Key Management

deals with generation, exchange, storage, use, destruction, and replacement of keys
depending on the cryptosystem, the way these are done will vary

key length
    security of the cryptosystem depends on the strength of the algorithm and the length of the key
    key length directly dictates the time it takes to brute-force a cryptosystem

    symmetric cryptography key length
        an 8-bit symmetric key will have $2^8=256$ different possible keys
        a 128-bit symmetric key will have $2^{128}=3.4*10^{38}$ different possible keys
            which would take approximately 10 years to brute-force
        the longer the key, the better the security
        Michael J. Wiener designed a machine that could crack DES with 56-bit keys in less than 7 hours
            DES is not considered secure anymore

    asymmetric cryptography key length
        as you know, RSA is based on the factorization of large numbers
        this is considered a one-way trapdoor function
        the problem is "probably" hard

        calculations in asymmetric cryptography tend to be computationally expensive
        obviously, it's easier to compute with shorter keys
        but using short keys compromises security
        it's not easy to balance using key length for security with computational requirements

    comparing symmetric and asymmetric cryptography
        for keys of equal length, symmetric cryptography is considerably more secure
        comparable security and key length for both types of cryptography:

Symmetric (AES) Key Length (bits)	Asymmetric (RSA) Key Length (bits)
56	384
64	512
80	769
112	1,792
128	2,304

attacks on keys
    brute force attack involves knowing the ciphertext
    sometimes, we try to find the key in order to then obtain the plaintext
    sometimes we actually know the plaintext and try to find the key
        e.g., we intercept ciphertext but also know the plaintext decrypted at the receiver
        this is usually much faster

    how to efficiently brute force a key
        distributed attacks have many machines brute force a subset of the keyspace
        attackers sometimes use software for this
        they also sometimes use viruses and botnets

birthday attack

based on the birthday paradox and is mostly useful in finding hash collisions

scenario 1: given a birthday (e.g., June 6)

~253 people must be in the room to find another person with the same birthday

i.e., where the probability of this is at least 50%

like flipping a coin (one side will absolutely turn up)

try in the class

$\left(\frac{364}{365}\right)^n$ gives us the probability of n people not having the same birthday

(i.e., having it on one of the other 364 days of the year

to solve for n when we want 50% probability:

$$\begin{aligned}
\left(\frac{364}{365}\right)^n &= 0.5 \\
\log\left(\frac{364}{365}\right)^n &= \log(0.5) \\
n\log\left(\frac{364}{365}\right) &= \log(0.5) \\
n &= \frac{\log(0.5)}{\log\left(\frac{364}{365}\right)} \\
n &= 252.65
\end{aligned}$$

scenario 2: if **no** day is specified

~23 people must be in the room to find two people who share a common birthday

try in the class

the idea:

P(event occurs) + P(event doesn't occur) = 1

for this:

P(two people share a birthday) + P(no two people share a birthday) = 1

P(two people share a birthday) = 1 – P(no two people share a birthday)

what's the probability that no two people share a birthday?

the first person can have any birthday

the second must have a different birthday (that's 364 other days)

the probability that two people have different birthdays is $\frac{365}{365} * \frac{364}{365} = 0.9973$

adding a third means having $\frac{363}{365}$ other days to choose from

and so on…

P(three people have different birthdays) = $\frac{365}{365} * \frac{364}{365} * \frac{363}{365} = 0.9918$

and so on…

P(n people have different birthdays) = $\frac{365}{365} * \frac{364}{365} * \ldots \frac{365-n+1}{365}$

the numerators express how many ways 365 days can be ordered, n at a time that's just n permutations of 365

and is well known to be $\frac{365!}{(365-n)!}$

collectively, the denominators are just 365^n

therefore, for a group size of n: $\frac{365!}{365^n(365-n)!}$

of course, the probability that two people have the same birthdays is: $1-\frac{365!}{365^n(365-n)!}$

we can try values for n to get 50% probability:

$$1-\frac{365!}{365^{23}(365-23)!}=1-\frac{365!}{365^{23}342!}=0.5073$$

generally, if a one-way hash function is secure and produces an m-bit output

finding a message that hashes to a given value (scenario 1) would take 2^m tries

finding two messages that hash to the same value (scenario 2) would only take $2^{\frac{m}{2}}$

key secrecy

maintaining key secrecy is one of the most challenging tasks in cryptography

usually, it is easier to use social engineering to obtain the keys than to attack the actual cryptosystem

think about this!

