

Cryptography In The Digital World

Jack Bradbrook

Topics

- Cryptography: Direct communication Vs Over a network
- Block Ciphers
- Stream Ciphers
- Hashing and hash Functions
- Digital Signatures

Nothing individually is that hard today but we've got a lot of ground to try and cover.

After today we've finished the bulk of the super technical stuff and will be looking more at practical aspects of cryptography, protocols legal issues etc.

Relevance to the Exam

- By the end of today these types of questions should hopefully be ok:

8 A way of verifying both the sender of information and the integrity of a message is using which of the following?

- A Digital signatures.
- B Digital certificates.
- C Public key encryption.
- D Private key encryption.

9 Why would a user check that a software patch is **CORRECTLY** signed by the software publisher?

- A To ensure that the user does not exceed the software license.
- B To ensure the patch downloaded correctly.
- C To ensure that it is a legitimate patch issued by the correct party.
- D To ensure the patch is compatible with the user's version of software.

10 Which of the following is the **CORRECT** description of ciphers?

- A Stream ciphers encrypt continuous streams of data.
- B Block ciphers encrypt blocks of data of variable size.
- C Polyalphabetic substitution ciphers keep the substitution alphabet constant for every symbol.
- D Transposition ciphers take groups of characters and shift them according to a random system.

14 Which of the following is **NOT** a block cipher operating mode?

- A Cipher Block Chaining (CBC) mode.
- B Electronic Codebook (ECB) mode.
- C Cipher Feedback (CFB) mode.
- D Asymmetric Key Cipher (AKC) mode.

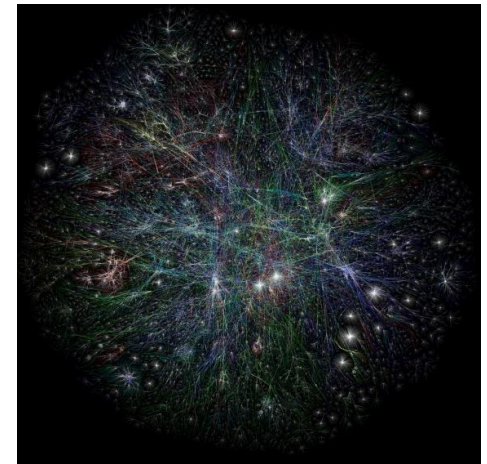
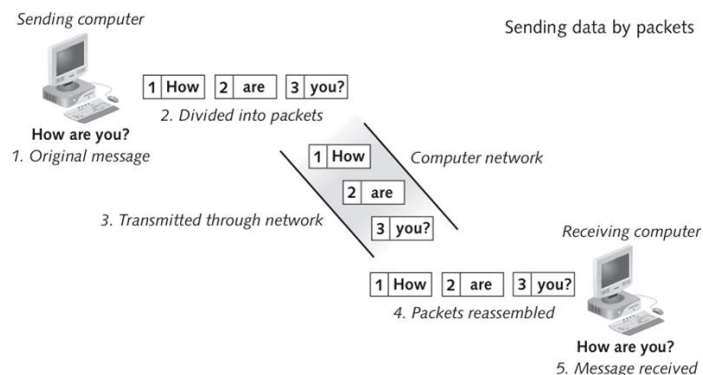
Practical Challenges of modern cryptography

- The way data is communicated in the 21st century is much more complicated than was the case historically
- In Classical cryptography the Cipher Text (CT) message is sent from Alice to Bob directly over a physical medium. E.g. a carrier Pidgeon, cable or radio waves, typically using Morse code to encode the encrypted characters as pulses
- Once received the CT message is translated as one block into the correspondent Plain Text (PT).
- One single block of data to preform encryption and decryption on.



Encrypting Messages Over Networks

- When data is transferred between computers it's sent in Packets, small blocks of data which take different routes across a network and are re-assembled at the receiving end of the communication to form bigger messages.
- The amount of data being encrypted and sent digitally can be much greater than in classical cryptography, an image or video file is huge when compared to a file of just text.
- The infrastructure of the internet means parts of the message can take different routes to their destination and can be lost more easily than with direct communication over a wire.



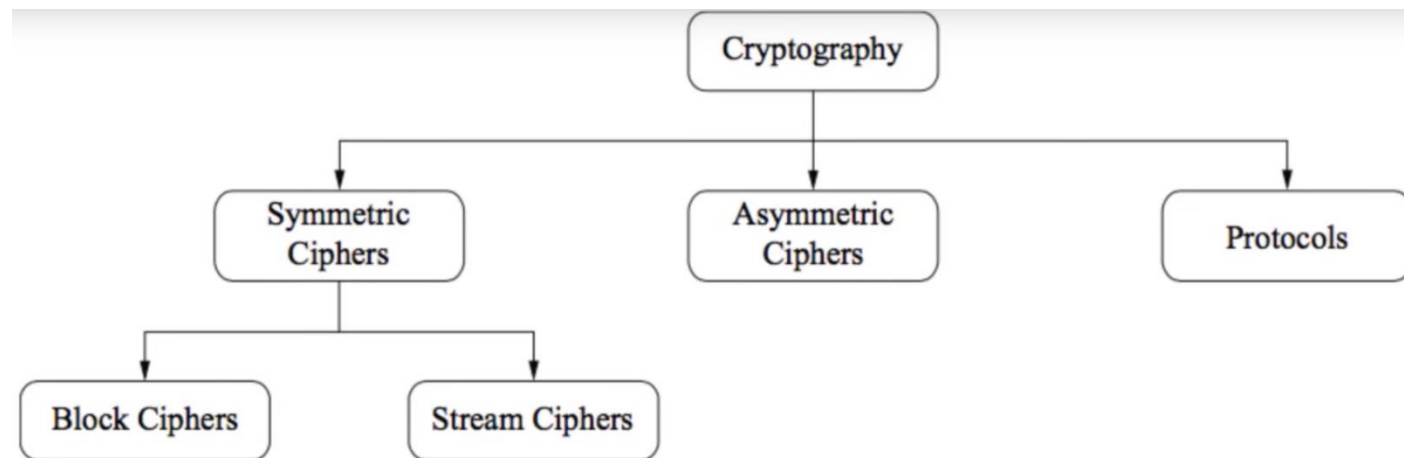
How do we solve these problems?

- We need to select the most appropriate ciphers and communication protocols depending on the specific application.
- Some considerations:
 - Is the network stable, are we likely to lose packets?
 - How frequently is data being communicated, is it frequent enough to slow down the device or drain its power if the encryption is too heavy?

Block and Stream Ciphers

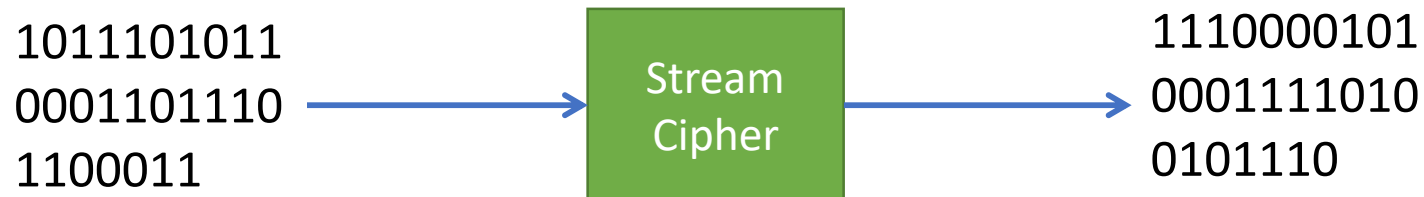
Block Ciphers vs Stream Ciphers

- Both are forms of symmetric encryption (both encryption and decryption use the same key)
- Block ciphers: Encrypt data in blocks of a specific length e.g. 64 or 128 bits
- Stream Ciphers: Encrypt bits individually (A stream cipher is a block cipher with a block length of 1 bit)



Stream Ciphers

- Very fast, Data flows through the cipher like a stream.
- Works on the Binary representation of the data
- Requires very little energy and computational power to execute (efficient)
- Commonly used on
 - Mobile devices, e.g. voice data when making a phone call,
 - Live video, e.g. conference calls, where lots of data needs encrypting fast.



