Asymmetric Cryptography

Jack Bradbrook

Feedback on last week

- How was the pace?
  - Too fast/ too slow?

- Did you feel topics not enough detail others too much?

- What did you enjoy, if anything, what wasn't so interesting?

- Was there too much content or would you have liked more than what was presented?

- What would you liked to have seen?

One thing I forgot from last week

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

Topics

- Digital Cryptography Vs Classical Cryptography
- Symmetric Vs Asymmetric cryptography
- The Diffie-Hellman Key Exchange
- Kerckhoffs's principle

Digital Cryptography Vs Classical Cryptography

- Encrypting bytes instead of Encrypting pain text, this means you could be encrypting anything which can be represented in binary, video, audio etc.

Problems posed by modern Digital Cryptography:

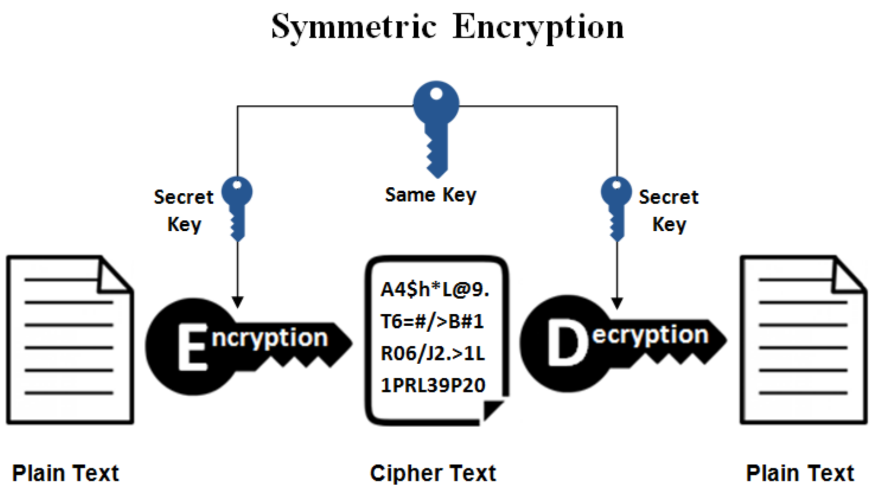
- Cannot physically transfer keys from one person to another as was the case with Enigma. Communication, verification and transactions need to be able to happen quickly across the world between parties which have never met.

- Security cannot rely on keeping the cipher secret, algorithms can be examined and transferred easily over the internet if leaked.
  - Algorithms need to be carefully checked and peer reviewed in order for them to become trusted by a large number of people.

- Encryption and Decryption need to be performed efficiently to be used for modern applications and on low power mobile devices

Symmetric Key Cryptography

- Symmetric cryptography: one key used to encrypt and decrypt everything e.g. The Caesar cipher or the enigma system, both parties need to know the same key for encryption and decrypt, if a third party also knows it then the system is compromised.



The Problem with Symmetric Key Cryptography

- Both parties need to already know the key!
- How are you supposed to tell another party at the other side of the world what the key is without a third party being able to potentially see the key and then decrypt all your messages?
- You need another system to securely share your Symmetric Key with the other party before you can start safely communicating using with it.
- In cryptography this is known as “The Key Exchange problem”

Asymmetric Cryptography: Diffie-Hellman key exchange

- ***Asymmetric cryptography*** uses multiple keys, which are either public or private.

- ***The Diffie–Hellman key exchange*** was invented in 1976 to solve the “The Key Exchange problem”, one of the earliest and simplest methods of Asymmetric Cryptography.

- Relies on the idea of a ***one way function***.
  - A function where given a value x calculating $f(x) = a$ is easy
  - But given a calculating x using the inverse of f: f^{-1} is difficult:
  - $f^{-1}(a) = x$ <- this is hard!

How it works

- Two public values are agreed which act as the public key, a generator, commonly known as *g*, and a prime number known as n both parties, and any third party can see the value of *g*.
- Two private keys are created, typically referred to as *a* and *b*, each private key is only known to **the party that created it** and **never** shared with the other party.
  - can therefore never be known by a third party!
- Party 1 (Alice) combines their private key *a* with *g* **using a one way function** to create *ag*
- Party 2(Bob) combines their private key *b* with *g* **using a one way function** to create *bg*

How it works

- Each party shares the combination of its private key and the generator with the other party.

- So both *ag* and *bg* are made public knowledge

- Alice privately knows *a* and publicly knows bg
  - So Party 1 can combine both of these to effectively get *abg*

- Bob privately knows b and publicly knows *ag* so can combine b and *ag* to get the same abg that Alice has, this is the symmetric key.

Why is this secure?

