

# 2550 Intro to cybersecurity

## L11: Passwords

abhi shelat

Thanks Christo for slides!


# Choosing Passwords

Bad Algorithms  character classes 3 = 12

Better Heuristics

Password Reuse 

# Password Reuse

- People have difficulty remembering >4 passwords
    - Thus, people tend to reuse passwords across services
    - What happens if any one of these services is compromised?
  - Service-specific passwords are a beneficial form of compartmentalization
    - Limits the damage when one service is inevitably breached
  - Use a password manager — *1ocKwise*
  - Some service providers now check for password reuse
    - Forbid users from selecting passwords that have appeared in leaks
- 

## Sites

Sort By: Folder (a-z)

Favorites (8)



AirBnB  
fan@lastpass.com

Amazon  
fan@lastpass.com

Launch

Best Buy  
fan@lastpass.com



Dropbox  
fan@lastpass.com



Evernote  
fan@lastpass.com

Facebook  
fan@lastpass.com



Pocket  
fan@lastpass.com

Twitter  
fan@lastpass.com

Banking and Finance (3)

Read Only • Shared Folder

Bank of America  
fan@lastpass.com

Fidelity  
fan@lastpass.com

Mint  
fan@lastpass.com





# Two Factor Authentication

Biometrics

SMS

Authentication Codes

Smartcards & Hardware Tokens

# Types of Secrets

- Actors provide their secret to **log-in** to a system
- Three classes of secrets:

1. Something you know

- Example: a password

2. Something you have

- Examples: a smart card or smart phone

3. Something you are

- Examples: fingerprint, voice scan, iris scan

"have"

# Biometrics

- ancient Greek: bios = "life", metron = "measure"
- Physical features
  - Fingerprints
  - Face recognition
  - Retinal and iris scans
  - Hand geometry
- Behavioral characteristics
  - Handwriting recognition
  - Voice recognition
  - Typing cadence
  - Gait

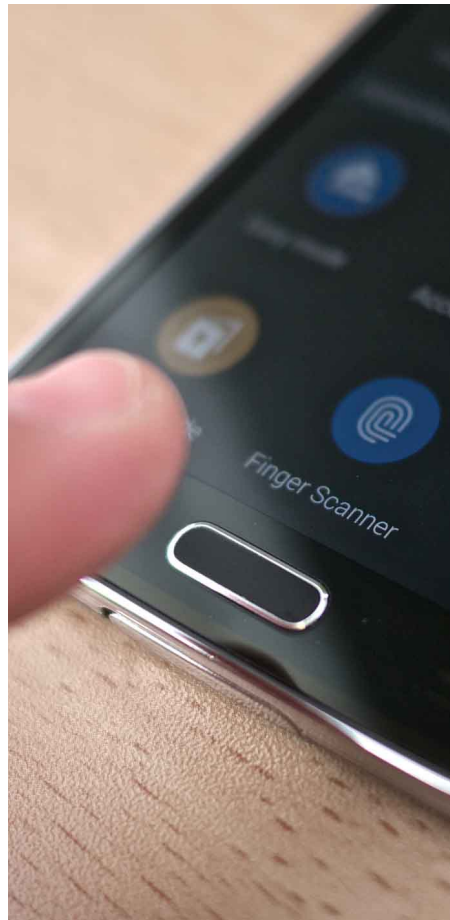


# Fingerprints

- Ubiquitous on modern smartphones, some laptops
- Secure?
  - May be subpoenaed by law enforcement
  - Relatively easy to compromise
    1. Pick up a latent fingerprint (e.g. off a glass) using tape or glue
    2. Photograph and enhance the fingerprint
    3. Etch the print into gelatin backed by a conductor
    4. Profit ;)

[https://www.theregister.co.uk/2002/05/16/gummi\\_bears\\_defeat\\_fingerprint\\_sensors/](https://www.theregister.co.uk/2002/05/16/gummi_bears_defeat_fingerprint_sensors/)

*a loss of finger*



# Facial Recognition

- Popularized by FaceID on the iPhone X
- Secure?



# Facial Recognition

- Popularized by FaceID on the iPhone X
- Secure?
  - It depends



# Facial Recognition

- Popularized by FaceID on the iPhone X
- Secure?
  - It depends
- Vulnerable to law enforcement requests
- Using 2D images?
  - Not secure
  - Trivial to break with a photo of the target's face



# Facial Recognition

- Popularized by FaceID on the iPhone X
- Secure?
  - It depends
- Vulnerable to law enforcement requests
- Using 2D images?
  - Not secure
  - Trivial to break with a photo of the target's face
- Using 2D images + 3D depth maps?
  - More secure, but not perfect
  - Can be broken by crafting a lifelike mask of the target





Specially processed area

2D images

Silicone nose

3D printed frame



## By Press Association

---

*Saturday, October 19, 2019 - 01:20 PM*

Google has confirmed the Face Unlock system on its new Pixel 4 smartphone can allow access to the device even when the user has their eyes closed.

Early testers of the phone, as well as security experts, have raised concerns it could lead to unauthorised access to the device.

It has been suggested someone else could gain access to the phone by holding it in front of the face of its sleeping owner, but Google said it meets security requirements.

The technology giant unveiled the new phone earlier this week.

In a statement, Google said: "Pixel 4 Face Unlock meets the security requirements as a strong biometric and can be used for payments and app authentication, including banking apps.

"It is resilient against unlock attempts via other means, like with masks.

"If you want to temporarily disable Face Unlock, you can use lockdown mode to temporarily require a PIN/pattern/password.

# Voice Recognition

---

- Secure?
  - Very much depends on the implementation

→ CS2017

"noise in your  
cellphone call"

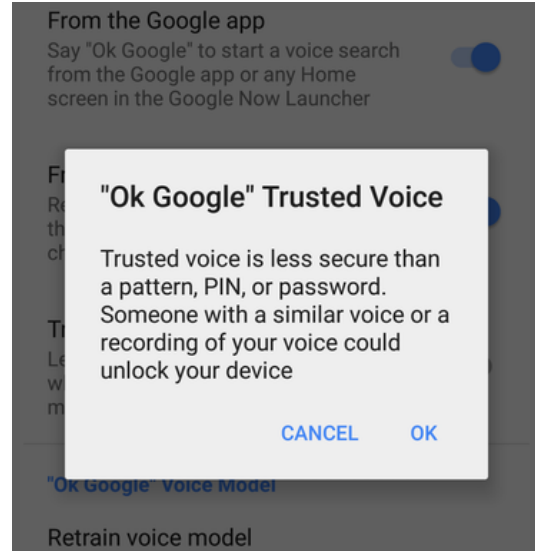


# Voice Recognition

- Secure?
  - Very much depends on the implementation
- Some systems ask you to record a static phrase
  - E.g. say “unlock” to unlock
  - This is wildly insecure
    - Attacker can record and replay your voice

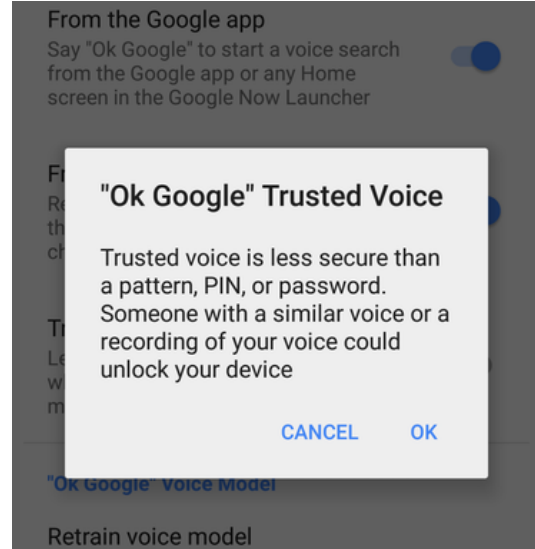
# Voice Recognition

- Secure?
  - Very much depends on the implementation
- Some systems ask you to record a static phrase
  - E.g. say “unlock” to unlock
  - This is wildly insecure
    - Attacker can record and replay your voice



# Voice Recognition

- Secure?
  - Very much depends on the implementation
- Some systems ask you to record a static phrase
  - E.g. say “unlock” to unlock
  - This is wildly insecure
    - Attacker can record and replay your voice
- Others ask you to train a model of your voice
  - Train the system by speaking several sentences
  - To authenticate, speak several randomly chosen words
  - Not vulnerable to trivial replay attacks, but still vulnerable
    - Given enough samples of your voice, an attacker can train a synthetic voice AI that sounds just like you



# Fundamental Issue With Biometrics

- Biometrics are immutable
  - You are the password, and you can't change
  - Unless you plan on undergoing plastic surgery?
- Once compromised, there is no reset
  - Passwords and tokens can be changed
- Example: the Office of Personnel Management (OPM) breach
  - US gov agency responsible for background checks
  - Had fingerprint records of all people with security clearance
  - Breached by China in 2015, all records stolen :(

# Something You Have

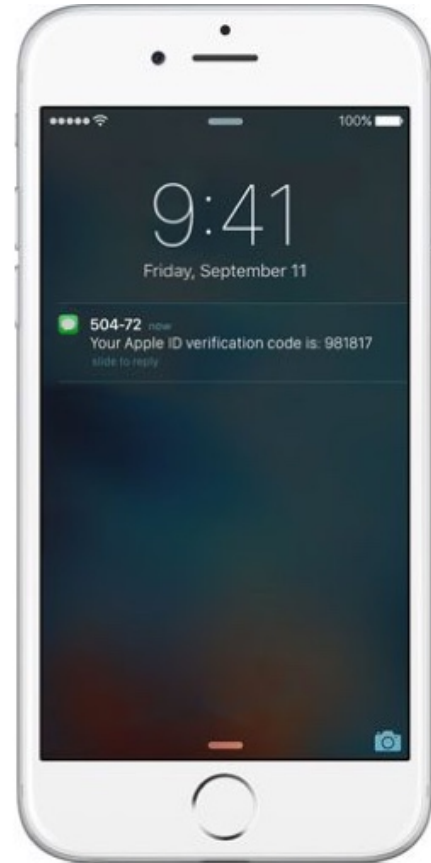
---

- Two-factor authentication has become more commonplace
- Possible second factors:
  - SMS passcodes ✓
  - Time-based one time passwords ✓
  - Hardware tokens ✓

best

# SMS Two Factor

- Relies on your phone number as the second factor
- Key assumption: only your phone should receive SMS sent to your number



# SMS Two Factor

- Relies on your phone number as the second factor
  - Key assumption: only your phone should receive SMS sent to your number
- SMS two factor is deprecated. Why?

↑  
this  
assumption  
is unsound,  
broken in  
practice,



# SMS Two Factor

- Relies on your phone number as the second factor
  - Key assumption: only your phone should receive SMS sent to your number
- SMS two factor is deprecated. Why?
- Social engineering the phone company
  1. Call and pretend to be the victim
  2. Say “I got a new SIM, please activate it”
  3. If successful, phone calls and SMS are now sent to your SIM in your phone, instead of the victim
- Not hypothetical: successfully used against many victims

*SIM  
JACKING  
SWAPPING*

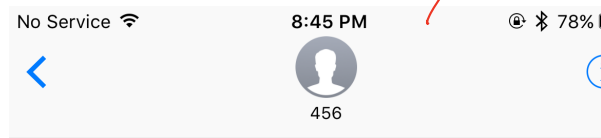




First, criminals call a cell phone carrier's tech support number pretending to be their target. They explain to the company's employee that they "lost" their SIM card, requesting their phone number be transferred, or ported, to a new SIM card that the hackers themselves already own. With a bit of social engineering—perhaps by providing the victim's Social Security Number or home address (which is often available from one of the many data breaches that have happened in the last few years)—the criminals convince the employee that they really are who they claim to be, at which point the employee ports the phone number to the new SIM card.

