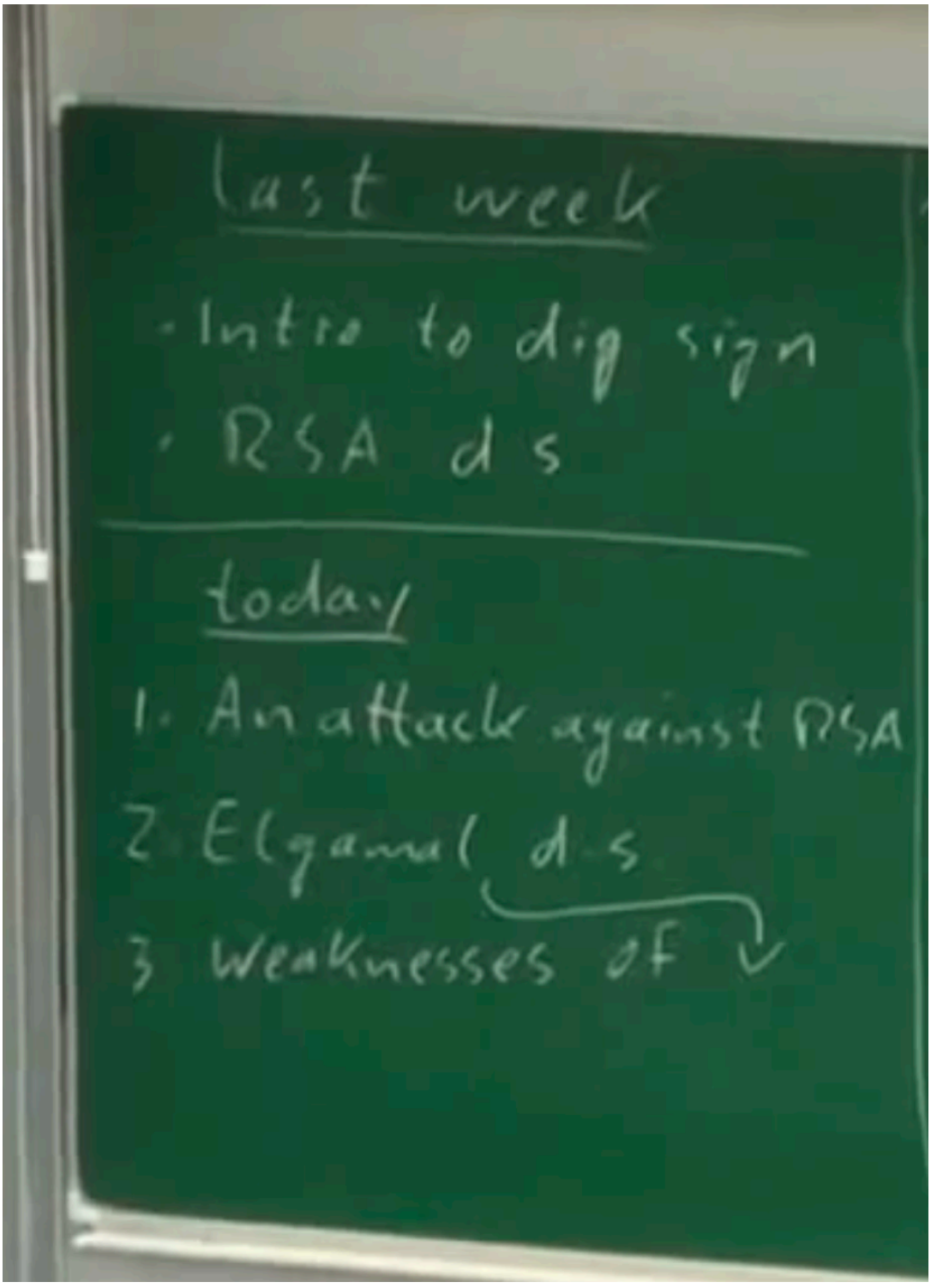
Lecture 19: Elgamal Digital Signature



Last Lecture:

1. we learnt about DSA then
2. we talk about Security services and then
3. we talked about RSA Digital Signature.

Today:

1. Attacks against RSA Digital Signature which we were not able to do in lastlecture
2. Elgamal Digital Signature
3. Weeknesses/Attacks of Elgamal Digital Signature

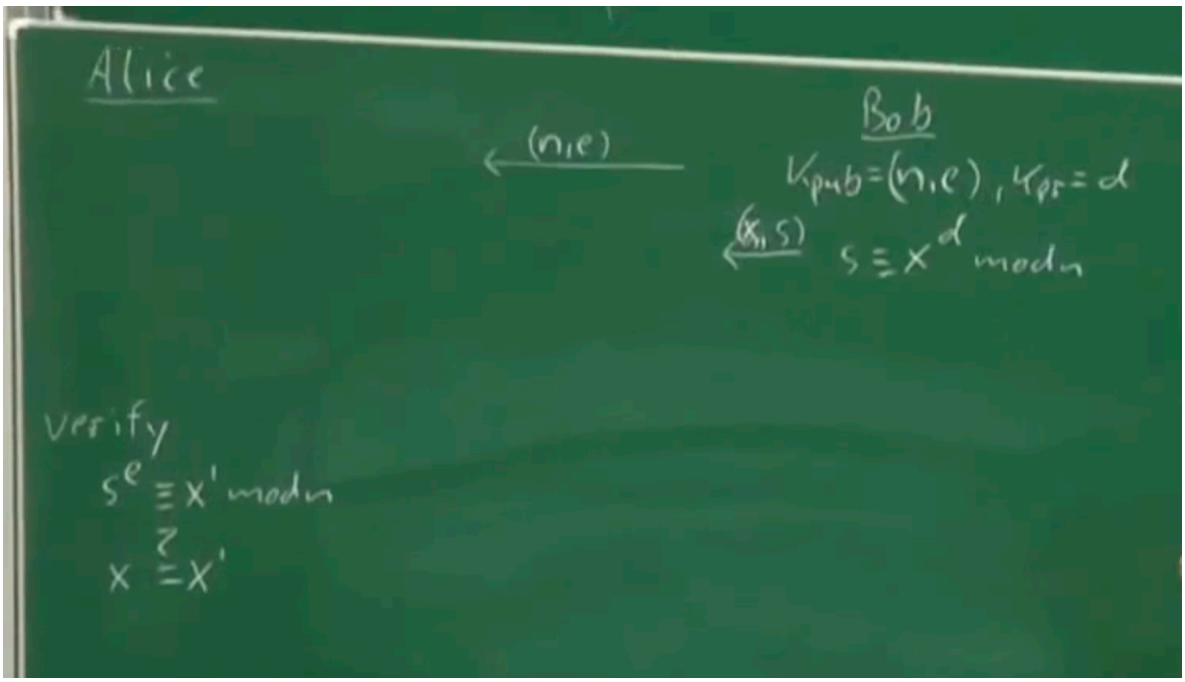
Attack against RSA:
Existential Forgery attack against RSA digital signature:
Protocol from last lecture:

Alice **Bob**
Kpub = (n,e), Kpri = (d)
S = X^d mod n

<———(S,X)———

Verify(S,X)
X` = S^e mod n
if(X` == X) ?

