

2550 Intro to cybersecurity

L10: Passwords

abhi shelat

Thanks Christo for slides!



abhi



Enter Password

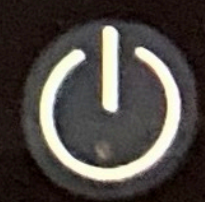
Touch ID or Enter Password



Sleep

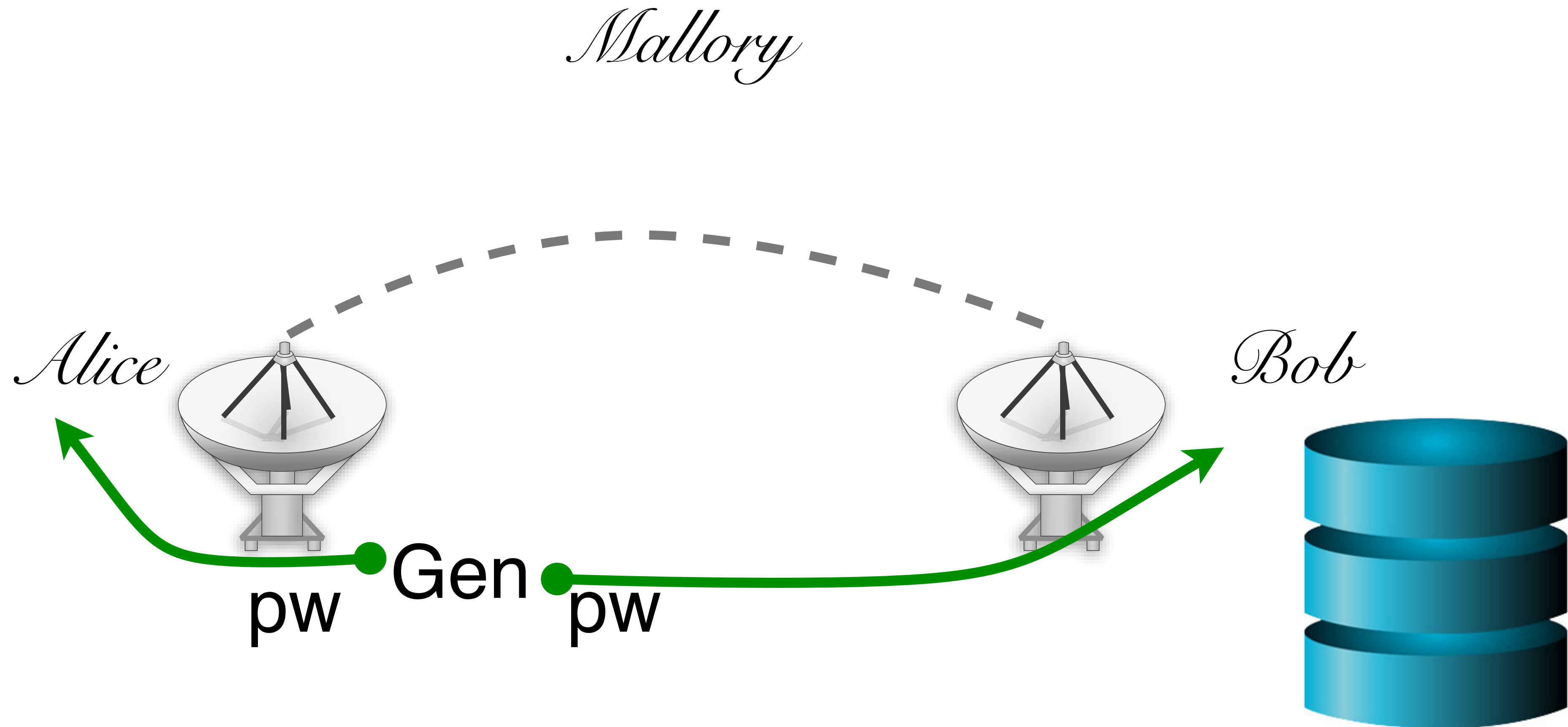


Restart



Shut Down

Password Security game



Operating
Systems

R. Stockton Gaines
Editor

Password Security: A Case History

Robert Morris and Ken Thompson
Bell Laboratories

This paper describes the history of the design of the password security scheme on a remotely accessed time-sharing system. The present design was the result of countering observed attempts to penetrate the system. The result is a compromise between extreme security and ease of use.

Key Words and Phrases: operating systems, passwords, computer security

CR Categories: 2.41, 4.35

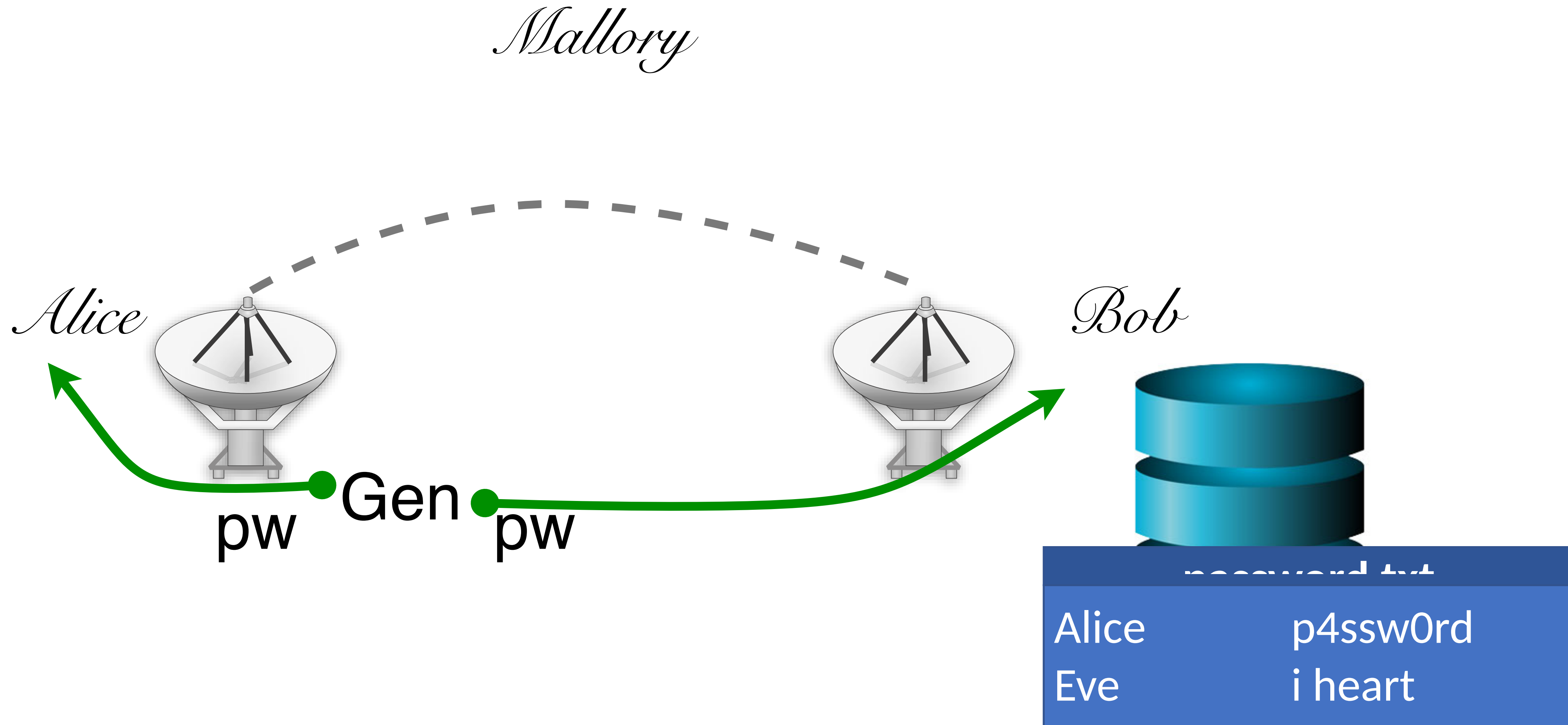
"The UNIX system was first implemented with a password file that contained the actual passwords of all the users, and for that reason the password file had to be heavily protected against being either read or written. Although historically, this had been the technique used for remote-access systems, it was completely unsatisfactory for several reasons."

Checking Passwords

- System must validate passwords provided by users
- Thus, passwords must be stored somewhere
- Basic storage: plain text

password.txt	
Alice	p4ssw0rd
Eve	i heart doggies
Charlie	93Gd9#jv*0x3N
bob	security

Password Security game Attack



RockYou Hack: From Bad To Worse

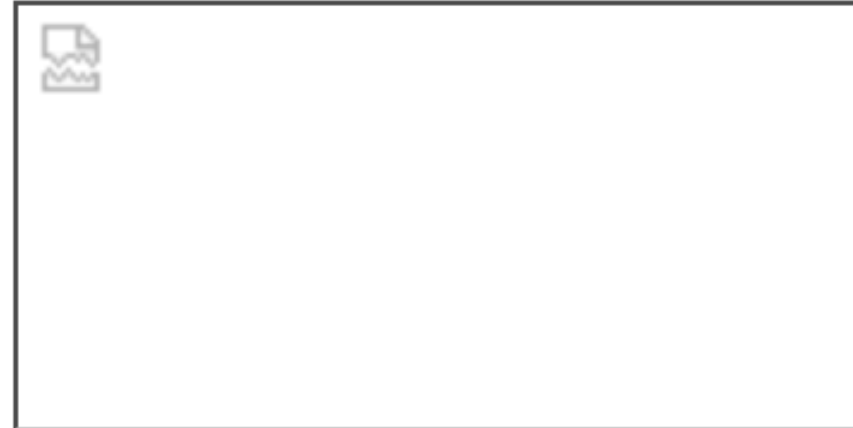


Nik Cubrilovic




@nikcub / 2:42 am EST • December 15, 2009

Comment



[Earlier today news spread](#) that social application site [RockYou](#) had suffered a data breach that

resulted in the exposure of over 32 Million user accounts. To compound the severity of the security breach, it was found that **RockYou**  are storing all user account data in plain text in their database, exposing all that information to attackers. RockYou have yet to inform users of the breach, and their blog is eerily silent – but the details of the security breach are going from bad to worse.

Hashed Passwords

- **Key idea: store “hashed” versions of passwords**
 - Use one-way cryptographic hash functions
 - Examples: MD5, SHA1, SHA256, SHA512, bcrypt, PBKDF2, scrypt
- **Cryptographic hash function transform input data into scrambled output data**
 - Deterministic: $\text{hash}(A) = \text{hash}(A)$
 - High entropy:
 - MD5('security') = e91e6348157868de9dd8b25c81aebfb9
 - MD5('security1') = 8632c375e9eba096df51844a5a43ae93
 - MD5('Security') = 2fae32629d4ef4fc6341f1751b405e45
 - Collision resistant
 - Locating A' such that $\text{hash}(A) = \text{hash}(A')$ takes a long time (hopefully)
 - Example: 2^{21} tries for md5

Hashed Password Example



User: Charlie

hashed_password.txt

charlie	2a9d119df47ff993b662a8ef36f9ea20
greta	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Hashed Password Example



User: Charlie



MD5('p4ssw0rd') =
2a9d119df47ff993b662a8ef36f9ea20



hashed_password.txt	
charlie	2a9d119df47ff993b662a8ef36f9ea20
greta	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Hashed Password Example



User: Charlie



MD5('p4ssw0rd') =
2a9d119df47ff993b662a8ef36f9ea20

hashed_password.txt

charlie	2a9d119df47ff993b662a8ef36f9ea20
greta	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Hashed Password Example



User: Charlie

MD5('p4ssw0rd') =

2a9d119df47ff993b662a8ef36f9ea20



MD5('2a9d119df47ff993b662a8ef36f9ea20')

= b35596ed3f0d5134739292faa04f7ca3

hashed_password.txt

charlie	2a9d119df47ff993b662a8ef36f9ea20
greta	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Hashed Password Example



User: Charlie

MD5('p4ssw0rd') =

2a9d119df47ff993b662a8ef36f9ea20



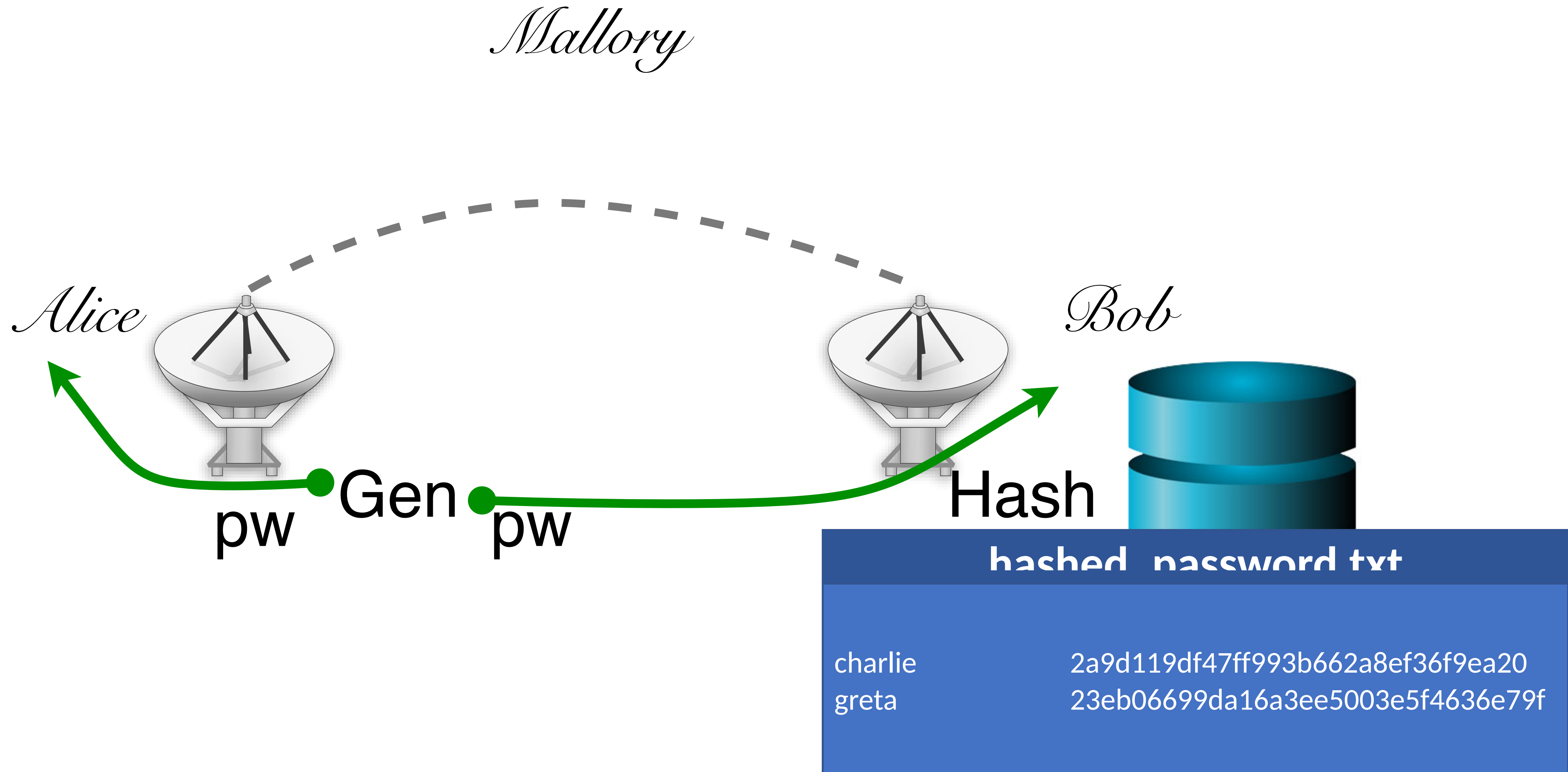
MD5('2a9d119df47ff993b662a8ef36f9ea20')

= b35596ed3f0d5134739292faa04f7ca3



hashed_password.txt	
charlie	2a9d119df47ff993b662a8ef36f9ea20
greta	23eb06699da16a3ee5003e5f4636e79f
alice	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

Password Security game



Attacking Password Hashes

- Recall: cryptographic hashes are collision resistant
 - Locating A' such that $\text{hash}(A) = \text{hash}(A')$ takes a long time (hopefully)
- Are hashed password secure from cracking?
 - Attack 2:

Attacking Password Hashes

- Recall: cryptographic hashes are collision resistant
 - Locating A' such that $\text{hash}(A) = \text{hash}(A')$ takes a long time (hopefully)
- Are hashed password secure from cracking?
- Attack 1: • Attack 2:

The authors have conducted experiments to try to determine typical users' habits in the choice of passwords when no constraint is put on their choice. The results were disappointing, except to the bad guy. In a collection of 3,289 passwords gathered from many users over a long period of time,

15 were a single ASCII character;

72 were strings of two ASCII characters;

464 were strings of three ASCII characters;

477 were strings of four alphanumerics;

706 were five letters, all upper-case or all lower-case;

605 were six letters, all lower-case.

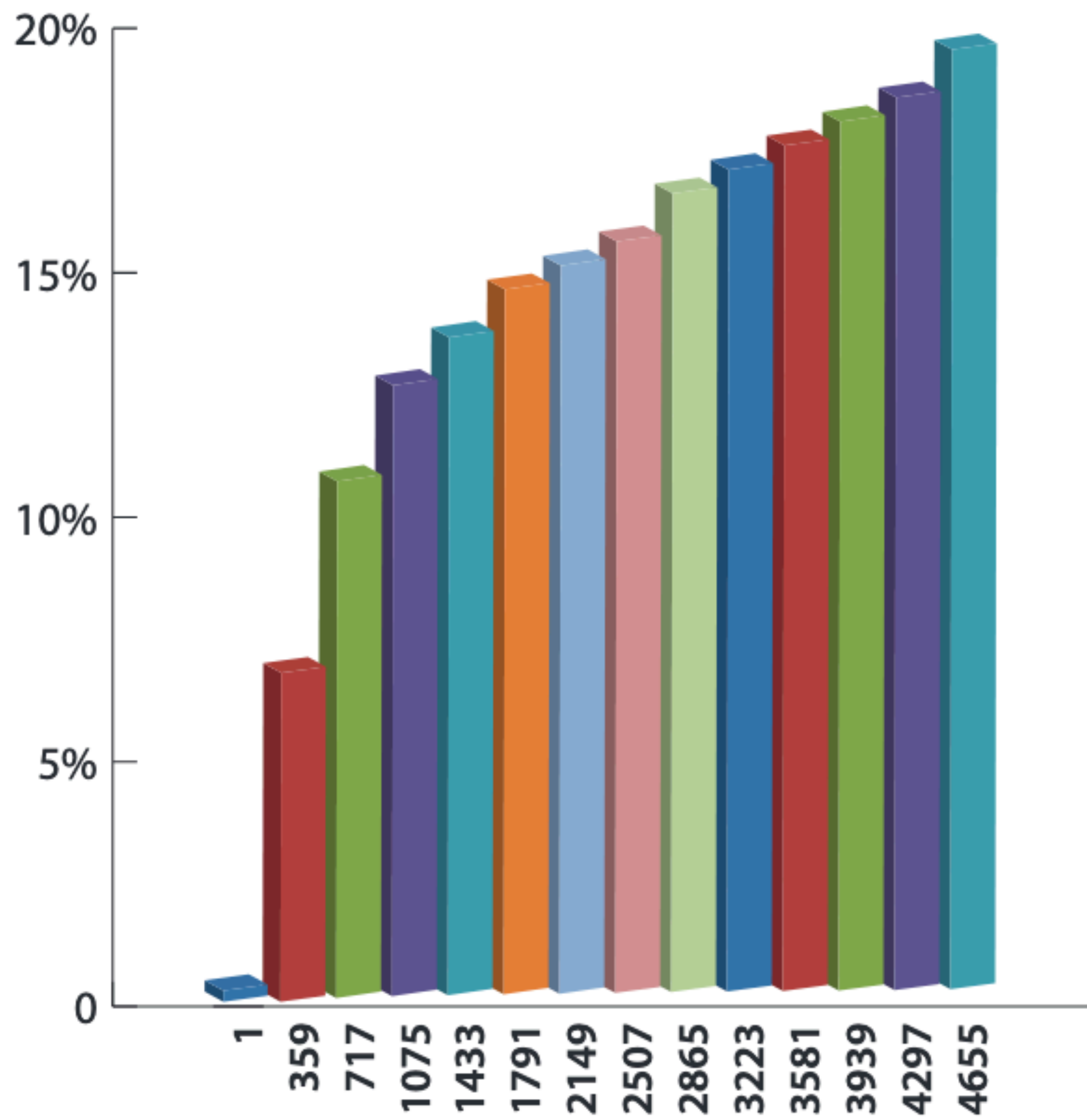
An additional 492 passwords appeared in various available dictionaries, name lists, and the like. A total of 2,831 or 86 percent of this sample of passwords fell into one of these classes.

From Rockyou breach

Rank	Password	Number of Users with Password (Absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Password Popularity—Top 20

Rank	Password	Number of Users with Password (Absolute)
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856

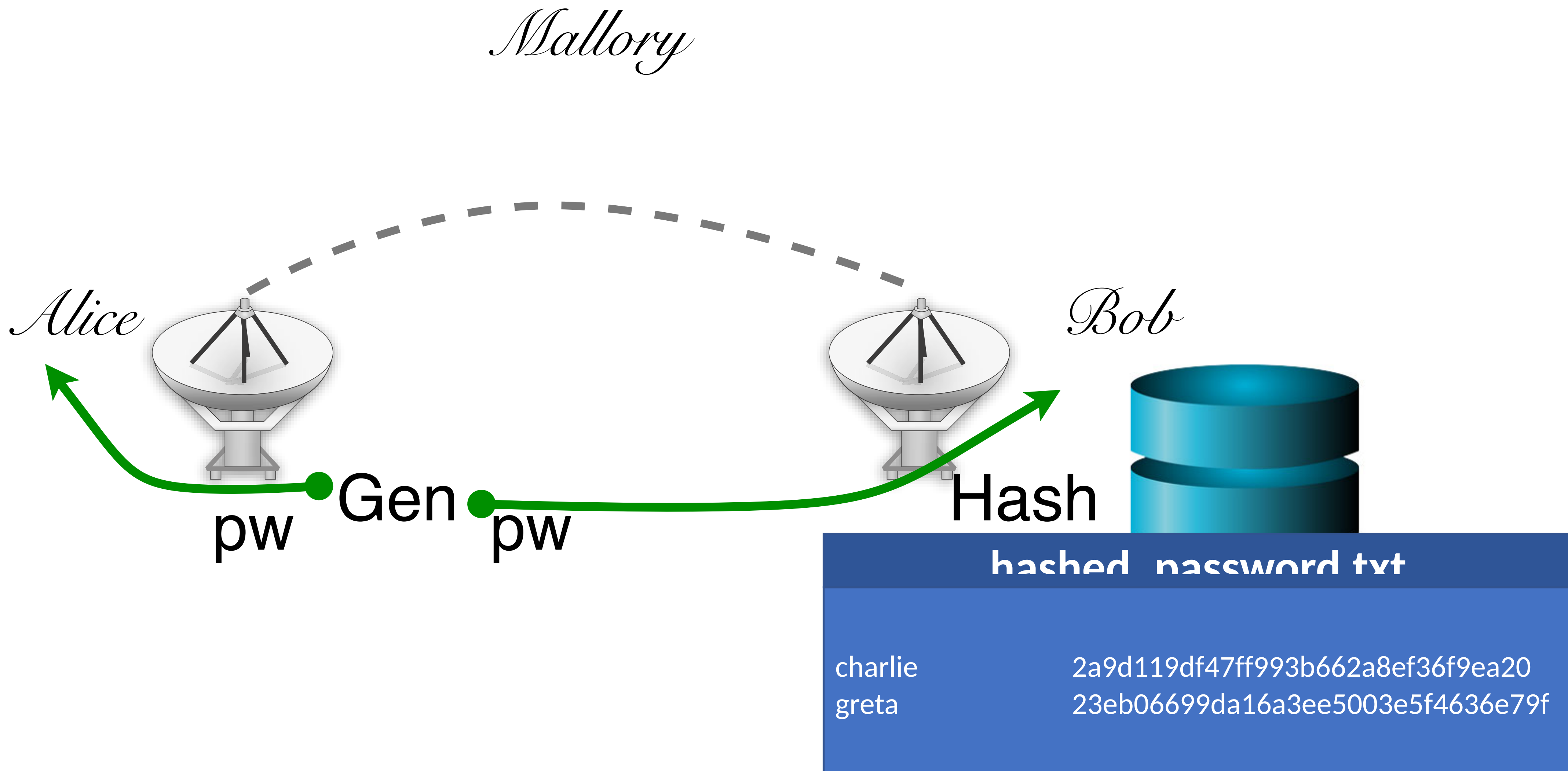


Accumulated Percent of Dictionary Attack Success

Most Common Passwords

Rank	2013	2014
1	123456	123456
2	password	password
3	12345678	12345
4	qwerty	12345678
5	abc123	qwerty
6	123456789	123456789
7	111111	1234
8	1234567	baseball
9	iloveyou	dragon
10	adobe123	football

Attack 1



Dictionary Attacks

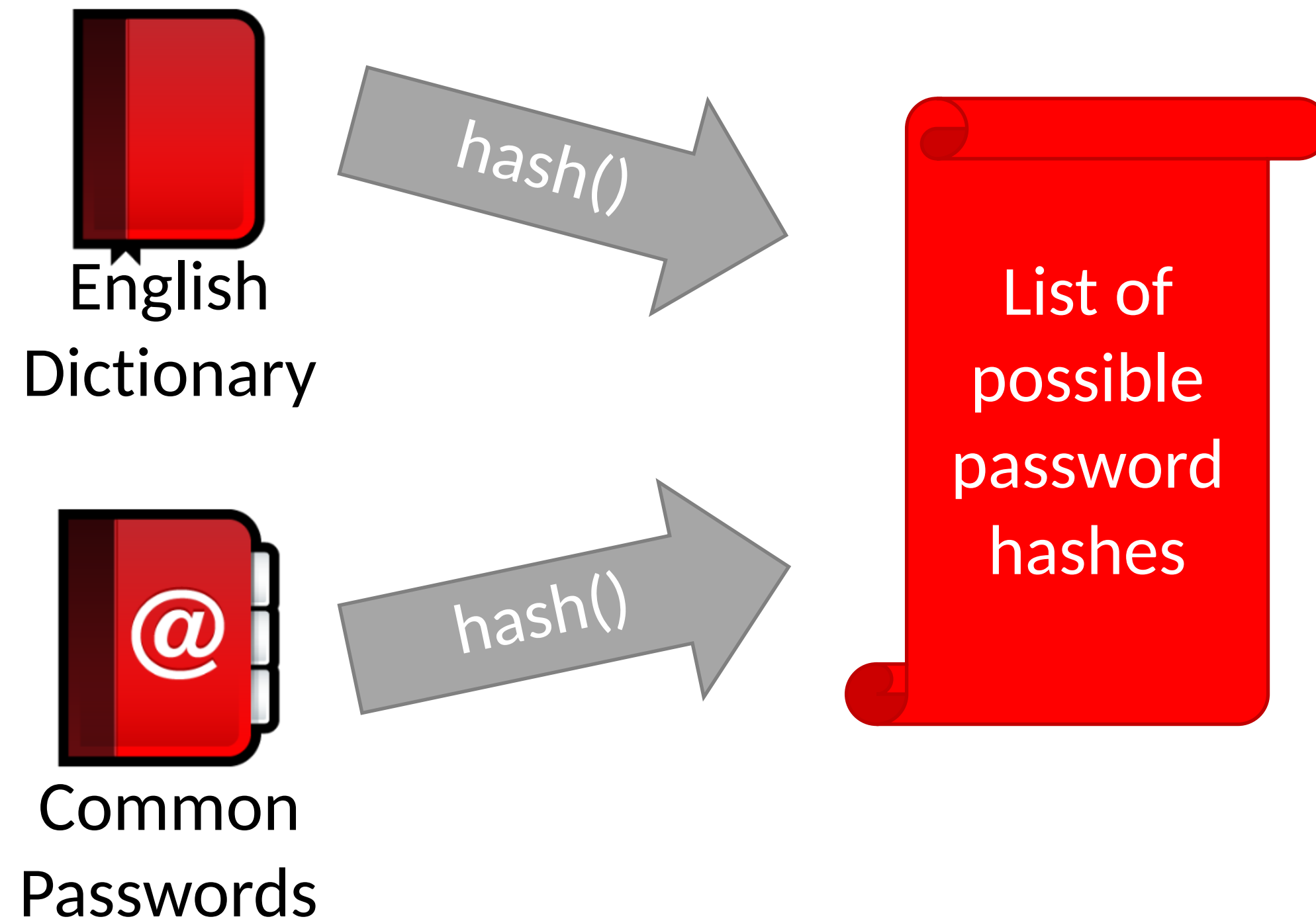


English
Dictionary

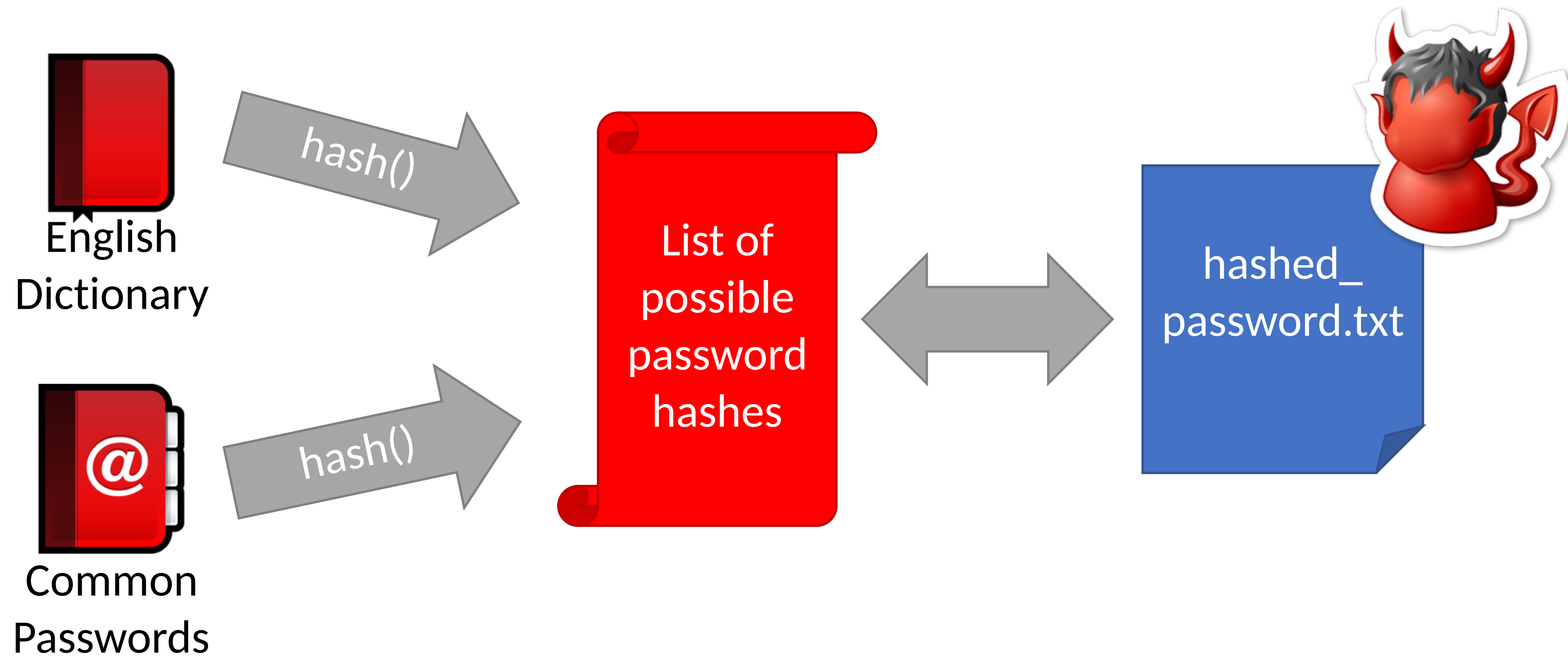


Common
Passwords

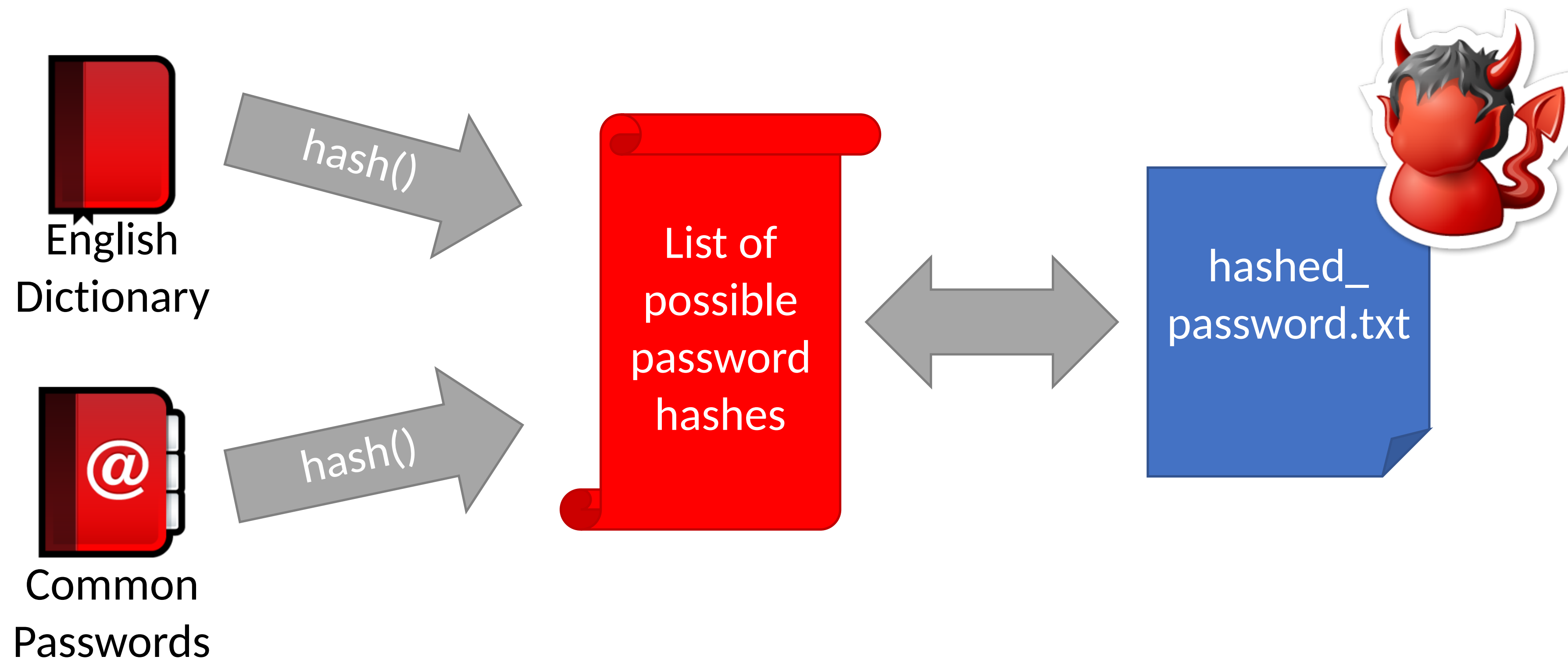
Dictionary Attacks



Dictionary Attacks



Dictionary Attacks



- Common for 60-70% of hashed passwords to be cracked in <24 hours

Brute force attack estimates

How big is the alphabet from which pwd are chosen?

