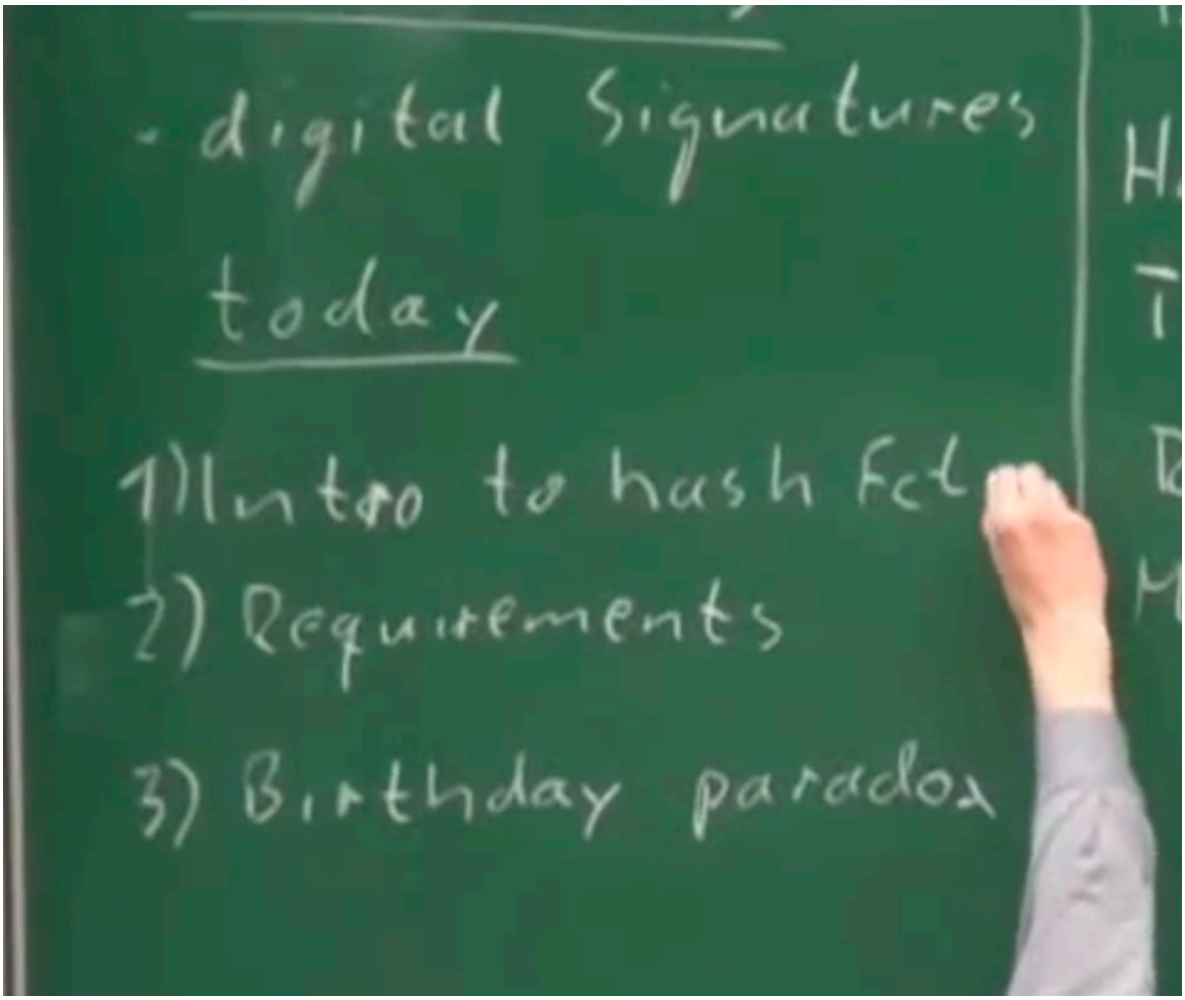
Lecture 20: Hash Functions



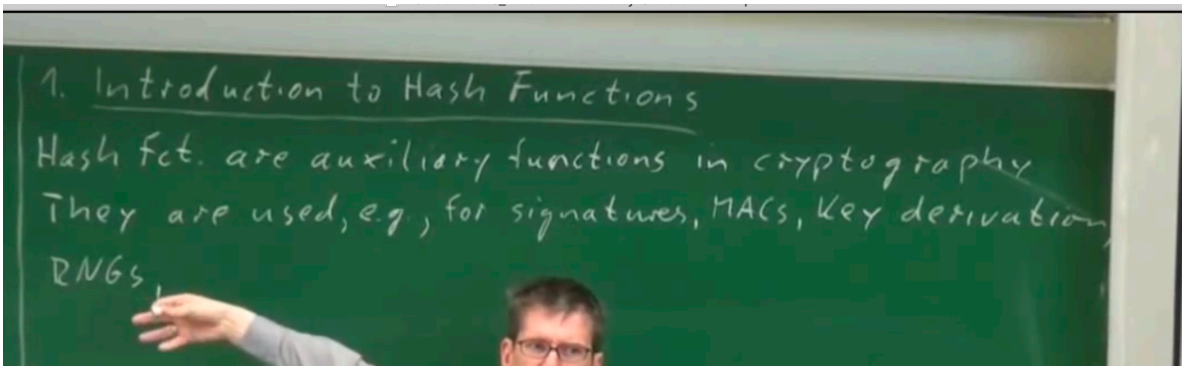
Definition: These are Auxiliary functions, they are used in conjunction with other cryptographic mechanism but they are super important. In cryptography they are used for e.g.

1. For **Signatures** which we are doing **today**,
2. **Mac's** which we are going to read in **upcoming lectures**,
3. **Key derivation** which we are going to do at the **end of the course**,
4. **RNG's** (Random Number Generators)

and many more.

Important point by Professor Paar is that incase we are going to look at real world crypto implementation, there is a very high chance that they have hash functions.

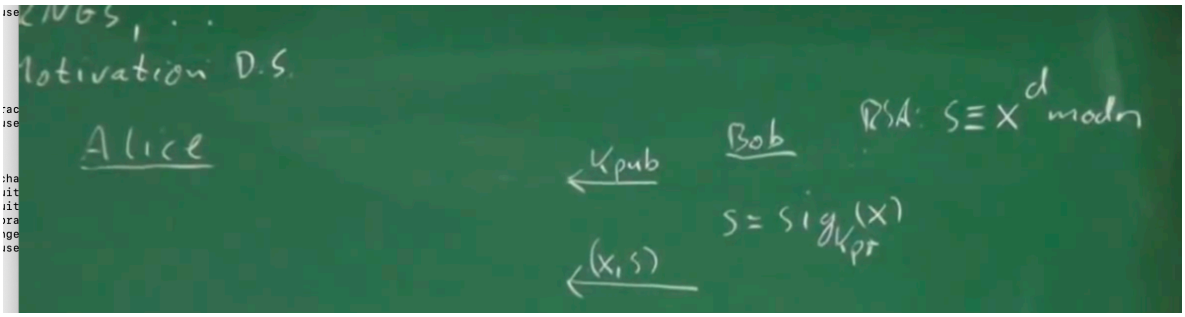
Introduction to Hash Functions:



Motivation:
Historically HashFunctions are used in Digital Signatures so we are going to look at it today.

