Asymmetric/Public key cryptography

Lecture 5a

Overview

- Revisiting: intro to asymmetric crypto and key change issue
- Applications of asymmetric cryptography
- Key maths concept in asymmetric cryptography
- RSA cipher

  1) General process

  2) Examples: example01 and example02

  3) Security issues with RSA

  4) Timing attacks

Diffie Hellman Exchange

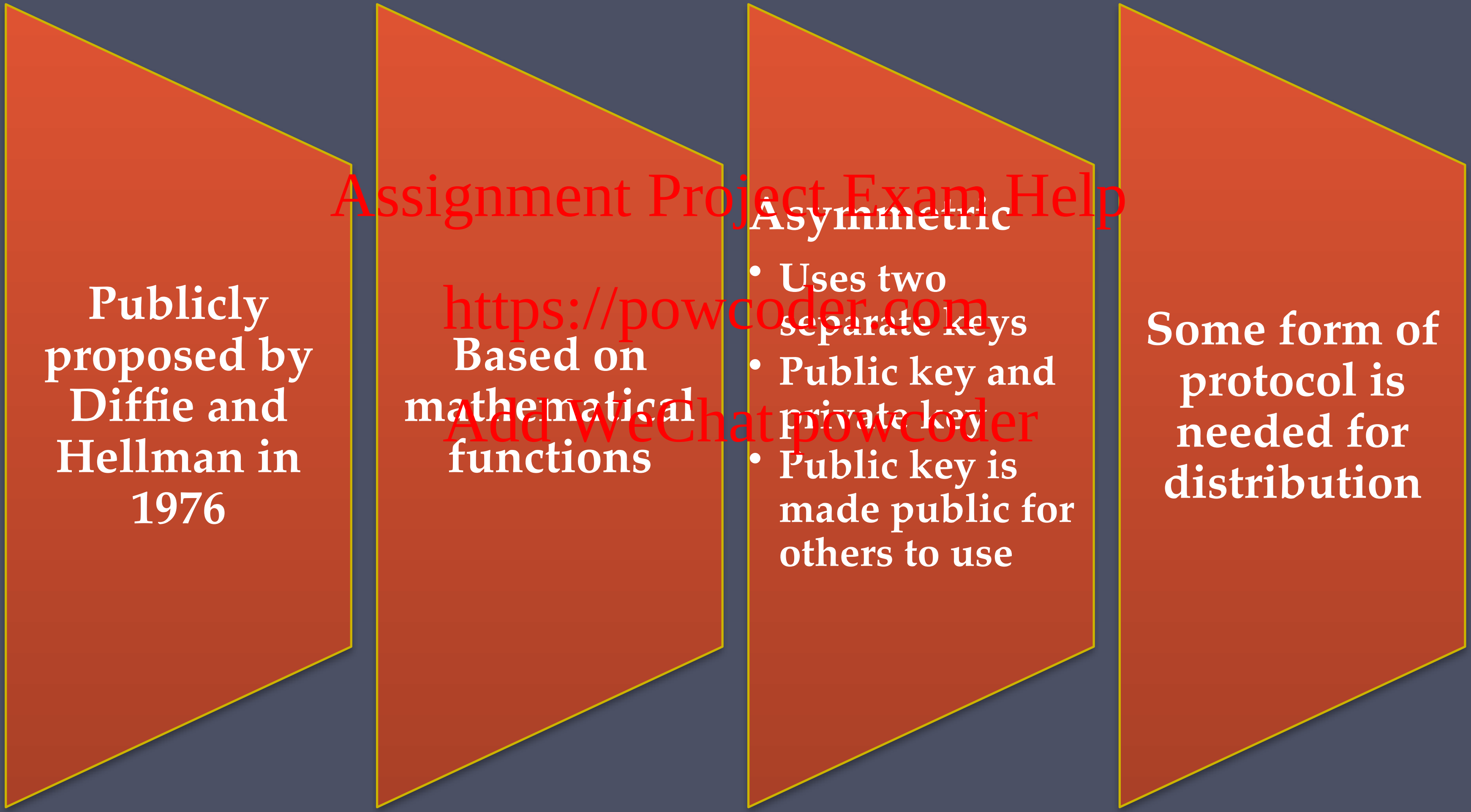
  1) Intro

  2) General process

  3) Examples

  4) Security issues: Man-in-the-middle attack

Public-Key Encryption Structure



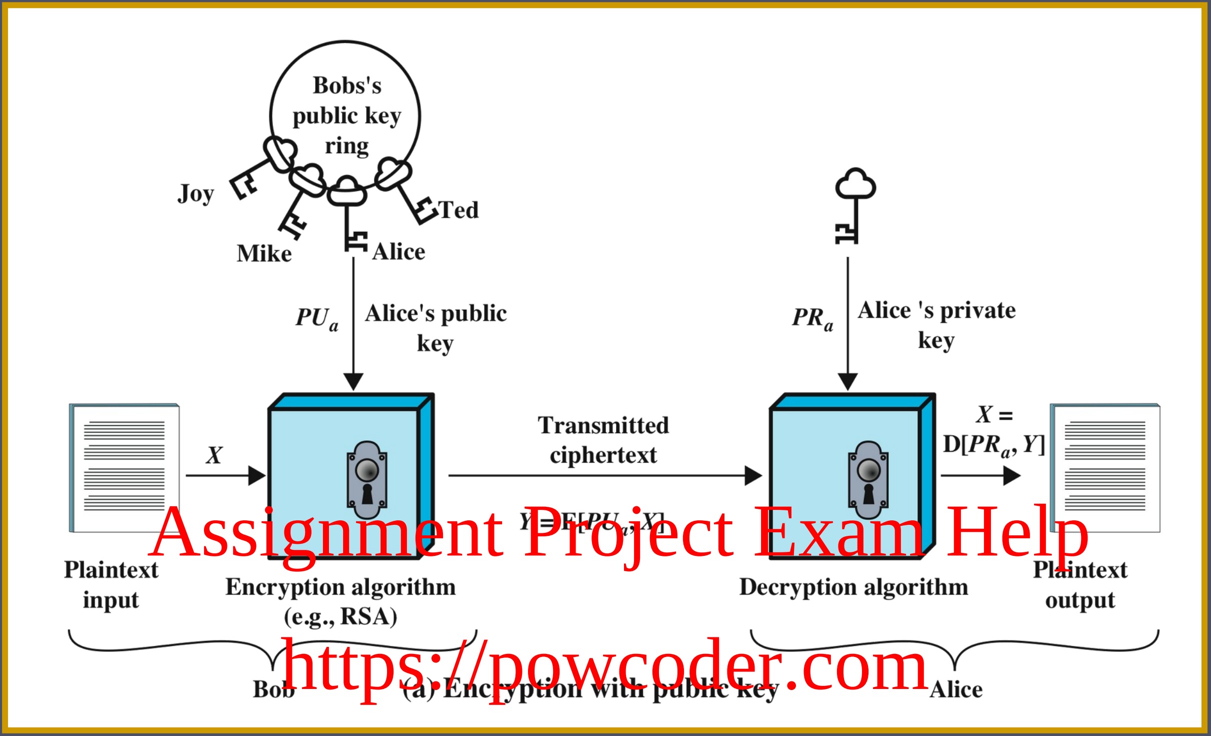
Publicly proposed by Diffie and Hellman in 1976