Encoding data to binary

- We use different encoding standards to convert our different types of data to binary, e.g. ASCII
- JPEG, GIF, PNG are all methods for encoding different binary data in Image / graphic form.
- We use different encodings depending on the task we're trying to do



ASCII Code: Character to Binary

0	0011 0000	O	0100 1111	m	0110 1101
1	0011 0001	P	0101 0000	n	0110 1110
2	0011 0010	Q	0101 0001	o	0110 1111
3	0011 0011	R	0101 0010	p	0111 0000
4	0011 0100	S	0101 0011	q	0111 0001
5	0011 0101	T	0101 0100	r	0111 0010
6	0011 0110	U	0101 0101	s	0111 0011
7	0011 0111	V	0101 0110	t	0111 0100
8	0011 1000	W	0101 0111	u	0111 0101
9	0011 1001	X	0101 1000	v	0111 0110
A	0100 0001	Y	0101 1001	w	0111 0111
B	0100 0010	Z	0101 1010	x	0111 1000
C	0100 0011	a	0110 0001	y	0111 1001
D	0100 0100	b	0110 0010	z	0111 1010
E	0100 0101	c	0110 0011	.	0010 1110
F	0100 0110	d	0110 0100	,	0010 0111
G	0100 0111	e	0110 0101	:	0011 1010
H	0100 1000	f	0110 0110	;	0011 1011
I	0100 1001	g	0110 0111	?	0011 1111
J	0100 1010	h	0110 1000	!	0010 0001
K	0100 1011	I	0110 1001	'	0010 1100
L	0100 1100	j	0110 1010	"	0010 0010
M	0100 1101	k	0110 1011	{	0010 1000
N	0100 1110	l	0110 1100	}	0010 1001
				space	0010 0000

The XOR (Exclusive Or) Operator

- Given a possible 1 or 0 and another 1 or 0 if the two values are different the result is 1, if the two values are the same then the result is 0
- \oplus symbol is often used to denote XOR.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Exercise

- XOR the two strings A and B
- A: 1011101000
- B: 0111010011
- That is give $A \oplus B$

Answer + Another Exercise

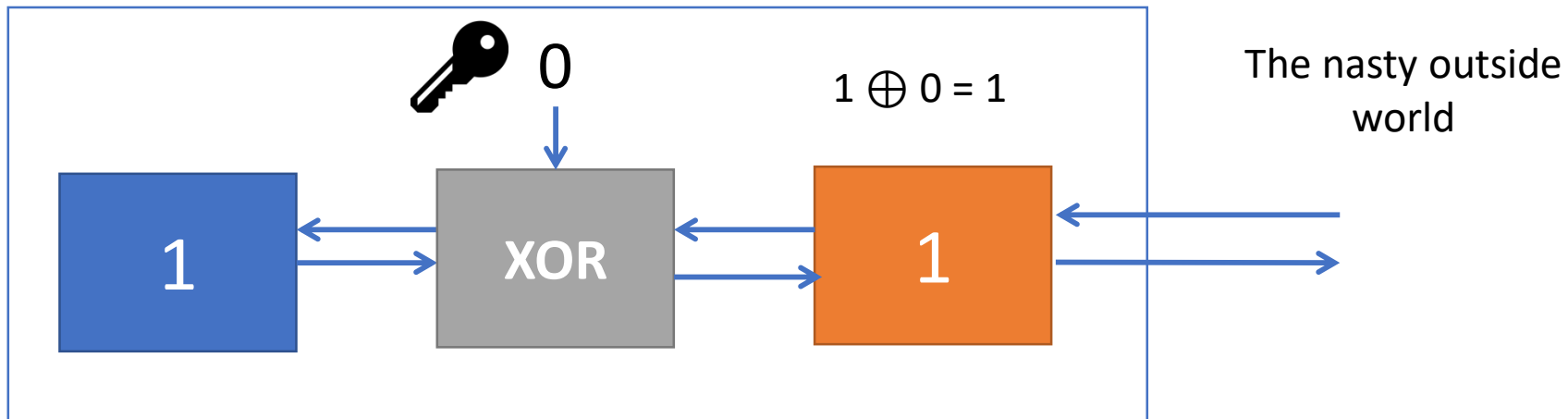
- A: 1011101000
- B: 0111010011
- C: $A \oplus B = 1100111011$
- Now you've done this do $C \oplus B$, what do you notice?

Undoing an XOR

- If you do $C \oplus B$ you get A back again, likewise $C \oplus A$ gives you B back again.
- The implication of this is that an XOR operation can be undone easily using the same operator.
- This is why we can use the same key to perform the encryption and the decryption.

Stream Ciphers

- Stream ciphers XOR the key with the plain text to generate the Cipher Text
- The Cipher text can then be XORed with the key to retrieve the plain text at the other end.
- Key is a pseudo random string of bits and generated from a seed.



Block Ciphers

- Transform **fixed size** blocks of plain text into identical size blocks of cipher text
- The key is used to encrypt and decrypt each of the blocks
- When decrypted the blocks can then be re assembled into the message at the far end.
- Used for many internet applications.
- Slower but more secure than a stream cipher










Confusion and Diffusion

- Diffusion – If we change a single bit in the key then statistically at least 50% of all the bits in the ciphertext should change
 - This means we can't isolate how one part of a cipher text corresponds with the key, this is because the key changes the cipher text in lots of different places.
- Confusion – Each bit of the ciphertext should depend on several parts of the key,
 - This makes it harder for the key to be deduced from the cipher text.
- Stream ciphers do not add much confusion or diffusion, each part of the key corresponds with one part of the message.

Block Cipher Modes

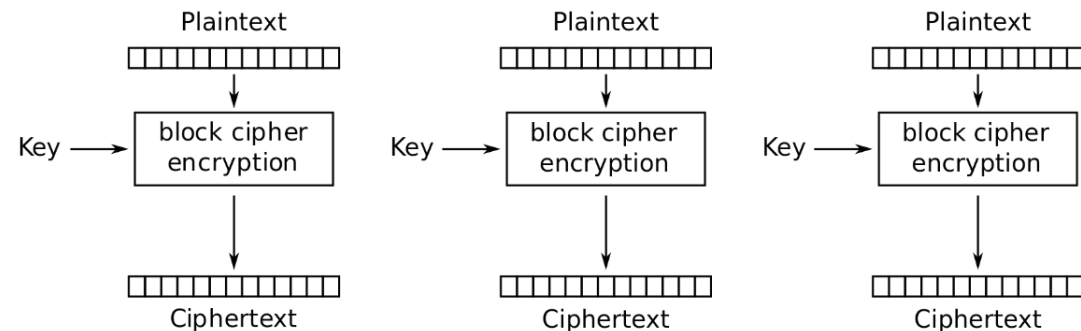
- Block Cipher modes are different methods used to apply encryption with a block cipher, there are many of them but the most well known ones are:

- Electronic Codebook (ECB)
- Cipher Block Chaining Mode (CBC)
- Cipher Feedback CFB Mode (CFB)

 CBC All	 OFB modes	 GCM are
 XEX beautiful	 ECB not you	 CTR and
 OCB deserve	 CFB to be	 XTS used

The Electronic Codebook (ECB)

- Most basic block cipher mode
- Split cipher text into blocks and encrypt each block separately using the same key for each block.
- The resulting cipher text is all of the cipher blocks joined together.
- Any repetition that appears in the plain text will also appear in the cipher text, this is a problem when encrypting longer messages!
- No Confusion or Diffusion



