Encryption

Review

1. Find the inverse of 5 mod 11 using the Extended Euclidean Algorithm.

2. Compute $31^{67} \bmod 9$ using fast modular exponentiation.

Solve for s:
$\gcd(5, 11) = \mathbf{1} = \boldsymbol{s}(\mathbf{5}) + \boldsymbol{t}(\mathbf{11})$

$11 = 5(2) + 1$

$1 = -2(5) + 1(11)$

$s = -2$
$s = -2 + 11 = \mathbf{9}$

$67 = 64 + 2 + 1 \qquad 31^{67} = 31^{64} * 31^{2} * 31^{1}$

$31^{1} \bmod 9 = 4$
$31^{2} \bmod 9 = 4 * 4 \bmod 9 = 7$
$31^{4} \bmod 9 = 7 * 7 \bmod 9 = 4$
$31^{8} \bmod 9 = 4 * 4 \bmod 9 = 7$
$31^{16} \bmod 9 = 7 * 7 \bmod 9 = 4$
$31^{32} \bmod 9 = 4 * 4 \bmod 9 = 7$
$31^{64} \bmod 9 = 7 * 7 \bmod 9 = 4$

$31^{67} \bmod 9 = 4 * 7 * 4 \bmod 9 = 28 * 4 \bmod 9 = 1 * 4 \bmod 9 = \mathbf{4}$

Encryption

- Private Key Encryption (symmetric encryption)
  - Same key is used to encrypt and decrypt a message
  - Caesar Cipher

- Public Key Encryption (asymmetric encryption)
  - One key is used to encrypt a message
  - A different key is used to decrypt a message
  - RSA (Rivest-Shamir-Adleman)
  - HTTPS is built using RSA. Enables web economy.
  - See browser demonstration

$p:$ plaintext
$c:$ ciphertext

$c = (p + k) \bmod 26$
$p = (c - k) \bmod 26$

Example

RSA

Use modular exponentiation to compute these quickly

e, N: public key
d: private key

p: a large prime number

q: another large prime number

N: $p * q$ (must be larger than the message)

It is computationally difficult to factor N to find p and q.

t: $(p - 1) * (q - 1)$ (called the *totient*)

e: The encryption exponent. Must be coprime to t.

Usually $2^{16} + 1 = 65537$

65537 is prime, which makes it coprime to t.

In binary it is 10000000000000001. Only two 1's, so performing modular exponentiation is fast and easy.

d: The decryption exponent. It is the inverse of e under mod t

$d * e \bmod t = 1$ (use Extended Euclidean Algorithm to find d)

To encrypt a message m:

$c = m^e \ \mathbf{mod}\ N$

To decrypt a message c:

$m = c^d \ \mathbf{mod}\ N$

In Python:

```
c = pow(m, e, n)
m = pow(c, d, n)
```

Example

How is RSA used?

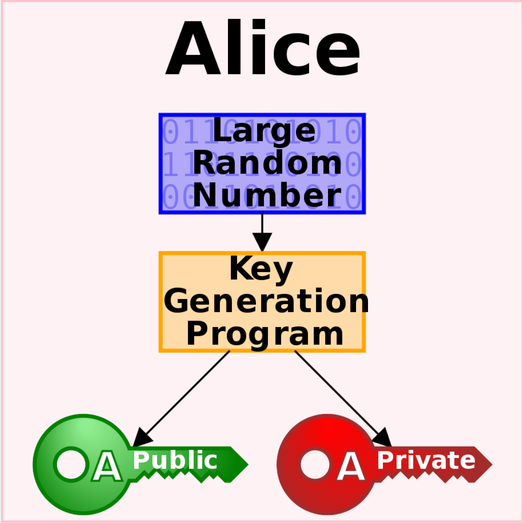
Scenario 1: Secure communication

Alice and Bob want to communicate securely.

