CS203 Gene's Class - Lec 3 - Cryptography

I will give you a transcribed version of class recording. you will read it very thorughly and understand. I have some tasks for you

Sounds good! Go ahead and share the transcribed class recording whenever you're ready. Once I have it, I'll read through it carefully and get ready for the tasks you have in mind.

where instead of doing what you see here, every block is encrypted with a different key.
Now, you start with the same key, but for every block that you encrypt,
you derive a unique key based on the master key and the block number, the position of the block.
So that makes sure that every block knows its place and cannot be moved around or deleted, okay?
So there are other modes that I'm not going to go into here, but if you're in, because that's crypto, right,
then we're trying to kind of bulldoze for crypto fairly quickly because you should know something about it.
Now, as you also know, there's two types of crypto, symmetric or conventional, and public key or asymmetric.
All right, and the differences are fairly obvious.
In a symmetric crypto, both parties, assuming there are two parties to communicate,
both parties need to know the same key, okay?
And the biggest problem there is how to distribute that key,
how to, who comes up with the key, and how does the other party learn the key, right?
So it's key distribution or key management.
In a public key crypto, you have this radically, totally different approach, right,
where the public key is known for everybody, but private keys are not, right?
So, typical situation, I put my public key on my webpage or on Canvas.
As enrolled students, you will see it as public, and then any one of you can send me a secure message
using my public key.
As long as you trust that it is mine, okay, you can send me a secure message,
and none of you can interfere with other people's messages or read them, right, or anything like that, right?
So only I can decrypt because I have, I am the only one who has a private key.
So that kind of solves, sort of, kind of the distribution problem,
because my public key is not secret, so there's no, like, headache of how do I distribute it, right?
I just post it somewhere.
Yeah?
So in public key crypto, the plain text message goes for the encryption algorithm,
which is computed with the message itself.
It must be allergies.
Ah, Bob's public key is known to Alice.
Alice uses the message, and Bob's public key to produce the ciphertext.

That's the notation we use in this lecture.
I'm borrowing from the book.
And then Bob, who knows a private key, not the same as the public key, okay,
that's the whole point, will somehow compute a transformation on the ciphertext
that will negate the effects of encryption and spit out the same message.
That's where magic happens.
In order for this magic to work, you need to have some functions, right?
Encrypt and decrypt functions, where essentially these operations with two different keys reverse each other.
And this is different from symmetric crypto, where you can say, well, XOR, for example, right?
They use one-time path, XOR with a key encrypts the plaintext, and, again, XORing the ciphertext with a key decrypts,
because XOR of something with something, the same cancels it out.
But here, we have two different keys, and so, in addition, if you know the public key,
like in my example, I'm the instructor, I post my public key on the bulletin board somewhere,
that public key should not allow you to let my secret key, or my private key.
If that was the case, it would be useless, right?
So, it's not impossible, because obviously there is some relationship between my public key and my private key, right?
So, by saying impossible, that's a high bar.
I'm not saying it's impossible, but it should be computationally very hard for you to learn my private key.
Right? An example of that, well, Diffie-Hellman, but it's not truly an encryption scheme, it's more of a key management scheme.
The best known example is RSA, right?
Rivest-Chemeyer Edelman.
Dating back to 1978, the public key as a concept was actually invented in 1974 by a guy who doesn't get enough credit,
partially because he's super weird, even for a computer scientist, his name is Ralph Merkle.
And if you study computer security, you probably come across his name because there's something called Merkle-Hastries,
and if you haven't heard of him, you should know his name, Merkle, M-E-R-K-L-E, Ralph Merkle.
He invented the very first public key method that at least was published, right?
That he doesn't get credit for, for some reason.
Anyway, RSA, how it works, you first, to set up the system, you have to choose two large prime numbers.
By large, I mean, while it says 2048 bits each, probably even 1,000 bits is okay.
So, you pick two large numbers, let's say they are 1,044 bits long.
When you multiply them together, which you should, you get another large integer, which is no longer prime, right,
because it's a multiplication of two prime numbers.
It is a composite number, but it has an interesting property that it can only be factored in one way, right, with two primes.
And prime numbers, as you know, have no factors.
So, you come up with this giant number, which is a product of two large primes.
The large primes must be about the same in length.
They are not the same, same is bad, they are different, but they are about the same length, in terms of, like, digits of bits.
And there is a good reason for that.
So, you compute the product, which is not difficult, just another huge number.
And then you compute another product, which is called Z, which is a P-1 times Q-1.

Okay?
P-1 times U-1.
Notice, if P and Q are prime, there's no way P-1 is prime or Q-1 is prime, right, in fact, you'll know they are even.
Yeah?
Right.
So, that product is not, is not a product of two prime.
All right.
Then, you choose a number E.
That E is very important.
Of course, that E is less than N, making greater than N makes absolutely zero sense.
And it must have no common factors with Z.
But P, but N, Z, must have no common factors with Z.
When two numbers have no common factors, they are called, in mathematics, relatively prime.
Which means, E and Z must be relatively prime for each other.
To be relatively prime, the numbers don't need to be prime.
Right?
Are you following me?
I'm waking up.
Right?
They don't need to be prime.
They just need to have no common factors.
Now, if you choose E to be prime, that helps.
Because then, it obviously has no common factors with Z minus 1, sorry, with Z.
Right?
That's okay.
You can, in fact, choose E to be 3.
If it helps you think about it.
It doesn't have to be large.
Can't be, but it doesn't have to be.
Then, so these 3 and 4 are the crucial steps.
If you've never heard of it, then I see your eyes are not glazing over,
which means some of you have probably never heard of this.
Amazing.
You pick a D.
Another number, D.
But D, you don't pick.
Actually, you compute it.
You compute it that it must be the case that E times D minus 1 is divisible by Z.
So it's not like you actually pick.
Right?
E you pick.
D you compute.
So it must be the case, in other words, so if this doesn't make sense, think of this.
E times D must be 1 mod Z.
Remember the mod operator?
Everybody loves mod operators, right?
So, in other words, they are called inverses of each other mod Z.
Now, there's a whole number theoretic background that goes into this, which, again, the 134 slides that I thought you should go in excruciating detail.
So, if you really love this stuff, and you really want to understand, go there, okay?

