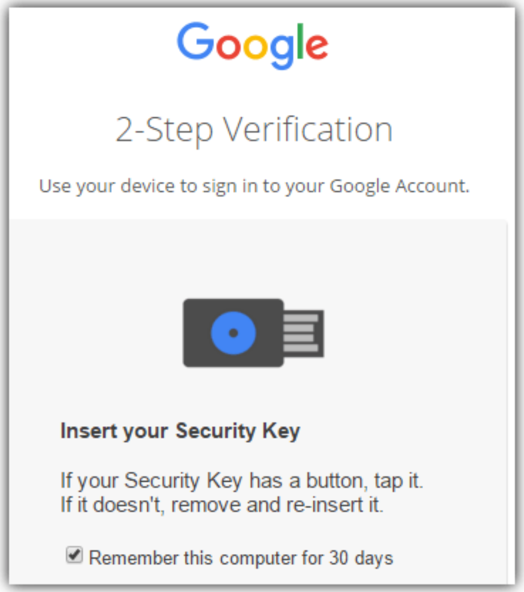
2550 Intro to cybersecurity

L5

abhi shelat

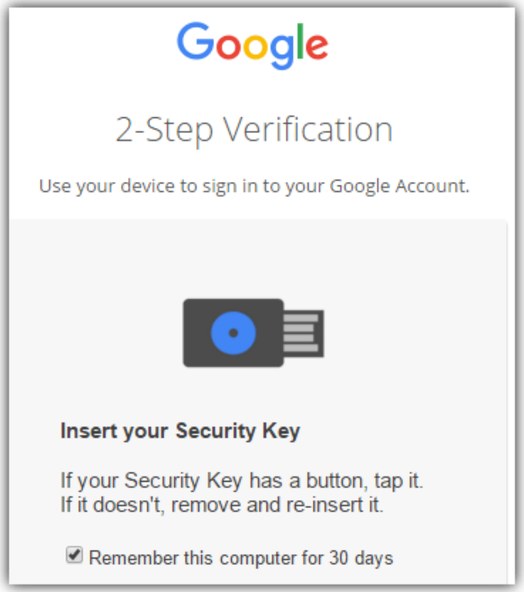
Universal 2nd Factor (U2F)

- Supported by Chrome, Opera, and Firefox (must be manually enabled)
- Works with Google, Dropbox, Facebook, Github, Gitlab, etc.



Universal 2nd Factor (U2F)

- Supported by Chrome, Opera, and Firefox (must be manually enabled)
- Works with Google, Dropbox, Facebook, Github, Gitlab, etc.
- Pro tip: always buy 2 security keys
  - Associate both with your accounts
  - Keep one locked in a safe, in case you lose your primary key ;)



How does U2F work?

(THIS protocol has a few flaws)

Website (Relying Party)

Init

Make a signing key (sk,pk)

unguessable

{register} ←

{register} ←

{ pk, sign_sk("username") } →

User, pk

Login

Sign challenge using sk

{login, ch} ←

{login, challenge ch} ←

$s \leftarrow \text{Sign}_{sk}(ch)$

{ s } →

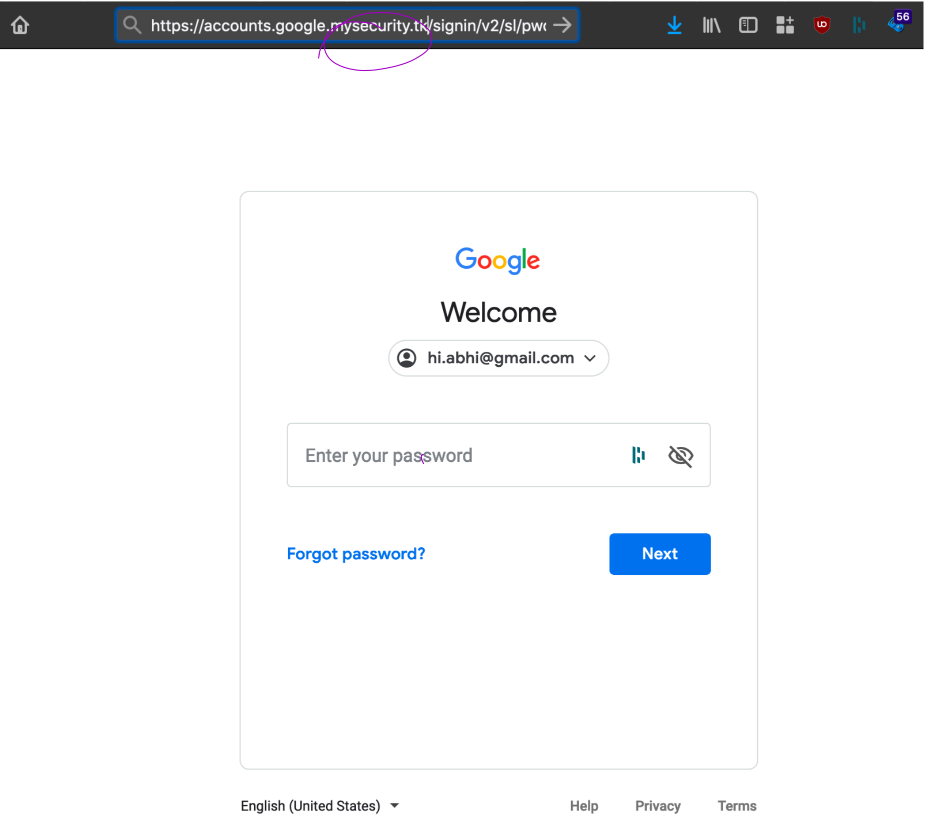
$\text{Verify}_{pk}(ch)$

Vulnerable to simple attack

It solves a problem with

guessable pwds.

But it still has a big flaw.


https://accounts.google.mysecurity.tk/signin/v2/sl/pwd
56
Google
Welcome
hi.abhi@gmail.com
Enter your password
Forgot password?
Next
English (United States)
Help
Privacy
Terms

Simple Phishing

Lure: A spammed email with a call to action from a seemingly legitimate source encouraging the user to visit a hook website.

Hook: A website designed to mimic legitimate site and collect confidential information.

Google

Someone has your password

Hi William

Someone just used your password to try to sign in to your Google Account

Details:

Tuesday, 22 March, 14:9:25 UTC
IP Address: 134.249.139.239
Location: Ukraine

Google stopped this sign-in attempt. You should change your password immediately.

CHANGE PASSWORD

Best,
The Gmail Team

You received this mandatory email service announcement to update you about important changes to your Google product or account.

bitly

TOUR ENTERPRISE RESOURCES ABOUT

MY ACCOUNT

MAR 19

http://myaccount.google.com-securitysettingpage.tk/security/signinoptions/password?e=am9obi5wb2Rlc3RhQGdtYWlsLmNvbQ%3D%3D&fn=Sm9obiBQb2Rlc3Rh&n=Sm9obg%3D%3D&img=Ly9saDQuZ29vZ2xldXNlcmNvbnRlbnQuY29tLy1RZVlPbHJkVGp2WS9BQUFB...

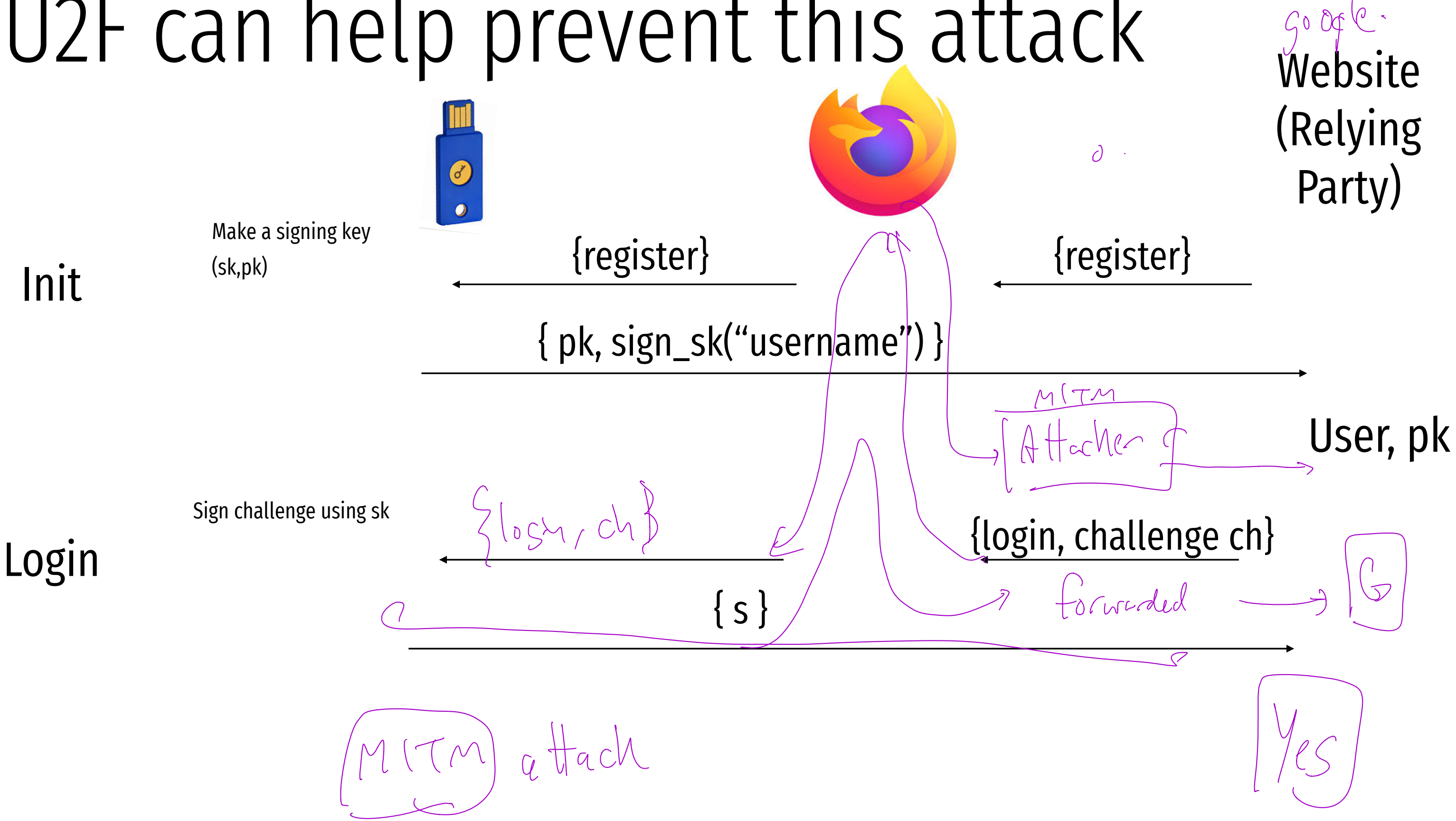
http://myaccount.google.com-securitysettingpage.tk/security/signinoptions/password?e=am9obi5wb2Rlc3RhQGdtYWlsLmNvbQ%3D%3D&fn=Sm9obiBQb2Rlc3Rh&n=Sm9obg%3D%3D&img=Ly9saDQuZ29vZ2xldXNlcmNvbnRlbnQuY29tLy1RZVlPbHJkVGp2WS9BQUFBQUFBQUFBSS9BQUFBQUFBQUFCTS9CQldVOVQ0bUZUWS9waG90by5qcGc%3D&id=1sutlodlwe

bitly.com/[redacted] COPY

2

CLICKS

U2F can help prevent this attack



U2F can help prevent this attack

Website (Relying Party)

Init

Make a signing key (sk,pk)

{register} ← {register}

{ pk, sign_sk("username") } →

User, pk

Login

Sign challenge using sk

{login, ch, origin, tls_id} ← {login, challenge ch}

$s \leftarrow \text{Sign}_{sk}(ch, \text{url}, \text{tls}_{id})$

{ s } →

$\text{Verify}_{pk}(ch, s, \text{url}, \text{tls}_{id})$

How U2F foils phishing

User,sk

My browser

1. In the beginning, I register with G and setup 2FA.

User,pk

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.

