Putting it all together: Cryptography

- We now have all the tools we need to understand a method for secure information transition on the Internet.
- This is called RSA public key encryption.
- Before we look at that, let's review the basic premises of encoding, and discuss the differences between public and private key encryption.

Encryption

- The basic idea of encryption is to take a message (string of symbols) and to apply a function to it, called the encryption function.
- Let A be the set of possible messages, and B be the set of possible encrypted messages. We seek a function $e : A \to B$ which has an inverse $d : B \to A$, called the decryption function. There is one extra requirement: knowledge of how to compute d should not be obtainable easily, say by looking at a lot of different encoded messages.
- Often, some data called a key can be given telling how to compute e (the *encryption key*), or how to compute d (the *decryption key.*)

Private key systems

- In these systems, the sender of a message uses an encryption key, and somehow passes that key (or a key easily obtainable from it) to the receiver, to use as a decryption key. This key transaction has to be in private.
- The real issue here is how to pass the key. Once that has happened, messages can be sent back and forth for quite a while, presuming that a malicious attacker can only see the message streams.
- We'll look at some examples of private-key systems.

Caesar Shift Encryption

- Start by mapping the 26 letters of the alphabet to $\mathbb{Z}_{25}$. Thus $a \mapsto 0, b \mapsto 1, \ldots, z \mapsto 25$. Call this function α.
- Extend α to strings of letters $a_1 \ldots a_n$ using

$$\alpha(a_1 \ldots a_n) = (\alpha(a_1), \ldots, \alpha(a_n)).$$

  If x is a string of n letters, $\alpha(x)$ is a finite sequence of n numbers. As an example

$$\alpha(\text{"}scrooge\text{"}) = (18, 2, 17, 14, 14, 6, 4).$$

- Now the idea is to shift all the letters over a certain number of spaces. Thus, $a \mapsto d$, $b \mapsto e, \ldots, w \mapsto z, x \mapsto a$. This can be described by the shift function $s_3 : \mathbb{Z}_{25} \to \mathbb{Z}_{25}$: $s_3(m) = (m+3) \bmod 26$. This function can be defined on n-tuples:

$$s_3(m_1, \ldots, m_n) = ((m_1 + 3) \bmod 26, \ldots, (m_n + 3) \bmod 26).$$

- To encode a string w we just compute

$$e(w) = (\alpha^{-1} \circ s_3 \circ \alpha)(w),$$

  where α^{-1} is the inverse function of α. So $e(\text{"}scrooge\text{"}) = \alpha^{-1}(21, 5, 20, 17, 17, 9, 7) = \text{"}vfurrjh\text{"}$.
- The decoding function d is

$$\alpha^{-1} \circ s_{23} \circ \alpha.$$

- The encoding key is 3, and the decoding key is 23, once it's known that shifting is being used.

Security

- Shifting might have been OK for Caesar, but it's completely insecure. For one thing, there are only 26 encoding functions.
- We could get fancier, using $s(x) = (ax + b) \bmod 26$, where a and 26 are relatively prime, but still these are easily cracked.
- Is there a completely secure private-key code? YES.

The one-time pad

- Instead of mapping characters to $\mathbb{Z}_{25}$, we map characters to bitstrings of a fixed length. Say $a \mapsto 00000001, b \mapsto 00000010$, and so on. (This allows a lot more characters to be encoded.)
- A message then maps to a sequence of bitstrings, or just one long bitstring.
- When the sender has digitized the message this way, he uses a random number generator to generate a one-time key k, a bitstring exactly as long as the message. This private key is somehow delivered to the receiver.
- Let m be the digitized message. Then put

$$e_k(m) = m \oplus k$$

  where $\oplus$. is the (bitwise) exclusive or function.
- The decoding function $d_(n) = n \oplus k$. That is, the same key can be used to decode. To see this, note that

$$d_k(e_k(m)) = (m \oplus k) \oplus k = m \oplus (k \oplus k) = m \oplus 0 = m.$$

- Furthermore, if the original message can be recovered from the encoded one, then so can the key. This is because

$$e_k(m) \oplus m = m \oplus k \oplus m = (m \oplus m) \oplus k = 0 + k = k.$$

  Thus if the key is really transmitted securely, and the message is very long, there's no way to decode it an any reasonable amount of time.

