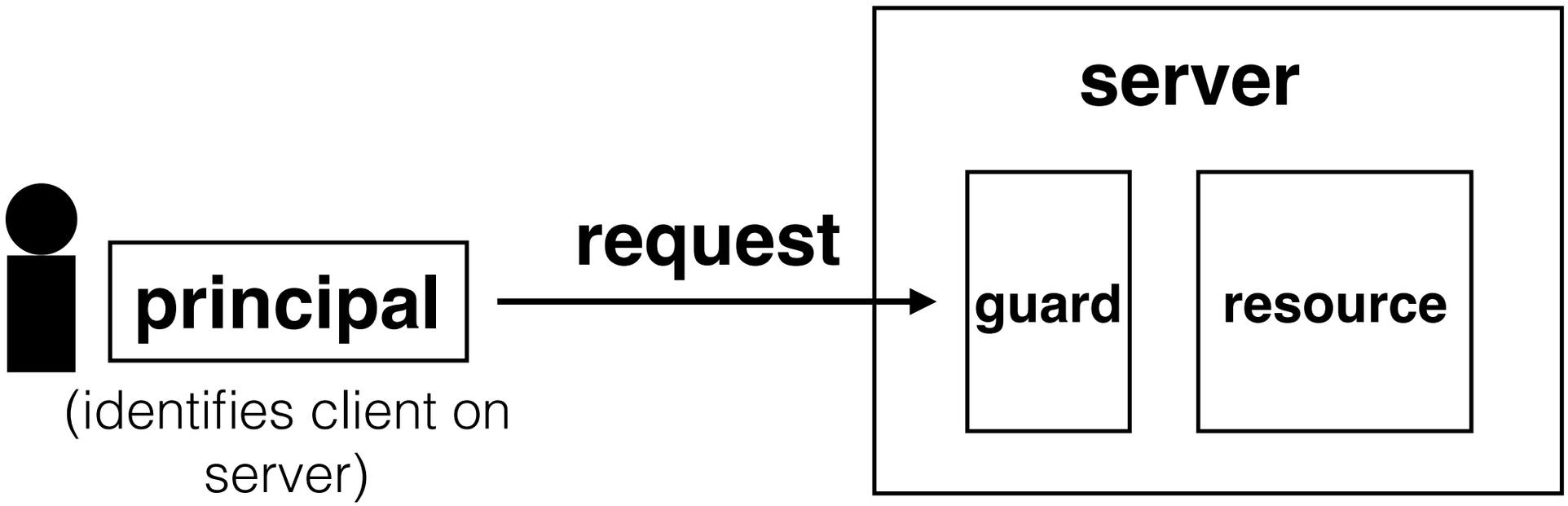
6.033 Spring 2019

Lecture #21

- **Principal Authentication via Passwords**

complete mediation: every request for resource goes through the guard



guard often provides:

authentication: is the principal who they claim to be?

authorization: does principal have access to perform request on resource?

Rank	2012	2013	2014	2015	2016	2017	2018
1	password	123456	123456	123456	123456	123456	123456
2	123456	password	password	password	password	password	password
3	12345678	1234567	12345	12345678	12345	12345678	123456789
4	abc123	qwerty	12345678	qwerty	12345678	qwerty	12345678
5	qwerty	abc123	qwerty	12345	football	12345	12345
6	monkey	1234567	123456789	123456789	qwerty	123456789	111111
7	letmein	111111	1234	football	123456789	letmein	1234567
8	dragon	1234567	baseball	1234	1234567	1234567	sunshine
9	111111	iloveyou	dragon	1234567	princess	football	qwerty
10	baseball	adobe12	football	baseball	1234	iloveyou	iloveyou
11	iloveyou	123123	1234567	welcome	login	admin	princess
12	trustno1	admin	monkey	123456789	welcome	welcome	admin
13	1234567	1234567	letmein	abc123	solo	monkey	welcome
14	sunshine	letmein	abc123	111111	abc123	login	666666
15	master	photosho	111111	1qaz2wsx	admin	abc123	abc123
16	123123	1234	mustang	dragon	121212	starwars	football
17	welcome	monkey	access	master	flower	123123	123123
18	shadow	shadow	shadow	monkey	passw0rd	dragon	monkey
19	ashley	sunshine	master	letmein	dragon	passw0rd	654321
20	football	12345	michael	login	sunshine	master	!@#$%^&*
21	jesus	password	superman	princess	master	hello	charlie
22	michael	princess	696969	qwertyuiop	hottie	freedom	aa123456
23	ninja	azerty	123123	solo	loveme	whatever	donald
24	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx	password1
25	password1	000000	trustno1	starwars	password1	trustno1	qwerty123