- As a third party I know *ag* and *bg* but not *a* or *b* because a one way function is used to generate *ag* and *bg* It's extremely hard for me to uncover what either *a* or *b* from *ag* or *bg*.

- Therefore I cannot combine *ag or bg* with the corresponding *b* or *a* to get *abg* which is the key that Alice and Bob want to exchange

- A visual Example:

https://youtu.be/MsqqpO9R5Hc

The Mathematics behind Diffie-Hellman

- Don’t worry about not understanding this straight away, we will review it. Can take time to sink in.

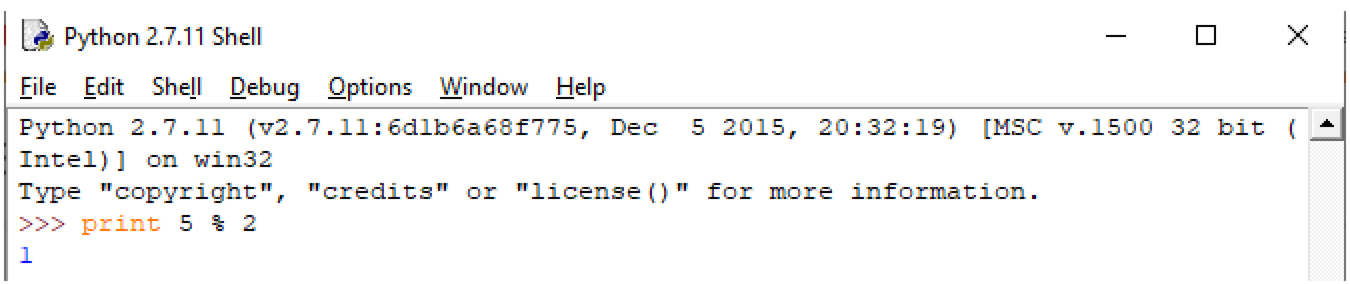
- Try to get your head around the general concepts.

Important mathematics concepts

- To understand how this process is performed using maths we need to first appreciate the following concepts:
  - The Modulo operator – how we “lock” the message
  - Exponent properties – how we “unlock” the message

The modulo operator

- Do people already know how this works?
- Fizz buzz anyone?
- When doing computer programming the “mod” operator is typically represented using % sign like so:



How modulo works

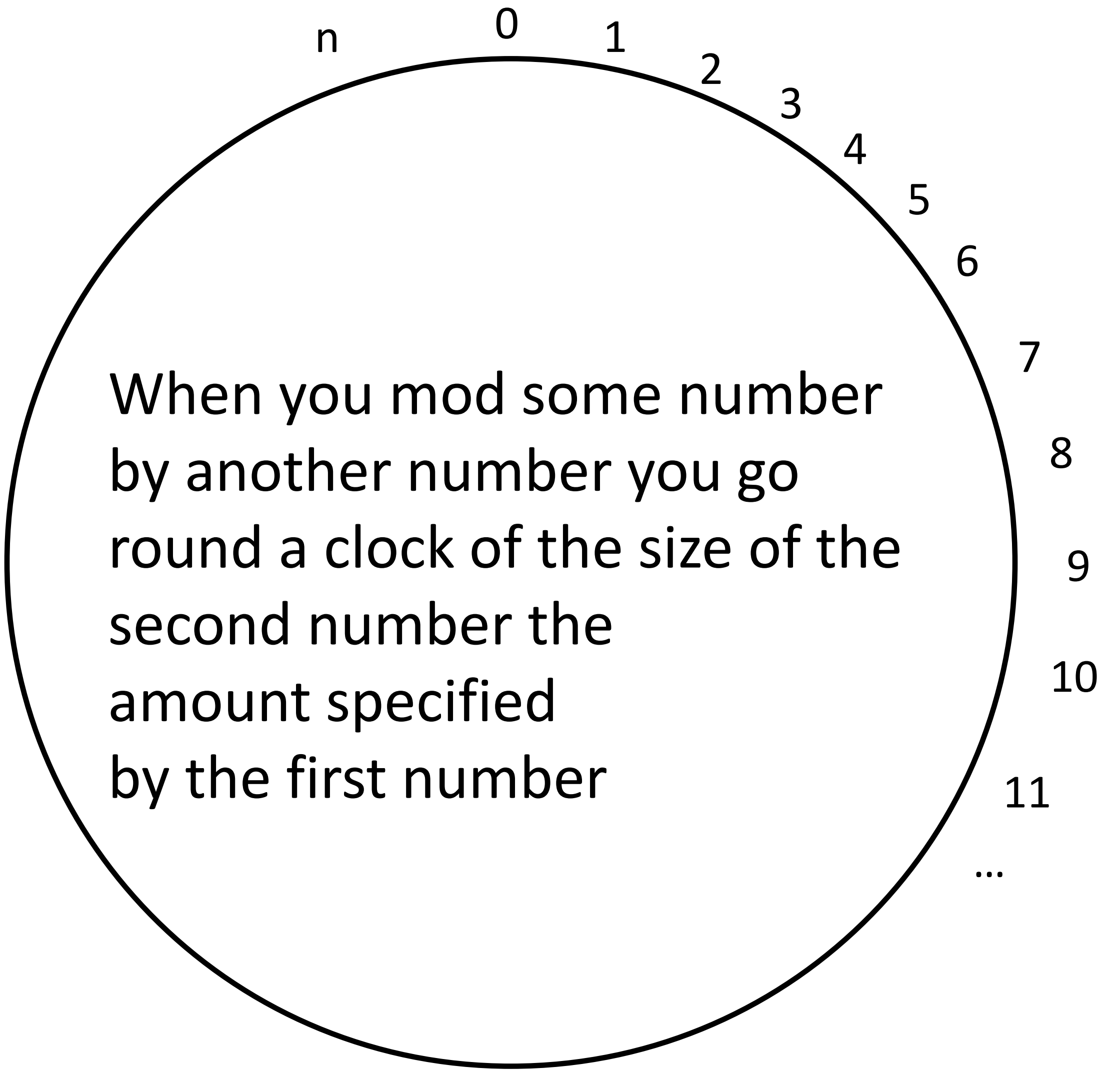
- 5 mod 2
  - This is the same as asking “What is the remainder value when I split the number 5 into groups of 2”
  - 5 when broken down into groups of two is made up of two groups of two with a remainder of one. Note 5 cannot be made simply out of 2s (it’s not a multiple of 2)
- 5 = 2 + 2 + 1
- Therefore 5 mod 2 = 1 (The remainder)
- If a number is a multiple of the value it is being moduloed by then the resulting value is 0
  - E.g. 6 mod 2 = 0

