



Security - TLS

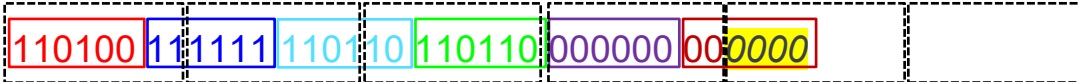
- ▶ On completion of this section, you should be able to:
 - ▶ Explain what base 64 is and the difference between it and base64url
 - ▶ Explain the properties, basic operation and use of the following security approaches:
 - ▶ Hashing
 - ▶ Message Authentication Code
 - ▶ Symmetric key encryption
 - ▶ Diffie-Hellman key exchange
 - ▶ Asymmetric key encryption
 - ▶ Digital Signature
 - ▶ Explain the properties, basic operation and use of a digital certificate
 - ▶ Explain how to request and deploy a digital certificate
 - ▶ Explain the chain of trust
 - ▶ Explain how the browser uses the certificate to establish trust with the server
 - ▶ Create and deploy a digital certificate for an application
 - ▶ Explain the TLS handshake process
 - ▶ Explain the generation and use of session keys

- ▶ This is a very broad subject
- ▶ We will focus only on the underlying principles of Transport Layer Security used to enable https
- ▶ https enables encryption in transit to provide secure communication
- ▶ TLS uses the following crypto processes, so we need to study them briefly:
 - ▶ Hashing
 - ▶ MAC / HMAC
 - ▶ Symmetric key encryption
 - ▶ Asymmetric key encryption
 - ▶ Digital signature
 - ▶ Key exchange
 - ▶ Digital Certificate

Base 64

- ▶ Data is a sequence of binary bits
- ▶ The binary number block is split into 6 bit blocks then stored as bytes. i.e. 4 chars in 3 bytes
- ▶ The 6 bits represent the position in the base64 alphabet where 'A'=0, 'a'=26, '0'=52, '+' = 62, '/' = 63. Split into 6 bit blocks
- ▶ Say we have a file of only 4 bytes: D3 FD B6 00

11010011111110110110110000000000



- ▶ 0 / 2 2 A A = = In base 64 -
- ▶ As the smallest byte sequence to fit 6 bit numbers is 24 bits or 3 bytes, we need to pad a byte with zeros then indicate how many bytes are missing from a 3 byte block. Pad the last byte to create a char = A in this case. As we are 2 bytes short of a 3 byte block, we add two = to indicate that the last triple has 3 bytes missing converting 0/22AA back into binary
- ▶ Doesn't work with urls as +, / and = are used for other things in internet-based data. + is a space and / is a path separator. = means = so has been dropped. Because base64-url is used for structured data, e.g., JWT, OAuth tokens, URL query parameters etc, the receiver knows how to interpret this data as it can be inferred by the protocol or an expected number of bytes

Index	Binary	Char	Index	Binary	Char	Index	Binary	Char	Index	Binary	Char
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/

