Distributed Systems
Lecture 9

Secure Sockets and JSON

Joseph Phillips

Last modified 2019 June 2

Reading

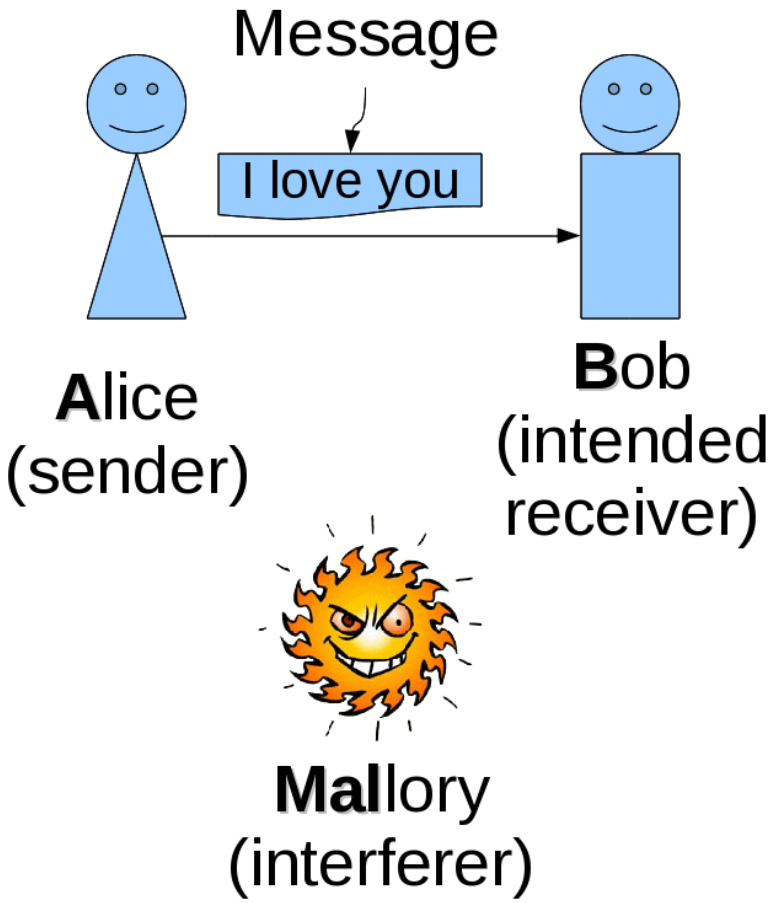
- Chapter 10 “*Secure Sockets*” of Elliotte Rusty Harold “*Java Network Programming: 4th Ed.*”

Topics

- Secure sockets in Java
- JSON

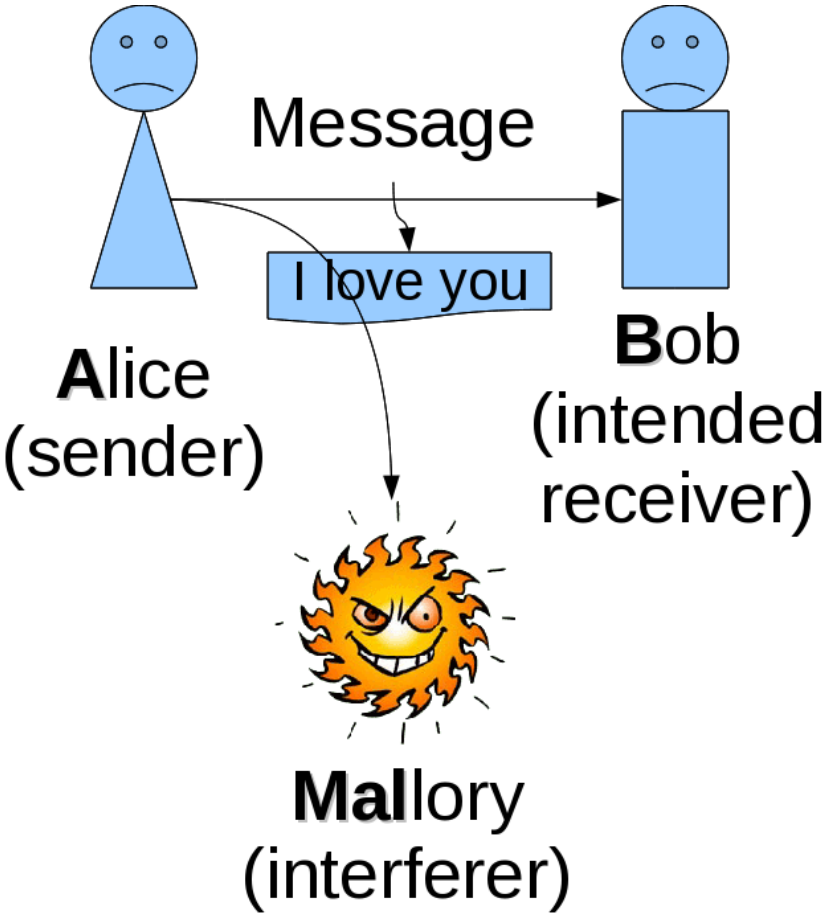
Life without secure sockets: How bad code it be?

- Introducing our cast of characters:
- ***Alice***: the ***legitimate sender*** of a message
- ***Bob***: the ***legitimate receiver*** of a message
- ***Mallory***: the one trying to ***muck-up communication*** from Alice to Bob



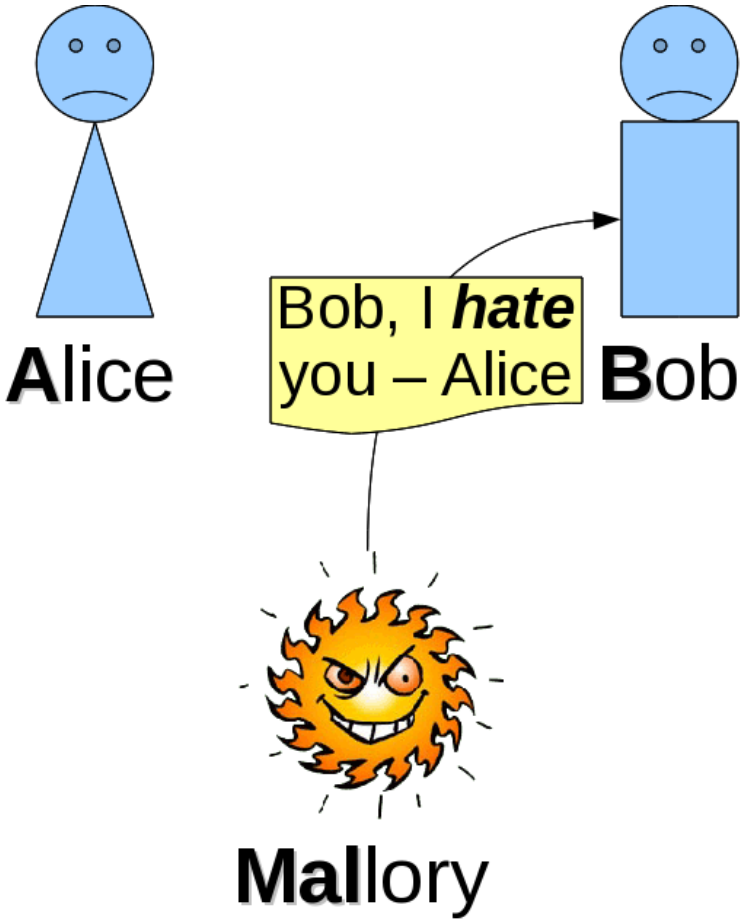
Life without secure sockets: How bad code it be?

