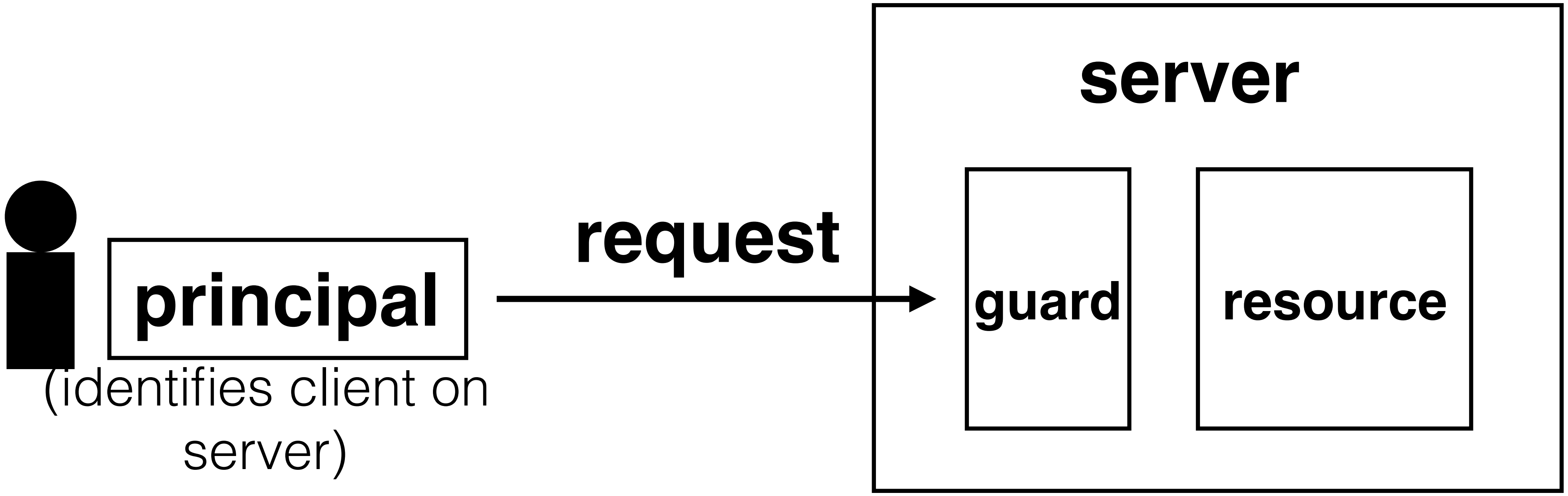
6.033 Spring 2018

Lecture #21

- **Principal Authentication via Passwords**

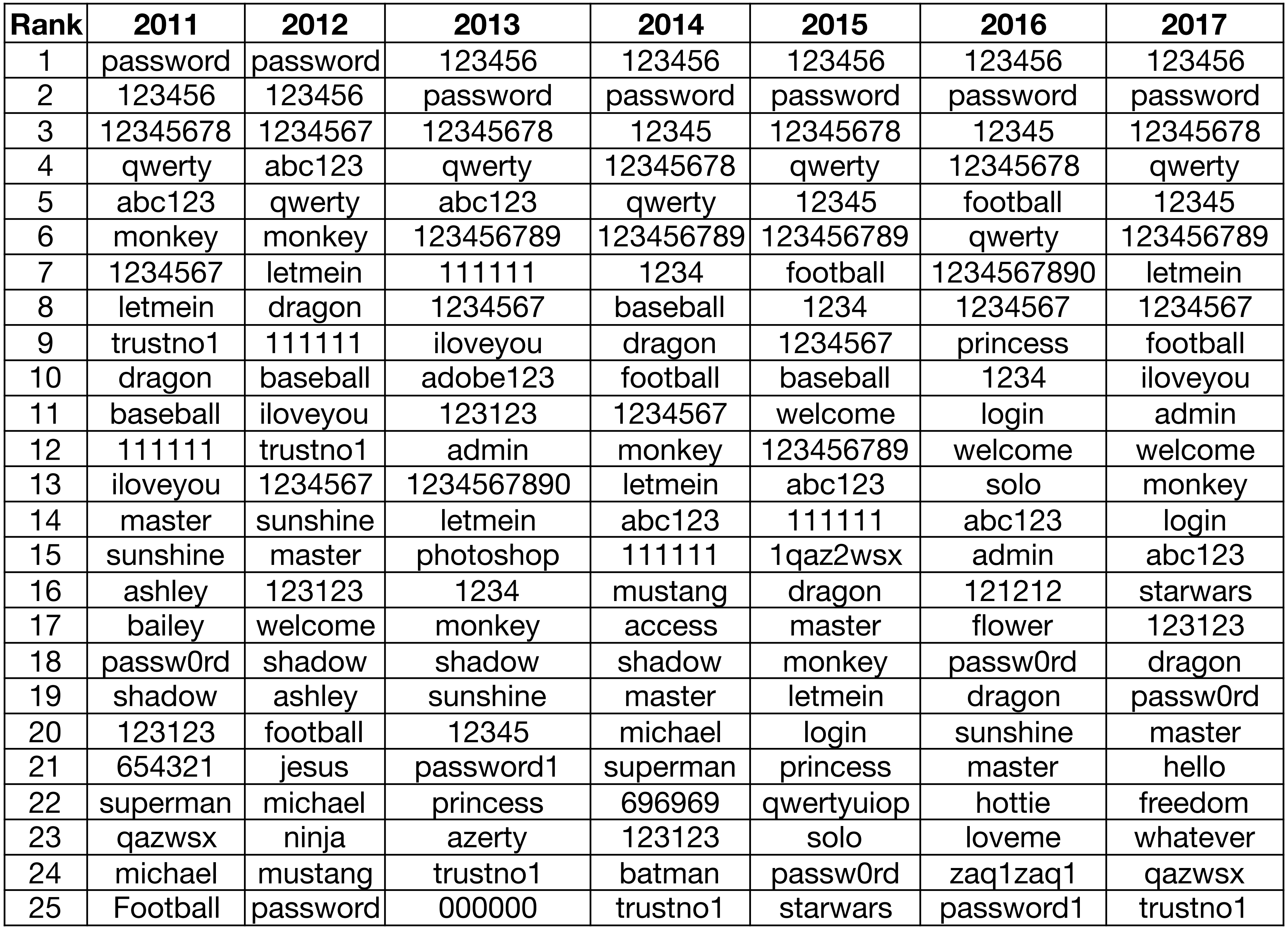
complete mediation: every request for resource goes through the guard



guard typically provides:

authentication: is the principal who they claim to be?

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



Rank	2011	2012	2013	2014	2015	2016	2017
1	password	password	123456	123456	123456	123456	123456
2	123456	123456	password	password	password	password	password
3	12345678	1234567	12345678	12345	12345678	12345	12345678
4	qwerty	abc123	qwerty	12345678	qwerty	12345678	qwerty
5	abc123	qwerty	abc123	qwerty	12345	football	12345
6	monkey	monkey	123456789	123456789	123456789	qwerty	123456789
7	1234567	letmein	111111	1234	football	1234567890	letmein
8	letmein	dragon	1234567	baseball	1234	1234567	1234567
9	trustno1	111111	iloveyou	dragon	1234567	princess	football
10	dragon	baseball	adobe123	football	baseball	1234	iloveyou
11	baseball	iloveyou	123123	1234567	welcome	login	admin
12	111111	trustno1	admin	monkey	123456789	welcome	welcome