Based on mathematical functions

Asymmetric

- Uses two separate keys
- Public key and private key
- Public key is made public for others to use

Some form of protocol is needed for distribution



- **Plaintext**
  - Readable message or data that is fed into the algorithm as input
- **Encryption algorithm**
  - Performs transformations on the plaintext
- **Public and private key**
  - Pair of keys, one for encryption, one for decryption
- **Ciphertext**
  - Scrambled message produced as output
- **Decryption key**
  - Produces the original plaintext

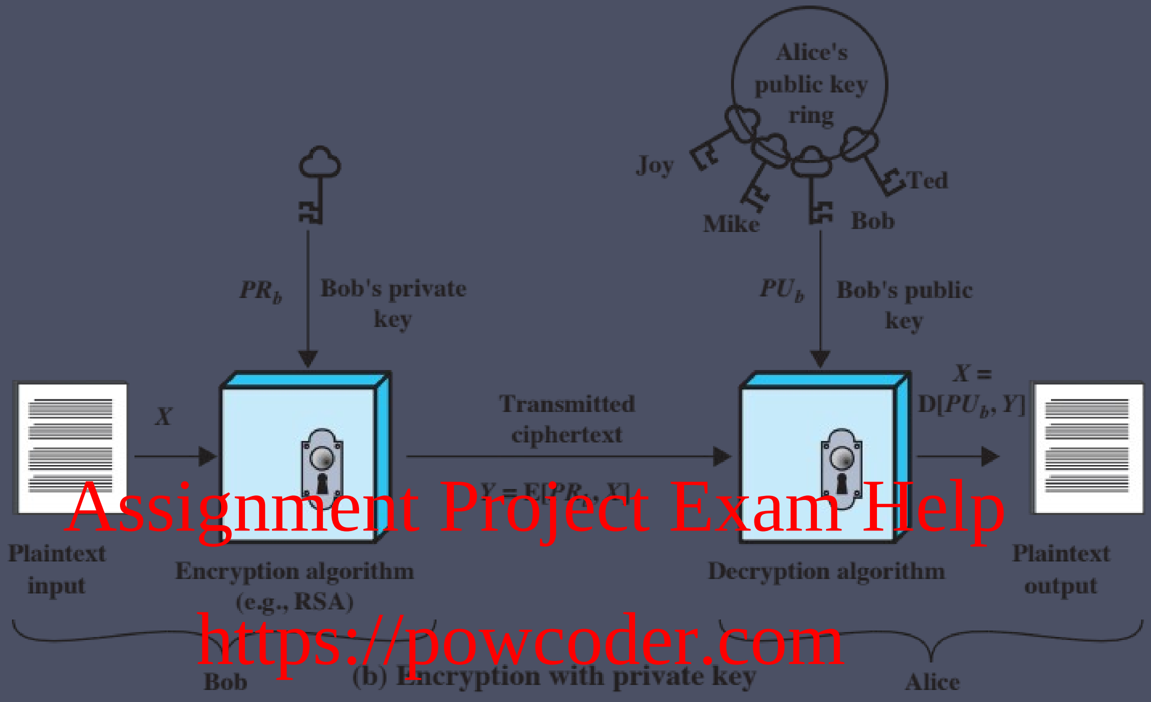


Figure 2.6 Public-Key Cryptography

- User encrypts data using his or her own private key

- Anyone who knows the corresponding public key will be able to decrypt the message

Requirements for Public-Key Cryptosystems

Computationally easy to create key pairs

Computationally easy for sender knowing public key to encrypt messages

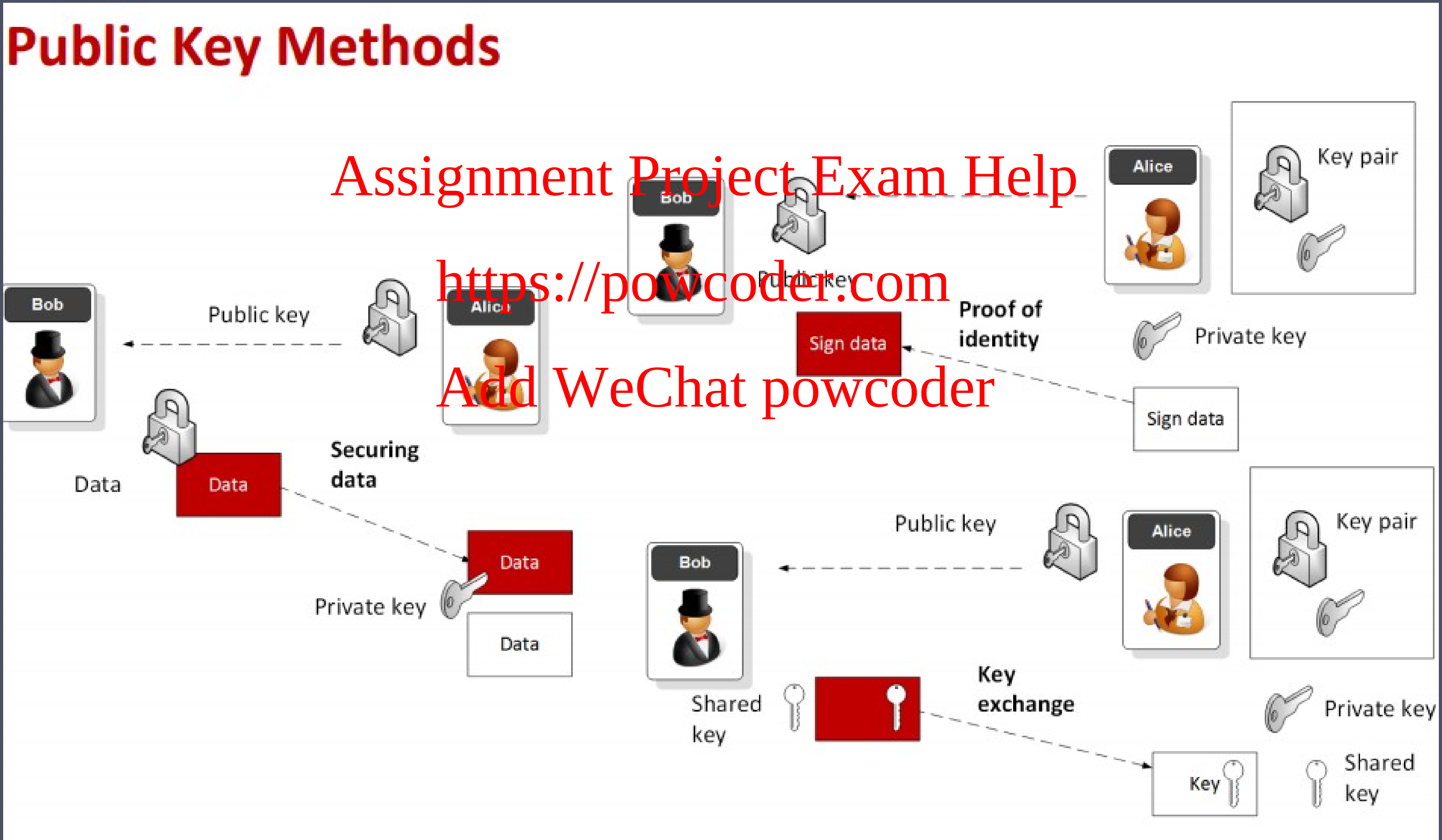
Computationally easy for receiver knowing private key to decrypt ciphertext