How does alice know that the Signature is valid ? She compares the X` with X.



So now question is What can oscar do ?
Oscar is interested in generating the Message with valid signature, ie say Alice is a bank and Oscar wants to generate a message like send 200 euro's to Oscar which is signed by the bob (With valid signature)

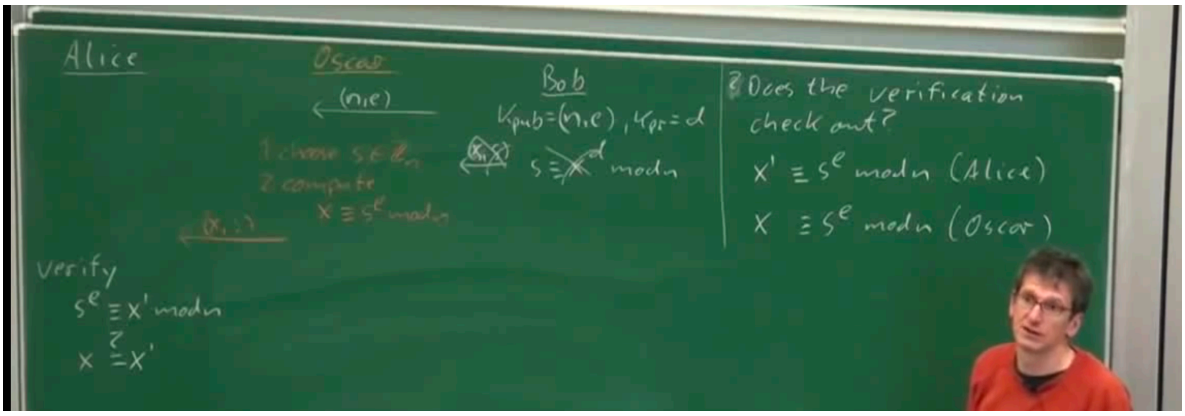
Pay Check like system is there where bank verifies the Signature of the sender. Pay checks need Integrity, Authenticity and Non Repudiation and all these are the important security services for **Digital Signatures.**

Lets see if Oscar can do this ?
So what oscar does is:

1. Oscar chooses a Signature S from Zn (Entire Space)
2. Then he computes X = S^e mod n as e and n are public components which is known to everyone.

and say Oscar sends (X,S)

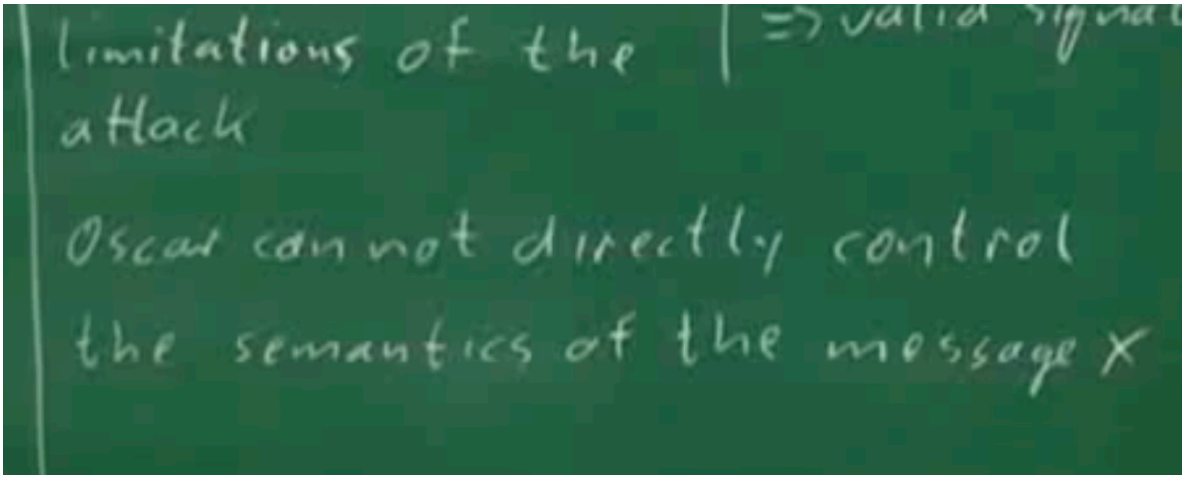
Question is does the verify function at Alice/Bank side passes or verify successfully and answer is Yes.



So from here you can see that both Alice and Oscar compute same X and verification function also passes.

Now this is a problem as Oscar generated a valid pair message signature which alice cannot detect.

Now question is what is the limitation of attack from Oscar side ?
Problem is X is some random message and it might not makes any sense and it is quite difficult for Oscar to generate a valid message like transfer 200 euro's to oscar's account.

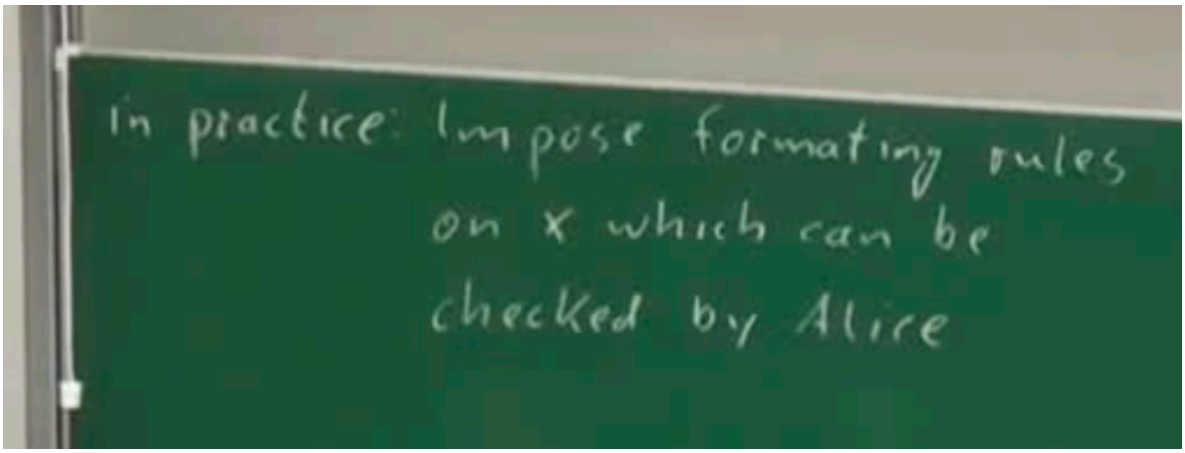


So we can say that it is a Theoretical attack but at Alice side automatically it cannot be verified if the X is semantically correct as per cryptographically it will be recognised as a valid signature but we want to detect such attacks.