Game over.

"With someone's phone number," a hacker who does SIM swapping told me, "you can get into every account they own ~~within minutes~~ and they can't do anything about it."



Text Message  
Today 8:38 PM

Free T-Mobile Msg: The SIM card for [REDACTED] has been updated. Account activity details at [my.t-mobile.com](https://my.t-mobile.com). Call 1-800-937-8997 if change is unauthorized.

# One Time Passwords

- Generate ephemeral passcodes that change over time
- To login, supply normal password and the current one time password
- Relies on a shared secret between your mobile device and the service provider
  - Shared secret allows both parties to know the current one time password

Changes every few minutes

*PRF (time)  
sk*



 AUTHY

ACME INC TOKEN IS:

6883932 



Duo Mobile



Lastpass Authenticator



Google Authenticator

# Time-based One-time Password Algorithm

$T0$  = <the beginning of time, typically Thursday, 1 January 1970 UTC>

$Tl$  = <length of time the password should be valid>

$K$  = <shared secret key>

$d$  = <the desired number of digits in the password>

$TC = \text{floor}(\frac{\text{unixtime}(\text{now}) - \text{unixtime}(T0)}{Tl}),$

$\text{TOTP} = \text{HMAC}(K, TC) \% 10^d$  → last 6 characters

Specially formatted  
SHA1-based signature

# Time-based One-time Password Algorithm

$T0$  = <the beginning of time, typically Thursday, 1 January 1970 UTC>

$T1$  = <length of time the password should be valid>

$K$  = <shared secret key>

$d$  = <the desired number of digits in the password>

$TC = \text{floor}((\text{unixtime}(\text{now}) - \text{unixtime}(T0)) / T1),$

$\text{TOTP} = \text{HMAC}(K, \text{TC}) \% 10^d$

Specially formatted  
SHA1-based signature

Given  $K$ , this algorithm can  
be run on your phone and by  
the service provider

# Secret Sharing for TOTP

"Gen"

## Enable Two-Step Sign in

An authenticator app generates the code automatically on your smartphone. Free apps are available for all smartphone platforms including iOS, Android, Blackberry and Windows. Look for an app that supports time-based one-time passwords (TOTP) such as Google Authenticator or Duo Mobile.

To set up your mobile app, add a new service and scan the QR code.



(SK)  
PRF

If you can't scan the code, enter this secret key manually: fvxo

USE SMS INSTEAD

CANCEL

NEXT STEP

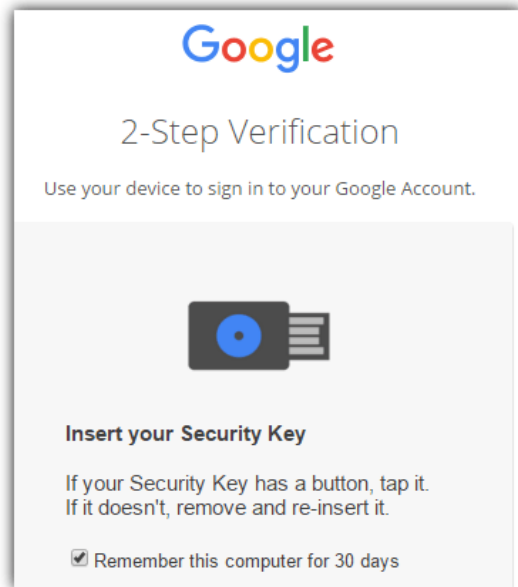
# Hardware Two Factor

- Special hardware designed to hold cryptographic keys
- Physically resistant to key extraction attacks
  - E.g. scanning tunneling electron microscopes
- Uses:
  - 2<sup>nd</sup> factor for OS log-on
  - 2<sup>nd</sup> factor for some online services
  - Storage of PGP and SSH keys



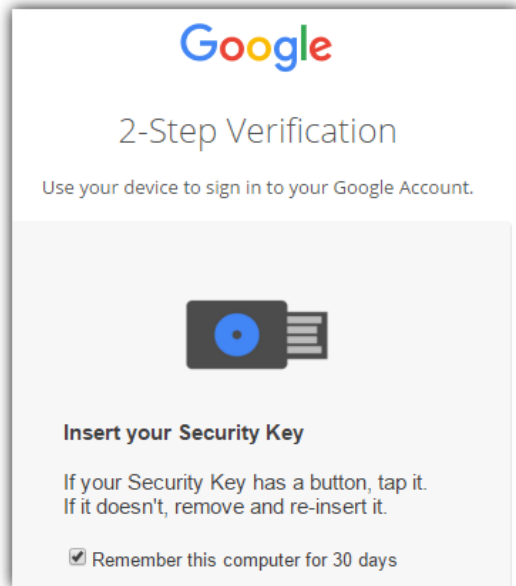
# Universal 2<sup>nd</sup> Factor (U2F)

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



# Universal 2<sup>nd</sup> 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 ;)



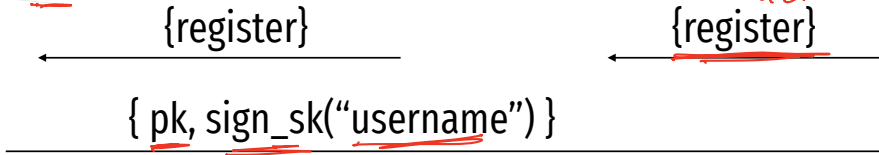


# How does U2F work?



**Init**

Make a signing key  
 $(sk, pk)$



**User, pk**

**Login**

Sign challenge using sk

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



Verify<sub>pk</sub><sup>s<sub>r</sub></sup>(ch)

Vulnerable to simple attack

# Simple Phishing

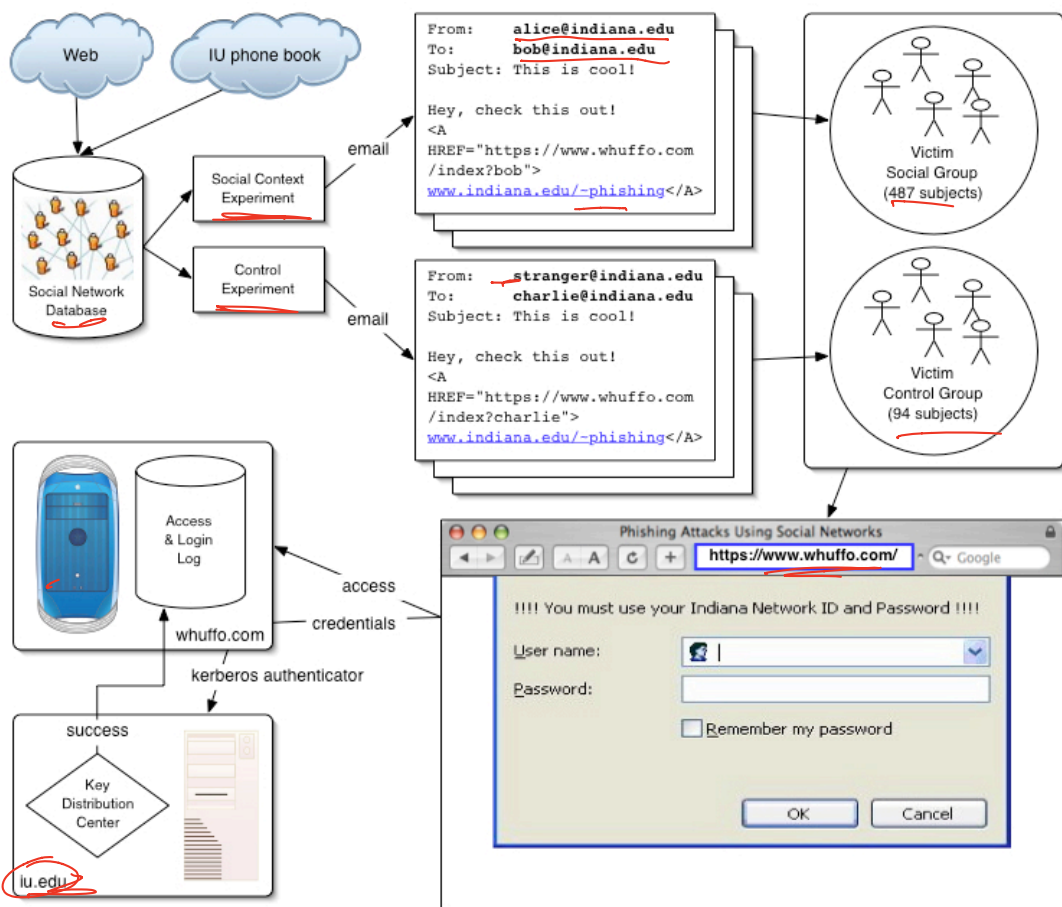
**Lure:** A spammed email with a call to action from a seemingly legitimate source encouraging the user to visit a hook website.

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

# Spear Phishing @ IU

A red underline is drawn under the text "Spear Phishing @ IU". It consists of a long horizontal line under "Spear Phishing" and a shorter, slightly curved line under "@ IU".

Experiment by T. Jagatic, N. Johnson, M. Jakobsson, F. Menczer.



# Control Phishing Success Rate:

# 9-23%

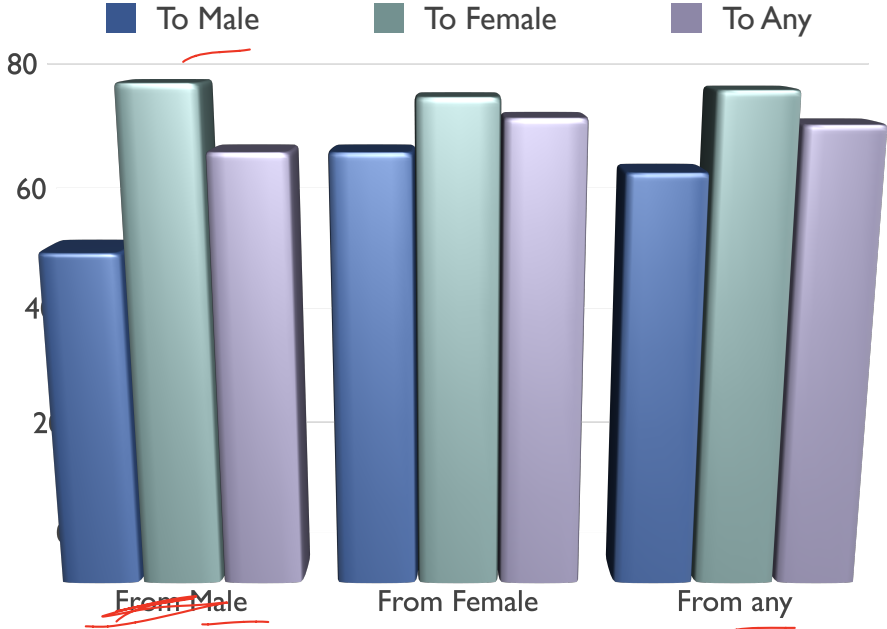
with 95% Confidence Interval

**Spear** Phishing Success Rate:

**68-72%**

with 95% Confidence Interval

# Spear Phishing Success Rate by Gender





# VOIP Phishing

**Lure:** Get victim to call a bogus 800... number about their account.

**Hook:** Have the human on the other end extract the victim's information.

From: FlagStar Bank <[usflag60536@flagstar.com](mailto:usflag60536@flagstar.com)>

Date: 11 Sep 2007 10:55:21 -0400

To: <[samyers@indiana.edu](mailto:samyers@indiana.edu)>

Subject: You have one new private message

Dear FlagStar Bank card holder,

You have one new private message.

