# PI-Grau
# (Internet Protocols)

José M. Barceló Ordinas
Departamento de Arquitectura de Computadores
(UPC)

- **Topic 2: Application and Services I.**
  - Objectives
    - Understand **Web Services** architectures
    - Identify the **main components of Secure Services** and be introduced in certificates and public keys

# Topic 2: Applications and Services.

- **Service** - a mechanism to enable **access** to one or more capabilities, where the access is provided using a prescribed **interface** and is exercised consistent with **constraints and policies** as specified by the service description

  OASIS SOA Reference Model

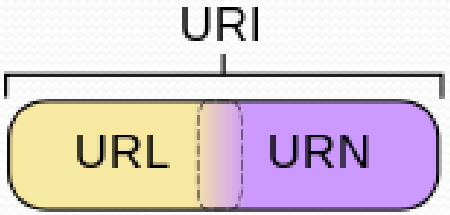
- **Service Oriented Architectures (SOA)** – a **paradigm** for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.
  - It provides a uniform means to offer, discover, interact with use capabilities to produce desired effects consistent with measurable preconditions and expectations.

  OASIS SOA Reference Model

## Topic 2: Applications and Services.

- **Distributed System:**
  - Consists of diverse, discrete **software agents** that must work together to perform some tasks.
  - The agents in a distributed system do not operate in the same processing environment, so they must communicate by hardware/software protocol stacks over a network.
  - SOA is a kind of distributed system.
- **Distributed Object Systems:**
  - Distributed object systems are distributed systems in which the semantics of object initialization and method invocation are exposed to remote systems by means of a proprietary or standardized mechanism to broker requests across system boundaries

- **URI (Uniform Resource Identification)**
  - is a string of characters used to identify a name of a web resource.
  - A **resource** can be anything that has identity. Familiar examples include an electronic document, an image, a service and a collection of other resources




URL: Uniform Resource Locator
URN: Uniform Resource Name

A URI is a URL, a URN or both

A **URL** is a URI that, in addition to identifying a web resource, specifies the means of acting upon or obtaining the representation: providing both the primary access mechanism, and the network "location".

the URL http://example.org/wiki/Main_Page refers to a resource identified as /wiki/Main_Page at location example.org.

A **URN** is a URI that identifies a resource by name, in a particular namespace.

The **URN** *urn:isbn:0-395-36341-1* is a URI that specifies the identifier system, but it does not specifies a location (URL)

- **Web Services Architecture (W3C working group)**
  - SOA is a generic term that accounts for any software architecture. If these softwares are in net → Web Services
  - **Web Services** provide a **standard** means of interoperating between different software applications, running on a variety of platforms and/or frameworks.
    - It allows, thus, the implementation of a SOA architecture or in other words, an implementor can build a SOA using Web Service standards.
    - Can be seen as application components
  - The **architecture** does not attempt to specify how Web Services are implemented, and imposes no restriction on how Web Services might be combined.

- **Web Services (in simple words)**
  - Let's imagine a Web page that needs a piece of software handle by other application (e.g. we construct a web page that includes the Barcelona weather forecast).
  - The Catalan Forecast Institute has an application that upon request gives you a map of today's weather in any town in Catalonia.
  - You build your Web page an automatically, every day, you issue a request to the Catalan Forecast Institute asking for the Barcelona weather's map.
  - Web services:
    - allows you **to re-use application components**,
    - allows you to **exchange data** between different applications and different platforms,
    - **solves interoperability** problems by giving different applications a way to link their data,

