Last time

- Internet Application Security and Privacy
  - Public-key encryption
  - Integrity

This time

- Internet Application Security and Privacy
  - Authentication
  - Security controls using cryptography
  - Link-layer security: WEP, WPA, WPA2

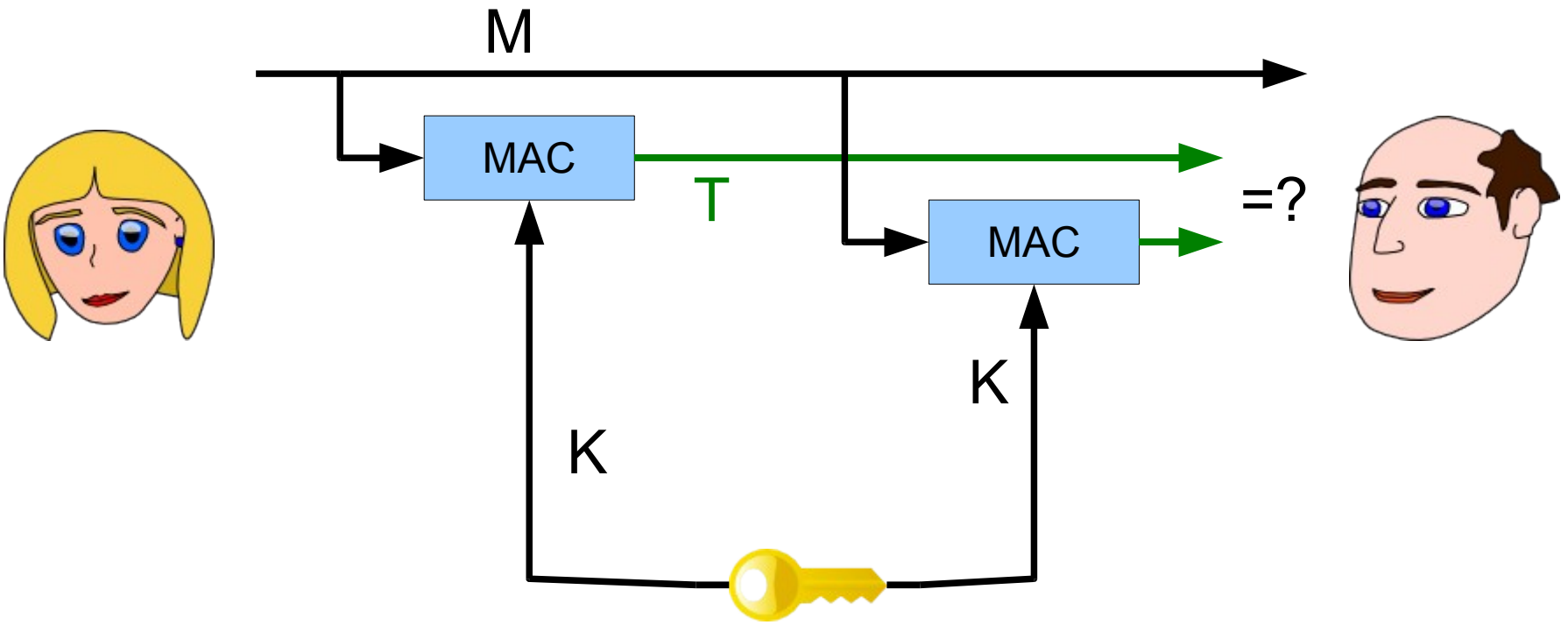
Using hashes for integrity

- Remember that hash functions can only guarantee integrity if the hash itself can be sent over a secure channel
  - Why?
- 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

