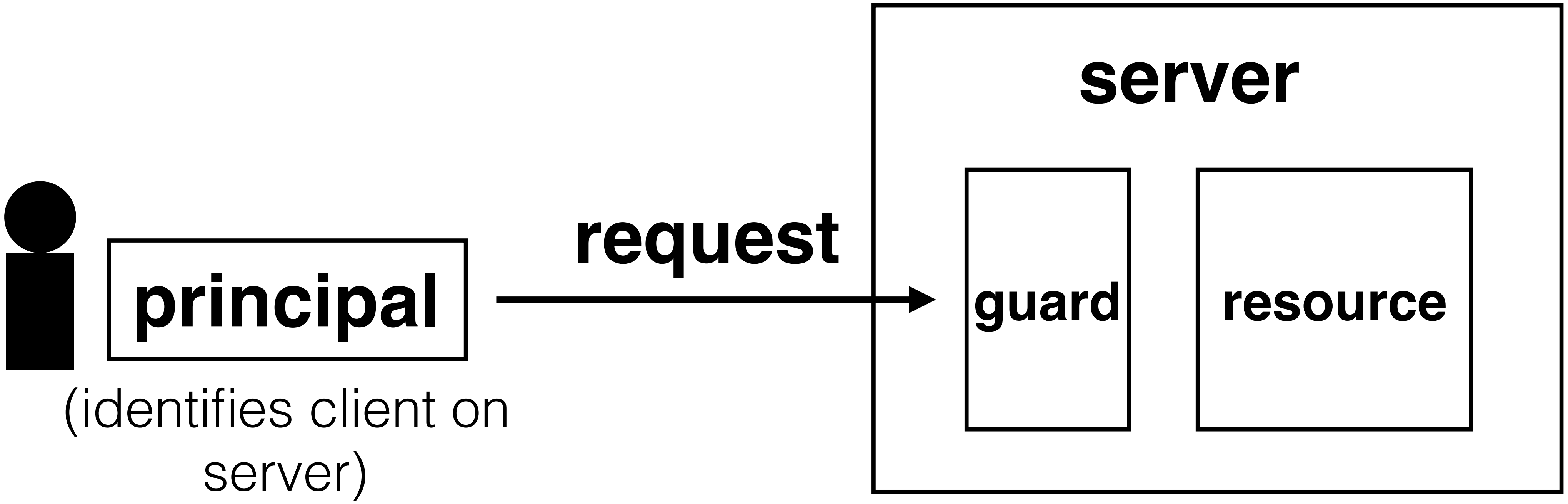
6.033 Spring 2017

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?

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	JXYWVPkpoQ6W1tbA21t6c66G4QUoWAS
roman	Xn5U1QvQz5MGOzdfJWgF8OiDFv1q7qe
tej	lo5WIidPPZePoSyMB2O.fUz3fLeZkm

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

top 10 passwords from a leak of 32 million passwords in 2009

source: Imperva, “Consumer Passwords Worst Practices”

password	number of users
123456	290,731
12345	79,078
123456789	76,790
Password	61,958
iloveyou	51,622
princess	35,231
rockyou	22,588
1234567	21,726
12345678	20,553
abc123	17,542

password usage has not improved in recent years. see, e.g.,
https://www.yahoo.com/tech/here-are-500-passwords-you-probably-shouldnt-be-using-96467697789.html
http://adamcaudill.com/2012/07/12/yahoos-associated-content-hacked/
http://www.huffingtonpost.com/2012/06/08/linkedin-password-leak-infographic_n_1581620.html
http://blogs.wsj.com/digits/2010/12/13/the-top-50-gawker-media-passwords/

username	slow_hash(password)
dom	gamynjSAIeYZ4iOBT4uaO3r5ub8O
han	JXYWVPkpoQ6W1tbA21t6c66G4QUoWAS
roman	Xn5U1QvQz5MGOzdfJWgF8OiDFv1q7qe
tej	lo5WIidPPZePoSyMB2O.fUz3fLeZkm

