6.033 in the news

APRIL 30, 2021

Scientists discover new vulnerability affecting computers globally

by Audra Book, University of Virginia School of Engineering and Applied Science

A team of University of Virginia School of Engineering computer science researchers has uncovered a line of attack that breaks all Spectre defenses, meaning that billions of computers and other devices across the globe are just as vulnerable today as they were when Spectre was first announced. The team reported its discovery to international chip makers in April and will present the new challenge at a worldwide computing architecture conference in June.

The researchers, led by Ashish Venkat, William Wulf Career Enhancement Assistant Professor of Computer Science at UVA Engineering, found a whole new way for hackers to exploit something called a "micro-op cache," which speeds up computing by storing simple commands and allowing the processor to fetch them quickly and early in the speculative execution process. Micro-op caches have been built into Intel computers manufactured since 2011.

Venkat's team discovered that hackers can steal data when a processor fetches commands from the micro-op cache.

https://techxplore.com/news/2021-04-scientists-vulnerability-affecting-globally.html

6.033 Spring 2021

Lecture #20: Authentication

authenticating users through the magic of (salted) hash functions

Katrina LaCurts | lacurts@mit.edu | 6.033 2021

steps towards building a more secure system

steps towards building a more secure system

1. be clear about goals (**policy**)

steps towards building a more secure system

1. be clear about goals (**policy**)
2. be clear about assumptions (**threat model**)

policy: provide **authentication** for users

username	password
user1	fhxiyfioncvgxvrb
user2	pmhdtzyxjpwbsjwv
user3	gxrdqlbxrvhdopjg
user4	hncobhdvimrtbpmy
user5	jprfyxpznunmbabp
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	mwccvpbesqqkuwsu
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	qopgbawviyrdzjhz
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	vlcfilrfcdjgtvqn
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	abpuvcbxqaoeaujq
user23	ykzpwiijhisbqlpw

policy: provide **authentication** for users

username	password
user1	fhxiyfioncvgxvrb
user2	pmhdtzyxjpwbsjwv
user3	gxrdqlbxrvhdopjg
user4	hncobhdvimrtbpmy
user5	jprfyxpznunmbabp
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	mwccvpbesqqkuwsu
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	qopgbawviyrdzjhz
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	vlcfilrfcdjgtvqn
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	abpuvcbxqaoeaujq
user23	ykzpwiijhisbqlpw

policy: provide **authentication** for users

to authenticate users:

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

username	password
user1	fhxiyfioncvgxvrb
user2	pmhdtzyxjpwbsjwv
user3	gxrdqlbxrvhdopjg
user4	hncobhdvimrtbpmy
user5	jprfyxpznunmbabp
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	mwccvpbesqqkuwsu
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	qopgbawviyrdzjhz
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	vlcfilrfcdjgtvqn
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	abpuvcbxqaoeaujq
user23	ykzpwiijhisbqlpw

policy: provide **authentication** for users

to authenticate users:

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

is this what passwords really look like?

username	password
user1	fhxiyfioncvgxvrb
user2	111111
user3	123456
user4	hncobhdvimrtbpmy
user5	password
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	123456
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	111111
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	123456
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	password
user23	ykzpwiijhisbqlpw

highlighted users are using common passwords

to authenticate users:

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

is this what passwords really look like?

absolutely not. in reality, some passwords are *far more common* than others

username	password
user1	fhxiyfioncvgxvrb
user2	111111
user3	123456
user4	hncobhdvimrtbpmy
user5	password
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	123456
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	111111
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	123456
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	password
user23	ykzpwiijhisbqlpw

highlighted users are using common passwords

policy: provide **authentication** for users

to authenticate users:

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

username	password
user1	fhxiyfioncvgxvrb
user2	111111
user3	123456
user4	hncobhdvimrtbpmy
user5	password
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	123456
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	111111
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	123456
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	password
user23	ykzpwiijhisbqlpw

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

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

username	password
user1	fhxiyfioncvgxvrb
user2	111111
user3	123456
user4	hncobhdvimrtbpmy
user5	password
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	123456
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	111111
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	123456
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	password
user23	ykzpwiijhisbqlpw

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

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

problem: the adversary (with access to the stored table) can just read the passwords directly

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

H is **collision-resistant**: if $x_1 \neq x_2$, then the probability that $H(x_1) = H(x_2)$ is virtually zero

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

H is **collision-resistant**: if $x_1 \neq x_2$, then the probability that $H(x_1) = H(x_2)$ is virtually zero

H is **one-way**: given *x*, it is easy to compute *H(x)*. given *H(x)* (without *x*), it is virtually impossible to determine *x*

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

H is **collision-resistant**: if $x_1 \neq x_2$, then the probability that $H(x_1) = H(x_2)$ is virtually zero

H is **one-way**: given *x*, it is easy to compute *H(x)*. given *H(x)* (without *x*), it is virtually impossible to determine *x*

there are **very few** such functions, and the ones that exist are well-known

in the context of our quest for authentication, this means that we cannot rely on a “secret“ hash function; there is no such thing

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

H is **collision-resistant**: if $x_1 \neq x_2$, then the probability that $H(x_1) = H(x_2)$ is virtually zero

H is **one-way**: given *x*, it is easy to compute *H(x)*. given *H(x)* (without *x*), it is virtually impossible to determine *x*

there are **very few** such functions, and the ones that exist are well-known

in the context of our quest for authentication, this means that we cannot rely on a “secret“ hash function; there is no such thing

we are going to use hash functions to (attempt to) solve our authentication problem

username	password
user1	fhxiyfioncvgxvrb
user2	111111
user3	123456
user4	hncobhdvimrtbpmy
user5	password
user6	abbsocpnkpnqirmf
user7	ybtsnxthnpunjrep
user8	eauohxeagxkejbyk
user9	viyigjfjdtllgdgd
user10	123456
user11	jpzxomdscorejjcb
user12	aqpyiegxyiftfpaa
user13	111111
user14	ordkyoqkugmrzudj
user15	ygeblazpolvfufox
user16	bslrsuhqmsgfvfjx
user17	pwcdmcpucurnzmjy
user18	123456
user19	jfkwdbgdprxnotuz
user20	cmhuajeqecvbdlyn
user21	tnfdfjiytbsomleb
user22	password
user23	ykzpwiijhisbqlpw