13	iloveyou	1234567	1234567890	letmein	abc123	solo	monkey
14	master	sunshine	letmein	abc123	111111	abc123	login
15	sunshine	master	photoshop	111111	1qaz2wsx	admin	abc123
16	ashley	123123	1234	mustang	dragon	121212	starwars
17	bailey	welcome	monkey	access	master	flower	123123
18	passw0rd	shadow	shadow	shadow	monkey	passw0rd	dragon
19	shadow	ashley	sunshine	master	letmein	dragon	passw0rd
20	123123	football	12345	michael	login	sunshine	master
21	654321	jesus	password1	superman	princess	master	hello
22	superman	michael	princess	696969	qwertyuiop	hottie	freedom
23	qazwsx	ninja	azerty	123123	solo	loveme	whatever
24	michael	mustang	trustno1	batman	passw0rd	zaq1zaq1	qazwsx
25	Football	password	000000	trustno1	starwars	password1	trustno1

problem: users pick terrible passwords

source: SplashData, via wikipedia

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

```
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)
dom	e5f3c4e1694c53218978fae2c302faf4a817ce7b
han	365dab99ab03110565e982a76b22c4ff57137648
roman	ed0fa63cd3e0b9167fb48fa3c1a86d476c1e8b27
tej	0e0201a89000fe0d9f30adec170dabce8c272f7c

