerberos (1): Authorize, Authenticate



Alice

Authentication Server (AS)

"I'm Alice and want to talk to Bob"

{ "Alice" || "Bob" } →

If Alice is allowed to talk to Bob, generate session key, $k_{A,B}$

← { "Bob's server", T, k_S } $k_{A,B}$

Alice decrypts this:

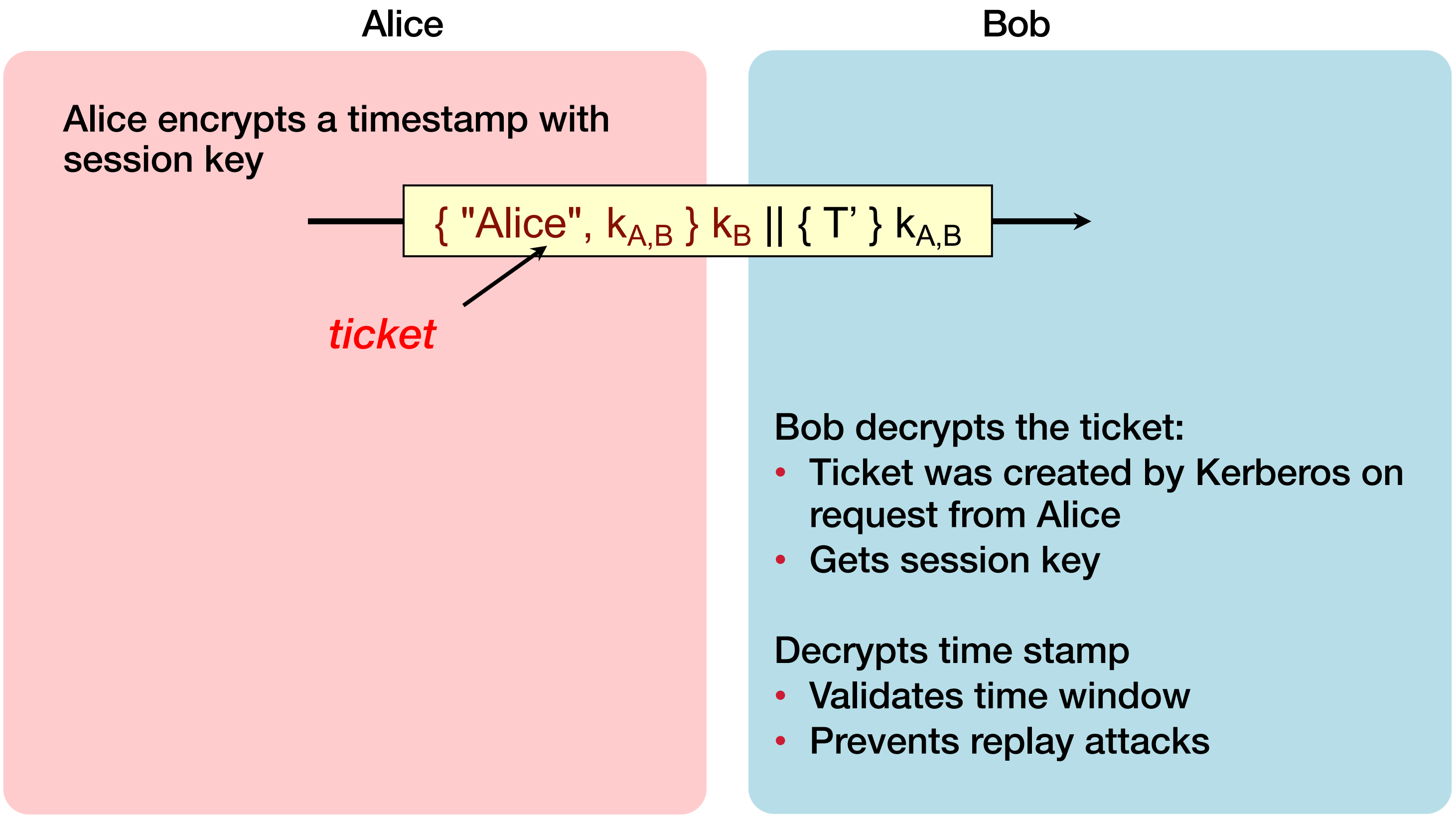
- Gets ID of "Bob's server"
- Gets session key & timestamp
- *Knows message came from AS*

TICKET

← { "Alice", T, $k_{A,B}$ } k_B

eh? (Alice can't read this!)

Kerberos (2): Send key



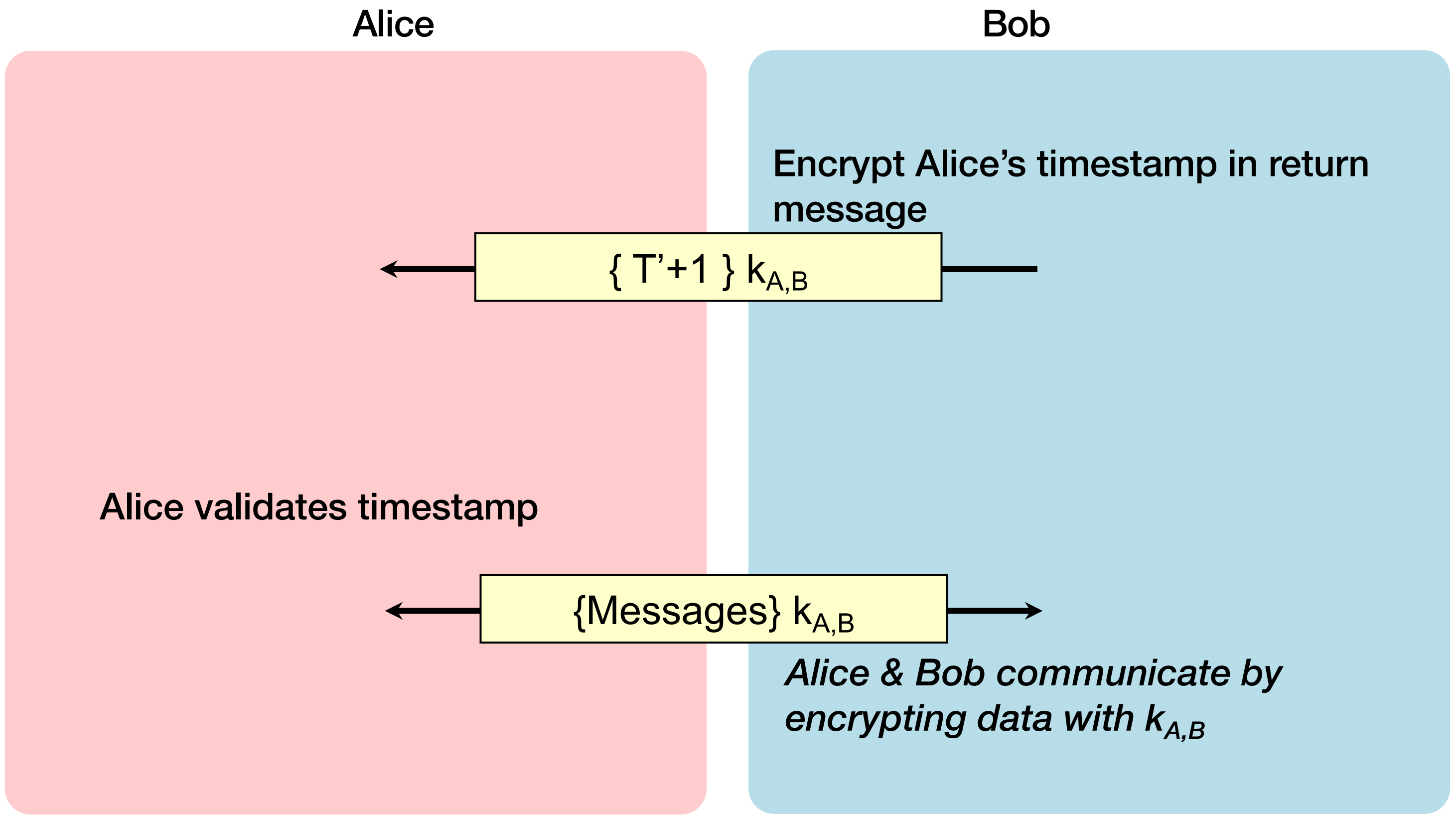
Alice encrypts a timestamp with session key

$\{ \text{"Alice"}, k_{A,B} \} k_B \,||\, \{ T' \} k_{A,B}$

Bob decrypts the ticket:
- Ticket was created by Kerberos on request from Alice
- Gets session key

Decrypts time stamp
- Validates time window
- Prevents replay attacks

Kerberos (3): Authenticate recipient of message



Encrypt Alice's timestamp in return message

Alice validates timestamp

Alice & Bob communicate by encrypting data with $k_{A,B}$

Kerberos key usage

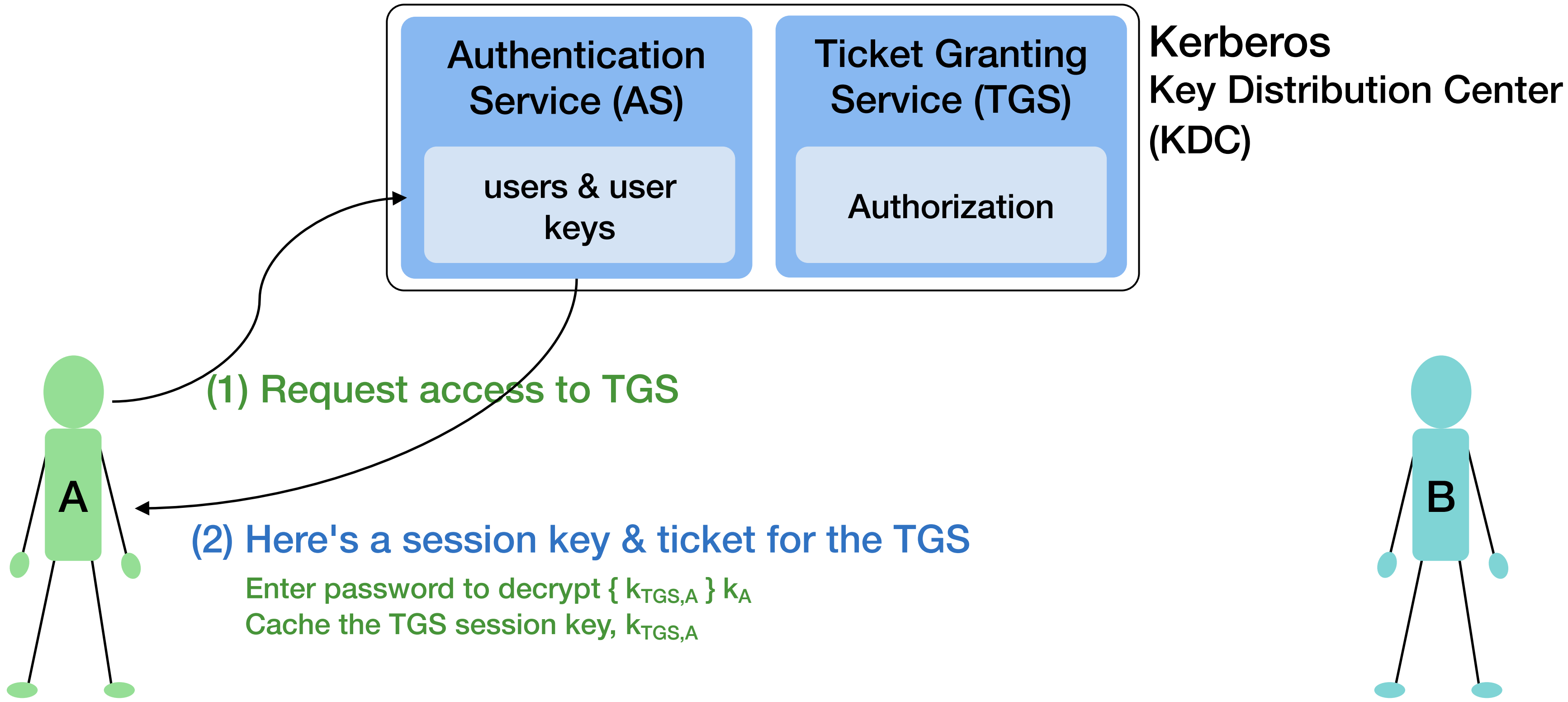
- **Every time a user wants to access a service**
  - User's password (key) must be used to decode the message from Kerberos
- **We can avoid this by caching the password in a file**
  - Not a good idea
- **Another way: create a temporary password**
  - We can cache this temporary password
  - It's just a session key to access Kerberos – to get access to other services
  - Split Kerberos server into

Authentication Service + Ticket Granting Service

Ticket Granting Server (TGS)