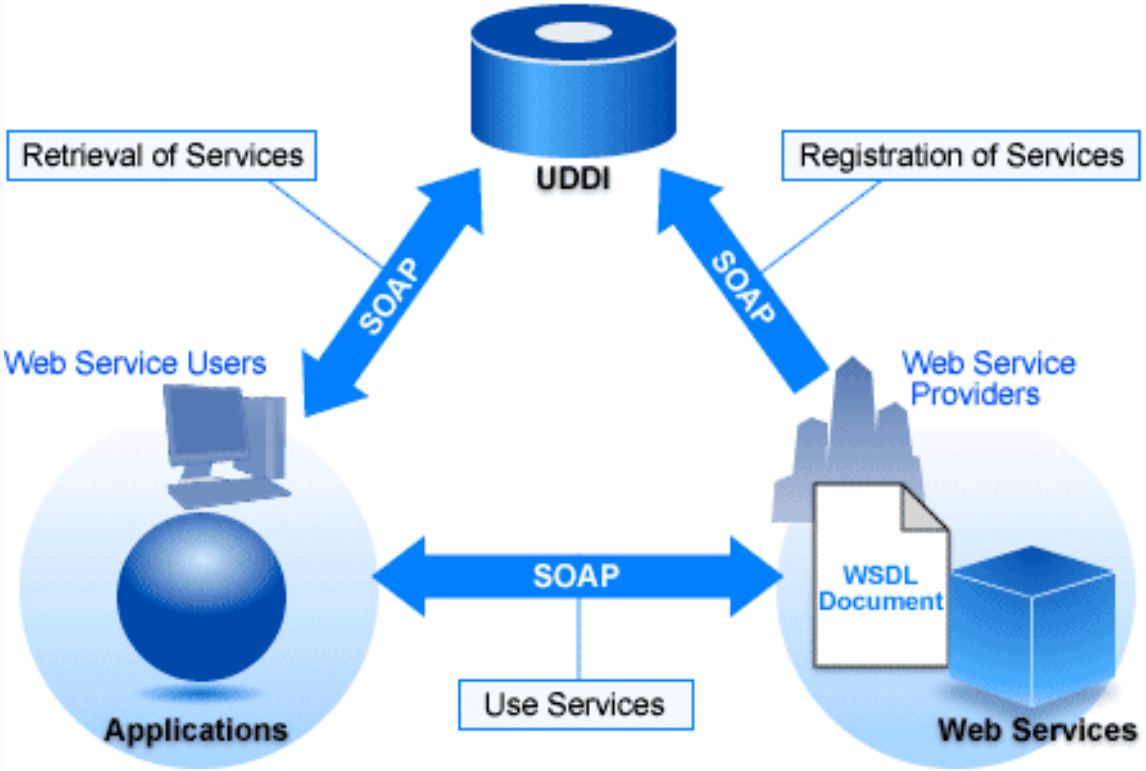
- **Web Services Architecture (W3C working group)**
  - **Two classes of Web Services:**
    - **REST-compliant Web services (REST: Representational State Transfer)**, in which the primary purpose of the service is to manipulate XML/JSON representations of Web resources using a uniform set of "stateless" operations;
    - and **arbitrary Web services**, in which the service may expose an arbitrary set of operations

- **REST (Representational State Transfer) architecture:**
  - Developed by W3C, consists on clients (request services) and servers (response that are built on transfers of representation of resources)
  - The transfer of representation of resources consists on a document with the state of the resource.
  - REST uses XML or JSON as languages and typically uses HTTP methods for transfers (similar to CRUD: Create/Read/Update/Delete in databases) of resources
    - POST (create or Insert)
    - GET (read or Select)
    - PUT (Update)
    - DELETE (delete)

- **XML Web Services Architecture (W3C working group)**
  - **Reference architecture and definition:**
    - A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
    - It has an interface described in a machine-processable format (specifically **WSDL** - Web Services Description Language).
    - Other systems interact with the Web service in a manner prescribed by its description using **SOAP** (Simple Object Access Protocol,) messages, typically conveyed using **HTTP** with an **XML serialization** in conjunction with other Web-related standards.

# Topic 2: Applications and Services.

- **XML Web Service reference architecture**
  - **Service requester:** requests the execution of a Web service
  - **Service provider:** processes a Web service request
  - **Discovery agency:** agency through which a Web service description is published and made discoverable




**Providers** are responsible for publishing a description of the service(s) they provide. **Requesters** must be able to find the description(s) of the services.