Please call free 800-870-8124 to listen to your private message.

Copyright ©2007 FlagStar Bank

**Source: Steven Myers, IU**

From: FlagStar Bank <[usflag60536@flagstar.com](mailto:usflag60536@flagstar.com)>

Date: 11 Sep 2007 10:55:21 -0400

To: <[samyers@indiana.edu](mailto:samyers@indiana.edu)>

Subject: You have one new private message

Dear FlagStar Bank card holder,

You have one new private message.

Please call free 800-870-8124 to listen to your private message.

Copyright ©2007 FlagStar Bank

**Source: Steven Myers, IU**



## 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

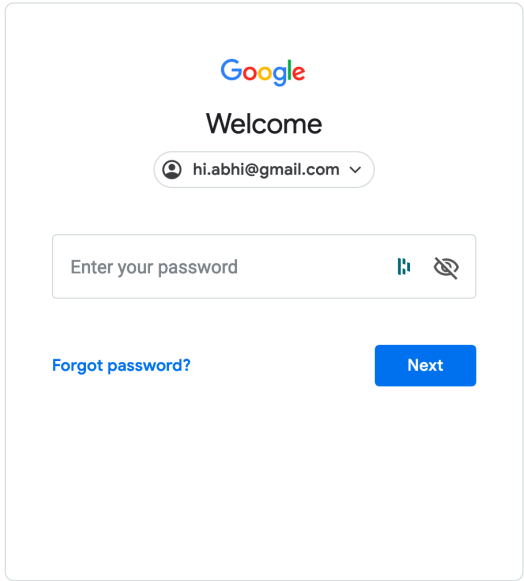
Location: Ukraine

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

[CHANGE PASSWORD](#)

Best,  
The Gmail Team

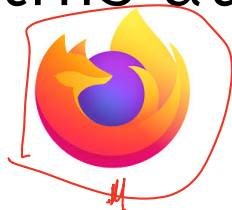




The image shows a Google sign-in form. At the top is the Google logo, followed by the word "Welcome". Below that is a dropdown menu showing the email address "hi.abhi@gmail.com". A password input field contains the text "Enter your password" and has icons for voice search and password visibility. At the bottom left of the form is a link for "Forgot password?" and at the bottom right is a blue "Next" button.

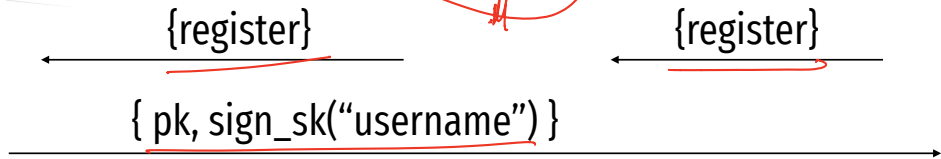
# U2F can help prevent this attack

Website  
(Relying Party)



Init

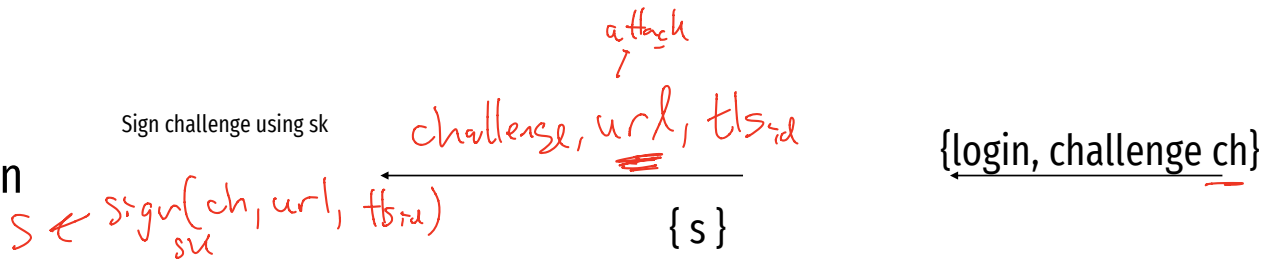
Make a signing key  
(sk,pk)



User, pk

Login

Sign challenge using sk



Verify(ch, url, tls\_id)

# U2F can help prevent this attack



Website  
(Relying  
Party)

Init

Make a signing key  
(sk,pk)

$\xleftarrow{\{\text{register}\}}$   $\xleftarrow{\{\text{register}\}}$

$\xrightarrow{\{\text{pk, sign\_sk}(\text{"username"})\}}$

User, pk

Login

Sign challenge using sk

$\xleftarrow{\{\text{login, ch, origin, tls\_id}\}}$   $\xleftarrow{\{\text{login, challenge ch}\}}$

*added  
by  
browser*

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

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



# U2F can help prevent tracking

Init

Make a signing key  
(sk,pk)



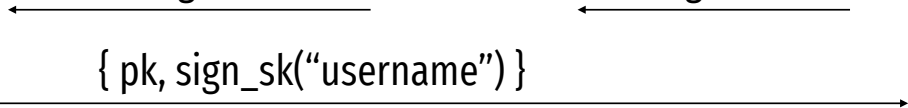
Website  
(Relying  
Party)

{register}

{register}

{ pk, sign\_sk("username") }

User, pk



# U2F can help prevent tracking

Init

Make a signing key  
(sk,pk)  
And link it with  
appid, and create  
A token "h"



Website  
(Relying  
Party)

{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  
(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

{login, h, ch, origin, tls\_id}

{login, appid, challenge ch}

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

{ s, h }

Verify<sub>pk</sub>(ch, url, tls<sub>id</sub>)  
Check h

Sending request with appId: https://u2f.bin.coffee

```
{
  "version": "U2F_V2",
  "challenge": "uQnl3M4Rj3FZgs6WjyLaZAfwRh4"
}
```

Got response:

```
{
  "clientData": "eyJjaGFsbgVuZ2UiOiJ1UW5sM000UmozRlpnczZXanlMYVpBZndSaDQiLCJvcmlnaW4iOiJodHRwciovL3UyZi5iaW4uY29mZmVlIiwidHlwIjoibmF2",
  "errorCode": 0,
  "registrationData": "BQRSuRlPv0p5udQ55vVhucf3N50q6...",
  "version": "U2F_V2"
}
```

Key Handle: 0r0Z0p0F0E0-0d0W0c0Q0b0X0i020C0w0-0E0v0h0t0T0T0P0\_0-090\_0a050P0e030u0b0z010K0Q0r000f0u030\_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

# 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<sup>nd</sup> factors

# Status Check

- At this point, we have discussed:
  - How to securely store passwords
  - Techniques used by attackers to crack passwords
  - Biometrics and 2<sup>nd</sup> 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?

# Authentication in Unix/Linux

- Users authenticate with the system by interacting with *login*
  - Prompts for username and password
  - Credentials checked against locally stored credentials
- By default, password policies specified in a centralized, modular way
  - On Linux, using Pluggable Authentication Modules (PAM)
  - Authorizes users, as well as environment, shell, prints MOTD, etc.



# Example PAM Configuration

```
# cat /etc/pam.d/system-auth
#%PAM-1.0

auth required pam_unix.so try_first_pass
auth optional pam_permit.so
auth required pam_env.so

account required pam_unix.so
account optional pam_permit.so
account required pam_time.so

password required pam_unix.so try_first_pass nullok sha512 shadow
password optional pam_permit.so

session required pam_limits.so
session required pam_unix.so
session optional pam_permit.so
```

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

# Unix Passwords

- Traditional method: *crypt*
  - 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*

`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:$1$l3RxU5F1$: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`

`amislove:x:1002:1000:Amislove:/home/amislove/~/bin/sh`

`$<algo>$<salt>$<hash>`

Algo: 1 = MD5, 5 = SHA256, 6 = SHA512

`/etc/shadow`

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

`cbw:$1$0nSd5ewF$0df/3G7iSV49nsbAa/5gSg:9479:0:10000:::`

`amislove:$1$l3RxU5F1$:8172:0:10000:::`

# Distributed Authentication

- Early on, people recognized the need for authentication in distributed environments
  - Example: university lab with many workstations
  - Example: file server that accepts remote connections
- Synchronizing and managing password files on each machine is not scalable
  - Ideally, you want a centralized repository that stores policy and credentials

# The Yellow Pages

- Network Information Service (NIS), a.k.a. the Yellow Pages
  - Developed by Sun to distribute network configurations
  - Central directory for users, hostnames, email aliases, etc.
  - Exposed through *yp*\* family of command line tools
- For instance, depending on */etc/nsswitch.conf*, hostname lookups can be resolved by using
  - */etc/hosts*
  - DNS
  - NIS
- Superseded by NIS+, LDAP,



# NIS Password Hashes

- Crypt based password hashes
- Can easily be cracked
- Many networks still rely on insecure NIS

```
[cbw@workstation ~] ypcat passwd
afbjune:qSAH.evuYFHAM:14532:65104:::/home/afbjune:/bin/bash
philowe:T.yUMej3XSNAME:13503:65104:::/home/philowe:/bin/bash
bratus:2omkwsYXWiLDo:6312:65117:::/home/bratus:/bin/tcsh
adkap:ZfHdSwSz9WhKU:9034:65118:::/home/adkap:/bin/zsh
amitpoon:i3LjTqgU9gYSc:8198:65117:::/home/amitpoon:/bin/tcsh
kcole:sgYtUs0tyk38k:14192:65104:::/home/kcole:/bin/bash
david87:vA06wxjJEUGBE:13055:65101:::/home/david87:/bin/bash
loch:6HgIQrVkcBeiw:13729:65104:::/home/loch:/bin/bash
ppkk315:s6CTSAkqqr/nU:14061:65101:::/home/ppkk315:/bin/bash
haynesma:JYWaQUARSqDQE:14287:65105:::/home/haynesma:/bin/bash
ckubicek:jYpwYhqqr3tA:10937:65117:::/home/ckubicek:/bin/tcsh
mwalz:wPIa5Bv/tFVb2:9103:65118:::/home/mwalz:/bin/tcsh
sushma:G6XNe18GpeQj.:13682:65104:::/home/sushma:/bin/bash
guerin1:n0Da2Tm09MDBI:14512:65105:::/home/guerin1:/bin/bash
```

# Distributed Authentication Revisited

- Goal: a user would like to use some resource on the network
- File server, printer, database, mail server, etc.

cbw



Auth Server

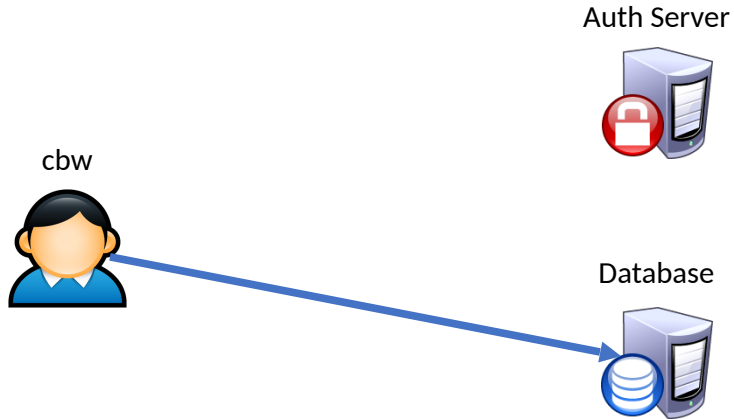


Database



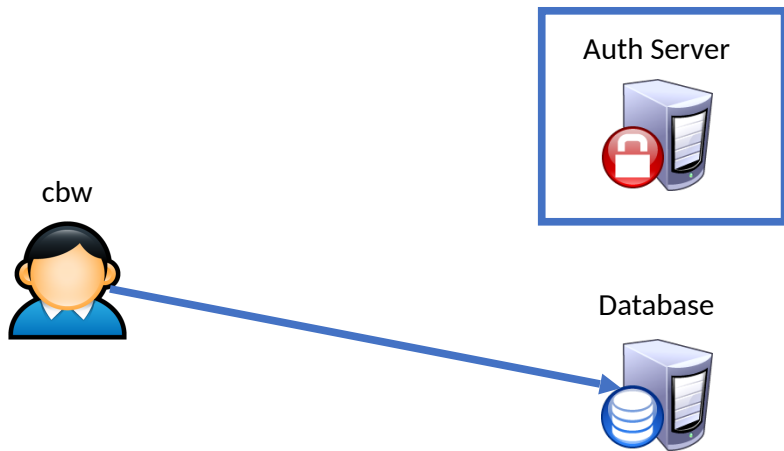
# Distributed Authentication Revisited

- Goal: a user would like to use some resource on the network
- File server, printer, database, mail server, etc.