Are there any counter measures ?

As we know that the RSA which we have done so far is School book RSA but in real world or in practise it is different.

So what kind of other things which can be done to change the school book rsa ?

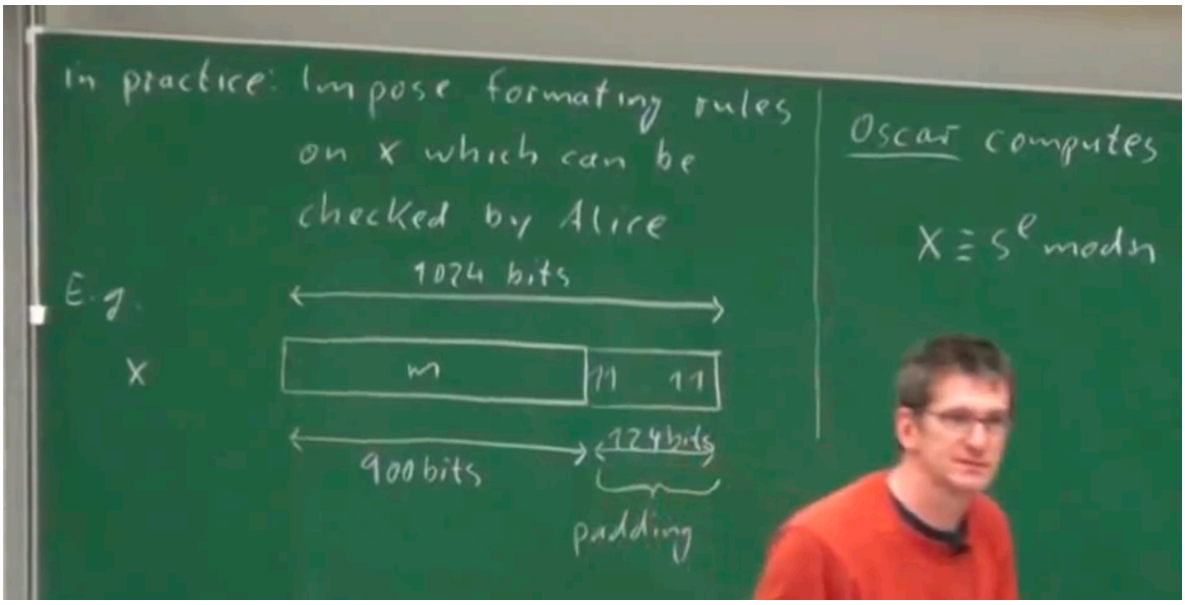


In Practice there are some rules added on the Message which needs to be verified by Alice.

Till now there were no requirements on what X can be , now we change it. Simple Idea for learning is say total length is 1024 but we say that we will only allow 900 bits and 124 bits left are padded with some arbitrary bits or say we say that all those 124 bits as 1.

So if every message that Bob signs is having this format. Now as Oscar has no control over the message bits so he need to do extra check to match the format so now lets see what is the probability of Oscar to arrive at the 124 bit 1's pattern.

if last LSB bit is 1 then what is the probability ? 50% ie 1/2 same we last 2 bits as 1 , 1/4 .....
last 124 bits as 1 => 2^-124 which is quite less and on an average Oscar need to choose 2^124 times or compute 2^124 times which is quite large as we know 2^80 with current computers takes years.



Look at the text book for real world padding. This is the basic idea.

As professor paar has already mentioned that we are not going to do any new algorithms from now onwards and we are going to use what ever we had learned so far.

So we will learn now about Elgamal Digital Signature.

Since January we talked about public key encryption, public key key-exchange and now we are learning Public Key DSA and as we know there are mainly three families of Public Key cryptography namely:

1. RSA -> Integer Factorisation problem
2. Discrete Logarithm Problem
3. Elliptical Cryptography

Why RSA is Integer Factorisation problem ?
Because if you really want to break RSA then you need to compute phi and phi is quite tough if we don't know about prime factorisation of "N" so hardness is based on "phi of n" which is indirectly saying depends on Prime Factorisation.

In this lecture we are going to study about Discrete Logarithm based schema called **Elgamal Digital Signature:**

All public Key schemes have Setup phase so in case of Elgamal also we will have the same.

Alice Bob

As this is a Discrete Logarithm problem so we need

domain parameters namely large prime P and Alpha

Question is what is a discrete logarithm problem ?
Y = alpha^X mod P where X is private Key and Y is Public Key.

Random Kpr = d which is X in the example above.

Compute Beta which is Kpub.

(Beta, P and Alpha)
<————————————————

Important point is that RSA Digital Signature and Encryption algorithm are almost identical but Elgamal DSA is quite different that Elgamal Encryption.

In Elgamal DSA we are going to have 2 Public Keys. we are having Beta which is long term public Key but we have a temporary private Key/public key.