How long is a password?

Size of password domain:

Brute force attack estimates

Size of password domain: 95^8 6,634,204,312,890,625

Built by [Edward Betts](#). Comments welcome: edward@4angle.com

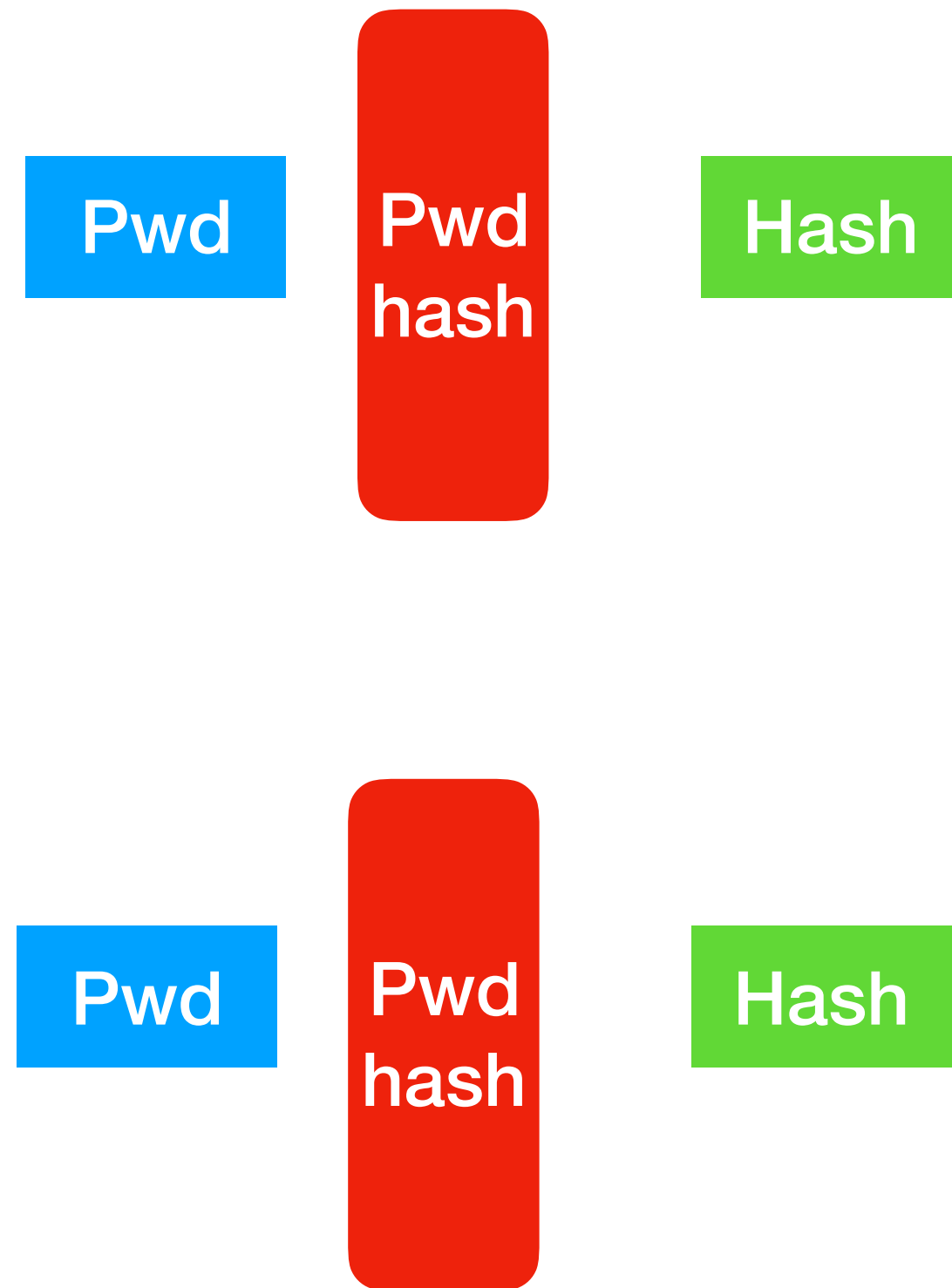
Last updated: 07 February 2020.

3.5" internal drives

Price per TB	Price	Size	Drive
\$18.75	\$149.99	8TB	Seagate BarraCuda ST8000DM004 8TB 5400 RPM 256MB Cache SATA 6.0Gb/s 3.5" Internal Hard Drive Bare Drive
\$22.25	\$88.99	4TB	WD Blue 4TB Desktop Hard Disk Drive - 5400 RPM SATA 6Gb/s 64MB Cache 3.5 Inch - WD40EZRZ
\$22.50	\$89.99	4TB	Seagate BarraCuda ST4000DM004 4TB 5400 RPM 256MB Cache SATA 6.0Gb/s 3.5" Hard Drives Bare Drive - OEM
\$22.52	\$135.12	6TB	Seagate BarraCuda ST6000DM003 6TB 5400 RPM 256MB Cache SATA 6.0Gb/s 3.5" Internal Hard Drive Bare Drive
\$23.33	\$139.99	6TB	WD Blue 6TB Desktop Hard Disk Drive - 5400 RPM SATA 6Gb/s 256MB Cache 3.5 Inch - WD60EZZZ
\$23.92	\$334.88	14TB	Seagate Exos X16 ST14000NM001G 14TB 7200 RPM 256MB Cache SATA 6.0Gb/s 3.5" Hard Drives, 512E/4KN
\$24.00	\$71.99	3TB	WD Blue 3TB Desktop Hard Disk Drive - 5400 RPM SATA 6Gb/s 64MB Cache 3.5 Inch - WD30EZRZ
\$24.06	\$384.99	16TB	Seagate Exos 16TB Enterprise HDD X16 SATA 6Gb/s 512e/4Kn 7200 RPM 256MB Cache 3.5" Internal Hard Drive ST16000NM001G
\$24.42	\$292.99	12TB	Seagate 12TB HDD Exos X14 7200 RPM 512e/4Kn SATA 6Gb/s 256MB Cache 3.5-Inch Enterprise Hard Drive (ST12000NM0008)
\$24.66	\$73.99	3TB	Seagate BarraCuda ST3000DM007 3TB 5400 RPM 256MB Cache SATA 6.0Gb/s 3.5" Hard Drives
\$25.00	\$49.99	2TB	Seagate BarraCuda ST2000DM008 2TB 7200 RPM 256MB Cache SATA 6.0Gb/s 3.5" Hard Drive Bare Drive
\$25.00	\$99.99	4TB	Seagate IronWolf 4TB NAS Hard Drive 5900 RPM 64MB Cache SATA 6.0Gb/s 3.5" Internal Hard Drive ST4000VN008
\$25.40	\$253.99	10TB	Seagate Exos Enterprise Capacity 3.5" HDD 10TB (Helium) 7200 RPM SATA 6Gb/s 256MB Cache Hyperscale 512e Internal Hard Drive ST10000NM0016
\$25.67	\$307.99	12TB	Seagate Exos Enterprise Capacity ST12000NM0007 12TB 7200 RPM SATA 6Gb/s 256MB Enterprise Hard Drive (Helium & 3.5 inch)
\$25.75	\$102.99	4TB	WD Purple 4TB Surveillance Hard Disk Drive - 5400 RPM Class SATA 6Gb/s 64MB Cache 3.5 Inch WD40PURZ
\$26.00	\$103.99	4TB	Seagate SkyHawk 4TB Surveillance Hard Drive 64MB Cache SATA 6.0Gb/s 3.5" Internal Hard Drive ST4000VX007

[more](#)

Classic Time-memory tradeoff



Classic Time-memory tradeoff

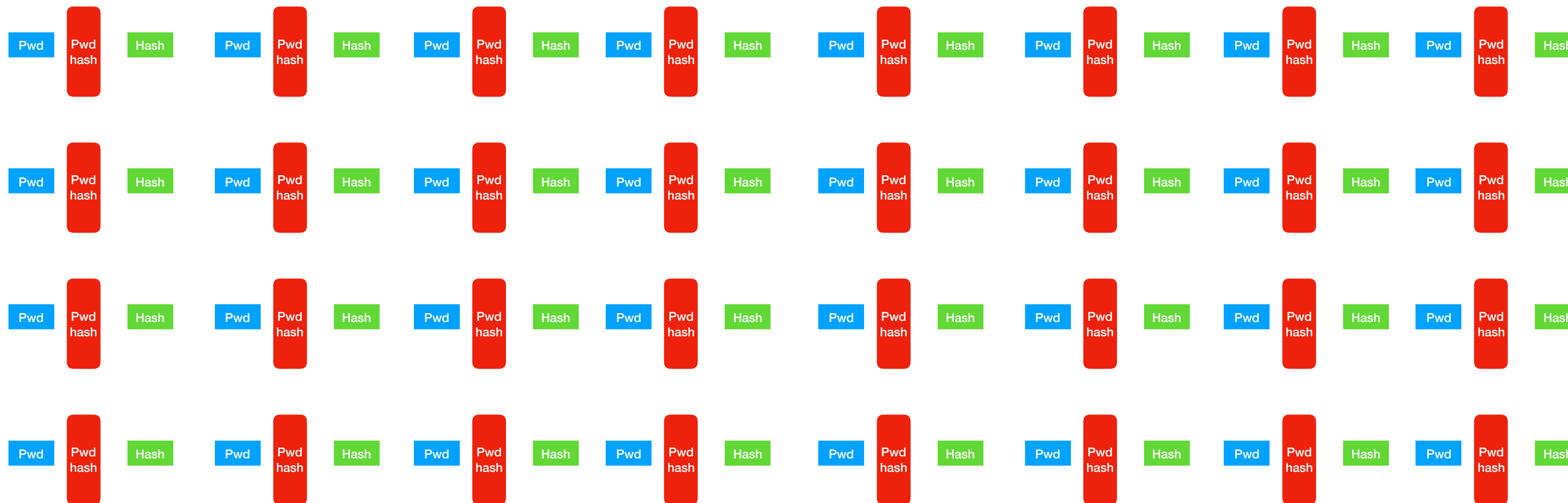
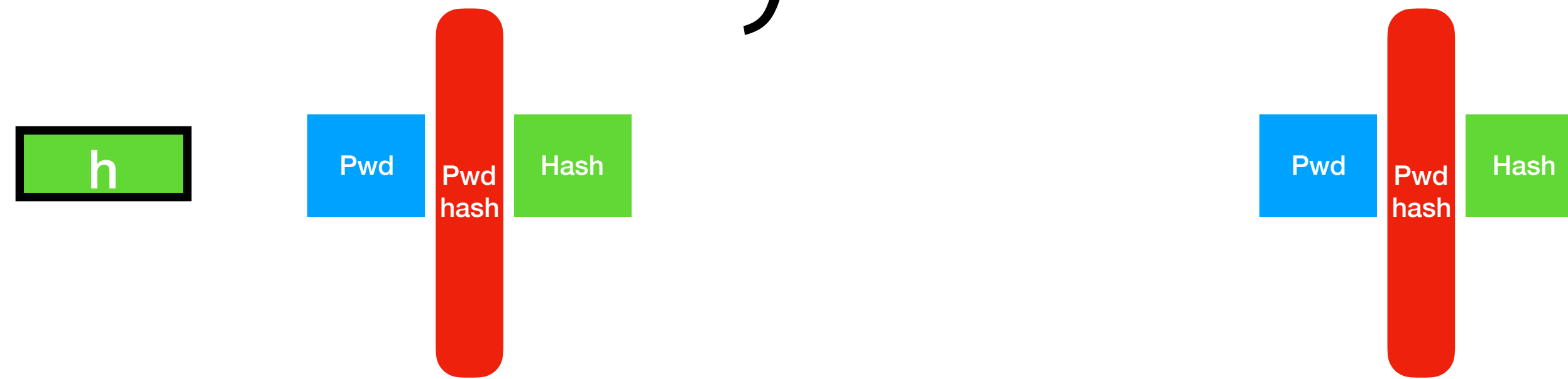




Given a hash [h] that you want to invert, you can:

h

Classic Time-memory tradeoff



SHA1 Rainbow Tables

Table ID	Charset	Plaintext Length	Key Space	Success Rate	Table Size	Files	Performance
☰ sha1_ascii-32-95#1-7	ascii-32-95	1 to 7	70,576,641,626,495	99.9 %	52 GB 64 GB	Perfect Non-perfect	Perfect Non-perfect
☰ sha1_ascii-32-95#1-8	ascii-32-95	1 to 8	6,704,780,954,517,120	96.8 %	460 GB 576 GB	Perfect Non-perfect	Perfect Non-perfect
☰ sha1_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	221,919,451,578,090	99.9 %	127 GB 160 GB	Perfect Non-perfect	Perfect Non-perfect
☰ sha1_mixalpha-numeric#1-9	mixalpha-numeric	1 to 9	13,759,005,997,841,642	96.8 %	690 GB 864 GB	Perfect Non-perfect	Perfect Non-perfect
☰ sha1_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	104,461,669,716,084	99.9 %	65 GB 80 GB	Perfect Non-perfect	Perfect Non-perfect
☰ sha1_loweralpha-numeric#1-10	loweralpha-numeric	1 to 10	3,760,620,109,779,060	96.8 %	316 GB 396 GB	Perfect Non-perfect	Perfect Non-perfect

RainbowCrack Software Features

- High performance hash cracking on PC (> 10,000,000,000,000 plaintext tests per second)
- Optimized implementation of time-memory trade-off algorithm
- GPU acceleration with NVIDIA and AMD GPUs
- GPU acceleration with multiple GPUs
- Supports 64-bit Windows operating system
- Easy to use



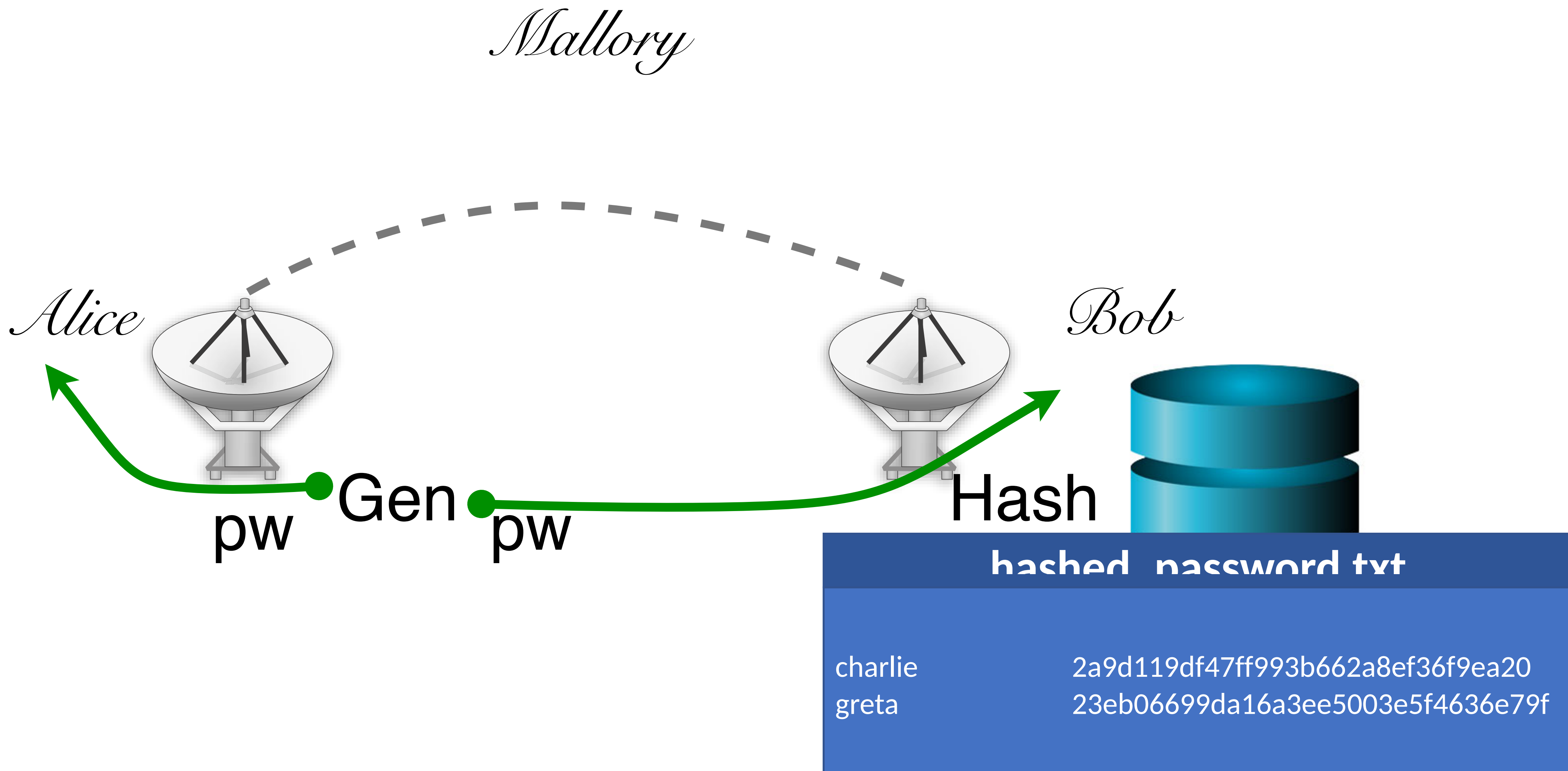
- [RainbowCrack 1.7 software](#)
- One [Seagate BarraCuda 6TB ST6000DM003 \(SATA\)](#) hard drive containing rainbow tables and software
- License in USB dongle



The attack is highly effective

<https://www.youtube.com/watch?v=TkMZJ3fTgrM>

Attack 2: offline brute force



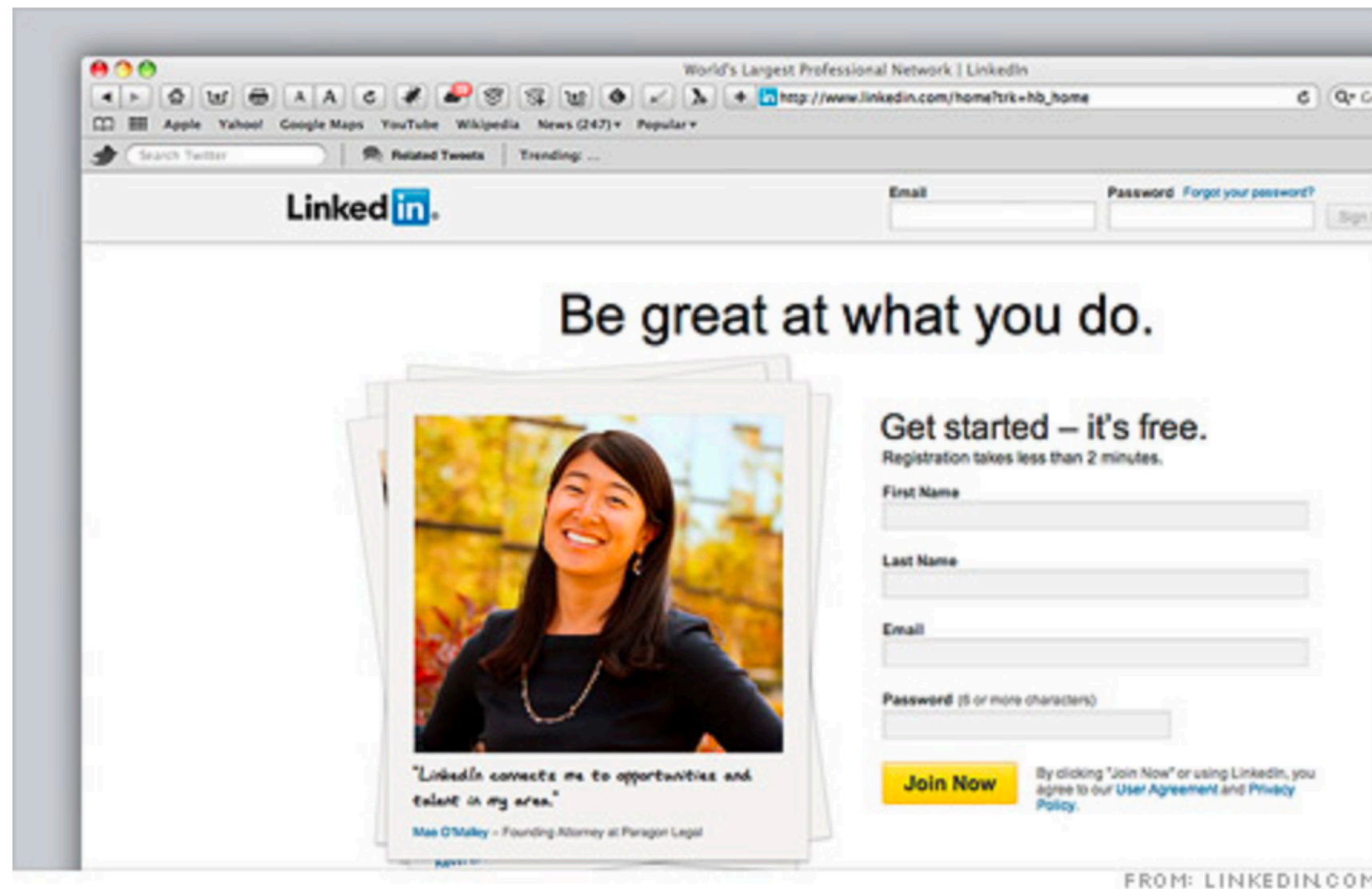
Pwd breaches

CNN Money

THE CYBERCRIME ECONOMY

More than 6 million LinkedIn passwords stolen

By David Goldman @CNMMoneyTech June 7, 2012: 9:34 AM ET



Researchers say a stash of what appear to be LinkedIn passwords were protected by a weak security scheme.

NEW YORK (CNMMoney) -- Russian hackers released a giant list of passwords this week, and on Wednesday security researchers identified their likely source: business social networking site LinkedIn.

2012: 6.5 million hashes leaked onto Internet 90% cracked in 2 weeks

2016: 177.5 million more hashes leaked 98% cracked in 1 week

2012 LinkedIn Breach had 117 Million Emails and Passwords Stolen, Not 6.5M


May 18, 2016



Long time users of LinkedIn users may very well need to change their passwords once more



Related Posts

- Web Skimming Attack on Blue
-  Bear Affects School Admin

by Paul Ducklin



One month ago today, we wrote about Adobe's [giant data breach](#).

As far as anyone knew, including Adobe, it affected about 3,000,000 customer records, which made it sound pretty bad right from the start.



But worse was to come, as recent updates to the story bumped the number of affected customers to a [whopping 38,000,000](#).

We took Adobe to task for a lack of clarity in its breach notification.

OUR COMPLAINT

One of our complaints was that Adobe said that it had lost *encrypted* passwords, when we thought the company ought to have said that it had lost

```
4464 ① User ID yahoo.com-|-g2B6PhWEH36 ⑤ Password hint try: qwerty123 --
4465-|-|-xxxxx@jcom.home.ne.jp-|-Eh5tLomK+N+82csoVwU9bw==|-|?????|--
4466-|-|-xx@hotmail.com-|-ahw2b2BELzgrTWYvQGn+kw==|-quiero a...|--
4467-|-|-xxx@yahoo.com-|-leMTcMPEPcjioxG6CatHBw==|-|--
4468-|-username ② Username e.com-|-2GthVrmsERzioxG6CatHBw==|-|--
4469-|-|-xxxxx@yahoo.com-|-4LSlo772tH4= ④ Password data (base64)
4470-|-|-xxx@hotmail.com-|-xxxxx@xxxxx.com-|-xG6CatHBw==|-|--
4471-|-|-xxxx@yahoo.com ③ Email address xG6CatHBw==|-myspace|--
4471-|-|-xxx@hotmail.com-|-kby1918wDrrioxG6CatHBw==|-regular|--
```

Adobe password data	Password hint
110edf2294fb8bf4	-> numbers 123456
110edf2294fb8bf4	-> ==123456 ① 123456
110edf2294fb8bf4	-> c'est "123456"
8fda7e1f0b56593f e2a311ba09ab4707	-> numbers
8fda7e1f0b56593f e2a311ba09ab4707	-> 1-8 ② 12345678
8fda7e1f0b56593f e2a311ba09ab4707	-> 8digit
2fca9b003de39778 e2a311ba09ab4707	-> the password is password
2fca9b003de39778 e2a311ba09ab4707	-> password ③ password
2fca9b003de39778 e2a311ba09ab4707	-> rhymes with assword
e5d8efed9088db0b	-> q w e r t y
e5d8efed9088db0b	-> ytrewq tagurpidi ④ qwerty
e5d8efed9088db0b	-> 6 long qwert
ecba98cca55eabc2	-> sixxone
ecba98cca55eabc2	-> 1*6 ⑤ 111111
ecba98cca55eabc2	-> sixones

How to hamper offline brute force attacks?

Mallory

hashed password.txt

charlie	2a9d119df47ff993b662a8ef36f9ea20
greta	23eb06699da16a3ee5003e5f4636e79f

Hardening Password Hashes

- **Key problem: cryptographic hashes are deterministic**
 - $\text{hash}(\text{'p4ssw0rd'}) = \text{hash}(\text{'p4ssw0rd'})$
 - This enables attackers to build lists of hashes

Hardening Password Hashes

- **Key problem: cryptographic hashes are deterministic**
 - $\text{hash}(\text{'p4ssw0rd'}) = \text{hash}(\text{'p4ssw0rd'})$
 - This enables attackers to build lists of hashes
- **Solution: make each password hash unique**
 - Add a random **salt** to each password before hashing
 - $\text{hash}(\text{salt} + \text{password}) = \text{password hash}$
 - Each user has a unique, random salt
 - Salts can be stores in plain text

Example Salted Hashes

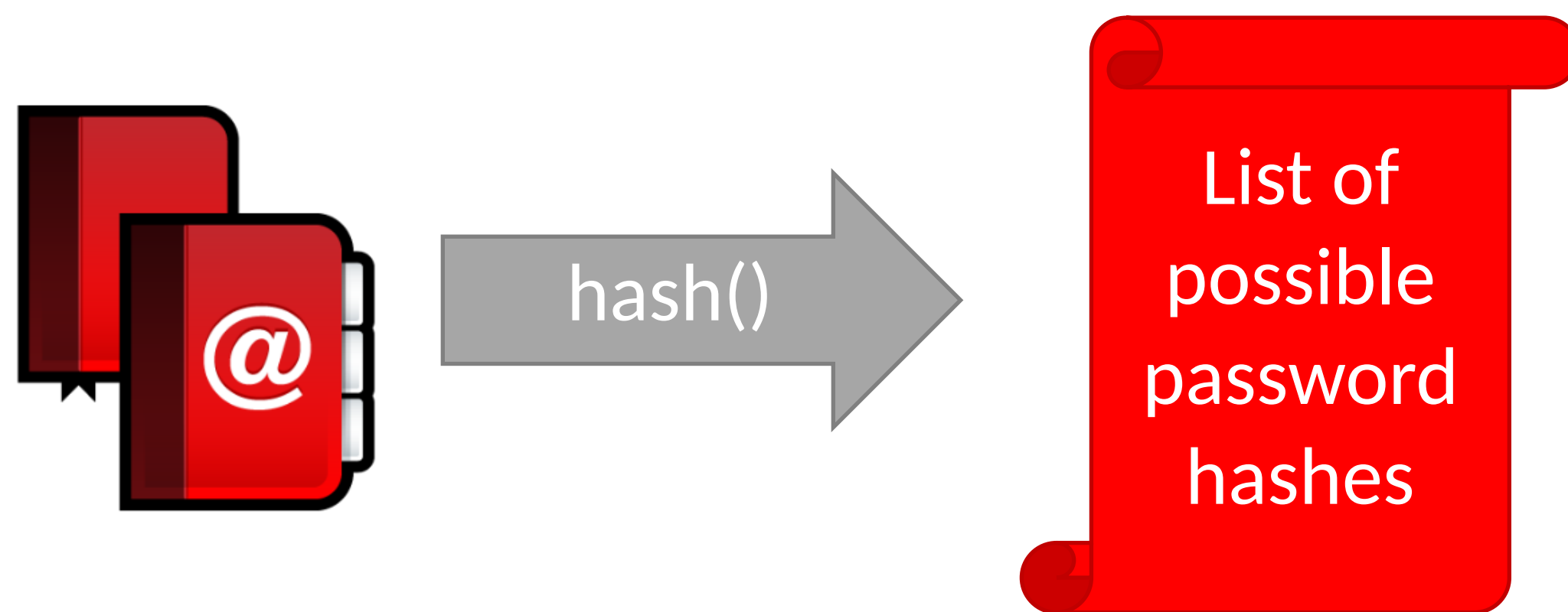
hashed_password.txt

cbw	2a9d119df47ff993b662a8ef36f9ea20
sandi	23eb06699da16a3ee5003e5f4636e79f
amislove	98bd0ebb3c3ec3fbe21269a8d840127c
bob	e91e6348157868de9dd8b25c81aebfb9

hashed_and_salted_password.txt

cbw	a8	af19c842f0c781ad726de7aba439b033
sandi	0X	67710c2c2797441efb8501f063d42fb6
amislove	hz	9d03e1f28d39ab373c59c7bb338d0095
bob	K@	479a6d9e59707af4bb2c618fed89c245

