CS203 Gene's Class - User Authentication Summary

LEC2-user-auth.pdf
PDF

read this lecture slide on user authentication, read very thoroughly

Thanks! I've thoroughly reviewed the **Lecture 2: User Authentication** slides from CS203 / NetSys 240. Here's a structured summary with important insights and intuitive explanations, slide-by-slide grouped by topic:

🔐 Fundamental Concepts

- **Main Problem**: Proving identity to a system — "How do you prove you are who you claim to be?"
- **Methods of Authentication**:
  - **What you know**: Passwords, secret answers
  - **What you have**: Hardware token, smartphone
  - **What you are**: Biometrics (e.g., fingerprint)
  - **Where you are**: IP, phone number, etc. (less secure)

🧠 Password-Based Authentication

- **Common Issues**:
  - Usability vs. security (hard passwords are hard to remember)
  - Vulnerabilities: eavesdropping, keyloggers, social engineering
  - Attack vector: After gaining access, attackers steal more passwords (via sniffers/keyloggers)
- **UNIX-style system**:
  - Stores **hashed** passwords using one-way hash functions
  - Historically used DES with limitations (e.g., 8-char truncation)
- **Dictionary attacks**:
  - Effective due to human-chosen, low-entropy passwords
  - /etc/passwd was world-readable → enabled offline cracking

---

🧂 Salting & Shadowing

- **Salt**: Random value added to password before hashing

- Prevents identical hashes for same passwords
- Defeats precomputed rainbow tables
- **Shadow passwords**:
  - Hashes stored in **/etc/shadow**, accessible only to root
  - Increases protection against offline attacks

🔍 Real-World Password Breaches

- Common causes: reused passwords, weak choices, plain text storage
- Examples:
  - **Hotmail (2009)**: Phishing → weak passwords like “123456”
  - **RockYou (2009)**: SQL injection → 32M passwords leaked, all stored in the clear

---

🧠 Human Factors

- People use:
  - Same passwords across sites
  - Easy passwords (“ZotZot123”, “password1!”)
  - Insecure recovery questions (e.g., pet name, car)
- **Password vaults**:
  - Pros: strong, unique passwords per site
  - Cons: single point of failure

🖼️ Graphical Passwords & Passfaces

- Exploit visual memory for improved usability
- Example: **Passfaces**
  - User recognizes pre-selected faces from 3×3 grids
  - More intuitive than recall-based passwords
  - Downside: face bias (race, attractiveness), shoulder surfing

---

🧬 Biometric Authentication

- Based on **what you are**: fingerprint, iris, voice, etc.
- Pros: nothing to remember, can’t be shared
- Cons:
  - Forgery (e.g., Play-Doh fingers)
  - Hard/infeasible to revoke

    - Expensive, infrastructure-heavy
- **Error types**:
    - **False positive (fraud)** vs **False negative (insult)**
    - Tradeoff must be tuned carefully

🧠 Cognometrics (New Class)

- Recognition-based authentication using human cognition (e.g., Passfaces)
- Human brain is better at recognizing faces than recalling text

---

🔁 Challenge-Response Authentication

- Secure against replay attacks
- System sends random challenge → user replies with f(key, challenge)
- Mitigates network sniffing vs. plain passwords
- Still vulnerable to **MitM attacks** without mutual authentication

---

🔐 Advanced Mechanisms

- **Lamport's Hash Chain** (S/Key OTP):
    - One-time passwords using a precomputed hash chain
    - Each authentication uses the next link
    - Only the final hash (root) is stored server-side
- **RSA SecurID**:
    - Hardware token using a shared secret + time-based counter
    - Generates new 6-digit codes every 60 seconds

⚠️ Risks & Attacks Summary

- **Offline attacks**: Dictionary/rainbow table
- **Online attacks**: Guessing, phishing
- **Hardware/software keyloggers**
- **Social engineering**: Tricking helpdesks, users
- **Replay attacks**, **MitM**
- **Biometric subversion**: Forgery, unrevokable IDs

Would you like me to extract this into slides, prepare quiz-style questions, or compare with SSH/Kerberos next?

this is the transcribed class lecture

like SSL that we're going to go over later and there's some wireless, I think it ends
with some firewalls which also will go in depth and wireless.
Any questions from us?
So again, I would like to have your new proposals by next Tuesday, like last time.
If you have any fundamental questions there about what I expect, ask.
I'll have officers on Monday.
Again, I want to deal with two people.
No double dipping means not to use material that you did already or you're doing in parallel
in some other course.
So there has to be something distinct that you do for this part.
And if you plan to do writing, survey or something like that, no chatbots.
That should be you.
If you do use some tools that you are not sure, or you want to use some tools, you're not quite sure
if this transcends the boundary, just ask.
It's easier to ask now than to explain later.
Okay, so user authentication, one of the basic topics in computer security.
How do we authenticate humans?
That means us.
All right, basic problem.
Somebody claims to be Alice or Bob, and now how do you prove that you are that Alice or Bob?
Now why do we do authentication?
We do authentication in order to enforce access control.
All right, just authentication for the sake of authentication isn't useful.
What's the point of authenticating you if nothing happens?
And any form of, any time you are asked for anything that identifies you and authenticates you, it's
for some purpose.
Whether it's a policeman that stops you and says, where's your driver's license?
The driver's license is to let you go.
If you authenticate you, you let you go.
Then you get a ticket.
And without the driver's license, you might get arrested.
So no access to driving privileges, right?
If you work for a company, you use a badge and maybe sometimes a camera to authenticate you as
an employee, right?
So you can have access to the premises.
You get into the UCI's Google space, right?
Google Docs and Drive space.
You authenticate it via a dual and password username so you can get access to those resources, et
cetera, et cetera.
So there's always a reason for why you are being authenticated, right?
There is something that you are given access to if you succeed in authentication.
Okay.
So this pops up just about everywhere.
So, of course, as you know, there are many ways to prove that you are.
Say you are.
The most familiar ones are passwords.

Their demise has been predicted for the last 40 years, and yet they are still here.
Moreover, they show no signs of going away anytime soon.
An interesting parallel, email.
Email has been around since early 80s.
Pretty much the same way it is used to be.
Maybe the WYSIWYG changed, you know, the here and there, but the interfaces, et cetera.
But email is pretty much the same.
Same format email addresses, and every year somebody says, yeah, I guarantee it.
You know, it's like a doomsday.
You say, oh, yeah, the end of the world is coming tomorrow.
Just you wait.
And tomorrow comes, ah, you know what?
My clock was wrong.
I was like, yeah, it missed you.
Anyway, ah, the stars did not align.
So the same thing about email and password, doomsday predictions.
Email is still here.
Password is still here.
No signing going on.
Right.
So passwords we know, and nobody loves them, but we're used to them.
And then we have passphrases, right?
Or answers to personal questions.
Like, what color eyes do you like the most?
Or, what was the make of your first automobile?
Really difficult questions, right?
And then there are, of course, secret keys, right?
So we can use some secret keys.
If you can remember them, you can use them, right?
Or you can write them down, which will be a lot of fun.
But these are things you know, presumably.
Pins are, of course, another form.
It's a variation of the theme of passwords, right?
Because they're supposed to be numbers, not alphanumeric characters.
Then you can do authentication based on where you are.
So that is, if you are in some space, and you can prove you are in some space, geographical location, then you must have access to something.
You might be giving access to something.
Like, if you are in some secure room in the Pentagon, by virtue of being in that secure room, you're allowed to access some computer system or some database.
I'm not saying that's how it should be, but it can be like that.
It could also be an IP address given to you by your internet provider.
So, for example, if you live in the campus housing, your entry is provided by UCI, right?
As opposed to, say, me, I also live on campus, Canada campus, I think on Union Hills here, next to campus, my address is provided by Cox.
And all my neighbors are in the cost of Google to buy them.
So the IP address tells you something.
If you believe me, the IP address, of course.
It could be home phone, right?
Of course, people don't use home phones anymore, right?
Landlines are rare.

Cell phone.
Cell phone used to be able to, many, many, many years ago, if you had a certain cell phone number, then you had to be in a particular space, because back then there was no roaming, right?
So if your phone number worked, that means you were, you know, with a 949 area code, you were in North County.
It doesn't work anymore.
Now, MAC addresses, could they be used for identifying where you are, right?
Or for anything at all?
Everybody knows what MAC addresses is, right?
Theoretically, MAC addresses supposed to tell you something about the interface of the device.
Any ideas?
No?
Too early?
MAC addresses have a special format.
They actually should tell you who the manufacturer of the device, or the interface.
It doesn't tell you necessarily where you are, and that's why I put a question mark there.
Also, MAC addresses are impermanent.
There was a time when most devices that have MAC addresses, you couldn't change them.
Today, you can change MAC addresses on most laptops, industrial or desktop systems, most laptops, cell phones, and other guys.
Also, even if MAC addresses were impermanent, MAC addresses are only used for the one hub, right?
That's a data link layer.
So, in this room, the MAC address is only visible to the access point.
But beyond the access point, it has no meaning, right?
There's nothing.
It doesn't longer go there.
Okay, then there's what you are as a human, right?
Now, what you are are your intrinsic features, like your DNA, right?
Your fingerprints, your eyes, your face geometry, your hand geometry, your vein pattern on your wrist.
There's gait, where you are, not a very good biometric, but that's in you.
All kinds of other things.
And then there is the one that you are familiar with the most, which is what you have, right?
It's a possession of something.
We know this, right?
This is the most frequent two-factor.
You have the app on your phone, and it's visually secure.
You trust the app.
The app will pop up, right?
Either with the approve, disapprove, or a code, right?
The temporary code, that essentially, by entering that code or hitting approve, you're saying, I'm in possession of this phone.
And because this phone has been pre-registered as associated with you, there's a reasonable assumption that because you have this phone, and previously you entered the right password and username, must be the case that you are a police agent.
And if you think of it, any other?
Might I miss something?
Any other basis upon which you can be authenticated?
What you know, where you are, what you are, what you have.

Some people suggested where you've been, but that's hard to remember.
So, how do our passwords work?
No.
Quiet track.
No.
Yeah.
Without a sign.
So, must not be able to remember.
Right.
So, you have a secret password.
You pick the secret password, and the system checks to authenticate that password.
Now, sometimes, of course, you don't pick the secret password,
sometimes the secret password is given to you.
Like, how many of you have used a browser to select a secret password?
Then you don't actually know anything.
Right?
All you do is, what?
You log into your computer, or your, uh, smartphone computer, like that.
And, the idea is that the browser, especially something like Chrome, let's say.
I want to pick, I use Firefox sometimes here, but let's say Chrome, which is popular.
Um, you also have to log into your profile, right?
On Chrome.
So, you log into the computer, you log into the browser, but the browser doesn't know what you log into the computer.
So, in fact, you log into your profile on Chrome, and that essentially unlocks the so-called password vault, or password bundle, that is stored in your profile on Chrome.
Not super reassuring, right?
But, convenient, because as long as you use that Chrome, right?
Actually, it doesn't have to be Chrome there, right?
You could be using Chrome somewhere else.
You know, if you set it up correctly, that password that Chrome picked for you will be available once you authenticate with the right folder.
Right.
So, initialization, but typically the password initialization is you pick the password based on some password rules.
Which vary, of course, from site to site, from time to time.
And, um, how do you actually communicate with the log in time for the first time, right?
Or for subsequent times, right?
There is always this risk of eavesdropping.
Right?
So, if you imagine the most naive way of communicating a password is you type your password and then
it just goes somewhere.
Now, if it's a local computer, maybe you can have a little more trust.
You trust the operating system that the log in prompt is actually from the operating system.
Right?
So, when you turn on your laptop, you have tablets and laptops, and when you open it, so turn it on, and the log in prompt occurs, and says, usually it doesn't say username, right?
You can't, right?
But usually it just ends your password.
Do you know who's asking you for a password?

Who's still a computer?
Do you know who's asking?
Should be the operating system.
Should be the operating system.
Is it the operating system?
Do you have any way to verify that, in fact, the window that shows you the login is the operating system window?
I say no.
I don't think you know.
Maybe if you have an encrypted disk, would you type in your password and connect it for
But even then, I mean, if it just scrolls the right things, it says bias, blah blah blah blah, authenticating, initializing, enterprise, you don't know.
So, I'm not trying to cause panic here, but you should be aware that, so you have a little more sure when your computer is doing a cold boot, right?
Because it's functionally like turned off, although that's a whole other story.
How do you know your device is turned off?
We will touch upon this in a couple of days.
You often tell it to turn off, you press the button, and how do you know it's off?
Well, you say, well, you know what, I'll just unplug it from the wall.
Are you sure that's on?
Are you sure there's no battery?
Do you know what's inside the device?
They have CMOS battery, too.
Ah, yeah, but you should like disassembling the device, voiding the warranty, and causing electric shock, potentially.
So, actually, you don't.
So, that's a depressing thought, right?
You actually don't know who is asking for the password.
But we trust.
Okay, but that's the password that goes to your operating system, right?
It doesn't go on the wire, right?
It's not sent anywhere.
What about if you're logging into Facebook, or, you know, your Instagram, or whatever, your favorite website, or any kind of service?
Like, you're going into UCI.
A window pops up that looks like UCI's little pop-up, with blue and stuff, you know, all the right colors.
How busy is this to copy?
So, you enter your password, and then it goes.
So, you don't know.
But let's suspend our disbelief, and say, okay, we trust that it goes, it goes to the right place, okay, okay, okay.
But what about how is it sent?
Is it sent in clear text?
That's bad, right?
It can't be sent in clear text, because then it gets used to the other part.
Okay, well, you hope that actually in reality, it sends over a secure connection.
And sort of prior to you entering the password, whatever the stub application, whether it's a browser, a client browser, or some other application on your laptop, or your phone,
has already established a secure, usually TLS, or VPN connection, to the other end.
But even then, you might not trust that connection.

