6.1800 Spring 2025

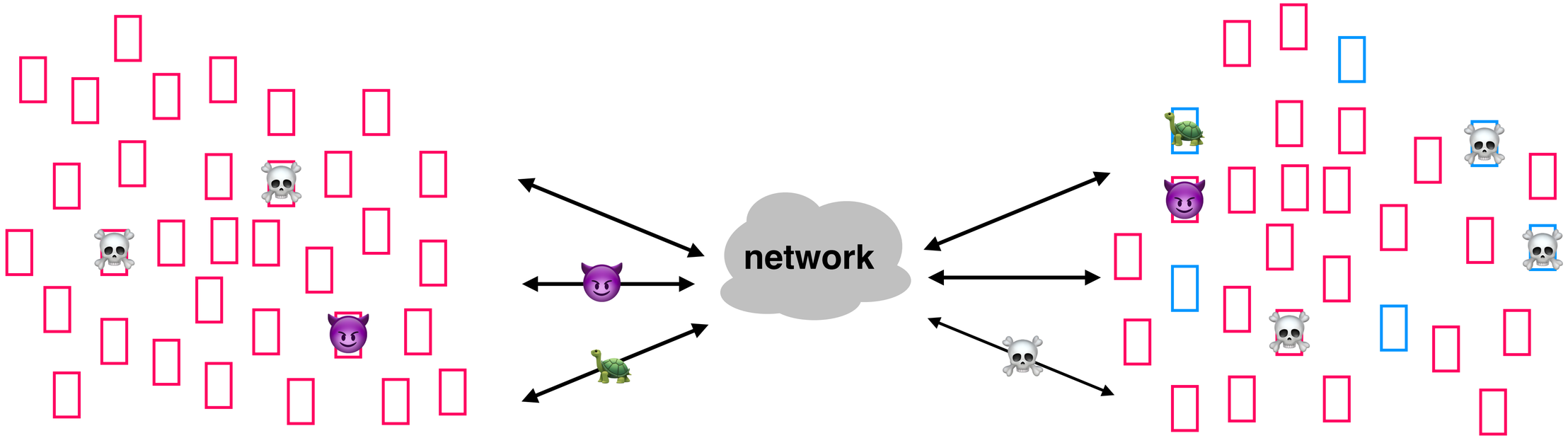
Lecture #21: Authentication

authenticating users through the magic of (salted) hash functions

Katrina LaCurts | lacurts@mit.edu | 6.1800 2025

scalability: how does our system behave as we increase the number of machines, users, requests, data, etc.?

fault-tolerance/reliability: how does our system deal with failures (☠️)? machines crashing, network links breaking, etc.



security: how does our system cope in the face of targeted attacks (😈)?

performance: how do we define our performance requirements, and know if our system is meeting them? what do we do if performance is subpar (🐢)?

who is impacted by our design and implementation choices?
who makes those choices?

steps towards building a more secure system

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

username	password (plaintext)
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
user24	inztzlzqdtzfmblv
user25	bzemazvpqtuhukhk

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

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	hash(password)
user1	14cdac7adb383312176cf4376d6bd594 d4823e675567a545a0e13ada7f89c995
user2	adab20bca80250418a02196ca80d01d8 5456675b29816a40212eaa8d4c56ad22
user3	02c0d53273d41d74221c69c539315796 5182254904aa0653aa43f3a206e6eaa1
user4	21f110ed053c24004463526b3ba06999 c52580abeff8a85e89f7d9af966446e3
user5	2fae96a772f8e7c1bc5ae87ba291e467 1b66792a024b324e87305ca551c34a80
user6	e5034d3265313a5b4fbac596b1e3f954 805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb41 57c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2 d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9 c5d2f8d39ff106035d74757c140b527d
user10	1f7dbe35470e3f7364736afb9a02787c 832ea8b5cdd958d53d27aa69c834e63b
user11	062f82f9ef701cfa2573aa40f2e7cc98 79789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57 d27d29648e770deb91dadb4b6ddd5f3a
user13	92df155300535fa28fefe4a65f2059fb 3bedfe6409d01cbf29dd45b1163234b4
user14	964205c227974d9048fe67a485f73f71 2d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c 69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6 a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e9 4ae5d8f1948f6de483801fa12ed4056b

