CS 858 –
Mobile Privacy and Security

Fall 2016

Background

Workshops/Conferences of Interest

- Mobile Security and Privacy: SPSM, MoST, WiSec
- Security: USENIX Security, ACM CCS, IEEE Security & Privacy, NDSS
- Privacy: PETS
- Usability: SOUPS (including this year's authentication workshop), CHI, USEC (NDSS workshop)
- Pervasive/mobile computing: UbiComp, PerCom, Pervasive, MobiSys

Survey Articles on Mobile Privacy And Security

- SoK: Privacy on Mobile Devices – It's Complicated, Spensky et al., PETS 2016
  - Very readable, but quite brief
- SoK: Lessons Learned From Android Security Research For Appified Software Platforms, Acar et al., Oakland 2016
  - Good discussion of solved/still unsolved problems
- Securing Android: A Survey, Taxonomy, and Challenges and Toward Engineering a Secure Android Ecosystem: A Survey of Existing Techniques, ACM Computing Surveys
  - Very dense, but good coverage and useful if there's a particular topic that you are interested in

Cryptography/Security Books

- Pfleeger & Pfleeger, Security in Computing
- Doug Stinson, Cryptography- Theory and Practice
- Bruce Schneier, Applied Cryptography
- Ross Anderson, Security Engineering
  http://www.cl.cam.ac.uk/~rja14/book.html
- Viega & McGraw, Building Secure Software
- Cranor & Garfinkel, Security and Usability

Survey Articles on Mobile Privacy And Security

- Previous articles all ignore smartphone authentication

- Continuous User Authentication on Mobile Devices: Recent progress and remaining challenges, Patel et al., IEEE Signal Processing Magazine
- Surveying the Development of Biometric User Authentication on Mobile Phones, Meng et al., IEEE Communications Surveys Tutorials

Android Security

- Android Security, Google

- Understanding Android Security, Enck et al., IEEE Security and Privacy

Presentations:

- Android Security Essentials, Pragati, Oscon
- Understanding Android's Security Framework , Enck
- Android's security architecture, Elenkov, Android Security Symposium

iOS Security

- iOS Security, Apple
- Behind the Scenes with iOS Security, Krstic, Blackhat

Related Courses

- Every term
  - CS 658 – Computer Security and Privacy
- Fall 2016:
  - CS 858 – Computing on Encrypted Data
- C&O 485/685 The Mathematics of Public-Key Cryptography
- C&O 487 Applied Cryptography

What is Security?

- In the context of computers, security generally means three things:
- Confidentiality: Access to systems or data is limited to authorized parties
- Integrity: When you ask for data, you get the “right" data
- Availability: The system or data is there when you want it
- A computing system is said to be secure if it has all three properties
  - Well, usually

What is Privacy?

- There are many definitions of privacy
- A useful one: “informational self-determination“
- This means that you get to control information about you
- “Control" means many things:
  - Who gets to see it
  - Who gets to use it
  - What they can use it for
  - Who they can give it to
  - etc.

How secure should we make it?

- Principle of Easiest Penetration
  - “A system is only as strong as its weakest link“
  - The attacker will go after whatever part of the system is easiest for him, not most convenient for you
  - In order to build secure systems, we need to learn how to think like an attacker!

- Principle of Adequate Protection
  - “Security is economics“
  - Don't spend $100,000 to protect a system that can only cause $1,000 in damage

Access Control

- In general, access control has three goals:
  - Check every access: Else system might fail to notice that access has been revoked
  - Enforce least privilege: Grant program access only to smallest number of objects required to perform a task
    - Often violated for Android apps, see “Android Permissions Demystified” by Felt et al.
  - Verify acceptable use: Limit types of activity that can be performed on an object
    - E.g., for integrity reasons (ADTs)

User Authentication

- Computer systems often have to identify and authenticate users before authorizing them
- Identification: Who are you?
- Authentication: Prove it!

Authentication Factors

- Four classes of authentication factors
- Something the user knows
  - Password, PIN, pattern, answer to “secret question”
- Something the user has
  - ATM card, badge, browser cookie, physical key, uniform, smartphone
- Something the user is
  - Biometrics (fingerprint, voice pattern, face,…), behaviour (swiping pattern, visited websites,…)
- Something about the user's context
  - Location, time, devices in proximity

Combination of Auth. Factors

- Different classes of authentication factors can be combined for more solid authentication
  - Two- or multi-factor authentication

Cryptography

- What is cryptography?
- Related fields:
  - Cryptography ("secret writing"): Making secret messages
    - Turning plaintext (an ordinary readable message) into ciphertext (secret messages that are "hard" to read)
  - Cryptanalysis: Breaking secret messages
    - Recovering the plaintext from the ciphertext
- Cryptology is the science which studies these both
- The point of cryptography is to send secure messages over an insecure medium (like the Internet)

Dramatis Personae

When talking about cryptography, we often use a standard cast of characters

- Alice, Bob, Carol, Dave
  - People (usually honest) who wish to communicate
- Eve
  - A passive eavesdropper, who can listen to any transmitted messages
- Mallory
  - An active Man-In-The-Middle, who can listen to, and modify, insert, or delete, transmitted messages
- Trent
  - A Trusted Third Party

Building blocks

- Cryptography contains three major types of components
  - Secrecy components
    - Preventing Eve from reading Alice's messages
  - Integrity components
    - Preventing Mallory from modifying Alice's messages
  - Authenticity components
    - Preventing Mallory from impersonating Alice

Kerckhoffs' Principle (19th c.)

The security of a cryptosystem should not rely on a secret that's hard (or expensive) to change

- So don't have secret encryption methods
  - Then what do we do?
  - Have a large class of encryption methods, instead
    - Hopefully, they're all equally strong
  - Make the class public information
  - Use a secret key to specify which one you're using
  - It's easy to change the key; it's usually just a smallish number

Kerckhoffs' Principle (19th c.)