problem: users pick terrible passwords

source: SplashData, via wikipedia

username	password
arya	valarMorghul1s
jon	w1nterIsC0ming
sansa	LemonCakesForever
hodor	hodor

```
check_password(username, inputted_password):
    stored_password = accounts_table[username]
    return stored_password == inputted_password
```

problem: adversary with access to server can get passwords

username	hash(password)
arya	de5aba604c340e1965bb27d7a4c4ba03f4798ac7
jon	321196d4a6ff1372021914898 95e58c29475ccab
sansa	6ea7c2b3e08a3d19fee5766cf9fc51680b267e9f
hodor	c6447b82fbb4b8e7dbcf2d28a4d7372f5dc32687

```
def check_password(username, inputted_password):
    stored_hash = accounts_table[username]
    inputted_hash = hash(inputted_password)
    return stored_hash == inputted_hash
```

problem: hashes are fast to compute, so adversary could quickly create a data structure mapping a *lot* of passwords to their hashes

username	slow_hash(password)
arya	0W4uHhyIOxp5pTFnKJ7iGkx4mH7bteG
jon	34P1U08R9BUYdE.FQJg8c9LBN5Lfepm
sansa	6DKZlRkB7WYvR2J15CjJT20NXYKUeh6
hodor	Wf1BujzkzRezVmNmCsBBqfYI23L3X.6

```
def check_password(username, inputted_password):
    stored_hash = accounts_table[username]
    inputted_hash = slow_hash(inputted_password)
    return stored_hash == inputted_hash
```

problem: adversary can still create a data structure mapping the most common passwords to their (slow) hashes

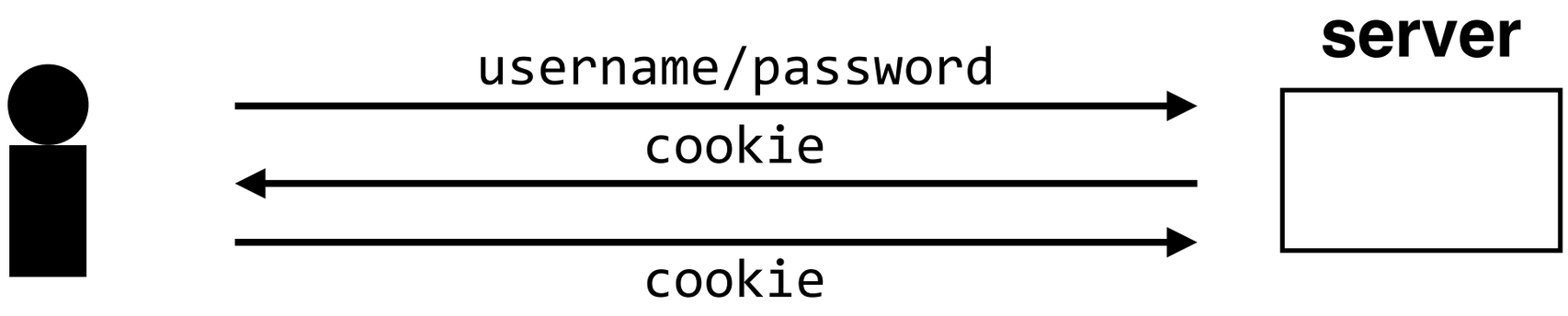
stored in plaintext

username	salt	slow_hash(password | salt)
arya	a7tON1uxjhnoiOfHpQleDO	P2l081lU4KbPr6zZmwgUR/kv4fliGOm
jon	.yQszx42GRwtIG.Kxq5Vfe	o.mCJOo/2USSxkwaWkoYegLcRLMrLIa
sansa	RKtMTWvnCe3GnTqWgtvxNe	SUe3PTY8KNbrpnTepuqlVfM2ApsAXw2
hodor	tL8Cq9R1zi.UNaUuxJ3N4e	V2ItTHW170lGYmQSyddtzr0cZPCb4RO

```
check_password(username, inputted_password)
  stored_hash = accounts_table[username].hash
  salt = accounts_table[username].salt
  inputted_hash = slow_hash(inputted_password | salt)
  return stored_hash == inputted_hash
```