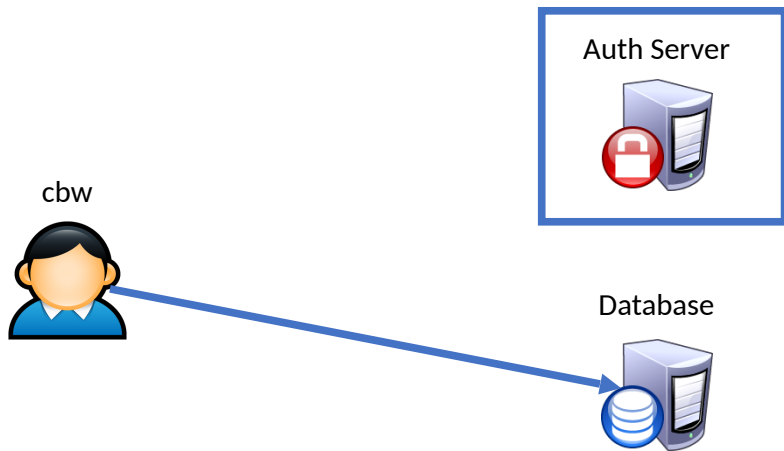
# Distributed Authentication Revisited

- Goal: a user would like to use some resource on the network
  - File server, printer, database, mail server, etc.
- Problem: access to resources requires authentication
  - Auth Server contains all credential information
  - You do not want to replicate the credentials on all services



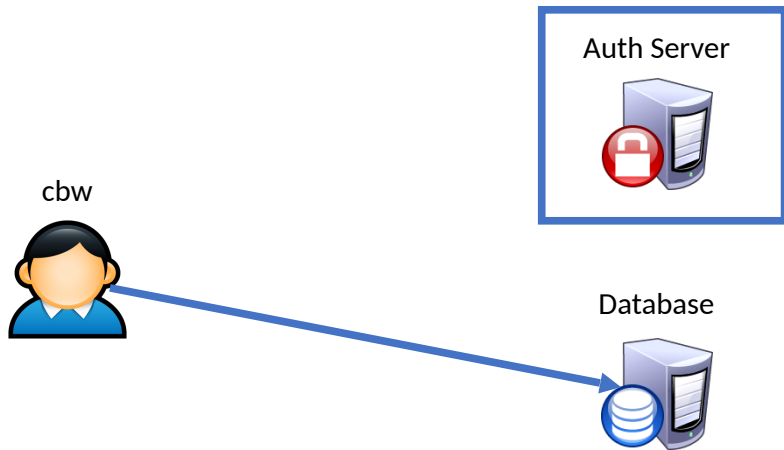
# Attacker Goals and Threat Model

- Goal: steal credentials and gain access to protected resources
- Local attacker – may spy on traffic
- Active attacker – may send messages
- In some cases, may be able to steal information from users



# Attacker Goals and Threat Model

- Goal: steal credentials and gain access to protected resources
- Local attacker – may spy on traffic
- Active attacker – may send messages
- In some cases, may be able to steal information from users



# (Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server

cbw



Auth Server

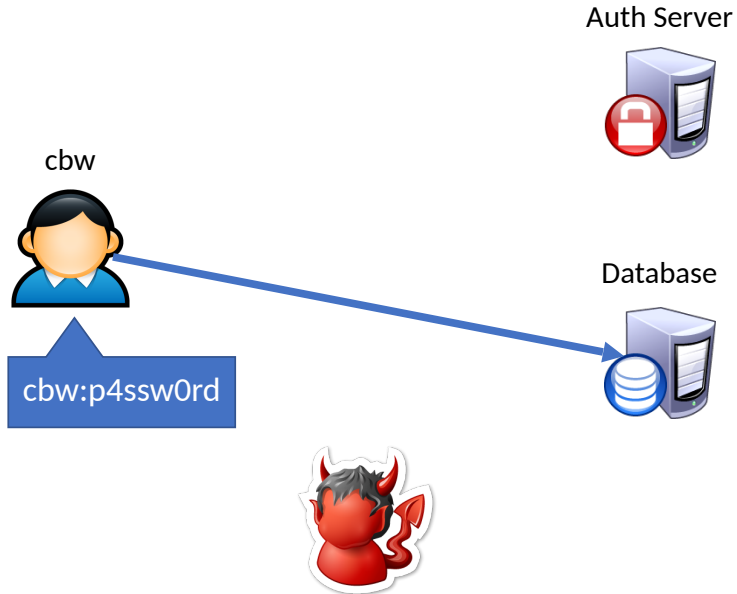


Database



# (Bad) Distributed Auth Example

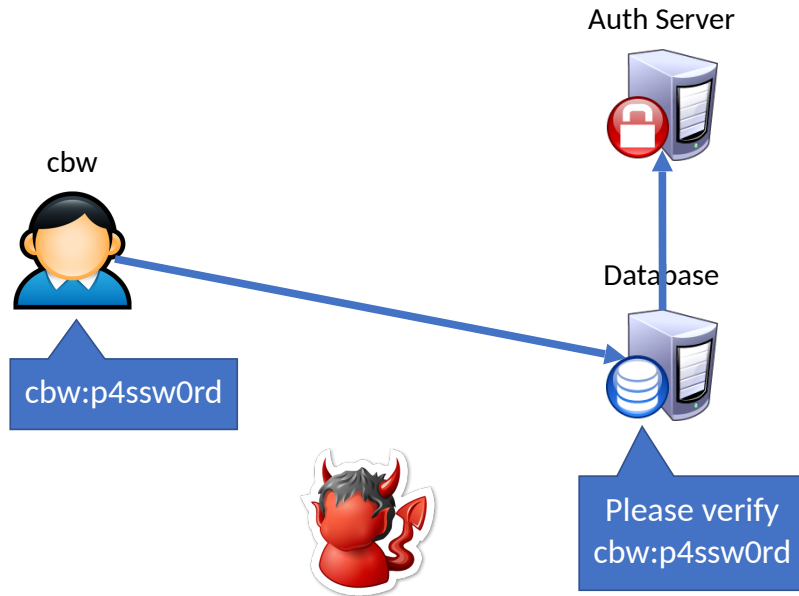
- Idea: client forwards user/password to service, service queries Auth Server





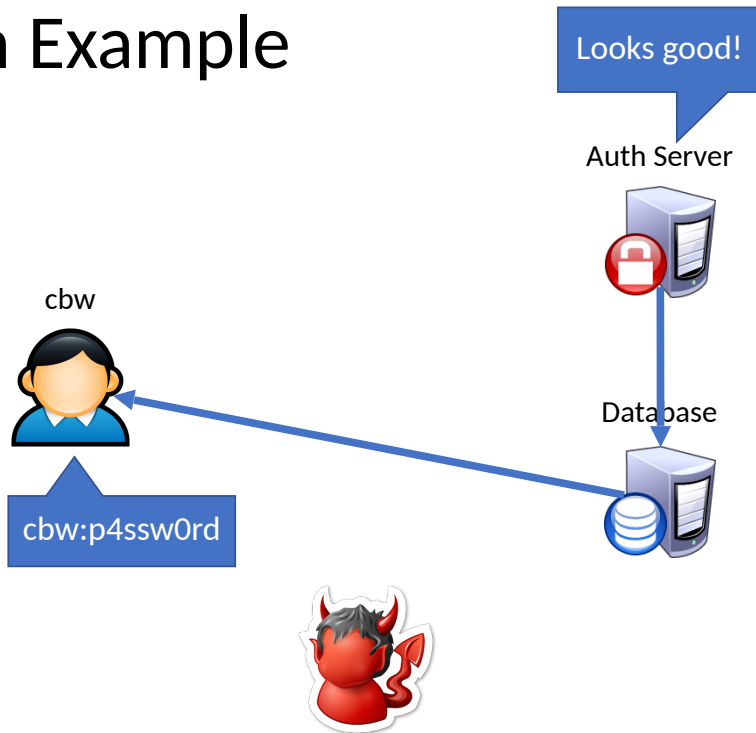
# (Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server



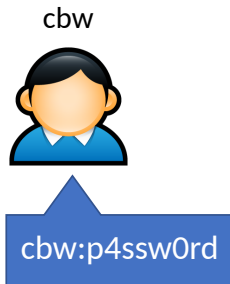
# (Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server



# (Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server
- Problems:



Looks good!

Auth Server

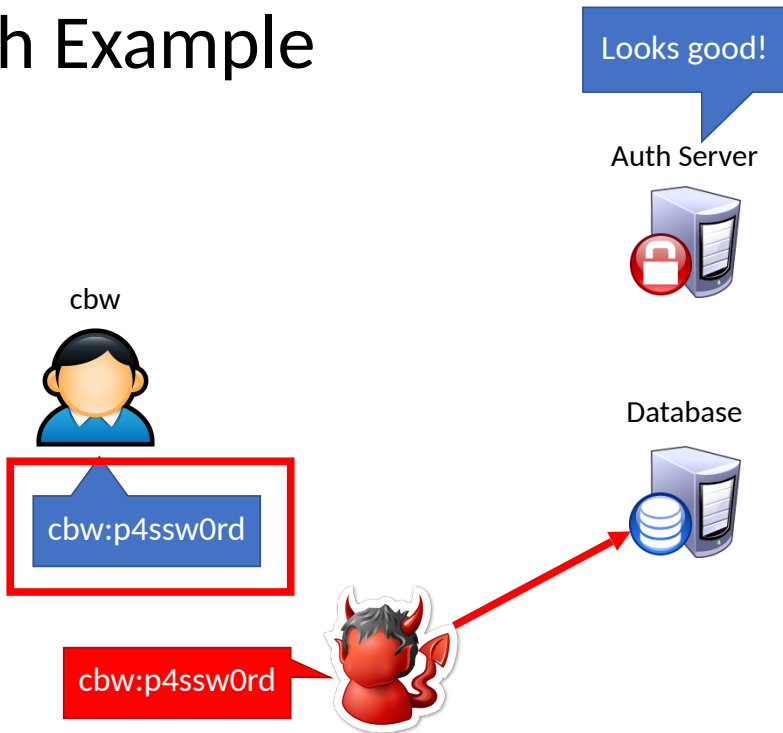


Database



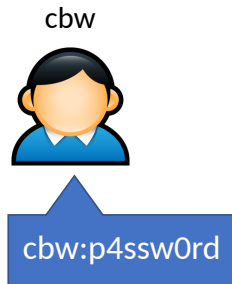
# (Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server
- Problems:
  - Passwords being sent in the clear
  - Attacker can observe them!
  - Clearly we need encryption



# (Bad) Distributed Auth Example

- Idea: client forwards user/password to service, service queries Auth Server
- Problems:
  - Passwords being sent in the clear
    - Attacker can observe them!
    - Clearly we need encryption
  - Database learns about passwords
    - Additional point of compromise
    - Ideally, only the user and the Auth Server should know their password

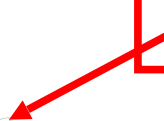
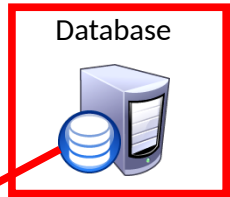


Looks good!