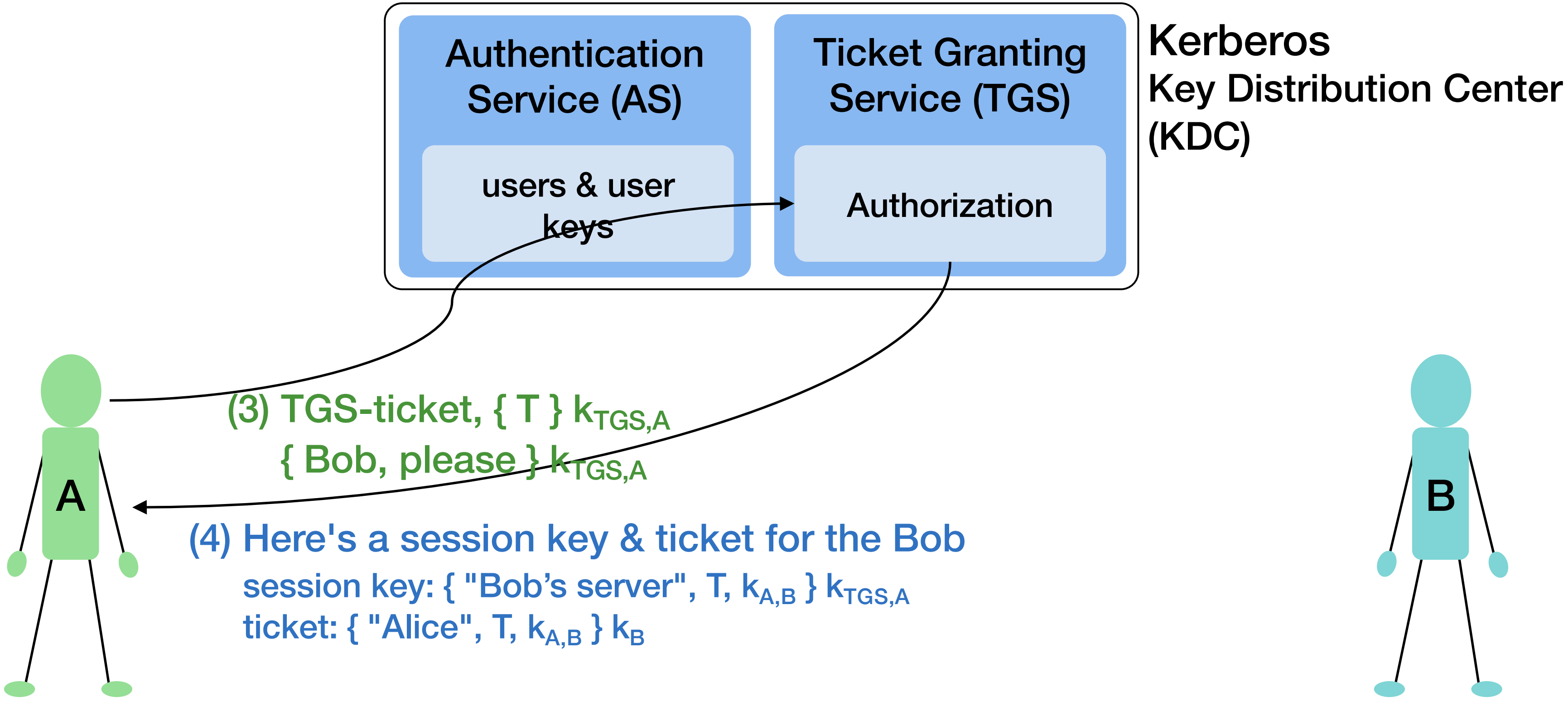
- **TGS works like a temporary ID**
- **User first requests access to the TGS**
  - Contact Kerberos Authentication Service (AS knows all users & their keys)
    - Gets back a ticket & session key to the TGS – these can be cached
- **To access any service**
  - Send a request to the TGS – encrypted with the TGS session key along with the ticket for the TGS
  - The ticket tells the TGS what your session key is
  - It responds with a session key & ticket for that service

Kerberos AS + TGS: Step 1


Authentication Service (AS)
users & user keys
Ticket Granting Service (TGS)
Authorization
Kerberos
Key Distribution Center (KDC)
(1) Request access to TGS
(2) Here's a session key & ticket for the TGS
A
B

Enter password to decrypt { $k_{TGS,A}$ } k_A
Cache the TGS session key, $k_{TGS,A}$

Kerberos AS + TGS



Authentication Service (AS)
users & user keys
Ticket Granting Service (TGS)
Authorization
Kerberos
Key Distribution Center (KDC)
A
B

(3) TGS-ticket, { T } $k_{TGS,A}$

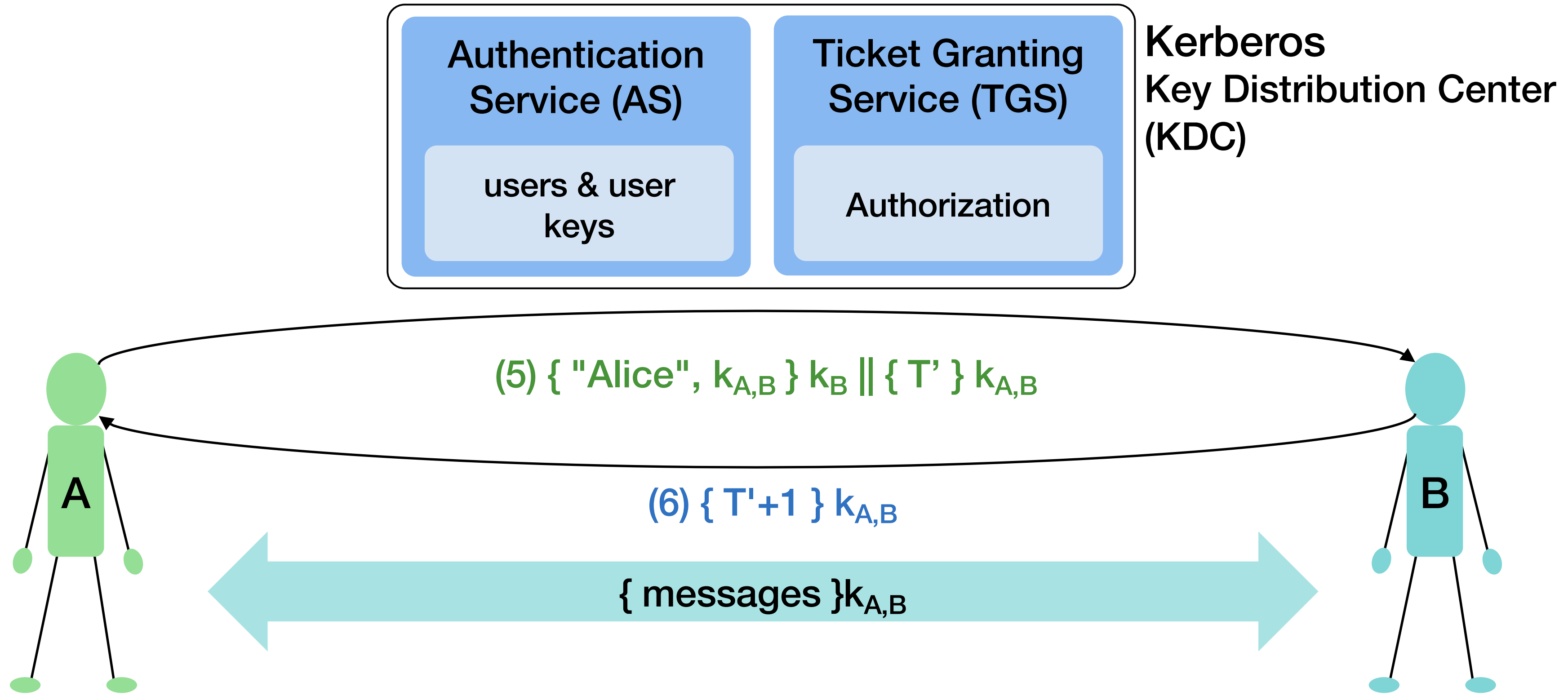
{ Bob, please } $k_{TGS,A}$

(4) Here's a session key & ticket for the Bob

session key: { "Bob's server", T, $k_{A,B}$ } $k_{TGS,A}$

ticket: { "Alice", T, $k_{A,B}$ } k_B

Kerberos AS + TGS



Authentication Service (AS)
users & user keys
Ticket Granting Service (TGS)
Authorization
Kerberos Key Distribution Center (KDC)
(5) { "Alice", $k_{A,B}$ } k_B || { T' } $k_{A,B}$
(6) { T'+1 } $k_{A,B}$
{ messages }$k_{A,B}$
A
B

Using Kerberos

```
$ kinit
```

Password: *enter password*

ask AS for permission (session key) to access TGS

Alice gets:

$\{\text{"TGS"}, T, k_{A,TGS}\}\, k_A$ ← *Session key & encrypted timestamp*

$\{\text{"Alice"}, k_{A,TGS}\}\, k_{TGS}$ ← *TGS Ticket*

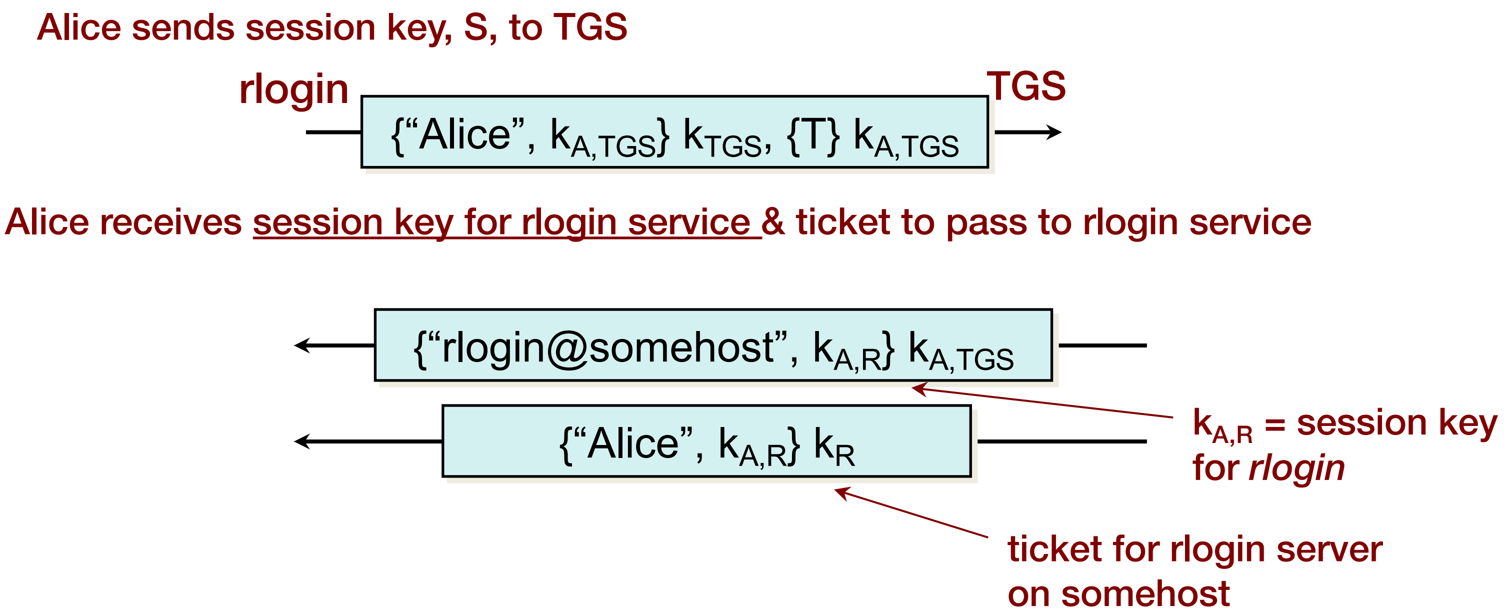
Compute key (A) from password to decrypt session key $k_{A,TGS}$ and get TGS ID.

You now have a ticket to access the Ticket Granting Service

Using Kerberos

```
$ rlogin somehost
```

rlogin uses the TGS Ticket to request a ticket for the *rlogin* service on *somehost*



Alice sends session key, S, to TGS
rlogin
TGS
{"Alice", $k_{A,TGS}$} k_{TGS}, {T} $k_{A,TGS}$
Alice receives session key for rlogin service & ticket to pass to rlogin service
{"rlogin@somehost", $k_{A,R}$} $k_{A,TGS}$
{"Alice", $k_{A,R}$} k_R
$k_{A,R}$ = session key for *rlogin*
ticket for rlogin server on somehost

Combined authentication & key exchange

Basic idea with symmetric cryptography:

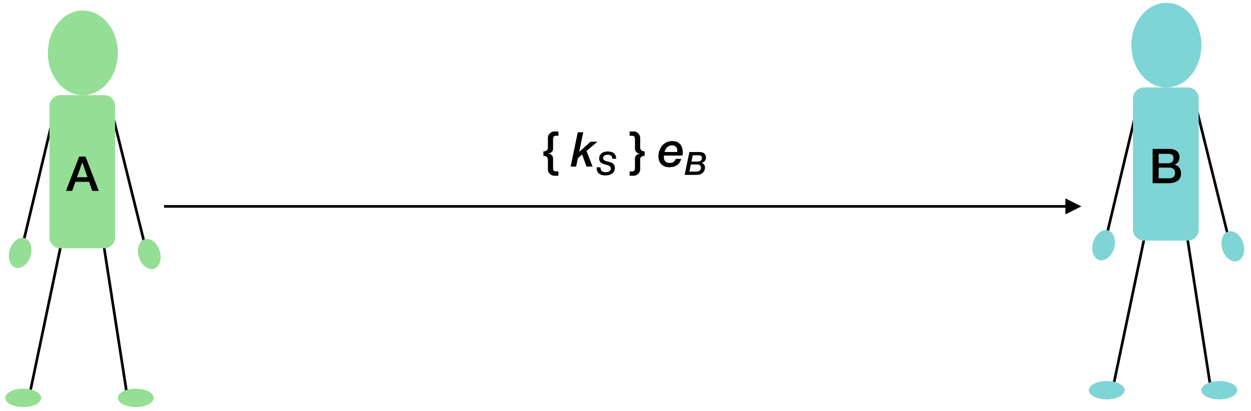
Use a trusted third party (Trent) that has all the keys

- **Alice wants to talk to Bob: she asks Trent**
  - Trent generates a session key encrypted for Alice
  - Trent encrypts the same key for Bob (ticket)
- **Authentication is implicit:**
  - If Alice can decrypt the session key, she proved she knows her key
  - If Alice can decrypt the session key, he proved he knows his key
- **Weaknesses that we had to fix:**
  - Replay attacks – add nonces – Needham-Schroeder protocol
  - Replay attacks re-using a cracked old session key
    - Add timestamps: Denning-Sacco protocol, Kerberos
    - Add session IDs at each step: Otway-Rees protocol

