Human - is the most complex factor in computer security

Skim read the news blog listed below– while we are waiting to start session

URL is also in the chat

• • •

https://www.bbc.co.uk/news/technology-54591761

Introduction to computer security: Symmetric key

By.

Dr. Imran Ullah Khan

Informatics, University of Sussex

For further reading

1)
Information on these slides is taken from Chapter 02 Cryptography online book written by Bill Buchanan available online in Sussex Library
http://asecuritysite.com/crypto02
http://asecuritysite.com/encryption

2) Computer Security by William Stallings and Lawrie Brown
Part of chapter 20

Classified along three independent dimensions:

The type of operations used for transforming plaintext to ciphertext

- Substitution – each element in the plaintext is mapped into another element
- Transposition – elements in plaintext are rearranged

The way in which the plaintext is processed

- Block cipher – processes input one block of elements at a time
- Stream cipher – processes the input elements continuously

The number of keys used

- Sender and receiver use same key – symmetric
- Sender and receiver each use a different key - asymmetric

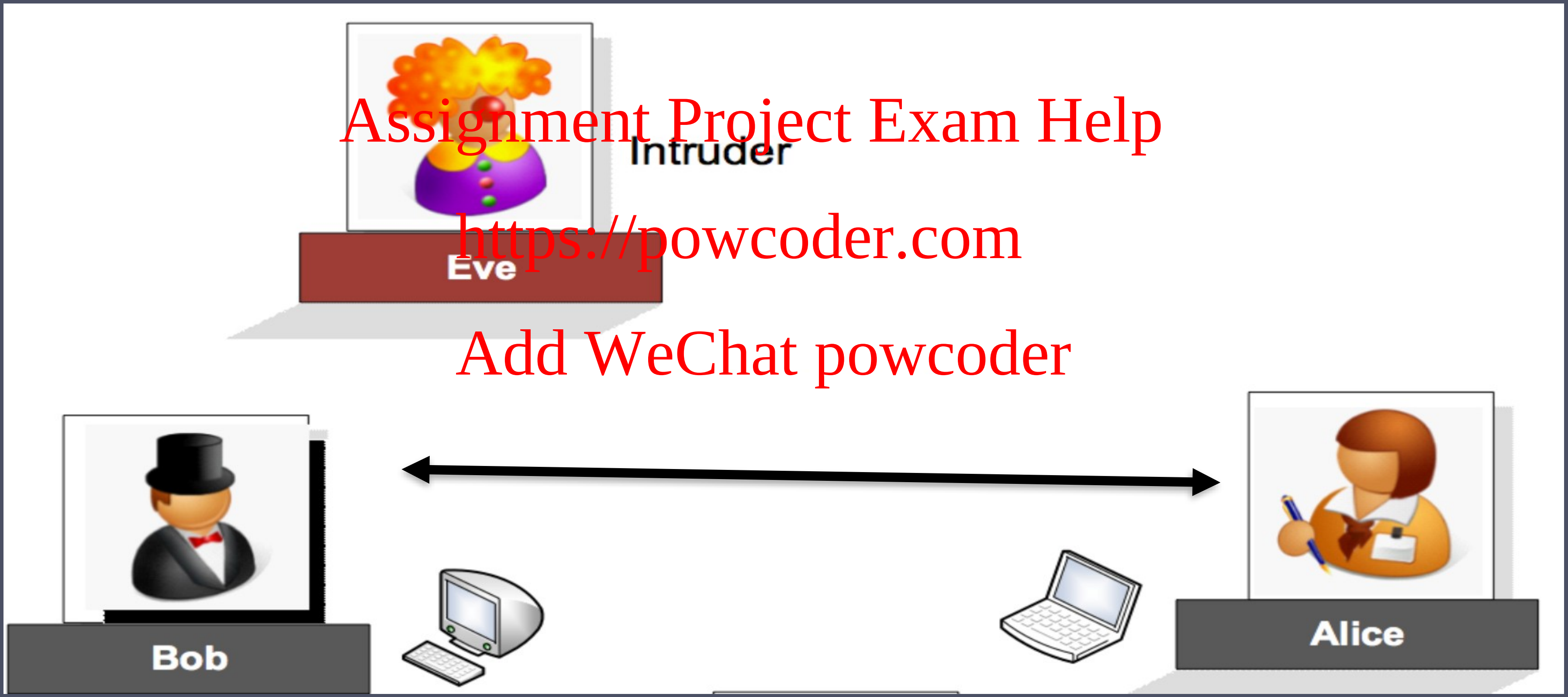
Completed last week	Today	Next

Overview

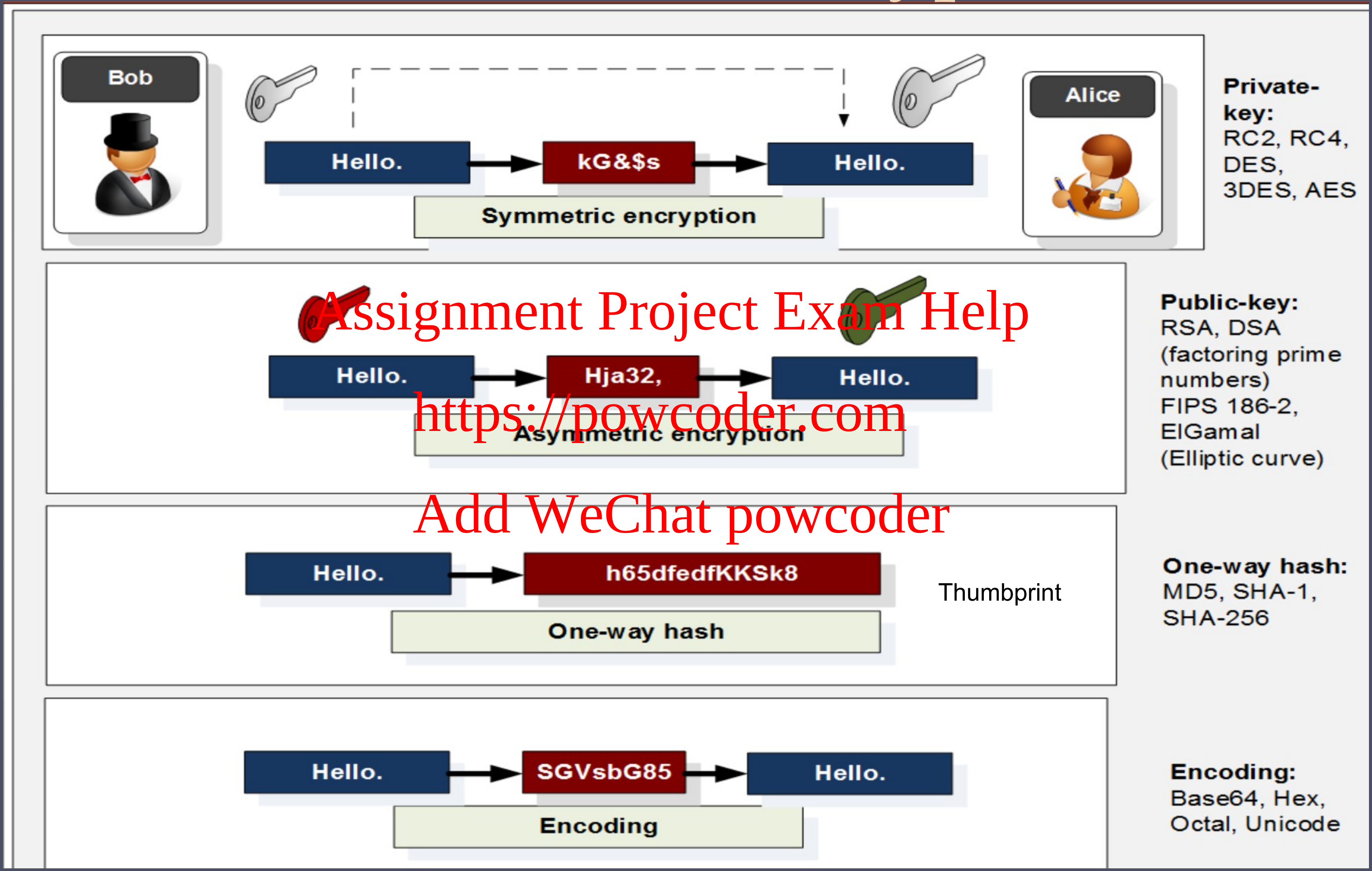
- All forms of encryption
- Block versus stream
- Block cipher: Padding
- Salting
- Time to crack an encrypted asset

