Security

Read from the textbook

- Ch16.1 The Security Problem
- Ch16.4 Cryptography as a Security Tool
- Ch16.5 User Authentication

Cryptography

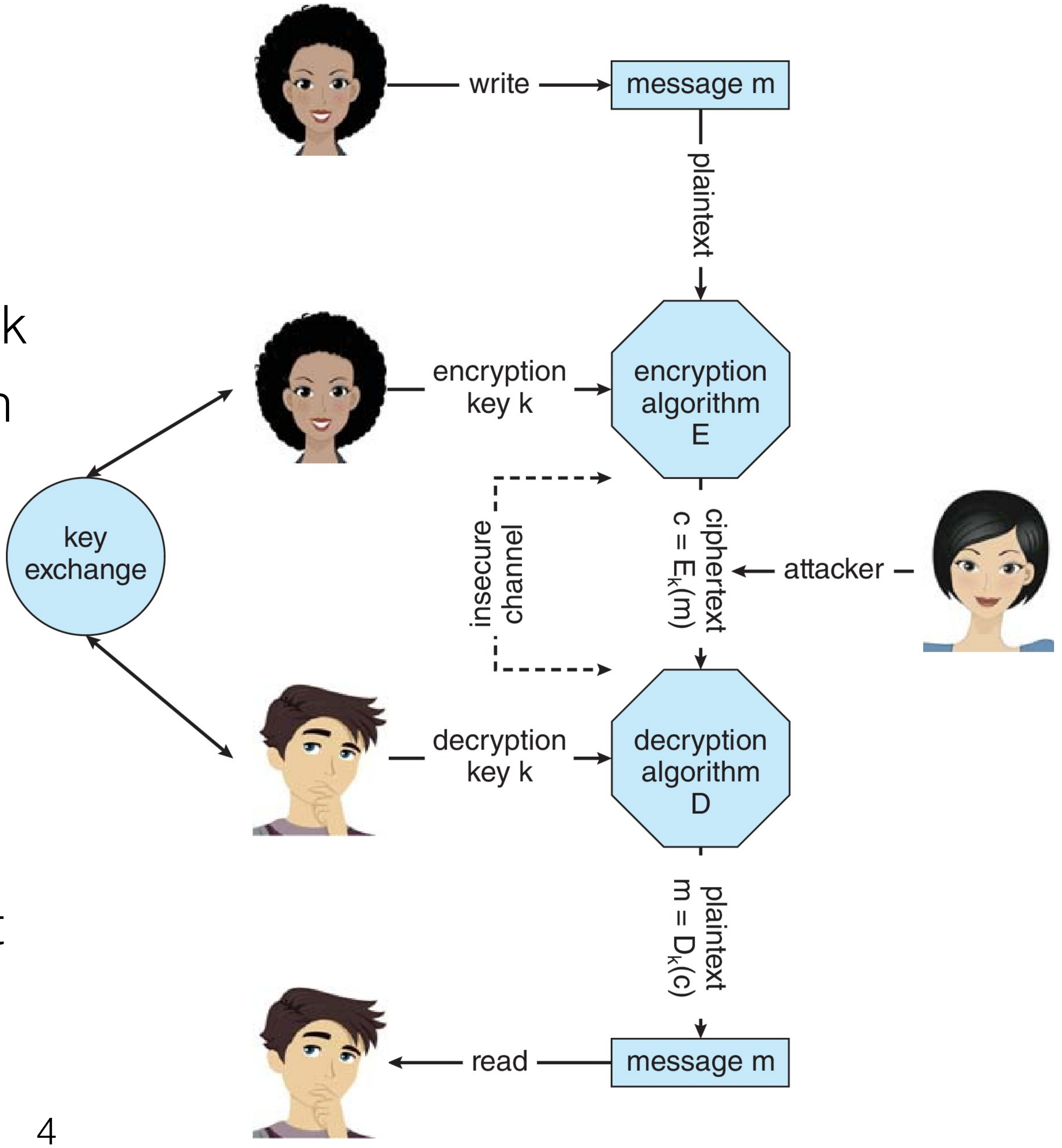
- Means to constrain potential senders (sources) and / or receivers (destinations) of messages
  - Based on secrets (keys)
  - Enables
    - Confirmation of source
    - Receipt only by certain destination
    - Trust relationship between sender and receiver