1. Alice generates a public/private key pair.

   In RSA, the public key is (N, e) and the private key is (N, d)

   Note that anyone can view Alice's public key, but it won't help them find her private key



https://en.wikipedia.org/wiki/Public-key_cryptography

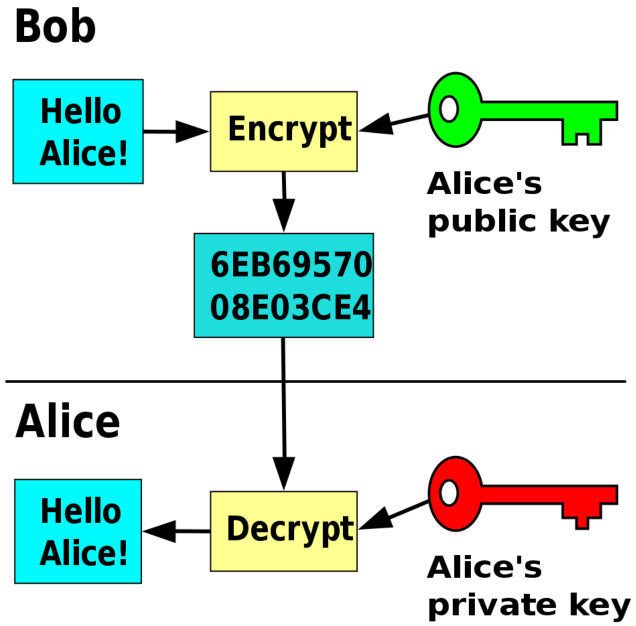
How is RSA used?

2. Alice gives Bob her public key. It is OK if an eavesdropper sees this key.

3. Bob uses Alice's ***public*** key to encrypt a message.

4. Alice uses her ***private*** key to decrypt the message.

Note: Alice must keep her private key a secret. The only way to decrypt a message that was encrypted with her public key is to use the private key.



https://en.wikipedia.org/wiki/Public-key_cryptography

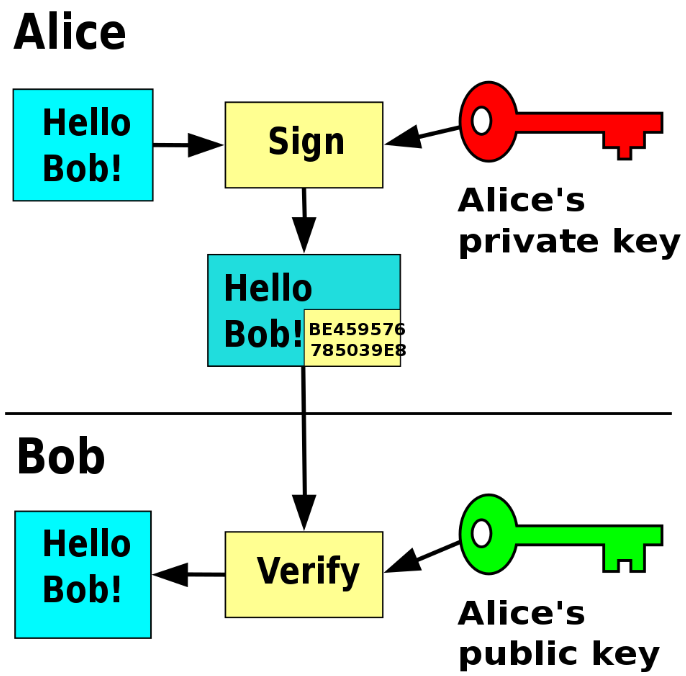
How is RSA used?

Scenario 2: Digitally signing messages

Alice wants to send a message to Bob and prove that she is the person who sent it.

1. Alice encrypts a hash of the message using her *private* key and attaches this encrypted hash to the end of the message.

2. Anyone, including Bob, can view her message. Only Alice's *public* key can decrypt the hash, which proves that the hash must have been encrypted using Alice's *private* key. Thus, only Alice could have sent the message.



https://en.wikipedia.org/wiki/Public-key_cryptography

With a partner:

Use the RSA algorithm to encrypt and decrypt the message "134", using the following values:

$p = 17$
$q = 11$
$e = 7$

1. Find N and t where $\boldsymbol{N = p * q}$ and $\boldsymbol{t = (p - 1) * (q - 1)}$.

2. Use Euclid's extended algorithm to find d, which is the inverse of e mod t.

3. Encrypt the message "134" by computing $\boldsymbol{m^e \bmod N}$. Use fast exponentiation to help you.

4. Now decrypt the encrypted message by computing $\boldsymbol{c^d \bmod N}$. Again, use fast exponentiation to help you.