```
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 “rainbow table”

username	slow_hash(password)
dom	gamynjSAIeYZ4iOBT4uaO3r5ub8O
han	JXYWVPkpoQ6W1tbA21t6c66G4QUo
roman	Xn5U1QvQz5MGOzdfJWgF8OiDFv1q
tej	lo5WIidPPZePoSyMB2O.fUz3fLeZ

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

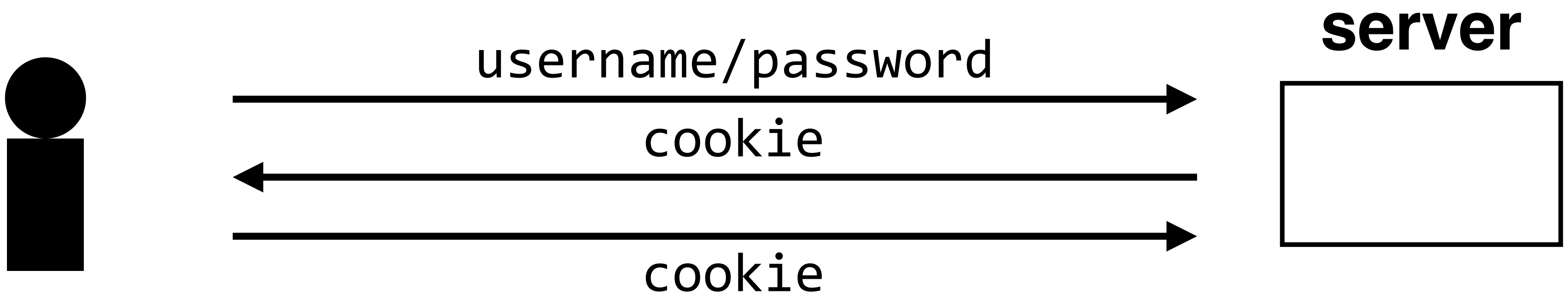
problem: adversary can still create rainbow tables for the most common passwords stored in plaintext

```
username | salt                      | slow_hash(password | salt)
dom      | LwVx6kO4SNY3jPVfOpfYe.  | M4ayLRWuzU.sSQtjoteIrIjNXI4UX
han      | UbDsytUST6d0cFpmuhWu.e  | Y8ie/A18u9ymrS0FgVh9IOVx2Qe48
roman     | CnfkXqUJz5C5OfucP/UKIu | 3GDJu07gk2iL7mFVquOzPt3L3IITe
tej       | cBGohtI6BwsaVs0SAo0u7. | 8/v1Kl6rImUMYVw/.oGmA/BaRAlgC
```

```
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
```

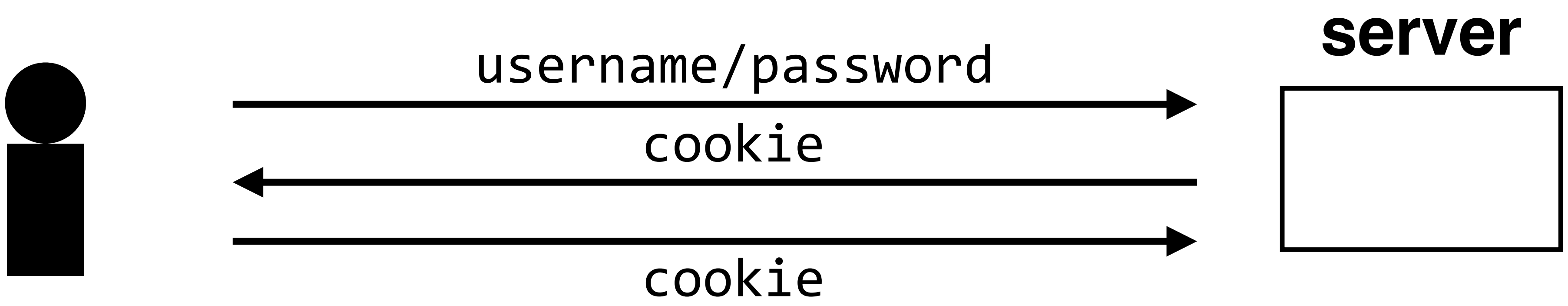
adversary would need a separate rainbow table for every possible salt

how can we avoid transmitting the password over and over?



