

Introduction to Blockchain Technology & Applications
Prof. Sandeep Shukla
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture - 02

So welcome to lecture 2 of Blockchain Technology and Applications on NPTEL. So today's topic is more on the cryptography and the basic cryptographic concepts that are being used in the construction of blockchains and protecting data integrity of data in blockchain.

(Refer Slide Time: 00:42)



So the material in this lecture are from various lectures and courses and writings of various colleagues around the world. So I should acknowledge them first and then we will move on to the topics.

(Refer Slide Time: 00:55)

Today is all about essential Cryptography for Blockchain

- crypto basics that are essential for blockchain technology
- Hash functions and their properties
- Public Key Cryptosystems
- Digital Signatures
- Hash Puzzles
- Hash Pointers
- Merkle Data Structures

So today's topics are first of all, the basic of cryptography and this is not a cryptography course. So therefore we will only focus on the very basic essentials that are being used for blockchain technology. And one of the most important part of constructing a blockchain is hash functions. And hash functions are suitable because of certain properties of hash function that determines what function could actually be used as a hash function.

So we will talk about that in detail. We will talk a little bit about public key cryptosystem because in most blockchain, the public key of an individual is used as his address, address of an account. So public key cryptosystem understanding how it works is very important. Another thing that is very important in blockchain is digital signatures because all transactions must be signed digitally by the appropriate person or appropriate account.

And then we will talk a little bit about hash puzzles and hash pointers. And finally, we will talk about Merkle data structures, which is the data structure that is most commonly used in storing large amounts of data in blockchain. So basic concepts of cryptography.

(Refer Slide Time: 02:22)

Cryptography: Basic Terminology

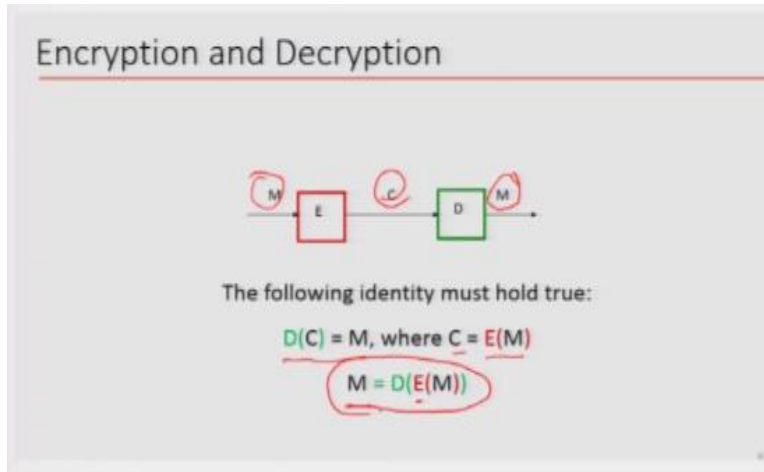
- Plaintext (or cleartext)
 - The message.
- Encryption (encipher)
 - Encoding of message.
- Ciphertext
 - Encrypted message.
- Decryption (decipher)
 - decoding of ciphertext

So some of the terms that you will hear a lot when you talk about cryptography are plaintext, or cleartext. That is the message that you want to encrypt. So it is plaintext, because if you read it, you get it, right. The second thing is encryption or encipher. That is an action that you apply on the cleartext or the plaintext and you encode it in a form so that only people who have a very specific knowledge can actually recover the original plaintext or original message, others will see garbage, right?

So that is the idea of encryption. This is a very old idea, you know we have, we know that Julius Caesar used Caesar cipher. So this is not something that is very modern. But in modern day, the technology, and the algorithms for encryption or enciphering has changed drastically from the old days. Then after the encryption, what we get, we call them ciphertext. It is basically the encrypted message.

And in order to read the message, you have to do a decryption or decipher. And this basically converts the ciphertext to plaintext.

(Refer Slide Time: 03:46)



So now if you look at this as some kind of a data flow, then this is your message or plaintext here. And this plaintext goes into an encryption algorithm or here it is shown as an encryption box. It could be hardware, it could be software and then what comes out is the ciphertext.

The ciphertext is the encrypted version of the original message. And then you want to read it, then you have to pass it through the decryption or deciphering algorithm or the hardware and then comes out the cleartext or the original message. So the following identity must hold. So if D applied to C is M, and C is E applied to M. Then M must be recoverable by a composition of D and E.

That means, if you apply E first on the message, you get the ciphertext and then on the ciphertext, if you apply D, then you should get back the plaintext. So that is the idea, that is the property of all encryption systems.

(Refer Slide Time: 05:01)

Cryptography: Algorithms and Keys

- A method of encryption and decryption is called a **cipher**.
- Generally there are two related functions: one for encryption and other for decryption.
- Some cryptographic methods rely on the secrecy of the algorithms.
 - Such methods are mostly of historical interest these days.
- All modern algorithms use a **key** to control encryption and decryption.
- Encryption key may be different from decryption key.

So in cryptography, as we said that the method of encryption and decryption is called a cipher. So generally there are two related functions, as we showed in the previous slide, that there is one for encryption and one for decryption. And in the old days, the how you encrypted and how you decrypt it were kept secret, right? That is the only secret that stood in the way of somebody an eavesdropper.

Somebody who is trying to capture the message on its way and trying to understand it, it is secret to them, and therefore, they cannot do it, unless somehow they figure out what the algorithm is. And this is what happened in the 1940s when the Germans used this Enigma code to instruct their naval ships and U-boats. And this messages were going encrypted.

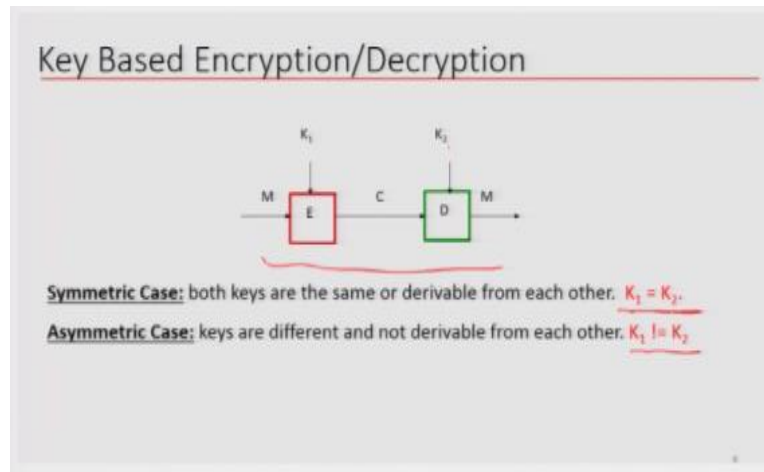
And then the Britishers, which was a team led by Alan Turing, actually constructed a method and device that would reverse the process. And without knowing the algorithm that the Germans were using, they had to actually come up with the algorithm that is being used. And then they had to figure out how to reverse it. And that is how they actually encrypted the Enigma code.

But nowadays, we do not keep the algorithm secret, because keeping an algorithm secret is quite difficult. So we have to use some other method to keep the secrecy and that is where the idea of a key comes in. So a cryptography key is basically the secret that is an ingredient into the encryption process. And the key is also an ingredient in

the decryption process. So and the encryption key and the decryption key, maybe same or maybe different.

And based on whether they are same or they are different we have two types of encryption or decryption method, which are called the symmetric encryption or asymmetric encryption.

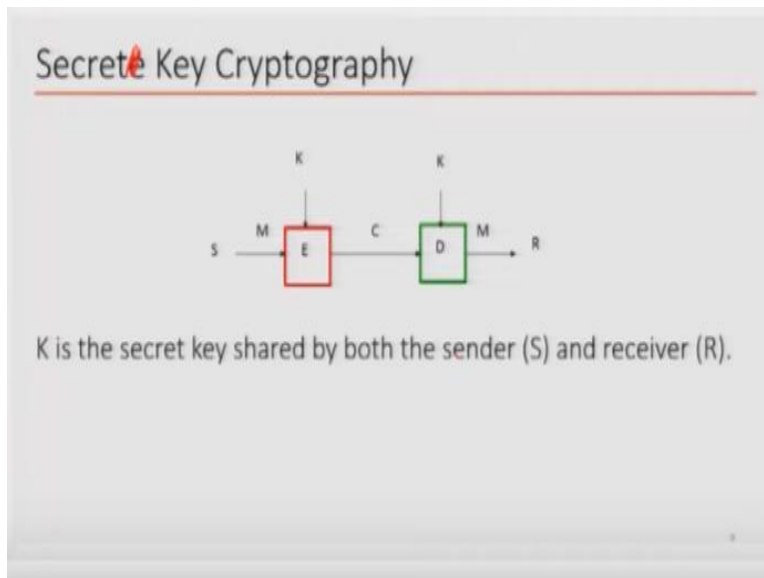
(Refer Slide Time: 07:13)



Now you see that the picture I showed earlier, the second inputs to this boxes were not there. So now we have the encryption in encryption box which takes a message and a key, in this case K_1 . And then outcomes, the ciphertext. And then when you put that ciphertext to the decryption algorithm, then you need another key or maybe the same key depending on whether it is a symmetric case or asymmetric case.

And then you get the M . Now symmetric basically means that the two keys K_1 and K_2 are the same or they are derivable from each other. And in case the two keys are separate and not derivable from each other, then we say that it is an asymmetric algorithm. And we will see that both of them have their own importance in the whole literature of cryptography or the practice of cryptography. And we will talk about that.

(Refer Slide Time: 08:22)



So this is also called a secret key cryptography. So secret key cryptography means that the key has to be known and kept completely secret from outsiders. So only the person encrypting it should know the key and the person who is supposed to decrypt it should know the key. So that means this key is shared between the encrypted and decrypted. And this sharing is a big problem because how do you let the decrypted know the secret key?

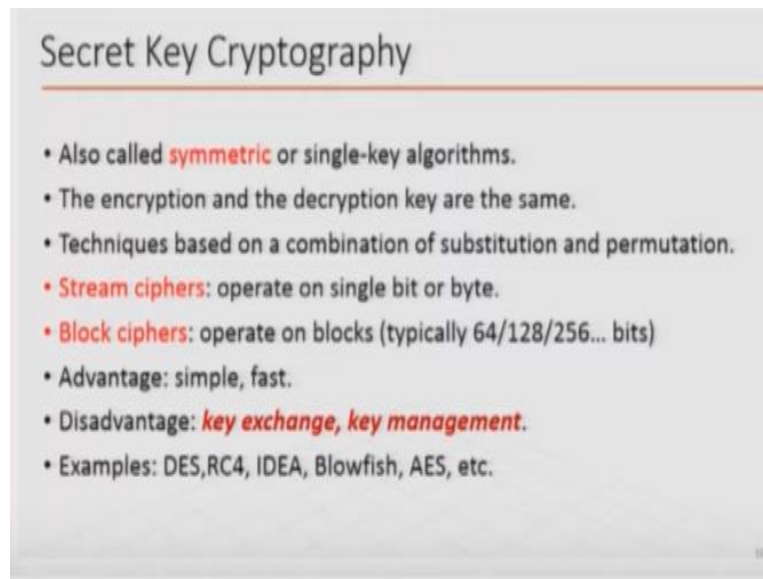
So one possibility is that you do it out of band which means that you use some other means like a, like a telephonic call or some other kinds of method in which you send the key to the receiver, and then you send the message and then the receiver will use the key. But if you send the key with the message, then anybody who is listening on the communication channel will read the ciphertext, but then also read the key.

And therefore, they will use the key to decipher because algorithms are not secret, only the keys are secret. So therefore, there has to be a way in which the key is shared between the two parties. And that is one of the problems that the symmetric key cryptography suffers from. And since before you share a key, you cannot encrypt the key, right? Because then there is a problem that when you have the key, then you can decipher it.

But if you do not have the key, then there has to be a way to share it. And most of the time, this has to be done out of band. And which means that there has to be some other

way of communicating between the sender and the receiver. So this is one of the problem of symmetric key.

(Refer Slide Time: 10:10)



So this symmetric key is also called single key algorithm because you have only one key shared between the receiver and sender. And obviously, two keys are the same. And most of the time this encryption algorithms are also pretty simple. They are simple, but they are tedious. So you take the information that you want to encrypt.

And with the key using the key, you decide on certain substitutions and the part of the message or permutations of the bits or symbols of the message and you do this for many rounds. And how you do it depends on the key, the individual bits of the key. And therefore, when you want to decrypt it, you have to reverse the permutations and reverse the substitutions in order to get this message back.

And since you have the key, you know what the substitution and permutation sequences were, and by how much the permutations happened, by how much the substitutions happened. So similarly, you will do the opposite in the other side. One good thing about this is this substitution and permutation are pretty easy to do in hardware.