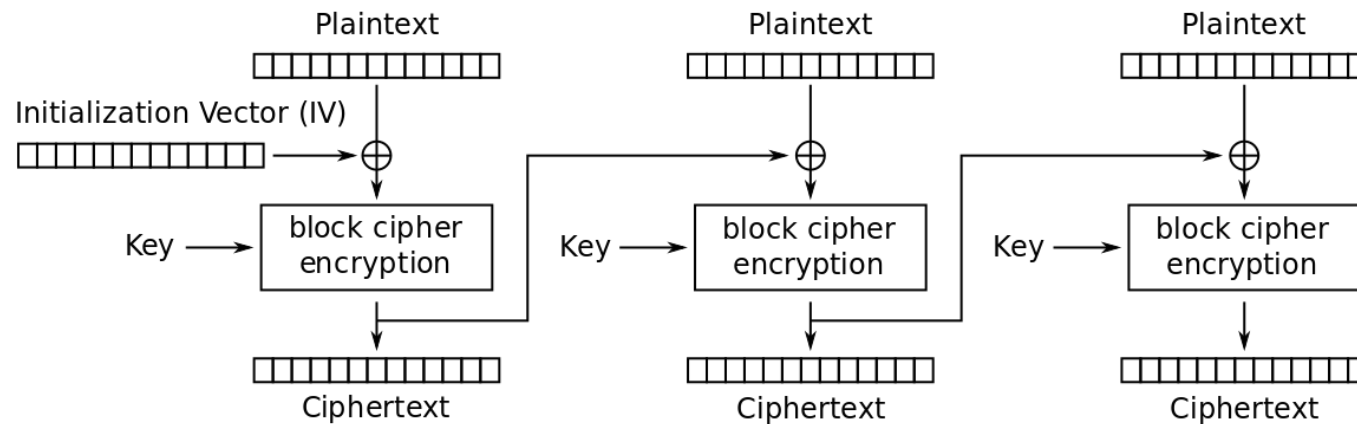
Electronic Codebook (ECB) mode encryption

Iteration Cipher Encryption processes

- Most Block ciphers are Iteration Ciphers
- They repeatedly apply an encryption function known as the **round** function to add security
 - Think about the enigma machine with the three wheels, each wheel adds a “**round**” of encryption
- Decryption is done by reversing the iteration stages

Cipher Block Chaining (CBC)

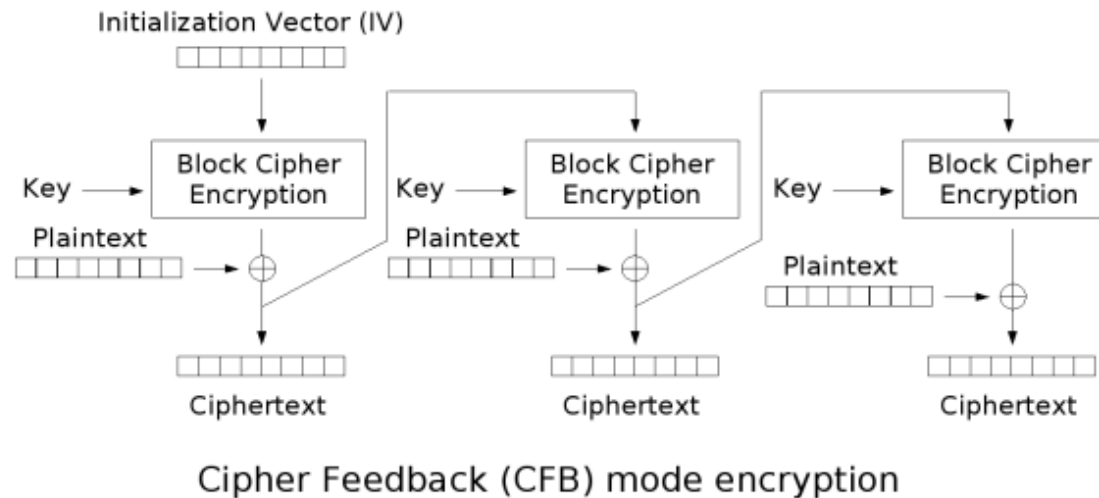
- Take plain text and split it into blocks.
- XOR the Encrypted output of the previous block with the plain text, then encrypt using the key.
- Initialisation Vector: set of value we use for first round of encryption before previous block can be used, doesn't need to be secret.



Cipher Block Chaining (CBC) mode encryption

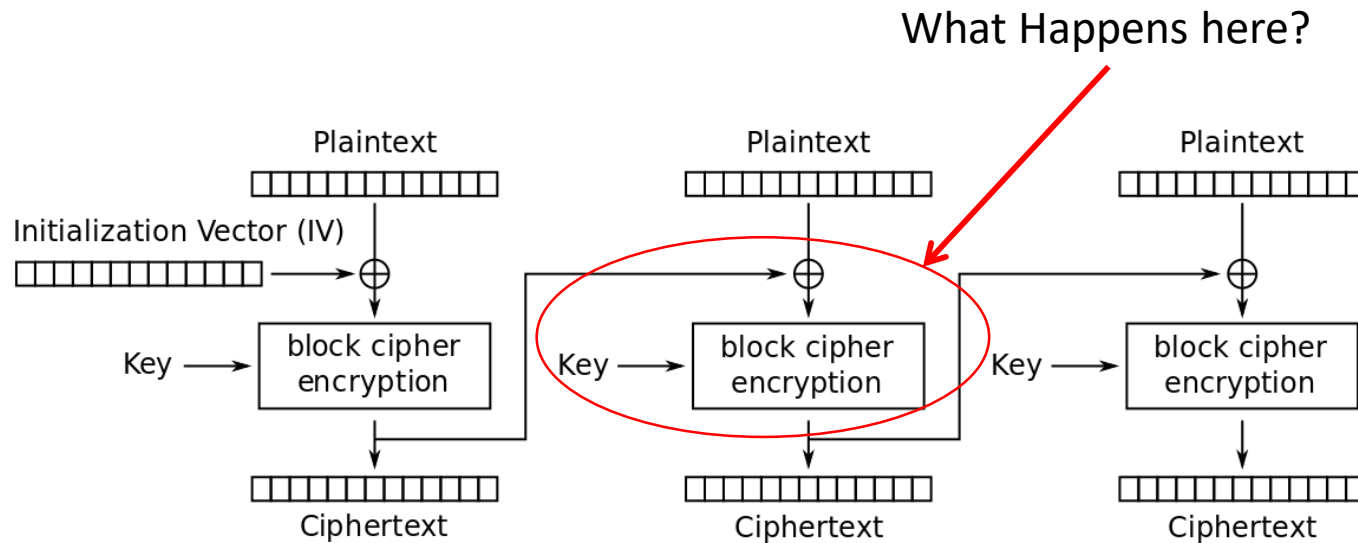
Cipher Feedback CFB Mode (CFB)

- Similar to CBC, but a subtle difference
- Instead of XORing the Plain Text for each block with the previous cipher text then encrypting that we encrypt the output from the previous block again, then XOR that with the plain text.



The Actual Encryption Process?

- The next question is: for each round what is done to apply the encryption process before moving onto the next round?



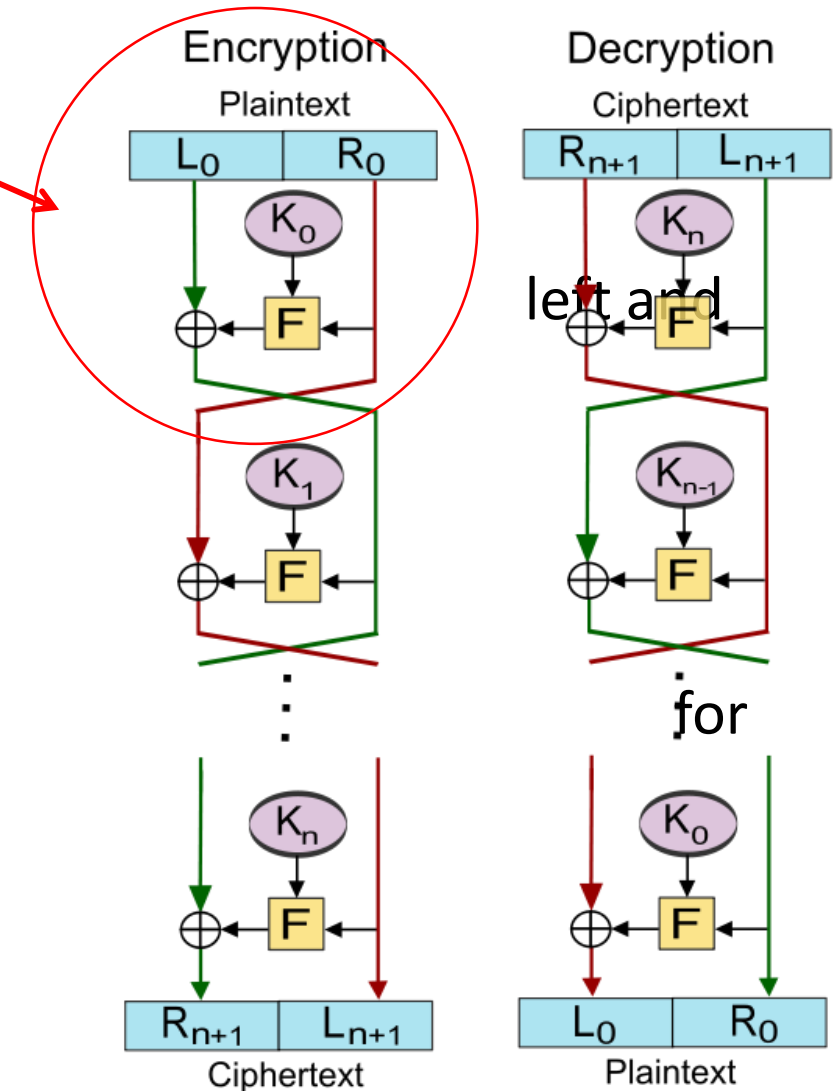
Cipher Block Chaining (CBC) mode encryption

