CS 458 / 658
Computer Security and Privacy

Module 5
Internet Application Security and Privacy

Winter 2012

Module outline

1. Basics of cryptography
2. Symmetric-key encryption
3. Public-key encryption
4. Integrity
5. Authentication

Module outline

6. Security controls using cryptography
7. Link-layer security
8. Network-layer security
9. Transport-layer security and privacy
10. Application-layer security and privacy

Module outline

1. Basics of cryptography
2. Symmetric-key encryption
3. Public-key encryption
4. Integrity
5. 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 that studies these both
- The point of cryptography is to send secure messages over an insecure medium (like the Internet)

The scope of these lectures

- The goal of the cryptography unit in this course is to show you what cryptographic tools exist, and information about using these tools in a secure manner
- We won't be showing you details of how the tools work
  - For that, see CO 487, or chapter 12 of the text

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
  - Confidentiality 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

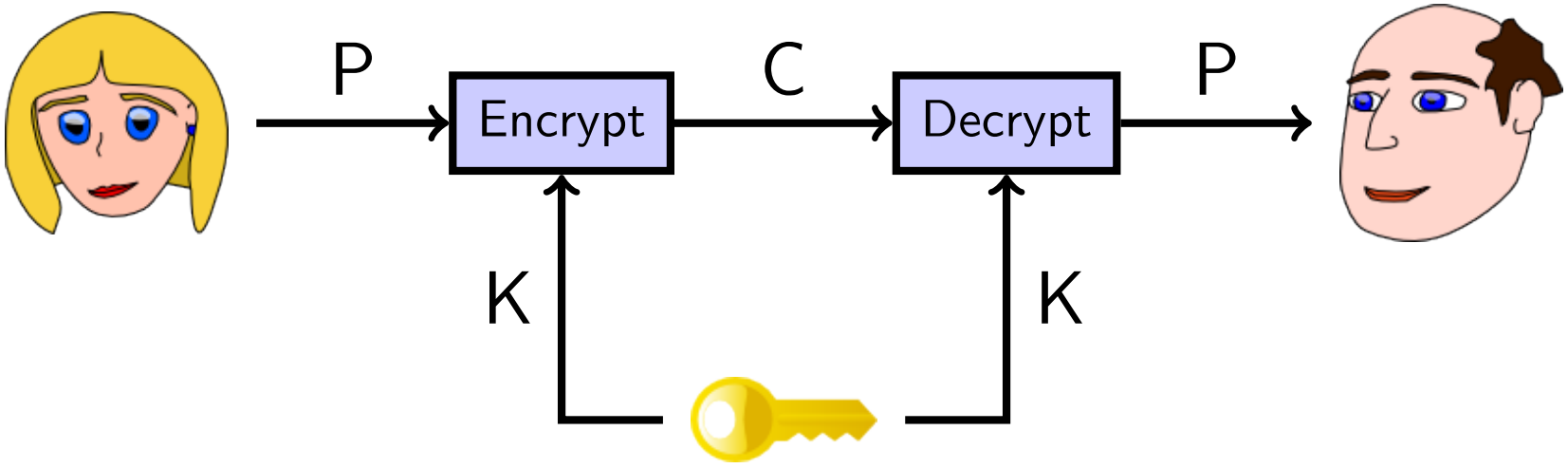
- What information do we assume the attacker (Eve) has when she's trying to break our system?
- She may:
  - Know the algorithm (the public class of encryption methods)
  - Know some part of the plaintext
  - Know a number (maybe a large number) of corresponding plaintext/ciphertext pairs
  - Have access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

- And we still want to prevent Eve from learning the key!

Module outline

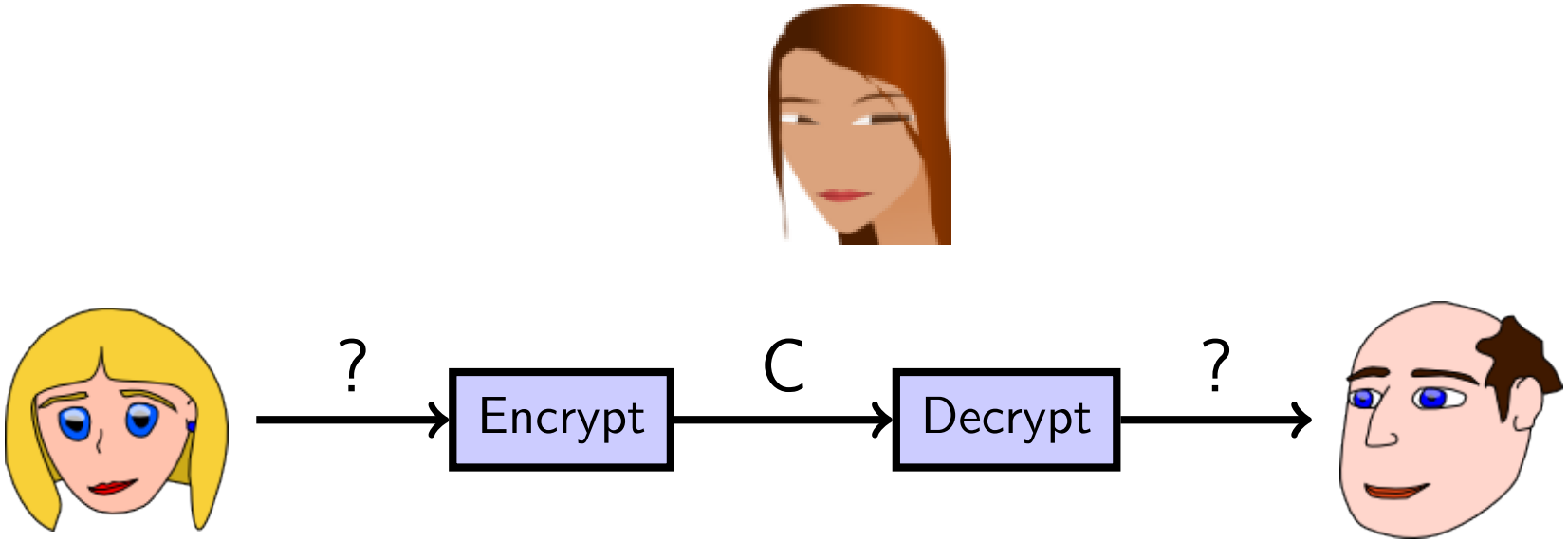
Symmetric encryption

- 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

- Q: Why does "try every key" not work here?
- 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!
- Q: How do you share that much secret key?
- Used in the Washington / Moscow hotline for many years

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

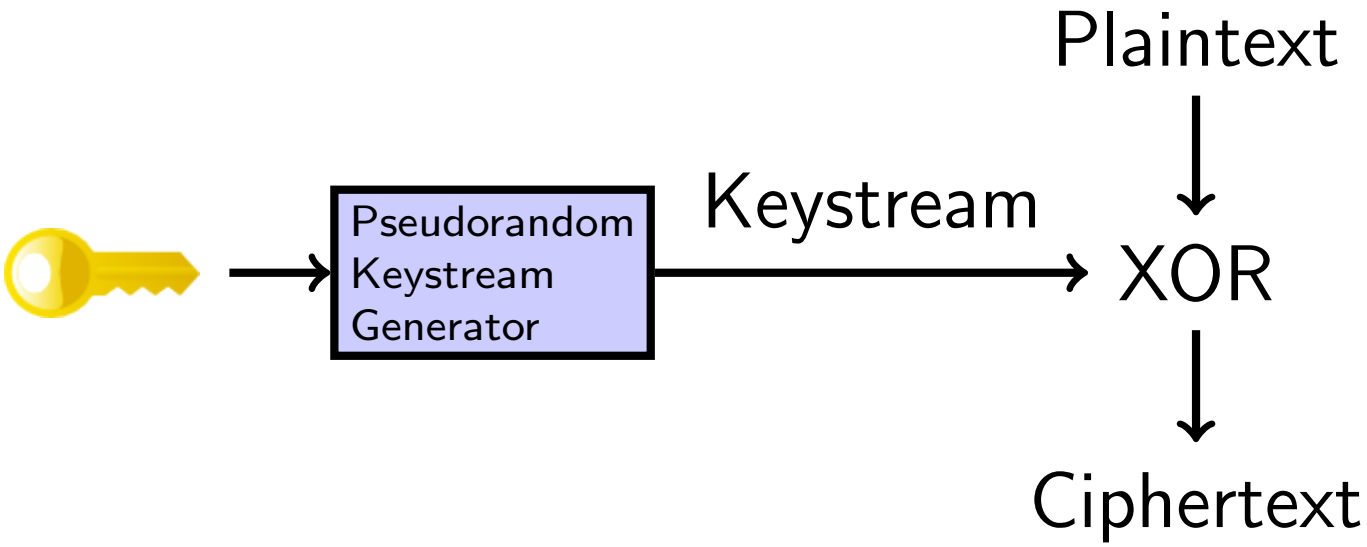
- Don't break the crypto at all!
- There are always weaker parts of the system to attack
  - Remember the Principle of Easiest Penetration
- The point of cryptography is to make sure the information transfer is not the weakest link