Because you are basically taking bits and pushing them through shufflers and parameters and all that stuff. And you do this many times. They are called rounds. And therefore, the secret key cryptography has the advantage of being very efficient

and fast when you have a hardware support for doing cryptography. So there are chips that will do this for you in a very fast turnaround time.

So now how you encrypt with the help the key is based on whether you are doing it continuously as you see bits of the message and you do something with the bits or you divide the message into blocks, chop the message into multiple smaller messages, and then you apply the key as a whole on each of this chopped portions. So those are the depending on how you do it, whether you want to do it on the stream of bits or on the blocks of bits.

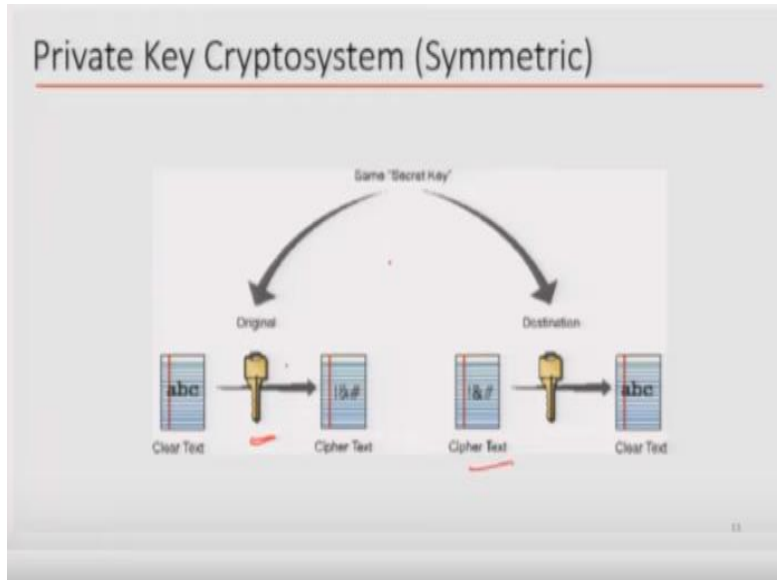
These are called block ciphers and stream ciphers and block ciphers usually divide the message into chunks of like 64 or 128 or 256 or 512 bit blocks. So as I said that the symmetric cryptography is very simple and very fast. So disadvantage, as I said, is that how do you exchange the key between the sender and the receiver. And if you want to keep the key in some kind of a, you know database or something, there is a possibility of the key being stolen or read by somebody.

Because if you keep it in an encrypted form, then you have to first give everybody some kind of a key to decrypt them, right. So you have to keep it in plaintext that is not safe. And therefore, key exchange and key management is a big problem in symmetric key cryptography. And that is where the asymmetric cryptography comes to help because the symmetric cryptography is used to actually do the key exchange.

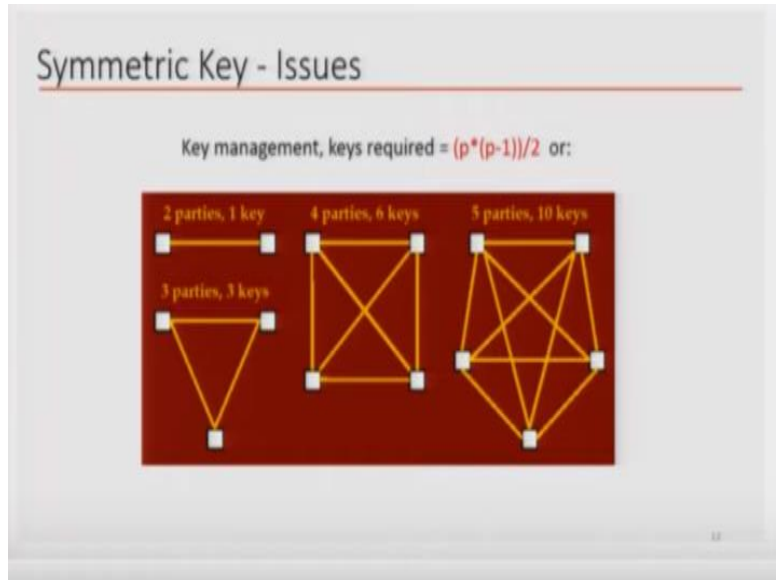
And once you have done the key exchange, you can apply symmetric key cryptography. Now you might ask, why should not I then only use asymmetric cryptography? The problem is asymmetric cryptography is not as efficient and it is time consuming. Therefore, you only use it very sparingly. And in most cases, it is used for key exchange and after the key exchange has happened you can then apply symmetric cryptography on very large amount of data, because it is fast.

So examples of symmetric key cryptography are DES, like Data Encryption Standard RC4, IDEA and Blowfish, and most common nowadays is the AES, right. So AES is the current most common standard and there are variants of AES, which are also used.

(Refer Slide Time: 14:31)



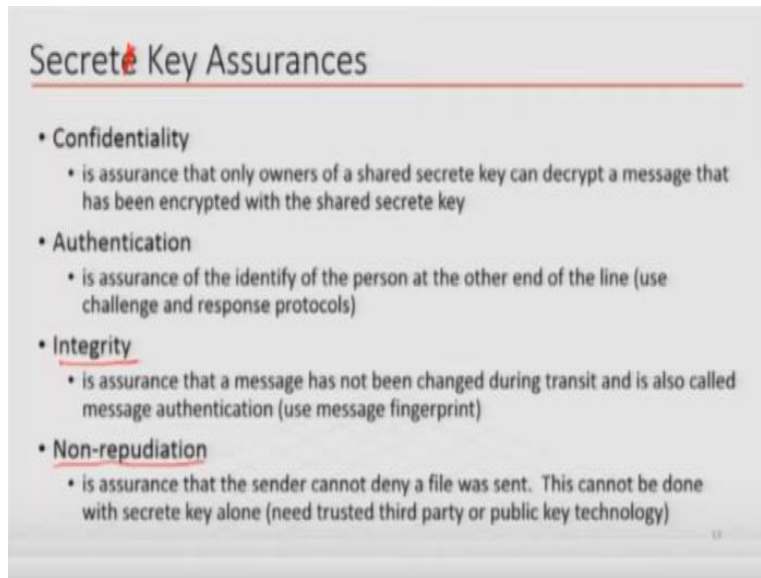
So pictorially if you look at this, the private key cryptosystem has the cleartext to ciphertext with a key and then the same key can be used for, it is the same key here, and here from ciphertext to the cleartext. The only issue is how do you share the key.
(Refer Slide Time: 14:54)



The other problem is that if you have 2 parties, then you have 1 key but if you have 3 parties, then you need, you know 3 combination 2, which is 3 keys. If you have 4 parties 4 combination 2 six keys, and if you have 5 parties, then you have, you know 5 combination 2, which is 10 keys. So this way, the number of keys between parties will also increase very quickly in quadratic speed.

And that is another problem that you know you have to maintain so many keys for every pair of communicators. So that is another issue with the symmetric key.

(Refer Slide Time: 15:33)



Secret Key Assurances

- **Confidentiality**
 - is assurance that only owners of a shared secret key can decrypt a message that has been encrypted with the shared secret key
- **Authentication**
 - is assurance of the identify of the person at the other end of the line (use challenge and response protocols)
- **Integrity**
 - is assurance that a message has not been changed during transit and is also called message authentication (use message fingerprint)
- **Non-repudiation**
 - is assurance that the sender cannot deny a file was sent. This cannot be done with secret key alone (need trusted third party or public key technology)

So let us see what are the properties of secret key or symmetric key algorithms. First of all, it assures confidentiality. So which means that if you have already exchanged the key, and nobody else has the key, then when you send the ciphertext on the way, if somebody reads your ciphertext by tapping on to your network, they would not be able to figure out what is being sent because unless they have the key they cannot decipher.

So confidentiality is one of the primary reason why we do cryptography. So cryptography basically, the symmetric key cryptography basically achieves that privacy or confidentiality of information, provided the key is kept secret. Also if the key is kept secret, then authentication. That is, if you receive a message from somebody, you want to know that it is that person who I suppose to send you the message is indeed sending you the message.

And somebody is not pretending to be him or her and sending you the message. Now if you have done the key exchange beforehand with the authentic person, then when you receive the message and you apply your key for decryption and you can decrypt, then that means the anybody who has sent this message must have the key.

Now assuming that the key has not been stolen from the sender who you actually exchange the key with, then the message has to be coming from an authentic source. So authenticity is also proven, provided the key is kept really secret between the two

parties. Message integrity, integrity means that the message or the data has not been tampered with, nobody has changed the data while the data was in transit.

So that is also assured because if somebody replace the message, and then he has to also encrypt the message with the key. Otherwise, you would not be able to decipher it. When you try to decipher it with a key and it was actually encrypted with a different key then you will get back garbage. So therefore if the if you can decrypt the message, then it is actually showing that the message could not have been tampered with.

So message integrity is also proven if you can decipher the message. And non-repudiation. Non repudiation means that the sender cannot later say, oh no, I did not send this message. Because if the sender has used the key, and he is the one who knows the key and nobody else, then he has to have sent it. Otherwise, there are two possibilities. One, the key is stolen or he is lying.

Assuming the key is not stolen, then it is a non-repudiable. That is that the sender cannot admit to have sent the message, he cannot deny. So now let us look at non repudiation a little more closely by playing out a scenario. So let us say and in cryptography literature, you will always see these names Alice, Bob, Eve. So usually Alice and Bob are the sender receivers, and Eve is an eavesdropper, right.

So who actually tries to capture the message while the message is on transit and tries to decipher. So he is Eve is usually the attacker, or threat.

(Refer Slide Time: 19:20)

Example: non-repudiation

- Scenario 1:
 - Alice sends a stock buy request to Bob
 - Bob does not buy and claims that he never received the request
- Scenario 2:
 - Alice sends a stock buy request to Bob
 - Bob sends back an acknowledge message
 - Again, Bob does not buy and claims that he never received it
 - Alice presents the ack message as proof
- Can she prove that the ack message was created by him?

So in this case, suppose Alice sends a stock buy request to Bob, and Bob does not buy and claims to have never received the message. So at the end, Alice would suffer some loss because the stock might have gone up in price. And then Alice says to Bob that I was supposed to have bought this when the price was low, and I sent you message but you did not buy it. So I have suffered loss. Bob can say that no, I never got the message so I never bought that stock.

So he can deny to have received the message. In scenario 2, suppose Alice sends a stock buy request to Bob. And Bob sends back an acknowledgement that yeah, I am going to do it, I received your request. And then again Bob fails to buy it. And then later when Alice comes and says I have suffered loss because of you. Bob can say that no, I did not get the message.

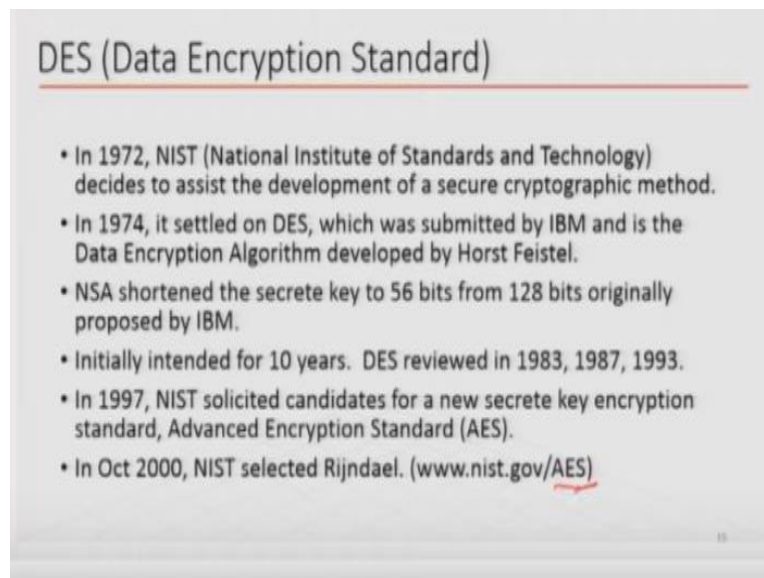
Then Alice shows him that look, you sent me an acknowledgement message. And therefore, you must have received the message. Now in case they are not using encryption, or any kind of method of non-repudiable communication, then Bob can say that well, it looks like a message from me. It has on its header that it is from my email address, but somebody fabricated it.

And then without the use of cryptography for Alice, there is no way that she can prove that the acknowledgement message was indeed from Bob. Maybe somebody, like Eve might have captured Alice's request in the middle of the transit, and then

block the message to receive go to Bob. And she faked a message from Bob to Alice that okay fine, I am going to buy it, but she does not do it. So Bob is not the culprit.

So in this case, Bob has repudiability. But in case of, in case the whole thing was done cryptographically, then Bob could not have done repudiation unless he claim that my key was stolen, which is another issue outside the scope. So in a court of law, under the circumstances we described, Alice cannot prove that the acknowledgement message was indeed from Bob. So that is about non repudiation.