Example 2: Feisal Cipher

Method:

1. Split the Plain Text or the cipher text into right hand sides
2. XOR the left hand side with the key
3. Swap the XORed left side with the right side the following round
4. Repeat this for multiple rounds

This is one round



Substitution Permutation Network

- For each block of plain text:
 - Perform a substitution of the blocks
-> Swap a block of bits for another block of bits

2. Permute the data

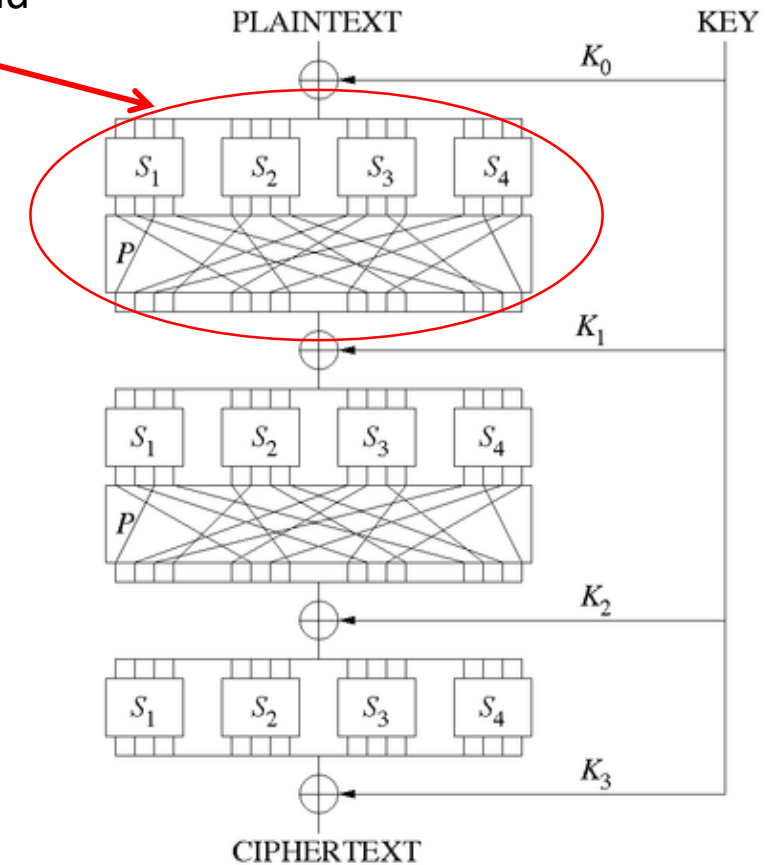
-> Take the outputs from all

Of the blocks in one round and move the data around into
Different block for the next round
(similar to enigma machine wheels)

3. Repeat this for multiple rounds

- Adds lots of Diffusion and Confusion!
- Hard to break with enough rounds.

This is one round



Hashing Data

Hashing

- Uniquely map an input value to some output value
- Extremely powerful concept in computing!
- Has anyone here used a Dictionary or HashTable object in a programming language before?

Quick Knowledge Check

- Have we done much programming before?
- Do we know what Arrays are?
- Do we know how to use an array in a general programming language?
 - Make one
 - Store data in it
 - Access the data
 - Change the data
- Before we look at cryptographic hashing we need to familiarise ourselves with some other concepts first.

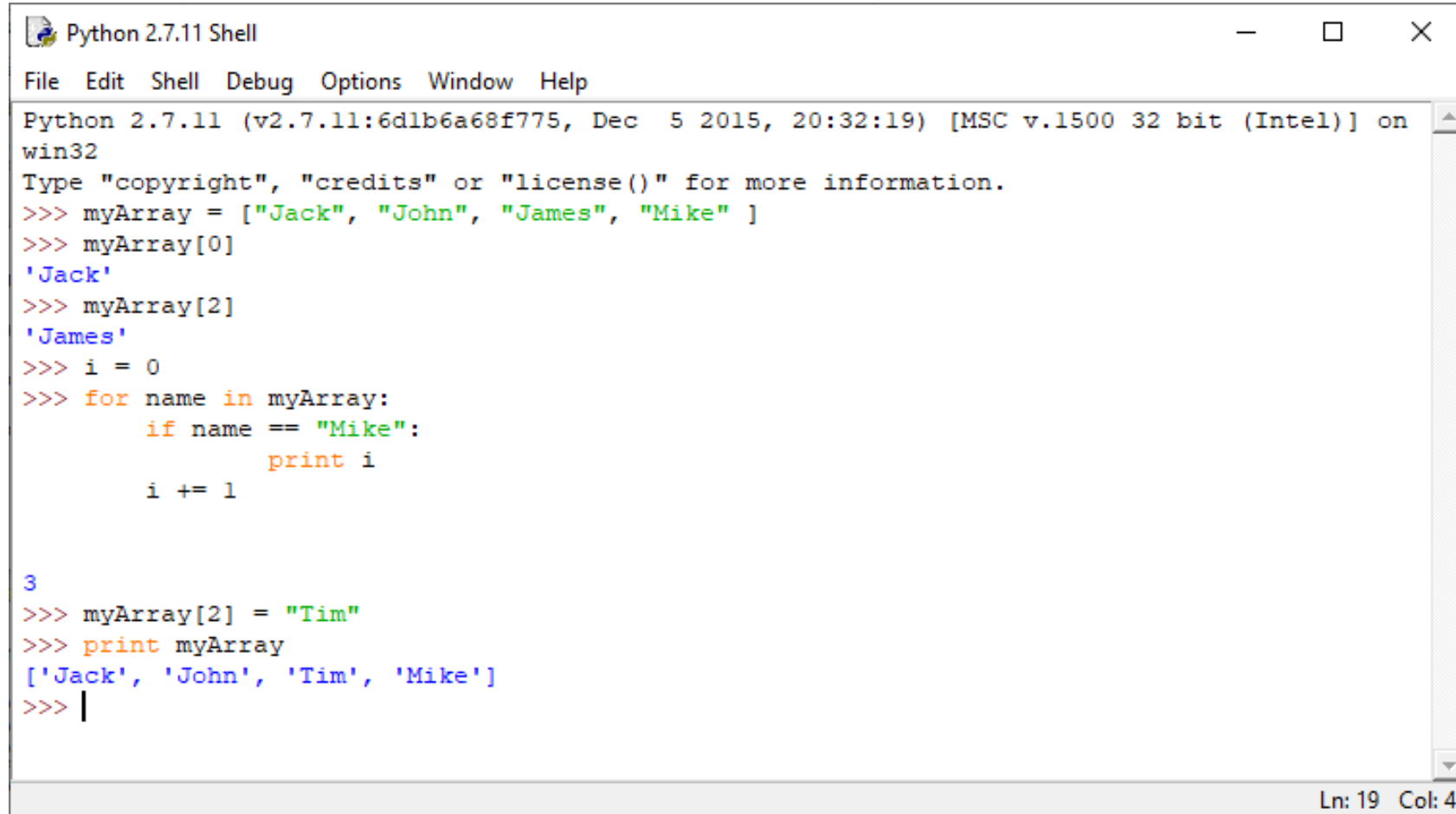
Arrays

- An array is a contiguous block of data
-(a bunch of data next to one another without any spaces), think of a list of items.
- We know that items stored in an array follow one another, there is a first item, second item, third item etc up to a final item.
- We can store, or change the value in the array using an index which specify the position values are stored at.
- The index is an integer is between 0 and the length of the array – 1 (the last value in the array)
- To get a data out of an array we need to do one of the following:
 - Know the specific index at which the data is stored.
 - Loop through the array until we find the data we are looking for.