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

highlighted users are using common passwords

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

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

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594 d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad 5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf 0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999 c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc62927 73603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954 805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb41 57c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2 d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9 c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf 0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc98 79789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57 d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad 5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f71 2d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c 69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6 a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e9 4ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

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

now the adversary cannot just read passwords directly from the table

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

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

now the adversary cannot just read passwords directly from the table

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

password	hash(password)
	. . .
111111	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
123456	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
password	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
ybtsnxthnpunjrep	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
ygeblazpolvfufox	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
abbsocpnkpnqirmf	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
	. . .

an adversary can quickly make a table ahead of time, mapping passwords to their hashes, and use that to determine the user passwords

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

password	hash(password)
	. . .
111111	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
123456	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
password	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
ybtsnxthnpunjrep	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
ygeblazpolvfufox	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
abbsocpnkpnqirmf	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
	. . .

it takes so little time to calculate a hash that it's feasible for the attacker to have the hash of many of our passwords, not just the most common ones

(it took my laptop 8 seconds to compute 10 million hashes)

an adversary can quickly make a table ahead of time, mapping passwords to their hashes, and use that to determine the user passwords

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user3	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

highlighted users are using common passwords

password	hash(password)
	. . .
111111	bcb15f821479b4d5772bd0ca866c00ad5f926e3580720659cc80d39c9d09802a
123456	8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92
password	5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
ybtsnxthnpunjrep	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
ygeblazpolvfufox	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
abbsocpnkpnqirmf	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
	. . .

it takes so little time to calculate a hash that it's feasible for the attacker to have the hash of many of our passwords, not just the most common ones

(it took my laptop 8 seconds to compute 10 million hashes)

an adversary can quickly make a table ahead of time, mapping passwords to their hashes, and use that to determine the user passwords

key facts: given *x*, it is easy to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

interlude: hash functions

hash functions are not normal functions! they have a number of exciting properties

a **hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

H is **collision-resistant**: if $x_1 \neq x_2$, then the probability that $H(x_1) = H(x_2)$ is virtually zero

H is **one-way**: given *x*, it is easy to compute *H(x)*. given *H(x)* (without *x*), it is virtually impossible to determine *x*

there are **very few** such functions, and the ones that exist are well-known

in the context of our quest for authentication, this means that we cannot rely on a “secret“ hash function; there is no such thing

current problem: an adversary can easily pre-compute hashes for *a lot* of passwords

interlude: *slow* hash functions

hash functions are not normal functions! they have a number of exciting properties

a ***slow* hash function *H*** takes an input string of arbitrary size and outputs a fixed-length string

H is **deterministic**: if $x_1 = x_2$, then $H(x_1) = H(x_2)$

H is **collision-resistant**: if $x_1 \neq x_2$, then the probability that $H(x_1) = H(x_2)$ is virtually zero

H is **one-way**: given *x*, it is easy — **but slow** — to compute *H(x)*. given *H(x)* (without *x*), it is virtually impossible to determine *x*

there are **very few** such functions, and the ones that exist are well-known

in the context of our quest for authentication, this means that we cannot rely on a “secret“ hash function; there is no such thing

these slow hash functions are technically called “key derivation functions”, but we want to emphasize the relationship between them and the hash functions you’re already familiar with

they also have some additional properties, but not ones that we’re concerned with for this lecture

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

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

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password)
	. . .
111111	
123456	
password	
ybtsnxthnpunjrep	
ygeblazpolvfufox	
abbsocpnkpnqirmf	
	. . .

is it feasible for an adversary to make the same table as before?

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

highlighted users are using common passwords

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password)
	. . .
111111	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
123456	4JTDRz7y99aSELdzzcueTseTpO/5fA6
password	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
	. . .

is it feasible for an adversary to make the same table as before?

yes, but for fewer passwords

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password)
	. . .
111111	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
123456	4JTDRz7y99aSELdzzcueTseTpO/5fA6
password	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
	. . .

because it takes much longer to calculate a single hash, there's less incentive for an attacker to include uncommon passwords

(it's not that they *can't*, but it would take a very long time)

is it feasible for an adversary to make the same table as before?

yes, but for fewer passwords

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password)
	. . .
111111	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
123456	4JTDRz7y99aSELdzzcueTseTpO/5fA6
password	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
	. . .

because it takes much longer to calculate a single hash, there's less incentive for an attacker to include uncommon passwords

(it's not that they *can't*, but it would take a very long time)

is it feasible for an adversary to make the same table as before?

yes, but for fewer passwords

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

we would like to make it more difficult for adversaries to pre-compute hashes, even for common passwords

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	slow_hash(password)
user1	qAkXG2gGdJlx.xozulzEKnhXbtKbsya
user2	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user3	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user4	2CxFUAKBJsJLm.maayIhsb7lJvHIXXi
user5	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user6	PzHog2J/0ya8onDrA4MzZFKN7Orrph.
user7	ZU/1z6GKB0L7zNNraJ8P4y4mXrU6kne
user8	G7pXMqNyiRcjjSj/nZLwlX00UZjhLg6
user9	UcY2dIlUa9gGSNY3n35YBRZs5dS//hq
user10	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user11	brZbM2NgvCu.jOOJo5F0KO5rJs42KLi
user12	x0MqBzW5PQnpW9iGrYbLq5ztNjQVT9i
user13	oyMB4m/7QcaQMvxP2TQt0zWCdTzBn0m
user14	hsHIdCPh2o/HjcdE9uiUftKGvonuLQ.
user15	GnF8kGCIOt1ds7l0Jj57bcyiJuVJbsC
user16	VCrN2r8DevjEnWzuKAKcTpmzrkOSywi
user17	U85eomVd1xESFUiahkh4rAOBfZAvzzq
user18	4JTDRz7y99aSELdzzcueTseTpO/5fA6
user19	zQDC3VWHX2T2y8K1HMtYwEoTzA7rwIe
user20	kpTnPCXD062S7a7MEYLSdWWnTw/ls.q
user21	wB3zo4Ies6lrM46Sk.f6TixOOSJlEB2
user22	ptHn4kJl89vKJorSOCLKq0mEqVsFVka
user23	.iSZZDUgdclmZWbA3z5OJDuQF.NlfyG