User,sk

My browser

Fake Website

Com-settingssecurity.tk

login

pwd

User,pk

{login, challenge ch}

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.

User,sk

My browser

Fake Website

Com-settingssecurity.tk

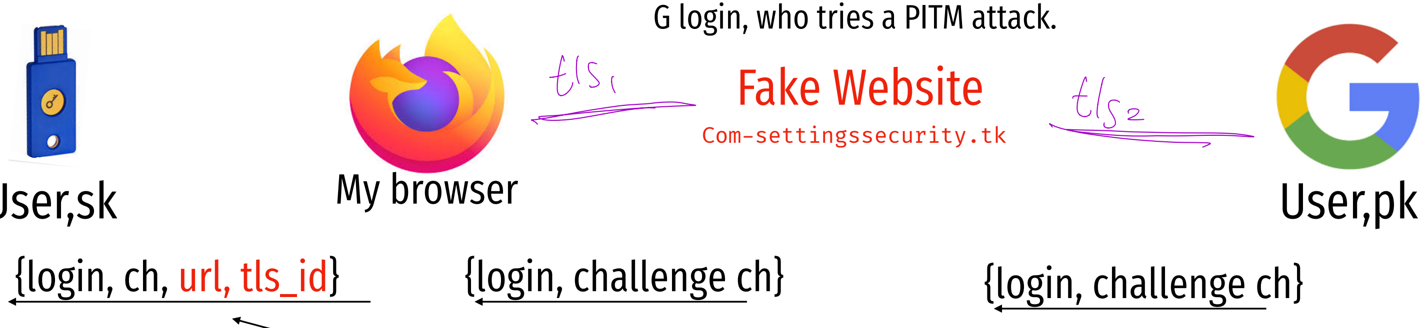
User,pk

{login, challenge ch}

{login, challenge ch}

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.



My browser knows the origin is “com-settingssecurity.tk” instead of google.com, and passes this string as url.

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.

Fake Website

Com-settingssecurity.tk

User,sk

My browser

User,pk

{login, ch, url, tls_id}

{login, challenge ch}

{login, challenge ch}

My browser knows the origin is “com-settingssecurity.tk” instead of google.com, and passes this string as url.

$s \leftarrow \mathsf{Sign}_{sk}(ch, \mathsf{url}, \mathsf{tls}_{id})$

Sign challenge using sk

The 2FA key signs this with url=com-settings...

what does G do ??

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.

User,sk

My browser

Fake Website

Com-settingssecurity.tk

User,pk

{login, ch, url, tls_id}

{login, challenge ch}

{login, challenge ch}

My browser knows the origin is "com-settingssecurity.tk" instead of google.com, and passes this string as url.

$s \leftarrow \text{Sign}_{sk}(ch, \text{url}, \text{tls}_{id})$

Sign challenge using sk

{ s }

$\text{Verify}_{pk}(ch, s, \text{url}, \text{tls}_{id})$

The 2FA key signs this with url=com-settings...

Google reject the authentication and detects the attack!

The Tracking problem

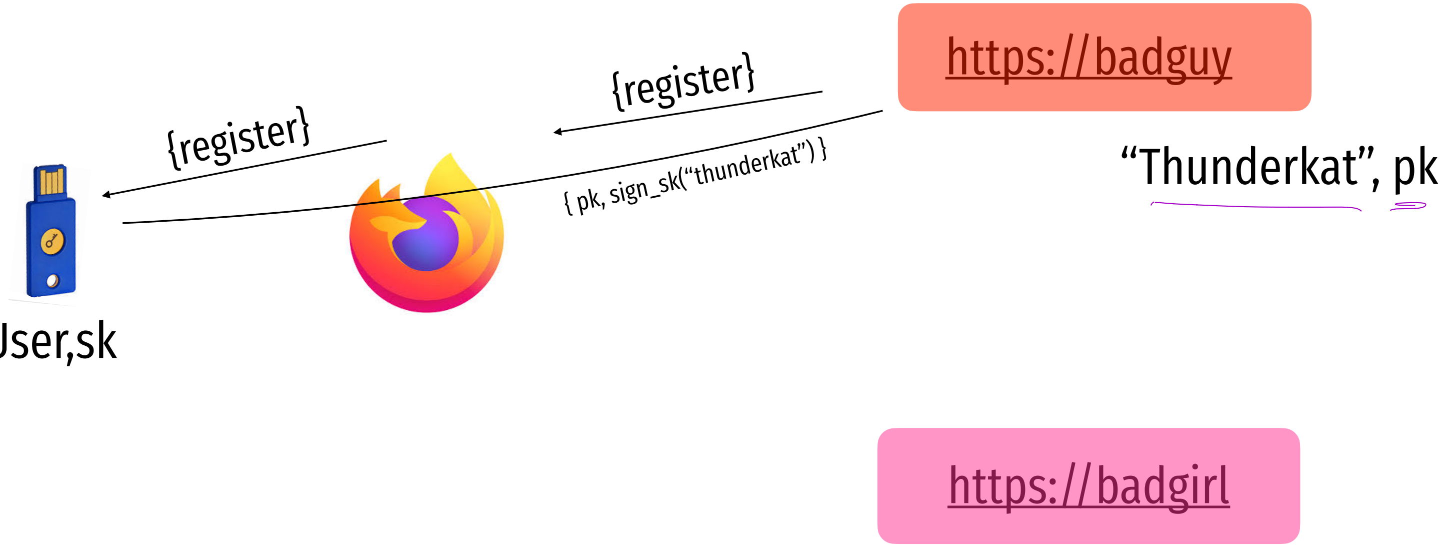
https://badguy

You

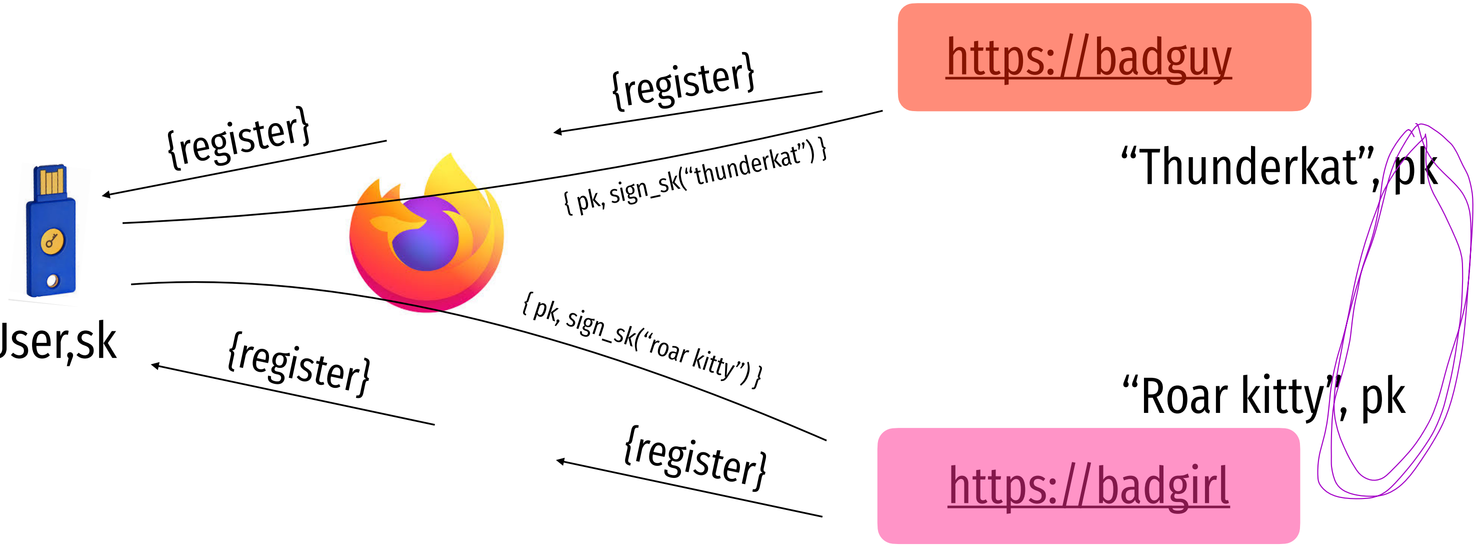
User,sk

https://badgirl

The Tracking problem



The Tracking problem



U2F can help prevent tracking

Website (Relying Party)

Init

Make a signing key (sk,pk)

And link it with appid, and create A token "h"

{appid, register}

{appid, register}

{ h, pk, sign_sk("username") }

User, h, pk

U2F can help prevent tracking

Website (Relying Party)

Init

Make a signing key with aphid (sk,pk)

And link it with appid, and create A token “h”

{appid, register} ←

{appid, register} ←

{ h, pk, sign_sk(“username”) } →

User, h, pk

Login

Lookup sk using h
Sign challenge using sk

$s \leftarrow \text{Sign}_{sk}(ch, \text{url}, \text{tls}_{id})$

{login, h, ch, origin, tls_id} ←

{login, h, challenge ch} ←

{ s,h } →

$\text{Verify}_{pk}(ch, s, \text{url}, \text{tls}_{id})$

Check h

```
Sending request with appId: https://u2f.bin.coffee
{
  "version": "U2F_V2",
  "challenge": "uQnl3M4Rj3FZgs6WjyLaZAfwRh4"
}

Got response:
{
  "clientData": "eyJjaGFsbGVuZ2UiOiJ1UW5sM000UmozRlpnczZXanlMYVpBZndSaDQiLCJvcmlnaW4iOiJodHRwczovL3UyZi5iaW4uY29mZmVlIiwidHlwIjoibmF2
  "errorCode": 0,
  "registrationData": “BQRSuRLPv0p5udQ55vVhucf3N50q6…”,
  "version": "U2F_V2"
}

Key Handle: 0r0Z0p0F0E0-0d0W0c0Q0b0X0i020C0w0-0E0v0h0t0T0T0P0_0-090_0a050P0e030u0b0z0l0K0Q0r0O0f0u030_0P020B0J0M0x0D050J0_0d0P0Q0e0j0
Certificate: 3082021c3082…
Attestation Cert
Subject: Yubico U2F EE Serial 14803321578
Issuer: Yubico U2F Root CA Serial 457200631
Validity (in millis): 1136332800000
Attestation Signature
R: 00b11e3efe5ae5ac7ca0e0d4fe2c5b5cf18a2531c0f4f70b11c30b72b5f946a9a3
S: 0f37ab2d4f93ebcdaed0a51b4b17fb93403db9873f0e9cce36f17b1502734bb2
[PASS] Signature buffer has no unnecessary bytes.: 71 == 71
[PASS] navigator.id.finishEnrollment == navigator.id.finishEnrollment
[PASS] uQnl3M4Rj3FZgs6WjyLaZAfwRh4 == uQnl3M4Rj3FZgs6WjyLaZAfwRh4
[PASS] https://u2f.bin.coffee == https://u2f.bin.coffee
[PASS] Verified certificate attestation signature
[PASS] Imported credential public key
Failures: 0 TODOs: 0
```

Future without passwords?

Authentication Protocols

Unix, PAM, and crypt

Network Information Service (NIS, aka Yellow Pages)

Needham-Schroeder and Kerberos

Status Check

- At this point, we have discussed:
  - How to securely store passwords
  - Techniques used by attackers to crack passwords
  - Biometrics and 2^{nd} factors

Status Check

- At this point, we have discussed:
  - How to securely store passwords
  - Techniques used by attackers to crack passwords
  - Biometrics and 2nd factors
- Next topic: building authentication systems
  - Given a user and password, how does the system authenticate the user?
  - How can we perform efficient, secure authentication in a distributed system?