(Refer Slide Time: 21:53)



DES (Data Encryption Standard)

- In 1972, NIST (National Institute of Standards and Technology) decides to assist the development of a secure cryptographic method.
- In 1974, it settled on DES, which was submitted by IBM and is the Data Encryption Algorithm developed by Horst Feistel.
- NSA shortened the secrete key to 56 bits from 128 bits originally proposed by IBM.
- Initially intended for 10 years. DES reviewed in 1983, 1987, 1993.
- In 1997, NIST solicited candidates for a new secrete key encryption standard, Advanced Encryption Standard (AES).
- In Oct 2000, NIST selected Rijndael. (www.nist.gov/AES)

So now let us look at what are the different symmetric key algorithms how they came about. So the National Institute of Standards and Technology in the US in 1972, decided that cryptography is becoming important. More people are communicating through various networkings. So let us define cryptography. So what NIST does is that they opens challenge for anybody to respond to.

So IBM sent one of the entries to the challenge, and it was called the DES and turns out that was accepted by the community as the NIST as the standard. And although IBM initially said in their proposal that it should be a 128 bit key, but NSA, the National Security Agency of the US, they decided that 56 bits is enough. So a 56 bit key will protect the information. And they said we will do this for 10 years, then we will revise the standard.

It was reviewed several times until 1993. And then they realized that we need a new key, secret key algorithm. And that time again they solicited response from people and eventually in 2000, they selected Rijndael, which is actually was originated in Belgium. And that is the standard called AES, Advanced Encryption Standard. And that is what is in use mostly these days.

(Refer Slide Time: 23:35)

The slide is titled "Cycling through DES keys" and contains a bulleted list of historical events related to DES. Handwritten red ink notes are present: "10⁶/s" and "10⁶ [IAS]" in the top right corner, and a red underline under "2**57" in the "Double and Triple DES" section.

- In 1977, a 56-bit key was considered good enough.
 - Takes 1,000 years to try all keys with 56 1's and 0's at one million keys per second
- In Jan 1997, RSA Data Security Inc. issued "DES challenge"
 - DES cracked in 96 days
 - In Feb 1998, distributed.net cracked DES in 41 days
 - In July 1998, the Electronic Frontier Foundation (EFF) and distributed.net cracked in 56 hours using a \$250K machine
 - In Jan 1999, the team did in less than 24 hours
- Double and Triple DES
 - Double DES only gives 2**57 = 2 x 2**56, instead of 2**112, due to meet-in-the-middle attack.
 - Triple DES recommended, but managing three keys more difficult

Now what is the reason why DES, you know failed to be the standard for too long. Remember that 56 bit keys means that there are 2 to the 56 possible keys. Now if I am an attacker, and I do not know the key, so I will try all possible combinations 2 to the like 000 like 56 zeroes to 56 ones and any other combinations in the middle, so there are 2 to the 56 such possible combinations. And I will try those on the message.

And if one of them unlocks the message, then I would know that is the key. However, with 1970s computing power and even if they try out 1 million keys per second, even then, which is actually in those times was impossible, 1 million is 10 to the 6 so per second would mean that each key can be tried in 10 to the -6 that is 1 microsecond, which was not even possible in those days.

But even if it was possible, then they calculated that it will take somebody thousand years in those computers with the you know assuming that the computer is pretty fast and it can do one attempt in one microsecond, it would take 1000 years. So then RSA the company, another cyber security and encryption company, they issued a DES challenge in 97.

And very quickly, some of the people who took the challenge they cracked DES in 96 days as in 97 the computing was lot more fast and lot of more memory was available plus lot of parallel computing was available. In 98, another player in the challenge, they reduced it to 41 days. And then in 98, the Electronic Frontier Foundation and the distributed.net, they reduced it to 56 hours.

And in 99, they did it in less than 24 hours. So at that point, they decided that this is not safe, because attackers could use very you know especially if the attacker is from a big nation, the government is behind it, then they can use it to crack the keys very fast and therefore, they decided that this is not safe. So one possibility is you double the DES right? So you say that first you apply DES once, and then use another key to do the DES on the encrypted one again.

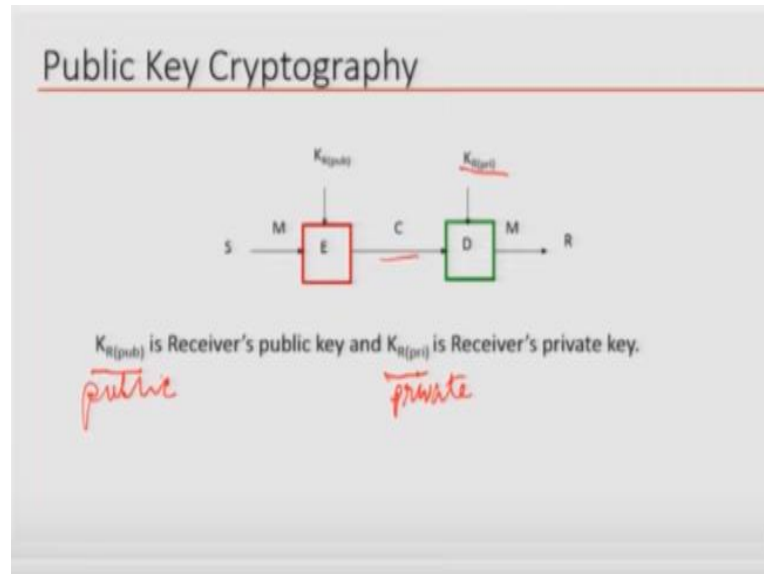
So if somebody has to unencrypt it, then they have to do the opposite that is they have to first find a key for the second encryption and then apply it and then they have to do another time with the second key and the first key and then they will be able to decipher it. So that would mean that they have to try 56 keys first and 56 keys second. So that is 56 times plus 56 that is 112 bit strength.

But it turns out that there is a meet in the middle attack by which you try from both sides. And therefore, it turns out the double DES is equivalent to having a strength of 57 bits rather than 112 bits. So the NIST recommended that you have to use triple DES which is safer. Now triple DES means that you have to have 3 keys for each pair of communicators, right. So three keys for each pair of communicators would basically make this key management issue even more problematic than before.

So therefore, public key cryptography is something that is almost essential in actually doing cryptography today. In practice, for example, when you communicate over the web, and you do ecommerce on the web, you use HTTPS, which uses SSL, secure socket layer and there the key exchange happens through public key cryptography. Once the key exchange happens it is called a session key.

And then that session key is used as a symmetric key between the two parties. And public key cryptography in the beginning, allows them to establish that secret key between the two parties.

(Refer Slide Time: 28:19)



So what is a public key cryptography as we said before? In public key cryptography, there are two keys. One is kept secret. The other one is kept public or known to everybody. And this pair of keys here we are calling it $K_R(pub)$ and $K_R(pri)$. This one is private. And this is the public key. So what you do is suppose you want to send me a message and I tell you my public key.

I can tell anybody my public key, and that time, what you do is you encrypt with that public key, transcend me the ciphertext. Since I have the private key and nobody else has a private key corresponding to this public key, only I can decipher the message. And I do not tell this key to anybody, I keep it very closely secret. So therefore, the key, I have two key or I can say it is a two part a key system.

One part, the public part, I can announce to the world that if you want to send message to me, then send me encrypted with this key. And the way it is designed that only person who can actually unencrypt the message or decipher the message is one possessing the private key. Therefore, it has to be only me who can read it. So that is the idea of public key cryptography, right.

(Refer Slide Time: 29:49)



So now the question is, how is it used to establish a shared secret, because as I said that public key cryptography is usually based on a very complex mathematical problem compared to secret key cryptography or symmetric key cryptography where bit shuffling and bit permutation, this kind of stuff is used. So it is very fast to implement.

Whereas in this case, normally what we do is we try to solve a very complex mathematical problem, rather than a bit shuffling kind of thing, simple things. Therefore, it is a, it is not very fast. So I have to eventually use a symmetric key cryptography. But to establish the secret key between the two parties, I need to use public key cryptography. So that is what public key cryptography is normally used for.

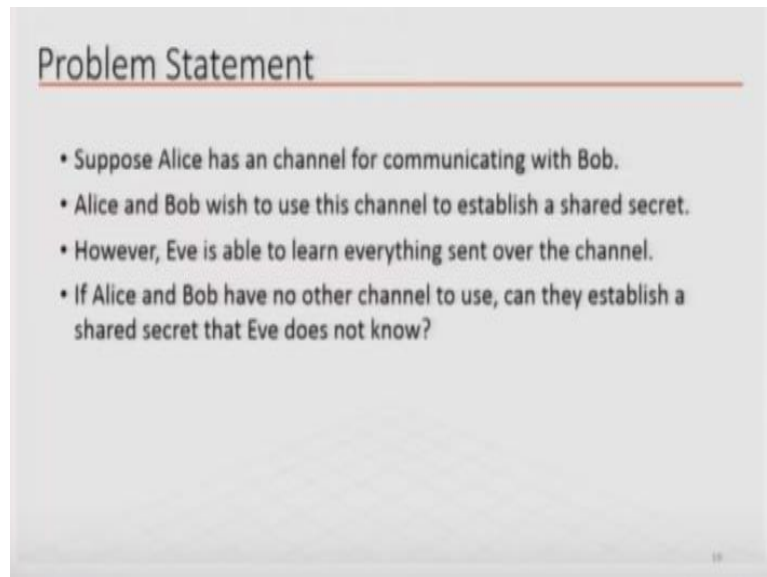
So I am sending a message over the Internet to my friend. Here in the internet, there are lurking behind many of the computers, there are people who can capture my message. And then if they know the key, then they will decipher it. So I have to somehow make the system such that the symmetric key that I am going to use to communicate very large amount of messaging or file uploads and all that I will not let anybody else know the key. So public key cryptography helps me by the following.

If you use public key cryptography, then you never send the secret key over the network. So what you do is you establish the secret key by using public key cryptography without having to ever sent the key on the network. So everybody

creates the key on their own computer. And because of the mathematical process that is followed, the key this party establishes and the key the other party establishes becomes the same.

So that becomes a shared secret without having gone through the network.

(Refer Slide Time: 31:48)

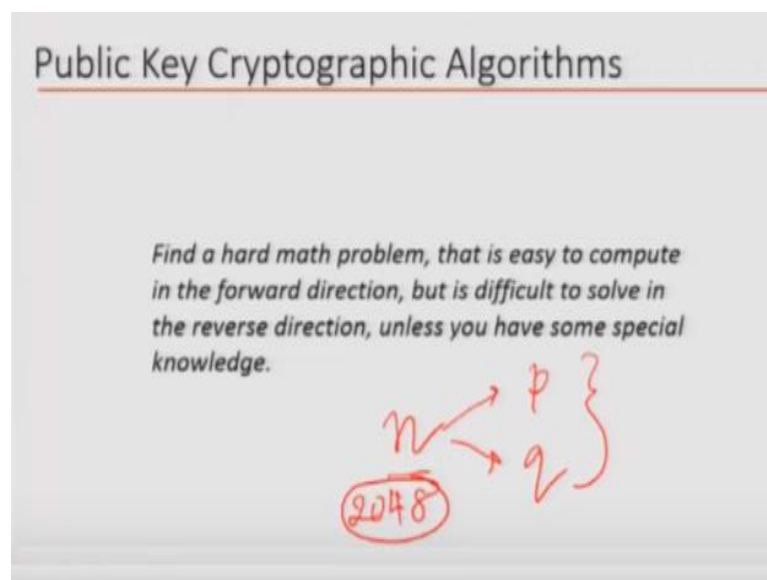


Problem Statement

- Suppose Alice has a channel for communicating with Bob.
- Alice and Bob wish to use this channel to establish a shared secret.
- However, Eve is able to learn everything sent over the channel.
- If Alice and Bob have no other channel to use, can they establish a shared secret that Eve does not know?

So the problem statement here would be that Alice has a channel for communicating with Bob and they want to use this channel to establish a shared secret. However, Eve is able to learn anything that is sent over the channel. So if Alice and Bob has no other channel to use, how can they establish a shared secret?

(Refer Slide Time: 32:07)



Public Key Cryptographic Algorithms

Find a hard math problem, that is easy to compute in the forward direction, but is difficult to solve in the reverse direction, unless you have some special knowledge.

