SCC.311: Security

Prof. Barry Porter

So far…

- We’ve seen two different kinds of communication middleware:
  - Remote Invocation (like RMI)
  - Group Communication

…today

- We examine core challenges and solutions to security in distributed systems

- These approaches can be embedded in any middleware platform (or any distributed system in general)

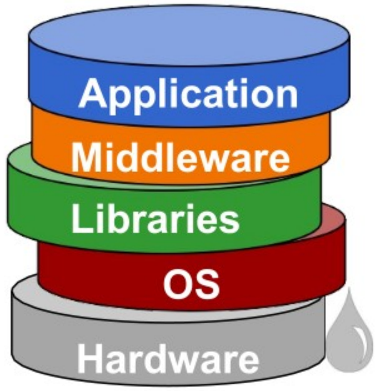
Disclaimer

- This is not a cyber-security course
- We're not going to cover cryptography theory or pen-testing
- We will cover the most common kinds of security in distributed systems, and how they're implemented and used

Security is hard

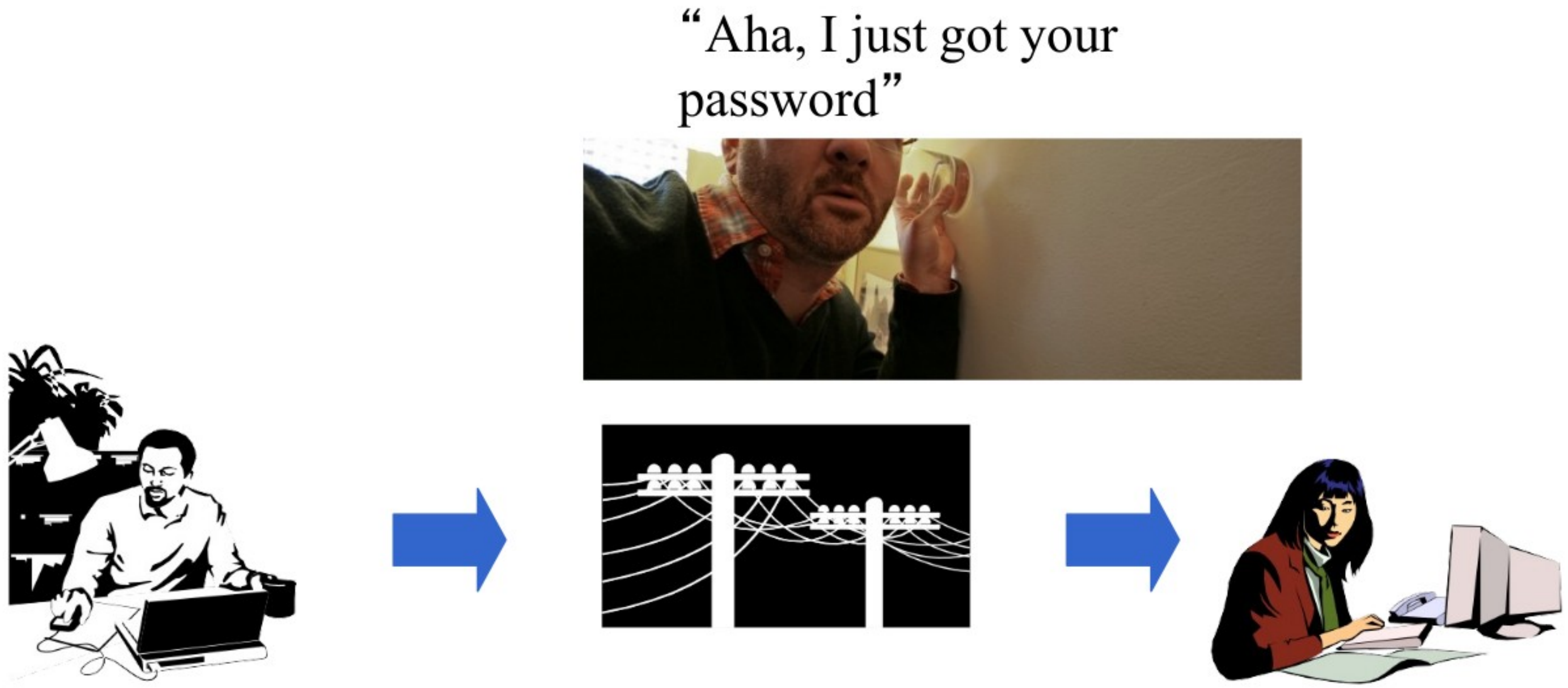
- Securing a system is about ensuring that bad things never happen
  - This means we need to know what every possible bad thing is

- A defender must understand and eliminate **every** possible threat and vulnerability; an attacker only needs to find **one** to succeed



