How secure is RSA? very very

- Public key (n, e) is known to everybody. Private key (n, d) is only known to receiver
- need to guess the value of d, e is known and d is related to e: (ed-1) mod z = 0
- To find z, one has to know p and q, p and q is related to n: n = p*q
- Hence, one has to be able to perform prime factorization of n to know p and q
- If n is sufficiently large, e.g: 1024 bits, it is hard to find the correct prime factors of n (exponential complexity).
- To crack a simple 128-bit key, a supercomputer requires ~10^(39) seconds. The universe is younger than that.

How Practical is RSA?

- exponentiation in RSA is computationally exhaustive (DES is 100x faster than RSA)
- **Use Session Key:**

1. Use public-key cryptography to establish a secure connection (meaning we know for sure who the host at the end of network line is)
2. Generate symmetric session key & exchange b/w sender & receiver
3. Use this symmetric session key (instead of public key) for lots of subsequent comm throughout session

Application of Cryptography

provides: authentication, confidentiality, msg integrity, availability, access

prevents: eavesdrop, alter msgs, impersonate, denial of service, hijack

1) Providing Authentication:

> Need to prevent **impersonation**
A--msg--> B | T---fake msg--> B
> Need to prevent I + **spoofing**
A--IP+msg--> B | (spoofing) T --A's IP+msg--> B
> Need to prevent I + S + **eavesdropping**
A--IP+secret PW+msg--> B |
(eavesdrop atk) T--A's IP + A's sPW+msg-->B
> Need to prevent I+S+E+ **playback**
A--IP+encrypted secret PW+msg--> B
(playback atk) T--A's IP+A's esPW+msg--> B
T can just record and playback the 'encrypted secret password', B cannot tell if its really A or T who is sending this
> Need to prevent I+S+E+P: Use encryption + **nonce**
A <-- Nonce N---B | B sends Nonce, A encrypts it using A -- Ka-(N) & Ka+--> B | Ka-

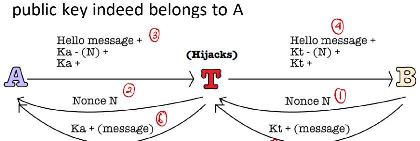
Encrypting message with private key is called digital signature

Failed by **man-in-the-middle attack**

The loophole: B doesn't know whether A's public key he received indeed belongs to A

Bob receives eth Alice sends, & vice versa but Trudy receives all msgs as well! Attack is totally transparent to A&B

Solution: Besides Nonce, you also need an external party (called Certificate Authority: CA) to verify that A's public key indeed belongs to A



→ a client verifies the authenticity of the certificate using the well-known public key of the CA, e.g: Global Sign, Verisign, etc

1. PROVIDING AUTHENTICATION

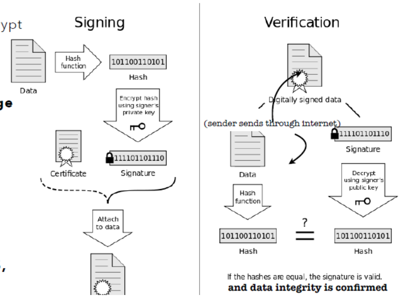
Practical usage using **message digest**

It is expensive to encrypt long messages using private key (RSA)

Solution: use message digest

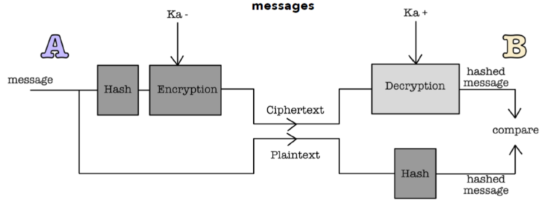
Message digest: Hash long (arbitrary sized) messages to produce a **fixed size** ***message digest***

Hash functions: MD5, SHA1



2. PROVIDING AUTHENTICATION AND MESSAGE INTEGRITY

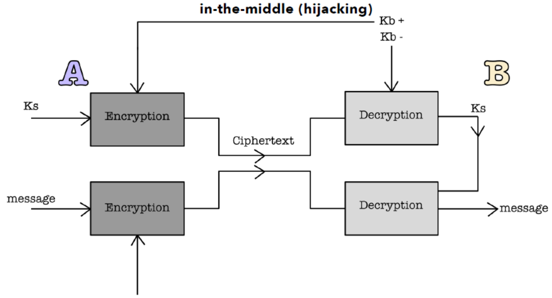
Prevent impersonation, spoofing, eavesdropping, playback, and altering of messages