    Parallel Computing
- Quantum computing

General scenario

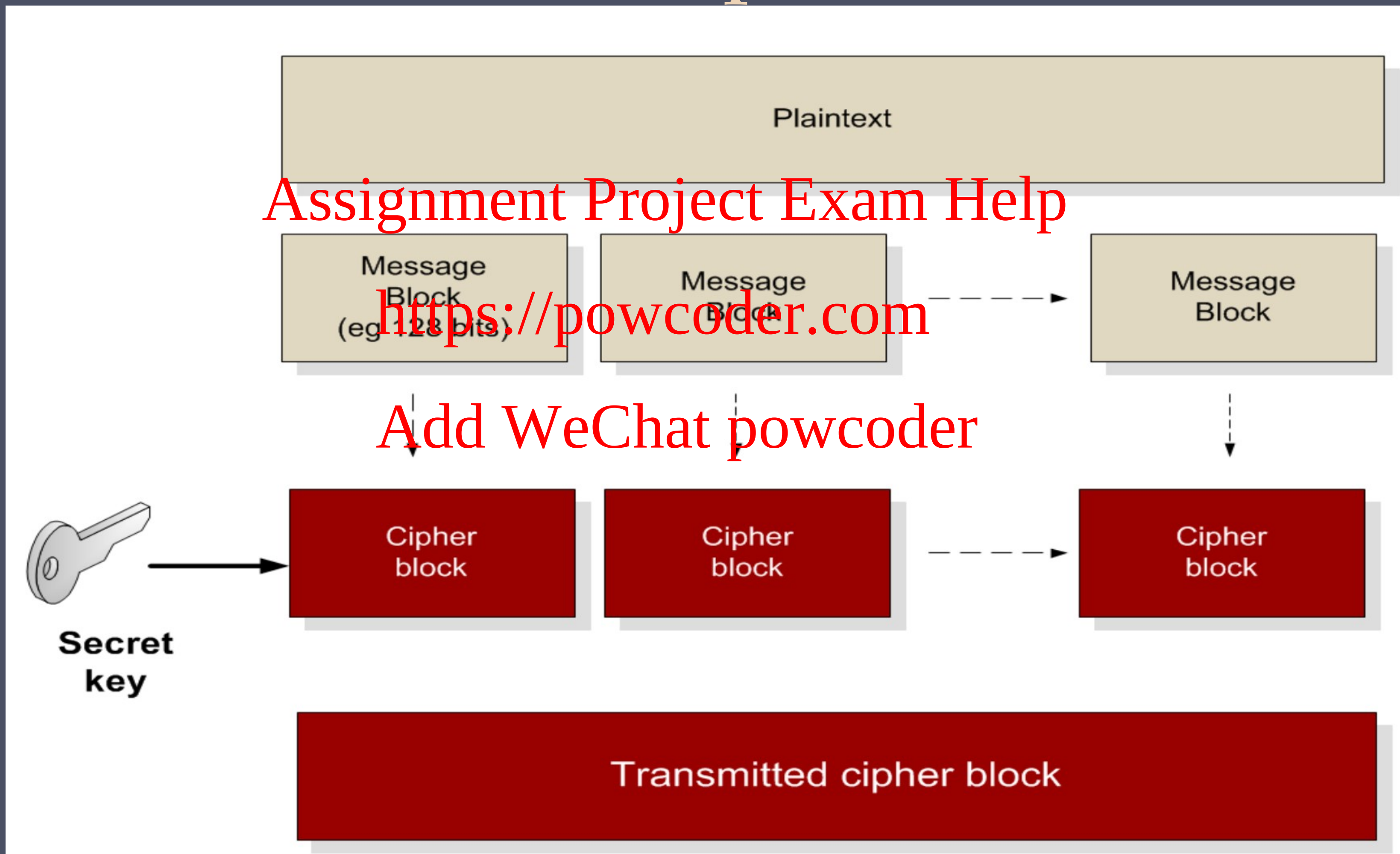


All forms of encryption



Block & Stream cipher

Block cipher



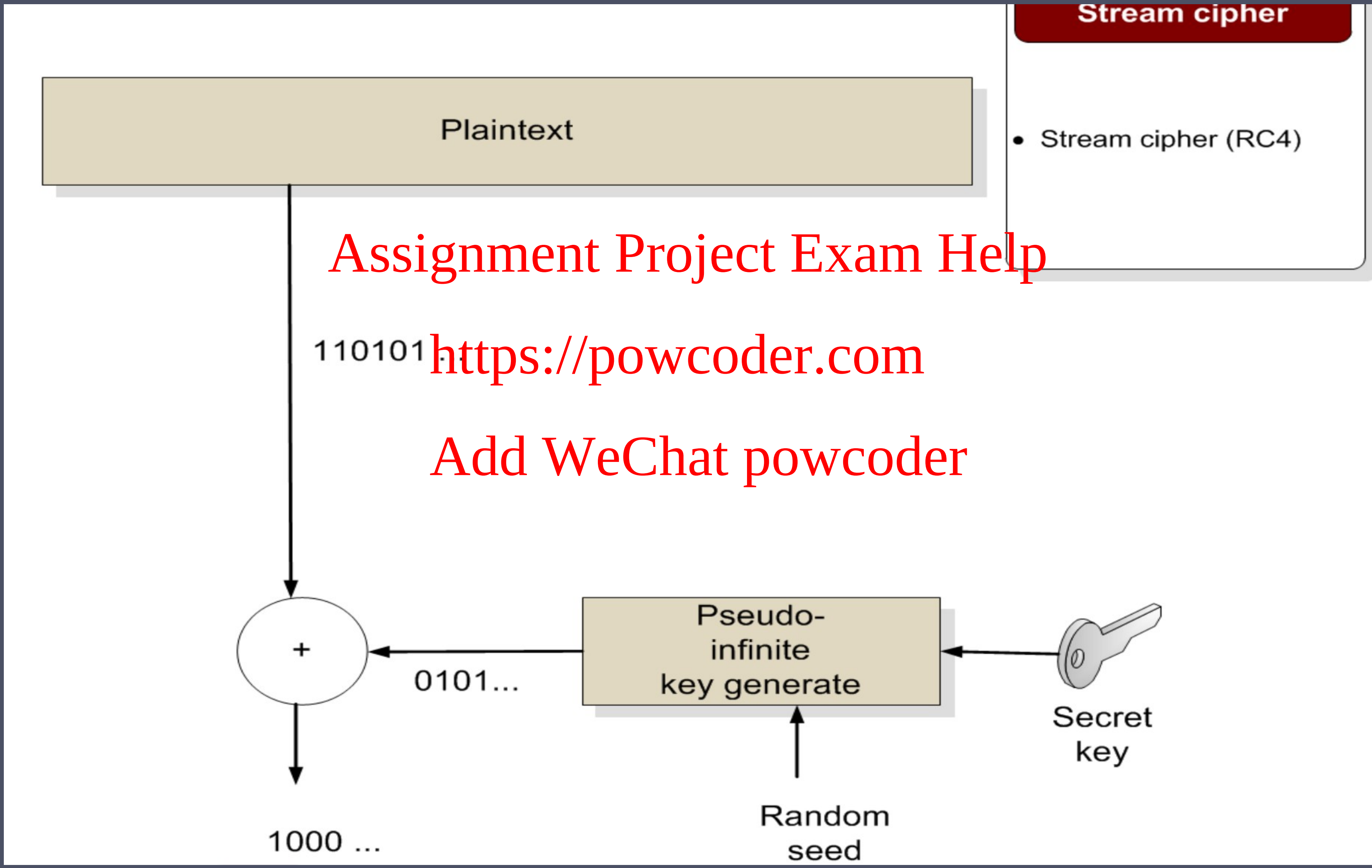
Stream Ciphers