So, probably not a good idea to send the password unencrypted, even over the connection.
So, what's typically done is the password is, like, hashed.
Remember hash functions, right?
Yeah.
Now the question is, where is it hashed?
So, who gets to see your password locally?
Maybe it's hashed in software, with a client software.
Maybe it's hashed in hardware.
Maybe there's a way for the login prompt, or if it presents you with a password prompt,
to communicate directly to some hardware inside your laptop of a smartphone,
that actually takes that password, hashes it, and that's it.
And then there's a template resistant, because if it's not, maybe, you know, somebody can hack it.
And then, how does the other end check the password?
Well, and of course, is your password guessable?
Right?
None of this other stuff is important if your password is weak.
Right?
If I can, with reasonable probability, guess your password,
all the other measures to prevent theft and misuse of password, they're useless.
Because guessing a password means somebody is going to try to log in, just like you would.
So, whatever measures are taken after you enter a password and hit return,
they're useless if the password is being guessed.
Also, on most operating systems, it's difficult to keep a password file secret.
Especially like multi-user systems.
So, I know that you are all used to, essentially, single-user computers, right?
I mean, how many people usually have, like, multiple profiles on their Android?
I mean, you can do it, but it's a pain.
How many of you share your laptop with others?
My guess is not many, if any.
Maybe when you were younger, and you had, like, a sibling, and you shared your laptop.
Or the parent will let you use their laptop and create an account.
But most people, like, you're 80, they don't share laptops.
So, this does not seem relevant to you, but think about a bigger world out there.
Like, an organization.
Where you have multiple users, multiple, many users,
and they all need to log in and access the same system.
So, you need to store the password database somewhere, the password file, right?
And so, in modern systems, at least one of the Unix derived,
historically, the password file is accessible.
It's called world-readable.
And the reason it's world-readable is because it's not just the password file.
It's, essentially, the user database.
And so, programs that run on Linux, or Unix, and whatever,
all these Unix-derived operating systems,
they need to consult that file, not for the password's sake,
but for account's sake, to figure out what accounts have rights for what resources.
So, it's mostly for access control.
Okay?
This was done a long time ago, when Unix first was born, in the 80s.
Now, with hard-to-remember passwords,
if you don't pick a weak password, you pick a hard-to-remember password, you're good.

And if your memory is good, wonderful.
But what about reusing passwords?
A lot of people can remember one difficult password.
Right?
I'm sure you can remember one difficult password.
That is not, by any standard, weak.
But then, can you remember 10, 15?
These days, most of us have well over 10,
some of us, I would say, around 20 different accounts.
There have been some studies about this, but I don't think it's an immediate number.
You could carry around an object, like some kind of a device, like a smartphone,
maybe, that has your passwords all stored there in plain text.
And you unlock your phone, you access your password, and you're like, oh, great.
What if you don't have your phone?
What if somebody steals your phone?
What if a malicious app on the phone exfiltrated that file?
So there is a cottage industry out there called, you know, Password Vault.
Maybe, have anybody used Password Vault?
I mean, some of them are, there are some free, but most of them are cost-mine.
And some companies do use those.
Okay, but guess what?
Well, when you have a company that runs the services, Password Vault Vault,
a third-party Password Vault Vault, they become like honey or a swarm of bees.
They are a super attractive attack target, and just like this DA that we talked about last time,
it's like, attack me.
Because this is where the friendly jewels are.
You get into a password-drawn company, you get into their databases,
whether through outside or inside attack, and you wind up with a big dirt.
Big prize, right?
Tons of passwords.
So I'm not clear that that's a good solution.
Then there's like denial of service issues, right?
If you store your passwords on your phone, if the phone is lost,
or like what happened to me a couple of months ago,
the phone just died, and there's no way.
I mean, it's dead beyond all salvation.
There's no way you can take anything out.
So now it's stuck with having to change all the passwords.
It's like a denial of service.
And for a while, your account's essentially off-limits to you.
One of the more interesting and super easy attacks, by the way,
with a denial of service goal.
Let's say you want to attack a particular company, right?
Let's say you have a company called XYZ Incorporated
that provides access to, I don't know, cute squirrel videos.
And they specialize in cute squirrels only.
No ugly squirrels.
And all the squirrel lovers in the world create accounts there
and just love, you know, it's like popular, right?
So all this community, probably not large,
because how many squirrel lovers out there,

they actively use this service,
and then somebody who wants to just piss them off
or maybe has a competing company decides to mount it in our service.
All they need to do is not break any passwords.
All they need to do is get a large enough sample of the usernames
or people who have an account.
All you do, you can program this.
You can run into this by hand.
Essentially, try to log in.
So for every account, let's say Alice has an account.
Write the username Alice, random password, click.
Username Bob, random password, click.
Do it three times, do it four times,
whatever the little threshold.
Use it, it's a little threshold.
For number of failed logins, right?
What happens?
You're locked out.
So when real users come back to log in,
due to the large number of successive failed login attempts,
you need to contact this 1-800 number in Mauritania.
Good luck.
With awful hours.
With, yes, of course.
We're only open between 3.30 and 4.30 p.m. Hawaii time.
Good luck.
Good luck, right?
Because then, of course, they want to make sure it's you.
Even if it's done in automated fashion
with some other second factor,
all back email, blah, blah, blah, blah, blah, blah.
It's a giant waste of time.
And a logistical nightmare.
So, a great attack.
You can program a bot to do this, right?
You can go for, like, even random usernames.
Eventually, you're going to find usernames that are existing.
Login, login, login.
Bam.
Locked out.
So, no passwords need to be broken.
So, let's see.
Then there's social engineering.
Social engineering is, like, this whole thing in its own right.
You can teach a class on that, just on social engineering.
But it's not really computer science.
Social engineering is all about how to trick people.
So, maybe more appropriate than psychology or some other department.
But it's a wonderful opportunity for attackers, right?
And a lot of spectacular hacks in this world that you've heard about
that sound really, really technical.

I suspect it's TouchNet and Mirai and all these other things.
I think they all, I'm pretty sure,
they all somehow involve elements of social engineering.
Social engineering means human.
Right?
Human interaction is elicitation of sensitive information
in verbal form.
Non-analog form.
Okay?
The typical social engineering attack
probably doesn't work at night today,
but, you know,
a CIO or a, let's say,
no, it's an IT person,
in the middle of the night,
in some department of some random company,
receives a panicked phone call
from the vice president
of, I don't know,
toilet affairs,
and says,
I am stuck in
Timbuktu
on a
very important company trip.
I'm dealing with clients.
All of a sudden,
my phone broke.
I cannot log in.
Help.
Millions of dollars
of business are at stake.
The panicked IT clerk,
who's sleepy,
and probably has,
I don't know,
DeVry Institute course
or education or something,
says,
oh, of course,
yes, sir.
And says,
okay,
here's a one-time login code,
go in this,
do this.
Yeah,
that's it.
That's the end of it.
That's the back door.
That allows the attacker

to get in.
It's still super common.
Yeah,
I'm sure,
you know,
every once in a while,
in a research literature,
or even at, like,
Black Hat,
DEFCON,
you will hear talks
about people saying,
okay,
we did this kind of
probing attack
against 500 companies
than, like,
Brazil,
I heard one talk.
And the particular talk
I heard is that,
you know,
when this was about
the attacks
on the Brazilian banks,
right,
and essentially utilizing
what I just described,
calling at random hours
and pretending to be somebody.
They didn't even invent somebody.
They actually pretended
to be somebody
who worked for the company.
So, you know,
the hackers know in advance.
They've done their research.
They've looked
for the company charts
and said,
who is awake?
Oh, this guy is on vacation.
So,
it is really believable
when most of it is true.
So, they call it,
say,
I'm so-and-so.
I'm on vacation
in Florida.
Oh,

so-and-so.
Yes,
he's on vacation.
Element of truth
goes a long way.
Right?
That's kind of
human psychology.
It works.
In the U.S.,
there were studies that,
oh, you know,
just like infomercials.
I don't know
if you even know
what infomercials are,
but this is a curious thing.
When Americans listen
to a voice
of a salesman,
it helps
if their voice
has a refined British accent.
Maybe you should wonder why.
If somebody speaks like they're
from the deep south,
forget it.
If somebody speaks like they're
from Brooklyn,
forget it.
But if somebody speaks
with a refined British accent,
carefully enunciating
every single syllable,
I'll do every single word
they say.
Most of us,
this country,
oh, yes,
I'm you.
What can I do?
Right?
So that's part
of social engineering.
Right?
So,
another reason is,
another way that it works
is the attacker
will know some information
about you.

Right?
So if they want to impersonate you
or want to get into your account,
they will know information.
Social security numbers,
days of birth,
those are a lot of difficult
things to find out.
Right?
Social security numbers
used to be believed
to be secret.
Nobody believes
to be secret in these days.
Even though we still
keep them private,
but I can buy
your social security number
if you have one.
It's not going to cost me
very much.
And so if I buy it,
even if it costs me
a couple hundred dollars,
it allows me essentially,
well,
not just social security,
but along with
other information
like David Burr,
driving the last time
in December,
maybe open a bank account
online.
Great.
If I open a bank account,
I can rent something
in your name.
Like an apartment.
Establish a gas company
relationship.
Electrical company.
Right?
Essentially,
start stealing your identity.
Yeah,
so,
this is like entertainment,
but anyway,
passwords are a pain.
So we're going to focus

for a moment
on the passwords.
There have been
tons of breaches
in the middle,
but by the way,
this is super old.
All these examples
are super old,
and yet,
I don't refresh them.
They're like 20 years old.
But,
this happens every year.
The similar things
that happened
20,
25 years ago
still happen.
Right?
If you go to the
website,
passwordpresearch.com,
they have lots
of interesting
articles.
Right?
So,
why do people
want your password?
So,
of course,
one of the reasons
they may want your password
is just to log in
and just delete your files.
They just hate you
and they want to
suck.
That doesn't happen
very often.
It can.
I mean,
you make some enemies,
you probably can.
But,
this is very personal.
Generally,
what somebody wants
is more than just
your password.

So,
installing a key log
or a sniffer
is another way.
And then,
basically,
you sniff for more passwords.
Especially if your account,
getting into your account,
is privileged.
Right?
So,
maybe it will be able
to run a sniffer
that will sniff
on other people's passwords.
The other thing
is to steal the password file.
And once you steal
the password file,
well,
then you can run
password cracking too much.
Because the password file,
as you can already imagine,
doesn't actually store
a real password,
but stores hashes.
Right?
So,
all you need to do
is try a brute force adapt.
Alright?
So,
Unix-style passwords
work like this,
right?
You enter your password,
it hashes,
and then this is
the etc.
etc. password file
that has one line
for a user,
generally,
and it contains,
among other things,
so the line,
the one line is like,
it's a text file.
So,

one line for a record,
oh,
one line is a record,
and this record
is for a given user,
and it has some
non-human accounts,
okay?
net, like root,
and admin,
or whatever,
some other,
some other cron account,
but it also has one
for every user.
And so,
when you enter a username,
the username is also
stored there in the clear.
So,
people realized early on
that it's not a good idea
to store a password there,
so why don't we store a hash?
So,
great,
the system doesn't know
your password,
it only knows the hash,
and because,
presumably,
we use a good cryptographic hash,
like the kind we talked about
briefly last time,
hash is supposed to be
one way.
Yeah?
One way means that,
yeah,
the inverting,
like knowing a hash,
and learning what the input
to the hash was,
is hard.
But is it really?
If the input space
is very large,
it is hard.
But if the input space
is constricted,
it might not be so hard.

But wait,
there's more.
Hashing is better
than encryption,
maybe you will realize why.
You know why?
I mean,
encrypting a block,
it should be,
it's usually roughly equivalent
to hashing a block.
All you think encryption
is not used.
And hashing is used.
No decryption.
No decryption.
Yeah,
that's a reason,
it's not that.
Major reason.
I hope you've never
had to see the thing
that's passed through
and that's been honest.
Okay,
now you're getting close.
What else does encryption
mean?
You want to encrypt the question.
No,
it leaks the length,
doesn't it?
Hmm.
So you want to have
the uniform
value,
right?
There are other reasons,
then you can think of some.
Encryption is not,
now encryption,
that's,
let me rephrase
what I said,
be very careful.
Encryption as in
something that can be reversed
is not used.
You can use encryption
to build a hash function.
An encryption function

can be used
to build a hash function.
Okay,
so it's okay
to use encryption function
to build a hash function
that you can use
for this purpose.
But what we don't do
is encrypt
the password itself.
Okay?
So remember the
properties of hash functions,
right?
So if you are a password hacker
and you want to
break the password,
what property
would you like
the hash function
not to have?
Well,
one way is we agree.
We all must have,
right?
Because it should be
difficult to go back
and figure out
from the hash
what the password
is going to be.
But what about
these other things?
Remember,
there's a strong
collision resistance
and weak collision resistance.
Which do we care
about?
Yeah.
Strong
would be nice to have
but not super important
because
the password
is specific,
right?
It's given.
It's given
to the hacker

but it's fixed
by the hash,
right?
So in order
to break
the system
without learning
the password,
the hacker
has to find
another password
that hashes
into the same value.
Does that make sense?
So if your password
is Apple
and it hashes
into
345
and I pick
Pear
and it also
hashes into
the 345
I don't need
to know
Apple.
I can just
log into your account
by typing Pear.
OK?
So
that weak collision
resistance
and that's
what we need.
So
I'm not sure
how many
to do it
but historically
that's what
has been done
is that
Linux
was using
DES
encryption
as a hash
function.
Now how

does it do it?
Well,
it takes
a null
string
a 64-bit
now you remember
DES is a 64-bit block
so it takes
a string of
0
64-bits
long
and then
encrypts it
repeatedly
OK?
Repeatedly
25 times
using
a DES
key
that is
derived
from the
password.
OK?
Make sense?
So it converts
your password
into a key
now that's
also
an interesting
detail
because
DES keys
are actually
56-bits
long
I mean
they're 64
but 8-bits
are parity-bits
so
that's an
interesting one
so we'll come
back to that
in a second.
The idea

so why do
they do this
or why did
they do this
because in
order to
mount a
brute force
attack
on a password
you would
have to
guess the
password
and perform
25
encryptions
or
if you like
25
decryptions
and at
the end
OK?
If you
hit 0
if what
you decrypt
after 25
decryptions
and 0
and a
string of
0s
you guessed
correctly.
OK?
Or
alternatively
you would
take
a string
of 0s
and encrypt
it 25
times
and see
if that
matches
the result
in the

password
file.
Am I
with you?
Any
questions?
So
it's a
clever idea
right?
The idea
was to
slow down
the
attacker
25
that is
for every
guess
of a
password
or every
possible
password
the attacker
guesses
it would
have to
perform
25
encryption
or decryption
operations.
that's
a lot
and back
then
DESC was
implemented
mostly in
hardware
it was
very slow
in software
so it
seemed like
a good idea.
And then
of course
passwords aren't
really random

right?
Doesn't matter
how old you
are, doesn't
matter what
culture
or which
country
which
ethnicity
you come
from
within the
group that
you come
from
you're
likely to
pick a
password
that is
not
truly
random.
There's
no such
thing as
humans
picking
truly
random
passwords.
Even if
you say
well
I know
a certain
date
where
let's
say
I
don't
know
I
graduated
from
college
or
I
had