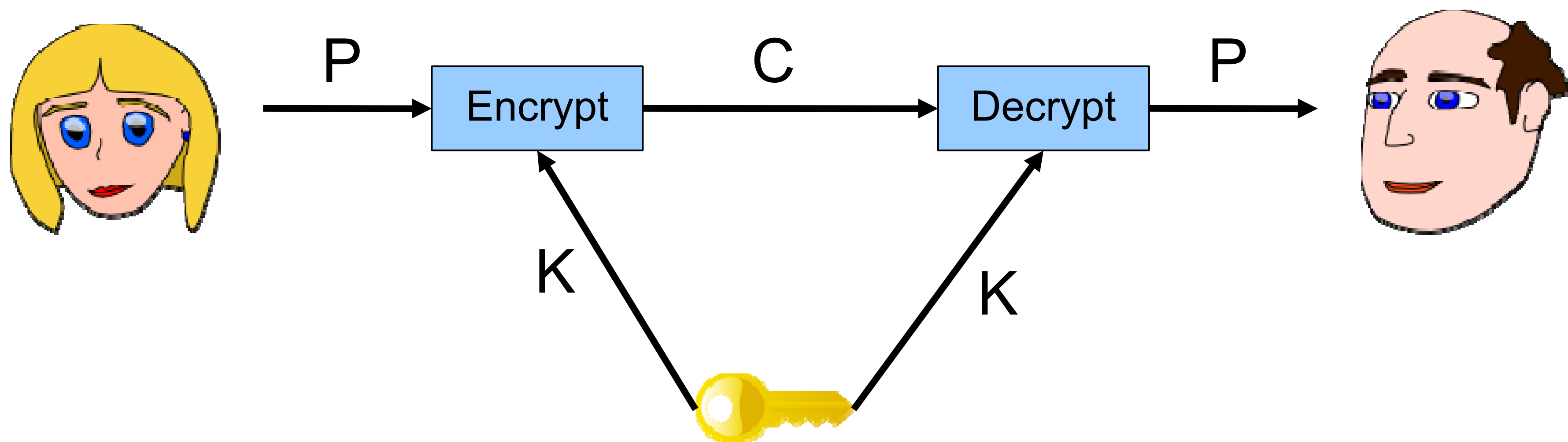
- This has a number of implications:
  - The system is at most as secure as the number of keys
  - Eve can just try them all, until she finds the right one
  - A strong cryptosystem is one where that's the best Eve can do
    - With weaker systems, there are shortcuts to finding the key
  - Example: newspaper cryptogram has 403,291,461,126,605,635,584,000,000 possible keys
  - But you don't try them all; it's way easier than that!

Strong cryptosystems

- What information do we assume the attacker (Eve) has when she's trying to break our system?
- She may:
  - Know the algorithm (the public class of encryption methods)
  - Know some part of the plaintext
  - Know a number (maybe a large number) of corresponding plaintext/ciphertext pairs
  - Have access to an encryption and/or decryption *oracle*
- And we still want to prevent Eve from learning the key!

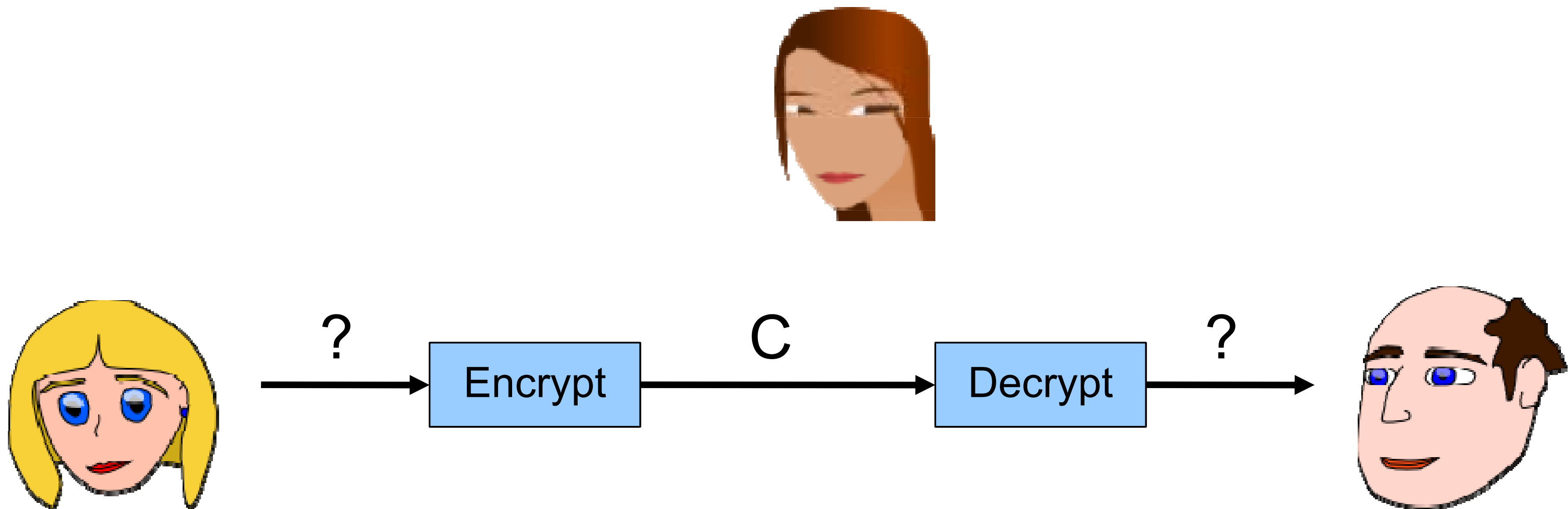
Symmetric encryption

- Symmetric encryption is the simplest form of cryptography
- Used for thousands of years
- The key Alice uses to encrypt the message is the same as the key Bob uses to decrypt it



Symmetric encryption

- Eve, not knowing the key, should not be able to recover the plaintext



Perfect symmetric encryption

- Is it possible to make a completely unbreakable cryptosystem?
- Yes: the One-Time Pad
- It's also very simple:
  - The key is a truly random bitstring of the same length as the message
  - The “Encrypt” and “Decrypt” functions are each just XOR

One-time pad

- But! It's very hard to use correctly
  - The key must be truly random, not pseudorandom
  - The key must never be used more than once!
    - A “two-time pad” is insecure!
- Used in the Washington / Moscow hotline for many years
- Q: Why does “try every key” not work here?
- Q: How do you share that much secret key?