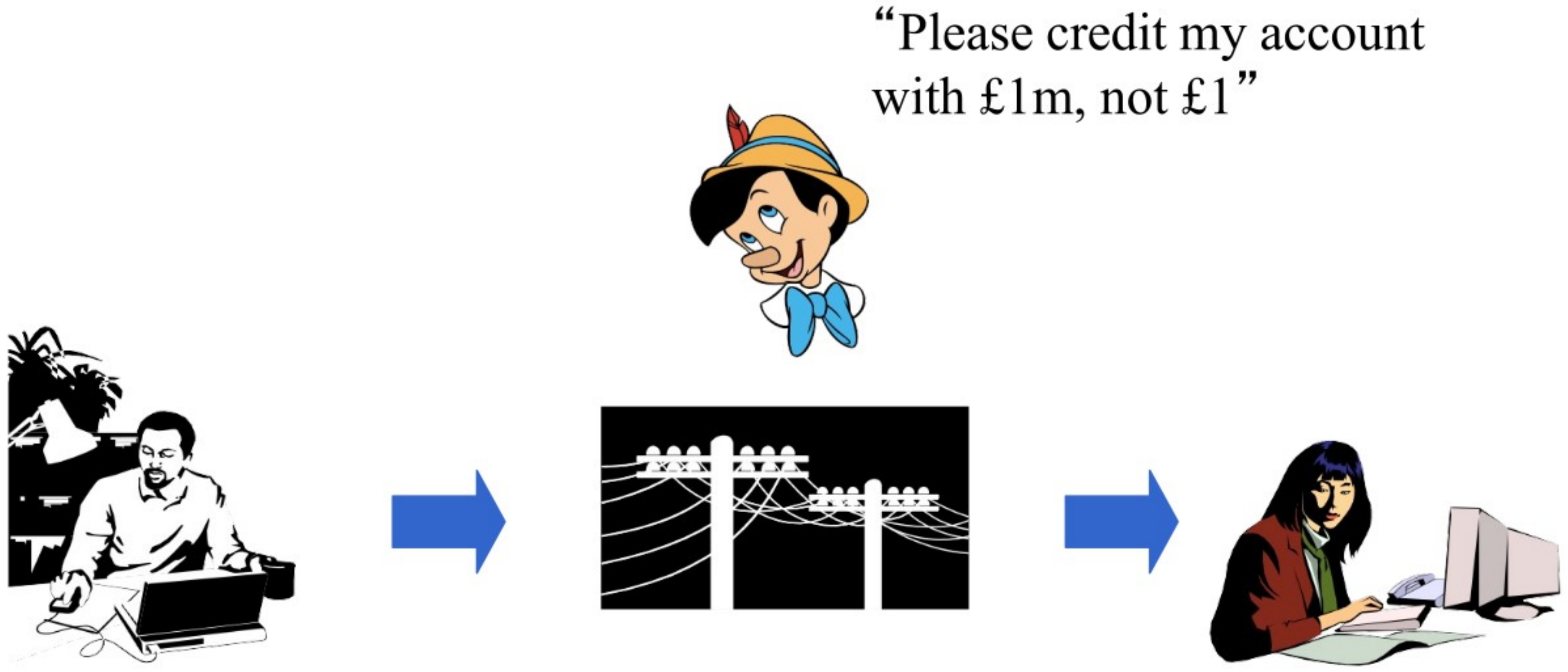
Broad classes of security threat in Distributed Systems

- 1. Eavesdropping



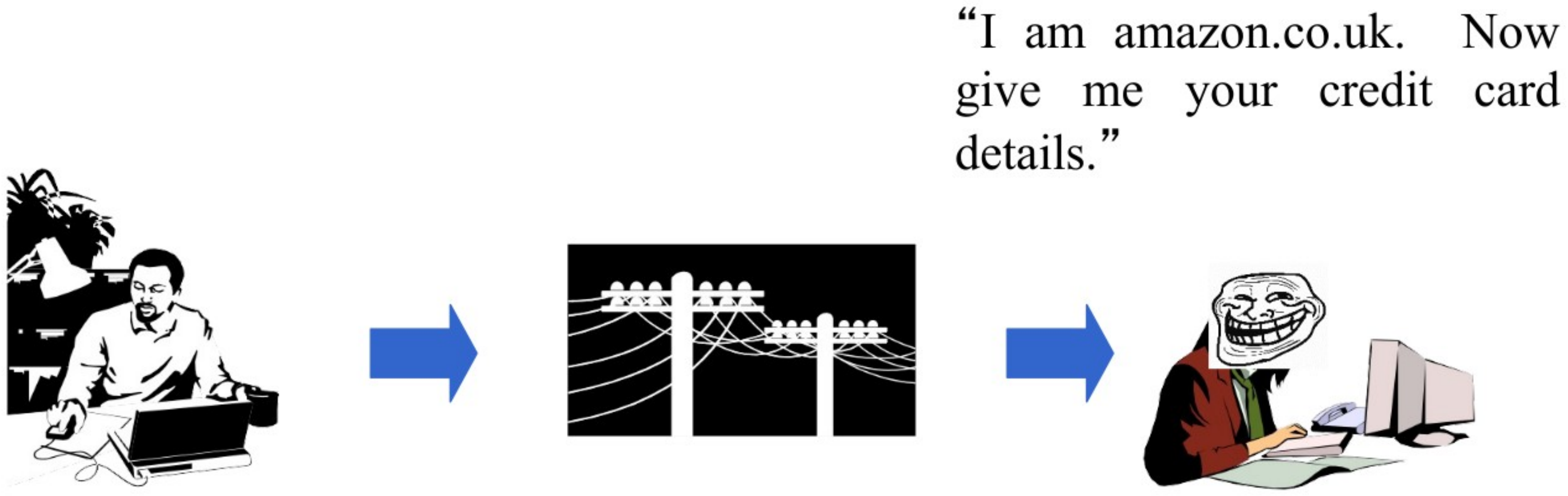
Broad classes of security threat in Distributed Systems