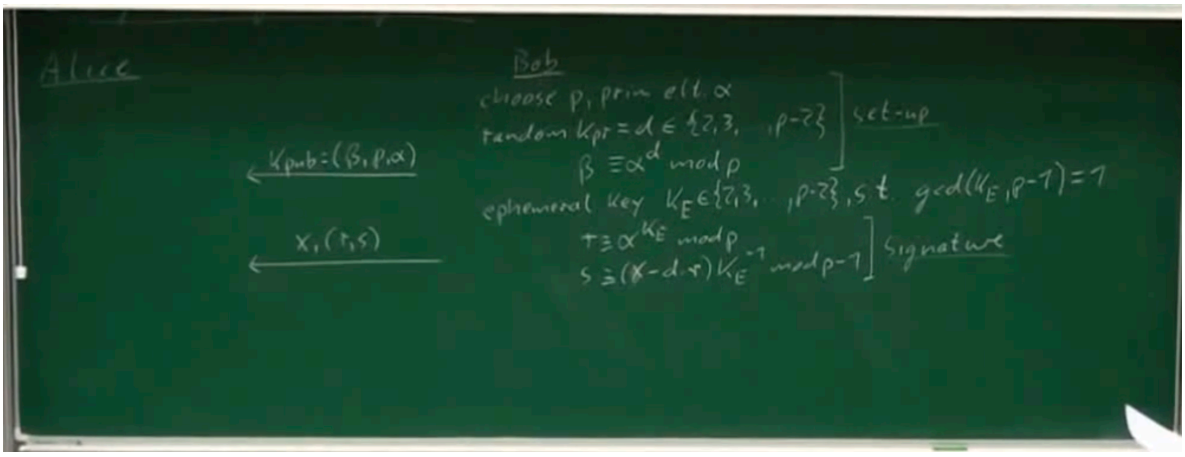
ephemeral Key, Ke take from {2,3, ….,P-1}

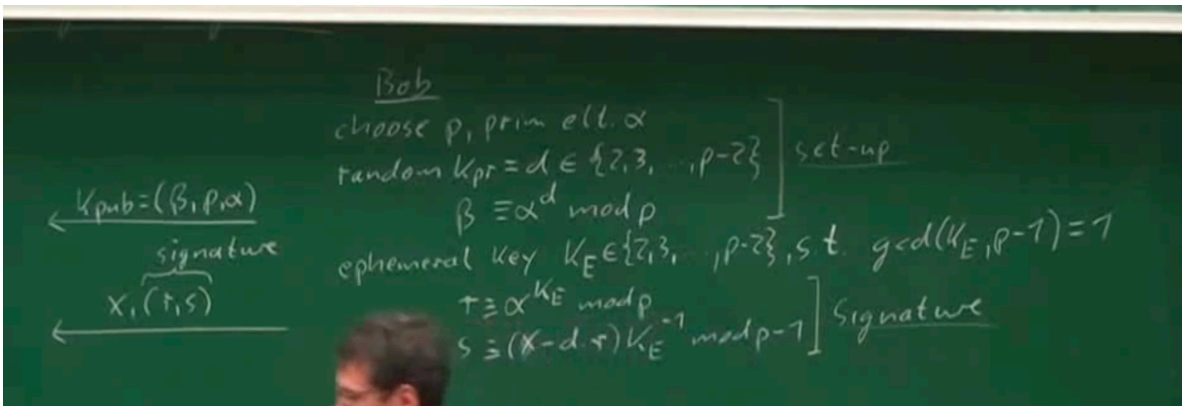
s.t. gcd(Ke, P-1} = 1
r = Alpha ^ Ke mod P

Question is why we need gcd(Ke, P-1) = 1 ? it is necessary condition for finding **inverse**.

X is message
Signature s = (X - d*r)* Ke^-1 mod P-1

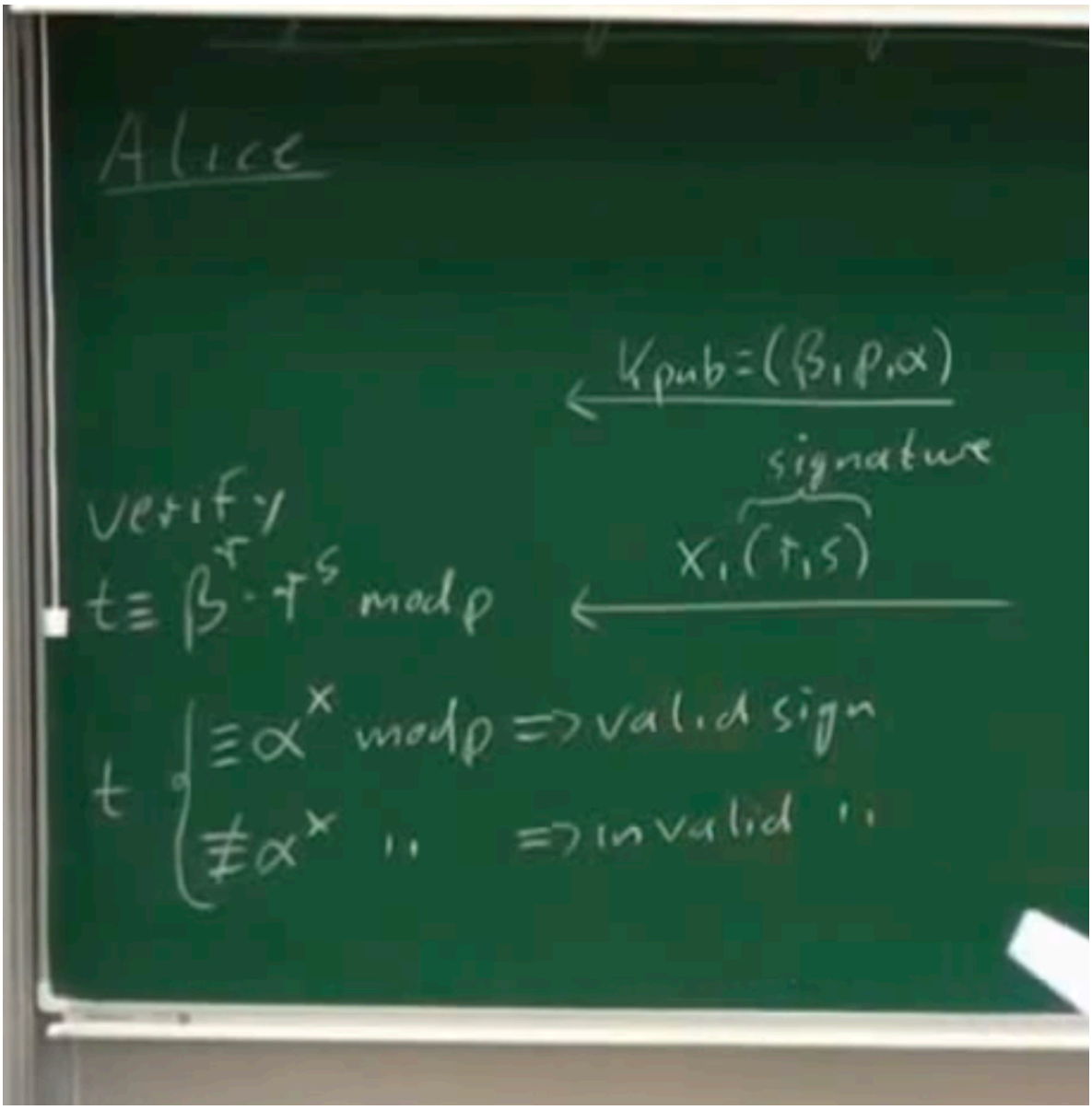
Signature in case of Elgamal is 2048 bit which is twice.
r + s combined is Signature.
<————————————————

