Public key cryptography

New type of cryptography that distinguished between encipherment and decipherment keys.

One of keys – publicly known. Other – kept private by owner.

To send a message ,encipher it with recipient’s public key. Then send it.

Recipient can decipher message using his private key

One key is public ,other private key is secret ,public key cryptosystem must meet few conditions.

1)Must be easy to encipher or decipher a message given appropriate key.

2)Must be infeasible to derive private key from public key.

3)Infeasible to determine private key by a plaintext attack.

A close-up of a computer code

Description automatically generated

RSA can’t be used to encrypt large texts as it will result in very large ciphertext.

RSA is exponentiation cipher.

An asymmetrical algorithm invented in the 1970s

Choose 2 large prime numbers p and q , n=pq .

Totient phi(n)– number of numbers less than n with no factors in common with n.

Phi(n) = (p-1) (q-1)

Choose Integer e relatively prime to phi(n)

c = m^e mod n

Issues – computation of modular exponentiation and finding 2 large primes.

RSA can’t be used to encrypt large texts as it will result in very large ciphertext.

Providing both

Use of public key cryptography – provides technical type of non-repudiation of origin.

recipient Bob decrypts outer layer using Alice’s public key.

Public key is inverse of pvt key ,only Alice who knows pvt key could have enciphered her message.

To provide both confidentiality and authentication, must first encipher with recipients public key and then encipher that with sender's private key.

* **Initial Encryption**: To send a confidential message to Bob, Alice first encrypts the message using Bob's public key. This ensures that only Bob, who possesses the corresponding private key, can decrypt and read the message, providing confidentiality.
* **Further Encryption for Authentication**: To add an authentication layer, Alice can then encrypt the already encrypted message with her own private key. This is known as double encryption.
* **Decryption Process**: Upon receiving the message, Bob first uses Alice's public key to decrypt the outer layer, authenticating that the message is indeed from Alice (since only her private key could have encrypted it). He then uses his private key to decrypt the inner layer to read the confidential message.

The correct answer is: A symmetrical  encryption algorithm and substitutional cipher that can in principle be unbreakable → One time pad, An asymmetrical algorithm invented in the 1970s → RSA, A symmetrical algorithm with key length of 64 bits → DES, A symmetrical algorithm with keyspace up to 2256 → AES

Asymmetric cryptography is neat idea

I'll introduce a new idea to you. Wouldn't it be nice if I could take a

document and encrypt it? Encryption I'm showing by showing you a padlock. The

idea being I can take something and put it into a box and put the padlock on so

you can't see it. Okay. Wouldn't it be nice if I could have a key like this

whereby if I encrypt something with that key it doesn't take the same key to

decrypt it as we've seen before but it takes a second key like this to decrypt.

Why is that a nice idea? Well now I can take this. This is a key pair. I'll start

introducing some terminology. We have a key pair because they're somehow

intimately related. If they can do this there's some kind of connection

between the two so they match in some way. And then I can take one of these

keys. I'll take the red one. That looks kind of dangerous. The red I'm gonna put

here in my breast pocket close to my heart and if I've got it there then I'm

pretty sure that nobody else will get hold of it without me noticing. It's

good and safe. The red key, sorry I'll write this in red to make the connection,

the red key I'm going to call my private key. Back in the pocket. The green key I'm

going to call the public key. And that means that I can leave this in the open.

I can now give this key to you. Not only to you, you can take that key and you can

copy it and give it to anybody else you like. It's publicly available. So why is

this a good idea? Well if you wanted to send me a secret now you take this green

key, encrypt something and then send that to me. Knowing that these two keys belong

together you understand that the only person who's going to be able to decrypt

that is now me. And that is very handy immediately because you realise that now

anybody in the world who has a copy of this key can encrypt something and pass

it to me. Whereas before anybody in the world I would have had to have something

like seven billion different symmetric keys to be able to do that. But now I can

do it with a simple pair of keys. Okay. But I want to expand on the idea. I'm going to

take back the key. That's a nice idea but I'm greedy I want even more. I also want

to be able to do like this. I want to start with the red key and sometimes I

want to be able to encrypt with the red key. And if I do that, I do understand

that if I do that I can't go back that way. No I have to either follow the

outside ring or what will now be an inside ring. There you go. I understand

that it goes that way or it goes that way. And that's nice too. Why? I'll give

you back that. Because now I can take a document, let's call it an exam paper. I

can take an exam paper and encrypt it with my key that nobody else has. And

then anybody in the world can take the green key and decrypt it. So in that case

we understand it's not about secrecy. It's not confidentiality in that case

because anybody can take that key and open it. It is of course an issue of

integrity again. If I do that and you can decrypt, anybody can decrypt it, then in

