School of Computer Science

University of St.Gallen

“From insight to impact”

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBAS ACCREDITED

Fundamentals of Computer Science for Business Studies

Week 6: Guided Exercise

Prof. Dr. Simon Mayer

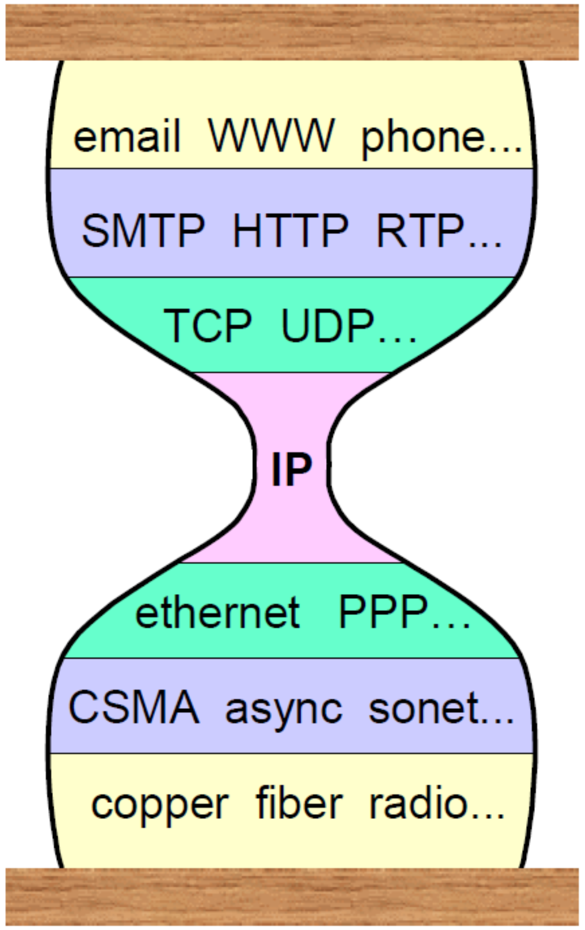
simon.mayer@unisg.ch

Computer Networks and Distributed Systems

Down to the individual **bit on the cable** and up to the **user-facing application**, communication among machines is defined by a set of **protocols.**

Network interfaces of individual machines are identified using **MAC addresses.** These addresses are needed for managing access to a transmission medium (e.g., a cable)

The **Internet** is a network of networks that all use the **Internet Protocol**
- **IP** provides addresses for Internet hosts and forms a narrow waist of the networking stack
- UDP is used for unreliable best-effort communication on top of IP
- TCP is used for more reliable, in-order communication and flow control on top of IP
- **DNS** assigns stable (host)names to IP addresses
- The **Web** is one of the main applications of the Internet, its main protocol is HTTP(S)