- 2. Tampering



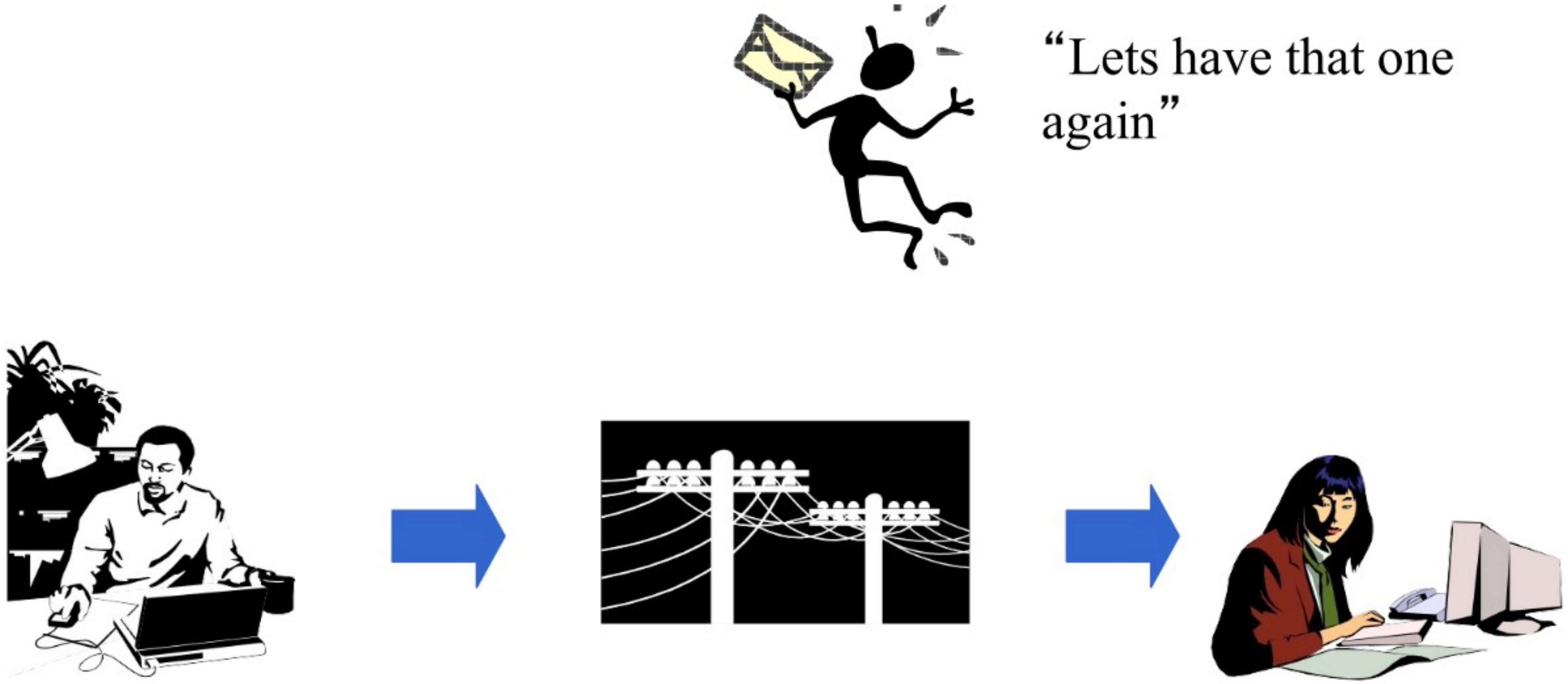
Broad classes of security threat in Distributed Systems