But there's, like, a whole mathematical foundation about why this works and why this is important.
So, okay, at the end, step five, what is your public key?
Your public key is N, modulus, the product of T and Q, and E, that number you pick.
Together, N and E are your public key.
You throw them on your web page, you put them on your canvas, whatever.
You are free to give it away.
But your private key, well, it says N and D, that's actually incorrect.
D, P, and Q are your private key.
Because N is public.
So, if you really want to know, it's D.
That D is the one you're going to use to be the private key.
And the whole trick with the RSA is that it seems, we believe, right?
You're not the mathiest, you understand.
We believe that knowing E, N, and E, it is really, really hard
to compute a D.
We don't have a proof.
We, what I mean, the world, at least.
The math, the mathematicians, the computer scientists.
There is no formal proof that it is computationally difficult.
But there has been no evidence to the contrary.
Proof by hundreds of demons, right?
It's, yes, proof by repetition and trial, repetition and trial.
People like me keep repeating it.
We think it's hard, and we think it's hard.
And other people try, and other people try and break it, and yet they fail.
One day somebody might succeed.
In which case, we're screwed.
At least a little bit.
Because a lot of today's secure communications are sort of based on RSA problem.
Okay?
Any questions?
Just an anecdote.
Until 2002, everybody thought that primality testing, does everybody know what primality
testing means?
I'll give you an integer, tell me if it's primed.
I don't mean like 17.
I mean like a thousand digit integer.
I'll give you a huge integer, and I'll tell you, is it primed?
That problem is called primality testing.
Until 2002, it was believed to be computationally hard.
Well, then in 2002, there was a paper by two very bright Indian students, and a no-good professor.
We just tagged them along.
But the students did the work.
I know it's for a fact.
And they proved, formally, mathematically, that primality testing is, in fact, polynomial.
So, the entire math world was super surprised.
So, these things happen.
That's why we can never be absolutely sure.
Right.
Okay.
So, RSA encryption.

How does it work?
Given modulus N.
N is called the modulus.
Okay?
And that exponent E.
E is called E because it's exponent, and also because it's for encryption.
Right?
E.
And D is for decryption.
Right?
So, that's why we call it D.
To encrypt a message, you simply compute, take a message, raise it to the power E, mod N.
Everybody should know how the mod operator works, right?
So, N to the E, mod N.
The mod part is a crucial one.
If you don't do mod, if you simply take a message and exponentiate to E, first, you'll have an expansion.
Right?
Because exponentiation expands.
Right?
The message will grow.
And if E is a large number, the message will grow out of bounds.
Way out of bounds.
That's one problem.
The other problem would be insecure.
Because even given a very large number, to take a discrete, sorry, to create a log, not a discrete one, a log of a number, okay, is easy.
Or take a square root of an number, it's easy.
Take a cube root of a number, it's easy.
And E is public, right?
So, if I give you M to the E, and I say, can you take an E root of that and give me M?
That's not a hard problem.
That's polynomial.
But if I give you M to the E mod N, and ask you to take an E root of that, that's a hard problem.
So, again, we believe.
So, the magic is here.
Right?
It's difficult, apparently, to take an E root of the cypertext, which is M to the E mod N.
But also, and this is real magic, if you take M to the E mod N, and then raise it to the power D, you get back M.
Even though E and D are different.
Okay?
There is a reason for this.
So, if you've never, now, to be honest, there's no penalty.
Is anybody here who has not really seen the guts of the public kicker therapy before?
One, two.
Anybody else?
I'm guessing.
Okay.
So, the way you think about public kicker therapy is like this.

This is like what I tell the guts.
Anybody use a U.S. Postal mailbox recently?
Remember those blue boxes?
You touch one inappropriately.
You molest it.
It's a federal offense.
You'll go to federal prison.
But don't.
Now, the interface that the box gives you is what?
A hole.
A slit.
Right?
Open it.
You stick your mail in.
You close it.
Once you throw it in, you can't.
You can't.
Right?
You try to do anything, federal prison.
And if you're not a citizen these days, God knows.
God help.
Yeah.
So.
Bad joke.
I don't know.
Too soon.
So, the thing is, anybody can use a mailbox, right?
You can just come up, open, slide letter in.
To get things out, you need to be a postman with a key.
That's a public decryptor.
Anybody can encrypt.
Right?
The public key interface is public.
But to decrypt, you need to have a private key.
So, this is a trickery with RSA.
Let's look at an example.
You can trust me.
It works.
You can verify in your favorite calculator.
If we pick toy example, right?
P equals Q.
You will never pick such strong numbers.
First one.
Ever.
But this is just a toy example.
If P equals Q, P equals 5, Q equals 7.
These are prime numbers.
P minus 1 times Q minus 1 is 24.
And N is 35.
Still correct?
Of two primes.

We pick E equals 5.
Okay?
I know, silly, but okay.
Then, the inverse of P, mod 24, is 29.
Is that possible?
What's 29 times 5?
145.
Right?
145, mod 24.
Mod 24.
1.
Now, I didn't pick this example.
This is from the Kuroza-Kuroza book.
Why did I pick 29?
I should have picked 5 there instead of 49.
Right?
Because 49 does not exist.
It's greater than 24.
So, actually, in this silly example, which I did not come up with, but I should have really corrected it, E and D are the same.
It will never happen in real life.
So, that's because 29 is actually 5 mod Z.
So, let's look at the encryption, let's follow along.
Yeah, I want to encrypt the letter I, that's not a 1, that's an M.
And so, I is the 12th letter of the alphabet, so actually, what the message is, it's M is 12.
So, then, you raise it to the power E, which is 29.
Stupid.
You get this number, which, actually, mod N is 17.
So, that's the Cypher case.
17.
Then, when you raise it to the D, again, this is stupid.
It should be, actually, 5.
Raised to the 5, not 29, which, in that case, would be just about as long as that.
Not that number, but as long as that.
It will be 12, which is that.
All right.
So, the whole point is, to convince you, to demonstrate that encryption and decryption are inverses of each other.
Even though you encrypt with one number and decrypt with another number, you get back the original plaintext.
Not only that, but if you first apply the D, meaning that you first decrypt a plaintext, and then encrypt it, the same thing works.
Which means that encryption and decryption are commutative.
Why is that important?
Anybody know?
Why would you want to decrypt a message that's not encrypted?
No idea?
Do you have the same thing in decrypting?
Yes.
It's true.
If you reverse those, yes.

But on a more serious note, have you all ever heard of signatures?
Digital signatures?
Well, a digital signature is computed by taking a plaintext message, signing it with your private key, then releasing the message, then releasing the message, and the signature.
Notice in that case, I said releasing the message, not hiding the message.
Public key encryption is about hiding.
Public key signatures are not about hiding the message.
It's to prove that you generated the message.
So let's come back to our classroom example.
You went to my classroom page, a web page, and a canvas, and obtained my public key.
Now you have nothing to tell me.
You're not sending me anything.
But I have something to tell you.
And now what I have to tell you is this class is canceled.
Okay?
Due to low attendance and poor performance, the class is hereby canceled.
Don't come to class next time.
The message sounds a little weird, unhinged, bizarre.
Okay?
Is it really me?
Or is it some hacker in Krakistan, you know, hiding in his mom's basement, sending this?
Well, you verified my signature on the message.
By using my public key to check that the signature was computed with my private key that only I could have known.
Make sense?
In that case, I'm not hiding the message to you all.
You are verifying the origin of the message and its integrity.
So that's where decrypting the plaintext, essentially, is useful.
Yeah?
So in theory, like, you always have had some trust with public key photography,
because you have to have some trusted public key release.
Ah.
Okay.
What you're talking about is how do you distribute public keys.
Right.
Okay.
That's the new headache.
Symmetric photography has this big headache of how do you distribute secret key.
You have to distribute secret.
A little bird has to sit on the shelf and tell you.
God has to talk to you.
I don't know.
I have to meet you in physically, meet you, give you something.
Our phones have to touch NFCs, right, in order to transmit the secret key.
But with public key, the problem is not as bad, right?
It's more like, do I trust this is your public key?
Right.
So, very good question.
That is very true.
It's called a public key distribution problem.
Again, I emphasize, it is not nearly as nasty as the symmetric key distribution problem.