DES private-key encryption

- The one-time pad depends on the message being sent. DES (Data Encryption Standard) does not depend on this – it uses fixed size keys, but of a size big enough to foil decryption in almost every practical case.
- From the Internet:

  DES has a 64-bit block size and uses a 56-bit key during execution (8 parity bits are stripped off from the full 64-bit key). DES is a symmetric cryptosystem, specifically a 16-round Feistel cipher, and was originally designed for implementation in hardware. When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt the message.

  Feistel ciphers are a special class of iterated block ciphers, where the ciphertext is calculated from the plaintext by repeated application of the same transformation or "round function". In a Feistel cipher, the text being encrypted is split into two halves. The round function f is applied to one half using a subkey and the output of f is exclusive-ored with the other half. The two halves are then swapped. Each round follows the same pattern except for the last round where there is no swap. A nice feature of a Feistel cipher is that encryption and decryption are structurally identical, though the subkeys used during encryption at each round are taken in reverse order during decryption.

Public Key Cryptosystems

- A persistent problem with private key systems is the need to exchange the private keys. Once this is actually done, short-term security (say for an internet session transmitting a credit card number) can be established with DES.
- You can use another method, called public key cryptography, to encrypt the private key to send to another recipient. (You can also use the method for continuing messages, but it is more expensive and time-consuming.)
- Public key systems work in the following way. There is a sender (Bob) who wishes to *get* encrypted messages from members of a large group. He does this by creating two keys, one for encryption and one for decryption. He keeps the decryption key secret, and publishes the public one to the group. Anybody in this group (say Gloria) can send Bob an encrypted message using Bob's published key. Bob can decrypt the message with his secret key.
- The crucial point is that anyone seeing the public key has no way of figuring out the private, secret key in any time short of millions of years.

Digital Signatures

- Public key cryptography has another nice feature: you can use the system to electronically sign a message in such a way that the recipient knows that it was you who sent it.
- Go back to Bob. Bob can send a signed message to Gloria (the message is not encoded, so watch it, Bob :) He does this as follows. He encrypts a message m using his *decryption* key d and sends Gloria both $d(m)$ and m.
- Gloria computes $e(d(m))$ using Bob's public key. Since e and d are inverse functions, the result should be m. If it is, then Gloria knows that m came from Bob.
- This assumes that Bob's public key really came from him. There are ways to guarantee this, which is where companies like Verisign get into the act – they generate so-called digital certificates for people like Bob.
- All of this is incorporated into your browser. The most common protocol is called SSL – Secure Sockets Layer.