- The ***eavesdropping attack***
  - The bad guy (Mallory) ***reads your message***



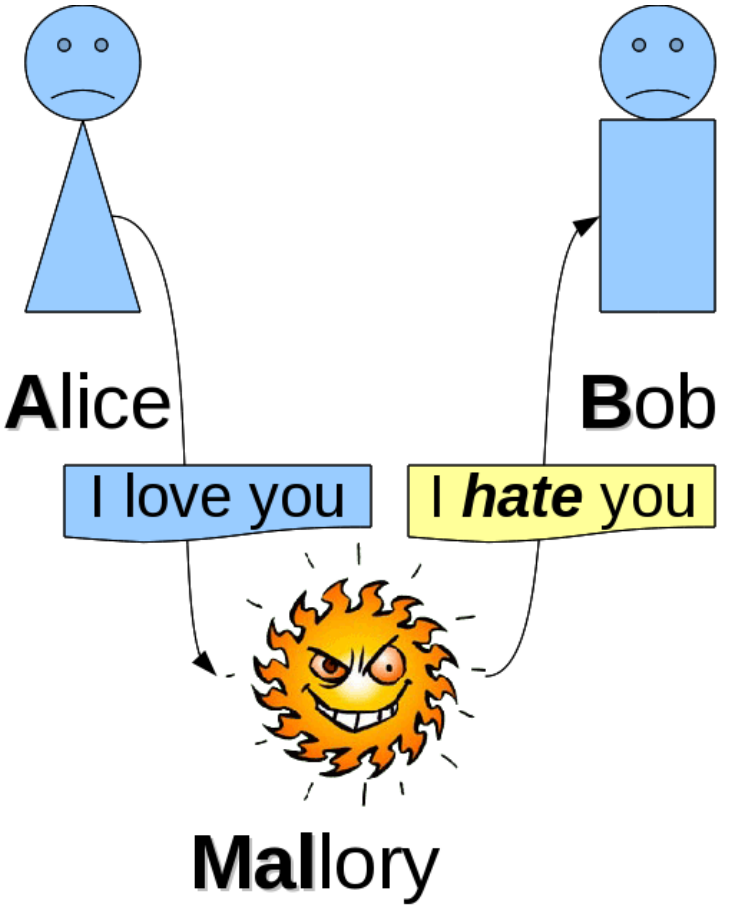
Life without secure sockets: How bad code it be?

- The ***masquerading attack***
  - The bad guy (Mallory) ***pretends to be someone else*** and ***fabricates a message***



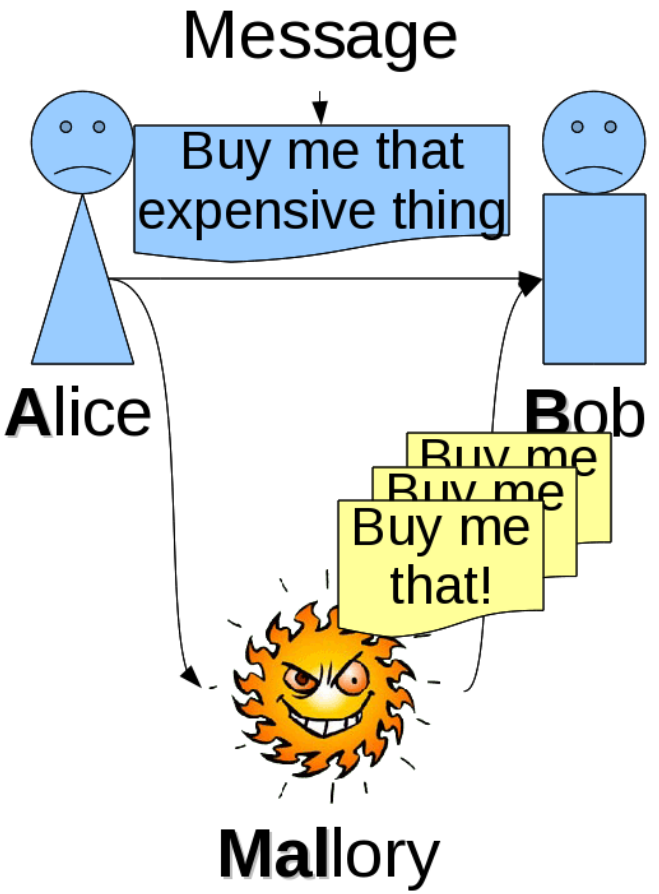
Life without secure sockets: How bad code it be?