# XML Web Service reference architecture

- **Service Provider:** From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that hosts access to the service. Its role in the client-server message exchange patterns is that of a **server**.
- **Service Requestor:** From a business perspective, this is the business that requires certain function to be satisfied. From an architectural perspective, this is the **application** that is looking for and invoking or initiating an interaction with a service. Its role in the client-server message exchange patters is that of a **client**.
- **Discovery Agency (optional):** This is a searchable set of service descriptions where service providers publish their service descriptions.
  - The service provider can send the description directly to the client, or
  - The description can be obtained via service registration, FTP site, URL, …

- **XML Web Services Architecture**
  - **Formal definition:**
    - A **XML Web service** is a **software system** identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems.

    These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols.

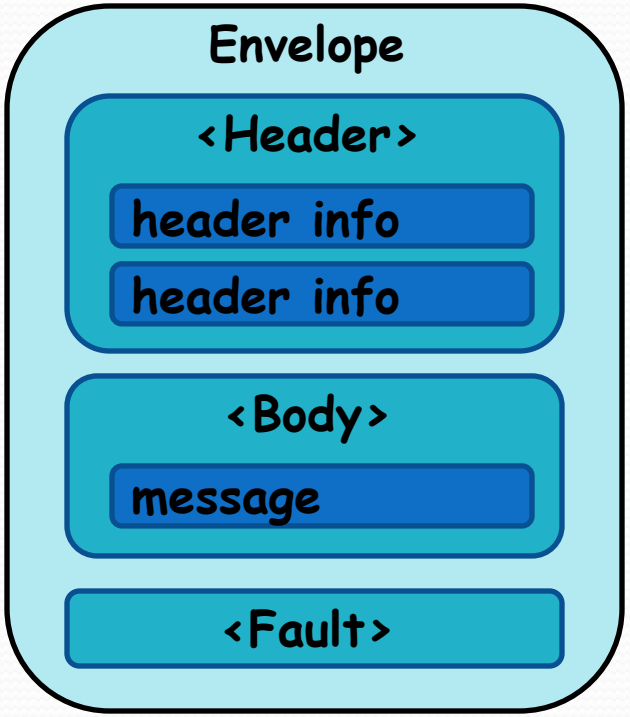
    This definition does not presuppose the use of SOAP as processing model or WSDL as service description language.

    Thus, raw http (data transfer protocol) and XML (format) can be used to produce Web services and the **reference architecture** uses SOAP and WSDL

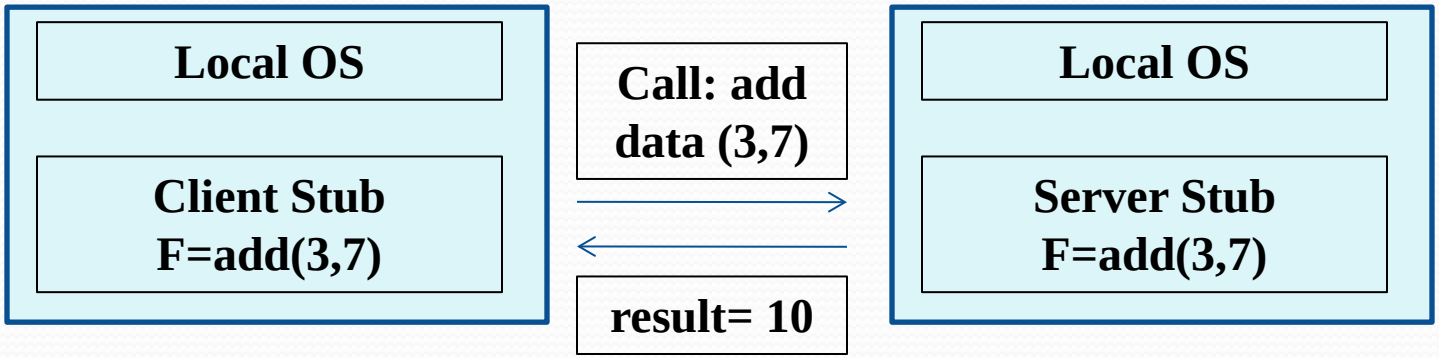
- **XML Web Services components:**
  - **SOAP (Simple Object Access Protocol):** protocol specification for exchanging structure information between client and servers in Web Services
    - **Message Format**: uses XML
    - **Communication:** uses application protocols such as HTTP, SMTP or XML-RPC