a key must be as secure as the data

if the key is compromised, the entire cryptosystem is compromised

a few things to avoid

using reduced key space

using poor keys

using non-random keys

secure keys can be generated using key crunching

the process of using some passphrase to generate a secret key – and hashing that key

the hash value is used as the key

this is what's used in WEP and WPA2, for example

key exchange

some protocols deal with key exchange or key distribution

these protocols are mainly known as key establishment protocols

key transport protocol

one party selects the key and sends it to another party

the chances of selecting common or weak keys is quite high

key agreement protocol

both parties work together to mutually agree on a key

there is a very low chance of generating weak keys

since both parties use protocols involving secure computations

- key distribution in symmetric cryptography
  - naive approach
    - each user shares secret keys with all of the other users
    - the appropriate shared key is used to share some message secretly with another user
    - the n^2 key distribution problem
      - establishes pairwise secret keys among all users
      - the total number of keys (stored): $n(n-1)=n^2$ (approximately)
        - this is quadratic complexity
      - the total number of key pairs: $\frac{n(n-1)}{2}=\binom{n}{2}$
      - the total number of keys required is HUGE
    - adding a new user
      - basically, awfully painful
      - a new key pair will have to be generated for the new user
        - with each of the other users in the network
  - key distribution center (KDC) approach
    - uses a central trusted authority
    - the authority shares one key (known as the "key encryption key")
    - the authority only encrypts session keys whenever needed – with every user
    - the authority issues a session key whenever one user wants to send something securely to another
    - the authority generates a session key
      - and encrypts this session key using the keys shared with the receiver and sender
    - the authority sends this key to both parties so that they can securely communicate
    - the total number of keys (stored): $2n$
      - this is linear complexity
    - the total number of key pairs: n
    - a new user can be added by
      - establishing only one secure channel between the KDC and the user
      - and exchanging a key between them
    - Kerberos, a widely-used network authentication protocol, uses KDC
    - some limitations of KDC:
      - if it fails, everything fails (basically, a single point of failure)
      - there's no perfect forward secrecy
        - if the "key encryption key" is compromised, so is everything in the past too
      - vulnerable to replay attacks and key confirmation attacks
        - playing back sniffed packets could get someone information
        - pretending to be the KDC could cause a user to send information
- key distribution in asymmetric cryptography
  - three families of asymmetric cryptography
    - RSA (already discussed)
    - discrete log (discussed later in the course)
    - elliptic curve (discussed later in the course)

asymmetric cryptography is computationally expensive
so it is usually only used to securely distribute/exchange the key for symmetric cryptography

Merkle's puzzle

- this is an early form of a public-key cryptosystem
- but it doesn't technically use public-key cryptography as we know it now
- the idea is very close though

- in this protocol, two parties agree on a shared secret key
- they do this by exchanging messages without having any prior secret in common

- the protocol:
  - Alice and Bob want to communicate
  - Bob creates a large number of puzzles, each of moderate complexity
    - i.e., Alice must be able to solve these puzzles in a reasonable amount of time
  - each puzzle has an encrypted message that is encrypted with an unknown key
    - the key must be short enough to allow brute-forcing
  - the encrypted message contains an identifier (ID) and a session key
  - Alice brute forces one of the encrypted messages by trying all possible keys
  - Alice sends the resulting (decrypted) ID
    - to let Bob know which session key she is going to use
  - both parties now know the secret session key

  - if Eve was eavesdropping, she would have to solve all the puzzles to find the key
  - computationally, it is much more difficult for Eve
    - so this protocol is "fairly" secure