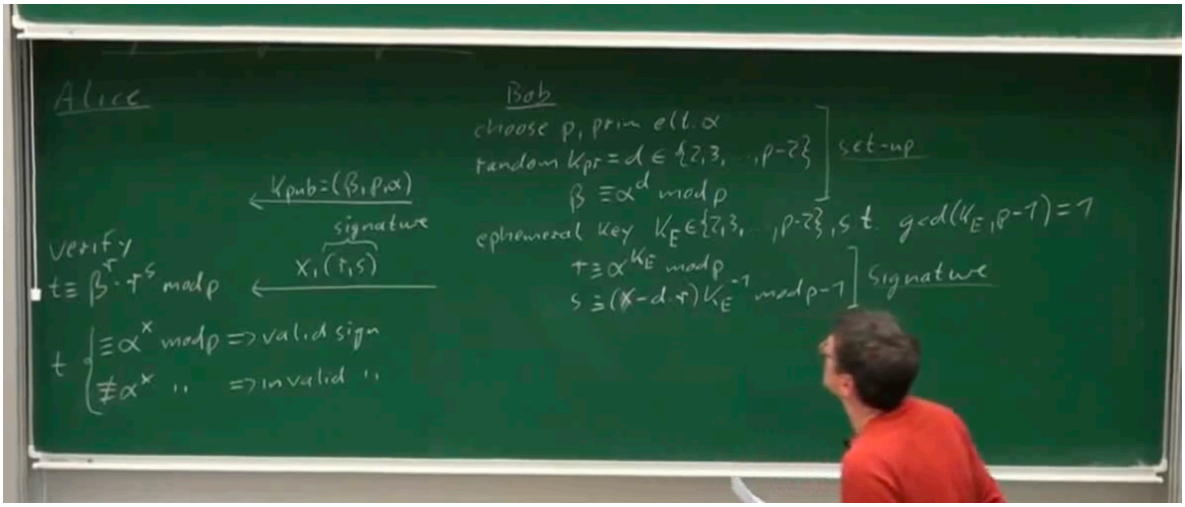




Now on receiving Signature and Message, alice has to verify.

So verification done at alice side is :





Above things are quite weird, so let take a look at
Proof of Correctness:

My way to prove:
Alice: (Beta^r)(r^s) mod p => ((Alpha^d)^r) *((Alpha^Ke)^s)mod p

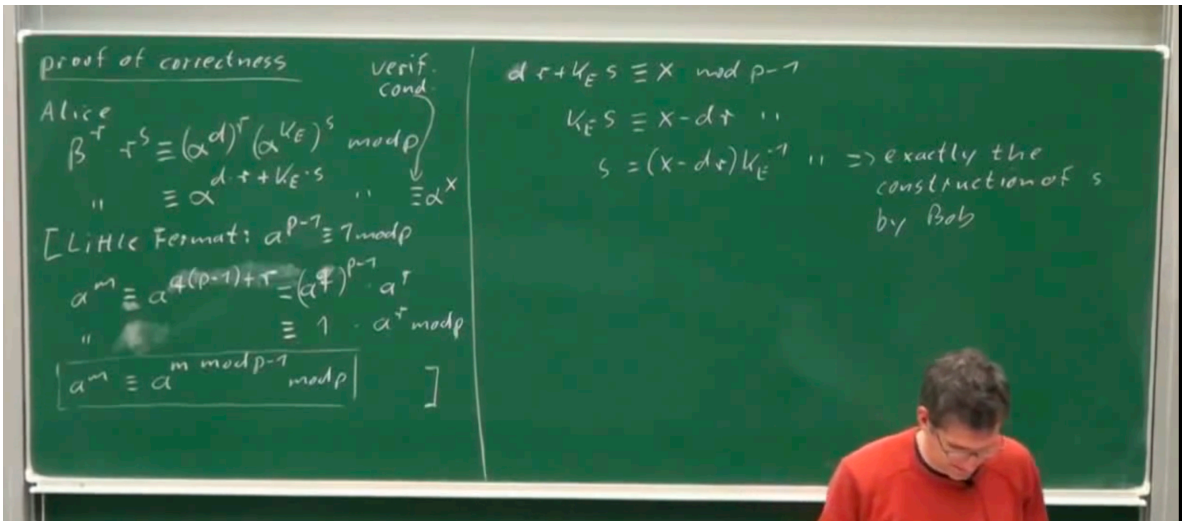
=> (Alpha ^ (d*r + Ke*s)) mod p
as we know s = (X - d*r)Ke^-1 mod p so replace s
(Alpha ^ (d*r + Ke*((X - d*r)Ke^-1))) mod p
===> Alpha ^ (d*r + X - d*r) mod p = Alpha^X mod p which is the verify equation.

Professor Paars way:
Alice: (Beta^r)(r^s) mod p => ((Alpha^d)^r) *((Alpha^Ke)^s)mod p
=> Alpha^(d*r + Ke*s) mod p
Fermat's Little theory:
a^p-1 = 1 mod p
so if we have "m" which is any integer then we can write
a^m mod p = a^(q*(p-1) + m mod p-1) mod p
ie we have written m in the from of p-1

=> (a ^ p-1)^q mod p * (a ^ m mod p-1) mod p
from fermats little theorm 1st part becomes 1 so
=> (a^m mod p-1) mod p

so final equation is
a^m = a^(m mod p-1) mod p
And as per Paar if we have p as prime and exponent then in exponent we can do arithmetic modulo p-1
Hence: m = m mod p-1
so from the above equation we will try to solve the equation:
 Alpha^(d*r + Ke*s) mod p = Alpha ^ X
d*r + Ke*s = X mod p-1
Ke^s = X-d*r mod p-1
s = (X-d*r)*Ke^-1 mod p-1

So s is same as what was computed by Bob. Hence Proved.