Building authentication systems

Example PAM Configuration

```
abhi@l2:~$ cat /etc/pam.d/common-password
#
# /etc/pam.d/common-password - password-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define the services to be
# used to change user passwords.  The default is pam_unix.

# Explanation of pam_unix options:
#
# The "sha512" option enables salted SHA512 passwords.  Without this option
# the default is Unix crypt.  Prior releases used the
#
# The "obscure" option replaces the old `OBSCURE_CHECK
# login.defs.
#
# See the pam_unix manpage for other options.

# As of pam 1.0.1-6, this file is managed by pam-auth-
# To take advantage of this, it is recommended that yo
# local modules either before or after the default block, and us
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
password [success=1 default=ignore]pam_unix.so obscure sha512
# here's the fallback if no module succeeds
password requisite     pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
password required      pam_permit.so
# and here are more per-package modules (the "Additional" block)
# end of pam-auth-update config
```

- Use SHA512 as the hash function
- Use /etc/shadow for storage

Unix Passwords

- Traditional method: *crypt* md5crypt
  - 25 iterations of DES on a zeroed vector
  - First eight bytes of password used as key (additional bytes are ignored)
  - 12-bit salt
- Modern version of *crypt* are more extensible
  - Support for additional hash functions like MD5, SHA256, and SHA512
  - Key lengthening: defaults to 5000 iterations, up to $10^8 - 1$
  - Full password used
  - Up to 16 bytes of salt

Password Files

- Password hashes used to be in */etc/passwd*
  - World readable, contained usernames, password hashes, config information
  - Many programs read config info from the file…
  - But very few (only one?) need the password hashes

Password Files

- Password hashes used to be in */etc/passwd*
  - World readable, contained usernames, password hashes, config information
  - Many programs read config info from the file…
  - But very few (only one?) need the password hashes
- Turns out, world-readable hashes are **Bad Idea**
- Hashes now located in */etc/shadow*
  - Also includes account metadata like expiration
  - Only visible to root

Password Storage on Linux

/etc/passwd

username:x:UID:GID:full_name:home_directory:shell

uid groupid homedir shell

cbw:x:1001:1000:Christo Wilson:/home/cbw/:/bin/bash
amislove:1002:2000:Alan Mislove:/home/amislove/:/bin/sh

/etc/shadow

username:password:last:may:must:warn:expire:disable:reserved

cbw:$1$0nSd5ewF$0df/3G7iSV49nsbAa/5gSg:9479:0:10000::::
amislove:1l3RxU5F1$:8172:0:10000::::

Password Storage on Linux

/etc/passwd

username:x:UID:GID:full_name:home_directory:shell

cbw:x:1001:1000:Christo Wilson:/home/cbw/:/bin/bash
n Mislove:/home/amislove/:/bin/sh

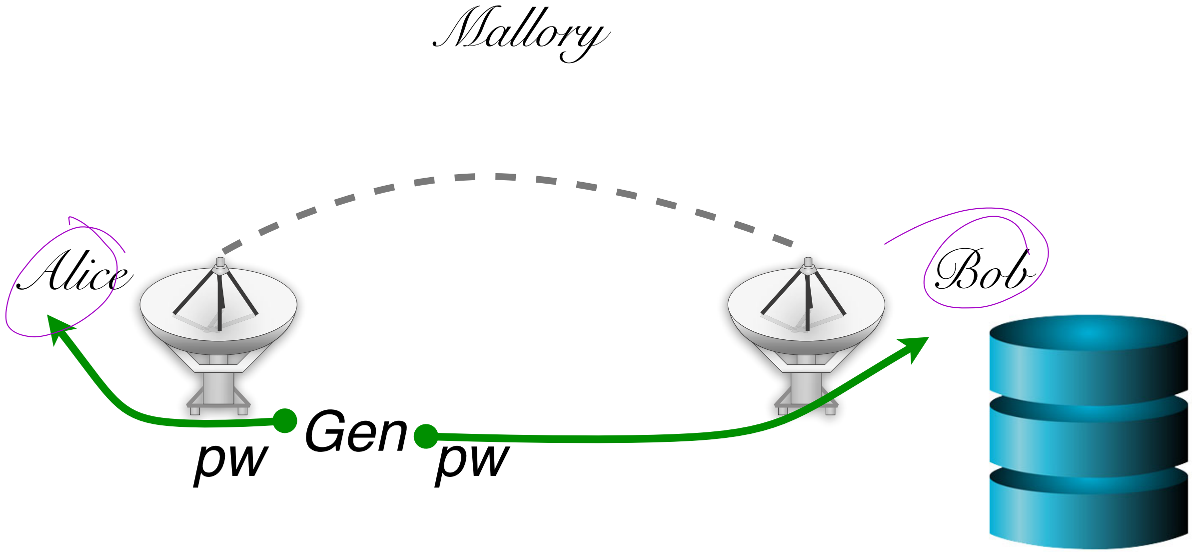
$<algo>$<salt>$<hash>
Algo: 1 = MD5, 5 = SHA256, 6 = SHA512

/etc/shadow

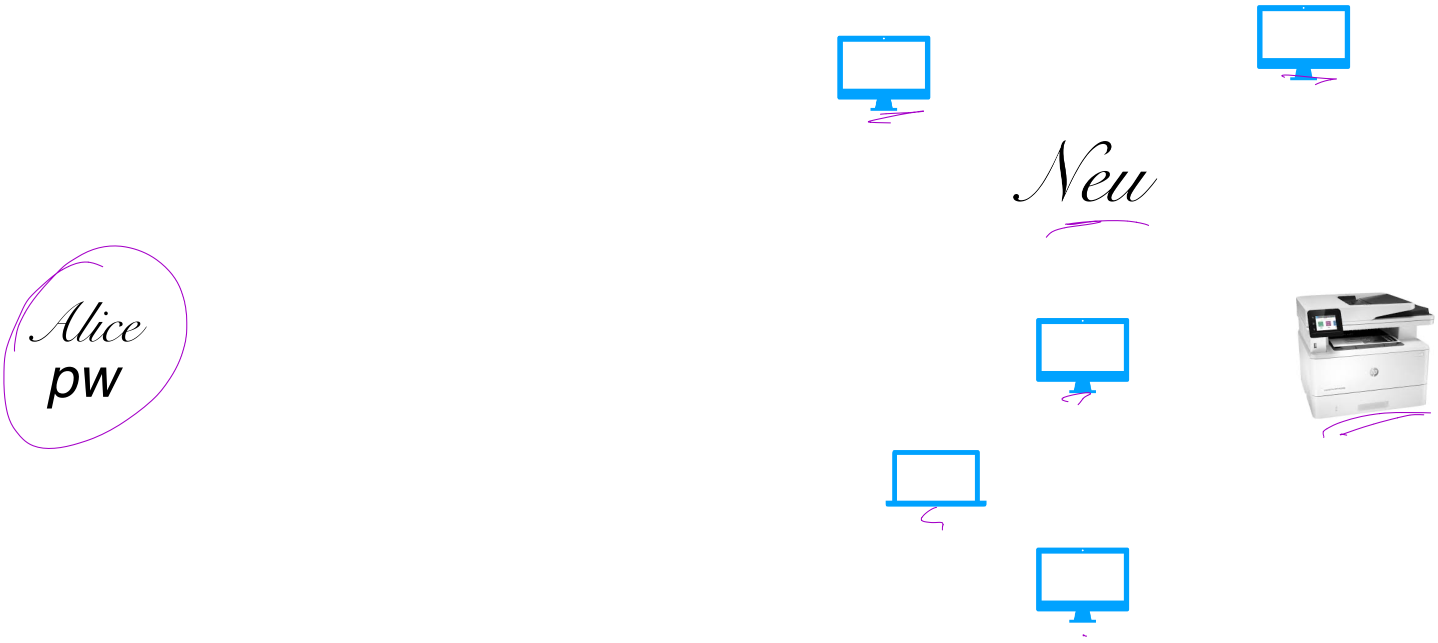
ername:password:last:may:must:warn:expire:disable:reserved

cbw:$1$0nSd5ewF$0df/3G7iSV49nsbAa/5gSg:9479:0:10000::::
amislove:1l3RxU5F1$:8172:0:10000::::

Password Security game



More realistic picture of the world



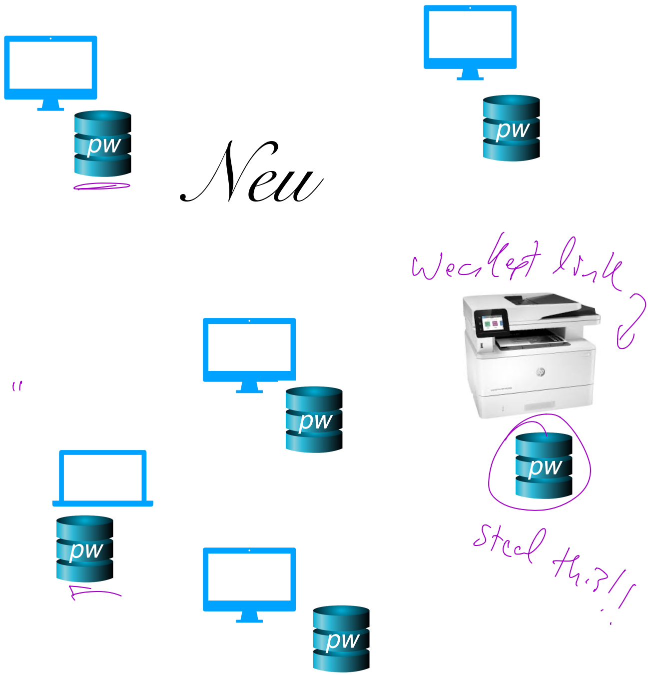
More realistic picture of the world

What are the problems with this solution?

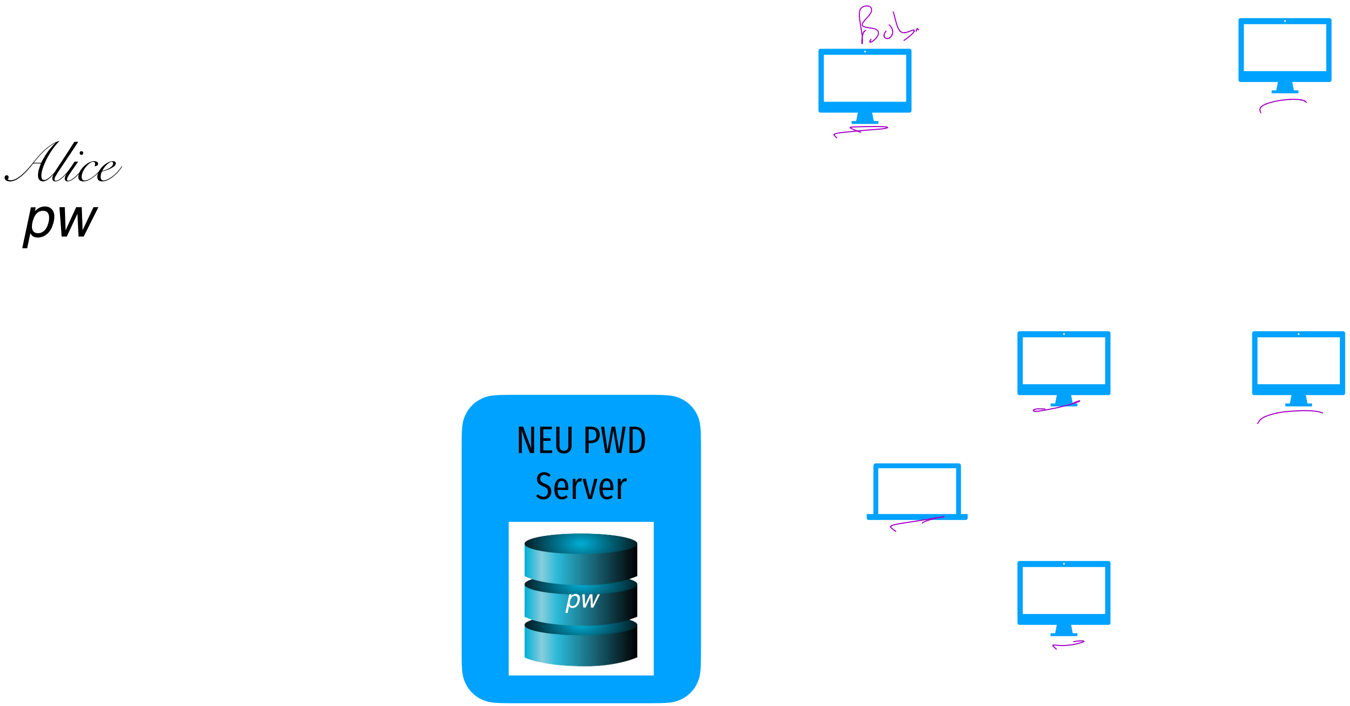
- update difficulty, sync.
- weakness among devices
  - increases "attack surface" (huge)