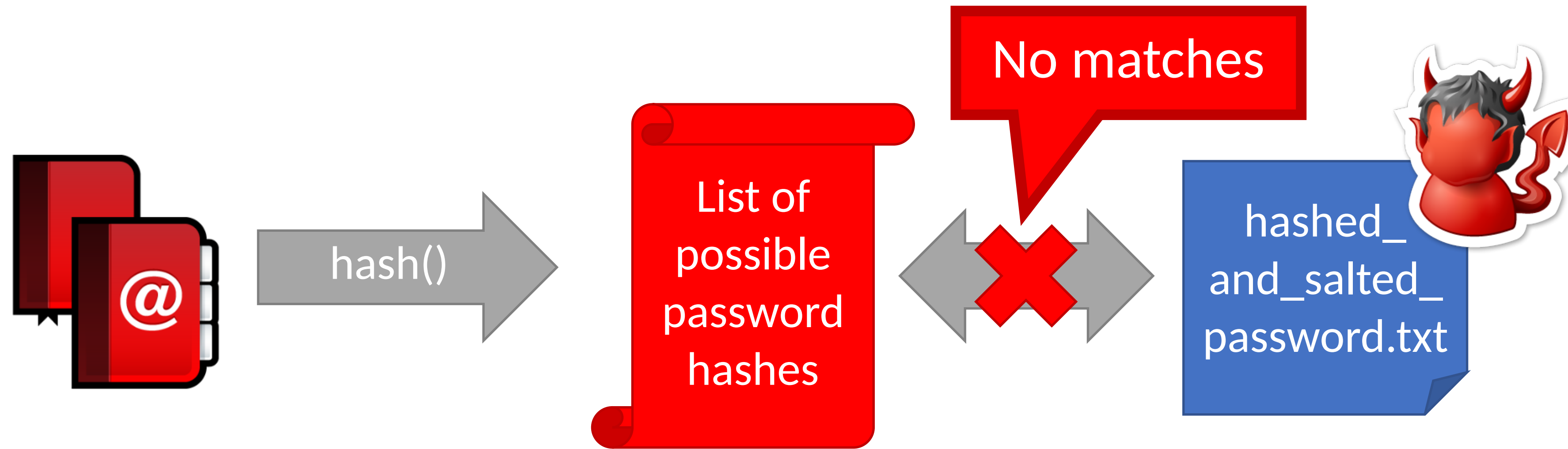
Attacking Salted Passwords



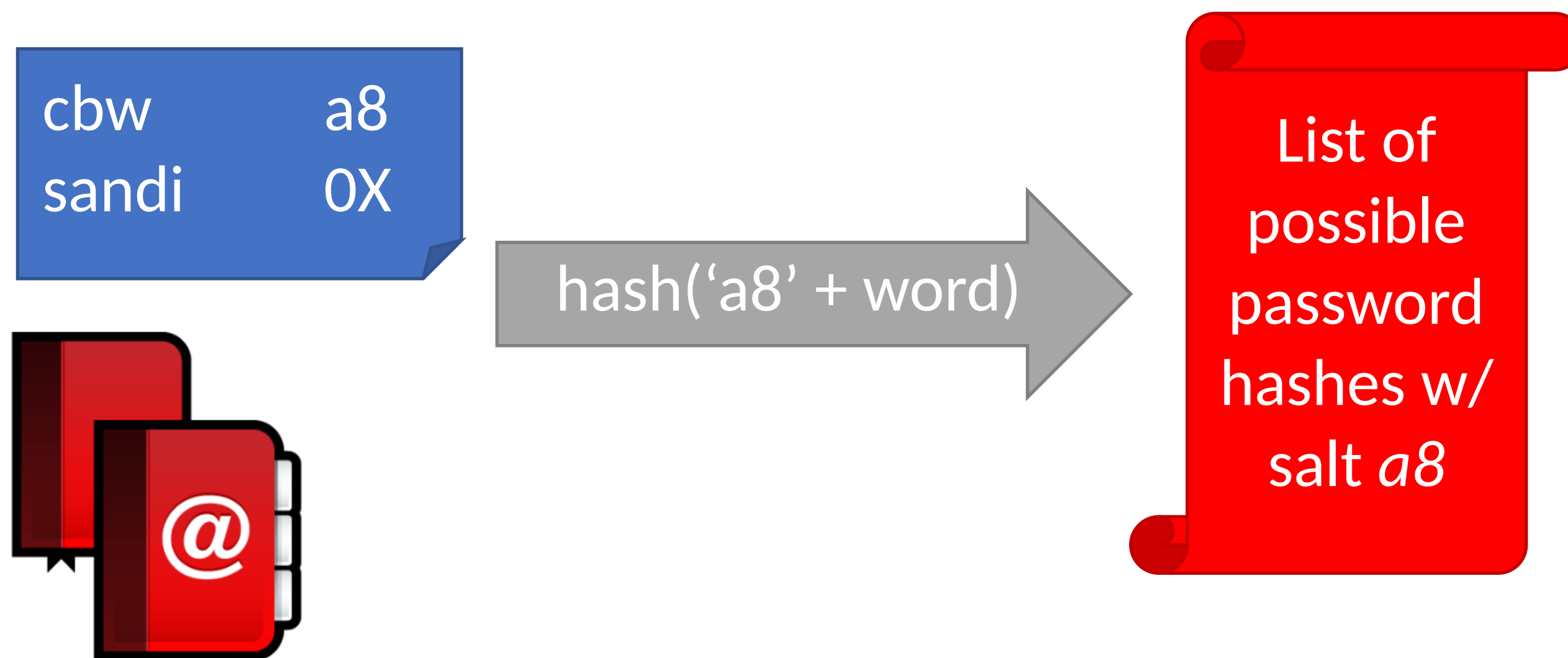
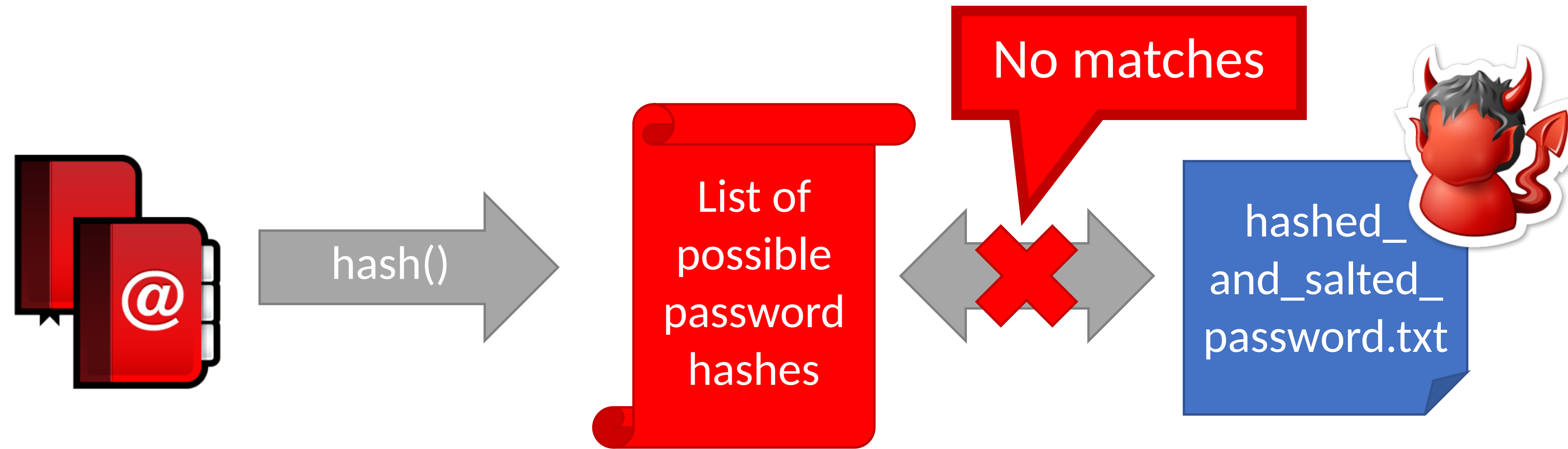
Attacking Salted Passwords



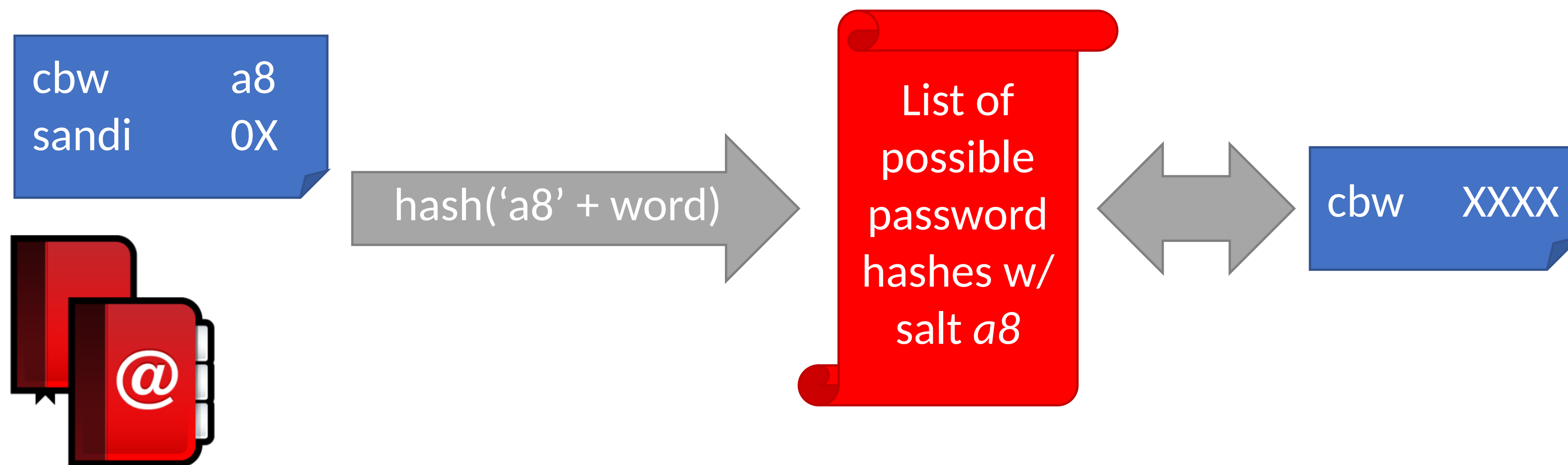
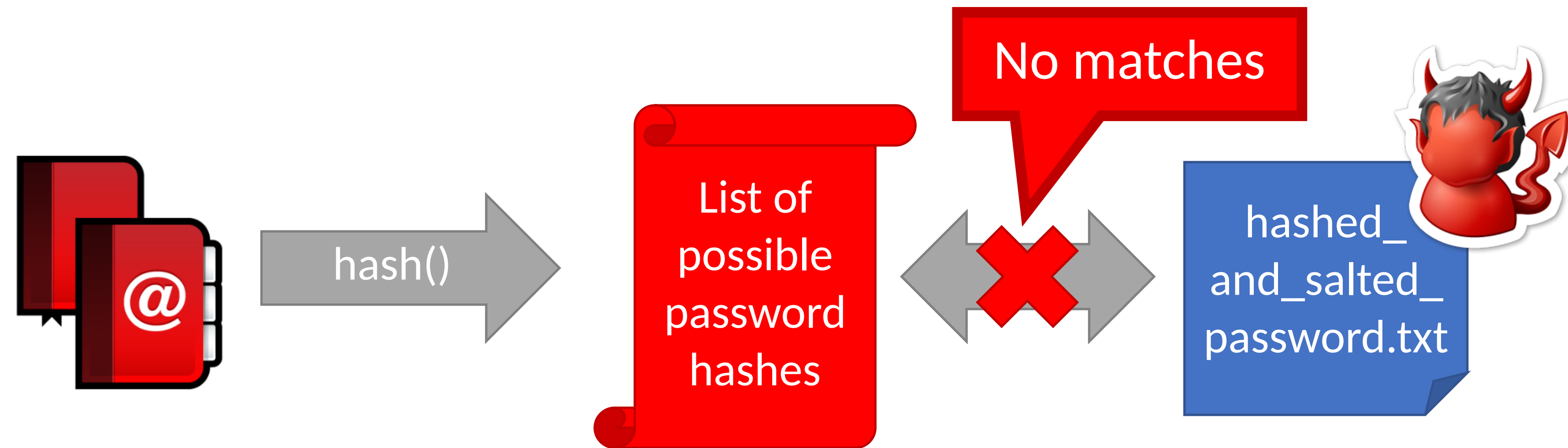
Attacking Salted Passwords



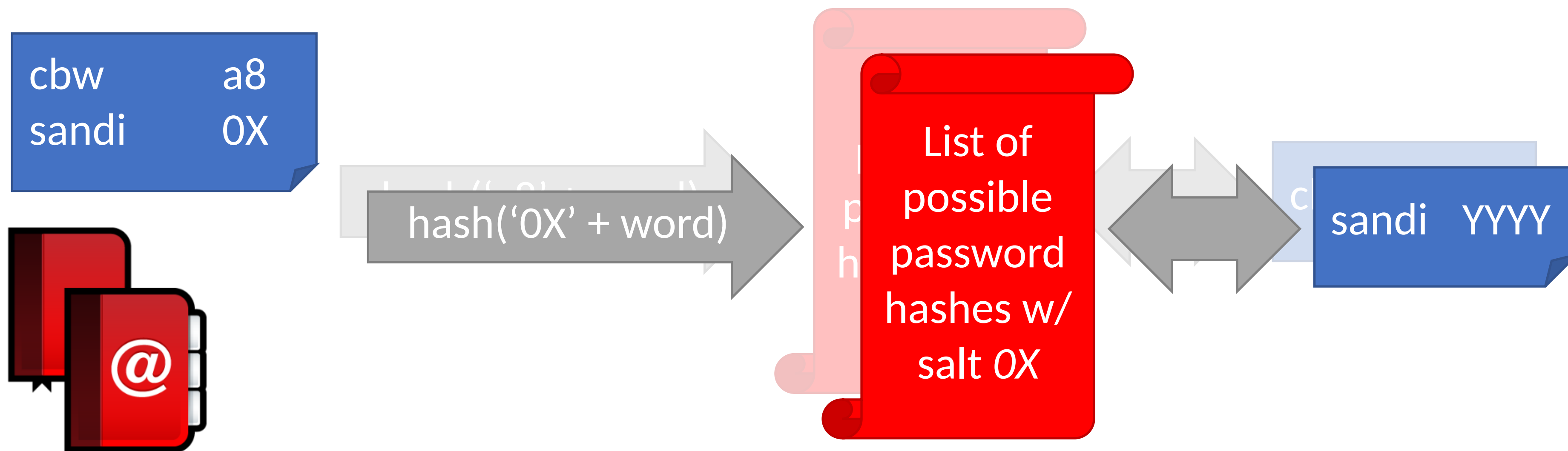
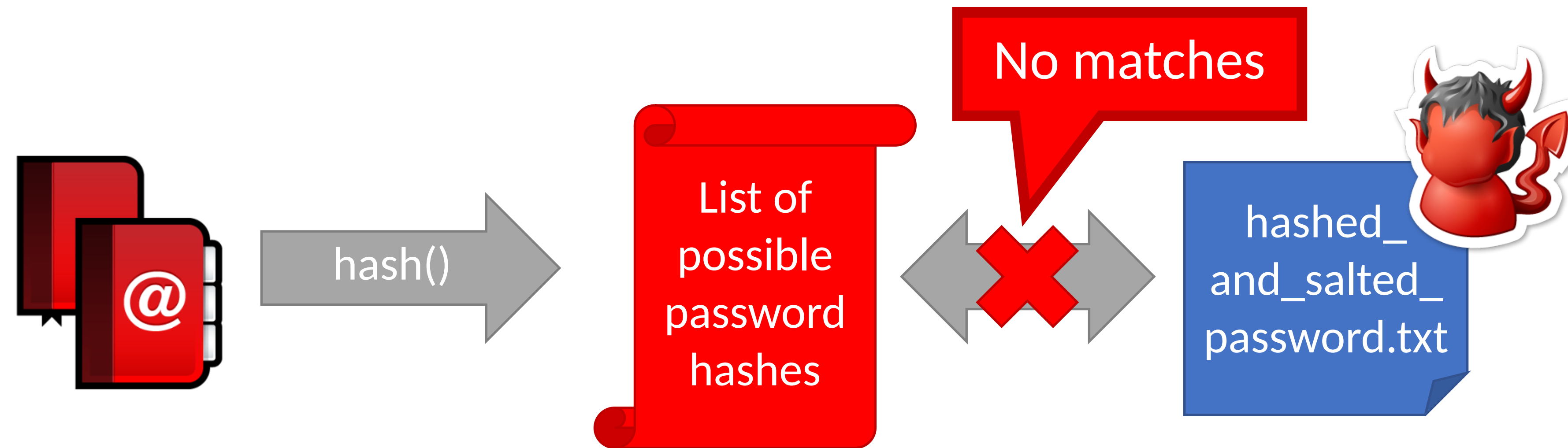
Attacking Salted Passwords



Attacking Salted Passwords



Attacking Salted Passwords



Breaking Hashed Passwords

- **Stored passwords should always be salted**
 - Forces the attacker to brute-force each password individually

Breaking Hashed Passwords

- **Stored passwords should always be salted**
 - Forces the attacker to brute-force each password individually
- **Problem: it is now possible to compute hashes very quickly**
 - GPU computing: hundreds of small CPU cores
 - nVidia GeForce GTX Titan Z: 5,760 cores
 - GPUs can be rented from the cloud very cheaply
 - \$0.9 per hour (2018 prices)

Examples of Hashing Speed

- A modern x86 server can hash all possible 6 character long passwords in 3.5 hours
 - Upper and lowercase letters, numbers, symbols
 - $(26+26+10+32)^6 = 690$ billion combinations

Examples of Hashing Speed

- A modern x86 server can hash all possible 6 character long passwords in 3.5 hours
 - Upper and lowercase letters, numbers, symbols
 - $(26+26+10+32)^6 = 690$ billion combinations
- A modern GPU can do the same thing in 16 minutes

Examples of Hashing Speed

- A modern x86 server can hash all possible 6 character long passwords in 3.5 hours
 - Upper and lowercase letters, numbers, symbols
 - $(26+26+10+32)^6 = 690$ billion combinations
- A modern GPU can do the same thing in 16 minutes
- Most users use (slightly permuted) dictionary words, no symbols
 - Predictability makes cracking much faster
 - Lowercase + numbers $\rightarrow (26+10)^6 = 2B$ combinations

Hardening Salted Passwords

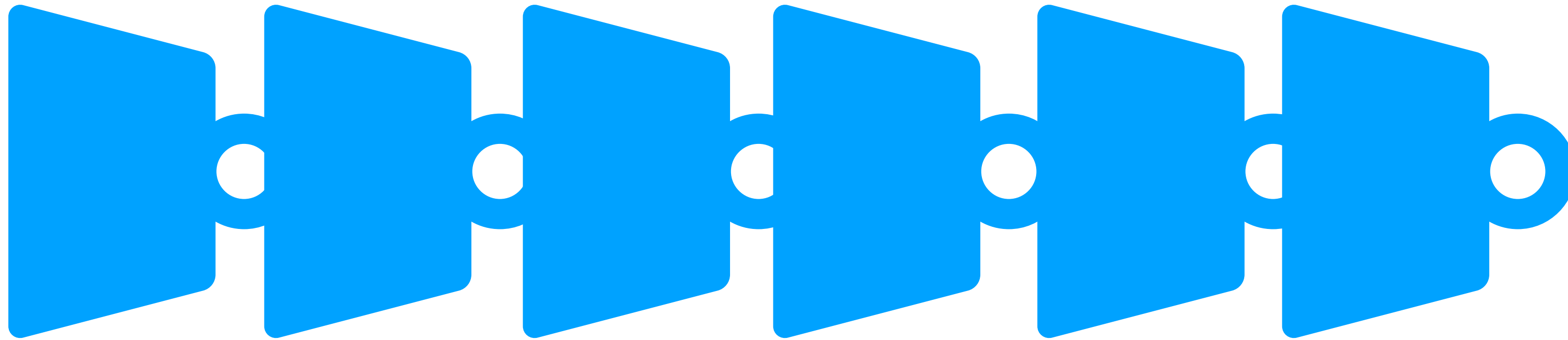
- **Problem:** typical hashing algorithms are too fast
 - Enables GPUs to brute-force passwords
- **Old solution:** hash the password multiple times
 - Known as **key stretching**
 - Example: *crypt* used 25 rounds of DES
- **New solution:** use hash functions that are designed to be **slow**
 - Examples: bcrypt, PBKDF2, scrypt
 - These algorithms include a **work factor** that increases the time complexity of the calculation
 - scrypt also requires a large amount of memory to compute, further complicating brute-force attacks

Slow hash movement



Iterated hash function {x times}

Pw
Salt



Hashed pwd

bcrypt Example

- Python example; install the *bcrypt* package

```
[cbw@localhost ~] python
>>> import bcrypt
>>> password = "my super secret password"
>>> fast_hashed = bcrypt.hashpw(password, bcrypt.gensalt(0))
>>> slow_hashed = bcrypt.hashpw(password, bcrypt.gensalt(12))
>>> pw_from_user = raw_input("Enter your password:")
>>> if bcrypt.hashpw(pw_from_user, slow_hashed) == slow_hashed:
...     print "It matches! You may enter the system"
... else:
...     print "No match. You may not proceed"
```

Work factor

Best practices so far:

Dealing With Breaches

Dealing With Breaches

- Suppose you build an extremely secure password storage system
 - All passwords are salted and hashed by a high-work factor function
- It is still possible for a dedicated attacker to steal and crack passwords
 - Given enough time and money, anything is possible
 - E.g. The NSA
- Question: is there a principled way to detect password breaches?

Honeywords

- Key idea: store multiple salted/hashed passwords for each user
 - As usual, users create a single password and use it to login
 - User is unaware that additional **honeywords** are stored with their account

Honeywords

- Key idea: store multiple salted/hashed passwords for each user
 - As usual, users create a single password and use it to login
 - User is unaware that additional **honeywords** are stored with their account
- Implement a **honeyserver** that stores the index of the correct password for each user
 - Honeyserver is logically and physically separate from the password database
 - Silently checks that users are logging in with true passwords, not honeywords

Honeywords

- Key idea: store multiple salted/hashed passwords for each user
 - As usual, users create a single password and use it to login
 - User is unaware that additional **honeywords** are stored with their account
- Implement a **honeyserver** that stores the index of the correct password for each user
 - Honeyserver is logically and physically separate from the password database
 - Silently checks that users are logging in with true passwords, not honeywords
- What happens after a data breach?
 - Attacker dumps the user/password database...
 - But the attacker doesn't know which passwords are honeywords
 - Attacker cracks all passwords and uses them to login to accounts
 - If the attacker logs-in with a honeyword, the honeyserver raises an alert!

Honeywords example

Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1

Honeywords example



Bob

Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1

Honeywords example



Bob

SHA512("fl" | "p4ssW0rd") → bHDJ8l

Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1

Honeywords example



Bob

SHA512("fl" | "p4ssW0rd") → bHDJ8l

Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1



Honeywords example



Bob



SHA512("fl" | "p4ssW0rd") → bHDJ8l

Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1

Honeywords example



Bob

SHA512("fl" | "p4ssW0rd") → bHDJ8l



Cracked Passwords

User	PW 1	PW 2	PW 3
Bob	123456	p4ssW0rd	Turtles!
sandi	puppies	iloveyou	blizzard
Alice	coff33	3spr3ss0	qwerty



Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1

Honeywords example



Bob

SHA512("fl" | "p4ssW0rd") → bHDJ8l



Cracked Passwords

User	PW 1	PW 2	PW 3
Bob	123456	p4ssW0rd	Turtles!
sandi	puppies	iloveyou	blizzard
Alice	coff33	3spr3ss0	qwerty



Database



User	Salt 1	H(PW 1)	Salt 2	H(PW 2)	Salt 3	H(PW 3)
Bob	aB	y4DvF7	fl	bHDJ8l	52	Puu2s7
sandi	0x	pIDS4F	K2	R/p3Y8	8W	S8x4Gk
Alice	9j	0F3g5H	/s	03d5jW	cV	1sRbJ5

Honeyserver



User	Index
Bob	2
sandi	3
Alice	1

Multiple layers of storage

Password Storage Summary

- 1. Never store passwords in plain text**
 - 2. Always salt and hash passwords before storing them**
 - 3. Use hash functions with a high work factor**
 - 4. Implement honeywords to detect breaches**
- **These rules apply to any system that needs to authenticate users**
 - Operating systems, websites, etc.

Password Recovery/Reset

- Problem: hashed passwords cannot be recovered (hopefully)



“Hi... I forgot my password. Can you email me a copy? Kthxbye”

- This is why systems typically implement password **reset**
 - Use out-of-band info to authenticate the user
 - Overwrite `hash(old_pw)` with `hash(new_pw)`
- Be careful: its possible to crack password reset

Cracking Password Reset

- Typical implementations use **Knowledge Based Authentication (KBA)**
 - What was your mother's maiden name?
 - What was your prior street address?
 - Where did you go to elementary school

Cracking Password Reset

- Typical implementations use **Knowledge Based Authentication (KBA)**
 - What was your mother's maiden name?
 - What was your prior street address?
 - Where did you go to elementary school
- Problems?

Cracking Password Reset

- Typical implementations use **Knowledge Based Authentication (KBA)**
 - What was your mother's maiden name?
 - What was your prior street address?
 - Where did you go to elementary school
- **Problems?**
 - This information is widely available to anyone
 - Publicly accessible social network profiles
 - Background-check services like Spokeo

Cracking Password Reset

- Typical implementations use **Knowledge Based Authentication (KBA)**
 - What was your mother's maiden name?
 - What was your prior street address?
 - Where did you go to elementary school
- **Problems?**
 - This information is widely available to anyone
 - Publicly accessible social network profiles
 - Background-check services like Spokeo
- **Experts recommend that services not use KBA**
 - When asked, users should generate random answers to these questions

Choosing Passwords

Bad Algorithms

Better Heuristics

Password Reuse

Mental Algorithms

- Years of security advice have trained people to generate “secure” passwords

Mental Algorithms

- Years of security advice have trained people to generate “secure” passwords
 1. Pick a word
 2. Capitalize the first or last letter
 3. Add a number (and maybe a symbol) to the beginning or end

Mental Algorithms

- Years of security advice have trained people to generate “secure” passwords
 1. Pick a word
 2. Capitalize the first or last letter
 3. Add a number (and maybe a symbol) to the beginning or end

- 1. Pick a word
 2. Replace some of the letters with symbols (a → @, s → \$, etc.)
 3. Maybe capitalize the first or last letter

Human Generated Passwords

Password				
Computer3@				nsf
cl4ssr00m				nsf
7Dogsled*				nsf
Tjw1989&6	54	Weak	Easy	Users initials and birthday, obvi
B4nk0f4m3r1c4!	83	Medium	Easy	Includes service name, obvious

Human Generated Passwords

Password	Entropy (bits)			
Computer3@	60			
cl4ssr00m	47			
7Dogsled*	54			
Tjw1989&6	54	Weak	Easy	Users initials and birthday, obvious
B4nk0f4m3r1c4!	83	Medium	Easy	Includes service name, obvious

Human Generated Passwords

Password	Entropy (bits)	Strength	Crackability	Problem
Computer3@	60	Weak	Easy	Dictionary word, obvious transf
cl4ssr00m	47	Weak	Easy	Dictionary word, obvious transf
7Dogsled*	54	Weak	Easy	Dictionary word, obvious transf
Tjw1989&6	54	Weak	Easy	Users initials and birthday, obvi
B4nk0f4m3r1c4!	83	Medium	Easy	Includes service name, obvious

Human Generated Passwords

Password	Entropy (bits)	Strength	Crackability	Problem
Computer3@	60	Weak	Easy	Dictionary word, obvious transfo
cl4ssr00m	47	Weak	Easy	Dictionary word, obvious transfo
7Dogsled*	54	Weak	Easy	Dictionary word, obvious transfo
Tjw1989&6	54	Weak	Easy	Users initials and birthday, obvi
B4nk0f4m3r1c4!	83	Medium	Easy	Includes service name, obvious

- Modern attackers are sophisticated
 - No need for brute force cracking!
 - Use dictionaries containing common words and passwords from prior leaks
 - Apply common “mental” permutations

Password classes

1 character class and 8 characters minimum

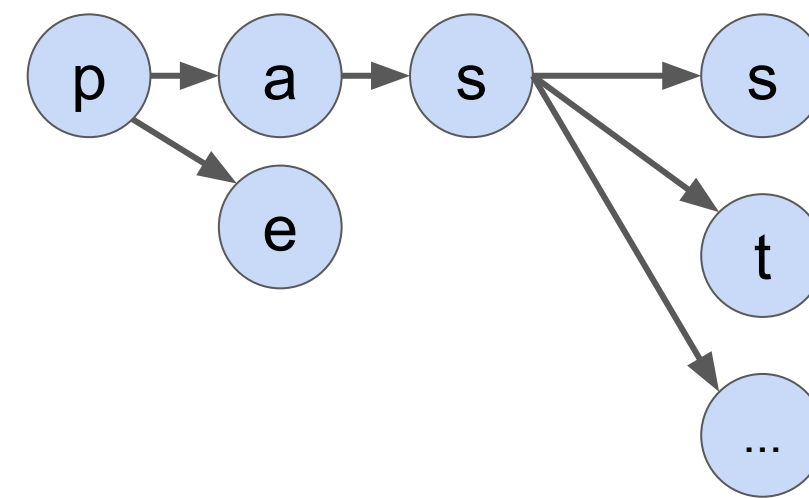
4 character class and 8 characters minimum

Programs

John the ripper

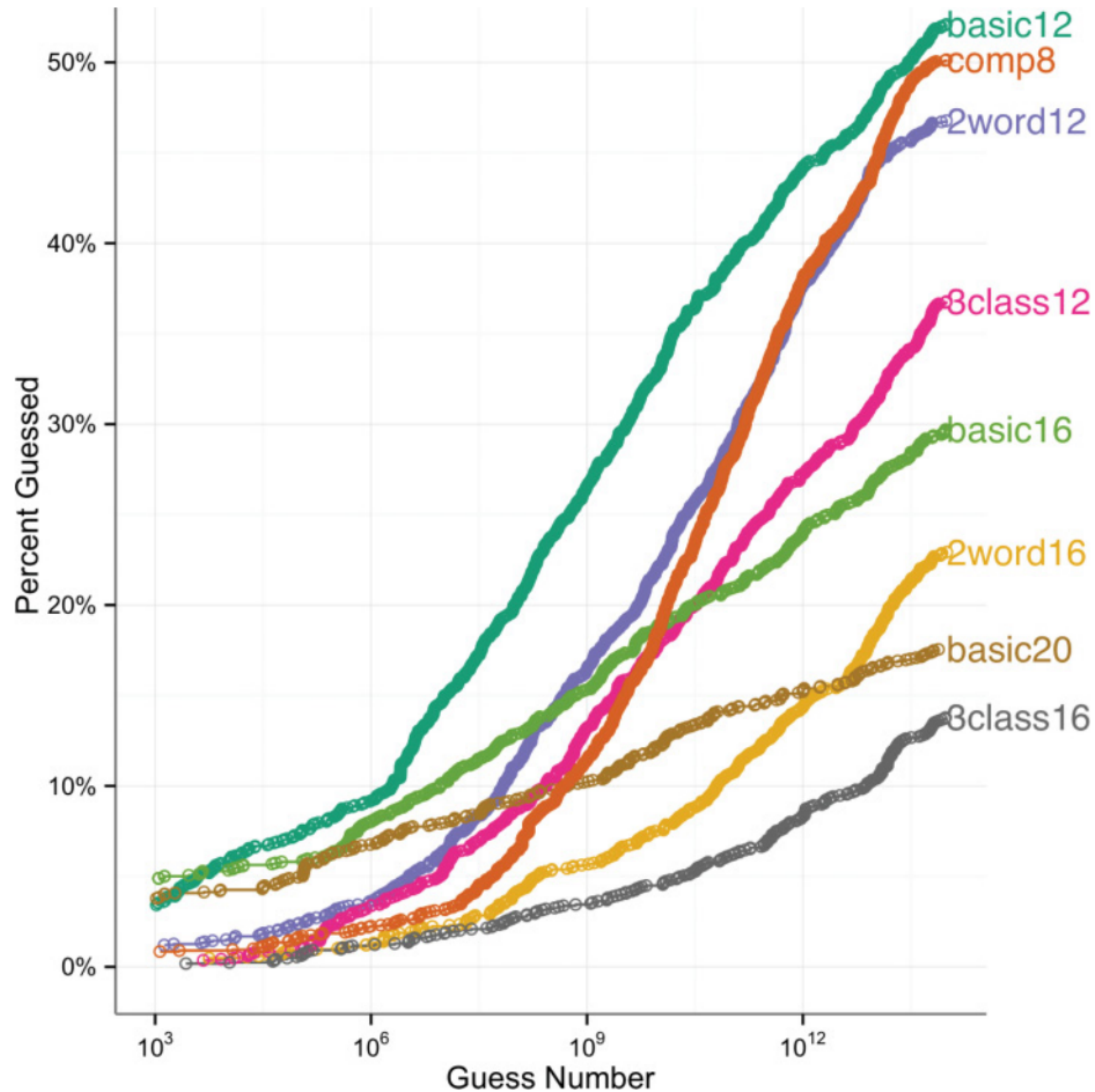
hashcat

pcfg



Password Requirements

- $\text{comp } n$ and $\text{basic } n$: use at least n characters
- k word n : combine at least k words using at least n characters
- d class n : use at least d character types (upper, lower, digit, symbol) with at least n characters



Neural Nets

Fast, Lean, and Accurate: Modeling Password Guessability Using Neural Networks

Melicher et al, Usenix'2016

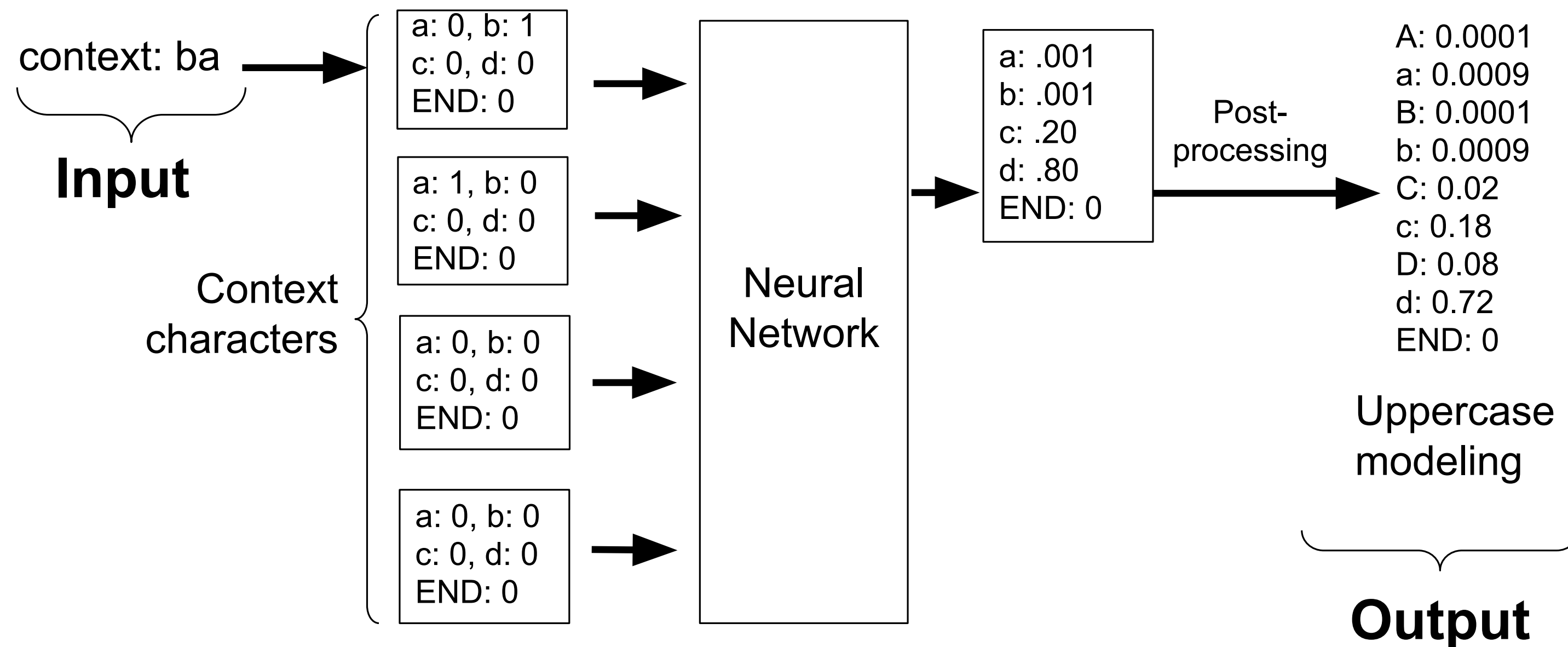
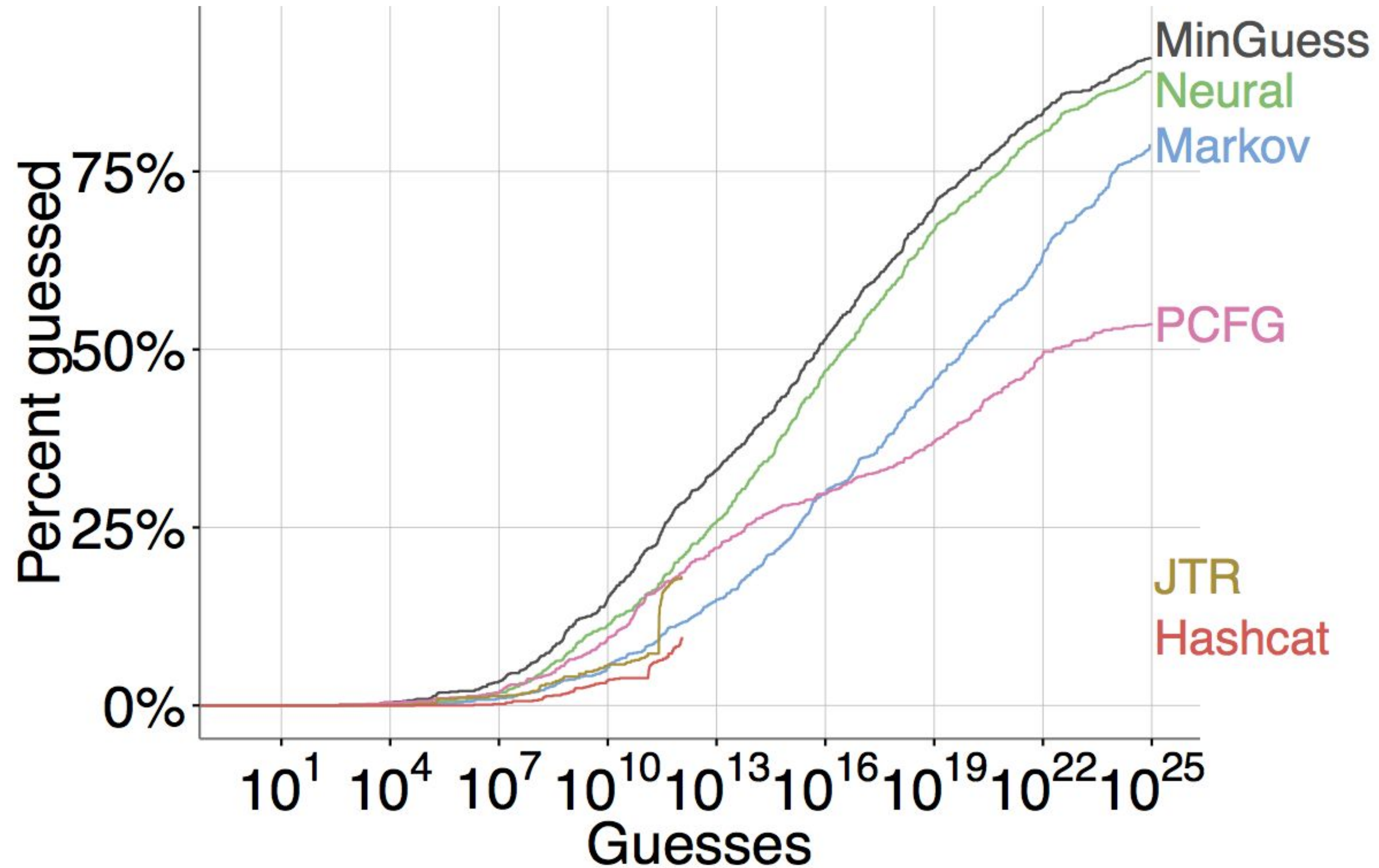


Figure 1: An example of using a neural network to predict the next character of a password fragment. The network is being used to predict a 'd' given the context 'ba'. This network uses four characters of context. The probabilities of each next character are the output of the network. Post processing on the network can infer probabilities of uppercase characters.