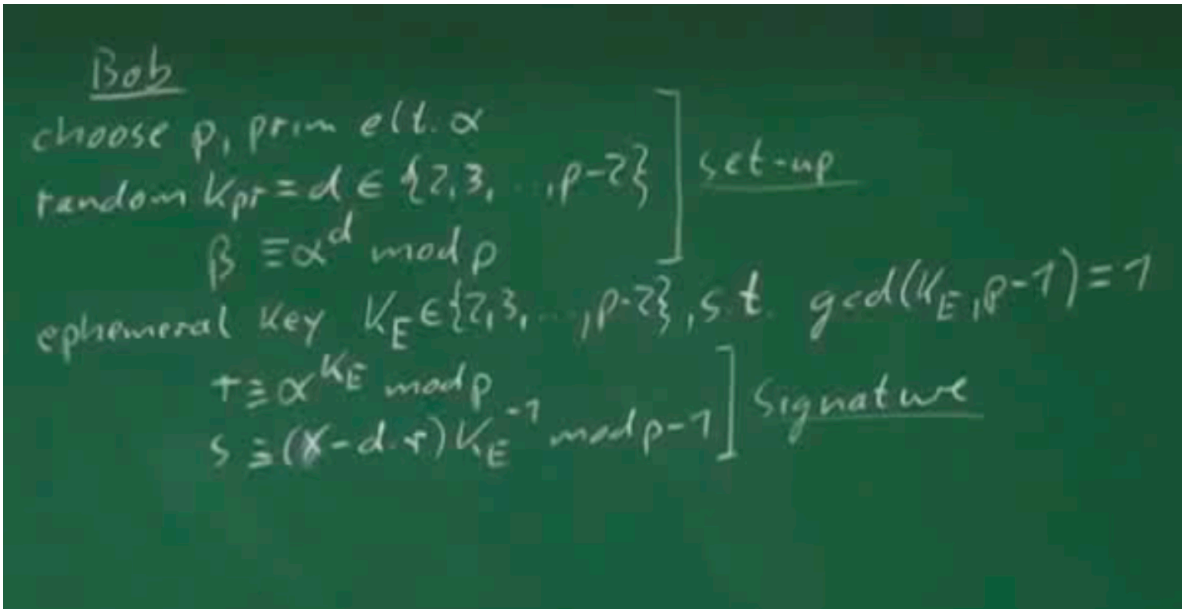
Practical remark:

1. If P is 2048 bits that means the block size or we can say X is 2048 bits then Signature is 4096 bits why ? because Signature in elgamal is (s,r) where s is 2048 bits and r is also 2048 bits. it might not be an issue in case of youtube /internet because millions of bits are transfered per second but if it is in credit card or smaller devices then it can be quite slow. — **Hashing is coming in my mind, let's see in upcoming lectures.**
2. Elgamal digital signature is based of DSA which is very popular Digital Signature Algorithm and DSA is very widely used.

Chapter 3: Weaknesses of Elgamal Digital Signature:
We are going to look at 2 attacks/weaknesses/pitfalls

Reuse of the Ephemeral Key:
As we know we are using temporary key and by reuse means we will use same key say twice to create Digital Signature.

Question: why some one do this kind of a thing?
Answer: If you look at below image:

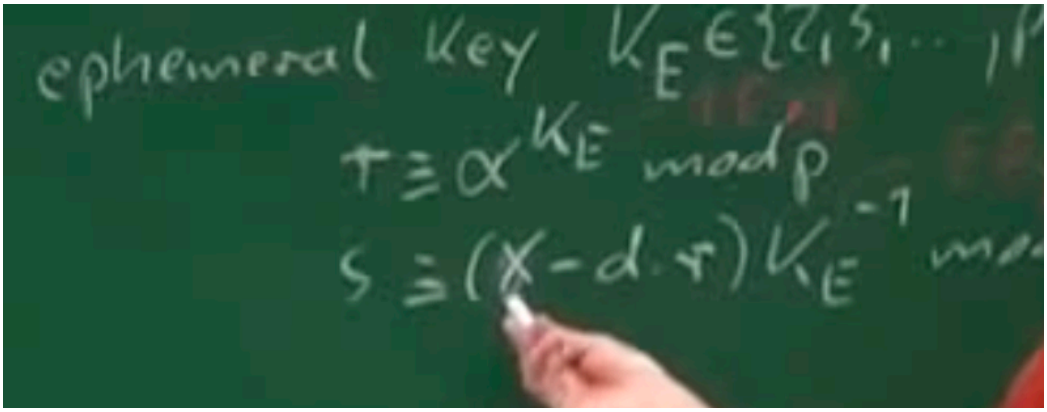


The d ie Kprivate choosen only once and it doesn't takes much time but what takes time is computation of exponent ie Beta but as we know that setup phase is done only once so it doesn't hurt.

Coming to Ephemeral Key as we know "**r**" is computed using exponent and that might be time consuming as we need to do this for each message and then if we come to "**S**" then here we need to compute Ke^-1, inverse which requires **EEA** which is also computationally time consuming so for saving computation time people might reuse the same Ke so that **"r" computation and Ke^-1 computations** are not required to be done again and again and other operations like Subtraction and multiplication are quite simple and doesn't impact/ very less impact the performance.

Setup phase doesn't require speed consideration but other phase os done again.
Question: How to compute the exponentiation ?
we can use Algorithm called **Square and Multiply algorithm**
if we use same ephemeral key then it is quite fast to compute signature but that is STUPID.

Question: Why is using same Ke twice is considered bad ?
Assume say Bob has used same Ke for message X1 and X2 then we know that "r" will be same for both the messages but S is different because S is computed using arithmetic which involves Message ie X1 and X2.



What oscar sees is X1, r, S1 for first message, and X2, r, S2 for second message.

Because of same r parameter Oscar can finds that Ke is same for both the messages as r is generated from S.

what will be the signature equations?

S1 = (X1 - d*r)Ke^-1 mod p-1
S2 = (X2 - d*r)Ke^-1 mod p-1

From above equations question is why **r** is same and **d** is same in both the equations ?
Answer is because **r** is computed as "**alpha^Ke mod P -1**" and as Ke is same so r is also same and d is private key which is generated in setup phase and it doesn't change in messages as it is generated once.