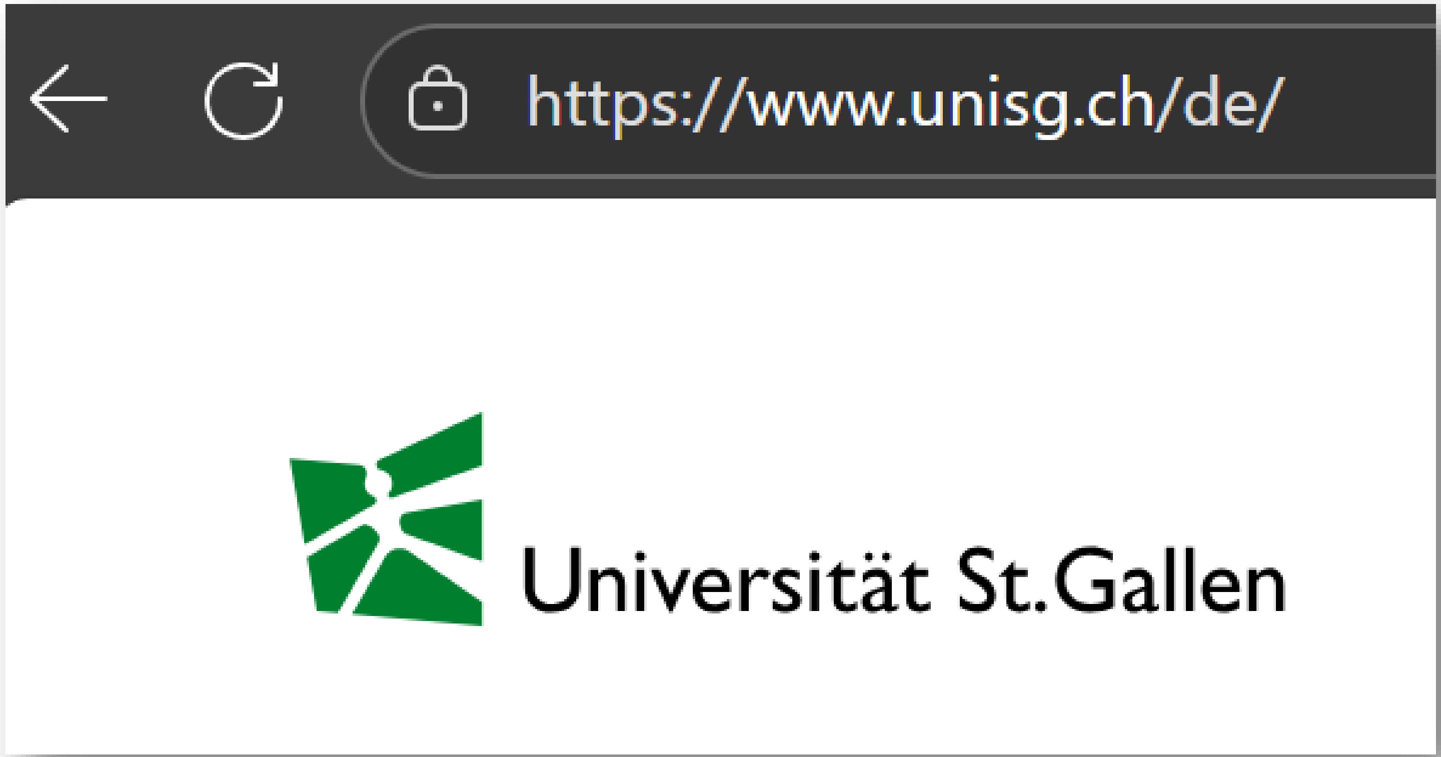


This Week's Menu

Motivation & Web APIs
Computer Networks Basics
The Networking Stack
The World Wide Web
Briefing: Security on the Web

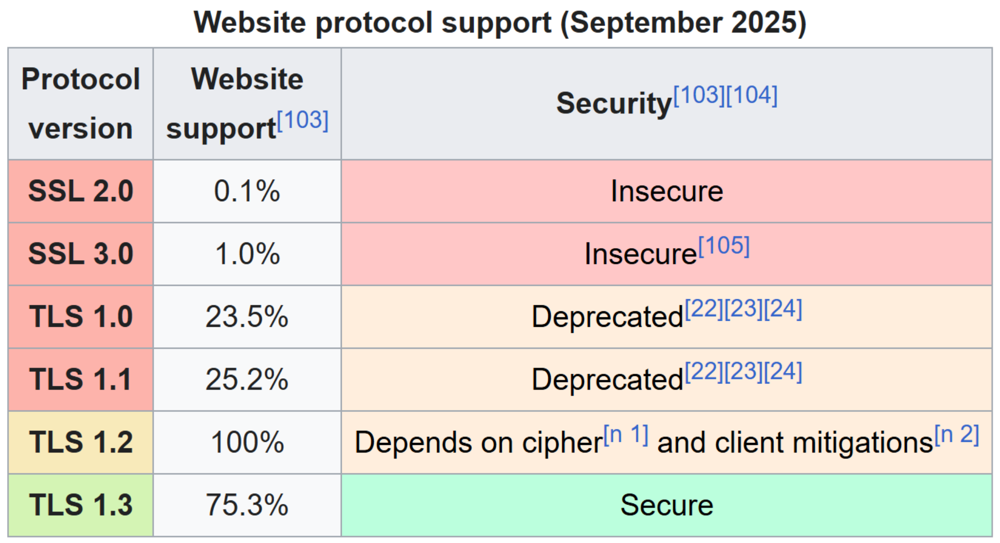
I'm showing demos (available on Canvas) **locally on my machine.** You can run them as well if you want.

1. Install a **python interpreter.** This is available and described in https://www.python.org/downloads/
2. Put code into **.py files** and then run it from your **Terminal** (e.g., PowerShell on Win or Terminal on Mac)
3. Download FCS-BWL-06-DistributedSystems-Demos.zip from Canvas


https://www.unisg.ch/de/
Universität St.Gallen

Why HTTP«S»?

HTTPS connections use a protocol that is called **Transport Layer Security** (TLS) and that provides data privacy, data integrity, and authentication – at the same time!



Website protocol support (September 2025)

Protocol version	Website support[103]	Security[103][104]
SSL 2.0	0.1%	Insecure
SSL 3.0	1.0%	Insecure[105]
TLS 1.0	23.5%	Deprecated[22][23][24]
TLS 1.1	25.2%	Deprecated[22][23][24]
TLS 1.2	100%	Depends on cipher[n 1] and client mitigations[n 2]
TLS 1.3	75.3%	Secure