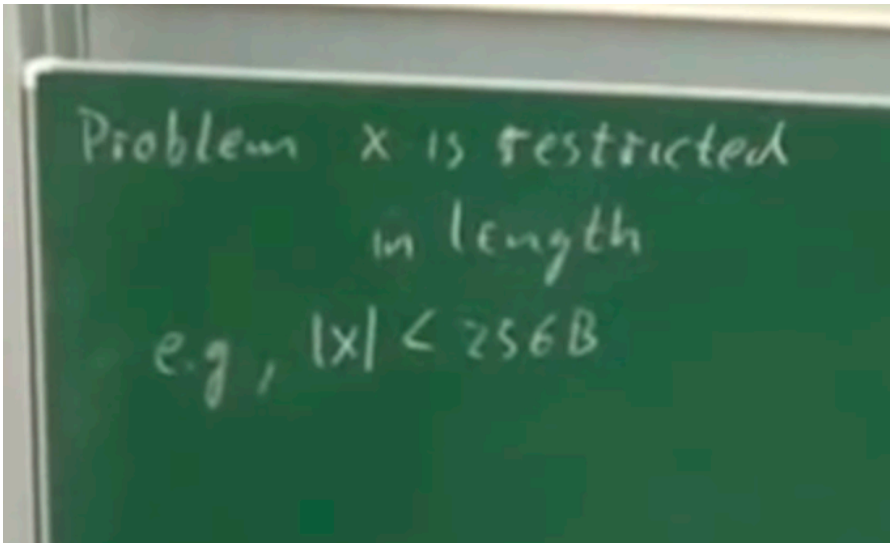
Alice Bob
Kpub
<————
(X,S)
<———— S = Sig(X) using Kpr

Lets discuss more about the implementation details:
if we use RSA then we would compute **S = X^d mod n**

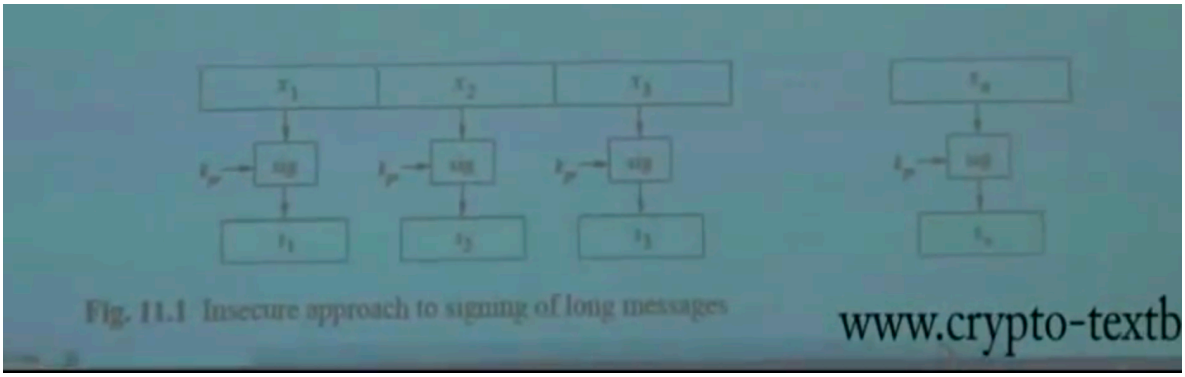


Question: What is the problem now ? if a PDF is given as email attachment.
Answer: Problem is we need to create a signature for each block. if you look at above formula **n** is 2048 bits only and we know that we can only sign max of 2048 bits or 256 bytes but typically our messages/pdfs are much longer than 256 bytes.

Problem: X is restricted in length e.g. |X| < 256 Byte



So what is the adhoc way to handle this problem ?



Actually above is the way most people think they can solve this problem i.e. chopping the message in blocks and each block is signed individually.

This is what we did in AES too.
The above way is really stupid approach and it is quite insecure.
Question: What is the problem with the above approach ?

1. You can exchange the blocks i.e. lets say X is a big message a Mail with attachments and if say someone drops a block which is very important i.e. Xn and Sn then remaining signature is still valid but the information is lost and there is no way to find the integrity of the message is damaged. Even you can reorder the messages too.

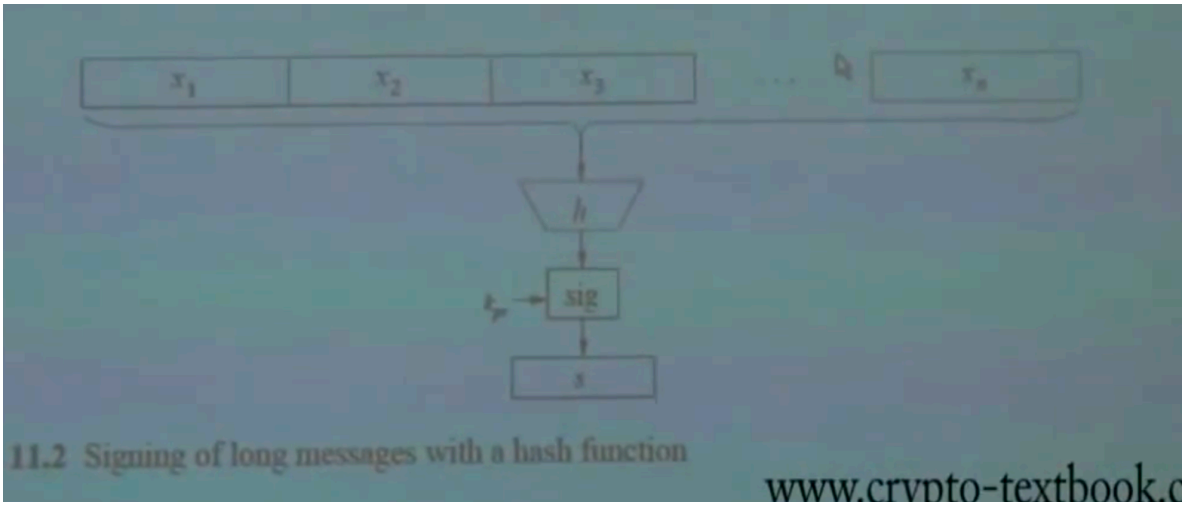
It has the same problem that we had in ECB mode i.e. Here we have Integrity, Non-Repudiation and blah...blah but only at block level not at the entire message.

2. It is a lot of work and takes a lot of time. for 256 MB PDF requires 256 MB of signature and we need to compute 1 million RSA exponentiation and this will be really really slow.

What else we can do ?

Solution: Somehow “compress” the message X prior to signing, i.e. make smaller before signing.

So what are we going to do ?



So what we will do is we are going to feed the big PDF file 256 MB in a function called h which is a hash function and that h function gives hopefully short output which we feed into RSA signature function.

RSA signature function is slow but we are going to do operation only once i.e. on a shorter output and the final output is **s**.

By Looking at above picture we are deriving the requirements for Hash Functions. (Just by looking at the picture and not about any arithmetic properties)

Basic Protocol for Digital Signature with H function i.e. Hash Functions (Little Modification to the protocol above):