Because what we're distributing here is not fundamentally secret, right?
What we're distributing here has to be reliable.
It has to be like, yes, it has to be my public key, not some other person's public key.
So, well, we're coming to that.
But to make a shortcut, you have to basically trust the place you give it from.
So, of course, if I give it to you personally, yes, it's mine, but then how is this different from symmetric?
If you trust the UCI ICS webpage, you go there and say my public key is there, yeah, fine.
If you trust Canvas, right, and I post my public key there and you trust that I authenticate it, it all has to be like a chain of trust, right?
That if the Canvas authenticated me based on my UCI credentials and blah, blah, blah, and therefore only I could have posted my public key and my Canvas page, et cetera, et cetera.
It's only going to be as secure as the weakest link.
So, there's no, like, protocol that exists that would not be.
Ah, there are.
There are protocols.
There are.
But ultimately, there's no, like, Big Bang, like some elegant description of how all this is done.
Think about your browser.
What happens when you install your browser on, let's say, you buy a new computer and it comes with some god-awful thing like Safari?
I don't know, I don't know, or worse yet before you used to be an Internet Explorer, before you died in that time you got it.
And, you know, and then you wanted to install Chrome, or Tor, perhaps, or Brave, I don't know.
And what would you do?
You went to that, you used a very crappy browser that you got there, right, to go somewhere and, like, download the code that did this game, I didn't try to package it, depending on what's that business.
And you had to trust that place.
And then, when you install the browser, assuming you're kind of like, okay, I trust that, it came with, like, some called roots of trust.
With these kind of, what's called CAs, right, Certification Authority keys that certify all the other keys for all the websites that you deal with using HTTPS or TLS, right?
So, you see, now, in operation today, when you use your browser, 99% of the time, you see HTTPS, if you pay attention.
And that means you're using TLS with whatever web server you're talking to.
But the original connection established, when you first went to that web server, had to have what's called a handshake, and we'll cover that later.
And that handshake involved, public key, of that web server.
Not of yours.
Usually, you are not involved.
You don't have a public key like HTTPS.
I mean, you can't.
It's obscure.
Usually, you want to authenticate your web server.
You want to go, I'm going to Wells Fargo, and not, you know, .com, and not whatfargo.cn, or .fargo.kr, I don't know, North Korea somewhere, right?
So, that's important.
And that's all based on the roots of trust that the original browser package came with, so, get that in?
Yeah.

Yeah.
Anyway, so, back to RSA, where does it work?
Well, there's lots of number-theoretic results, and I'm going to explore you with, but essentially, the reason, remember, we picked E and D, so they're inverses of each other, mod Z.
Well, it turns out, when you do things, you're modular, mod N, right?
So, if you say, take X to the Y, you raise X to the power of Y, mod N, in the exponent, things are also modular.
But, not mod N, they're mod what's called phi of N, or Z.
See, mod V minus 1 times Q minus 1.
So, everything gets wrapped around in the exponent.
And so, when you take X and raise it to the power of E, and then again raise it to the power of D, the two exponents multiplied, right?
E times D.
But, we pick them so that E times D, mod E minus 5 times Q minus 1 is 1.
And that's the explanation.
Right?
To use RSA, you don't need to understand this.
You just need to see the result, okay, that it works.
And it's been working for the last 40 years, almost 40 years.
So, this is what I alluded to before, is that you can use the encryption key, then decrypt, you know, using the private key.
Or you can first use the private key, and then follow that by decryption, sort of decryption with a public key.
But then, when you use RSA for digital signatures, that's exactly what happens.
So, yes, I take a message, I apply the private key to the message.
I take that result, right?
This decryption of the message.
And I send it to you, together with the actual plain text of the message.
So, now you have a plain text of the message, and decryption of the message with my private key.
Yeah?
With me so far?
Now, then what you do is to verify that it really comes from me.
You take the decryption of the message, encrypt it using E.
Remember, E reverses D, D reverses E.
So, you should get back the message.
You compare it to the plain text.
Is it the same?
If it's the same, only I could have sent it.
And it has not been modified.
Okay?
So, that's the basics of digital signatures.
So, and data integrity.
So, typically, we need both data integrity and origin authentication.
Right?
These terms are trivial, right?
Integrity means something has not been modified.
It has integrity.
So, and origin authentication means it comes from a place I think it comes from.
Like, if GeneSutic sign it, well, then you know it comes from GeneSutic.
I can verify.
Okay?

So, but what we don't do is we don't use digital signatures on messages.
Because messages, files, can be very long.
And digital signatures are based on public key cryptography.
So, if we're going to exponentiate, and trust me, this is the problem with public key cryptography.
Not only, I mean, there are other problems, but one of them is the usage problem is that it's slow.
It does not use efficient primitives like XORs and SHIFTS and ORs and ANDs, like the symmetric cypress.
It uses large number arithmetic.
A large number arithmetic requires all kinds of tricks to be implemented efficiently in software.
And so, anything that is in public key world can never match the speed of symmetric cryptography.
No way.
No chance.
It's several orders of magnitude slow.
So, using public key cryptography on bulk data is wasteful and senseless.
So, we don't do that.
So, in particular, when it comes to signing a message, or signing a file, or signing a video, right?
Or signing a software distribution, which is, by the way, when you get, you know, your next software update for the operating system, it's signed.
And the delay you experience often in, like, a pre-installation is the verification of the signature.
So, we don't sign it directly with RSA or anything like that.
What we do is compute a hash function over the thing.
And a hash function is essentially like an integrity token.
And then we sign the hash function.
Make sense?
Can I ask one question?
Yeah.
What output should be a fixed length value HM?
The who?
What output should be a fixed length value HM rather than...
Ah, fixed length.
Well, because we want to make sure that what we compute is a canonical length, right?
If we vary the length, if, like, a very large input produces a larger output, you know, we cannot write standard code, right?
It's harder.
But it can also be, like, we hash a long value into a short value, like, if we have some...
Well, remember, all of this is done because it's not humans who operate on this, right?
It's software and no hardware that operates on these hash functions, computes them.
So, in the implementation, that would compute variable length output would be much strictly less efficient than that that computes a standard length output.
So, if we are not considering efficient, but purely security is a bad...
Eh, purely security.
Well, for pure security, you may not really need...
You can just sign every block.
Right?
You can just sign every block using publicly crypto.
You don't need a hash function.
See, the hash function, the beauty of having a hash function is the function acts as a checksum of sort, right?
As an integrity protector of the message.
It allows you to detect manipulations of the message without having to inspect every block.
Because what I think is that, for example, for some encrypto algorithm, for example, RSA, I think