Processes input elements continuously

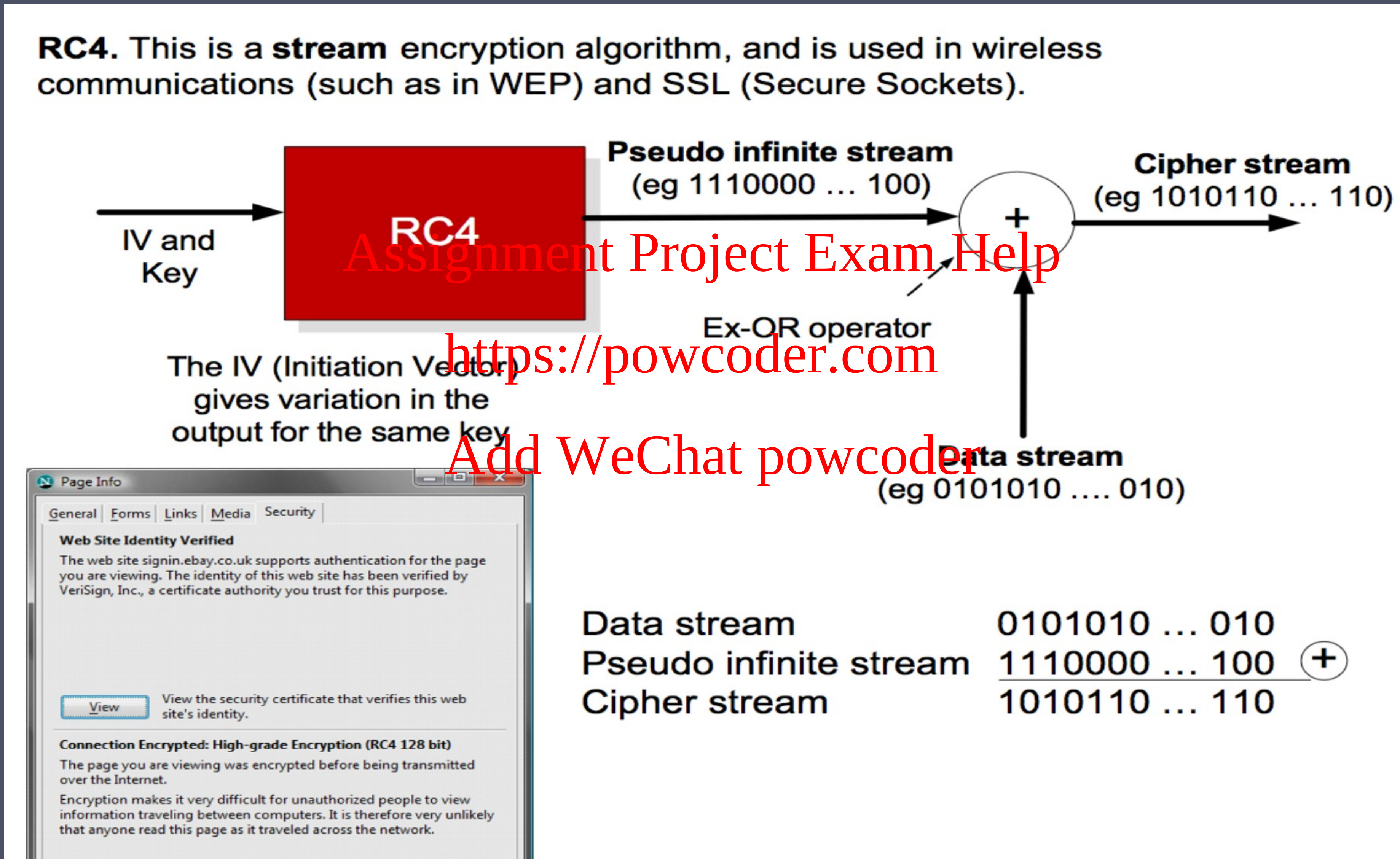
Key input to a pseudorandom bit generator

- Produces stream of random like numbers
- Unpredictable without knowing input key
- XOR keystream output with plaintext bytes