**Structure of SOAP message**

- **Envelope:** delimits SOAP Message
- **Header:** contains options for message processing
- **Body:** contains the request/response message
- **Faults:** signals error in the processing of the request



- **RPC (Remote Call Procedure):**
  - Allows the invocation of a function across a network. It is similar to a Java RMI (Java - Remote Method Invocation)
  - An RPC is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters.
    - The arguments are serialized to a wire-format
  - The remote server sends a response to the client, and the application continues its process.
    - The results are serialized to a wire-format



- **XML-RPC:**
  - An RPC that uses **XML** for serializing parameters
  - Uses HTTP as **transport protocol**
  - Defines few **data formats**: boolean, integer, double, string, date/time, base64 (for binary data), struct and array
  - Does NOT use XML Schema and thus does not support automatic validation
  - Does not consider naming and discovery capabilities

- **Example of XML-RPC:**

**XML-RPC call**

```
<?xml version="1.0"?>
<methodCall>
   <methodName> Nombre_Comunidad_Autonoma </methodName>
   <params>
          <param>
                 <value> <i4>  5 </i4> </value>
          </param>
   </params>
</methodCall>
```

**XML-RPC response**

```
<?xml version="1.0"?>
<methodResponse>
   <params>
          <param>
                 <value><string> Aragón </string></value>
          </param>
   </params>
</methodResponse>
```

- **SOAP example:** get the Stock Price of a Company

**SOAP request (using http as communication protocol)**

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"
<?xml version="1.0"?>
   <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
         <soap:Header>
         </soap:Header>
         <soap:Body>
                  <xmlns:m="http://www.example.org/stock">
                  <m:GetStockPrice ">
                           <m:StockName>  IBM  </m:StockName>
                  </m:GetStockPrice>
         </soap:Body>
   </soap:Envelope>
```

XML namespaces: SOAP specific

XML namespaces: application specific

http://www.w3schools.com/soap/soap_example.asp

- **SOAP example:** get the Stock Price of a Company

**SOAP response**

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
<?xml version="1.0"?>

<soap:Envelope
    xmlns:soap=http://www.w3.org/2001/12/soap-envelope
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body xmlns:m="http://www.example.org/stock">
            <m:GetStockPriceResponse>
            <m:Price> 34.5 </m:Price>
            </m:GetStockPriceResponse>
    </soap:Body>
</soap:Envelope>
```

XML namespaces: SOAP specific

XML namespaces: application specific

http://www.w3schools.com/soap/soap_example.asp

- **XML Web Services components:**
  - **WSDL (Web Services Description Language – WuzDuL)**: XML-based language that provides a model for describing the services that offers and their localization and the parameters and methods that supports
    - A client program connecting to a Web service can read the WSDL file to determine what operations are available on the server
    - The client can then use SOAP to actually call one of the operations listed in the WSDL file using XML or HTTP

- **WSDL (Web Services Description Language ):**
  - A WSDL document defines a web service using XML elements:
    - **<portType>: defines the Web Service** and the possible operations and messages that supports that service. It is similar to a function definition.
    - **<message> defines the data elements** that participates in an operation. Can be considered as the parameters that are passed in a function.
    - **<types>** for **defining the data types** that the Web Service uses. For that, it uses XML Schema.
    - **<binding>** for defining the **message format** and **communication protocols** that each port uses
    - **<port>** a combination of a binding and a network address, **providing the target address of the service communication**.
    - **<service> maps** the binding to the port and include any extensibility definitions.