Another way to think about modulo

- Mathematics using modulo is also called clock arithmetic.

- If you think about seconds since the start of the month you can get
  – The minutes as seconds mod 60
  – The hours as minutes mod 60
  – The Days as hours mod 24

Visualising modulo



When you mod some number by another number you go round a clock of the size of the second number the amount specified by the first number

Why is modulo useful?

- Can be used to convert an input number of any size to another number between a specific maximum and minimum size:

- E.g. 2982 mod 12 = 6

- I.e. Any input number mod 12 will always be between 0 and 12 (again think about a clock)

- This is useful in cryptography because it makes it very easy to convert some input number say 2982 to 6 but very hard to do the opposite and determine a key value such as 2982 from 6 without using a computationally expensive brute force search.

Exercises: have a go!

- What is:
  - 28 mod 4
  - 9 mod 6
  - 13 mod 1
  - 5 mod 13

Answers

- 28 mod 4 = 0
  - 4, 8, 12, 16, 20, 24, 28
- 9 mod 6 = 3
  - 6, then we have to add 3 to 6 to get 9.
- 13 mod 1 = 0
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
- 5 mod 13 = 5
  - We go round the clock of 13 hands by 5 so we have 5 left over at the end, we never complete a full round of the clock.

Exponents recap

- When we write 2^3
- we are saying raise 2 to the power of 3.
  - We refer to 3 as the “exponent” of 2
  - 2 is the “base” of the exponent
- In other words $2^3 = 2 \times 2 \times 2$
  - Multiply 2 by itself 3 times

Exponent properties

- There are various properties of exponents we can use when performing algebra involving them

- One such property is useful in cryptography:

- If we have an exponent (5^2) i.e. 5 x 5 and we raise that exponent to another exponent:
  - e.g. $(5^2)^3$ we can simplify this expression by multiplying the two exponents e.g. $5^{2*3} = 5^6$

  - In other words $(5^2)^3 = 5^6 = 15,625$

Exercises

- Calculate: $(7^2)^4$ both using the exponent property to simplify the exponent first and without doing this (use a calculator)
  - If you don't have a scientific calculator use wolfram alpha https://www.wolframalpha.com/
  - Can write this the following way in the search box:

- Calculate $(11^5)^9$ mod 8
  - ask yourself is there any difference between this value and $(11^9)^5$ mod 8

- This last step is worth checking and thinking about to make the next part easier

Answers

- $(7^2)^4 = 5764801$
  - $7^8 = 5764801$

- $(11^5)^9 \bmod 8 = 3$
  - $11^{45} \bmod 8 = 3$

Diffie-Hellman variables

- Given the previous explanation where we start with n, g, a and b
- n (public) is a very large Integer (needs to be big)
- g (public) is typically a small prime number
- a (private) and b (private) are numbers chosen by Alice and bob respectively