Computational security

- In contrast to OTP's “perfect” or “information-theoretic” security, most cryptosystems have “computational” security
  - This means that it's certain they can be broken, given enough work by Eve
- How much is “enough”?
- At worst, Eve tries every key
  - How long that takes depends on how long the keys are
  - But it only takes this long if there are no “shortcuts”!

Some data points

- One computer can try about 17 million keys per second
- A medium-sized corporate or research lab may have 100 computers
- The BOINC project has 2 million computers

Berkeley Open Infrastructure for Network Computing

- Remember that most computers are idle most of the time (they're waiting for you to type something); getting them to crack keys in their spare time doesn't actually cost anything extra

40-bit crypto

This was the US legal export limit for a long time

2^{40} = 1,099,511,627,776 possible keys

- One computer: 18 hours
- One lab: 11 minutes
- BOINC: 30 ms

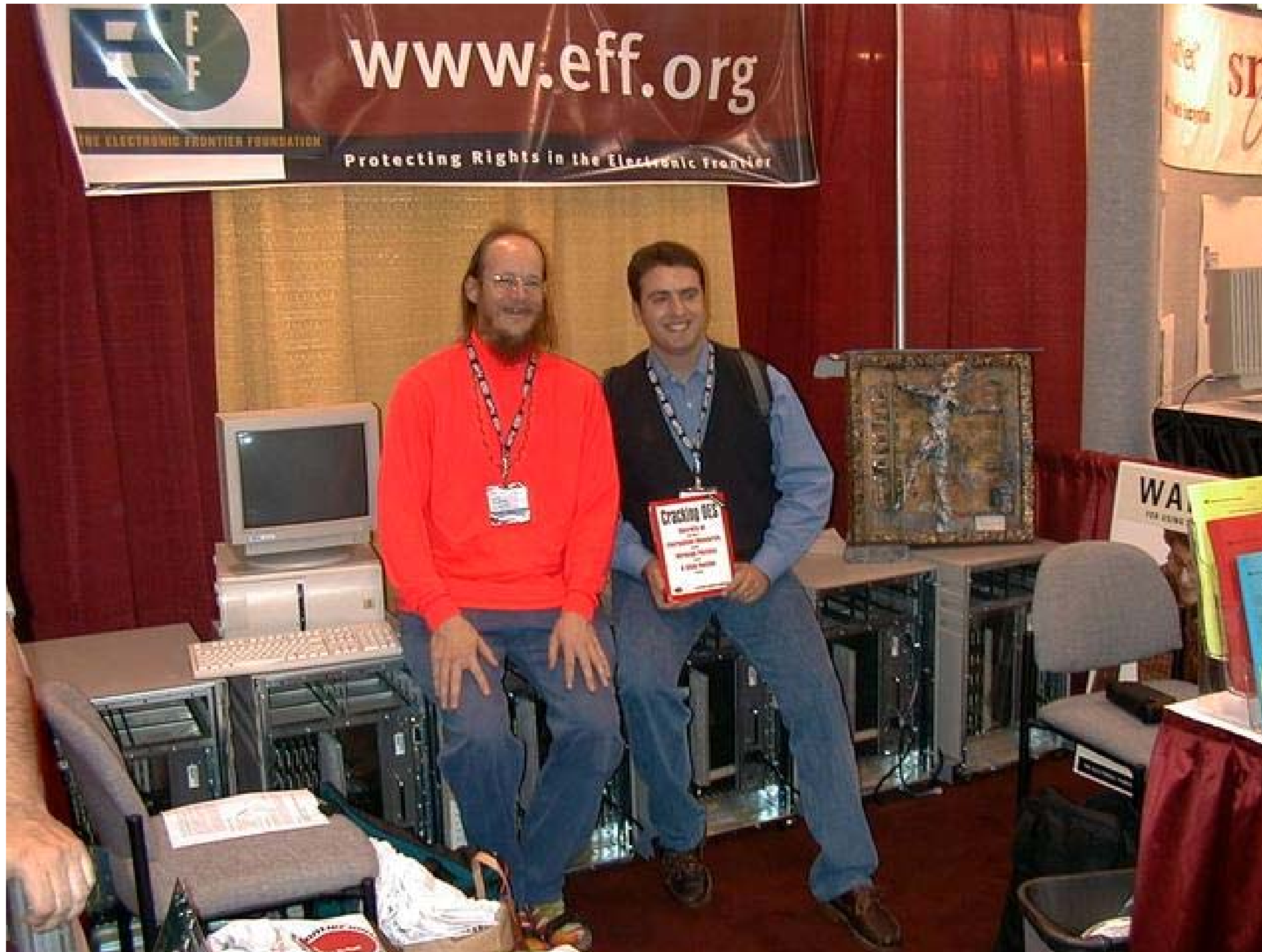
56-bit crypto

This was the US government standard (DES) for a long time

2^{56} = 72,057,594,037,927,936 possible keys

- One computer: 134 years
- One lab: 16 months
- BOINC: 36 minutes

Cracking DES



“DES cracker” machine of Electronic Frontier Foundation

128-bit crypto

This is the modern standard

2^{128} = 340,282,366,920,938,463,463,374,607, 431,768,211,456 possible keys

- One computer: 635 thousand million million million years
- One lab: 6 thousand million million million years
- BOINC: 300 thousand million million years

Well, we cheated a bit

- This isn't really true, since computers get faster over time
  - A better strategy for breaking 128-bit crypto is just to wait until computers get 2^{88} times faster, then break it on one computer in 18 hours.
  - How long do we wait? Moore's law says 132 years.
  - If we believe Moore's law will keep on working, we'll be able to break 128-bit crypto in 132 years (and 18 hours) :-)
    - Q: Do we believe this?