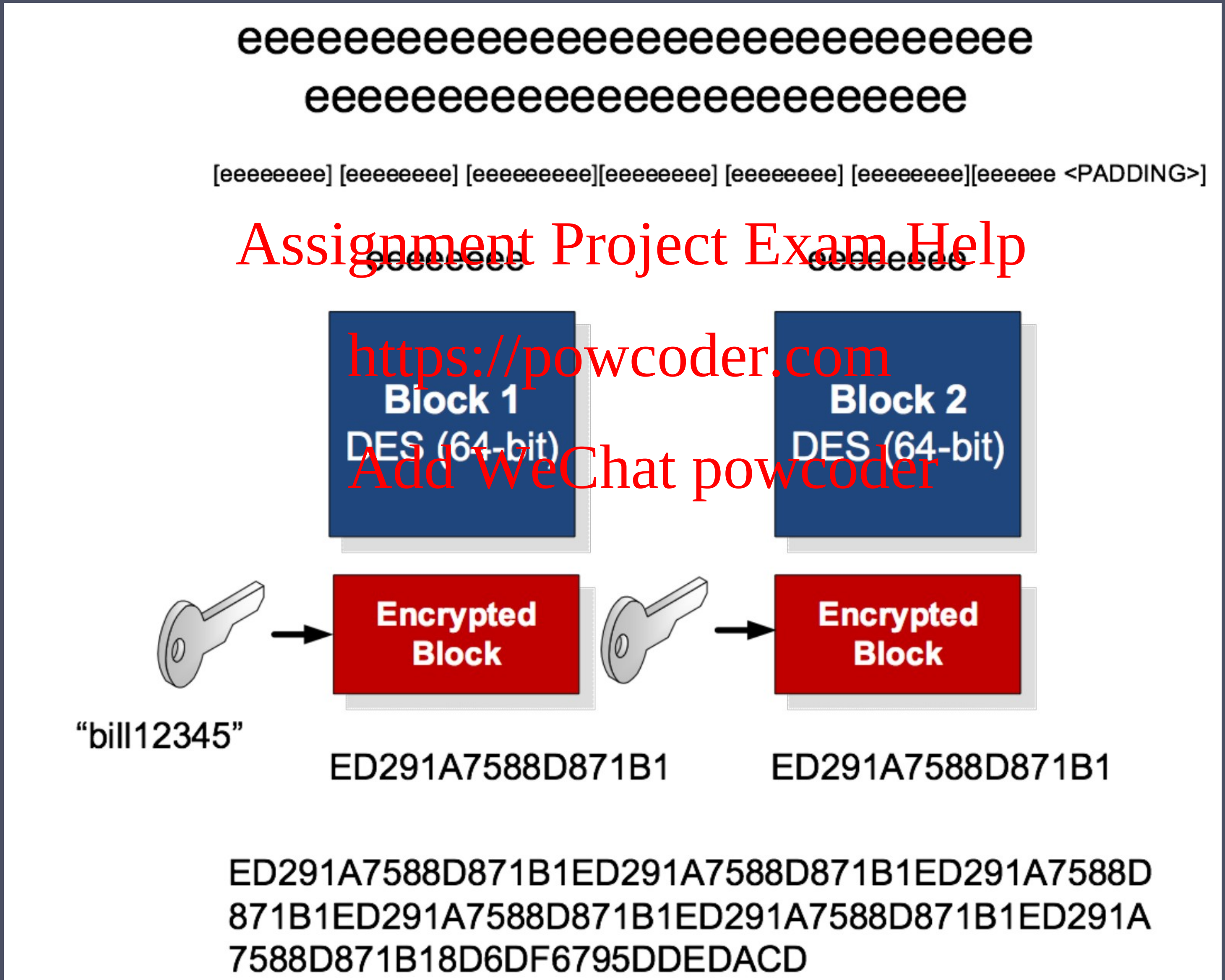
Stream cipher



Stream cipher: RC4 example



Block cipher: Padding



Padding

Padding

- **CMS** (Cryptographic Message Syntax). This pads with the same value as the number of padding bytes. Defined in RFC 5652, PKCS#5, PKCS#7 and RFC 1423 PEM.
- **Bits**. This pads with 0x80 (10000000) followed by zero (null) bytes. Defined in ANSI X.923 and ISO/IEC 9797-1.
- **ZeroLength**. This pads with zeros except for the last byte which is equal to the number (length) of padding bytes.
- **Null**. This pads will NULL bytes. This is only used with ASCII text.
- **Space**. This pads with spaces. This is only used with ASCII text.
- **Random**. This pads with random bytes with the last byte defined by the number of padding bytes.

Padding examples

Plaintext: hello where h=68, e=65 and so on ...

68=h,e=65 [0b in hexadecimal = 11]

After padding (CMS): 68656c6c6f0b0b0b0b0b0b0b0b0b0b0b

Cipher (ECB): 0a7ec77951291795bac6690c9e7f4c0d

Message hex [80=128 by Bruce] zeros bytes

After padding (Bit): 68656c6c6f80000000000000000000000

Cipher (ECB): 731abffc2e3b2c2b5caa9ca2339344f9

ASCII Check values here http://www.asciitable.com/

Afterpadding(ZeroLen):

[Number of padding bytes ten, excluding 0a (hex=10)]

68656c6c6f0000000000000000000000a

Cipher (ECB): d28e2f7e8e44e068732b292bde444245

After padding(Null): 68656c6c6f00000000000000000000000

Cipher (ECB): 44479742246045 3d95856eb2a1520ece

After padding (Space): 68656c6c6f00000000000000000000000

Cipher (ECB): 444797422460453d95856eb2a1520ece

Error this is actually 20……

[Number of random bytes]

After padding (Random): 68656c6c6*ffc6ecfd884a38798d62a***0a**

Cipher (ECB): c2c88b4364d2c2dc6f2cac9ab73c995d

Another example of padding:

Plaintext: hello123

For CMS with AES,

AES use 16 bytes

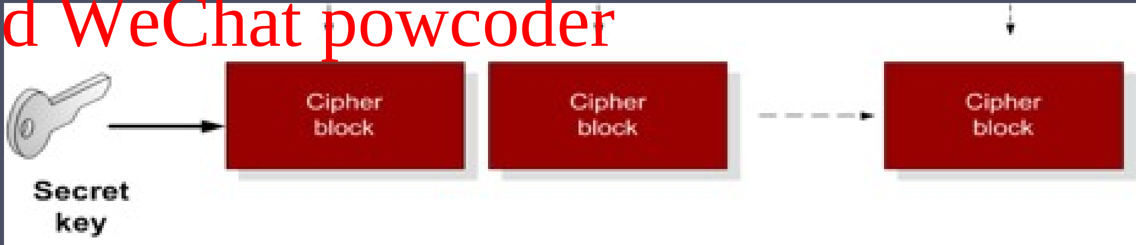
The plaintext will use 8 bytes (count letters in plaintext)

Padding bytes = 16 – 8 (plaintext bytes) = 8 bytes

```
After padding (CMS): 68656c6c6f3132330808080808080808
Cipher (ECB): a20bd93e1af5c0433b68e537ddc70d9a
decrypt: hello123
```

Electronic Codebook (ECB)

- Simplest mode
- Plaintext is handled *b* bits at a time and each block is encrypted using the same key
- "Codebook" is used because there is an unique ciphertext for every *b*-bit block of plaintext
  - Not secure for long messages since repeated plaintext is seen in repeated ciphertext