Public Key Exchange

We saw how this works…

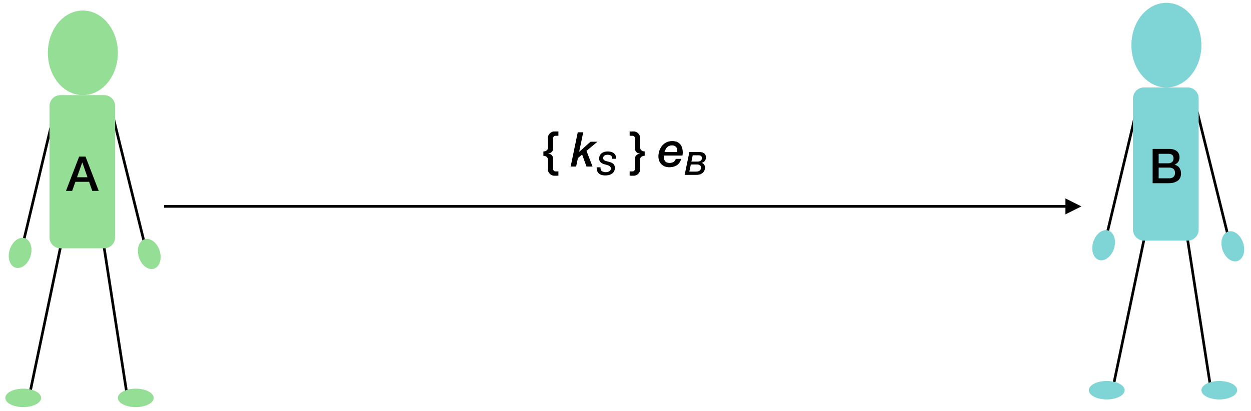
- Alice's & Bob's public keys known to all: e_A, e_B
- Alice & Bob's private keys are known only to the owner: d_a, d_b
- Simple protocol to send symmetric session key: k_S



Public Key Exchange

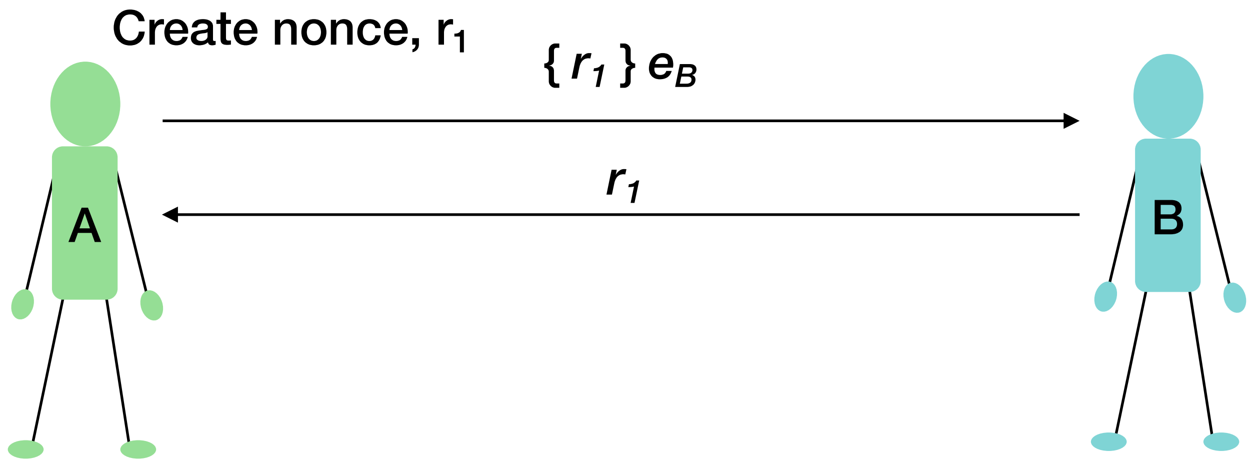
We saw how this works…

- Alice's & Bob's public keys known to all: e_A, e_B
- Alice & Bob's private keys are known only to the owner: d_a, d_b
- Simple protocol to send symmetric session key: k_S



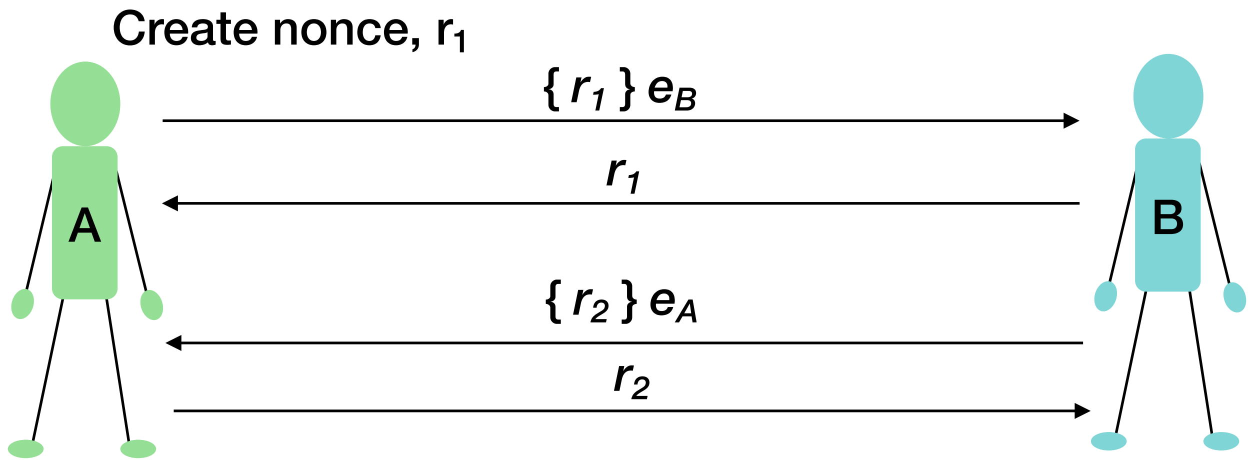
Adding authentication

- **Have Bob prove that he has the private key**
  - Same way as with symmetric cryptography – prove he can encrypt or decrypt



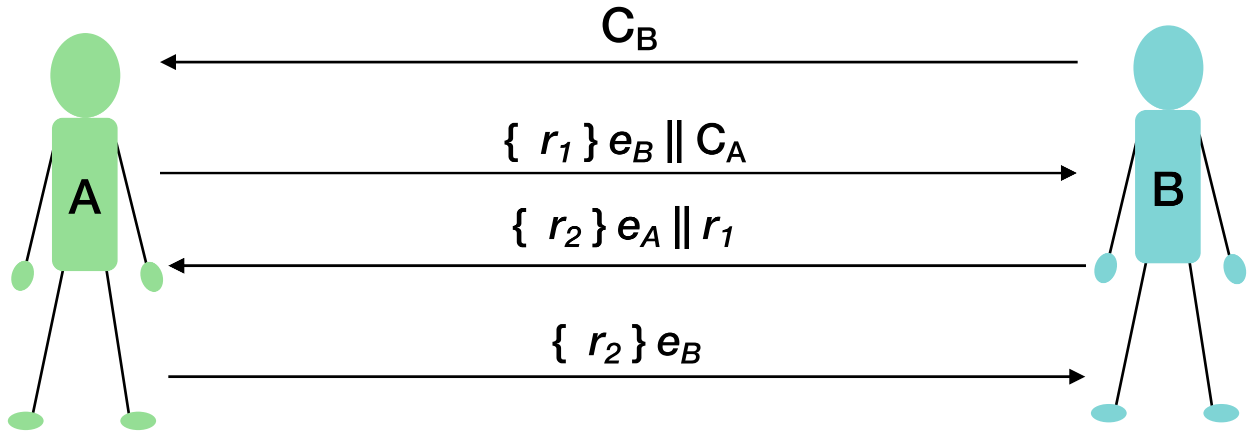
Adding authentication

- **Have Bob prove that he has the private key**
  - Same way as with symmetric cryptography – prove he can encrypt or decrypt



Adding identity binding

- **How do we know we have the right public keys?**
- **Get the public key from a trusted certificate**
  - Validate the signature on the certificate



Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- Hash functions
- Random number generators

User Authentication

Three Factors of Authentication

1. Ownership **Something you have**	*Key, card*	*Can be stolen*
2. Knowledge **Something you know**	*Passwords, PINs*	*Can be guessed, shared, stolen*
3. Inherence **Something you are**	*Biometrics (face, fingerprints)*	*Requires hardware* *May be copied* *Not replaceable if lost or stolen*

Multi-Factor Authentication

Factors may be combined

- **ATM machine: 2-factor authentication (2FA)**

– ATM card	something you have
– PIN	something you know

- **Password + code delivered via SMS: 2-factor authentication**

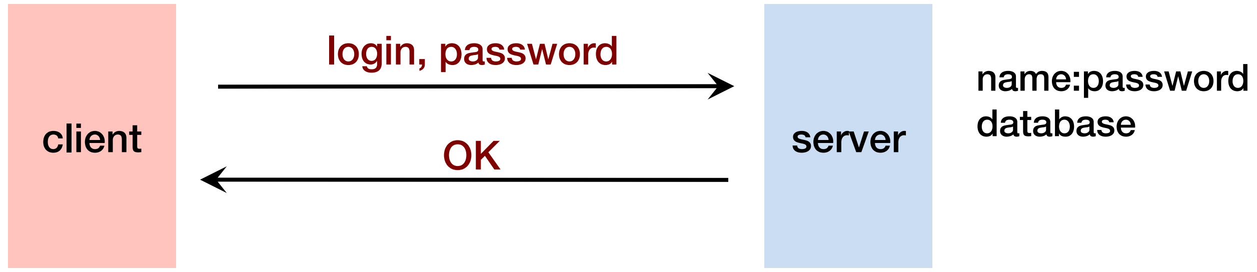
– Password	something you know
– Code	something you have: your phone

Two passwords ≠ Two-factor authentication

The factors must be different

Authentication: PAP

Password Authentication Protocol



- Unencrypted, reusable passwords
- Insecure on an open network
- Also, the password file must be protected from open access
  - But administrators can still see everyone's passwords
    What if you use the same password on Facebook as on Amazon?

Passwords are bad

- **Human readable & easy to guess**
  - People usually pick really bad passwords
- **Easy to forget**
- **Usually short**
- **Static … reused over & over**
  - Security is as strong as the weakest link
  - If a username (or email) & password is stolen from one server, it might be usable on others
- **Replayable**
  - If someone can grab it or see it, they can play it back

It's not getting better

Recent large-scale leaks of password from servers have shown that people DO NOT pick good passwords

Rank	2012	2013	2014	2015	2016	2017	2018	2019
1	password	123456	123456	123456	123456	123456	123456	123456
2	123456	password	password	password	password	password	password	123456789
3	12345678	12345678	12345	12345678	12345	12345678	123456789	qwerty
4	abc123	qwerty	12345678	qwerty	12345678	qwerty	12345678	password
5	qwerty	abc123	qwerty	12345	football	12345	12345	1234567
6	monkey	123456789	123456789	123456789	qwerty	123456789	111111	12345678
7	letmein	111111	1234	football	1234567890	letmein	1234567	12345
8	dragon	1234567	baseball	1234	1234567	1234567	sunshine	iloveyou

Past eight years of top passwords from SplashData's list

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

Policies to the rescue?

Password rules

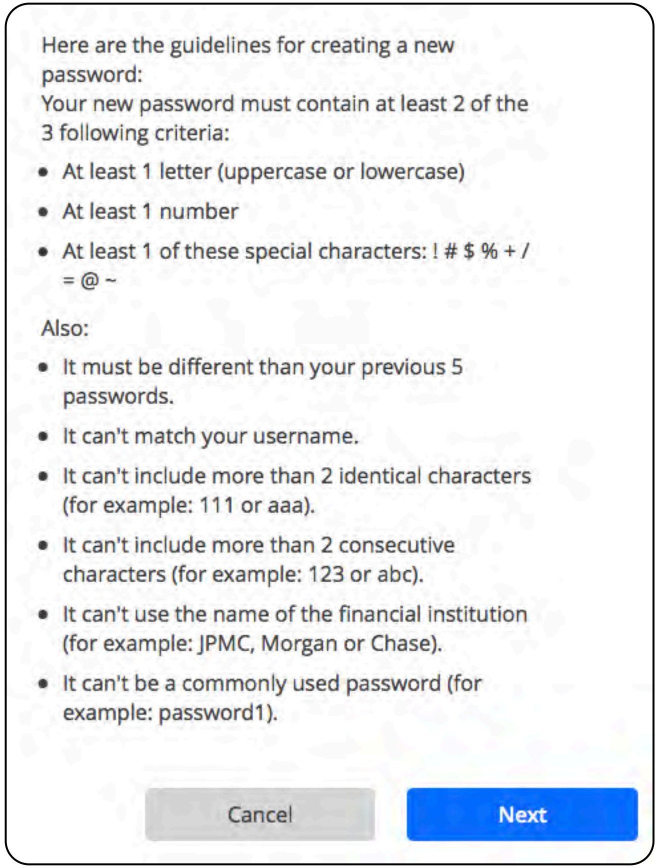
“Everyone knows that an exclamation point is a 1, or an I, or the last character of a password. $ is an S or a 5. If we use these well-known tricks, we aren’t fooling any adversary. We are simply fooling the database that stores passwords into thinking the user did something good”

— Paul Grassi, NIST

Periodic password change requirement problems

– People tend to change passwords rapidly to exhaust the history list and get back to their favorite password
– Forbidding changes for several days enables a denial of service attack
– People pick worse passwords, incorporating numbers, months, or years

https://fortune.com/2017/05/11/password-rules/
https://pages.nist.gov/800-63-3/sp800-63b.html#sec5


Here are the guidelines for creating a new password:
Your new password must contain at least 2 of the 3 following criteria:

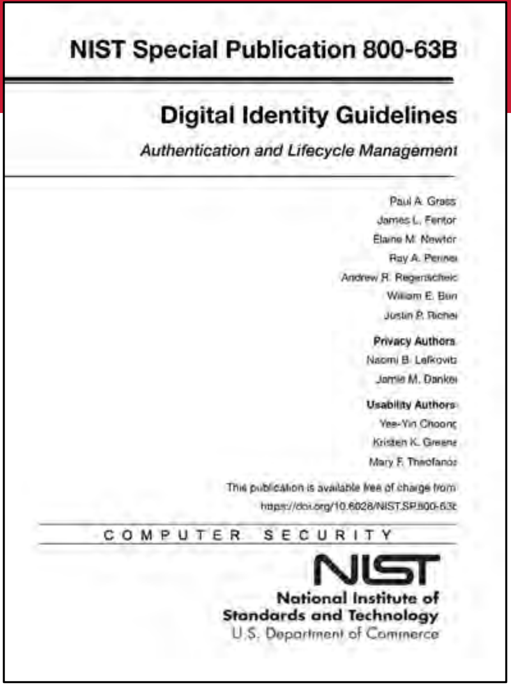
- At least 1 letter (uppercase or lowercase)
- At least 1 number
- At least 1 of these special characters: ! # $ % + / = @ ~

Also:

- It must be different than your previous 5 passwords.
- It can't match your username.
- It can't include more than 2 identical characters (for example: 111 or aaa).
- It can't include more than 2 consecutive characters (for example: 123 or abc).
- It can't use the name of the financial institution (for example: JPMC, Morgan or Chase).
- It can't be a commonly used password (for example: password1).