- **WSDL (Web Services Description Language ):**

```
<definitions>

<types>
   definition of types.....
</types>

<message>
   definition of a message....
</message>

<portType>
   <operation>
         definition of a operation.....
   </operation>
</portType>

…
```

```
…

<binding>
   definition of a binding....
</binding>

<service>
   definition of a service....
</service>

</definitions>
```

- **WSDL example:**
  - **Example:** Assuming the service provides a single publicly available function, called *sayHello*. This function expects a single string parameter and returns a single string greeting. For example if you pass the parameter *world* then service function *sayHello* returns the greeting, "Hello, world!".
    - **Definition:** HelloService
    - **Port Type:** sayHello **operation** that consists of a request and response service.
    - **Message:**
      - sayHelloRequest : firstName parameter
      - sayHelloResponse: greeting return value
    - **Type:** Using built-in data types and they are defined in XMLSchema.
    - **Binding:** Direction to use the SOAP HTTP transport protocol.
    - **Service:** Service available at http://www.examples.com/SayHello/.
    - **Port:** Associates the binding with the URI http://www.examples.com/SayHello/ where the running service can be accessed.

http://www.tutorialspoint.com/wsdl/wsdl_example.htm

- **WSDL example:**

```
<definitions name="HelloService"
(1)         targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
            xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
(2,3,4,5) ……..
</definitions>
```

• The definitions element is a container of all the other elements.
• The definitions element specifies that this document is the *HelloService*.
• The definitions element specifies a *targetNamespace* attribute.
• The definition element specifies a default namespace. All elements without a namespace prefix, such as *message* or *portType*, are therefore assumed to be part of the default WSDL namespace.
• It also specifies numerous namespaces (soap, tns, xsd) that will be used throughout the remainder of the document.

- **WSDL example:**

```
<definitions name="HelloService"
(1)    ……..

(2)   <portType name="Hello_PortType">
          <operation name="sayHello">
                <input message="tns:SayHelloRequest"/>
                <output message="tns:SayHelloResponse"/>
          </operation>
      </portType>
(3,4,5) ……..
</definitions>
```

• The **<portType>** element combines multiple message elements to form a complete one way or round-trip operation. For example, a <portType> can combine one request and one response message into a single request/response operation.

•The portType element defines a single operation, called *sayHello*.
• The operation itself consists of a single input message *SayHelloRequest*
• The operation itself consists of a single output message *SayHelloResponse*

- **WSDL example:**

```
<definitions name="HelloService"
(1,2)    ……..

(3)        <message name="SayHelloRequest">
                    <part name="firstName" type="xsd:string"/>
           </message>
           <message name="SayHelloResponse">
                    <part name="greeting" type="xsd:string"/>
           </message>

(4,5) ……..
</definitions>
```

• Each Web Service has two messages: **input** and **output**. The input describes the parameters for the Web Service and the output describes the return data from the Web Service.
• Each **<part>** parameter associates with a concrete type defined in the **<types>** container element.

- **WSDL example:**
  - The **<binding>** element provides specific details on how a *portType* operation will actually be transmitted over the wire.
  - The bindings can be made available via multiple transports, including HTTP GET, HTTP POST, or SOAP.
  - The bindings provide concrete information on what protocol is being used to transfer *portType* operations.
  - The bindings provide information where the service is located.
  - For SOAP protocol, the binding is **<soap:binding>**, and the transport is SOAP messages on top of HTTP protocol.

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
```

  - You can specify multiple bindings for a single *portType*.

- **WSDL example:**

```
<definitions name="HelloService"
(1,2,3)    ……..

(4) <binding name="Hello_Binding" type="tns:Hello_PortType">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="sayHello">
        <soap:operation soapAction="sayHello"/>
            <input>
            <soap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="urn:examples:helloservice"
                    use="encoded"/>
            </input>
            <output>
            <soap:body
                    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    namespace="urn:examples:helloservice"
                    use="encoded"/>
            </output>
        </operation>
    </binding>
(5)  ……..
```

- **WSDL example:**

```
(1,2,3,4) ……….
(5)  <service name="Hello_Service">
          <documentation>
                    WSDL File for HelloService
          </documentation>
          <port binding="tns:Hello_Binding" name="Hello_Port">
                    <soap:address   location="http://www.examples.com/SayHello/">
          </port>
     </service>