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

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

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594 d4823e675567a545a0e13ada7f89c995
user2	adab20bca80250418a02196ca80d01d8 5456675b29816a40212eaa8d4c56ad22
user3	02c0d53273d41d74221c69c539315796 5182254904aa0653aa43f3a206e6eaa1
user4	21f110ed053c24004463526b3ba06999 c52580abeff8a85e89f7d9af966446e3
user5	2fae96a772f8e7c1bc5ae87ba291e467 1b66792a024b324e87305ca551c34a80
user6	e5034d3265313a5b4fbac596b1e3f954 805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb41 57c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2 d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9 c5d2f8d39ff106035d74757c140b527d
user10	1f7dbe35470e3f7364736afb9a02787c 832ea8b5cdd958d53d27aa69c834e63b
user11	062f82f9ef701cfa2573aa40f2e7cc98 79789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57 d27d29648e770deb91dadb4b6ddd5f3a
user13	92df155300535fa28fefe4a65f2059fb 3bedfe6409d01cbf29dd45b1163234b4
user14	964205c227974d9048fe67a485f73f71 2d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c 69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6 a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e9 4ae5d8f1948f6de483801fa12ed4056b

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

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

question: suppose an adversary reads the hash of `user3`'s password. can the adversary figure out what the password was with that information alone?

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594 d4823e675567a545a0e13ada7f89c995
user2	adab20bca80250418a02196ca80d01d8 5456675b29816a40212eaa8d4c56ad22
user3	02c0d53273d41d74221c69c539315796 5182254904aa0653aa43f3a206e6eaa1
user4	21f110ed053c24004463526b3ba06999 c52580abeff8a85e89f7d9af966446e3
user5	2fae96a772f8e7c1bc5ae87ba291e467 1b66792a024b324e87305ca551c34a80
user6	e5034d3265313a5b4fbac596b1e3f954 805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb41 57c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2 d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9 c5d2f8d39ff106035d74757c140b527d
user10	1f7dbe35470e3f7364736afb9a02787c 832ea8b5cdd958d53d27aa69c834e63b
user11	062f82f9ef701cfa2573aa40f2e7cc98 79789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57 d27d29648e770deb91dadb4b6ddd5f3a
user13	92df155300535fa28fefe4a65f2059fb 3bedfe6409d01cbf29dd45b1163234b4
user14	964205c227974d9048fe67a485f73f71 2d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c 69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6 a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e9 4ae5d8f1948f6de483801fa12ed4056b

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, nor can they "reverse" the hash function to get the password

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*

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

username	hash(password)
user1	14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995
user2	adab20bca80250418a02196ca80d01d85456675b29816a40212eaa8d4c56ad22
user3	02c0d53273d41d74221c69c5393157965182254904aa0653aa43f3a206e6eaa1
user4	21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3
user5	2fae96a772f8e7c1bc5ae87ba291e4671b66792a024b324e87305ca551c34a80
user6	e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf
user7	5de31d1e08319d086c7fee8de384eb4157c6b02a6e575b7c49a6a9d8c7388836
user8	b8cd12c77b504b4e103dcc12cce948b2d737638d80c4e8c221cbbe13d1776bb7
user9	bbd108d7967db8489432dc3d1899d4d9c5d2f8d39ff106035d74757c140b527d
user10	1f7dbe35470e3f7364736afb9a02787c832ea8b5cdd958d53d27aa69c834e63b
user11	062f82f9ef701cfa2573aa40f2e7cc9879789512d55bafb2b72486ba262612d0
user12	224c37554776a15372b768a2c9232a57d27d29648e770deb91dadb4b6ddd5f3a
user13	92df155300535fa28fefe4a65f2059fb3bedfe6409d01cbf29dd45b1163234b4
user14	964205c227974d9048fe67a485f73f712d950dea64bf3c94cbf93fadec91d657
user15	907c14aae02712091b9682e5e08b997c69094d15b92e55b1463d247e10c9c235
user16	e30cff73a266cda4e41ad3e28d7770c6a029316de6da4c2976cba547b2efdb46
user17	6efa1445959a560e0148464ff4d316e94ae5d8f1948f6de483801fa12ed4056b