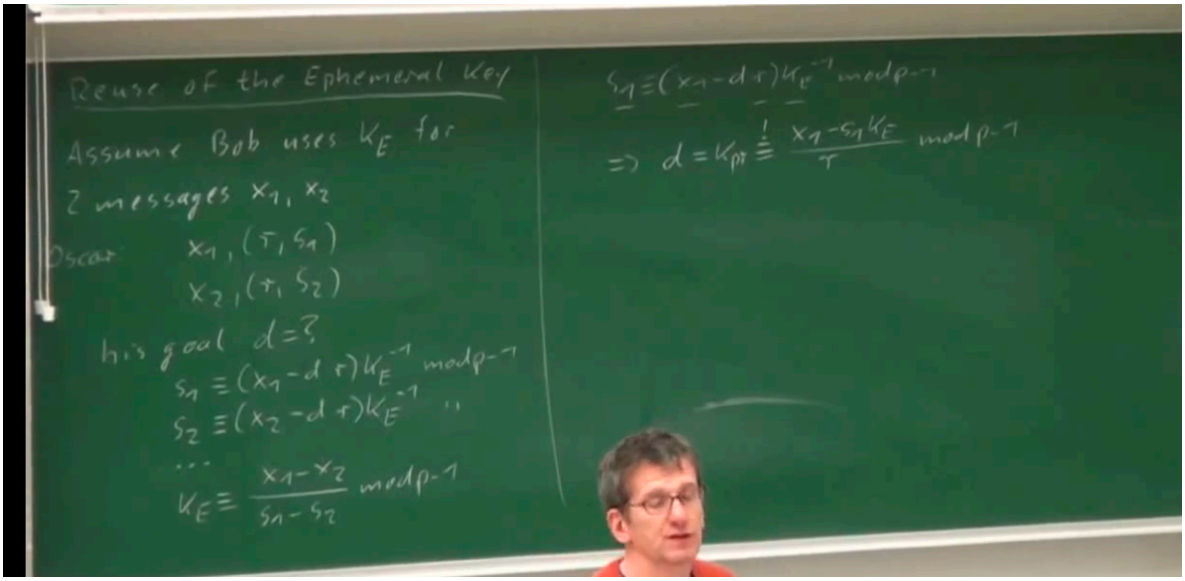
So now if we try to solve both the equations we can simplify the equations like:

1. **Ke*S1 = X1 - d*r mod p-1**
2. **Ke*S2 = X2 - d*r mod p-1**

if we subtract 2 from 1 we get Ke(S1 - S2) = X1 - X2
so what is unknown ? unknown is Ke so we can easily find Ke ?

Now question is why are we solving above equations, What we want to finally find ?
Because we want to find the private key which is d and if we find d then we can generate signatures and others will surely trust them.

so now as we know Ke, S1, X1, r so by equation 1 we can find d too and then we get the private key and now we can easily impersonate the user.
Hence this is stupid idea to use the same Ephemeral Key twice.



one he has private key he can generate message and also digital signature. Ie once he has d, he can do anything. **he can impersonate Bob.**
Remark: Don't reuse Ephemeral Key.

Existential Forgery attack on Elgamal Digital Signature:

Alice	**Oscar**	**Bob**
		Kpub = (P, Alpha, Beta) Kpr = d
	Kpub ie (P, Alpha, Beta) <———————————	

Oscar's goal is generate valid pair of **X and signature (r, S)** so that the verify method works properly.

so oscar

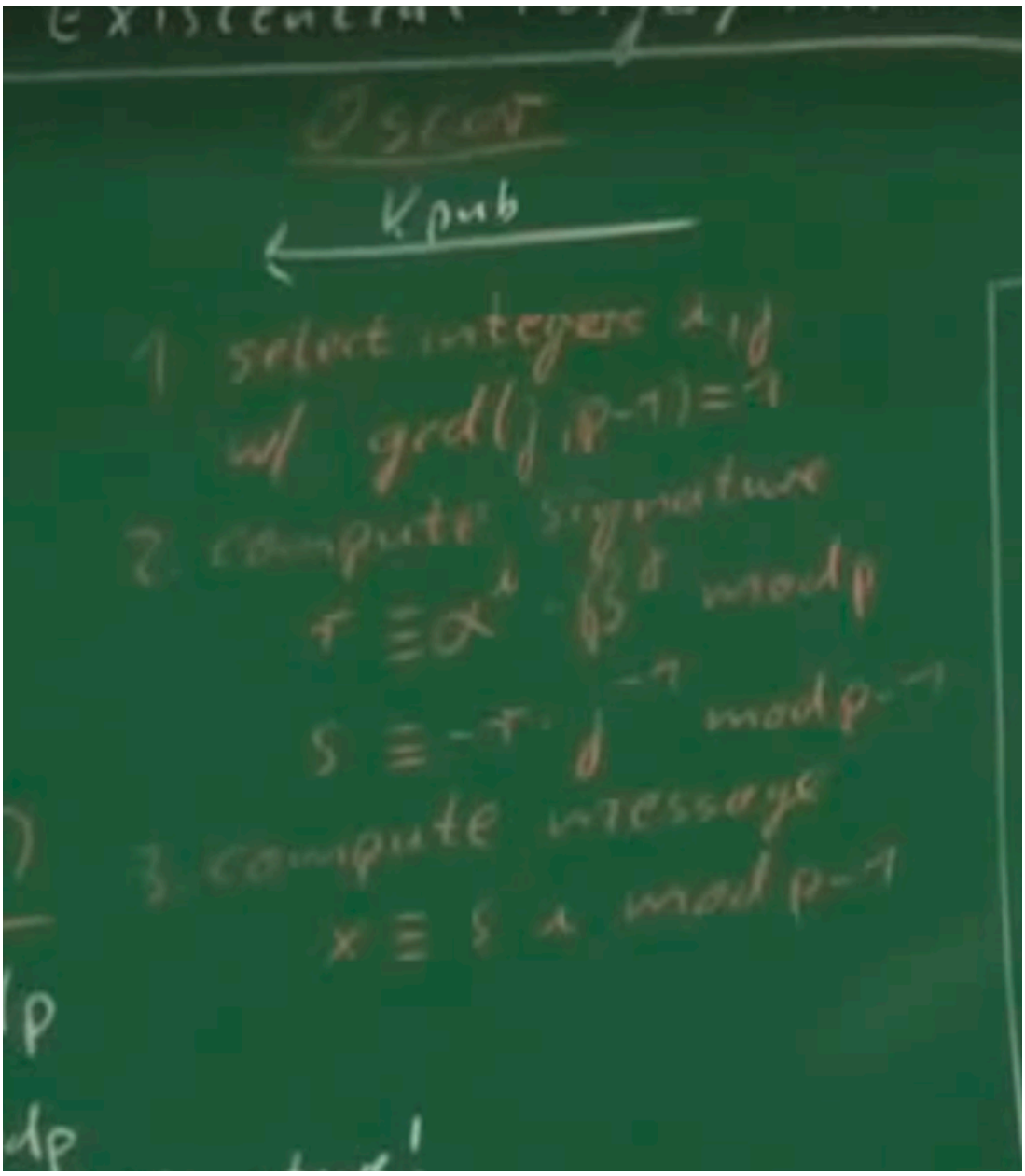
1. **selects 2 integers, i, j where gcd(j, p-1) = 1** , hence we got that in the way we need to invert j at some place.
2. Computes Signature:
    1. r = (Alpha^i)*(Beta^j) mod p
    2. s = -r*j^-1 mod p-1