demo using <https://cryptii.com/pipes/hex-to-base64>

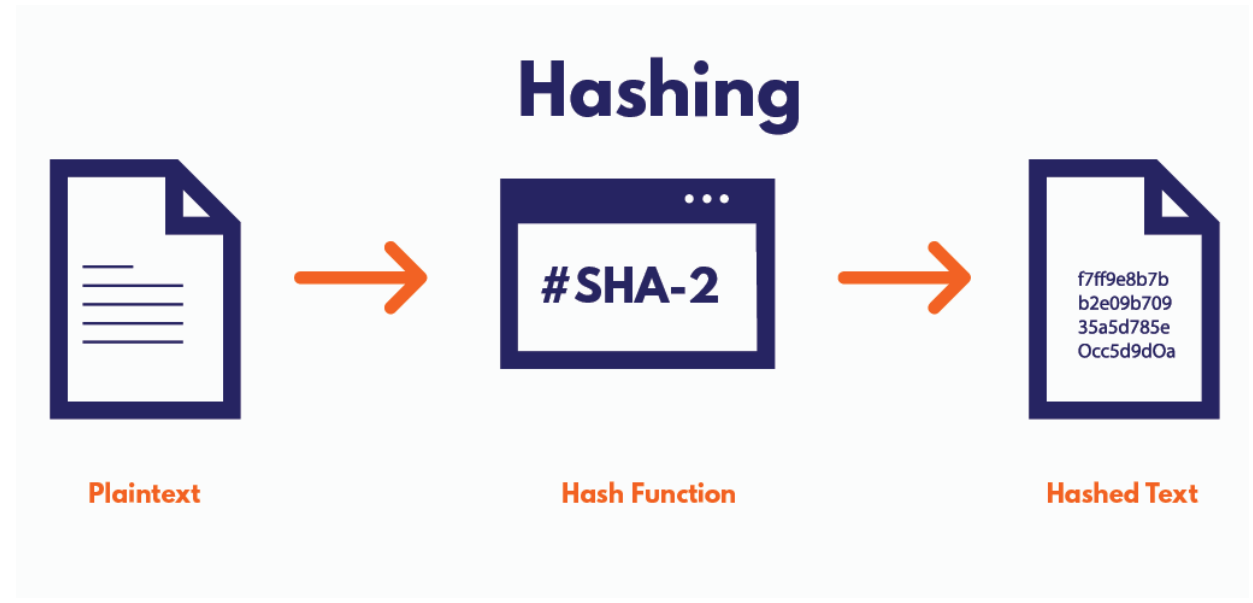
Security Aims of TLS

- ▶ **Confidentiality** - no one other than the intended person can read the data - i.e. secret
 - ▶ **Integrity** - we can be assured that the data has not been interfered with or corrupted
 - ▶ **Authenticity** - we know for sure who / where the data came from
 - ▶ **Non-repudiation** - the provider of data cannot claim at a later date that they didn't send it
-
- ▶ TLS provides crypto approaches that deal with most of these assurances
 - ▶ TLS doesn't provide non-repudiation but a technique it uses can - we will discuss each
 - ▶ TLS enables us to use https which is security of data in transit through encryption and authentication

Hashing (Integrity)

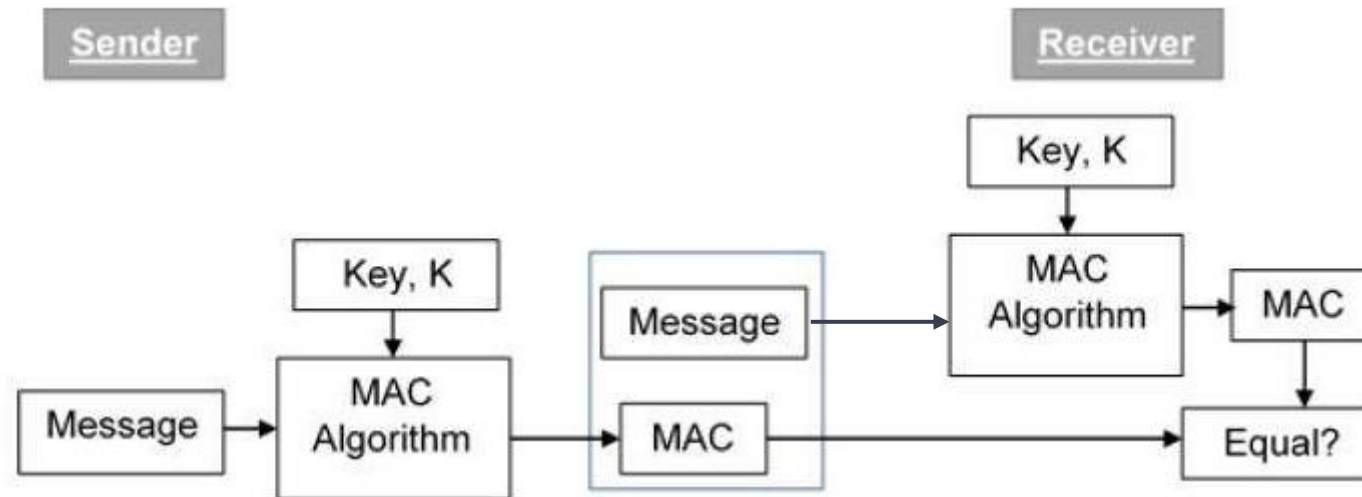
- ▶ Fast search
- ▶ **One-way crypto function**
- ▶ **Fixed length number (digest)**
- ▶ Use cases include

- ▶ password protection
 - ▶ Pa55w0rd : SHA 256=590561E975AA1CBDAA6FF00DA013DFB85656990EEF11E504E9FEBE48A87D3642
 - ▶ Can't be reversed but rainbow table may expose the password so still need strong password
 - ▶ Brute force attack
 - ▶ Use long complex passwords and a password manager
 - ▶ Use Single sign-on
 - ▶ Use Multi factor authentication
- ▶ file / message integrity check
 - ▶ File data integrity, photo steganography