[0]	[1]	[2]	[3]	[4]
73	98	86	61	96

arr[0]	→	73
arr[1]	→	98
arr[2]	→	86
arr[3]	→	61
arr[4]	→	96

Array Example in Python:

A screenshot of a Python 2.7.11 Shell window. The window has a title bar with the text "Python 2.7.11 Shell" and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main area of the window contains a Python interpreter session. The session starts with the version and environment information: "Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:32:19) [MSC v.1500 32 bit (Intel)] on win32". It then shows the user typing several lines of code: creating a list named "myArray" with elements "Jack", "John", "James", and "Mike"; accessing the first element "myArray[0]" which returns "Jack"; accessing the third element "myArray[2]" which returns "James"; initializing a counter "i = 0"; and using a for loop to iterate over "myArray", printing the index "i" for each element. The output shows "0" for "Jack", "1" for "John", "2" for "James", and "3" for "Mike". After the loop, the user changes "myArray[2]" to "Tim" and prints the entire array, which now shows "['Jack', 'John', 'Tim', 'Mike']". The status bar at the bottom right indicates "Ln: 19 Col: 4".

```
Python 2.7.11 Shell
File Edit Shell Debug Options Window Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:32:19) [MSC v.1500 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.
>>> myArray = ["Jack", "John", "James", "Mike" ]
>>> myArray[0]
'Jack'
>>> myArray[2]
'James'
>>> i = 0
>>> for name in myArray:
    if name == "Mike":
        print i
    i += 1

3
>>> myArray[2] = "Tim"
>>> print myArray
['Jack', 'John', 'Tim', 'Mike']
>>> |
```

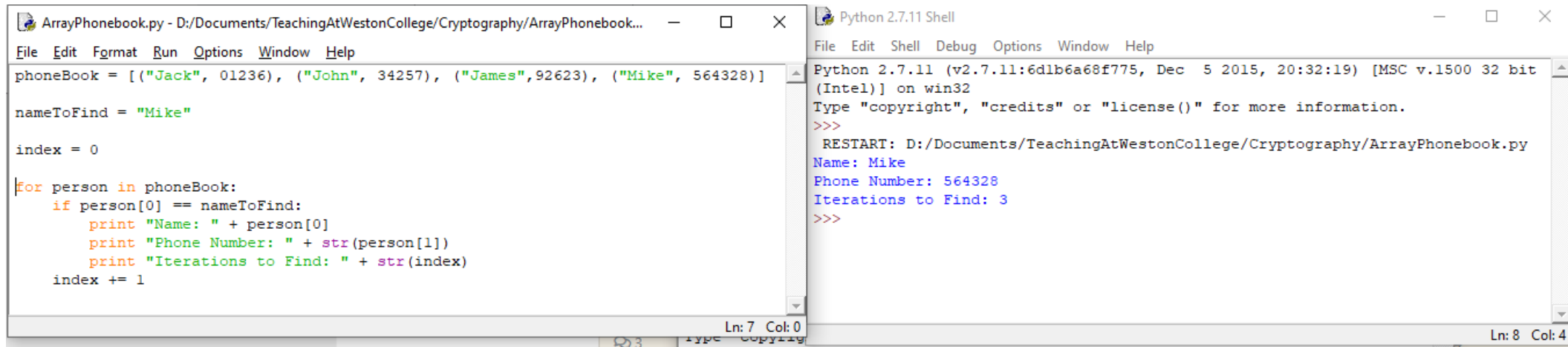
Ln: 19 Col: 4

Exercise (Assuming we've done programming):

- Say I want to program a phone book where I have:
 - A name
 - A phone number associated with that name
- I then want to give my program a the name of a person and have it give me back the phone number
- How would I do this?
- Have a quick think!

The problem with Arrays.

- The only information I can use to go straight to an item in an array is the index.
- As a human I can't remember my contacts using indexes, I only know them by their names.
- In other words I can remember the name but not the page number in the phone book.
- If I can't remember their index I have to look through the whole phonebook until I find the name of the person I want to contact.
 - This is ok for a small phonebook, problem for a massive one, will be slow!



The image shows two windows from a Windows operating system. The left window is a text editor titled 'ArrayPhonebook.py' containing a Python script. The script defines a list 'phoneBook' with four tuples, each containing a name and a phone number. It sets 'nameToFind' to 'Mike' and 'index' to 0. A 'for' loop iterates through 'phoneBook', and inside, an 'if' statement checks if the current name matches 'nameToFind'. If it does, it prints the name, phone number, and the current index, then increments the index. The right window is a 'Python 2.7.11 Shell' showing the execution of the script. It displays the output of the print statements: 'Name: Mike', 'Phone Number: 564328', and 'Iterations to Find: 3'.

```
ArrayPhonebook.py - D:/Documents/TeachingAtWestonCollege/Cryptography/ArrayPhonebook...
File Edit Format Run Options Window Help
phoneBook = [("Jack", 01236), ("John", 34257), ("James", 92623), ("Mike", 564328)]

nameToFind = "Mike"

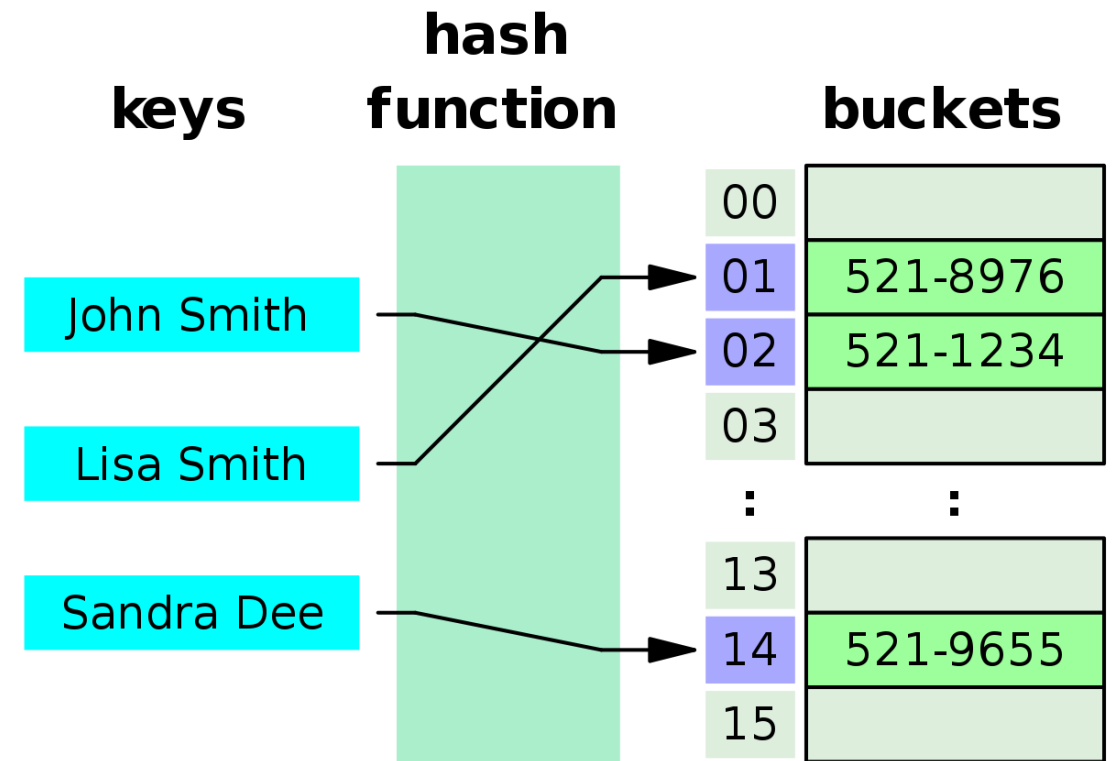
index = 0

for person in phoneBook:
    if person[0] == nameToFind:
        print "Name: " + person[0]
        print "Phone Number: " + str(person[1])
        print "Iterations to Find: " + str(index)
        index += 1
Ln: 7 Col: 0
```

```
Python 2.7.11 Shell
File Edit Shell Debug Options Window Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:32:19) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:/Documents/TeachingAtWestonCollege/Cryptography/ArrayPhonebook.py
Name: Mike
Phone Number: 564328
Iterations to Find: 3
>>>
Ln: 8 Col: 4
```


Hash Tables: The solution!