- 3. Masquerading



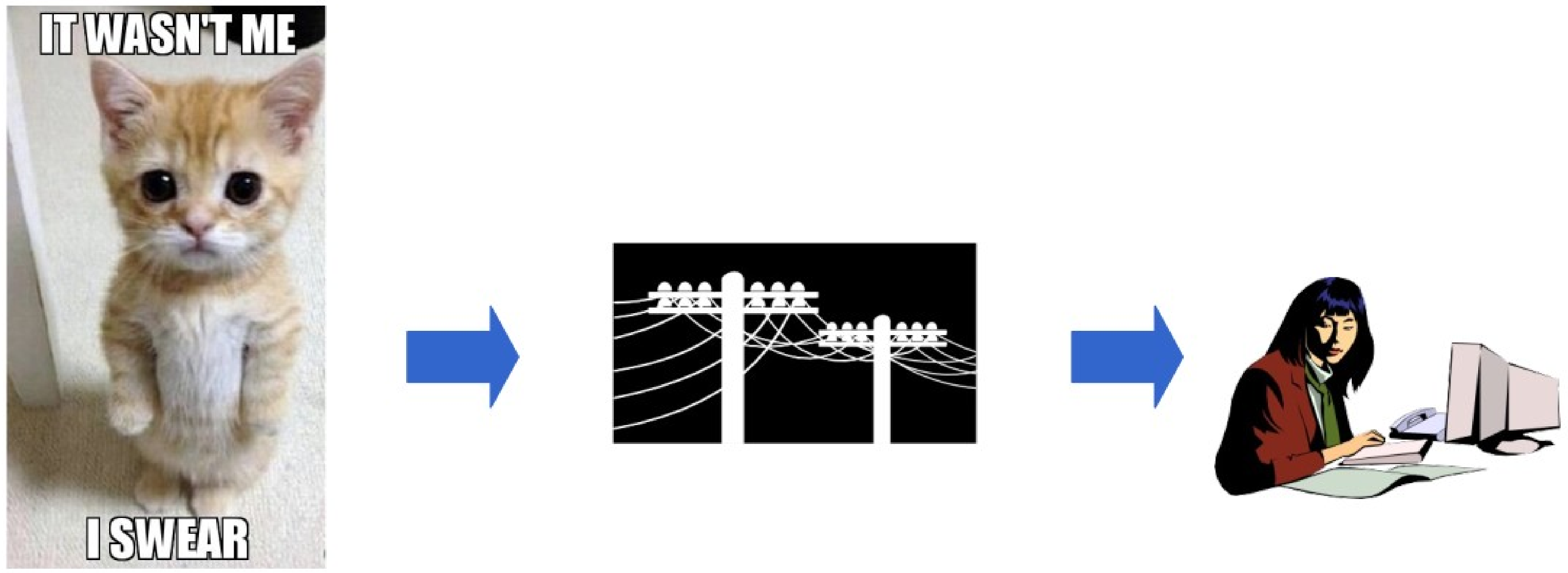
Broad classes of security threat in Distributed Systems

- 4. Replaying



Broad classes of security threat in Distributed Systems

- 5. Non-Repudiation



Security Mechanisms for Distributed Systems

- Encryption
  - To implement *confidentiality* of messages
- Authentication
  - To verify claimed identities of all parties
- Digital Signatures
  - To prevent tampering
- Authorisation
  - To define access rights to a given verified party
- Auditing
  - For the analysis of security breaches

Cryptography

Introduction to Cryptography

- There are two main kinds of cryptography:
  - Symmetric
  - Asymmetric

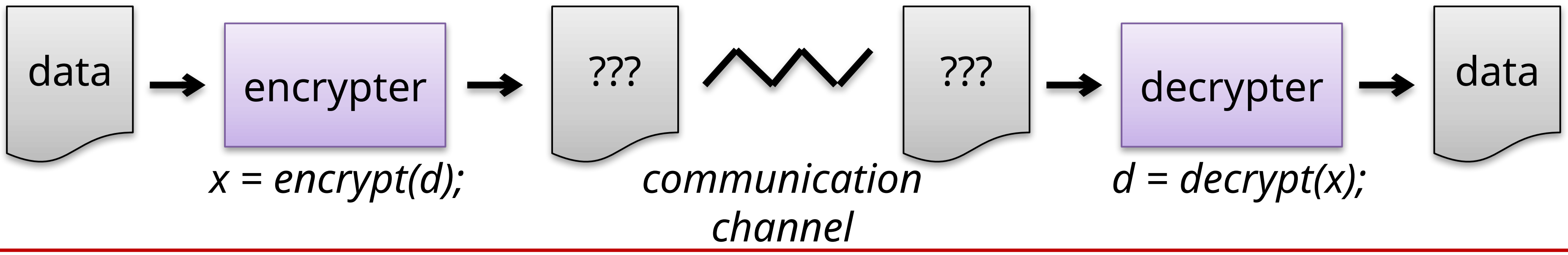
- Both kinds of cryptography are useful in different ways, and are often used in combination

Introduction to Cryptography

- Terminology:
  - **Plain text** means the un-encrypted, original-format data. It isn't necessarily "text", and could be binary, image data, etc.
  - **Cipher text** means the encrypted version of that data.