Encryption

- Constrains the set of possible receivers of a message
- Encryption algorithm consists of
  - Set K of keys
  - Set M of Messages
  - Set C of ciphertexts (encrypted messages)
  - A function $E : K \to (M \to C)$. That is, for each $k \in K$, E_k is a function for generating ciphertexts from messages
    - Both E and E_k for any k should be efficiently computable functions
  - A function $D : K \to (C \to M)$. That is, for each $k \in K$, D_k is a function for generating messages from ciphertexts
    - Both D and D_k for any k should be efficiently computable functions

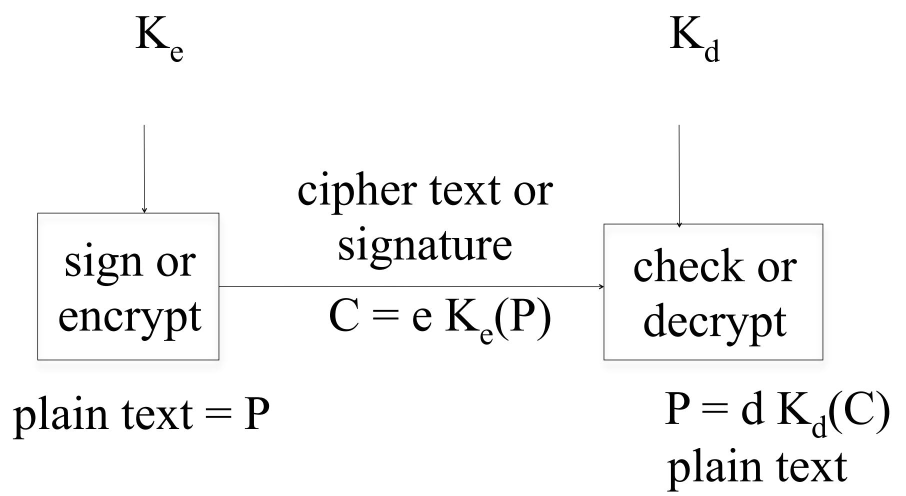
Encryption (cont.)

- An encryption algorithm must provide this essential property: Given a ciphertext c ∈ C, a computer can compute m such that Ek(m) = c only if it possesses k
    - Thus, a computer holding k can decrypt ciphertexts to the plaintexts used to produce them, but a computer not holding k cannot decrypt ciphertexts
    - Since ciphertexts are generally exposed (for example, sent on the network), it is important that it be infeasible to derive k from the ciphertexts



Key Pairs

- A little bit of information on methods of cryptography. Capabilities use cryptography.
- Information is encoded using a key.
- It is decoded using either the same or another key.
- Symmetric algorithms – same key, must be kept secret.
- Asymmetric algorithms use different keys for encrypting and decrypting (one key can be public). The idea is that although the keys are mathematically related it is *impossible* to produce one from the other.



Public Key Use

We can use public key algorithms in two ways.

1. Make the encoding key public – then anyone can encrypt messages but only the holder of the private key can decrypt.

   Useful for talking to a site without interception.

2. Make the decoding key public – then anyone can decrypt the messages but only the holder of the private key can encrypt them.

   Useful for proving the message came from the holder of the private key.