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

we would like to make it more difficult for adversaries to pre-compute hashes, even for common passwords

idea: add randomness

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	salt
user1	TU6kbcuPm7jA./IQYZG.8O
user2	y7oSC2QsrvXTzEZlDZFdwu
user3	4ncYRSB5v3rWiU1nPpA0iu
user4	SK9H4x4Ha0Owz4NOTwj20.
user5	j8YyeDX.9GnsT5Hu94z7tO
user6	CIqY72CGM8KNQId1CqXY7.
user7	OGtMXrEZEx0L544Odvrhbe
user8	RFeT9TVol8cmpQdhqMCV5.
user9	rDAhDK5V6n3TUS3ahf2Z9e
user10	nvOYvT0/oczOW5lmbVZSUO
user11	yNL/e3PpBsfBYgwi0Ai/gu
user12	1zroUl0scwDzgG3GY86pFO
user13	TAkv7nBQ5amY4V.aIjez0u
user14	1J796dTzufUC8ItVIKIyOu
user15	/x.Vk/XhUILbk3XjgyVyfO
user16	hyg8T0JPDX3dCf92Zkx4Yu
user17	YbaYOSdkA01IF.drWa6CXO
user18	yaE.gULeQg.K2SelX191Q.
user19	NLt0SA/QPo2IIbtb7G56lO
user20	RFFSWUGGFeX5XNyW8rLToe
user21	YWEgwinWuKrNUFvgzQKUNe
user22	ukqUgoOZWCqIQjH3DwC4xe
user23	sPRFpmFnu5G4lAPUkV0wrO

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

we're going to associate a random value — a *salt* — with each user. these salts are stored in plaintext; they are not a secret

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

we're going to associate a random value — a *salt* — with each user. these salts are stored in plaintext; they are not a secret

instead of storing a hash of the password, we will concatenate the password and the salt, and store the hash of that string

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

to authenticate users:

```
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
```

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password|salt)
. . .	
111111	?
123456	?
password	?
. . .	

is it feasible for an adversary to make the same table as before?

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password|salt)
. . .	
111111	?
123456	?
password	?
. . .	

is it feasible for an adversary to make the same table as before?

to pre-compute the table, the adversary would need a separate table for *every possible salt*; this is infeasible

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

password	slow_hash(password|salt)
. . .	
111111	?
123456	?
password	?
. . .	

is it feasible for an adversary to make the same table as before?

to pre-compute the table, the adversary would need a separate table for *every possible salt*; this is infeasible

to target a specific user — say user1 — an adversary could read user1's salt and make a table using that salt. if user1 was using a common password it would be revealed, but the table can't be pre-computed without knowing the salt

key facts: given *x*, it is easy but slow to compute *H(x)*; given *H(x)*, it is virtually impossible to determine *x*. there are **very few** such functions, and the ones that exist are well-known

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

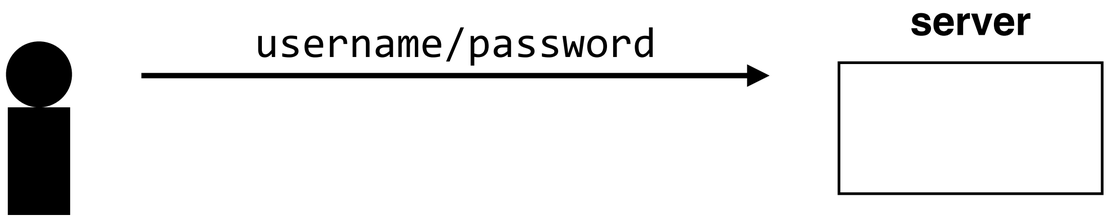
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

server

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

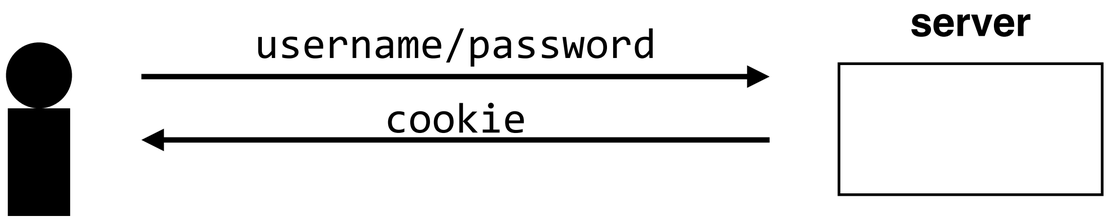
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords



limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

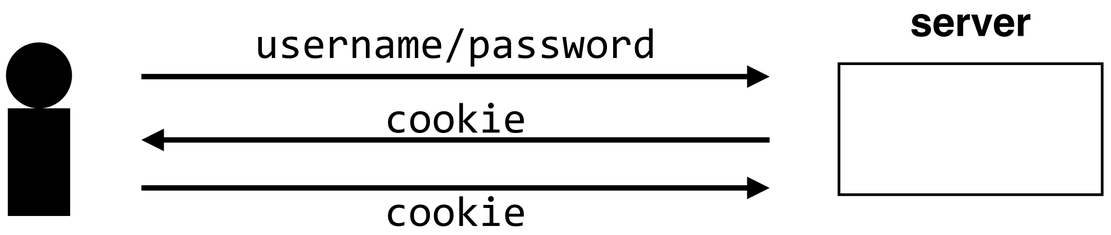
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords



limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

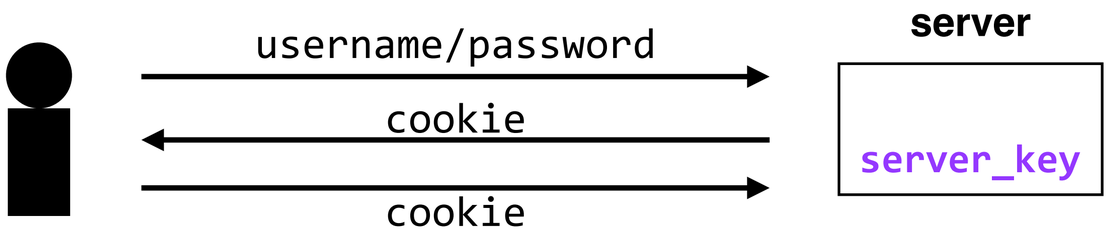
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords



limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

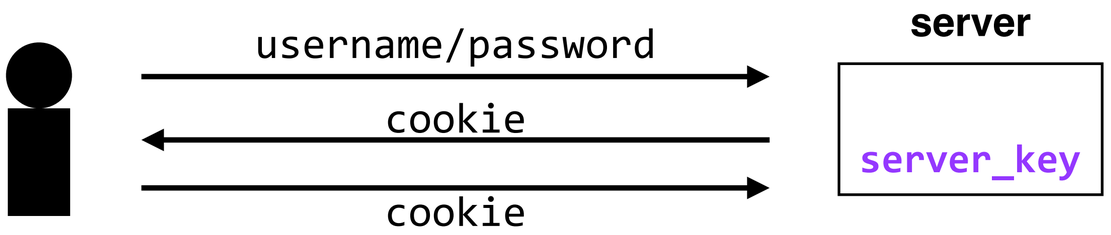
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords



limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords



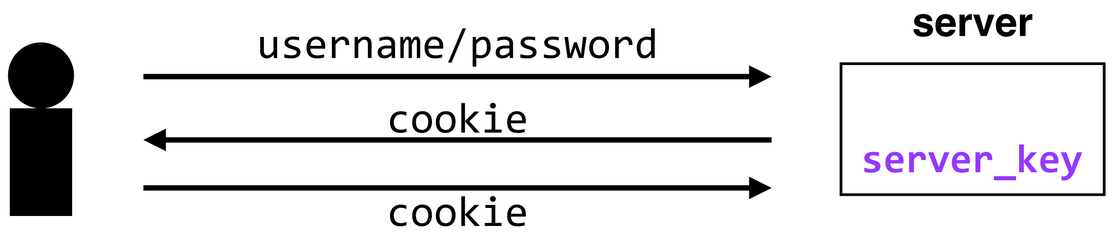
`cookie = {username, expiration, H(server_key | username | expiration)}`

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



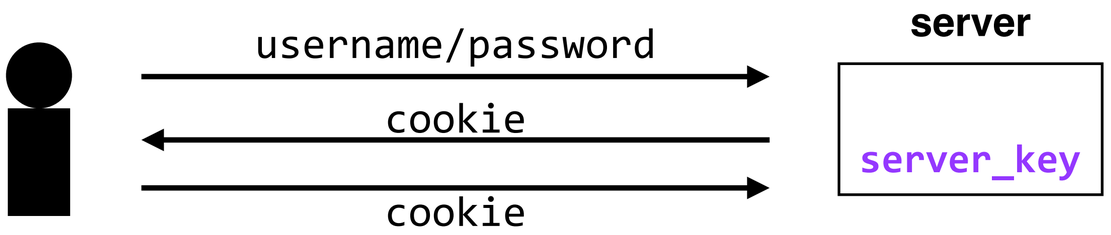

```
cookie = {username, expiration, H(server_key | username | expiration)}
```

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time




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

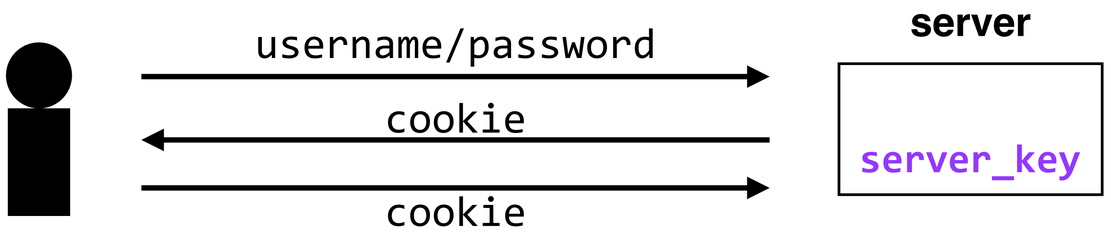
challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

valid server

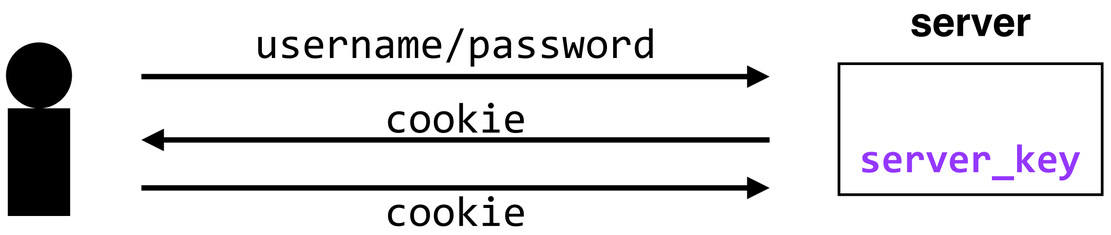
username	password
user1	fhxiyfioncvgxvrb

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

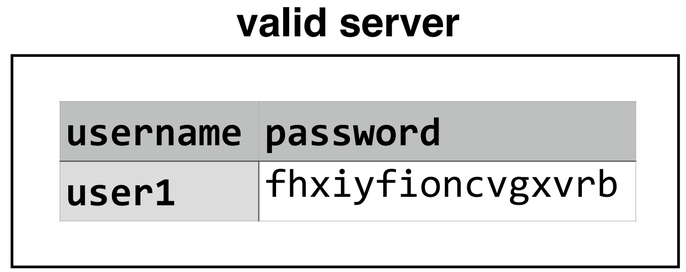
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