3. PROVIDING AUTHENTICATION, MESSAGE INTEGRITY, AND CONFIDENTIALITY

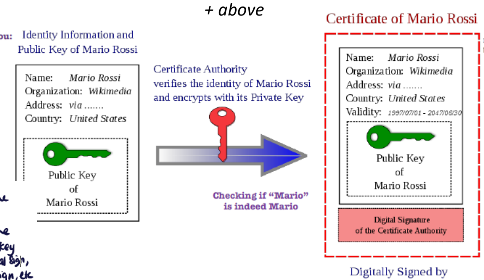
Eg: **secure e-mail, ssh, https**

Prevents prevent impersonation, spoofing, eavesdropping, playback and man-in-the-middle (hijacking)



1. *Authentication: CA cert + Nonce*
2. *Authentication & Integrity: Signed digest + CA cert + above*
3. *Auth & int & confidentiality: CA cert + Ks (session key, symmetric) + above*

Lets say Mario sends you:



Digital signatures

simple digital signature for message m:

- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$

cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document, establishing he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed the document (*non-repudiation*)
- Can we use symmetric key for non-repudiation?
- suppose Alice receives msg m, with signature: m, $K_B^-(m)$
- Alice verifies m signed by Bob by applying Bob's public key K_B^+ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
- If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

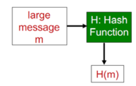
Alice thus verifies that:

- ü Bob signed m
- ü no one else signed m
- ü Bob signed m and not m'

Now we really achieve non-repudiation:

- ✓ Alice can take m, and signature $K_B^-(m)$ to court and prove that Bob signed m

Message digests



computationally expensive to public-key-encrypt long messages

goal: fixed-length, easy-to-compute digital "fingerprint"

- apply hash function H to *m*, get fixed size message digest, *H(m)*.

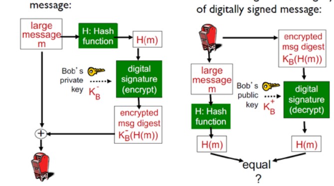
Hash function properties:

- produces fixed-size *message digest* (fingerprint), generally much smaller than message
- many-to-one (collisions possible, but hopefully rare)
- given message digest x, computationally infeasible to find m such that x = H(m)

Digital signature = signed message digest

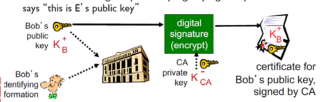
Bob sends digitally signed message:

Alice verifies signature, integrity of digitally signed message:

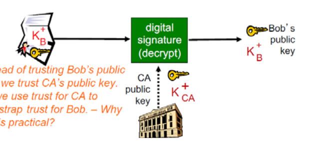


Certification authorities

- *certification authority (CA):* binds public key to particular entity, E. Can be government (e.g., IDA) or well known provider (e.g., VeriSign)
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



- when Alice wants Bob's public key:
  - gets Bob's certificate (from Bob *or elsewhere*).
  - apply CA's public key to Bob's certificate, get Bob's public key



Instead of trusting Bob's public key, we trust CA's public key. So we use trust for CA to bootstrap trust for Bob. – Why is this practical?

A **nonce** is a number that a protocol will use only once in a lifetime. That is, once a protocol uses a nonce, it will never use that number again. Our *ap4.0* protocol uses a nonce as follows:

1. Alice sends the message "`I am Alice`" to Bob.
2. Bob chooses a nonce, R, and sends it to Alice.
3. Alice encrypts the nonce using Alice and Bob's symmetric secret key, K_{A-B}, and sends the encrypted nonce, K_{A-B}(R), back to Bob. As in protocol *ap3.1*, it is the fact that Alice knows K_{A-B} and uses it to encrypt a value that lets Bob know that the message he receives was generated by Alice. The nonce is used to ensure that Alice is live.
4. Bob decrypts the received message. If the decrypted nonce equals the nonce he sent Alice, then Alice is authenticated.

2. I ran the command **dig MX sutd.edu.sg** and got the following (edited and excerpted) output:

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21444
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 6

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;sutd.edu.sg.                 IN      MX

;; ANSWER SECTION:
sutd.edu.sg.          3600    IN      MX      5 post.global.frontbridge.com.
sutd.edu.sg.          3600    IN      MX      0 sutd1.eo.olook.com.