Alice Bob

Kpub
<————————

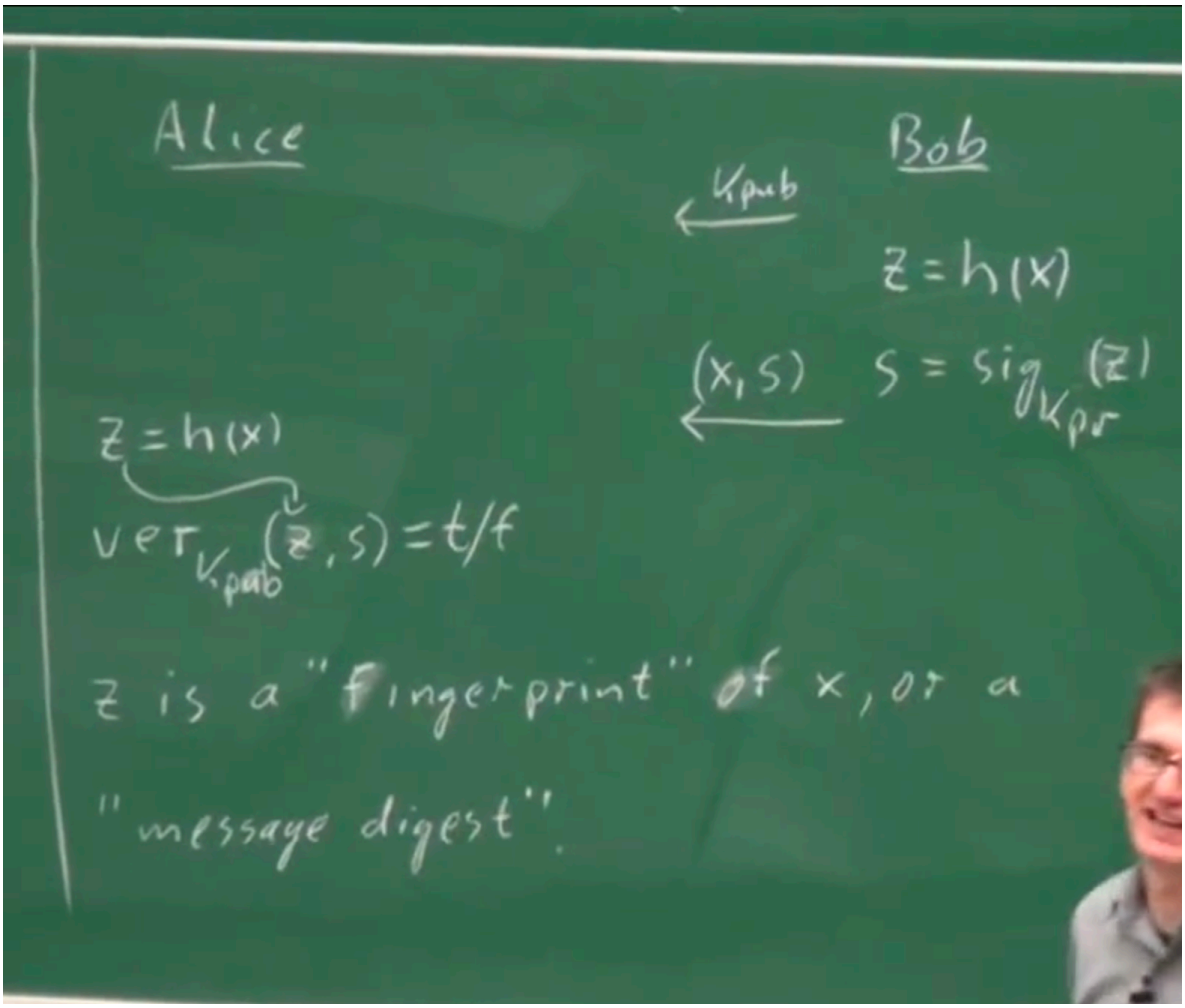
Z = h(X)
S = Sign(Z) using Kpr

S,X
<————————

Verification:
Ver(h(X), S) using Kpub

As X is not involved in Signing and Verification so this can be a concern and this is a concern (It is opening an **Attack**) which we are going to discuss later in the Lecture.

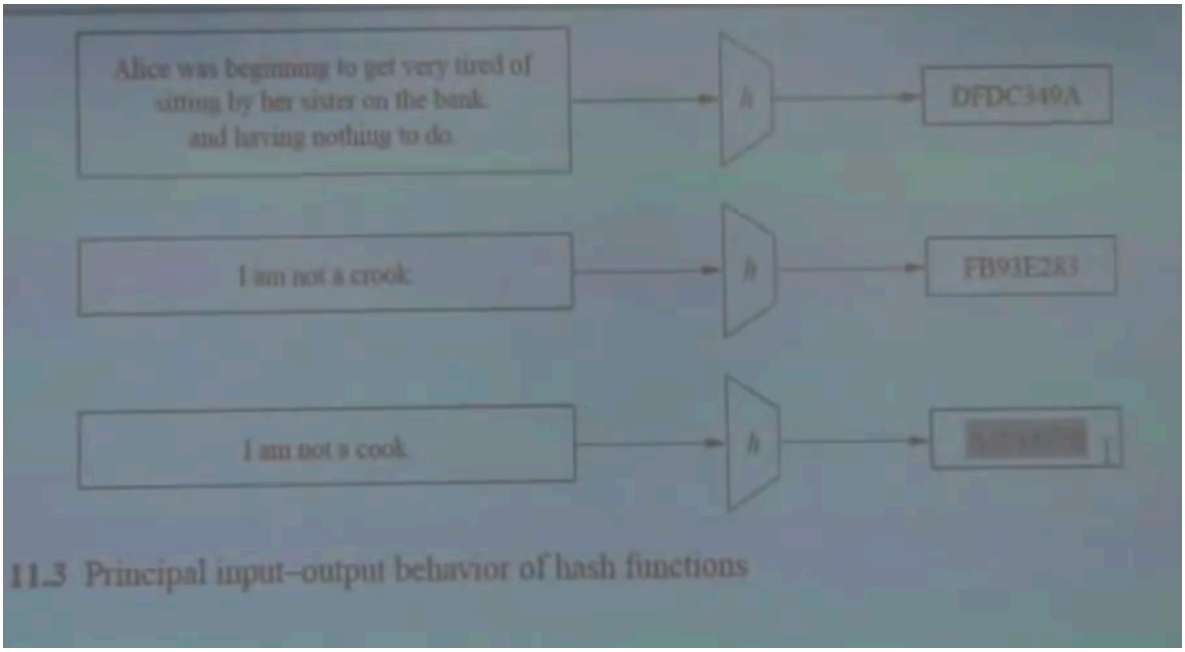
Here **Z** is called the **fingerprint** of the message “X” or **Message Digest.**



Chapter 2: Requirements
Normally professor Paar gives the algorithm and then we discuss about it and requirements but today we are going to do requirement engineering.

Question : Why we want this HashFunction ?
Because we are having this real problem of Signing Long messages.
Hence we came up with first requirement.

1. Arbitrary input length, i.e. it can be an Email or a Hard disk
2. Fixed, Short output length
3. One point which we discussed above is that Signing should be fast. In Naive approach the problem is it is quite slow so we don’t want to do that. So requirement is it should be **Efficient**

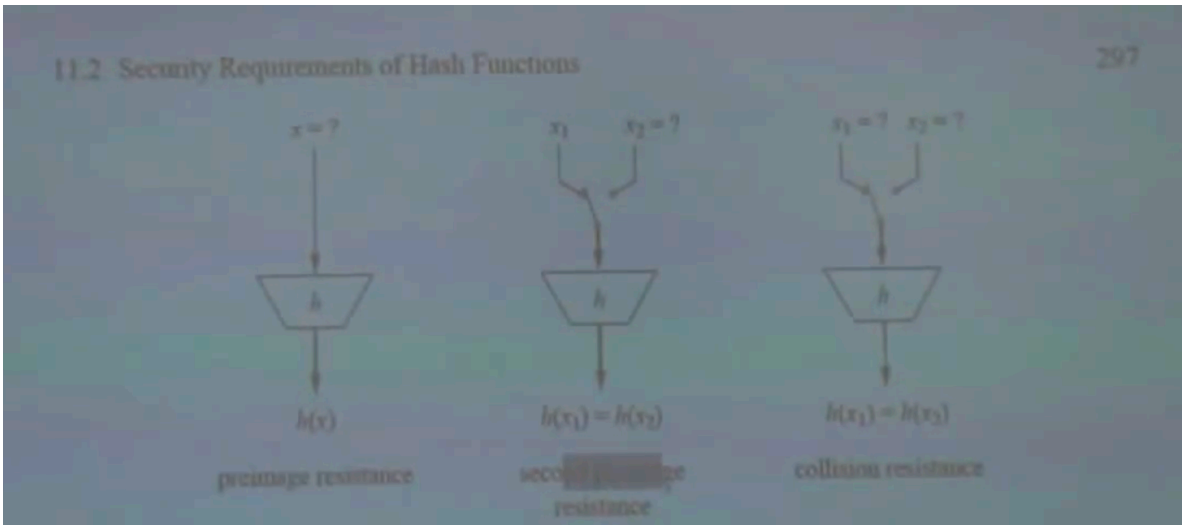


First Input is quite long still we get the same length output.