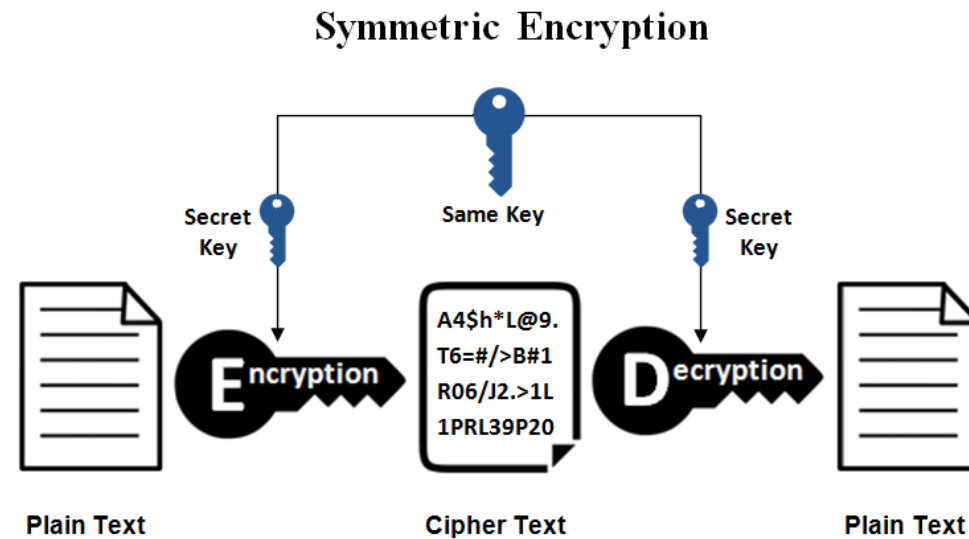
Message Authentication Code (Integrity and authenticity)

- ▶ Hashing is good for integrity but not security
- ▶ MAC fixes this as it adds authenticity as long as the shared secret is kept secure
- ▶ HMAC is a MAC generated using a hashing function



- ▶ Cannot provide non-repudiation - I can be assured of who it came from and it's not been tampered with, but what if the sender claims they didn't send it, what proof do I have? I have a message and hmac which I could have created myself and claimed it was sent to me

Symmetric key encryption (confidentiality)

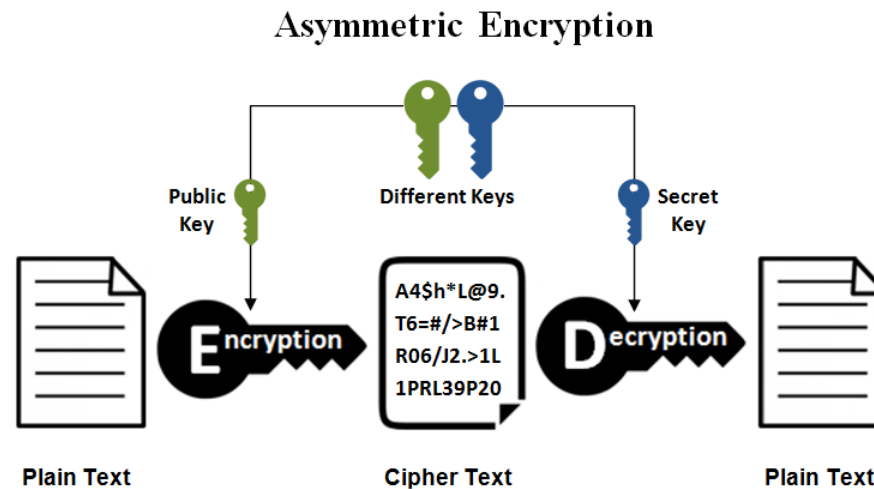


- ▶ The same key is used to encrypt as to decrypt
- ▶ Advanced Encryption Standard is most common
- ▶ AES has a 128 bit key (can accommodate 192 and 256 bit keys)
- ▶ AES128 for home routers & https, Bitlocker & Whatsapp use AES256 and NSA approves it for Secret level data
- ▶ Very fast so good for large data blocks
- ▶ Challenge: getting the key to the second party securely

Secure key agreement (not exchange)

- ▶ Diffie-Hellman is a very popular method of key agreement
- ▶ Each party creates values that are exchanged and from those values, the secret keys are computed
- ▶ The beauty of this method is that the secret keys themselves are not exchanged so can't be intercepted
- ▶ The values which are exchanged, if intercepted, cannot be used to compute the shared secret
- ▶ If there is a man in the middle attack, they would need to intercept both exchanges and substitute both values
 - ▶ To protect against this, the server value also provides a digital signature (later) so the value can't be changed without detection
- ▶ From the exchanged values, both parties can compute the same value which is the shared secret
- ▶ Although the shared secret could be used now for encryption, generally create working keys based on these shared values as will be seen in TLS later