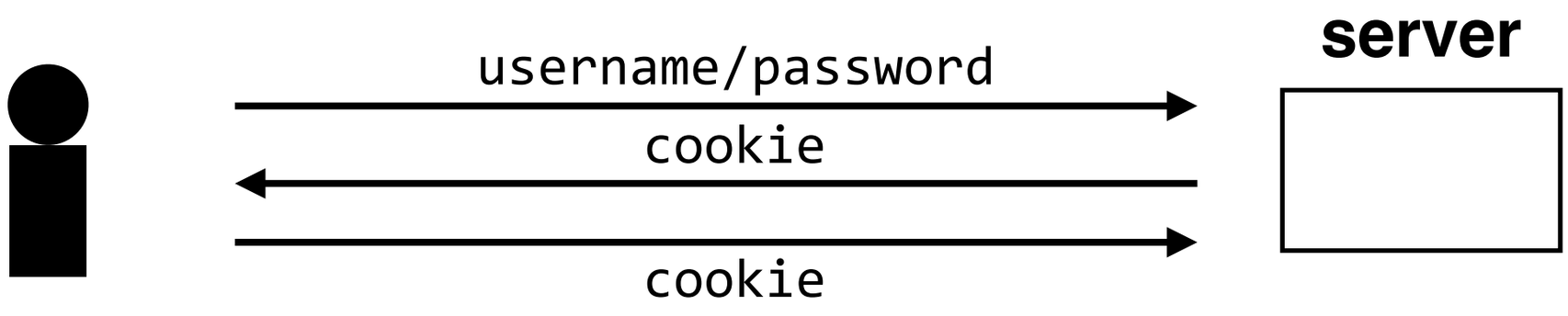
adversary would need a separate password⟶hash(password | salt) mapping for every possible salt

how can we avoid transmitting the password over and over?



once the client has been authenticated, the server will send it a “cookie”, which the client can use to keep authenticating itself for some period of time

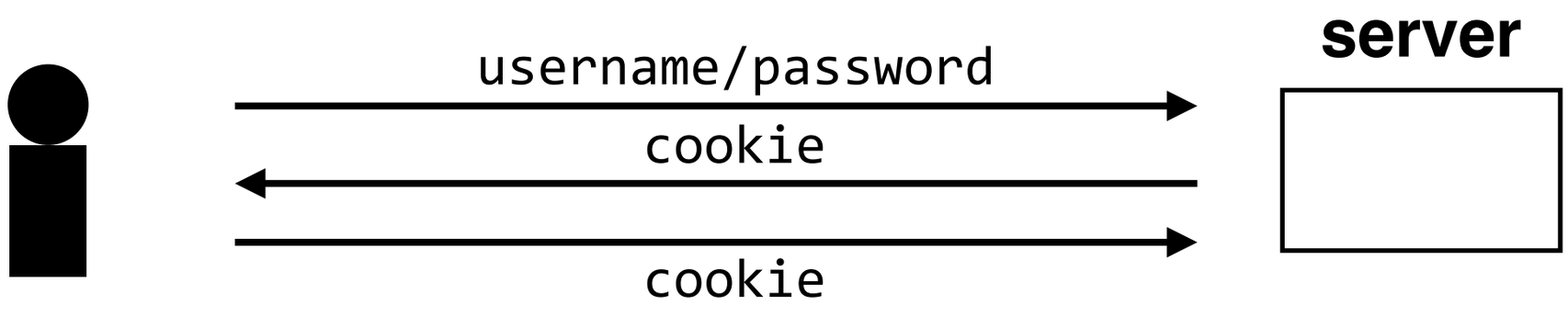
how can we avoid transmitting the password over and over?




```
cookie = {username, expiration} ?
```

problem: adversary could easily create their own cookie and masquerade as this user