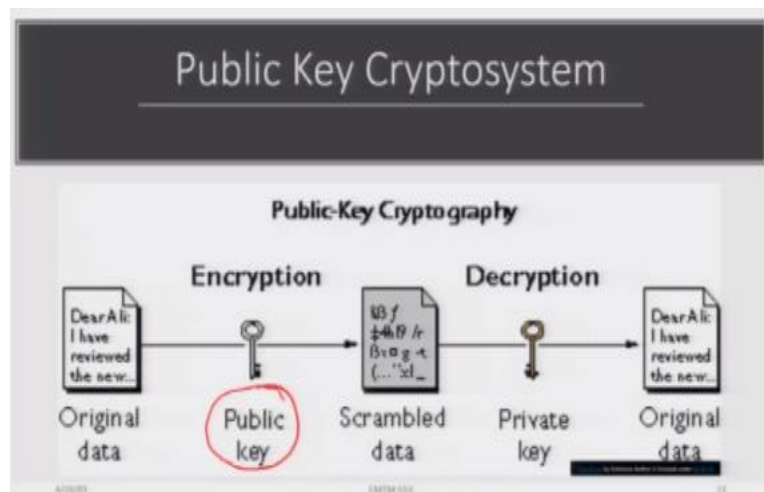
Handwritten diagram: A circled '2048' has an arrow pointing to 'n'. 'n' has two arrows pointing to 'p' and 'q', which are grouped by a bracket on the right.

So public key cryptography is as I said is based on a hard mathematical problem. So for example, RSA is dependent on the fact that if I give you a very large integer n and the n can be factored into two prime numbers, p and q , finding those p and q to factor n is a very hard problem. So it is called the integer factorization problem. So figuring that this problem is not very fast and if n is a very large number like 2048 bits.

This is a 2048 bit number. So it is a very large number, let us say. And then you have to know what are the two prime numbers that produced this number by being multiplied, it is a hard problem. And this kind of hard problems are used for public key cryptography. So these are also called one way functions because multiplying p and q to produce n is easy.

And then if you do not tell people p and q , and just give the n for people to know the p and q is hard. So reversing the process is difficult.

(Refer Slide Time: 33:23)



So public key cryptography as you know that the person who wants to send you the message, you basically send it encrypted with the public key. And then, when you decipher the message, you do it with the private key.

(Refer Slide Time: 33:38)

General Strategy

- A public key is used to encrypt a message that can be decrypted only by the matching private key.
- Bob can use Alice's public key to encrypt messages. Only Alice can decrypt the message.
- Similarly, Alice can also use Bob's public key.
- Alice and Bob exchange information, each keeping a secret to themselves.
- The secrets that they keep allow them to compute a shared secret.
- Since Eve lacks either of these secrets she is unable to compute the shared secret.

So this is what we already talked about. The public key is used to encrypt a message that can be decrypted by matching private key. So Bob can use Alice's public key to encrypt messages sent to Alice. Alice is the only one who has the private key, so she can decipher the message. And Alice can also use Bob's public key to send message back to Bob. And Alice and Bob can exchange information, each keeping a secret to themselves.

And the secrets they keep allow them to compute a shared secret. And so since Eve does not have any of the secrets, she is unable to compute the shared secret. That is the general strategy.

(Refer Slide Time: 34:23)

Asymmetric Algorithms

- Also called public-key algorithms.
- Encryption key is different from decryption key.
- Furthermore, one cannot be calculated from other.
- Encryption key is often called the **public key** and decryption key is often called the **private key**.
- Advantages: better key management.
- Disadvantages: slower, more complex.
- Both techniques are complementary.
- Examples: RSA, Diffie-Hellman, El Gamal, etc.

Handwritten notes:

Public
Digital Signature
private key

RSA

$$d = e^{-1} \pmod{\phi(n)}$$
$$de = 1 \pmod{\phi(n)}$$
$$c = m^e \pmod{n}$$
$$m = c^d \pmod{n}$$

$\{e, n\}$ $\{d, n\}$

So this public key algorithms there are many. So for example, RSA Rivest-Shamir-Adleman algorithm is one of the most popular algorithm for public key or asymmetric key cryptography. There is also Diffie-Hellman, there is El Gamal. So there are multiple different algorithms. And encryption key is different from decryption key, one cannot be calculated from each other.

And encryption key is called the public key and decryption key is called the private key and you can do better key management. And disadvantage is that is slower and more complex. And both techniques are complimentary, which means that asymmetric is used to establish a shared secret, which is then used as secret key for symmetric algorithm. So let us quickly tell you how the RSA works.

So in RSA what you do is, so one party decides two very large prime numbers, p and q . Prime numbers, you know that, if I guess an integer to check whether it is a prime or not, is a polynomial time, and due to an algorithm that was invented in IIT Kanpur by Professor Manindra Agarwal, Nitin Saxena and Neeraj Katyal. Now you take two large primes p and q , and then you multiply them and you get a number n .

Then you compute another number, $p - 1$ times $q - 1$ and you call it ϕn and then you forget this numbers. So you do not tell anybody the numbers, throw away these numbers and you have the n and ϕn , right? Then you choose a number e , which is mutually prime with ϕn . Mutually prime means if you take the GCD of e and ϕn , that should become 1, so they have no common factor.

Then you compute d and the d is a multiplicative inverse of e . In other words, d times e is equal to 1 modulo ϕn . If you do not understand modular arithmetic, that is fine. We are just giving you this information, it is not necessary for this course to actually know how the algorithm works. But just to give you an idea, so then the public key is e, n and it is published. And the private key is d, n , which is kept secret.

And then when you want to get a message through, what you do is you take the message and raise it to the power e and then take modulo n . Now you say how do I do this because message m is basically anything inside the computer, is a string of bits.

So you can actually think of the message as a large number or you can chop the message into multiple chunks, which are each smaller than n .

Because otherwise you would not be able to do modular arithmetic, and then you raise each of them by e . So that becomes your ciphertext. So when the message goes on the other side, the one to the one who possesses the private key that is this, then he does c to the d , which becomes $m \cdot e^d \pmod{n}$ being $1 \pmod{\phi(n)}$, this becomes basically m . So I keep d and n .

So nobody else knows d , and it is difficult to know d unless you know $\phi(n)$. But how do you know $\phi(n)$ because $\phi(n)$ is $(p-1)$ times $(q-1)$. To know $\phi(n)$ you have to factor n first into p and q , then you can get $(p-1)$ and $(q-1)$. So that is the whole idea of RSA asymmetric cryptography. So this is how you establish a shared secret, and then you go on doing things.

But what we will see is that there is another use of this, which is suppose I want to prove authenticity. See earlier I said that in symmetric key cryptography if I have shared the key with you, and if I get a message that is encrypted by that key, I know that it is authentic because you are the only one who knows the key.

In public key that is not possible if somebody sends me a message encrypted with my public key, how do I know it is not the one who is claiming it to be and somebody else because public key is known to everybody. So anybody actually can send me encrypted with my public key, right? So therefore, I do not have authenticity, like that.

So what I do to prove my authenticity is I use my private key to encrypt something and send you. And you know my public key because everybody knows my public key. So you use the public key on that message, and you decipher the message. Now you know it is me because I only have the private key. If it was not encrypted with my private key, then my public key would not have unlocked it.

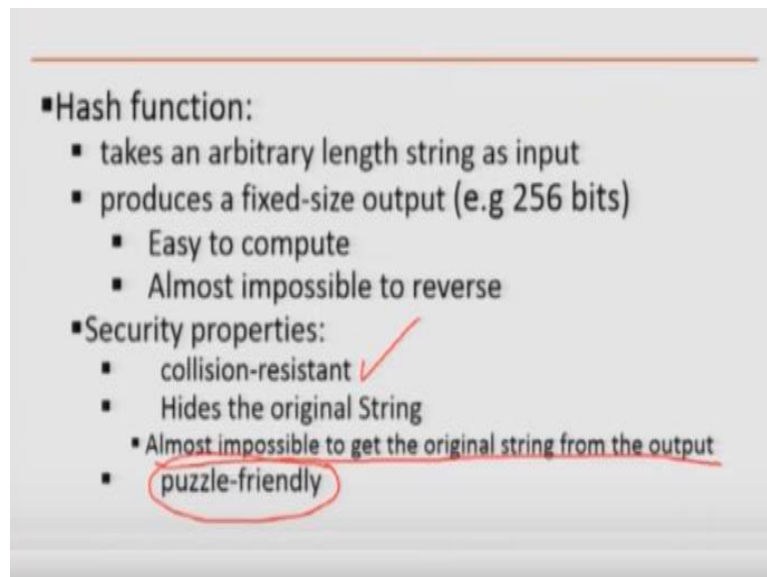
So this is how digital signatures work. And we will talk a little more about that later. So we will see a number of things that are very useful in blockchain from public key

cryptography. One is the public key. Public key for each person is very important. And the digital signature is very important. And because public key is there so private key is also very important because digital signature is done with private key.

Also anything that is encrypted with public key will have to be deciphered with private key. So these concepts, as we will see will come in blockchain all the time. And they basically constitute a key backbone in the technology of blockchain. So examples as I said are RSA, Diffie-Hellman, El Gamal, etc. So now we go into the idea of cryptographic hash functions.

So far we were only talking about encryption and decryption and encryption and decryption are used for confidentiality, for authenticity, etc. For integrity, some of the times we have to use a different technique. And we will see how hash functions are used.

(Refer Slide Time: 41:42)



So hash function is a function that takes any length string and creates a fixed length string. So if you give me a million bit string, I can apply a hash function and let us say the hash function is a 256 bit hash function, then it will give me a 256 bit output. So if you give me even larger, let us say 10 million length string, it will still produce me a 256 bit output. So and the property of hash function should be such that it should not take too long to compute them, because we have to use it time and again.

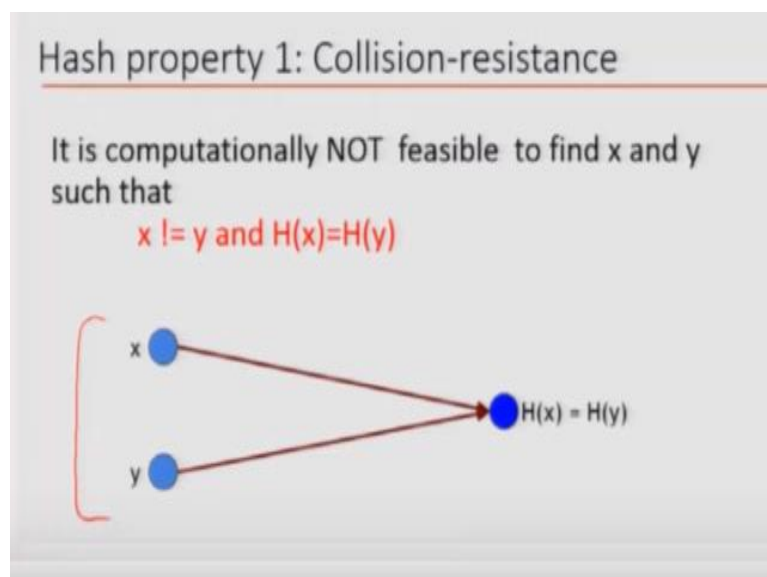
So it has to be easy to compute, but you cannot reverse it, which means that if I give you the output, you should not be able to say which you know string was put into the hash function to produce this output. So it is almost impossible to reverse. Now it has to have some security properties. So any function that makes it impossible to reverse is not necessarily a good hash function. So it has to have certain properties.

So one important property is collision resistance. So it must be and I will explain to you what collision resistance means in a few slides down. But collision resistance is one of the most important property of hash functions. And we will see why. And certainly it hides the original string. So if you are going to say I want to prove I know this string. So I but I do not want to tell you the string.

Then what I can do is I can send you the hash. And then the other party should have a way to check that this is the hash of that string. So even if the other party does not know the string itself, but he might actually somehow have the hash and therefore he can match them. So it hides the original string. And as I said that the hash property, the property of a hash function should be that it should be almost impossible to get the original string.

And then the third property that is very important for our purpose in blockchain especially for cryptocurrency type of blockchains is that it has to be something called puzzle friendly. And we will see what that means very soon.

(Refer Slide Time: 44:09)



So collision resistance, what does it mean? So it means that if you have x and y , and their hash is the same, then we say that x and y collides, means their outputs are same hash. So now the question is, if I get a hash, can I find two distinct x and y , whose hash is that particular string. So I take a hash value. And I ask you that give me two strings, which are different, but they are giving me the same hash output.

And that saying that finding a collision and a good hash function should have the property that it is not feasible to find such x and y , who collide through this hash function. So now you say why is this important? It is important for the following reason. So when you download a open source software, for example, the originators of the software, so when you are downloading over the internet, it is possible that the bits that you are downloading might get corrupted.

So therefore, all of these things come with a hash. So the entire file is hashed and it is sent with the file to you. And your computer then checks by applying the hash function on the entire binary and check the hash matched, right. If the hash match, then it knows that the file was never modified or corrupted on the way to download to on your machine. So therefore, the hash here works as some kind of a signature of the file.