An even better strategy

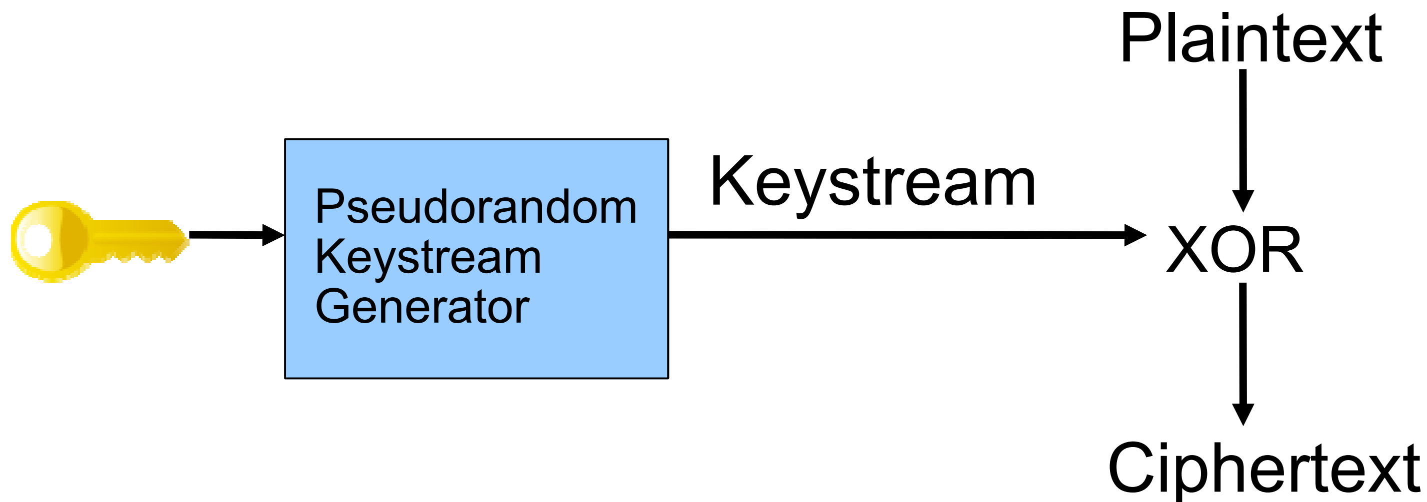
- Don't break the crypto at all!
- There are always weaker parts of the system to attack
- The point of cryptography is to make sure the information transfer is not the weakest link

Types of symmetric ciphers

- Symmetric ciphers come in two major classes
  - Stream ciphers
  - Block ciphers

Stream ciphers

- A stream cipher is what you get if you take the One-Time Pad, but use a pseudorandom keystream instead of a truly random one



- RC4 is the most commonly used stream cipher on the Internet today

Stream ciphers

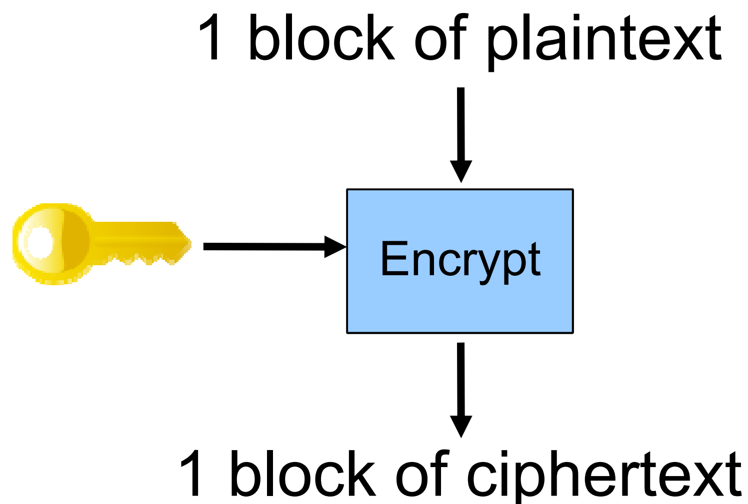
- Stream ciphers can be very fast
  - This is useful if you need to send a lot of data securely
- But they can be tricky to use correctly!
  - What happens if you use the same key to encrypt two different messages?
  - How would you solve this problem without requiring a new shared secret key for each message?
- WEP, PPTP are great examples of how not to use stream ciphers

Block ciphers

- Notice what happens in a stream cipher if you change just one bit of the plaintext
  - This is because stream ciphers operate on the message one bit at a time
- We can also use block ciphers
  - Block ciphers operate on the message one block at a time
  - Blocks are usually 64 or 128 bits long
- AES is the block cipher everyone should use today
  - Unless you have a really, really good reason

Modes of operation

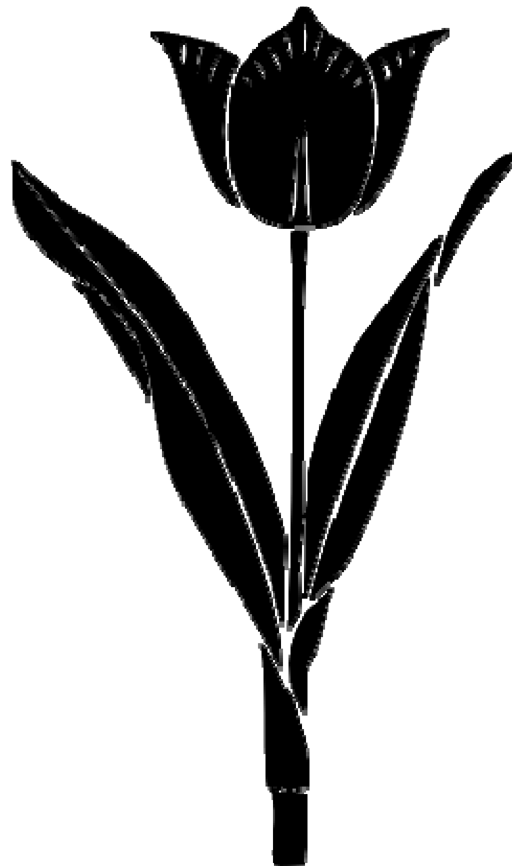
- Block ciphers work like this:

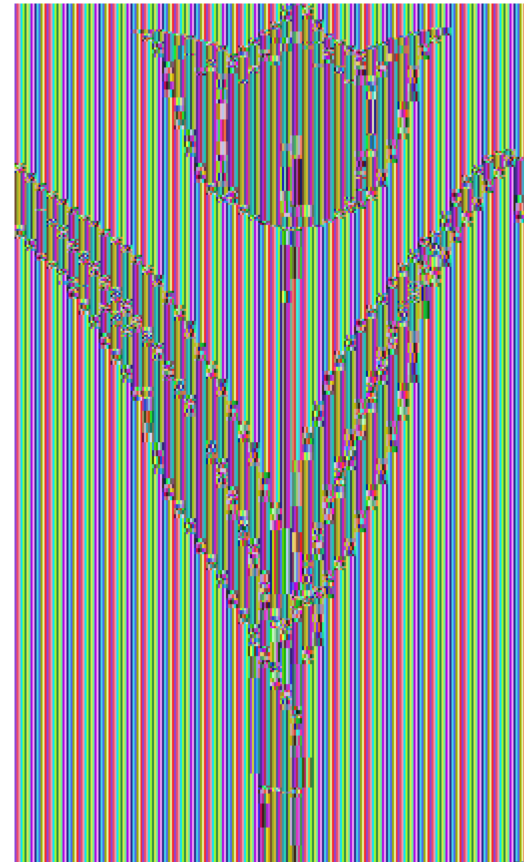


- But what happens when the plaintext is larger than one block?
  - The choice of what to do with multiple blocks is called the mode of operation of the block cipher

Modes of operation

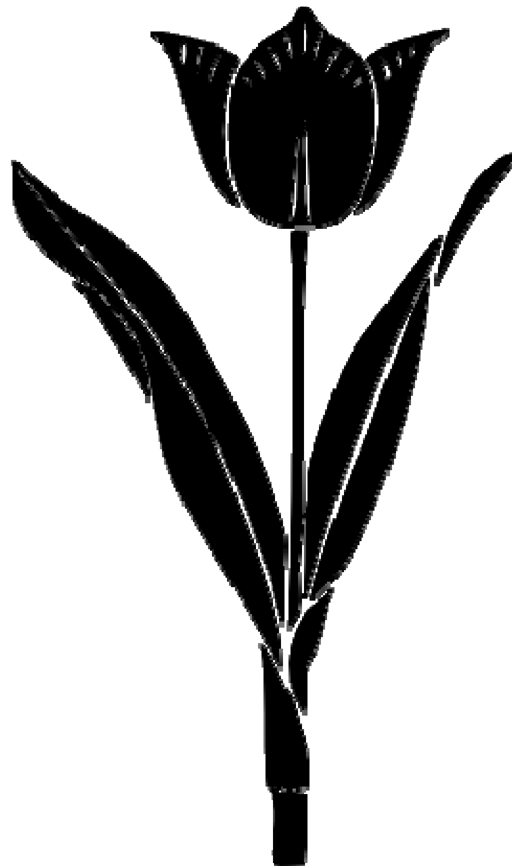
- The simplest thing to do is just to encrypt each successive block separately.
  - This is called Electronic Code Book (ECB) mode
- But if there are repeated blocks in the plaintext, you'll see the same repeating patterns in the ciphertext:

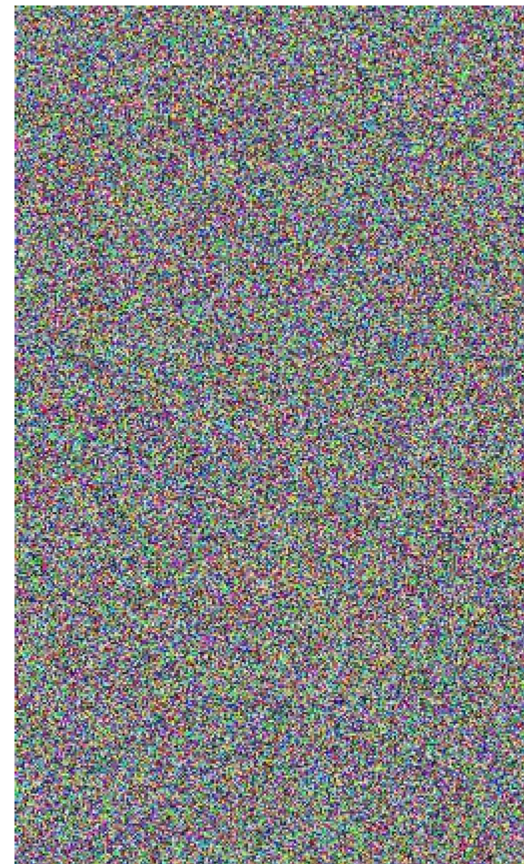




Modes of operation

- There are much better modes of operation to choose from
  - Common ones include Cipher Block Chaining (CBC) and Counter (CTR) modes
- Patterns in the plaintext are no longer exposed
- But you need an IV (Initial Value), which acts much like a salt





Key exchange

- The hard part of symmetric ciphers is:
  - How do Alice and Bob share the secret key?
    - Meet in person; diplomatic courier
  - In general this is very hard
- Or, we invent new technology...