- The ***tampering attack***
  - The bad guy (Mallory) ***fabricates changes a legitimate message***



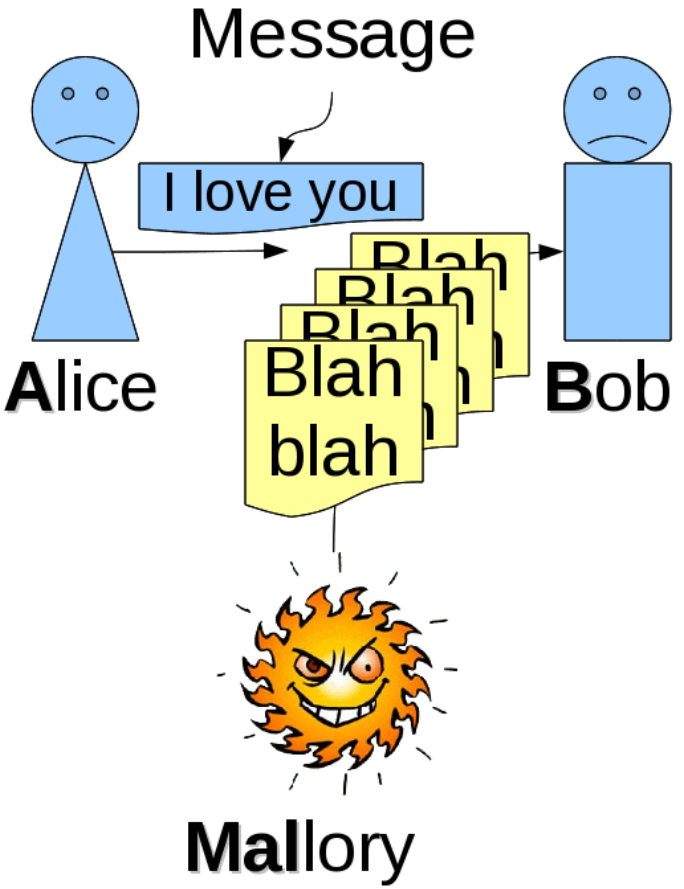
Life without secure sockets: How bad code it be?

- The ***replay attack***
  - The bad guy (Mallory) ***copies a legitimate message and sends it multiple times***



Life without secure sockets: How bad code it be?

- The ***denial of service attack***
  - The bad guy (Mallory) ***sends a lot of crap so legitimate messages cannot get through***



Motivation (SSL)

We can communicate client-to-server

We can communicate server-to-client

But how do we do so ***securely?***

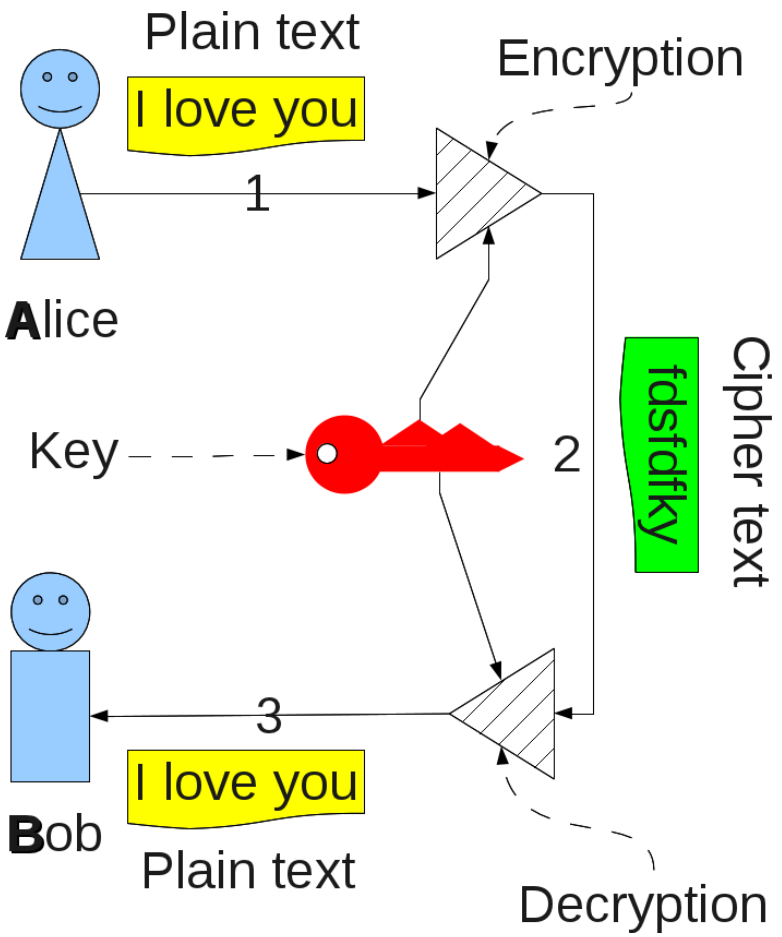
Answer: With ***secure sockets!***

Ideally we would like:

- ***Secrecy***: no one else knows what message says
  - Thwarts which attack?
- ***Integrity***: detect when message was tampered with
  - Thwarts which attack?
- ***Authentication***: know who we're talking too
  - Thwarts which attack(s)?
- ***Non-repudiation***: know who's talking to us
  - Thwarts which attack(s)?

Some of our main tools

- Symmetric key encryption
  - Pros: ***fast***
  - Cons: ***must keep key secret!***



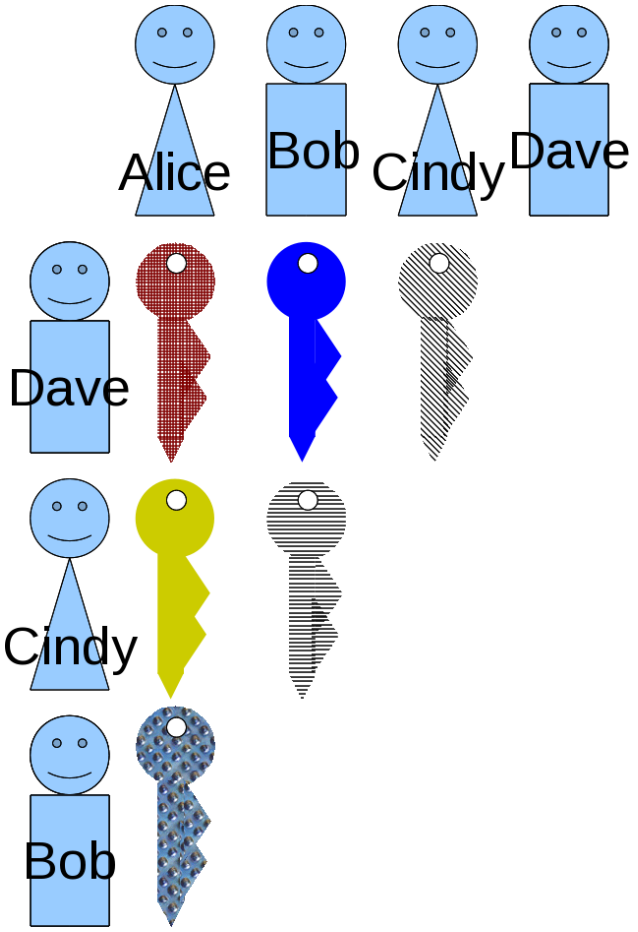
Problems with symmetric keys cont'd

- What if Alice and Bob have never met?