OpenAI

my
first
date
or
something
auspicious
happened
on a
particular
date
and
nobody
else
knows
it
just
you
you
think
that
might
be
random
but
actually
it's
not.
I
think
about
are
dates
really
random?
There's
a lot
of
people
use
them
professionally.
Think
about
how
we
structure
dates.
Forget
hours
a
minute

just
dates.
I
mean
depending
on
which
notation
it
used
to
them
in
the
United
States
we
use
month
day
year
right?
How
many
months
are
there?
Well
last time I
said
how many
days are
there?
Let's be
generous and
say 51
in every
month
right?
They're
honest
right?
Let's say
they're
out because
they're
the maximum
and how many
years is it?
Well
there are a lot

of years
but you're not
likely to pick
a year from
dark ages
history
history
and
yeah
if you're
a fan
of history
you might
pick
something
ancient
Egyptian
era
or
something
from
the
time
of
William
the
Conqueror
but
most of
us
would pick
something
reasonably
more
1940
21st
century
so
that
does not
be
vulnerable
that's
not
random
and the
space
of that
is
very
small

so
don't
pick
dates
in any
format
right
so
but if
you just
look at
the
52
upper
and
lower
case
letters
that is
being
generous
assuming
that
whatever
system
we're
working
distinguishes
between
upper
and
lower
case
because
some
systems
do
not
right
they'll
just
convert
everything
to
upper
case
or
lower
case
so
52

in English
20
that's
26
26 times
2
plus
10
digits
and
32
punctuation
symbols
although
many
password
rules
will not
allow you
to use
punctuation
they might
allow you
to use
like
exclamation
point
the question
mark
the dot
and maybe
that's it
but okay
let's be
generous
so that
gives us
52
plus 10
plus 32
that's
94
and let's
say we
limit to
8 characters
because
that's
pretty much
how many
of you

have
passwords
that I
exceed
8 characters
yeah
I buy
a lot
how many
exceed
12 characters
look at
you
15
okay
you like
entering
passwords
don't you
because
there's
a big
trade-off
here
right
the big
trade-off
is
okay
so
let's
say
I'm
not going
to
triangulate
your
password
here
but let's
say you
have
15
character
password
and
he has
an 8
character
password
now we

have
usability
issue
let's
say
you're
young
you're
smart
you can
remember
15
characters
obviously
you can
right
but what
about
making
mistakes
in
those
15
characters
as you
type
have
we
never
did
mistakes
right
how likely
let's
put it
another way
how likely
are you
to make
a mistake
in an
8
character
password
as opposed
to a
15
character
password
much more
likely

there are
some
people
fatigued
right
I mean
finger
fatigued
you have
to stare
at the
keyword
so
there's
that
second
issue
is the
time
it just
takes
longer
to enter
that
password
so
while I
find it
commendable
that you
guys use
this long
password
that's
great
that doesn't
mean you're
out of the
woods
with attacks
you know
but
there is a
trade-off
right
and most
people
I mean
they're okay
with like
not super

short
passwords
but like
research
show that
8 is
about
the right
mix
but with
8
characters
which is
I think
for most
people
and I
suspect
most people
your age
have picked
some better
passwords
than let's
say your
uncles
and grandparents
and parents
because you
know
who are less
confused
having
you're going
to wind up
with roughly
6 quadrillion
possible passwords
well to a
kid this
sounds really
impressive
right
you just
explain how
many zeros
are
but we're
all adults
here
6 quadrillion

is not a
large number
okay
not at all
right
and
since
given
freedom
to pick
right
without like
actual
restriction
most of us
will pick
things like
human and
pet names
and dictionary
words
we're going to
wind up
with
20
2 to the
20
essentially
a 20-bit
password
which means
that
even if
your password
is 8
characters
the real
entropy
there
is
about
20
minutes
meaning
that
human
death
names
and all
the dictionary
words

will roughly
be about
2 to the
point
that's
assuming
the language
is known
so you
assume
your password
is English
like
if you
are
Swahili
speaker
then a whole
different
thing
will apply
it will
be based
on the
vocabulary
of that
language
but
you
have
to
assume
that
hacker
knows
about
that
yeah
so
and
if you
think
pins
are
better
they're
not
because
you
get
with

maximum
2 to the
27
with
8
digit
pins
and
that's
something
that
any
software
on your
phone
you can
do
in
seconds
just
iterate
for all
of those
so
that's
not a
scary
number
on
the
websites
they often
restrict
length
for
I get
they have
to store
in a
database
somehow
but why
in the
world
would they
ever
that
takes
no sense
to me
either

symbols
they
so
I think
the reason
okay
you're asking
two questions
why do they
restrict
the length
and why
they don't
allow
symbols
none of
those two
none of
those two
make sense
to me
but I have
a potential
explanation
for why
they don't
allow certain
symbols
is because
they think
these are
executable
some kind
of like
they're
trigger symbols
that somehow
some of
these things
like
slash slash
or back
slash back
dollar sign
can be used
if you somehow
cause a buffer
overflow
they can
so there's
some like

suspicion
of these symbols
dating back years
nobody really
knows why
but that's
my suspicion
why
do they
truncate
and allow
only
fixed
up to a certain
number of
characters
I don't know
that
that may be
it makes no
sense because
it's hashed
and it may be
again that
whoever programmed
the front end
no
they're also
afraid of
these kind
of buffer
overflows
and so on
and memory
bugs
because
you allow
unlimited number
of characters
they're thinking
you're going to
enter 2,000
characters
something is going
to break
in the code
I know
I know
I'm with you
on that
I think that's

silly
right because
there's no
justification
it's only the
front end
because of the
back end
they're hashed
and they're all
the same way
it's a heat
map
some of the
stuff I'll show
you is really
dated
dated as in
like 10 years
old
more than 10
years old
none of it
changed since
people still
think
similar
faster
because the
fundamental
human
mentality
hasn't
changed
what is
a heat
map
what does
it mean
is that
you see
where
things are
really
hot
there
yellow
right
and so
over here
it's a heat

map generated
from a
leaked
password
database
a giant
leaked
password
database
okay
so
this is
a pin
sorry
this is not
a pin
heat map
and so
over here
where it's
really hot
dates
dates are
different
from other
pins
this is
you know
day
day
month
month
type
dates
right
and then
related couplets
meaning repeated
things like
0101
0505
667
they're all
along this
diagonal thing
okay
and these are
years of
specific years
of 19 something
because probably

OpenAI

birth years
people
for people
to take
passwords
back
so
as I said
before
the
Unix
and all
the other
Unix
like systems
have this
etsy
password file
and it's
world readable
for reasons
I already
explained
so
if you don't
do anything
and just
store the
hashed
passwords
in that
file
the attacker
will
easily
before
even knowing
what that
file is
before breaking
in or getting
an accident
the attacker
can precompute
a large table
of all the
hashed
passwords
yeah
I mean
passwords are

passwords right
you can just
pick all the
dictionary words
all the dictionary
words are
reversed
all the
two word
combination
let's restrict
our discussion
to English
for now
okay
because
similar things
apply to
other languages
but let's say
yeah
all the date
in a standard
US format
all the
even times
of day
using
you know
after a second
let's say
granularity
you can precompute
hashes of all
these values
combinations
of all these
values
pet names
city names
state names
university
mascot names
right
all of these
can be precomputed
in a large
database
stored in a large
database
that is something

a task that
an adversary
could do
and would take
maybe a day
or two
maybe less
this can be
stored
and it can be
done at your
own leisure
now
at the time
of intrusion
when the
adversary gains
access to
the password
file
all the
adversary
needs to
do
is extract
the hashed
passports
and do it
database
basically
am I with you
very simple
so a little bit
of work ahead
of it
and then
no work
essentially
once the
password file
becomes available
not only that
but the
adversary can
break into
multiple systems
and reuse
that same
work that
we did
earlier

right
it's not like
he has to
brute force
anything again
just a bunch
of database
who comes
so that's
nice
for the
adversary
bad for
us
right
this is
called an
offline
attack
because the
adversary has
done the
bulk of
the work
offline
and online
just does
lookouts
so that's
why salt
unlike what
the doctors
say is good
for you
we don't
store hash
passwords in
new
password file
we store
salted
passwords
salted
means
at the
time of
the initial
creation of
a user
account
the system

or the
administrator
will pick
a random
number
called a
salt
that random
number is
only used
there in
the password
file
it isn't
stored in
the plain
text
but it's
not reused
so different
systems will
have different
salts for the
same user
so you
have a record
something that
looks something
like this
this is maybe
a little outdated
that green box
you see
gts would be my
account name
the salt will be
that number
right
the first blue
number
the second is
the hash
then you have
like user
identity
uid
i think that
the 14510 is
uid
30 i don't
remember

group id or
something
then the full
name
like what my
actual name is
and then the
login shell
right
and today is
probably like a
lot more
at least that
but we're not
interested in
the rest of the
stuff
the important
thing is
the salt
and the
password
that has
together
okay
so what
happens
very simple
little mandate
right
but makes a
big difference
that offline
attack
is no longer
possible
why not
well let me
give you an
example
suppose the
salt was
one decimal
digit
would the
offline
attack
be
possible
one
just one

decimal
digit
meaning
it's a
number
from zero
to ten
would it
be possible
to mount
this offline
attack
of course
it's possible
why not
right
just the work
increases tenfold
right
but okay
now it depends
on how long
exactly you make
it
but suppose
you make it
128 bits
oh
now
one digit
128 bits
instead of
like what
one digit
is like
three digits
right
so from three
bits to 128
bits
now the
adversary
cannot do
cannot do
an offline
attack
because remember
for every
guess of the
password
that the

adversary
makes
he has to
multiply
do it
two to the
128 times
per password
so it
will be
two to the
128 times
whatever
even two to
the 20
right
is
beyond reach
of the
adversary
today and
for
what it
would be
so originally
they made the
salt
32 bits
I'm not sure
exactly what
it is
because it
varies
I think the
length of the
salt varies
from unique
flavor
to unique
flavor
but if I
were using
it today
I would
want the
salt to
be at
least
80 but
probably
more like

128
than salt
is
healthier
okay
but salt
makes a big
difference
now once the
attacker breaks
in
okay
now there's a
crucial difference
between offline
and online
attack
once the
attacker
breaks in
and reads
the
lc password
file
okay
the attacker
sees the
salt
so then the
attack becomes
much cheaper
but it can
only be
mounted
once he
breaks in
not ahead
because
by knowing
the salt
you shave
off the
2 to the
128
okay
and that
basically
the explanation
is the same
thing
just take a

second
to read it
well you shave
off the
2 to the
number 28
but it's
no longer
only off
line
fully attack
no no
it's online
it's online
it's not
offline
at all
the attacker
breaks in
learns the
salt
and now
has to do
with that
salt
whatever
2 to the
20
2 to the
40
however many
password
guesses
there are
per user
yeah
so there's
still a bit
of work
but it's
not a scary
amount of work
I'm sorry
I mean
we can
let the salt
expire
well
you can
let the salt
expire

but you
have to
replace it
in the
password
file
so
what you're
saying
is that
you know
if you
know
that
they
have
the
so
it
takes
at least
24
hours
to
mount
an
online
attack
once
he
learns
the
salt
then
you
could
change
the
salt
every
24
hour
you could
but the
problem is
okay
so this is a good question
let's think about this
that what you say
would be useful
or would be

possible
if
just
a password
attack
a brute force
password
attack
would take
24
hours
but remember
once the adversary
breaks in
and learns
the salt
he then
also knows
the hash of the salt
and the password
right
so the adversary
knows this
right
that
the
well
that
the adversary
knows this
right
once he breaks in
he knows this
and he also knows
the salt
so you can change
the salt
but he learned
the previous salt
and the hash
of the previous salt
plus password
he's gonna break
the password
see
he's gonna break
the password
so changing the salt
at that point
not effective
not effective

because
if he's breaking
it locally
maybe
but
you know
others
he's gonna
exfiltrate
and break
it on his own
path
so
the password
hasn't changed
that's the main
problem
but here
it says
12
I think
I may be wrong
it wasn't
30
I think
it was even
less
it was 12
bits
originally
well today
12 bits
is laughable
right
so what
is it
saying
if the
passwords
come from
I don't
know
2 to the
20
space
and multiplied
by 2 to the
12
you still
get 2 to
32

which is
a laughably
small number
or password
bracket
right
it takes
a minute
or less
to go for
the 2 to the
32
password
choice
right
so
instead
so
what happens
these days
is that
most systems
actually
kind of
separate
the two
things
they separate
the password
from
the account
information
in Unix
and that's
kind of
a good
practice
anyway
so
they create
what's called
shadow
password
and so
instead of
looking like
what you saw
before
the account
record
in the