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

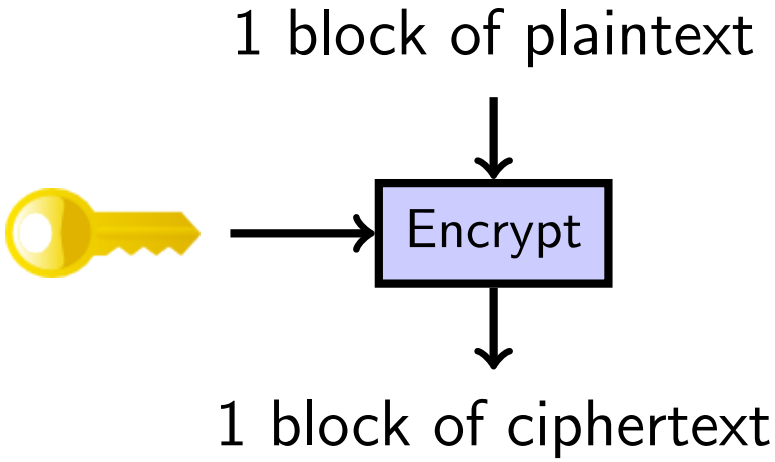
- Stream ciphers can be very fast
  - This is useful if you need to send a lot of data securely
- But they can be tricky to use correctly!
  - What happens if you use the same key to encrypt two different messages?
  - How would you solve this problem without requiring a new shared secret key for each message? Where have we seen this technique before?
- 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

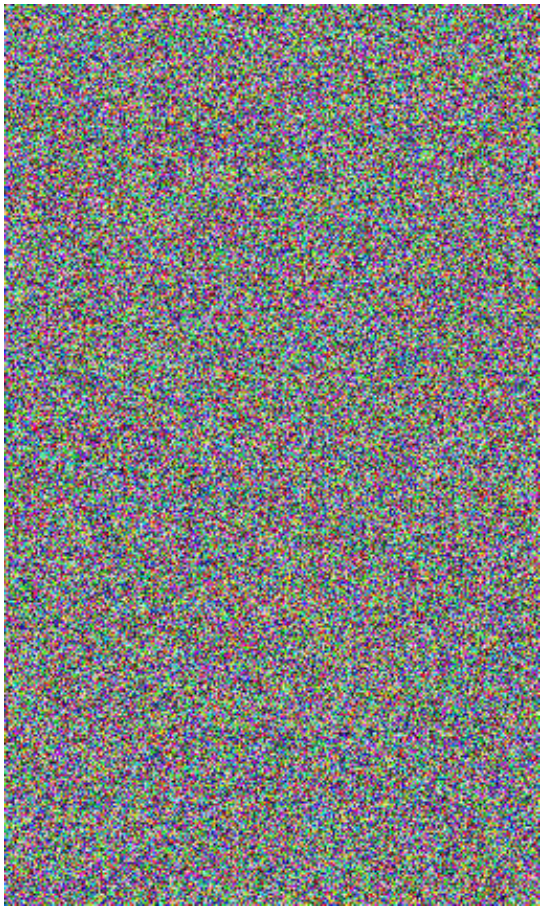
- But if there are repeated blocks in the plaintext, you'll see the same repeating patterns in the ciphertext:

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...

Module outline

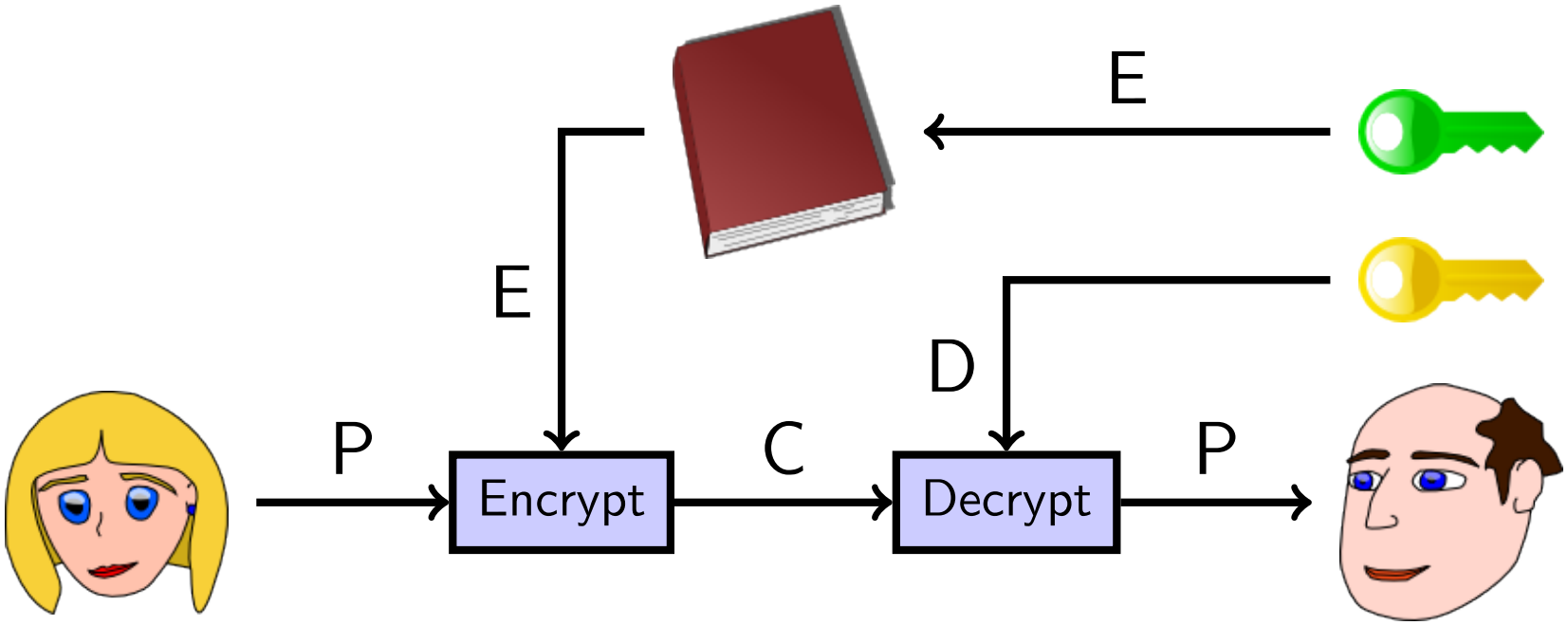
Public-key cryptography

- 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
    - The textbook even gets this wrong!
- How do we make sure that Bob gets the same message Alice sent?