Computationally infeasible for opponent to determine private key from public key

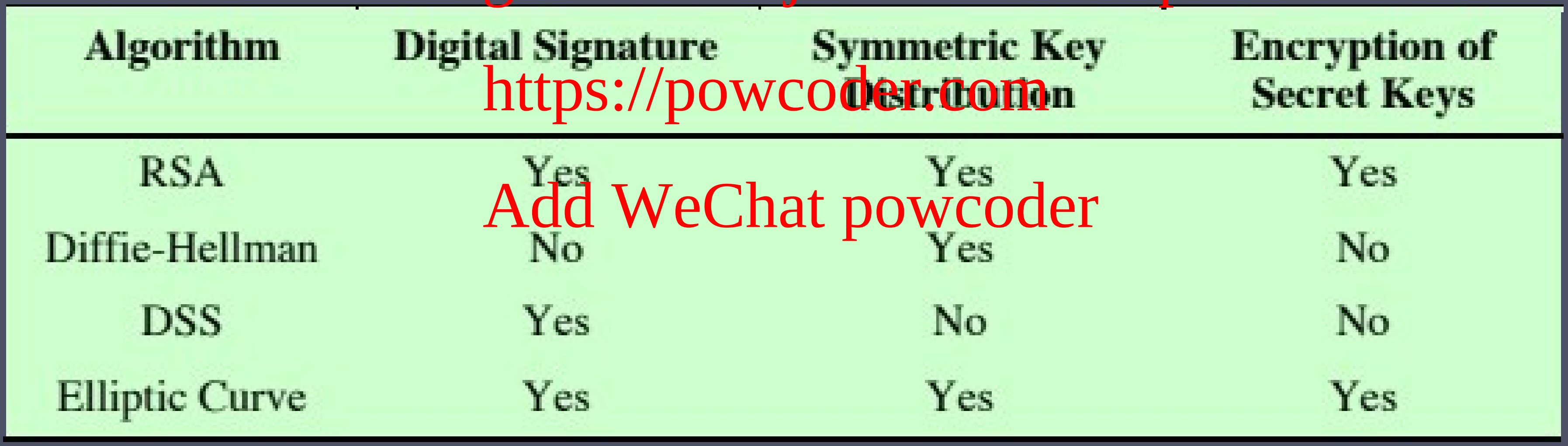
Computationally infeasible for opponent to otherwise recover original message

Useful if either key can be used for each role

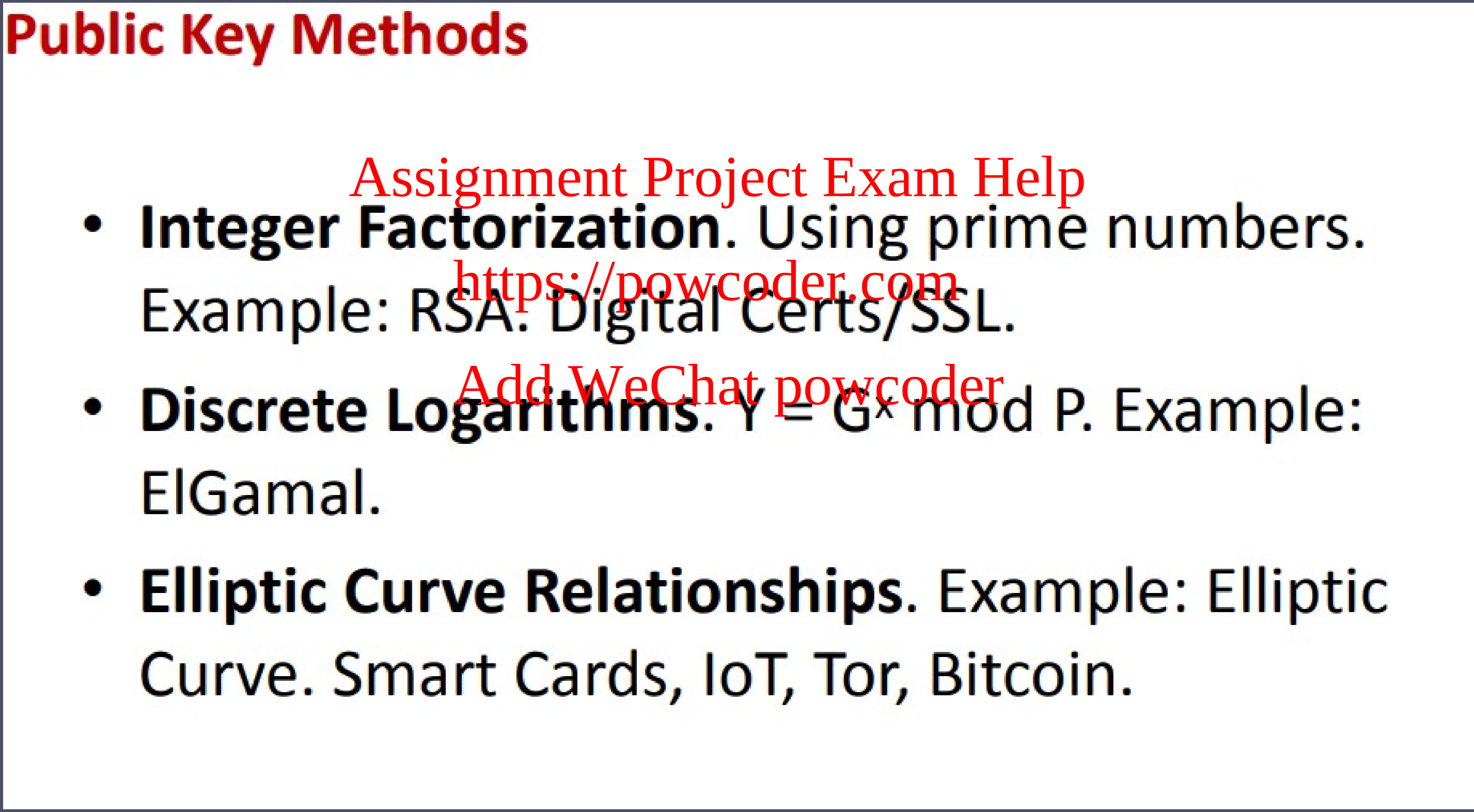
Public key methods



Applications for Public-Key Cryptosystems



Algorithm	Digital Signature	Symmetric Key Distribution	Encryption of Secret Keys
RSA	Yes	Yes	Yes
Diffie-Hellman	No	Yes	No
DSS	Yes	No	No
Elliptic Curve	Yes	Yes	Yes