etsy
password
file
has
the name
and then
all the other
information
but there's
like an
x
following the
accounting
that says
actual password
isn't here
it's a
special
hashed
password file
which is not
world readable
like it took
only 30 years
for people to realize
that the passwords
and user account
information
shouldn't be stored
together
well that's what
happens
right
because all
those other
programs
application programs
running on
this Unix
like operating
system
they still
do need
the account
information
to figure out
who can access
what
etc
and distinguish
between users

but
they don't
need to know
the passwords
right
so
let's create
a file
that has
all the account
information
that also
store passwords
in
passwords
should only
be visible
to the root
or the admin
account
right
so instead
it's stored
in an etsy
shadow file
which is only
readable
by the
system
administrator
okay
also
very trivial
thing
and of course
it also
added expiration
date
for passwords
right
so that's
why on
most
production
Unix
like systems
you'll be
asked to
renew
or
change

another
password
every so
often
okay
so
you've probably
heard of
keyloggers
right
so these
are things
you generally
have to be
afraid of
not so much
on your own
devices
because
it's not
all that
easy
to install
a keylogger
unless
somebody
has
direct
access
and an
account
on your
device
but
more like
when you're
using
public
facility
like
any of
you ever
use
ICS
terminals
or ICS
machines
but that's
where you
have to
be worried

about
because
there you
have no
idea
what software
is running
who has
been there
before
you
essentially
these
machines
are
sort of
like
public
toilet
you have
no idea
what's
going on
there
and
what
is
running
there
really
so
if
these
drugloggers
are
problem
they
can be
hardware
based
or
software
based
hardware
based
means
somebody
actually
inserted
something
you

know
external
keyboards
not the
kind
you have
right now
in front
of you
but
actual
keyboards
move on
they have
a wire
some of them
do
some of them
don't
but most
have a wire
well
the wire
goes in the
back of the
computer
somewhere
or
some
USB
interface
well
somebody
could have
installed
something
between
the USB
interface
and the
back
of the
computer
you cannot
see
especially
publicly
and
essentially
it
sees

every
keystroke
you have
okay
some
government
computers
in various
agencies
do this
on purpose
they tell
employees
your
keystrokes
are locked
and there's
a keylogger
for
auditability
and logging
purposes
but
do you
yourself
whenever
you use
a public
or at least
not personal
but do you
have a look
to see
that the
wire
goes directly
into the
USB port
in the
back
of the
computer
or
there's
some
other
little
creepy
crawling
that
is in

between
I bet
most people
don't
with
wireless
oh
with wireless
it's even worse
because the
wireless
keyboard
typically
use
encryption
but the
kind of
encryption
you use
is often
homegrown
by the
manufacturer
or really
crappy
and so
you don't
need to
put a
hardware
device
you can
just sniff
on it
externally
right
because
they use
some
kind of
either
infrared
or some
other
kind
of
wireless
so
beware
but
attacks

keylogger
attacks
especially
hardware ones
are not
common
they are
targeted
right
somebody
has to
target
a specific
computer
a specific
person
in order
to do
that
software
keyloggers
are different
that often
happens
with buggy
browsers
or buggy
it used
to happen
once in
a while
with microsoft
windows
where there
would be
some malware
that would
warm itself
and usually
again
with some
degree of
social engineering
preceding it
and then
will
install
itself
and actually
take over
the keyboard

driver
or worse
yet
update
the keyboard
driver
right
because
the keyboard
is a
peripheral
right
there's a
driver
for that
peripheral
to
insert an
illegitimate
fake update
of the
driver
that would
contain
a keylogger
not nice
and then
there are
like
lower tech
ways of
sniffing
on
keyboard
input
shoulder
surfing
everybody
familiar
with
shoulder
surfing
yeah
people are
getting
too close
to you
too close
for comfort
you're entering
your pen

whether it's
on the
door
block
or
on your
bank
ATM
somebody's
like
breathing
down
your neck
that's
shoulder
surfing
I mean
easy to
avoid
but you have
to be
coming
center
right
but
shoulder
surfing
means that
human
eyes are
looking
at something
but it's not
always human
eyes
it could be
a tiny little
drone
it could be
a camera
from farther
away
with a
telephoto
lens
that is
shoulder
surfing
right
so
it could be

think about
it
you go
let's
say
let's
take
example
of
the
ATM
anybody
ever
go
to
the
ATM
anymore
I think
I go
once
twice
a year
but
still
people
do
I can
see you
every time
I go
to the
ATM
there
are
people
so
you don't
trip
in
do you know
how many
cameras are
looking at
you
well
there's
a camera
in the
ATM
right

they usually
have
a camera
but what
about
other
cameras
on the
street
do you
know
if
they have
the
right
angle
maybe
they
do
maybe
there's
something
right
above
you
that's
looking
directly
at
the
ATM
keypad
no
there are
thermal
cameras
that's
something
also
fairly
scary
thermal
cameras
what does
that mean
it means
when you
type
your
password
or

pin
on
some
surface
like
keyboard
or
pin
tab
what
happens
is
your
fingers
touch
the
surface
you
are
human
last
time
you
checked
you
were
human
last
time
you
checked
you
were
alive
which
means
as a
live
human
you have
a certain
temperature
on your
body
and your
fingers
reflect that
temperature
and when
your
live

human
fingers
touch
the
dead
plastic
cold
plastic
this
what
happens
heat
transfer
even
if
you
go
boom
boom
well
that's
the force
right
that generates
heat
cut off
mainly
the
skin
so
if
you
type
in
your
pin
or
your
password
and
then
somebody
snaps
a
photo
of
the
pad
or
the
keyboard

they
will
see
heat
map
they
will
see
the
keys
that
won't
press
most
recently
they
might
not
directly
leak
pin
or
password
information
but
it
leaks
a lot
information
that's
about
in
particular
which
keys
were
pressed
recently
maybe
not
exactly
the
sequence
but
how many
combinations
that
are
right
so
something to worry about

then there is something called acoustic emanation
there's a whole body of research literature on this
starting in like 25 years ago
people realize
people realize
that when you type on any surface
be it a keyboard
or a
thing
even on the phone
on the
on screen keyboard
there are sounds that are being named
and sound propagates
right
when you type
especially in some of those
people like me
etc
aggressive
times
sound propagates
farther
than the human ears
right
so if you put a very sensitive microphone
you know
ten yards away
people still pick up
keyboard emanations
isn't that scary
I think it's very cool
I have a lot of research on this
still these days
there are people who work at it
but
the sound
that your
fingers make
when you type in
anything
not just the classroom
is unique
not because
so much
that you make it
but the keyboard
itself
produces a different sound
for any different key
every key

on your plastic keyboard
makes a different sound
the same
to a lesser extent
old story about the keyboard
you have
but every time you press a key
the sound that it makes
when you press a Z
is different from the sound
that it makes you
when you press a D
or slash
or a question mark
or space
okay
there are different sounds
and so
if I record those sounds
I essentially can key log
without installing a key log
isn't that fun
but it gets worse
much worse
even if the manufacturers
and some made an effort
some didn't
to make all the keyboard
sounds sound the same
it's possible
it makes people
more expensive
there's still a problem
does anybody see the problem?
yourself as a person
what about yourself?
I mean
some people
have just
and they find
and they just
make themselves
because of the finger
and
uh-huh
yeah
yeah
so remember
it's true
that
what I said before

is that there's you
who types differently
depending on which key you type
and then there's the actual keyboard
that makes different sounds
so even if you press
the same way
on the same
like
you have this finger
like you're one of those
hunter tech
people like me
like
you move the keyboard
hit
move the keyboard
hit it
so you see
my angle doesn't change
so I hit the same way
right
but the keyboard sound
is different
but
manufacturers suppose
now fix the keyboard
that all the sounds
are the same
but I haven't changed
right
so
when I use my
this finger
that goes straight down
on this one letter
it sounds one way
when I have to angle my finger
to press shift
right
or press question mark
it sounds differently
not because of the keyboard
but because of the way
I angle my fingers
and the force I apply
correct
but it gets worse than that
that
so
you can indeed

still mine the information
like I said
about the individual
humans typing
characteristics
but
it's harder
it's harder
it's easier to extract
from the manufacturer
introduced
distinction
though
the problem is
what did you say
time
like time between keys
time between keys
that's the problem
turns out
on a standard
QWERTY keyboard
let's assume
that we're using
QWERTYs
right
which is an American
keyboard
as you move
from key to key
right
from let's say
typing key
to typing H
to typing E
to typing N
to typing E
to typing X
to typing E
right
you type one letter
at a time
even if you are
touch typist
you type one letter
at a time
well
the intervals
the timing intervals
between successive letters
are unique to you

also
and the keyboard
they sound like
that's right
they're slightly different
yeah
yeah
but
that's right
they sound
that's true
but that's actually
related more
to what he said
but just
the sheer timings
between the keystrokes
so you don't have
to have any idea
but
guess what
there are
let's say
forget the
exclamation points
forget those
special characters
let's suppose
there are
I don't know
26 keys
on the keyboard
and you're just
typing words
right
so how many
possible
two word
combinations
are there
two letter
combinations
are there
26 times 26
right
so after a while
if you
record
the sounds
of somebody
typing

a lot of text
right
over the day
or two days
or whatever
you're going to
start distinguishing
between transitions
between keys
right
believe me
it's totally doable
because
assuming
somebody
is typing
English text
right
like writing
email messages
or filling out
forms
well
the probability
of occurrence
of two successive
letters
are quite distinct
and essentially
becomes
a simple
decision
so anyway
that was a little
detour
but what about
good vibrations
well good vibrations
refers to another
thing that
actually
if you're using
your phone
and that has
been some
published research
on this
and your phone
is like this
on your
desk

or whatever
in front of you
right
and
as you press
every
on-screen
button
right
on-screen keyboard
your phone
makes like a tiny
vibration sound
like a moment
that the other
phone do
right
just to confirm
that you actually
type
well
those vibrations
when you put a
microphone
underneath your
desk
they can actually
calculate your
typing
other analysis
include taking
photos of
smartphone screens
and looking for
quadrants
little areas
on your screen
that are
most
like smudged
that
leaks information
about your
password
or the
unlock pattern
you know
this
like
anybody use
unlock patterns

like five years
ago
everybody in this
class
was like
yes
I guess
they are
faster now
but apparently
you know
if your human
fingers leave
like oil
residue
on the screen
and so
if you take
a high resolution
photo of
somebody's
smartphone screen
you've got to
figure out
what the
unlock pattern
is
yeah
so all this
is good stuff
but the problem
is of course
like that
the passwords
right
why do I
talk about
all this
previous stuff
because that
would be
not only just
a way of
snoop on
what you're
doing
but of course
to snoop on
your passwords
right
that's the way

to actually
and sort of
learn your
password
the worst
thing we can
do is write
them down
because then
writing them
down
whether it's
on a device
or a piece
of paper
that becomes
a way to
like cause
denial of
service
etc
we can
pick passwords
that are
easy to
remember
some places
don't even
have passwords
I've seen
a few
systems
websites
which
completely
have no
passwords
but instead
have these
special
questions
like
they give
you five
special
questions
every time
you want to
log in
you have to
answer four

out of five
friends
but they all
tend to be
of this
variety
what's your
favorite pet's
name
rover
bio
jerry
and they
give you
some of
them are
worse
than others
some of
them allow
you free
text entry
which is
better
and some
of them
give you
a choice
like
out of
these five
pet names
which one
is your
favorite
really
seriously
terrible
easily
guessable
alright
I think
we should
be
done
for today
yeah
I'll
forget
this
candle

to me
ok

Thanks — this transcription adds a rich layer of insight to the lecture slides. Here's a high-level synthesis combining both the **slides** and the **transcribed lecture**, with key themes, intuitive takeaways, and nuanced clarifications that reflect your professor's commentary.

🧠 Core Theme: Why Authenticate?

- **Authentication = Access Control**: It's not just to prove identity — it's to *grant or restrict access* (physical, logical, or informational).
- **Analogy**: Like showing a driver's license to get driving privileges or a UCI login to access your Google Drive.

🛠️ 4+1 Categories of Authentication

1. **What You Know**: Passwords, PINs, secret answers.
2. **What You Have**: Phones, hardware tokens, authenticator apps.
3. **What You Are**: Biometrics (face, fingerprint, iris, gait).
4. **Where You Are**: Geolocation, IP address, or secure premises.
5. *(Possibly mentioned)*: **When/How You Behave**: Timing attacks, keystroke dynamics (used in behavioral biometrics).

🔐 Password Systems: Then and Now

- **Persistence**: Despite predictions of their demise, passwords are still dominant.
- **Old Unix Systems**:
  - `passwd` file used to store all hashes and user info — *world-readable*.
  - Used DES as a hash: 25 rounds of DES encryption on a block of 0s using key derived from password.
  - Salts were originally only **12 bits** — now laughably small.
- **Modern Defenses**:
  - **Salted hashes**: Stops rainbow table and precomputed attacks.
  - **Shadow files**: Separates password hash into `/etc/shadow`, only readable by root.

🧂 The Role of Salt

- Salt = Random nonce stored per user account
- Prevents **offline precomputation** attacks by requiring new hash computations for each salt.

- Lecture Note: If the attacker gets the salt *after intrusion*, they can still brute-force **online**.
- Ideal salt length today: ≥ **128 bits**.

🧠 Human Behavior Weakens Password Security

- Most people:
  - Use short or reused passwords.
  - Pick dictionary words, dates, pet names.
- Lecture Commentary:
  - Even 8-character passwords have low *entropy* (~20 bits).
  - Entropy ≠ Length — the actual randomness matters more.
- Heatmap Analysis:
  - Visualizations of password frequency (e.g., `123456`, birthdates, repeated digits).

🧱 Common Weaknesses

- **Password Vaults**:
  - Single point of failure — attractive to attackers.
- **Security Qs**:
  - "What's your favorite pet's name?" = terrible design.
- **Password Managers**:
  - Lecture cautions against browser-integrated managers (e.g., Chrome's vault).
- **Reused Passwords** = Cross-site compromise.