Symmetric Cryptography

- The most basic encryption is for the sender to have an *encrypt(x)* function and the receiver to have a matching *decrypt(x)* function
- The receiver knows which encrypt algorithm is being used by the sender, and matches it



Symmetric Cryptography

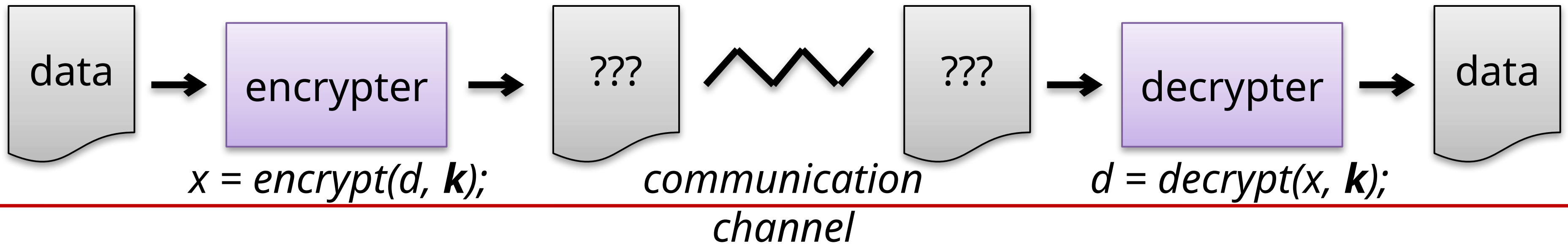
- An example of this is a “caeser cipher”
  - We shift each letter *n* places, where *n* is hard-coded in the algorithm
  - “hello” becomes “khoor” with *n* of 3

Symmetric Cryptography

- This approach has problems…
  - it's based on a "shared secret" between the sender and receiver (i.e. which encryption algorithm is used)
  - all clients have to use the same encryption algorithm, so the probability of the shared secret becoming a non-secret is quite high
  - there aren't that many different encryption algorithms around, so it's easy to "guess" which one is being used
  - data diffusion is non-existent…etc…

Symmetric Cryptography

- We therefore use a “key” ***k*** to modify the behaviour of an encryption algorithm
- In the most extreme case, a unique key can be generated for each interaction, then discarded
  - the problem then becomes “how to safely agree on a key” between a sender and a receiver



Symmetric Cryptography

- The strength of a cryptographic function is largely determined by two things:
  - the length of time that the same key is used
  - the size (or complexity) of the key itself

  - the general approach to cracking a cryptographic algorithm is to take the plain-text version of a message, with a copy of the encrypted version, and then try every possible key to generate a match
    - once we know the key, decrypting further messages is simple
    - the more complex the key, the more permutations we try

when we say “128-bit encryption”, we mean that the ***key*** *is 128 bits long –*
so it has 340,282,366,920,938,463,463,374,607,431,768,211,456 permutations

Symmetric Cryptography

- …the general approach to cracking a cryptographic algorithm is to take the plain-text version of a message, with a copy of the encrypted version, and then try every possible key to generate a match
    - how do we know the plain text version?
    - common protocols like HTTP have a predictable first *n* bytes

*when we say "128-bit encryption", we mean that the **key** is 128 bits long – so it has 340,282,366,920,938,463,463,374,607,431,768,211,456 permutations*

Symmetric Cryptography – Aims

- Obscurity (confusion): cipher is distant from plaintext
  - Hard to recover plaintext from cipher without knowing the secret (key)
- Diffusion:
  - small changes in the plaintext cause big changes cipher
  - diffusing the information about the plaintext's structure across the cipher
    - Substitutions and permutations
  - hard to detect simple useful facts, e.g. repeated patterns in the messages
- Fast: cheap to compute without compromising strength
- Infeasible to break
  - as opposed to impossible to break
- Open methods based on secrets
  - Encryption/Decryption methods are public

Symmetric Cryptography – Approaches

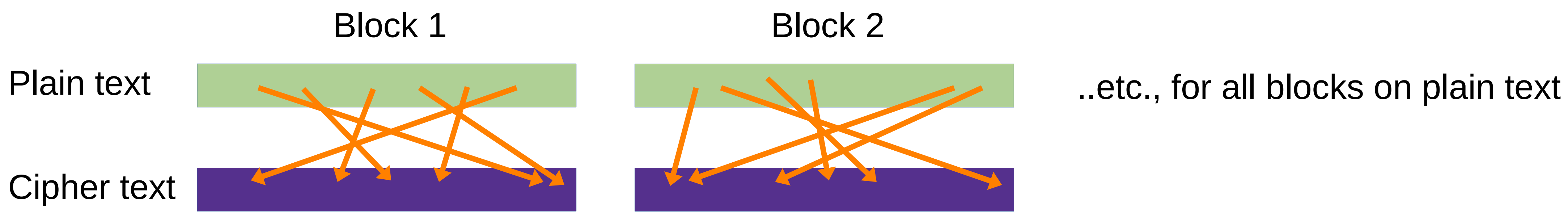
- DES, 1974
- Triple-DES: more complex (and much slower) than DES
- IDEA (International Data Encryption Algorithm)
  - developed by ETH in Zurich in 1990
  - 128-bit key
- AES (Advanced Encryption Standard):
  - replaced DES as the most commonly-adopted standard in 2001
  - uses the Rijndael algorithm
  - configurable key length
- Others:
  - Blowfish, Twofish, RC4, RC6, SEAL, Serpent, etc.

Symmetric Cryptography – Diffusion

- What does “diffusion” mean?
  - Most cryptographic functions work in terms of fixed-size blocks
  - We break our plain-text input data into blocks of (e.g.) 256 bits, encrypt that block, then take the next plain-text block of the same size, encrypt that, etc.