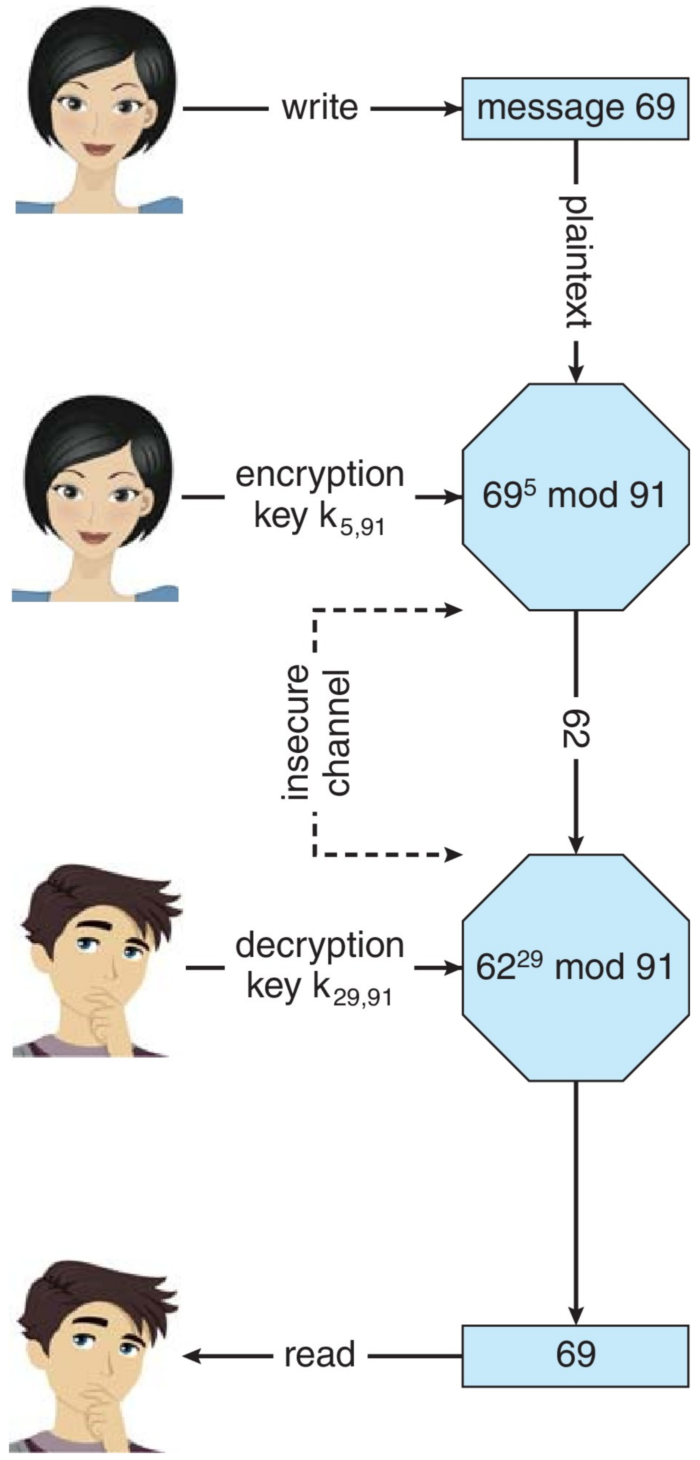
   Capabilities can be of this form.

Asymmetric Encryption

- Most common is **RSA** block cipher
- Efficient algorithm for testing whether or not a number is prime
- No efficient algorithm is know for finding the prime factors of a number
- Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so k_e need not be kept secret and can be widely disseminated
    - k_e is the **public** key
    - k_d is the **private** key
    - N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 512 bits each)
    - Encryption algorithm is $E_{ke,N}(m) = m^{k_e} \bmod N$, where k_e satisfies $k_e k_d \bmod (p-1)(q-1) = 1$
    - The decryption algorithm is then $D_{kd,N}(c) = c^{k_d} \bmod N$

Asymmetric Encryption Example

- Suppose p = 7 and q = 13
- We then calculate N = 7∗13 = 91 and (p−1)(q−1) = 72
- We next select k_e relatively prime to 72 and< 72, yielding 5
- We calculate k_d such that $k_e k_d$ mod 72 = 1, yielding 29
- We how have our keys
  - Public key, $k_{e,N}$ = 5, 91
  - Private key, $k_{d,N}$ = 29, 91
- Encrypting the message 69 with the public key results in the cyphertext 62
- Cyphertext can be decoded with the private key
  - Public key can be distributed in cleartext to anyone who wants to communicate with holder of public key



Digital Signing

- We need a way of proving that messages come from who they say they are and haven't been altered on the way.
- The sender puts the message through a hash algorithm to produce a message digest (e.g. SHA-256).
- Then encrypts the digest with its secret key. This is the digital signature for this message.
- Sends the message and the signature.
- The recipient uses the sender's public key to decrypt the signature.
- Also calculates the message digest on the message (SHA-256). (Note: Microsoft, Google, Mozilla etc do not accept SHA-1 certificates after 2017.)
- Checks the local digest value with the one sent.
- If they match, the message was sent by the proper subject and was not modified on the way.

Sharing Keys

There are problems with distributing keys.

- If the key is public the receiver still needs a way of checking that the key is authentic.
- If the key is private we need to ensure that only the right domains get the key.