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

hash(password)	password
. . .	
14cdac7adb383312176cf4376d6bd594d4823e675567a545a0e13ada7f89c995	fhxiyfioncvgxvrb
adab20bca80250418a02196ca80d01d85456675b29816a40212eaa8d4c56ad22	pmhdtzyxjpwbsjwv
02c0d53273d41d74221c69c5393157965182254904aa0653aa43f3a206e6eaa1	gxrdqlbxrvhdopjg
21f110ed053c24004463526b3ba06999c52580abeff8a85e89f7d9af966446e3	hncobhdvimrtbpmy
2fae96a772f8e7c1bc5ae87ba291e4671b66792a024b324e87305ca551c34a80	jprfyxpznunmbabp
e5034d3265313a5b4fbac596b1e3f954805c0662c2f9cca6ca3efaa8282080cf	abbsocpnkpnqirmf
. . .	

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

(it took my laptop 1.22 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

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	h/PQkKPhG8IZ8r9pzyrdJaFEN84L7jG
user2	FiuUw/UnyUnUSTtowroAEOoOnrwpJiC
user3	xgHxAoz56N1J3k/fgG1nl18LhJTP/c.
user4	tOJcJLNP5GG7CVCzMsRNXW93rAKkz0C
user5	/kC9h5wwWFc9Ghm0GBZu3La/rikLSOe
user6	NrvsbVFRCUrpu/kI0vUgdZRt5UgaE.O
user7	uYw28g4ny/EJfaTdkHLTV3VR0phdbnW
user8	pf4vjqPlyDA7qGPOjsXVNwJVTOz1zVS
user9	oHJdvc1N0U5.bddasJICdClBTc9G1hm
user10	euXVvvxyAJyfBiIJqCD/03Nf0tKvBq2
user11	RWxFuID2F7L4/xMgIIr8NInvEyygLzC
user12	p4Qs7e936KHs3MVH4PojD29Y7whDSM2
user13	M9IwuX1Bdws6tIu4O/0pwXAkJBymZje
user14	bc3EL.yS1Olon5b.FQ/DljiW0uCTYOK
user15	9QfuMPuIHkXCyo/f5MdthG47gGGXLJe
user16	VBOA7y0TvobH5938ch.J7d7e6u0.MtG
user17	qkycmlPRA0BRObYpt/qIfxvgeEocqF6
user18	vJl/7e1gWI3GeD7o/gYyUGCIxXjQgLS
user19	nMvi3XUvnkZCNVQ8rtmnpK7DqbtlXwS
user20	z5nwjrJghXhwIgZbyuCJzAABuZQOa.u
user21	fKCQIUYcXc1aqUUbLLF3WR1nILtMSNe
user22	4f9e6oyZmFav2ux1BU8blaWNVojqEaa
user23	M/RExDQsV.Vhtysn6bH9SRLj.sc8ubu
user24	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user25	fQPGG3/hQJFxTtO6MNPs4.twWy5jbqu