  Require private coordination to share keys

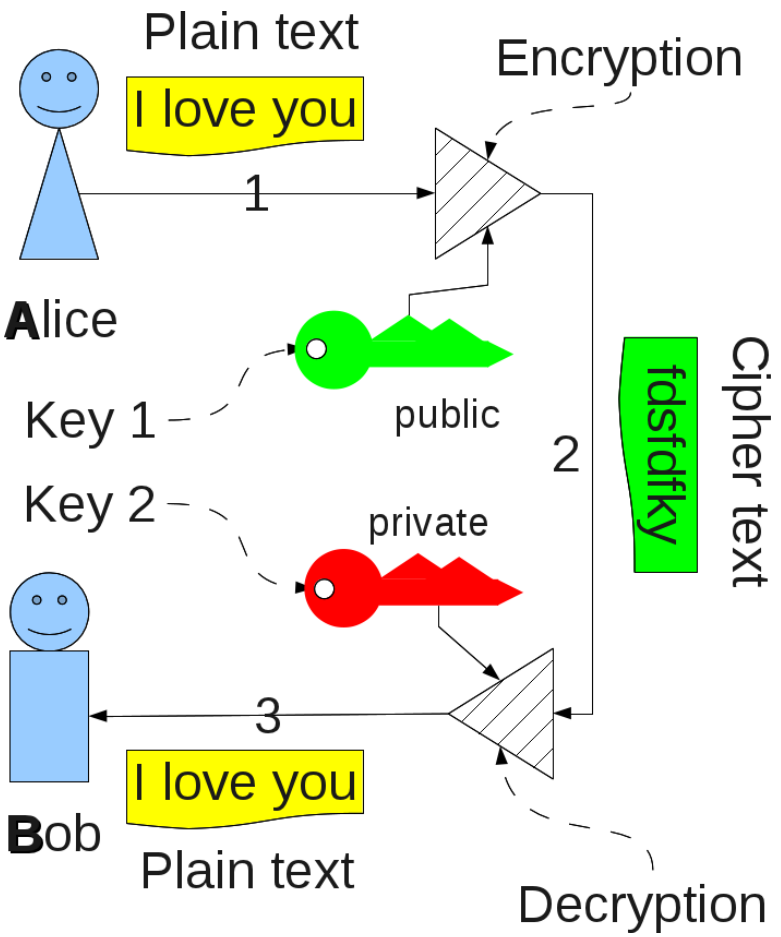
- N people?

  Need N*(N-1)/2 keys!



Some of our main tools

- Asymmetric key encryption
- Encrypt with one key (e.g. “public”)
- Decrypt with another (e.g. “private”)
  - Pros: ***You are encouraged to share the public key***
  - So more people can securely talk to you
  - Cons: ***Slow!***



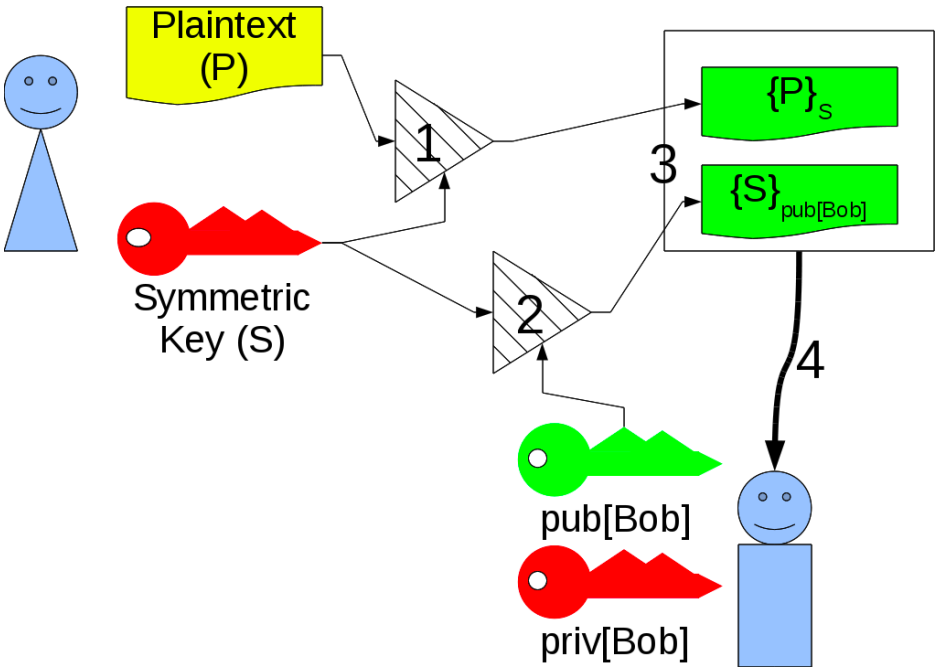
Encrypting Documents

(1) Alice encrypts her plain text with a symmetric key

- It's quick!

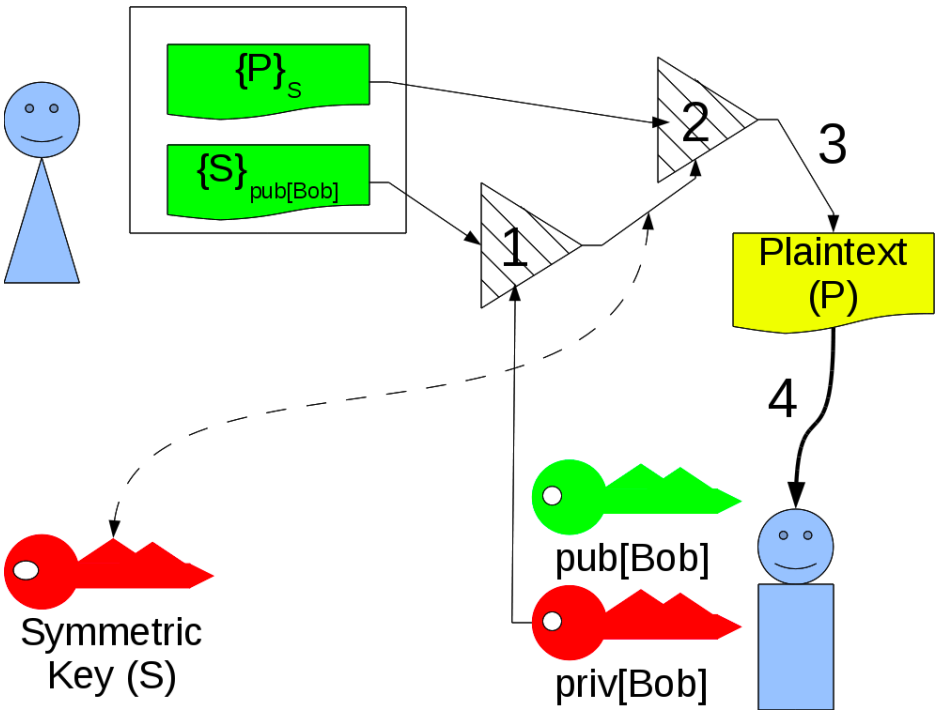
(2) Then the encodes her symmetric key with Bob's public key