Auth Server



Database



# Needham-Schroeder Protocol

- Let Alice  $A$  and Bob  $B$  be two parties that trust server  $S$
- $K_{AS}$  and  $K_{BS}$  are shared secrets between  $[A, S]$  and  $[B, S]$
- $K_{AB}$  is a negotiated session key between  $[A, B]$
- $N_i$  and  $N_j$  are random **nonces** generated by  $A$  and  $B$

1)  $A \rightarrow S: A, B, N_i$

2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$

5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

# Needham-Schroeder Protocol

- Let Alice  $A$  and Bob  $B$  be two parties that trust server  $S$
- $K_{AS}$  and  $K_{BS}$  are shared secrets between  $[A, S]$  and  $[B, S]$
- $K_{AB}$  is a negotiated session key between  $[A, B]$
- $N_i$  and  $N_j$  are random **nonces** generated by  $A$  and  $B$

1)  $A \rightarrow S: A, B, N_i$

2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$

5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

Challenge nonce forces  $A$  to acknowledge they have  $K_{AB}$

# Needham-Schroeder Protocol

- Let Alice  $A$  and Bob  $B$  be two parties that trust server  $S$
- $K_{AS}$  and  $K_{BS}$  are shared secrets between  $[A, S]$  and  $[B, S]$
- $K_{AB}$  is a negotiated session key between  $[A, B]$
- $N_i$  and  $N_j$  are random **nonces** generated by  $A$  and  $B$

1)  $A \rightarrow S: A, B, N_i$

$K_{AS}$  is not sent in the clear, authenticates  $S$  and  $A$

2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$

5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

Challenge nonce forces  $A$  to acknowledge they have  $K_{AB}$



# Needham-Schroeder Protocol

- Let Alice  $A$  and Bob  $B$  be two parties that trust server  $S$
- $K_{AS}$  and  $K_{BS}$  are shared secrets between  $[A, S]$  and  $[B, S]$
- $K_{AB}$  is a negotiated session key between  $[A, B]$
- $N_i$  and  $N_j$  are random **nonces** generated by  $A$  and  $B$

1)  $A \rightarrow S: A, B, N_i$

$K_{AS}$  is not sent in the clear, authenticates  $S$  and  $A$

2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

$K_{BS}$  is not sent in the clear, authenticates  $B$

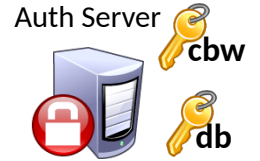
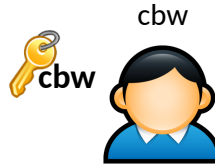
4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$

5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

Challenge nonce forces  $A$  to acknowledge they have  $K_{AB}$

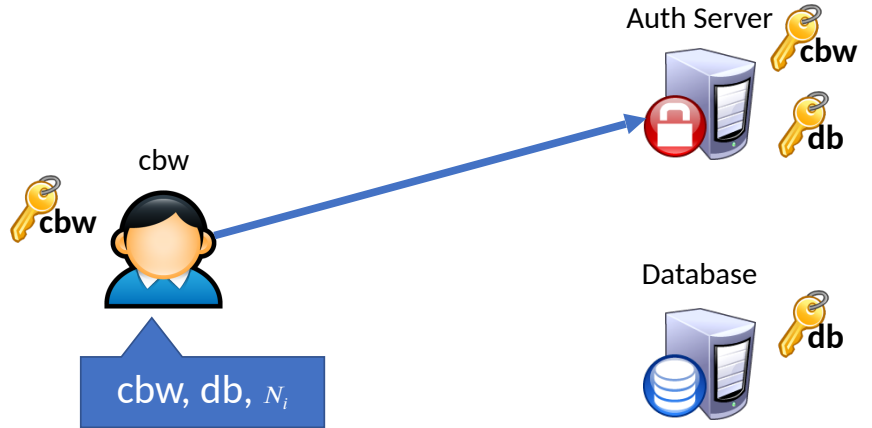
# Needham-Schroeder Example

1)  $A \rightarrow S: A, B, N_i$



# Needham-Schroeder Example

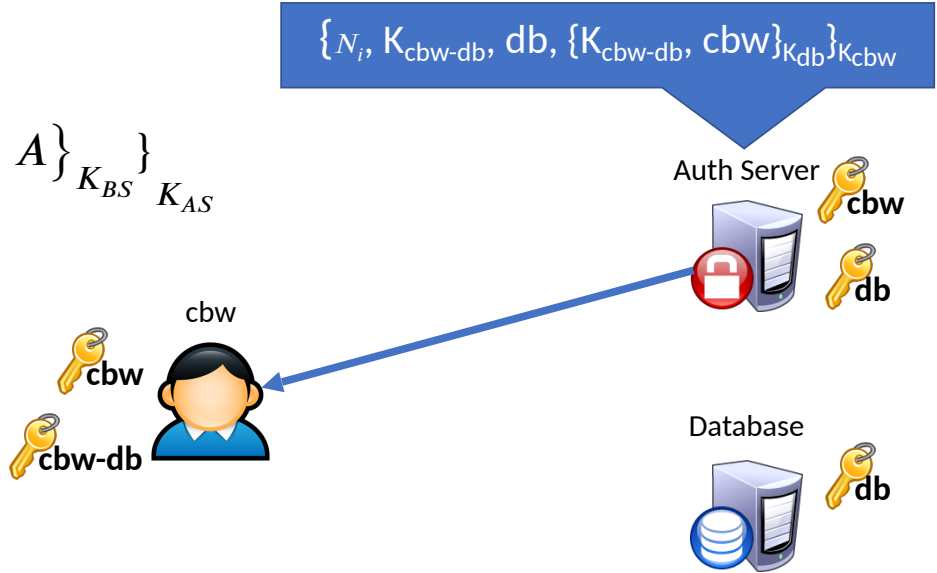
1)  $A \rightarrow S: A, B, N_i$



# Needham-Schroeder Example

1)  $A \rightarrow S: A, B, N_i$

2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

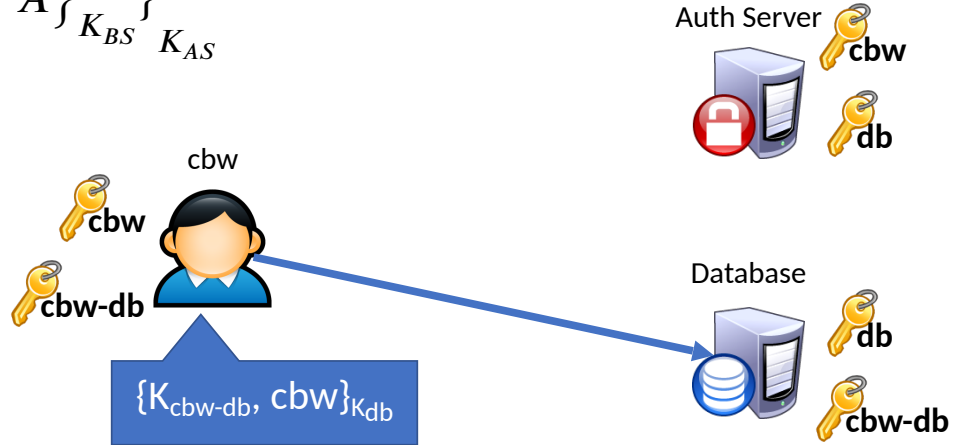


# Needham-Schroeder Example

1)  $A \rightarrow S: A, B, N_i$

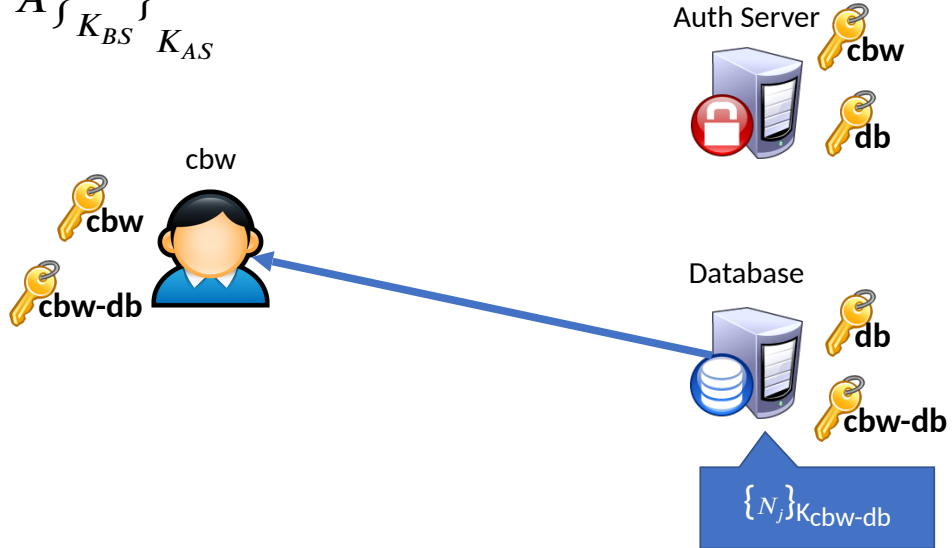
2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$



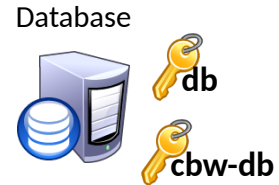
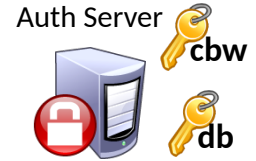
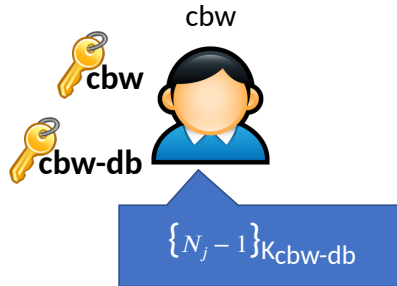
# Needham-Schroeder Example

- 1)  $A \rightarrow S: A, B, N_i$
- 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
- 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$

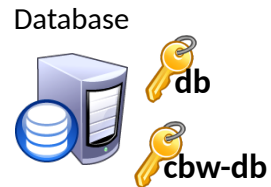
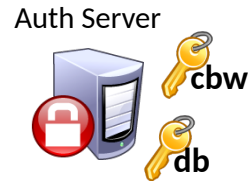
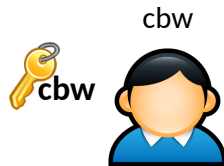


# Needham-Schroeder Example

- 1)  $A \rightarrow S: A, B, N_i$
- 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
- 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$
- 5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$