Alice

pw



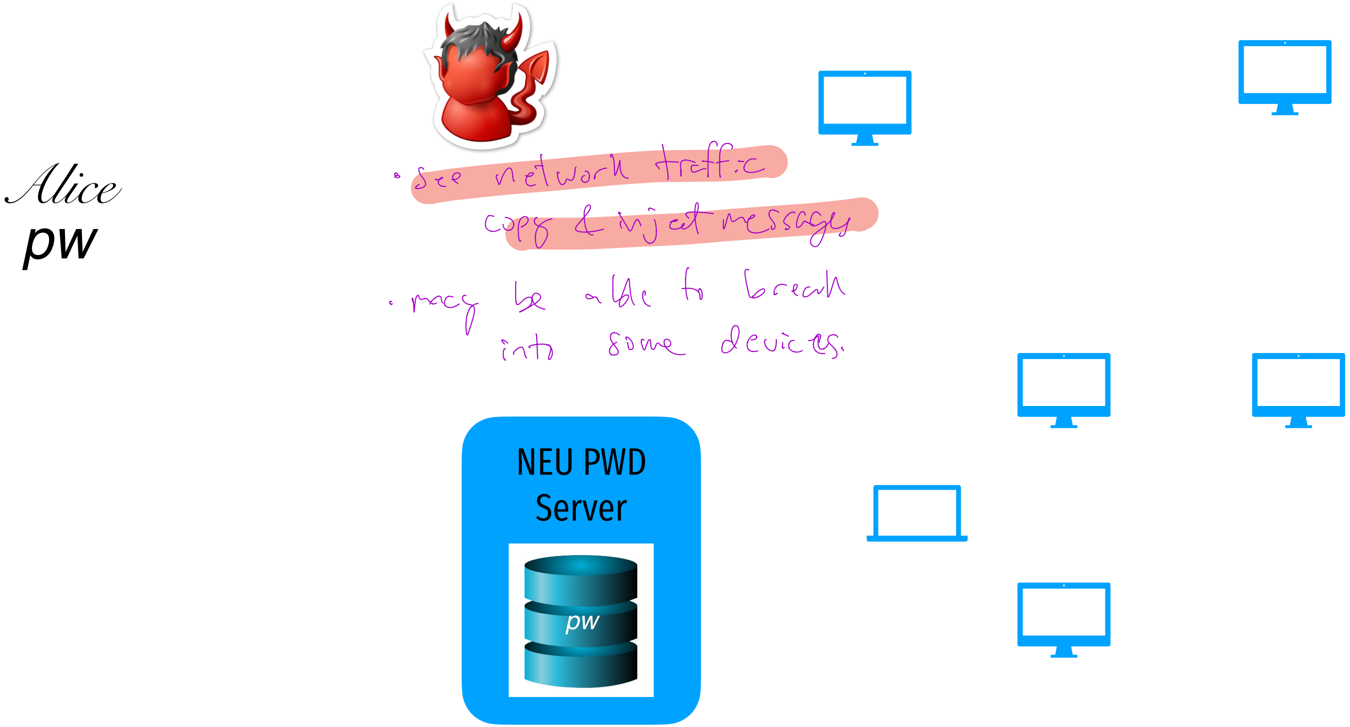
The problem of distributed authentication



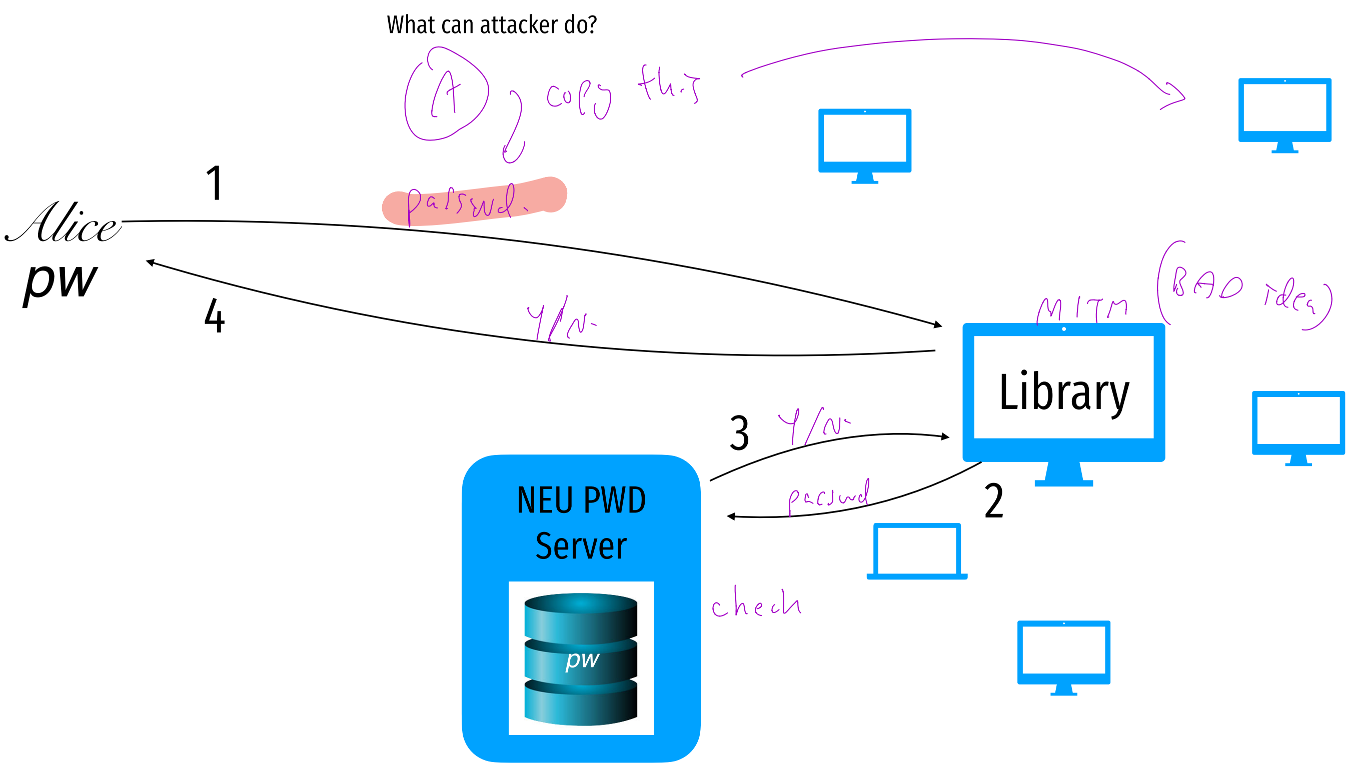
Distributed authentication: Attacker model

What can attacker do?

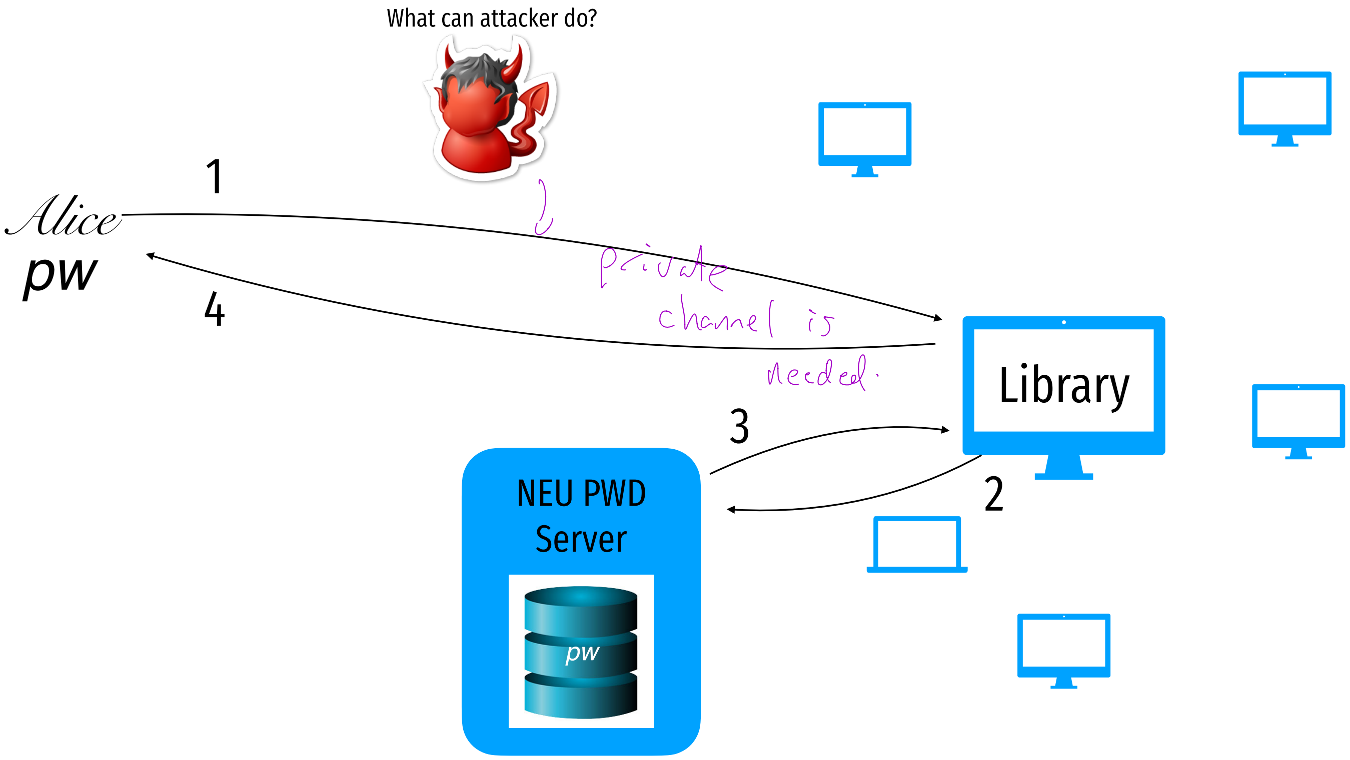
- see network traffic
  copy & inject messages
- may be able to break into some devices.