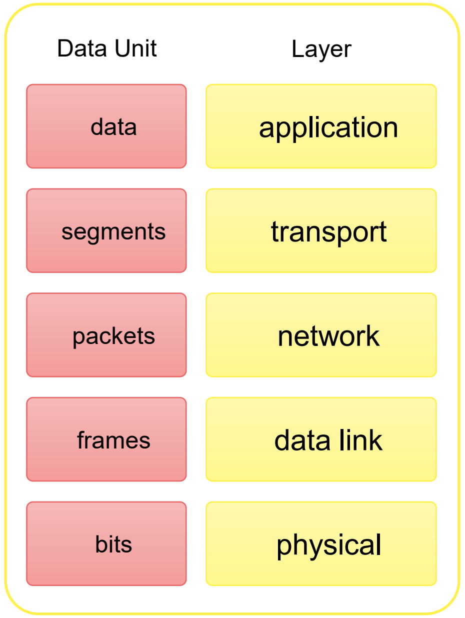
https://en.wikipedia.org/wiki/Transport_Layer_Security

Data **Privacy**:	Only the intended recipient **can read** what the sender sends.
Data **Integrity**:	Noone **can modify** the data that the sender sends.
Authentication:	The recipient can **verify** that the sender is who they claim they are.

HTTPS and TLS

HTTPS (“HTTP-over-TLS”) = Security on the **transport layer!**

– The entire request **above transport is encrypted** but nothing at or below the transport layer



Data Unit	Layer
data	application
segments	transport
packets	network
frames	data link
bits	physical

So what does that mean **in practice**? When I send an HTTPS request, …

…can an eavesdropper **modify the request** sent over HTTPS?

…can they see the **request contents**?

…can they see the **request URL**?

…can they see the **host IP address**

Yes!

Why?

…can they see the **host port number**?

Yes!

…can they see the **HTTP request method**?

…can they see the **amount of data** transferred?

Yes!

…can they see the **duration** of the interaction with the host?

Yes!

But this does not mean that encrypted data is never leaked in another way…

Case Study: Information Leakage through DNS and SNI

If Bob's **online behavior was actually fully encrypted**, we should not see **any of it on the wire** – correct?

First Try: Domain Name System (DNS; remember Wednesday)

Next Try: Server Name Indication (SNI; here is more information)

Brief Introduction to Secure Communication

```
EBIIL TLOIA
```

This is a so-called „substition cipher" where each letter is replaced by another letter

The characters **seem random**… any two five-letter words (out of ~171'000 english words) **seem equally likely**

Substitution Cipher / Caesar Cipher

To encrypt, each letter of a word is replaced with the corresponding letter from the cipher.

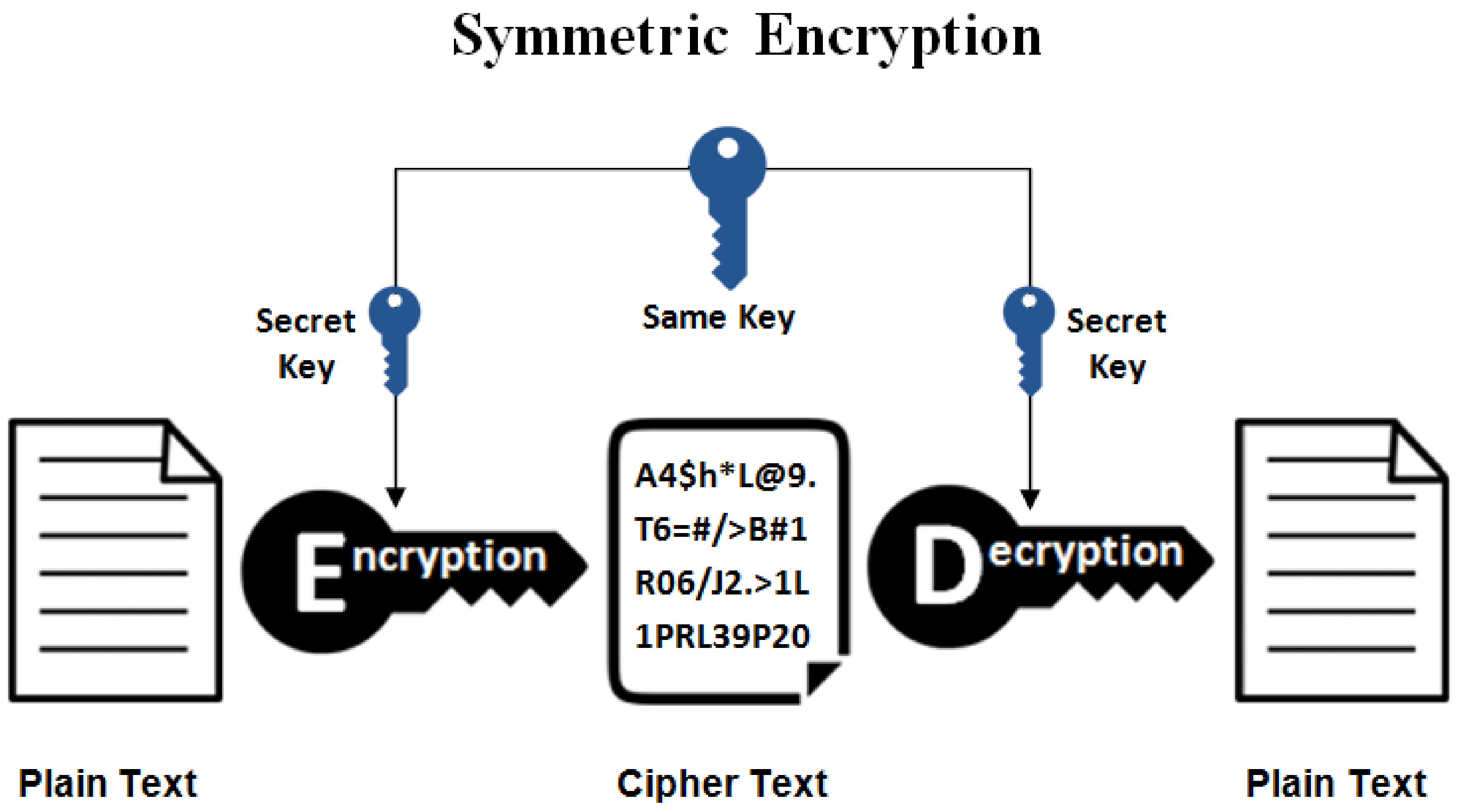
```
Plain Letters:                  ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher / Encrypted Letters:     XYZABCDEFGHIJKLMNOPQRSTUVW
```

```
Plaintext:                      HELLO WORLD
Ciphertext:                     EBIIL TLOIA
```