Solutions to checking key authenticity:

- the key can be signed and we have the key to check the authenticity of the signature – a trust chain
- trusted third parties – can certify that the key (or certificate) came from the correct source
- contact the source and check some *fingerprint*

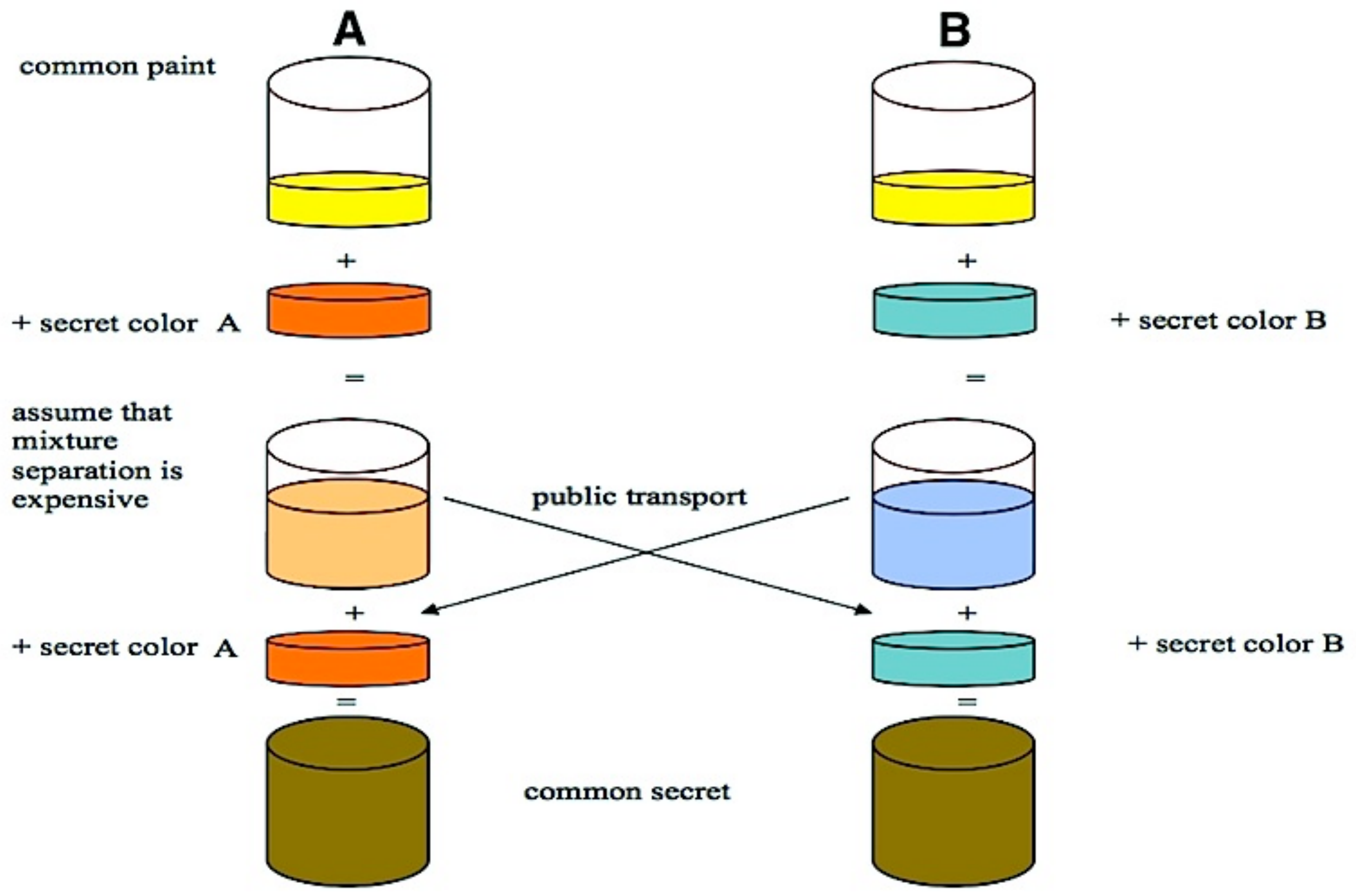
Sharing Secret Keys

- There are methods to cooperatively form secret keys during sessions e.g. the Diffie-Hellman protocol.
- The parties then share a secret key but they don’t know who the other party is.
- This can be solved but requires knowing signature verification keys.
- For OSs the usual approach is to use a server that holds secret keys for all domains. Then we can use the Needham-Schroeder Protocol.