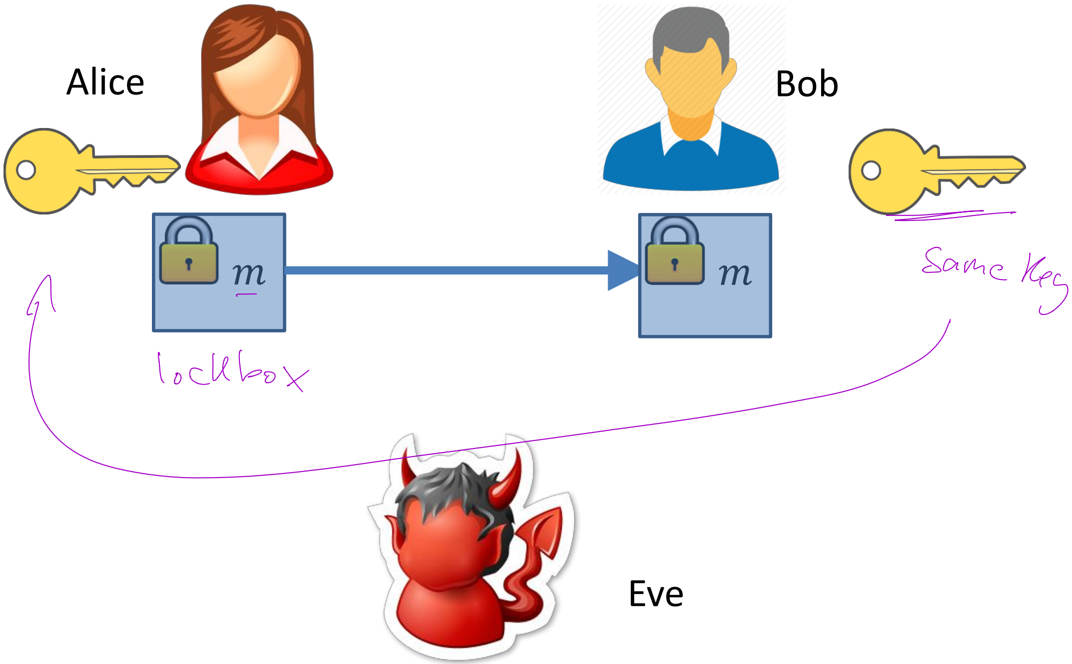
Distributed authentication: Bad Solution



Distributed authentication: Bad Solution



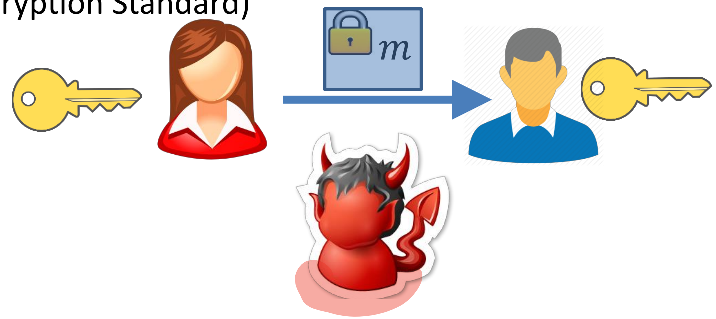
Basic tool: symmetric encryption



Basic tool: symmetric encryption

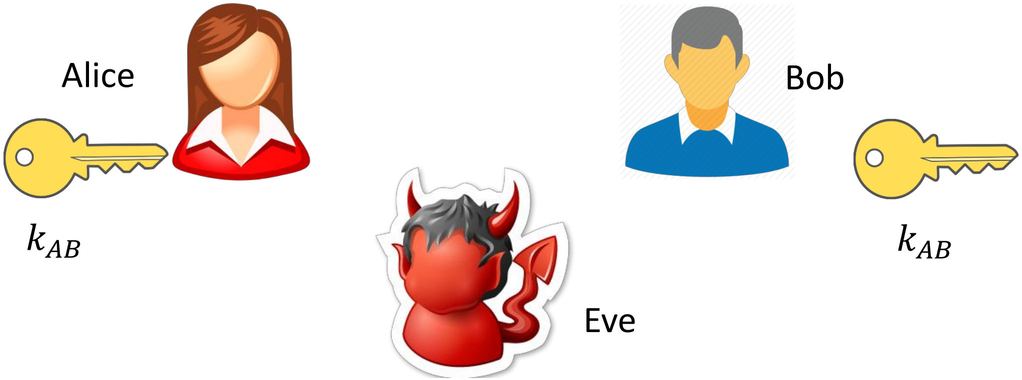
- Gen: generates secret key k
- Enc: given k and m output a ciphertext c
  Denote $Enc_k(m)$, $E_k(m)$, $\{m\}_k$
- Dec: given k and c output a message m
- Security (informal):
  Whatever Eve can learn on m given c can be learned without c
- Examples:
  - DES (Data Encryption Standard)
  - AES (Advanced Encryption Standard)

Eve can "copy" this lockbox

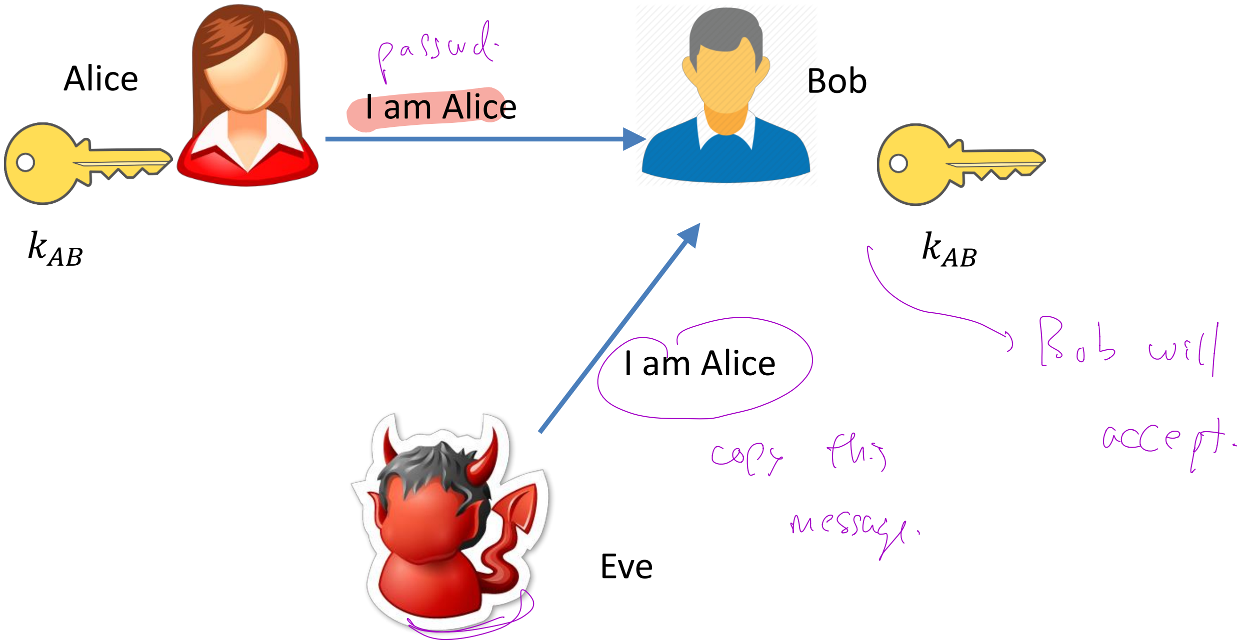


Authentication from Encryption

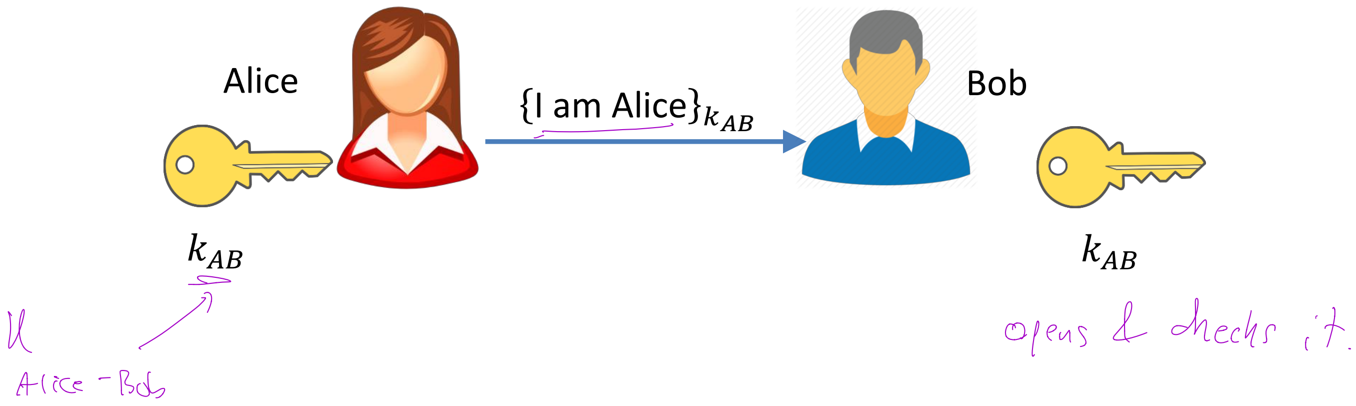
- Alice and Bob share a key
- They communicate over an insecure channel
- Alice wants to prove her identity to Bob
- Eve's goal: impersonate Alice



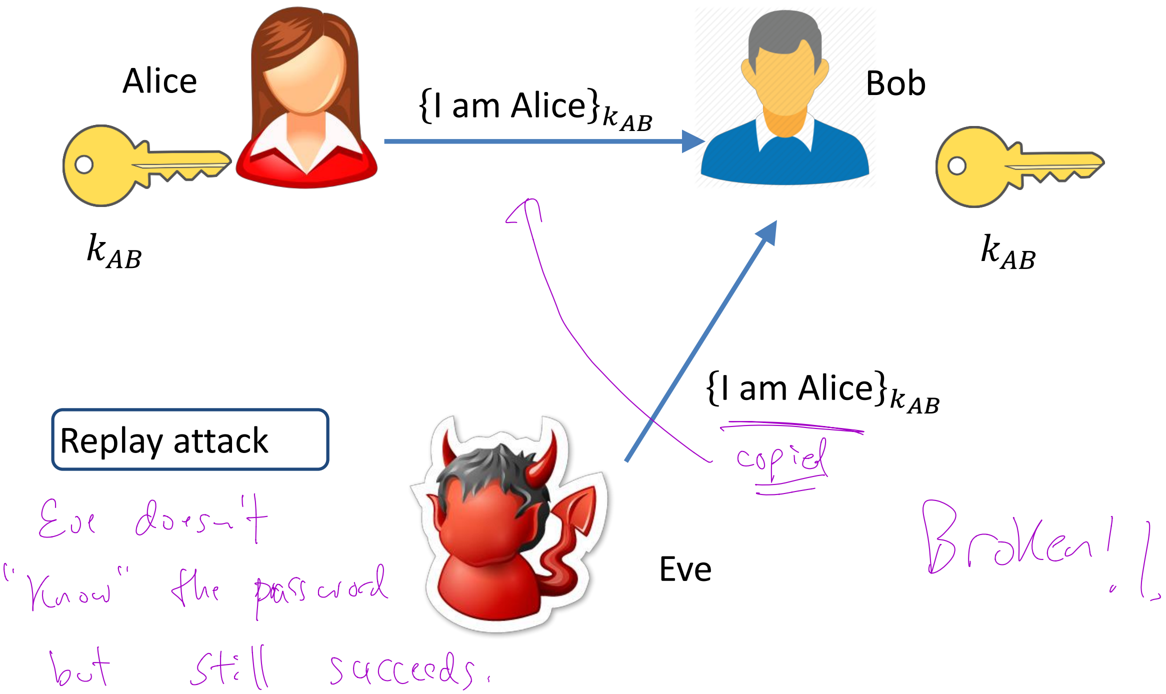
Attempt #1


Alice
passwd.
I am Alice
Bob
k_{AB}
k_{AB}
I am Alice
Bob will accept.
copy this message.
Eve