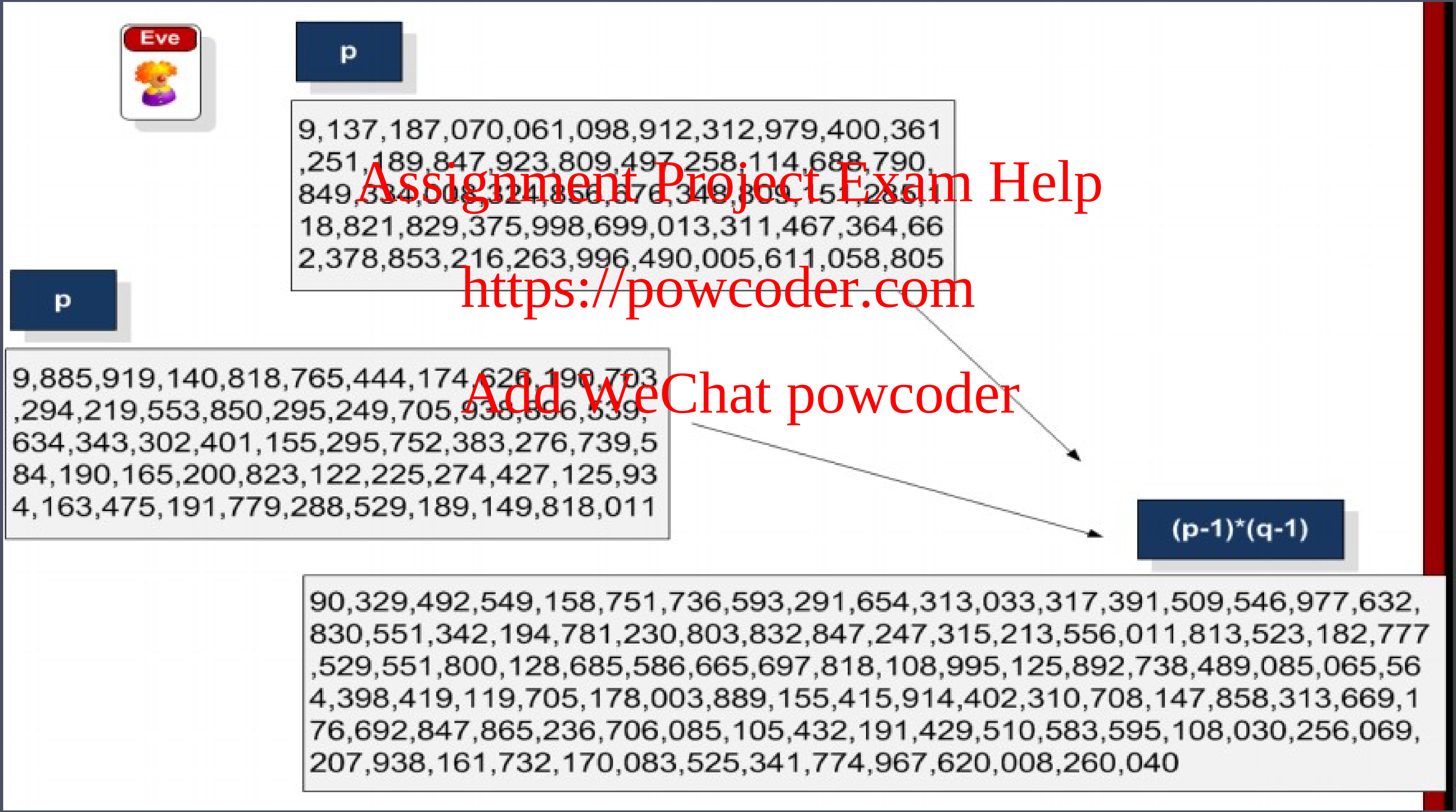
Public Key Methods

- **Integer Factorization**. Using prime numbers. Example: RSA. Digital Certs/SSL.
- **Discrete Logarithms**. Y = G^x mod P. Example: ElGamal.
- **Elliptic Curve Relationships**. Example: Elliptic Curve. Smart Cards, IoT, Tor, Bitcoin.

RSA Public-Key Encryption

- By Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key algorithm
- Uses exponentiation of integers modulo a prime
- Encrypt: $C = M^e \bmod n$
- *Decrypt:* $M = C^d \bmod n = (M^e)^d \bmod n = M$
- Both sender and receiver know values of n and e
- Only receiver knows value of d
- Public-key encryption algorithm with public key $PU = \{e, n\}$ and private key $PR = \{d, n\}$

P and q numbers in real



p

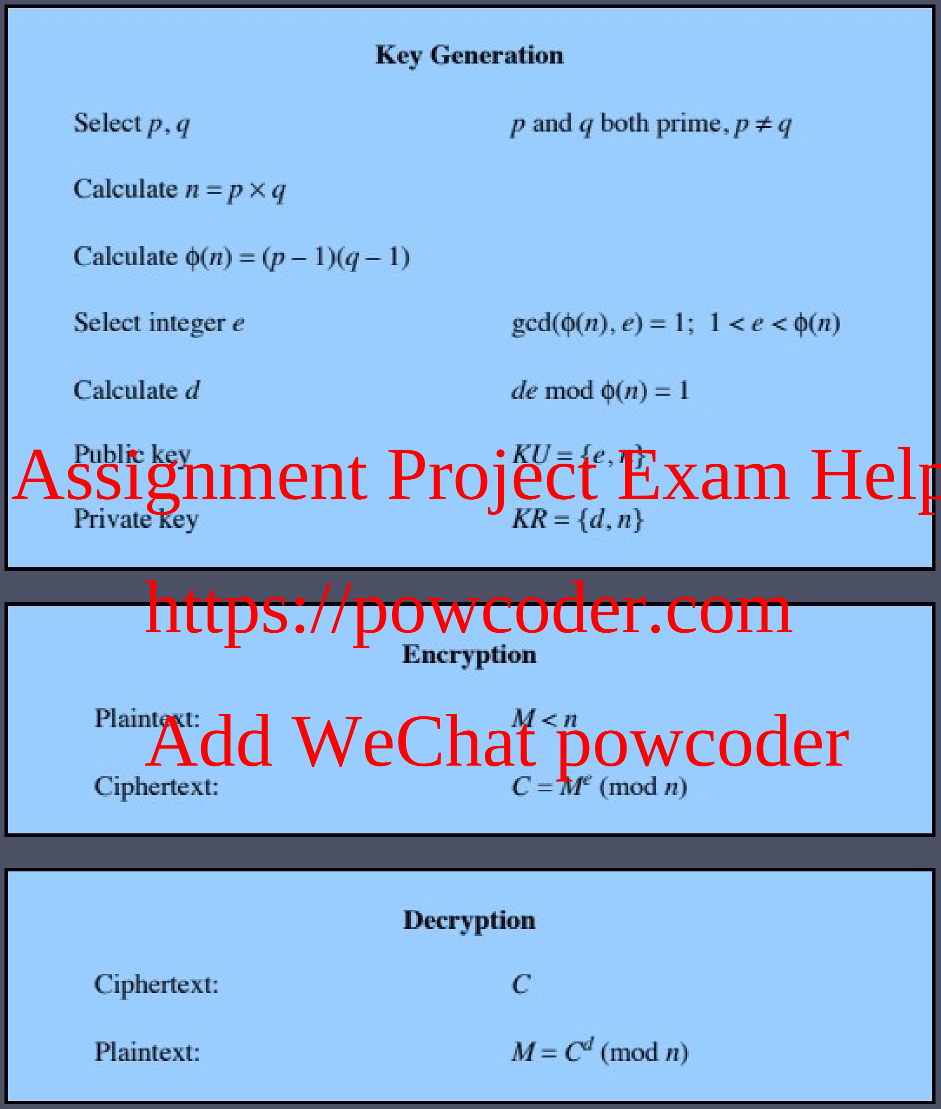
9,137,187,070,061,098,912,312,979,400,361
,251,189,847,923,809,497,258,114,688,790,
849,334,008,324,856,876,348,809,151,288,1
18,821,829,375,998,699,013,311,467,364,66
2,378,853,216,263,996,490,005,611,058,805