NIST recommendations

- **Remove periodic password change requirements**
- **Drop complexity requirements (numbers, letters, symbols)**
- **Choose long passwords**
- **Avoid**
  - Passwords obtained from databases of previous breaches
  - Dictionary words
  - Repetitive or sequential characters (e.g. ‘`aaaaa`’, ‘`1234abcd`’)
  - Context-specific words, such as the name of the service, the username, and derivatives thereof



NIST Special Publication 800-63B

Digital Identity Guidelines

Authentication and Lifecycle Management

Paul A. Grass
James L. Fentor
Elaine M. Newtor
Ray A. Perlnei
Andrew R. Regenscheic
William E. Burr
Justin P. Richei

Privacy Authors
Naomi B. Lefkovitz
Jamie M. Dankei

Usability Authors
Yee-Yin Choong
Kristen K. Greene
Mary F. Theofanos

This publication is available free of charge from
https://doi.org/10.6028/NIST.SP.800-63t

COMPUTER SECURITY

NIST
National Institute of
Standards and Technology
U.S. Department of Commerce

https://pages.nist.gov/800-63-3/sp800-63b.html

PAP: Reusable passwords

Problem #1: Open access to the password file

What if the password file isn't sufficiently protected and an intruder gets hold of it? All passwords are now compromised!

Even if an admin sees your password, this might also be your password on other systems.

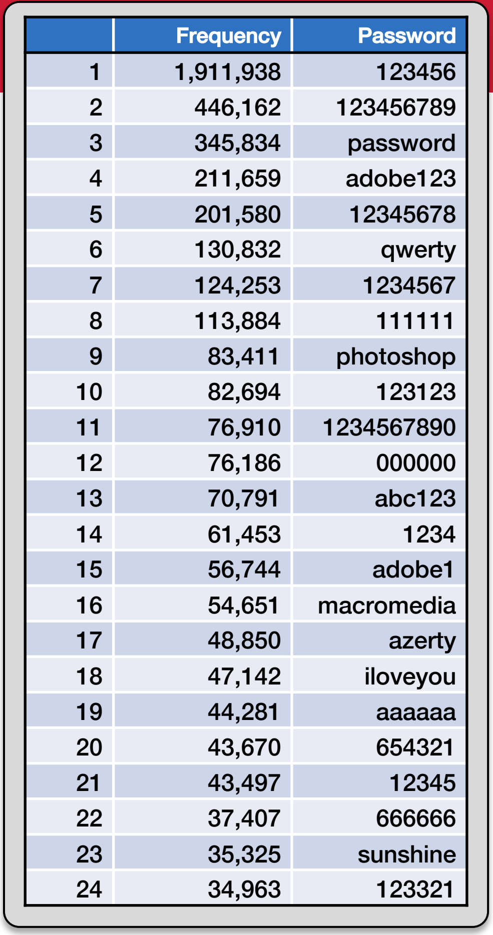
How about encrypting the passwords?

- **Where would you store the key?**
- **Adobe did that**
  - 2013 Adobe security breach leaked 152 million Adobe customer records
  - Adobe used encrypted passwords
    - But the **passwords were all encrypted with the same key**
    - If the attackers steal the key, they get the passwords

Poor Password Management

Adobe security breach (November 2013)

- 152 million Adobe customer records …
  with encrypted passwords
- Adobe encrypted passwords with a symmetric key algorithm
  … and used the same key to encrypt every password!



	Frequency	Password
1	1,911,938	123456
2	446,162	123456789
3	345,834	password
4	211,659	adobe123
5	201,580	12345678
6	130,832	qwerty
7	124,253	1234567
8	113,884	111111
9	83,411	photoshop
10	82,694	123123
11	76,910	1234567890
12	76,186	000000
13	70,791	abc123
14	61,453	1234
15	56,744	adobe1
16	54,651	macromedia
17	48,850	azerty
18	47,142	iloveyou
19	44,281	aaaaaa
20	43,670	654321
21	43,497	12345
22	37,407	666666
23	35,325	sunshine
24	34,963	123321

Top 26 Adobe Passwords

PAP: Reusable passwords

Solution:

Store a hash of the password in a file

- Given a file, you don't get the passwords
- Have to resort to a dictionary or brute-force attack
- Example, passwords hashed with SHA-512 hashes (SHA-2)

What is a dictionary attack?

- **Suppose you got access to a list of hashed passwords**
- **Brute-force, exhaustive search: try every combination**
  - Letters (A-Z, a-z), numbers (0-9), symbols (!@#$%...)
  - Assume 30 symbols + 52 letters + 10 digits = 92 characters
  - Test all passwords up to length 8
  - Combinations = $92^8 + 92^7 + 92^6 + 92^5 + 92^4 + 92^3 + 92^2 + 92^1 = 5.189 \times 10^{15}$
  - If we test 1 billion passwords per second: ≈ 60 days
- **But some passwords are more likely than others**
  - 1,991,938 Adobe customers used a password = “123456”
  - 345,834 users used a password = “password”
- **Dictionary attack**
  - Test lists of common passwords, dictionary words, names
  - Add common substitutions, prefixes, and suffixes

Easiest to do if the attacker steals a hashed password file – so we read-protect the hashed passwords to make it harder to get them

How to speed up a dictionary attack

Create a table of precomputed hashes

Now we just search a table for the hash to find the password

SHA-256 Hash	password
8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92	123456
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8	password
ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532cc95c5ed7a898a64f	12345678
1c8bfe8f801d79745c4631d09fff36c82aa37fc4cce4fc946683d7b336b63032	letmein
…	…

Salt: defeating dictionary attacks

Salt = random string (typically up to 16 characters)

- Concatenated with the password
- Stored with the password file (it's not secret)

"`am$7b22QL`" + "`password`"

Even if you know the salt, you cannot use precomputed hashes to search for a password (because the salt is prefixed to the password string and becomes part of the hash)

Example:

SHA-256 hash of "`password`", salt = "`am$7b22QL`"= *hash*("`am$7b22QLpassword`") =

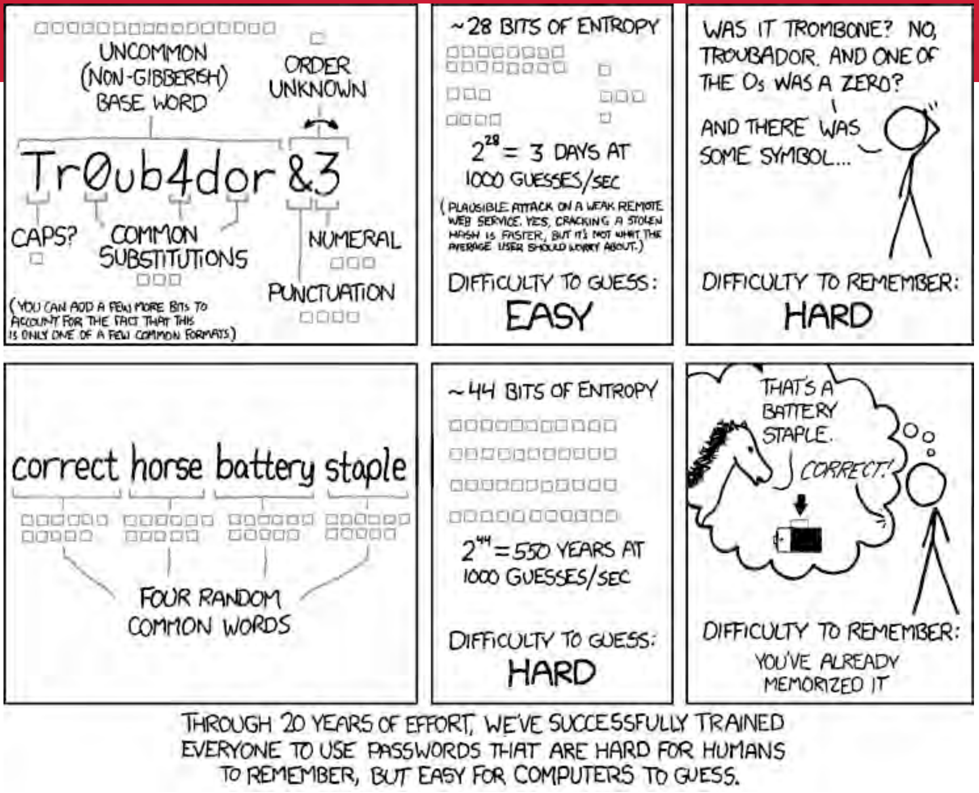
`076bb015496e1db7b57e4c3b1c34d69504358b13fae4c50270cf40dfa92a1996`

You will __not__ have a precomputed hash("am$7b22QLpassword")

Longer passwords

English text has an entropy of about 1.2-1.5 bits per character

Random text has an entropy ≈ $\log_2(1/95)$ ≈ 6.6 bits/character


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Assume 95 printable characters

Defenses

- **Use longer passwords**
  - But can you trust users to pick ones with enough entropy?
- **Rate-limit guesses**
  - Add timeouts after an incorrect password
    - Linux waits about 3 secs – and terminates the *login* program after 5 tries
- **Lock out the account after *N* bad guesses**
  - But this makes you vulnerable to denial-of-service attacks
- **Use a slow algorithm to make guessing slow**
  - OpenBSD *bcrypt* Blowfish password hashing algorithm

People forget passwords

- **Especially seldom-used ones – how do we handle that?**
- **Email them?**
  - Common solution
  - Requires that the server be able to get the password (can't store a hash)
  - What if someone reads your email?
- **Reset them?**
  - How do you authenticate the requester?
  - Usually send reset link to email address created at registration
  - But – what if someone reads your mail? …or you no longer have that address?
- **Provide hints?**
- **Write them down?**
  - OK if the threat model is electronic only

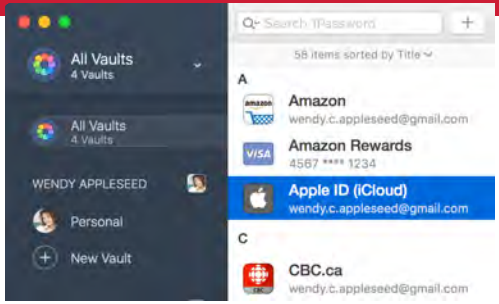
Reusable passwords in multiple places

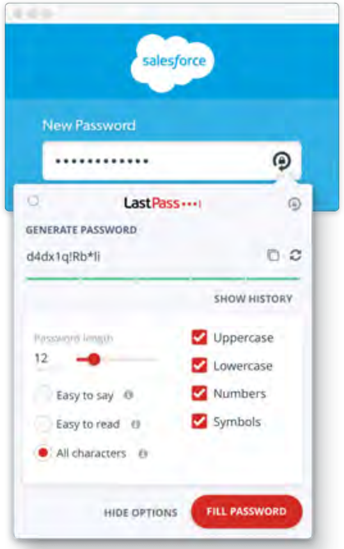
- **People often use the same password in different places**
- **If one site is compromised, the password can be used elsewhere**
  - People often try to use the same email address and/or username
- **This is the root of phishing attacks**

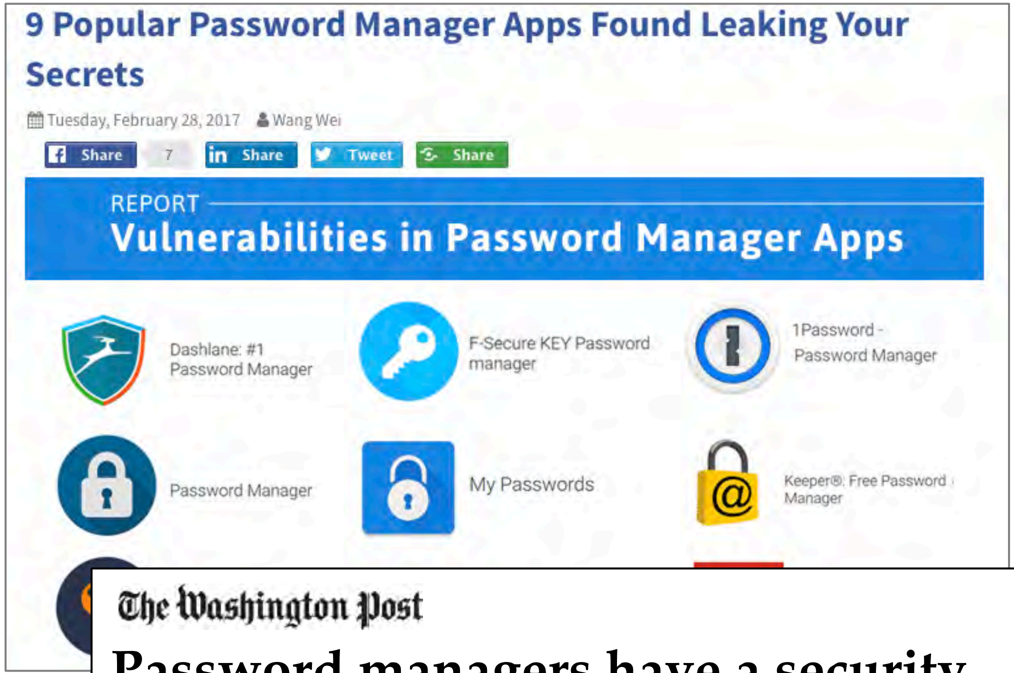
Password Managers

Software that stores passwords in an encrypted file

- **Do you trust the protection?**
  - The reputation of the company & its security policies
  - The synchronization capabilities?
- **Can malware get to the database?**
- **In general, these are good**
  - Way better than storing passwords in a file
  - Encourages having unique passwords per site
  - Password managers may have the ability to recognize web sites & defend against phishing







9 Popular Password Manager Apps Found Leaking Your Secrets

Tuesday, February 28, 2017 Wang Wei

Share
7
Share
Tweet
Share
REPORT
Vulnerabilities in Password Manager Apps
Dashlane: #1 Password Manager
F-Secure KEY Password manager
1Password - Password Manager
Password Manager
My Passwords
Keeper®: Free Password Manager

The Washington Post

Password managers have a security flaw. But you should still use one.