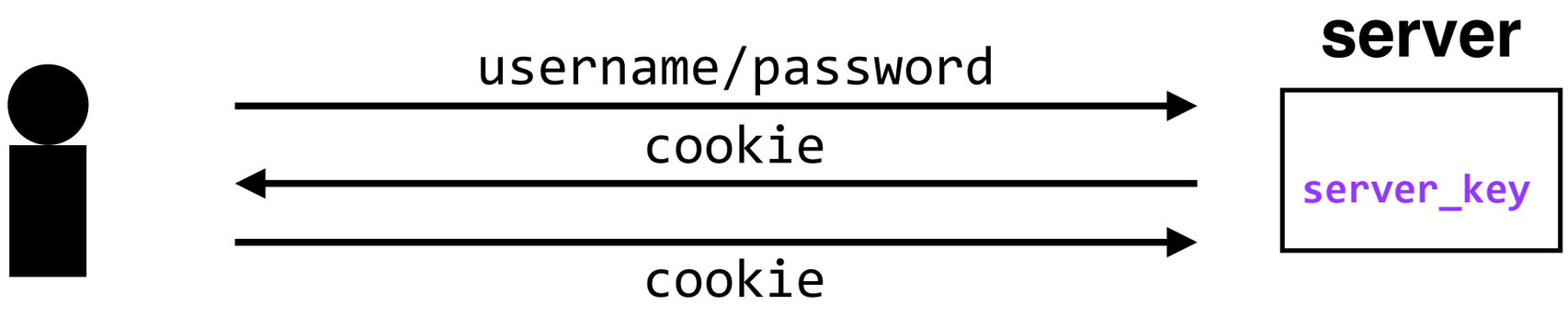
how can we avoid transmitting the password over and over?




```
cookie = {username, expiration, H(username | expiration)} ?
```

problem: adversary could easily create their own cookie and masquerade as this user

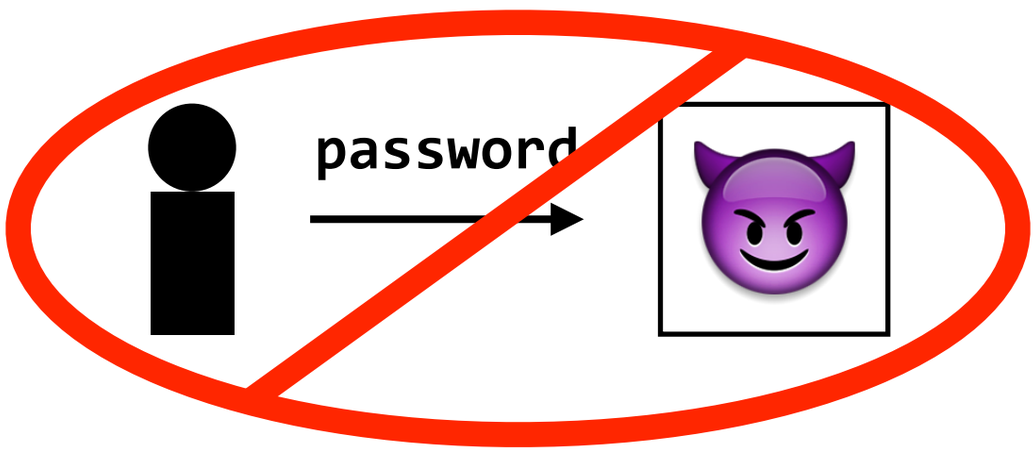
how can we avoid transmitting the password over and over?



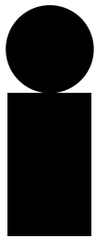
{username, expiration, H(server_key | username | expiration)}

how can we protect against phishing attacks, where an adversary tricks a user into revealing their password?

must avoid sending the password to the server entirely, but still allow valid servers to authenticate users



challenge-response protocol



(random number)

458653

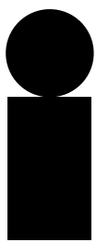
ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(valarMorghul1s | 458643)

valid server

username	password
arya	valarMorghul1s
jon	w1nterIsC0ming
sansa	LemonCakesForever
hodor	hodor

challenge-response protocol



(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(valarMorghul1s | 458643)

valid server

username	password
arya	valarMorghul1s
jon	w1nterIsC0ming
sansa	LemonCakesForever
hodor	hodor

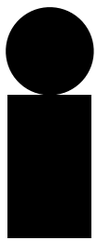
server computes

H(valarMorghul1s | 458643)