Doing Diffie-Hellman maths

Alice	Public	Bob
a	n g	b
ag = $(g^{a})\text{mod } n$		*bg* = $(g^{b})\text{mod } n$

First Alice and bob need to generate *ag* and *bg* using the one way function,

The Diffie-Hellman process with maths

Alice	Public	Bob
a	n g	b
$(g^{a})\text{mod } n$		$(g^{b})\text{mod } n$
$(g^{b})\text{mod } n$		$(g^{a})\text{mod } n$

Bob sends Alice the value of *ag*

And Alice sends bob the value *bg*

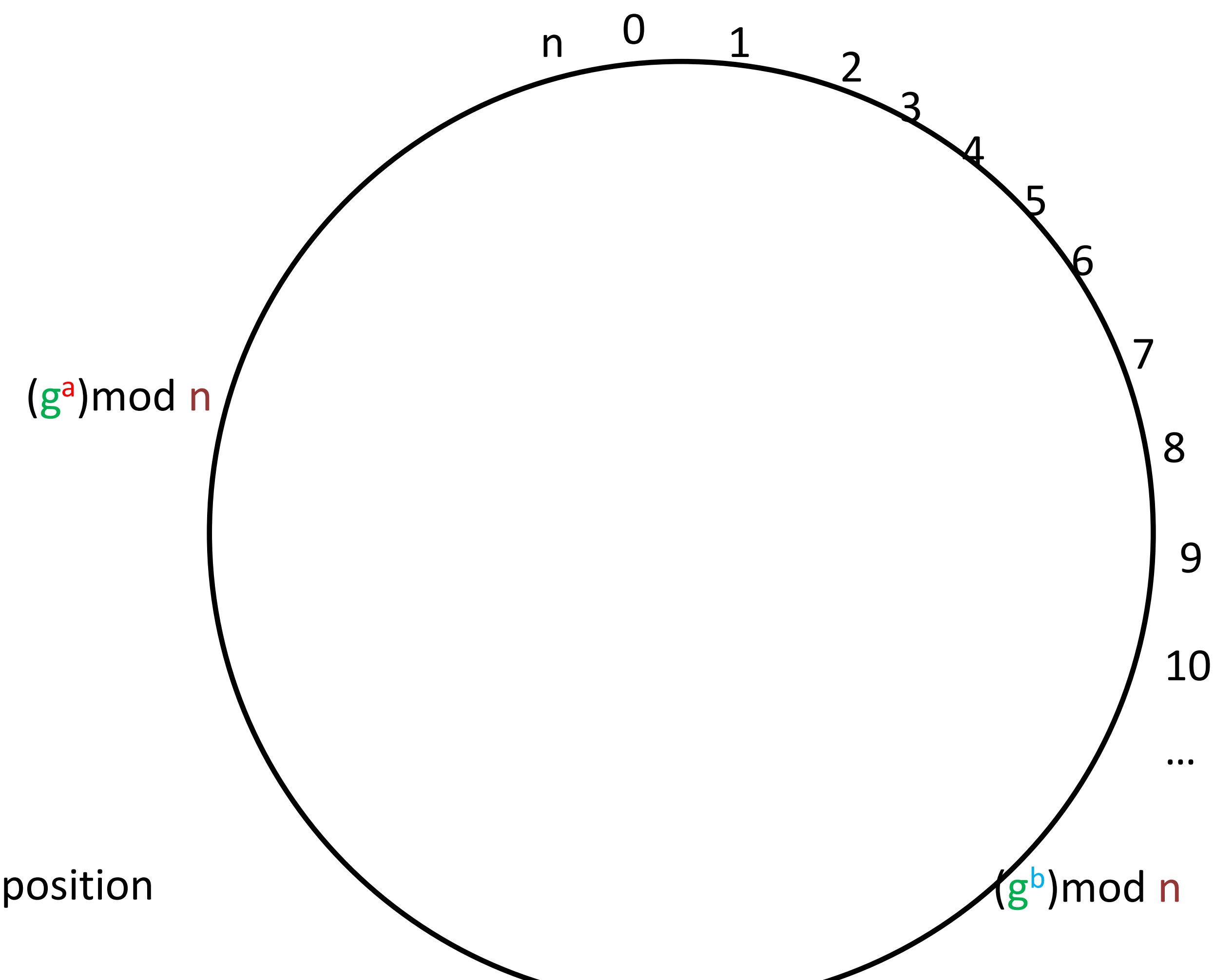
Why $(g^x) \bmod n$ is secure

- Because (g^a), (g^b) and n are huge numbers and the modulo operator makes the function cyclic, it's extremely hard for an observer to determine a or b from the result of