Asymmetric Key Encryption (confidentiality)



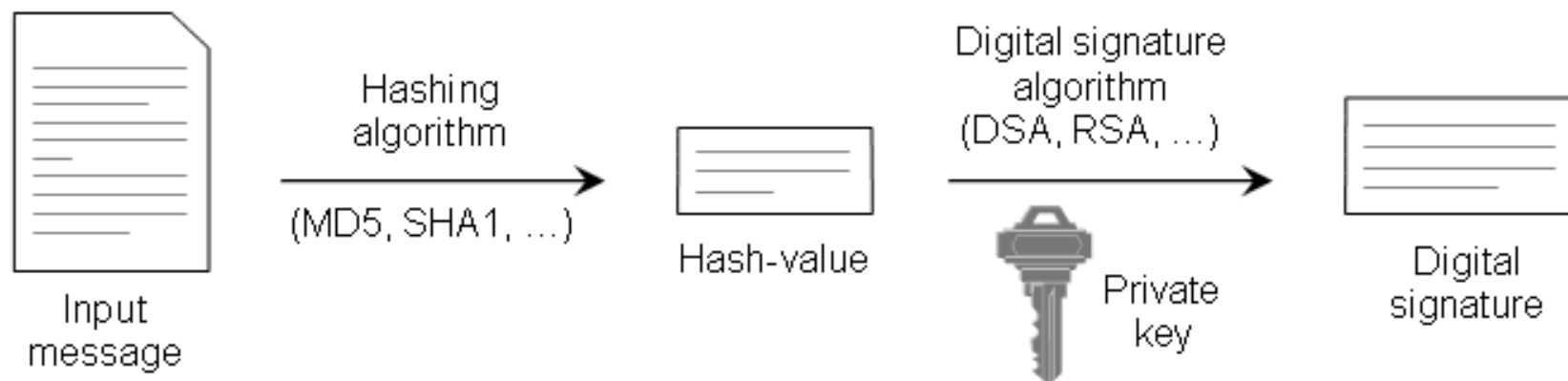
- ▶ Uses a pair of complementary but different keys: private key and public key
- ▶ Public key is made public
- ▶ A message encrypted with a public key is decrypted with the secret key - this is used for **confidentiality**
- ▶ A message encrypted with the secret key can be decrypted with a public key - not used for confidentiality, it's just a fact
- ▶ But who sent the encrypted message?
- ▶ How do we know your public key came from **you** when we downloaded it or received it in an email? - i.e. someone could claim to be Alice, spoof her email and send an encrypted message to Bob using his public key
- ▶ Complex maths so slow. Not good for large data blocks

</demos/crypto/asymmetricKeyGen.js>
</demos/crypto/asymmetricEncrypt.js>
</demos/crypto/asymmetricDecrypt.js>