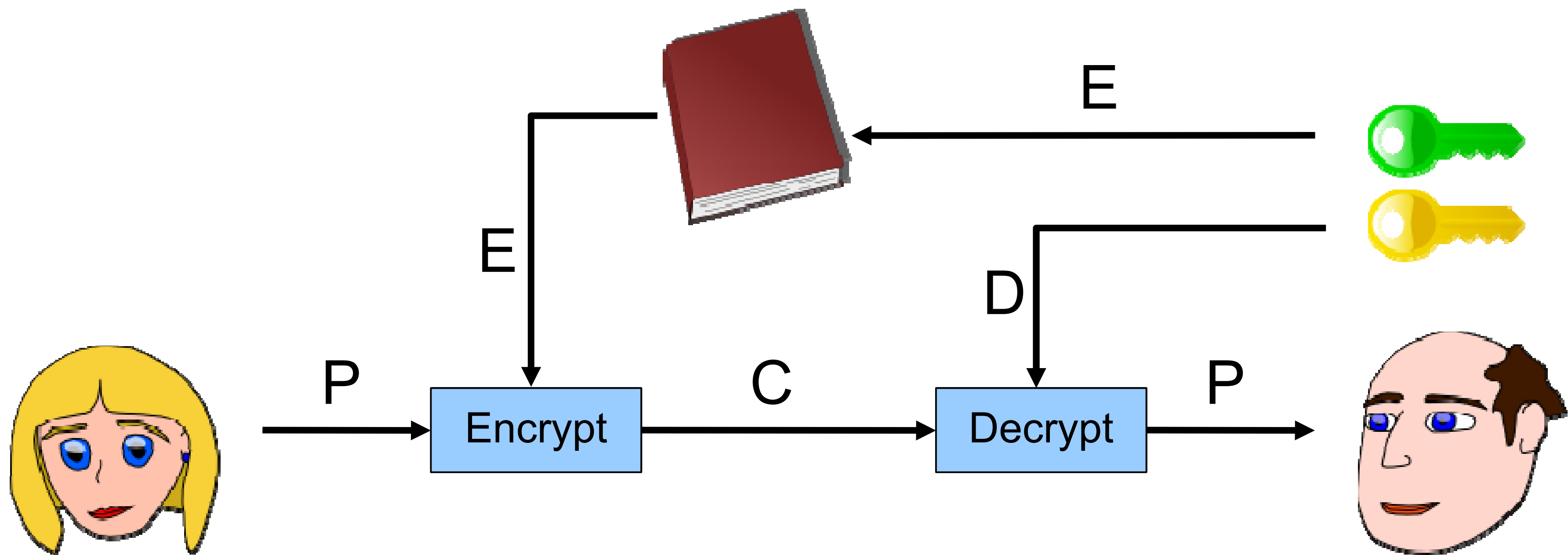
Public-key cryptography

- Invented (in public) in the 1970's
  - Allows Alice to send a secret message to Bob without any prearranged shared secret!
  - In symmetric crypto, the same key “locks” the message as “unlocks” it
  - In asymmetric (or “public-key”) crypto, there's one key for locking, and a different key for unlocking!
- Some common examples:
  - RSA, ElGamal, ECC

Public-key cryptography

- How does it work?
  - Bob gives everyone a copy of his public locking key. Alice uses it to lock (encrypt) a message, and sends the locked message to Bob
  - Bob uses his private unlocking key to unlock (decrypt) the message.
    - Eve can't unlock it; she only has the locking key.
    - Neither can Alice!
- So with this, Alice just needs to know Bob's public key in order to send him secret messages
  - These public keys can be published in a directory somewhere

Public-key cryptography



Public key sizes

- Recall that if there are no shortcuts, Eve would have to try 2^{128} things in order to read a message encrypted with a 128-bit key
- Unfortunately, all of the public-key methods we know do have shortcuts
  - Eve could read a message encrypted with a 128-bit RSA key with just 2^{33} work, which is easy!
  - If we want Eve to have to do 2^{128} work, we need to use a much longer public key

Public key sizes

Comparison of key sizes for roughly equal strength

AES	RSA
80	1024
116	2048
128	2600
160	4500
256	14000

Hybrid cryptography

- In addition to having longer keys, public-key crypto takes a long time to calculate (as compared to symmetric-key crypto)
  - Using public-key to encrypt large messages would be too slow, so we take a hybrid approach:
    - Pick a random 128-bit key for a symmetric-key cryptosystem
    - Encrypt the large message with that symmetric key (AES)
    - Encrypt the 128-bit key with a public-key cryptosystem
    - Send the symmetric-encrypted message and the public-encrypted key to Bob
  - This hybrid approach is used for almost every cryptography application on the Internet today

Is that all there is?

- It seems we've got this “sending secret messages” thing down pat. What else is there to do?
  - Even if we're safe from Eve reading our messages, there's still the matter of Mallory
  - It turns out that even if our messages are encrypted, Mallory can sometimes modify them in transit!
  - Mallory won't necessarily know what the message says, but can still change it in an undetectable way
    - e.g. bit-flipping attack on stream ciphers
  - This is counterintuitive, and often forgotten
- How do we make sure that Bob gets the same message Alice sent?

Integrity components

- How do we tell if a message has changed in transit?
- Simplest answer: use a checksum
  - For example, add up all the bytes of a message
  - The last digits of serial numbers (credit card, ISBN, etc.) are usually checksums
  - Alice computes the checksum of the message, and sticks it at the end before encrypting it to Bob. When Bob receives the message and checksum, he verifies that the checksum is correct

This doesn't work!

- With most checksum methods, Mallory can easily change the message in such a way that the checksum stays the same
- We need a “cryptographic” checksum
- It should be hard for Mallory to find a second message with the same checksum as any given one

Cryptographic hash functions

- These cryptographic checksums are called hash functions
  - Common examples: MD5, SHA-1, SHA-256
- Hash functions generally have two properties:
  - One-way:
    - Given a hash value, it's hard to find a message which hashes to that value (a “preimage”)
  - Collision-resistant:
    - It's hard to find two messages which hash to the same value (a “collision”)

What is “hard”?