Exclusive: A new study finds bugs in five of the most popular password managers. So how is it safe to keep all your eggs in one basket?

By Geoffrey A. Fowler • Feb 19, 2019

THE VERGE

LastPass fixes bug that could let malicious websites extract your last used password

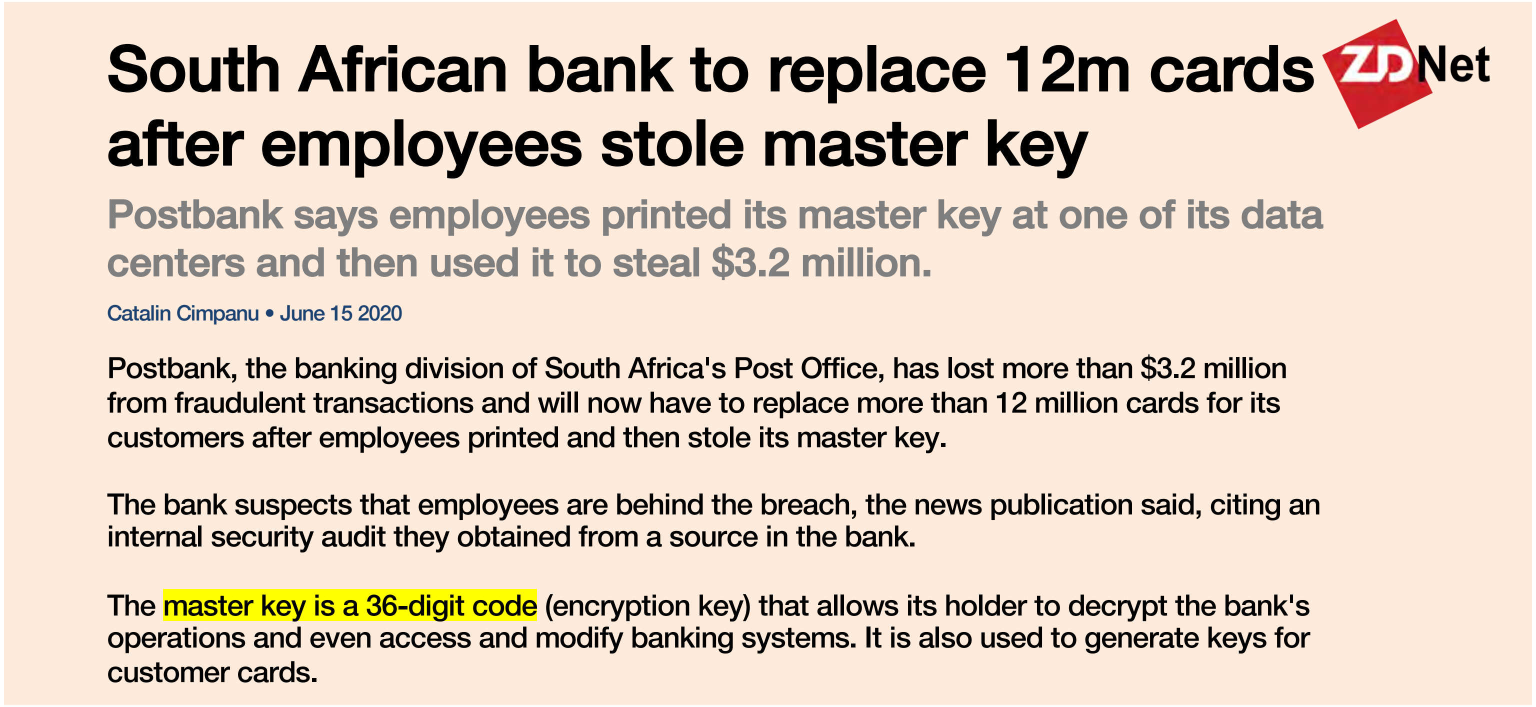
Even password managers have security bugs

By Jon Porter • Sep 16, 2019

LastPass has patched a bug that would have allowed a malicious website to extract a previous password entered by the service's browser extension. ZDNet reports that the bug was discovered by Tavis Ormandy, a researcher in Google's Project Zero team, and was disclosed in a bug report dated August 29th. LastPass fixed the issue on September 13th, and deployed the update to all browsers where it should be applied automatically, something LastPass users would be smart to verify.

Key management risks

Password managers are a form of key storage



South African bank to replace 12m cards after employees stole master key

ZDNet

Postbank says employees printed its master key at one of its data centers and then used it to steal $3.2 million.

Catalin Cimpanu • June 15 2020

Postbank, the banking division of South Africa's Post Office, has lost more than $3.2 million from fraudulent transactions and will now have to replace more than 12 million cards for its customers after employees printed and then stole its master key.

The bank suspects that employees are behind the breach, the news publication said, citing an internal security audit they obtained from a source in the bank.

The master key is a 36-digit code (encryption key) that allows its holder to decrypt the bank's operations and even access and modify banking systems. It is also used to generate keys for customer cards.

https://www.zdnet.com/article/south-african-bank-to-replace-12m-cards-after-employees-stole-master-key/

PAP: Reusable passwords

Problem #2: Network sniffing or shoulder surfing

Passwords can be stolen by observing a user's session in person or over a network:

- Snoop on http, telnet, ftp, rlogin, rsh sessions
- Trojan horse
- Social engineering
- Key logger, camera, physical proximity
- Brute-force or dictionary attacks

Solutions:

(1) **Use an encrypted communication channel**

(2) **Use one-time passwords**

(3) **Use multi-factor authentication, so a password alone is not sufficient**

One-time passwords

Use a different password each time

– If an intruder captures the transaction, it won't work next time

Three forms

1. **Sequence-based**: password = *f*(previous password)
2. **Time-based**: password = *f*(time, secret)
3. **Challenge-based**: *f*(challenge, secret)

S/key authentication

- One-time password scheme
- Produces a limited number of authentication sessions
- Relies on one-way functions

S/key authentication

<u>Authenticate Alice for 100 logins</u>

- pick random number, R
- using a one-way function (e.g., a hash function), $f(x)$:

$$\begin{aligned}
x_1 &= f(R) \\
x_2 &= f(x_1) = f(f(R)) \\
x_3 &= f(x_2) = f(f(f(R))) \\
&\dots \quad \dots \\
x_{100} &= f(x_{99}) = f(\dots f(f(f(R)))\dots)
\end{aligned}$$

Give this list to Alice

- then compute:

$$x_{101} = f(x_{100}) = f(\dots f(f(f(R)))\dots)$$

S/key authentication

<u>Authenticate Alice for 100 logins</u>

Store x_{101} in a password file or database record associated with Alice

alice: x_{101}

S/key authentication

Alice presents the *last* number on her list:

Alice to host: { “alice”, x_{100} }

Host computes $f(x_{100})$ and compares it with the value in the database

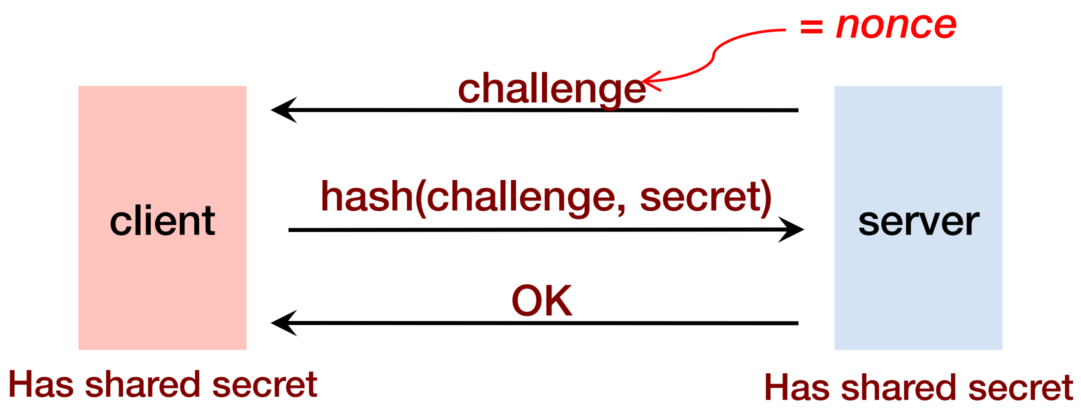
```
if  f(x100 provided by alice) = passwd(“alice”)
        replace x101 in db with x100 provided by alice
        return success
else
        fail
```

next time: Alice presents x_{99}

If someone sees x_{100} there is no way to generate x_{99}.

Authentication: CHAP

Challenge-Handshake Authentication Protocol

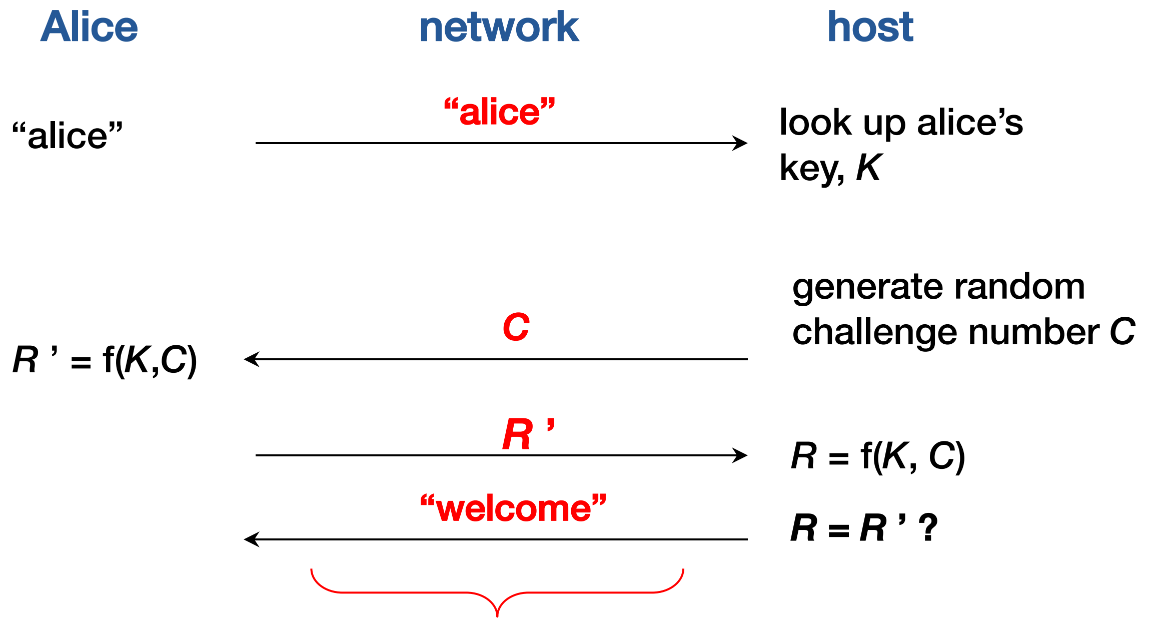


The challenge is a *nonce* (random bits).

We create a hash of the nonce and the secret.

An intruder does not have the secret and cannot do this!

CHAP authentication



an eavesdropper does not see K

SMS/Email Authentication

- **Second factor = your possession of a phone (or computer)**
- **After login, sever sends you a code via SMS (or email)**
- **Entering it is proof that you could receive the message**
- **Dangers**
  - – SIM swapping attacks (social engineering on the phone company)
    - Viable for high-value targets
  - – Social engineering to get email credentials

Time-Based Authentication

Time-based One-time Password (TOTP) algorithm

- **Both sides share a secret key**
  - Sometimes sent via a QR code so the user can scan it into the TOTP app
- **User runs TOTP function to generate a one-time password**

  one_time_password = *hash*(secret_key, time)

- **User logs in with:** ***name*, *password*, and *one_time_password***

- **Service generates the same password**

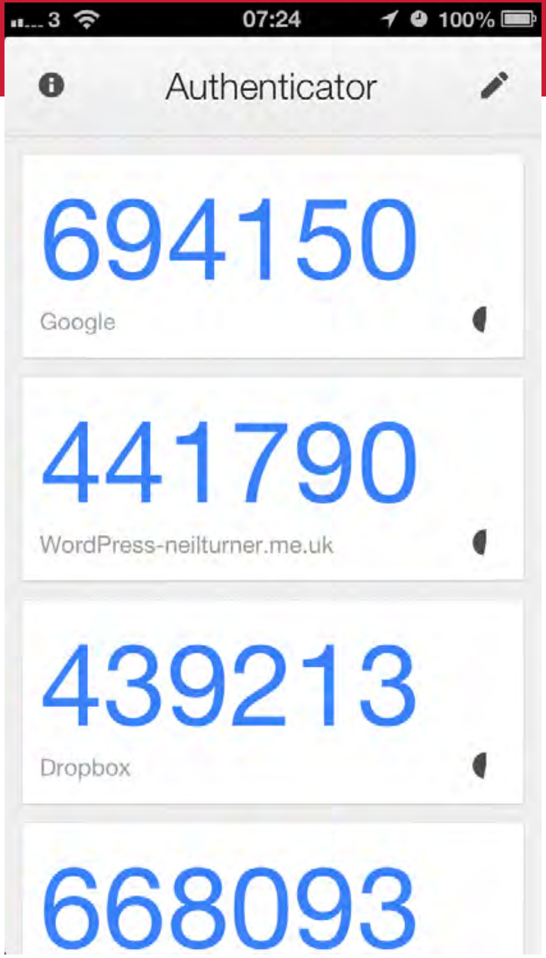
  one_time_password = *hash*(secret_key, time)