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

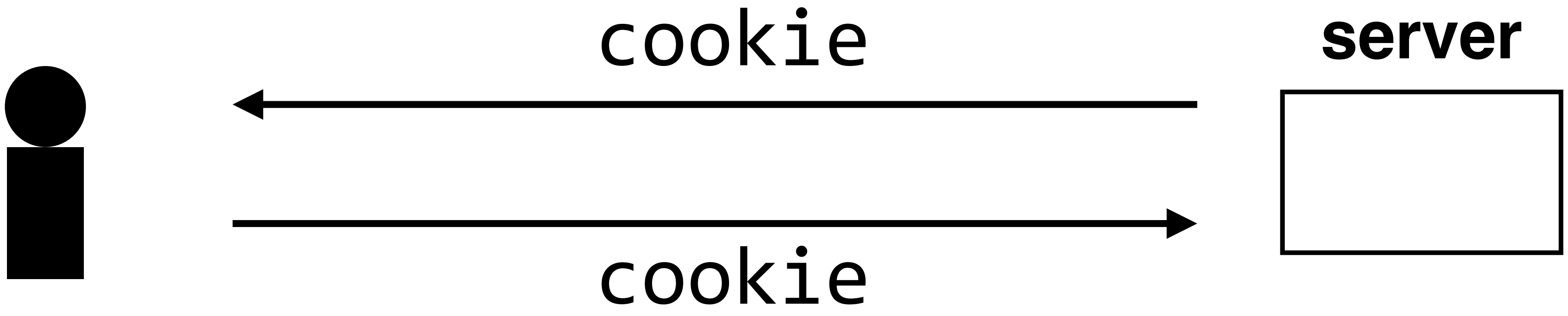
problem: adversary can still create rainbow tables for the most common passwords stored in plaintext

```
username | salt                         | slow_hash(password | salt)
dom      | LwVx6kO4SNY3jPVfOpfYe.       | M4ayLRWuzU.sSQtjoteIrIjNXI4UXta
han      | UbDsytUST6d0cFpmuhWu.e       | Y8ie/A18u9ymrS0FgVh9IOVx2Qe48lO
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
```

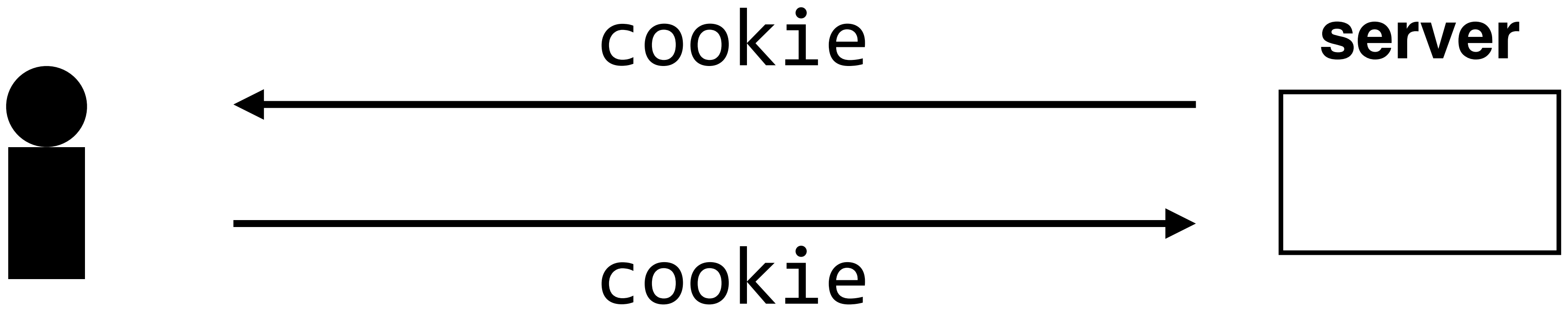
adversary would need a separate rainbow table for every possible salt

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

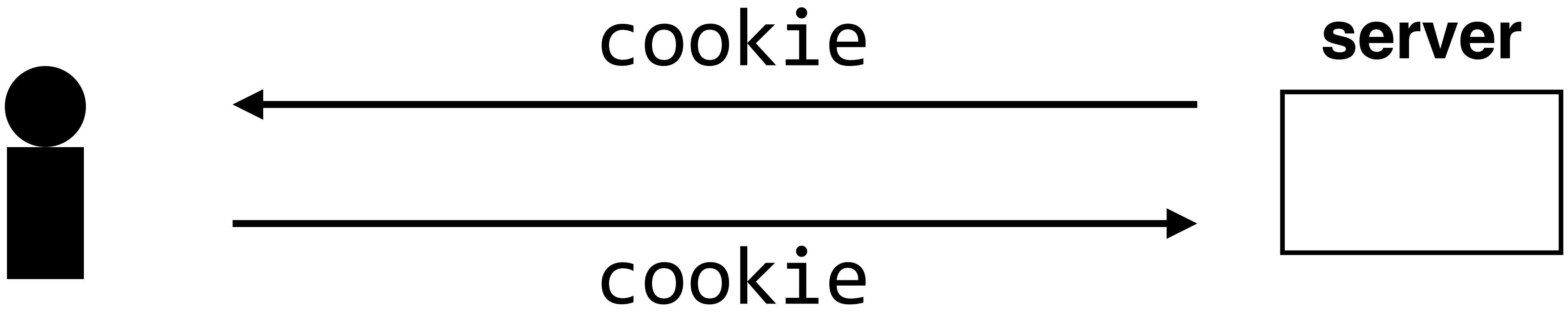
how can we avoid transmitting the password over and over?