- The most basic example of Hashing
- Can map one kind of data to another in a single computing operation (super fast)
- Data doesn't need to be in order
 - There's no order relationship between a name and a phone number
 - So long as I can get from the name to the number it's all good.



What is a Hash Table?

- A hash table is what is known as a **Data Structure**
- A **Data Structure** is a structure in computer memory with certain valid operations that can be applied to it.
- An array is also a Data structure.
 - We know how data is stored in array
 - We're only allowed to do certain things with an array

The Hash Table

- Essentially the idea is we have an array and we access it via some information other than a number index.
- In the case of the phone book we can access it using a name.
 - E.g. `hashTable["Jack"]`
- A hash table is built using an Array to store the data but It has extra functionality that allows the data to be accessed differently.

The problem with accessing data

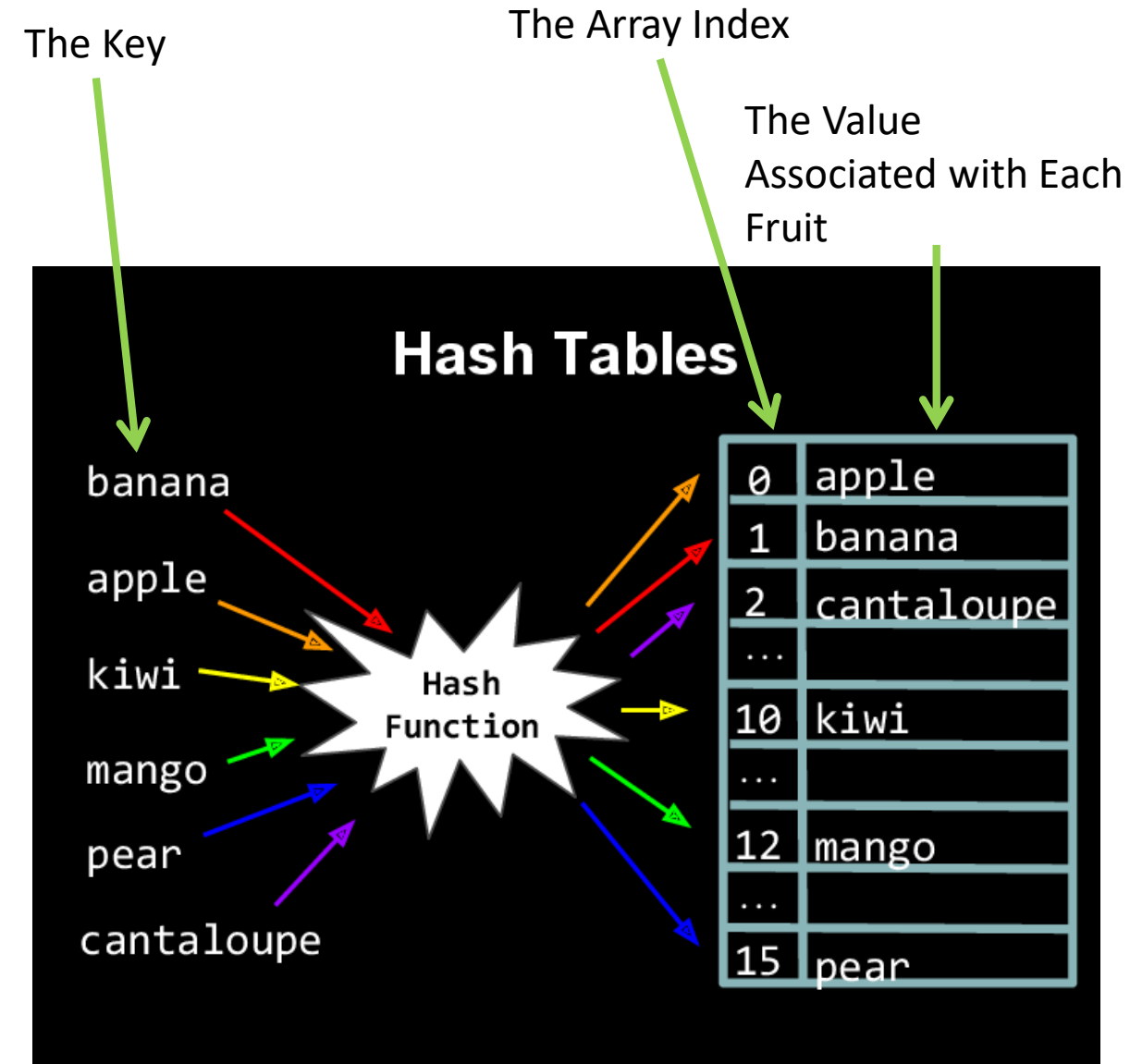
- In an array we can reference the position using the number index, its position relative to the other values stored inside it.
 - E.g. 0, 1, 2, 3, 4, 5...
- So if I say `array[2]` I'm referencing the 3rd position in the array (Arrays start at 0).
 - This is really easy for the hardware in the computer to do.
 - Computer memory is physically laid out like this.
- If I wanted to do `hashTable["Jack"]` there's nothing about the string "Jack" that refers to the position in memory at which the data about Jack is stored at.
- In other words `hashTable["Jack"]` doesn't make any sense to the computer by default.
- The computer needs to be told how to find the phone number associated with "Jack"

```
1 myArray = ["Jack", "John", "James", "Mike"]
2
3 myArray["John"]
```

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    myArray["John"]
TypeError: list indices must be integers, not str
```

The Hash Function.

- To solve this problem we need what's called a **hash function**.
- The hash function is responsible for converting some arbitrary data type into an index which we can use to find the data in the array.
- This index is then used to access the data in the array associated with "Jack" and retrieve the **Value** for "Jack"
- We say "Jack" is the **key** and the phone number for Jack is the **value**
- The hash function relates the key to the value



HashTable Example in Python

- You can see I've made a hash table in Python called **phoneBookHashTable** which contains some names and phone numbers.
- I access the table using "Mike" as the **key**
- The HashTable construct within python then uses a hash function behind the scenes to convert "Mike" into an index, this is hidden.
- It then accesses an array stored within **phoneBookHashTable** (also hidden) using that index.
- The **value** I get back is the phone number associated with Mike.

```
HashTablePhonebook.py - D:/Documents/Teac...
File Edit Format Run Options Window Help
phoneBookHashTable = {"Jack": 01236,
                      "John": 34257,
                      "James": 92623,
                      "Mike": 564328
                      }

nameToFind = "Mike"

value = phoneBookHashTable["Mike"]

print "Key: " + nameToFind
print "Value: " + str(value)

Ln: 9 Col: 34
```

```
Python 2.7.11 Shell
File Edit Shell Debug Options Window Help
Python 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:32:19) [MSC v.1500 32 bit (Int
el)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:/Documents/TeachingAtWestonCollege/Cryptography/HashTablePhonebook.py
Key: Mike
Value: 564328
>>>