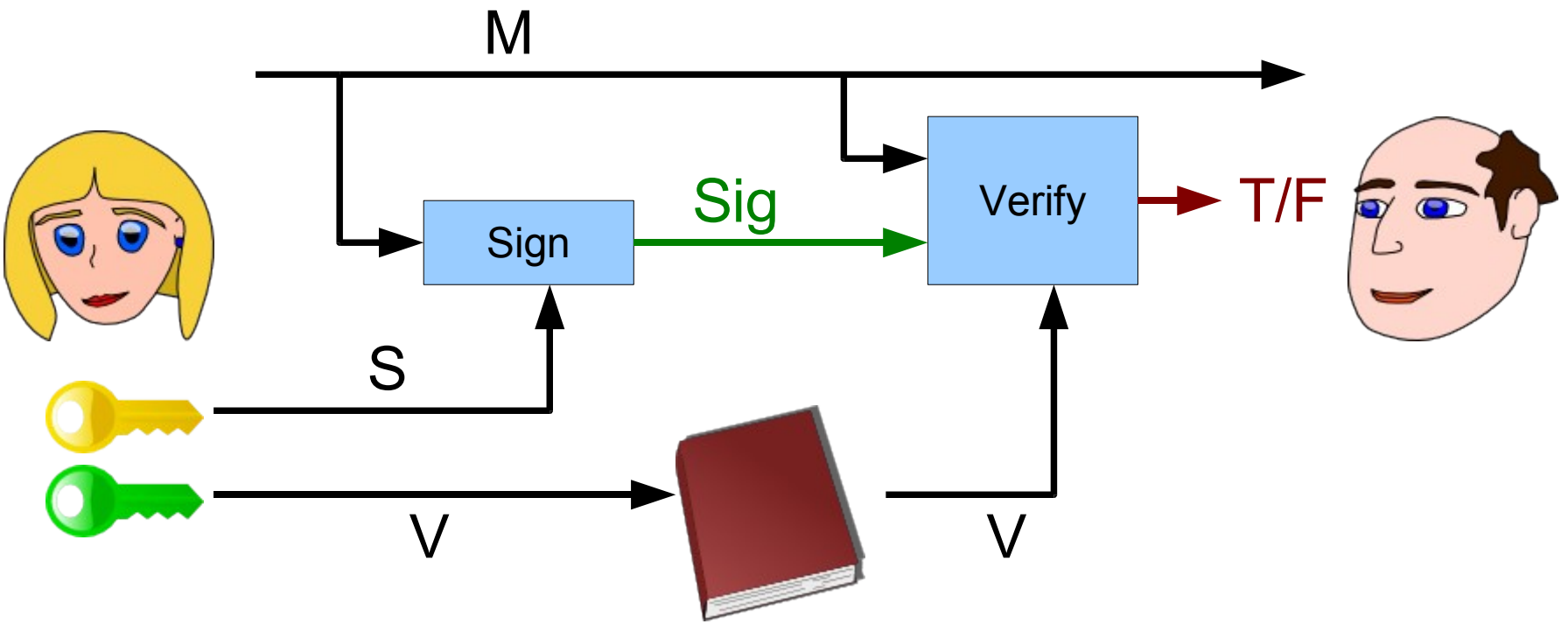
- For non-repudiation, what we want is a true digital signature, with the following properties:
- If Bob receives a message with Alice's digital signature on it, then:
  - Alice, and not an impersonator, sent the message,
  - the message has not been altered since it was sent, and
  - Bob can prove these 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 locks the message with her secret **signature key**.
- To verify Alice's signature:
  - Bob unlocks the message with his copy of Alice's **verification key**.
  - If it unlocks 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 secrecy 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)
    - SSL does 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 part, as well as a bunch of personal info, to the CA.
  - 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.
- Everyone is assumed to have a copy of the CA's signature key, so they can verify the signature on the certificate.

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!

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 which will only execute encrypted code
- The processor will decrypt instructions before executing them
- The encryption key is processor-dependent
- Malware won't be able to spread without knowing the processor key
- Downsides?

OS authentication

- Authentication mechanisms sometimes use cryptography

- Unfortunately, people are bad at doing cryptography in their heads, so some hardware token is needed

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, PGP, OTR, Mixminion

Link-layer security controls

- Intended to protect local area networks
- Most common example today: 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 <*M,c(M)*> 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 <*M',c'*>
  - 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?

Recap

- Internet Application Security and Privacy
  - Authentication
  - Security controls using cryptography
  - Link-layer security: WEP

Next time

- Internet Application Security and Privacy
  - Link-layer security: WEP, WPA, WPA2
  - Network-layer security: VPN, IPSec
  - Transport-layer security and privacy: TLS / SSL, Tor