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

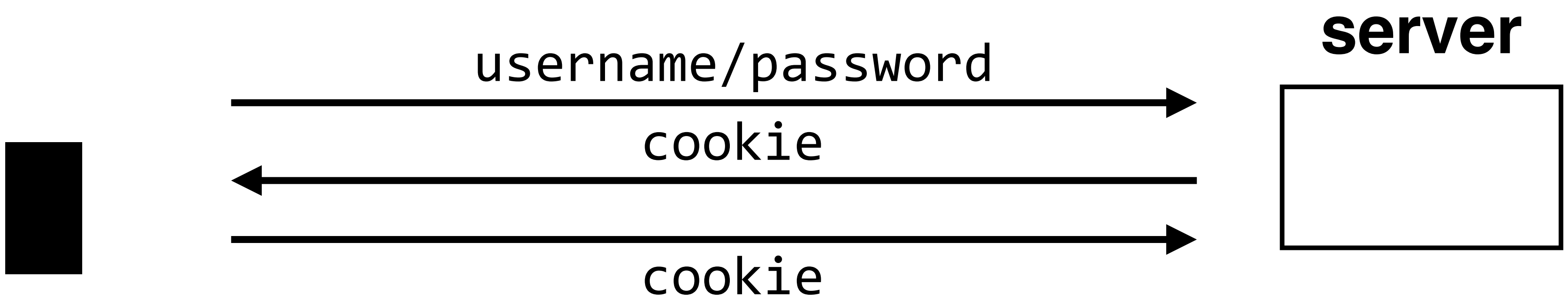
how can we avoid transmitting the password over and over?



`cookie = {username, expiration} ?`

problem: adversaries could easily create their own cookies

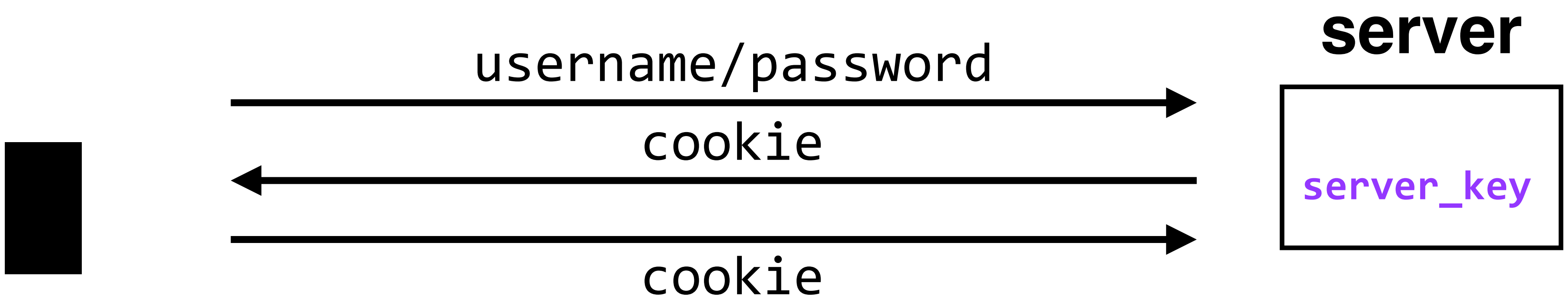
how can we avoid transmitting the password over and over?



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

problem: adversaries could still easily create their own cookies

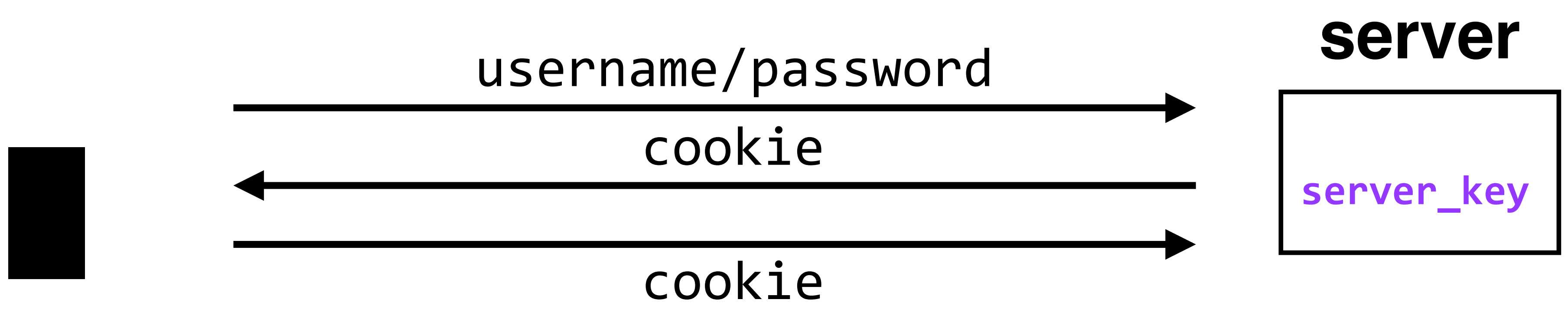
how can we avoid transmitting the password over and over?