</definitions>
```

- The **<port>** element has two attributes - the name attribute and the binding attribute.
- The name attribute provides a unique name among all port types defined within in the enclosing WSDL document.
- The binding attribute refers to the binding using the linking rules defined by WSDL.
- A port MUST NOT specify more than one address.
- A port MUST NOT specify any binding information other than address information.
- The **<service>** element defines the ports supported by the Web service. For each of the supported protocols, there is one port element. The service element is a collection of ports.

## Topic 2: Applications and Services.

- **XML Web Services components:**
  - **UDDI (Universal Description Discovery and Integration):** XML-based registry where applications are found - it is like Yellow Pages-like directory of services
    - **Message Format**: uses XML
    - **Communication:** uses SOAP
  - **UDDI has two parts:**
    - A registry of all a web service's metadata including a pointer to the WSDL description of a service
    - A set of WSDL port type definitions for manipulating and searching that registry
- Looking for web services:
  - http://www.xmethods.net

- **JSON Web Services:**
  - JSON format is often used for **serializing** and transmitting structured data over a network connection. It is used primarily to transmit data between a server and web application, serving as an alternative to XML.
  - JSON-RPC instead of XML-RPC or SOAP, allows for notifications (information sent to the server that do not require a response) and for multiple calls to be sent to the server that may be answered out of order.
  - JSON schema still a draft standard (JSON is schema-less)
  - JSON is less complex than XML (many people don't need namespaces and they don't need mixed content documents)

What happens if you want richer Web API or do you want to federate JSON data across several application domains (need a namespace) ?

- **Ongoing Topics:**
  - **HTML5:** an attempt to define a single markup language that can be written in either HTML or XHTML syntax
  - **Semantic Web:** XML standards for interoperation of web services specify only **syntactic** interoperability, not the **semantic** meaning of messages.
    - **RDF (Resource Description Framework):** it is based upon the idea of making statements about resources (in particular web resources) in the form of subject-predicate-object expressions.
    - **OWL (Web Ontology Language):** is a family of knowledge representation languages for authoring ontologies.