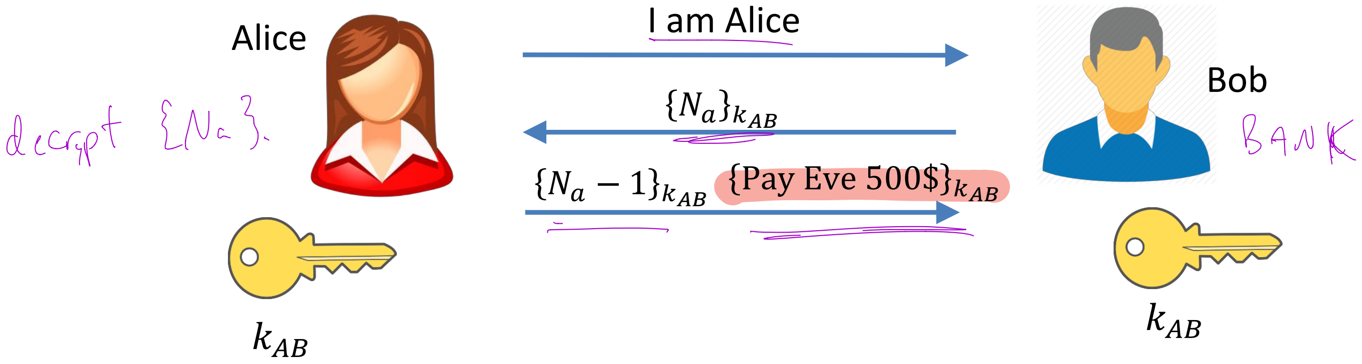
Attempt #2: use the key



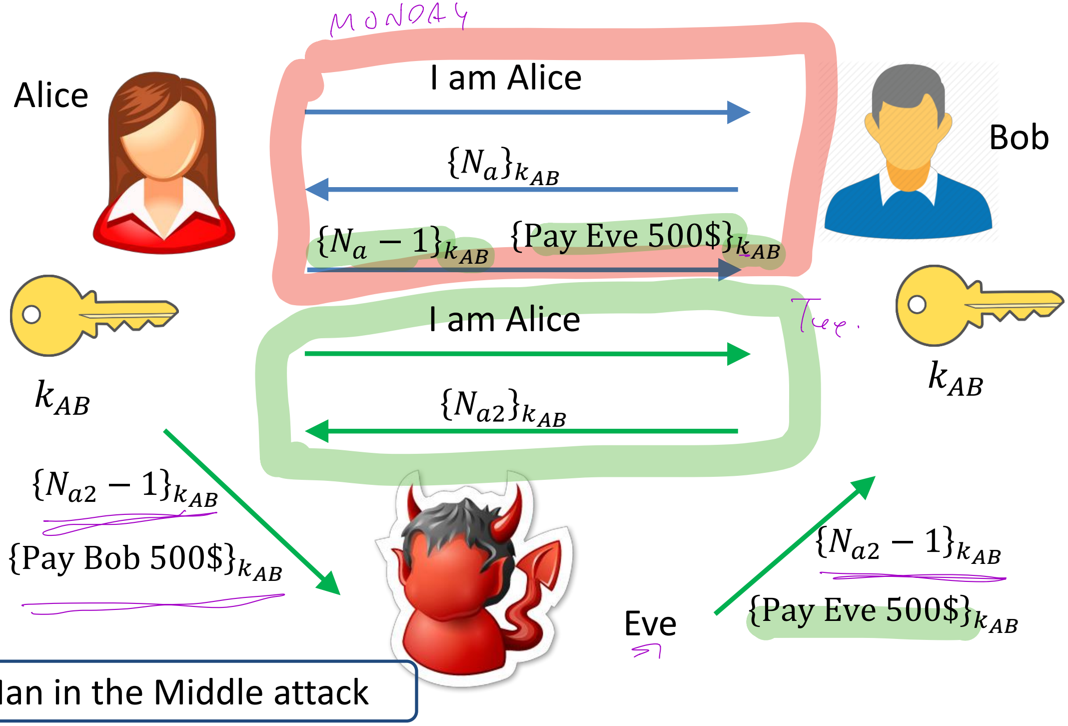
Attempt #2: use the key



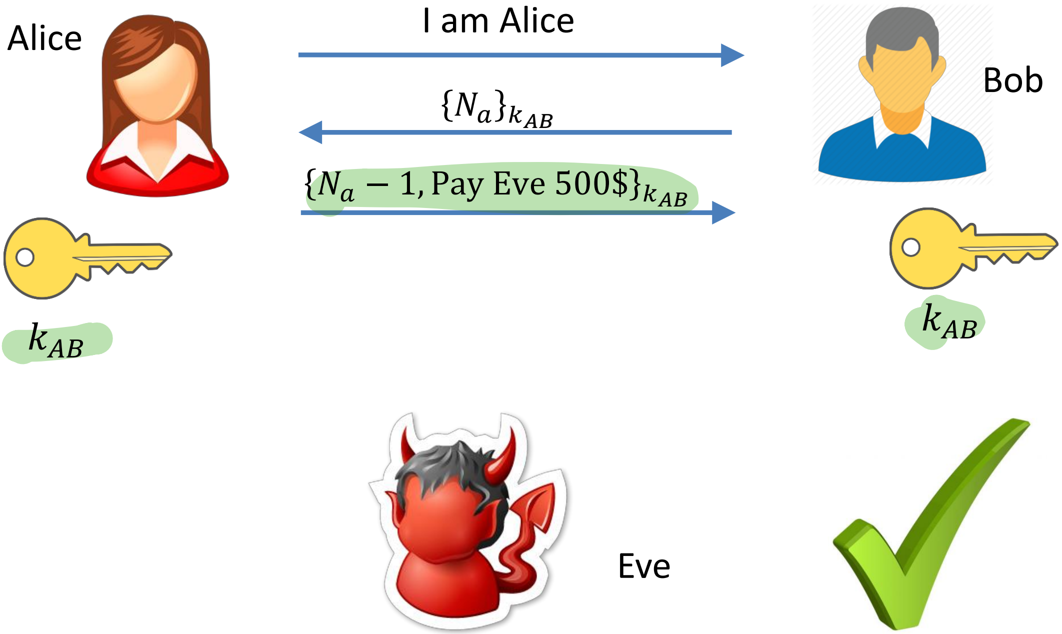
Attempt #3: use nonce



Attempt #3: use nonce



Attempt #4



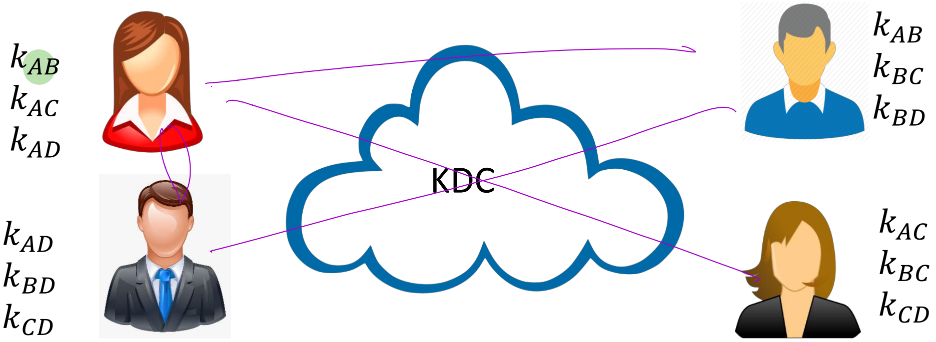
Key establishment

- The protocol worked because Alice and Bob shared a key
- How do parties agree on a key?
  - Run a key agreement protocol (later in the semester)
  - Use a trusted third party (this lecture)
- Key distribution center (KDC):
  - Shares a key with each entity
  - Single point of failure
  - Reasonable assumption for organizations
  - Not useful for open environments (e.g. the Internet)

Naïve solution

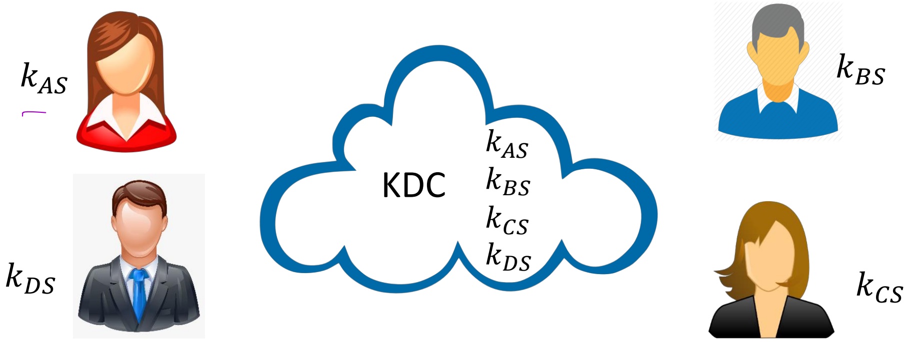
- KDC generates a key for each pair
- Number of keys $n(n-1)$, number of key pairs $\frac{n(n-1)}{2} = \binom{n}{2}$
- Drawbacks:
  - Quadratic number of keys
  - Adding new users is complex
- May be useful for static small networks

$O(n^2)$ Keys.

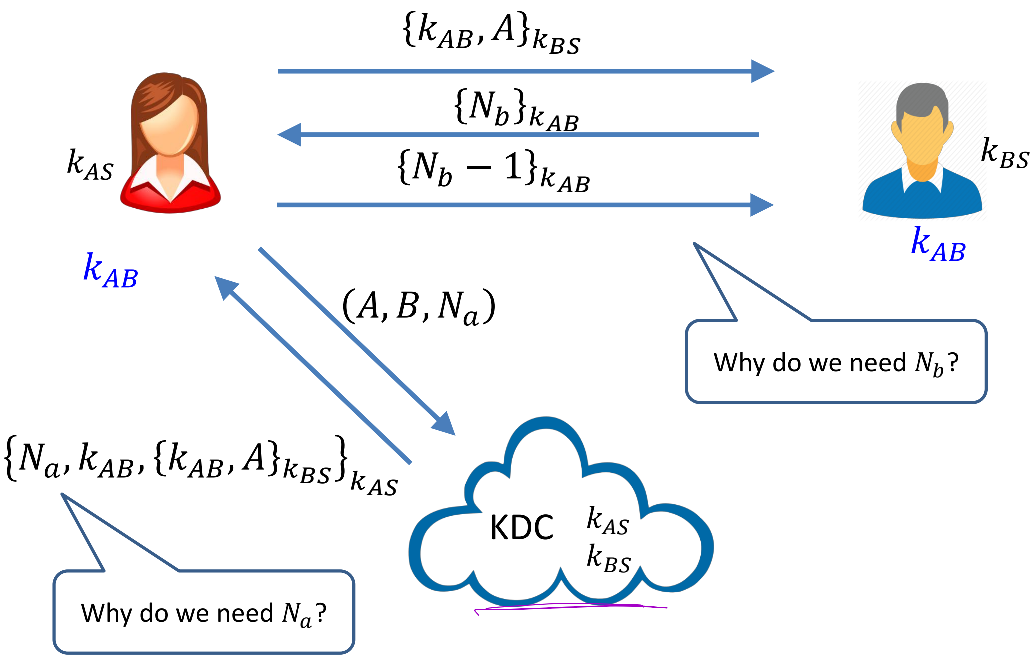


Desire: solution with linear keys

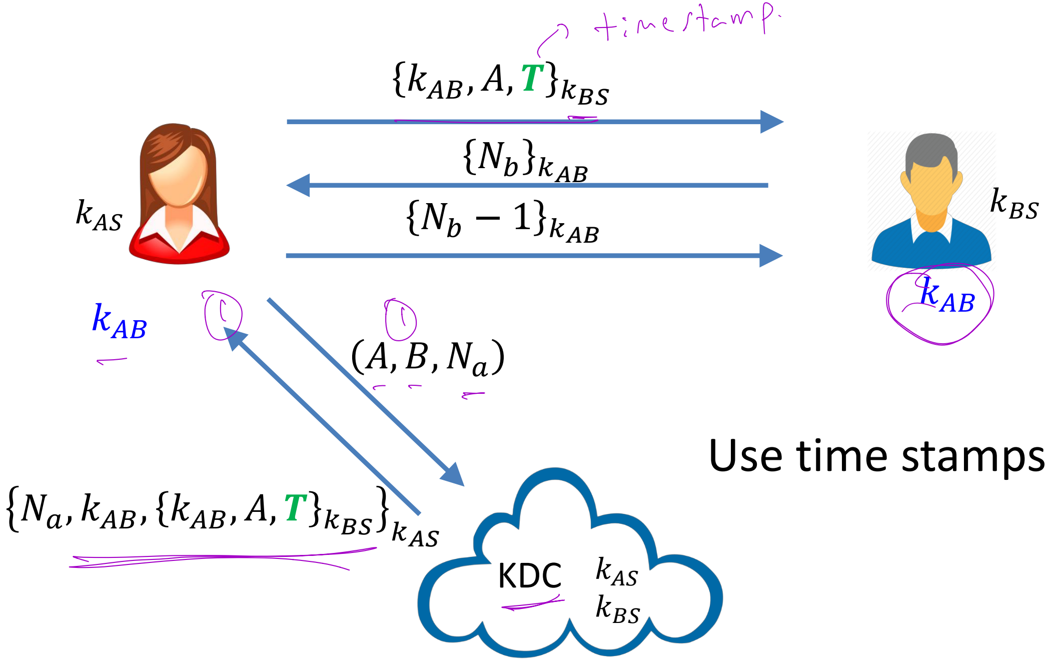
- KDC shares a key with each user
- Number of keys $2n$ $O(n)$ total # of keys in the system.
- Number of key pairs n
- These are long-term keys
- Alice and Bob establish a fresh session key