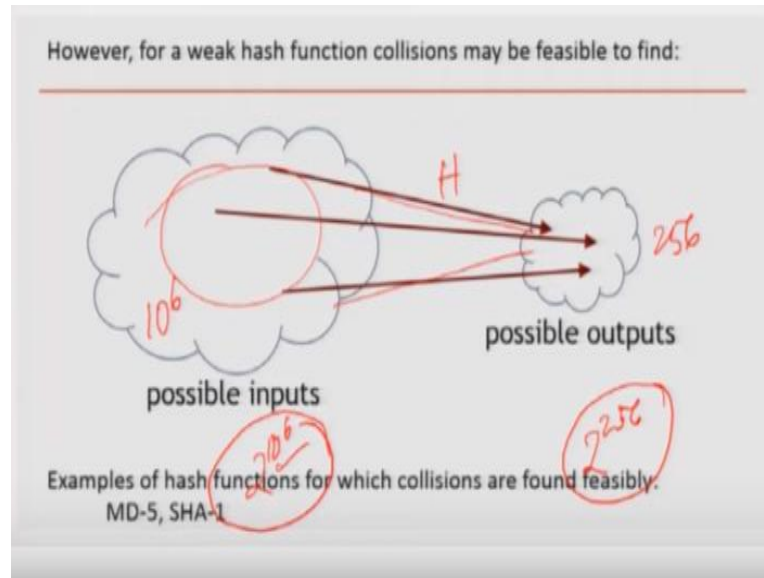
So if hash is not corrupt and if the binary is not corrupt, then binary's hash should match the hash that came with the file. Now suppose a cyber attacker wants you to download a malicious file and somehow makes you believe that this is the real file. So then you download that file. And then if you know the hash of the original file, then your computer will say that okay, so I the file that I have downloaded, I hashed it.

And its hash does not match because I have got the hash of the original file elsewhere. And it does not match. So this is how we use hash to detect the integrity of a file. Now suppose I am an attacker and I know how to find collision in the hash function that is being used. Then what I can do is I can suitably modify the file, put my malware content in the file and suitably modify.

So that the hash of the file, even though the file has changed, the hash of the file remains same as the original file, which means I have found a collision. And if I find

a collision, then hash is no longer a good method to check the integrity of the file. So therefore, the hash function should be such that it should not be easy to find collision and this is called collision resistance. Now the collision is not an impossibility. We said it is not feasible.

(Refer Slide Time: 47:38)



And the difference between feasibility and impossibility is that feasibility means that it is not easy to compute in a fast. Impossibility would have meant that it does not even exist, such collisions does not exist. And that cannot be the case. Because, as you can understand, I said that I have arbitrary length strings here, which are then mapped to a fixed length string. Let us say this is 10 to the 6 length strings, and these are 256 bit strings.

Then I have all the possibilities here is 2 to the 10 to the 6 and here I have 2 to the 256, which is much smaller than 10 to the 6. So this number is much bigger than this number. So if the hash function is mapping all the strings here two strings here then obviously there will be collisions. But the point is, so it is not like if the collisions do not exist. So we do not say a hash function is collision free.

We say a hash function is collision resistant, which means that people should not be able to quickly find a collision. So that is collision resistance. Example hash functions on which collisions were found are MD-5, SHA-1, SHA is the secure hash function, version 1. This have been now abundant as suitable hash functions because people found that collisions for this functions can be found.

So until it was found that they have collisions easily in a feasible, then they were used quite a bit. But as we know now that they have collisions, and it is not difficult to compute the collision, so therefore, they are not being used that commonly.

(Refer Slide Time: 49:21)

Brute-forcing collision

try 2^{130} randomly chosen inputs
99.8% chance that two of them will collide

This works no matter what H is ...
... but it takes too long to matter

Even if each input checking takes 100 ms, we are talking about $2^{130} \times 0.1$ seconds which is $\frac{1}{5} \times 2^{129}$ seconds = 4.3×10^{30} years

So collision, of course can be found by brute forcing the collision. So if you have a 130 bit string mapped to a let us say whatever number, let us say a 64 bit hash, then you have to try 2 to the 130 possible inputs and apply hash to each of them. And you will find collisions because collisions do exist. So you can compute that you know 99.8% with chance you can compute these probabilities with simple combinatorics that collisions will be found.

The point is that suppose for each input and making hash of them and so on takes let us say hundred millisecond, then we are talking about 2 to the 130 times 0.1 second, which I computed approximately, will become 4 times 10 to the 30 years. That is a huge number of years it will take to try this, right. So therefore, it should be not possible to brute force the collision.

Unless you find some weakness in the hash functions construction itself, it should not be possible to find the collision by just brute force method.

(Refer Slide Time: 50:37)

Are there any hash functions for which efficient collision finding algorithms exist?

- For some H
- But for others none known yet (SHA-256, SHA3 etc)

No H has been proven collision-free. *resistant*

Many H has been proven to have collisions.

Examples: SHA-1

<https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html>

So there are hash functions for which efficient collision finding algorithms exist. For example, the SHA-1 and here is a link that you can go to see what how the researchers found that SHA-1 collision is 256 SHA-256, SHA3 etc. are safe. That does not mean they are collision free or collision resistant. But it only says that so far we have not found a weakness in this functions that will make the collision easy to find. Okay, so now we will look at some applications of hash function.

(Refer Slide Time: 51:19)

Application: Hash as message digest

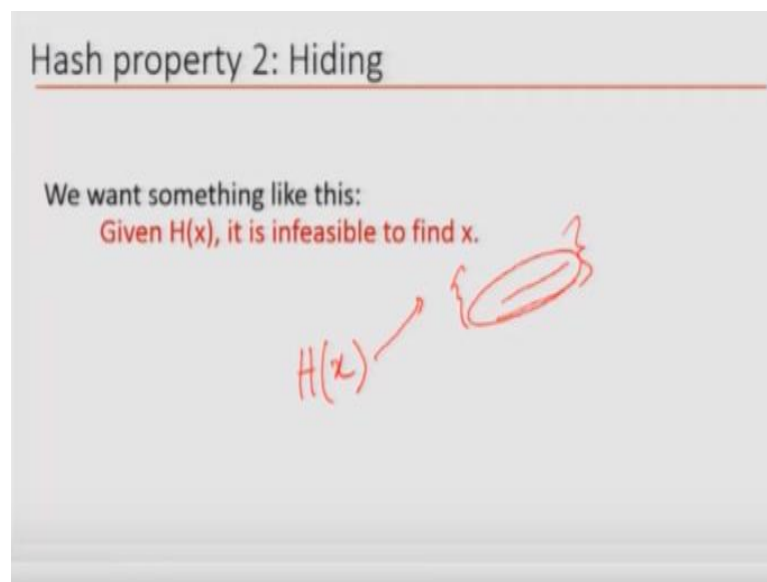
- If we know $H(x) = H(y)$,
 - it's safe to assume that $x = y$.
- To recognize a file that we saw before,
 - just remember its hash.
- Useful because the hash is small.

So hash can as I already alluded to, hash can be used as a digest of some very large message a file. And the reason is that since the collisions are not common, so we will assume that if hashes match, then they are they must be the same. So when I get a file downloaded, if the hash matches then I know the file is fine. And this is based on the

assumption that nobody has found a collision and replaced the file with a collision file.

So it is often used for recognizing a file that we saw before. So when we have antivirus, the antivirus tries to figure out whether this virus has been seen before. So if this virus has remained unchanged, then by computing the hash, they figure out if that virus has been already in their database. And this is very useful because hash is usually small, like 256 bits or 512 bits, so it is easy to use.

(Refer Slide Time: 52:16)



Second property is of course, hiding. That is given the hash, finding out the original message or original file is infeasible. Again, it is not impossible, but it is infeasible. If there is a hash function for which it is not infeasible, it is easy to find, then it is a bad hash function. For example, suppose you have a hash functions that maps things to only finite number of possibilities, right.

So in that case, you can create a table for every x from your domain from which you are taking inputs, and compute this possibilities and have a table. So when you see the hash, you have to just reverse lookup the table and say these are the possible inputs to which it is mapping. So unless this number is small this is not possible. If this number is very large like 2 to the 256 or 2 to the 512 then it is difficult and then those hash functions are workable.

(Refer Slide Time: 53:21)

Hash property 2: Hiding

- Hiding property:

- If r is chosen from a probability distribution that has *high min-entropy*, then given $H(r \parallel x)$, it is infeasible to find x .

- High min-entropy means

- the distribution is "very spread out", so that no particular value is chosen with more than negligible probability.

To know a bit more on this: <https://crypto.stackexchange.com/questions/66097/why-is-min-entropy-significant-in-cryptography>

So the hiding property is more formally defined as follows. So if you have a random number chosen from a distribution with high mean entropy, then given the concatenation of r with x , it is infeasible to find x . So what is high mean entropy means? If you want to know in a very broad terms, it means that the distribution is pretty uniform.

Because if the distribution is very skewed, then the numbers you generate may not be, certain numbers are more likely and certain numbers are less likely to be picked when you pick the random number. So the distribution should be more or less uniform or spread out so that the r cannot be guessed with high probability. And then if you take this r , pick this r and mix it with x and compute the hash.

Then finding x would be very difficult. And that is what we actually use in what we call hash puzzles. And if you want to know more about this formal property of mean entropy and why they play a significant part in cryptography, you can read various other sources. Here I have given one possible quick source.

(Refer Slide Time: 54:42)

Hash property 3: Puzzle-friendly

Puzzle-friendly:

For every possible output value y ,

if k is chosen from a distribution with high min-entropy,

then it is infeasible to find x such that $H(k | x) = y$.

$$y = H(k | x)$$

So the third property that we want to emphasize is puzzle friendliness. So what is puzzle friendliness? Which means that given an output, right, of a hash function, and given that the part of the input we know and it is a random number, then finding the rest is very difficult or infeasible. And that is the whole idea of a puzzle friendliness.

So if I know the hash, and if I know part of the input, and then I concatenate it with some x and compute the hash, and I get y then given y and given k , finding x should be infeasible. And we will see how this is used very effectively in cryptocurrency mining. Basically, it is a puzzle that given y given k give me x , right. And the only way it should be brute forcing. There should be no other shortcut way by exploiting the weakness of the hash function, or weakness of the distribution of the k .

(Refer Slide Time: 55:56)

Application: Search puzzle

Given a "puzzle ID" id (from high min-entropy distrib.),

and a target set Y :

Try to find a "solution" x such that

$$H(id | x) \in Y.$$

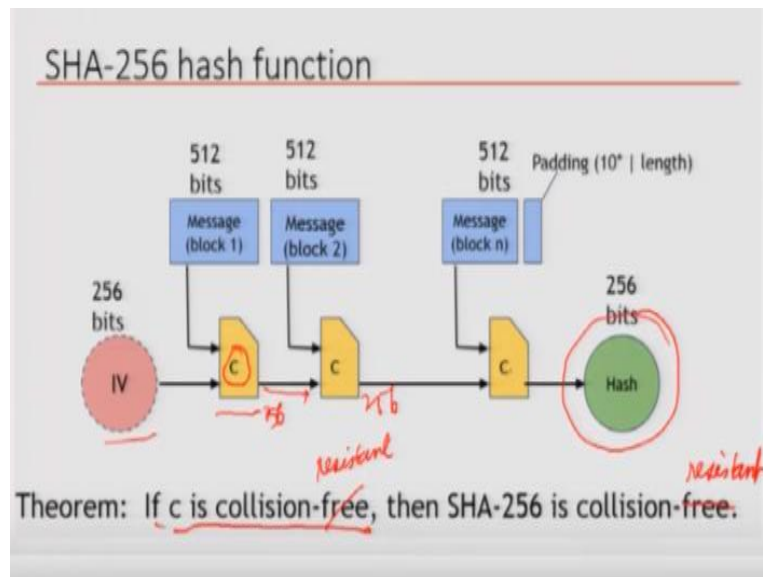
Puzzle-friendly property implies that no solving strategy is much better than trying random values of x (*Brute-force*)

In a puzzle form we can say that given a puzzle ID and a target set y try to find a solution x such that hash of ID followed by x belongs to this y , right? Earlier we were talking about this single y given to you and k is given to you, you have to find x . Now we can also make it a little bit easier by saying that it is not a fixed y , but I will give you a set of y 's.

And you give me an x , such that hash of this ID concatenated with x will give me a number in that set y . And the idea is that if the hash function is puzzle friendly, that means that there is no shortcut to find this x , except brute forcing, except trying all possible x 's. I know if I know the y . And then I try 0000001 and all that stuff. And then at some point, hit a number as hash which will belong to y .

But it will take you a very long time to find out because you are going to do brute force.

(Refer Slide Time: 57:10)



So let us look at one hash function that is actually used in Bitcoin blockchain quite a bit. It is a SHA-256, secure hash function 256. This is a NIST standard. So what you do? Suppose you have a very long message and you want to create the digest. You basically use a kind of a seed on this side, and then you send, so C is a function that basically does a lot of bit shifting, slicing, and permutation and so on.