$(g^a) \bmod n$

Or

$(g^b) \bmod n$



The resulting values

Can be anywhere

around the clock,

without a or b

Indicating the clock position

Doing Diffie-Hellman maths

Alice	Public	Bob
a	n g	b
bg = $(g^b) \bmod n$ $bg^a \bmod n$ = *abg*	Once the values have been exchanged the final step can be taken so both Alice and Bob can generate the same *abg* value Note properties of exponents	*ag* = $(g^a) \bmod n$ $ag^b \bmod n$ = *abg*

The final calculation

- So Alice performs the overall calculation:
- $abg = (g^b \bmod n)^a \bmod n$
  - But she sees this part as a single value (This is calculated by Bob)
- Bob performs the calculation:
- $abg = (g^a \bmod n)^b \bmod n$
  - But he sees this part as a single value (This is calculated by Alice)
- Crucially: both of these values are always identical.

Exercise

- *Let $n = 11$,*
- *Let $g = 7$,*
- *Let $a = 2$,*
- *Let $b = 4$*

- *Calculate in turn*
- g^b mod n for Bob
- g^a mod n for Alice

- Then calculate the value *abg* for both bob and Alice and check the results are the same.

Answers

- Performing exercise in python

Diffie-Hellman.py - D:/Documents/TeachingAtWestonCol...

File Edit Format Run Options Window Help

```python
n = 11
g = 7
a = 2
b = 4
ag = pow(g, a) % n
bg = pow(g, b) % n

bob_abg = pow(ag, b) % n
alice_abg = pow(bg, a) % n

print "ag is: " + str(ag)
print "bg is: " + str(bg)

print "abgs are the same: " + str(alice_abg == bob_abg)

print "Alice abg: " + str(alice_abg)
print "Bob abg: " + str(bob_abg)
```

Ln: 3 Col: 0

Python 2.7.11 Shell

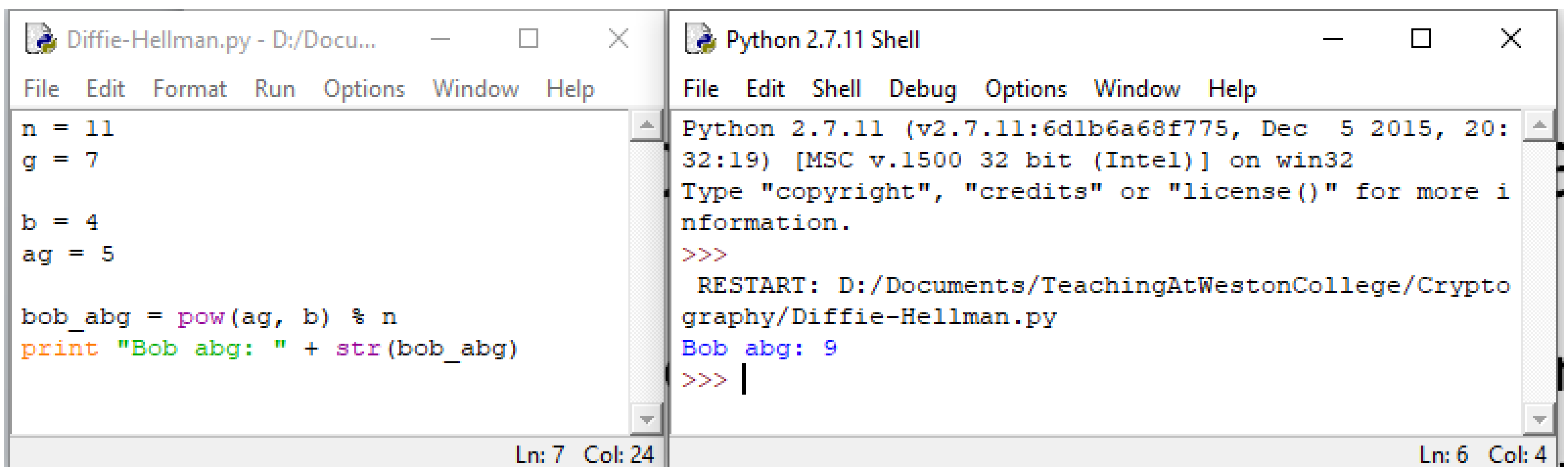
File Edit Shell Debug Options Window Help

```
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/Diffie-Hellman.py
ag is: 5
bg is: 3
abgs are the same: True
Alice abg: 9
Bob abg: 9
>>> 
```

Ln: 10 Col: 4

Another example to clarify

- Look at the modified code from bobs perspective
- Note how as Bob I can still calculate the value 9 with just the values *b* and *ag* even though I have no idea what *a* value was used to produce *ag*



Kerckhoffs's principle

- Kerckhoffs's principle: “A cryptographic system should be secure even if everything about the system, except the key, is public knowledge.”