p

9,885,919,140,818,765,444,174,626,199,703
,294,219,553,850,295,249,705,938,696,539,
634,343,302,401,155,295,752,383,276,739,5
84,190,165,200,823,122,225,274,427,125,93
4,163,475,191,779,288,529,189,149,818,011

(p-1)*(q-1)

90,329,492,549,158,751,736,593,291,654,313,033,317,391,509,546,977,632,
830,551,342,194,781,230,803,832,847,247,315,213,556,011,813,523,182,777
,529,551,800,128,685,586,665,697,818,108,995,125,892,738,489,085,065,56
4,398,419,119,705,178,003,889,155,415,914,402,310,708,147,858,313,669,1
76,692,847,865,236,706,085,105,432,191,429,510,583,595,108,030,256,069,
207,938,161,732,170,083,525,341,774,967,620,008,260,040



Key Generation

Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$de \bmod \phi(n) = 1$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption

Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption

Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 21.7 The RSA Algorithm

RSA – example01

Encryption	Decryption
Public key: (5,14) Plaintext: B → 2 index C= mod N (mod) 14 = 32 (mod 14) = 4 (mod) 14 = D = 4 index	Private key (11, 14) Note: 14 is the same Ciphertext: D→ 4 M= mod N (mod)14 = 4194304 (mod 14) = 2 (mod 14) = B = 2 index

How does it work?

1st step: two primes number p and q

p=2 and q=7

2nd step: product of p and q = p x q = 14 = N

which is mod in public and private key, it is publicise

3rd step: (pronounced as PHI(N) = (p-1)(q-1)

=(2-1)(7-1)

= 6 = total number of co-prime

4th step: Choose e 1< e < (N) = 2,3,4,5

{ co-prime with N, (N) = 2,3.4.5

N=14, (N)=6;

public key = 5, 14

5th step: choose d: de (mod (N)) = 1

5d (mod 6) = 1

d should be such a number that when it multiplies with 5 and find mod by 6, it should give you 1

Coprime
1=1x1 3=3x1 5=5x1 9=3x3 11=11x1 13=13x1

How many coprime below 14?	
14=2x7	2=2x1 4=2x2 6=3x2 8=2x2x2 12=2x2x3
14=2x7	1=1x1 3=3x1 5=5x1 7=7x1 9=3x3 11=11x1 13=13x1

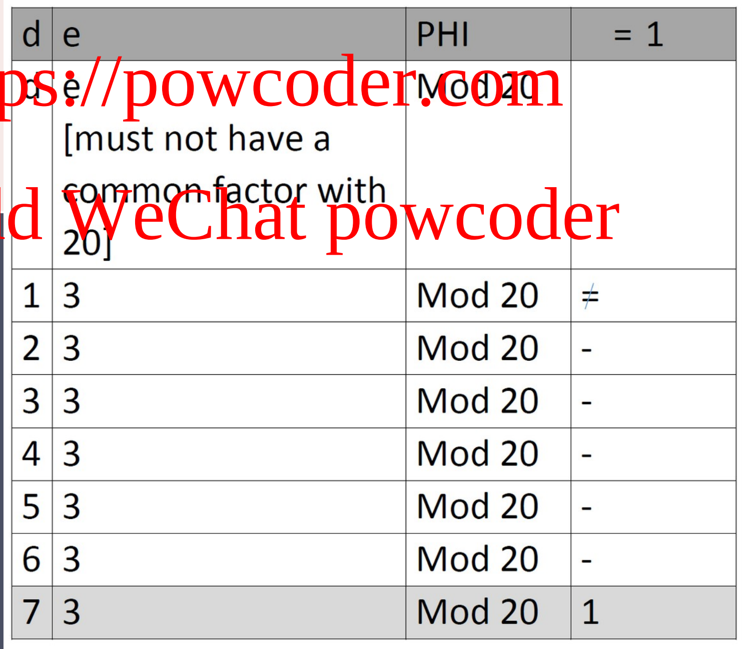
1
2
3
4
5
6
7
8
9
10
11
12
13
14

d	1	2	3	4	5	…..
5d	5	10	15	20	25	……
mod 6	5	4	3	2	1	0

This pattern repeat, pick any number that give you mod 1

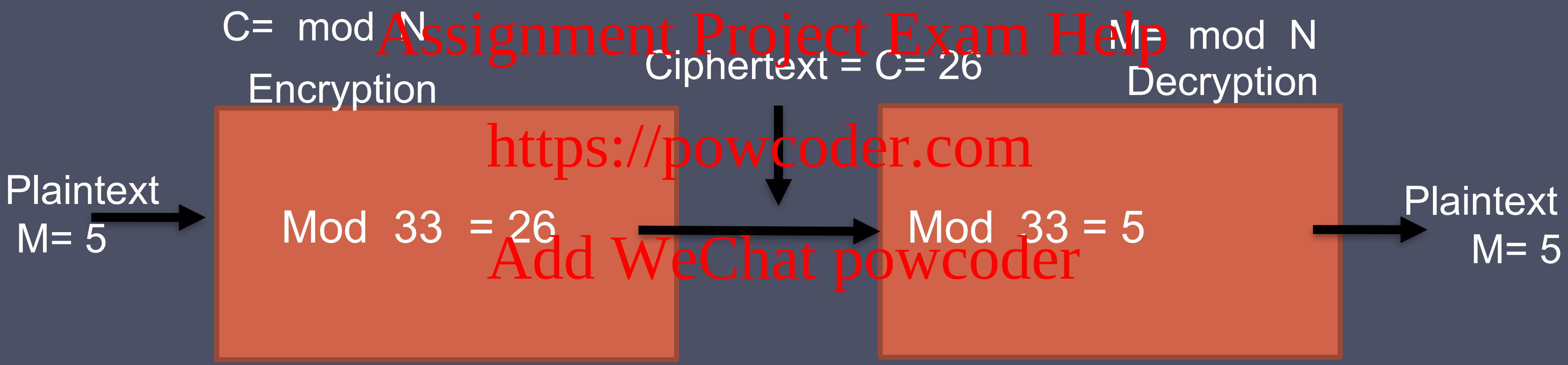
Example02

Encryption	Decryption
two primes p x q ; p=3, p=11 N = p x q = 3 x 11 = 33 (N) = (p-1)(q-1) = (3-1) (11-1) = 2 x 10 = 20 [this will be our mod] = Both parties will have this value	
Selecting e 1< e < (N) = 1<e<20 { co-prime with N, (N) e=3 public key = [3, 33]	(d x e) mod (N) = 1 (d x 3) mod 20 = 1