Diffie-Hellman Protocol

- A method of securely exchanging cryptographic keys over a public channel
- A and B are the parties that want to communicate securely. They need a shared secret key.
- There are two public values (one is a large prime p and the other is related to the prime (primitive root modulo p)).
- Both parties generate a random private key. A produces key 'a' and B produces key 'b'.
- From these keys and the two original public values, both can produce public values they transmit to each other.
- A shared secret key can then be produced by combining the public key each gets from the other with their private keys.
- This shared key cannot easily be broken, with the public values.

Diffie-Hellman Protocol



http://en.wikipedia.org/wiki/Diffie-Hellman

Diffie-Hellman Example

Public prime value p = 23, number primitive root mod p = 5.
Alice chooses random number 6.
  Generates 5**6 mod 23 = 8 (sends to Bob)
Bob chooses random number 15.
  Generates 5**15 mod 23 = 19 (sends to Alice)
Alice then does 19**6 mod 23 = 2
and Bob does 8**15 mod 23 = 2

So 2 can be used as the secret key.
In use, p would be a huge number, 300 digits and the random numbers would have about 100 digits.

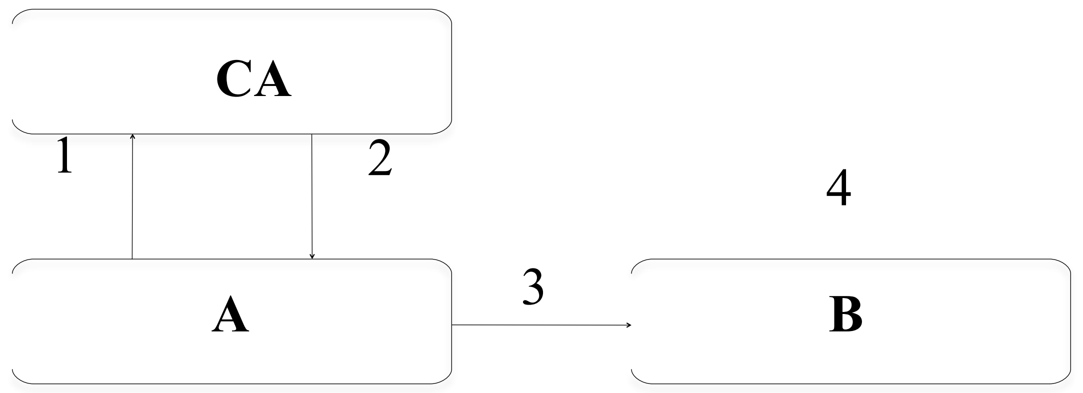
Certification

Diffie-Hellman is susceptible to Man-in-the-Middle attacks.

•So identities need to be proved. (MQV (Menezes-Qu-Vanstone) is based on Diffie-Hellman but uses the pre-existing public keys of both parties to include authentication.)

Certification Authorities

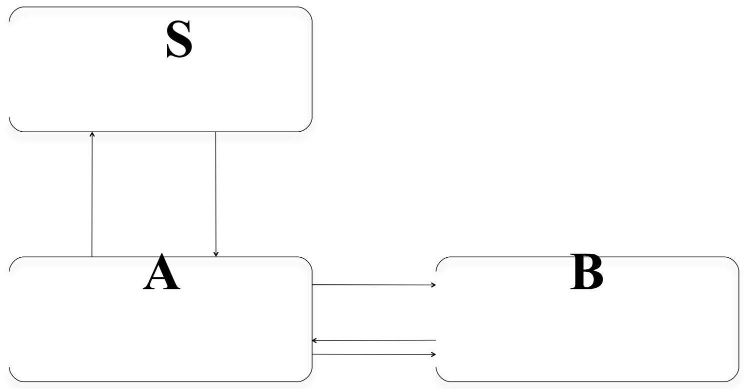
A wants to prove its identity to B.



1. A makes a request to the certification authority for a signed certificate. Includes its name and public key.
2. CA signs the message with its private key to produce a signature (the message and signature is now the certificate).
3. A sends the certificate to B.
4. B checks certificate using CA's public key.

Needham-Schroeder Protocol

But if we have a server with everyone's secret keys, we can use the Needham-Schroeder symmetric key protocol.