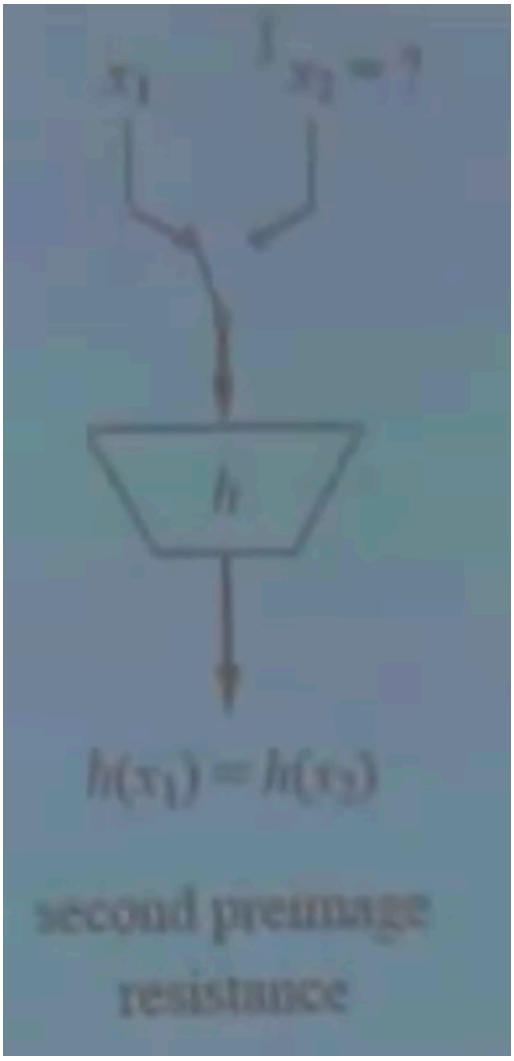
One bit flip should give something very different output depicted in second and third input.

Next 3 requirements are related to Security which are quite interesting.

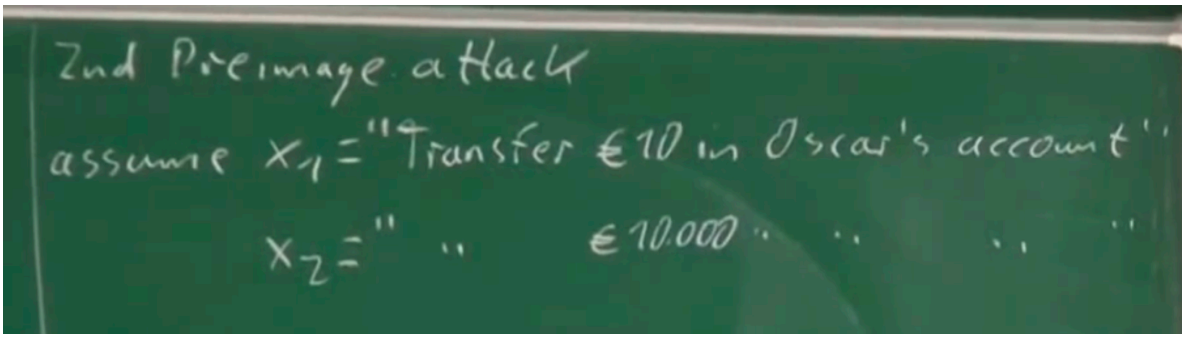
4. **Preimage resistence** or **one wayness**/one way function i.e. from hash output it should be impossible to compute the X. For Digital signature it is not that important but for many other applications it is quite important like "**Key Derivation**"

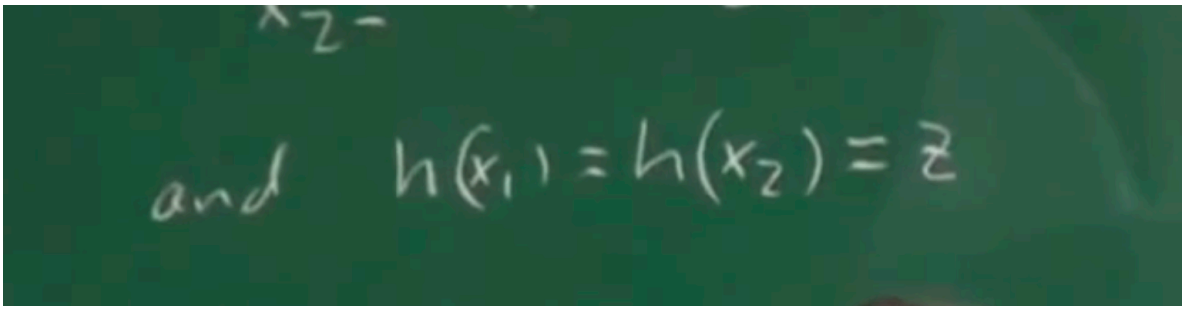


5. **Second Preimage resistence:** Say given X1 and output and now attacker should not be able to compute second X2 with same output i.e. same hash output.