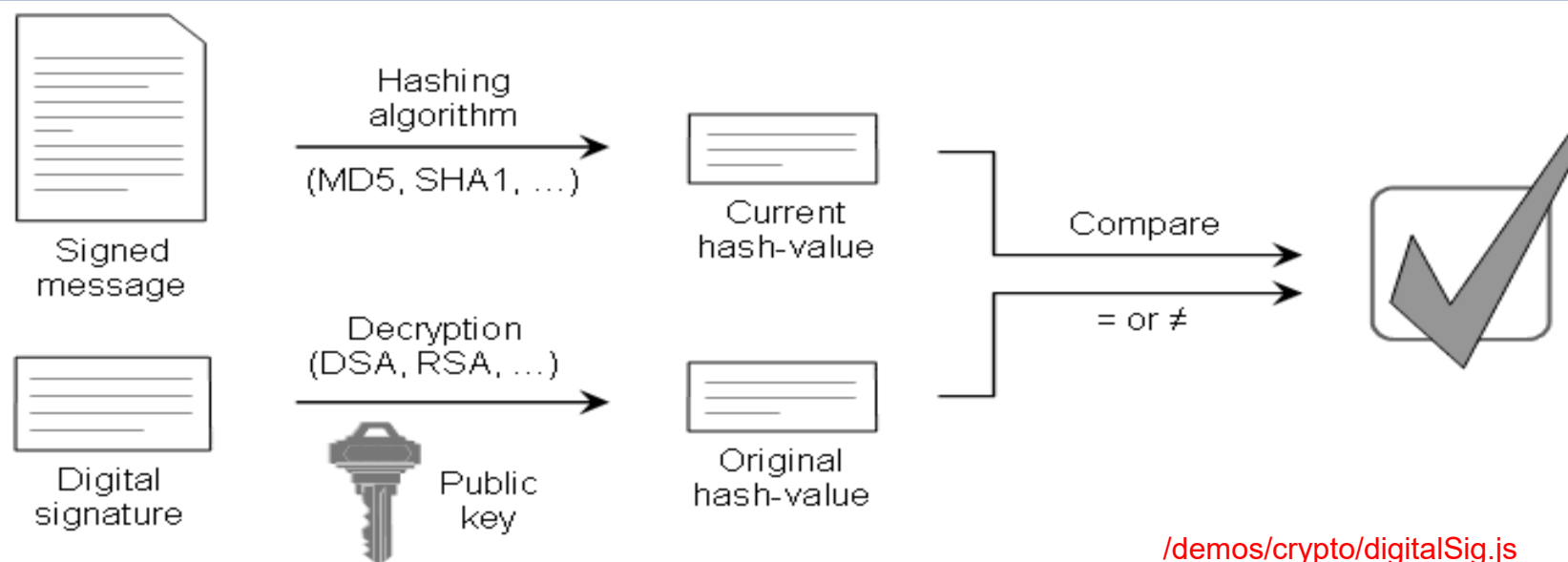
- ▶ Popular asymmetric algorithms are RSA and Elliptic curve
- ▶ RSA (N) where N is the key size
- ▶ The National Institute of Standards and Technology (NIST) define crypto strength in "bits of security"
 - ▶ 112 bits is good until about 2030
 - ▶ Approximate equivalencies:
 - ▶ 2048 bit RSA key
 - ▶ 224 bit ECC key
 - ▶ 128 bit AES

Digital Signature (authenticity, integrity, non-repudiation)

- ▶ A digital signature has a wealth of features:
 - ▶ More secure alternative to a handwritten signature
 - ▶ Guarantees **authenticity** of the sender
 - ▶ Provides **non-repudiation** - as long as the secret key isn't shared so for legal stuff, use a HSM, smart card or similar to hold the secret key
 - ▶ Ensures **integrity**
- ▶ Bob wants to send me a binding contract of employment
 1. Bob sends me his public key
 2. Bob hashes the contract to create a unique number (digest)
 3. Bob encrypts the digest using his private key
 4. I receive the email with the plain text contract and the encrypted digest (digital signature)
 5. I decrypt the digital signature using Bob's public key to get the hash value (not confidentiality)
 6. I hash the contract file to get a digest (hash) to check it's the same as Bob's
- ▶ We need a diagram



Create a Digital Signature



Verify a Digital Signature

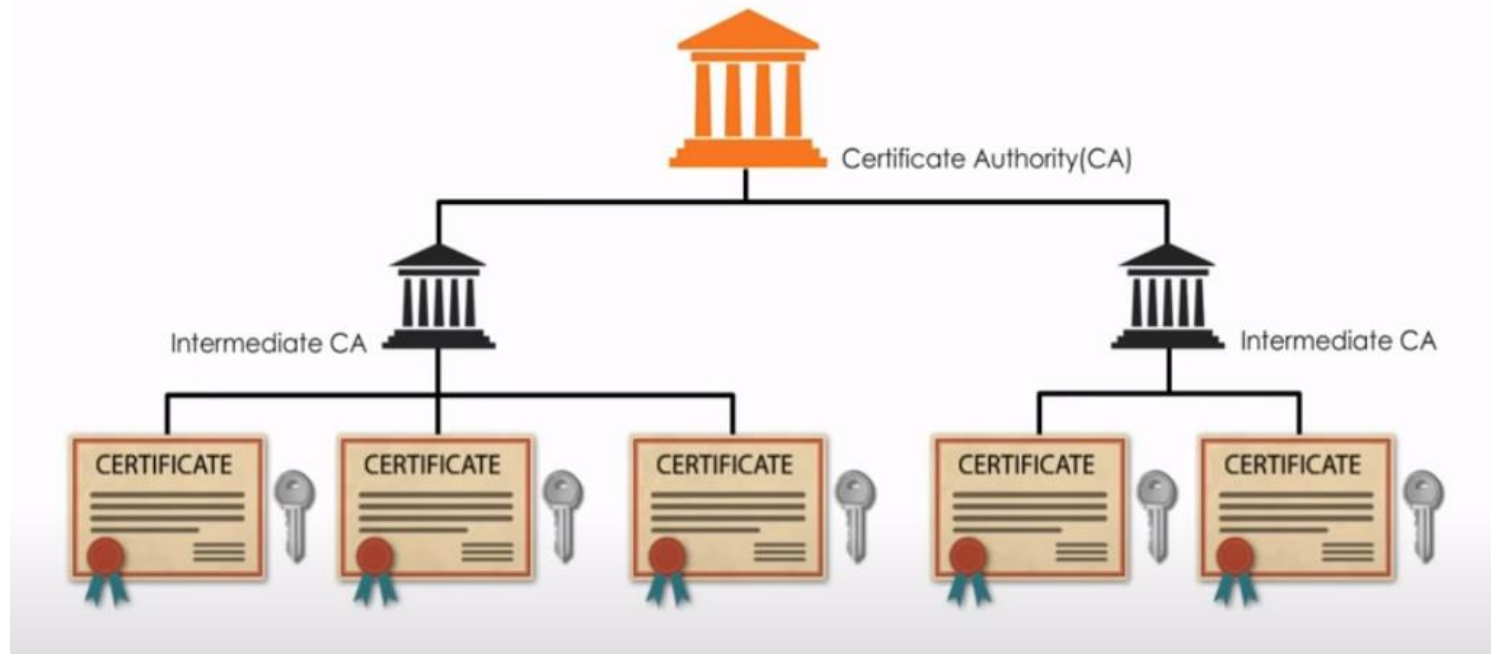
</demos/crypto/digitalSig.js>
</demos/crypto/digitalSigCheck.js>