- A and B are the parties that want to communicate securely.
- S is the server with secret keys for A and B.

  So A can communicate securely with S and B can communicate securely with S.
- A tells S it wants to talk to B.
- S gives A a key for A and B to talk K_{AB}. It also includes a verifying message for B using B's secret key.
- A sends the verifying message to B.
- B checks the message (which includes A's identity and the key K_{AB})
- This algorithm is extended by Kerberos (and hence AFS).

How things go wrong?

The three “C”s of security failure:

- Change
- Complacency
- Convenience

Q: Which of the following is most likely to produce a problem based on “convenience”?

1. Making a quick patch to enable a program to work on a different system.
2. Allowing a superuser to access all machine state.
3. Not checking the bounds on an array.
4. Providing all users with an electronic card key.

Complacency

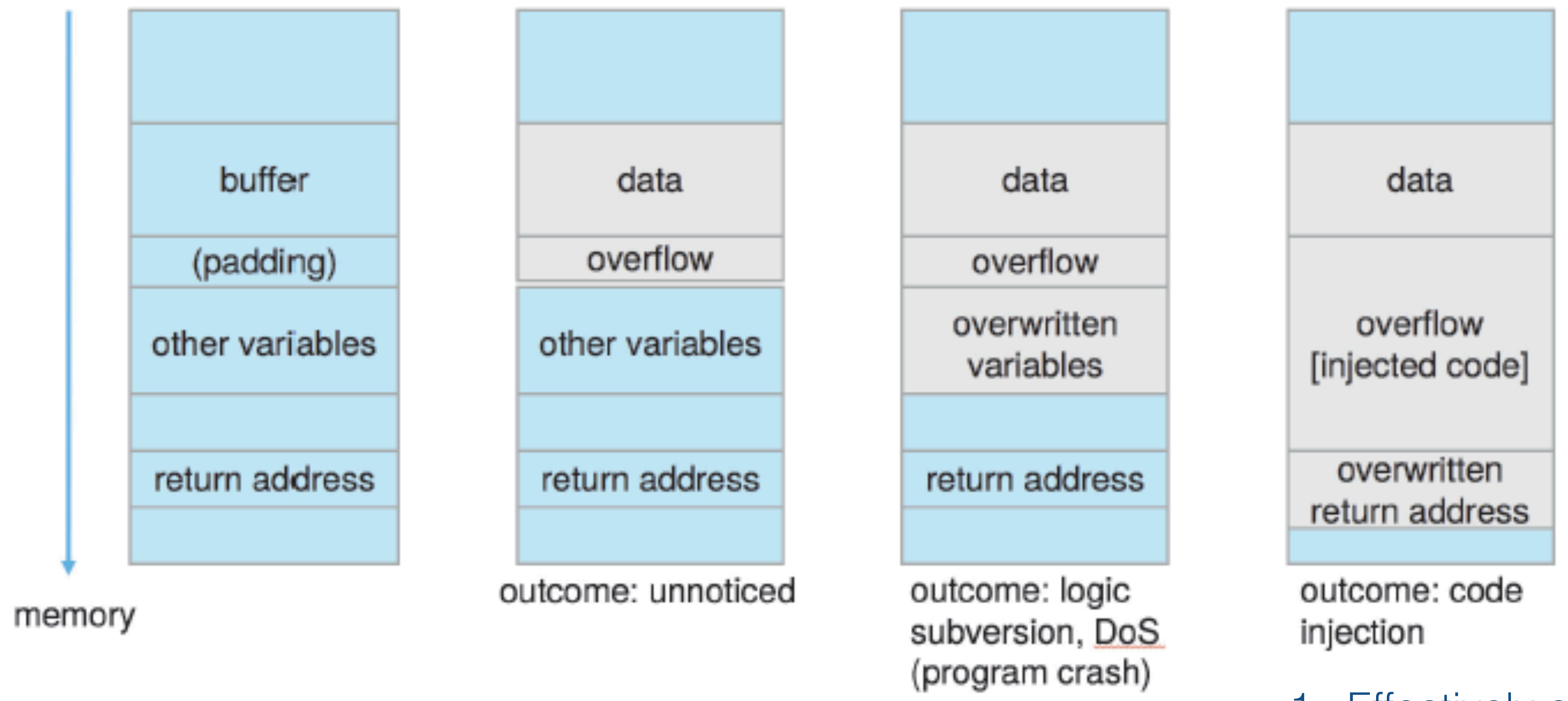
Bounds checking

- `fingerd` – process running to handle *finger* requests
  - Used `gets` library routine to get input into a buffer.
  - No length check.
  - If you know the architecture and OS you can overflow the allocated stack space for the buffer and leave a return address to an exec method call on the stack.
  - A buffer overflow attack. **See page 16.2.2 of the textbook.**
- The same sort of thing is possible with many OSs and there are thousands of cases of unchecked buffers causing security problems under all varieties of Windows and Unixes.
- VMS login – users could specify the machine they wanted to access `username/DEVICE=<machine>`