👀 Side-Channel Attacks & Threats

1. **Keyloggers**:
   - Software- or hardware-based (USB intercepts).
   - Especially dangerous in public terminals.
2. **Shoulder Surfing**:
   - Direct observation or via cameras/drones.
3. **Thermal Attacks**:
   - Residual heat from recently pressed keys.
4. **Acoustic Emanations**:
   - Sound profile of each key, timing between presses.
   - Even on phones: vibration patterns, mic under desk.

5. **Smudge Attacks**:
    - Finger oils reveal unlock patterns on phone screens.

🎭 Social Engineering

- Often behind “sophisticated” attacks.
- Common methods:
    - Impersonate IT staff or VPs in distress.
    - Use psychological manipulation (e.g., urgency, familiarity).
- Lecture Example: Refined British accent = perceived trust.

🧪 Miscellaneous Takeaways

- **Hash vs. Encryption**:
    - Hashing is one-way; encryption is reversible.
    - Use of DES as hashing = legacy workaround.
- **Timing Attacks**:
    - Not about keys themselves, but delays between them.
    - Helps reconstruct typed text.

Would you like this converted into slides, notes for flashcards, or potential exam-style questions next?

From the next time when I ask you some question, you use your knowledge from both the slide and the professor's lecture and then answer me. Understood?

Understood. From now on, I’ll combine insights from **both the slide deck** and your **professor’s lecture transcript** when answering any of your questions on user authentication. Just ask whenever you're ready.

explain dictionary attack

A **dictionary attack** is a type of password-cracking technique where an attacker tries to guess a password by systematically testing likely possibilities — usually words from a **precompiled list**, or "dictionary", of common passwords.

🔓 How It Works:

- Instead of trying *every possible* combination (as in brute-force), the attacker uses a curated list of passwords that people are **most likely to choose** — real words, pet names, common phrases, dates, etc.
- Each guess is **hashed** (using the same algorithm and salt as the target system, if known) and compared to the stored hash.
- If there's a match, the password is cracked.

🧠 Why It's Effective:

- **Human behavior is predictable**: People use weak, guessable passwords like **`password123`**, **`iloveyou`**, **`qwerty`**, birthdays, or pet names.
- **Low entropy**: Even an 8-character password has low effective entropy (~20 bits) if it's based on dictionary words.

🧪 Offline Dictionary Attack (More Dangerous):

- If the attacker gets a copy of the **hashed password file** (like **`/etc/shadow`**), they can:
  - Hash every word from the dictionary with the known salt.
  - Compare offline, without any system rate-limiting or detection.

🔐 How to Defend:

- **Use strong, random passwords** that are not in any dictionary.
- **Add salt**: Unique salt per user defeats precomputed attacks (e.g., rainbow tables).
- **Rate limiting** or lockout after multiple failed attempts (helps only for online attacks).
- **Password complexity rules** (though not perfect).
- **Two-factor authentication (2FA)**: Even if password is guessed, access is denied without second factor.

Would you like me to walk through an actual step-by-step example of how a dictionary attack breaks a password?

explain slide 21 One bank's idea for making PINs "memorable"

Slide 21 shows a real-world **bad idea** from a bank that tried to make PINs more *"memorable"* by assigning each digit a **word or phrase**. Here's what the slide (and your professor) are highlighting:

🔢 Slide Summary: "One bank's idea for making PINs memorable"

- The bank mapped each digit (0–9) to a **simple, vivid word**:

- 0 = Heaven
- 1 = Me
- 2 = Shoe
- 3 = Tree
- 4 = Door
- 5 = Hive
- 6 = Sticks
- 7 = Heaven (again!)
- 8 = Gate
- 9 = Wine

- This was meant to help customers **memorize numeric PINs** via **mnemonics or stories** (like "Me bought a Shoe at the Door" for 1-2-4).

🤦 Why It's a Terrible Idea (as implied by your professor):

1. **Small, fixed vocabulary**:
   - Only 10 possible options per digit = easy to reverse-engineer.
2. **Uniform mapping**:
   - Anyone who knows the scheme can map the story back to the digits.
3. **Predictable user behavior**:
   - Users will create familiar or meaningful combinations like "Me-Heaven-Gate-Wine" = 1-0-8-9.
4. **No entropy gain**:
   - It only adds a **layer of indirection**, not real randomness or security.
   - Attackers can automate matching known phrases back to PINs.

🧠 Lecture Insight:

Your professor mentioned this in a critical tone, showing how **"clever tricks" to improve usability** often backfire by introducing **predictable patterns** — which attackers love.

✅ Better Approach:

- **Use real randomness** for PINs.
- Use **2FA** or **biometric fallback** for usability.
- If memory aids are needed, encourage secure password managers rather than lossy word mappings.

Want me to show how an attacker might use this mnemonic scheme to guess a PIN more easily?

explain assword Vaults or
Password Managers

Sure. Here's an explanation of **Password Vaults** (also called **Password Managers**) using insights from both your **lecture slide** and **professor's class discussion**.

🔐 What is a Password Vault / Manager?

A **Password Vault** is a secure application that:

- Stores all your passwords (and sometimes 2FA tokens, notes, etc.).
- Encrypts them in a **vault** using a master password.
- Automatically fills in login forms for websites and apps.

Examples: LastPass, 1Password, Bitwarden, Keeper, Chrome/Firefox built-in password managers.

✅ Benefits (from the slides and common usage):

1. **Convenience**: You only need to remember *one* master password.
2. **Strong, unique passwords**: The manager can generate high-entropy passwords (e.g., `@o3&Uj93#P!z`) for each site.
3. **Avoids reuse**: Helps you avoid using the same password across services.
4. **Sync across devices**: Cloud-based managers can work across phone and desktop.

⚠️ Risks and Criticisms (professor emphasized this):

1. **Single point of failure**:
   - If the vault provider is hacked (e.g., LastPass breach), attackers can steal *all* stored credentials — even if they're encrypted, metadata (URLs, usernames) might leak.
   - This is what the professor called **"honey for hackers"** — these companies are **prime targets**.
2. **Device compromise**:
   - If your device is compromised (e.g., malware, keylogger), attackers may access the unlocked vault.
3. **Malicious apps**:
   - A rogue app could steal stored vault data or exfiltrate passwords.
4. **Trust and dependency**:
   - You're trusting a third party with all your secrets. If the service shuts down or has a bug, you could lose access.

🧠 Lecture Insight:

Your professor emphasized:

> "They become like honey or a swarm of bees... Attack me! Because this is where the crown jewels are."

He also discussed **browser-based vaults** (like Chrome):

- They're convenient but less trustworthy.
- E.g., Chrome unlocks the vault when you log in — and that password might itself be saved elsewhere insecurely.

🔒 Safer Use Guidelines:

- Choose **open-source or well-audited** managers (like Bitwarden).
- **Use a strong master password** that you never reuse.
- Enable **2FA** for accessing the vault.
- Don't save the vault's password in your browser or phone notes.
- Prefer **local** vaults if you distrust the cloud (e.g., KeePass).

Would you like a comparison table of popular password managers with pros and cons?

explain passface

Passface is a graphical password authentication system that uses **faces instead of text** — an attempt to make passwords easier to remember *and* harder to guess.

🧠 Core Idea:

Humans are **much better at recognizing faces** than remembering random strings. Passface uses this psychological strength by showing you a grid of faces and asking you to pick the ones you recognize.

👀 How It Works:

1. **Registration (setup phase)**:
   - The user selects **4 or more faces** from a large database of photos.
   - These faces become the user's secret password.
2. **Login phase**:

- The system presents a grid (e.g., 3×3 or 4×4) of faces.
- In each grid, only **one face is the user's chosen one**, the rest are decoys.
- The user must identify their face in **multiple rounds** (e.g., 4 rounds for 4 faces).

✅ Advantages:

- **Easy to remember**: Recognizing familiar faces is much easier than recalling alphanumeric passwords.
- **Hard to brute-force**: Large image databases make brute-forcing harder than cracking short passwords.
- **Language-agnostic**: No need to remember words, so it works globally.

❌ Weaknesses (discussed in security literature):

1. **Shoulder surfing risk**:
    - An attacker watching you select faces (or taking screen recordings) could learn your passface.
2. **Bias in face selection**:
    - Users often pick faces **similar to their own race, age, or gender**.
    - This makes chosen faces more predictable, lowering entropy.
    - Your professor didn't go into this, but studies (e.g., Brostoff & Sasse) found significant **demographic bias**.
3. **Replay attacks** (if not protected):
    - Without proper randomness in grid layout or session-specific scrambling, an attacker could replay your selections.
4. **Login speed**:
    - Slower than typing a password — you need to go through multiple rounds of face selection.

🧪 Related to Lecture Themes:

This ties into the broader discussion on **what you know vs. what you are**:

- Passface is a **"what you know"** scheme, but it *feels* like it draws on your ability to recognize **visual features**, almost like a mix of cognition and possession.

Summary Table:

Feature	Passface
Type	Graphical password
Strength	Easy to remember, hard to brute-force

Feature	Passface
Weaknesses	Shoulder-surfing, bias, slow login
Vulnerable to	Observation attacks, selection bias
Mitigation Techniques	Shuffle grids, increase decoy pool

Would you like a visual mock-up or sketch of how a Passface grid looks?

What if user uploads his own passface choices?

Excellent question — and this dives into a **critical security risk** with Passface systems.

🤔 What if users upload their *own* passfaces?

If users are allowed to upload their own face images — or choose images from their personal collection (e.g., family, friends, celebrities) — this opens up serious vulnerabilities:

🔒 Security Risks:

1. Guessability & Social Engineering:

- Users often upload **faces of people they know or admire**:
  - Friends
  - Partners
  - Celebrities
- These are **guessable** by someone who knows the user (e.g., from social media).
- Example:
  - If you upload a face of "Taylor Swift" or your girlfriend, an attacker who knows you could guess those as candidates.

2. Publicly Available Images:

- If attackers can scrape the same images from public sources (e.g., Facebook, LinkedIn, Instagram), they can run **targeted dictionary attacks** using face-matching AI.
- Similar to textual dictionary attacks, but with *face databases*.

3. Bias Exploitation:

- As your professor alluded to with password patterns: people follow predictable behavior.
- If attackers know the user is likely to pick images that:

- Match their race, gender, or age group
- Come from certain categories (e.g., attractive celebrities)
- Then attacker can **narrow the face search space** dramatically.

4. Face Recognition Leakage:

- If the system stores *user-uploaded faces*, and that database is breached, it leaks not just passwords but **personally identifying facial data**.
- That's a **privacy nightmare**.

🔐 Mitigations (Best Practices):

1. **Only allow system-provided face sets**:
    - The face images are random, diverse, and carefully curated.
    - This avoids the predictability and leak risk from personal uploads.
2. **Face grid randomization**:
    - Don't show face images in the same position every time — shuffle to prevent shoulder-surfing or replay.
3. **Avoiding predictable choices**:
    - Could use constraints or nudges to push users toward random-looking selections rather than predictable faces.

🧠 Takeaway:

Letting users upload their own passface images:

> **Completely undermines the entropy and security assumptions** of the Passface system.

It's the visual equivalent of letting users choose "123456" or "password" — it feels personal and memorable, but it's dangerously guessable.

Would you like a short summary of an academic paper that analyzed Passface guessability in real-world user studies?

Experimental study with graduate students at
UMBC
– 4 types of passwords: Passfaces with mouse,
Passfaces with keyboard, dictionary text password,
non-dictionary text password (random words and
numbers)
• Result: non-dictionary text password most

vulnerable to shoulder surfing
– Why do you think this is the case?

Great question — this is a **counterintuitive result**, and it reveals a lot about **human behavior and attacker advantage during observation**.

🧪 Recap of the UMBC Experiment:

Graduate students at UMBC were tested on their vulnerability to **shoulder surfing** across 4 password types:

1. **Passfaces with mouse input**
2. **Passfaces with keyboard input**
3. **Dictionary-based text password**
4. **Non-dictionary (random) text password** ← *most vulnerable*

❓ Why was the non-dictionary text password the most vulnerable to shoulder surfing?

Here's the breakdown:

🔍 1. Visual attention drawn to keyboard

- Non-dictionary passwords contain **random letters, numbers, symbols**.
- Users **type slowly**, often looking down at the keyboard.
- This makes it **easier for attackers to observe keystrokes** — either directly or by timing or pattern.

👁️ 2. Unfamiliarity leads to slower typing

- These passwords are harder to memorize and require **deliberate attention**.
- Users often type them **character by character** (vs. fluid muscle memory for common words).
- The slower and more deliberate the input, the **easier it is to observe or reconstruct**.

🤔 3. Nothing to "hide behind"

- Dictionary passwords can be partially inferred but are harder to fully guess without context.
- Passfaces introduce **multiple-choice distractions** (many decoy faces on screen), and selection is harder to monitor precisely.
- Random passwords? There's no visual clutter or masking — it's just a stream of precise inputs.

🎯 4. Every character matters

- A single wrong character = login failure.
- So, attackers watching closely just need to record **one successful login**, and they'll get the full sequence.

📌 In contrast:

- **Passfaces with mouse**: harder to track unless attacker sees screen **and** cursor.
- **Passfaces with keyboard**: even tougher — attacker must correlate key presses with hidden face positions.
- **Dictionary passwords**: faster typing, less observation time, partial guessing is less useful.

✅ Summary:

Password Type	Vulnerability to Shoulder Surfing	Why?
Passfaces (mouse)	Low–Medium	Needs screen + mouse tracking
Passfaces (keyboard)	Low	Faces hidden behind keystrokes
Dictionary passwords	Medium	Fast input, attacker may guess partially
Non-dictionary passwords	**High**	Slow, deliberate, easy to observe, every char critical

---

Would you like a short mock experiment setup of how you might replicate this with your own participants?

discuss everything the prof told about biometric authentication

Here's a **comprehensive summary of everything your professor said about biometric authentication** in the lecture, organized and explained clearly:

🔍 1. What is Biometric Authentication?