- Asymmetric encryption is slow, but we are only encrypting a relatively short key, not the whole message



Decrypting Documents

- Bob decrypts Alice's symmetric key with his private key
- Bob then decrypt's Alice's message with the decrypted symmetric key

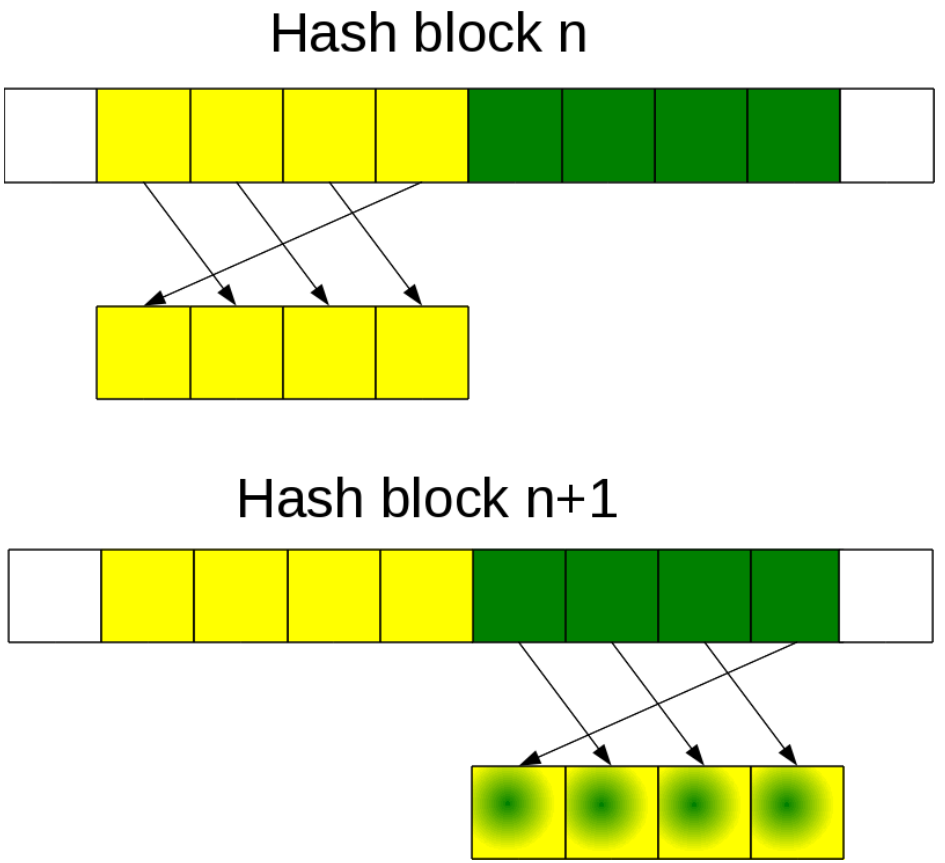


Your turn!

But we are still vulnerable?
To which attacks?

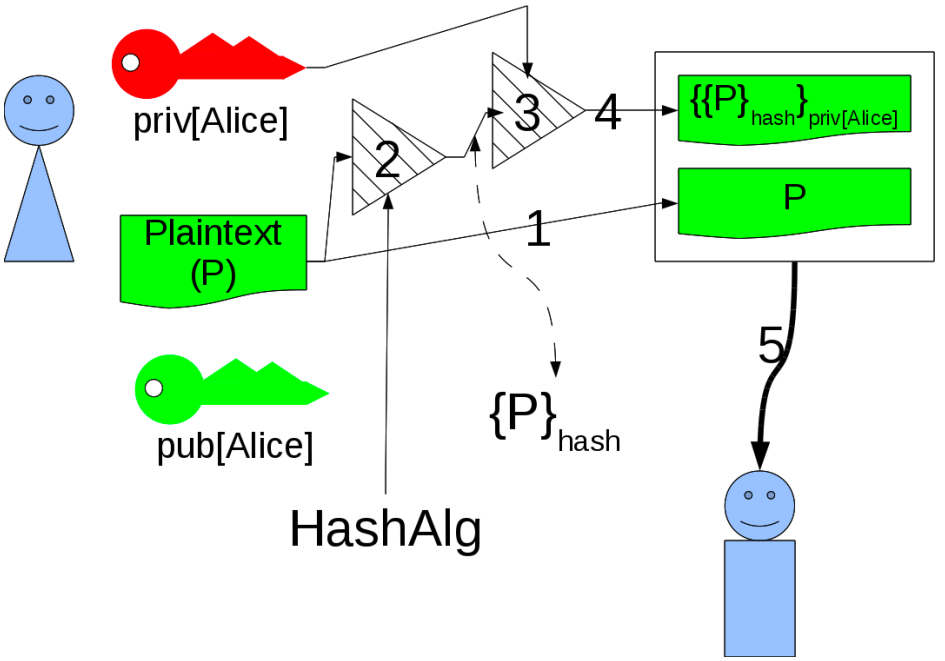
Some of our main tools

- Cyprotographic hash



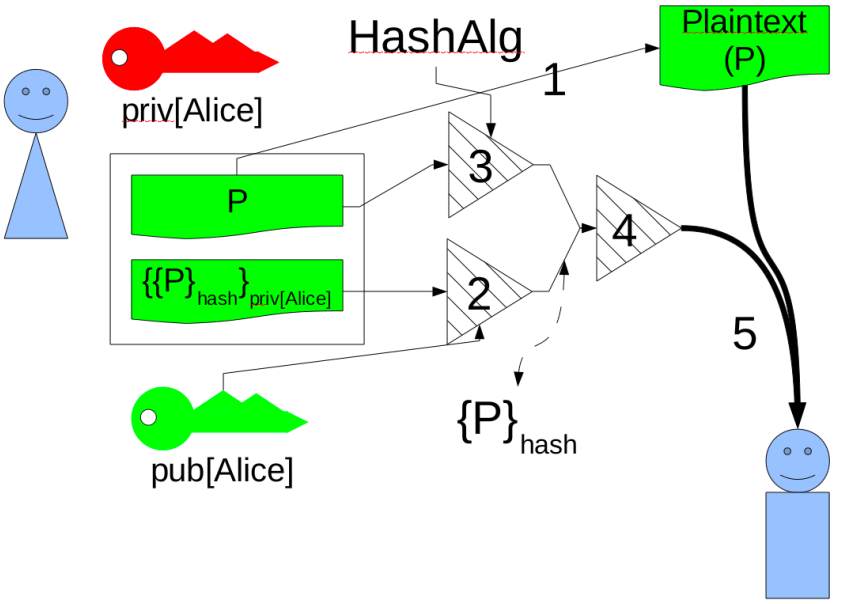
Signing documents

- Alice makes a hash of her plain text
- Then she encrypts this with her private key