Module outline

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 that hashes to that value (a "preimage")
  - Collision-resistant:
    - It's hard to find two messages that 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 a few years ago
  - 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 confidentiality!
- Mallory can change the message and just compute the new hash value himself

Cryptographic hash functions

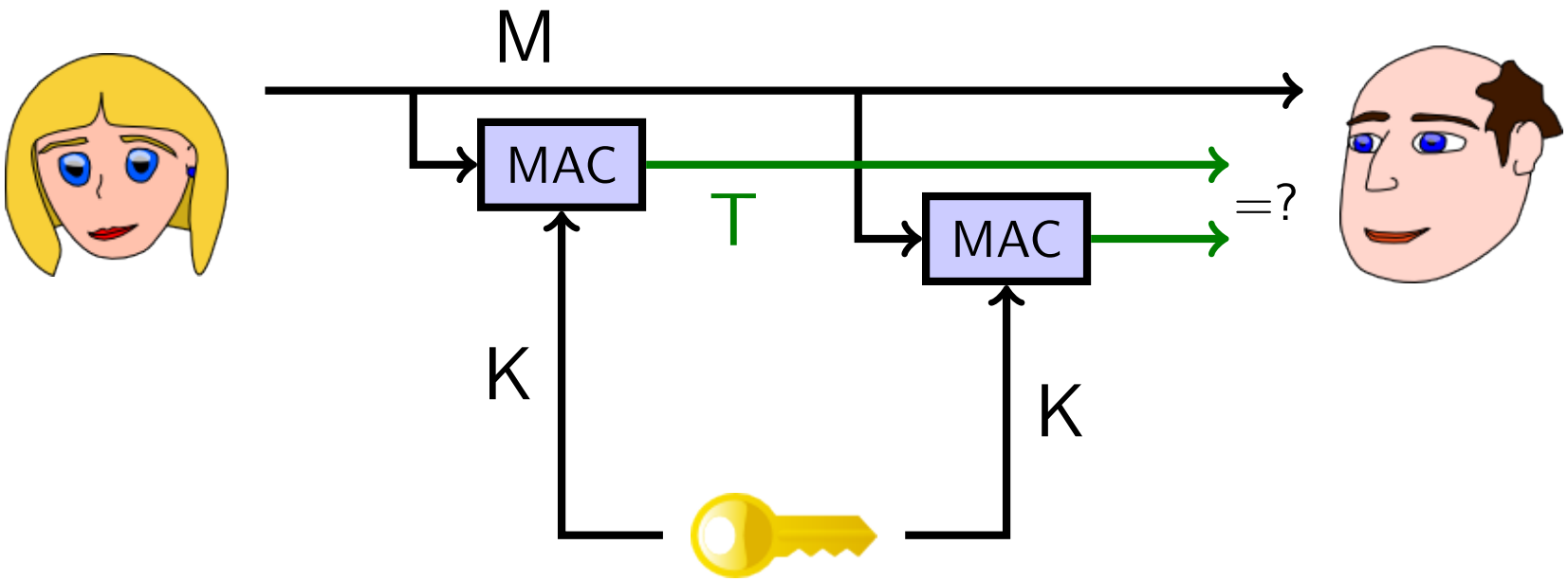
- Hash functions are useful only when there is a secure way of sending the hash value
  - For example, Bob can publish a hash of his public key on his business card
  - Putting the whole key on there would be too big
  - But Alice can download Bob's key from the Internet, hash it herself, and verify 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

Module outline

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

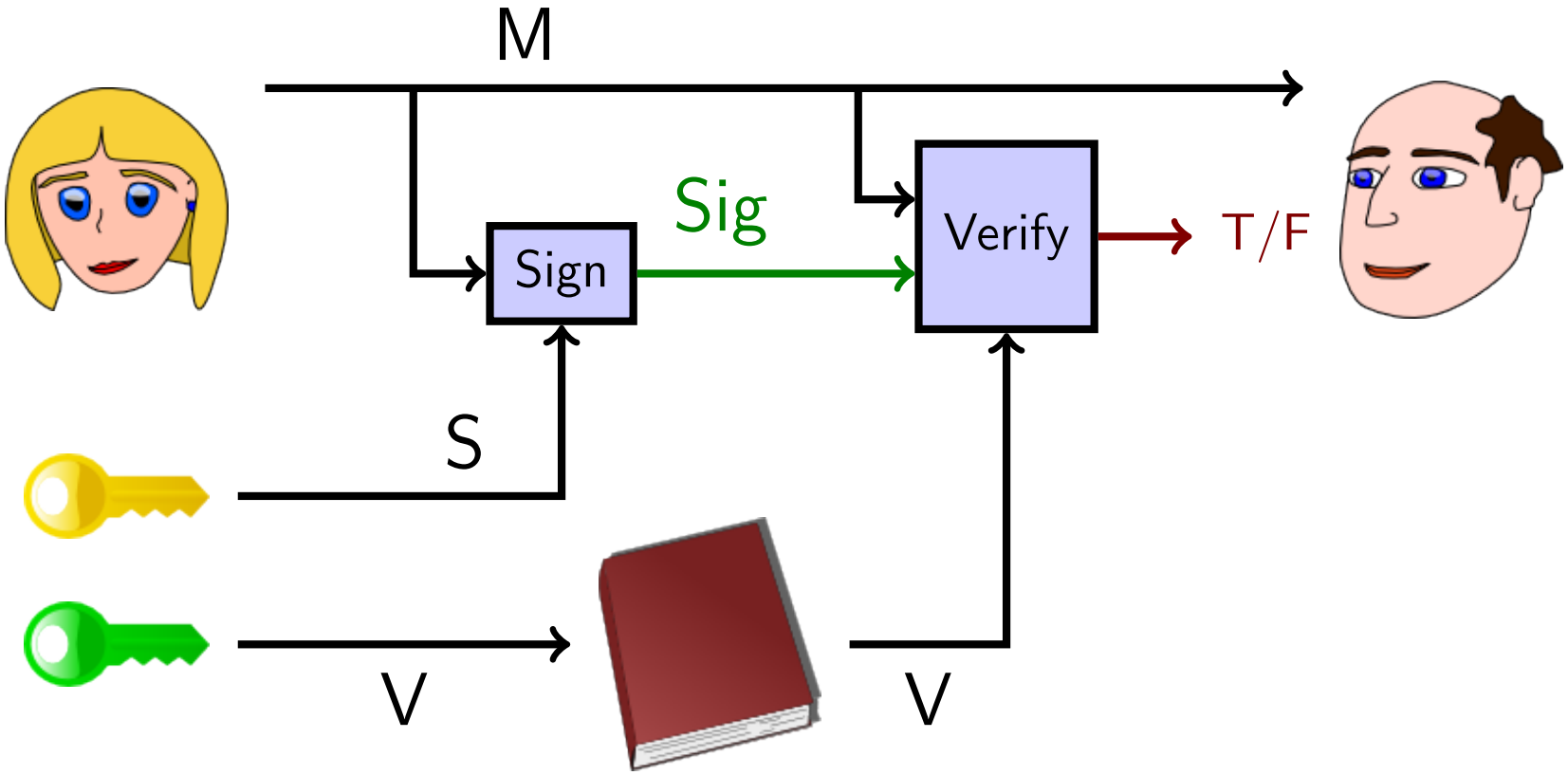
Digital signatures

- 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



- Note that (Encryption, Decryption) key pairs for public-key encryption are not the same thing as (Signature, Verification) key pairs for 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 confidentiality are separate; if you want both, you need to do both

The Key Management Problem

- One of the hardest problems of public-key cryptography is that of key management
- If Alice wants to send an encrypted message to Bob, how does she find out Bob's public key?
  - She can know it personally (manual keying)
    - SSH does this
  - She can trust a friend to tell her (web of trust)
    - PGP does this
  - She can trust some third party to tell her (CA's)
    - TLS / SSL do this

Certificate authorities

- A CA is a trusted third party who keeps a directory of people's (and organizations') public keys
- Bob generates a public and private key pair, and sends the public key, as well as a bunch of personal information, both signed with Bob's private key, to the CA
- The CA ensures that the personal information and Bob's signature are correct
- The CA generates a certificate consisting of Bob's personal information, as well as his public 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 public key of root CA to verify certificate chain

Putting it all together

- We have all these blocks; now what?
- Put them together into protocols
- This is HARD. Just because your pieces all work, doesn't mean what you build out of them will; you have to use the pieces correctly
- Common mistakes include:
  - Using the same stream cipher key for two messages
  - Assuming encryption also provides integrity
  - Falling for replay attacks or reaction attacks
  - LOTS more!