3class12: Neural Networks Guess Better



Password feedback

<https://cups.cs.cmu.edu/meter/>

Better Heuristics

- Notice that in $S = L * \log_2 N$, length matters more than symbol types
 - Choose longer passwords (16+ characters)
 - Don't worry about uppercase, digits, or symbols



Better Heuristics

- Notice that in $S = L * \log_2 N$, length matters more than symbol types
 - Choose longer passwords (16+ characters)
 - Don't worry about uppercase, digits, or symbols
- Use mnemonics
 - Choose a sentence or phrase
 - Reduce it to the first letter of each word
 - Insert random uppercase, digits, and symbols



Better Heuristics

- Notice that in $S = L * \log_2 N$, length matters more than symbol types
 - Choose longer passwords (16+ characters)
 - Don't worry about uppercase, digits, or symbols
- Use mnemonics
 - Choose a sentence or phrase
 - Reduce it to the first letter of each word
 - Insert random uppercase, digits, and symbols

I double dare you, say “what” one more time



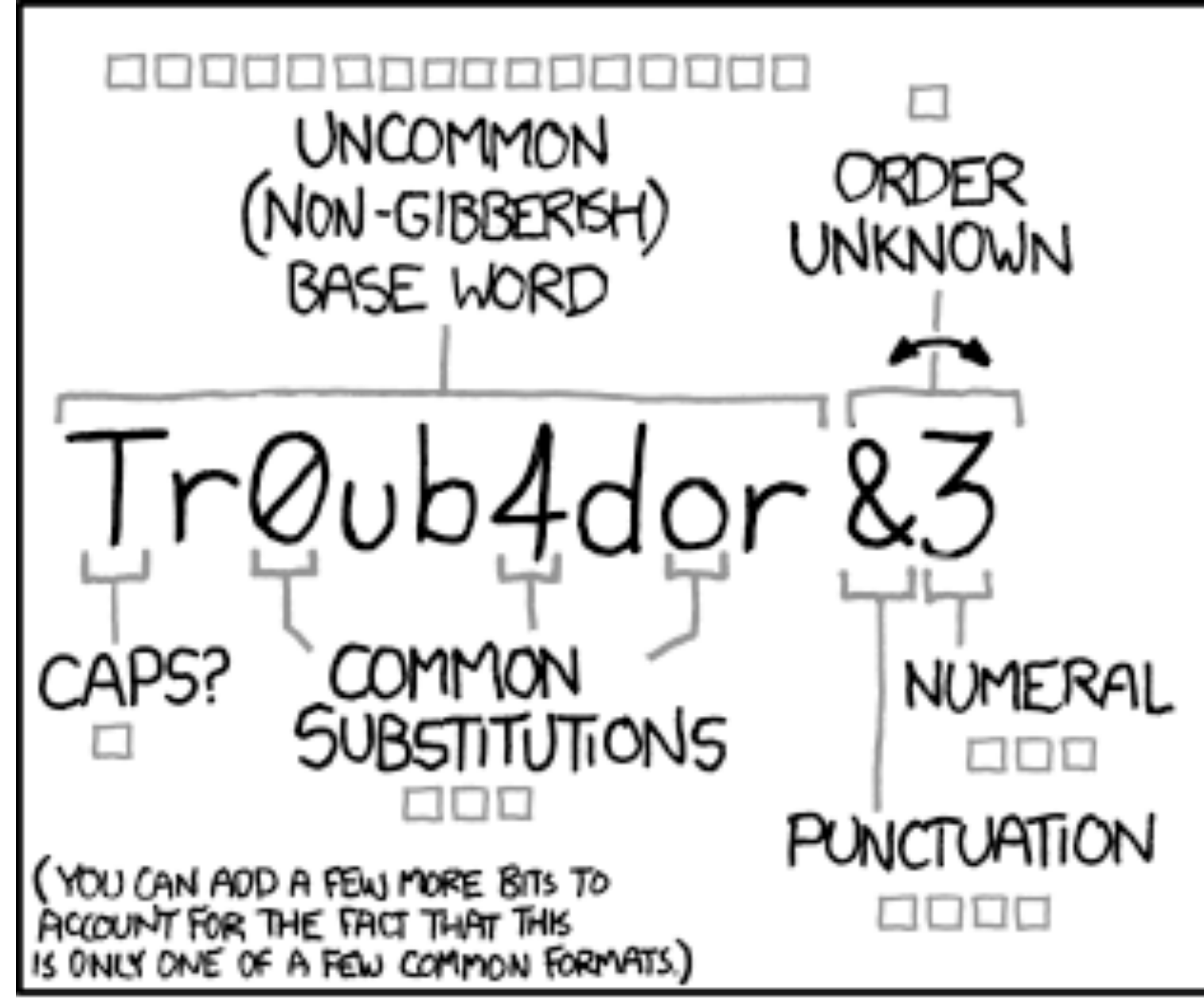
Better Heuristics

- Notice that in $S = L * \log_2 N$, length matters more than symbol types
 - Choose longer passwords (16+ characters)
 - Don't worry about uppercase, digits, or symbols
- Use mnemonics
 - Choose a sentence or phrase
 - Reduce it to the first letter of each word
 - Insert random uppercase, digits, and symbols

I double dare you, say "what" one more time

i2Dy,s"w"omt





~28 BITS OF ENTROPY

□□□□□□□□ □

□□□ □□□

□□□□ □

$2^{28} = 3$ DAYS AT 1000 GUESSES/SEC

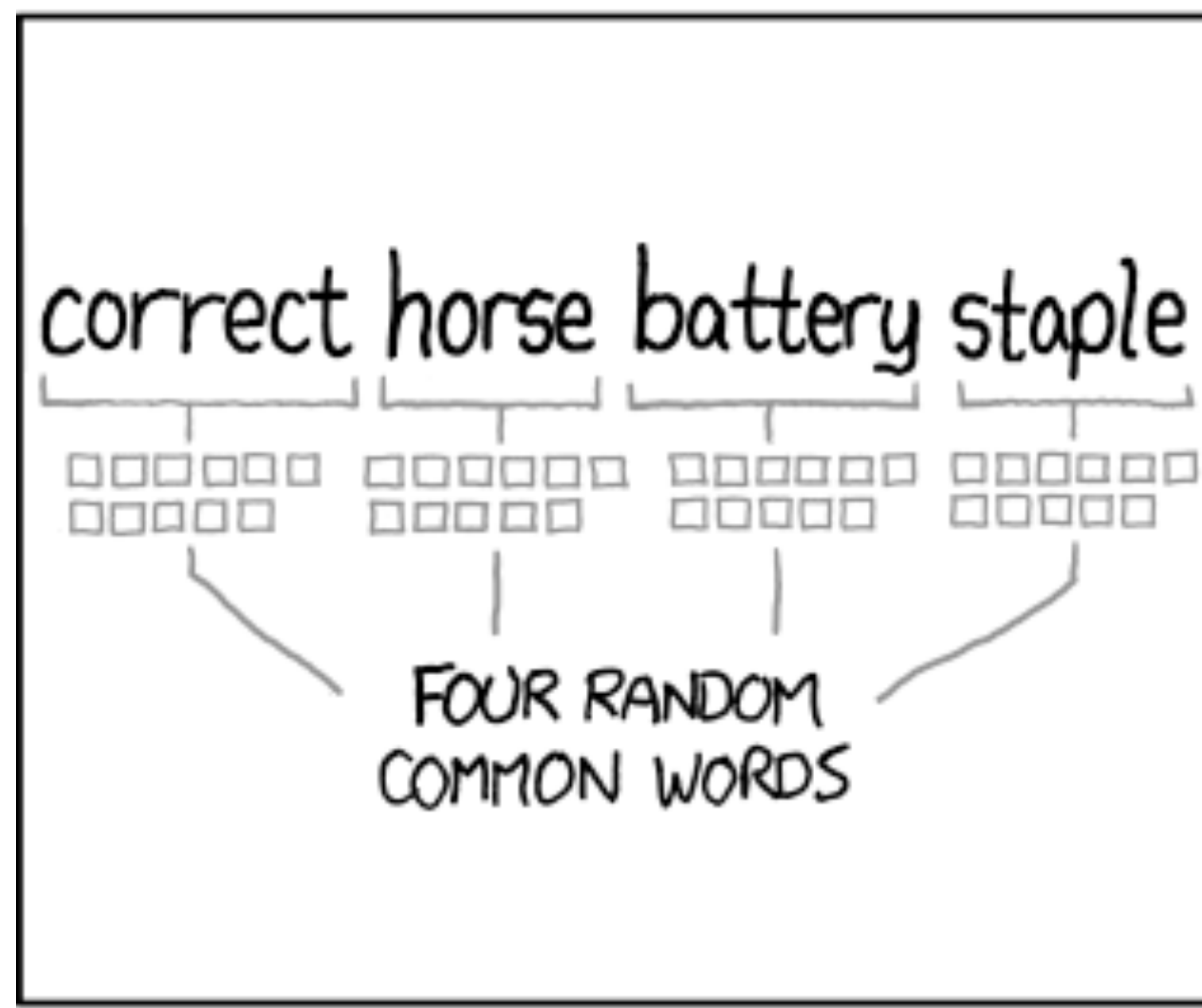
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)

DIFFICULTY TO GUESS: **EASY**

WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE 0s WAS A ZERO?

AND THERE WAS SOME SYMBOL...

DIFFICULTY TO REMEMBER: **HARD**



~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.

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
- Some service providers now check for password reuse
 - Forbid users from selecting passwords that have appeared in leaks

Sites



Favorites (8) ▾

AirBnB
fan@lastpass.comAmazon
fan@lastpass.com

Launch

Best Buy
fan@lastpass.comDropbox
fan@lastpass.comEvernote
fan@lastpass.comFacebook
fan@lastpass.comPocket
fan@lastpass.comTwitter
fan@lastpass.com

Banking and Finance (3) ▾

Read Only • Shared Folder

Bank of America
fan@lastpass.comFidelity
fan@lastpass.comMint
fan@lastpass.com



Home

Notify me

Domain search

Who's been pwned

Passwords

API

About

Donate

';--have i been pwned?

Check if you have an account that has been compromised in a data breach

264

pwned websites

4,859,717,682

pwned accounts

61,081

pastes

59,268,789

paste accounts

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

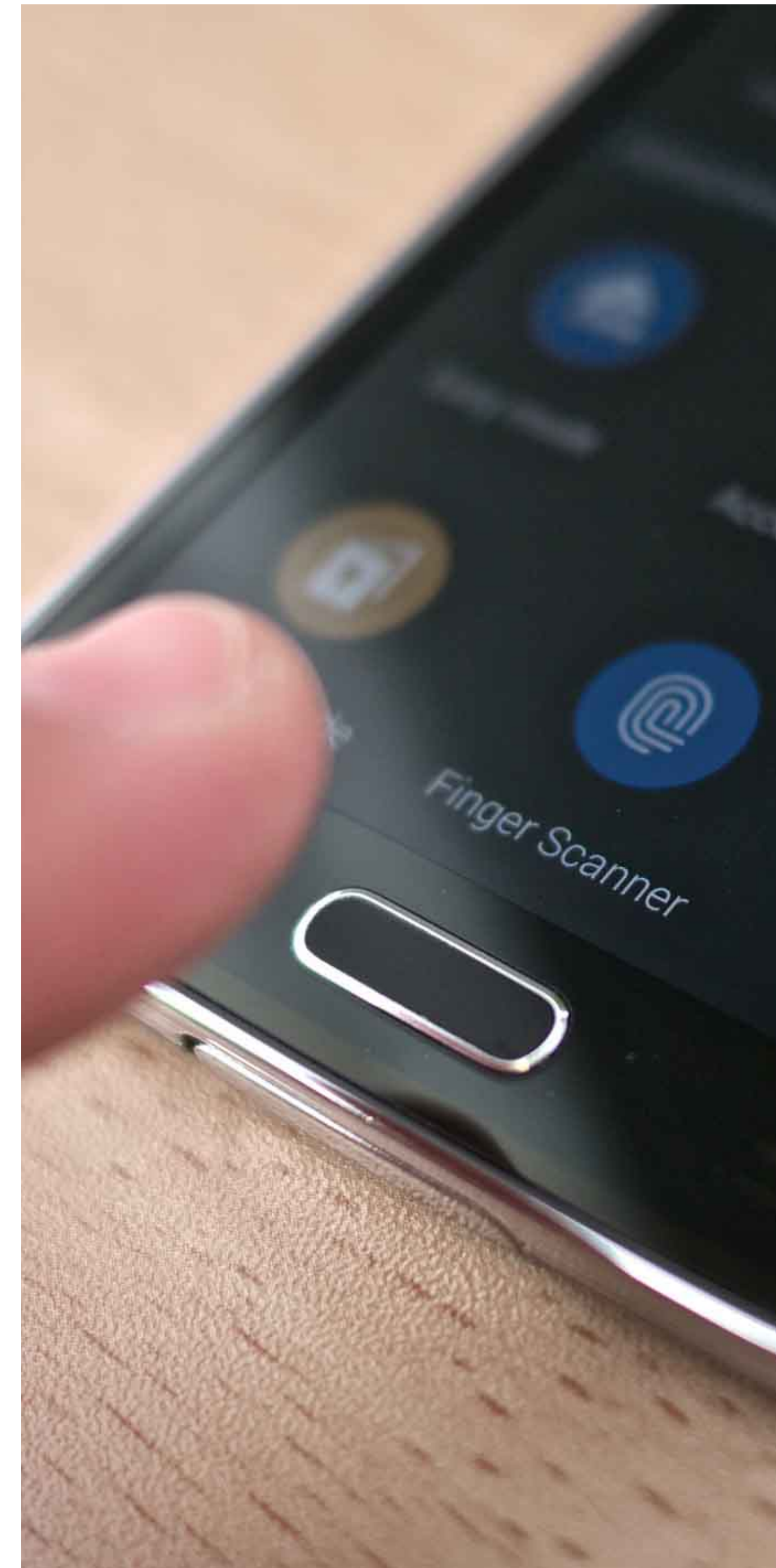
Biometrics

- 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/



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



Voice Recognition

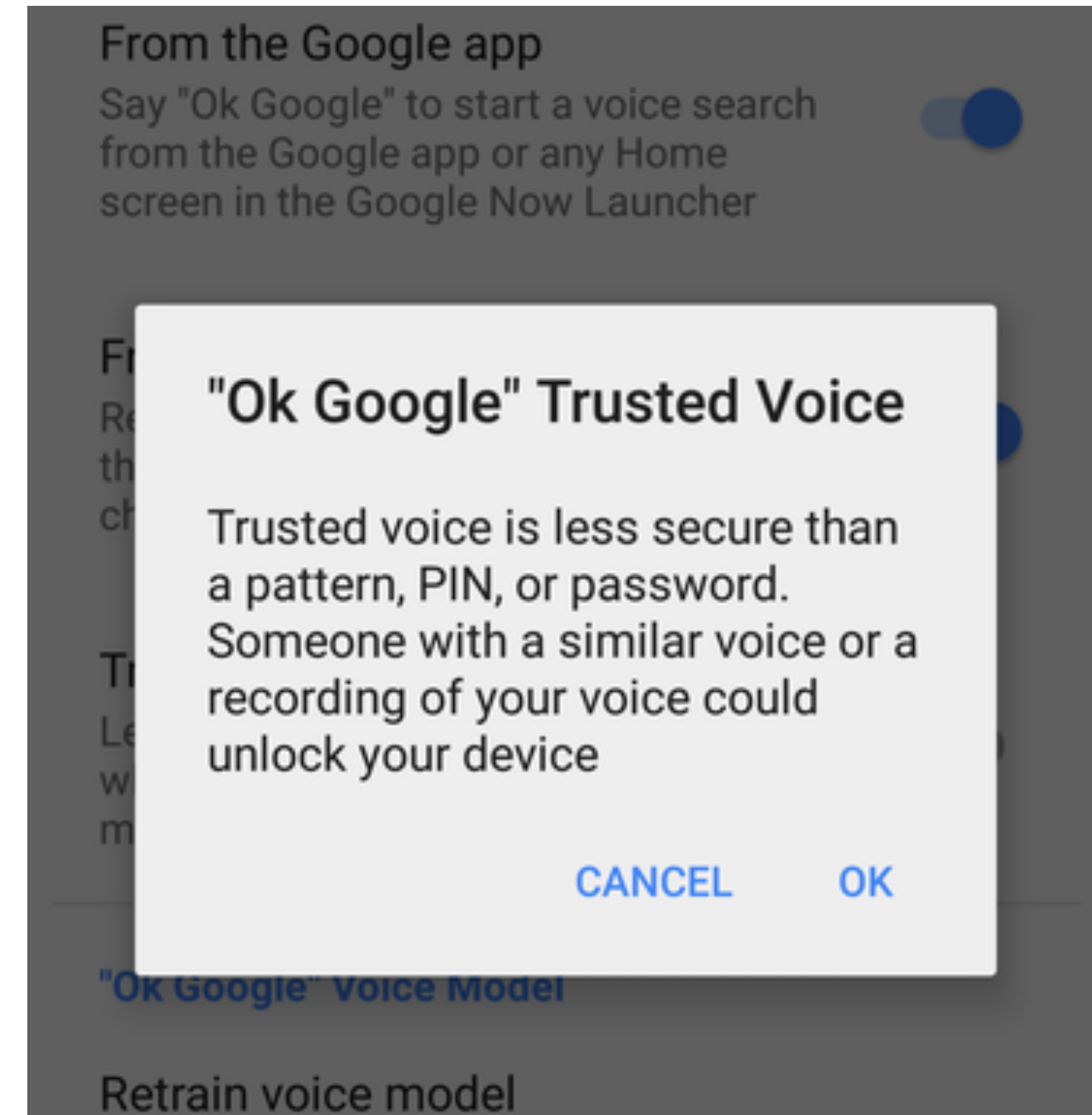
- Secure?
 - Very much depends on the implementation

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

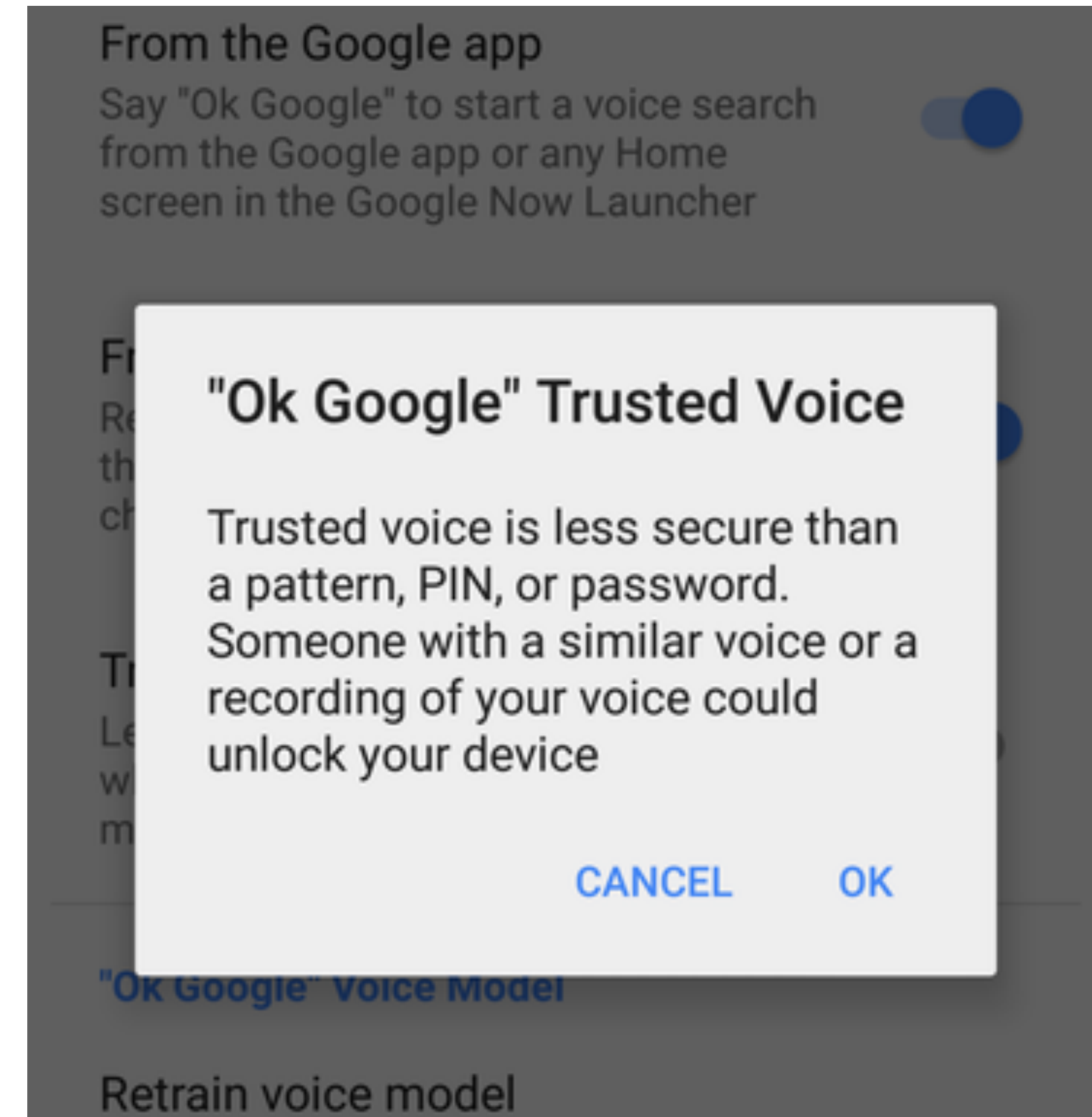
Voice Recognition

- 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

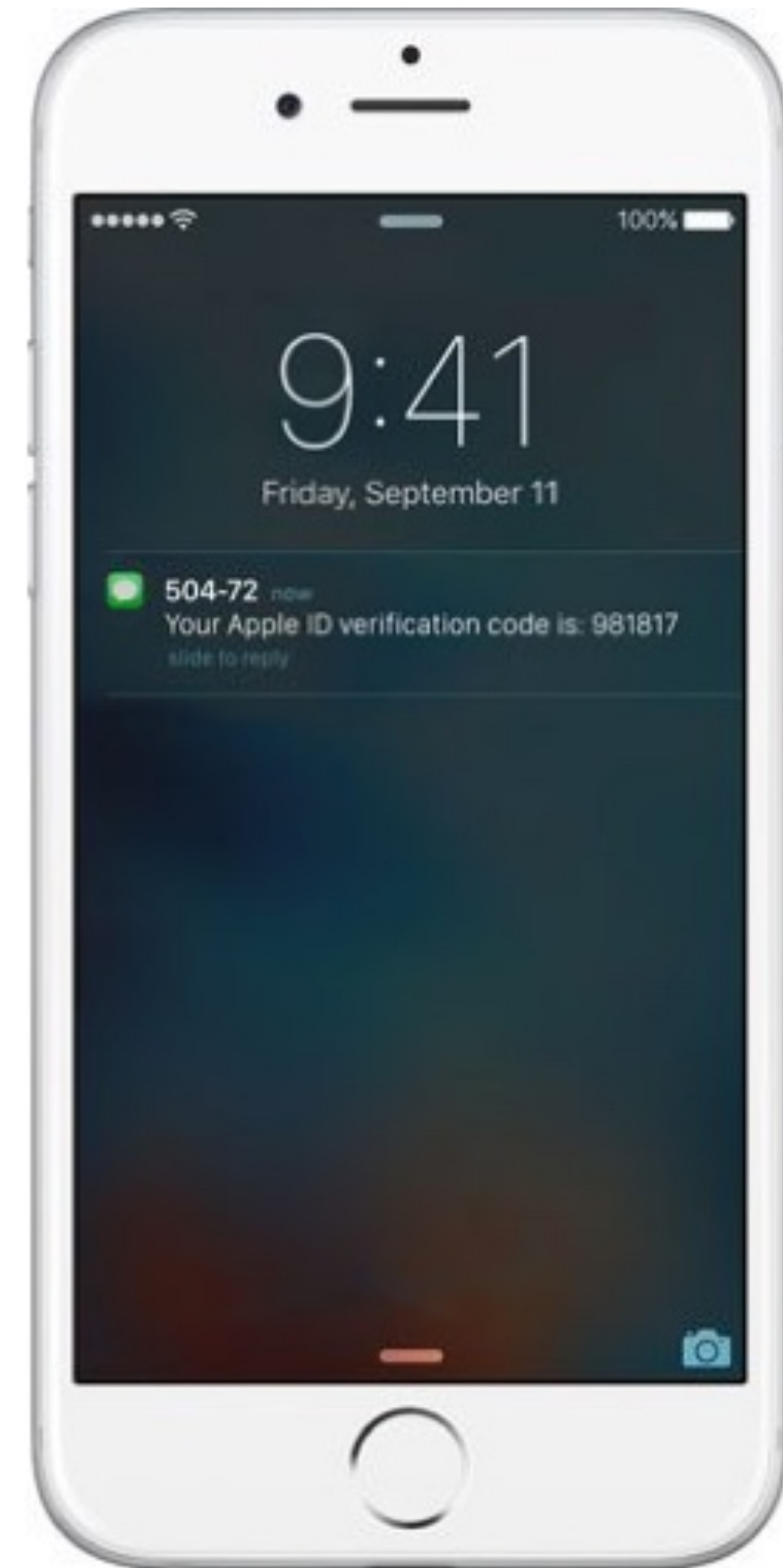
- Biometrics are immutable
 - You are the password, and you can't change
 - Unless you plan on undergoing plastic surgery?
- Once compromised, there is no reset
 - Passwords and tokens can be changed
- Example: the Office of Personnel Management (OPM) breach
 - US gov agency responsible for background 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

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?



SMS Two Factor

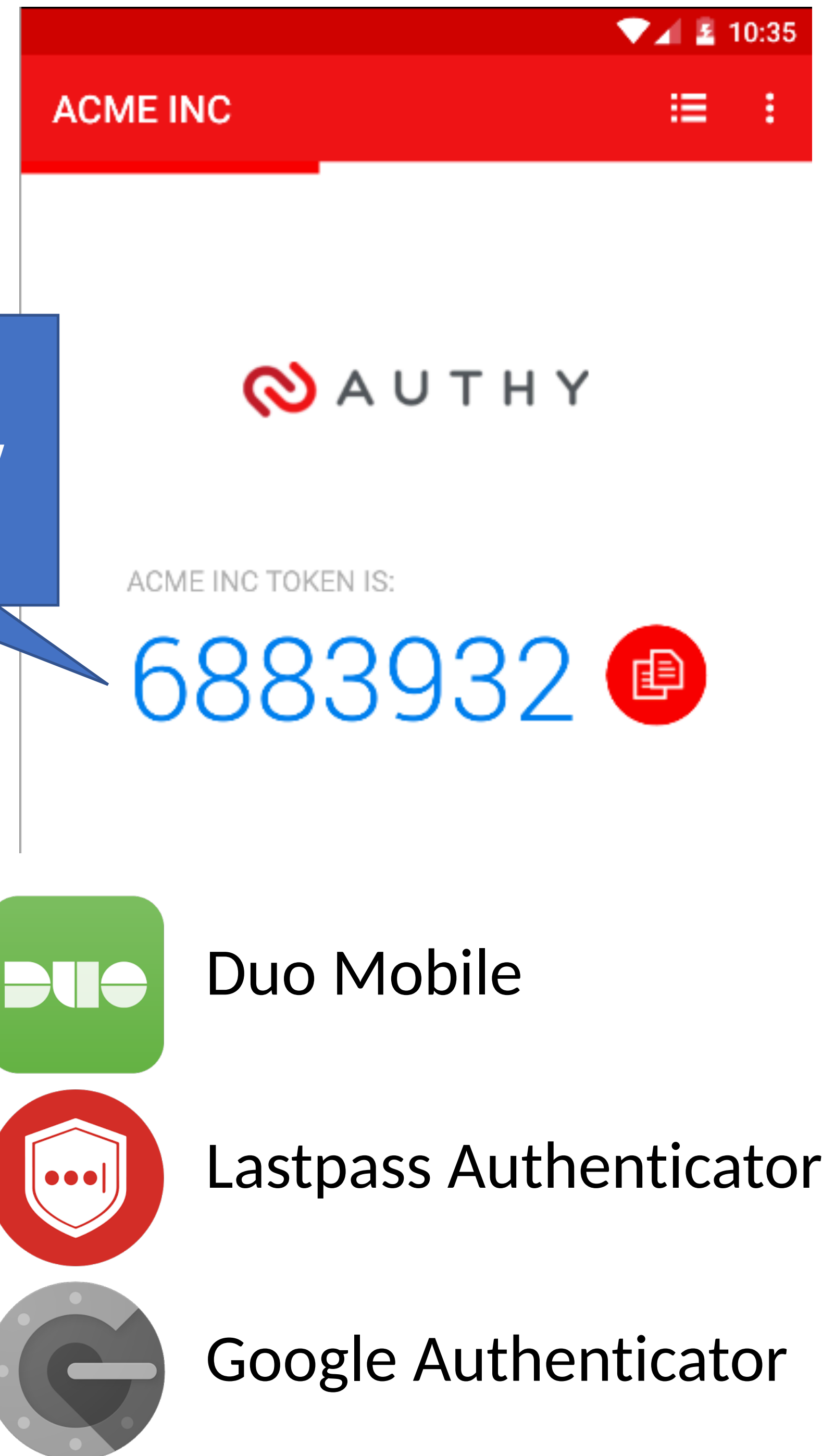
- Relies on your phone number as the second factor
 - Key assumption: only your phone should receive SMS sent to your number
- SMS two factor is deprecated. Why?
- Social engineering the phone company
 1. Call and pretend to be the victim
 2. Say “I got a new SIM, please activate it”
 3. If successful,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



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



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

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



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>

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

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

Specially formatted
SHA1-based signature

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

Secret Sharing for TOTP

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.