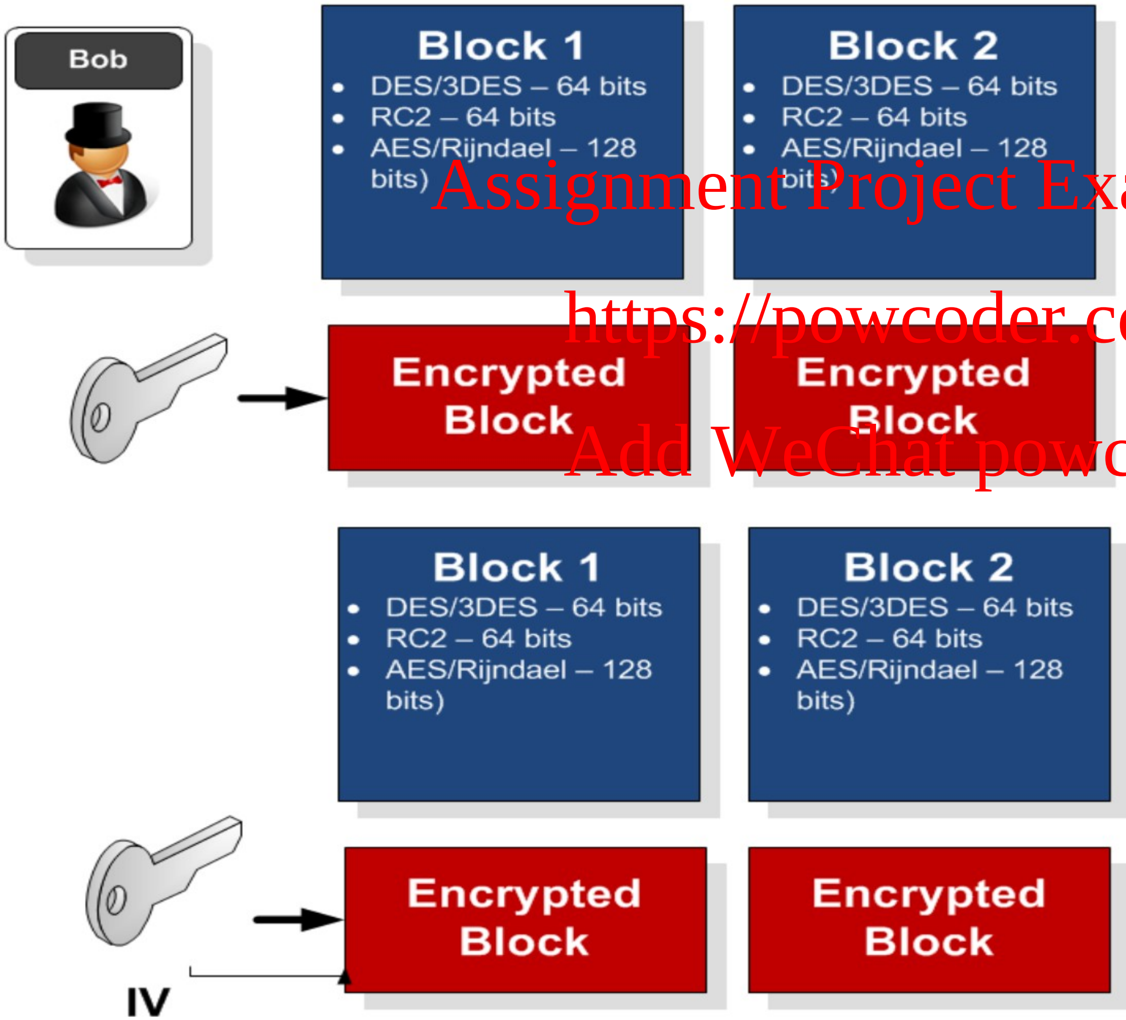


- To overcome security deficiencies you need a technique where the same plaintext block, if repeated, produces different ciphertext blocks

Salting

A major problem in encryption is playback where an intruder can copy an encrypted message and play it back, as the same plain text will always give the same cipher text.

The solution is to add **salt** to the encryption key, as that it changes its operation from block-to-block (for block encryption) or data frame-to-data frame (for stream encryption)



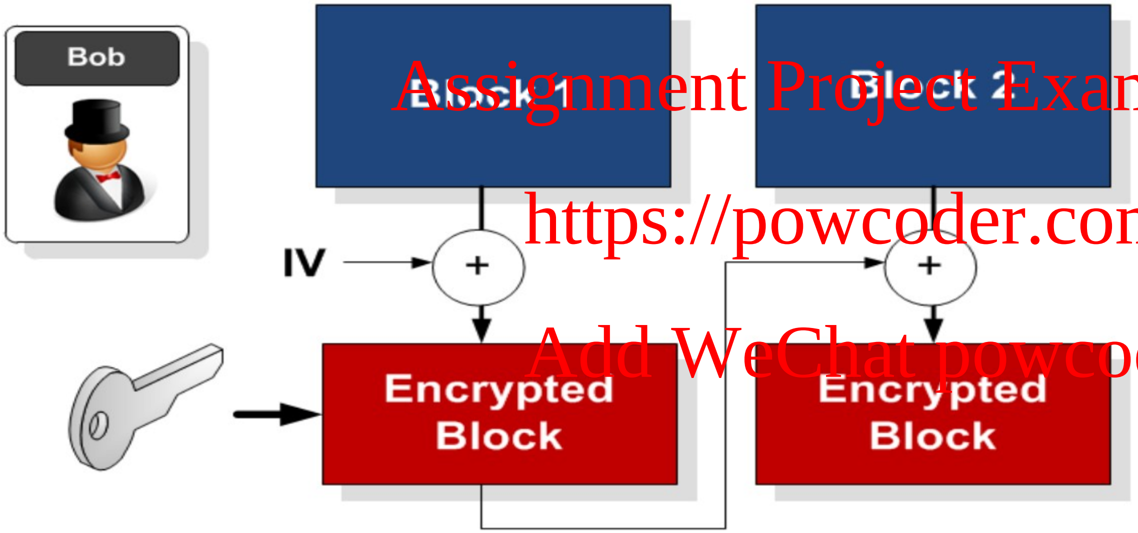
Electronic Code Book (ECB) method. This is weak, as the same cipher text appears for the same blocks.

Hello → 5ghd%43f=
Hello → 5ghd%43f=

Adding salt. This is typically done with an IV (Initialisation Vector) which must be the same on both sides. In WEP, the IV is incremented for each data frame, so that the cipher text changes.