Module outline

6. Security controls using cryptography
7. Link-layer security
8. Network-layer security
9. Transport-layer security and privacy
10. Application-layer security and privacy

Security controls using cryptography

- In what situations might it be appropriate to use cryptography as a security control?
- Remember that there needs to be some separation, since any secrets (like the key) need to be available to the legitimate users but not the adversaries
- In some situations, this may make symmetric-key crypto problematic
- If your web browser can decrypt its file containing your saved passwords, then an adversary who can read your web browser probably can, too
- How is this solved in practice?

Program and OS security

- Using symmetric-key crypto can be problematic for the above reason
  - But public-key is OK, if the local machine only needs access to the public part of the key
  - So only encryption and signature verification; no decryption or signing
  - Common example: programs allow upgrades only if digitally signed
  - OS may allow execution of programs only if signed

Encrypted code

- There is research into processors that will only execute encrypted code
- The processor will decrypt instructions before executing them
- The decryption key is processor-dependent
- Malware won't be able to spread without knowing a processor's key
- Downsides?

Encrypted data

- Harddrive encryption protects data when laptop gets lost/stolen
- It often does not protect data against other users who legitimately use laptop
- Or somebody installing malware on laptop
- Or somebody (maybe physically) extracting the decryption key from the laptop's memory

OS authentication

- Authentication mechanisms often use cryptography
  - E.g., salted hashes (see Module 3)
- Unfortunately, people are bad at doing cryptography in their heads, so some mechanisms require hardware token

Photo from http://itc.ua/

Network security and privacy

- The primary use for cryptography
  - "Separating the security of the medium from the security of the message"
- Entities you can only communicate with over a network are inherently less trustworthy
  - They may not be who they claim to be

Network security and privacy

- Network cryptography is used at every layer of the network stack for both security and privacy applications:
  - Link
    - WEP, WPA, WPA2
  - Network
    - VPN, IPSec
  - Transport
    - TLS / SSL, Tor
  - Application
    - ssh, Mixminion, PGP, OTR

Module outline

Link-layer security controls

- Intended to protect local area networks
- Widespread example: WEP (Wired Equivalent Privacy)
- WEP was intended to enforce three security goals:
  - Confidentiality
    - Prevent an adversary from learning the contents of your wireless traffic
  - Access Control
    - Prevent an adversary from using your wireless infrastructure
  - Data Integrity
- Unfortunately, none of these is actually enforced!

WEP description

- Brief description:
- The sender and receiver share a secret k
  - The secret k is either 40 or 104 bits long
- In order to transmit a message M:
  - Compute a checksum $c(M)$
    - this does not depend on k
  - Pick an IV (a random number) v and generate a keystream $RC4(v, k)$
  - XOR $\langle M, c(M) \rangle$ with the keystream to get the ciphertext
  - Transmit v and the ciphertext over the radio link

WEP description

- Upon receipt of v and the ciphertext:
  - Use the received v and the shared k to generate the keystream $RC4(v, k)$
  - XOR the ciphertext with $RC4(v, k)$ to get $\langle M', c' \rangle$
  - Check to see if $c' = c(M')$
  - If it is, accept M' as the message transmitted

- Problem number 1: v is 24 bits long
  - Why is this a problem?

WEP data integrity

- Problem 2: the checksum used in WEP is CRC-32
  - Quite a poor choice; there's already a CRC in the protocol to detect random errors, and a CRC can't help you protect against malicious errors.
- The CRC has two important properties:
  - It is independent of k and v
  - It is linear: $c(M \text{ XOR } D) = c(M) \text{ XOR } c(D)$
- Why is linearity a pessimal property for your integrity mechanism to have when used in conjunction with a stream cipher?

WEP access control

- What if the adversary wants to inject a new message F onto a WEP-protected network?

- All he needs is a single plaintext/ciphertext pair
- This gives him a value of v and the corresponding keystream $RC4(v, k)$
- Then $C' = \langle F, c(F) \rangle$ XOR $RC4(v, k)$, and he transmits v, C'
- C' is in fact a correct encryption of F, so the message must be accepted

WEP authentication protocol

- How did we get that single plaintext/ciphertext pair we needed just now?
  - Problem 3: It turns out the authentication protocol gives it to the adversary for free!

- This is a major disaster in the design!
- The authentication protocol is supposed to prove that a certain client knows the shared secret k
- But if I watch you prove it, I can turn around and execute the protocol myself!
  - "What's the password?"

WEP authentication protocol

- Here's the protocol:
  - The access point sends a challenge string to the client
  - The client sends back the challenge, WEP-encrypted with the shared secret k
  - The base station checks if the challenge is correctly encrypted, and if so, accepts the client

- So the adversary has just seen both the plaintext and the ciphertext of the challenge
- Problem number 4: this is enough not only to inject packets (as in the previous attack), but also to execute the authentication protocol himself!

WEP decryption

- Somewhat surprisingly, the ability to modify and inject packets also leads to ways the adversary can decrypt packets!
  - The access point knows k; it turns out the adversary can trick it into decrypting the packet for him and telling him the result.
- Note that none of the attacks so far:
  - Used the fact that the stream cipher was RC4 specifically
  - Recovered k

Recovering a WEP key

- Since 2002, there have been a series of analyses of RC4 in particular
  - Problem number 5: it turns out that when RC4 is used with similar keys, the output keystream has a subtle weakness
    - And this is how WEP uses RC4!
- These observations have led to programs that can recover either a 104-bit or 40-bit WEP key in under 60 seconds, most of the time
  - See the optional reading for more information on this

Replacing WEP

- Wi-fi Protected Access (WPA) was rolled out as a short-term patch to WEP while formal standards for a replacement protocol (IEEE 802.11i, later called WPA2) were being developed
- WPA:
  - Replaces CRC-32 with a real MAC (here called a MIC to avoid confusion with a Media Access Control address)
  - IV is 48 bits
  - Key is changed frequently (TKIP)
  - Ability to use 802.1x authentication server
    - But maintains less-secure PSK (Pre-Shared Key) mode for home users
  - Able to run on most older WEP hardware

Replacing WEP

- The 802.11i standard was finalized in 2004, and the result (called WPA2) has been required for products calling themselves "Wi-fi" since 2006
- WPA2:
  - Replaces the RC4 and MIC algorithms in WPA with the CCMP algorithm, which uses AES
  - Considered strong, except in PSK mode
    - Dictionary attacks still possible

Module outline

Network-layer security

- Suppose every link in our network had strong link-layer security
- Why would this not be enough?