Biometric authentication is based on **"what you are"** — using **intrinsic physical or behavioral traits** of a human to verify their identity.

Your professor grouped this under one of the **four major bases of authentication**:

- What you **know** (e.g., passwords)
- What you **have** (e.g., phone, token)
- Where you **are** (e.g., location/IP)
- What you **are** → **Biometrics**

🧬 2. Examples of Biometrics Mentioned

Your professor listed several commonly used biometric identifiers:

- **DNA** — extremely strong and unique, but impractical for real-time systems
- **Fingerprints** — widely used, including in smartphones and border control
- **Eyes** — often refers to iris or retina scanning
- **Face geometry** — used in facial recognition systems
- **Hand geometry** — measures size and shape of the hand
- **Vein pattern on wrist** — an emerging biometric (e.g., palm vein readers)
- **Gait** — how you walk (not reliable, but sometimes used in surveillance)

All of these are **based on physical traits**.

🔐 3. Why Use Biometrics?

Your professor implied that biometric methods can **strengthen authentication**, especially when combined with another factor (e.g., 2FA).

Advantages:

- **Harder to forget** than passwords
- **Difficult to guess or steal** (though not impossible)
- **Convenient** (e.g., face unlock or fingerprint scan is fast)

⚠️ 4. Problems and Challenges (Implied or Known)

Though not deeply elaborated, the professor hinted at important **biometric limitations**, many of which are discussed in security literature:

❌ Not revocable

- If your fingerprint or face is compromised (e.g., leaked data), you **can't change it** like a password.
- Biometric breaches are **permanent security incidents**.

🧱 Device trust assumptions

- If the biometric data is captured or stored **insecurely** (e.g., not in secure enclaves), attackers could spoof or intercept it.

🕵️ Privacy concerns

- Biometric data is **deeply personal**.
- Systems collecting it (like facial recognition in public spaces) raise serious privacy issues.

🤖 Spoofing attacks

- Attackers have successfully bypassed biometric systems using:
  - Fake fingerprints (e.g., gummy fingers)
  - Photos or 3D masks (for facial recognition)
  - High-res iris photos

🧠 Additional Observations:

While your professor didn't go deep into biometric-specific failures, his broader **skepticism of authentication methods** (like trusting login screens or believing "secure" storage) suggests:

> He would **not blindly trust** biometric systems — especially without careful hardware and protocol-level protections.

💡 Biometric Authentication in Context of the Lecture

Biometrics were presented as **one part of a diverse authentication strategy**, often used in **multi-factor authentication (MFA)** alongside:

- Passwords (something you know)
- Hardware tokens or phones (something you have)
- Location/IP address (where you are)

🧠 TL;DR Summary:

Aspect	Explanation
What it is	Authentication based on **"what you are"**
Examples	DNA, fingerprints, iris, face, hand geometry, vein pattern, gait
Benefits	Convenient, harder to guess, always with you
Risks	Not revocable, privacy concerns, spoofable, storage & device issues
Prof's position (implied)	Useful, but should be **combined with other factors** and **not blindly trusted**

Would you like me to also compare biometric authentication vs. password authentication across usability and attack surface in a table?

explain Pulse Response Biometric

Pulse Response Biometric is a lesser-known but fascinating form of biometric authentication that uses the unique **electrical properties of the human body** — specifically how your body responds to small electric pulses.

🧬 What Is Pulse Response Biometric?

It's based on this idea:

> Every person's body has **unique electrical impedance characteristics**, due to variations in **body composition, skin conductivity, tissue distribution, and hydration**.

When a small (safe) electrical signal is passed through the body — often via electrodes touching the skin — the system measures the **response waveform**. This waveform acts like a fingerprint for your internal body structure.

⚙️ How It Works

1. **A device sends a low-power electrical signal** into the body, typically through contact points (e.g., fingertips or wrist).
2. The signal travels through the body and is altered by:
    - Bone density
    - Tissue composition
    - Skin conductivity
    - Muscle mass
3. The **return signal (the pulse response)** is recorded.
4. A **template** is created from this response and used for future matching during authentication.

🔐 Why It's Interesting for Authentication

- **Hard to spoof**: Unlike a fingerprint or face, your body's internal electrical response is not visible and not easy to replicate.
- **Liveness detection**: It confirms the person is **alive** (a static image or dead finger won't work).
- **Low risk of reuse/stealing**: You can't easily leave behind your pulse response like a fingerprint or typed password.

🔬 Known Use Cases / Research Prototypes

- Mostly seen in **academic prototypes** and **wearables**.
- Some experimental smartwatch systems have explored using pulse response to ensure the watch is being worn by the correct user.
- Not yet widely adopted due to complexity, but holds promise for **continuous, passive authentication**.

⚠️ Limitations

Limitation	Explanation
Hardware dependency	Requires skin-contact electrodes and precise measurement hardware.
Environmental sensitivity	Sweat, humidity, or motion can affect the signal.
Not mainstream	Not widely deployed in commercial products yet.
Template aging	Your body's impedance may change over time (e.g., due to hydration, illness, weight).

🧠 TL;DR Summary

Pulse Response Biometric is an advanced biometric technique that uses how your body alters an electrical pulse to authenticate you. It's **inherently secure**, **difficult to spoof**, and enables **liveness detection**, but is not yet common in real-world systems due to hardware complexity and variability.

Would you like a 1-slide summary version of this explanation for your presentation or notes?

explain the user study from slide 54-62

Slides 54–62 of your lecture describe a **user study** involving **novice users** performing **online banking tasks** under different phishing and authentication scenarios. Below is a structured breakdown of everything covered across those slides:

🧪 STUDY GOAL (Slide 54)

Objective:
To evaluate **how effective different anti-phishing strategies** and **authentication mechanisms** are in helping users avoid phishing attacks.

👤 PARTICIPANTS (Slide 55)

- **Participants**: 57 users
- **Profile**: All had **no prior experience with the online bank** used in the study
- **Environment**: Conducted in a **controlled lab setting**

💼 TASK (Slide 55)

Participants were asked to perform **3 types of tasks** using the bank's website:

1. **Login**
2. **Check balance**
3. **Pay a bill**

But here's the twist...

🕵️ PHISHING ATTACKS INCLUDED (Slide 56)

Participants were **intentionally exposed to phishing websites** during the study. These phishing pages varied in:

- **URL realism** (e.g., `securebank.com` vs `secur3bank-login.net`)
- **Look-and-feel** resemblance to the real site

Some sites were **real**, others were **fake** — and users had to distinguish.

🔐 AUTHENTICATION VARIANTS TESTED (Slide 57)

The study tested **different authentication designs** to see how they affect phishing resilience:

1. **Standard login** (username/password)
2. **Image-based authentication** (like Passfaces or site keys)
3. **Mutual authentication** (bank proves itself to user, and vice versa)
4. **One-time password via email or phone**

📊 KEY RESULTS (Slides 58–60)

🧠 1. Users are bad at spotting phishing

- Many **couldn't tell fake from real** websites.
- **URL inspection** was rarely used.
- Even when sites looked slightly off, **users still entered their credentials**.

🔒 2. Mutual authentication helps

- Sites that displayed **user-specific cues** (e.g., pre-chosen images or phrases) made phishing harder.
- Users were **more likely to detect phishing** if their expected image was missing.

💬 3. Feedback matters

- Giving **real-time feedback** (e.g., “This is a secure connection” or “URL mismatch”) improved detection rates.

- Some participants still ignored feedback — **habituation** is real.

🤯 4. Too much security = confusion

- Complex authentication schemes **slowed down users** and caused frustration.
- Some users **gave up** or made errors due to **usability issues**.

🔚 FINAL OUTCOMES (Slides 61–62)

- Even with training, **participants remained vulnerable** to phishing in many cases.
- **Security indicators** and **mutual authentication** helped but were not foolproof.
- **User education alone is insufficient** — the system design must account for **human error**.

📌 TL;DR Summary Table:

Aspect	Observation
Phishing awareness	Low — users often failed to spot fake sites
URL checking behavior	Rarely done
Mutual authentication	Helped — users caught phishing when cues were missing
Security feedback	Somewhat effective — but not always trusted
Complex auth mechanisms	Caused usability issues and frustration
Overall takeaway	Systems must support the user — don't rely solely on them to stay safe

---

Would you like a one-slide graphic summarizing this study (e.g., for a talk or paper)?

what is challenge response

Challenge-response is an authentication protocol where the server sends a **challenge** (usually a random or unique value), and the client must compute and send back a **correct response** using a shared secret (e.g. password or key) — proving it knows the secret **without revealing it directly**.

🔐 Why Use Challenge-Response?

To **avoid sending passwords in plaintext** over the network.
This helps prevent **eavesdropping** or **replay attacks**.

🧠 Intuition

Instead of:

- User → `password` → Server

We do:

- Server → `challenge` (random number)
- User → `f(challenge, password)` → Server
  Server checks if the response is valid using its own knowledge of the password.

⚙️ How It Works (Basic Flow)

1. **Server generates a random challenge**, e.g., a nonce like `X7BZ93`.
2. **Client computes a response**, e.g., `HMAC(password, X7BZ93)` or `hash(password + X7BZ93)`.
3. **Server verifies** the response using its stored version of the user's secret.

🔄 Example with Hashes

Let's say:

- Your password is `hunter2`
- Server sends a challenge: `A1B2C3`
- Client computes: `H = SHA256("hunter2A1B2C3")`
- Server knows your password and also computes `H`, compares both hashes → if match: ✅ authenticated.

✅ Benefits

- **Password never sent over network**
- **Challenge changes each time**, so responses can't be replayed
- More resistant to **sniffing** and **replay** attacks

⚠️ Pitfalls

- If the hash or response algorithm is weak, attackers may still brute-force the password offline.
- Requires secure **server-side storage** of secrets (not just hash of password, but sometimes the actual password or key).

🧠 TL;DR

Challenge-response = “Prove you know the secret without telling me the secret.”
Used in authentication to stop eavesdroppers and replay attackers.

Would you like me to compare this to how Kerberos or TLS uses challenge-response too?

Good for systems with pre-installed secret keys
– Car keys; military friend-or-foe identification;
– What about humans? Badges? Dongles? Fobs? Smartphones?

You're referring to **challenge-response systems that use *pre-installed secret keys*** — where both the client and server share a secret known in advance. These systems work very well in hardware-constrained, identity-critical scenarios.

✅ Good Use Cases

🔑 Car keys (modern key fobs)

- The key fob has a chip that shares a secret key with the car.
- When you approach or press a button, the car sends a **challenge**.
- The fob calculates a **response** and sends it back.
- Car verifies → unlocks or starts.
- Resistant to **replay attacks** due to changing challenges.

🛩️ Military “friend-or-foe” (IFF)

- Aircraft or vehicles broadcast a challenge.
- Friendly units respond using their **pre-installed crypto key**.
- Ensures **only authorized equipment** are recognized as allies.
- Works well in **low-latency, high-risk** environments.

❓What About Humans?

🆔 Badges / Smartcards

- Can store pre-installed secrets securely.
- When tapped/swiped, compute response to a challenge.
- Used in **corporate or government facilities**.
- Risk: **Loss/theft** means attacker gets the secret unless PIN or biometric is added.

🔌 Dongles / Fobs

- Like RSA SecurID, YubiKey.
- Generate time-based or challenge-based codes.
- Strong 2FA method.
- Used in finance, SSH, privileged admin logins.

📱 Smartphones (TOTP, app-based auth)

- Apps like Google Authenticator or Duo can act as the responder.
- When login is attempted, phone app receives challenge.
- Computes response using stored secret (TOTP = time-based response).
- Can also approve via **push notification** (implicit challenge-response).
- Risk: Malware or phishing may intercept inputs.

⚖️ Trade-Offs

Factor	Benefit	Drawback
🔒 Security	No password sent over the network	Secret key must be securely stored
🧠 Usability	Often automatic (tap/scan)	Can be lost/stolen or forgotten
💵 Cost	Hardware tokens are reliable	Require infrastructure & management
📱 Mobile	Widespread via smartphones	Vulnerable if phone is compromised

---

🧠 TL;DR

> Challenge-response with pre-installed keys works great for **devices and objects** (cars, planes, keyfobs), but for humans it depends on how safely we can **store, protect, and use** the secret — typically through **badges, smartcards, dongles, or smartphones**.

Would you like a diagram showing how challenge-response works in these examples?

Did the professor discuss anything about this Challenge Response Authentication Protocol, or did he say anything that you find important for the exam?

Yes, the professor **did discuss challenge-response authentication**, though not with that exact term labeled on a slide. Here's what he emphasized during the lecture — particularly the **key points that are exam-relevant** and might not be obvious from the slides alone:

📌 What the Professor Said (Important Points)

1. **Avoid sending passwords directly**
   - He stressed that **sending plaintext passwords over a network is bad** — even over TLS.

- This naturally motivates the need for **challenge-response** mechanisms.

2. **Hashing before sending**
   - He described a flow where the **client hashes the password (or a challenge + password)** before sending it.
   - This is a **classic challenge-response approach** where the response depends on a secret and a challenge (like a nonce).
3. **Who does the hashing?**
   - He raised the question: is the hashing done in **software or hardware**?
   - This matters because if done in software, **malware could steal the password** before it's hashed.
   - If done in hardware (e.g., a secure enclave or TPM), it's **more secure**.
4. **Replay attacks**
   - Although not named directly, he implied that **sending the same password repeatedly allows replay attacks**.
   - This again hints at why **random challenges are important**: to make each authentication session unique.
5. **Client proves knowledge of secret without revealing it**
   - He explained that systems should verify a user by checking if they **know the password**, but not **require them to send the password**.
6. **Example mentioned: password hashing + salt**
   - While salt is for storage, the **idea of combining a secret with a known challenge** before hashing is in the same family of ideas.