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

problem: adversaries could *still* easily create their own cookies

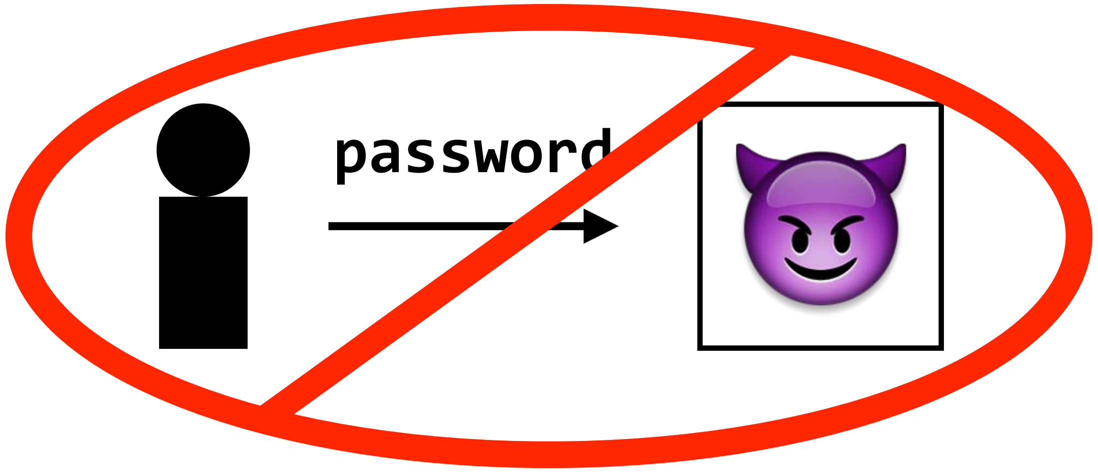
how can we avoid transmitting the password over and over?



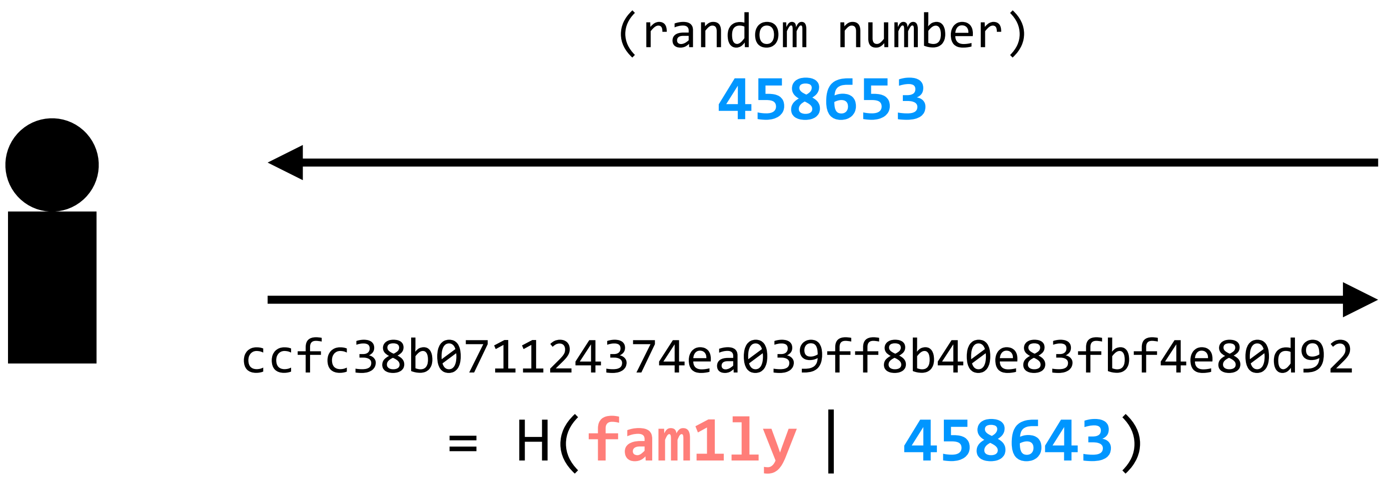
`{username, expiration, H(server_key | username | expiration)}`