- Crucially even though we know we all now know how Diffie-Hellman works and we know what information is made public for the key exchange we still can’t easily break it when it is used by others for sending messages

  – This also means we can ourselves implement the algorithm ourselves (just did in python) and check there isn’t a back door allowing somebody to read our messages without us knowing.

- For any modern cryptographic system to be considered “secure” it must adhere to Kerckhoffs's principle
  - Caesar cipher does not adhere, if we know how the cipher works we can easily break it.

Breaking the Encryption

- Say I'm trying to break the encryption
- The only way to do this is to figure out what the value of a or b is.
- To do this *ag* or *bg* has to be split up.
- I need to solve "The Discreet Logarithm Problem" to do this.

The Discreet Logarithm Problem

- If I know$(g^a) \bmod n = ag$

- I know everything but the private value a e.g.

$3^x \bmod 17 = 12$

- How do I calculate the value of x (the private key) from this?

- This is the process of trying to perform the inverse of the one way function mentioned earlier.

The Discreet Logarithm Problem

- $(g^a) \bmod n = ag$ is very hard when n is a large prime number

- The only way to get a is to try every value of it one after another until you get the correct value.

- Would take a super computer thousands of years to solve for a sufficiently large n value.

Man in the middle attacks

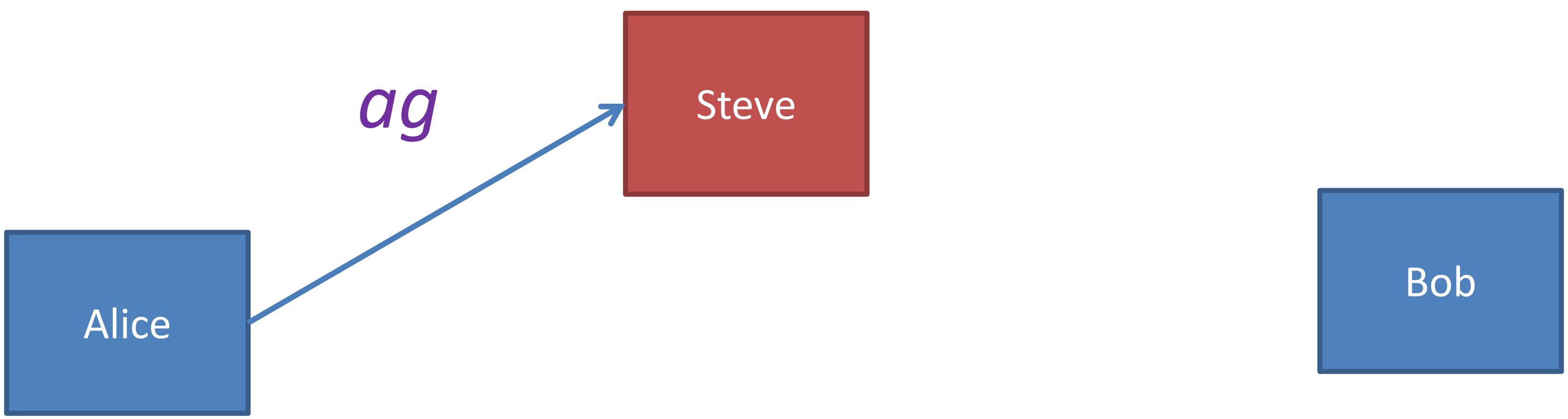
- The main problem with Diffie–Hellman is it Is vulnerable to what’s known as a “Man in the Middle Attack”

- This is when a third party say Steve pretends to be either Alice or Bob to the corresponding party