Symmetric Cryptography – Diffusion

- What does “diffusion” mean?
  - The diffusion of data by a cryptographic function means that the ciphertext bits, in each encrypted output block, are distant from their original position in the plaintext block
  - Where the distance is based on some factor of the key, and is mathematically hard to reverse without knowing the key

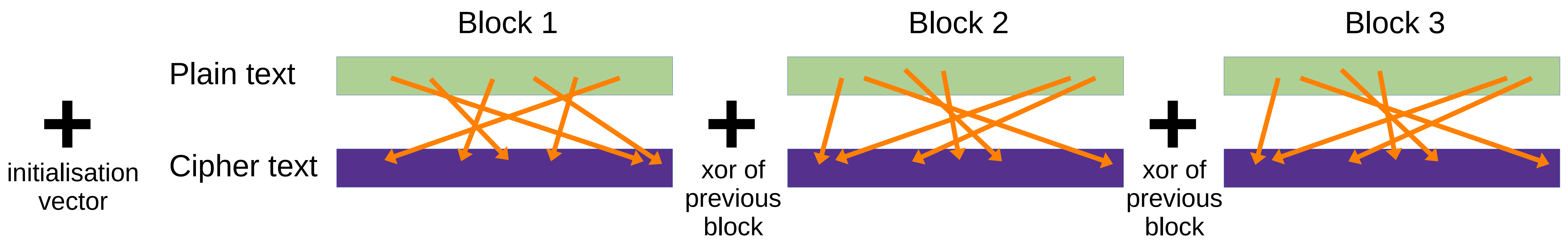


Symmetric Cryptography – Diffusion II

- Even with this, block-based ciphers have a potential problem: if the same plaintext sub-string appears more than once in the plain text input, the corresponding ciphertext output blocks can look the same
  - This can make it easier to crack keys and decrypt data

Symmetric Cryptography – Diffusion II

- We can use a technique called **cipher block chaining** to remove this problem:



Symmetric Cryptography

- Putting this all together, we have a cryptographic function which can take the **same** input plain text, and two different keys, and produces **different** output ciphertext, with good diffusion:

AES-CBC with 256-bit keys

Key: sam

Plain text: Hello! I love my distributed systems coursework!

$\longrightarrow$

Cipher text (base64): 1d6ed980f4c38f74b07c215caef02cbad0704f656924fd24085b37faee0f529f29759b613938338d59001e1b3a550357f18217cce1ee4be55c2e228e06775833

Key: alex

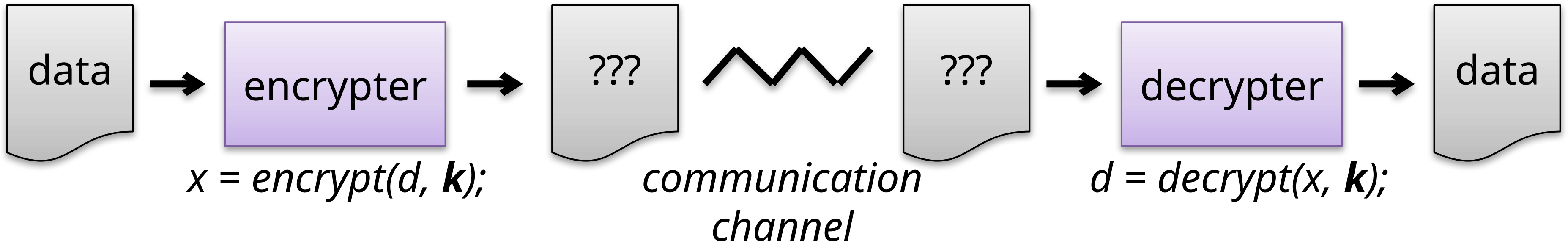
Plain text: Hello! I love my distributed systems coursework!

$\longrightarrow$

Cipher text (base64): 358a5a82db2bd8fdcf456ac2817c208251904668c3f8ddaba5ea924bdfa5ce7c7bff1abae4c125d4d1f9af3862f9d32454be2c80ce80afb7ca0e908da9cc874a

Symmetric Cryptography

- The remaining problem here is "how to safely agree on a key" between a sender and a receiver...?



Asymmetric Cryptography

- Instead of using a single, shared, key for both encryption and decryption, asymmetric encryption uses a **key pair**
  - One element of the key pair (the *public* key) can *only* **encrypt** data
  - The other element (the *private* key) can *only* **decrypt** data
  - Furthermore, the private key can only decrypt data that was encrypted by its matching public key

Asymmetric Cryptography

- The concept of asymmetric, or public-key cryptography, was invented by Ralph Merkle, Whitfield Diffie, and Martin Hellman, in 1976
  - The first algorithm was known as “Diffie-Hellman”
  - The public and private keys are mathematically related at the point of creation, but it’s infeasible to derive one from the other

Asymmetric Cryptography

- The current standard algorithm (at least on the Internet) is called RSA, named after its inventors Rivest, Shamir, and Adleman, and was invented in 1977