they encrypt information with different length, right?
Yes.
That's not a problem.
Yes.
That's not a problem.
Block site, whether it's a symmetric key, crypto, or public key.
But you never want to use public key more than absolutely necessary.
Yes.
Because it's several orders of magnitude slower.
So, when you want to protect integrity of a message or a file of data, and that data is more than one block long, which is whatever the block is in the same order, you want it's much cheaper to compute a hash function.
Standard length, standard length, standard size, where standard varies, right?
Depending on what's that.
20 years ago, 128 bits were okay.
Right.
Now, at least 256 bits.
Okay?
And that's a good reason for that.
So, you want to agree on the implementation of a hash function that produces a fixed length output regardless of the length of the input, right?
Because that facilitates faster, more efficient implementation.
And also, the security level remains the same, right?
Because what we don't have time to cover is hash functions that are chosen for many reasons.
One of them is the length of the output.
If the length of the output is too short, then functions become susceptible to something called the birthday paradox.
Has anybody heard of the birthday paradox?
Some of them have.
Okay?
Anyway, the common wisdom is the hash function has to be roughly double what an encryption should
be for a block cipher.
So, for example, today, we're okay using a 128-bit encryption key for a block cipher.
But the minimum hash length we use today, I think, should be like 256 or so.
Right?
And that's one of the reasons is the birthday paradox.
Which, if you all like, I could cover.
Sounds like a lot of you don't know what it is.
But anyway, so it is important to compute the canonical fixed length output regardless of the input.
And these three properties are very important.
So, that is for convenience.
Right?
For convenience, it's nice to have a fixed length output.
Okay?
So, you could also say, well, why not compute a minimum 256 and go from there.
And then for a long way, you could.
But again, it will not be efficient.
But these three important properties must hold for a function to be a cryptographic hash function.
So, first of all, it must be one-way.
One-wayness means that if I give you an output of a hash function,

it must be computationally hard for you to come up with input.
Unless I give you the input.
Okay?
So again, right?
Given z, finding x such that h of x is z must be computationally different.
Okay?
This is also important.
They're all equally important.
Given x, finding another value, y is a different value, such that they have what's called a collision, a hash function collision.
So, finding another value such that h of x equals h of y.
That process of finding another y must be computationally hard.
Okay?
Now, we know for a fact that collisions exist, right?
Because a hash function, as we just discussed, takes on arbitrary input and reduces it to a fixed length output.
It is impossible not to have collisions, right?
A block cipher will not have collisions, right?
Because it takes a plaintext and with a given key transforms into a ciphertext.
You cannot have collisions.
It's impossible for two plaintexts under the same key to produce the same ciphertext.
Do you see that?
But here, collisions are given.
They exist.
But finding them must be hard.
Okay?
And this last one is subtle.
And the subtle is different from this.
Here, this last third product, this third product, this is called strong collision resistance.
This is called weak collision resistance.
This is called strong collision resistance.
Here, you are not given x.
You are said, you are told, find me any two that collide.
Any two distinct values that produce the same hash output.
That's called strong collision resistance.
All today's hash functions that you will come across have all of these properties.
Now, are they proven?
Formally, mathematically?
No.
They are beaten to death with testing and evaluation and so on.
But they are not formally mathematically proven.
That's how the world is.
So if you look at something like IP, right, to the IPv4.
So IPv4 has what's called header checksum.
Have you ever seen it in network class maybe?
It's a very trivial checksum, right?
It's just like a once complement and once complement of summation of all the words in the header.
And so checksums are weak hash.
We don't, of course, use the prescriptor guide to cache because they have collisions.
Right?
And this is just an example of how if you use insufficiently long or trivial checksums in place of a real

hash function, it's easy to find collisions.
Okay, so let's come back to another real word application, which is message authentication codes.
Now, this is not to be confused with digital signatures.
Right?
This is symmetric crypto.
And this is what happens when Alice and Bob share a secret.
Okay?
And this is, in fact, what happens in TLS, right, after the initial TLS entry or in IPsec after the keys down.
Alice has a secret, Bob has the same secret, and Alice wants to send a message to Bob.
Now, in this example, the message M in deep blue is not encrypted.
That's because encryption isn't always important, right?
In some cases, the message is not confidential, but Bob wants to make sure that the message hasn't been modified in transit and really comes from Alice.
So what happens here?
Well, Alice has a secret key K. Bob presumably has the same secret key K.
The message is hashed, okay, and the message and the hash of the message and the key is sent along.
So you see the blue part, the green part, that gets sent over the internet or whatever communication channel.
The Bob receives them and then recomputes the hash and compares to itself.
So now, again, keep in mind, this is not a digital signature.
This is message authentication code with symmetric character.
Okay?
Now, in the real world, this is what they use.
This is actually a standard canonical way of computing a key-based, symmetric key-based MAC.
It's used all over the place in many different purposes, HMAC, hashMAC, and it involves this, right?
You take a message, M, you see the message M over there on the red.
The very first thing you do is you take the secret key, XOR it with a constant.
iPad is a constant, not an Apple device.
You XOR it with a constant.
The constant is public.
And then you hash that the result of an XOR concatenated with a message.
Then you take the result of that.
That returns a fixed length result, right?
This H.
And then you take the same key, XOR it with an outer path, concatenated with that result, and hash it again.
So there's an inner hash and an outer hash.
The inner hash is long, meaning it computes over the entire message M, and the outer hash just computes over two values.
And that's supposed to resist a lot of attacks and so on.
So this is, you're going to implement a M, a scheme that needs a message authentication code.
This is what you should use.
On inventor-owned.
Alright, so for hash functions, historically, like when I was growing up, essentially we used MD5.
Now, if you see MD5 run away, because that's not secure for a number of years.
It was only 128 hits long.
Insecurity of it was, essentially, its resistance to the birth repair was about 2 to the 64, which is considered insecure.
Which means it takes 2 to the 64 trials to find, on average, to find a collision in MD5.