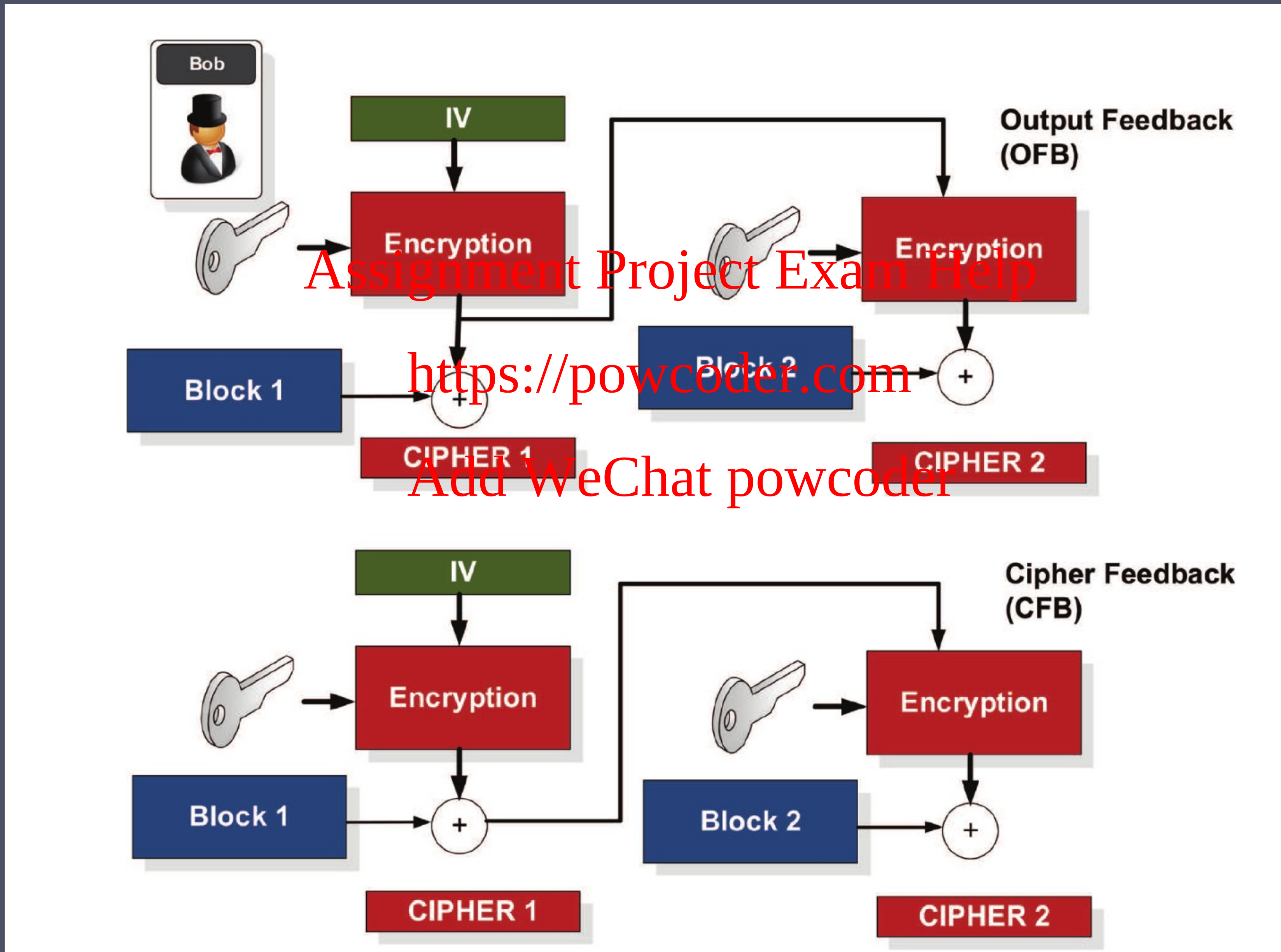
Block Cipher Modes of Operation

Mode	Description	Typical Application
Electronic Codebook (ECB)	Each block of 64 plaintext bits is encoded independently using the same key.	•Secure transmission of single values (e.g., an encryption key)
Cipher Block Chaining (CBC)	The input to the encryption algorithm is the XOR of the next 64 bits of plaintext and the preceding 64 bits of ciphertext.	•General-purpose block-oriented transmission •Authentication
Cipher Feedback (CFB)	Input is processed *s* bits at a time. Preceding ciphertext is used as input to the encryption algorithm to produce pseudorandom output, which is XORed with plaintext to produce next unit of ciphertext.	•General-purpose stream-oriented transmission •Authentication
Output Feedback (OFB)	Similar to CFB, except that the input to the encryption algorithm is the preceding DES output.	•Stream-oriented transmission over noisy channel (e.g., satellite communication)
Counter (CTR)	Each block of plaintext is XORed with an encrypted counter. The counter is incremented for each subsequent block.	•General-purpose block-oriented transmission •Useful for high-speed requirements



Cipher Block Chaining (CBC). This method uses the IV for the first block, and then the results from the previous block to encrypt the current block.

Salting: OFB and CFB



Cracking an encrypted asset: time consideration

Time to crack

- **Clock speed** measures the number of cycles your **CPU** executes per second

- Hertz - one **cycle** per second is known as 1 hertz.

- For example, a **CPU** with a **clock** speed of 2 gigahertz (GHz) can carry out two thousand million (or two billion) **cycles** per second.

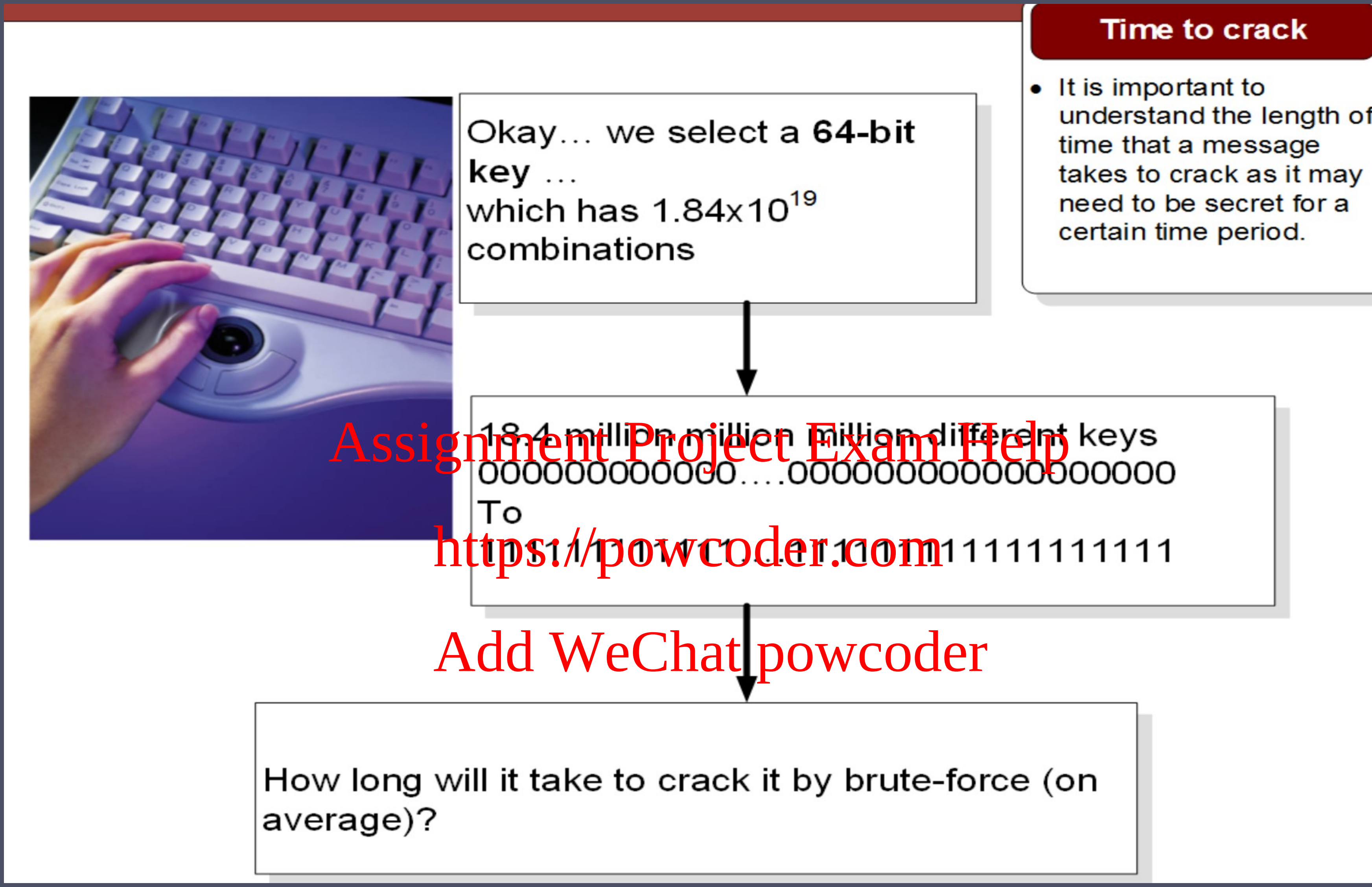
- The higher the **clock** speed a **CPU** has, the faster it can process instructions.