- **Typically 30-second granularity for time**

Time-based One-time Passwords

Used by

- Microsoft Two-step Verification
- Google Authenticator
- Facebook Code Generator
- Amazon Web Services
- Bitbucket
- Dropbox
- Evernote
- Zoho
- Wordpress
- 1Password
- Many others…



694150
Google

441790
WordPress-neilturner.me.uk

439213
Dropbox

668093

RSA SecurID card

Username:

```
paul
```

Password:

$$\underbrace{1234}_{\text{PIN}}\ \underbrace{032848}_{\text{passcode from card}}$$

PIN + passcode from card

Something you know

Something you have

Passcode changes every 60 seconds

1. Enter PIN
2. Press ◊
3. Card computes password
4. Read password & enter

Password:

```
354982
```

SecurID card

Same principle as Time-based One-Time Passwords

- **Proprietary device from RSA**
  - SASL mechanism: RFC 2808

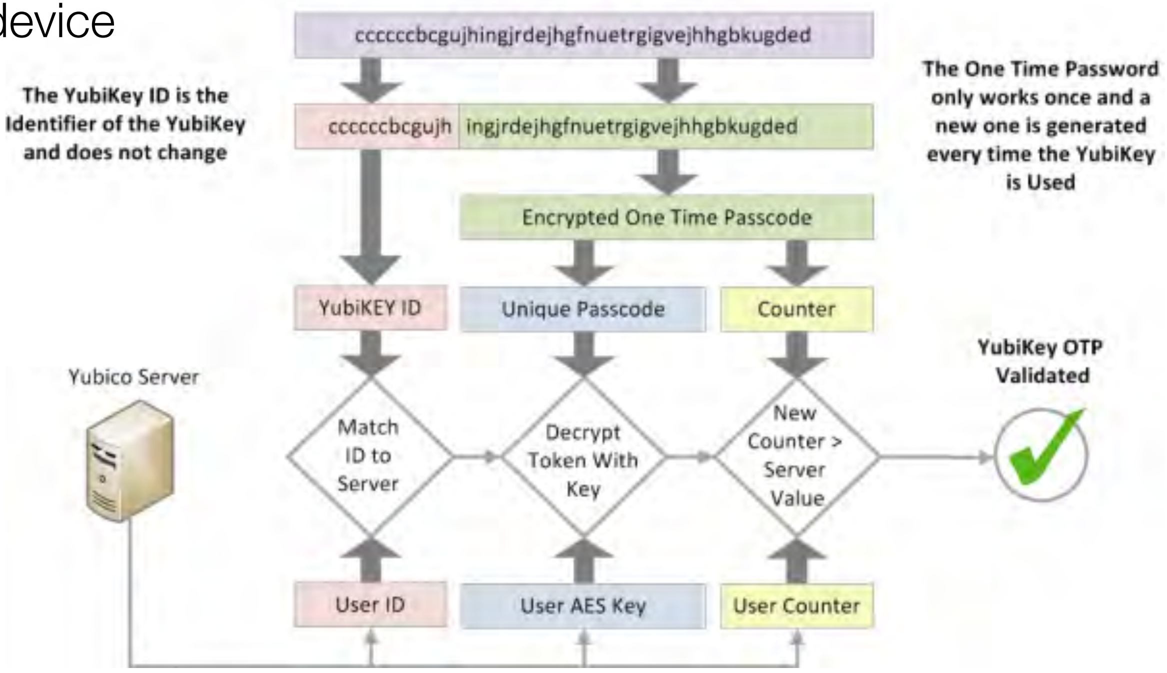
- **Two-factor authentication based on:**
  - Shared secret key (seed) ← **Something you have**
    - stored on authentication card
  - Shared personal ID – PIN ← **Something you know**
    - known by user

Yubikey: Yubico One Time Password

HOTP = Hash-based One-Time Password

OTP = *hash*(hardware_id, passcode, counter)

Passcode generated on the device
from session counters,
previous values,
other sources

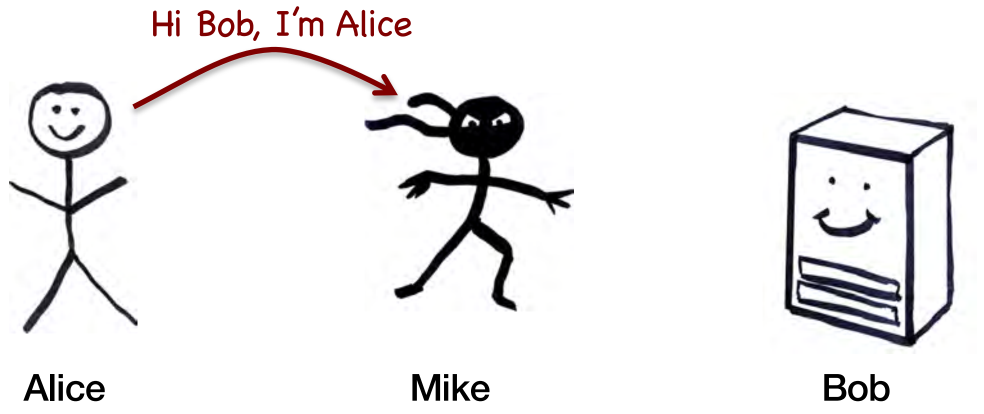


cccccbcgujhingjrdejhgfnuetrgigvejhhgbkugded
cccccbcgujh
ingjrdejhgfnuetrgigvejhhgbkugded
The YubiKey ID is the Identifier of the YubiKey and does not change
The One Time Password only works once and a new one is generated every time the YubiKey is Used
Encrypted One Time Passcode
YubiKEY ID
Unique Passcode
Counter
Yubico Server
Match ID to Server
Decrypt Token With Key
New Counter > Server Value
YubiKey OTP Validated
User ID
User AES Key
User Counter

Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

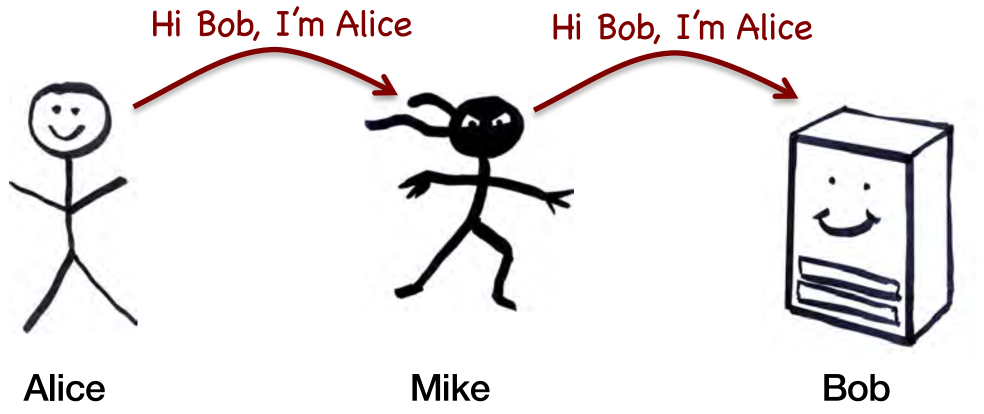
– Attacker acts as the server



Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

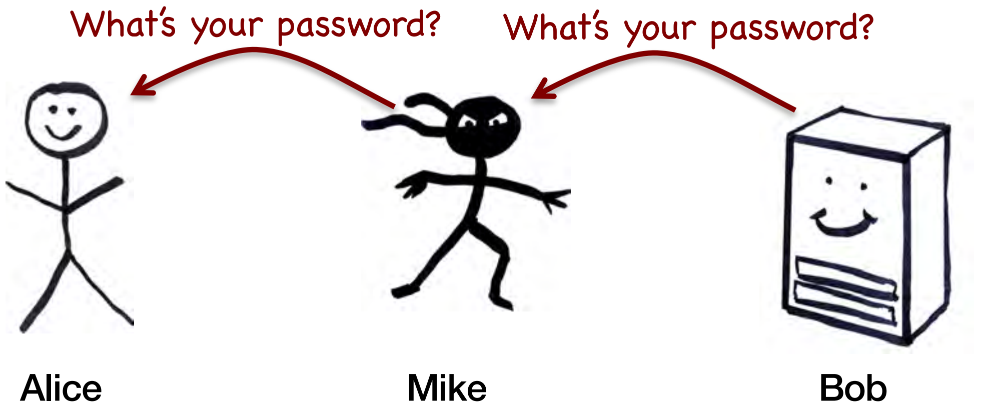
– Attacker acts as the server



Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

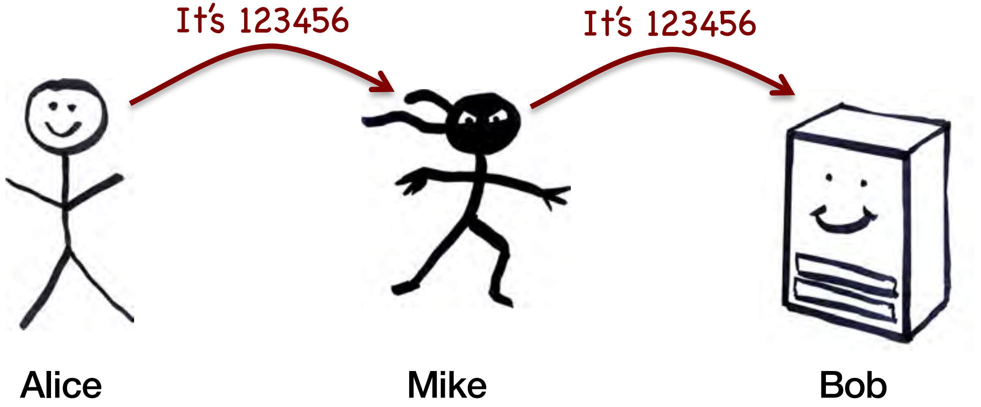
– Attacker acts as the server



Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

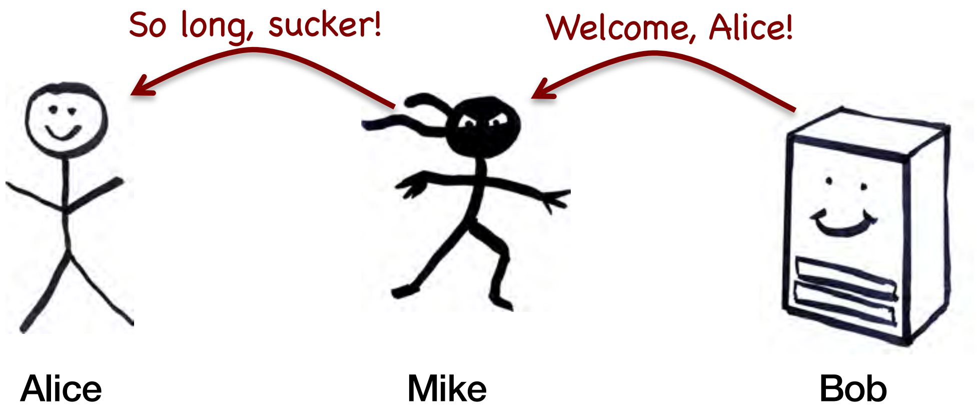
– Attacker acts as the server



Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

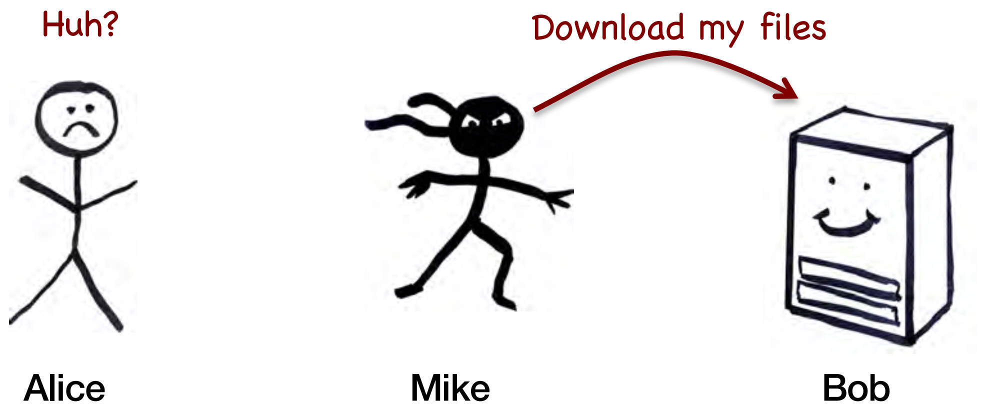
– Attacker acts as the server



Man-in-the-Middle Attacks

Password systems are vulnerable to man-in-the-middle attacks

– Attacker acts as the server



Guarding against man-in-the-middle attacks

- **Use a covert communication channel**
  - The intruder won't have the key
  - Can't see the contents of any messages
  - But you can't send the key over that channel!
- **Use signed messages for all communication**
  - Signed message = { message, private-key-encrypted hash of message }
  - Both parties can reject unauthenticated messages
  - The intruder cannot modify the messages
    - Signatures will fail (they will need to know how to encrypt the hash)
- **But watch out for replay attacks!**
  - May need to use session numbers or timestamps

Biometric Authentication

Biometrics

- Identify a person based on physical or behavioral characteristics

```
scanned_fingerprint = capture();
if (scanned_fingerprint == stored_fingerprint)
    accept_user();
else
    reject_user();
```