`cookie = {username, expiration} ?`

problem: adversaries could easily create their own cookies

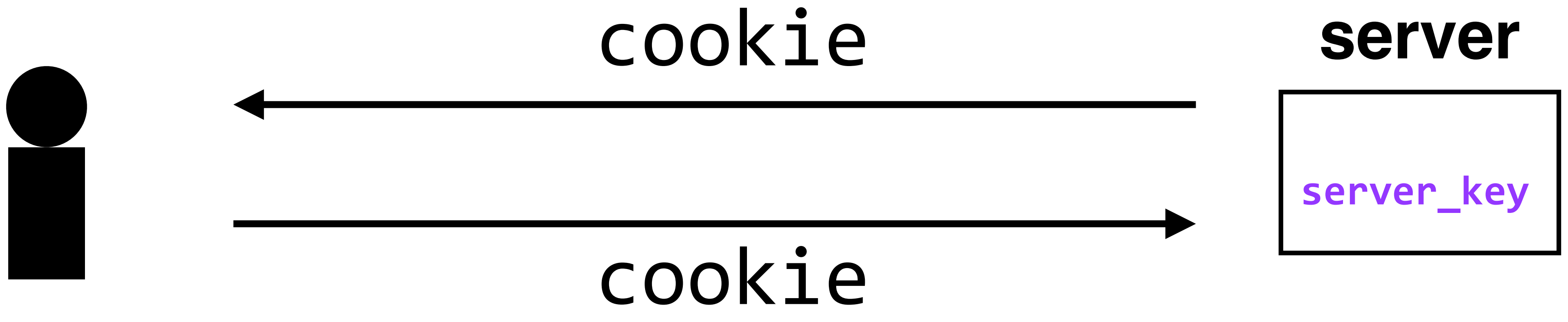
how can we avoid transmitting the password over and over?



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

problem: adversaries could still easily create their own cookies

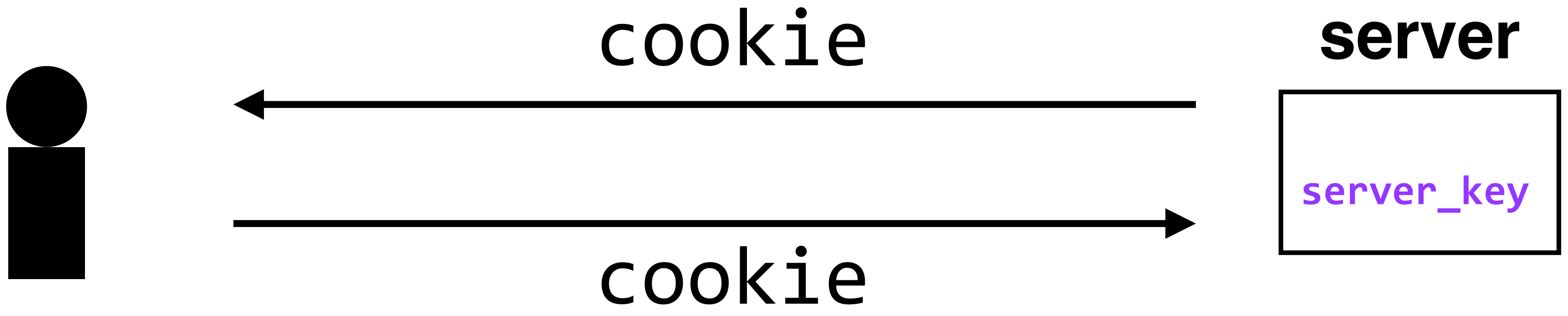
how can we avoid transmitting the password over and over?