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
```

it would take my laptop almost 19 days
to compute 10 million slow hashes,

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	h/PQkKPhG8IZ8r9pzyrdJaFEN84L7jG
user2	FiuUw/UnyUnUSTtowroAEOoOnrwpJiC
user3	xgHxAoz56N1J3k/fgG1nl18LhJTP/c.
user4	tOJcJLNP5GG7CVCzMsRNXW93rAKkz0C
user5	/kC9h5wwWFc9Ghm0GBZu3La/rikLSOe
user6	NrvsbVFRCUrpu/kI0vUgdZRt5UgaE.O
user7	uYw28g4ny/EJfaTdkHLTV3VR0phdbnW
user8	pf4vjqPlyDA7qGPOjsXVNwJVTOz1zVS
user9	oHJdvc1N0U5.bddasJICdClBTc9G1hm
user10	euXVvvxyAJyfBiIJqCD/03Nf0tKvBq2
user11	RWxFuID2F7L4/xMgIIr8NInvEyygLzC
user12	p4Qs7e936KHs3MVH4PojD29Y7whDSM2
user13	M9IwuX1Bdws6tIu4O/0pwXAkJBymZje
user14	bc3EL.yS1Olon5b.FQ/DljiW0uCTYOK
user15	9QfuMPuIHkXCyo/f5MdthG47gGGXLJe
user16	VBOA7y0TvobH5938ch.J7d7e6u0.MtG
user17	qkycmlPRA0BRObYpt/qIfxvgeEocqF6
user18	vJl/7e1gWI3GeD7o/gYyUGCIxXjQgLS
user19	nMvi3XUvnkZCNVQ8rtmnpK7DqbtlXwS
user20	z5nwjrJghXhwIgZbyuCJzAABuZQOa.u
user21	fKCQIUYcXc1aqUUbLLF3WR1nILtMSNe
user22	4f9e6oyZmFav2ux1BU8blaWNVojqEaa
user23	M/RExDQsV.Vhtysn6bH9SRLj.sc8ubu
user24	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user25	fQPGG3/hQJFxTtO6MNPs4.twWy5jbqu

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

slow_hash(password)	password
. . .	
	fhxiyfioncvgxvrb
	pmhdtzyxjpwbsjwv
	gxrdqlbxrvhdopjg
	hncobhdvimrtbpmy
	jprfyxpznunmbabp
	abbsocpnkpnqirmf
. . .	

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

it is for fewer passwords!

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

6.1800 in the news

Rank	Password	Time to crack it	Count
1	123456	< 1 second	3,018,050
2	123456789	< 1 second	1,625,135
3	12345678	< 1 second	884,740
4	password	< 1 second	692,151
5	qwerty123	< 1 second	642,638
6	qwerty1	< 1 second	583,630
7	111111	< 1 second	459,730
8	12345	< 1 second	395,573
9	secret	< 1 second	363,491
10	123123	< 1 second	351,576

https://nordpass.com/most-common-passwords-list/

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

username	slow_hash(password)
user1	h/PQkKPhG8IZ8r9pzyrdJaFEN84L7jG
user2	zx63UU3DeiSKRJOKjifNS5rd79cIbYy
user3	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user4	tOJcJLNP5GG7CVCzMsRNXW93rAKkz0C
user5	zx63UU3DeiSKRJOKjifNS5rd79cIbYy
user6	NrvsbVFRCUrpu/kI0vUgdZRt5UgaE.O
user7	uYw28g4ny/EJfaTdkHLTV3VR0phdbnW
user8	pf4vjqPlyDA7qGPOjsXVNwJVTOz1zVS
user9	oHJdvc1N0U5.bddasJICdClBTc9G1hm
user10	v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq
user11	RWxFuID2F7L4/xMgIIr8NInvEyygLzC
user12	p4Qs7e936KHs3MVH4PojD29Y7whDSM2
user13	v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq
user14	bc3EL.yS1Olon5b.FQ/DljiW0uCTYOK
user15	9QfuMPuIHkXCyo/f5MdthG47gGGXLJe
user16	VBOA7y0TvobH5938ch.J7d7e6u0.MtG
user17	qkycmlPRA0BRObYpt/qIfxvgeEocqF6
user18	WxZXb3zJRZFfHK6bNNyKagWXTmD5CaS
user19	nMvi3XUvnkZCNVQ8rtmnpK7DqbtlXwS
user20	z5nwjrJghXhwIgZbyuCJzAABuZQOa.u
user21	fKCQIUYcXc1aqUUbLLF3WR1nILtMSNe
user22	N0mMl0u8DYOQ9dAHTK829ij1zCqtft.
user23	M/RExDQsV.Vhtysn6bH9SRLj.sc8ubu
user24	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user25	fQPGG3/hQJFxTtO6MNPs4.twWy5jbqu

slow_hash(password)	password
. . .	
	fhxiyfioncvgxvrb
	pmhdtzyxjpwbsjwv
	gxrdqlbxrvhdopjg
	hncobhdvimrtbpmy
	jprfyxpznunmbabp
	abbsocpnkpnqirmf
. . .	

in reality, our table of slow hashes would look more like this

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	h/PQkKPhG8IZ8r9pzyrdJaFEN84L7jG
user2	zx63UU3DeiSKRJOKjifNS5rd79cIbYy
user3	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user4	tOJcJLNP5GG7CVCzMsRNXW93rAKkz0C
user5	zx63UU3DeiSKRJOKjifNS5rd79cIbYy