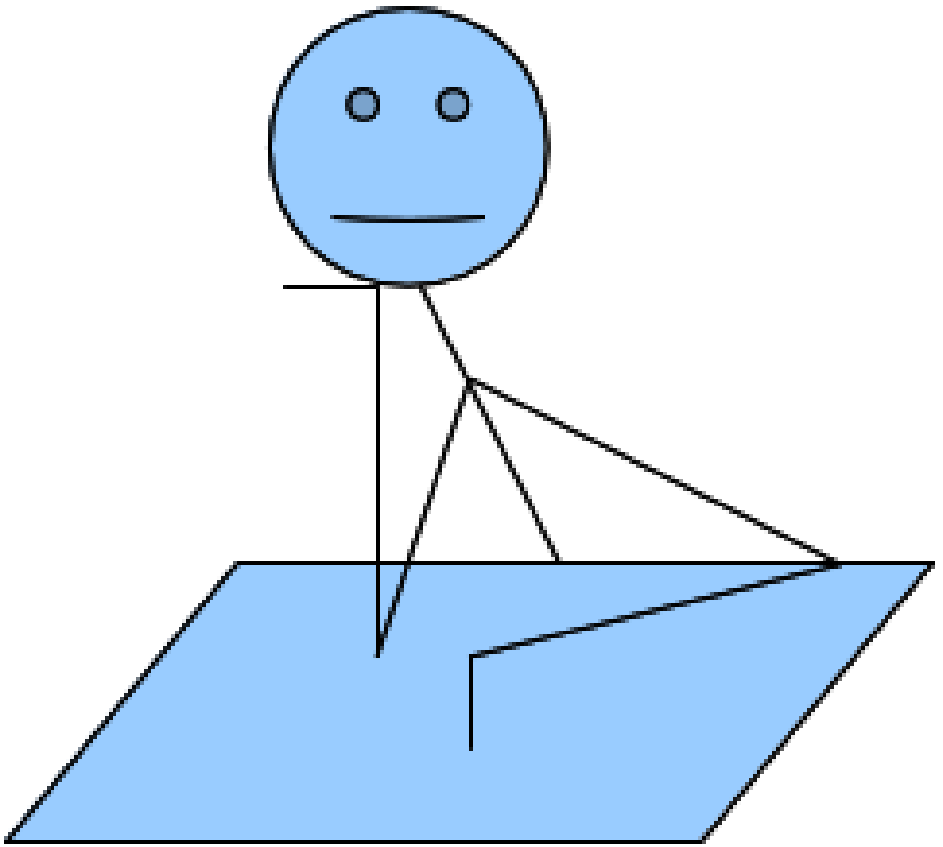
Signing documents

- After Bob decrypts the plaintext, he hashes it too
- He then decrypts the encrypted hash value using Alice's public key
- If they match, we assume Alice signed it



Excited Student

- Sounds nifty! How do we do it?



With Secure Sockets!

- All the fun of regular sockets . . .
- . . . but encrypted too!

SSLSocketFactory

- Getting an SSLSocketFactory
  - SocketFactory factory = SSLSocketFactory.getDefault();
  - Throws InstantiationException if find no appropriate class
- SSLSocketFactory methods
  - public abstract Socket createSocket(String host, int port)
  - public abstract Socket createSocket(InetAddress host, int port)
  - public abstract Socket createSocket(String host, int port, InetAddress interface, int localPort)
  - public abstract Socket createSocket(InetAddress host, int port, InetAddress interface, int localPort)
  - public abstract Socket createSocket(Socket proxy, String host, int port, boolean autoCreate)

HttpsClient.java

```
import  java.io.*;
import  javax.net.ssl.*;

public class HttpsClient
{
  public static void main (String[] args)
  {
    if  (args.length == 0)
    {
      System.out.println("Usage:\tjava HttpClient <host>");
      System.exit(1);
    }

    int         port    = 443;  // Default HTTPS port
    String      host    = args[0];

    SSLSocketFactory    factory = (SSLSocketFactory)
                                  SSLSocketFactory.getDefault();
    SSLSocket           socket  = null;

    try
    {
      socket    =
(SSLSocket)factory.createSocket(host,port);

      // Enable all the suites
      String[]  supported= socket.getSupportedCipherSuites();
      socket.setEnabledCipherSuites(supported);

      Writer    out = new
OutputStreamWriter(socket.getOutputStream(),"UTF-8");

      // https requires the full URL in the GET line
      out.write("GET http://" + host + "/ HTTP/1.1\r\n");
      out.write("HOST: " + host + "\r\n");
      out.write("\r\n");
      out.flush();

      // read() response
      BufferedReader    in = new BufferedReader
                        (new InputStreamReader
                              (socket.getInputStream()
                              )
                        );

      // read() the header
      String s;

      while  ( !(s = in.readLine()).equals("") )
      {
        System.out.println(s);
      }
```

HttpsClient.java

```
System.out.println();

// read the length
String    contentLength   = in.readLine();
int       length          = Integer.MAX_VALUE;

try
{
  length  = Integer.parseInt(contentLength.trim(),16);
}
catch  (NumberFormatException ex)
{
  //  This server does not send the content-length
  //  in the first line of the response body
}

System.out.println(contentLength);

int       c;
int       i       = 0;

while  ( (c = in.read()) != -1 && i++ < length)
{
  System.out.write(c);
}
  System.out.println();
 }
 catch  (IOException ex)
 {
  System.err.println(ex);
 }
 finally
 {
  try
  {
   if  (socket != null)
   {
    socket.close();
   }
  }
  catch  (IOException e)
  {
  }
 }
}
}
```