1. $N = p * q = 17 * 11 = 187$, $t = (p-1)*(q-1) = 16 * 10 = 160$

2. Find d.

Find $\gcd(7, 160)$, then solve $1 = a(7) + b(160)$.

$160 = 7(22) + 6$	$6 = 160 - 22(7)$
$7 = 6(1) + 1$	$1 = 7 - 1(6)$

$$1 = 7 - 1\big(160 - 22(7)\big)$$
$$1 = 23(7) - 1(160)$$

$$d = 23$$

3. Encrypt the message $m = 134$ using $\boldsymbol{m^e} \textbf{mod N}$

$134^7 \bmod 187$ Note that $134^7 = 134^4 * 134^2 * 134^1$

$134^1 \bmod 187 = 134$
$134^2 \bmod 187 = 134 * 134 \bmod 187 = 17956 \bmod 187 = 4$
$134^4 \bmod 187 = 4 * 4 \bmod 187 = 16$

$134^7 \bmod 187 = 16 * 4 * 134 \bmod 187 = 161$

The encrypted message c is 161

4. Decrypt $c = 161$ using $c^d \bmod N$

$161^{23} \bmod 187$ Note that $161^{23} = 161^{16} * 161^4 * 161^2 * 161^1$

$161^1 \bmod 187 = 161$
$161^2 \bmod 187 = 25921 \bmod 187 = 115$
$161^4 \bmod 187 = 115 * 115 \bmod 187 = 13225 \bmod 187 = 135$
$161^8 \bmod 187 = 135 * 135 \bmod 187 = 18225 \bmod 187 = 86$
$161^{16} \bmod 187 = 86 * 86 \bmod 187 = 7396 \bmod 187 = 103$

$161^{23} \bmod 187 = 103 * 135 \ * 115 \ * 161 \bmod 187 = 134$

The decrypted message is 134

RSA demo: Generate and use a real RSA public/private key

1. Generate a private key:

```
$ openssl genrsa -out private.pem 2048

Generating RSA private key, 2048 bit long modulus (2 primes)
..............+++++
.....+++++
e is 65537 (0x010001)
```