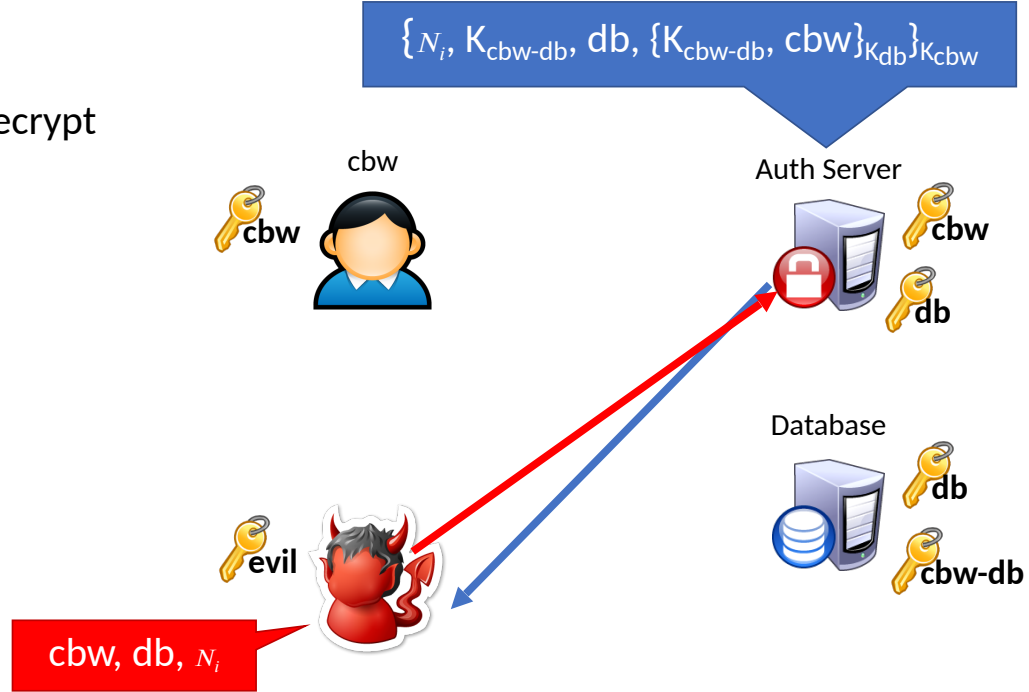
# Attacking Needham-Schroeder





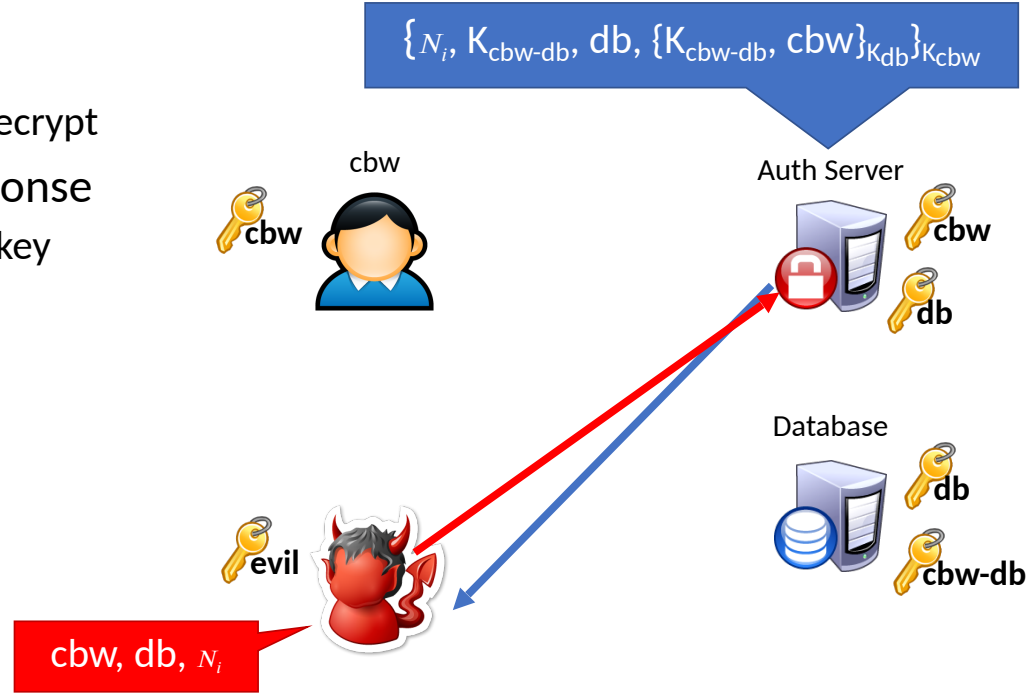
# Attacking Needham-Schroeder

- Spoof the client request
- Fail! Client key is needed to decrypt



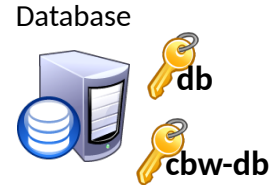
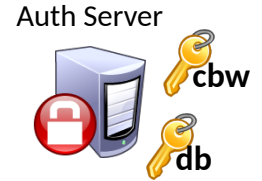
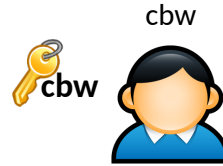
# Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key



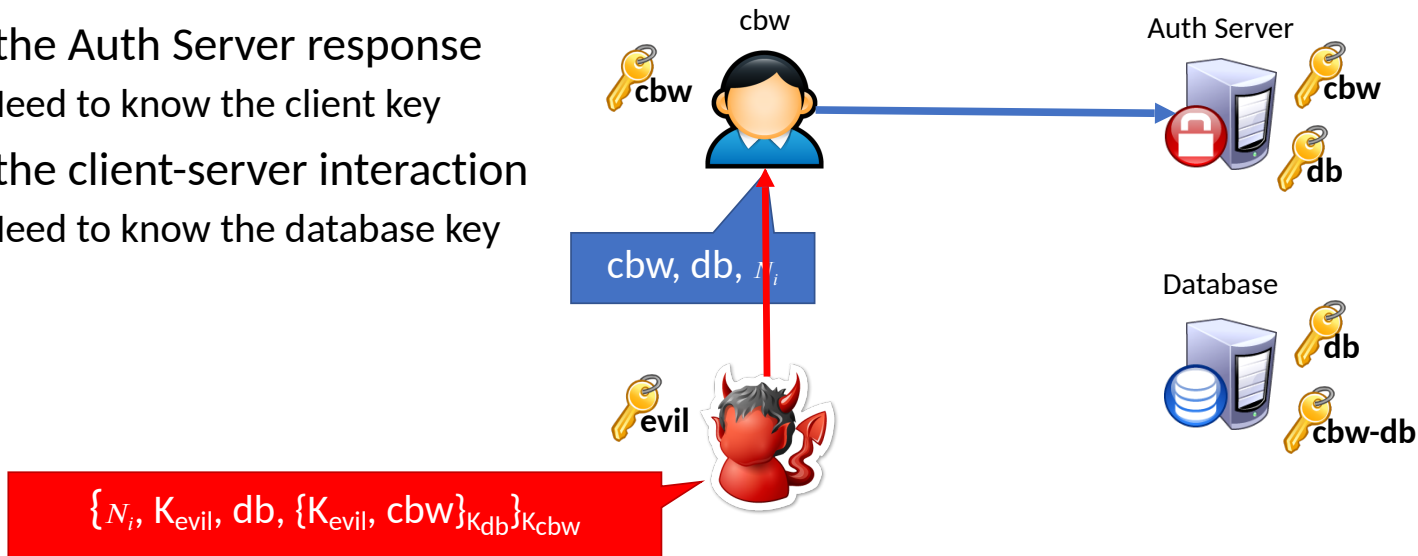
# Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key



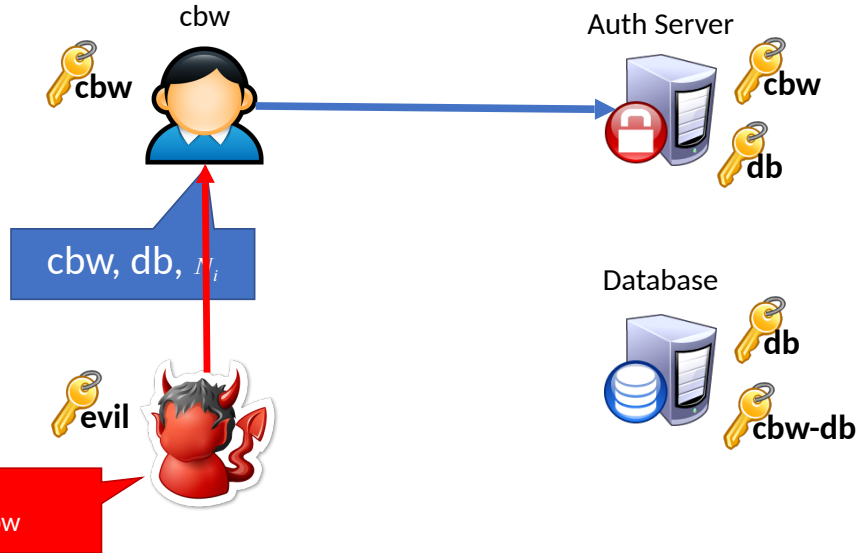
# Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key



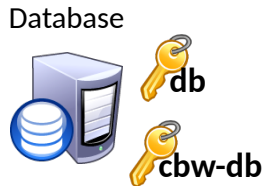
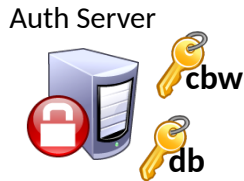
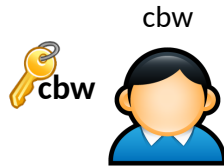
# Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key
- Replay the client-server interaction
  - Fail! Need to know the session key



# Attacking Needham-Schroeder

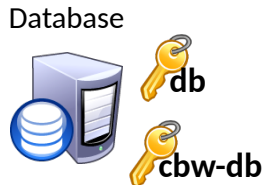
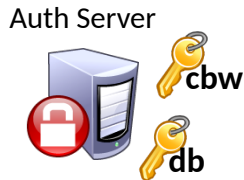
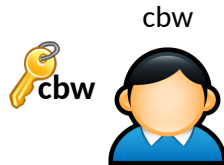
- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key
- Replay the client-server interaction
  - Fail! Need to know the session key



$\{K_{\text{evil}}, \text{cbw}\}_{K_{\text{db}}}$

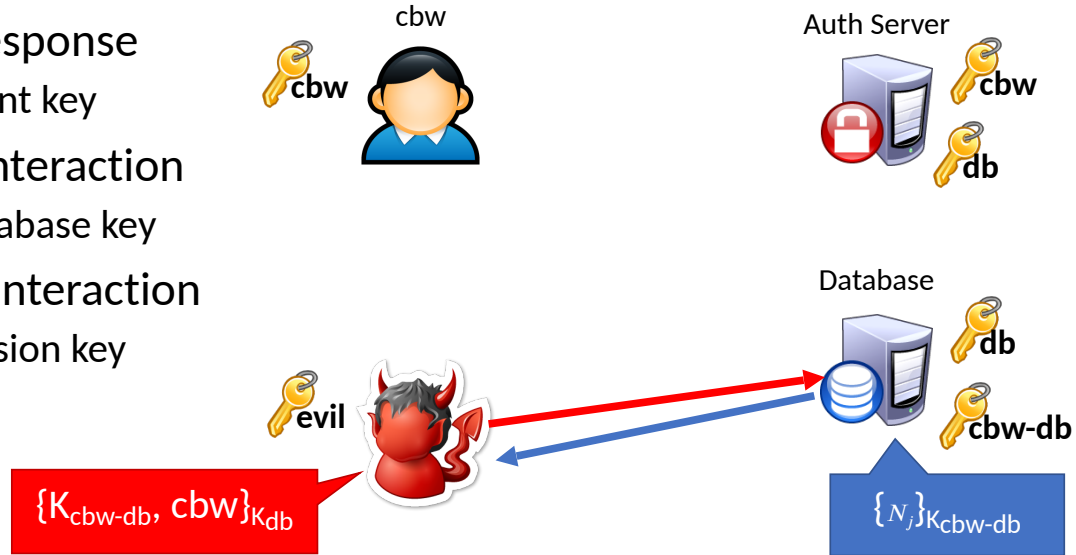
# Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key
- Replay the client-server interaction
  - Fail! Need to know the session key