- We need security across networks
  - Ideally, end-to-end

- At the network layer, this is usually accomplished with a Virtual Private Network (VPN)

Virtual Private Networks

- Connect two (or more) networks that are physically isolated, and make them appear to be a single network
  - Alternately: connect a single remote host (often a laptop) to one network
- Goal: adversary between the networks should not be able to read or modify the traffic flowing across the VPN
  - But DoS and some traffic analysis still usually possible

Setting up a VPN

- One host on each side is the VPN gateway
  - Could be the firewall itself, or could be in DMZ
  - In the laptop scenario, it will of course be the laptop itself on its side
- Traffic destined for the "other side" is sent to the local VPN gateway
- The local VPN gateway uses cryptography (encryption and integrity techniques) to send the traffic to the remote VPN gateway
  - Often by tunnelling
- The remote gateway decrypts the messages and sends them on to their appropriate destinations

Tunnelling

- Tunnelling is the sending of messages of one protocol inside (that is, as the payload of) messages of another protocol, out of their usual protocol nesting sequence

- So TCP-over-IP is not tunnelling, since you're supposed to send TCP (a transport protocol) over IP (a network protocol; one layer down in the stack)

- But IP-over-TCP is tunnelling (going up the stack instead of down), as are IP-over-IP (same place in the stack), and PPP (a link layer protocol; bottom of the stack) over DNS (an application layer protocol; top of the stack)

IPSec

- One standard way to set up a VPN is by using IPSec
- Many corporate VPNs use this (open) protocol
- Two modes:
  - Transport mode
    - Useful for connecting a single laptop to a home network
    - Only the contents of the original IP packet are encrypted and authenticated
  - Tunnel mode
    - Useful for connecting two networks
    - The contents and the header of the original IP packet are encrypted and authenticated; result is placed inside a new IP packet destined for the remote VPN gateway

Other styles of VPNs

- In addition to IPSec, there are a number of other standard ways to set up a VPN
- Microsoft's PPTP was an older protocol
  - It had about as many design flaws as WEP
  - Most users now migrating to IPSec
- VPNs based on ssh
  - Tunnel PPP over ssh
    - That is, IP-over-PPP-over-ssh-over-TCP-over-IP
    - Some efficiency concern, but extremely easy to set up on a standard Unix/Linux box
  - OpenSSH v4 supports IP-over-SSH tunnelling directly

Module outline

Transport-layer security and privacy

- Network-layer security mechanisms arrange to send individual IP packets securely from one network to another

- Transport-layer security mechanisms transform arbitrary TCP connections to add security
  - And similarly for "privacy" instead of "security"

- The main transport-layer security mechanism:
  - TLS (formerly known as SSL)
- The main transport-layer privacy mechanism:
  - Tor

TLS / SSL

- In the mid-1990s, Netscape invented a protocol called Secure Sockets Layer (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
- Historical note: there was a competing protocol called S-HTTP. But Netscape and Microsoft both chose HTTPS, so that's the protocol everyone else followed
- 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 public 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 (i.e., browser)?
  - Does the host name in the certificate match the host name of the web site that client wants to download?
- Client sends symmetric encryption key K, encrypted with server's public key
- Server decrypts with its private key and gets K
- Communication now proceeds using K and the chosen ciphersuite

Security properties provided by TLS

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

The success of TLS

- Though designed as a security mechanism, TLS (including SSL) has become the most successful Privacy Enhancing Technology (PET) ever

- Why?
  - It comes with your browser
    - Which encouraged web server operators to bother paying $$ for their certificates
  - It just works, without you having to configure anything
  - Most of the time, it even protects the privacy of your communications
    - Increasingly important due to the success of WiFi

Privacy Enhancing Technologies

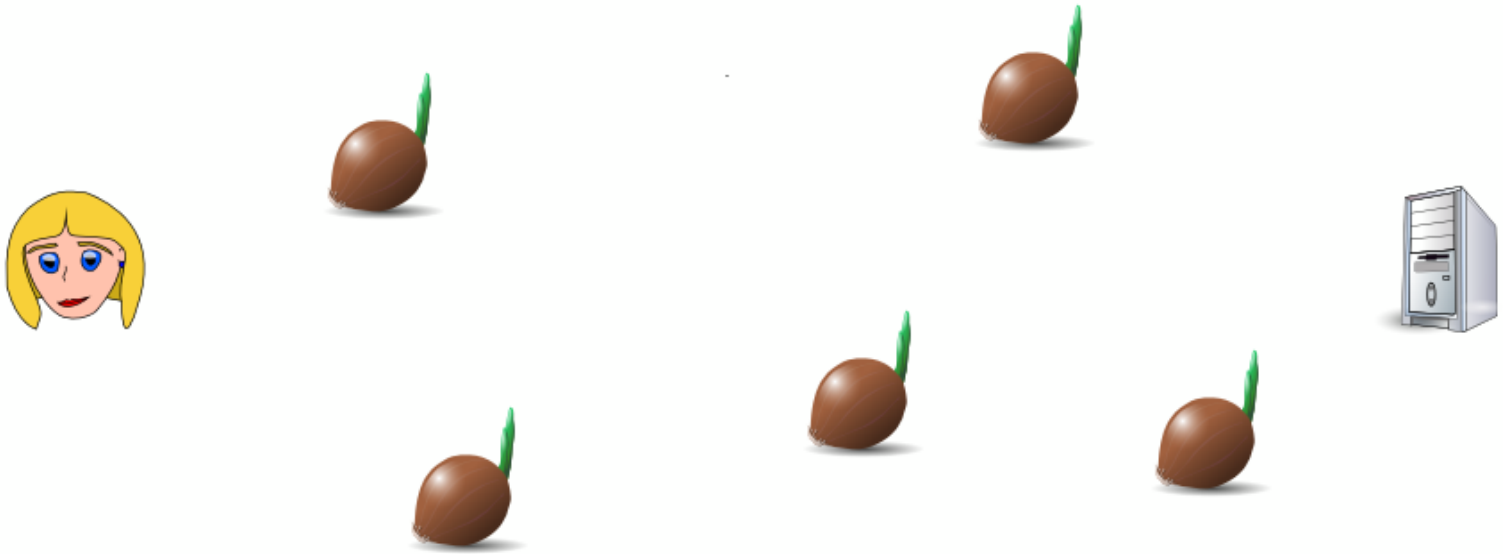
- So far, we've only used encryption to protect the contents of messages
- But there are other things we might want to protect as well!
- We may want to protect the metadata
  - Who is sending the message to whom?
  - If you're seen sending encrypted messages to Human Rights Watch, bad things may happen
- We may want to hide the existence of the message
  - If you're seen sending encrypted messages at all, bad things may happen

Tor

- Tor is another successful privacy enhancing technology that works at the transport layer
  - Hundreds of thousands of users
- Normally, any TCP connection you make on the Internet automatically reveals your IP address
  - Why?
- Tor allows you to make TCP connections without revealing your IP address
- It's most commonly used for HTTP (web) connections

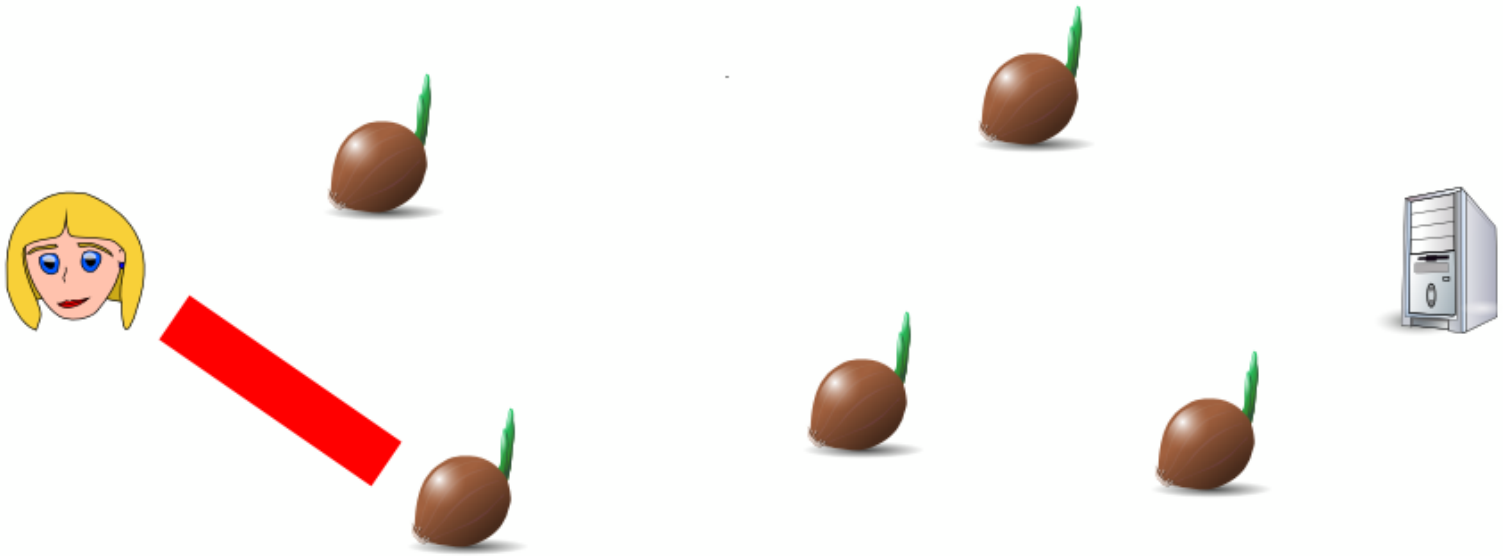
How Tor works

- Scattered around the Internet are about 1000 Tor nodes, also called Onion Routers
- Alice wants to connect to a web server without revealing her IP address