Note that *e* is 65537

RSA demo: Generate and use a real RSA public/private key

2. Display the private key:


```
$ cat private.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAv4QWTxhvoqA43Jp8aIi1fAdK8L/Fi3TcbKGp4iys/2lB7JkV
YFniRZrjwHaehiwY/32Qr2tAWeemimSFB1oMD9jiFYvwCC/EXZJapTo0AjAbjikJ
IBsgqYI5sR9f/lslJA/OFJVvckiB/Rkcpg6vmHY37QrV5zahCUEvF++BOPVrBuE1
ILjAv/ZCMbeTVz2D0B0VEb7/xJP6Fk/MtRcvNpK+h4WWmvv6U6MqnrmVoR8ogPBI
KcTQe6xScpYCK5MLIdiBh5YOdZAmWTpnrbdG4phc/70A4vqQ38sZH+7/IT+OpXvs
wT4GqsK5HPajJ+9bxWceATmVl6//O7ozfpRbvQIDAQABAoIBADhPGNQWpjQVkcJi
9bMyhtG5vzH74JQ5Ptn9ylpCsnbCNuzesadLpji66r3a4o8mhg8QlyrMiPYmIxRL
q8VWl3dNWGPnaiKuELlGX7p223XNOTzaGeHDuB2+uUG3rNy3ZbstnnT03vimEvoc
mQzGAqD8GUNrzSqXKE/1R2htNfxNwcvmBoUxN7suKPjhOsEzqRkrfliQVQ+LfEhc
8H/EDKLErn17aKYLDjHb7R3MQH/bKrgf9Ae+I+40IapMAnQpe/uLxEMUIY2am9qs
tc4kTXWo2MHJ3GzRZI86e+0T8N9oeGjBG3xMPNHvfNnngmzjeRvfp9qnsSGCnUIk
Wj1OOs0CgYEA6fDNpeeREH3at3qf0Nto4DksdX1l0CvvfagY+eD5k6rGP8IJ/B7Y
9y1HGBCH1mIXL7NT1F7h0nqJKG7GwiOiXJFwyUyrAaHnWA2LdiD9j8EfhKpzNOuy
zZoSueNqtPSzF2fsvErGQxleUo9bAHOFe32RDuxYvWTm89AsZgLjqpcCgYEA0ZMt
BbpSLUU0Xt9K1ZiP1DeU1vP7eXBu3iJ/I4rda2oyAcdHGNn2qW7zVEhtQzXlJIkT
PJhOYkrzhhQMg2ZjY0LOr32DewVfNfxjMi+hxEizSGblkULdTOO2OyHJtSUarVmE
xS0fLI/H0coKD51dS3AD0W6rGAhvK7GuMBflWssCgYAgROcZC2gnLjUNOgzxclJ1
G23WhwWUQXs3ighn73B9vgC6qrlV3atv7P7xgtY57C3mloXptWzQ67Yfragc/21V
93nnSnwMLZkLvFKQaNyRB8KhOiHKGvj/A1Gx8ny1mUta3yr4jhfrCYTJPpz+4vB2
qEtqE4/qEBELcJuvNpbQ4QKBgAhFAG/LbFaw9mIP+Yn4HSTIlKzur6uZDibhwZaL
cjU82YBMdre6UgtZf2yB9x3B0KriZcnsUJt8TaOqwtukfKN8DV/LWhbOnXUp4lgm
nQF7xOdEeCaFNupDyfsHWEg3kJfqglytbR97BO14RXFcSkQhxWTMMyAdpRVaS39G
UhyFAoGBAI2yrjyHyUbPQjNu2EJY2uvRV9gTV5OeItP6AyRTdXeSUyveMWhYbvsY
Ajs2hF6MPskMu0RA2zKsZfL7KdGL0kFmpCgzta9LSrBoX/GpTs07kMziUhnAzQ3V
teP6oiOB26ZfkkzuzxETKCr2abQvPrVtrSkTt0atK+48Zz5dB1U4
-----END RSA PRIVATE KEY-----
```

RSA demo: Generate and use a real RSA public/private key

3. Generate and display the public key:

```
$ openssl rsa -in private.pem -out public.pem -outform PEM -pubout
writing RSA key

$ cat public.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAv4QWTxhvoqA43Jp8aIi1
fAdK8L/Fi3TcbKGp4iys/2lB7JkVYFniRZrjwHaehiwY/32Qr2tAWeemimSFB1oM
D9jiFYvwCC/EXZJapTo0AjAbjikJIBsgqYI5sR9f/lslJA/OFJVvckiB/Rkcpg6v
mHY37QrV5zahCUEvF++BOPVrBuE1ILjAv/ZCMbeTVz2D0B0VEb7/xJP6Fk/MtRcv
NpK+h4WWmvv6U6MqnrmVoR8ogPBIKcTQe6xScpYCK5MLIdiBh5YOdZAmWTpnrbdG
4phc/70A4vqQ38sZH+7/IT+OpXvswT4GqsK5HPajJ+9bxWceATmVl6//O7ozfpRb
vQIDAQAB
-----END PUBLIC KEY-----
```