- I.e. Alice thinks Steve is Bob and Bob thinks Steve is Alice

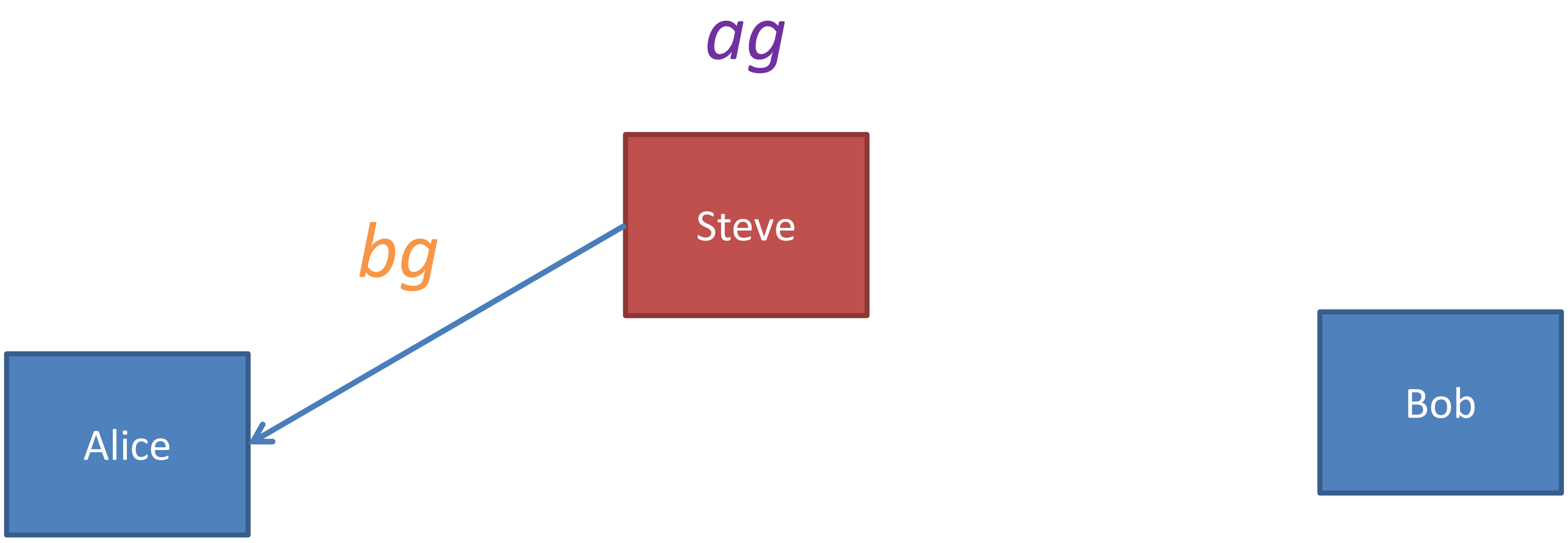
Man In the middle example step 1

- Alice calculates *ag* and sends this to Bob
- The message is received by Steve who prevents the message from being forwarded to Bob



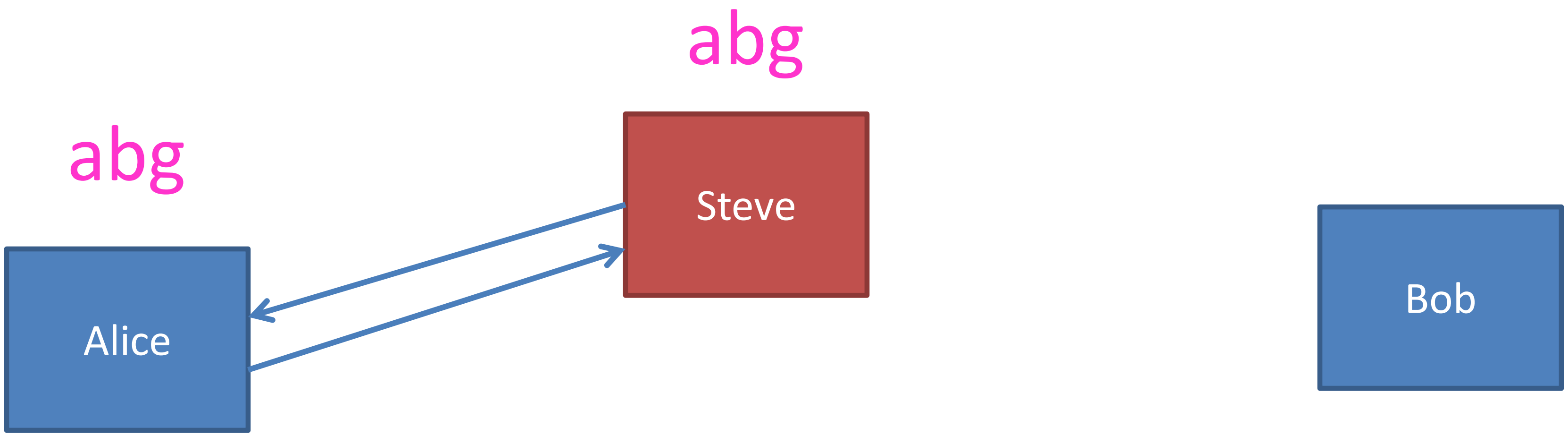
Man In the middle example step 2

- Steve then Generates his own *bg* value as per normal and sends it back to Alice