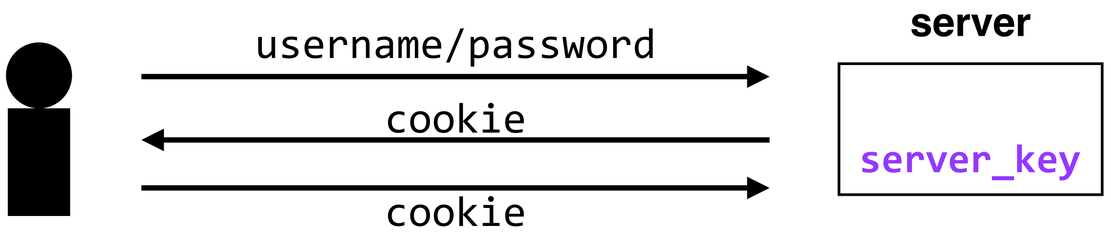


limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

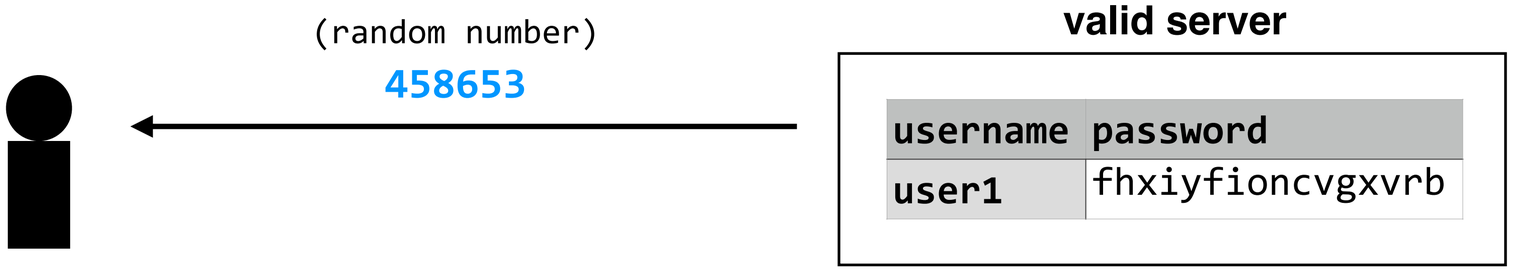
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

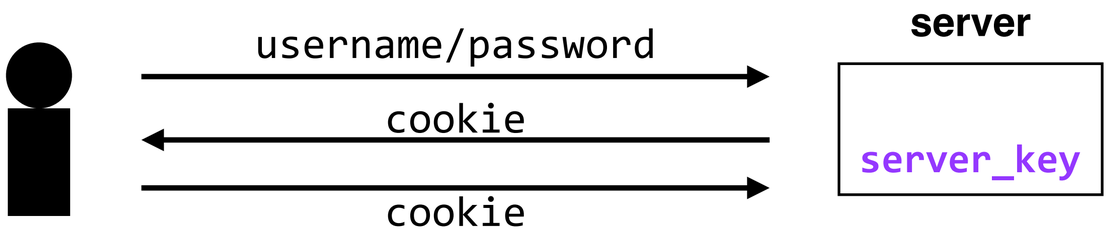


limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

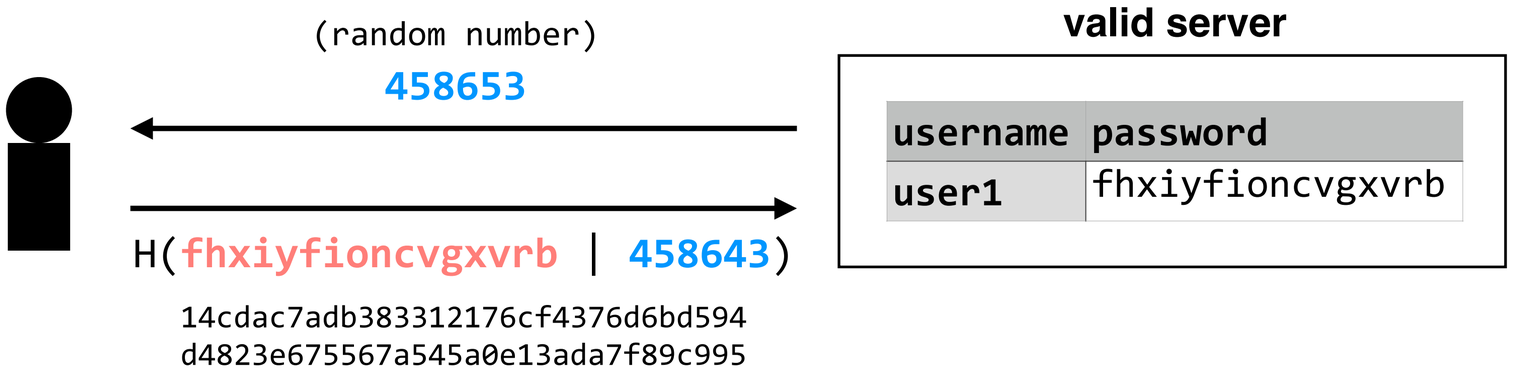
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