A Public Key

```
-----BEGIN PGP PUBLIC KEY BLOCK----- Version:  2.6.2 mQCNAy/CUuAAAAEE
nu8r/lGCWcrAGE3ZMUZmTET4WUWNpN6Lq/0p7iQRW8SyKl94KK1vnDmwQEZpuZ2c s1DY
tB5DaHJpcyBOZXdtYW4gPGNocmlzbitAY211LmVkdT6JAJUDBRAx4w9Pa5+6fmjZ LNkB
ZY6Xb6brrpypQ0HKlWIxTgUvM/jxflOWcOid2pdzWYTp/kwtaBFZq/bZ6H8CFR/G Llgr
Kme2PYuvFdQ95QEB0PsD/3QOkPY5F1pA9Oe3R4kYn77ryfy9tEWyyXV88hN0HIL5 2+eU
Mck2mAh3kvDzTAEQYgw2dP5qEDy5lKazdu6F5TkmAUEjRvxidj2rtgqi/U2oN3OI iQBV
jYSG9xzCF785wYTH7/qnvR87eDUAUDC0ZdoA6/43O81l8k9EuWmiyYkAlQMFEC/D d/9b
JnWqGrfRGthejXplzHeKRyw99jMayT1l6OZm21UCVVZCCnx0PZsQZkynpNHuGbfn ZOq0
iQCVAwUQMdtkwaky2tFFGJm1AQF6IQQAiTALF68UJnkfDYK/Prg7nzr7zl7bE5Yy RhTU
/kEIMcDjHWv2U4anaggZZrxkjBwGy4uXMiebZFZm+wyJMEfuBsVpRpHvD0FrOyTw RhLq
IYOhNemufSQrFhM1bWmtn3zSJ2pU5Ghd2Nsk6ZuogcGYWUjb4fyK1/80vb09GmkV AHFB
0uVkqyYp68PLxzC9pkxajYkAVQIFEDDKIf1SMVxYLwenQQEBo3gB/R57Mldr8L7e lm+T
HgbCkAOoStaJAJUDBRAwzCqi4uW74fteFRkBATYVBACqdefiRTl60Br5fqAlALnU wKeY
fu4vW1jmjSh7B63fLbQZDB3ZeDCOPbl+JNg8BtuoNExoEsiyNJcYqCxLtXP2fQjJ 0lWC
CVMsBMcstj+GWZZ/9W8Sk4eQG6Rqkq6yTKDR7RRTznFi9T7F5OvjOau2WNevO9uN z1pJ
KED4Y8w/f9lAfcBOOPKLefeWFqQyXxrAiQBVAgUQL84STCHgNn+d3oDNAQGBSwH+ Lcb4
7bRzPXs+FHWtjTqJNdlXXIkAlQMFEDCPv+/whgABgdZIZQEB1e4D/ix1WDnyf4yC JyrN
FzFFbgbXfjhk1pi2fHIL67tjumMYRS6z4tURpV38V0VzJelxptxVKLaqLEoqccuZ AQcv
U9AFMejL1orQzoxrwxgXi1FstHKGj7QPrxWg28mFduKgAYx5+lBCLP+PG8bB1yzR iomr
mY348dDji8dg3oigT0leBowE5UlVckWVNdjMpnMlmMmJAJUDBRAwjE/fDTCC1xNF MJEB
O2eAOcV66JtX/9ek1liAJxXP+YwrNZ9ArKfDmuv4I6b953pGZWesrTpCNtgsKcNg SSge
43QhMOLJlujOuQEBmvEEAKSJzinc/PTsGLAA6TTxtbLfaz7umbJLQRgPKNDZDkFH DOXn
```

```
tSjit/BjADSaVfxGrZUdOWt7zD4oRBJf2DvJQJm7YFV24L+xYRCWGUtWAY2x85rx iQCV
a7VjnnjmhvHmhqqAA07DBEEhR/ogt9uCusBr8a5lYWOfNaKIqE/CZtHYSjN55PX9 UK5E
tRF+i8M5PAmJAJUDBRAwh+aXqS78h8RKpnkBAdszBADDfbmazmCTQsPF5769C2n3 p6lK
WlMC9HN0Nr2/GgcTMb54zgAScAhDJkigerUa20s5RAlvqLgBMGVoOr6uj/8bf7qw EvnV
tXYWMfz4NXNn9QgjTFREtwx7U/GDk4GvBPIDxA2f15L/T1dl0oy9a6K3a1644oKY ub1n
ERhIYIoIjX1zmikjn/yY/aMWnfAcEaE8TuNaZOeipXcTUP8eMeXmcB/Fs56UhlpN LNdP
AhkBAedwAvoC2iNsqw4zuvRxUJ4XsVsNe2jse9y8MUs3vx+txfBOb2AvbJx0l91i 4Jcu
h36Bi0M78a6JAJUDBRAwh5oDFjW+UJpn240BASrDA/wPvOTZdcCQP8CCvDKG17rH Ia+G
uO9VlRNuW5z0w14JTTgu7VM2rPa2C5RPIFWBL8Zre0Hdl2HZGX6GyZ8ynzvDNWJI wCha
ozgh8f7e+a2aRh1hrayigiBmGBkeOoljXYbtmzaW0xRqO8xi0g9ZWlKW/1+IUglf AlYo
Ngp+KnjCiFAqVQEB3sQEAJ5a6aQcGD7HzPGBsWBxUuVADKyoxMOP736xohHAhzYk 88FZ
+8CnLPRtZOs5bKC0QwZxhg+zgqiv4ZRBtyrcqMR1AxSAO3rO2SIqJKep58fOph9u iQCV
u+YzL3+RVmlcIm0MOdOrIHI+dOexvpJp1lStK/WcTZewapbQa0Akoz2xtZtbVpHM ImH0
O0dQMTgyHpmJAJUDBRAwhuADAOCD9+WgeSkBAbP+A/4sPXJG2x5HTNfb5+GEtvbz W9in
m6NDG2T0wiF+sNMQqhW7XbfwZtoEeQyPjV6uf2p74jjZsAI/5lDB6B0A8v2A0q1D esJ3
owx8IFhv9O4AXtxKF8F7jHXUzrcqcT9ckwE0QP6368Hp42hN/lQVn5aI4ScAbNpK OuNt
bsgt+sbfxBLDntNzUdJlRYOHDqwjkGcoiQCVAwUQMIb5z8qsqgOTiAmRAQHyGwP/ S8dC
vvCXsri+r/5XsbSKG7uBifUGoz9CoBEtyz6Ztxfq38Of5uuF4TKK32ci4YXTgKdg Eg2a
x4sBAVCYArkBhCiofkbjcAfQ8q5+6T8YoQbEYoHvsilKrR/O0TZHAT3nZZtQook6 DZFb
iQBVAwUQL8YfaEay+yO2I8eRAQE7/AH/XCCOgh9FfjAjVg0cRws7e0bAhOpCTVAx ABDm
OMXFLUdtDb+QbQEBGPsD/2j3/3njMcqiQM9tAgLJ+prqaaB1mI76WPuGsdyALhSb eDUZ
VkZnUyQPxsnZEb73DUOUpfOhGideqJYPged6J9zhLDDR+WM39eFrKPQoZz6rTG0p iQCV
n9DeWfHj6WyUDkJZebAjTpywn9ey2l2RlfCUtmq1IxJQTS99vug7iRU1ptVsMvm8 7C+W
-----END PGP PUBLIC KEY BLOCK-----
```