Summary of basic properties

- ▶ **Symmetric keys** are the same for encryption and decryption - provides **confidentiality**
 - ▶ Encryption and decryption is very fast
 - ▶ Securely getting a symmetric key to the second party is the difficult bit. Diffie-Helman is a popular approach to solving that problem as the actual keys aren't exchanged but calculated based on exchanged data
 - ▶ Advanced Encryption Standard (AES) is popular for symmetric encryption (data in transit and at rest)
- ▶ **Hash** can be used for **integrity** as it is a unique fixed length number very sensitive to data change
- ▶ **MAC / HMAC** used to provide security of the hash by mixing in a symmetric key so provides
 - ▶ **authenticity** and **integrity**
- ▶ **Public key encryption.** Keys are a pair and are different: private (secret) and public
 - ▶ Public key used to encrypt data for **confidentiality**
 - ▶ Private key used to decrypt confidential data encrypted with public key
- ▶ **Private key encryption** can be used to create a digital signature.
 - ▶ Provides **integrity, authenticity & non-repudiation**
 - ▶ Digital signature is the equivalent to symmetric key HMAC but better as it can provide non-repudiation
- ▶ RSA and Elliptic Curve are popular for asymmetric encryption - e.g. digital signature and digital certificates

Digital Certificate

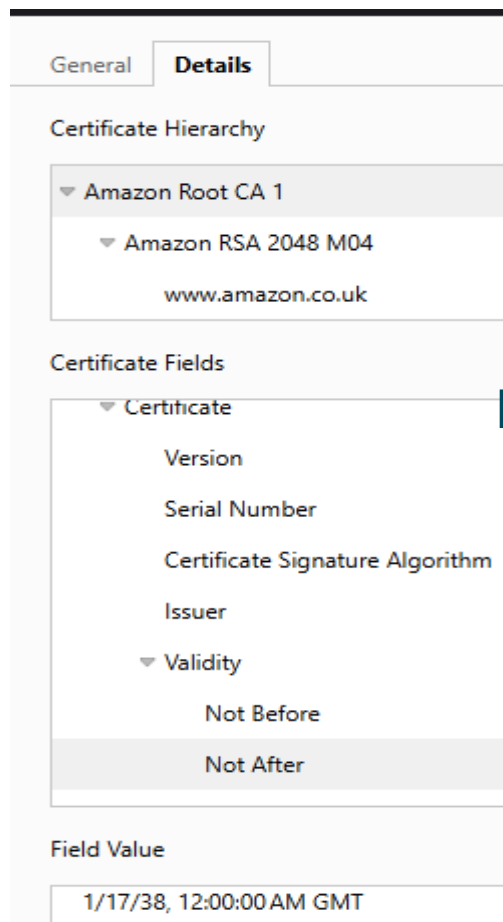
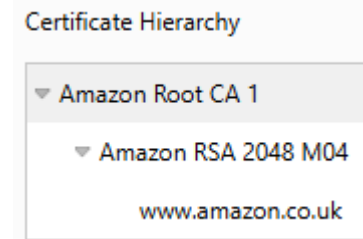
- ▶ We have almost nailed the security using the various approaches and digital signatures are simple, very effective and provide three of the security essentials we need: **integrity**, **authenticity** and **non-repudiation**. But we need a public key to be able to check the signature. How do we know where that came from?
- ▶ Digital certificates are the final piece in the puzzle
- ▶ Digital certs answer the question "Can we be sure that Bob's public key really is Bob's key?"
- ▶ Bob creates a public key then gets it embedded into a digital certificate by a trusted third party - certification authority
 - ▶ Bob needs to provide other info including the domain the certificate will be used e.g. bob.com. This is checked by the CA
- ▶ The CA, adds Bob's public key and his domain then signs the certificate with its digital signature to validate it
- ▶ When we want to use Bob's public key, we ask for the certificate. We can check the CA digital signature using the CA public key and if ok, we can trust Bob's public key so extract it from the certificate and use it
- ▶ In https, Bob would store his certificate on the server that is used to provide a service at the certified domain
- ▶ For practical purposes, there is more than one CA in the world, and the CA can delegate authority to intermediate CAs to make the process more manageable. This creates a chain of trust