So you divide the message in chunks of 512 bits. And also at the end, your original message may not be a multiple of 512 bits, so you have to do some padding at the end

to make all the chunks 512 bits. And then you apply the first 512 bits to the first function block with this seed, and then you do the output of that will also be 256 bits. So this is 256, this is 256 and so on. And this, this is 512.

They come out as another 256 bits and you apply this in a chain until all the 512 bit chunks including the one with the padding has finished. And then what you get out is a 256 bit number and that is your hash. And there is a theorem that shows that if each of these function blocks are collision free or collision resistant, then the entire composition of this functions is also collision resistant.

So that is the idea of a hash function SHA-256. What you do not really need to understand SHA-256 in detail, but what you need to remember is that when you have a very large message and you want to apply SHA-256, it basically chops the message into blocks of 512 bits, and use some padding to get the all the chunks to be 512 bits. And then it sends it through this rounds.

And each of this actually are going to be your this functions, which does bit shuffling and all. So it is easy to implement in hardware like the symmetric encryption, and therefore it is very fast to compute. It is actually can be computed very fast and with hardware support, it can be done even faster. So if you are doing in software, then you have to do a lot of bit operations and so on which is not as fast.

But if you do it in hardware, then it is really fast. So you have lot of cryptographic hardware implemented which are which can provide you very fast throughput in computing SHA-256 hash. And we will see that the need for this hardware support creates this enormous hunger for electric power and for the bitcoin mining machines because they use all these things in specialized hardware.

The next thing that we want to talk about is hash pointers and data structures.

(Refer Slide Time: 1:00:28)

Hash Pointers

- hash pointer is:
 - pointer to where some info is stored, and
 - (cryptographic) hash of the info
- if we have a hash pointer, we can
 - ask to get the info back (locate)
 - verify that it hasn't changed (integrity)

So what is a hash pointer? So earlier I said in the last class that a blockchain is actually a linked list. And then I said that unlike linked list that you see in C, C++ programming, where the pointer is actually a memory address for the location of the next block of data. Here the pointer cannot be memory address because it is replicated on many machines.

And all these machines are not only the same types of machines, they are actually of various it can be a phone or it could be a very large memory machine, 64 bit machine, it could be a 32 bit machine. Therefore, you cannot say that I will point to between blocks or between nodes using memory addresses. So what you do is you have to have something called hash pointer. So what is a hash pointer?

So you have a block of data and you want to locate that block of data, right. So you have to somehow point to that and keep that handle on that pointer. So what you do is you hash, compute the hash of the data and then you say that this is my handle on the data. Now assuming that for each block the hash will be different. So that hash value will tell you who are you pointing to, because in order to search that, all you have to do is that you have to hash all the blocks.

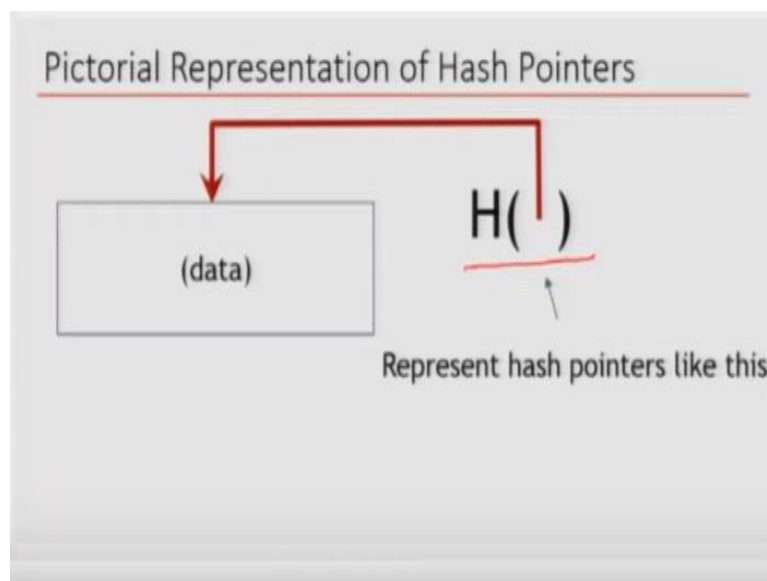
And see whose hash matches the one that you are holding, then you are pointing to that block, right? So you are locating a block by knowing its hash value. So that is why it is called a hash pointer. So it points to some info that is stored. And the way it points is by keeping the hash of the information. So having the hash, you can locate

the block of data, who you are pointing to, by checking whether its hash is matching with what you are holding.

And then you compute the hash. And therefore, as we said that hash of that data if you are holding it, and it matches that means that data has not been tampered with because if it was tampered with, then it will not match the hash. And in fact, if it was tampered with, then it will, you will not find the data. So if you have found the data by matching the hash that means your data was also not tampered with.

So data integrity and location can be found by using hash pointer. So although we call it a pointer, but it is not really like a pointer in the sense of C, C++.

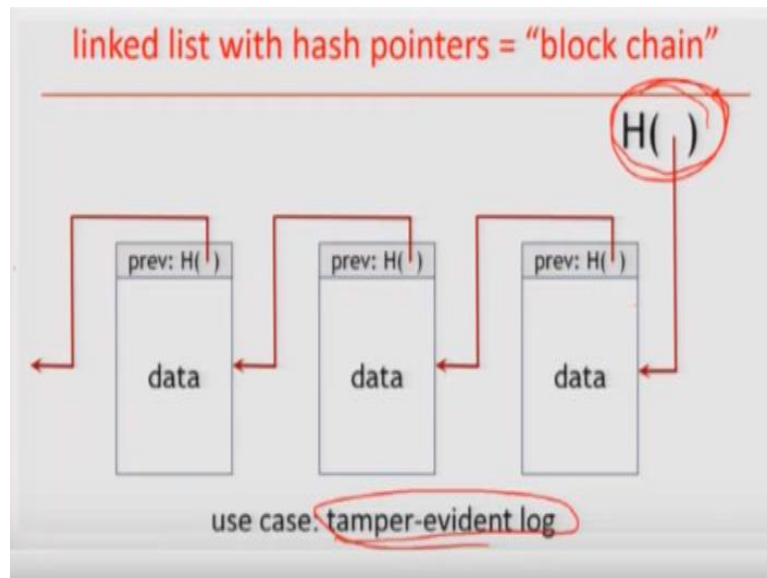
(Refer Slide Time: 1:03:12)



However, if you want to pictorially represent when we show you a blockchain, like last time also we were showing some representation of you know when you created a money with the coin creation function, then we gave it to somebody and we said that there are pointers. So we have to somehow represent pointers. So this hash pointers are usually represented by this H with this parenthesis and then this kind of an arrow.

And this is very useful pictorial representation to visualize the blockchain. So how do you build data structures with hash pointers?

(Refer Slide Time: 1:03:50)



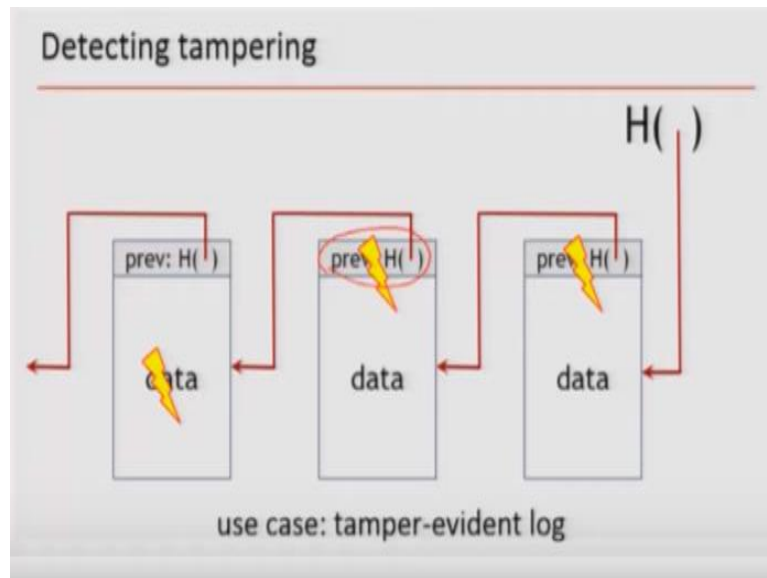
So suppose you want to create a list of blocks and you want that like a linked list, every block should be pointed to by the next block, right? So what you do is you take the data and keep a part of the data space for the hash of the previous block, whoever whatever is your previous block. So usually it is the temporarily previous block which means a block that was created before.

So when you create a new block, you compute the hash of the previous block and keep it inside your space. So then, this block, when it was created, it already knew what was created before it. So it keeps the hash of that here and this way, at the end, there will be a block that is called a Genesis block, which is not shown here. So before that, there will be no pointer. So that pointer will be pointing to nothing or it will be basically zero.

So this way, if you have handle to this, then you have handle to everything because first you have to find this. Then you have to look up the hash pointer to the previous block. You use that to find this and so on and so forth. So all you need to remember is the hash of the last created block.

And this is also tamper evident, which means that if somebody has tampered any part of the structure like whether they tampered here or tampered here or tampered here, it should be easily revealed by anybody who wants to check. How does that work?

(Refer Slide Time: 1:05:29)



Suppose somebody goes and changes some data here, adds or deletes or changes some value. Now this guy, this pointer is actually pointing to is actually the hash of this entire thing. Now if you change this data, then this hash the hash of this entire block has to change. And if it is changing, then it will be easily evident to somebody who is checking the hash of this against this.

And if it finds that this is not matching, then this data has been tampered with. So therefore, the perpetrator, the guy who is attacking the system also has to do this here, change it here. So he will have to go and change it here. As soon as he changes it here, since this is part of this big block, so this hash has to be changed. So then he has to go and go and change the next block's hash also.

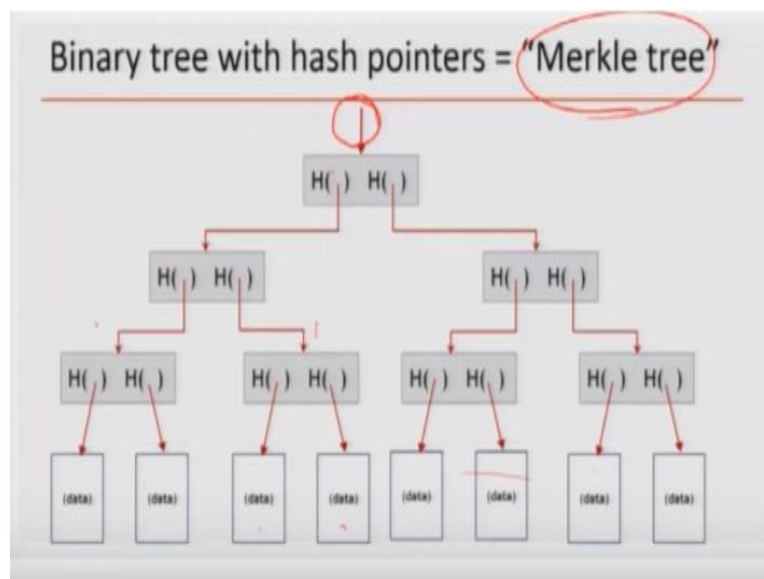
So therefore, deeper towards the Genesis block, you make change more places you have to make these changes. And as we will see later, is that you just cannot willy nilly go and change the hash or change the data. You have to actually solve a computational hash puzzle, which is very expensive to solve in terms of computation in order to change anything in any of these data structures right.

So therefore, if he have to change this not only he has to solve computing puzzle here once he has to computing solves another puzzle here and he has to solve another puzzle here and if this thing has lot more of these, then he has to keep solving those problems. Now to solve even one of those hash puzzles is quite difficult. So solving so many of them is computationally, you know very infeasible.

And that basically it gives you the tamper resistance or tamper evident log property of blockchain. So blockchains are basically blocks of data which are connected to each other consecutively with hash pointers. And the hash pointer gives you two properties. It tells you who is your previous block and also it tells you that the previous lock has not been tampered with and this goes transitively to all the previous blocks, to all the way through the Genesis block.

So to change anything is going to invite too much trouble. So therefore, once the data has been stored and hashed and the hash pointers remembered, you cannot really change it.

(Refer Slide Time: 1:08:15)



Now hash pointers are also useful like any pointer when you probably did C, C++ programming you use pointers to create binary trees and you know various kinds of trees. So here also we can create various kind of data structures. So one data structure that is most common and actually there are a lot of optimization done now on this data structure. But for now let us assume that the data structure is as simple as this, which is called a Merkle tree.

If you have a block of data, you have a hash pointed to it. Now if you have another block of data, you can have a hash pointed to it. And then what you do is you combine these two hash pointers as a block and compute the hash of those and create a hash pointer to those. Now you have two more blocks of data. So you have hashes of those.