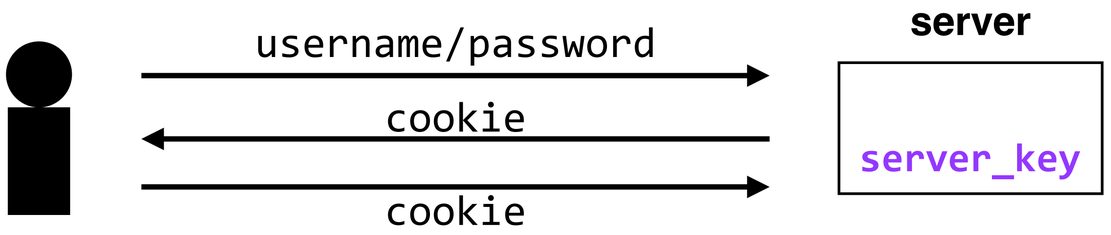


limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

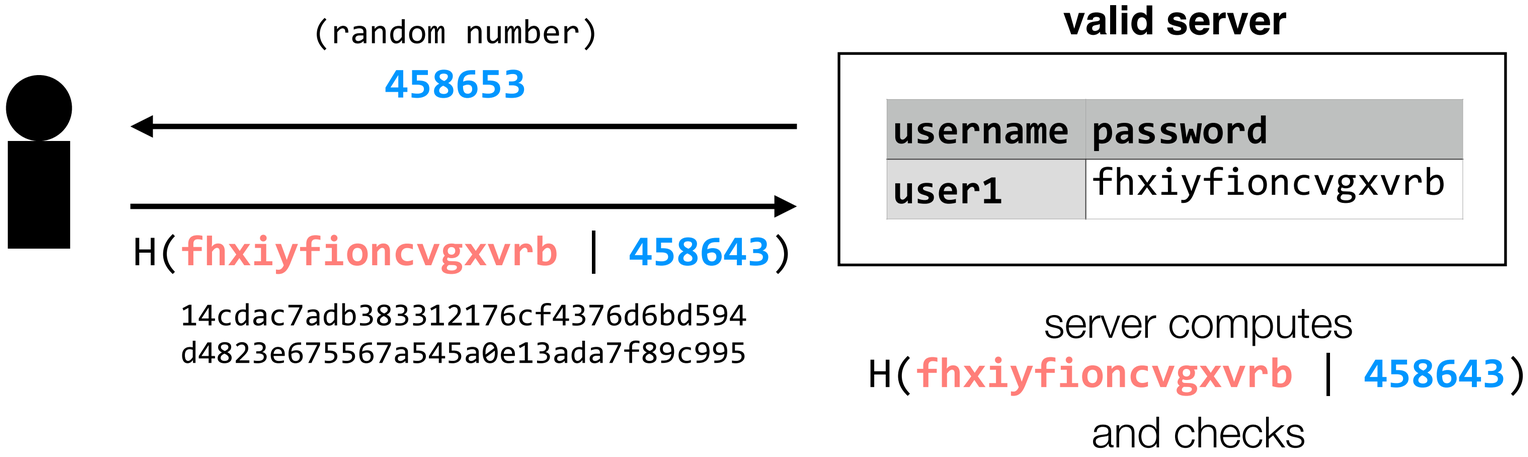
cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

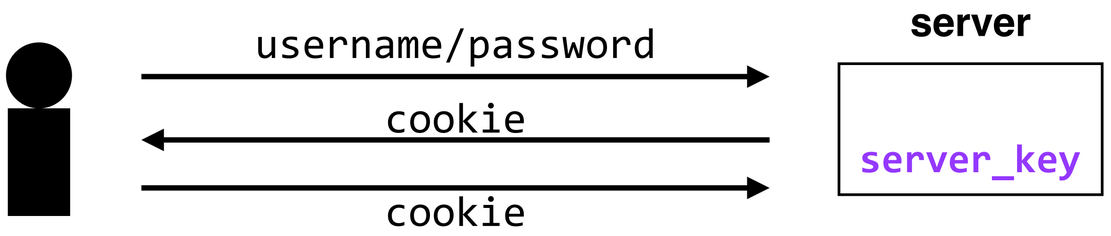


limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

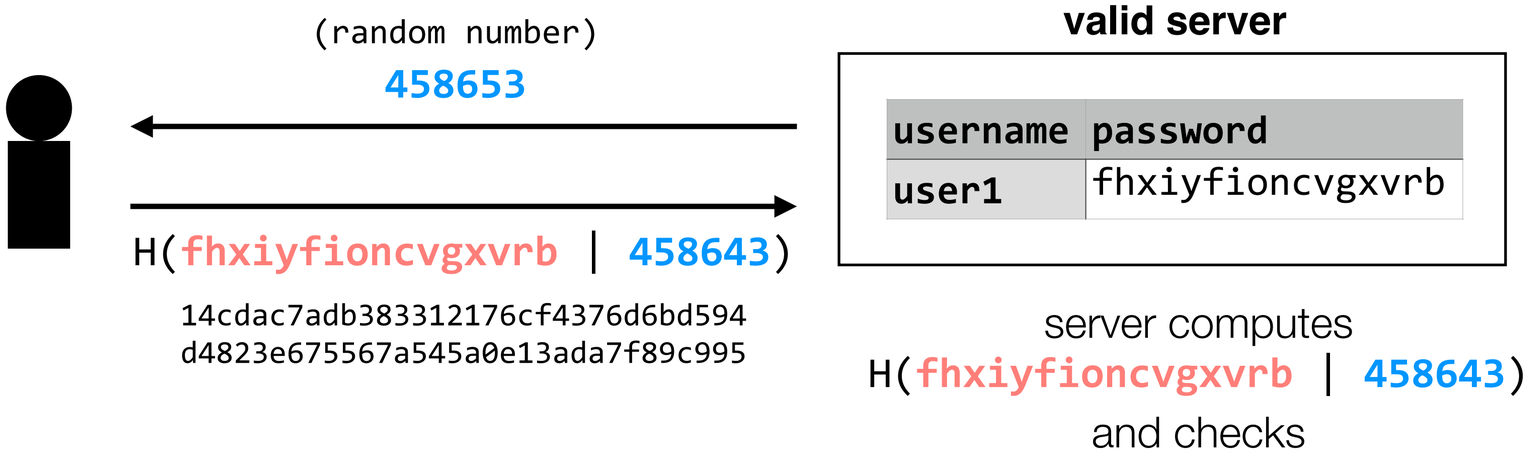
threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

threat model: adversary owns a server, and is trying to convince the user to send their password directly to the adversary