Today, we use SHA-2 and SHA-3.
SHA-2 is about 20 years old, more than 20 years old, but it's a US standard.
You want to do business with the government, you must use it.
If you want to get a stamp of like, what is it, ISO certification,
and at any company you work for using secure hash functions, you must adhere to this standard.
So, SHA-2 is a very flexible hash function, so is SHA-3, which is completely better, but slower.
So, it works with, SHA-2 works with 160, 224, 256, 384, 512 outputs.
That means, coming back to your question, with this hash function, you can configure it to produce output that you want.
Now, you cannot pick 259 if you want more, but you can pick from those.
So, if you need something faster, with smaller hashes, right, because maybe bandwidth or storage is an issue, you would do this.
If you need something super secure, that will be secure for like, I don't know, 20 years, you'll probably want to pick 512.
Okay?
This is more for efficiency.
Right?
Here, that, varying the block size means how many, how many bits at a time does the function process?
Because the hash function does not swallow a message as a whole.
It, like, like, like ciphers, like block ciphers, it processes one block at a time.
Okay?
So, this is just a block size.
It's not so much security relevant.
Right?
It's more like how efficient it is.
Okay?
Go ahead, question.
What do you mean by SHA-3 strictly better?
Well, it's better because it's redesigned.
Right?
So, it's not like they were, as I said, there were no, like, specific weaknesses found in SHA-2.
But, these functions tend to age.
And to describe, if you ask me, like, why and how, I really don't know.
Because, because designing hash functions is like designing symmetric ciphers, they're very similar principles using designing both.
That's like, not a science.
That's an art.
And, there's probably, like, a couple hundred people in the world who are good at it.
There are people who do it, but only about a couple hundred people who will do it well.
And so, they typically, these designs come from, like, well-known, and there's not, like, one genius behind any of them.
There's usually a team of people.
And, they're, they're repeatedly good at it.
Right?
They've come up with, like, several designs.
I know some of them.
And, they're very specialized breed.
So, they have, you know, it's like artists.
Right?
You don't know how, how they come up with what stuff they do.

But, generally, they have, I mean, they have to demonstrate everything.
So, these designs are public.
Everything is specified.
There's nothing secret.
Some of the criteria, why they take this over that, they might only know.
But, the designs are fully open.
And so, whenever they are, before they are standardized, right?
Before, well before, there's a competition.
Typically, the U.S. government, when it wants to have a standard for either encryption, or hashing, or signature, they open what's called a competition.
It's worldwide.
And, anybody in the world can enter and propose a candidate encryption function, or hash function.
And, in fact, this, I think both of these, are not from the U.S.
The U.S.
In fact, the AES, the advanced encryption stadium, we all used to be, also, is not from the U.S.
It originates from Belgium.
So, and then, anyways, it's like the American Idol.
I mean, it's a circuit competition with several rounds and eliminations.
And, in the end, only one wins.
Okay.
So, yeah, this one is, has more flexibility with block sizes, you see.
I don't know why it has these particular features, right?
Yeah.
From 576 to 152.
As far as output, you see, it does not go beyond, but it's supposed to be better.
It, it, it, it obviates a hundred, it does not do 160 anymore.
Output because, you can see this, because 160-bit output would be 2 to the 80 resistant.
First, the attacks, and 2 to the 80 is no longer considered secure, like, putting up five, six years from now.
So, FYI, today, if you're going to use encryption, 80 bits of entropy, 80-bit key, assuming it's random, is the absolute minimum you should use.
The absolute minimum.
And that's just for short term.
If you look for long term secrecy, you should not use 80 bits.
You should not use 80 bits.
You should use at least 128 bits.
Yeah.
Only.
All right.
So, the digital signatures, what I already mentioned before, basically, they are sort of people say, people say, again, these are slides I, from, from the book, from the Crows and Rose book.
They are, to a point, slightly analogous to handwritten signatures.
Right?
The way that we sign them.
But they're actually not really.
In a sense that, if our handwritten signatures, indeed, may be unique.
Right?
Especially if you record the patterns of movement and pressure.
You know, that's true.
But when we do sign a document, and remember signing a document in real life, like, actual

physical document, what is it actually you sign?
Like, I don't know, you sign some official legal document.
What is it you sign?
Well, generally, you sign a corner of it.
Right?
Some place on a page.
And maybe there is a document that has 20 pages, but you don't sign for every page.
Right?
And even if you do sign every page, you don't sign the page.
You sign a part of the page.
Somewhere.
Well, how difficult is it to, I don't know, get rid of your signature?
Or cut out your signature and replace it with another piece of paper.
Or take the pages of the document that you did not sign and replace them with some other pages.
Or delete pages.
Or insert pages.
Right?
So in a physical world, we kind of like, don't think about those things.
But they're totally possible.
Right?
Now you think about recent real estate contracts, or God knows why.
I mean, you sign one page and the rest may be your initial or something.
Like, how does initial mean?
Initial, there's no wait anywhere.
Right?
So, but in the digital world, digital signatures actually mean, it's like, when you sign something, you sign every bit of that.
Okay?
Every single bit of the message is signed.
So it's strictly stronger.
Right?
So if Bob signs a document, he establishes that he creates the entire document and he is the one who creates it.
Right?
Yeah.
So, with all that, and we already talked about it, right?
So Bob basically uses this private key.
I mean, I kind of jokingly say decrypt.
We don't say decrypt.
He signs.
But Bob signs with the private key.
And Alice verifies with the private key.
But, of course, as I said before, we don't actually sign the message.
We first hash the message and we sign the hash.
And the idea is that the message and the hash are like, there's like an umbilical cord between the two.
They're tied together.
Because if you can replace the message with another message that has the same hash, you won the game.
Right?

Okay.
So, this is all the same.
Uh, no, no, no.
Sign hash.
I said all that.
Boring.
Boring.
Okay.
Now, we come back to what you raised, which is, okay, so what about this subtle headache?
Not the big headache, but the subtle headache of how do we know, well, first of, where do we get the public key?
How do we know they have the right public key?
Well, that's called public key certification.
Welcome to hell.
Public key certification.
This is something that we deal with every day today, even without knowing it.
Right?
So, what Alice obtains while it's public key, how does she know it's public key apart?
Right?
And the solution is called CA's, or Certification Authority.
Maybe you've heard the term PKI.
Right?
Public key infrastructure.
Right?
That's what's required to solve this headache.
Public key infrastructure.
That is usually hierarchical structure of certification authorities.
With some God on top.
Right?
The root of trust.
Okay?
Okay?
Don't say to anybody I'm made by the religion.
It's just a God in terms of public keys.
Okay?
So, public key God on top, and then branches, like in a tree, right?
Going up to lower level certification authority.
Okay?
Okay?
So, and then at some point, you know, at the bottom of the tree, the leaves are users.
Or entities that are, that have public keys.
And Alice's and Bob's.
Or a web service.
Right?
In TLS there would be web service.
Because what happens underneath when you go to a website.
Right?
You do.
You type in a URL.
Or you click on a URL.
Your browser says, oh, cool.