- For SHA-1, for example, it takes 2^{160} work to find a preimage, and 2^{80} work to find a collision
  - Well, that's what we thought until last year
  - It turns out finding collisions in SHA-1 may be easier than we thought

- The difference is due to the well-known birthday paradox

Cryptographic hash functions

- You can't just send an unencrypted message and its hash to get integrity assurance
  - Even if you don't care about secrecy!
- Mallory can change the message and just compute the new hash value himself

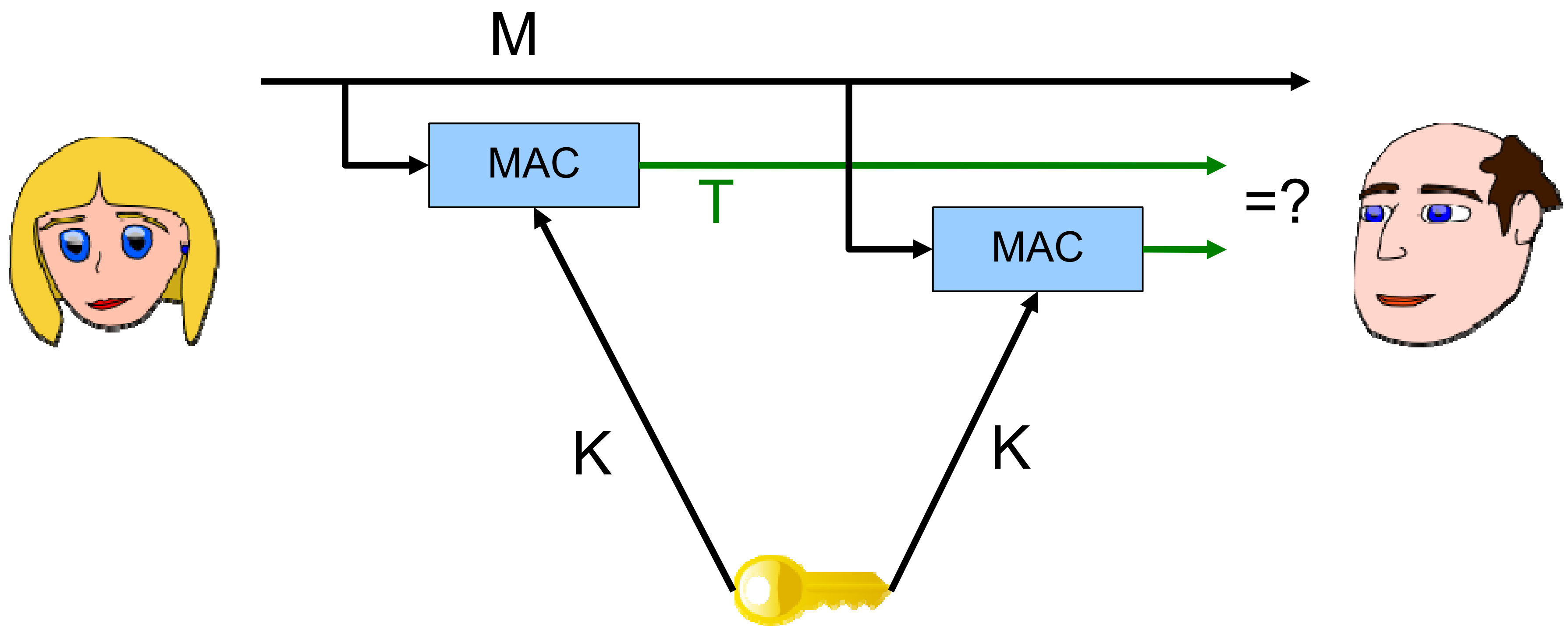
Cryptographic hash functions

- Hash functions are useful only when there is a secure way of sending the hash value
  - For example, Bob can publish a hash of his public key on his business card
  - Putting the whole key on there would be too big
  - But Alice can download Bob's key from the Internet, hash it herself, and verify that the hash matches the one on Bob's card
- What if there's no external channel to be had?
  - For example, you're using the Internet to communicate

Message authentication codes

- We do the same trick as for encryption: have a large class of hash functions, and use a shared secret to pick the right one
- Only those who know the secret can generate, or even check, the hash values
- These “keyed hashes” are usually called Message Authentication Codes, or MACs
- Common examples:
  - SHA-1-HMAC, SHA-256-HMAC, CBC-MAC

Message authentication codes



Message authentication codes

- Suppose Alice and Bob share a MAC key, and Bob receives a message with a correct MAC using that key
  - Then Bob can be assured that Alice is the one who sent that message, and that it hasn't been modified since she sent it!
  - This is like a “signature” on the message
  - But it's not quite the same!
  - Bob can't show that signature to Carol to prove Alice sent the message

Message authentication codes

- Alice can just claim that Bob made up the message, and calculated the MAC himself
- This is called repudiation; and we sometimes want to avoid it
- Some interactions should be repudiable
  - Private conversations
- Some interactions should be non-repudiable
  - Electronic commerce