To decrypt, we turn the procedure around.

```
Ciphertext:                     EBIIL TLOIA
Plaintext:                      HELLO WORLD
```

This is an example of a Symmetric Cryptosystem



https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

Attacks on Symmetric Cryptosystems

Key Search Attacks

Typical keys are **256 bits** long. Then, the number of possibilities of what the key might be is:

2^256 = 115792089237316195423570985008690000000000000000000000000000000000000000000000

How long does it take a system on average to find the correct key if it needs **1 nanosecond to check** one key?

1835871531540401373407708412745600000000000000000000000 **billion years**

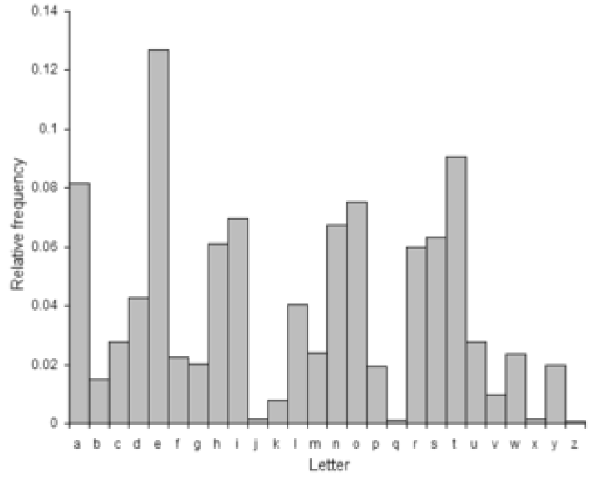
14 billion years (age of universe)

If the key lenghts is increased by 1 bit to **257 bits**, the system needs **double** as long.

Cryptanalysis

- Known Plaintext Attacks
- Chosen Plaintext Attacks
- Statistical Attacks
- Differential Analyses

The most frequent letter in English texts is „e“ (at around 12%). So if a system encodes letter-by-letter, and the attacker has access to a long plaintext/ciphertext pair….



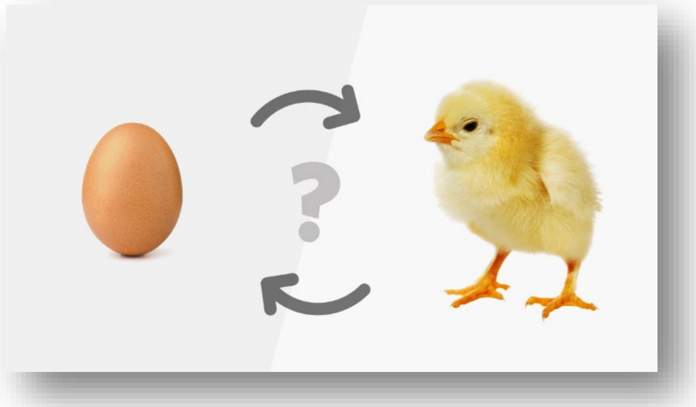
Symmetric Cryptosystems

Symmetric encryption and decryption can be performed **very fast**

If good, long, keys are used (ideally so-called „one-time pads“), this method is **very secure**