- Some kali Linux tools do provide some estimation, may not be 100% accurate still good enough
  oclhashcat   and john the ripper

Number of keys: the larger the key, the greater the key space

Code size	Number of keys	Code size	Number of keys	Code size	Number of keys
1	2	12	4,096	52	4.5×10^{15}
2	4	16	65,536	56	7.21×10^{16}
3	8	20	1,048,576	60	1.15×10^{18}
4	16	24	16,777,216	64	1.84×10^{19}
5	32	28	2.68×10^{8}	68	2.95×10^{20}
6	64	32	4.29×10^{9}	72	4.72×10^{21}
7	128	36	6.87×10^{10}	76	7.56×10^{22}
8	256	40	1.1×10^{12}	80	1.21×10^{24}
9	512	44	1.76×10^{13}	84	1.93×10^{25}
10	1024	48	2.81×10^{14}	88	3.09×10^{26}

Use online exponent calculator to find total possible keys
e.g. 2 exponent 1 = 2
https://www.rapidtables.com/calc/math/Exponent_Calculator.html



Time to crack.

Why is it important to understand the length of time that a message takes to crack as it may need to be secret for a certain time period.

Time to crack

- It is important to understand the length of time that a message takes to crack as it may need to be secret for a certain time period.

A 64-bit key has 1.84×10^{19} combinations and it could be cracked by brute-force in 0.9×10^{19} goes.

If we use a fast computer such as 1GHz clock (1ns), and say it takes one clock cycle to test a code, the time to crack the code will be:

9,000,000,000 seconds (150 million minutes)
… **2.5 million hours** (285 years)

1 Billionth of a second = nanosecond = 10^{-9}

$$T_{average} = 1.84 \times 10^{19} \times 1 \times 10^{-9} \div 2 \approx 9,000,000,000 \text{ seconds}$$

Average Time = (Total combination of keys X CPU Speed) / 2

Why divide by 2? Because base used to cal total keys was 2

Time to crack

If it takes 2.5 million hours (285 years) to crack a code. How many years will it take to crack it within a day?

Computers typically improve their performance every year … so assume a **doubling** of performance each year.

Moore's Law

Date	Hours	Days	Years
2017	2,500,000	104,167	285
2018	1,250,000	52,083	143

Time to crack

- It is important to understand the length of time that a message takes to crack as it may need to be secret for a certain time period.

Time to crack

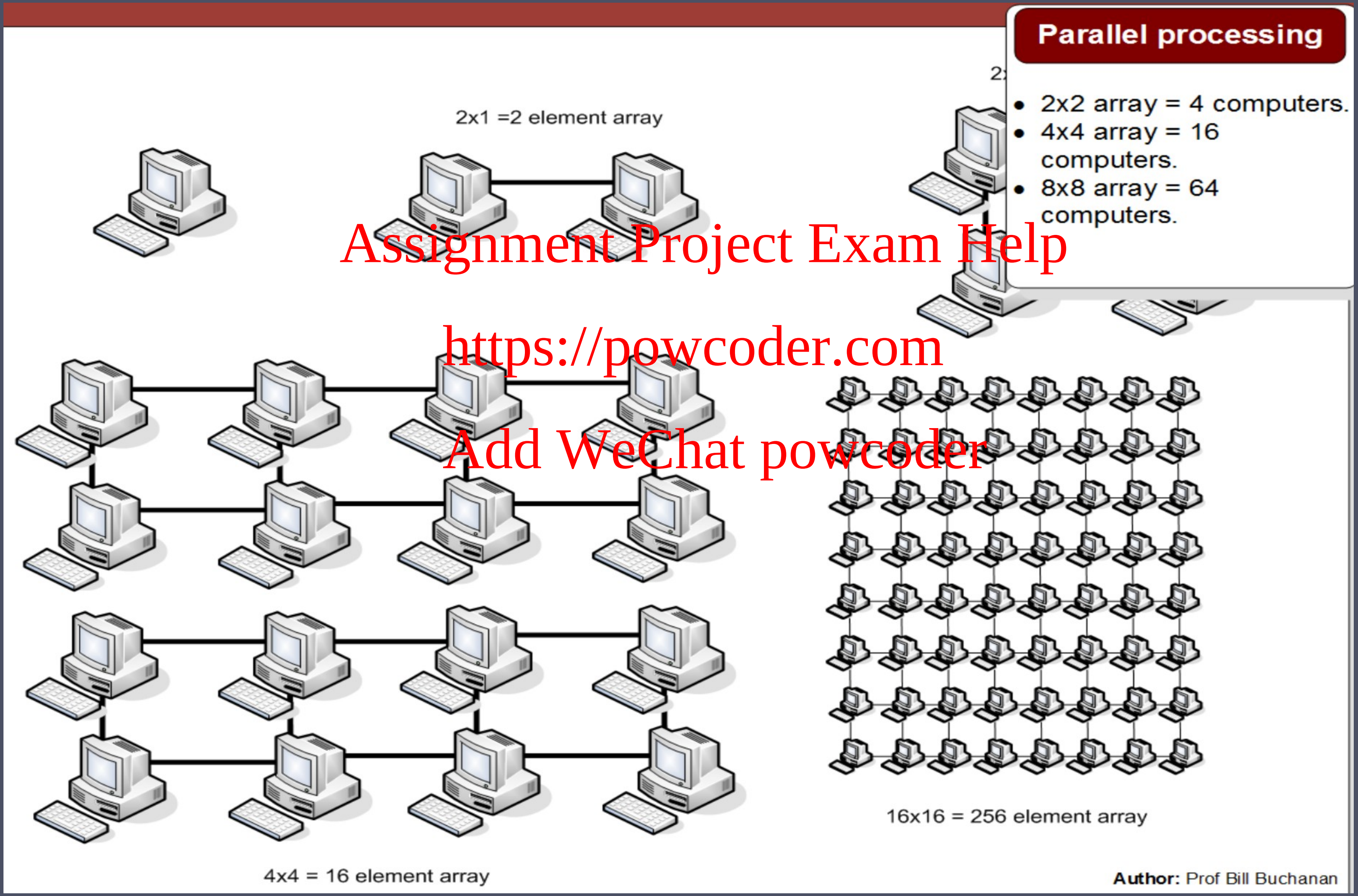
Date	Hours	Days	Years
2017	2,500,000	104,167	285
2018	1,250,000	52,083	143
2019	625,000	26,042	71
2020	312,500	13,021	36
2021	156,250	6,510	18
2022	78,125	3,255	9
2023	39,063	1,628	4
2024	19,532	814	2
+8	9,766	407	1
+9	4,883	203	1
+10	2,442	102	0.3
+11	1,221	51	0.1
+12	611	25	0.1
+13	306	13	0
+14	153	6	0
+15	77	3	0
+16	39	2	0
+17	**20**	**1**	**0**