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(fam1ly | 458643)

password is never sent directly

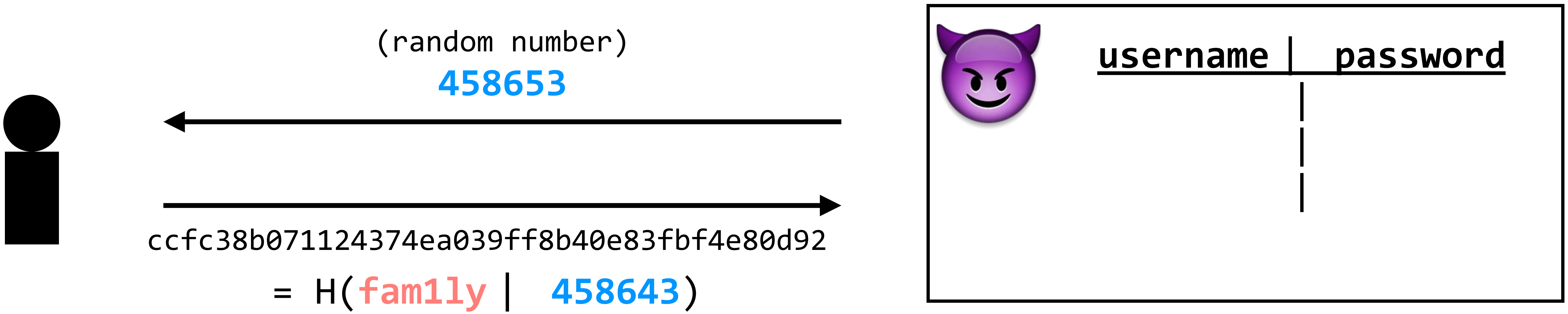
valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

server computes H(fam1ly | 458643) and checks

challenge-response protocol

adversary-owned server



(random number)

458653

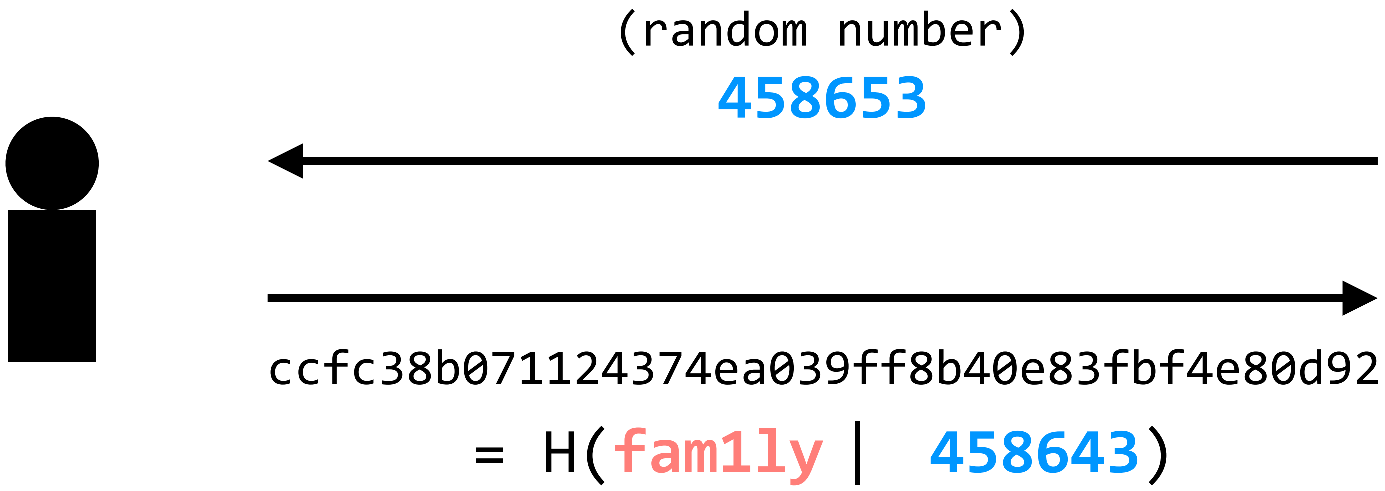
ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(fam1ly | 458643)

username | password

adversary only learns H(fam1ly | 458643); can’t recover the password from that

challenge-response protocol



(random number)
458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92
= H(fam1ly | 458643)

password is never sent directly

valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

server computes
H(fam1ly | 458643) and
checks

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.