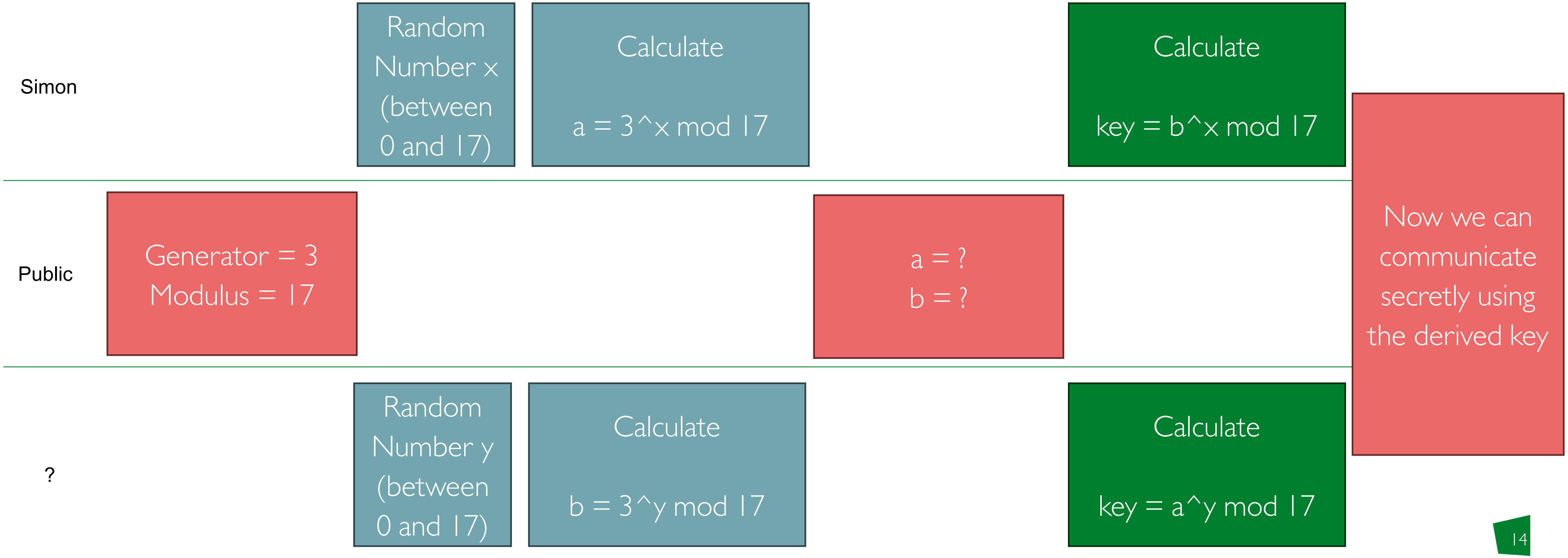
But the keys needs to be shared first

But **how**?