the process of decrypting they know that it was me that encrypted it. Not

only is this a matter of integrity, it's the deeper kind of integrity that we

have discussed. This is non-repudiation isn't it? You decrypt this and so long as

you understand the connection between these two keys, I can't deny that it was

me who encrypted it. Non-repudiation. So this certainly looks at the moment like a

lovely idea. The idea incidentally, Diffie and Hellman proposed this during the

70s, the 1970s. It's a new idea. And then a race began to try and discover a

method by which we could really do this. So let's hope somebody has discovered a

method by which we can do this.

RSA ,not all we are hoping

Well, I said that the idea of asymmetrical cryptosystems was invented in the 70s, the

1970s. Actually, that was only in the public space, and I did mention that. Since then,

it's turned out that before these Americans who invented the idea, published the idea,

it was very likely that the idea was discovered and implemented at Bletchley Park in Britain.

Bletchley Park is the place where Alan Turing worked during the war to decrypt German messages.

And of course, all the work that was done there was very, very secret, so they couldn't

tell anybody about their work. So let's put all the history lessons aside and say the

first big method in the public domain, it wasn't the first, but the first most practical

method is this one. RSA. And RSA stands for Rivest, Shamir and Adleman, who were the three

authors who collaborated on this. And here I've got some terminology, we're talking about a

cryptotext or ciphertext, we've been calling that same thing, crypto, cipher, and a message

could be a plain text, but now we're mostly interested in messages. And here we get the

idea of the public key and the private key and how we're going to get hold of it. And

this is basically how they do it. And at this point, you're probably worrying about the maths

involved. We're not going to dig into the maths, but I do at least want to show you a basic

problem that we still have. We can with RSA generate public keys and private keys.

They have all the requisite attributes that we'd like of these keys. You shouldn't ever be able to

look at a public key and from that work out what the private key looks like. Yeah, that would spoil

things if you could look at the green key and say, oh yeah, I know exactly what the red key looks

like. We can't have that. You shouldn't be able to look at the lock, i.e. you shouldn't be able

to look at the algorithm and then work out from that what either of the keys look like. So

everything is compartmentalised like that. And that works with RSA. Very good.

But there is a problem. The only thing I'd like you to understand from this picture here is that

to get the cryptotext, I need to take the message and raise it to the power of a number e.

Then I do division and take the remainder. Yeah, but the first step is to take the message

and raise it to the power e. That means that since this is working with numbers,

integers in fact, I'm going to have to translate my whole message into a number

and then apply the exponent to that with what is probably a large number.

Now imagine taking a message and turning it into a number. How big can a message be?

Well, it could easily be the complete works of Shakespeare.

So you're asking me to take the complete works of Shakespeare,

turn that into a number and then take an exponent and turn it into a number.

So I'm going to take the exponent of that number. At this point we realise that this formula is

dealing with very, very big numbers. So it's not your normal maths library, this.

But the bad news is that this puts a limit on what we can encrypt.

Unfortunately, we're not going to be able to encrypt the whole works of Shakespeare

because it's too big a number and it would take too much machine power and too much time.

So if we're going to use this method, it seems like we can do it so long as we keep to,

generally speaking, pretty short messages that are easy to translate into numbers that we can at least handle.

So we were almost there with this wonderful idea, but now it feels like it's slipped between our

fingers because we want to encrypt large things, but we can only encrypt small things.

But don't give up hope because with the knowledge that we all have at this point, I have a solution.

And the next step is to run through how it works in reali

Checksums

Alice wants to send Bob message of n bits.

Wants Bob to be able to verify message he receives is the same one that was sent. She applies checksum function to generate smaller set of k bits from original n bits.

Smaller set called checksum or message digest.

Alice then sends Bob message and associated checksum.

When Bob gets message, he recomputes checksum ,compares it with one Alice sent. If they match ,he assumes the message has not been changed.

If they don’t match message can’t be trusted.

Parity bit in ASCII – often used as single bit checksum.

If odd parity used sum of 1 bits in ASCII representation of character and parity bit is odd.

Assume Alice sends Bob letter “A”.