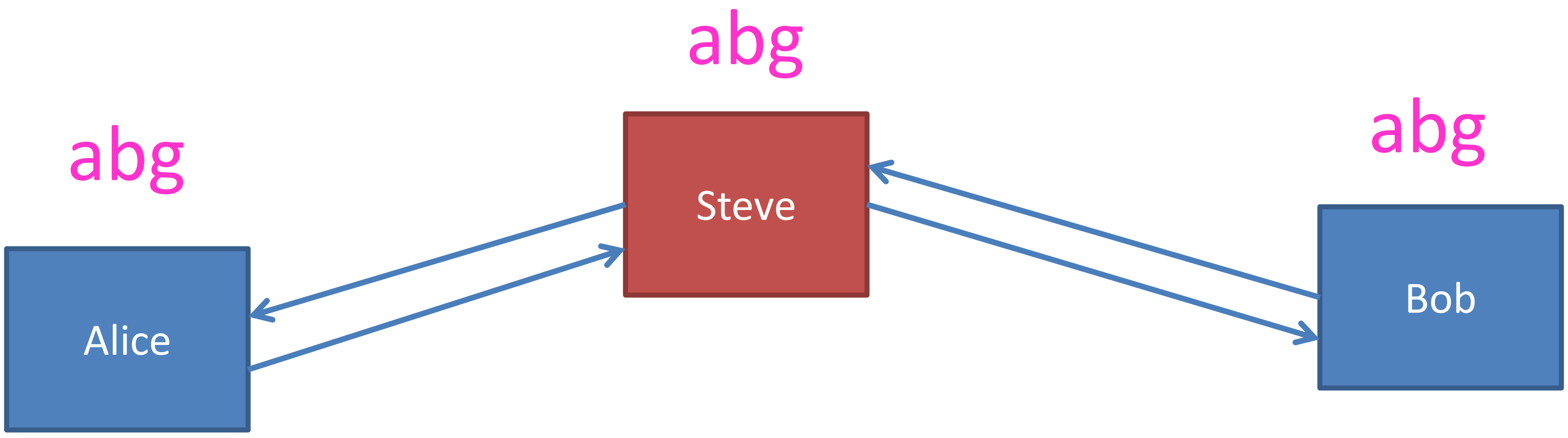
Man In the middle example step 3

- Both Alice and Steve Complete the key exchange and arrive at an abg value.
- They can now both communicate with one another using a regular symmetric cipher.



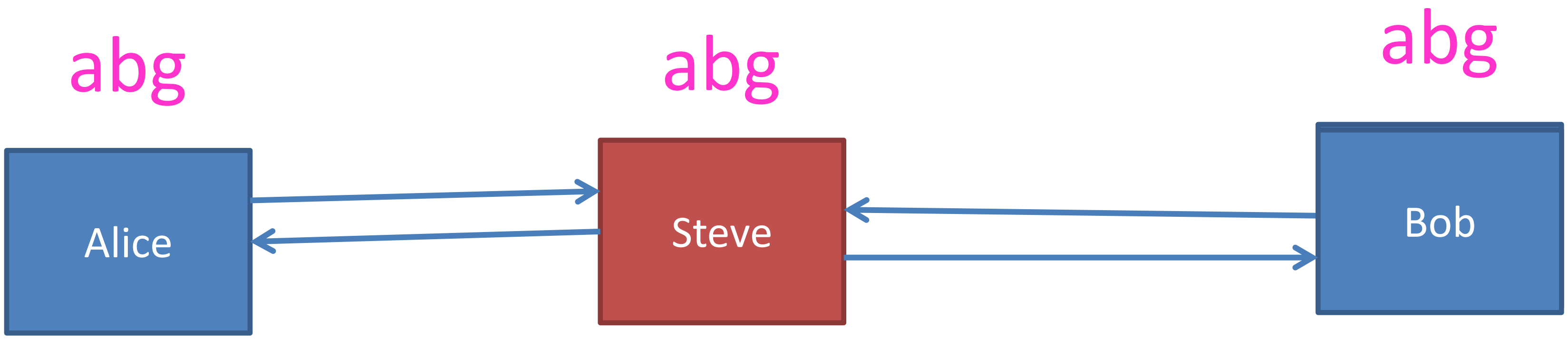
Man In the middle example step 3

- Steve then repeats the process for bob to establish communication with him
- Steve then forwards messages between Alice and Bob but crucially he has control!



Man In the middle example

- Steve can now do the following:
  - Read all of Alice and Bobs messages to one another
  - Prevent messages from Alice or Bob from reaching the other
  - Write his own messages which he can send pretending to be the other party



Diffie-Hellman extra material

- Further material on Diffie-Hellman:

https://youtu.be/MsqqpO9R5Hc

https://youtu.be/SL7J8hPKEWY