New things.
Extracts the domain name part of the URL.
Okay?
Looks up using DNS.
Right?
And then collect.
And then TLS, what it tries to do, it sees first checks in the local caches.
Do I have already talked to this website?
If I talk to this website, do I have a public key certificate for that?
And the key is up.
If yes, good.
If not, then he needs to go and get it.
And what is he going to get it?
He's going to get it from the server.
And the server's going to reply with his public key, but it better be a public key certificate because just saying a public key isn't enough.
So actually, what your browser is going to do is make sure that the public key certificate that that server sends you is traceable to the root of some public key infrastructure that your browser trusts.
And that comes back to that browser installation problem.
The browser, when you installed it, pray to God, securely, blah, blah, blah.
And it has some root of trust that says, I don't know, Panamanian Certification Authority.
All right?
Federal certification of Panamanian.
Panamanian.
Panamanian.
And you are going to a bank in Panama somewhere, right?
Your web browser.
You're actually trying to connect to that.
I don't know.
Banco Central de Panama.
And it gives you a public key certificate.
And the public key certificate is signed by the Central Certification Authority of Panama.
Okay, so what does your browser do?
Your browser says, oh, I trust the Central Authority of Panama and its public key.
I use its public key to verify the certificate of the central bank I'm going to.
Is it verified?
Yes.
Is it expired?
No.
Good.
This is a simple example.
It could go deeper.
Because the public key infrastructure could be deeper.
Okay.
So, in other words, we need these certification authorities as points of, like, trust.
And what they do, really, what they do is bind public keys to entities.
That's really the main function of the Certification Authority is to say, this guy has this public key.
This guy has that public key.
Okay?
Now, before it all begins, somebody has to ask for a public key certificate.

Meaning, if the bank in Panama wants to have a server and he wants that server to be securely connected to,
he needs to obtain a public key certificate, which means it needs to generate a public key and a private key,
go to the CA, and obtain a certificate.
Pay money.
Usually in the commercial world, you pay money to get a certificate.
And you need to verify your identity, meaning you need to bring credentials.
Right?
So, a lot of this process is actually administrative and it takes place offline.
Especially for, like, web servers, commercial entities.
For people like us, no, we can get a certificate, a low-grade certificate, often for free.
Okay?
But the certificate will contain a group, essentially, some kind of a binding.
Right?
Essentially, it says, oh, this is Alice Smith, and this certificate is to be used for email only, and it's valid from this date until this date.
And, um, I don't know, Alice lives in California.
And here's the public key.
I know it's called the public key.
And also attached to the certificate is the signature.
And, well, and the name of the CA, of the CA that issues the certificate and the signature.
So it's not unlike a driver's license or a passport, in a way, if you think about it.
Except your driver's license is a passport, well, actually, they do contain a key, but never mind you.
Well, the way you look at them today, right, the driver's license is a passport.
There are no cryptographic things in there.
At least you don't see them.
Now, if you have a recent driver's license or a fairly recent passport, chances are there are cryptographic keys inside them.
Because there's, like, RFID, data, all sorts of biometric information inside them.
That's not quite an analogy here.
Right?
So everybody understands?
So, certificates are absolutely necessary.
You cannot live without them today.
So, when Alice wants to get Bob's public key, she needs to get it from somewhere.
It doesn't matter, actually, where she finds it.
So, for example, my public key, you could have found it on the floor somewhere.
You could have picked it out from a garbage can.
It's not important how you got it.
And what's important is how you verify it.
Okay?
So, you get the certificate.
The first thing you do is you check for expiration.
Because it's cheap or something.
You look at it.
Is the certificate expired?
No, it's expired.
Why bother checking anything else?
Is it valid yet?
Meaning, is it a certificate that is not valid yet, but will be valid in the future?

It's not valid yet.
No point in checking anything.
Okay.
Now, you check for validity.
Meaning, you check the CA's public key.
Assuming you trust the CA's public key, very widely, the signatures.
Right?
Ah, but you're not done yet.
Not done.
Because a certificate is what's called in operating systems, or in general, a capability.
Maybe you've heard before in other classes.
A capability is like a bearer document.
Like a driver's license is a capability.
A passport is a capability.
It's given to you.
You have it.
But if you behave badly, it could be reformed.
A visa is another capability.
It could be reformed.
A driver's license could be reformed.
A passport could be reformed.
Well, same thing with public key certificates.
And because you could have found my public key certificate anywhere, you have no idea if it's been reformed.
Right?
Because it's still valid.
It says valid.
It expires in 2029.
And valid from 2021.
Great.
But how do you know I haven't behaved badly and it's been reformed since?
Or maybe I've lost my credibility.
There are many reasons it's been reformed.
Some of them have nothing to do with my bad behavior.
Maybe I've been robbed.
Maybe a gun was put to my head and I was forced to devolve my privacy.
Certificate is gone.
Right?
It should be revoked.
And the only person or the only entity in the world that knows that it's revoked is the issuing CA.
Right?
Don't ask me.
I might not want to tell you.
Or I'll tell you now why.
The only authoritative entity in the world that knows whether my certificate is revoked or not is the issuing CA.
Just like the only entity in the world that can tell you whether your U.S. passport is revoked is the U.S. State Department.
Okay?
Or the only entity that can tell you if your California license is valid is DMV in California.
So don't ask DMV in Illinois or federal government.

Ask DMV in California.
That's the CA.
Say again.
So that means somehow you need to check for revocation.
That's another headache.
Another big headache.
Revocation.
And once you've solved all those problems and passed all those tests, then you extract the public key and then you can verify the signature.
Amazing.
Right.
And, okay, this is a very old fashioned format.
Public key certificates don't look like that anymore because this is old.
But essentially it could look like that.
And if you go into your browser or if you use like a mail client like I do, like Thunderbird, you can go into it and the cuts of it will see what a public key, or should display what public key certificates look like.
And they look a bit more complex than what you see here.
But basically the idea is to show you, okay, it says this certificate belongs to, to whom it was issued.
Right?
This, in this case, the certificate belongs to another CA.
Right?
Because CA themselves have certificates.
Who was it issued to?
Oh, in this case, it was issued by itself.
Weird, right?
Certificate issued to an entity by the same entity.
Does that make any sense?
Not, not, not, not like immediately, but you think about it.
Who certifies God?
I mean, really, right?
If you're a God of your own public key infrastructure, right?
You're like a root of the tree.
Who's going to certify you?
The only choice you have is to certify yourself.
That's called a self-signed certificate.
Right?
So when you get your browser installation, for example, like a new browser installation, it comes with all these roots of trust.
They're usually self-signed.
Okay?
For that reason.
Because there's no global world authority.
You know what I mean?
Couldn't it be like a cycle or something?
Because there are multiple CA's, right?
Sometimes they sign each other.
Right.
Like I vouch for that guy.
And so that's called a peer sign.
Right.