In ASCII representation of “A” using odd parity – p1000001 in binary , p is parity bit .Because 2 bits set parity bit is 1 for odd parity.

When Bob gets message he counts number of 1 bits. Because it is odd message has arrived unchanged.

To increase probability a change to message or checksum will be detected, checksum function must have specific properties.

1)For any x belongs to A, h(x) is easy to compute

2)For any y belongs to B, it’s infeasible to find x belongs to A, such that h(x)=y.

Hash functions are one way functions. Impossible to go from hash value back to original data.

Collision resistance

3)Computationally impossible to x, x’ belongs to A such that x!=x’ and h(x)=h(x’).

Pigeonhole

If n containers for n+1 objects, atleast one will have 2 objects.

8 possible hashes for 32 files , atleast 4 files will have same hash.

Suppose checksum function computes hashes of 128 bits.

Yes, the output of a hash function, particularly in the context of computer science and cryptography, is typically represented as a sequence of bits, where each bit can only be 0 or 1. This binary representation is fundamental to how computers store and process data.

Prob finding message corresponding to given hash – 2^ (-128)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Keyed cryptographic checksum function requires a cryptographic key as part of the computation.

Keyless cryptographic checksum function does not.

The correct answer is: A symmetrical  encryption algorithm and substitutional cipher that can in principle be unbreakable → One time pad, An asymmetrical algorithm invented in the 1970s → RSA, A symmetrical algorithm with key length of 64 bits → DES, A symmetrical algorithm with keyspace up to 2256 → AES

Encrypt a message with public key ,recipient can decipher message using private key.

HMAC stands for Hash-based Message Authentication Code. It's a specific construction for creating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret cryptographic key. HMAC can be used to verify both the data integrity and the authenticity of a message.

The HMAC process involves taking the original message and a secret key, then applying a hash function in a specific way:

1. **Start with a cryptographic hash function** (like SHA-256) and a secret key that is known only to the sender and the intended receiver.
2. **Prepare the key**: If the key is longer than the block size of the hash function, hash it to reduce it to the hash function's block size. If it's shorter, pad it to the right length.
3. **Modify the key for inner and outer hashing**: Create two new keys from
4. the original key by XORing the key with two predefined constants: the inner pad (ipad) and the outer pad (opad). The ipad is typically a string of repeated bytes valued 0x36, and the opad is typically a string of repeated bytes valued 0x5C.
5. **Create the inner hash**: Prepend the inner key to the message, then apply the hash function to the result.
6. **Create the outer hash**: Prepend the outer key to the inner hash result, then apply the hash function again.

The final output is the HMAC value. This value is sent along with the message. When the receiver obtains the message, they perform the same HMAC process using the shared secret key and the received message. If the HMAC value they compute matches the one sent with the message, it can be concluded that the message was not altered in transit and that it was sent by a party possessing the secret key, thus confirming both integrity and authenticity.

HMACs are widely used in various security applications and protocols, including IPsec, TLS, and SSL. They provide security against certain attacks such as extension attacks, which are mitigated by the inclusion of the secret key in the hashing process, ensuring that the hash value cannot be computed without knowing the secret key.

Digital signatures

Think of Alice’s signature on a contract with Bob.

Bob not only has to know Alice is the signer and is signing it, he also must be able to prove to a judge that Alice signed it and the construct has not been altered since Alice signed it.

Construct that authenticates both origin and contents of a message in a way that it is provable to a disinterested third party.

Let m be a message.

The “proof” requirement introduces a subtlety. Let m be a message. Suppose Alice and Bob share a secret key k. Alice sends Bob the message and its encipherment using k. Is this a digital signature?

First, Alice has authenticated the contents of the message, because Bob deciphers the enciphered message and can check that the message matches the deciphered one. Because only Bob and Alice know k, and Bob knows that he did not send the message, he concludes that it has come from Alice. He has authenticated the message origin and integrity.

However, based on the mathematics alone, Bob cannot prove that he did not create the message because he knows the key used to create it. Hence, this is not a digital signature.

d -pvt key

e-public key

Public key cryptography solves this problem. Let d\_Alice and e\_Alice be Alice’s private and public keys, respectively. Alice sends Bob the message and its encipherment using d\_Alice. As before, Bob can authenticate the origin and contents of the message, but in this situation a judge can determine that Alice signed the message ,because only Alice know her pvt key with which message was signed.

The judge uses Alice’s public key to decipher message.