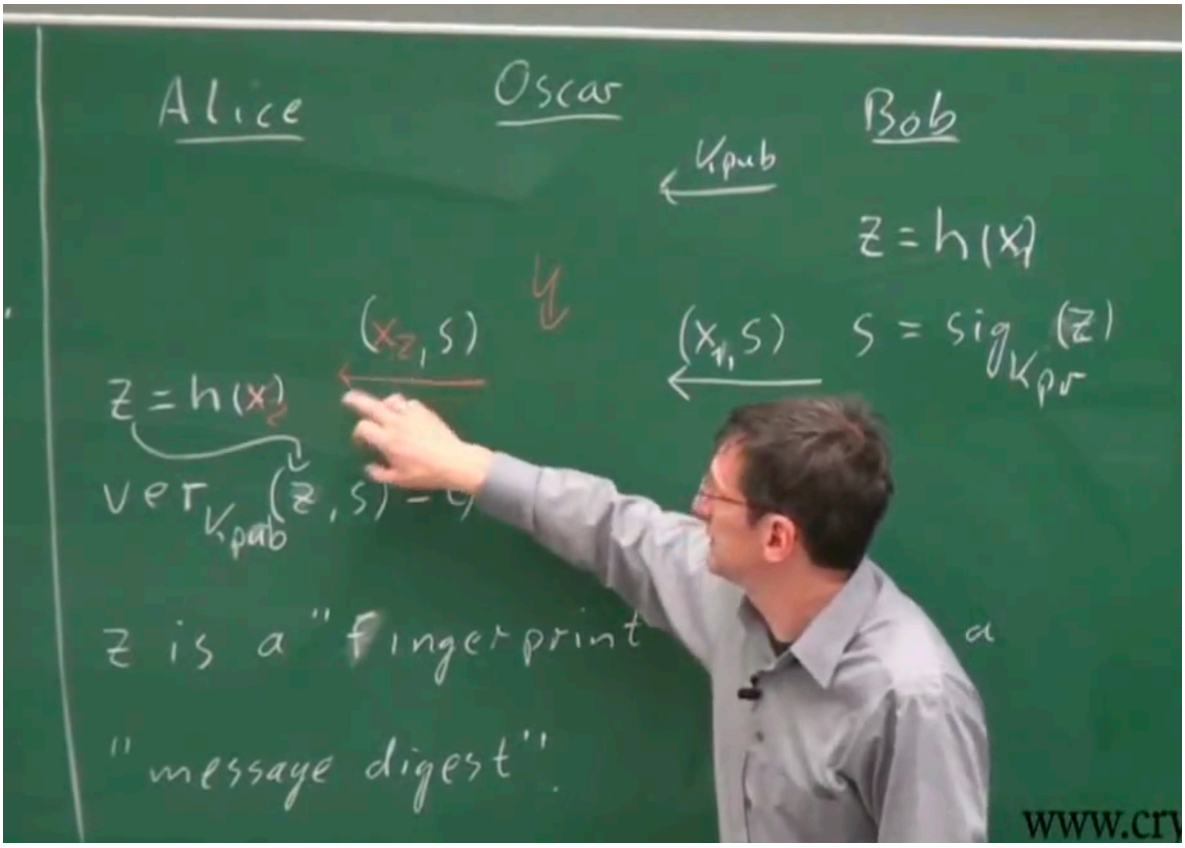


Assume the first message is :
X1 = “Transfer 10 Euro in Oscar’s account”which bob is sending to bank and Say oscar wiretaps so say Oscar changes 10 Euros to 10000 Euros. so X2 becomes “Transfer 10 Euro in Oscar’s account” and some how both X1 and X2 have same hash value i.e. “Z” now if oscar can do that then what will happen ?





Changes to the protocol:



Here Oscar wire taped and sent X2 and S and as we know X2's Z i.e. H(X2) matches H(X1) so S which is Encr(H(X1)) will also verify and user might think it is a valid case.

Thinking again:
Here we are Signing the Hash so if **Hash** of 2 messages is same then **Signature** will be same and hence **verification will pass for different message too**. This is one big issue.

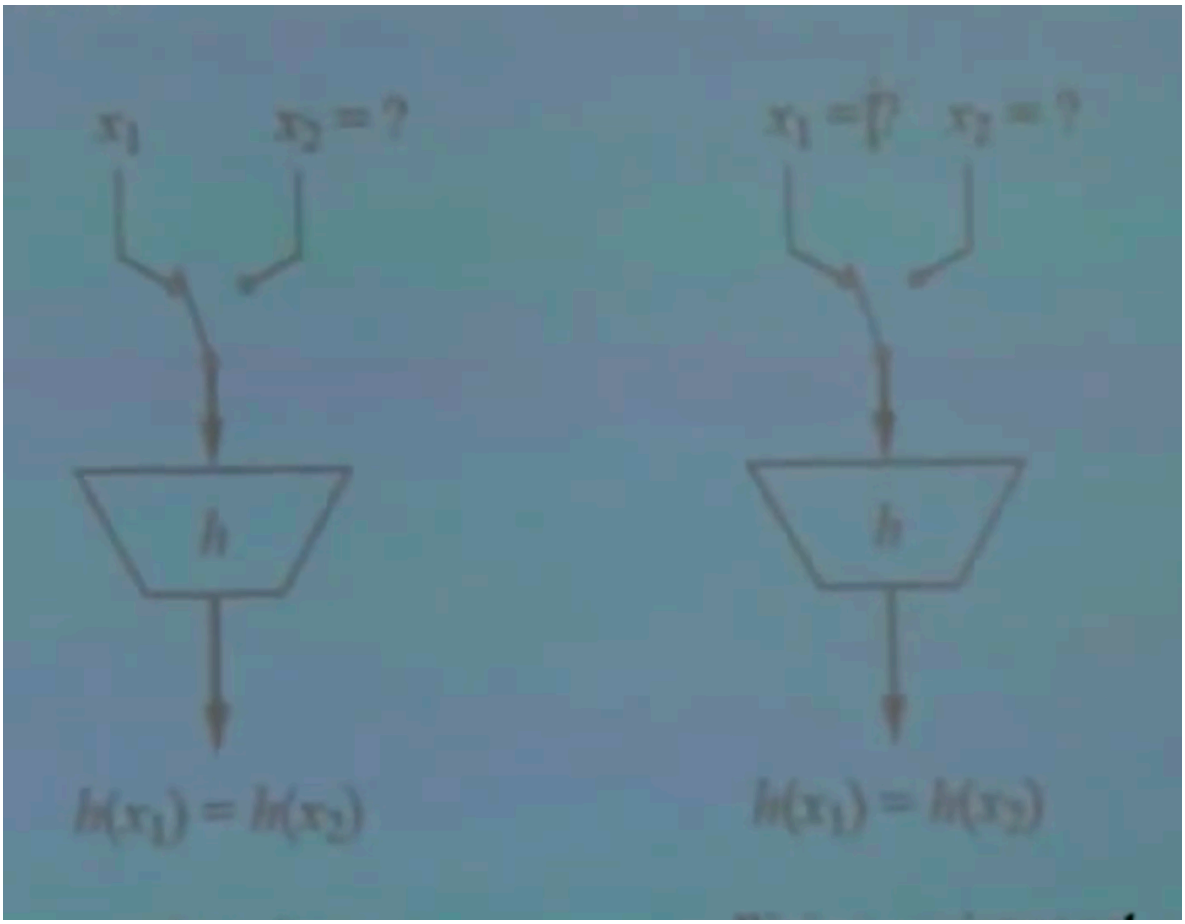
So actual issue is both signing and verification are operating on Z and they are not operating on the message so if you can find the Hash of 2 messages as same then it is like in older Digital signature protocol, both the message are same and hence signature and verification mechanism succeed.

So say you signed a message and now you signed another message and there are chances that verification might pass and this is what the issue is.

So Hash Functions should be Second preimage resistance.So we want to have some kind of mechanism such that this attack should not work so **Oscar, if he knows X1 should not be able to find X2.**

Second preimage is also called weak collision resistance.

6. **Collision resistance:** This is the most important attack which is the motivation for 2nd half of the lectures. This is almost the same as second pre image but what is the difference ?



In Preimage resistance X1 is given and oscar has to find the X2 however in Collision resistance oscar has to find both X1 and X2.
Question is what is the usecase and why this is important ?
Answer: Say oscar is able to find X1 and X2 which have the same hash value

Alice Oscar Bob

X1, X2 Kpub

<————————

Oscar tricks Bob in

signing X1 (somehow)