Needham-Schroeder Protocol (1978)



Fixed Needham-Schroeder

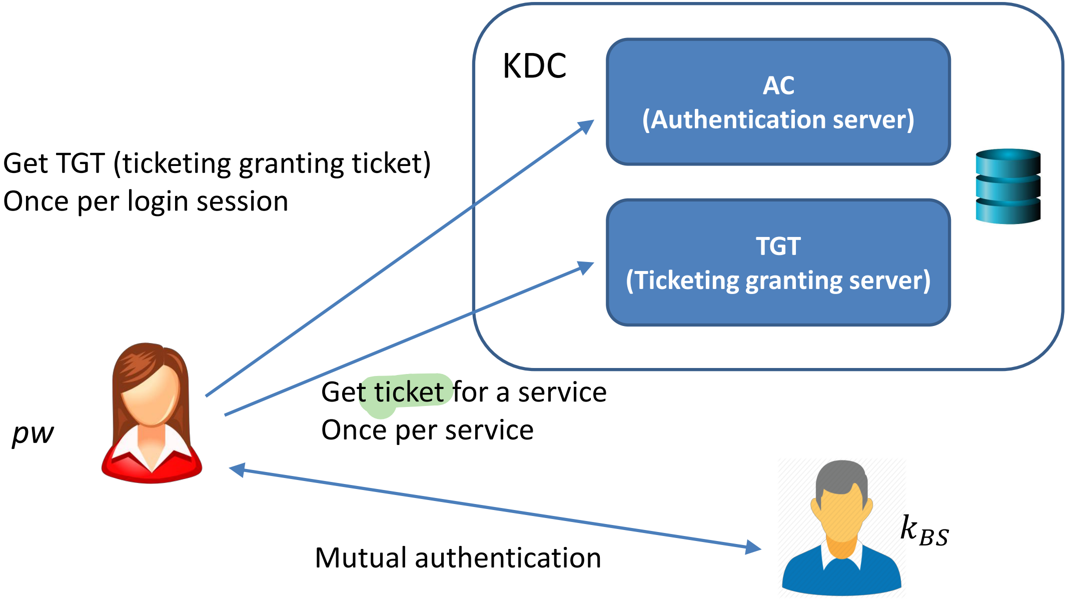


Use time stamps

Kerberos

- Developed in MIT in the ‘80s
- Based on Needham-Schroeder
  - Versions 1-3 not published
  - Version 4 not secure
  - Version 5 published in 1993
- Widely used nowadays:
  - The basis of Microsoft’s active directory
  - Many Unix versions

Kerberos

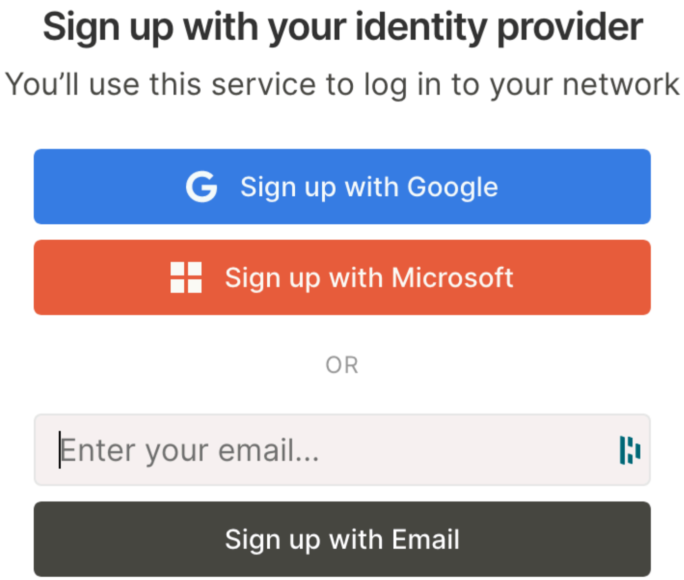

KDC
AC
(Authentication server)
TGT
(Ticketing granting server)
Get TGT (ticketing granting ticket)
Once per login session
Get ticket for a service
Once per service
pw
k_{BS}
Mutual authentication

Kerberos

- Passwords are not sent over the network
- Alice's key k_{AS} is a hash of her password

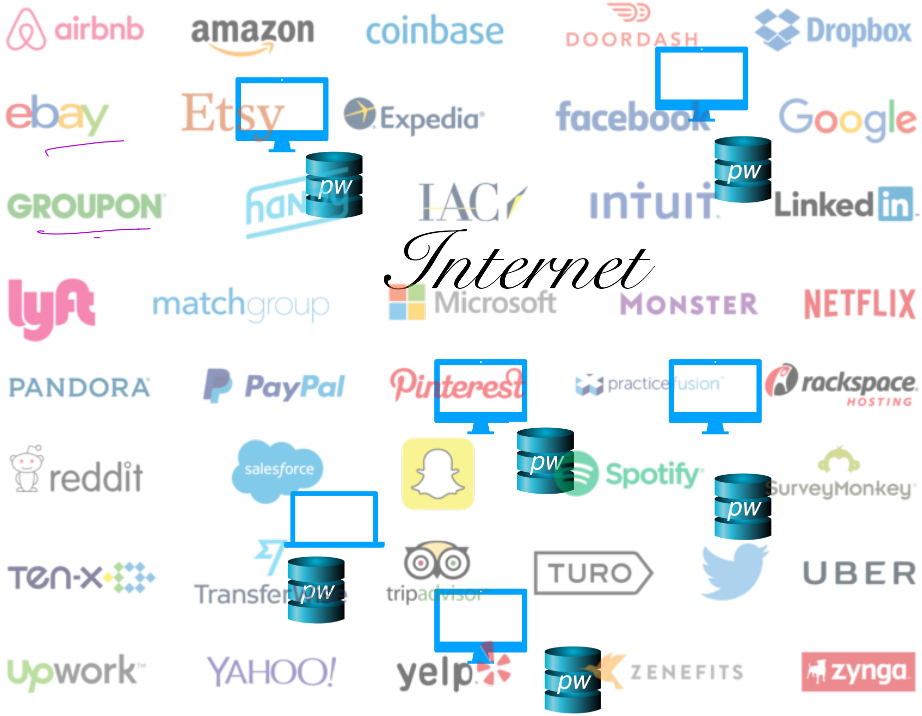
- Kerberos weaknesses:
  - KDC is a single point of failure
  - DoS the KDC and the network ceases to function
  - Compromise the KDC leads to network-wide compromise
  - Time synchronization is a very hard problem

“Single Sign on”



Same problem as before

Alice
pw



"Single Sign on"

Alice

pw

Sign up with your identity provider

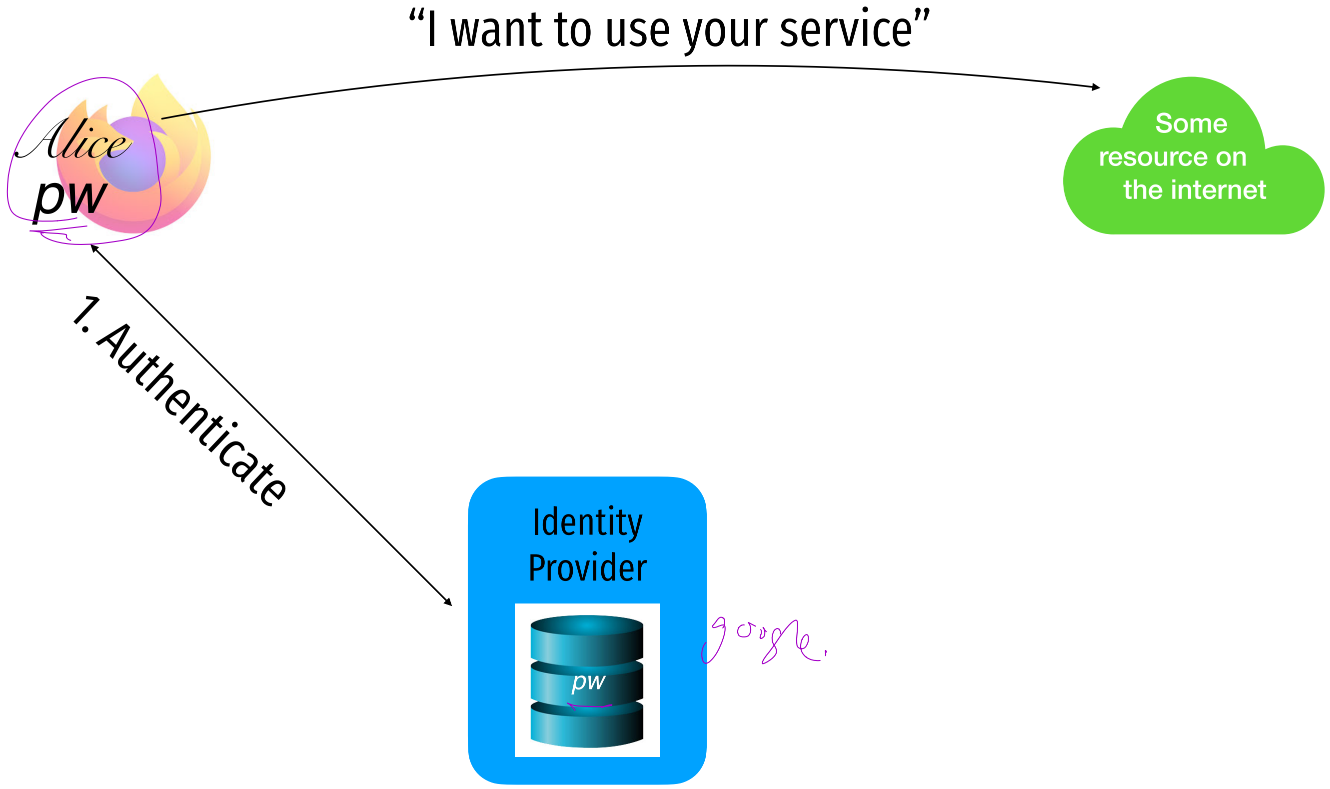
You'll use this service to log in to your network