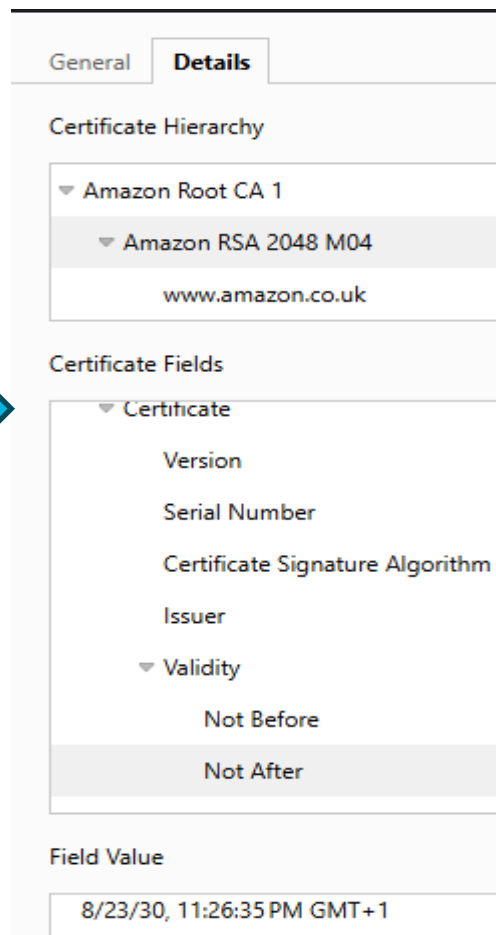
- ▶ Root certificate
 - ▶ A root certificate is self-signed as it must start somewhere. i.e. the CA digitally signs its certificate using its secret key
 - ▶ Root certificate authorities are known and trusted by most browsers - installed in OS and browser during updates
- ▶ Intermediate certificates
 - ▶ These can be created from the root and used by delegated organisations to sign user certificates. The lineage of certificate signing is held in the certificate
 - ▶ Having just a root CA is limiting in that it could be a bottleneck in providing certs and if its secret key is compromised then all its certs are worthless, so intermediates are where you tend to get your cert from
 - ▶ Intermediate certs and your cert should be installed on the server. The Root cert should be on the client in the OS trusted store

Chain of trust

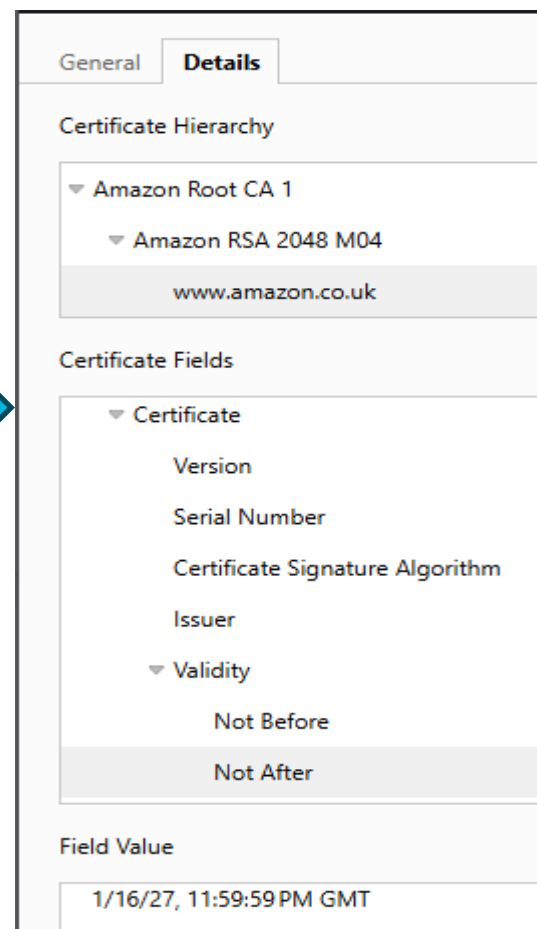
- ▶ Enter amazon.co.uk into your browser. It is secure by default
- ▶ Click on the padlock (now a different symbol) then certificate
- ▶ Certification path does what it says on the tin