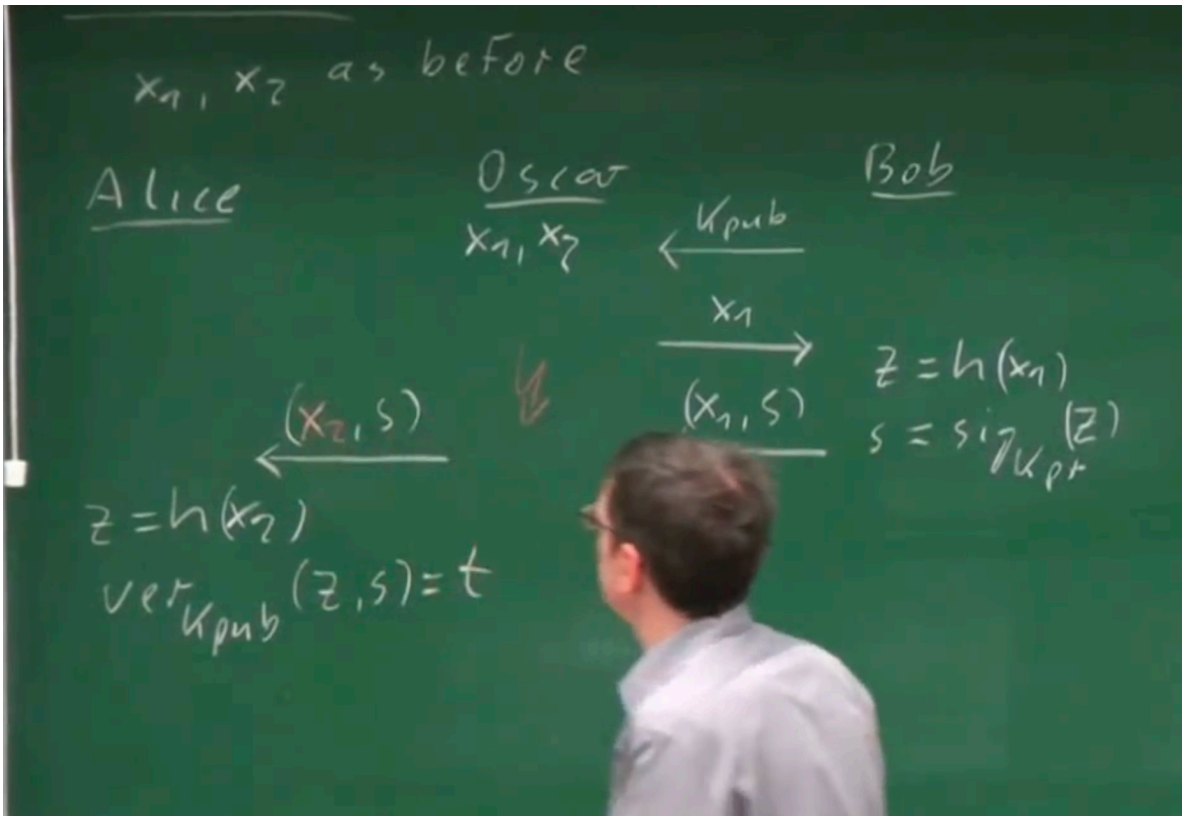
——————> Z = h(X1)
Sign = S(Z) with Kpri

(X1, Sign)
<——————

Oscar intercepts and
replaces X1 with X2
(X2, Sign)
<——————

Now Alice as a
bank so
alice tries to verify
Alice hashes X2 as
h(X2) which returns Z
as h(X1) == h(X2)

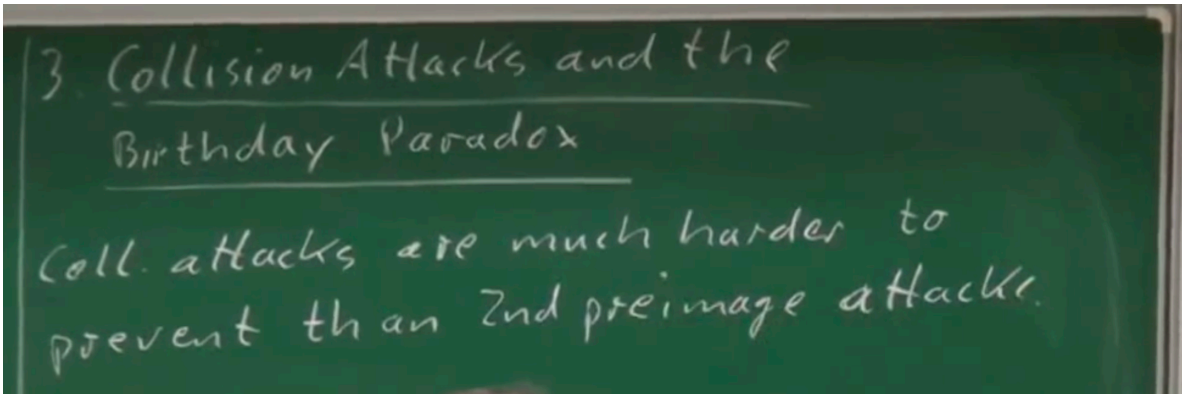
now alice tries to decrypt the
Sign and boom, it works.



So these are the requirements.

So now we come to the 3rd chapter of today. This is the main topic of today.

Chapter 3: Collision attacks and the Birthday Paradox



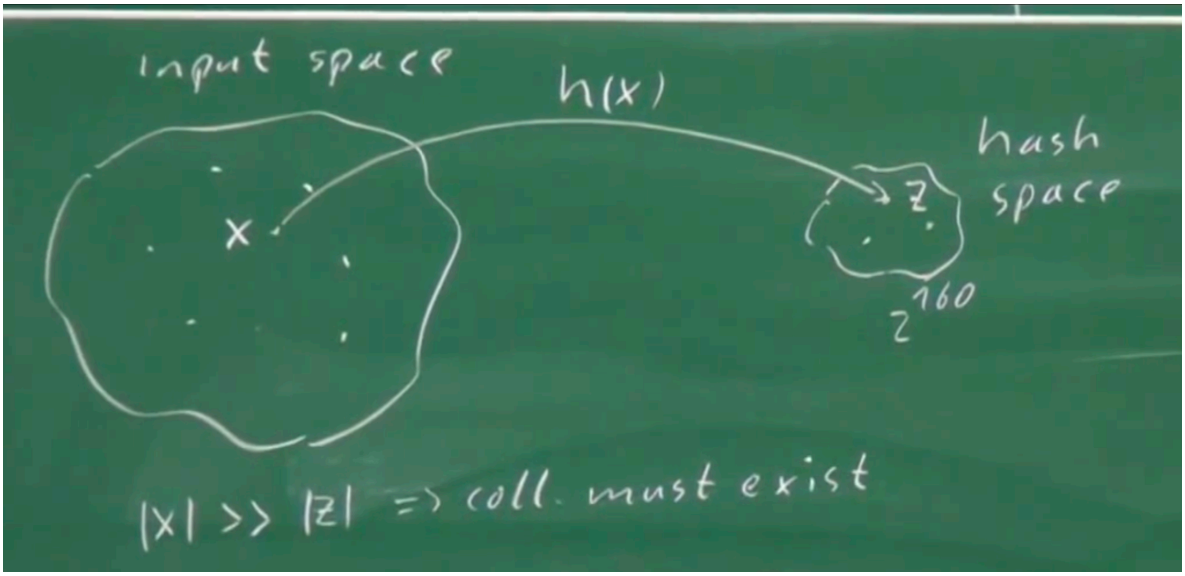
Collision Attacks are much harder to prevent than 2nd preimage attacks and professor paar is going to explain after some time.

In Preimage attack X1 was choose by Bob and X2 by Oscar and in case of collision attack, X1 and X2 both are choosen by Oscar they are almost same but it turns out that Collision attacks is much much much easier than preimage attack, as per paar it is SquareRoot times easier that preimage.

if 100 steps in preimage, 10 steps in collision attack.

This is a very current topic “Hash Functions” and the major requirement is Collision Attack prevention.

Question: Can we have Hash Functions without collisions ?
Answer: No why ?
Because condition 1 and 2 are the culprit i.e. Input can be any length and output is fixed length and hence many to one mapping cannot be overridden.
Say Output is 2^160 however if input is 2^160+1 then we have collision but as we know input size much much larger than output so collision must exists.