d	e	PHI	= 1
d	e [must not have a common factor with 20]	Mod 20	
1	3	Mod 20	≠
2	3	Mod 20	-
3	3	Mod 20	-
4	3	Mod 20	-
5	3	Mod 20	-
6	3	Mod 20	-
7	3	Mod 20	1

d = 7



C= mod N

Encryption

Ciphertext = C= 26

M= mod N

Decryption

Plaintext

M= 5

Mod 33 = 26

Mod 33 = 5

Plaintext

M= 5

Security of RSA

Brute force

- Involves trying all possible private keys

Mathematical attacks

- There are several approaches, all equivalent in effort to factoring the product of two primes

Timing attacks

- These depend on the running time of the decryption algorithm

Chosen ciphertext attacks

- This type of attack exploits properties of the RSA algorithm

Timing Attacks

- Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages
- Timing attacks are applicable not just to RSA, but also to other public-key cryptography systems
- This attack is alarming for two reasons:
  - It comes from a completely unexpected direction
  - It is a ciphertext-only attack

Timing Attack Countermeasures

Constant exponentiation time

- Ensure that all exponentiations take the same amount of time before returning a result
- This is a simple fix but does degrade performance

Random delay

- Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack
- If defenders do not add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays

Blinding

- Multiply the ciphertext by a random number before performing exponentiation
- This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack

Diffie-Hellman Key Exchange

- First published public-key algorithm
- By Diffie and Hellman in 1976 along with the exposition of public key concepts
- Used in a number of commercial products
- Practical method to exchange a secret key securely that can then be used for subsequent encryption of messages
- Security relies on difficulty of computing discrete logarithms

Global Public Elements

q	prime number
α	$\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

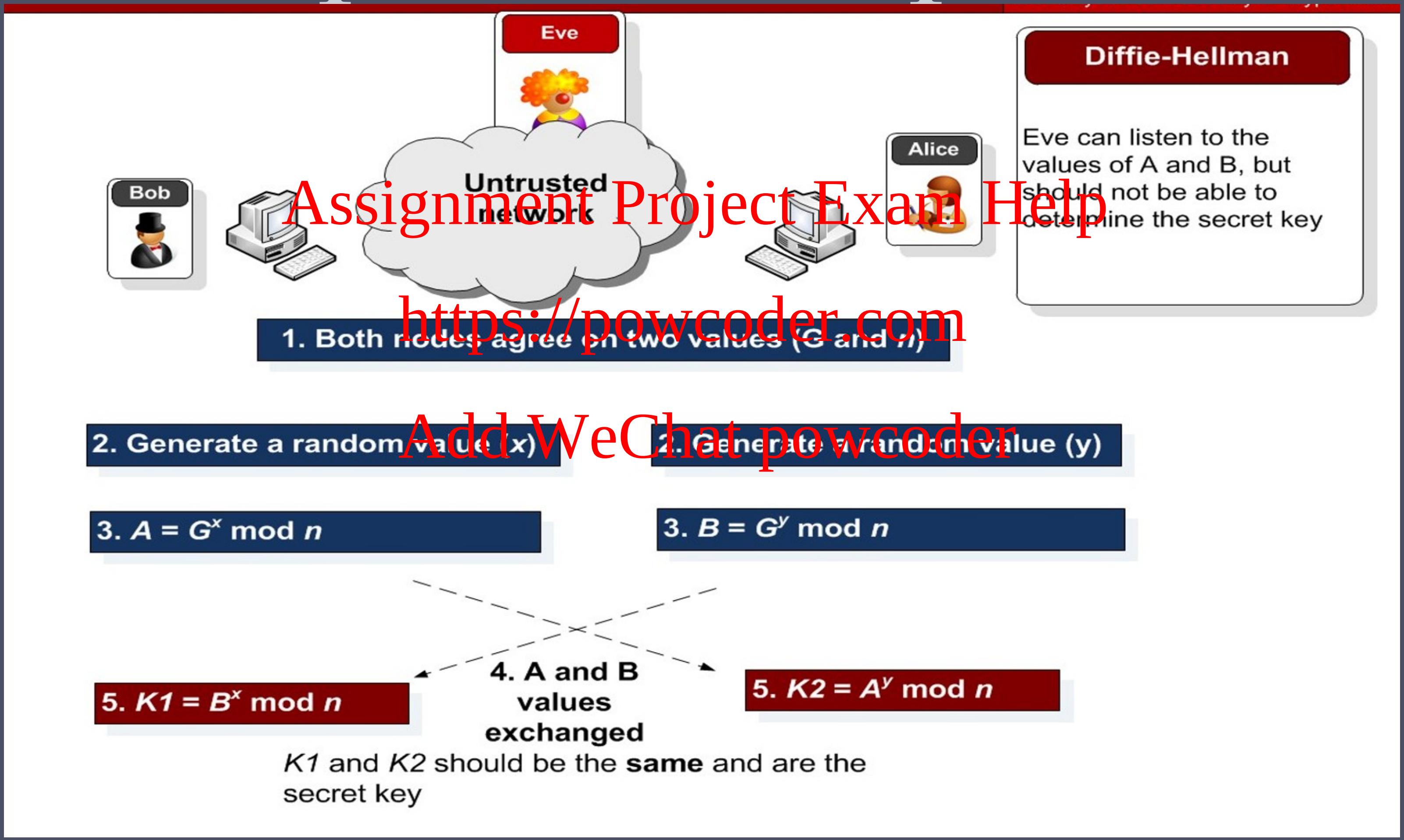
$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 21.9 The Diffie-Hellman Key Exchange Algorithm

Example of DH protocol



Diffie-Hellman

Eve can listen to the values of A and B, but should not be able to determine the secret key