They will do it.
And that's, that's, that happens.
That happens.
I don't think it happens in browsers, but it happens in other, in other applications.
Why would they allow self-signing if that is an option?
Ah, because at least it preserves the tech.
So, ah, good point, good point.
Um, self-signing in general is useful.
It's better than nothing because it tells you that whoever signed it knows the private key.
Otherwise they just say, here's my private public key and it's complete nonsense.
At least I know whoever, whoever self-signed knows the private key.
All right.
So it says who, to whom it belongs, by whom it was issued, always a serial number.
There is always a serial number.
It must be unique per seat, per issuer.
So you can definitely see two certificates with exactly the same serial number, but they apply different issues.
So the combination of issuer serial number must be unique.
Uh, that is valid from.
Okay.
So this is super old, 1996 to 2028.
Pretty generous, right?
32 year validity.
Nobody issued certificates this long.
This used to be the case.
People were much more optimistic in the beginning of the internet.
Uh, but yeah, it says, you must say from to.
Always.
Fingerprint is a hash of a certificate.
Right?
That's just for integrity and checking.
And then you see there's like other stuff there.
It's like, this certificate belongs to the certifying authority.
Accept the certificate for certifying human subject.
Like for example, this certificate should never be used for web authentication.
Right?
It will not work with TLS because, you see, this says only email.
So this is for sending email.
Only.
Now in reality, one second.
In reality, the formula certificate is pretty abstruse.
There's a standard called X509.
To be precise, X509 version 3.
That's an international standard.
And if you do a quick web search on X509v3, you'll see a ASN1 notation.
If you know what that means.
Specification, again, not for the faint of heart.
It's very sleep inducing.
It tells you all these fields that must be found in a certificate.
Some are mandatory.
Like most of the fields we've covered here are mandatory.

And then there are lots of options.
Okay?
Like some certificates don't even have a key.
There are like authorization certificates.
You know, some are like assigned to a goal or something like that.
Question?
Yes.
I'm just wondering why we need to pay for the certificate.
And the, yeah.
Well, you, you, so, so, you is a individual, right?
A student.
Yeah, yeah.
You can get a free certificate.
Like there are, there are, if you want, I can, I can tell you the next time.
There are services that will give you a low grade, low grade certificate, but just saying, oh, you know, here's an email address.
Give us your email address.
You enter your email address.
As long as you're a student at UCIEDU, they'll do a verification, like one of those, you know, send your code, right?
Verify that you own this email address or access it.
And they'll issue you a certificate.
Commercial entities or high grades.
Well, because the certification authorities are businesses.
What do they do?
Oh, they make money on this.
Yeah, yeah, yeah.
Oh, but okay.
Okay.
Not that I love them or have any connection to any situation.
But consider this.
If you are a CEA, right?
And you do have a for profit business, you need to be amazingly secure.
Because the key that you sign, certificates, is like the family jewels.
If somebody gets that key, they can now impersonate you and issue certificates.
This is super dangerous.
Your whole reputation is gone.
You lose all your customers overnight.
Right?
Your key leaks and your business is dead.
Like suicide immediately.
Now, does anybody know the word Stuxnet?
How many people heard of the word Stuxnet?
Not many.
God, guys, here we go.
Stuxnet, 2014 timeframe, was an incident when a malware made its way into Natanz nuclear reactor in Iran and severely and irreplaceably damaged centrifuges, or uranium centrifuges, producing centrifuges in that facility.
Okay.
Now, that malware jumped what's called air gaps.
Meaning that Iranian nuclear facility, Iranian is not stupid.

It is not connected to the internet.
Okay?
But it means there's an air gap between the internet and that facility.
Like it is in the United States and many places.
So that malware came in the form of a software update for a PLC, Programming Logic Controller, that was operating these turbines, the centrifuge turbines in the nuclear plant.
Why am I telling you this story?
Because the software update was signed, digitally signed, by a CA in Taiwan that was compromised.
Okay?
So, nobody knows for a fact who did it.
But everybody kind of knows that it's very likely Mossad and the NSA together mounted this joint operation to retard the Iranian nuclear weapons program.
Okay?
To do so, it was a massive logistic operation.
They had to patiently wait.
This malware did not infect computers outside Iran.
It actually was very like, am I in Iran?
No.
Sometimes it would delete itself.
And then it would like, look, am I running on a particular Windows industrial system, Windows control?
Oh, okay.
Then is it made by Siemens?
Because it particularly targeted specific turbines made by Siemens, which is a German company, right?
And they knew that whoever did this knew that the Siemens PLC turbines were running inside this nuclear reactor.
In Iran, it would make them essentially spin at such speed that they burned out or something like that.
I don't know the exact details.
What we do know is that they were damaged beyond repair and severely retarded the Iranian nuclear program.
Now, does that answer your question?
Or not yet?
The moral of the story is somebody got into the supply chain there and stole the private key of that CA in Taiwan.
Without that, this whole incident would have not happened because they started signing certificates for code distribution and then allowed whoever mounted the attack to sign fake code distribution.
Right?
The code updates.
Right?
You get the idea?
Yeah.
So one thing leads to another, to another, to another.
So that's why if you are a CA, if you're running a CA as a business, you need to have amazing, not just electronic, but physical security.
Because chances are all these, these kinds of attacks start with some amount of social engineering that involve humans and bribery, et cetera, et cetera.
So physical security for a CA business is extremely important.
So, and of course, disconnection from the internet.
Typically CA will not run accessible from the internet.

At least not high grade CA.
They will be disconnected.
They will actually be a human interface between, let's say requests will come in to issue certificates.
And then at the end of the day, somebody will like bunch up those requests and physically like carry them over to the actual CA.
So I'm in a bunker, you know, in a Faraday cage room and feed them to the CA and then get certificates and come out and turn off the CA.
So basically the idea is this has to be a very secure operation.
And so it's not, it's not, it's not like so easy to run a CA.
So today's, today, I mean, you have all kinds of CAs.
Of course, if you go to some third world place, like, I don't know, middle of nowhere, you know, Marshall Islands, you know,
their CA is not going to be as secure as a CA in France, you know, but the prices will also be different, right?
And the trust people will have in the certificate issued by a CA in like Solomon Islands in Vanuatu would be different from a, from a certificate issued, let's say by some reputable California authority.
Yeah.
So this is all, it's not super technical, right?
This is all very reputation.
Yeah.
All right.
So that's done to be done.
Yes.
See you on Thursday.
So if you haven't yet again, start thinking about who you're going to work with, what you're going to work on in the project.
If you have questions, by the way, was somebody there yesterday because I had some music playing in my office and not yesterday, Monday, yeah, Monday yesterday.
And I may have missed somebody not yesterday in the office hours.
No?
Okay.
Good.
Good.
Good.

Update memory?
YesNo
Got it—I've read through the full transcript carefully and have a solid grasp on the material. Go ahead and hit me with the tasks you have in mind!

prepare a detailed lecture note of this class. be as detailed as possible

Cryptography Lecture Notes

Here's a detailed lecture note based on the transcript you provided. It's organized by topic and covers all key concepts, examples, and takeaways from the class. If you want a visual PDF or slide version of this, or a more concise summary, I can help with that too!

include any discussion that prof did, not just topics

The lecture notes are now fully updated to include not just the technical content but also the professor's discussions, humor, real-world examples, and class anecdotes. Let me know if you want this converted into a study guide, PDF, or visual summary!

be more detailed on what prof discussed

Cryptography Lecture Notes

Lecture Notes: Cryptography and Public Key Infrastructure

---

1. Encryption Basics: Block Encryption and Unique Keys

- In secure encryption systems, each block of data can be encrypted using a unique key derived from a master key and the block's number.
- This prevents tampering with block order and integrity, ensuring each block "knows its place" ar cannot be moved or deleted.
- The professor emphasized that this avoids problems like block shuffling and replay attacks.
- He briefly referenced that there are other modes (e.g., CBC, ECB), but opted to skip deep cryptographic theory in favor of conceptual clarity.