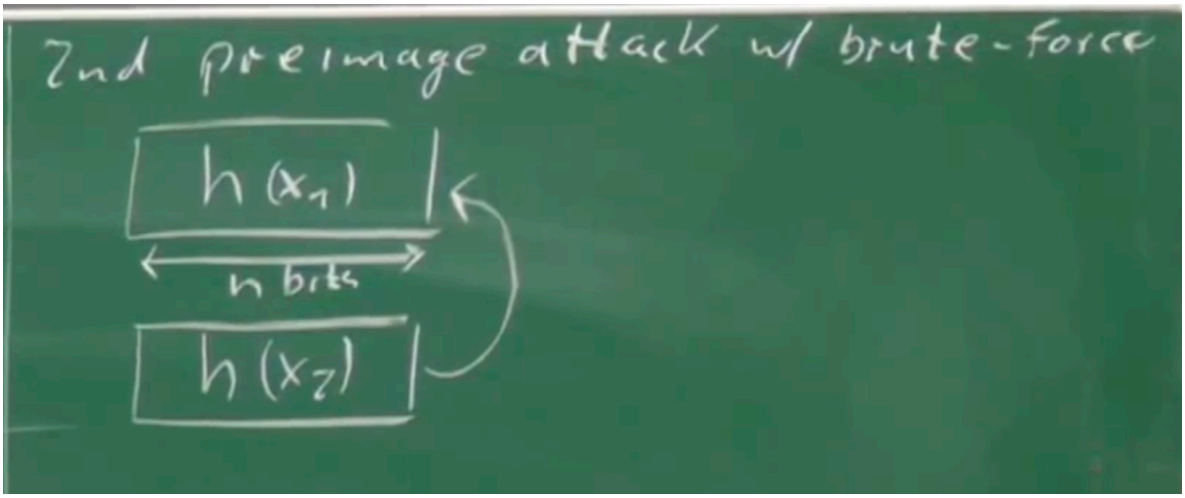
The name for the above problem is "**Pigeon Hole Principle**" say you have 20 pigeon and 19 boxes where they live then at least one has to share the box. (Collision)

We can say that if collisions always exists so we can always find the attack so we can think that HashFunctions might not be the solution but this is not true. so what can be done ? So next best thing which we can do is that we must have a situation such that **we must make collisions very hard to find.**

Let's Look at how would we actually do such kind of attack on 2nd Preimage ? bruteforce ->

Given X1, Oscar find H(X1) and now oscar likes to have colliding X2 (Colliding H(X2)).

Say message X1 is **transfer 10 Euro to oscar** and he tried to find X2 which is say **transfer 10000 Euro to oscar** so he hashes them and check if they are identical and chances are quite low because of the N bit length of the message.

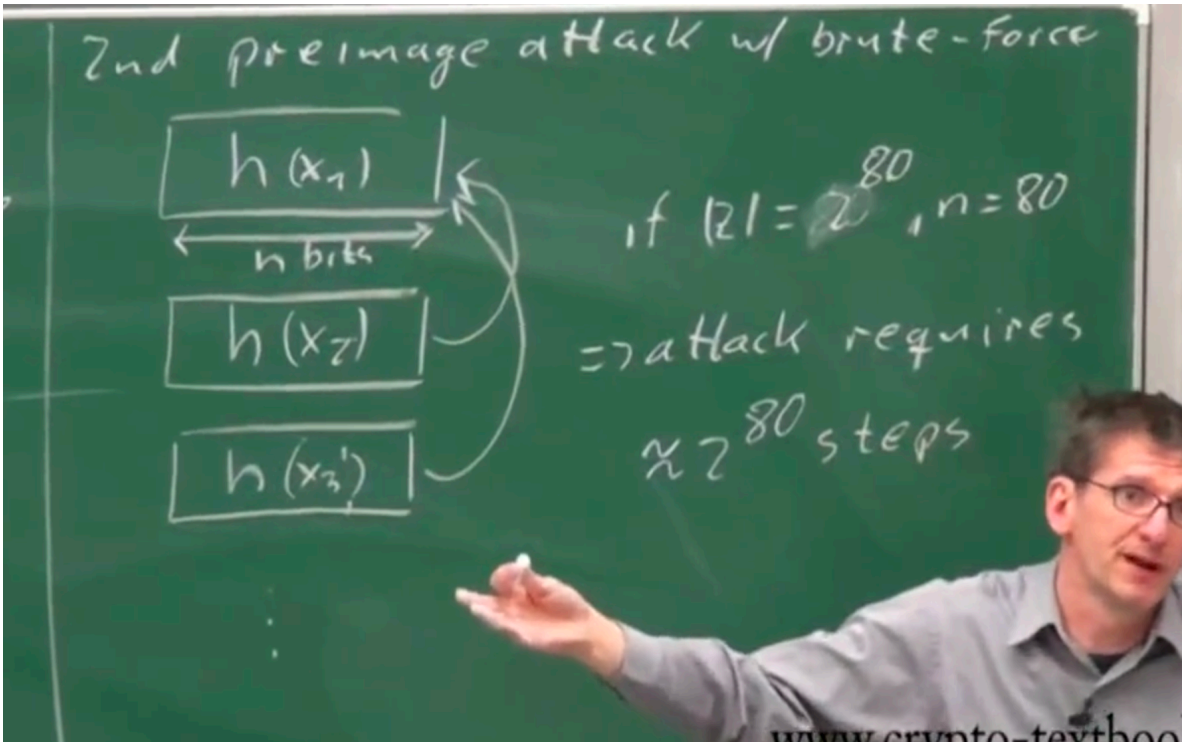


So you might think that it is safe but what oscar can do is, he construct X3 and what he can do is change the meaning of the message i.e. instead of 10000 Euros he can say 9999 euros etc.

However he can do other things too like, he can have the message with same content and adds a space in the end and that space is not visible to the bank and this space can change the Hash completely and if this doesn't work then take the next one.

Actually you can take the message and you can do variation at invisible places so **With the same meaning you can generate many many many messages and you can find the collision.**

May be at this point you are thinking that X2 might not be as per the format the X1 is like say X1 is **Transfer 10 Euro to Oscar** We can find the X2 but that might match the Hash value bit doesn't match the protocol or format of the X1 but it turns out that Say Oscar adds an extra ascii bit of Space character which is not seen by the bank and similar kind of stuff oscar can do.



If let's say we have cardinality of hash function i.e. output length is 80 bit instead of 160 then this attack requires 2^80 steps to find the same Hash. So if we have an input then we need to run it atleast 2^80 times to find the collision.

Now lets try to do the same thing with **Collision Attack**

Actually it turns out that lets have a party and how many people we have to

invite so that 2 people have birthday on the same day.

In case of Preimage birthday collision, question changes as my birthday is July 18 and How many people i need to invite so that one another person has birthday on July 18 on an average ?
Answer is 365.

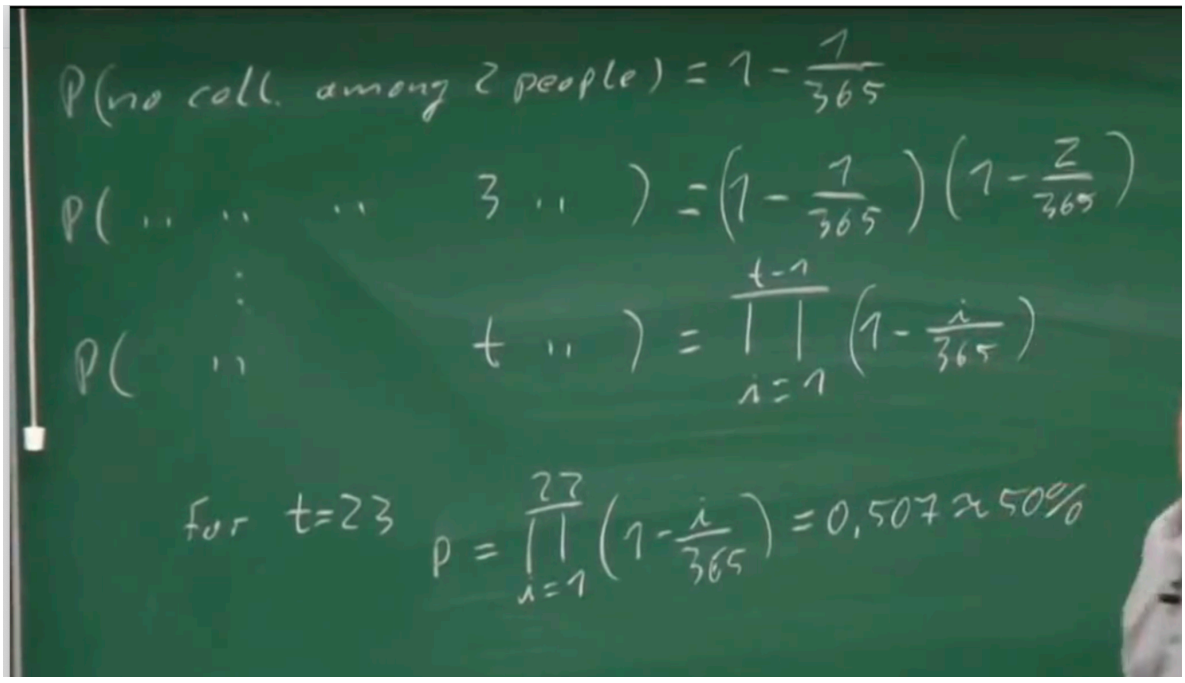
But if i change the requirement to say any two people having birthday collide on any day of the year.