RSA Encryption

- Public key cryptography was introduced in 1974 by Diffie and Hellman, at Stanford. They proposed a public key system which, as it turns out, was easy to break.
- In 1976 Rivest, Shamir, and Adleman (MIT) proposed the RSA system. It still is used today. (Visit www.rsa.com.)
- Their method uses the number theory we've been learning.
- Here's how you get an encryption key. Secretly find two huge prime numbers p and q Test primality using probabalistic primality tests. Then find another number e, relatively prime to $(p-1)(q-1)$. This is not hard either; fairly small prime numbers usually work.
- Publish the pair of numbers (pq, e). That's your public key. Note that if you don't say what p and q are, then factoring pq is thought to be an impossible task.

RSA encryption

- To encrypt a message:
- Encode the message into strings of numbers; e.g. "STOP" = 18191415.
- Break the message into blocks depending on the length of the representation of pq. For example, when $p = 43$, $q = 59$, and $e = 13$:

$$pq = 2537 : \quad M_1 = 1819; M_2 = 1415.$$

- Encode each block separately using

$$e(M) = M^e \bmod pq.$$

- For example, $e(M_1) = (1819^{13} \bmod 2537 = 2081$, and $e(M_2) = (1415)^{13} \bmod 2537 = 2182$. Send $e(M_1)$ followed by $e(M_2)$, etc.

RSA decryption

- You are the creator of the public key (pq, e), and you know p and q.
- Using Euclid, find a multiplicative inverse s of e modulo $(p-1)(q-1)$.
- Let C be (the value of) an encrypted block. To decrypt, calculate $d(C) = C^s \bmod pq$.
- We need to do two things: we need to show how modular exponentiation can be carried out rapidly, and we need to prove that e and d are functional inverses of each other.