user6	NrvsbVFRCUrpu/kI0vUgdZRt5UgaE.O
user7	uYw28g4ny/EJfaTdkHLTV3VR0phdbnW
user8	pf4vjqPlyDA7qGPOjsXVNwJVTOz1zVS
user9	oHJdvc1N0U5.bddasJICdClBTc9G1hm
user10	v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq
user11	RWxFuID2F7L4/xMgIIr8NInvEyygLzC
user12	p4Qs7e936KHs3MVH4PojD29Y7whDSM2
user13	v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq
user14	bc3EL.yS1Olon5b.FQ/DljiW0uCTYOK
user15	9QfuMPuIHkXCyo/f5MdthG47gGGXLJe
user16	VBOA7y0TvobH5938ch.J7d7e6u0.MtG
user17	qkycmlPRA0BRObYpt/qIfxvgeEocqF6
user18	WxZXb3zJRZFfHK6bNNyKagWXTmD5CaS
user19	nMvi3XUvnkZCNVQ8rtmnpK7DqbtlXwS
user20	z5nwjrJghXhwIgZbyuCJzAABuZQOa.u
user21	fKCQIUYcXc1aqUUbLLF3WR1nILtMSNe
user22	N0mMl0u8DYOQ9dAHTK829ij1zCqtft.
user23	M/RExDQsV.Vhtysn6bH9SRLj.sc8ubu
user24	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user25	fQPGG3/hQJFxTtO6MNPs4.twWy5jbqu

highlighted users are using common passwords

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

slow_hash(password)	password
. . .	
zx63UU3DeiSKRJOKjifNS5rd79cIbYy	111111
v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq	password
lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e	123456
BHFoF2NA5NLCt6rNUfOM0zy0O.kM1uK	12345
WxZXb3zJRZFfHK6bNNyKagWXTmD5CaS	123123
N0mMl0u8DYOQ9dAHTK829ij1zCqtft.	qwerty1
. . .	

because it takes much longer to make this table using slow hashes, the attacker is incentivized to concentrate on the most common passwords

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

it is for fewer passwords!

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	h/PQkKPhG8IZ8r9pzyrdJaFEN84L7jG
user2	zx63UU3DeiSKRJOKjifNS5rd79cIbYy
user3	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user4	tOJcJLNP5GG7CVCzMsRNXW93rAKkz0C
user5	zx63UU3DeiSKRJOKjifNS5rd79cIbYy
user6	NrvsbVFRCUrpu/kI0vUgdZRt5UgaE.O
user7	uYw28g4ny/EJfaTdkHLTV3VR0phdbnW
user8	pf4vjqPlyDA7qGPOjsXVNwJVTOz1zVS
user9	oHJdvc1N0U5.bddasJICdClBTc9G1hm
user10	v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq
user11	RWxFuID2F7L4/xMgIIr8NInvEyygLzC
user12	p4Qs7e936KHs3MVH4PojD29Y7whDSM2
user13	v7xhwRKEolIN1YUTN7sCLwX1yvN1cMq
user14	bc3EL.yS1Olon5b.FQ/DljiW0uCTYOK
user15	9QfuMPuIHkXCyo/f5MdthG47gGGXLJe
user16	VBOA7y0TvobH5938ch.J7d7e6u0.MtG
user17	qkycmlPRA0BRObYpt/qIfxvgeEocqF6
user18	WxZXb3zJRZFfHK6bNNyKagWXTmD5CaS
user19	nMvi3XUvnkZCNVQ8rtmnpK7DqbtlXwS
user20	z5nwjrJghXhwIgZbyuCJzAABuZQOa.u
user21	fKCQIUYcXc1aqUUbLLF3WR1nILtMSNe
user22	N0mMl0u8DYOQ9dAHTK829ij1zCqtft.
user23	M/RExDQsV.Vhtysn6bH9SRLj.sc8ubu
user24	lsWXfuHCZOez5Ed7fM7K//8KrmaiG7e
user25	fQPGG3/hQJFxTtO6MNPs4.twWy5jbqu

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

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
user24	KPh8kxp3T2wyfYpapO007O	7FmTSwEMiUHNI78Dmgkq34RmUWloRPW
user25	DXTiXzfPrmu3LAFwcMYhm.	Khvv0K4u7DSuWlnmPN3V7p/IgFwU43C

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

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
user24	KPh8kxp3T2wyfYpapO007O	7FmTSwEMiUHNI78Dmgkq34RmUWloRPW
user25	DXTiXzfPrmu3LAFwcMYhm.	Khvv0K4u7DSuWlnmPN3V7p/IgFwU43C

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

question: how do we authenticate users in this system? I.e., how does the below code need to change?

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

this is our previous `check_password` function

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
user24	KPh8kxp3T2wyfYpapO007O	7FmTSwEMiUHNI78Dmgkq34RmUWloRPW
user25	DXTiXzfPrmu3LAFwcMYhm.	Khvv0K4u7DSuWlnmPN3V7p/IgFwU43C

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
```