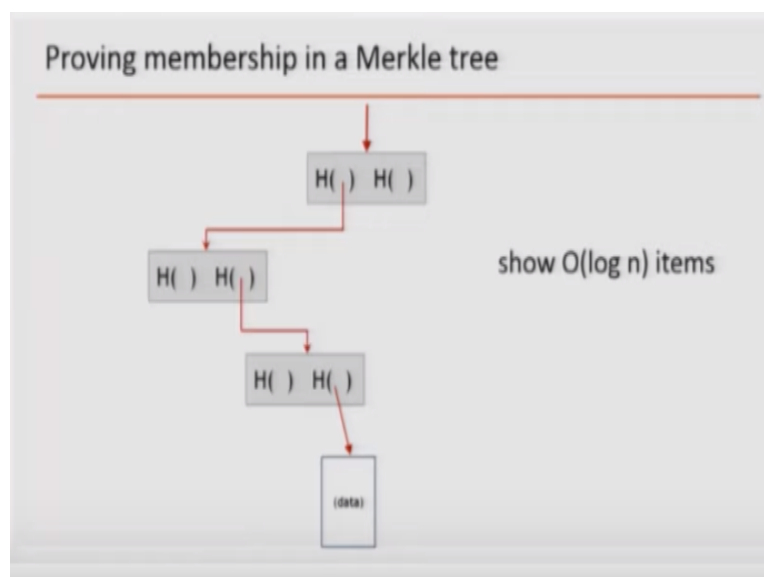
So once you have two hashes, what you can do is you can create the hash of that. So now you have another node with two hashes. Then you create a hash of that and point to that. So now if you have this hash, then you will be able to figure out this block and then you will find the two hashes. So then you can go to this block and this block and then eventually can go to the data and same thing happens when you have more blocks.

So and if you want to balance it, then you do it accordingly. So this is what is called a Merkle tree. So Merkle tree, now what is the advantage of doing this instead of chaining this data in the blockchain, in the chain data structure.

The advantage is in order to locate a block, if you have a chain then you have to do a completely linear search like you have to see if you have the hash pointed to this, then you have to compute hash of the previous blocks to see which one is just before it. And then you have to compute the hash before that to know which one is before that and so on and so forth. So that is a lot of work.

If instead of chaining them you have in the tree structure, and then you have the hash to this, then you know that this block has not been tampered with. So and then you locate this, and then you locate this, and you locate this, and then you locate this. So it is a logarithmic search for a particular block of data.

(Refer Slide Time: 1:10:39)



So to locate a block of data, you need to look at $\log n$ number of items. And whereas in a chain, you have to look at linear number of items.

(Refer Slide Time: 1:10:48)

The slide is titled "Advantages of Merkle trees" and contains the following text:

- Tree holds many items
 - but just need to remember the root hash
- Can verify membership in $O(\log n)$ time/space
- Variant: sorted Merkle tree
 - can verify non-membership in $O(\log n)$
 - (show items before, after the missing one)

Handwritten in red on the slide is a diagram of a Merkle tree structure with several nodes and connecting lines. At the bottom right of the slide, the word "Estonia" is written in red.

And then the advantage of Merkle tree is that you know with n levels you can handle 2^n blocks of data. And then you only need n depth, right. So which means logarithmic depth. So you can have verify membership. So suppose you want to know what is the membership of whether this particular data, piece of data is present. Then a logarithmic search to the trees depth from the root and then you will be finding that.

Now there is a variant called sorted Merkle tree where the members are also sorted. In that case, verifying non membership is also easy. So here verifying non membership is going to be expensive, whereas here verifying non membership is also going to be $\log n$, because data is sorted. So if you do not find it, you can also find what is before it and what is after it.

So this kind of structure is actually used in one particular case in a very successful deployment of this hash tree technique in Estonia. So in Estonia, the government actually maintains this hash trees for all kind of government documents. And what happens is that when documents are created, they are basically put into this tree structures through the routers under which this network belongs.

And then to the next router to the next router all the way. And then every, let us say every 5 seconds, whatever is created, whatever information is created is put in a tree

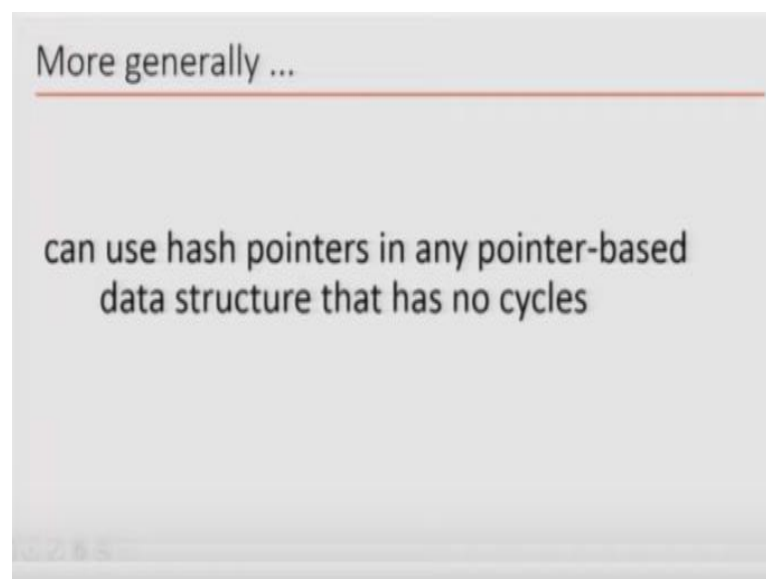
throughout the country. And then the trees route is then put into a list. And so the list contains, let us say, every 5 seconds, whatever is created the Merkle tree root of that. Then next 5 second Merkle tree root will be that.

And these things are also hash connected. Then the government publishes the latest hash. So you have a so you have a data structure which looks like this. So this is 5 seconds. So it is put in here. Then another 5 seconds data the tree underneath it will be put in here. But these are also hash connected. So this hash connected blocks is called a calendar blockchain.

So this calendar blockchain to convince people more into the government what they do is that every weekend they publish the latest hash value. The latest hash value that has been of the block that was created is published here in a newspaper. So if somebody changes something here some internal government employee, then he has to change the next one next one next one and then eventually this hash will change.

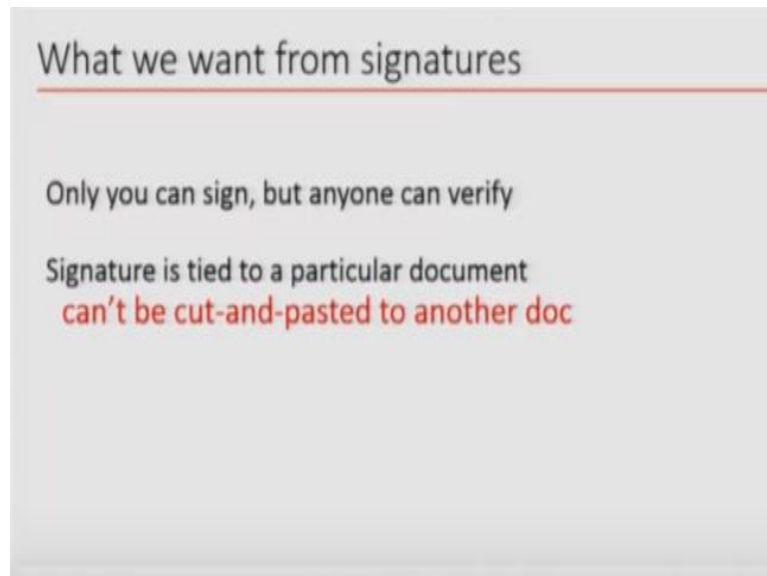
Now the public has this hash. Therefore, it will be not very easy if the public challenges that I believe that some data has been changed here. Therefore, this block got changed, therefore this block got changed, this block got changed, this block got changed. Then the government will be in trouble because public can challenge this information. So that is the idea of, you know Merkle tree.

(Refer Slide Time: 1:14:11)



So generally, any non-cyclic data structure can be created using hash pointers. So this is an important thing, so you can have more optimized data structures than even Merkle tree.

(Refer Slide Time: 1:14:28)

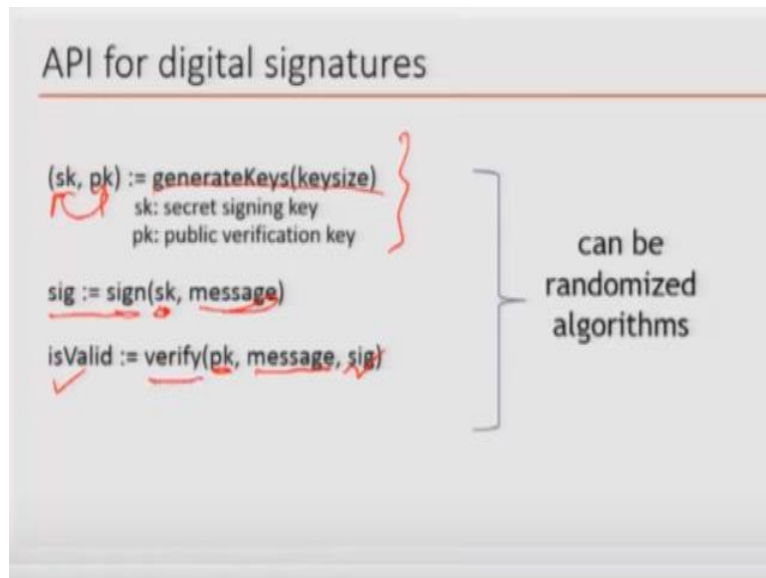


So the other thing that is very important here is the digital signatures because you always want to know in blockchain, whoever is doing transaction, the transaction before being recorded, one has to verify that it actually being done by somebody who claims that he is doing the transaction. So therefore, everybody has to have a public key and private key to be part of the blockchain ecosystem.

So I have a public key and I have a private key. Everybody knows my public key if they want to know whereas private key or secret key is kept with me. So if I sign a transaction with my private key, as I said before, digital signatures are basically some kind of an encryption done to prove that I know the private key. So anybody who can open decipher it with my public key knows that it is me who did this.

So whenever I put out a signature, then people would know that I did the transaction and if I was supposed to do the transaction, then that is a valid transaction. If I was not supposed to do the transaction, it is my signature, then the transaction will be considered invalid and will not go into the permanent blockchain. So how do the digital signatures work?

(Refer Slide Time: 1:15:42)



First, you have to have a function for generating keys. So you say that I wanted 256 bit key. So then you call the corresponding functions generate keys. It will give you a key pair that is secret key or private key and public key or pk. So then to sign a document, you take a message or a message digest, and then you encrypt it with your private key.

To verify that it is your signature, all I have to do is I have to check whether with your public key, I can decipher the message. And then if I can do that, then I know that this signature is valid. So there should be an isValid function. So anybody who wants to do the create key pair has to use this function generate keys. Anybody who wants to sign has to use the secret key and use the sign function.

And anybody who wants to check whether it is indeed the valid signature has to use the verify function.

(Refer Slide Time: 1:16:48)

Requirements for signatures

"valid signatures verify"

`verify(pk, message, sign(sk, message)) == true`

"can't forge signatures" ✓

adversary who:

knows pk

gets to see signatures on messages of his choice

can't produce a verifiable signature on another message

So only valid signatures, the digital signature must have the property that only the valid signatures will verify accept others will fail to verify if I am saying I am signing but I am using somebody else's secret key, then it should not verify as my signature. So you should not be able to forge somebody's signature. So unless so anybody who wants to challenge that it is my signature or not can use my public key to check this.

But he cannot produce my signature, because I keep the private key secret, unless of course my private key is stolen.

(Refer Slide Time: 1:17:24)

Practical stuff...

- algorithms are randomized
need good source of randomness
- limit on message size
fix: use Hash(message) rather than message
- trick: sign a hash pointer
signature "covers" the whole structure

So this is very important part of the blockchain technology that I should be able to have a verifiable digital signature. And I have to keep my private key secret. And also I have to be able to apply the signature on a message or a message digest. Now one

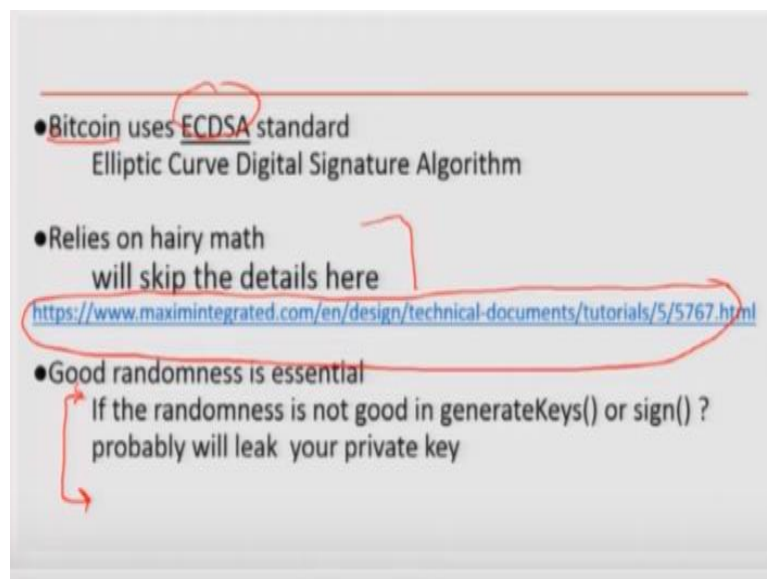
important thing that we must remember is that the algorithms that we use for all cryptography like generating keys and so on, requires a good source of randomness.