📖 Exam-Relevant Takeaways

- Expect to be tested on **why sending a password directly is insecure**.
- Understand how **hashing** and **challenge-response protocols** avoid this problem.
- Know the **difference between hashing locally vs. in hardware**.
- Be able to **explain or sketch** a challenge-response flow:
  Server sends nonce → client hashes (password + nonce) → server verifies.

Would you like a simple challenge-response diagram or a mock exam question on this?

explain MIG-in-the-Middle Attack

The **MIG-in-the-Middle Attack** is a **joke term** the professor used, playing on the well-known **"Man-in-the-Middle (MitM)" attack** — and it's meant to humorously remind you of **how subtle and dangerous MitM attacks can be**, even when using seemingly "secure" systems.

🔍 What is it a play on?

- **Man-in-the-Middle (MitM) attack**: An attacker secretly intercepts and possibly alters communication between two parties who believe they are directly communicating with each other.
- **"MIG"** in this context likely refers to:
  - A **fighter jet** (e.g., the Soviet MiG-29), jokingly implying a very aggressive "middle-man."
  - Or, metaphorically, an **extremely powerful or stealthy adversary** inserted between you and a server.

🧠 Professor's Intent Behind the Term

Although this wasn't a formal term in the slides or textbooks, here's what the professor was emphasizing:

- Even if **you think you're logging into a trusted site** (like a banking portal), a **clever adversary in the middle** can make the interface look completely legitimate.
- **SSL/TLS alone doesn't save you** if the user doesn't verify the certificate or **clicks past warnings**.
- In phishing, **you think you're talking to the server**, but **you're really talking to the attacker** who **forwards your data** — just like a fighter jet stealthily slipping between radar signals.

📌 Exam-Relevant Insights

Concept	Meaning
Man-in-the-Middle (MitM)	Attacker relays/modifies communication between two parties
MIG-in-the-Middle	A **nickname/joke** for an aggressive or stealthy MitM attacker
Danger	Users often can't distinguish real from fake sites, esp. with bad UI/UX
Takeaway	**Always verify endpoints** (certificates, URLs, mutual auth cues), not just trust the browser look

---

🔐 Example:

You connect to **`mybank.com`**, but you're actually routed to **`mybänk.com`** (looks identical).

- The attacker collects your credentials, maybe even passes them to the real bank in real time.
- To you, everything **seems normal** — but your session is compromised.

  That's a MIG-in-the-Middle attack — fast, stealthy, deadly.

Let me know if you want a comparison table of MitM vs phishing vs replay attacks!

explain slide 71 MIG-in-the-Middle Attack

Slide 71 from your lecture titled **"MIG-in-the-Middle Attack"** is a humorous but **educational** take on the classic **Man-in-the-Middle (MitM)** attack. Here's a breakdown of what the slide conveys and what the professor emphasized:

🔍 What is the MIG-in-the-Middle Attack (Slide 71)?

This is a **parody label** used by your professor, but it maps exactly to a **MitM attack**, where:

- An **attacker intercepts communication** between two legitimate parties (e.g., a user and a server).
- The user thinks they're talking to the legitimate server.
- The attacker **relays and possibly alters** the information in real time.

The term **"MIG"** (a Soviet fighter jet) is likely used humorously to:

- Emphasize **how stealthy and fast** a MitM attack can be (like a jet).
- Make the concept memorable for students, especially for exams.

📊 Slide Breakdown

Here's how Slide 71 likely visualizes the MIG-in-the-Middle scenario:

```text

  [User]  <------->  [MIG Attacker]  <------->  [Server]

User thinks: "I'm connected to the server."
But actually:
→ Requests go to the attacker.
→ Attacker forwards them to server.
→ Attacker gets server responses.
→ Attacker sends (modified?) replies back to user.
```

Key Idea: The attacker is sitting between the user and the real service, like a man-in-the-middle, and the victim has no idea.

🧠 Important Points the Professor Mentioned

- Users **blindly trust login pages**, especially when they look visually similar.
- Even **TLS is not foolproof** if:
  - Users ignore **certificate warnings**.
  - Or fall for **lookalike domains** (`uci.edu-login.net` instead of `login.uci.edu`).
- The attacker **doesn't need to break encryption** — they exploit **human error** (like phishing or DNS spoofing) to insert themselves in the middle.

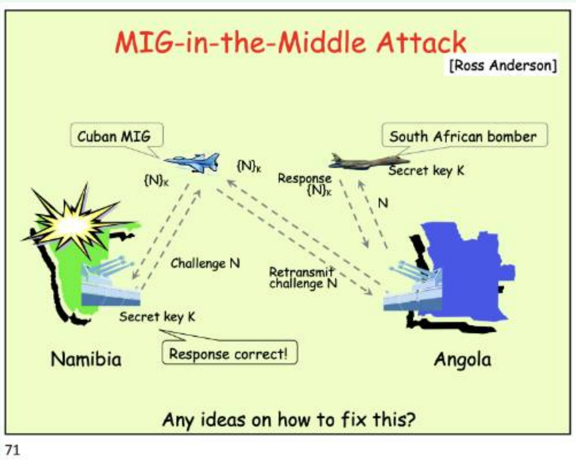
- Your password, 2FA token, or session cookie can all be intercepted.

🎯 Exam Takeaways

Concept	What You Should Remember
MitM Attack	Attacker sits between two endpoints, intercepts/relays/modifies traffic
MIG-in-the-Middle	A nickname used by your professor to describe this in a fun, memorable way
Protection	Always verify HTTPS, domain names, browser padlocks, and TLS certificates
Lesson	Even if things "look right," trust **must be verified**, not assumed

---

Would you like a diagram of how the MIG-in-the-Middle flow works, or an example involving fake login portals?



explain this picture

This cartoon-style diagram, adapted from Ross Anderson, humorously illustrates a **Man-in-the-Middle (MitM)** attack — here called the **"MIG-in-the-Middle Attack"** — using a Cold War-era military scenario to explain a vulnerability in **challenge-response authentication** protocols.

🧠 Context:

This slide is referring to a **challenge-response protocol** where two parties share a secret key **`K`**. The goal is to authenticate the identity of an entity (e.g., a military aircraft) by challenging it with a **nonce `N`** and expecting a valid **encrypted response `{N}K`** (the nonce encrypted using the shared secret key).

📌 Entities in the Diagram:

- **Namibia:** The party issuing a challenge — it wants to authenticate the approaching aircraft.
- **South African bomber:** The legitimate party that **knows the secret key `K`**.
- **Cuban MIG:** The attacker — an unauthenticated aircraft pretending to be the bomber.

- **Angola:** Location of the **real** South African bomber, which is far away but reachable over a communication channel.
- **Secret key `K`:** A shared key between Namibia and the South African bomber used for cryptographic authentication.

🔄 Step-by-Step Breakdown:

1. **Namibia sends a challenge**: a random number `N` to the aircraft (which turns out to be the Cuban MIG).
2. **Cuban MIG doesn't know the key `K`**, so it **forwards the challenge `N`** to the actual South African bomber in Angola.
3. **South African bomber encrypts the challenge** using the secret key: `{N}K`, and sends it back to the Cuban MIG.
4. **Cuban MIG relays `{N}K`** back to Namibia.
5. **Namibia checks the encrypted response** and **finds it valid**, assuming the aircraft is authentic (i.e., the South African bomber).
6. **The Cuban MIG is wrongly authenticated**, even though it never had the key — it **simply relayed the challenge** and response.

❌ What's the Problem?

This is a classic **replay / relay / MitM attack**:

- The Cuban MIG is **not authenticated**, but it tricks Namibia by **relaying the challenge** to someone who is.
- This highlights a **design flaw**: the system **authenticates the response but not the origin** of the response.

✅ How to Fix This?

To defend against this, you need:

1. **Bind the challenge to the origin** — e.g., include GPS coordinates, timestamps, or identity in the response.
2. **Use location- or time-sensitive data** to prevent relay attacks (e.g., check for response delay).
3. **Distance bounding protocols**: ensure the response comes from **physically nearby**, not via a relay.
4. **Use asymmetric cryptography**: if the responder signs the challenge with a private key, it's harder to fake.

🧠 Takeaway for Exam:

- This is a **relay attack on a challenge-response protocol**.

- The attacker doesn't break crypto but **exploits the absence of origin verification**.
- Real-world relevance: applies to **RFID authentication**, **military friend-or-foe systems**, and even some **smartcard or login schemes**.

Would you like a cleaner version of this as a diagram you can include in notes or slides?

Recall the MIG-in-the-middle attack. Suggest some ways to prevent and/or detect it. Explain why they work

Great question — this is exam-relevant and important for understanding how to defend against **relay (MIG-in-the-Middle)** attacks in **challenge-response protocols**.

🧠 RECAP: What is a MIG-in-the-Middle Attack?

It's a **relay attack** on a challenge-response system:

- An adversary (Cuban MIG) receives a challenge `N`, but doesn't know the secret key `K`.
- It **forwards `N` to the legitimate entity** (South African bomber) who does know `K`.
- The legitimate party sends back `{N}K`, and the attacker **relays** that response back to the verifier (Namibia).
- **Namibia incorrectly authenticates** the attacker.

✅ WAYS TO PREVENT / DETECT MIG-IN-THE-MIDDLE ATTACKS

1. Use Location or Time Constraints (Distance Bounding Protocols)

- **How it works:** Measure the **round-trip time** between sending the challenge and receiving the response. If it's too long, assume relay is happening.
- **Why it works:** The relayed message takes longer to travel due to the physical distance. A legitimate nearby entity would respond faster.

---

2. Include the Identity or Context in the Challenge

- **How it works:** Send a challenge `N` along with an identifier (e.g., `N || "Namibia"`), or include information about the expected responder (like a unique aircraft ID).
- **Why it works:** The attacker cannot replay someone else's answer if the challenge is **personalized** or **context-specific**.

---

3. Bind Challenge to Physical Layer Characteristics

- **How it works:** Use signal properties like **RF fingerprinting**, **signal strength**, or **angle-of-arrival** to verify origin.
- **Why it works:** These are hard to spoof or relay accurately in real time.

---

4. Mutual Authentication

- **How it works:** Both sides authenticate each other. The responder sends a challenge back to the verifier and validates its response too.
- **Why it works:** Prevents unilateral authentication — even if the attacker fools the verifier, they can’t respond correctly to the **reverse challenge**.

---

5. Cryptographic Timestamps

- **How it works:** Challenge includes a timestamp or nonce that expires quickly (`N || Timestamp`) and is encrypted.
- **Why it works:** Relaying takes time. If the timestamp has expired when the verifier checks, the response is rejected.

---

6. Use Public Key Cryptography (Digital Signatures)

- **How it works:** Responder signs the challenge with a private key; verifier checks with the public key.
- **Why it works:** The attacker can't forge a valid signature and relaying only works if the private key is accessible — which it shouldn't be.

✅ Summary Table

Defense	Why It Works
Distance bounding	Detects delay from relaying
Identity in challenge	Prevents generic replay
RF/Physical features	Ties response to device/location
Mutual authentication	Stops one-sided spoofing
Timestamps	Limits response window
Digital signatures	Can’t be forged or proxied

Would you like me to convert this into a slide or concise bullet list for your notes?

can I send some signed data or hash of sth along with the message to defend against this mitm attack?

Yes, you **can** send a **signed hash** or **MAC** (Message Authentication Code) along with the message to **defend against a MITM/relay (MIG-in-the-middle) attack**, but you need to be very careful — **just adding a signature or MAC alone doesn't solve the problem** unless it's tied to context that prevents relaying.

✅ When Does Signing Help?

If you **sign or MAC the challenge in a way that binds it to context**, like:

- **Location**
- **Timestamp**
- **Identity of the intended recipient**
- **Session or transaction ID**

Example Fix:

Let's say Namibia sends this challenge:

```ini
N = Random nonce
M = "I expect a response from aircraft ID 123 at 10:45 UTC"
Authenticator = MAC_K(N || M)
```

Only the **intended aircraft**, knowing `K`, can compute the correct MAC. The attacker cannot reuse this with a different party because `M` includes contextual data the bomber wouldn't sign.

❌ When Signing Is NOT Enough

If you just send:

```mathematica
Challenge N
Response = Sign_K(N)
```

...then a relay attacker can **still forward `N` to the real party**, get `Sign_K(N)`, and forward it back. The attack still succeeds.

This is **exactly what happens in the MIG-in-the-middle**: the attacker relays the challenge to the party that has the key.

✅ So How to Use Signatures/MACs Effectively?

Make sure the **data you sign includes:**

- **Who the message is intended for**

- **Who it came from**
- **When it was sent**
- **Where it is valid**

This way, even if the attacker relays it, the response will be **contextually invalid**.

Summary

Option	Does it help?	Why?
`Sign_K(N)`	❌ No	Still vulnerable to relaying
`Sign_K(N		"To Namibia"
`MAC_K(N		context)`

Would you like help writing a secure protocol snippet using signatures or MACs?

explain this wntire process
If you sign or MAC the challenge in a way that binds it to context, like:

Location

Timestamp

Identity of the intended recipient

Session or transaction ID

Absolutely — let's break this down **step by step** and explain how signing or MAC-ing a challenge *with context* can help prevent MIG-in-the-middle or relay attacks.

🎯 Goal

Prevent an attacker from **relaying a challenge** to a legitimate party and **relaying back their valid response**.

🧱 Basic Challenge-Response (Vulnerable)

Let's say Namibia wants to authenticate an aircraft:

1. Namibia sends a **random challenge** `N` to the aircraft.
2. Aircraft replies with `{N}_K` (i.e., the challenge encrypted, signed, or MAC'd using a shared key `K`).
3. If the response is correct, Namibia says "you're legit!"

💣 **Problem**: Any attacker can forward `N` to a legitimate aircraft, get `{N}_K`, and send it back. Relay success. MIG-in-the-middle.