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

[USE SMS INSTEAD](#)

[CANCEL](#)

[NEXT STEP](#)

[REFER A FRIEND](#)

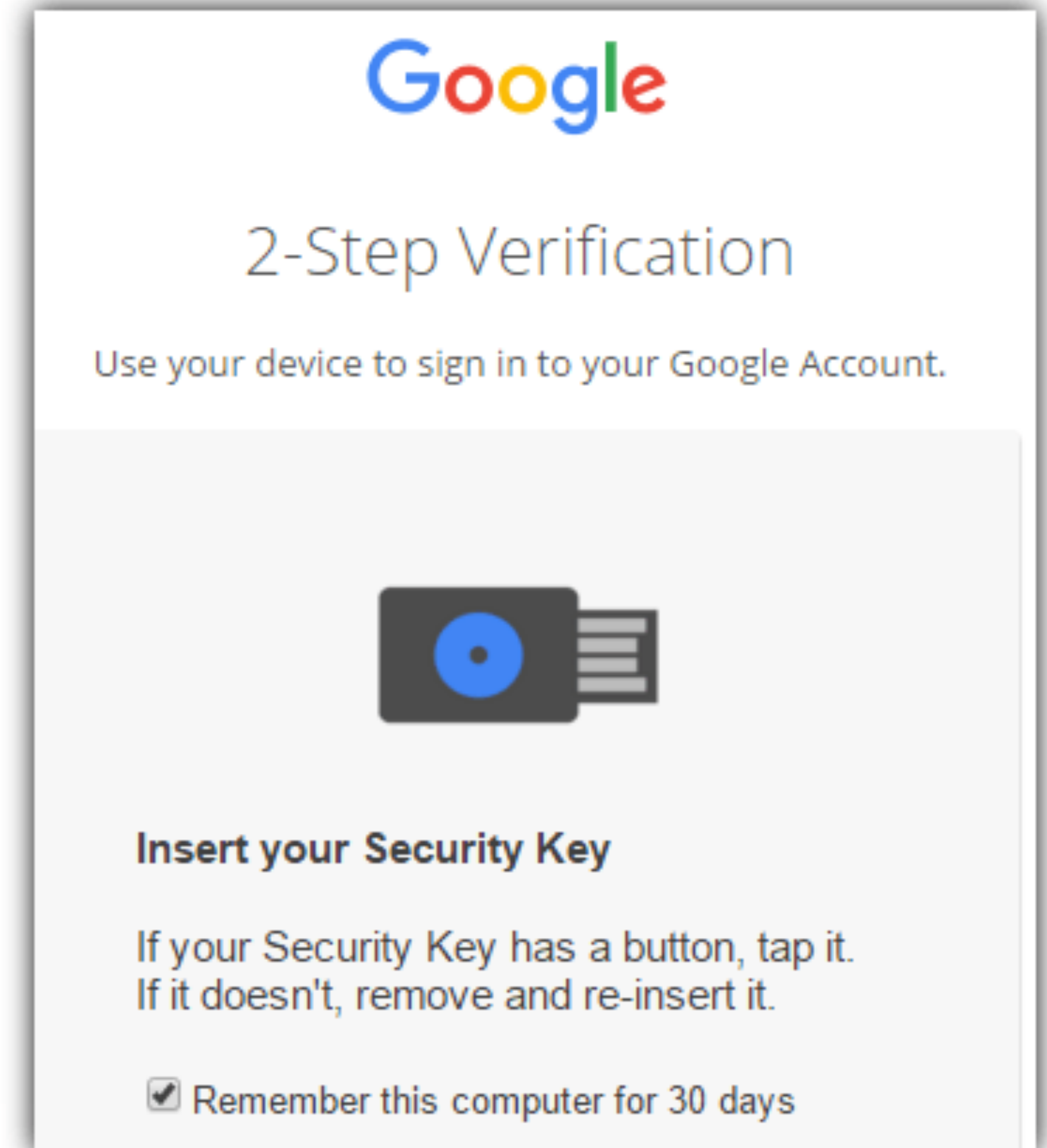
Hardware Two Factor

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



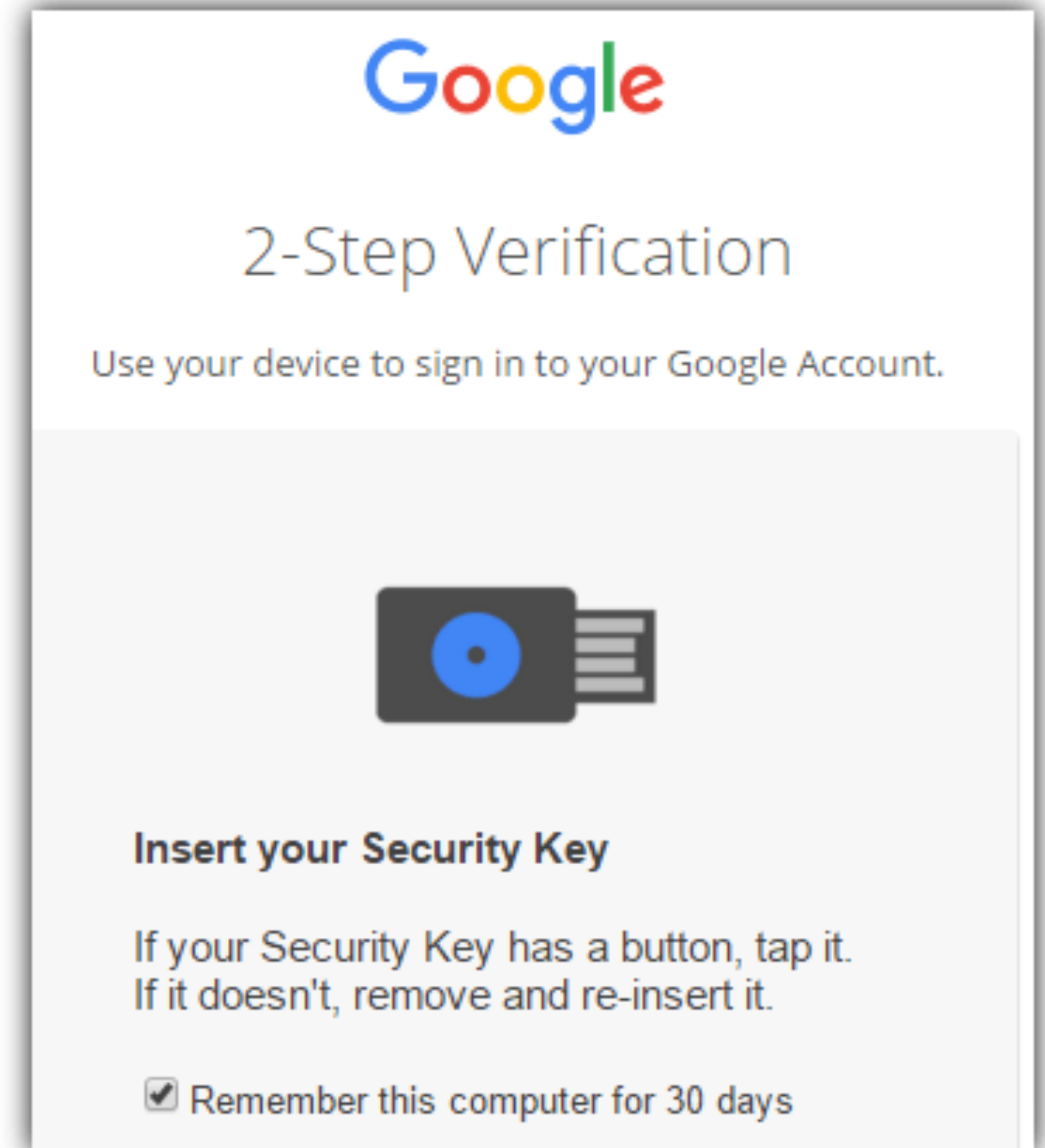
Universal 2nd Factor (U2F)

- 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 ;)



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 2nd 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?

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

n Mislove:/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.yUMej3XSNAM: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:jYpwYhqqvr3tA: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