Probability of having no collisions among 2 people is : 1 - 1/365
Probability of having no collisions among 3 people is: (1 - 1/365)(1 - 2/365)
Probability of having no collisions among t people is: (1 - 1/365)(1 - 2/365) (1 - 3/365)..... (1 - t/365)

For t = 23 we have 50 percentage chances of collision and this is roughly square root.

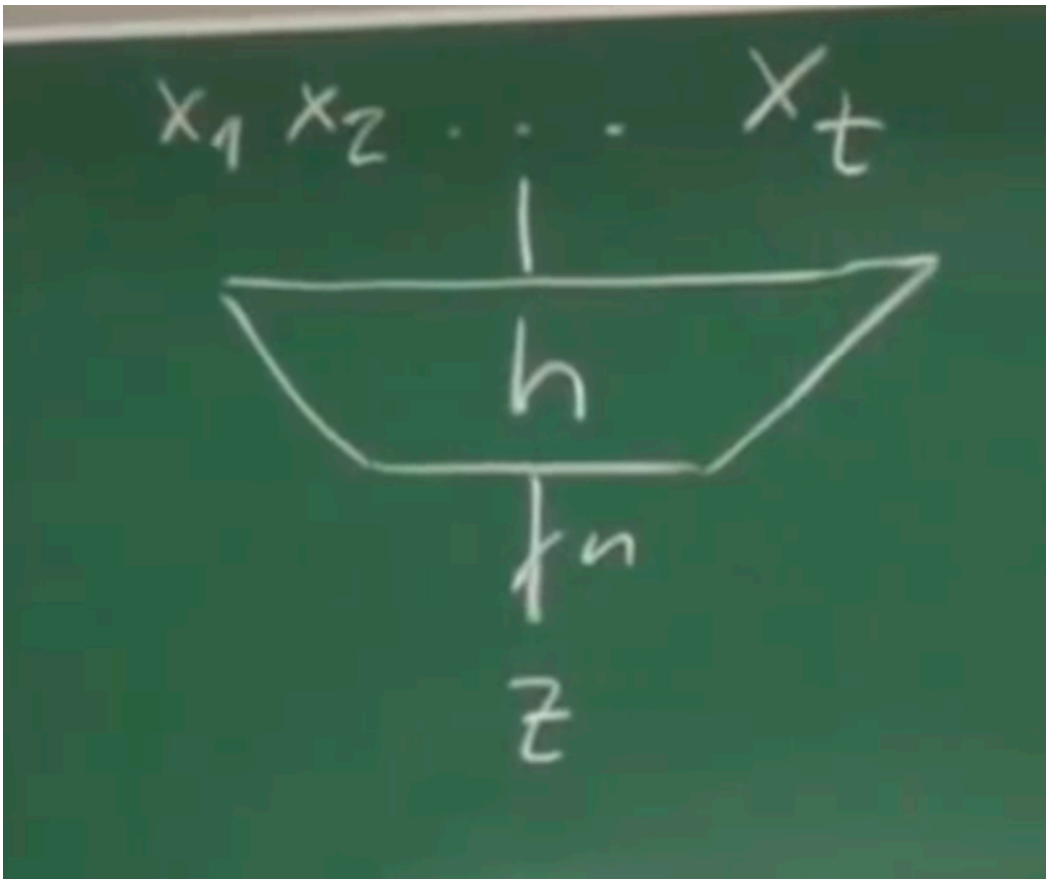
Can be though other way around too:
N c 2 => (N)(N-1)/2 is the changes and totol chances is 365 which becomes N(N-1)/2*365 and negation is 1 - (N*N-1)/2*365

We can also think this way:
If 2 person collide changes are 1/365, for 3 people chances are first 2 person collide + if first 2 doesn't then 3rd person can collide with any of them so 1/365 + (1-1/365)*(2/365) and so on.

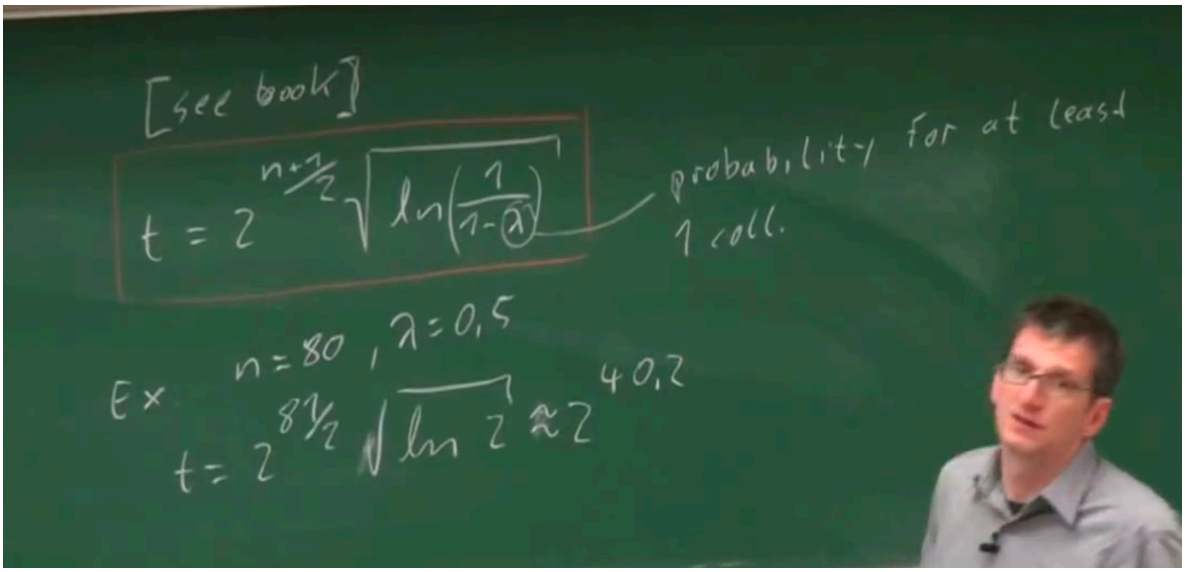


Now coming back to Hash Functions and applying the same Birthday paradox

for collision attack:



Now what is the current computation power of Crypto => 2^80 so with Second preimage we found that it takes 2^80 but lets come to Collision attack based on Crypto it comes out that for 50% probability it reduces the attack strength to 2^40 i.e. Half the strength and that might be the reason for increasing the n to 160.



Above function depicts the same square root strength.