RSA demo: Generate and use a real RSA public/private key

4. Inspect the key:

This is $p * q = N$

```
$ openssl rsa -text -in private.pem

RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:bf:84:16:4f:18:6f:a2:a0:38:dc:9a:7c:68:88:
    b5:7c:07:4a:f0:bf:c5:8b:74:dc:6c:a1:a9:e2:2c:
    ac:ff:69:41:ec:99:15:60:59:e2:45:9a:e3:c0:76:
    9e:86:2c:18:ff:7d:90:af:6b:40:59:e7:a6:8a:64:
    85:07:5a:0c:0f:d8:e2:15:8b:f0:08:2f:c4:5d:92:
    5a:a5:3a:34:02:30:1b:8e:29:09:20:1b:20:a9:82:
    39:b1:1f:5f:fe:5b:25:24:0f:ce:14:95:6f:72:48:
    81:fd:19:1c:a6:0e:af:98:76:37:ed:0a:d5:e7:36:
    a1:09:41:2f:17:ef:81:38:f5:6b:06:e1:35:20:b8:
    c0:bf:f6:42:31:b7:93:57:3d:83:d0:1d:15:11:be:
    ff:c4:93:fa:16:4f:cc:b5:17:2f:36:92:be:87:85:
    96:9a:fb:fa:53:a3:2a:9e:b9:95:a1:1f:28:80:f0:
    48:29:c4:d0:7b:ac:52:72:96:02:2b:93:0b:21:d8:
    81:87:96:0e:75:90:26:59:3a:67:ad:b7:46:e2:98:
    5c:ff:bd:00:e2:fa:90:df:cb:19:1f:ee:ff:21:3f:
    8e:a5:7b:ec:c1:3e:06:aa:c2:b9:1c:f6:a3:27:ef:
    5b:c5:67:1e:01:39:95:97:af:ff:3b:ba:33:7e:94:
    5b:bd

...
```

This is e

This is d

```
...

publicExponent: 65537 (0x10001)
privateExponent:
    38:4f:18:d4:16:a6:34:15:91:c2:62:f5:b3:32:86:
    d1:b9:bf:31:fb:e0:94:39:3e:d9:fd:ca:5a:42:b2:
    76:c2:36:ec:de:b1:a7:4b:a6:38:ba:ea:bd:da:e2:
    8f:26:86:0f:10:97:2a:cc:88:f6:26:23:14:4b:ab:
    c5:56:97:77:4d:58:63:e7:6a:22:ae:10:b9:46:5f:
    ba:76:db:75:cd:39:3c:da:19:e1:c3:b8:1d:be:b9:
    41:b7:ac:dc:b7:65:bb:2d:9e:74:f4:de:f8:a6:12:
    fa:1c:99:0c:c6:02:a0:fc:19:43:6b:cd:2a:97:28:
    4f:f5:47:68:6d:35:fc:4d:c1:cb:e6:06:85:31:37:
    bb:2e:28:f8:e1:3a:c1:33:a9:19:2b:7e:58:90:55:
    0f:8b:7c:48:5c:f0:7f:c4:0c:a2:c4:ae:7d:7b:68:
    a6:0b:0e:31:db:ed:1d:cc:40:7f:db:2a:b8:1f:f4:
    07:be:23:ee:34:21:aa:4c:02:74:29:7b:fb:8b:c4:
    43:14:21:8d:9a:9b:da:ac:b5:ce:24:4d:75:a8:d8:
    c1:c9:dc:6c:d1:64:8f:3a:7b:ed:13:f0:df:68:78:
    68:c1:1b:7c:4c:3c:d1:ef:7c:d9:e7:82:6c:e3:79:
    1b:df:a7:da:a7:b1:21:82:9d:42:24:5a:3d:4e:3a:
    cd
```

RSA demo: Generate and use a real RSA public/private key

5. Encrypt a message:

```
$ echo "Hello class" > plain.txt

$ cat plain.txt

Hello class

$ openssl rsautl -encrypt -inkey public.pem -pubin -in plain.txt -out cipher.txt

$ cat cipher.txt

}�4���|8H��x���_�N��,�n{�[�Ĩ�)�.�~��ug�?1������z�Y�%^fM������X���Ȟ�(^�������&
�Ôk;��JG�q�����ʜ%g�������p�搖l���Y��f��C��B�����E:v����9�s����3�Oa�*v
���`�~p�ǂ�S�*�O9�K�)�z��������X��n3�u�����kSmc�|_ ə��RWQ��#�����q��;��p�
```

6. Decrypt a message:

```
$ openssl rsautl -decrypt -inkey private.pem -in cipher.txt -out decrypted.txt

$ cat decrypted.txt

Hello class
```

Additional Exercises:

9.8.5

9.9.1

9.9.4