and checks

password is never sent directly

challenge-response protocol



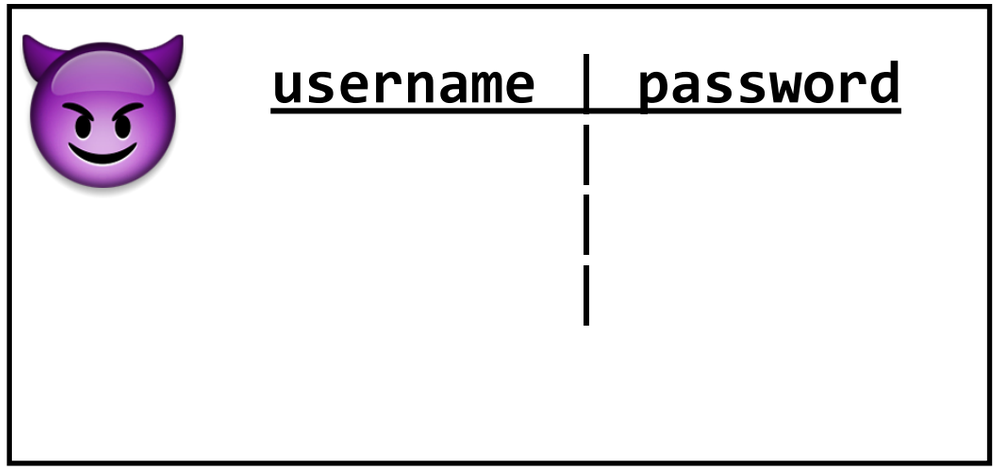
(random number)

458653

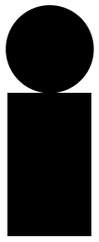
ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(valarMorghul1s | 458643)

adversary-owned server



challenge-response protocol



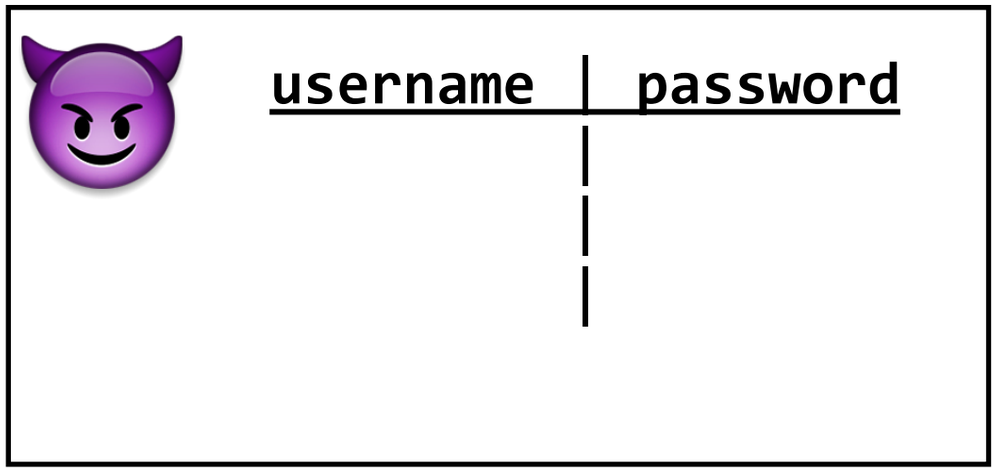
(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

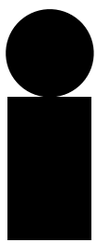
= H(valarMorghul1s | 458643)

adversary-owned server



adversary only learns
H(valarMorghul1s | 458643);
can’t recover the password
from that

challenge-response protocol



(random number)

458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(valarMorghul1s | 458643)

valid server

username	password
arya	valarMorghul1s
jon	w1nterIsC0ming
sansa	LemonCakesForever
hodor	hodor

server computes H(valarMorghul1s | 458643) and checks

password is never sent directly

adversary-owned servers (that don't know passwords) won't learn the password; client never sends password directly

problems arise when the server stores (salted) hashes — as it should be doing — but there are challenge-response protocols that handle that case

how do we initially set (“bootstrap”) or reset a password?

are there better alternatives to passwords?

- Using passwords securely takes some effort. Storing **salted hashes**, incorporating **session cookies**, dealing with **phishing**, and **bootstrapping** are all concerns.

- Thinking about how to use passwords provides more **general lessons**: consider human factors when designing secure systems, in particular.

- There are always **trade-offs**. Many “improvements” on passwords add security, but also complexity, and typically decrease usability.

Hacker Finds He Can Remotely Kill Car Engines After Breaking Into GPS Tracking Apps

"I can absolutely make a big traffic problem all over the world," the hacker said.

By reverse engineering ProTrack and iTrack's Android apps, L&M said he realized that all customers are given a default password of 123456 when they sign up.

At that point, the hacker said he brute-forced "millions of usernames" via the apps' API. Then, he said he wrote a script to attempt to login using those usernames and the default password.

https://motherboard.vice.com/en_us/article/zmpx4x/hacker-monitor-cars-kill-engine-gps-tracking-apps?wpisrc=nl_cybersecurity202&wpmm=1