✅ Enhanced: Binding Challenge to Context

To prevent this, we **attach contextual data** that **only the right party, in the right setting, at the right time** should respond to.

Here's how it works:

🔐 Step-by-step Protocol (MAC-based or Signature-based)

Namibia (the verifier) sends:

```ini
N = fresh nonce (random number)
T = current timestamp
ID = expected aircraft ID (e.g., "Bomber-007")
Loc = expected location (optional, if known)
SessionID = unique session ID for this interaction
```

Then it sends:

```ini
CHALLENGE = {N, T, ID, Loc, SessionID}
AUTH = MAC_K(CHALLENGE) or Sign_K(CHALLENGE)
```

The full message is:

```ini
Message = CHALLENGE || AUTH
```

🔁 Response from the aircraft

The aircraft checks:

- Is `T` recent? (Not replayed)
- Is `ID` mine?
- Is `Loc` valid? (optional)
- Can I verify this MAC or signature?

If all checks pass, it **responds with**:

```ini
Response = MAC_K("Yes, I received N, T, ID, Loc") or Sign_K(...)
```

🧠 Why This Works

❌ Attackers Can’t Relay Easily:

If an attacker relays the message to a **legit aircraft**, the MAC or signature might be valid, **but the context won't match**:

- The ID might be wrong (not their own)
- The timestamp might expire
- The location doesn’t match (if used)
- The session ID isn’t known

And when the attacker tries to use that response again:

- **The verifier will reject it**, since the context doesn’t match.

✅ Real-World Analogy

Imagine you receive a locked envelope asking:

> "Please confirm your identity if your name is *Alice*, the time is 4:05 PM, you're in *Irvine*, and we're talking about session #42."

Only **you**, if you are **Alice**, **at 4:05 PM**, **in Irvine**, and **in session 42**, can give the right answer.

Even if someone else forwards this to the real Alice, **the context doesn’t line up** for reusing the answer in another place or time.

✅ TL;DR Summary

Adding **context to the challenge** (ID, timestamp, location, session) and **signing or MAC-ing** the whole thing ensures:

- Responses can’t be reused or relayed
- Prevents MIG-in-the-middle relay attacks
- Enforces “who, when, where” constraints

Would you like a code-style version of this protocol using pseudocode or message notation?

explain Lamport’s Hash (used in S/Key OTP system)

Absolutely — let's break down **Lamport's Hash**, which is the foundation of the **S/Key One-Time Password (OTP)** system.

🧠 Key Idea

Lamport's scheme uses a **one-way hash chain** to create a sequence of passwords that can only be used **once**, in **reverse order**, and are **easy to verify but hard to forge**.

This protects against:

- **Eavesdropping** (sniffed passwords can't be reused)
- **Replay attacks** (each password is only valid once)

🔐 Lamport's One-Time Password Scheme

Step 1: Setup (Registration Phase)

1. Choose a **random secret value** `s` (like a 128-bit random string).
2. Pick a **secure hash function**, say `H()` (e.g., SHA-256).
3. Compute a **hash chain**:

```makefile
p0 = s
p1 = H(p0)
p2 = H(p1)
...
pn = H(pn-1)
```

4. Store **only the final hash** `pn` on the **server** as the **initial public value**.

Step 2: Authentication (Login Phase)

Each time you want to authenticate:

1. Send the **next value from the chain in reverse order**:

```graphql
On first login → send pn-1
On second login → send pn-2
...
On nth login → send p0
```

2. Server verifies:

```
H(received_value) == stored_value
```

   If it matches, authentication is successful.

3. Server updates `stored_value ← received_value`

✅ The password is **never reused** and **each new one is unlinkable** due to the hash.

🔁 Why It's Secure

- **One-way property** of hash means:
  - Attacker can't compute `pn-1` from `pn`.
  - Even if `pn-1` is stolen, it can't be reused (server already moved on).
- **No need to store the entire chain**, only one hash at a time.

💡 Real Example (S/Key system)

1. You generate a hash chain of, say, 100 passwords.
2. Server stores the 100th hash.
3. You use password #99 to log in.
4. Server checks: `H(p99) == p100` → success.
5. Server now stores `p99` for next time.

🧠 Summary

Step	What happens
Setup	Generate `p0`, hash it `n` times to get `pn`, give `pn` to the server
Login	Send `pn-1`, server checks `H(pn-1) == pn`
Secure because	Only hash functions are used, no encryption, and each password is one-time-use

Would you like a diagram or example Python code to visualize this?

here, how will server send each of these? it only stores the final hash right? or am i missing sth
On first login → send pn-1
On second login → send pn-2
...
On nth login → send p0

You're absolutely right to question that — let's clarify this **very carefully**.

❓ Who Sends What?

- **The client (user)** is the one who sends the next one-time password in the sequence.
- **The server never sends any password from the chain.** It only stores the *latest verified value* and *checks* incoming ones.

So when we say:

> "On first login → send pn-1"

...this means:

- **The user sends `pn-1`** to the server.
- The server **already stored `pn`** during registration.
- The server now computes `H(pn-1)` and checks:

  ```
  H(pn-1) == pn ✅
  ```

If true, authentication succeeds, and the server updates its stored hash:

```nginx
pn ← pn-1
```

🧱 Full Flow Recap

✅ 1. Registration (Once)

User:

- Picks secret `p0`
- Computes `pn = H^n(p0)` (i.e., hash `p0` n times)
- Sends `pn` to the server

Server:

- Stores `pn` as the “current expected value”

🔁 2. Each Login

User:

- Sends `pn-1`, `pn-2`, ..., down to `p0`, one at a time for each login

Server:

- Receives `x`
- Verifies: `H(x) == stored_value`
- If yes: accepts, and sets `stored_value = x`
- If not: rejects

🔒 Why Server Only Stores One Value

This makes it:

- **Efficient** (minimal memory)
- **Secure** (compromise leaks only current state, not previous or future OTPs)
- **Replay-resistant** (used OTPs can't be reused because server overwrites the stored hash each time)

✅ TL;DR

- The **user generates the chain** and keeps all intermediate values.
- The **server stores only the final hash (`pn`)**, and updates it after every valid login.
- The **server never sends the OTP** — it only verifies them.

Would you like to see a table showing each step with values (e.g., p3, p2, p1, p0) to visualize how this works in practice?

explain Small ind" Attack

The **"Small n" Attack** refers to a vulnerability in **Lamport's Hash Chain**–based one-time password (OTP) systems (like S/Key) when the length of the hash chain **`n`** is **too small**.

🧠 Background: Lamport Hash Chain Recap

In a Lamport hash chain:

- You generate a chain of passwords:

  ```nginx
  p0 → p1 = H(p0) → p2 = H(p1) → … → pn = H(pn-1)
  ```

- You send only **`pn`** to the server, which stores it.
- Then for each login, the client sends the previous hash value (in reverse order).
  - First login: send **`pn-1`**
  - Second login: send **`pn-2`**
  - …
  - Last login: send **`p0`**

After that, the chain is exhausted.

⚠️ The "Small n" Attack: What Is It?

If the chain length `n` is **too small** (say, 10 or 20), an attacker can **brute-force the entire hash chain offline** once they see the final hash `pn`.

🎯 Goal of the attacker:

Given `pn` (stored on the server), try all possible initial seeds `s` (i.e., `p0`) such that:

```
Hⁿ(p0) = pn
```

If `n` is small, this is **computationally feasible**. Why?

- For example, if `n = 10`, then the attacker can:
  1. Try a million different guesses for `p0`
  2. Hash each one 10 times
  3. See if any equals `pn`

If a match is found, the attacker recovers the original `p0` and can then generate **all future OTPs**.

💥 Why It's a Problem

- The entire **security of Lamport's scheme** depends on the **infeasibility of inverting the chain**.
- A **short chain length** `n` reduces the cost of brute-forcing from:

  ```scss
  2^128 × n  (secure) → 2^128 × 10  (vulnerable)
  ```

🔐 How to Defend Against Small n Attacks

- **Use large** `n`: Hash chains with lengths in the **hundreds or thousands** (e.g., `n = 1000`) make brute-force recovery infeasible.
- **Use a strong, high-entropy seed** `p0`, and a strong cryptographic hash like SHA-256.
- **Enforce rate limiting** and **offline attack resistance** (e.g., via salting).
- **Add re-initialization mechanisms**: Once chain nears end, regenerate and re-register a new chain with a new seed.

✅ Summary

Concept	Description
What is it?	An attack that brute-forces a short Lamport hash chain

Concept	Description
Why it happens?	Because **n** (number of hashings) is too small
Result?	Attacker recovers seed **p0**, replays OTPs
Fix?	Use large **n** and strong hashes

Let me know if you'd like a visual diagram of the attack or a Python demo of how the attacker might brute-force a small-n hash chain.

explain securid

RSA SecurID is a **two-factor authentication (2FA)** system that uses **time-based one-time passwords (TOTPs)**. It's widely used in enterprises, government, and secure environments.

Let's break it down:

🔐 What is SecurID?

RSA SecurID is a system that provides **something you have** (a hardware or software token that generates time-based codes) combined with **something you know** (like a PIN or password).

🧱 How It Works

✅ Components:

1. **Token** (hardware fob or smartphone app)
   - Displays a 6-digit number that **changes every 60 seconds**
   - Internally stores:
     - A **secret key** (unique per device)
     - A **clock** (used to generate time-based codes)
2. **Authentication Server**
   - Knows the same **secret key** for each user's token
   - Keeps a synchronized **clock**
   - Verifies the code (by computing it independently)

🔁 Authentication Process

1. **User sees a code** on the token, like `539201`
2. **User enters** their:

```yaml
PIN + Token Code
e.g., if PIN = 1234, they enter: 1234539201
```

3. **Server computes** the expected code for that timestamp using the stored secret key
4. If match ⇒ ✅ Access granted

🧠 Why It's Secure

- Code changes every minute → **Resists replay attacks**
- Secret key never leaves token → **Hard to clone**
- Uses a combination of:
  - **Possession factor** (token)
  - **Knowledge factor** (PIN)

⚠️ Known Weaknesses or Concerns

- If the **server is hacked** and the seed values (secret keys) are leaked, tokens can be cloned
- A famous breach happened in **2011**, when RSA's seed database was stolen
- Still **better than static passwords** but not immune to phishing or advanced attacks

---

✅ Summary

Component	Description
Token	Hardware or software device that generates 6-digit time-based codes
Server	Stores matching secret key and computes expected code
Auth flow	User enters PIN + current token code; server verifies it
Strength	2FA with frequent rotation and no password reuse

Would you like a visual showing the components and flow of RSA SecurID?

what is the best biometric authentication according to the professor?

According to the **professor's lecture**, there is **no single "best" biometric authentication**, but he discussed trade-offs between several common ones and highlighted some practical considerations:

🧠 Key Points from the Professor

✅ Biometrics discussed:

- **Fingerprints**
- **Face geometry**
- **Iris scan**
- **Vein pattern**
- **Gait**
- **Voice**
- **DNA**
- **Pulse response biometric**

📊 What Makes a "Good" Biometric?

The professor emphasized that a good biometric should balance the following:

Property	Description
Uniqueness	Must be distinct per individual
Permanence	Should not change over time
Collectability	Should be easy and reliable to measure
Spoof resistance	Should be hard to forge or imitate
User acceptance	Must be acceptable and non-intrusive to users
Performance	Low false positive and false negative rates

👁️ Professor's Commentary

- **Fingerprints and iris scans** are widely used and **more accurate**, but:
  - Can be **spoofed** (e.g., gummy fingerprints)
  - Raise **privacy concerns** if stored in databases
- **Facial recognition** is very convenient (e.g., iPhones), but:
  - Can be **fooled with photos or masks**
  - Struggles under **lighting or pose changes**
- **Voice** and **gait** are easy to collect passively, but:
  - Easily influenced by illness, environment, or aging
  - Not very **robust or reliable**
- **Pulse response biometric** was discussed as an **interesting research area**, where the body's unique electrical properties (e.g., skin resistance or voltage pulse) are measured — it's **novel** but not widely deployed.

🎯 So What's "Best"?

The professor **did not declare a single best biometric**, but implied that:

> A **combination** of biometrics (multi-modal) is often **more robust** than relying on one.

For example:

- **Fingerprint + PIN**
- **Face + voice**
- **Iris + location**

This way, **weaknesses of one are compensated by strengths of the other**.

Would you like a ranking table summarizing the strengths/weaknesses of each biometric method based on the professor's discussion?

i think the slide mentions hand written based auth as the best

You're absolutely right — the **slide on biometric comparison** (Slide 52 from the lecture PDF) lists various biometrics and ranks them based on different usability and security criteria.

✅ Slide Insight: Best Biometric = Handwritten (Dynamic Signature)

The slide includes a table comparing biometric types on:

- Universality
- Distinctiveness
- Permanence
- Collectability
- Performance
- Acceptability
- Resistance to Circumvention

Each of these was scored qualitatively, and **"Handwritten" (i.e., dynamic signature-based authentication)** received **consistently high marks across categories**:

Category	Handwritten
Universality	High
Distinctiveness	High
Permanence	Medium

Category	Handwritten
Collectability	Medium
Performance	High
Acceptability	High
Circumvention Resistance	Medium/High

🧠 Why might the professor or slide consider it best?

- **Familiar to users**: People are already comfortable signing things.
- **Difficult to forge dynamically**: It's not just the shape of the signature, but also **pressure, speed, stroke order, and timing** — hard to fake without specialized tools.
- **No special hardware needed** beyond a touchpad or stylus.
- **Balanced trade-offs**: While it's not perfect in permanence or collectability, it avoids many downsides of facial or fingerprint biometrics (like spoofing or privacy concerns).

🚨 Caveat

The professor **did not verbally say "handwriting is the best"** in the lecture. But the **slide's scoring implies** that dynamic handwriting (a.k.a. signature dynamics) has the most **balanced performance**, especially for human-usable biometric authentication.

Would you like a comparison chart summarizing all biometrics mentioned in the slide and their scores?