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

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

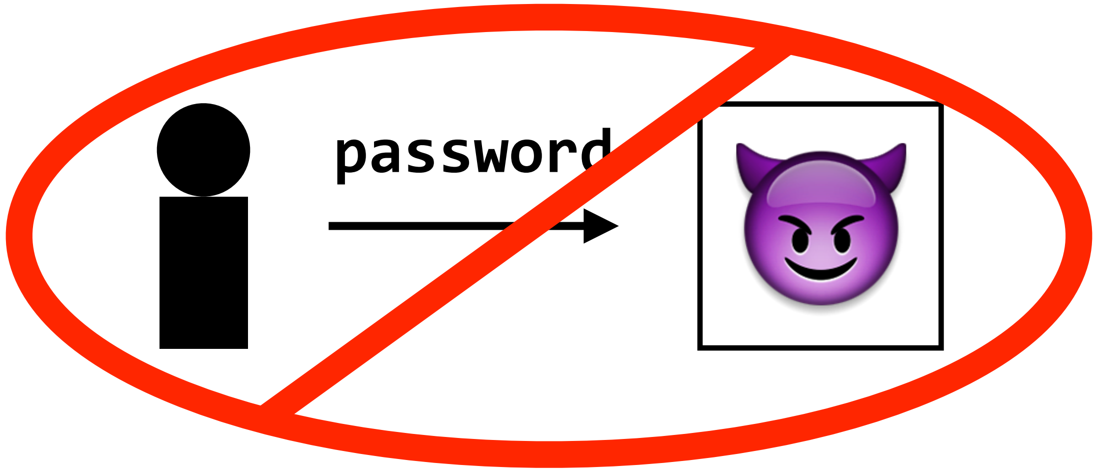
how can we avoid transmitting the password over and over?



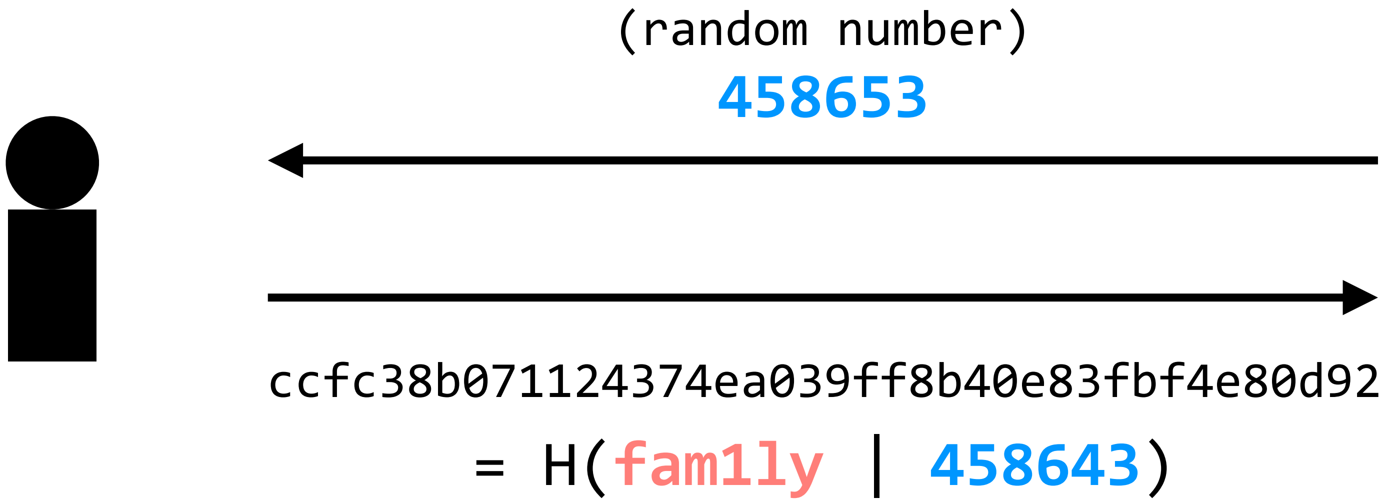
{username, expiration, H(server_key | username | expiration)}

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



ccfc38b071124374ea039ff8b40e83fbf4e80d92

= H(fam1ly | 458643)