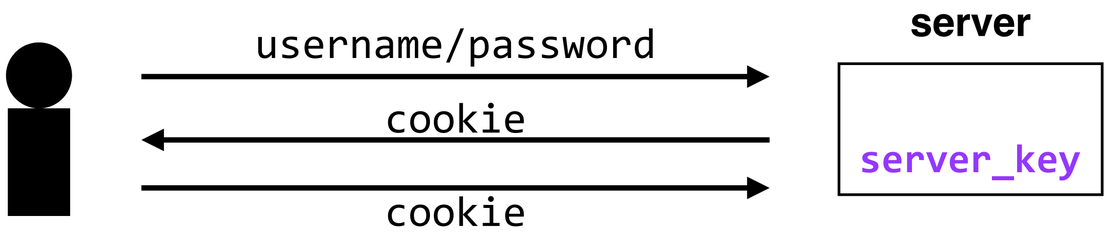


limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

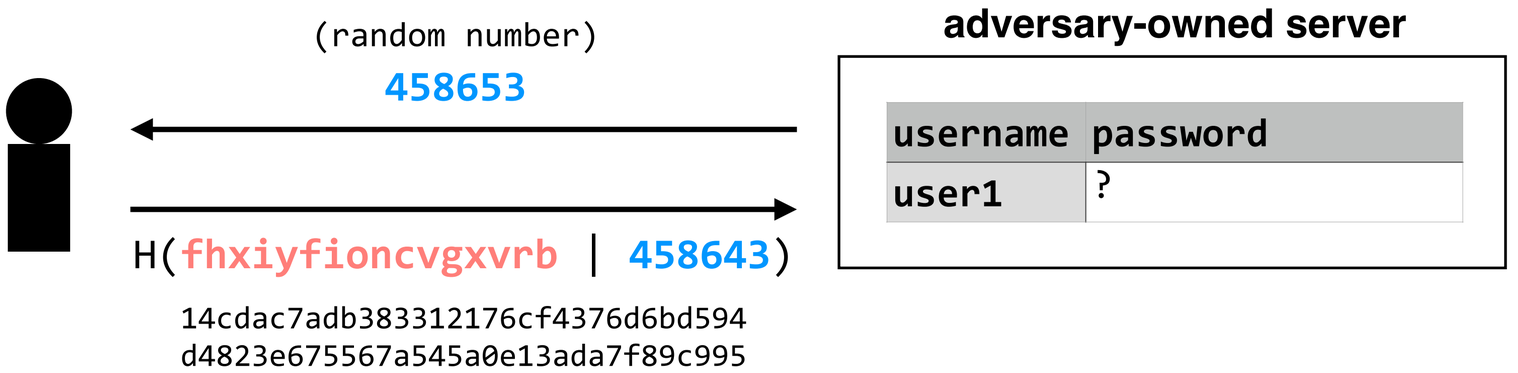
threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time




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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

threat model: adversary owns a server, and is trying to convince the user to send their password directly to the adversary

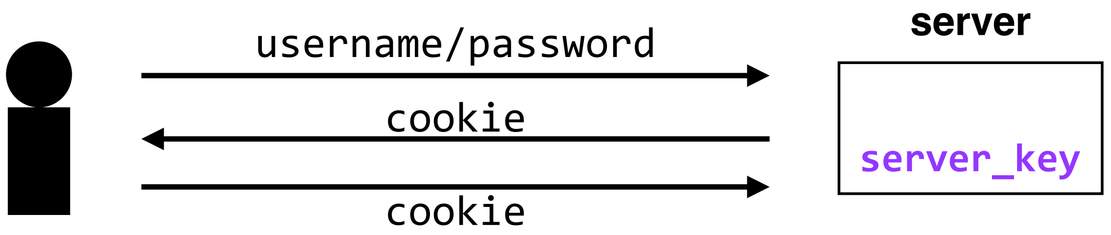


limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

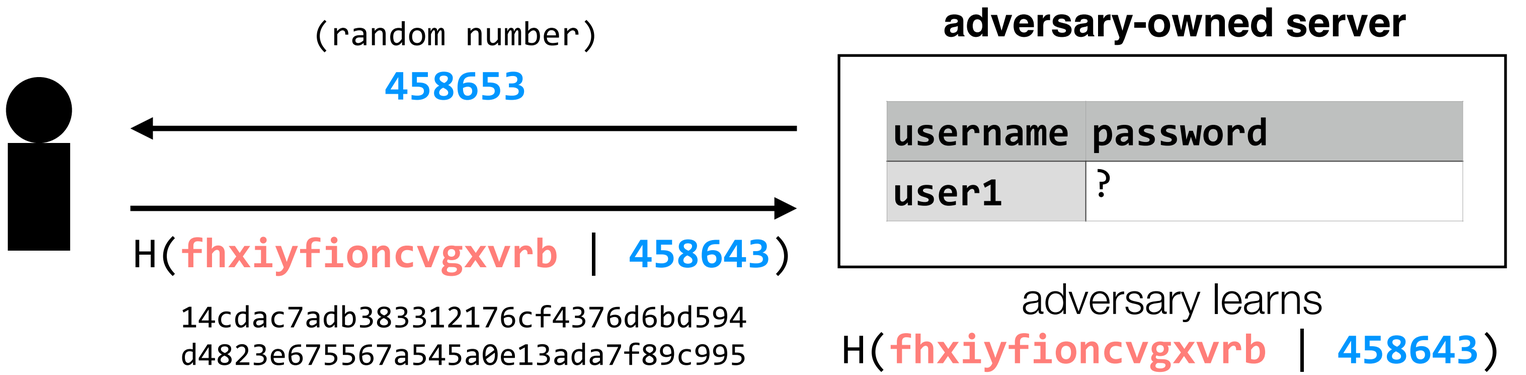
threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

threat model: adversary owns a server, and is trying to convince the user to send their password directly to the adversary



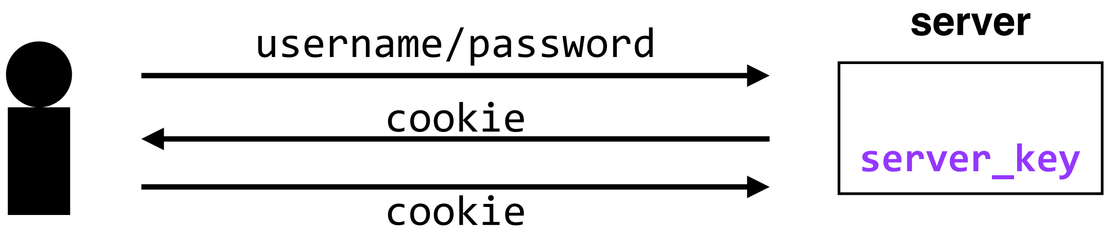
adversary learns
H(fhxiyfioncvgxvrb | 458643)
but can't reverse-engineer the password
(hashes are one-way)

limiting transmission of passwords

we want users to transmit passwords infrequently, because transmitting a password repeatedly can open a user up to other attacks outside of our current threat model

cookies allow users to authenticate themselves from some period of time, without repeatedly retransmitting their passwords