1. Both nodes agree on two values (*G* and *n*)

2. Generate a random value (*x*)

2. Generate a random value (*y*)

3. $A = G^x \bmod n$

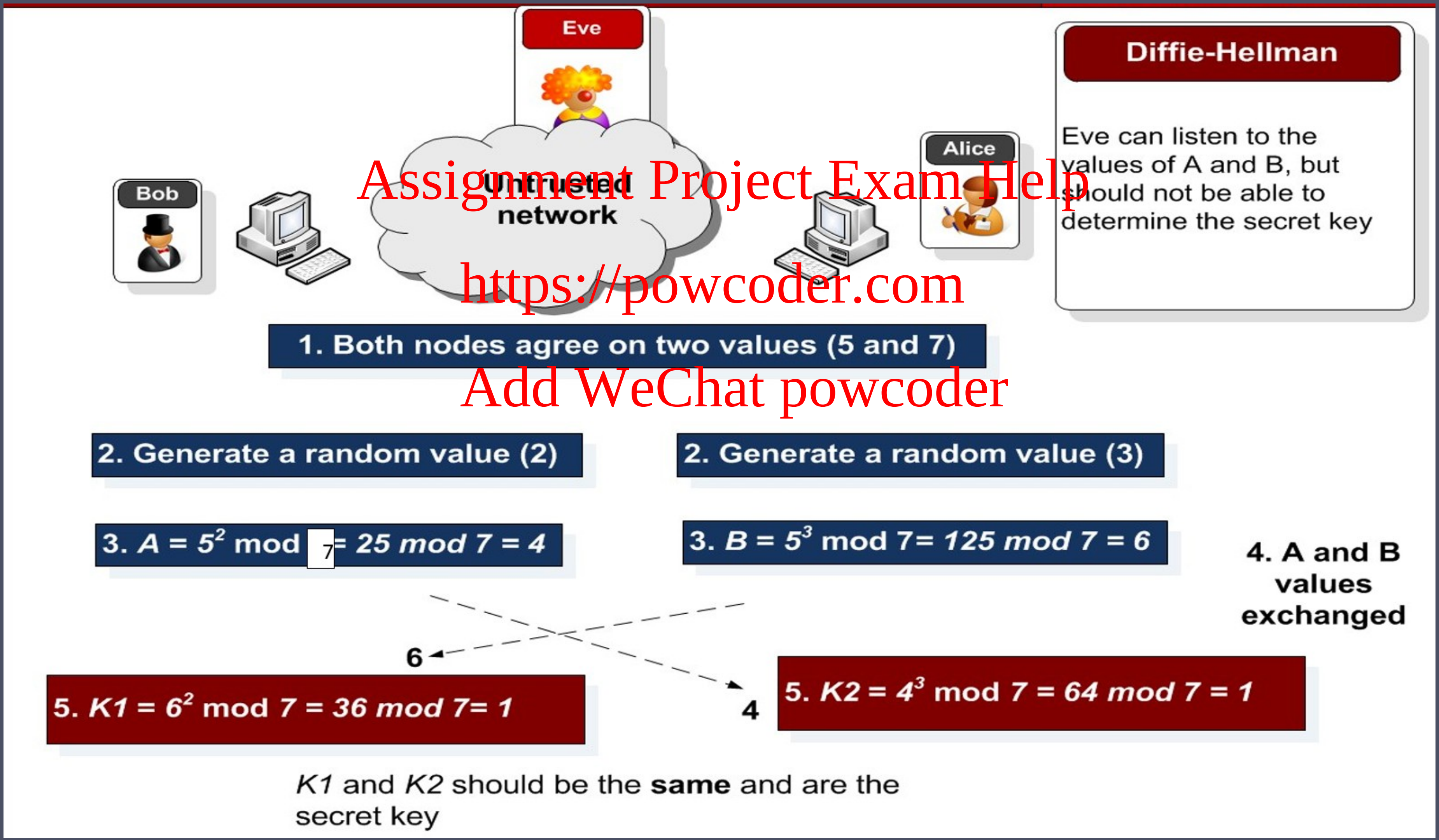
3. $B = G^y \bmod n$

4. A and B values exchanged

5. $K1 = B^x \bmod n$

5. $K2 = A^y \bmod n$

K1 and *K2* should be the **same** and are the secret key



Diffie-Hellman

Eve can listen to the values of A and B, but should not be able to determine the secret key