Ln: 7 Col: 4
```

Properties of a Hash Function

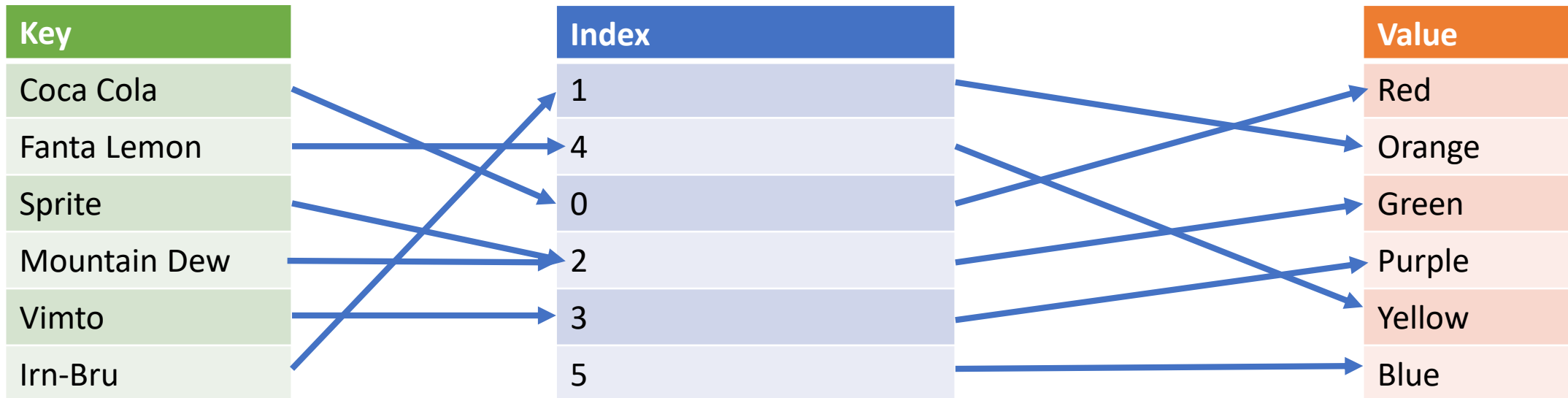
- A hash function must avoid converting two pieces of input data into the same index, in other words
- “Jack” -> 5
- “John” -> 5
- Should ideally never happen,
 - This is called a hash collision.
 - Key no longer corresponds to unique value.
- What would happen if it did do this?
- The input value e.g. “Jack” should also never result in an index bigger than the list of telephone numbers.
 - The hash function should always map a key to a valid index

Generation of hash functions

- When we're building hash tables the hash functions are typically generated automatically by the library we use when programming.
- How this is done depends on the input and the output data and the amounts of data used, it's quite a complicated topic!
- Would be possible to do many lectures just on this alone so for now you just need to know the basic concepts why they're important.

Exercise:

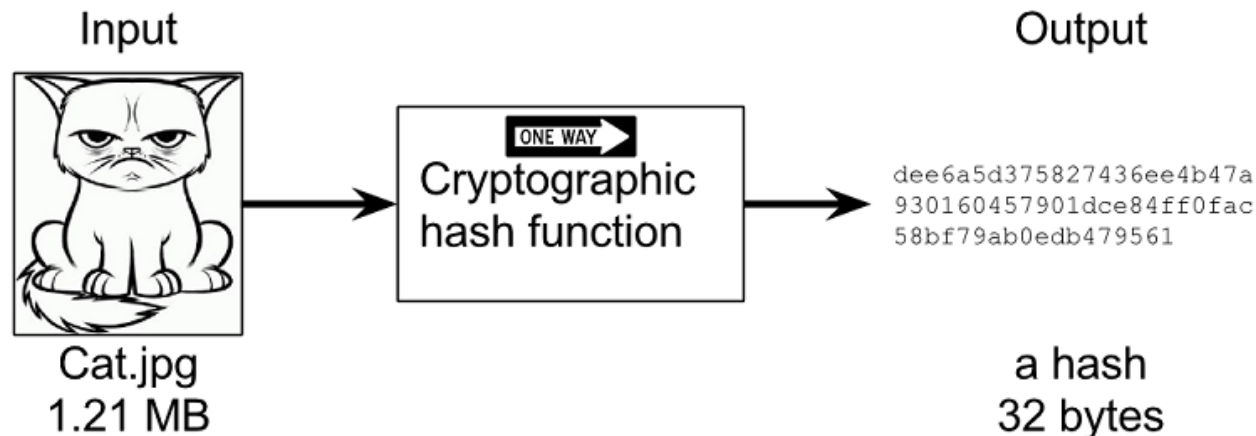
- Is the following Hash Table valid?
- Why is/ isn't it valid?



Cryptographic Hashing

Cryptographic Hashing

- Hashing is useful in cryptography.
- Allows us to authenticate data.
 - Authentication is the process of us proving or showing that data is in some way genuine.
- Another example of a one way function, easy to perform in one direction, hard in the reverse.
- No longer hashing to an index but hashing to an independent value



Cryptographic Hash Functions

- A cryptographic hash function takes an input **Key** and generates a pseudo random string value, **not an index**.
- If the input key changes even slightly then the output string is completely different -**Diffusion**
- All keys generate an output of the **same length**.
- We never decrypt a hash value as we would a message, hashing is a one way process.
- Hashes generated using SHA-1 algorithm:

Name: "Jack" -> "BC5351FFAE3EFE8067951F5DEBA4B294BF863F86"

Name: "John" -> "5753A498F025464D72E088A9D5D6E872592D5F91"

Example 1 / Exercise: Passwords

- Suppose I enter my password into a website in order to log into it
- Can you think of any reasons why the website storing my password in plain text is a problem? E.g. “JackBradbrook01Password”
- What could happen,
take 5 minutes to come up with some ideas.



Possibilities

- If my password is stored in plain text on the websites database anyone with access to the database can read it.
- How do I know the Admin isn't going to steal my password?
- What if the website experiences a security breach and the data is stolen by a hacker?

The Solution... You guessed it!

- If my password is hashed and the hash value is stored it's not possible for a hacker to figure out what my actual password was from the hash value:
- E.g. "JackBradbrook01Password" -> "Qhw12h432aMR8P2"
- Administrator of the site can only see the hash values if they access the database, they cannot steal my password.

Always hash passwords!

- Any systems involving user passwords should always hash those passwords, doing otherwise is absolutely not acceptable in 2019.
 - When user enter their password on a system, the password should be hashed immediately and compared with the hash value stored in the database.
- TalkTalk hack
 - Stored all the user passwords as plain text in SQL database, hacker injected SQL query -> returned the passwords in plain text
 - This Cost TalkTalk millions of pounds in damages.
 - Tarnished the companies reputation completely.



Example 2:

File Authentication with Checksums

- Say we download a file off the internet.
- How can we check that the file we have downloaded is actually the file we are after and not a malicious file that claims to be it?
- How can we trust that somebody is who they claim to be?
- The answer... Hashing!

Authentication

- If we hash a file then we know that if we change one part of the file then the hash value for the whole file will be very different
- We can use a reliable hashing algorithm to generate a hash for our downloaded file.
- We can check that this value is the same as the value associated with the file provided by another source.


Demo:

<https://www.python.org/downloads/release/python-374/>

<https://www.virustotal.com/gui/home/search>

Exercise

1. Find an Image, generate a hash value for it using the following website:
2. <http://onlinemd5.com/>
3. Cut and paste the value into the “Compare With” field

File checksum:	<input type="text"/>
Compare with:	<input type="text" value="2A4D3BDB7EA7CD6A9E0793F33134183A"/> 

4. Make a copy of that image and modify it just slightly using MS paint
5. Use the website to generate a hash value for the second image
6. Compare the hashes using the “File checksum” and “Compare with” field
7. What do you notice?
8. Do this again but re upload the original Image again or make another change

Hash Collisions

- If we have an input of some size x and generate a hash of some size y .
- If $x > y$ then at least some of the possible inputs that can exist will result in the same hash value.
- For example (Both Input Strings Length 48):

“Jack teaches his lectures every Friday afternoon” -> “BC5351FFAE”

“BkYPqDpYPDjxxxh8YMrqzKoBoc7Yk8ytaRiyXorK8qCdlccg” -> “BC5351FFAE”

Exercise: Speed of Cryptographic Hashing

- We don't want cryptographic hash algorithms to run too fast.