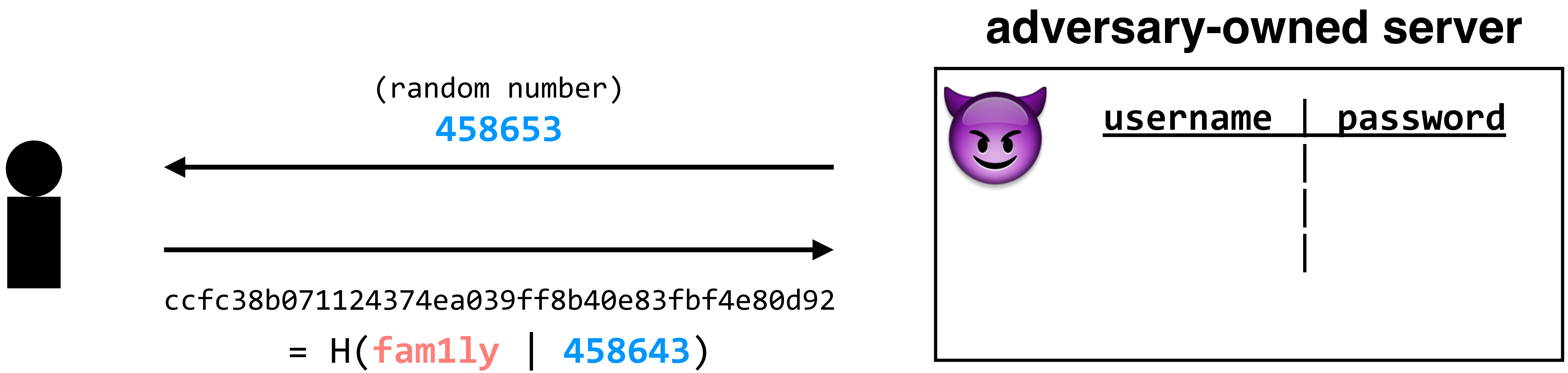
password is never sent directly

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

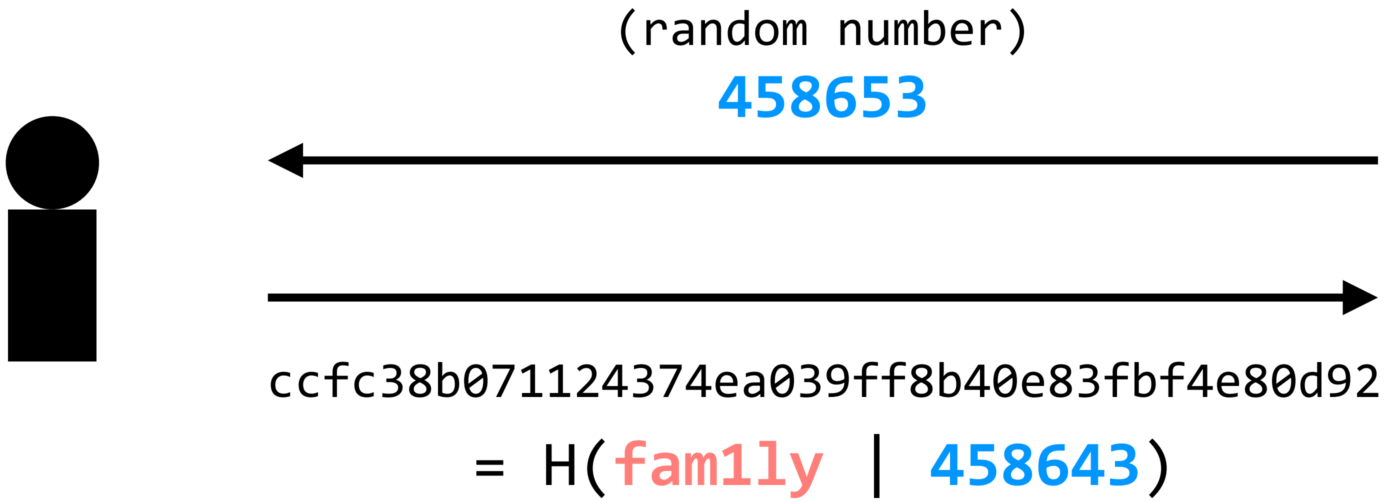
(random number)
458653

ccfc38b071124374ea039ff8b40e83fbf4e80d92
= H(fam1ly | 458643)

username | password

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

challenge-response protocol



valid server

username	password
dom	fam1ly
han	dr1ftnNt0ky0
roman	Lamb0s4ever
tej	31173h4ck3r

password is never sent directly

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.