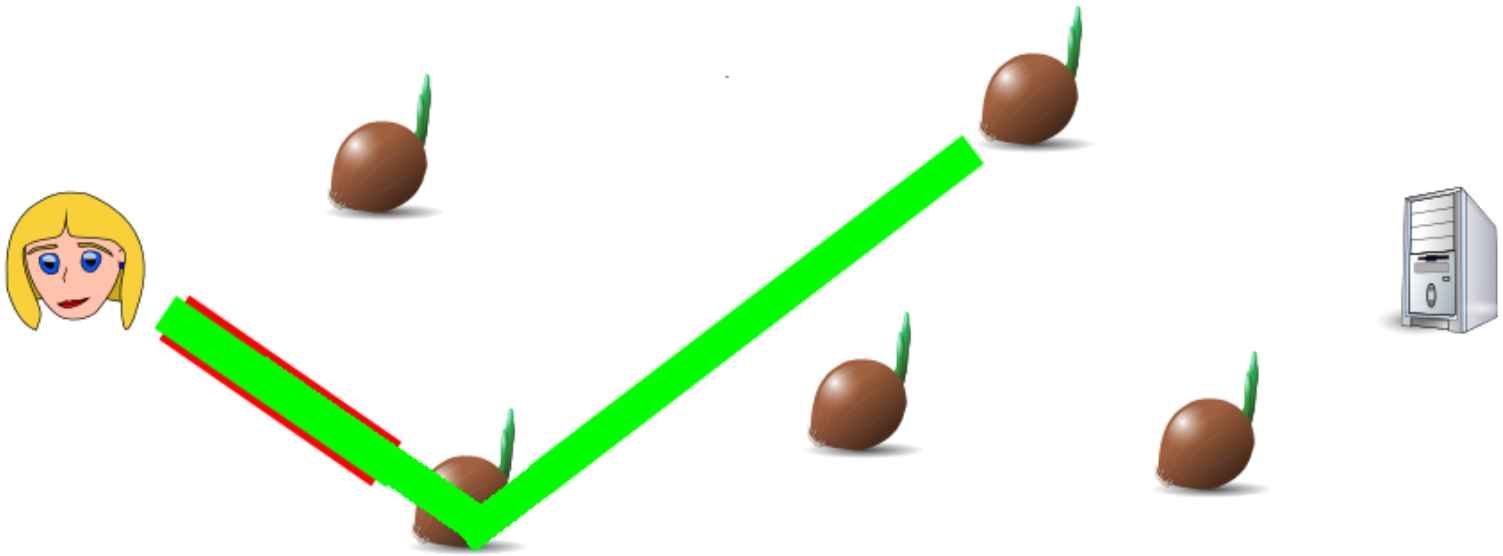
How Tor works

- Alice picks one of the Tor nodes (n1) and uses public-key cryptography to establish an encrypted communication channel to it (much like TLS)



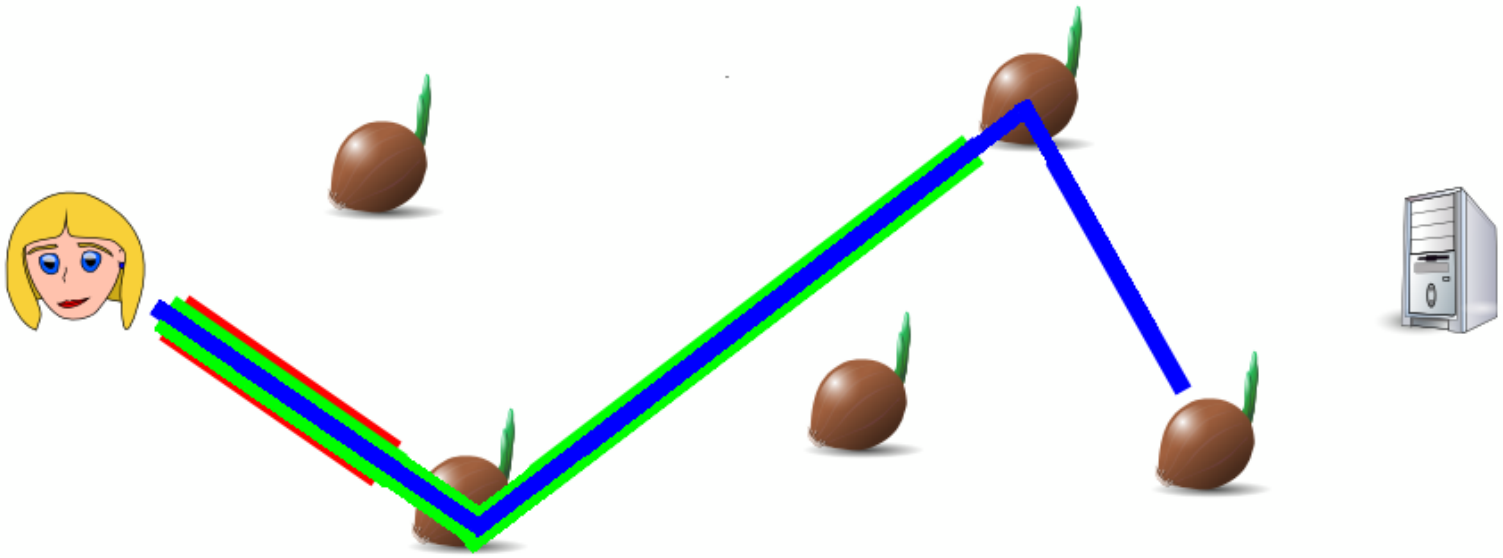
How Tor works

- Alice tells n1 to contact a second node (n2), and establishes a new encrypted communication channel to n2, tunnelled within the previous one to n1