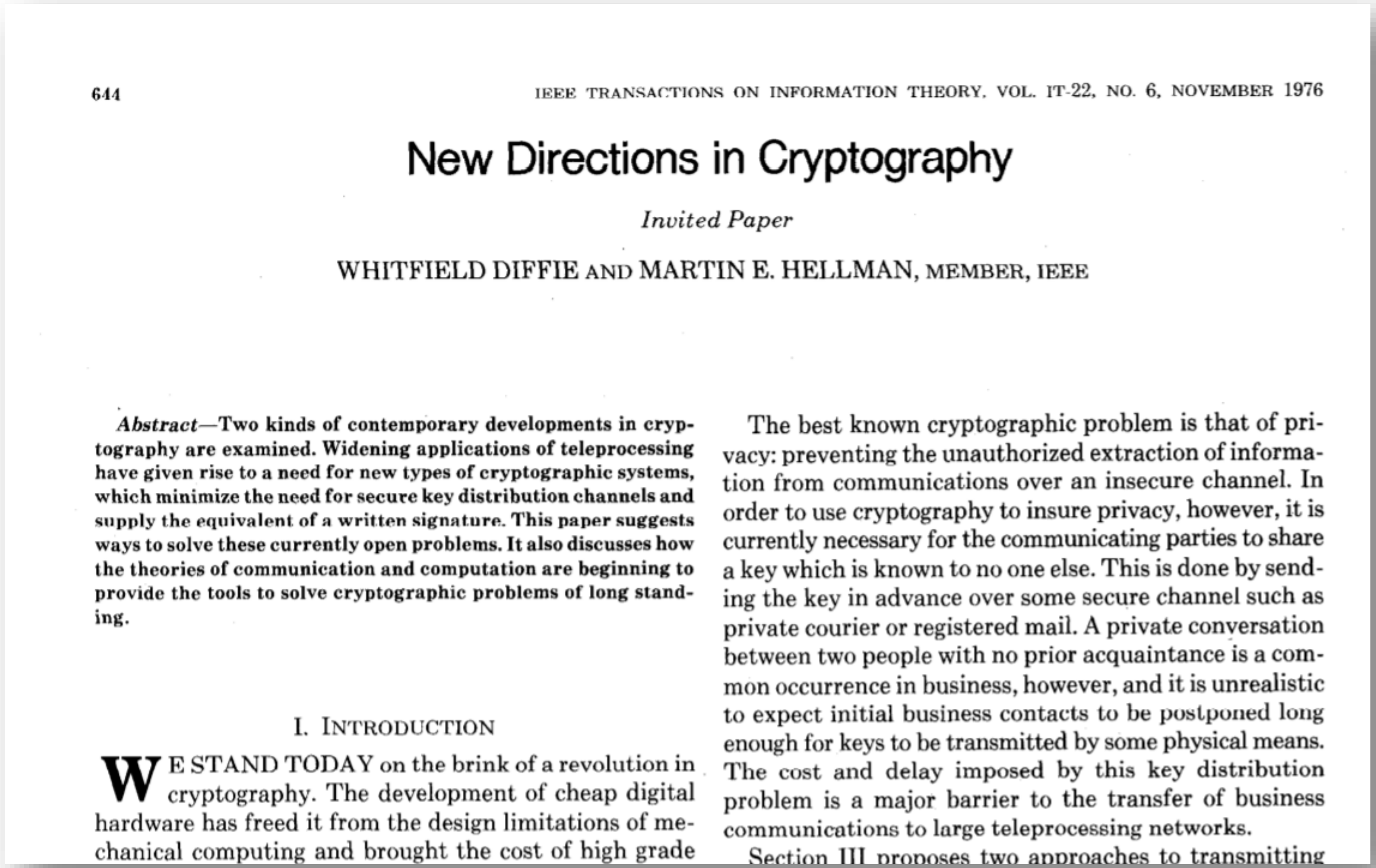


Some Magic ☺

I'll agree on a **shared secret number** (i.e., a key) with one of you even though we only communicate through the **public lecture hall**


Simon
Public
?
Random Number x (between 0 and 17)
Calculate a = 3^x mod 17
Calculate key = b^x mod 17
Generator = 3 Modulus = 17
a = ? b = ?
Random Number y (between 0 and 17)
Calculate b = 3^y mod 17
Calculate key = a^y mod 17
Now we can communicate secretly using the derived key

No Magic ☺



644 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

***Abstract*—Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.**

I. INTRODUCTION

WE STAND TODAY on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade

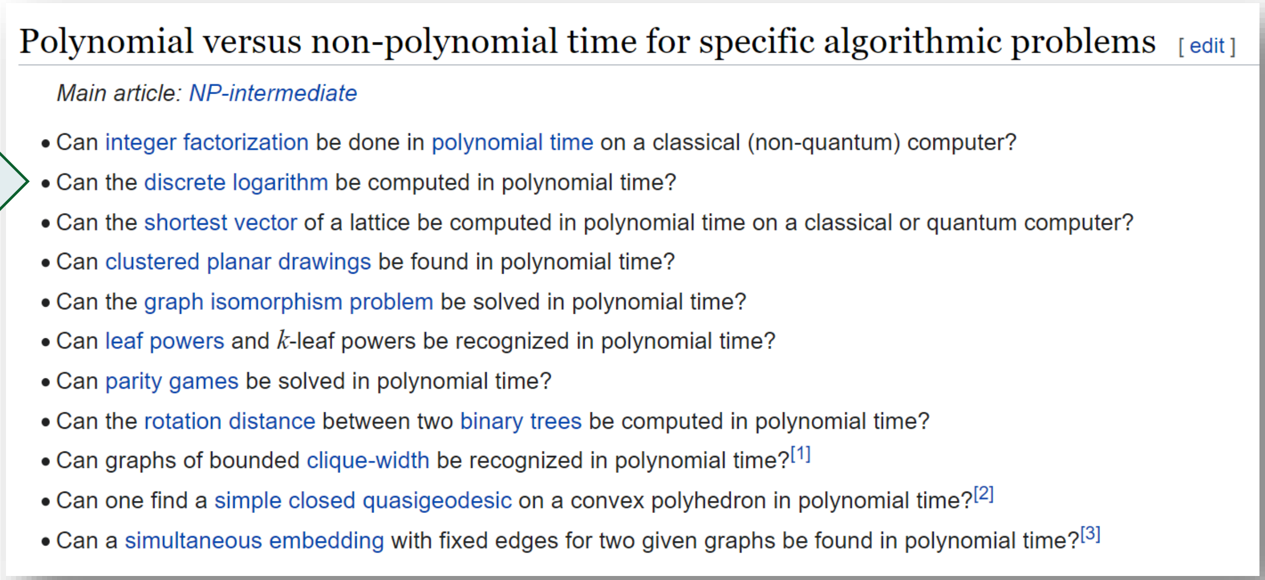
The best known cryptographic problem is that of privacy: preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such as private courier or registered mail. A private conversation between two people with no prior acquaintance is a common occurrence in business, however, and it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by some physical means. The cost and delay imposed by this key distribution problem is a major barrier to the transfer of business communications to large teleprocessing networks.