  The length of `<machine>` wasn't checked. The user's privilege mask was on the stack following the buffer. So you could overwrite the privilege mask to provide any desired privileges.

Stack and Buffer Overflow

- Exploiting bugs in system code that allows buffers to overflow
- Security breach caused by overflowing the buffer



1. Effectively changing where the program jumps to when it returns.
2. Insert some nefarious code.
3. Get the program to jump to our inserted code.

Preventing Stack Buffer Overflow Attacks

- Buffer length checks before storing data into the buffer/stack.
- Canaries - place a random value before the return address. If this value is modified then the stack has been compromised.
- Data Execution Protection (DEP) - use the memory management system to not allow execution of locations on the stack.
  - Return oriented programming can get around this (gadgets).
  - so the OS randomises the location of our code (ASLR - Address Space Layout Randomisation), making it harder to work out where code “gadgets” are.

More Complacency

- Syntax checking

```
rlogin
```

- Can remotely login to a machine with the rlogin command

```
rlogin –l username machine
```

- On Linux and AIX some versions did not check the syntax of the username and passed it straight on to the login command.

```
login username
```

- if the `username` was `–froot`

```
rlogin -l -froot machine
login -froot machine
```

- then the person was logged in without a password required; the `–f` flag means without a password to the login command.

Unthought Through Interactions

- Some programs cause security problems because of unexpected usage.

```
at
```

- The Unix at command is used to run programs at particular times.

```
at time –f file
```

- puts a copy of the file into a spool directory to be executed at the time.
- `at` doesn't check the read permissions when the file is put there.
- The user can read files they put in the spool directory (copies of files they were not allowed to read) – and then remove them to hide the evidence.

Convenience

- Adding security makes a system less usable.
- There is always a trade off between convenience and security.
- The most secure systems are very inconvenient.

Any expert will acknowledge that it's simple to create a totally secure computer: you simply unplug every connection, including power, encase the thing in concrete, and surround it with guards. By the same token, a pair of wire cutters provides the perfect network firewall: cut your Internet connections and we guarantee you won't suffer from Internet-based attacks.

from SideWinder site (no longer available)

Authentication

Even the best security system can be compromised if an intruder can successfully impersonate a legitimate user.

There also need to be policies that stop users ***sharing*** their identities with others.

From the OS point of view we need a way of allowing authenticated users access to the system but no one else.

We can use:

- possessions
- attributes
- knowledge

and combinations of these.

Possessions

- Keys or cards.
- Locks can be picked and smart cards can be analysed.
- Attackers (if they have access to a card) use techniques such as manipulating the power supply or clock to get secret information.
  - The hope is that the card will get into an unknown state and produce information it shouldn't.
  - A type of fuzzing - provide random, invalid, unexpected data and observe what happens. Used to find bugs and security problems.
- Or a side channel to extract information.
  - Even just observing how long it takes to perform computations can be used to cut down the possibilities when trying to guess a secret key.
- Of course keys and cards can be easily stolen as well.

Attributes

- Physical characteristics of the user.
  - palm prints
  - finger prints
  - iris patterns
  - retina patterns
  - voice print
- Also the way things are done.
  - typing patterns (different people type different sequences of characters at different speeds)
  - signatures – we include the speeds, directions and pressure of different strokes
- All of these biometric methods can suffer from false positives and false negatives.
- False positive – let someone have access who shouldn’t
- False negative – refuse access to a legitimate user
- The probabilities of each can be altered by changing parameters. Best to use a combination of techniques.

Before Next Time

Read from the textbook

- Ch16.2 Program Threats
- Ch16.3 System and Network Threats
- Ch16.6 Implementing Security Defenses
- Ch16.7 An Example: Windows 10