---

2. Symmetric vs. Asymmetric (Public Key) Cryptography

Symmetric Cryptography

- Both sender and receiver share the same secret key.
- Major challenge: **Key distribution**—how does the receiver get the secret key?
- The professor noted the absurdity of this with examples like birds whispering keys, meeting in person, phones touching, or God communicating the key.
- It was made clear that this headache scales terribly with large systems or open networks.

Asymmetric (Public Key) Cryptography

- Each user has a **public key** (shared openly) and a **private key** (kept secret).
- Anyone can encrypt using the public key; only the private key holder can decrypt.
- Professor illustrated this with a **U.S. postal mailbox analogy**:
  - The mailbox slot = public key. Anyone can drop letters in.

- The locked box = private key. Only the mail carrier (key holder) can retrieve the contents.
- Tampering or opening it without a key is a federal offense — a humorous warning.

---

3. RSA Cryptosystem

Historical Context

- Ralph Merkle introduced the public key idea in 1974. Despite his eccentricity, he deserves recognition.
- RSA (1978) became the de facto standard, created by Rivest, Shamir, and Adleman.

Setup Details

1. Choose two large primes `P` and `Q` (not equal).
2. Compute modulus `N = P * Q`.
3. Compute Euler's totient: `Z = (P-1)(Q-1)`.
4. Choose encryption exponent `E` (must be relatively prime to `Z`).
    - E.g., 3 or 65537 are commonly used.
5. Compute decryption exponent `D` such that `E * D ≡ 1 mod Z`.
    - This is the modular inverse.

Clarifications from Class

- If `E` is prime, it's easier to ensure it's relatively prime to `Z`.
- If `D` is derived properly, the RSA relationship holds: `M = (M^E)^D mod N`.
- Professor emphasized: knowing `N` and `E` should not help an attacker compute `D`.
- There is **no proof** RSA is hard—just long-term empirical confidence.

Example Discussion

- Toy Example: P = 5, Q = 7 → N = 35, Z = 24
- Chose E = 5, D = 29 (since 5 × 29 ≡ 1 mod 24)
- Encrypt: M = 12 → C = 12^5 mod 35 = 17
- Decrypt: 17^29 mod 35 = 12
- The professor noted this example is flawed: E and D shouldn't normally be the same mod Z, and real RSA wouldn't use such tiny numbers.

On RSA's Magic

- Highlighted the modular exponentiation trick.
- Without modulus, large exponentiation would be insecure and unbounded.
- But with modulus, recovering the message without `D` becomes computationally hard.

4. Digital Signatures and Commutativity

- Public key encryption enables **digital signatures**:
  - Sign using the **private key**.
  - Verify using the **public key**.
- Prof described a situation: “What if I send a bizarre message like 'class is canceled due to low attendance and poor performance’?”
  - Verifying the signature confirms it came from him—not a hacker.
- Digital signatures prove:
  - **Origin authentication**: Who sent it?
  - **Data integrity**: Was it altered?

---

5. Hash Functions

Concept and Purpose

- Maps variable-length input to fixed-length digest.
- Instead of signing large messages, sign a hash.
- Saves computation and is efficient.

Key Properties Explained

1. **One-wayness**: Given a digest, can't find original input.
2. **Weak collision resistance**: Given one input, can't find another with same hash.
3. **Strong collision resistance**: Can't find any two inputs with same hash.

Why Fixed Length?

- Easier implementation.
- Fixed security target (e.g., 256-bit digest).
- Avoids issues like birthday paradox.
  - Professor asked if anyone knew what this was—few hands.
  - Briefly discussed: It reduces the effort needed to find a hash collision.

Practical Commentary

- **MD5** = obsolete. **SHA-2** and **SHA-3** are modern standards.
- SHA-3 was selected through a public global competition.
- Designing secure hashes is an art; only a few experts can do it well.

---

6. Message Authentication Codes (MACs)

- Symmetric authentication mechanism.
- MAC ≠ Signature. It's not about identity—it's about integrity.
- Alice sends a message + MAC tag.
  - Bob recomputes the MAC and checks for equality.

Real-World Implementation: HMAC

- Used everywhere: TLS, IPsec.
- Includes an **inner and outer hash** to improve robustness:
  - `H(K ⊕ oPad || H(K ⊕ iPad || M))`
- Professor stressed that public-key crypto is slow; hashes and MACs are efficient for bulk data.

---

7. Public Key Infrastructure (PKI) and Certificates

The Trust Problem

- "How do you know a public key really belongs to Bob?"
- **Solution**: Certificates signed by trusted **Certification Authorities (CAs)**.

Certificates and Validation

- Contain:
  - Owner identity
  - Key usage (e.g., web server, email)
  - Validity period
  - CA signature
- **Steps to verify**:
  1. Check validity window (not before/after).
  2. Verify CA signature.
  3. Check revocation status.

Revocation and Trust Chains

- A revoked certificate may still appear valid unless explicitly checked.
- Only the issuing CA can confirm revocation.
- Analogy: Just like DMV controls driver's license status.
- Trust is transitive: Root CA → Intermediate CA → Entity cert

Self-Signed Certs

- Prof asked: “Who certifies God?”
  - Root CAs certify themselves.
  - Browsers come pre-installed with trusted root certs.

---

8. Real-World Example: Stuxnet Attack

- Malware that crippled Iran's nuclear centrifuges.
- It used a **forged software update**, signed with a stolen Taiwanese CA key.
- Malware delivered via USB into **air-gapped systems**.
- Prof emphasized this as a classic supply chain attack.
- Reinforced that CA private keys are “family jewels” and must be extremely well protected.

---

9. Final Notes and Humor

- Jokes sprinkled throughout:
  - Safari and Internet Explorer called “god-awful.”
  - Touching mailboxes improperly is a federal crime.
  - “God help you” if you're not a citizen and tamper with USPS.
- Encouraged students to explore crypto deeply if they enjoy the math.
- Urged forming project groups early.

Summary: This session explored core crypto mechanisms—block encryption, symmetric/asymmetric keys, RSA, signatures, hash functions, MACs, and the PKI ecosystem. Through detailed breakdowns, real-world analogies, and humor, the professor reinforced why each component matters for modern secure communications.

The lecture notes now include much more detailed coverage of the professor's discussions, analogies, side comments, and in-class anecdotes. Let me know if you'd like these formatted into slides, a study guide, or if you want another lecture processed the same way!