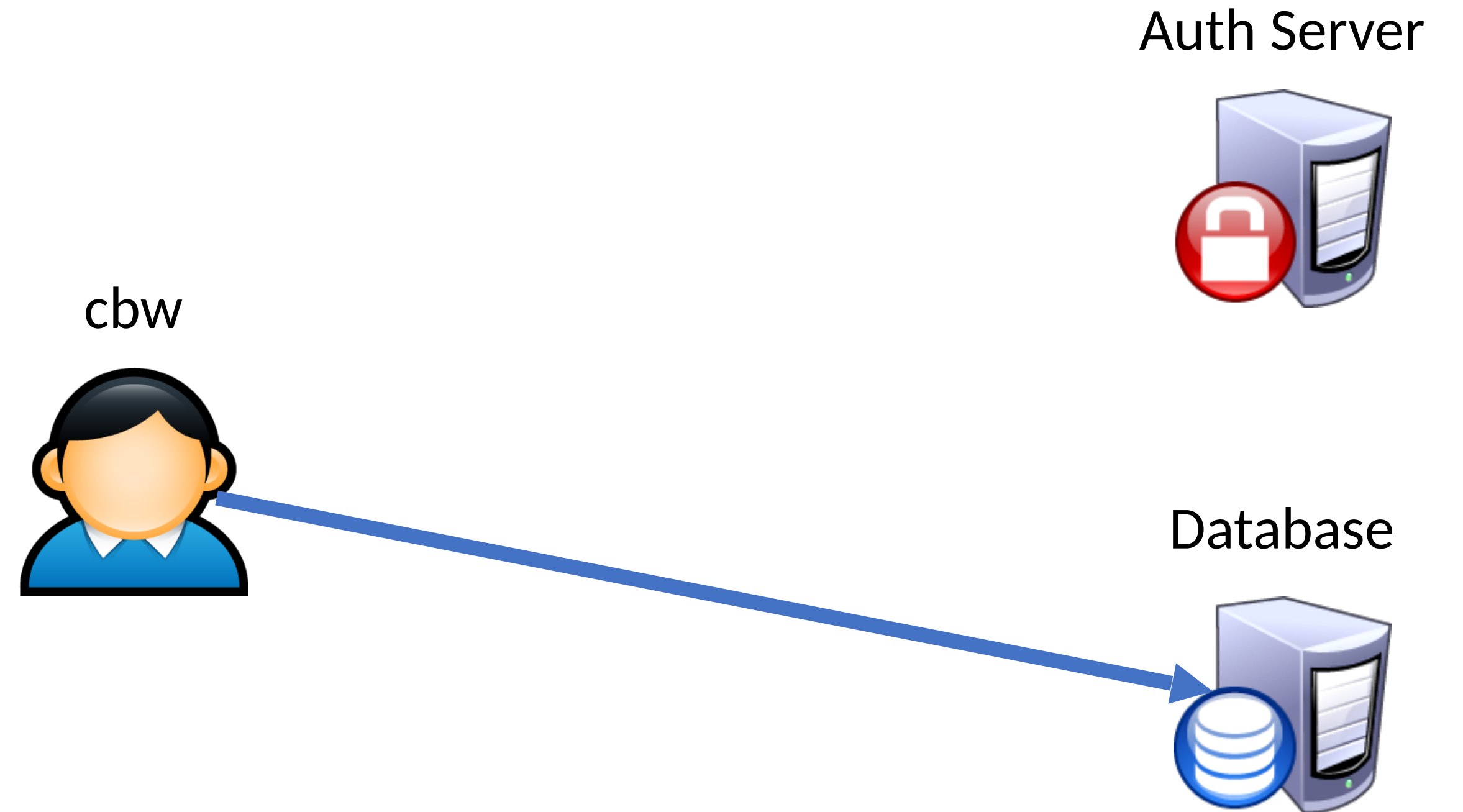


Database



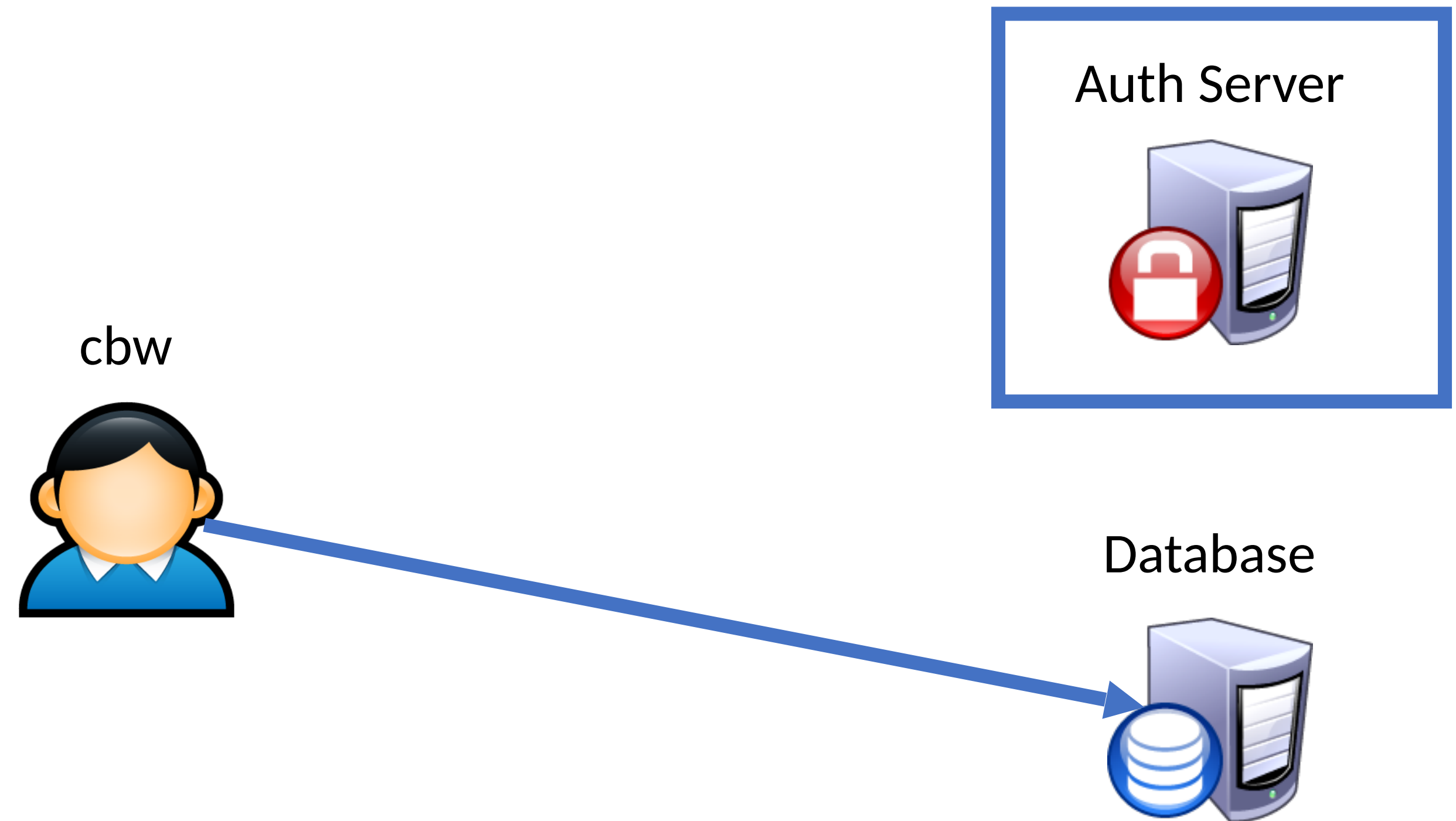
Distributed Authentication Revisited

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



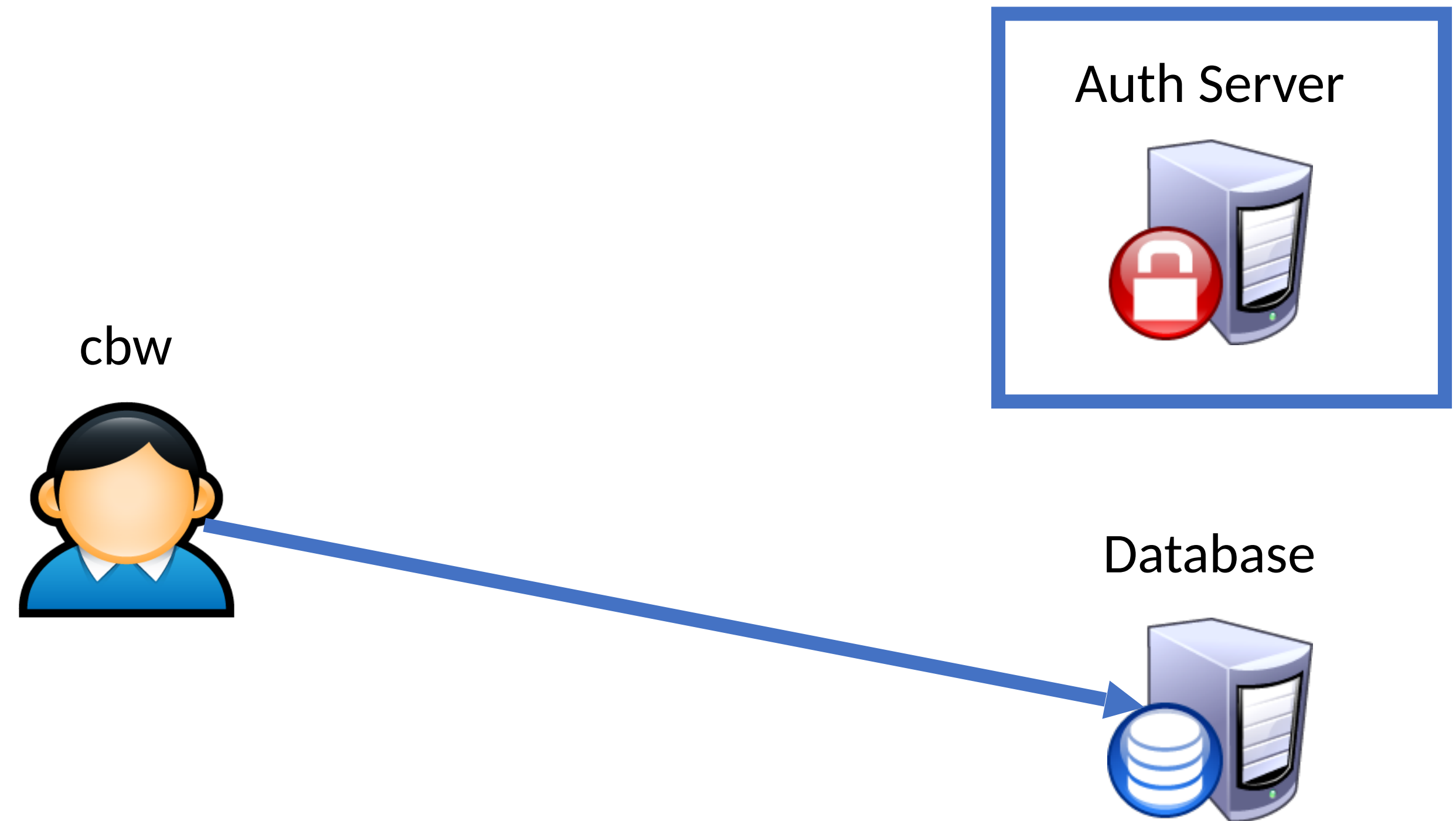
Distributed Authentication Revisited

- 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



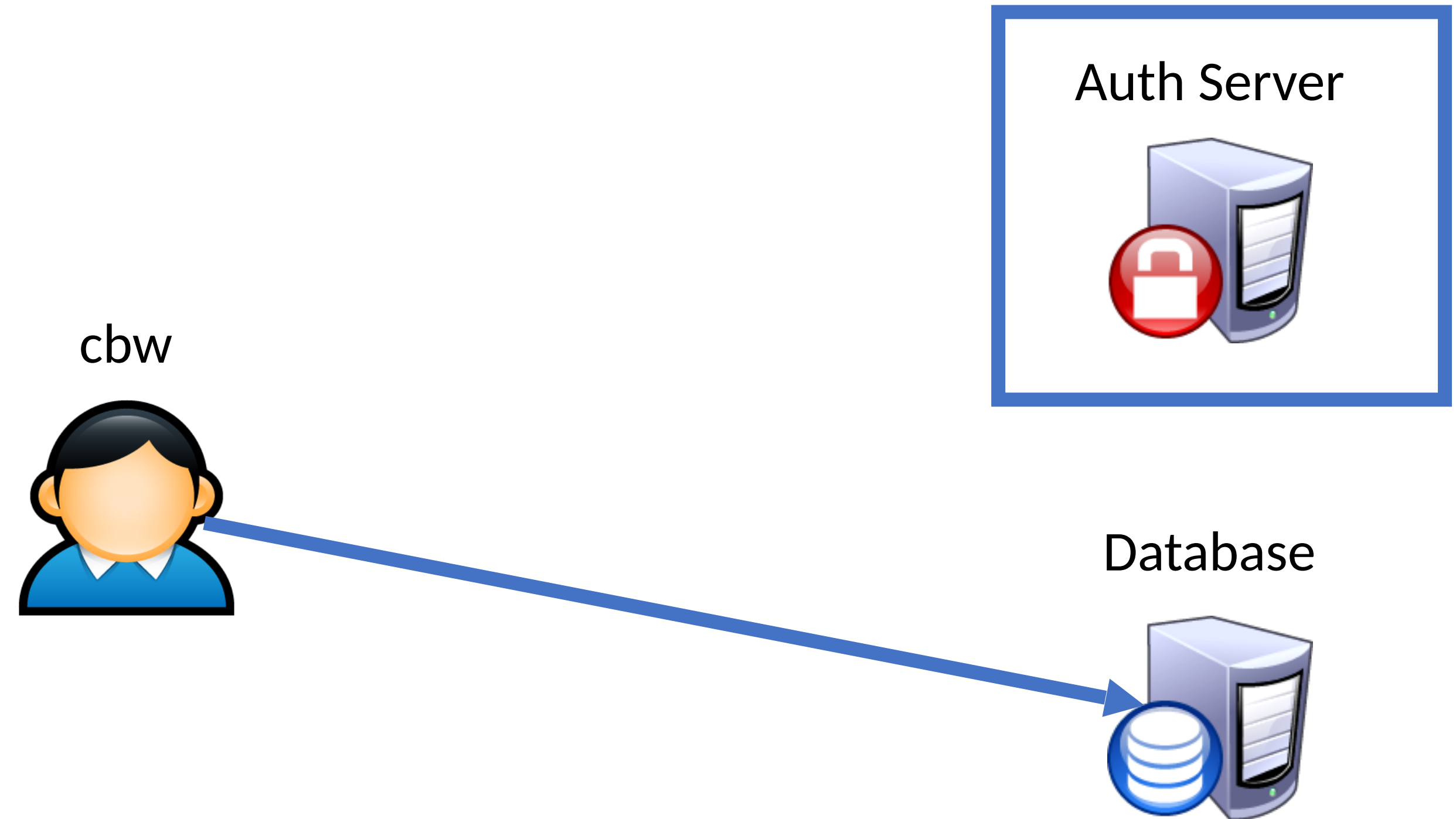
Attacker Goals and Threat Model

- 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

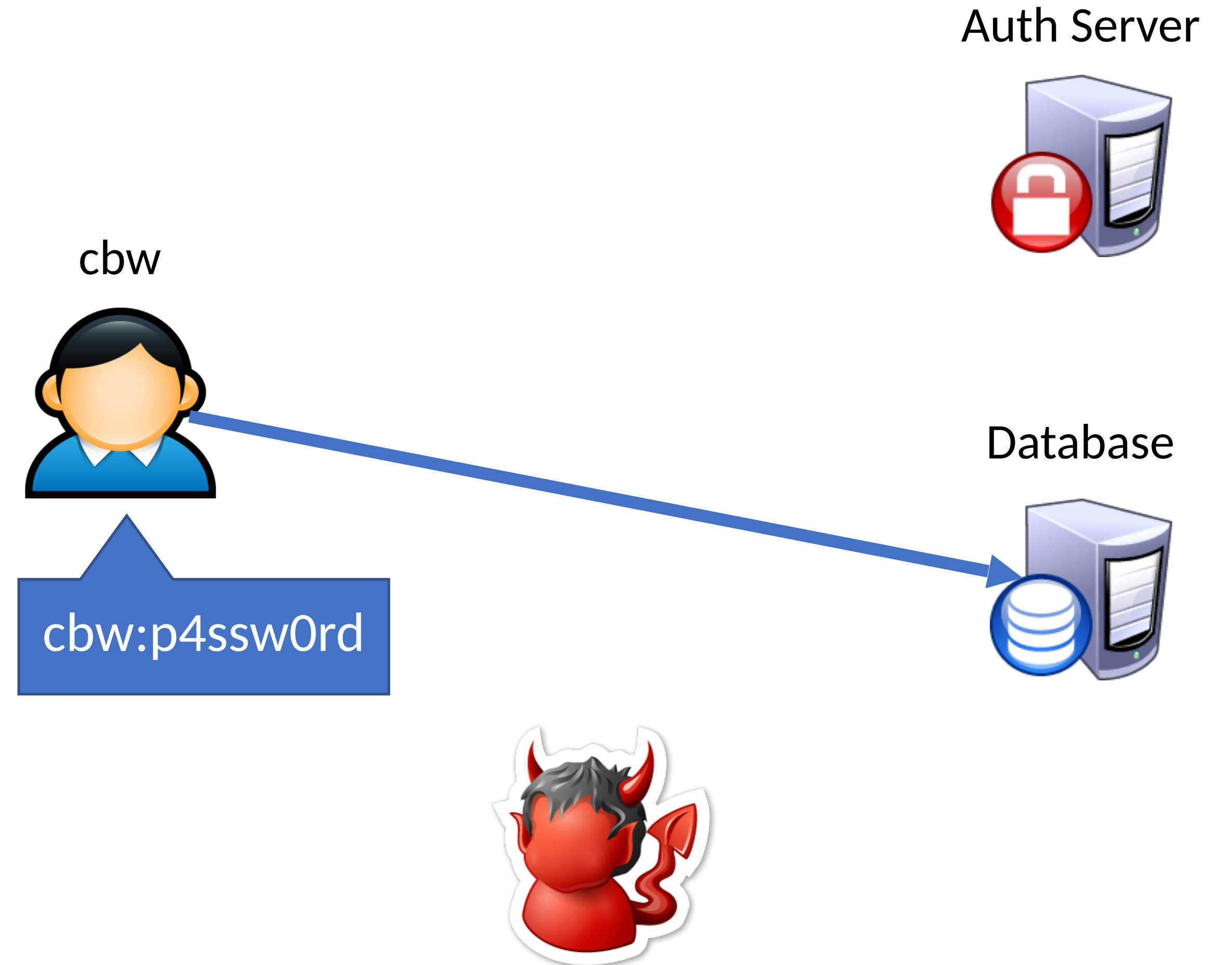


Database



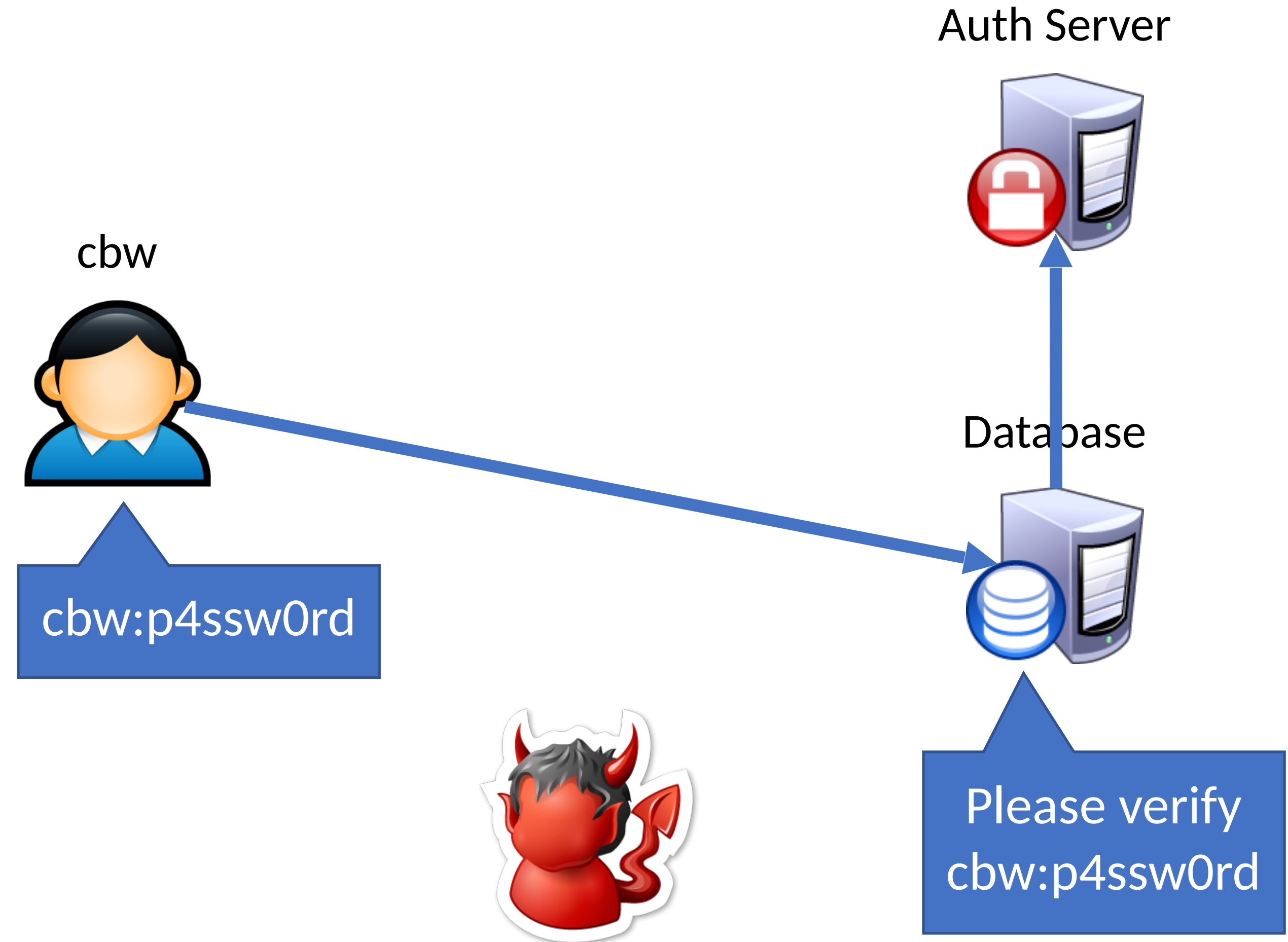
(Bad) Distributed Auth Example

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



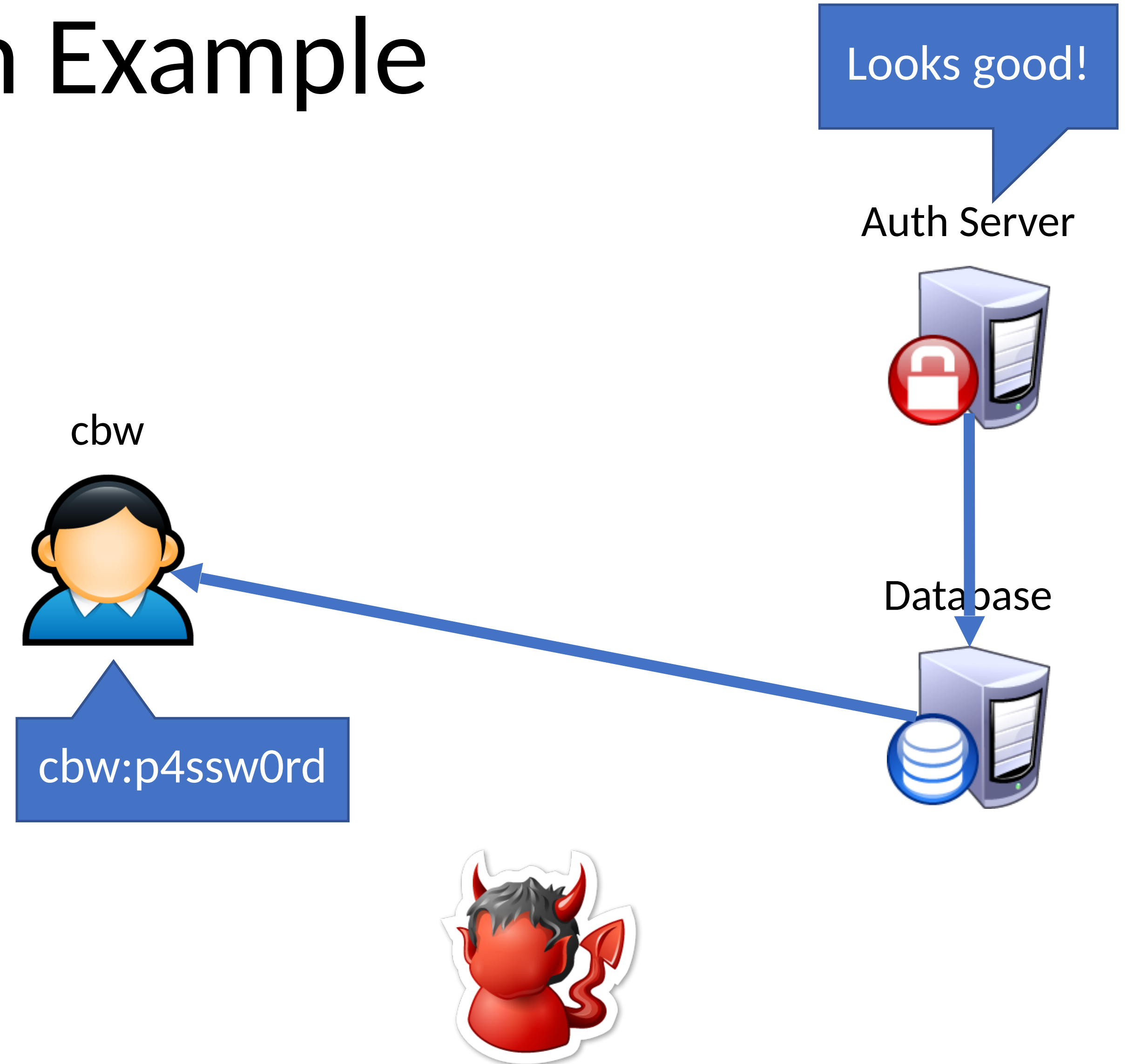
(Bad) Distributed Auth Example

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



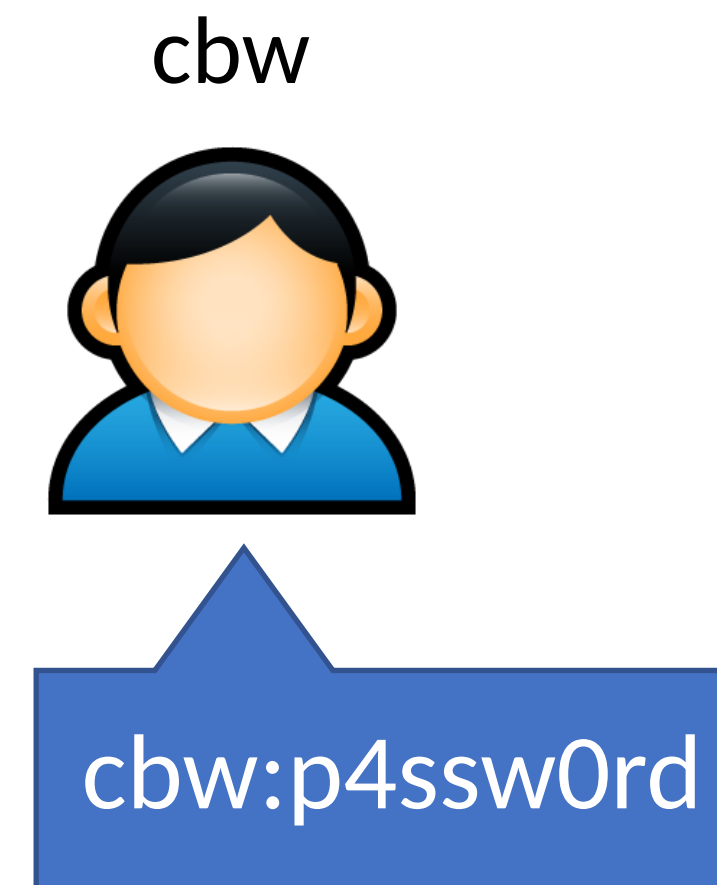
(Bad) Distributed Auth Example

- 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

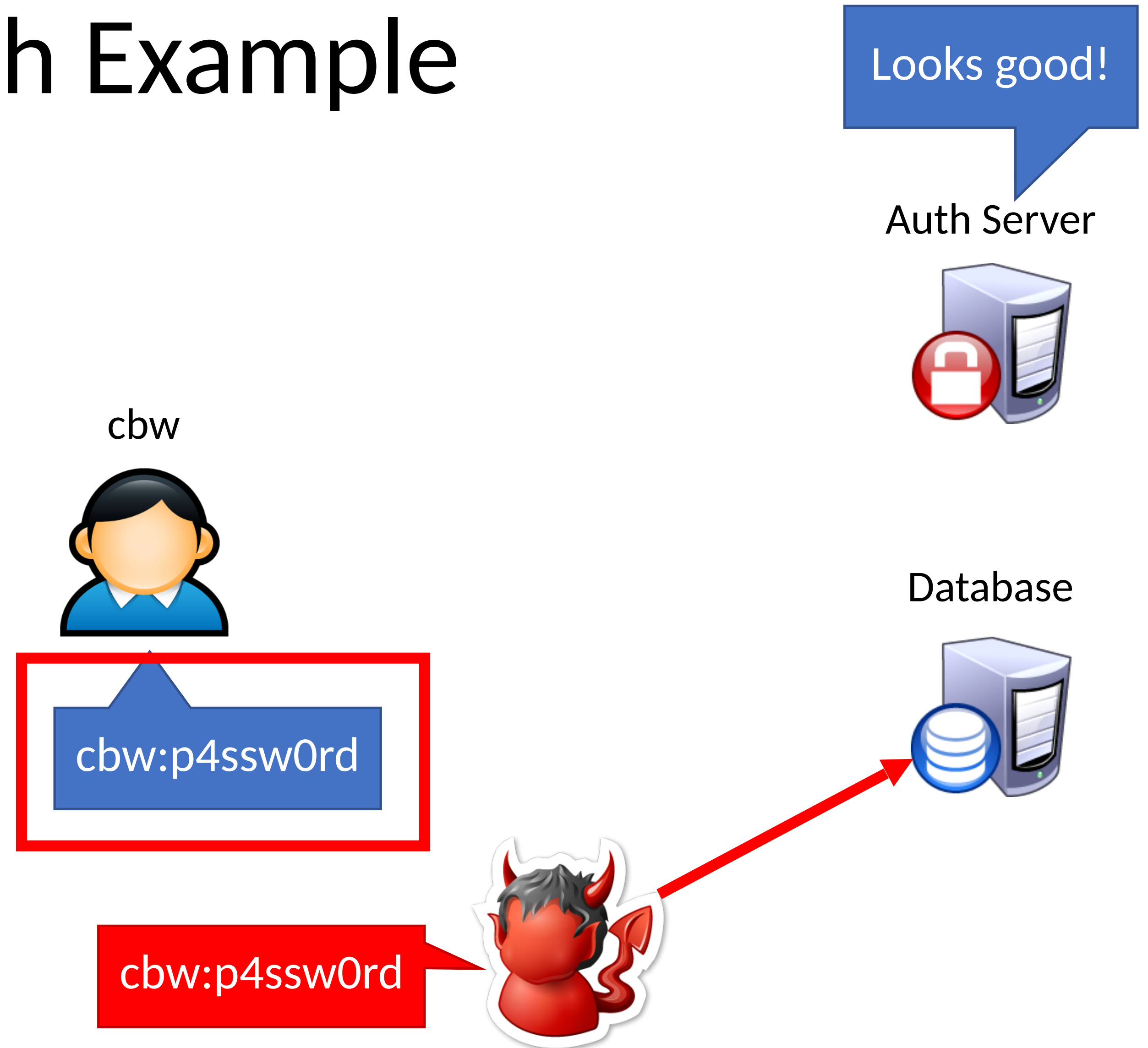


Database



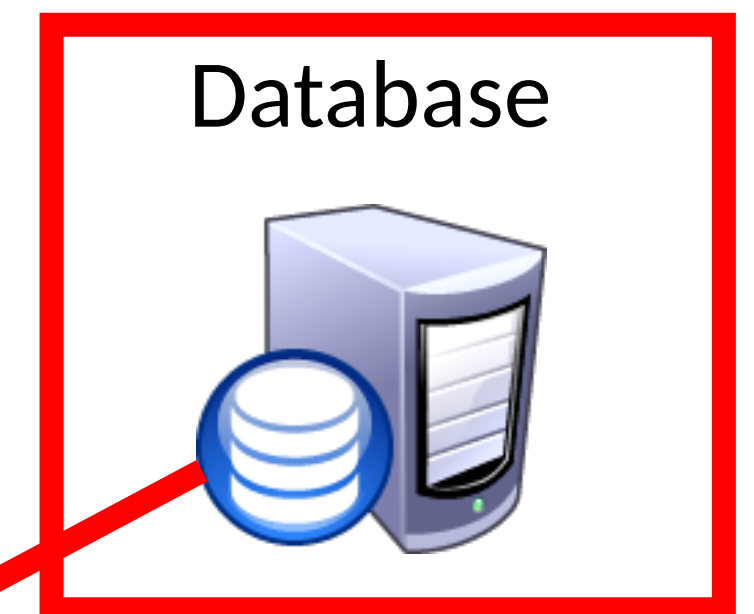
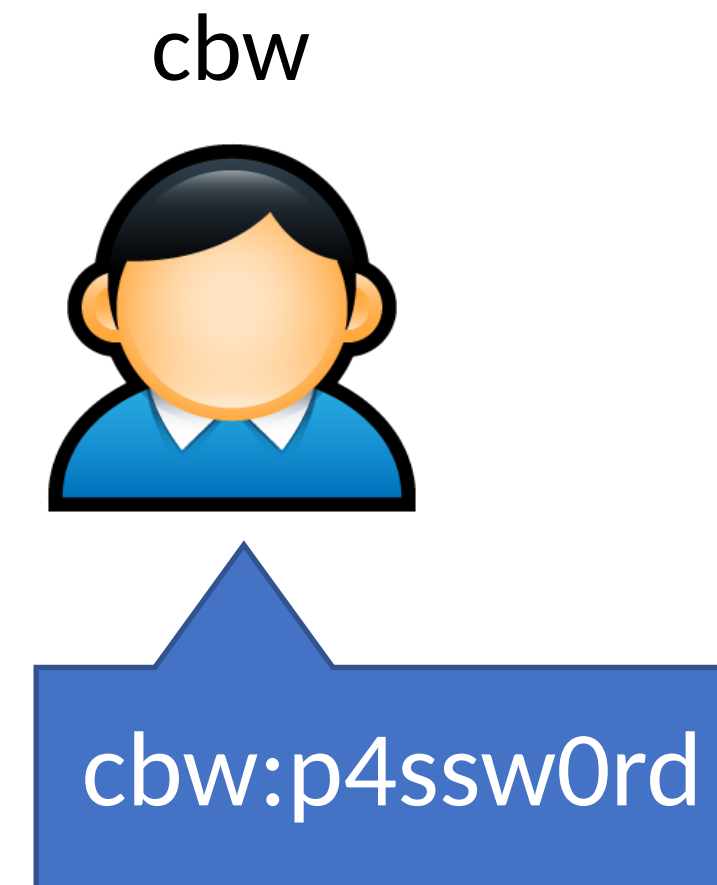
(Bad) Distributed Auth Example

- 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



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

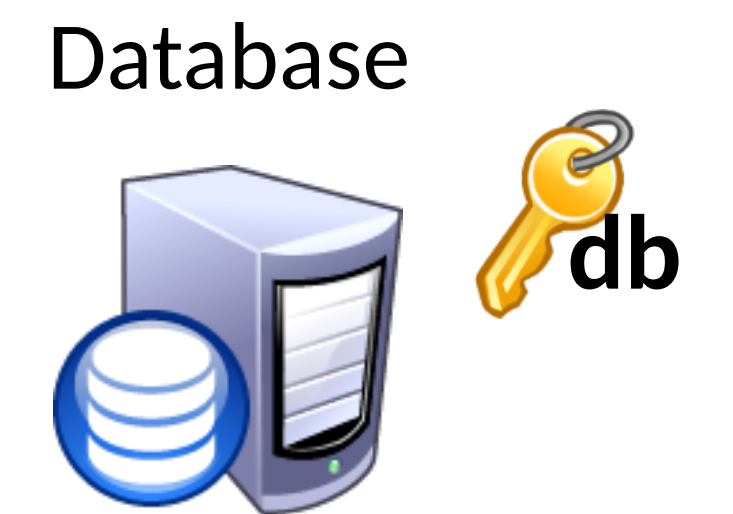
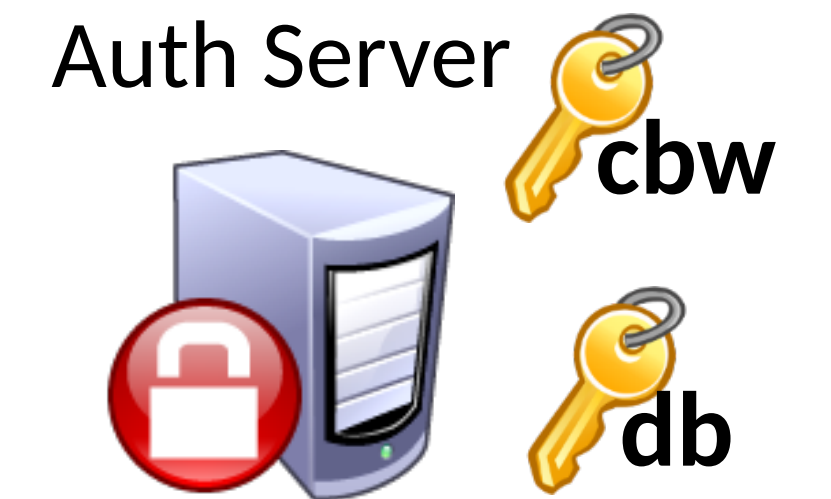
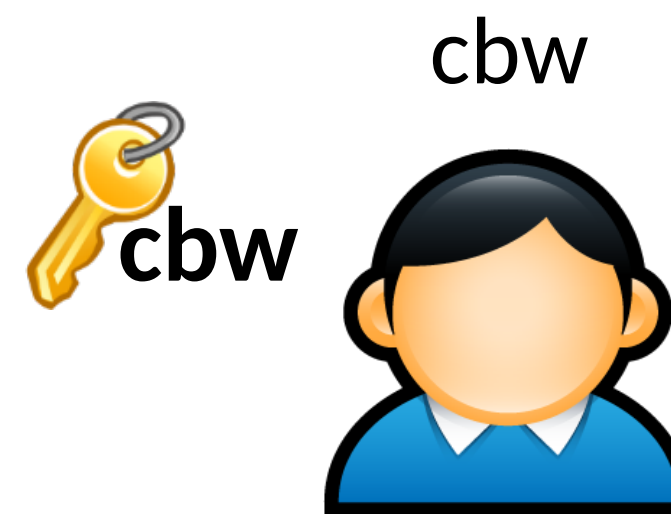
4) $B \rightarrow A: \{N_j\}_{K_{AB}}$

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

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

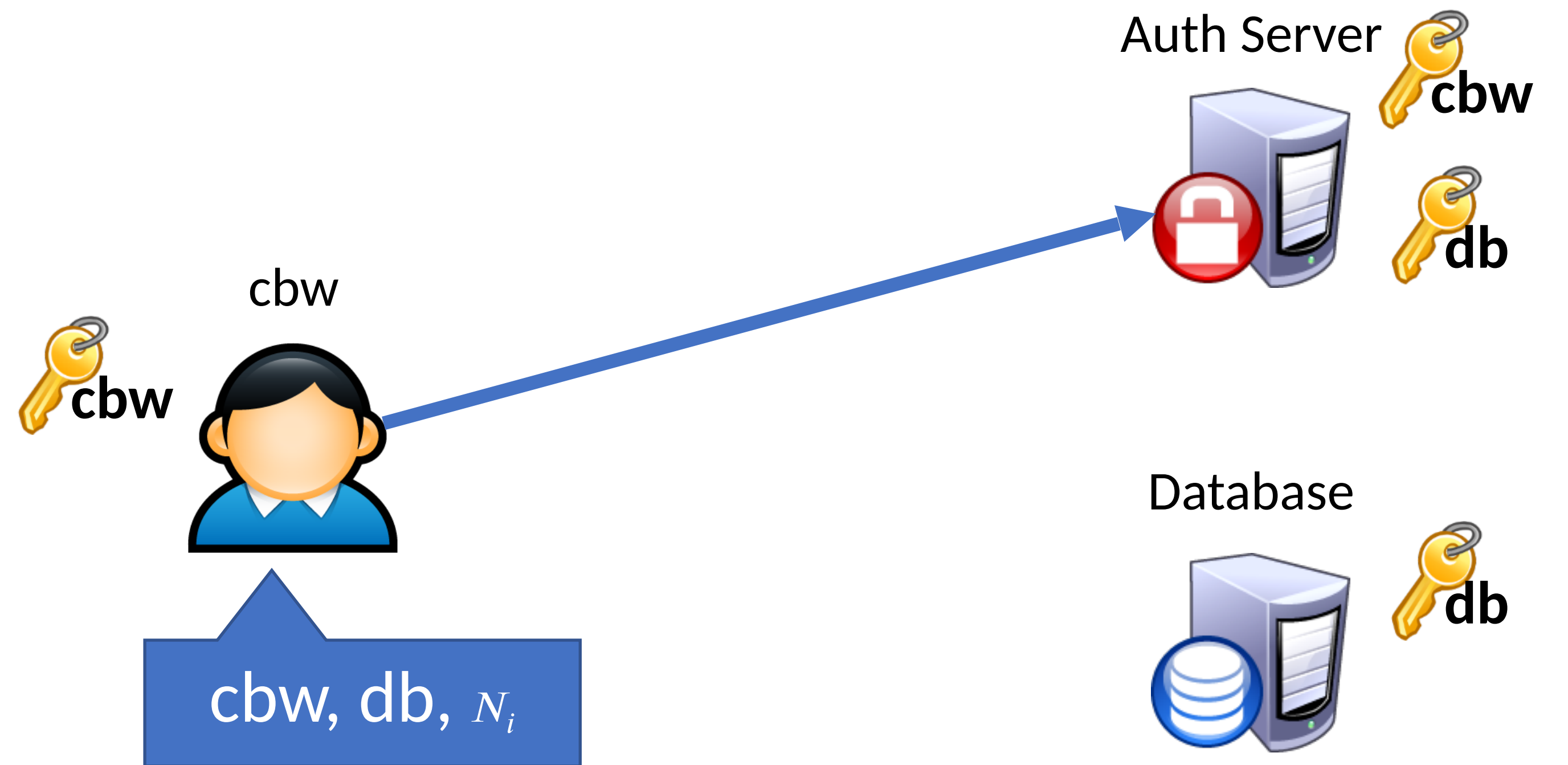
Needham-Schroeder Example

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



Needham-Schroeder Example

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

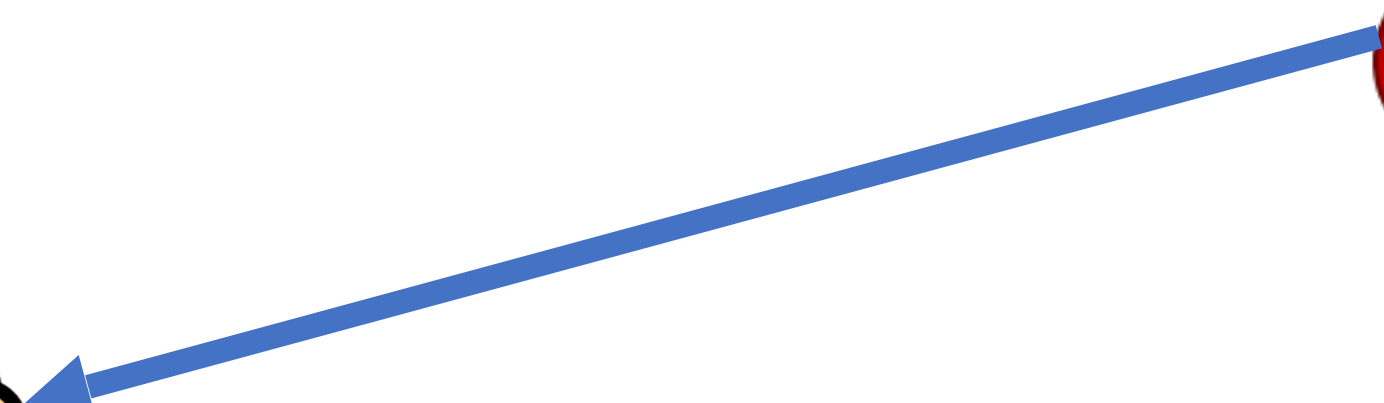
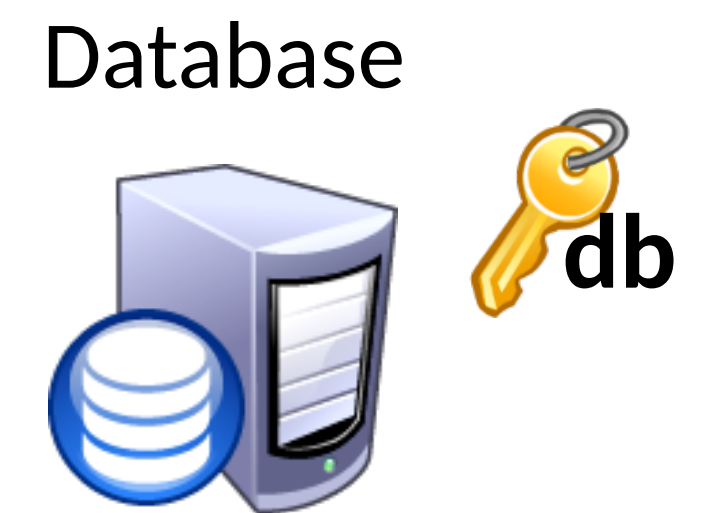
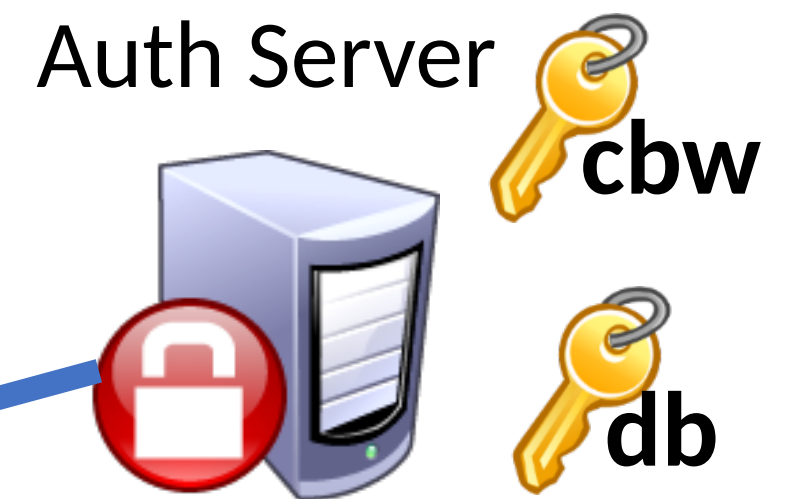
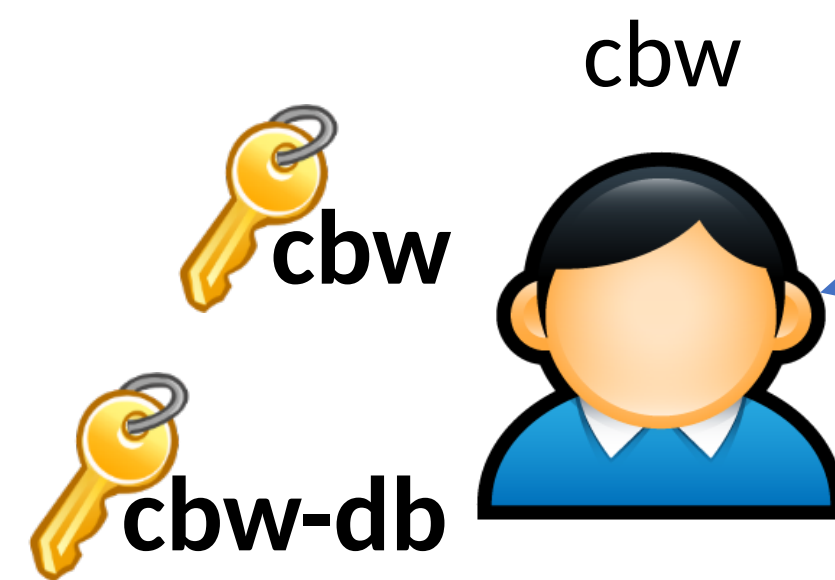


Needham-Schroeder Example

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

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

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

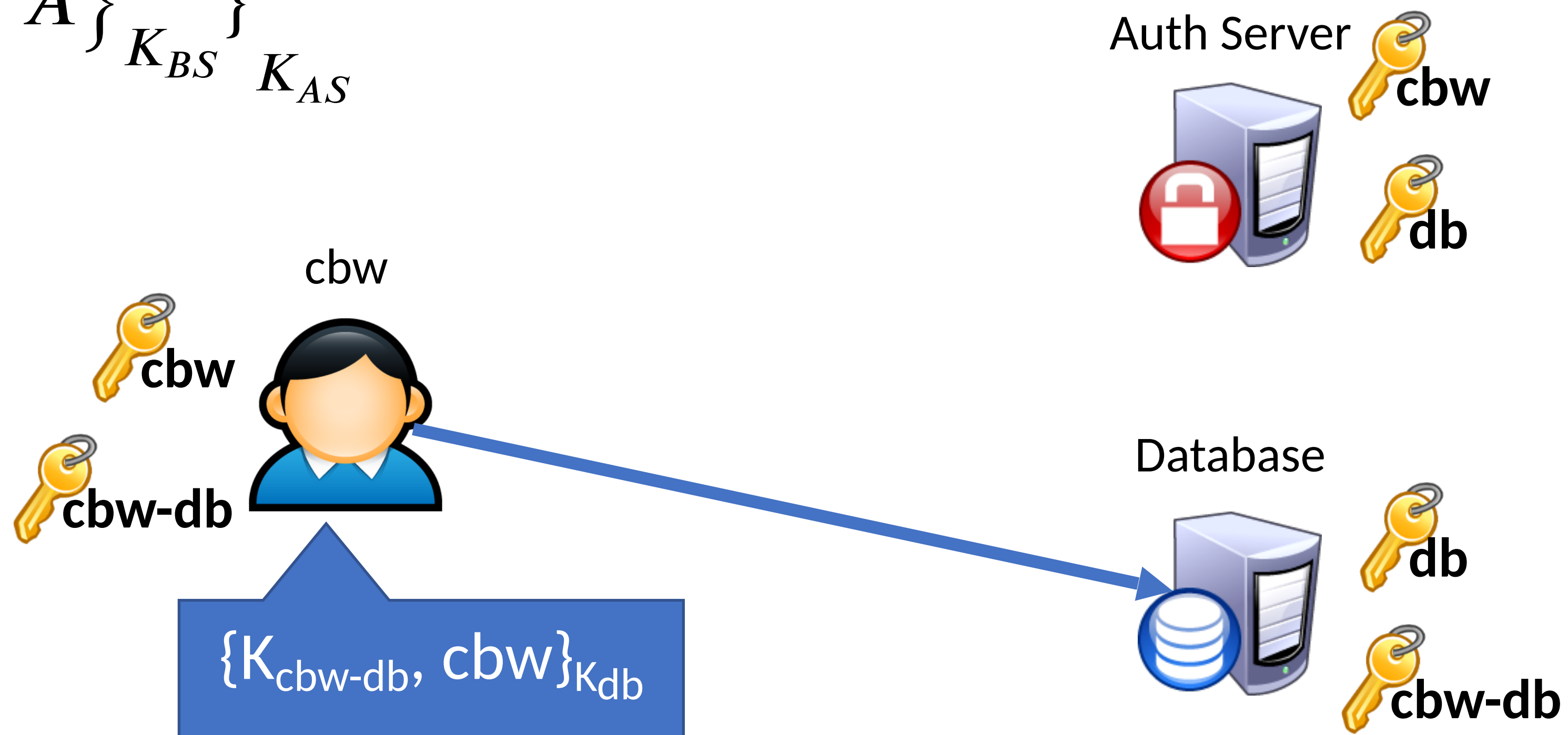


Needham-Schroeder Example

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



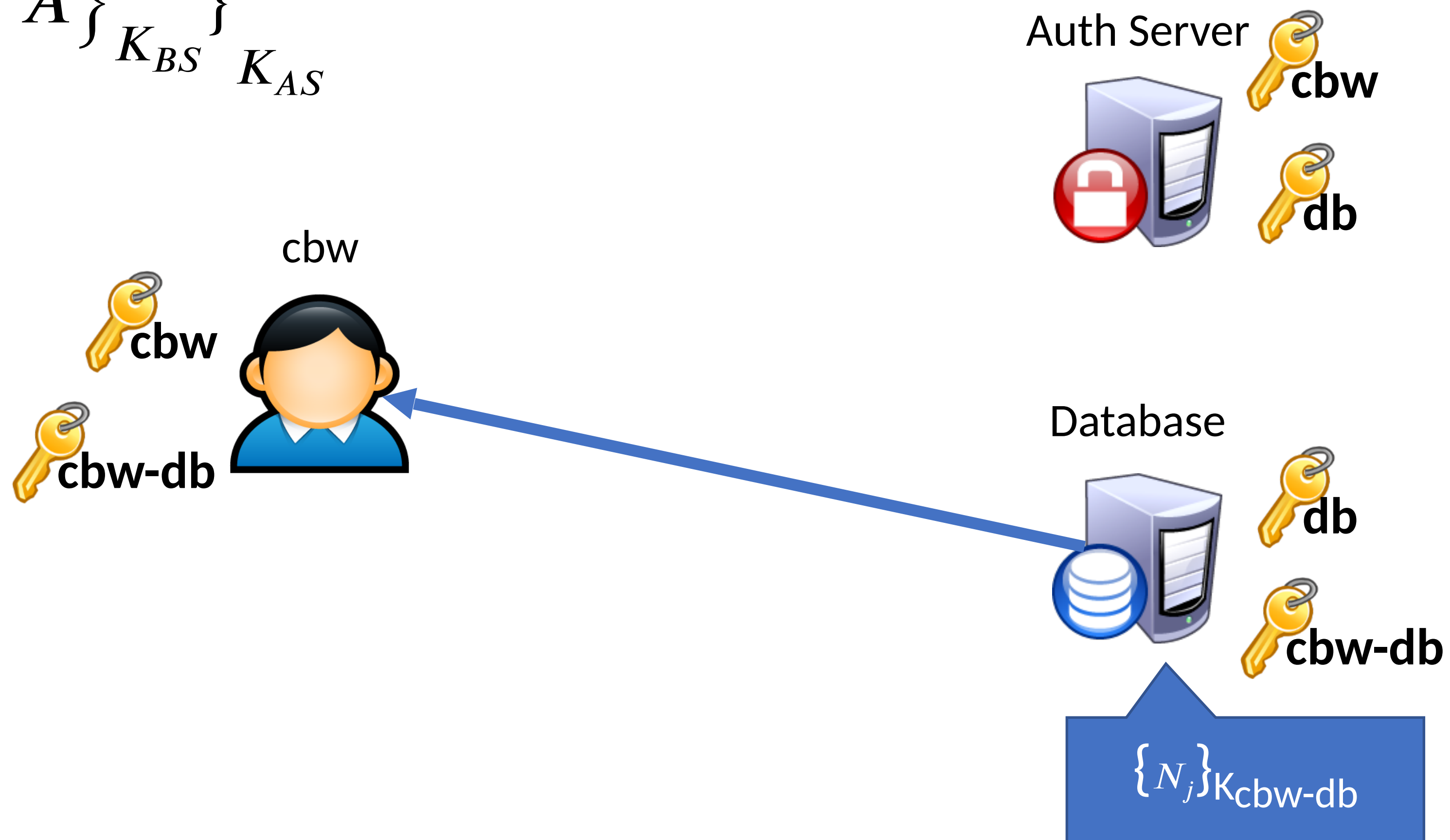
Needham-Schroeder Example

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

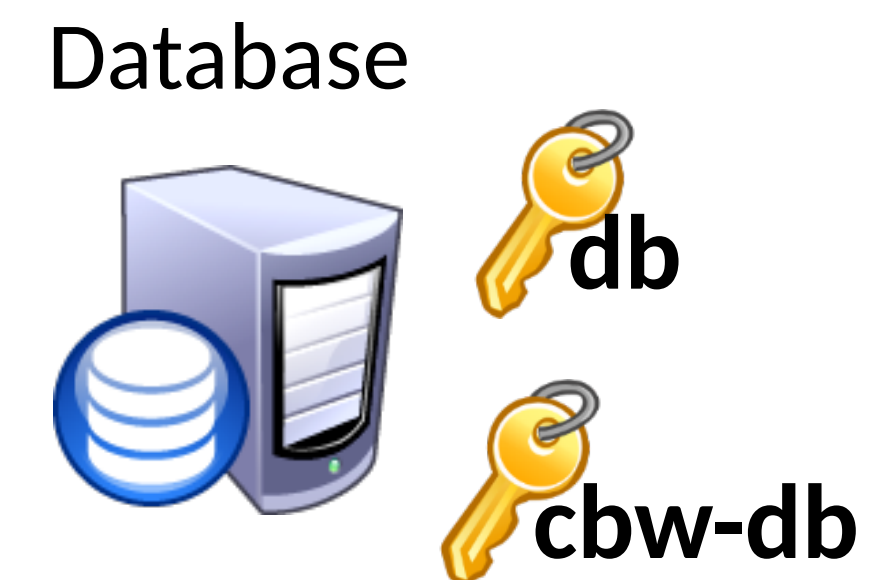
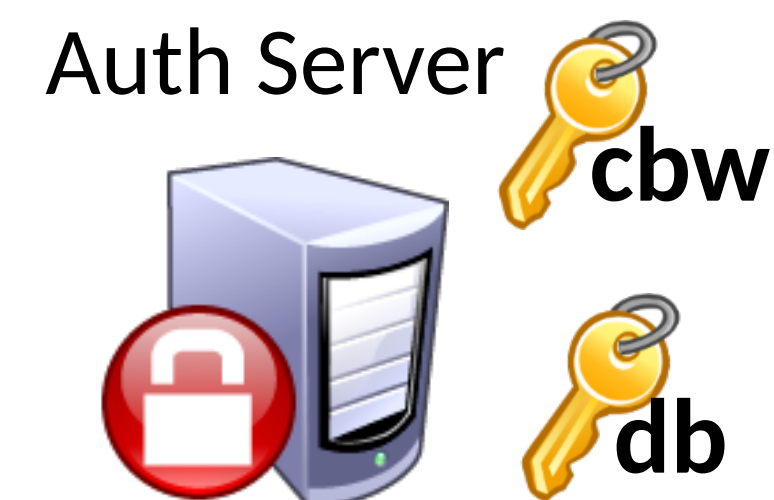
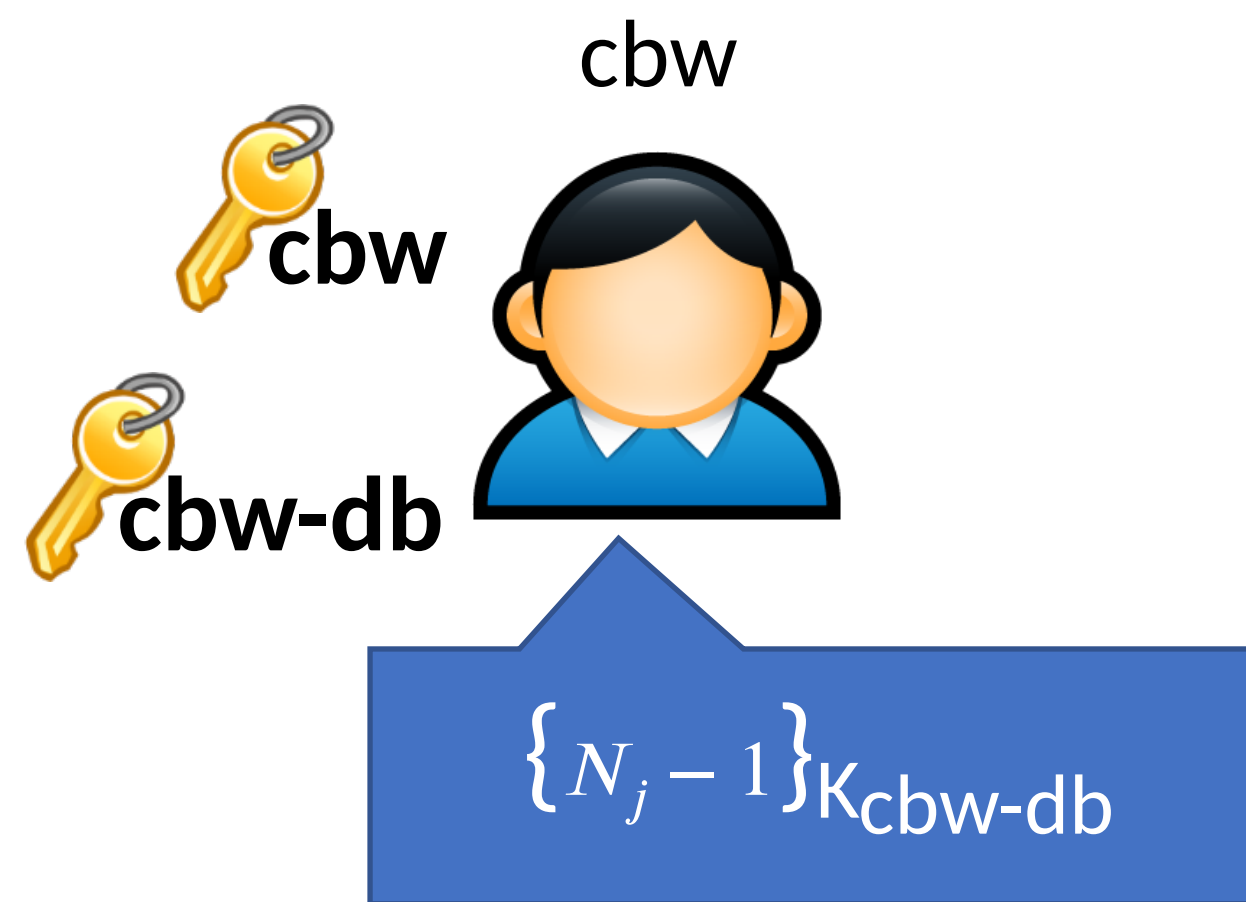
1) $A \rightarrow S: A, B, N_i$