- RSA is based on the use of very large prime numbers, which are computationally hard to work with, to generate its keys

RSA

In this example our “very large” prime numbers are 11 and 13, which makes the maths a little easier to follow.

key generation

N = 11 * 13 = 143

T = (11 – 1) * (13 – 1) = 120

pick any number != 1 that’s *coprime* to 120; call this “e”
e = 23

compute the *modular multiplicative inverse* of e (mod T(N)); call this “d”
d = 47

your **public key** is then (N, e) and your **private key** is (N, d)

RSA operation to encrypt plaintext value v to encrypted value x: $x = v^e \bmod N$
RSA operation to decrypt ciphertext value x to plaintext value v: $v = x^d \bmod N$

RSA

because everybody has access to **N**, the algorithm relies upon the difficulty of calculating the original factors of N

→ if we could calculate these factors we would replay the key generation algorithm to discover the private key

key generation

N = 11 * 13 = 143

T = (11 – 1) * (13 – 1) = 120

pick any number != 1 that's *coprime* to 120; call this "e"
e = 23

compute the *modular multiplicative inverse* of e (mod T(N)); call this "d"
d = 47

your **public key** is then (N, e) and your **private key** is (N, d)

only requires **public** key values N and **e**

requires the **private** key values N and **d**

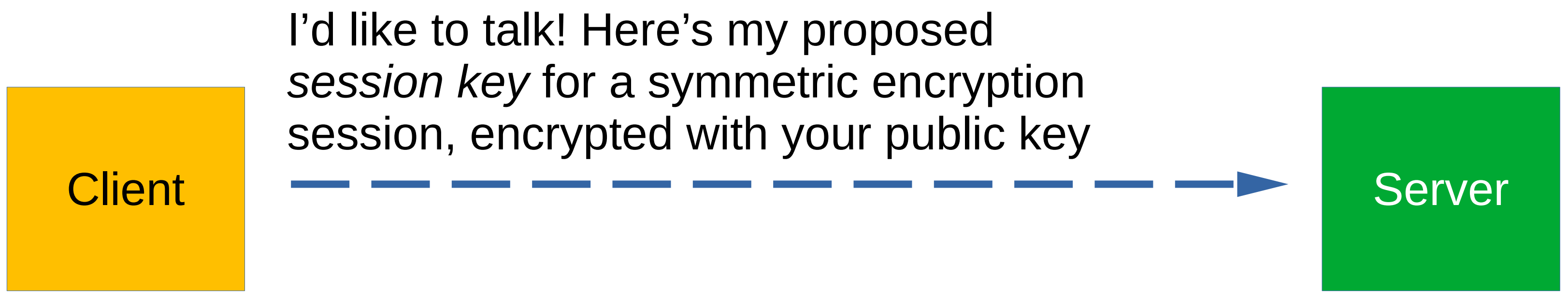
RSA operation to encrypt plaintext value v to encrypted value x: $x = v^e \bmod N$
RSA operation to decrypt ciphertext value x to plaintext value v: $v = x^d \bmod N$

Asymmetric Cryptography

- With our asymmetric cryptography system, we can safely give our public key to anyone, and allow people to send encrypted messages to us
  - in the knowledge that only our matching private key can decrypt those messages
  - it’s very difficult to derive the private key from the public one, assuming suitably large (and randomly-selected) prime numbers are used in the initial key generation

Asymmetric Cryptography applications

- Application #1: securely agreeing on a secret key
  - Symmetric encryption, like AES, has the problem that we need to somehow share our secret key before we can start talking

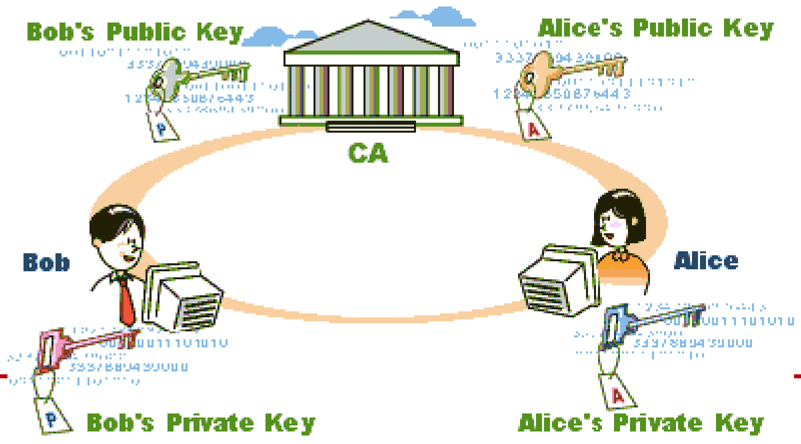

Client
I’d like to talk! Here’s my proposed *session key* for a symmetric encryption session, encrypted with your public key
Server