;; AUTHORITY SECTION:
sutd.edu.sg.          2350    IN      NS      nsrv.sutd.edu.sg.
sutd.edu.sg.          2350    IN      NS      dnssec1.singnet.com.sg.
sutd.edu.sg.          2350    IN      NS      dnssec3.singnet.com.sg.

;; ADDITIONAL SECTION:
dnssec1.singnet.COM.sg.   2328   IN   A      165.21.83.11
dnssec1.singnet.COM.sg.   2328   IN   AAAA   2001:c20:18:a::36
dnssec3.singnet.COM.sg.   2328   IN   A      165.21.100.11
dnssec3.singnet.COM.sg.   2328   IN   AAAA   2001:c20:10:a::37
ns.sutd.edu.sg.           13487  IN   A      202.94.70.254

;; Query time: 15 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
```

(a) Was this query iterative or recursive? **Recursive [2pts]. Because both the rd (recursion desired) and ra (recursion available) flags are set.**

(b) As far as you can tell from the output, what functions do the machines sutd1.eo.olook.com and nsrv.sutd.edu.sg serve respectively? **sutd1.eo.olook.com is an email server for the sutd.edu.sg domain [2pts]. nsrv.sutd.edu.sg is an authoritative DNS name server for the email server [2pts].**

Suppose that your department has a local DNS server for all the computers in the department. You are an ordinary user (i.e., not a system administrator). Can you determine if an external web site was likely accessed recently from a computer in your department? Explain. **Yes [1pt]. One possible method is to do a DNS lookup for the web server in question and note the time taken for getting the answer. If the time was small (e.g., LAN delay of a say few milliseconds), the answer was likely cached by the local server due to a recent access.**

2. What is a nonce? Explain why the use of a nonce in an authentication protocol can help defend against the playback attack. **A nonce is an identifier that is guaranteed to be fresh (i.e., never used before). Because a nonce is fresh, whoever replies to it must also be fresh and can't be a previously recorded version of the replier**

Non-repudiation is the assurance that someone cannot deny the validity of something.

3. Assume that Alice and Bob had previously established a secret symmetric key S for encryption of communication between them, and Alice obtained a message M from Bob encrypted by S. For each of the following statements, say if it is true or false and explain why.

a) Alice knows that the message must indeed have come from Bob. **True. Only Alice and Bob are able to generate the message, and Alice knows that she didn't do it.** ***NB: The intended meaning of "come from Bob" is that Bob authored the message. If student answers false and explains it by saying that someone else may have sent Bob's message to Alice, count the answer as correct.***

b) Alice can take the encrypted message to court with non-repudiation that the message indeed came from Bob. **False. Since besides Bob, Alice could also have generated the message, she can't prove to the court that the message came from Bob and not her.**

DNS: domain name system

Domain Name System:
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol:* hosts, name servers communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

iterated query:
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

recursive query:
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?

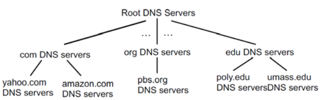
DNS: services, structure

DNS services
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one

why not centralize DNS?
- single point of failure
- traffic volume (bottleneck)
- distant centralized database (long delays for lookups)
- maintenance (add/delete/change translations)

A: *doesn't scale!*

DNS: a distributed, hierarchical database



client wants IP for www.amazon.com; 1st approx:
- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

DNS: root name servers

- contacted by local name server that can not resolve name
- root name server: (only 13 in the world)
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server

TLD, authoritative servers

top-level domain (TLD) servers:
- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp, sg
- Verisign Global Registry Services maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:
- organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- can be maintained by organization or service provider

Local DNS name server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - has local cache of recent name-to-address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

DNS: caching, updating records

- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire
  - who (client or server?) sets the TTL?
- update/notify mechanisms proposed IETF standard
  - RFC 2136

Rivest-Shamir-Adleman (RSA) algorithm (1977):

- Sender, receiver, do **not** share secret key
- **Public** encryption key **known to all**
- **Private** decryption key known only to receiver
- 1024 - 4096-bit asymmetric (public-private) key pair
- 1 round of encryption
- Input/output size: depends on key size

Two **important** requirements:
- Given **public** key, it is **impossible** to compute the private key
- Need a public-private key pair such that,
  - public(m) = encrypted message ------ **encryption**
  - private(encrypted message) = m ------ **decryption**

1. Choose 2 large **prime** number : p & q (1024 bits each)
2. Compute:

$$n = pq, \ z = (p-1)(q-1)$$

relatively prime: dont share common divisor except 1
eg 9 and 16
but 11 and 22 are not

3. Choose e, where e<n, and e,z are relatively prime to one another
4. Choose d, where (ed-1) mod z = 0
5. **Public key, is (n,e):**

$$c = m^e \mod n$$

m = message (unencrypted), where **m must be < n**
c = encrypted message

6. **Private key, is (n,d):**

$$m = c^d \mod n$$

7. An important property:

$$if\ k = m^d \mod n,$$

$$k^e \mod n = m = c^d \mod n$$

$$\boxed{x^y \mod n = x^{(y \mod z)} \mod n \text{ for any } x, y}$$

(note this is an example, so we choose small numbers for ease of calculation.

1. Choose ~~2 large~~ **prime** number : p=5 & q=7
2. Compute:

$$n = 5 * 7 = 35, z = (5-1)(7-1) = 24$$

3. Choose e = 5, where e<n, and e,z are relatively prime to one another
4. Choose d = 29, where (ed-1) mod z = 0
5. **Public key, is (n,e):**

$$c = 12^5 \mod 35 = 17$$

m = 00001100 (this is 12)
c = 00010001 (this is 17)

6. **Private key, is (n,d):**

$$m = 17^{29} \mod 35 = 12$$

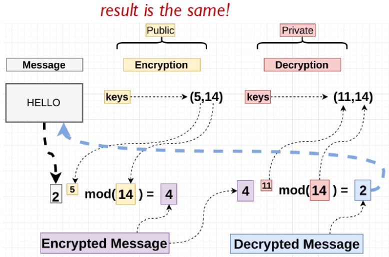
The following property will be *very* useful later

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

use public key first, followed by private key

use private key first, followed by public key

result is the same!



requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key K_B^+, it should be impossible to compute private key K_B^-

follows directly from modular arithmetic:

$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$
$= m^{de} \bmod n$
$= (m^d \bmod n)^e \bmod n$

$$[(a \mod n) \pm (b \mod n)] \mod n = (a \pm b) \mod n$$

$$[(a \mod n) * (b \mod n)] \mod n = (a * b) \mod n$$

$$[(a \mod n)^d] \mod n = a^d \mod n$$

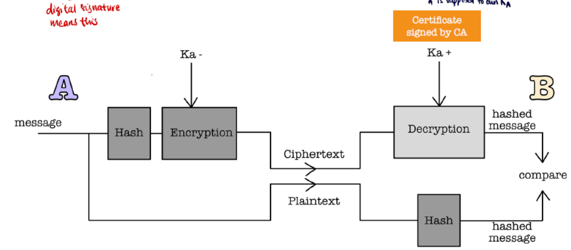
2. PROVIDING (AUTHENTICATION AND) MESSAGE INTEGRITY

Prevent impersonation, spoofing, eavesdropping, playback, and altering of messages.

Signing (encrypting using private key) + Certificate guaranteed by CA give **nonrepudiation** that data is authorised by A

digital signature means this

↳ because legally, only A is supposed to own K_A^-



https://websitename.com

THE DOMAIN NAME SYSTEM (DNS)

Application layer protocol,
Distributed database

32-bit IP address

After IP address is obtained, then end hosts can communicate with one another

When you browse website:
1. Key in the address: facebook.com (hostname)
2. Need to resolve facebook.com -> IP
3. can start requesting content to IP

DNS Services:
1. Hostname to IP translation
2. Host aliasing
3. Mail server aliasing
4. Load distribution

3. How does DNS, in usual practice, mitigate the problem of outdated answers being cached and used? Does this mechanism solve the problem completely? Why? **DNS lookup results are subject a specified time-to-live (TTL) that limits how long the results can be cached [2pts]. This increases the chance that the cached results will be refreshed before a user queries their outdated versions. No [1pt]. Since a DNS name-to-value mapping can change at any time, after the change, it is still possible for a user to query the name before the TTL expires [2pts].**