1. Both nodes agree on two values (5 and 7)

2. Generate a random value (2)

2. Generate a random value (3)

3. $A = 5^2$ mod 7 $= 25$ mod $7 = 4$

3. $B = 5^3$ mod $7 = 125$ mod $7 = 6$

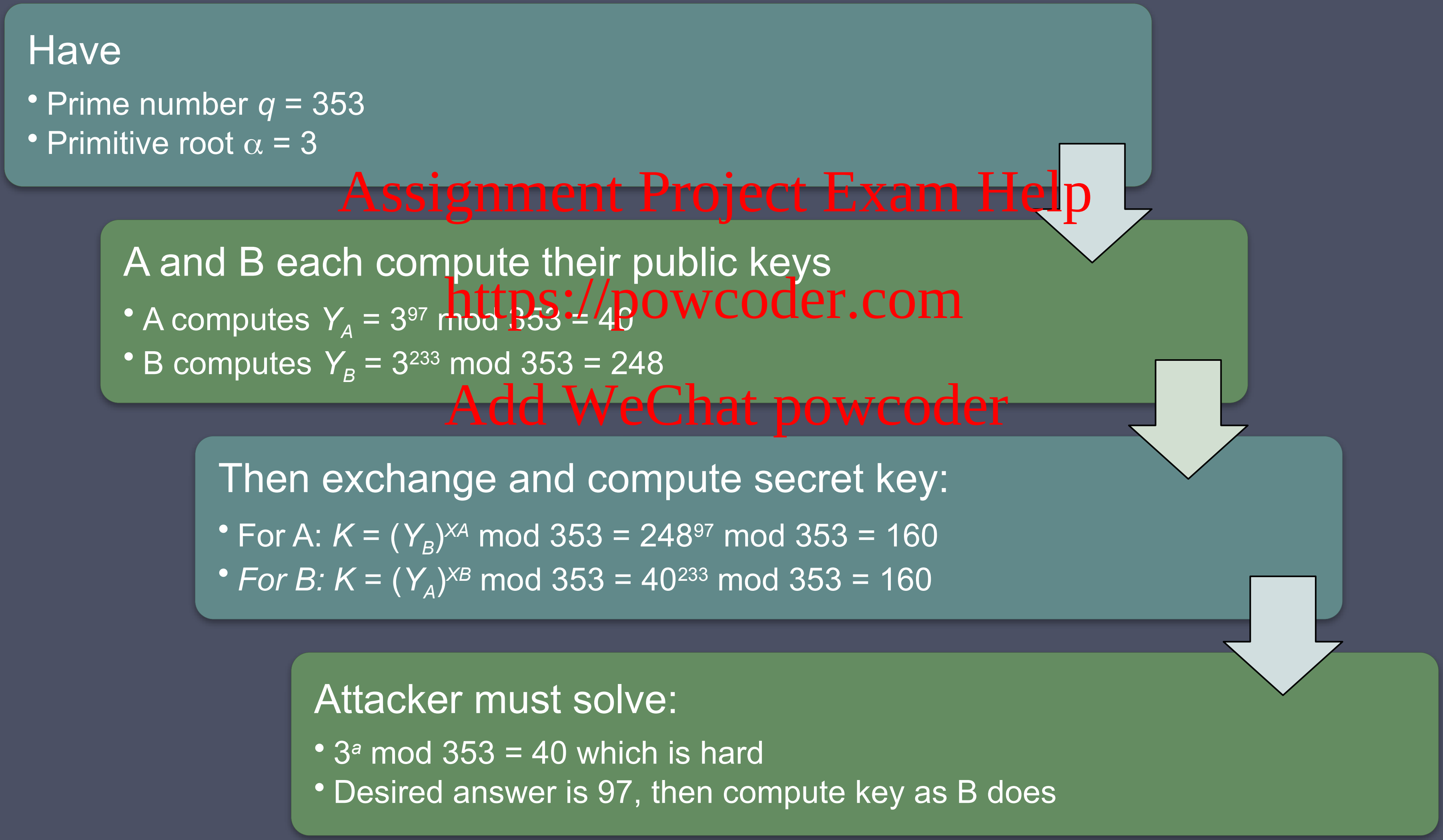
4. A and B values exchanged

5. $K1 = 6^2$ mod $7 = 36$ mod $7 = 1$

5. $K2 = 4^3$ mod $7 = 64$ mod $7 = 1$

K1 and *K2* should be the **same** and are the secret key

Diffie-Hellman Example-02



Have

- Prime number q = 353
- Primitive root α = 3

A and B each compute their public keys

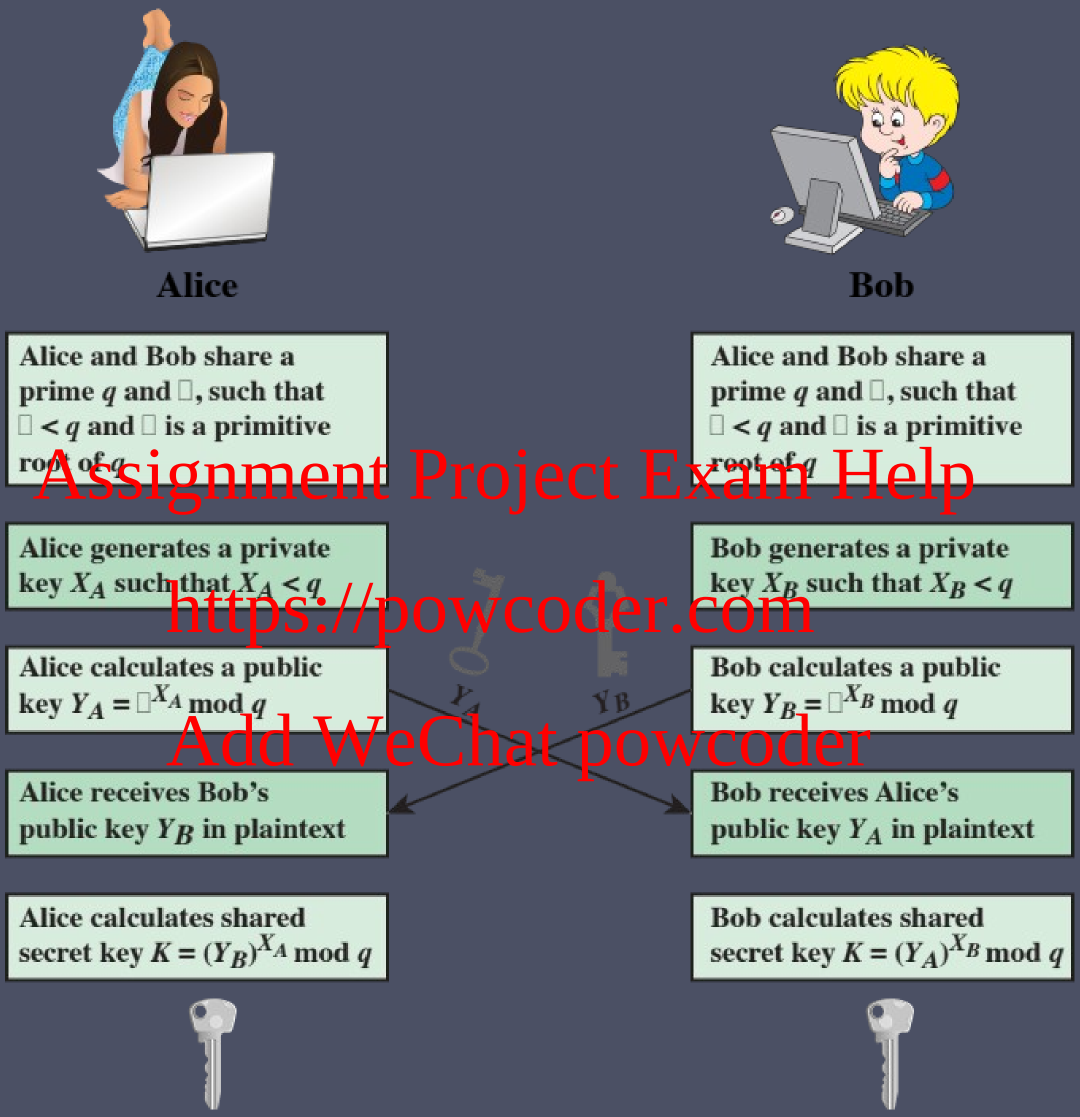
- A computes $Y_A = 3^{97}$ mod 353 = 40
- B computes $Y_B = 3^{233}$ mod 353 = 248

Then exchange and compute secret key:

- For A: $K = (Y_B)^{XA}$ mod 353 = 248^{97} mod 353 = 160
- *For B:* $K = (Y_A)^{XB}$ mod 353 = 40^{233} mod 353 = 160

Attacker must solve:

- 3^a mod 353 = 40 which is hard
- Desired answer is 97, then compute key as B does



Alice and Bob share a prime q and □, such that □ < q and □ is a primitive root of q

Alice generates a private key X_A such that $X_A < q$

Alice calculates a public key $Y_A = □^{X_A} \bmod q$

Alice receives Bob's public key Y_B in plaintext

Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$

Alice and Bob share a prime q and □, such that □ < q and □ is a primitive root of q

Bob generates a private key X_B such that $X_B < q$

Bob calculates a public key $Y_B = □^{X_B} \bmod q$

Bob receives Alice's public key Y_A in plaintext

Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

Figure 21.10 Diffie-Hellman Key Exchange

Man-in-the-Middle Attack

- Attack is:
  1. Darth generates private keys X_{D1} and X_{D2}, and their public keys Y_{D1} and Y_{D2}
  2. Alice transmits Y_A to Bob
  3. Darth intercepts Y_A and transmits Y_{D1} to Bob. Darth also calculates K2
  4. Bob receives Y_{D1} and calculates K1
  5. Bob transmits X_A to Alice
  6. Darth intercepts X_A and transmits Y_{D2} to Alice. Darth calculates K1
  7. Alice receives Y_{D2} and calculates K2
- All subsequent communications compromised

Other Public-Key Algorithms

Digital Signature Standard (DSS)

Elliptic-Curve Cryptography (ECC)

Next