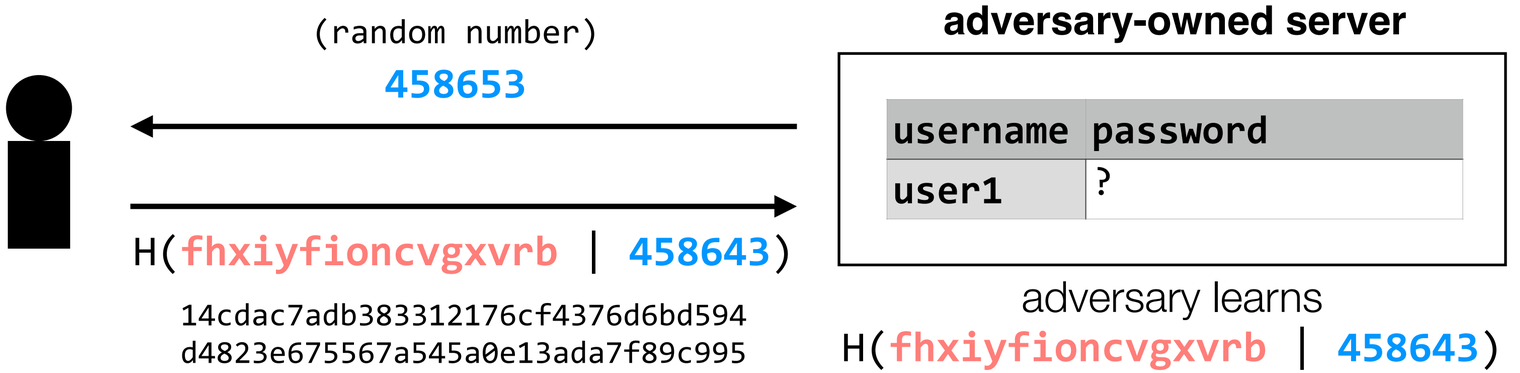
threat model: adversary can observe the cookie and is trying to learn the user's password, or to learn enough information such that it could create cookies that were valid for a long period of time



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

challenge-response protocols authenticate users without ever transmitting the password, and can help combat phishing attacks

threat model: adversary owns a server, and is trying to convince the user to send their password directly to the adversary



adversary learns
H(fhxiyfioncvgxvrb | 458643)
but can't reverse-engineer the password
(hashes are one-way)

in this specific setup, problems arise when the valid server is using salted hashes (as it should be), but challenge-response protocols exist to handle that case

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

are there alternatives to passwords?

username	salt	slow_hash(password|salt)
user1	TU6kbcuPm7jA./IQYZG.8O	rBda9fbnXhUCWi6c9MjlUtQFlK1I4Sq
user2	y7oSC2QsrvXTzEZlDZFdwu	tjFcpSrZN6ryOYueyrAtUfFnFaOui2C
user3	4ncYRSB5v3rWiU1nPpA0iu	hacrgRlfU44c9XnBckef2fu.ifuB.Ya
user4	SK9H4x4Ha0Owz4NOTwj20.	wWk3GjGeMspoqy3VcMghpbkE5OjHQXS
user5	j8YyeDX.9GnsT5Hu94z7tO	ViflhwGHl.5H3j0mawzBPdKTiXf5L6.
user6	CIqY72CGM8KNQId1CqXY7.	stk3mDJDaaH9Nfgf/ePJrkRoK15.Heu
user7	OGtMXrEZEx0L544Odvrhbe	A.7NaJc21Y6I3J6rdJtiIJXVpMvaMgG
user8	RFeT9TVol8cmpQdhqMCV5.	yVzcp0OjXBoNjcMWHpAxVulFqdM5W9m
user9	rDAhDK5V6n3TUS3ahf2Z9e	Af4wBH1YqLTvxrhBgVGP85IALXRya3C
user10	nvOYvT0/oczOW5lmbVZSUO	b4miFmYcRy0/TFVhttntbrrLPLjFDKu
user11	yNL/e3PpBsfBYgwi0Ai/gu	bbT5sTcmsklsyXVILfVdJ/HAIEonb..
user12	1zroUl0scwDzgG3GY86pFO	MG5LtQ6m/c4gVxbLalpPIJ4O3eXFPry
user13	TAkv7nBQ5amY4V.aIjez0u	LHPo8.0XJDGleWWgG87nPvY8/vNPa2G
user14	1J796dTzufUC8ItVIKIyOu	pAI7ZRWvVOhxBVW/sttFquJCl/74LTC
user15	/x.Vk/XhUILbk3XjgyVyfO	zx1P3YgW8d9m1n9lZ6GW7jsbBALniWi
user16	hyg8T0JPDX3dCf92Zkx4Yu	5Oh.8uSUrokBgqnByYYH/mDEH7my98C
user17	YbaYOSdkA01IF.drWa6CXO	ZKbZQtEh4UNoTflWsXs9hZ7wbnnzgC.
user18	yaE.gULeQg.K2SelX191Q.	E/syZIC.1.zg5.ZTMZwWX/RmkvpipNu
user19	NLt0SA/QPo2IIbtb7G56lO	eOX2p48XcKRXKFY87f56h3W.UEeO7Gi
user20	RFFSWUGGFeX5XNyW8rLToe	0W94ciFDN5stvqVzYsli4t/SNA2pwhS
user21	YWEgwinWuKrNUFvgzQKUNe	yatU0vWN//72Ul8OdxGHnClTLWdTfXe
user22	ukqUgoOZWCqIQjH3DwC4xe	jg1.OSatbZooR6l4taWv3HBpXNN5Xp2
user23	sPRFpmFnu5G4lAPUkV0wrO	mVpzAYGXEgs583nG894R98k1S3YmP1q

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

are there alternatives to passwords?

yes, and they have their own trade-offs

using passwords to provide **authentication** takes some effort. storing **salted hashes**, incorporating **session cookies** and **challenge-response protocols** can help mitigate common attacks

passwords provide more **general lessons** about security: consider human factors when designing secure systems, in particular

there are always **trade-offs**. many proposed improvements on passwords do add security, but often add complexity and decrease usability