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

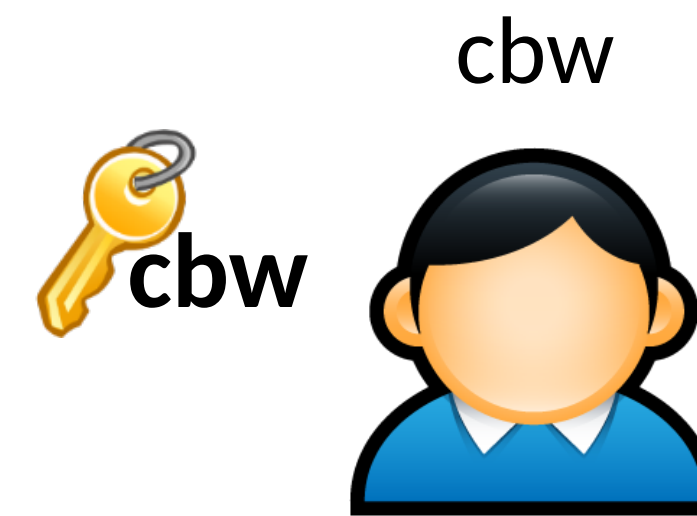
3) $A \rightarrow B: \{K_{AB}, A\}_{K_{BS}}$

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

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



Attacking Needham-Schroeder



Auth Server

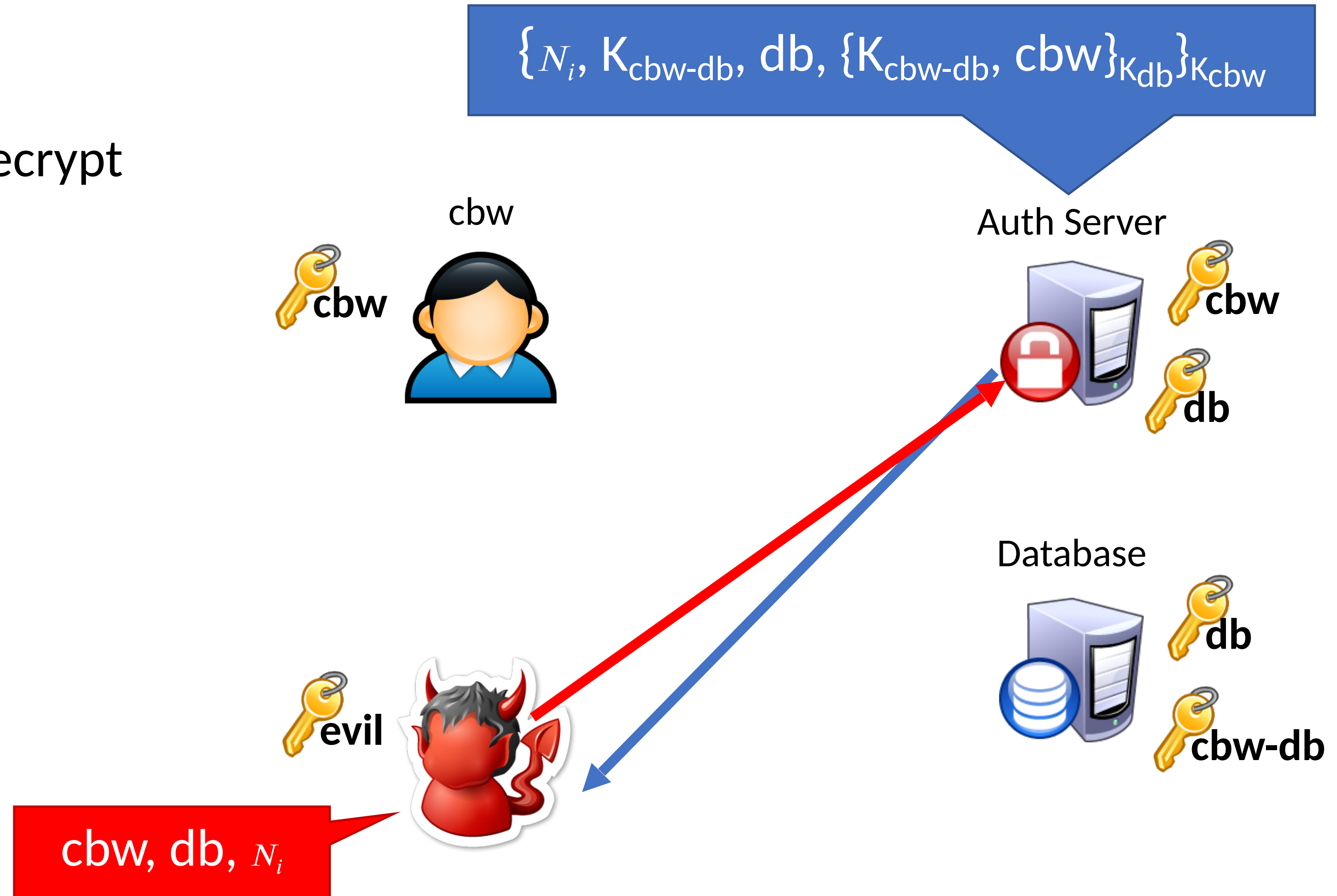


Database



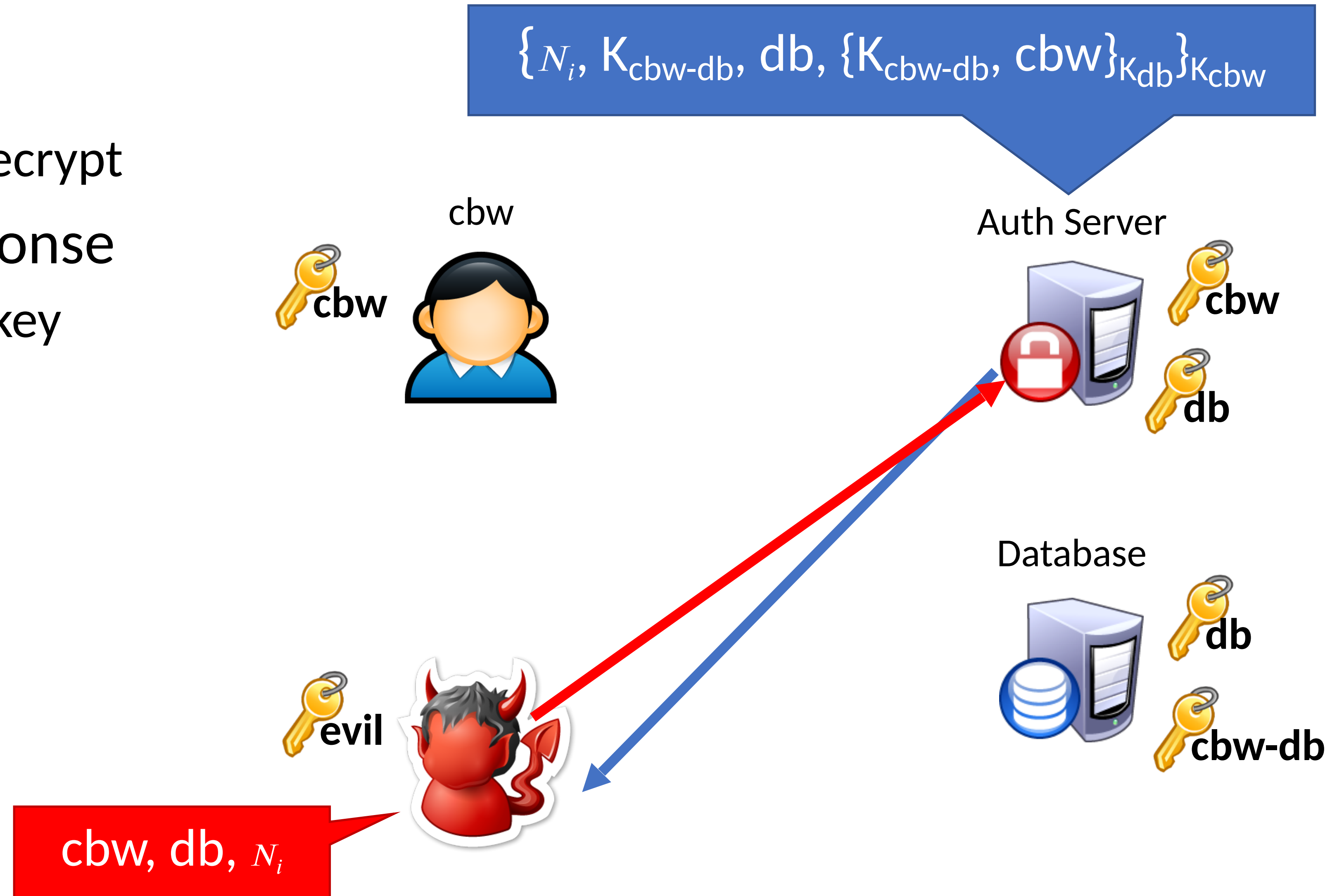
Attacking Needham-Schroeder

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



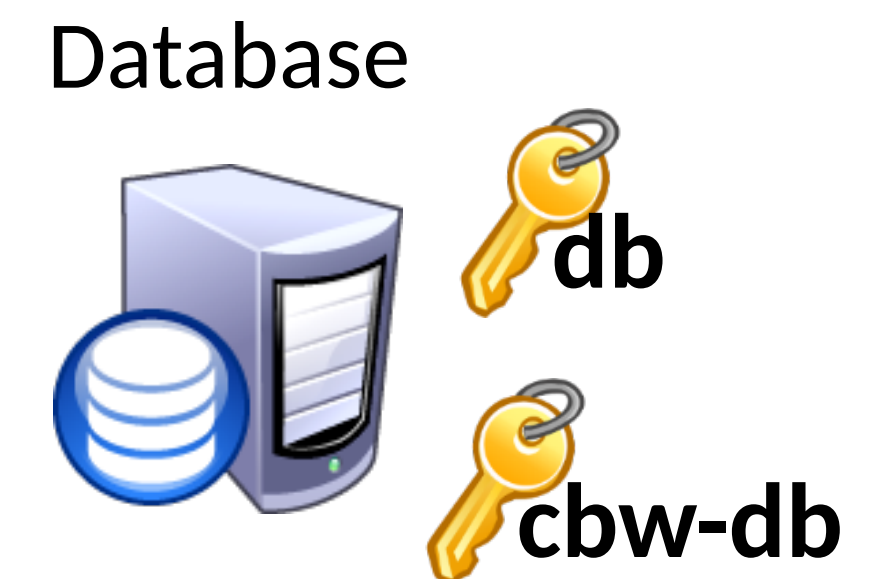
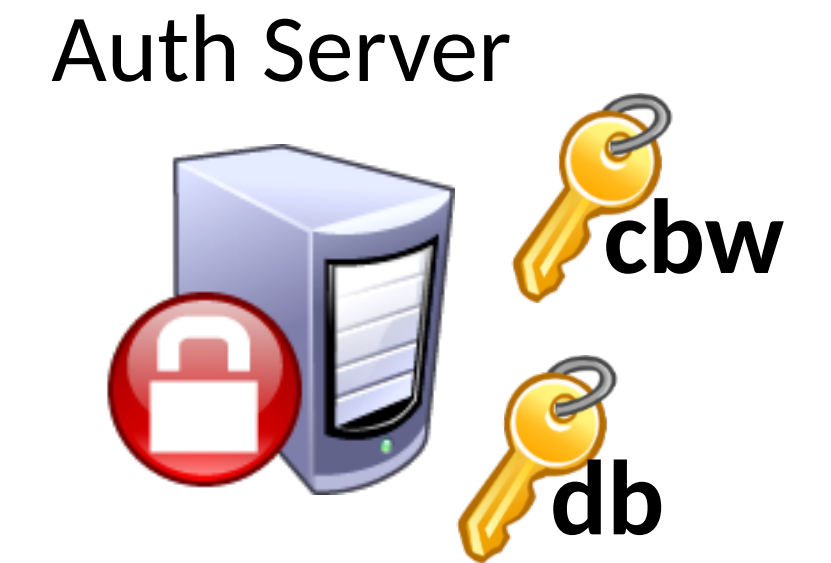
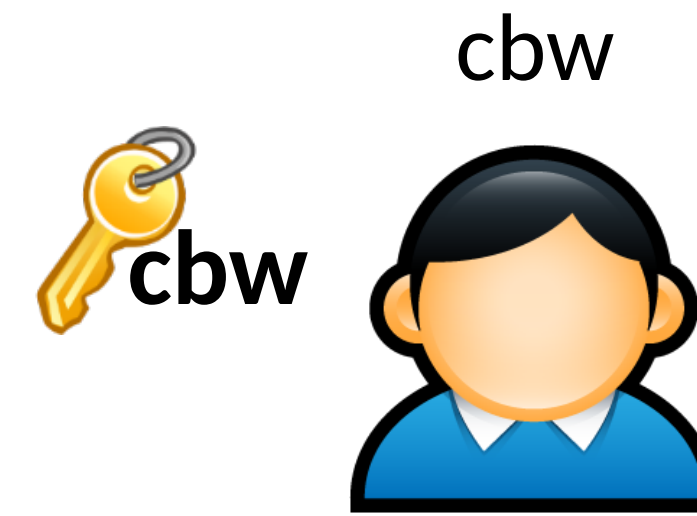
Attacking Needham-Schroeder

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



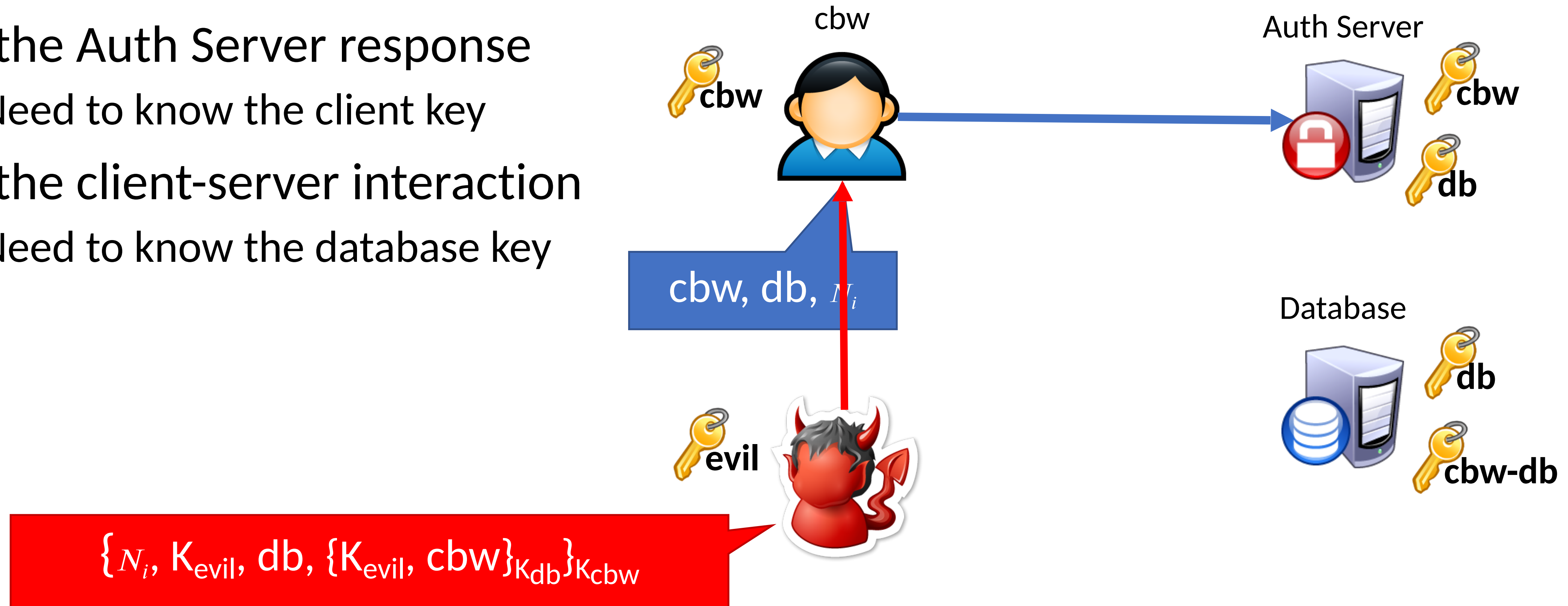
Attacking Needham-Schroeder

- 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



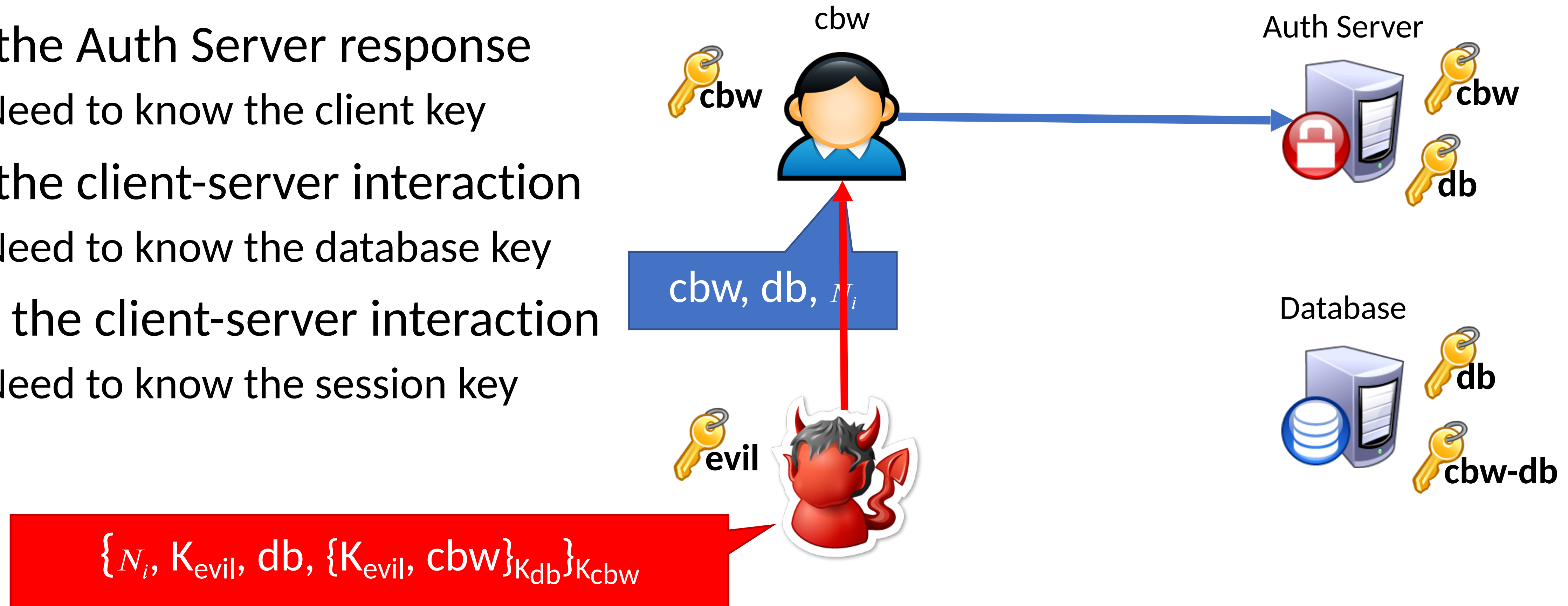
Attacking Needham-Schroeder

- 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



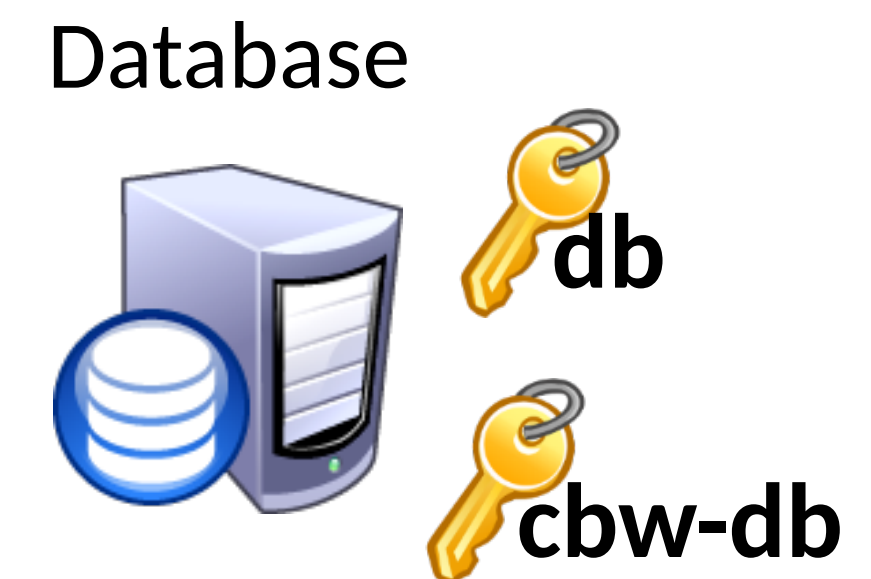
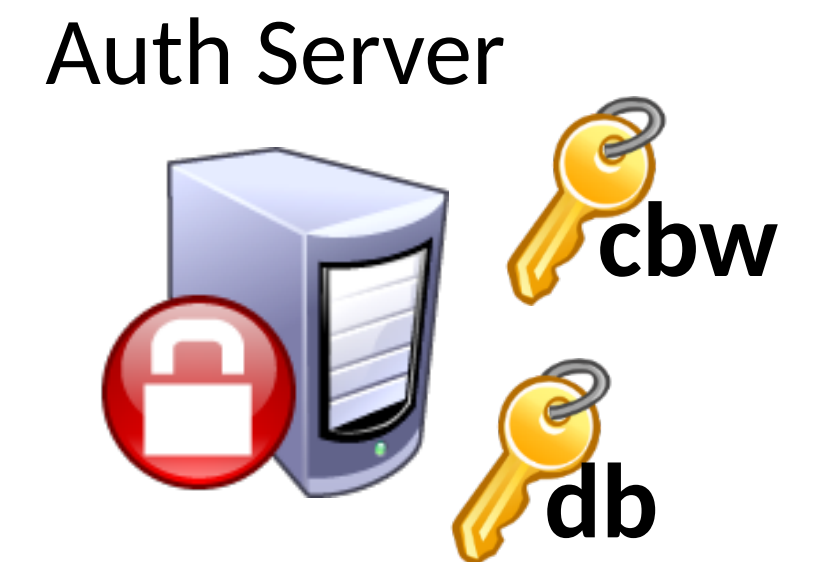
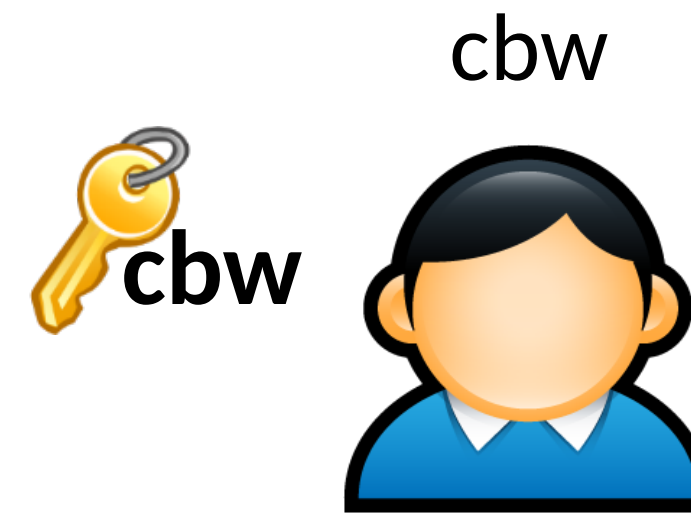
Attacking Needham-Schroeder

- 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

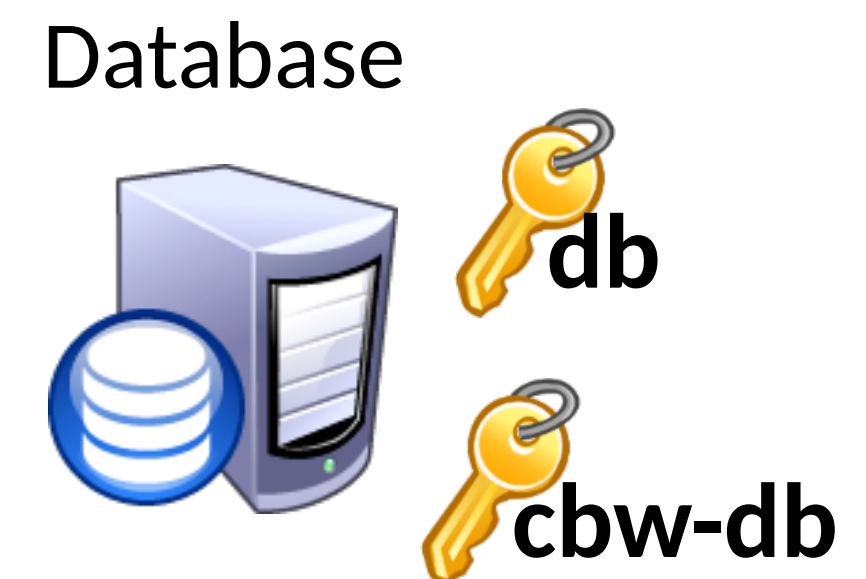
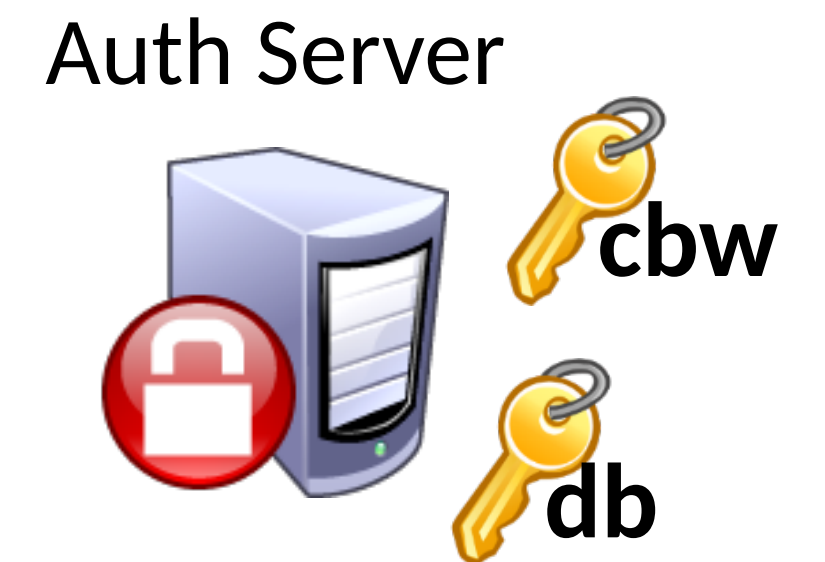
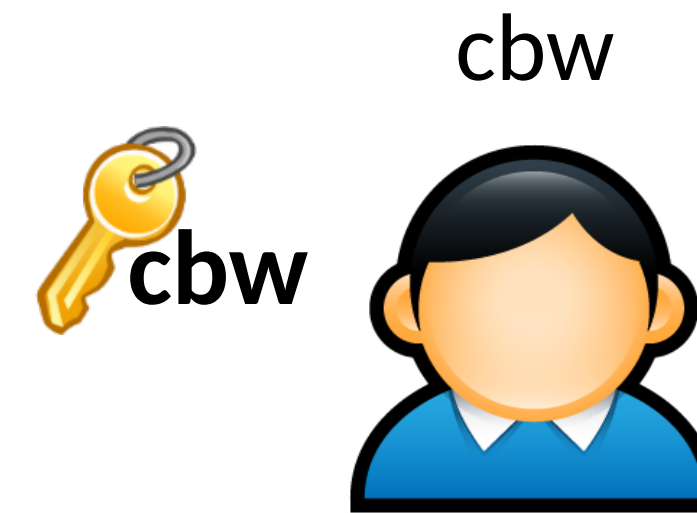
- Spoof the client request
 - Fail! Client key is needed to decrypt
- Spoof the Auth Server response
 - Fail! Need to know the client key
- Spoof the client-server interaction
 - Fail! Need to know the database key
- Replay the client-server interaction
 - Fail! Need to know the session key