# Attacking Needham-Schroeder

- Spoof the client request
  - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
  - Fail! Need to know the client key
- Spoof the client-server interaction
  - Fail! Need to know the database key
- Replay the client-server interaction
  - Fail! Need to know the session key





# Replay Attack

## Typical, Benign Protocol

- 1)  $A \rightarrow S: A, B, N_i$
- 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
- 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$
- 5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

# Replay Attack

## Typical, Benign Protocol

- 1)  $A \rightarrow S: A, B, N_i$
- 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}\}_{K_{BS}} \quad K_{AS}$
- 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$
- 5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

## Replay Attack

- 1)  $M \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 2)  $B \rightarrow M: \{N_j\}_{K_{AB}}$
- 3)  $M \rightarrow B: \{N_j - 1\}_{K_{AB}}$

# Replay Attack

## Typical, Benign Protocol

- 1)  $A \rightarrow S: A, B, N_i$
  - 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}\}_{K_{BS}} K_{AS}$
  - 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
  - 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$
  - 5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$
- Attacker must hack A to steal  $K_{AB}$ 
    - So the attacker can also steal  $K_{AS}$

## Replay Attack

- 1)  $M \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 2)  $B \rightarrow M: \{N_j\}_{K_{AB}}$
- 3)  $M \rightarrow B: \{N_j - 1\}_{K_{AB}}$

# Replay Attack

## Typical, Benign Protocol

- 1)  $A \rightarrow S: A, B, N_i$
- 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}\}_{K_{BS}} K_{AS}$
- 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$
- 5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

- Attacker must hack A to steal  $K_{AB}$ 
  - So the attacker can also steal  $K_{AS}$
- However, what happens after A changes  $K_{AS}$

## Replay Attack

- 1)  $M \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 2)  $B \rightarrow M: \{N_j\}_{K_{AB}}$
- 3)  $M \rightarrow B: \{N_j - 1\}_{K_{AB}}$

# Replay Attack

## Typical, Benign Protocol

- 1)  $A \rightarrow S: A, B, N_i$
- 2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
- 3)  $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$
- 5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

- Attacker must hack A to steal  $K_{AB}$ 
  - So the attacker can also steal  $K_{AS}$
- However, what happens after A changes  $K_{AS}$ 
  - Attacker can still conduct the replay attack! Only is  $K_{AB}$  necessary!

## Replay Attack

- 1)  $M \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$
- 2)  $B \rightarrow M: \{N_j\}_{K_{AB}}$
- 3)  $M \rightarrow B: \{N_j - 1\}_{K_{AB}}$

# Fixed Needham-Schroeder Protocol

- Let Alice  $A$  and Bob  $B$  be two parties that trust server  $S$
- $K_{AS}$  and  $K_{BS}$  are shared secrets between  $[A, S]$  and  $[B, S]$
- $K_{AB}$  is a negotiated session key between  $[A, B]$
- $N_i$  and  $N_j$  are random nonces generated by  $A$  and  $B$
- $T$  is a timestamp chosen by  $S$

1)  $A \rightarrow S: A, B, N_i$

2)  $S \rightarrow A: \{N_i, K_{AB}, B, \{K_{AB}, A, T\}_{K_{BS}}\}_{K_{AS}}$

3)  $A \rightarrow B: \{K_{AB}, A, T\}_{K_{BS}}$

4)  $B \rightarrow A: \{N_j\}_{K_{AB}}$

5)  $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

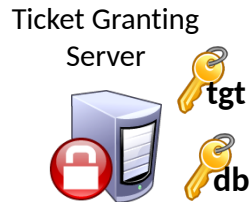
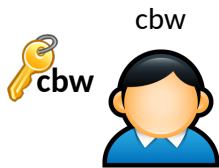


B only accepts requests  
with fresh timestamps

# Kerberos

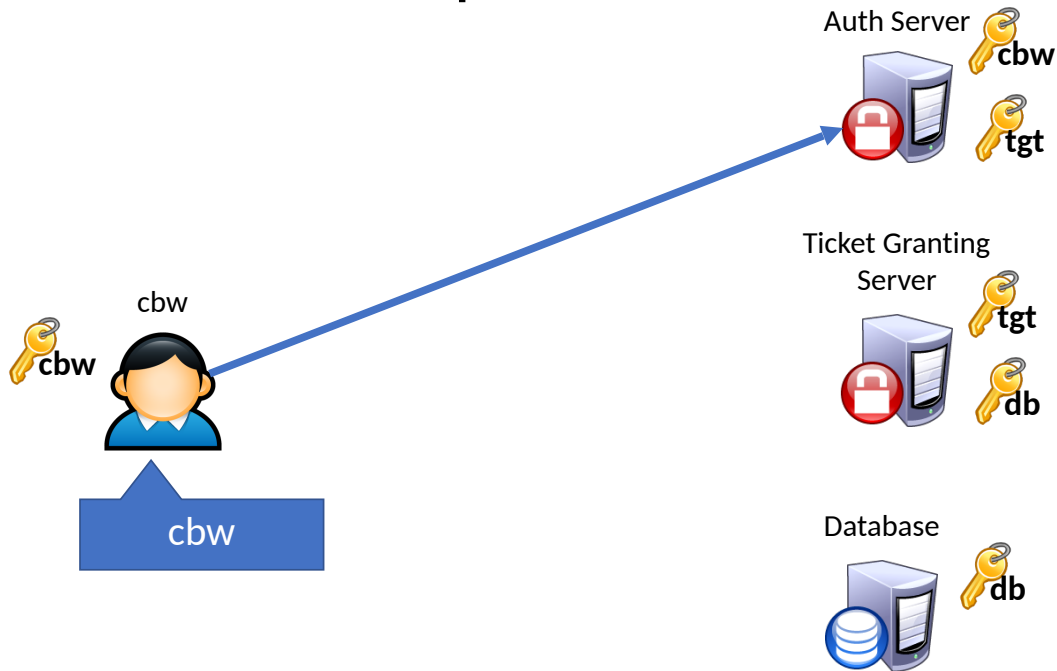
- Created as part of MIT Project Athena
  - Based on Needham-Schroeder
- Provides mutual authentication over untrusted networks
  - **Tickets** as assertions of authenticity, authorization
  - Forms basis of Active Directory authentication
- Principals
  - Client
  - Server
  - Key distribution center (KDC)
    - Authentication server (AS)
    - Ticket granting server (TGS)