- Can anybody guess why this is?
- Think about what an attacker can do if they know the hash algorithm used and can run it quickly.
- What are the implications of the fact that hash collisions exist?

The problem with collisions

- If an attacker can find another input value which generates the same output hash then they can potentially modify a file without breaking the authentication.
- I could therefore provide a download to Python which runs as a virus on your computer but has the same hash value as the real download.
- For all you know the python files you've downloaded are the legitimate ones.

Collision resistance

- Modern cryptographic hash functions are designed so that it's almost impossible to find two input values which result in the same Hash value.
- Would realistically take a fast computer lots of time with a brute force approach to find another input which produces the same output.
- Two common algorithms used:
 - MD5 – Old, now vulnerable due to more powerful computers.
 - SHA-1 – Was the current standard, now being replaced by SHA-2 and SHA-3 (similar to SHA-1)

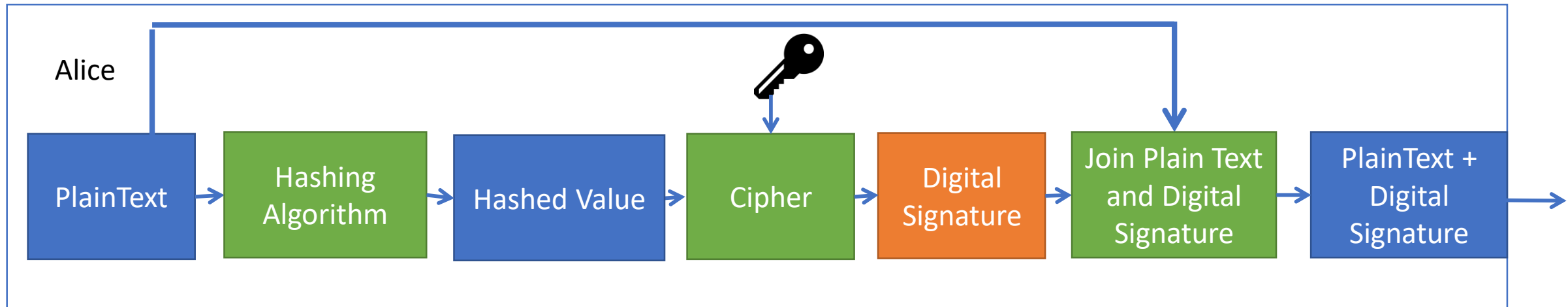
Digital Signatures – Bringing Everything
Together!

Digital Signatures

- A Digital Signature is equivalent to a handwritten signature.
- It is electronic verification of the sender of a message.
- A Digital Signature has three purposes:
 1. Authentication allows the receiver to have confidence that a the person sending the message is who they claim to be. (as we just saw with file downloads)
 2. Non-repudiation: The sender cannot deny having sent the message later on.
 3. Integrity: the signature ensures that the message was not meddled by a third party during communication.

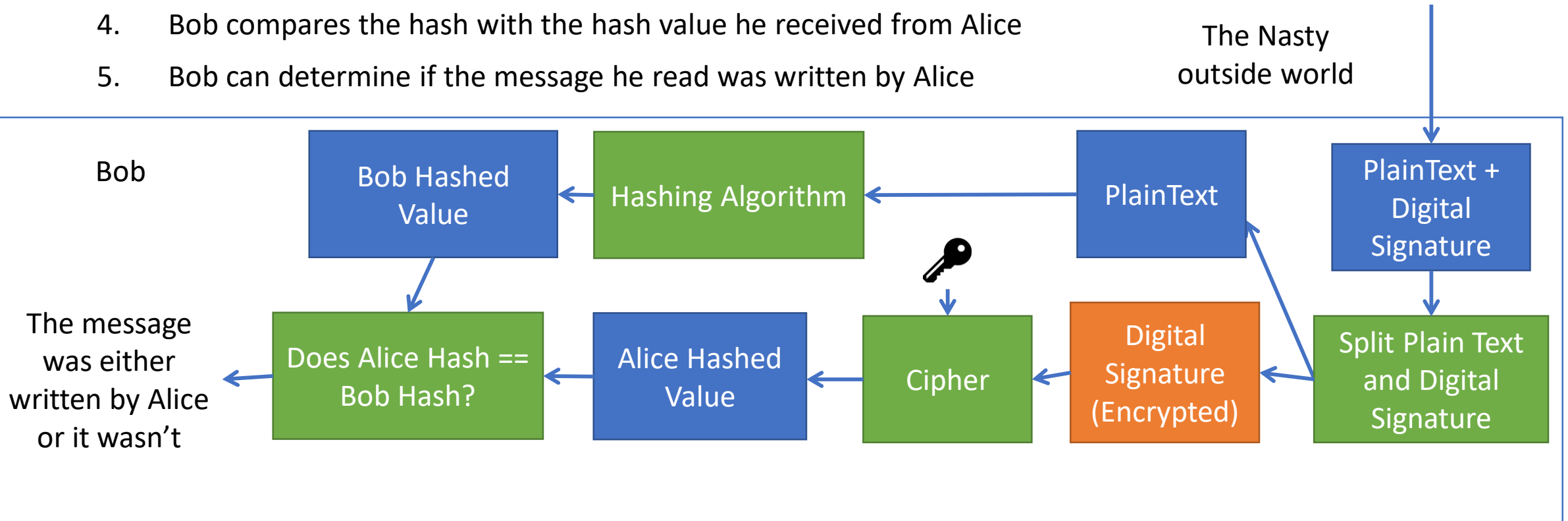
Digital Signature: How it works

1. Alice is going to send a message to Bob and provide a Digital Signature
2. Alice and Bob prepare to use an Asymmetric Encryption system like Diffie-Hellman (see last lecture)
3. They each make their private keys and agree on public keys as per usual
4. Alice makes a Hash value of the message she wants to send to Bob
5. She then encrypts the Hash value (not the message), just the hash value, to produce the digital signature
6. She attaches her digital signature to the message and sends both of them to Bob



Digital Signatures: How it works, part 2

1. Alice sends the plain text of the message with the encrypted hash value.
2. Bob Decrypts the digital signature using the public key and his private value
3. Bob hashes the plain text message using the same hash algorithm
4. Bob compares the hash with the hash value he received from Alice
5. Bob can determine if the message he read was written by Alice



Uses of Digital Signatures

- We use Digital Signatures for all kinds of stuff but some common uses are:
 - Emails
 - VPN Tunnels (Next lecture)
 - Software management
 - Online Banking
 - Online shopping
 - Verifying stock trades.
 - Auditing.



From the past paper

- So given what we now know lets have another crack at the following questions:

8 A way of verifying both the sender of information and the integrity of a message is using which of the following?

- A Digital signatures.
- B Digital certificates.
- C Public key encryption.
- D Private key encryption.

9 Why would a user check that a software patch is **CORRECTLY** signed by the software publisher?

- A To ensure that the user does not exceed the software license.
- B To ensure the patch downloaded correctly.
- C To ensure that it is a legitimate patch issued by the correct party.
- D To ensure the patch is compatible with the user's version of software.

10 Which of the following is the **CORRECT** description of ciphers?

- A Stream ciphers encrypt continuous streams of data.
- B Block ciphers encrypt blocks of data of variable size.
- C Polyalphabetic substitution ciphers keep the substitution alphabet constant for every symbol.
- D Transposition ciphers take groups of characters and shift them according to a random system.

14 Which of the following is **NOT** a block cipher operating mode?

- A Cipher Block Chaining (CBC) mode.
- B Electronic Codebook (ECB) mode.
- C Cipher Feedback (CFB) mode.
- D Asymmetric Key Cipher (AKC) mode.

Answers to Exam Questions:

8: We know a digital signature is a way to verify both the sender and the message integrity, **so the answer is: A**

9: We check a software patch is signed so we know that the patch is legitimate and we are actually downloading it from the software company instead of a fake, **So the answer is C**

10: We know block ciphers work on blocks of the same size so it's not B, we know stream ciphers do encrypt streams of data, **so the answer is A**

14: We now know what the three most common block cipher chaining modes are (ECB, CBC and CFB) we also know that block ciphers are symmetric ciphers, not asymmetric, **so the answer is D.**

Any Questions?

- We can Review these topics again in later weeks if people want a recap.
- I'd recommend going through the PowerPoint once or twice more by yourself and take some notes, you will probably forget otherwise.
- Having a working understanding of each topic yourself is how you become resilient to any tricky questions that may appear in an exam situation.

Enigma Machine Video

- https://www.youtube.com/watch?v=G2_Q9FoD-oQ