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
user24	KPh8kxp3T2wyfYpapO007O	7FmTSwEMiUHNI78Dmgkq34RmUWloRPW
user25	DXTiXzfPrmu3LAFwcMYhm.	Khvv0K4u7DSuWlnmPN3V7p/IgFwU43C

policy: provide **authentication** for users

threat model: adversary has access to the entire stored table

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

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

to pre-compute the table, even for just the most common passwords, 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. the more common `user1`'s password, the more likely it would be revealed, but it's unlikely an adversary would pre-compute this table since they would not know the salt ahead of time.

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

updated NIST guidelines for authentication

these are not the only updates, just the ones we're focused on today

1. **Length:** Passwords *must* be at least eight characters long. NIST recommends at least fifteen, and suggests that up to 64 characters be allowed.
2. **Complexity:** Password complexity — e.g., requiring certain special characters in every user's password — should *not* be imposed
3. **Password updates:** Users should *not* be required to change their passwords periodically, but should be forced to change if there's evidence that their password has been compromised.
4. **Hints:** Systems should not use password "hints" (e.g., "What was the name of your first pet?")
5. **Throttling:** Systems should set limits on how many attempts a user has to enter their password (NIST recommends no more than 100 attempts)

given what you've learned in today's lecture, what is the purpose of each of these guidelines?

https://pages.nist.gov/800-63-4/sp800-63b.html#passwordver

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
user24	KPh8kxp3T2wyfYpapO007O	7FmTSwEMiUHNI78Dmgkq34RmUWloRPW
user25	DXTiXzfPrmu3LAFwcMYhm.	Khvv0K4u7DSuWlnmPN3V7p/IgFwU43C

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