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

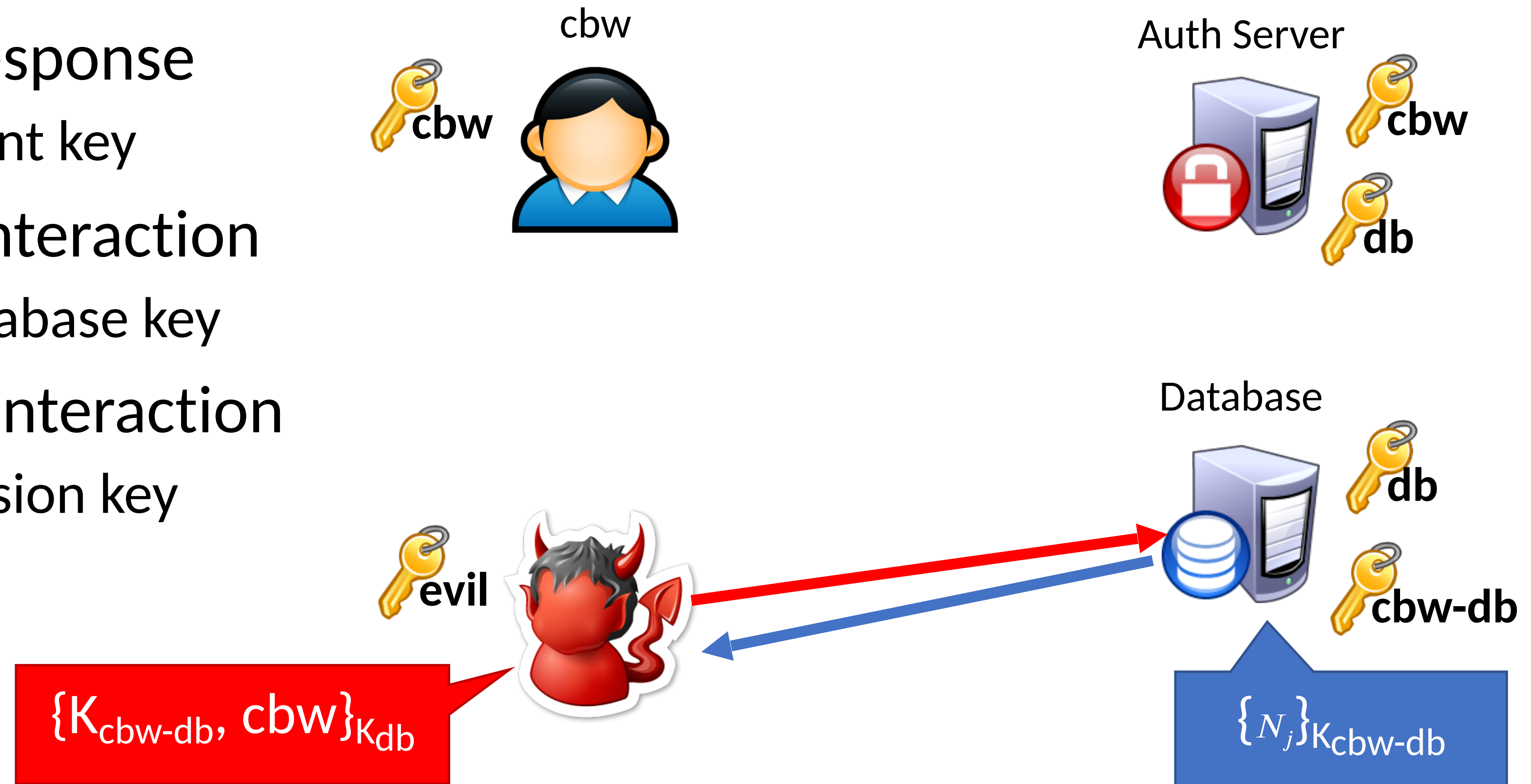
Attacking Needham-Schroeder

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

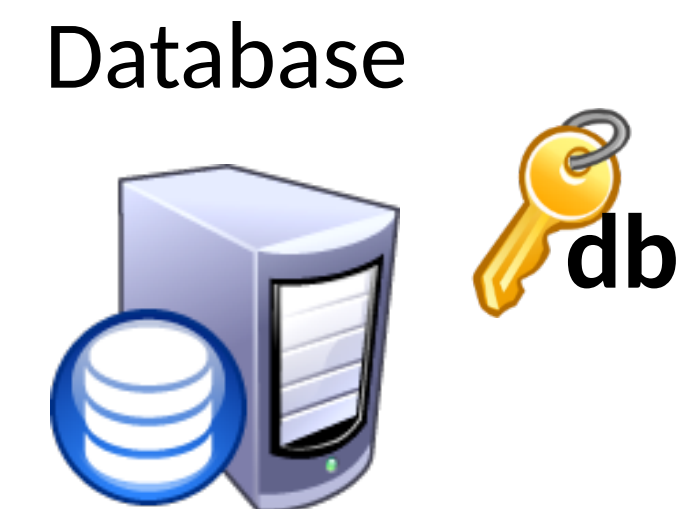
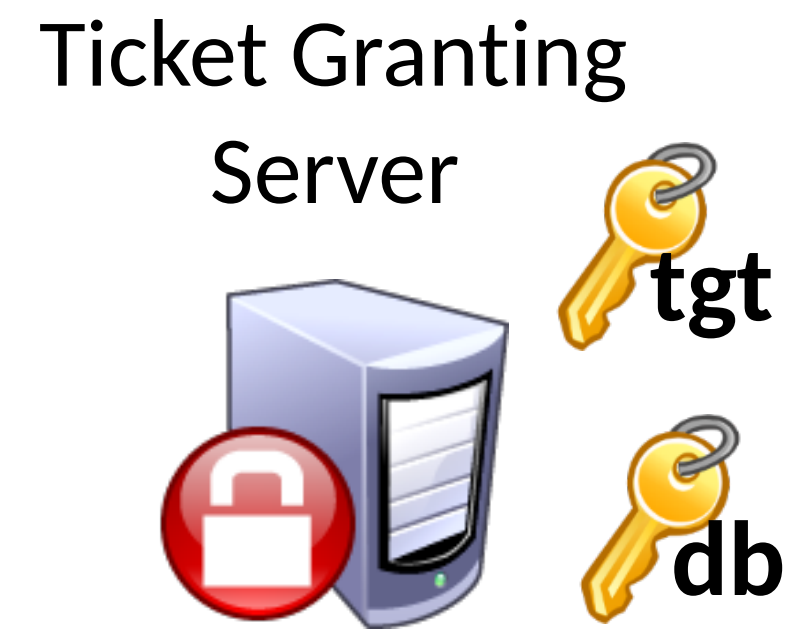
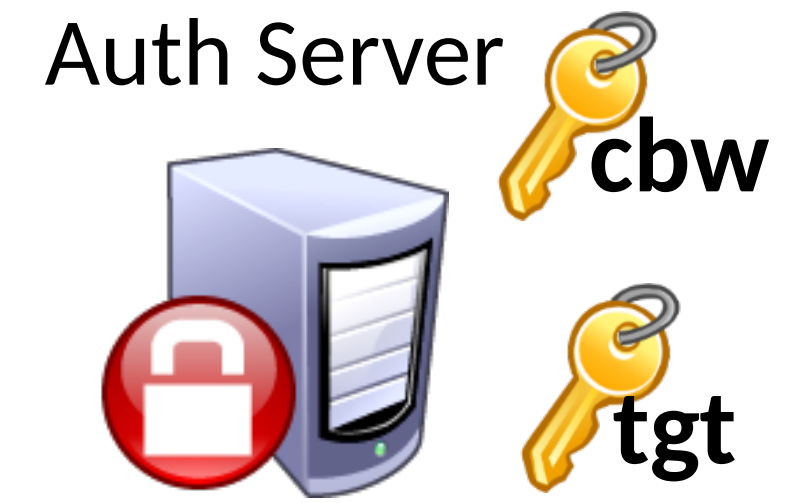
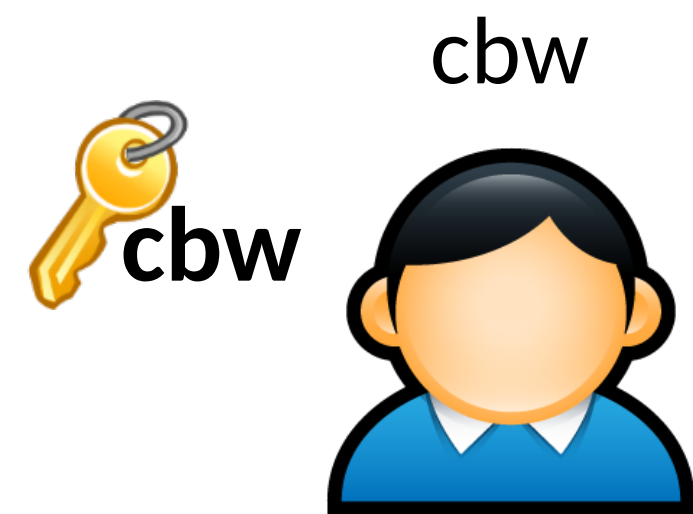
5) $A \rightarrow B: \{N_j - 1\}_{K_{AB}}$

B only accepts requests with fresh timestamps

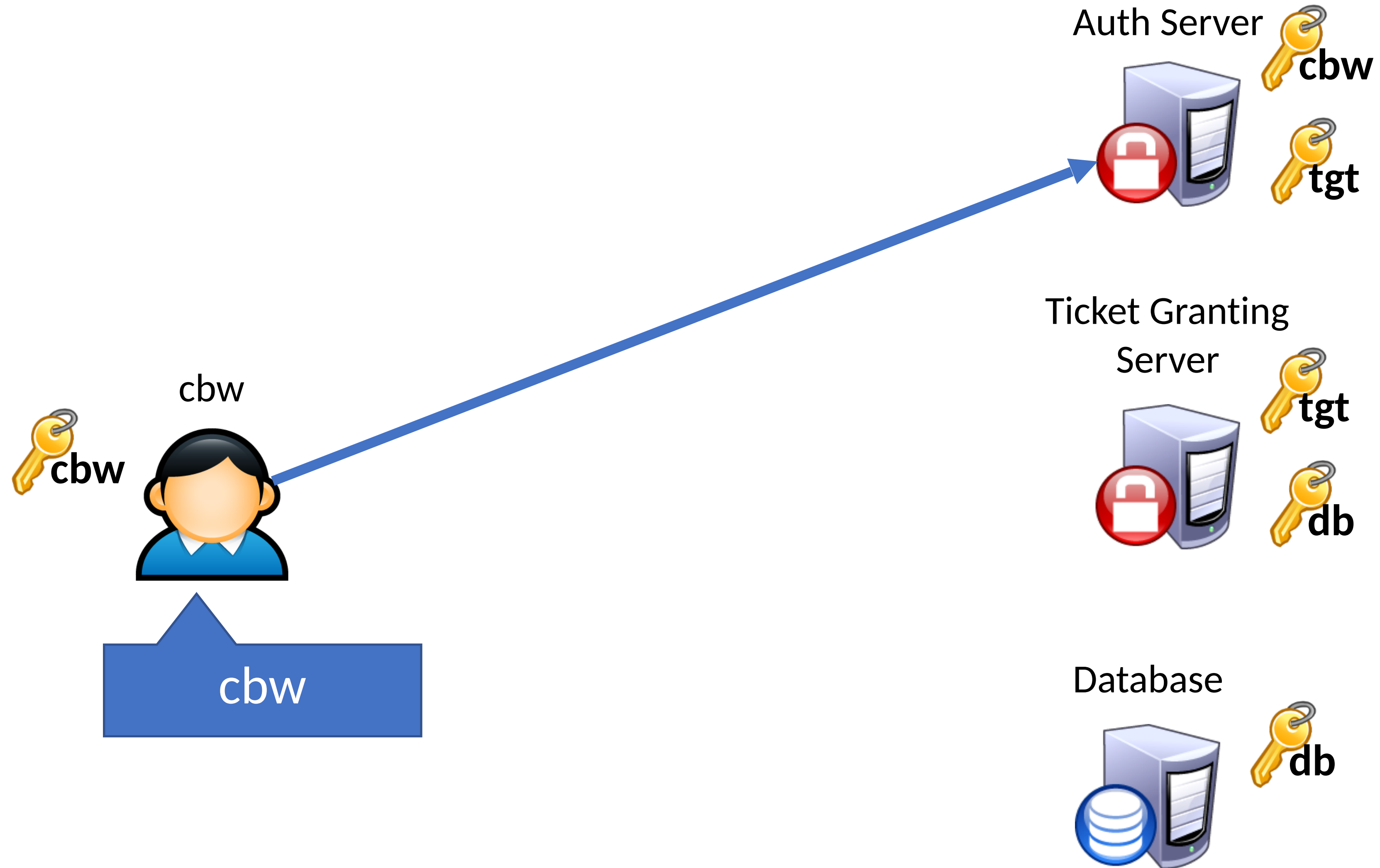
Kerberos

- 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**

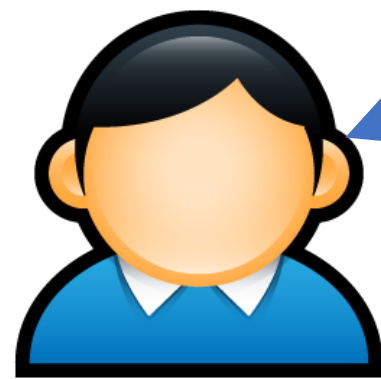
 **db**

Database



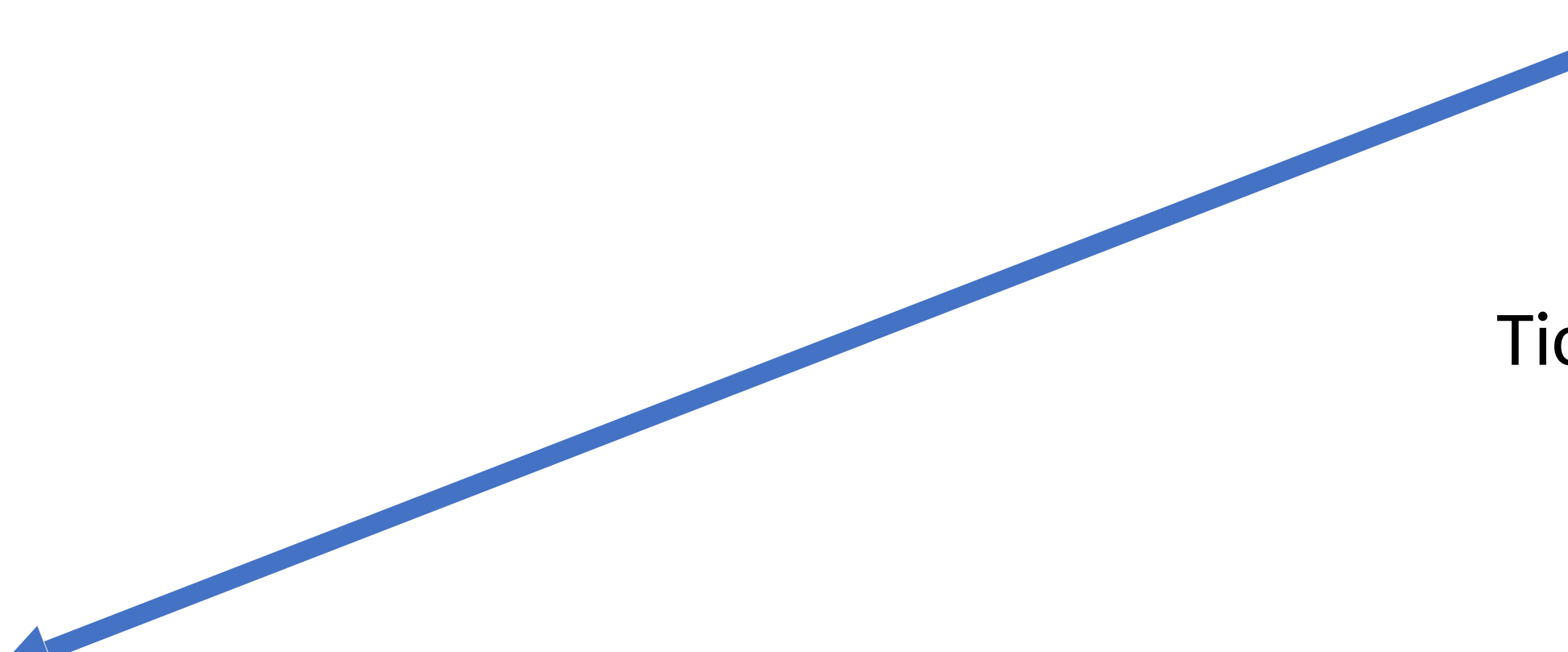
 **db**

cbw

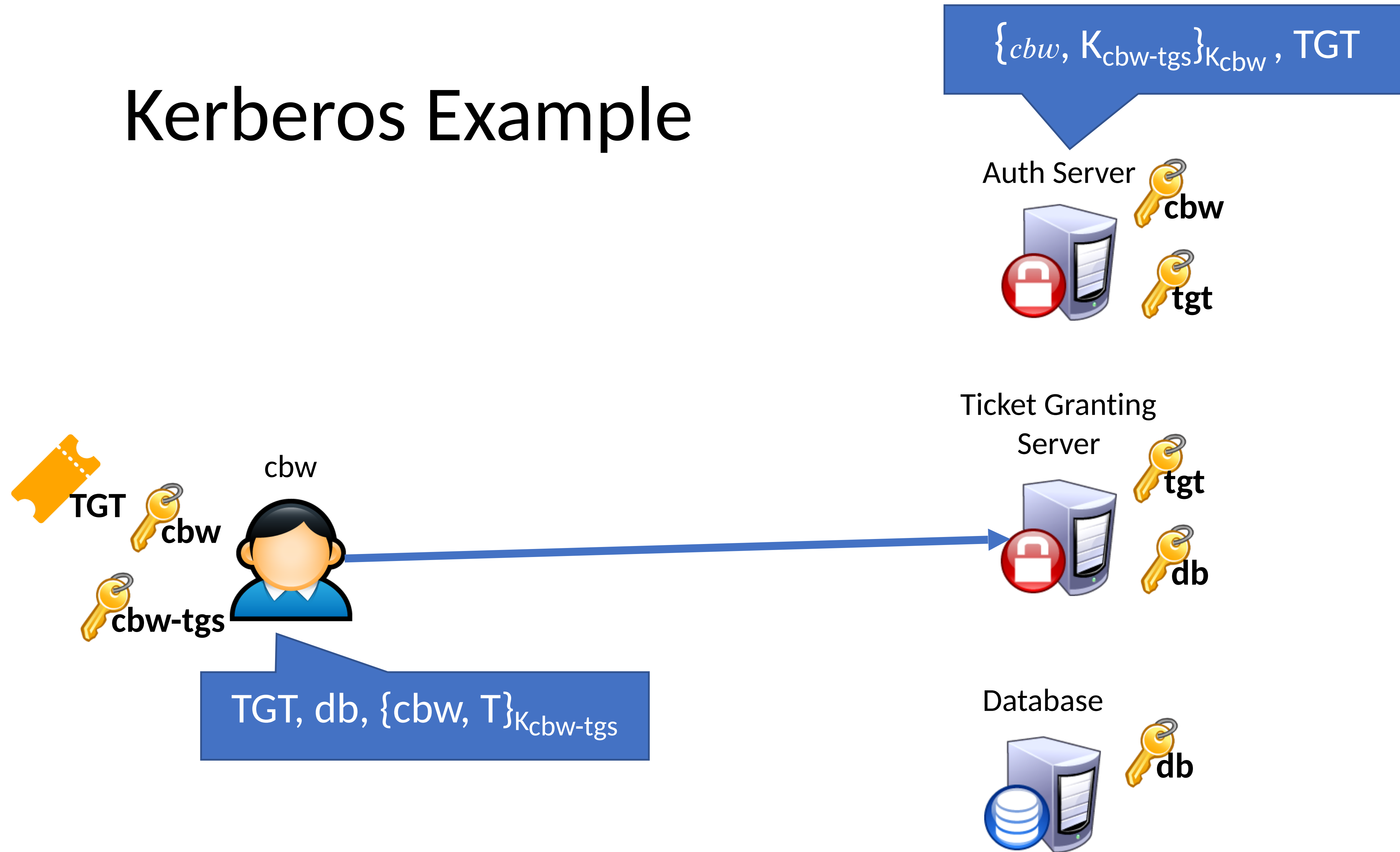


 **TGT**
 **cbw**

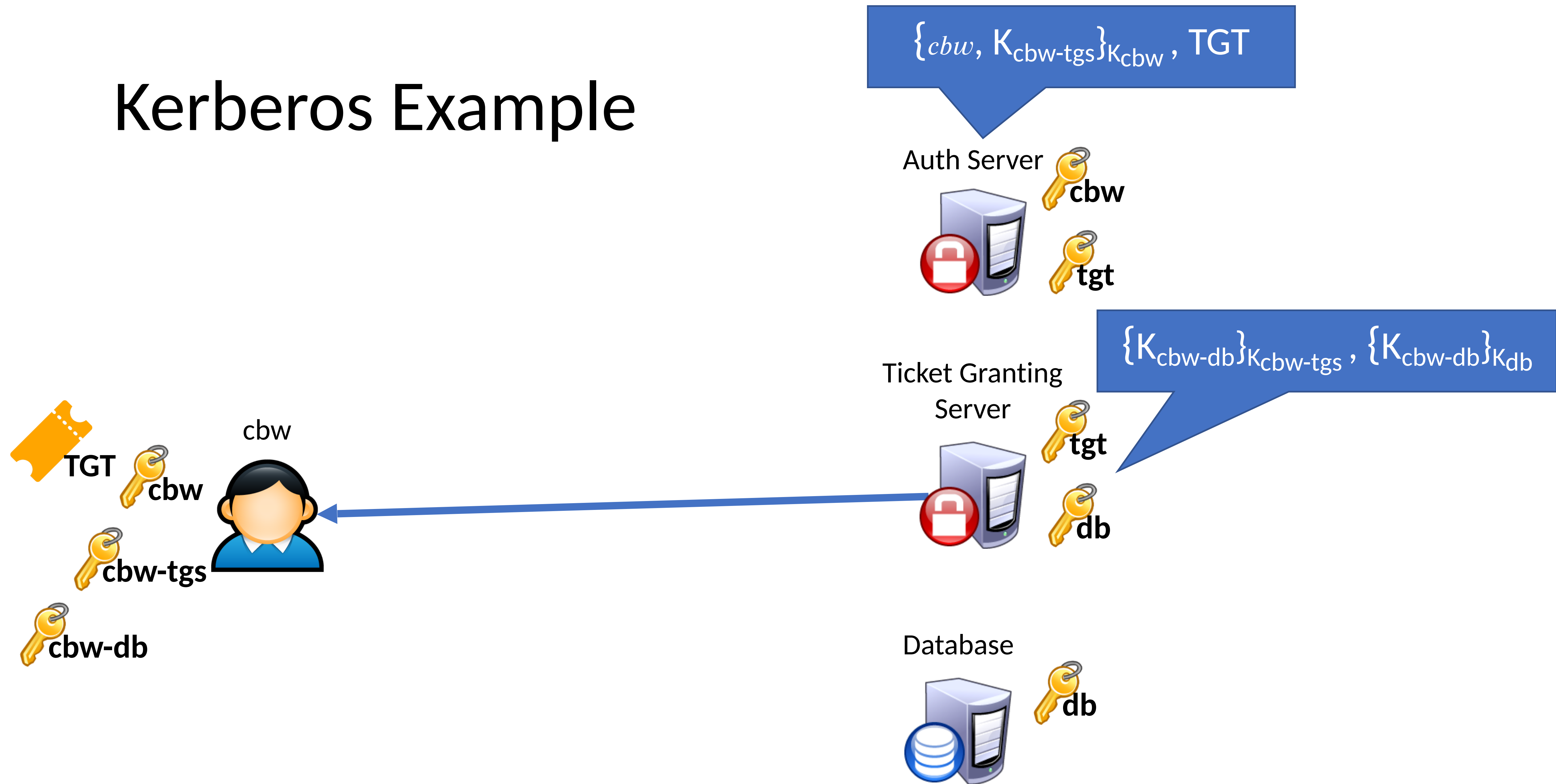
 **cbw-tgs**



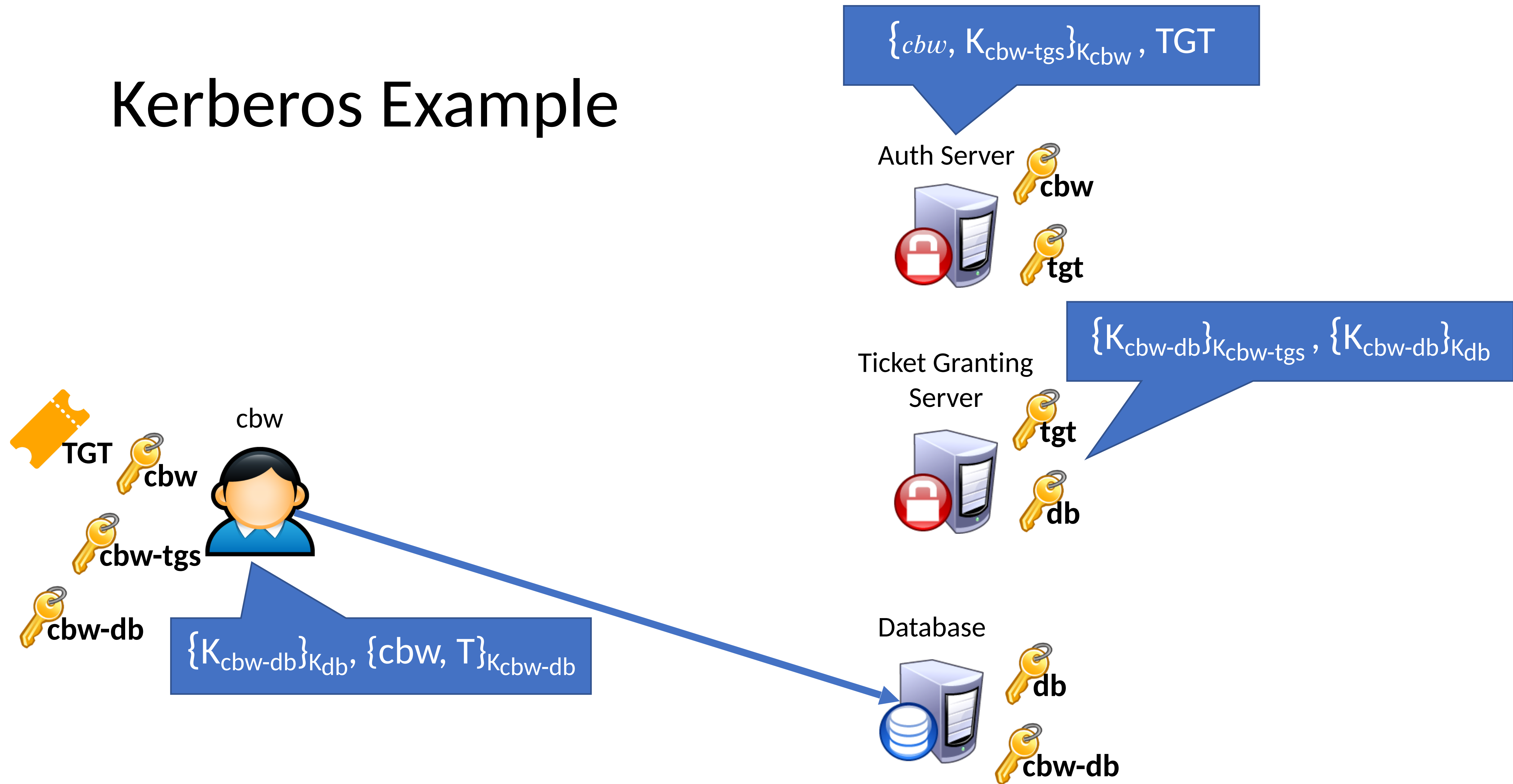
Kerberos Example



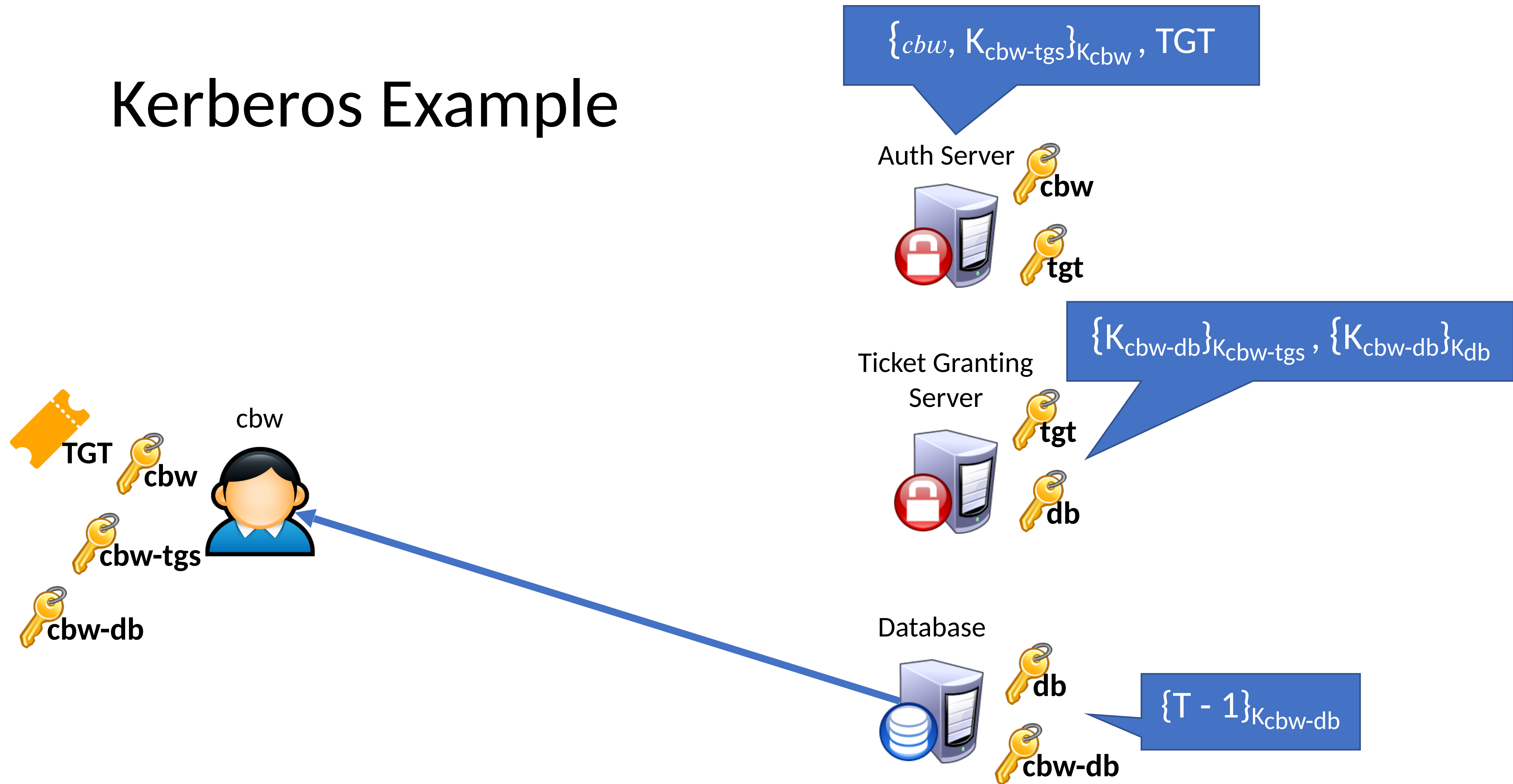
Kerberos Example



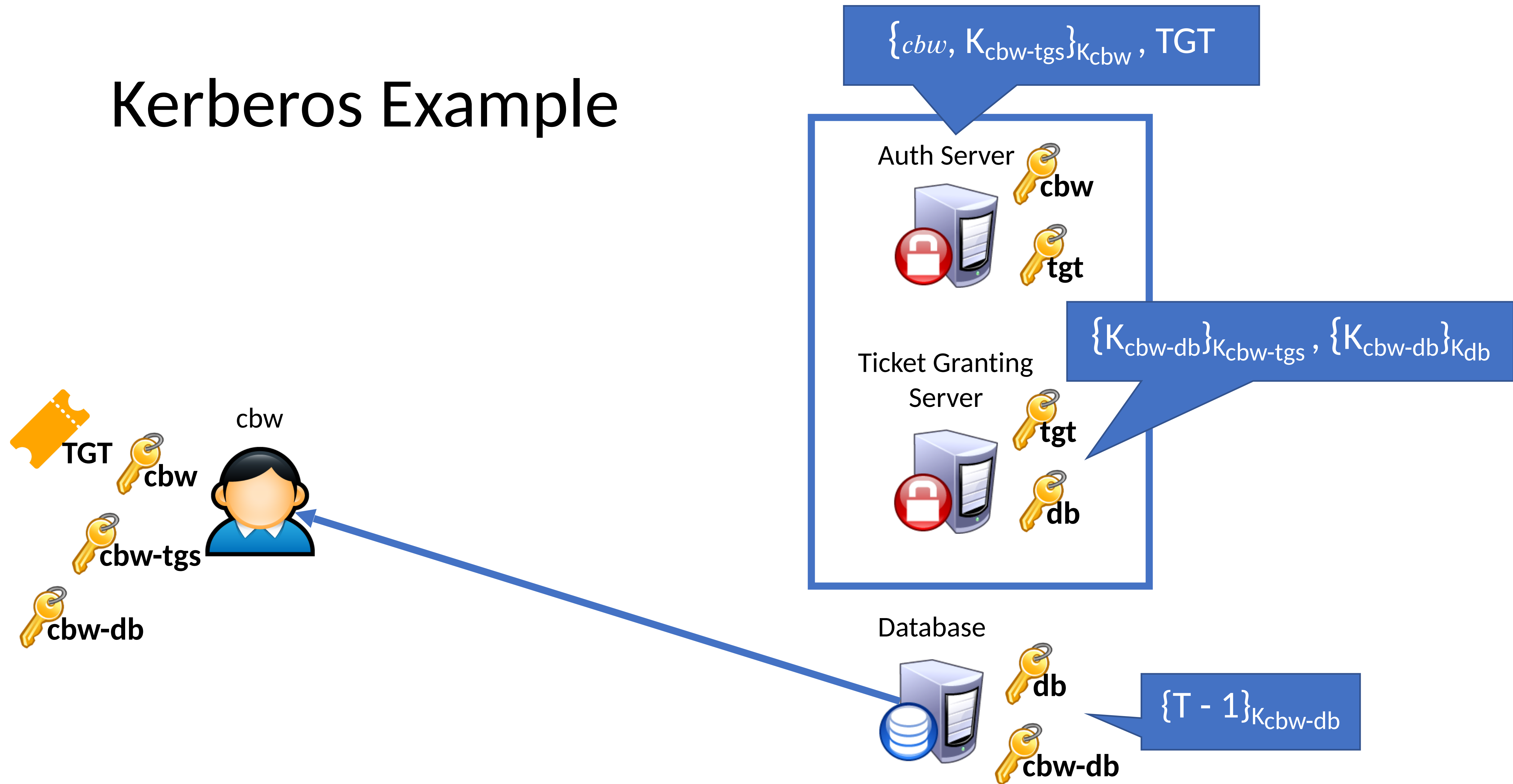
Kerberos Example



Kerberos Example



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>