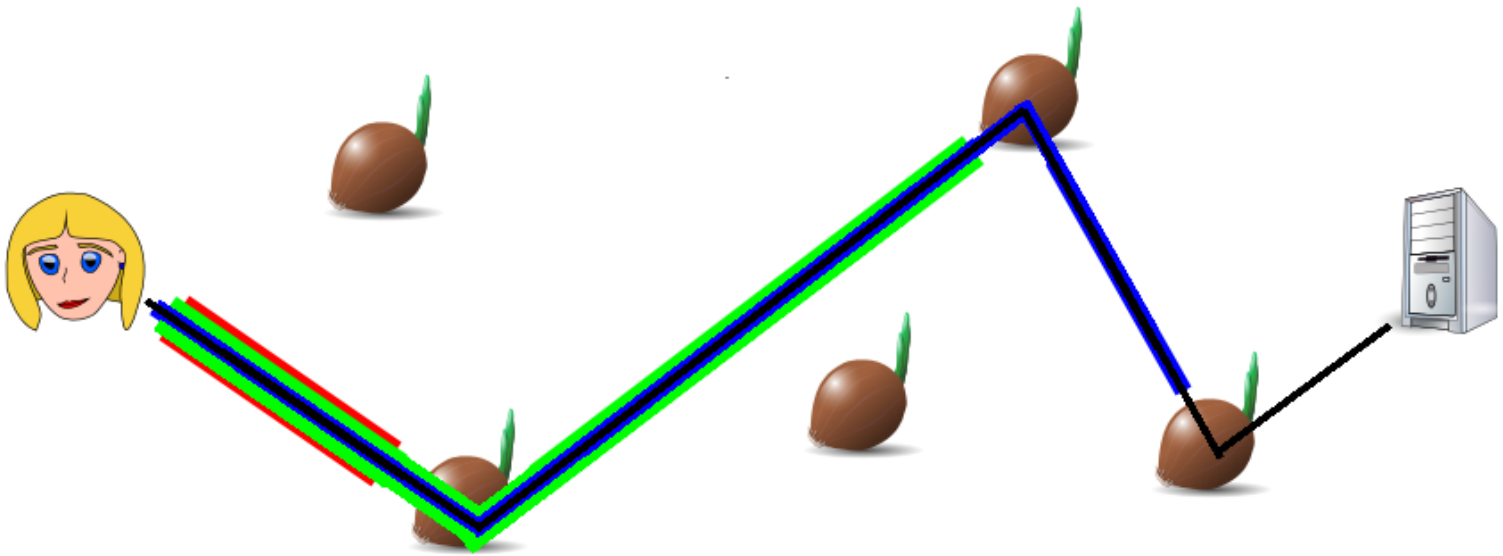
How Tor works

- Alice tells n2 to contact a third node (n3), and establishes a new encrypted communication channel to n3, tunnelled within the previous one to n2



How Tor works

- And so on, for as many steps as she likes (usually 3)
- Alice tells the last node (within the layers of tunnels) to connect to the website



Sending messages with Tor

- Alice now shares three symmetric keys:
  - $K1$ with n1
  - $K2$ with n2
  - $K3$ with n3
- When Alice wants to send a message M, she actually sends $E_{K1}(E_{K2}(E_{K3}(M)))$
- Node n1 uses $K1$ to decrypt the outer layer, and passes the result $E_{K2}(E_{K3}(M))$ to n2
- Node n2 uses $K2$ to decrypt the next layer, and passes the result $E_{K3}(M)$ to n3
- Node n3 uses $K3$ to decrypt the final layer, and sends M to the website

Replies in Tor

- When the website replies with message R, it will send it to node n3
  - Why?
- Node n3 will encrypt R with $K3$ and send $E_{K3}(R)$ to n2
- Node n2 will encrypt that with $K2$ and send $E_{K2}(E_{K3}(R))$ to n1
- Node n1 will encrypt that with $K1$ and send $E_{K1}(E_{K2}(E_{K3}(R)))$ to Alice
- Alice will use $K1$, $K2$, and $K3$ to decrypt the layers of the reply and recover R

Who knows what?

- Notice that node n1 knows that Alice is using Tor, and that her next node is n2, but does not know which website Alice is visiting
- Node n3 knows some Tor user (with previous node n2) is using a particular website, but doesn't know who
- The website itself only knows that it got a connection from Tor node n3

- Note: the connection between n3 and the website is not encrypted! If you want encryption as well as the benefits of Tor, you should use end-to-end encryption in addition
  - Like HTTPS

Anonymity vs. pseudonymity

- Tor provides for anonymity in TCP connections over the Internet, both unlinkably (long-term) and linkably (short-term)

- What does this mean?
  - There's no long-term identifier for a Tor user
  - If a web server gets a connection from Tor today, and another one tomorrow, it won't be able to tell whether those are from the same person
  - But two connections in quick succession from the same Tor node are more likely to in fact be from the same person

The Nymity Slider

- We can place transactions (both online and offline) on a continuum according to the level of nymity they represent:
  - Verinymity
    - Government ID, SIN, credit card #, address
  - Persistent pseudonymity
    - Noms de plume, many blogs
  - Linkable anonymity
    - Prepaid phone cards, loyalty cards
  - Unlinkable anonymity
    - Cash payments, Tor

The Nymity Slider

- If you build a system at a certain level of nymity, it's easy to modify it to have a higher level of nymity, but hard to modify it to have a lower level
- For example:
  - It's easy to add a loyalty card to a cash payment, or a credit card to a loyalty card
  - It's hard to remove identity information if you're paying by credit card
- The lesson: design systems with a low level of nymity fundamentally; adding more is easy

Module outline

Application-layer security and privacy

- TLS can provide for encryption at the TCP socket level
  - "End-to-end" in the sense of a network connection
  - Is this good enough? Consider SMTPS (SMTP/email over TLS)
- Many applications would like true end-to-end security
  - Human-to-human would be best, but those last 50 cm are really hard!
  - We usually content ourselves with desktop-to-desktop
- We'll look at three particular applications:
  - Remote login, email, instant messaging

Secure remote login (ssh)

- You're already familiar with this tool for securely logging in to a remote machine
- Usual usage (simplified):
  - Client connects to server
  - Server sends its public key
    - The client should verify that this is the correct key
  - Client picks a random session key, encrypts it with server's public key, sends to server
    - All communication from here on in is encrypted and MACd with the session key
  - Client authenticates to server
  - Server accepts authentication, login proceeds (under encryption and MAC)

Authentication with ssh

- There are two main ways to authenticate with ssh:
  - Send a password over the encrypted channel
    - The server needs to know (a hash of) your password
  - Sign a challenge with your private signature key
    - The server needs to know your public key
- Which is better? Why?

Anonymity for email: remailers

- Tor allows you to anonymously communicate over the Internet in real time
  - What about (non-interactive) email?
  - This is actually an easier problem, and was implemented much earlier than Tor
- Anonymous remailers allow you to send email without revealing your own email address
  - Of course, it's hard to have a conversation that way
  - Pseudonymity is useful in the context of email
    - Nymity Slider

Type 0 remailers

- In the 1990s, there were very simple ("type 0") remailing services, the best known being anon.penet.fi

- How it worked:
  - Send email to anon.penet.fi
  - It is forwarded to your intended recipient
  - Your "From" address is changed to anon43567@anon.penet.fi (but your original address is stored in a table)
  - Replies to the anon address get mapped back to your real address and delivered to you

anon.penet.fi

- This works, as long as:
  - No one's watching the net connections to or from anon.penet.fi
  - The operator of anon.penet.fi and the machine itself remain trustworthy and uncompromised
  - The mapping of anon addresses to real addresses is kept secret

- Unfortunately, a lawsuit forced Julf (the operator) to turn over parts of the list, and he shut down the whole thing, since he could no longer legally protect it

Type I remailers

- Cypherpunk (type I) remailers removed the central point of trust
- Messages are now sent through a "chain" of several remailers, with dozens to choose from
- Each step in the chain is encrypted to avoid observers following the messages through the chain; remailers also delay and reorder messages
- Support for pseudonymity is dropped: no replies!

Type II remailers

- Mixmaster (type II) remailers appeared in the late 1990s
- Constant-length messages to avoid an observer watching "that big file" travel through the network
- Protections against replay attacks
- Improved message reordering
- But! Requires a special email client to construct the message fragments
  - premail (a drop-in wrapper for sendmail) makes it easy

Nym servers

- Recovering pseudonymity: "nym servers" mapped pseudonyms to "reply blocks" that contained a nested encrypted chain of type I remailers. Attaching your message to the end of one of these reply blocks would cause it to be sent through the chain, eventually being delivered to the nym owner

- But remember that there were significant privacy issues with the type I remailer system

- Easier recipient anonymity: alt.anonymous.messages

Type III remailers

- Type II remailers were the state of the art until recently
- Mixminion (type III) remailer
  - Native (and much improved) support for pseudonymity
    - No longer reliant on type I reply blocks
  - Improved protection against replay and key compromise attacks
- But it's not very well deployed or mature
  - "You shouldn't trust Mixminion with your anonymity yet"

Pretty Good Privacy

- The first popular implementation of public-key cryptography.
- Originally made by Phil Zimmermann in 1991
  - He got in a lot of trouble for it, since cryptography was highly controlled at the time.
  - But that's a whole 'nother story. :-)
- Today, there are many (more-or-less) compatible programs
  - GNU Privacy Guard (gpg), Hushmail, etc.