Self signed root CA



Intermediate CA



Server Cert

Demo in browser

How to get a certified digital certificate

- ▶ You need to make a Certificate Signing Request (CSR)
- ▶ X.509 is the most common standard used
- ▶ You need to provide information proving who you are, and the details associated to you:
 - ▶ Your name – generally the host name of the server you want to use – e.g. your domain `tonynicol.com` as this is unique to you
 - ▶ Organisation – if you have one. This can be verified by the Certification Authority if it is registered
 - ▶ Division within the organisation – if relevant
 - ▶ City, State/County, Country, Email Address – they may contact you on this
- ▶ You need to provide verifiable details
 - ▶ Depending on the cert type, they may check your domain name provider, your payment provider, email, phone etc.
 - ▶ Extended Validation (EV): - Legal business name, Jurisdiction of incorporation, Registration number etc (e.g. `digicert.com`)
 - ▶ Organisational Validation (OV): Organisational details such as verified name, location (e.g. `Microsoft.com`)
 - ▶ Domain Validation (DV): simply prove you own the domain. You could be anyone (e.g. `Lancashire.ac.uk`)
- ▶ You create public / private key pair
 - ▶ Keep the private key secure and private
 - ▶ Public key will be sent to CA in the CSR and will be embedded in the signed certificate

Creating a simple DV cert

- ▶ Domain validation certs are very popular
- ▶ mkcert is a useful app from mkcert.org
- ▶ mkcert creates a CA on your client machine which enables you to create your own certificates
- ▶ mkcert adds itself to the secure vault in the trusted CAs in your OS so any certificates created from it and deployed to your server are trusted
- ▶ Useful for quick local ssh testing. Can use an IP with it in the domain registration
- ▶ The certs won't be trusted by anyone else's browser but it's good for development and learning
- ▶ Instructions are available at mkcert.org
- ▶ Get the app (Windows): winget install mkcert
- ▶ Once we have a CA trusted by our browsers we can create our own certificates
 - ▶ **mkcert -key-file key.pem -cert-file cert.pem domain**
 - ▶ this will create key.pem and cert.pem in the current directory. Domain needs to be localhost, 20.117.114.49 or wherever you will use the cert. The browser can check the IP it called and compare to this value
- ▶ If you want to look at the cert in a readable form:
 - ▶ **certutil -decode cert.pem** (on Windows)
 - ▶ Check the *X509v3 Subject Alternative Name: IP Address:20.117.114.49* or wherever, is correct if you are having problems
- ▶ We don't add certs into the container, or each container needs one and the secret key is in the image on Docker hub

Creating a let's encrypt real cert

- ▶ Cannot create a let's encrypt cert using an IP address - you need a domain name
- ▶ You install certbot on your server then using it in a single command it will
 - ▶ Send a server to let's encrypt (LE) server POST request with the domain name you want to register
 - ▶ Server receives a token from LE - basically a random string - this is a challenge step
 - ▶ Server POSTS back the token and a hash of the public key as a response to the challenge
 - ▶ Server POSTS a "Please validate the challenge" to LE
 - ▶ LE makes a get request to a web server on your server (typically port 80) asking for the token to check it matches
 - ▶ If it matches, certbot creates a key pair and a CSR containing the public key, domain names, and a digital signature proving ownership of the private key associated to the public key in the CSR and server sends it to LE
 - ▶ LE creates and returns cert.pem, chain.pem (the intermediate cert) and chain.pem (both in one file). We tend to use this one
- ▶ All that is done for you but you should know the sequence of events behind it
- ▶ It does create a CSR and it is stored on your server but you never see it unless you look for it
- ▶ If use digicert or similar, you create a CSR on their web page, add your public key and they send you the certs