# Kerberos Example





# Kerberos Example



# Kerberos Example

$\{cbw, K_{cbw-tgs}\}_{K_{cbw}}, TGT$

Auth Server



 **cbw**

 **tgt**

Ticket Granting Server



 **tgt**

 **db**

Database



 **db**

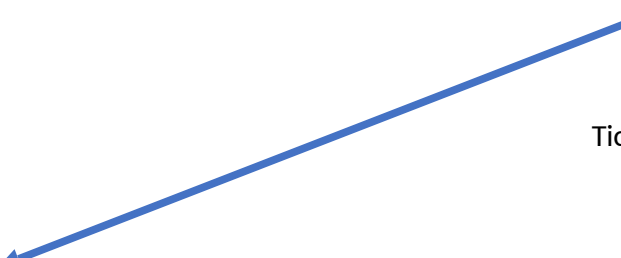
cbw



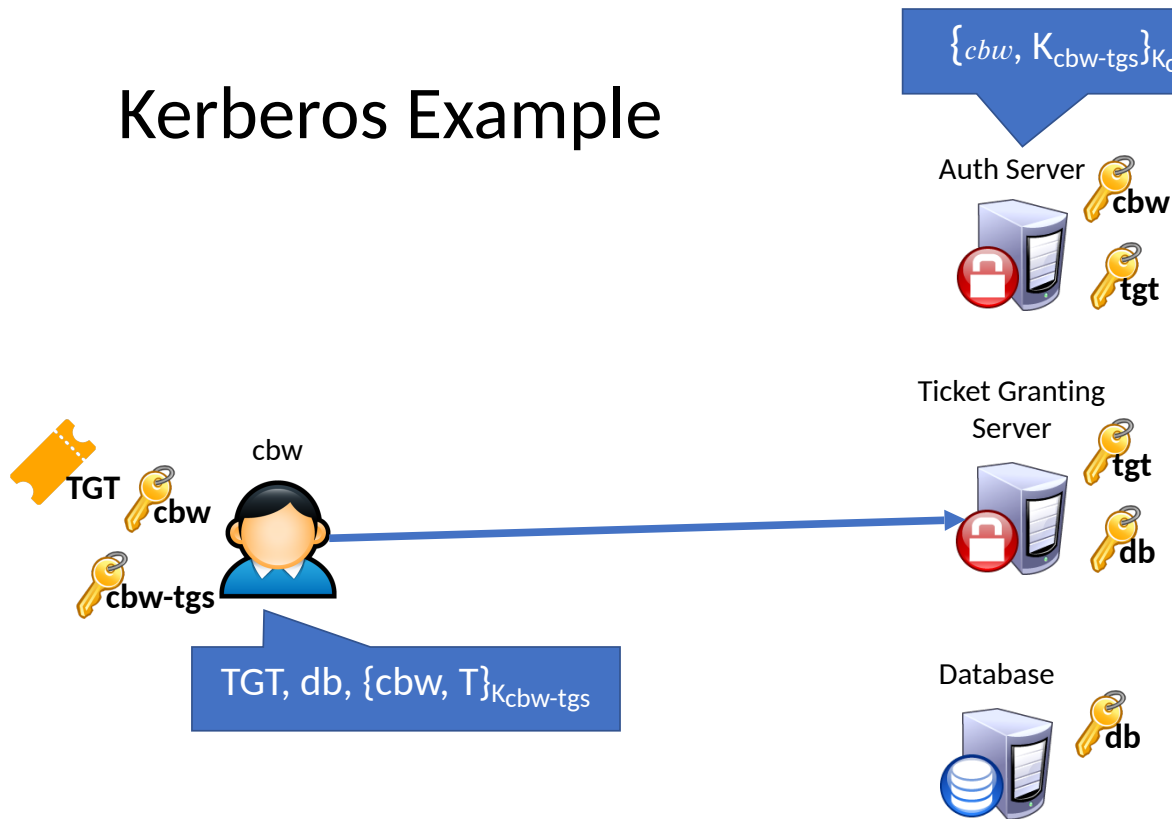
 **TGT**

 **cbw**

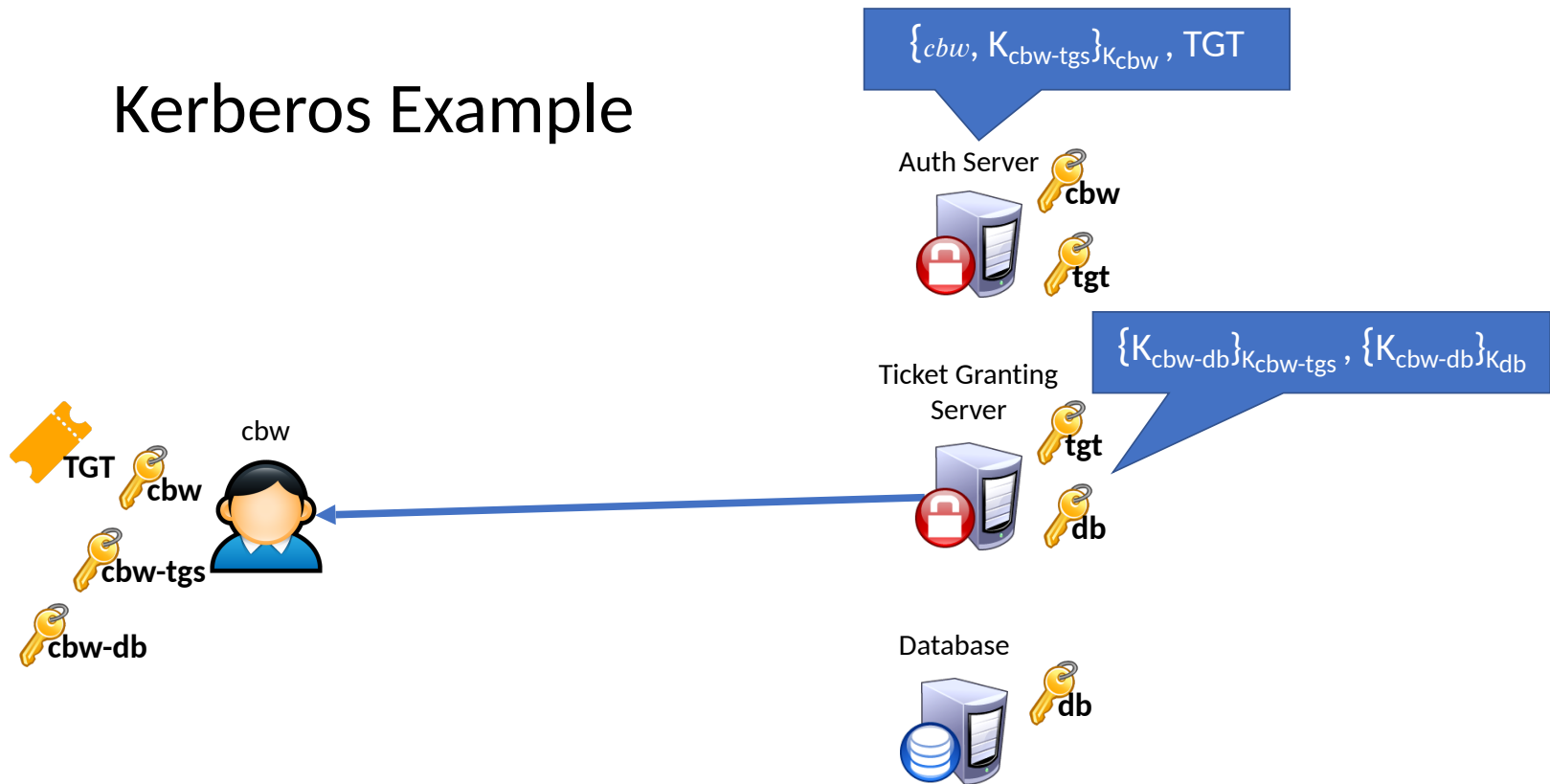
 **cbw-tgs**



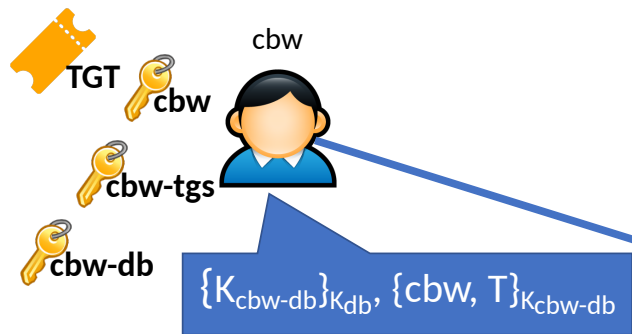
# Kerberos Example



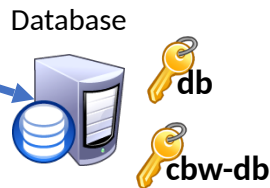
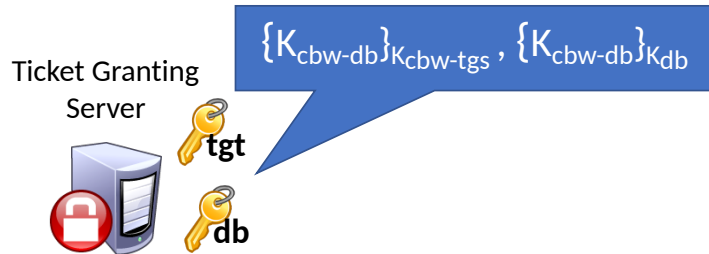
# Kerberos Example



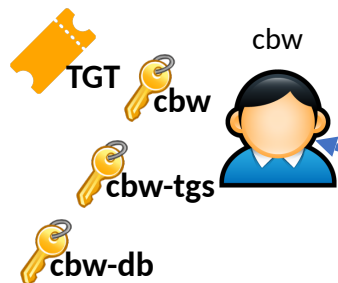
# Kerberos Example



$\{cbw, K_{cbw-tgs}\}_{K_{cbw}}, TGT$



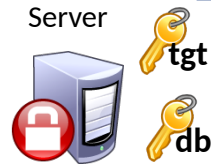
# Kerberos Example



$\{cbw, K_{cbw-tgs}\}_{K_{cbw}}, TGT$



Ticket Granting Server

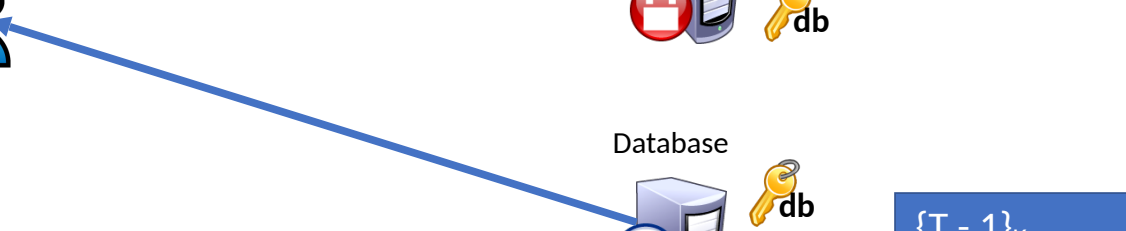


$\{K_{cbw-db}\}_{K_{cbw-tgs}}, \{K_{cbw-db}\}_{K_{db}}$

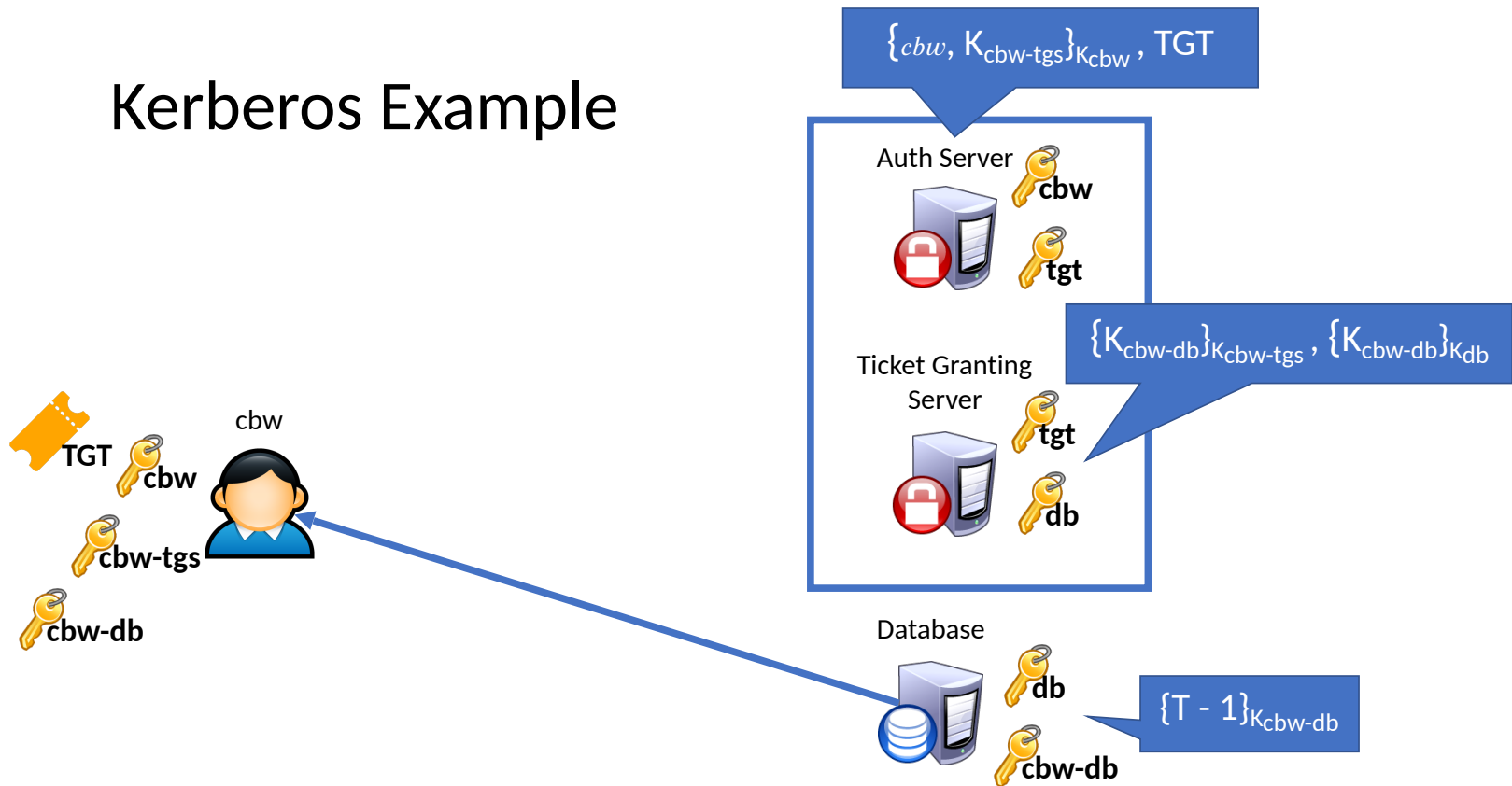
Database



$\{T - 1\}_{K_{cbw-db}}$



# Kerberos Example



# Attacking Kerberos

- Don't put all your eggs in one basket
  - The Kerberos Key Distribution Server (KDS) is a central point of failure
  - DoS the KDS and the network ceases to function
  - Compromise the KDS leads to network-wide compromise



# Attacking Kerberos

- Don't put all your eggs in one basket
  - The Kerberos Key Distribution Server (KDS) is a central point of failure
  - DoS the KDS and the network ceases to function
  - Compromise the KDS leads to network-wide compromise
- Time synchronization
  - Inaccurate clocks lead to protocol failures (due to timestamps)
  - Solution?

# Attacking Kerberos

- Don't put all your eggs in one basket
  - The Kerberos Key Distribution Server (KDS) is a central point of failure
  - DoS the KDS and the network ceases to function
  - Compromise the KDS leads to network-wide compromise
- Time synchronization
  - Inaccurate clocks lead to protocol failures (due to timestamps)
  - Solution?
  - Use NTP ;)

# Sources

1. Many slides courtesy of Wil Robertson: <https://wkr.io>
  2. Honeywords, Ari Juels and Ron Rivest: <http://www.arijuels.com/wp-content/uploads/2013/09/JR13.pdf>
- For more on generating secure passwords, and understanding people's mental models of passwords, see the excellent work of Blas Ur: <http://www.blaseur.com/pubs.htm>