Section III proposes two approaches to transmitting

https://www.youtube.com/watch?v=YEBfamv-_do from 2:08

https://networklessons.com/uncategorized/diffie-hellman-key-exchange-explained/

One-way Functions: Discrete Logarithm


Polynomial versus non-polynomial time for specific algorithmic problems [edit]

Main article: NP-intermediate

- Can integer factorization be done in polynomial time on a classical (non-quantum) computer?
- Can the discrete logarithm be computed in polynomial time?
- Can the shortest vector of a lattice be computed in polynomial time on a classical or quantum computer?
- Can clustered planar drawings be found in polynomial time?
- Can the graph isomorphism problem be solved in polynomial time?
- Can leaf powers and k-leaf powers be recognized in polynomial time?
- Can parity games be solved in polynomial time?
- Can the rotation distance between two binary trees be computed in polynomial time?
- Can graphs of bounded clique-width be recognized in polynomial time?[1]
- Can one find a simple closed quasigeodesic on a convex polyhedron in polynomial time?[2]
- Can a simultaneous embedding with fixed edges for two given graphs be found in polynomial time?[3]

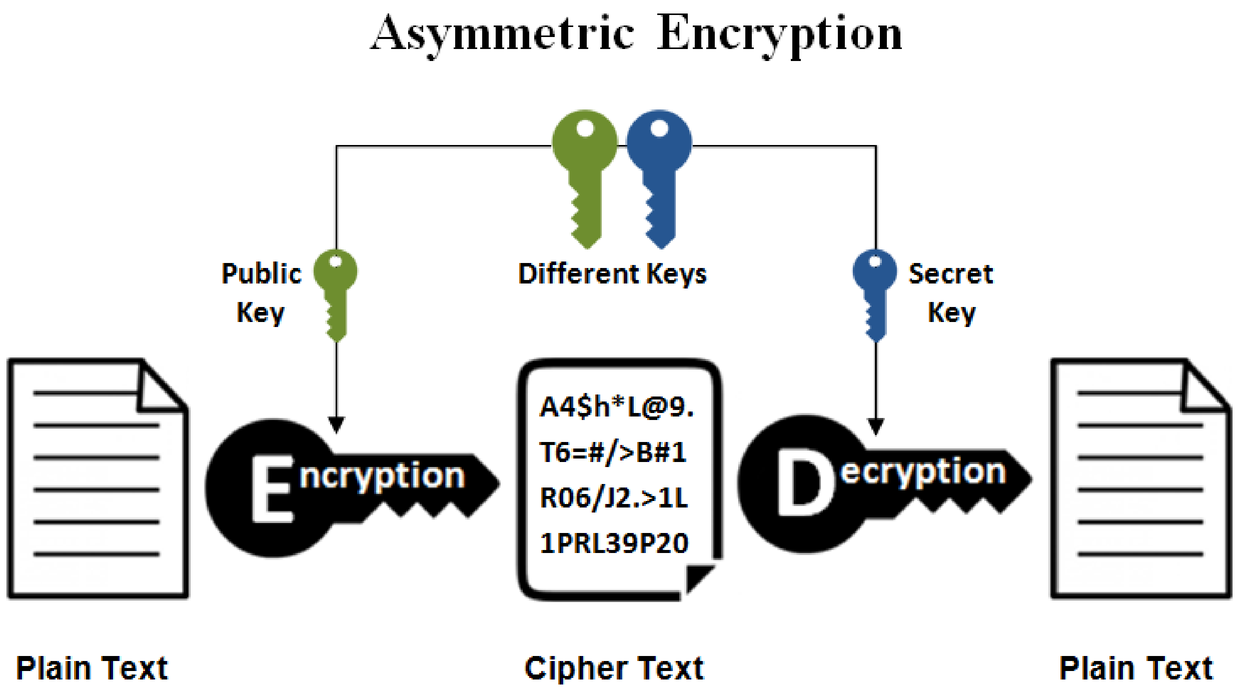
https://en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science

$$8 = 2^x \pmod{13}$$

In CS, one-way functions are frequently used as «locks»

Cryptographic Hash Functions are one-way functions

Waaay too much for us to treat in this lecture…



https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences

If you're interested, have a look at the following two slides and the linked information!