Pretty Good Privacy

- What does it do?
  - Its primary use is to protect the contents of email messages
- How does it work?
  - Uses public-key cryptography to provide:
    - Encryption of email messages
    - Digital signatures on email messages

Recall

- In order to use public-key encryption and digital signatures, Alice and Bob must each have:
  - A public encryption key
  - A private decryption key
  - A private signature key
  - A public verification key

Sending a message

- To send a message to Bob, Alice will:
  - Write a message
  - Sign it with her own signature key
  - Encrypt both the message and the signature with Bob's public encryption key
- Bob receives this, and:
  - Decrypts it using his private decryption key to yield the message and the signature
  - Uses Alice's verification key to check the signature

Back to PGP

- PGP's main functions:
  - Create these four kinds of keys
    - encryption, decryption, signature, verification
  - Encrypt messages using someone else's encryption key
  - Decrypt messages using your own decryption key
  - Sign messages using your own signature key
  - Verify signatures using someone else's verification key
  - Sign other people's keys using your own signature key (see later)

Obtaining keys

- Earlier, we said that Alice needs to get a copy of Bob's public key in order to send him an encrypted message
- How does she do this?
  - In a secure way?
- Bob could put a copy of his public key on his webpage, but this isn't good enough to be really secure!
  - Why?

Verifying public keys

- If Alice knows Bob personally, she could:
  - Download the key from Bob's web page
  - Phone up Bob, and verify she's got the right key
  - Problem: keys are big and unwieldy!


```
mQGiBDi5qEURBADitpDzvvzW+9lj/zYgK78G3D76hvvvIT6gpTIlwg6WIJNLKJat
O1yNpMIYNvpwi7EUd/lSNl6t1/AO22p7s7bDbE4T5NJdaOIOAgWeOZ/plIJC4+o2
tD2RNuSkwDQcxzm8KUNZOJla4LvgRkm/oUubxyeY5omus7hcfNrBOwjC1wCg4Jnt
m7s3eNfMu72Cv+6FzBgFog8EANirkNdC1Q8oSMDihWj1ogiWbBz4s6HMxzAaqNf/
rCJ9qoK5SLFeoB/r5ksRWty9QKV4VdhhCIy1U2B9tSTlEPYXJHQPZ3mwCxUnJpGD
8UgFM5uKXaEq2pwpArTm367kOtTpMQgXAN2HwiZv//ahQXH4ov30kBBVL5VFxMUL
UJ+yA/4r5HLTpP2SbbqtPWdeW7uDwhe2dTqffAGuf0kuCpHwCTAHr83ivXzT/7OM
```

Fingerprints

- Luckily, there's a better way!
- A fingerprint is a cryptographic hash of a key
- This, of course, is *much shorter*:
  - B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5
- Remember: there's no (known) way to make two different keys that have the same fingerprint

Fingerprints

- So now we can try this:
  - Alice downloads Bob's key from his webpage
  - Alice's software calculates the fingerprint
  - Alice phones up Bob, and asks him to read his key's actual fingerprint to her
  - If they match, Alice knows she's got an authentic copy of Bob's key
- That's great for Alice, but what about Carol, who doesn't know Bob
  - At least not well enough to phone him

Signing keys

- Once Alice has verified Bob's key, she uses her signature key to sign Bob's key
- This is effectively the same as Alice signing a message that says "I have verified that the key with fingerprint B117 2656 DFF9 83C3 042B C699 EB5A 896A 2898 8BF5 really belongs to Bob"
- Bob can attach Alice's signature to the key on his webpage

Web of Trust

- Now Alice can act as an introducer for Bob
- If Carol doesn't know Bob, but does know Alice (and has already verified Alice's key, and trusts her to introduce other people):
  - she downloads Bob's key from his website
  - she sees Alice's signature on it
  - she is able to use Bob's key without having to check with Bob personally
- This is called the Web of Trust, and the PGP software handles it mostly automatically

So, great!

- So if Alice and Bob want to have a private conversation by email:
  - They each create their sets of keys
  - They exchange public encryption keys and verification keys
  - They send signed and encrypted messages back and forth
- Pretty Good, no?

Plot twist

- Bob's computer is stolen by "bad guys"
  - Criminals
  - Competitors
  - Subpoenaed by the RCMP
- Or just broken into
  - Virus, trojan, spyware
- All of Bob's key material is discovered
  - Oh, no!

The bad guys can...

- Decrypt past messages
- Learn their content
- Learn that Alice sent them
- And have a mathematical proof they can show to anyone else!

- How private is that?

What went wrong?

- Bob's computer got stolen?
- How many of you have never...
  - Left your laptop unattended?
  - Not installed the latest patches?
  - Run software with a remotely exploitable bug?
- What about your friends?

What really went wrong

- PGP creates lots of incriminating records:
  - Key material that decrypts data sent over the public Internet
  - Signatures with proofs of who said what
- Alice had better watch what she says!
  - Her privacy depends on Bob's actions

Casual conversations

- Alice and Bob talk in a room
- No one else can hear
  - Unless being recorded
- No one else knows what they say
  - Unless Alice or Bob tells them
- No one can prove what was said
  - Not even Alice or Bob
- These conversations are "off-the-record"

We like off-the-record conversations

- Legal support for having them
  - Illegal to record conversations without notification
- We can have them over the phone
  - Illegal to tap phone lines
- But what about over the Internet?

Crypto tools

- We have the tools to do this
  - We've just been using the wrong ones
  - (when we've been using crypto at all)
- We want perfect forward secrecy
- We want deniable authentication

Perfect forward secrecy

- Future key compromises should not reveal past communication
- Use a short-lived encryption key
- Discard it after use
  - Securely erase it from memory

- Use long-term keys to help distribute and authenticate the short-lived key

- Q: Why do these new long-term keys not have the very same forward secrecy problem?

Deniable authentication

- Do not want digital signatures
  - Non-repudiation is great for signing contracts, but undesirable for private conversations
- But we do want authentication
  - We can't maintain privacy if attackers can impersonate our friends
- Use Message Authentication Codes
  - We talked about these earlier

No third-party proofs

- Shared-key authentication
  - Alice and Bob have the same MK
  - MK is required to compute the MAC
  - How is Bob assured that Alice sent the message?
- Bob cannot prove that Alice generated the MAC
  - He could have done it, too
  - Anyone who can verify can also forge
- This gives Alice a measure of deniability

Using these techniques

- Using these techniques, we can make our online conversations more like face-to-face "off-the-record" conversations
- But there's a wrinkle:
  - These techniques require the parties to communicate interactively
  - This makes them unsuitable for email
  - But they're still great for instant messaging!

Off-the-Record Messaging

- Off-the-Record Messaging (OTR) is software that allows you to have private conversations over instant messaging, providing:
- Confidentiality
  - Only Bob can read the messages Alice sends him
- Authentication
  - Bob is assured the messages came from Alice

Off-the-Record Messaging

- Perfect Forward Secrecy
  - Shortly after Bob receives the message, it becomes unreadable to anyone, anywhere
- Deniability
  - Although Bob is assured that the message came from Alice, he can't convince Charlie of that fact
  - Also, Charlie can create forged transcripts of conversations that are every bit as accurate as the real thing

Availability of OTR

- It's built in to Adium (a popular IM client for OSX)
- It's a plugin for pidgin (a popular IM client for Windows, Linux, and others)
  - With these two methods, OTR works over almost any IM network (AIM, ICQ, Yahoo, MSN, etc.)
- It's a proxy for other Windows or OSX AIM clients
  - Trillian, iChat, etc.
- Third parties have written plugins for other clients
  - Miranda, Trillian, Kopete

Recap

- Basics of cryptography
- Symmetric-key encryption
- Public-key encryption
- Integrity
- Authentication
- Security controls using cryptography
- Link-layer security
- Network-layer security
- Transport-layer security and privacy
- Application-layer security and privacy