Sign up with Google

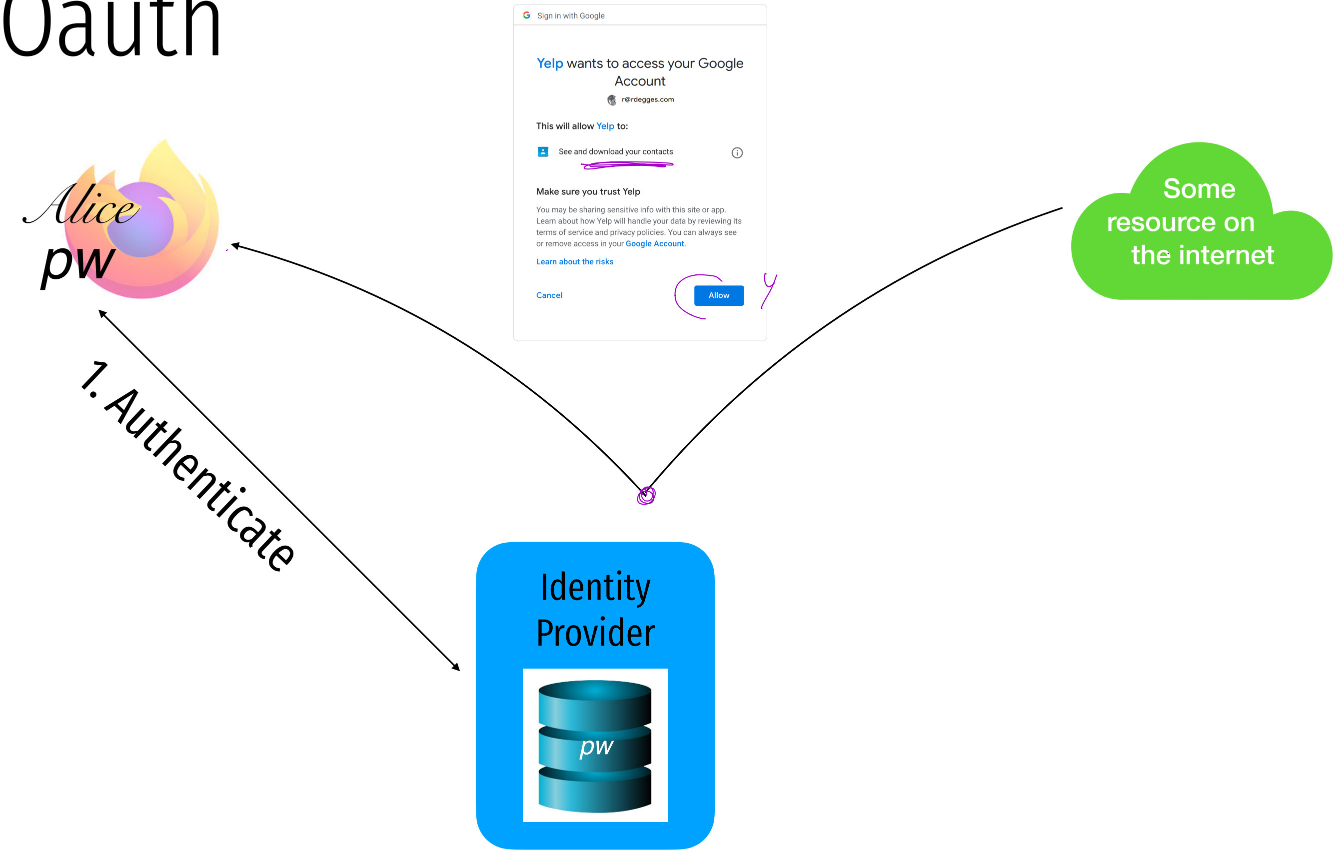
Sign up with Microsoft

OR

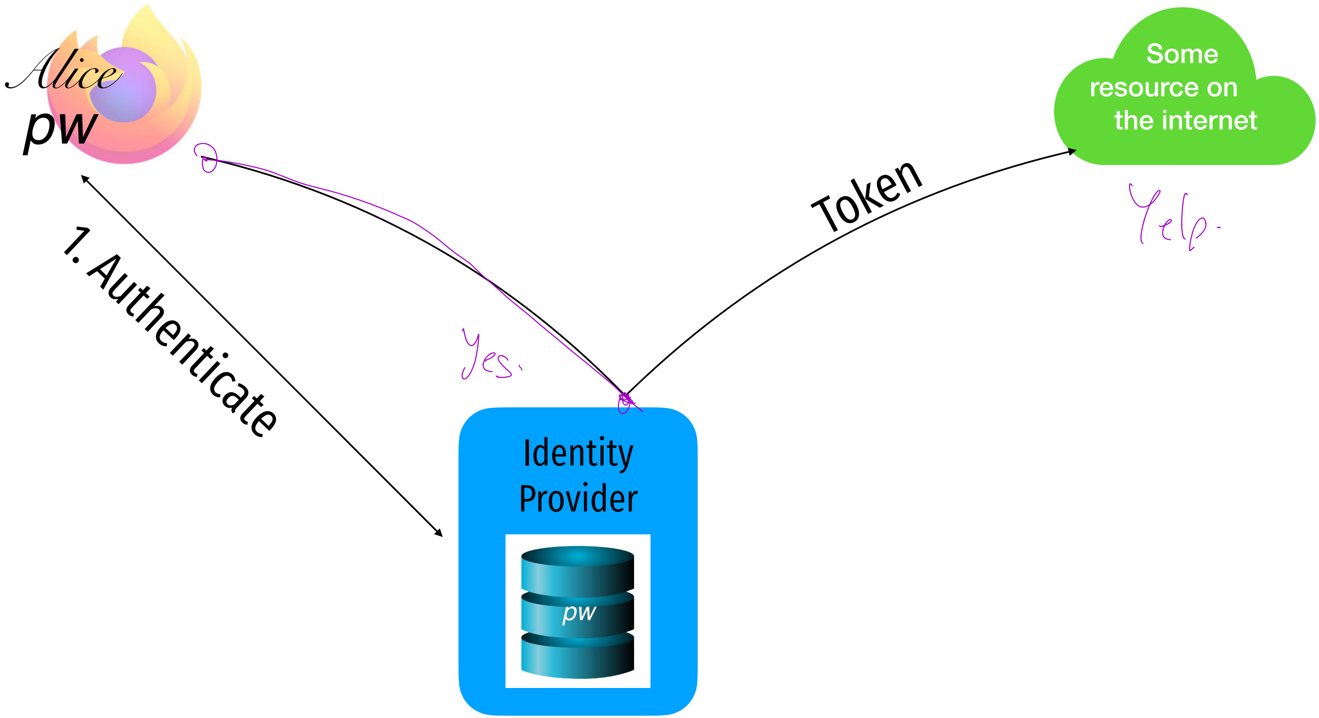
Oauth


"I want to use your service"
Alice
pw
Some resource on the internet
1. Authenticate
Identity Provider
pw

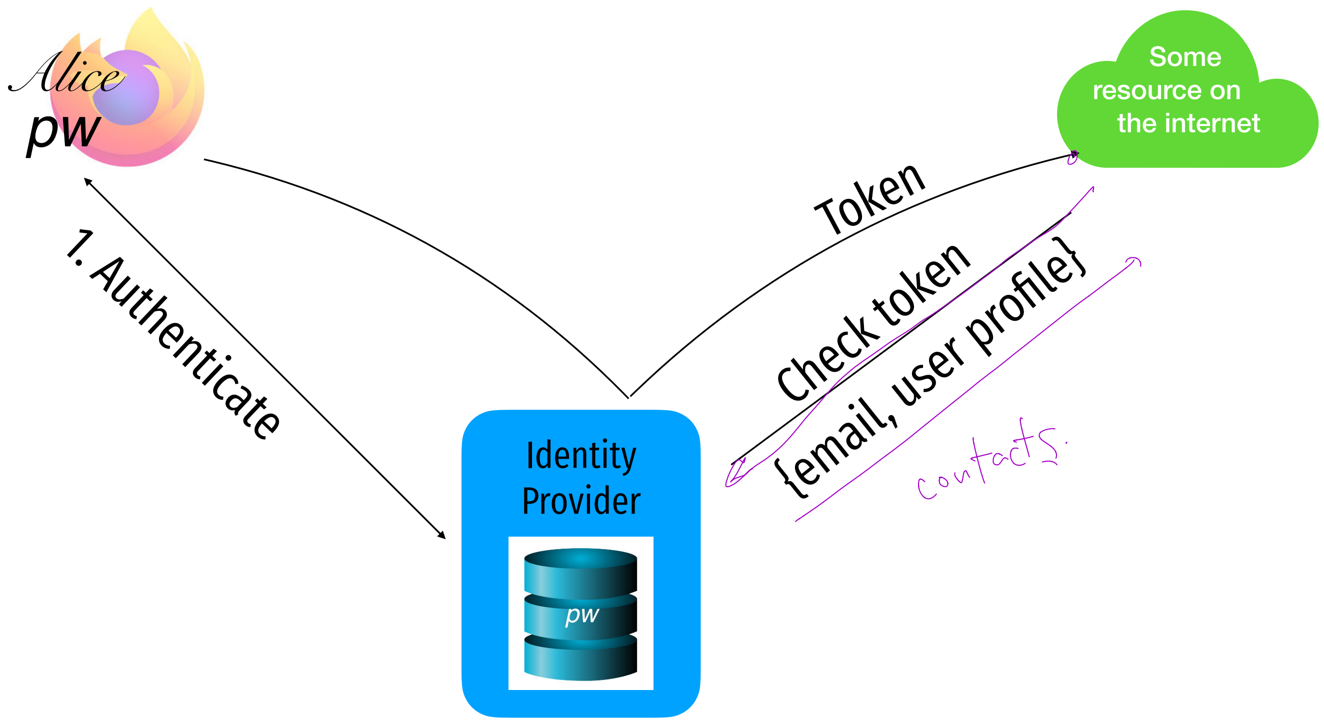
Oauth


Alice
pw
Sign in with Google
Yelp wants to access your Google Account
r@rdegges.com
This will allow Yelp to:
See and download your contacts
Make sure you trust Yelp
You may be sharing sensitive info with this site or app. Learn about how Yelp will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.
Learn about the risks
Cancel
Allow
Some resource on the internet
1. Authenticate
Identity Provider
pw

Oauth


Alice
pw
1. Authenticate
Yes.
Token
Some resource on the internet
Yelp.
Identity Provider
pw

Oauth



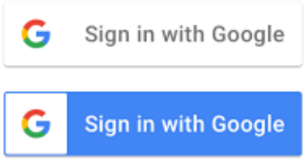
Alice
pw
1. Authenticate
Identity Provider
pw
Token
Check token
{email, user profile}
contacts
Some resource on the internet

Attacks against “Login with…” services

Log in with Twitter

Use Log in with Twitter, also known as Sign in with Twitter, to place a button on your site or application which allows Twitter users to enjoy the benefits of a registered user account in as little as one click. This works on websites, iOS, mobile, and desktop applications.

Sign in with Twitter



what is the main problem ??

single point of failure

Use Sign in with Apple on your Apple device

Using Sign in with Apple is quick and easy on any Apple device with the latest software. Make sure you're signed in with your Apple ID on your device.

1. Tap the Sign in with Apple button on the participating app or website.

   If the app or site has not requested any information to set up your account, check that your Apple ID is correct and go to Step 4.

   If you're asked to provide your name and email address, Sign in with Apple automatically fills in the information from your Apple ID. You can edit your name if you like and choose Share My Email or Hide My Email.

   Tap Continue and confirm with a quick Face ID, Touch ID, or device passcode to sign in. If you don't have Face ID, Touch ID, or a passcode set up, enter your Apple ID password.

Sources

1. Many slides courtesy of Wil Robertson: https://wkr.io
2. Many slides courtesy of Ran Cohen