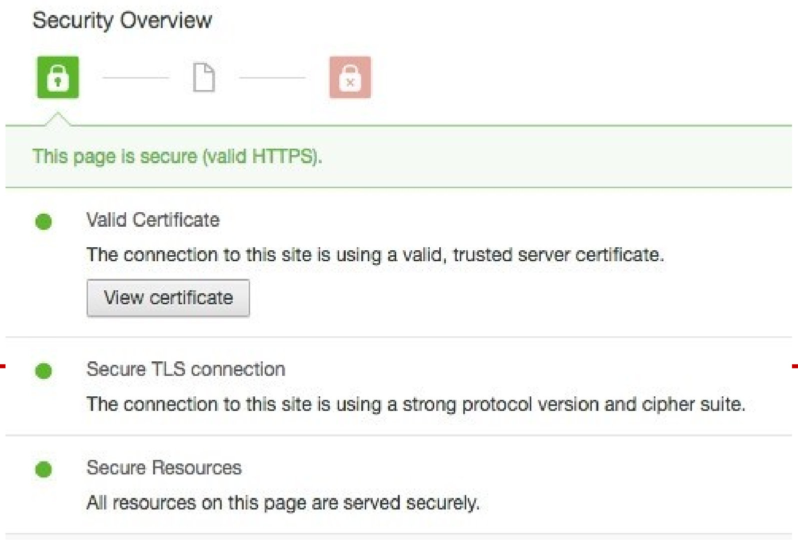
Asymmetric Cryptography applications

- Application #2: authentication
  - besides being able to securely communicate, another fundamental security challenge in distributed systems is verifying that the remote computer (like your bank) is really what it claims to be, before we send any sensitive data
  - this is called *authentication* (checking you are the authentic entity)
  - the simplest version of authentication is that you can only decrypt my messages if you have the matching private key of your public key
    - but how do I know that the public key is really yours?

Asymmetric Cryptography applications

- Application #2: authentication
  - we need a *trusted third party* to verify that a public key is really the right one
  - this is like asking somebody “is this public key actually that of my bank?”
  - on the Internet we use *Certificate Authorities* as trusted third parties
    - Certificate Authorities **sign** SSL Certificates for other organisations; an SSL certificate contains the public key of some organisation (like your bank)




Security Overview

This page is secure (valid HTTPS).

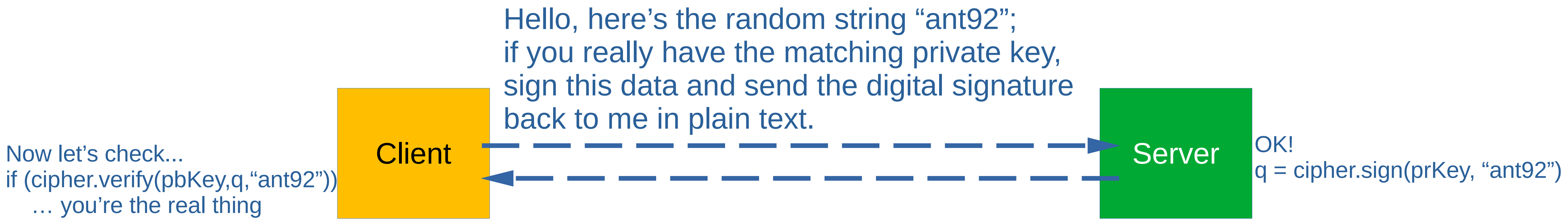
Valid Certificate
The connection to this site is using a valid, trusted server certificate.
View certificate

Secure TLS connection
The connection to this site is using a strong protocol version and cipher suite.

Secure Resources
All resources on this page are served securely.

Asymmetric Cryptography applications

- Application #3: digital signatures (for authentication)
  - we normally use a public key to encrypt data in asymmetric cryptography
  - but there's another trick we can do
    - we can use the **private key** to generate a **digital signature** of some plaintext data
    - a digital signature is a hashed value (non-reversible) of some plaintext data
    - we can use the public key of a key pair to **verify** that a digital signature was really generated by the matching private key



Hello, here's the random string "ant92";
if you really have the matching private key,
sign this data and send the digital signature
back to me in plain text.

OK!
q = cipher.sign(prKey, "ant92")

Now let's check...
if (cipher.verify(pbKey,q,"ant92"))
 … you're the real thing

Asymmetric Cryptography applications

- Application #3: digital signatures (for authentication)
  - SSL certificates are digitally signed using the private key of a certificate authority; if we trust we have the real public key of that certificate authority, we have reasonable grounds to trust that the SSL certificate is genuine
  - How do we know that we have the real public key of the certificate authority? Where do these keys come from, on which all other trust is based?

Asymmetric Cryptography applications

- Application #3: digital signatures (for tamper-prevention)
  - we can also use digital signatures to help assure that data hasn’t been tampered with in-transit
  - here we can encrypt our data for transport, then sign our entire encrypted data package with our private key, and send that signature too
  - the receiver can verify the signature using the public key, then only decrypt (or otherwise trust) the data if signature verification was successful

Summary

- Cryptography is at the heart of many security solutions in distributed systems, from encryption to authentication

- Understanding how to correctly use symmetric and asymmetric cryptography goes a long way to creating secure systems

Related reading

- CDBK, Chapter 11
- TvS, Chapter 9