Time to crack

- From 285 years to 1 day, just by computers increasing their computing power.

56-bit DES:
Developed 1975
30 years ago!
… now easily crackable

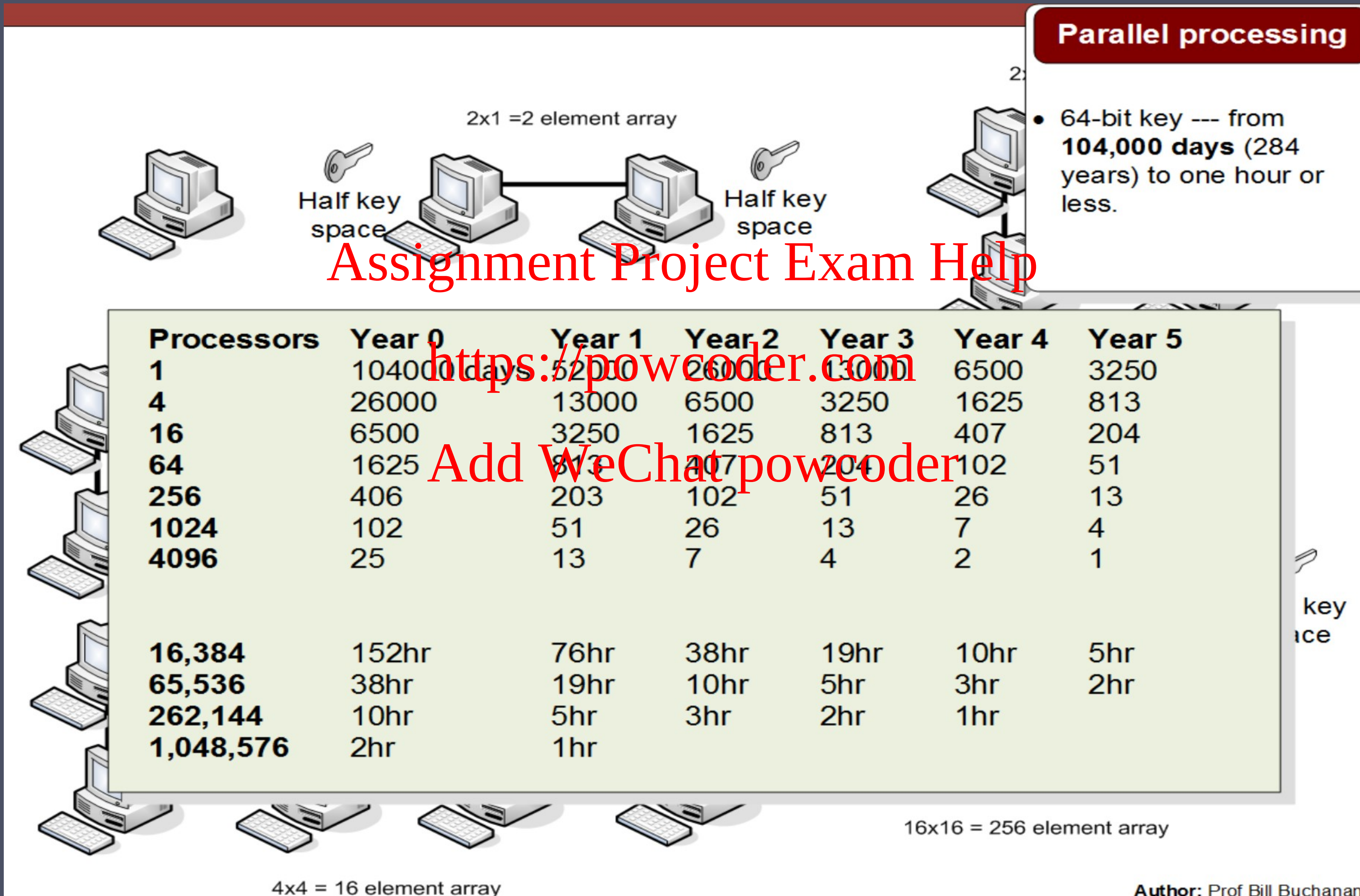
Time to crack: Parallel processing



Time to crack: parallel processing

Parallel processing

- 64-bit key --- from **104,000 days** (284 years) to one hour or less.



Processors	**Year 0**	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**
1	104000 days	52000	26000	13000	6500	3250
4	26000	13000	6500	3250	1625	813
16	6500	3250	1625	813	407	204
64	1625	813	407	204	102	51
256	406	203	102	51	26	13
1024	102	51	26	13	7	4
4096	25	13	7	4	2	1
16,384	152hr	76hr	38hr	19hr	10hr	5hr
65,536	38hr	19hr	10hr	5hr	3hr	2hr
262,144	10hr	5hr	3hr	2hr	1hr	
1,048,576	2hr	1hr				

Quantum computing

Important read

1)Read the white paper provided in Week 4 folder

2) https://www.cryptomathic.com/news-events/blog/quantum-computing-and-its-impact-on-cryptography

Key points

- Bits versus qubits
  - qubits store multiple states
- E-commerce depend on encryption
  - asymmetric cryptography:
    - (a) Integer factorisation
    - (b) Discrete logarithm
    - (c) Elliptic curve discrete logarithm
- Difficult to break but quantum computing will make it possible