If you **know the public key** of some entity (a server, a human user...) then you can establish a **secured data tunnel** with that entity (e.g. with HTTPS/TLS in your browser)

TLS Features

Setup

- The communicating parties use **symmetric encryption** to encrypt the transmitted data
- The **encryption keys** are **generated from a shared secret** and are unique for each connection
- The shared secret, encryption algorithm, and keys are **negotiated** at the start of each session ("TLS Handshake"), based on protocols such as DH
- Each entity has a **private and a public key**; these are mathematically coupled to get the following properties
- Each entity **advertises its public key** and **keeps the private key private**

Main Feature #1: Secure Connections

- The encrypting entity **encrypts messages** with the **receiver's public key**
- The receiver **decrypts messages** with its **own private key**

Main Feature #2: Authenticated Identities (usually: optional for the client but required for the server)

- The authenticating entity **signs messages** with its **private** key
- The authenticating entity also **advertises** its **public** key

Do you see any problem here?

- How do we know that the advertised public keys are correct?
- If we don't, so-called **machine-in-the-middle attacks** become feasible!
- Solution: The authenticity of the public keys are certified by a trusted third party called a **certification authority**

Asymmetric Encryption and Public Key Infrastructures (PKIs)

If you **know the public key** of some entity (a server, a human user...) then you can establish a **secured data tunnel** with that entity (e.g. with HTTPS/TLS in your browser)

But you need to be **sure** that the key you use actually belongs to that entity!

PKIs are used to **verify that someone actually is who they claim to be** and that they have the private key associated with a specific public key.

This moves the key distribution problem to **distributing keys of certification authorities** (instead of keys of users) and users trusting these authorities.

To enable this, „Root Certification Authorities" are **already included** in our operating systems and browsers.

Here is more (quality-controlled) background material about this: https://en.wikipedia.org/wiki/Public_key_infrastructure

This Week's Menu

Motivation & Web APIs
Computer Networks Basics
The Networking Stack
The World Wide Web
Briefing: Security on the Web

Done ☺

1. A Web / HTTP Server
2. A Web / HTTP Client

I'm showing demos (available on Canvas) **locally on my machine.** You can run them as well if you want.

1. Install a **python interpreter.** This is available and described in https://www.python.org/downloads/
2. Put code into **.py files** and then run it from your **Terminal** (e.g., PowerShell on Win or Terminal on Mac)
3. Download FCS-BWL-06-DistributedSystems-Demos.zip from Canvas

demos/*dynamic_webserver*

A Basic Web / HTTP Server

We have a python script that reads out the **current system time** and prints it on the screen

```
from datetime import datetime

def getCurrentDateTime():
        today = datetime.now()
        return str(today)
```

https://docs.python.org/3/library/datetime.html

We want to serve that functionality over the Web…

https://docs.python.org/3/library/http.server.html

If you can do this, this means that you can serve *any* **functionality that you can express in Python** over the Web (including, for instance, a robot controller ☺)

This Week's Menu

Motivation & Web APIs
Computer Networks Basics
The Networking Stack
The World Wide Web
Briefing: Security on the Web

Done ☺

1. A Web / HTTP Server
2. **A Web / HTTP Client**

I'm showing demos (available on Canvas) **locally on my machine.** You can run them as well if you want.

1. Install a **python interpreter.** This is available and described in https://www.python.org/downloads/
2. Put code into **.py files** and then run it from your **Terminal** (e.g., PowerShell on Win or Terminal on Mac)
3. Download FCS-BWL-06-DistributedSystems-Demos.zip from Canvas

A Basic Web / HTTP Client

```
# The requests library is most often used in HTTP clients
import requests

# Prompt for a pokemon name
pokemon = input("What is your favorite Pokemon? ")

# Define the API endpoint (for example, data about Pikachu)
url = "https://pokeapi.co/api/v2/pokemon/" + pokemon

# Send a GET request to the API
response = requests.get(url)

# Check that the request was successful
if response.status_code == 200:
    data = response.json()  # Parse JSON response
    print("Name:", data["name"])
    print("Base experience:", data["base_experience"])
    print("Abilities:")
    for ability in data["abilities"]:
        print("-", ability["ability"]["name"])
else:
    print("Failed to retrieve data:", response.status_code)
```

Library Import

Prompt User

Create correct URL

Make HTTP request

Check that request was successful

Parse HTTP response (response is in JSON format)

Cool that JSON can be directly fed into your favorite python structure ☺

Another (hidden) HTTP Client

```
# Import the OpenAI package
from openai import OpenAI

# Replace this with your API key
client = OpenAI(api_key="<put-your-key>")

# Formulate the request and send it
completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": 'You are a professor of computer science explaining how to use the ChatGPT python API.’},
    {"role": "user", "content": 'Greet the students who are waiting for their practical exercise!’}
  ]
)

# Extract the response
response = completion.choices[0].message.content

# Print it
print(response)
```

This is a custom program that accesses GPT.

„Underneath“, it uses an HTTP API (see the Documentation).

Any Questions / Comments / Doubts / Concerns?

This was basic HTTP-based interaction with Web services

This week's assignment is the same in principle, but with a few added complications

Play around and experiment with this. Build your own servers and clients.

And use it in your project ☺