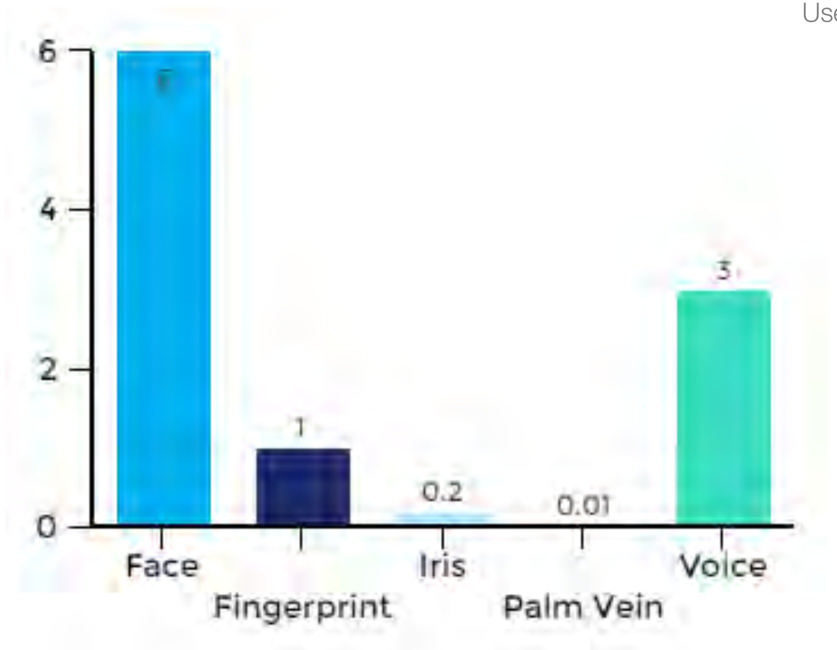
We'd like to use logic like this

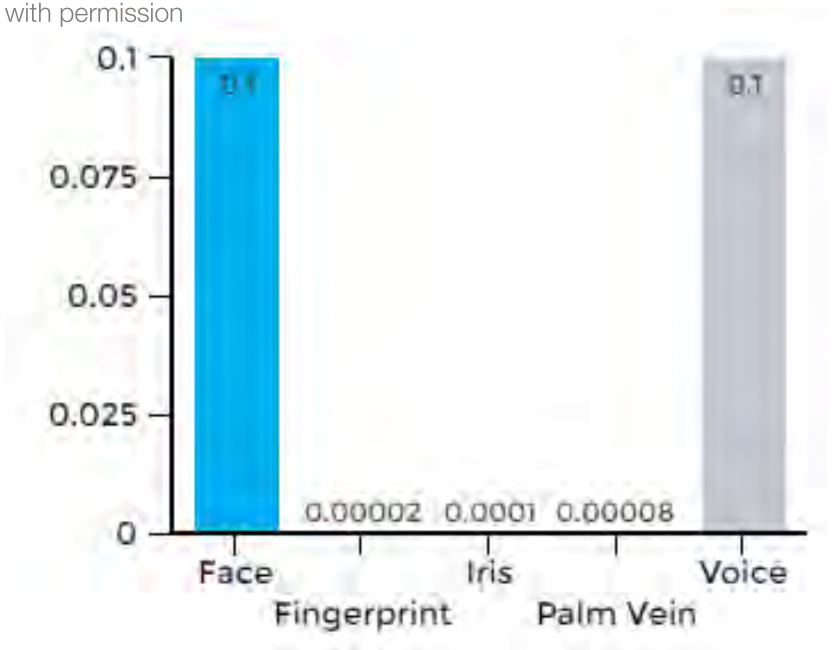
Biometrics

- **Rely on statistical pattern recognition**
  - Thresholds to determine if the match is close enough
- **False Accept Rate (FAR)**
  - Non-matching pair of biometric data is *accepted* as a match
- **False Reject Rate (FRR)**
  - Matching pair of biometric data is *rejected* as a match

FRR vs. FAR

Images from https://www.bayometric.com/biometrics-face-finger-iris-palm-voice/
Used with permission





False Rejection Rate (FRR)

Likelihood that the authentication will reject an authorized user

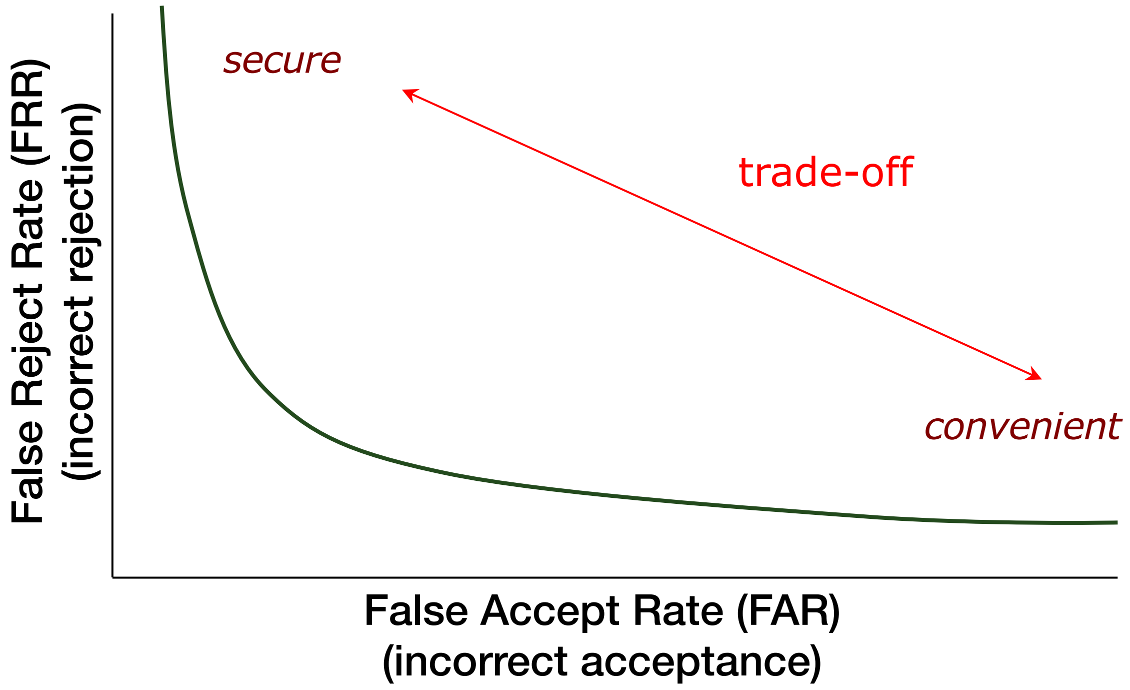
False Acceptance Rate (FAR)

Likelihood that the authentication will accept an unauthorized user

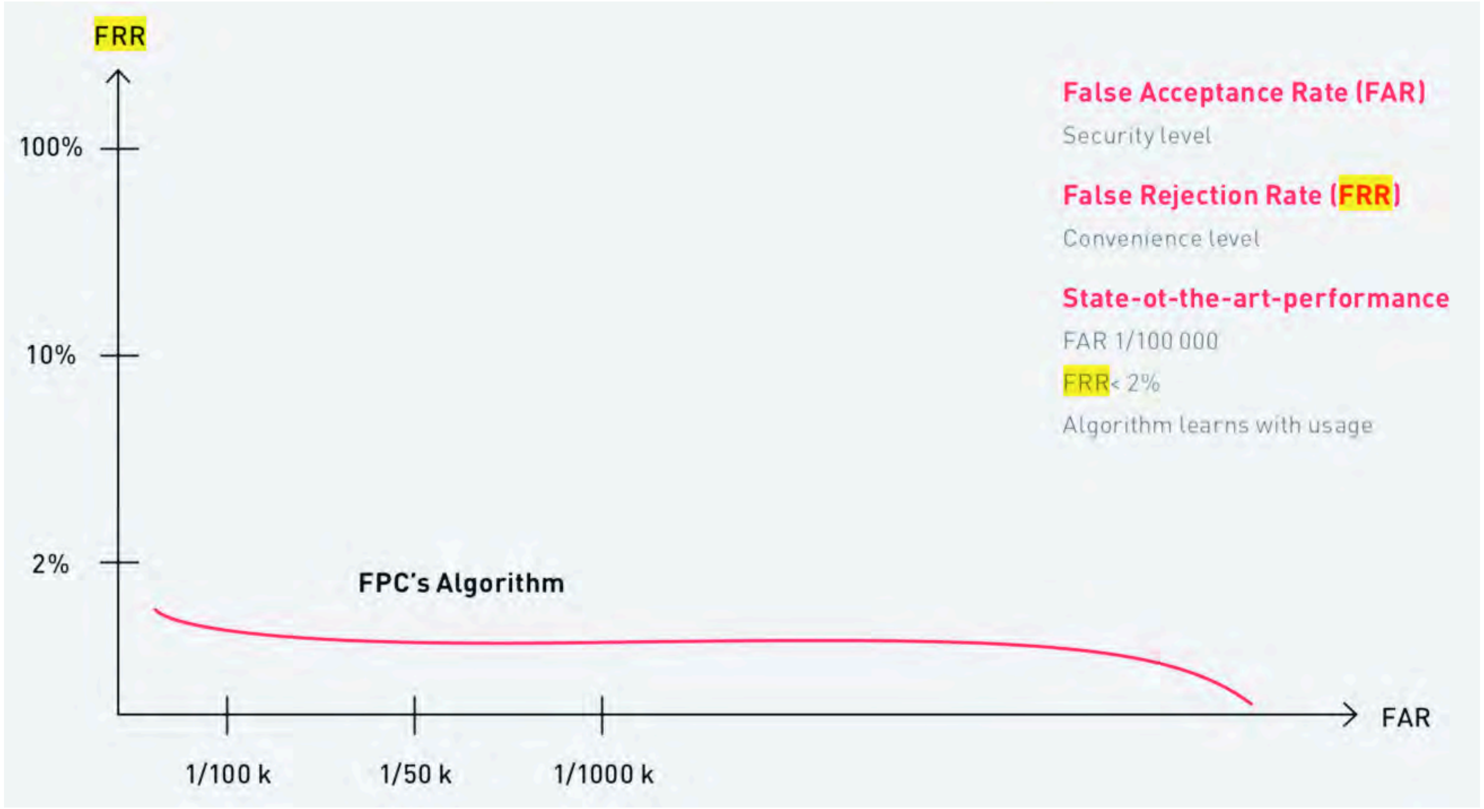
Biometrics: ROC Curve

Each biometric system has a characteristic ROC curve

(*receiver operator characteristic*, a legacy from radio electronics)



Sample ROC curve for an advanced capacitive fingerprint sensor



Source: https://www.fingerprints.com/uploads/2019/10/fpc_white_paper_digital.pdf

Galaxy S9 Intelligent Scan favors unlocking ease over security

An in-depth look at Samsung's new biometrics verification system -- and how it stacks up against the iPhone X’s Face ID — shows it's not quite safe enough for mobile payments.

Shara Tibken, Alfred Ng March 1, 2018 5:00 AM PST

Unlocking the Galaxy S9 might be faster -- but that doesn't mean it's more secure.

Samsung's newest smartphones, the Galaxy S9 and S9 Plus, include a new feature the company calls Intelligent Scan. The technology combines Samsung's secure iris scanner with its less-secure facial recognition unlock technology.

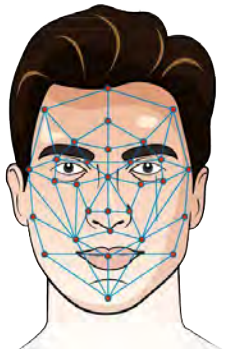
When unlocking your phone, it first will scan your face. If that fails to unlock the phone, the device then will check your irises. If both fail, Intelligent Scan will try to authenticate your identity using a combination of the two. And it all happens almost instantaneously.

https://www.cnet.com/news/samsung-galaxy-s9-intelligent-scan-unlock-favors-ease-over-security/

Biometric Modalities

Face

- Face geometry
- w/ 3-D imaging
- Thermographs
- Ear imaging



Eyes

- Iris - spokes
- Retina scans

Hands

- Fingerprints
- Vein scans
- Hand geometry
  - Finger length
  - Contours
  - Surface are

Signature, Voice

Behavioral vs. Physical system

John Hancock

Others

- DNA
- Odor
- Gait
- Driving habits
- …

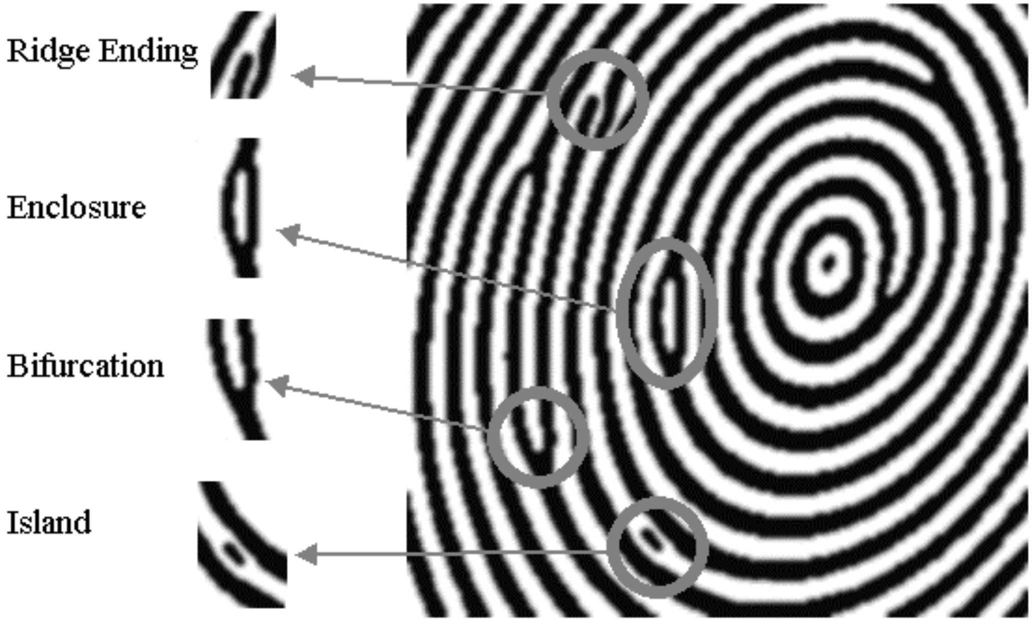
Biometrics: distinct features

Example: Fingerprints

Identify minutiae points and their relative positions

Minutiae (features)

- Arches
- Loops
- Whorls
- Ridge endings
- Bifurcations
- Islands
- Bridges



source: http://anil299.tripod.com/vol_002_no_001/papers/paper005.html

Biometrics: desirable characteristics

- **Robustness**
  - Repeatable, not subject to large changes over time
  - Fingerprints & iris patterns are more robust than voice
- **Distinctiveness**
  - Differences in the pattern among population
  - Fingerprints: typically 40-60 distinct features
  - Irises: typically >250 distinct features
  - Hand geometry: ~1 in 100 people may have a hand with measurements close to yours.