JSON: Motivation

- Now that we can send data:
  - client-to-server
  - server-to-client
  - high level with http
  - low level with sockets
  - securely (if need be) with SSL
- What should we say?
  - Text for humans? ==> text or html or PDF
  - Images for humans? ==> gif, jpeg or png

JSON: Motivation

- All very nice, but what if the ***software*** should understand what it being said too?
- **XML** (example from https://www.w3schools.com/js/js_json_xml.asp)

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
<lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName>
<lastName>Jones</lastName>
  </employee>
</employees>
```

JSON: Motivation

- All very nice, but what if the ***software*** should understand what it being said too?
- **JSON**

```
{"employees":[
  {  "firstName":"John", "lastName":"Doe" },
  {  "firstName":"Anna", "lastName":"Smith" },
  {  "firstName":"Peter", "lastName":"Jones" }
]}
```

- JSON is:
  - Shorter
  - Easier to parse

JSON

- (From www.json.org)
- Lightweight data-interchange format
- JSON is ***built on two structures***:
- A collection of ***name/value pairs***.
  - In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ***ordered list of values***.
  - In most languages, this is realized as an array, vector, list, or sequence.

Java JSON

- (From https://www.geeksforgeeks.org/parse-json-java/)
- JSONExample.json

```
{
      "lastName":"Smith",
     "address":{
         "streetAddress":"21 2nd Street",
          "city":"New York",
          "state":"NY",
          "postalCode":10021
     },
      "age":25,
      "phoneNumbers":[
             {
             "type":"home", "number":"212 555-1234"
             },
         {
             "type":"fax", "number":"212 555-1234"
         }
      ],
      "firstName":"John"
}
```

Java JSON

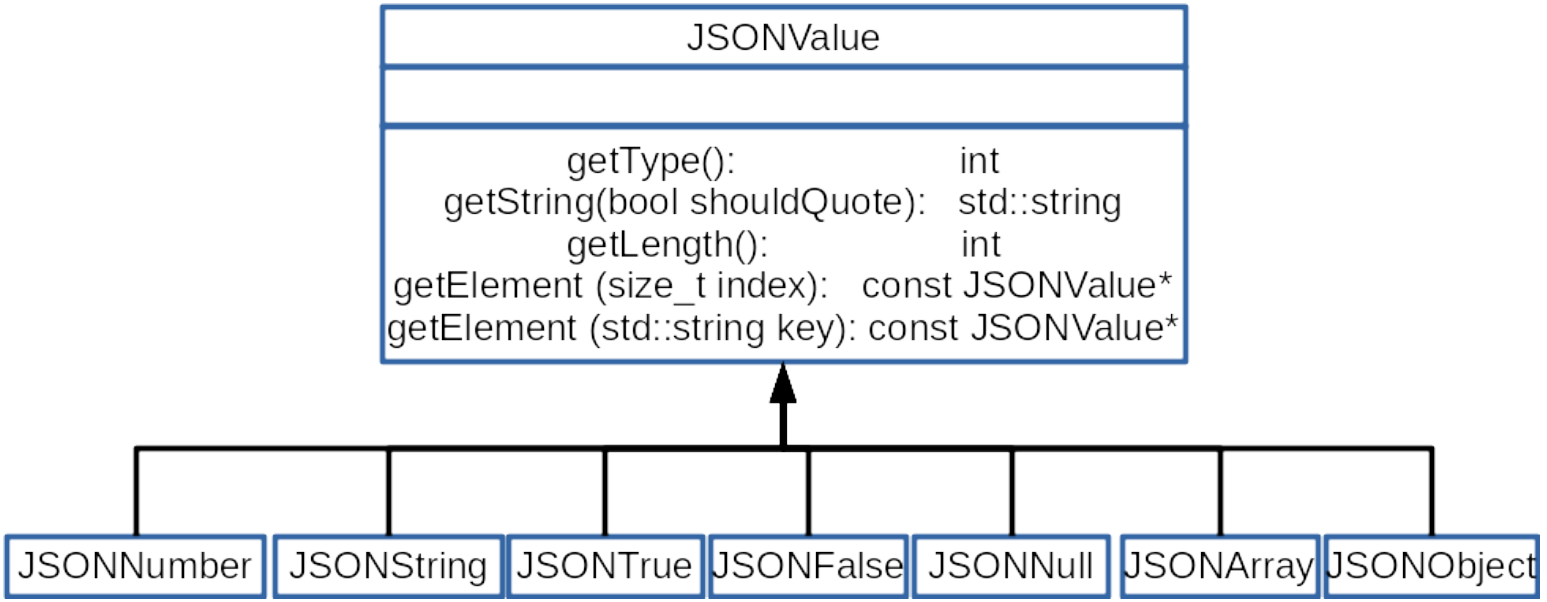
```
// Java program for write JSON to a file

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.LinkedHashMap;
import java.util.Map;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;

public class JSONWriteExample
{
  public static void main(String[] args)
         throws FileNotFoundException
  {
    // creating JSONObject
    JSONObject jo = new JSONObject();

    // putting data to JSONObject
    jo.put("firstName", "John");
    jo.put("lastName", "Smith");
    jo.put("age", 25);

    // for address data, 1st create LinkedHashMap
    Map m = new LinkedHashMap(4);
    m.put("streetAddress", "21 2nd Street");
    m.put("city", "New York");
    m.put("state", "NY");
    m.put("postalCode", 10021);

    // putting address to JSONObject
    jo.put("address", m);

    // putting address to JSONObject
    jo.put("address", m);

    // for phone numbers, first create JSONArray
    JSONArray ja = new JSONArray();

    m = new LinkedHashMap(2);
    m.put("type", "home");
    m.put("number", "212 555-1234");

    // adding map to list
    ja.add(m);

    m = new LinkedHashMap(2);
    m.put("type", "fax");
    m.put("number", "212 555-1234");

    // adding map to list
    ja.add(m);

    // putting phoneNumbers to JSONObject
    jo.put("phoneNumbers", ja);

    // writing JSON to file:"JSONExample.json"
    PrintWriter pw =
        new PrintWriter("JSONExample.json");
    pw.write(jo.toJSONString());

    pw.flush();
    pw.close();
  }
}
```

Java JSON

```
// Java program to read JSON from a file

import java.io.FileReader;
import java.util.Iterator;
import java.util.Map;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.*;

public class JSONReadExample
{
  public static void main(String[] args)
      throws Exception
  {
    // parsing file "JSONExample.json"
    Object obj = new JSONParser().
      parse(new FileReader("JSONExample.json"));