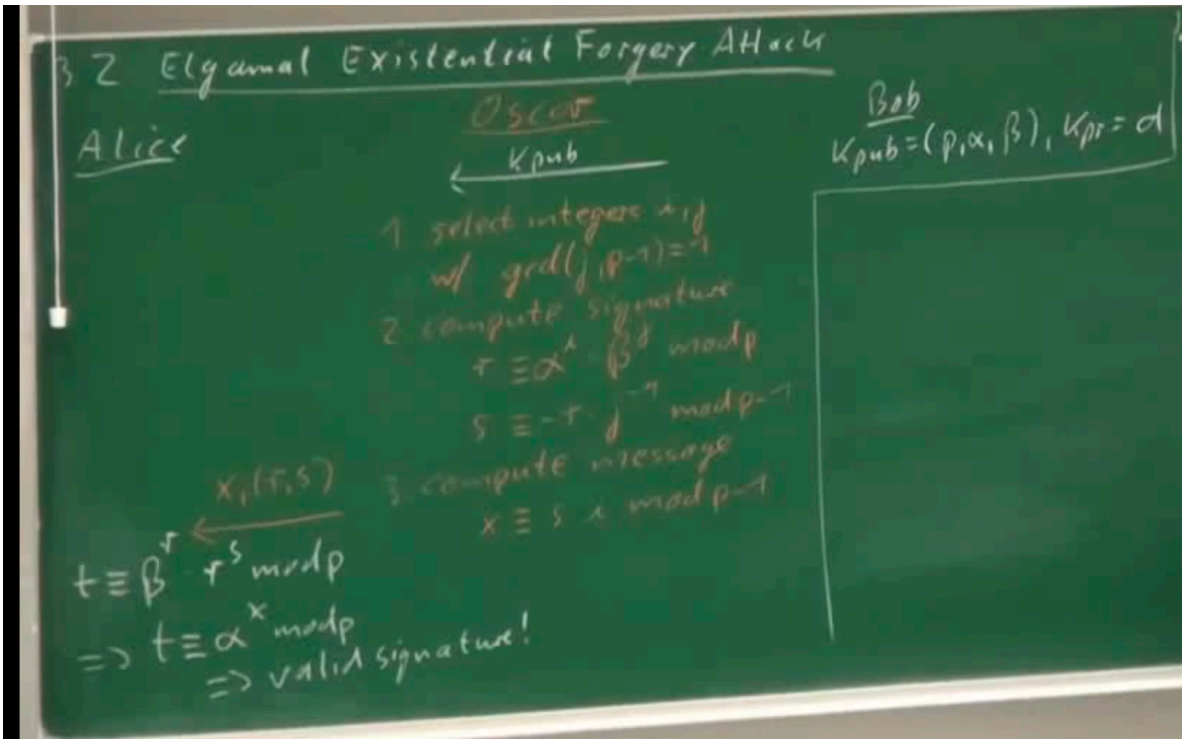
This complicated method we are doing so that verification process works properly.

Now what we have done in RSA we will try to do the same with Elgamal also ie we try to compute X from "Signature" which is not what Bob has done where bob has computed signature from X (message) and here we are generating message from signature.

3. Compute the message:
    1. X = s*i mod p-1

Now Alice needs to verify this entire information so alice computes (**Beta^r) * (r^s) mod p = t**
which comes out to be equal to **Alpha ^ X mod p** so alice concludes that this is a valid signature.





Proof of correctness:

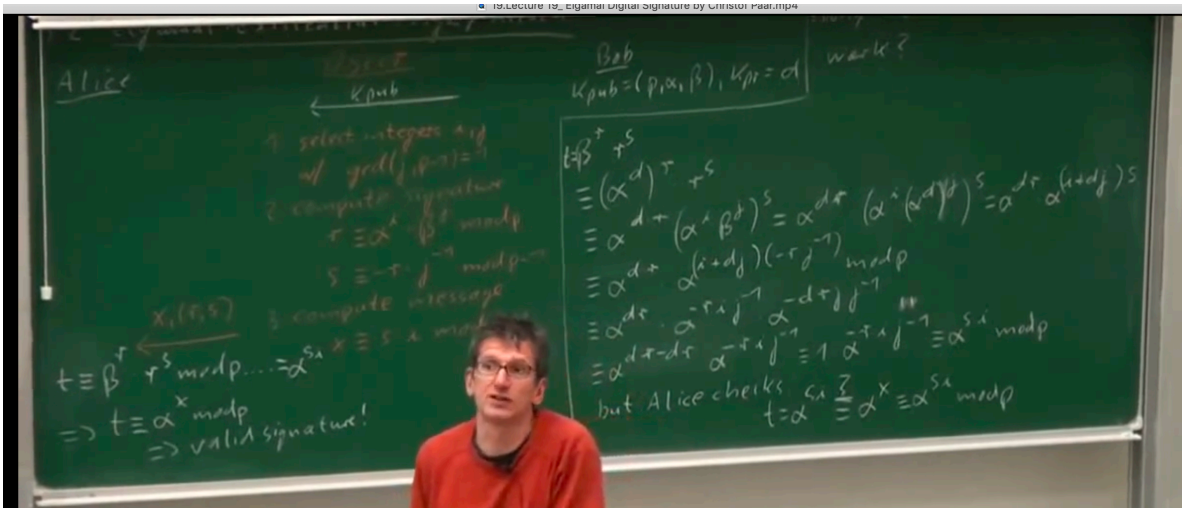
My proof:
Beta^r*r^s = Alpha^x
Beta^r*((Alpha^i)*(Beta^j))^s = Alpha^x

Beta^r*(Alpha^(i*s))*(Beta^(j*s)) = Alpha^x
Beta^(r + j*s)*Alpha^(i*s) = Alpha^x

Now opening rhs

Beta^(r + j*s)*Alpha^(i*s) = Alpha^(s*i)

it seems from here that **Beta^(r + j*s)** this should be 1.
lets open it
Beta ^ (r + j*(-r*j^-1)) => Beta ^ (r - r) => 1
H.P



Remark: I am not sure why Existential forgery attack is quite tough and how does they reach at this way of taking “i and j”. Need to look at it again.