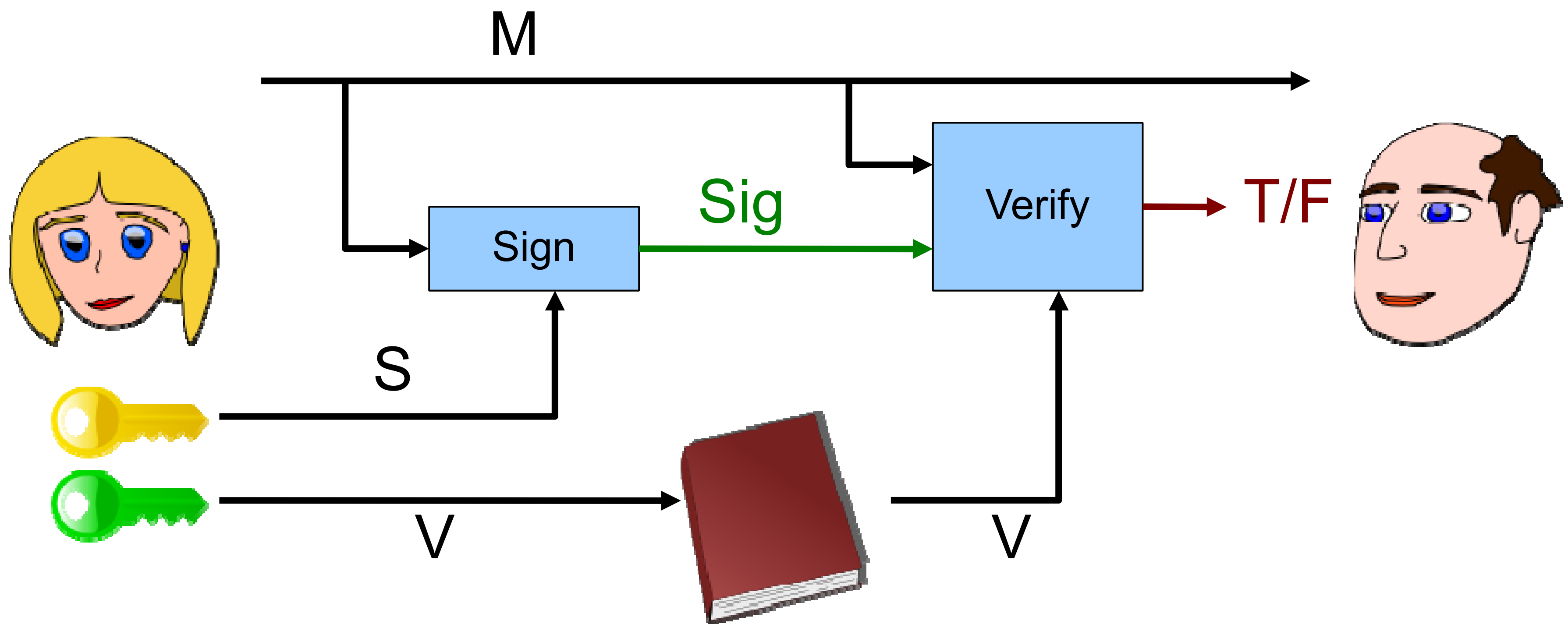
Digital signatures

- For non-repudiation, what we want is a true digital signature, with the following properties:
- If Bob receives a message with Alice's digital signature on it, then:
  - Alice, and not an impersonator, sent the message,
  - the message has not been altered since it was sent, and
  - Bob can prove these facts to a third party.
- How do we arrange this?
  - Use similar techniques to public-key cryptography

Making digital signatures

- Remember public-key crypto:
  - Separate keys for locking and unlocking
  - Give everyone a copy of the locking key
  - Keep the unlocking key secret
- To make a digital signature:
  - Alice signs the message with her private signature key
- To verify Alice's signature:
  - Bob verifies the message with his copy of Alice's public verification key
  - If it verifies correctly, the signature is valid

Making digital signatures



Hybrid signatures

- Just like public-key crypto, signing large messages is slow
- We can also hybridize signatures to make them faster:
  - Alice sends the (unsigned) message, and also a signature on a hash of the message
  - The hash is much smaller than the message, and so faster to sign and verify
- Remember that authenticity and secrecy are separate; if you want both, you need to do both

Combining public-key encryption and digital signatures

- Alice has two different key pairs: an (encryption, decryption) key pair and a (signature, verification) key pair
  - So does Bob
- Alice uses Bob's encryption key to encrypt a message destined for Bob
- She uses her signature key to sign the ciphertext
- Bob uses Alice's verification key to check the signature
- He uses his decryption key to decrypt the ciphertext
- Similarly for reverse direction

Relationship between key pairs

- Alice's (signature, verification) key pair is long-lived, whereas her (encryption, decryption) key pair is short-lived
  - Gives perfect forward secrecy
- When creating a new (encryption, decryption) key pair, Alice uses her signing key to sign her new encryption key and Bob uses Alice's verification key to verify the signature on this new key
- If Alice's communication with Bob is interactive, she can use secret-key encryption and does not need an (encryption, decryption) key pair at all (see TLS/SSH)

The Key Management Problem

- One of the hardest problems of public-key cryptography is that of key management
- How can Bob find Alice's verification key?
  - He can know it personally (manual keying)
    - SSH does this
  - He can trust a friend to tell him (web of trust)
    - PGP does this
  - He can trust some third party to tell him (CA's)
    - SSL does this

Certificate authorities

- A CA is a trusted third party who keeps a directory of people's (and organizations') verification keys
- Alice generates a (signature, verification) key pair, and sends the verification key, as well as a bunch of personal information, both signed with Alice's signature key, to the CA
- The CA ensures that the personal information and Alice's signature are correct
- The CA generates a certificate consisting of Alice's personal information, as well as her verification key. The entire certificate is signed with the CA's signature key

Certificate authorities

- Everyone is assumed to have a copy of the CA's verification key, so they can verify the signature on the certificate
- There can be multiple levels of certificate authorities; level n CA issues certificates for level n+1 Cas
  - Public-key infrastructure (PKI)
- Need to have only verification key of root CA to verify certificate chain

TLS / SSL

- In the mid-1990s, Netscape invented a protocol called Secure Sockets Later (SSL) meant for protecting HTTP (web) connections
  - The protocol, however, was general, and could be used to protect any TCP-based connection
  - HTTP + SSL = HTTPS
- SSL went through a few revisions, and was eventually standardized into the protocol known as TLS (Transport Layer Security, imaginatively enough)

TLS at a high level

- Client connects to server, indicates it wants to speak TLS, and which ciphersuites it knows
- Server sends its certificate to client, which contains:
  - Its host name
  - Its verification key
  - Some other administrative information
  - A signature from a Certificate Authority (CA)
- Server also chooses which ciphersuite to use

TLS at a high level (cont.)

- Client validates server's certificate
  - Is its signature from a CA whose public key is embedded in the client (e.g., browser, app)?
  - Does the host name in the certificate match the host name of the web site that client wants to access?
- Client and server run a key agreement protocol to establish keys for symmetric encryption and MAC algorithms from the chosen ciphersuite
  - Server signs its protocol messages with its signature key
- Communication now proceeds using chosen symmetric encryption and MAC algorithms

Security properties provided by TLS

- Server authentication
- Message integrity
- Message confidentiality
- Client authentication (optional)