    // typecasting obj to JSONObject
    JSONObject jo = (JSONObject) obj;

    // getting firstName and lastName
    String firstName =
      (String)jo.get("firstName");
    String lastName = (String) jo.get("lastName");

    System.out.println(firstName);
    System.out.println(lastName);

    // getting age
    long age = (long) jo.get("age");
    System.out.println(age);

    // getting address
    Map address = ((Map)jo.get("address"));

    // iterating address Map
    Iterator<Map.Entry> itr1 =
        address.entrySet().iterator();
    while (itr1.hasNext()) {
      Map.Entry pair = itr1.next();
      System.out.
        println(pair.getKey()+" : "+pair.getValue());
    }

    // getting phoneNumbers
    JSONArray ja = (JSONArray)jo.get("phoneNumbers");

    // iterating phoneNumbers
    Iterator itr2 = ja.iterator();

    while (itr2.hasNext())
    {
      itr1 = ((Map)itr2.next()).entrySet().iterator();
      while (itr1.hasNext()) {
      Map.Entry pair = itr1.next();
      System.out.
        println(pair.getKey()+" : "+pair.getValue());
      }
    }
  }
}
```

C++ JSON Library
(courtesy Applied Philosophy of Science)

- Base class: JSONValue
  - Subclasses: JSONNumber, JSONString, JSONTrue, JSONFalse, JSONNULL, JSONArray, JSONObject



JSONTrue, JSONFalse, JSONNNull

- Straight forward
- Constructors take no arguments
  - `JSONTrue()`, `JSONFalse()`, `JSONNNull()`
- Only methods (belongs to all classes)
  - std::string getString (bool shouldQuote)
    - Prints *this, with either strings quoted or not
- getType()
  - Returns `TRUE_JSON`, `FALSE_JSON`, or `NULL_JSON` respectively

JSONString

- Represents strings, or base objects
- Constructors:
  - `JSONString(const char* cPtr)`
  - `JSONString(std::string& str)`
- getType()
  - Returns `STRING_JSON`
- getString(bool shouldQuote)
  - Strings can either be quoted or unquoted

JSONNumber

- Represents numbers
- Constructors:
  - `JSONNumber (const std::string& text, int newInteger)`
  - `JSONNumber (const std::string& text, float newFloat)`
  - `JSONNumber (int newInteger)`
  - `JSONNumber (float newFloat)`
- Accessors:
  - `int getInteger ();`
  - `float getFloat ();`
  - `bool isInteger (int& integer, float& real);`
    - For integers: returns ‘true’ and sets ‘integer’
    - For floats: returns ‘false’ and sets ‘real’

JSONArray

- Holds arrays (silly!)
- Constructor:
  - JSONArray()
- Methods:
  - size_t getLength(); // How long array is
  - const JSONValue* getElement(size_t index);
    - // Get element at index, or NULL if at or beyond length
  - void add (JSONValue* jsonElePtr) // Adds at end
- getType()
  - Returns ARRAY_JSON

JSONObject

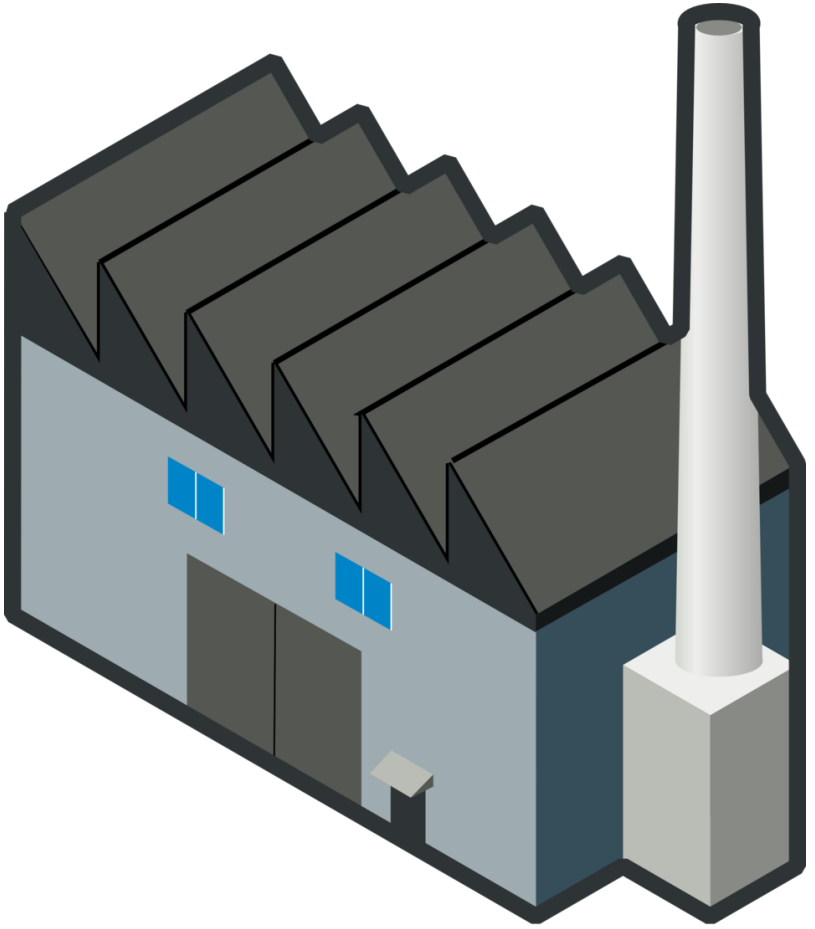
- Holds arbitrary objects
- Constructor:
  - JSONObject()
- Methods:
  - size_t getLength(); // How many (key,value) pairs are in *this
  - const JSONValue* getElement(const std::string& key);
    - // Get element indexed by key, or NULL if none
  - void add (const std::string& key, JSONValue* jsonElePtr)
    - // Adds jsonElePtr at key
- `getType()`
  - Returns `OBJECT_JSON`

Factory

- A public static method

```
JSONValue*
JSONValue::factory
(int  fd,
 bool shouldCloseFd
)
```

  - `fd`: File descriptor from which to read definition
  - `shouldCloseFd`: if true then will `close(fd)` after finished



Your turn!

Write a program to read a JSON definition and

References:

- Elliotte Rusty Harold “*Java Network Programming: 4th Ed.*”
- https://www.geeksforgeeks.org/parse-json-java/
- https://www.w3schools.com/js/js_json_xml.asp
- https://www.json.org/

Special thanks to:

Applied Philosophy of Science