Now we know that the creation of random numbers is very difficult. And many times the function that produces random numbers turns out to be not very random. And that has been a source of problem in many cryptographic techniques. And people have been able to decipher keys by using the fact that the randomness that you thought was random, was not very random, and they could know, they could reconstruct it.

So it is very important to have a good source of randomness. Without that the whole construction of cryptography digital signature will fail. Also when you sign, you can sign a message, or you can sign the digest of the message. Because digest of the message is always smaller than the message. So that makes your signature faster.

And you can also sign a hash pointer. So not only you keep a hash pointer, you actually then sign the hash pointer and that will cover the entire structure. So hash pointer covers only the part of the structure. But if you sign the hash pointer, it covers the entire structure.

(Refer Slide Time: 1:19:14)



Now bitcoin blockchain, when bitcoin digital signature is very crucial. So if you use bitcoin, then it uses the ECDSA that is the Elliptic Curve Digital Signature Algorithm, which we did not talk about. It is a elliptic curve cryptography is considered more

secure, and also what we call quantum resistant, which is if quantum computers come into existence tomorrow, then RSA for example, will no longer be secure.

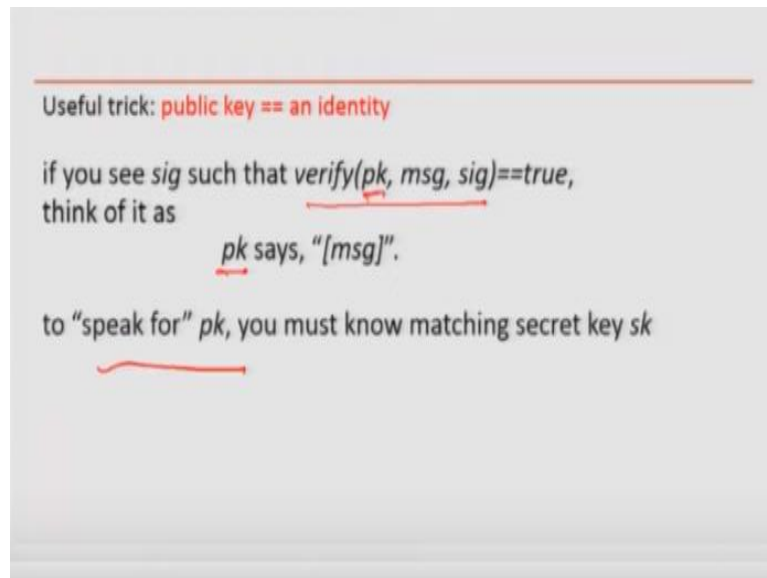
Because RSA uses the large integer factorization as its hard problem on which the whole construction is based. Now it is known that if there is Quantum Computing then factorization of large integers will be easy. Therefore, RSA by itself will not be very quantum proof or quantum safe. Whereas ECC the elliptic curve cryptography, ECC is considered to be quantum safe.

And so in that sense, Bitcoin has made a good choice, making ECDSA as your signature standard for digital signature in bitcoin blockchain. Now this ECDSA is not as simple as RSA where I was describing that it requires you to know prime numbers, it requires you to know a little bit about modular arithmetics. And it requires you to know a little bit about inverse, multiplicative inverse in a group.

And that is basically the mathematics that you require to understand how RSA works. Whereas in case of elliptic curve, curve cryptography, you need to know lot more Mathematics because you are going to do arithmetic on the points on an elliptic curve. And that you can learn if you are interested from many sources. One of the such source is here. And good randomness is a very important part.

And there is a whole area of research on pseudo randomness and random number generation and so on and so forth. But most of the times we rely on the randomness generation that comes with the system that we purchase, but if you are being very secure, then you have to worry about this source of randomness that is used to generate keys and so on.

(Refer Slide Time: 1:21:42)



So now final thing that is, we are going to discuss today is public key can also be used as an identity and that is what is done in bitcoin ethereum and this kind of blockchains the what we call public blockchains or permission free block chains where you as a person, it does not matter who you are. As long as you have a public key, you can become a player into the blockchain ecosystem.

You can try to do mining, you can make transactions, all that stuff and nobody would know who you are. And that is basically what is done is that you have to just tell this is my public key. And so anytime you want to send me a message, you basically use this public key and knowing that will help others to check your signature.

So when you sign a transaction, saying that I am making this transaction, since you have announced your public key as your identity, so they can use that to verify that it is indeed your signature. So the public key kind of speaks for you and nobody else can speak for you or nobody else can sign for you even if your public key is known because as we discussed before, the public key is not enough to do signature.

Signature requires the corresponding private key which you should keep to yourself. So to make a new identity in the blockchain, what you do is you create a new random key pair.

(Refer Slide Time: 1:23:09)

How to make a new identity

create a new, random key-pair (sk , pk)

pk is the public "name" you can use

[usually better to use $\text{Hash}(pk)$]

sk lets you "speak for" the identity

you control the identity, because only you know sk

if pk "looks random", nobody needs to know who you are.

We know how to do that with the create key function and pk becomes your public name and sk is kept as your hidden identity or key to your identity. So if you want to prove your identity you have to use sk to sign. Now if pk looks random, nobody needs to know who you are. So therefore, one of the big problem in blockchain is that I as one person can create hundreds of different identities.

I can create as number as many as key pairs, and I can use that to create multiple identities and do various things in under the various identities. And therefore people would not be able to know that I am the person who is doing all this.

(Refer Slide Time: 1:23:58)

Decentralized identity management

- Anybody can make a new identity at any time
make as many as you want!

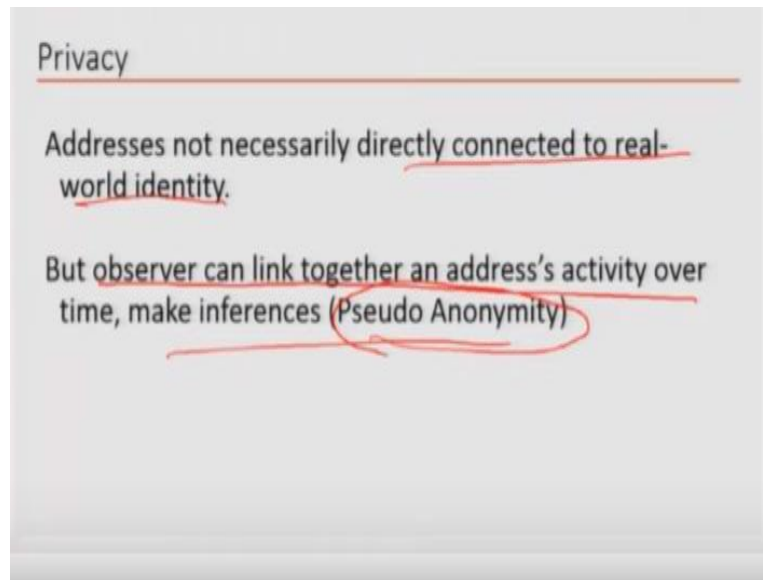
- No central point of coordination

These identities are called "addresses" in Bitcoin.

So that is what the problem is that anybody can make any new identity at any time and make as many as they want. And there is no central point of coordination. And

that was by design, because the anonymity of users was important for the originator of the bitcoin blockchain. And these identities are also called addresses in bitcoin. Because anybody who sends money to me, he sends it to my public key and that is why it is called an address.

(Refer Slide Time: 1:24:31)



And it is not connected to my real world identity. Nobody is checking. There is no other linking or anything. So therefore, you cannot discover ever who I am. However, there may be some investigative work people can do. And if you are not clever enough, then by linking together the various addresses that you are operating, what activity you are doing, and you are transferring money back to one particular identity and so on, they can somehow infer certain things.

And then once they know one of the identity, one of the public key's identity, real world identity, then all the other ones that are also created by you and somehow they establish the link between all this, they can make inference. So in that sense, bitcoin blockchain is not fully anonymous. It is called pseudo anonymous.

People have shown, researchers have shown that you can actually do various kinds of data mining algorithms to see which of the addresses are probably being maintained and operated by the same entity and things like that. But even then, discovering the real identity will require external investigative work that information cannot be found in the blockchain itself.

(Refer Slide Time: 1:25:53)

Summary for Lecture 2

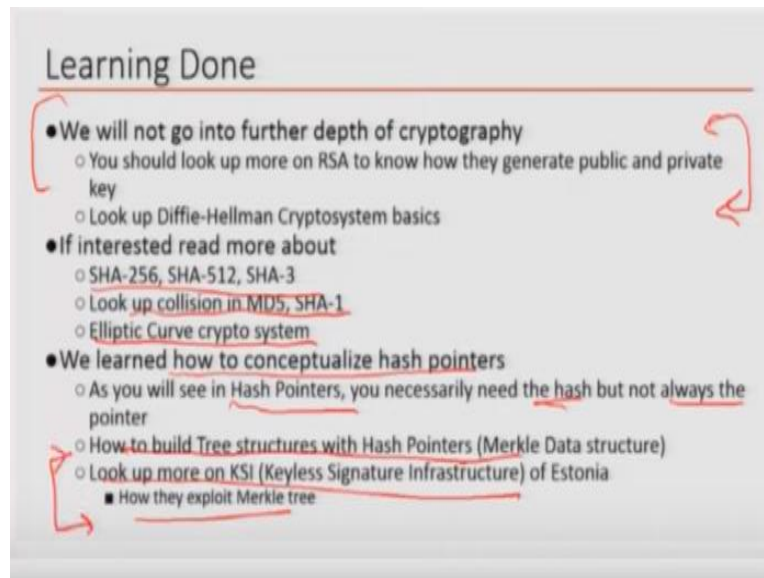
- Very basic concepts of cryptography that are used in blockchain
 - Secure Hash Functions
 - Collision Resistance
 - Hiding Property
 - Puzzle Friendliness
 - Very basic idea about SHA-256 which is used in bitcoin Blockchain
 - Digital Signatures
 - Public, Private key pair
 - Signature for authentication and non-repudiation
 - Hash Pointers
 - Linking blocks with hash pointers
 - Merkle tree data structure (Hash pointer based tree)
 - Public Key as identity

So summary of lecture 2 is that we covered very basic concepts of cryptography that are using blockchain. We talked about secure hash functions, the collision resistance, hiding property and puzzle friendliness. And this puzzle friendliness will come back when we discuss the basic operating principles of bitcoin blockchain. We talked about SHA-256 and little bit about its construct.

Digital signatures, how private keys use to do a digital signature and public keys used to verify it. And this is required for authentication and non-repudiation. We talked about hash pointers, what they are and how they are used to link blocks and Merkle tree as a data structure, which is used for efficient, locating efficient locating of data items. And finally, the public key as an identity.

So instead of having a username and password your public key becomes your identity.

(Refer Slide Time: 1:27:02)



Learning done so far in the course is that we will not discuss any more cryptography. You can learn on your own more about RSA, how it works. Also look up Diffie-Hellman cryptosystem, which is another public key cryptosystem based on discrete logarithm problem another hard problem. But we will not go into the depth of cryptography because that will require an entire course and I am sure there are a lot of courses on cryptography.

If you are interested to know more about this recent hash functions, which are being used in various blockchain lookup SHA-256 512, and SHA3, which is the latest, NIST standard. Look up the collision stories like how the collisions were found in MD5 and SHA-1. And look up a little more about elliptic curve cryptosystem. Because that is, you know ECDSA is the signature scheme that is being used in bitcoin.

So we learned how to conceptualize the hash idea of hash pointers. And as you will see the hash pointers you need the hash, but not always a pointer. In fact, the pointer is conceptual. We saw how to build tree structures with hash pointers like Merkle data structure and look up more about KSI, Keyless Signature Infrastructure of Estonia and how they use Merkle tree. That is a very instructive lesson.

You search on KSI Estonia, you will find a lot of information. I think it is a very interesting application that you should know about of hash pointer, Merkle tree etc.

(Refer Slide Time: 1:28:48)

Next Lecture

- A real simple blockchain example and exercise
 - Due to Prof. Pramod Subramanyan (IITK)

So in the next lecture we will see how to create a simple blockchain on your own and some exercise pertaining to that. And that was this simple blockchain example was created by my colleague, Pramod Subramanyan. So we will see that in the next class. Thank you.