Biometrics: desirable characteristics

Biometric	Robustness	Distinctiveness	Ease of Use	User Acceptance
Fingerprint	Moderate	High	Medium	Medium
Face	Moderate	Low	High	High
Hand Geometry	Moderate	Low	Medium	Medium
Voice	Moderate	Low	High	High
Iris	High	Ultra high	Medium	Medium
Retina	High	Ultra high	Low	Low
Signature	Low	Moderate	Low	High

Irises vs. Fingerprints

- **Number of features measured:**
  - High-end fingerprint systems: ~40-60 features
  - Iris systems: ~240 features

- **False accept rates (FAR)**
  - Fingerprints: ~ 1:100,000 (varies by vendor; may be ~1:500)
    - FRR ≈ 0 – 66%, FAR ≈ 0.01%
  - Irises: ~ 1:1.2 million
    - FRR ≈ 1%, FAR ≈ 0.1%
  - Retina scan ~1:10,000,000

Irises vs. Fingerprints

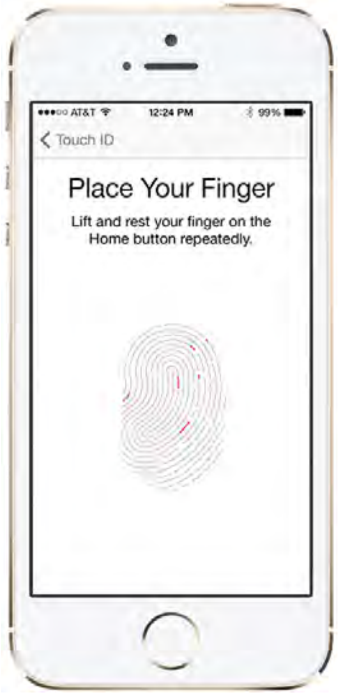
- **Ease of data capture**
  - More difficult to damage an iris … but lighting is an issue
  - Feature capture more difficult for fingerprints:
    - Smudges, gloves, dryness, …

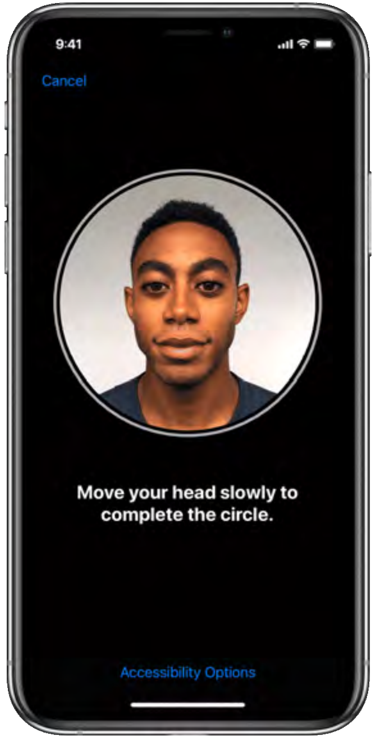
- **Ease of searching**
  - Fingerprints cannot be normalized
    1:many searches are difficult
  - Irises can be normalized to generate a unique IrisCode
    1:many searches much faster

Biometric: authentication process

0. Enrollment

- The user's entry in a database of biometric data needs to be initialized
- Initial sensing and feature extraction
- May be repeated to ensure good feature extraction





Biometric: authentication process

1. **Sensing**
   - User's characteristic must be presented to a sensor
   - Output is a function of:
     - Biometric measure
     - The way it is presented
     - Technical characteristics of sensor

2. **Feature Extraction**
   - Signal processing
   - Extract the desired biometric pattern
     - remove noise and signal losses
     - discard qualities that are not distinctive/repeatable
     - Determine if feature is of "good quality"

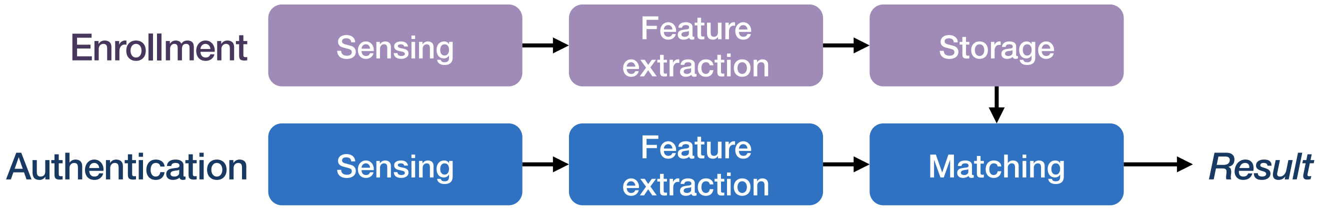
Biometric: authentication process

3. Pattern matching

- Sample compared to original signal in database
- Closely matched patterns have “small distances” between them
- Distances will hardly ever be 0 (perfect match)

4. Decision

- Decide if the match is close enough
- Trade-off:
  ↓ false non-matches leads to ↑false matches



Identification vs. Verification

- **Identification:** ***Who is this?***
  - *1:many* search

- **Verification:** ***Is this Bob?***
  - Present a name, PIN, token
  - *1:1* (or 1:small #) search

Biometrics: Essential characteristics

- Trusted sensor
- Liveness testing
- Tamper resistance
- Secure communication
- Acceptable thresholds

Biometrics: other characteristics

- **Cooperative systems (multi-factor)**
  - User provides identity, such as name and/or PIN
- **vs. Non-cooperative**
  - Users cannot be relied on to identify themselves
  - Need to search large portion of database
- **Overt vs. covert identification**
- **Habituated vs. non-habituated**
  - Do users regularly use (train) the system

Problems with biometric systems

- **Requires a sensor**
  - Camera works OK for iris scans & facial detection
    (but a good Iris scan will also take IR light into account)
- **Tampering with device or device link**
  - Replace sensed data– or just feed new data
- **Tampering with stored data**
- **Biometric data cannot be compartmentalized**
  - You cannot have different data for your Amazon & bank accounts
- **Biometric data can be stolen**
  - Photos, lifting fingerprints
  - Once biometric data is compromised, it remains compromised
    - You cannot change your iris or finger

naked security by SOPHOS

A photo will unlock many Android phones using facial recognition

By John E Dunn

08 JAN 2019 5

Security threats, Vulnerability

How easy is it to bypass the average smartphone's facial recognition security?

According to the Dutch consumer protection organisation Consumentenbond, in the case of several dozen Android models, it's a lot easier than most owners probably realise.

Its researchers tested 110 devices, finding that 42 could be beaten by holding up nothing more elaborate than a photograph of a device's owner.

Consumentenbond offers little detail of its testing methodology but it seems these weren't high-resolution photographs – almost any would do, including those grabbed from social media accounts or selfies taken on another smartphone.

While users might conclude from this test that it's not worth turning on facial recognition, the good news is that 68 devices, including Apple's recent XR and XS models, resisted this simple attack, as did many other high-end Android models from Samsung, Huawei, OnePlus, and Honor.

https://nakedsecurity.sophos.com/2019/01/08/facial-recognition-on-42-android-phones-beaten-by-photo-test/

BBC NEWS

Google Pixel 4 Face Unlock works if eyes are shut

Chris Fox • Technology reporter • 17 October 2019

Google has confirmed the Pixel 4 smartphone's Face Unlock system can allow access to a person's device even if they have their eyes closed.
One security expert said it was a significant problem that could allow unauthorised access to the device.

By comparison, Apple's Face ID system checks the user is "alert" and looking at the phone before unlocking.

Google said in a statement: "Pixel 4 Face Unlock meets the security requirements as a strong biometric."

https://www.bbc.com/news/technology-50085630

Samsung Galaxy S8 iris scanner tricked by photo, contact lens

c|net

Turns out the sophisticated tech can't tell the difference between your eye and a picture with a contact lens over the iris, a hacking club says.

Alfred NG. May 24, 2017 8:34 AM PDT

You won't believe your eyes. But maybe the Samsung Galaxy S8 will.

In the month since Samsung released its flagship device, hackers in Germany have figured how to break the phone's iris recognition lock. Samsung has touted the biometric technology as "one of the safest ways to keep your phone locked," claiming that a person's iris patterns are "virtually impossible to replicate."

But that's exactly what the hackers from the Chaos Computer Club say they did. The hackers used a photo shot in night mode and from a medium distance, about the same range that would pop up in a Facebook profile picture or a selfie. They then printed out a closeup of the person's eye and put a contact lens over the iris on the paper.

The lens is there to replicate the eye's curvature, the Chaos Computer Club said in a blog post this week. Someone then held up the piece of paper to the Samsung Galaxy S8's iris scanner, and it unlocked as if a real person had looked at it.

https://www.cnet.com/news/samsung-galaxy-s8-iris-scanner-tricked-photo-contact-lens/

THE WALL STREET JOURNAL.

Fraudsters Used AI to Mimic CEO's Voice in Unusual Cybercrime Case

Scams using artificial intelligence are a new challenge for companies

By Catherine Stupp • August 30, 2019

Criminals used artificial intelligence-based software to impersonate a chief executive's voice and demand a fraudulent transfer of €220,000 ($243,000) in March in what cybercrime experts described as an unusual case of artificial intelligence being used in hacking.

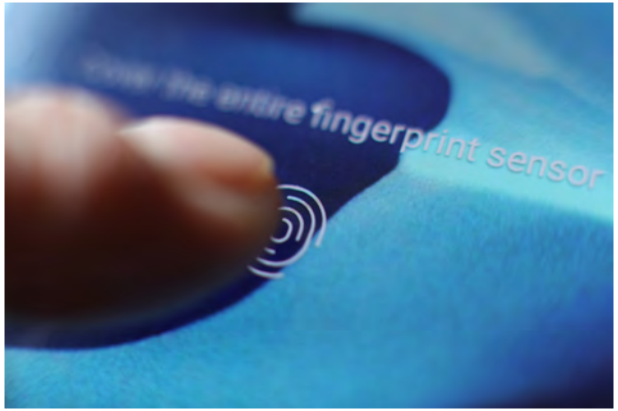
The CEO of a U.K.-based energy firm thought he was speaking on the phone with his boss, the chief executive of the firm's German parent company, who asked him to send the funds to a Hungarian supplier. The caller said the request was urgent, directing the executive to pay within an hour, according to the company's insurance firm, Euler Hermes Group SA.

https://www.wsj.com/articles/fraudsters-use-ai-to-mimic-ceos-voice-in-unusual-cybercrime-case-11567157402 engadget

Massive biometric security flaw exposed more than one million fingerprints

The system is used by banks, police and defence companies.

August 14, 2019 – Rachel England, @rachel_england



A biometrics system used by banks, UK police and defence companies has suffered a major data breach, revealing the fingerprints of more than one million people as well as unencrypted passwords, facial recognition information and other personal data.

Biostar 2, the biometrics lock system managed by security company Suprema, uses fingerprints and facial recognition technology to give authorised individuals access to buildings. Last month the platform was integrated into another access system -- AEOS -- which is used by 5,700 organizations across 83 countries, including the UK Metropolitan Police.

https://www.engadget.com/2019/08/14/biometric-security-flaw-fingerprints

THE VERGE

Samsung’s Galaxy S10 fingerprint sensor fooled by 3D printed fingerprint

It took 13 minutes to print up the fake

By Andrew Liptak • April 7 2019

… user darkshark outlined his project: he took a picture of his fingerprint on a wineglass, processed it in Photoshop, and made a model using 3ds Max that allowed him to extrude the lines in the picture into a 3D version. After a 13-minute print (and three attempts with some tweaks), he was able to print out a version of his fingerprint that fooled the phone’s sensor.

https://www.theverge.com/2019/4/7/18299366/samsung-galaxy-s10-fingerprint-sensor-fooled-3d-printed-fingerprint

Video: https://imgur.com/gallery/8aGqsSu

THE VERGE

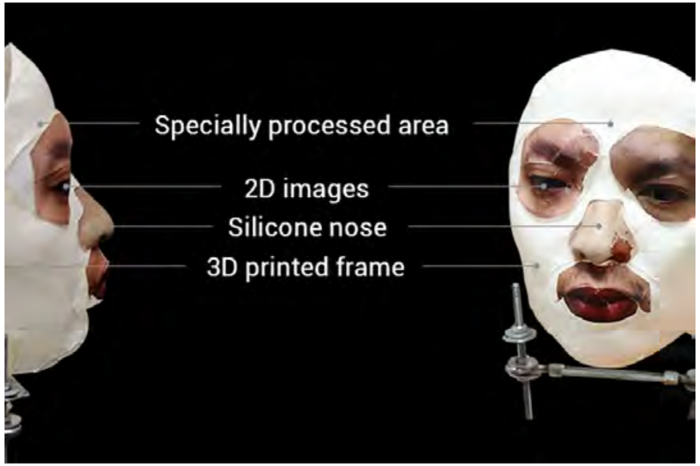
This $150 mask beat Face ID on the iPhone X

It’s just a proof of concept at the moment

By Thuy Ong • Nov 13 2017

Vietnamese cybersecurity firm Bkav claims it's been able to bypass the iPhone X's Face ID feature using a mask. The mask is made to trick Apple's depth mapping and the result is a kind of creepy hybrid monster head with realistic cutouts for the eyes, nose and mouth.

Bkav says the mask is crafted through a combination of 3D printing, makeup, and 2D images.



https://www.theverge.com/2017/11/13/16642690/bkav-iphone-x-faceid-mask

The End