  - **Cloud Computing:** network-based services which appear to be provided by real server hardware, which in fact are served up by virtual hardware, simulated by software running on one or more real machines.

# Topic 2: Applications and Services.

- **Computer Security** - generic name for the collection of tools designed to protect data and to thwart hackers
- **Network Security** - measures to protect data during their transmission
- * **Internet Security** - measures to protect data during their transmission over a collection of interconnected networks
- consider three aspects of information security:
  - **Security service**
  - **Security mechanism**
  - **Security attack**

* Security slides are based on Stalling book course slides and in Kurose book course slides

- **Security service:**
  - is something that **enhances the security** of the data processing systems and the information transfers of an **organization**
  - intended to **counter** security attacks
  - make use of one or more security mechanisms to provide the service
  - replicate functions normally associated with physical documents
    - E.g. have signatures, dates; need protection from disclosure, tampering, or destruction; be notarized or witnessed; be recorded or licensed

- **Security service:**
  - **Authentication** - assurance that the communicating entity is the one claimed
  - **Access Control** - prevention of the unauthorized use of a resource
  - **Data Confidentiality** –protection of data from unauthorized disclosure
  - **Data Integrity** - assurance that data received is as sent by an authorized entity
  - **Non-Repudiation** - protection against denial by one of the parties in a communication

- **Security mechanism:**
  - a **mechanism that is designed to** detect, prevent, or recover from a security attack
  - **no single mechanism** that will support all functions required
  - however one particular element underlies many of the security mechanisms in use: **cryptographic techniques**
  - hence our focus on this area

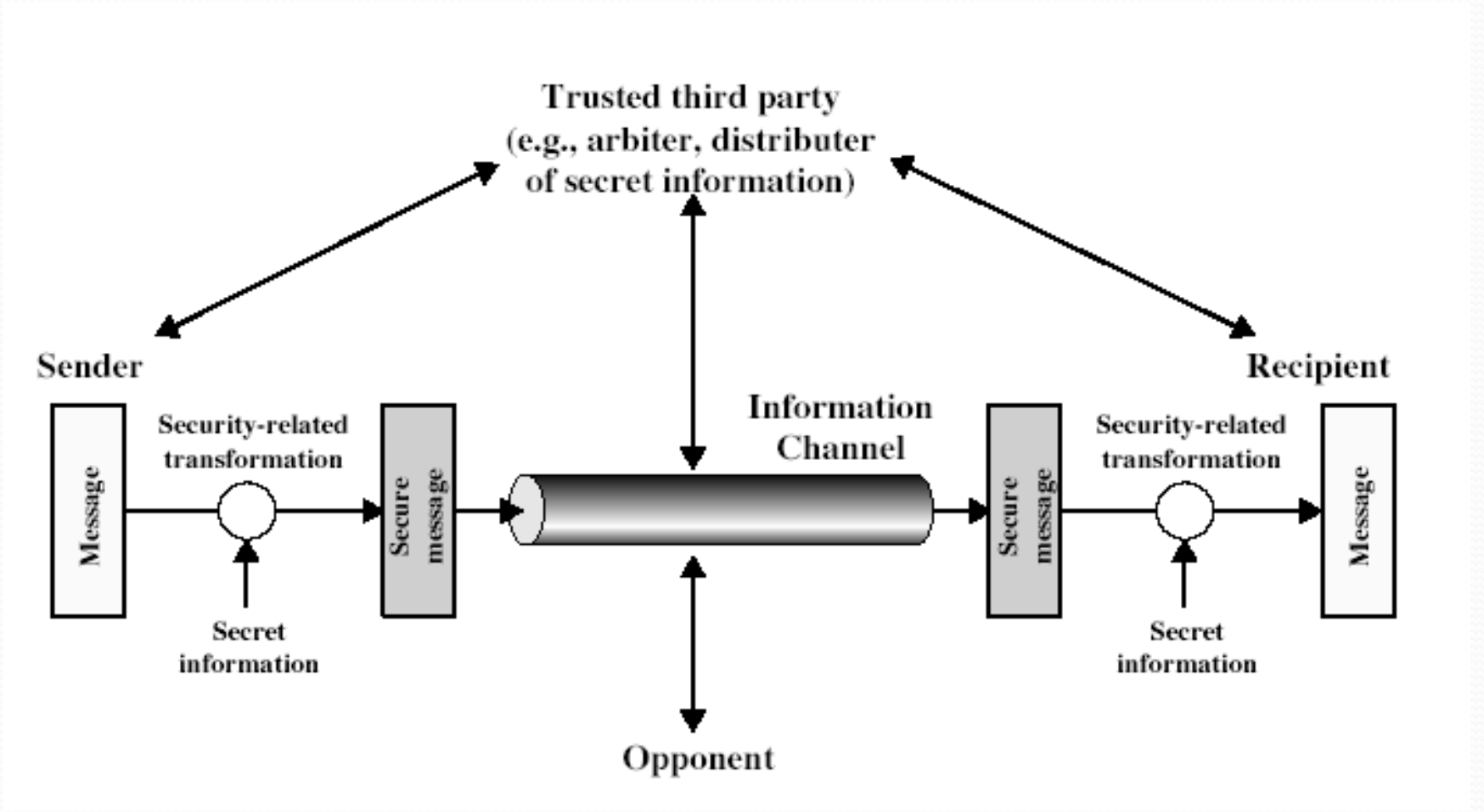
- **Security mechanism:**
  - specific security mechanisms:
    - encipherment, digital signatures, access controls, data integrity, authentication exchange, traffic padding, routing control, notarization
  - pervasive security mechanisms:
    - trusted functionality, security labels, event detection, security audit trails, security recovery

- **Security attack:**
  - any action that **compromises** the security of information owned by an organization
  - information security is about how to **prevent attacks**, or failing that, to detect attacks on information-based systems
  - have a wide range of attacks
  - can focus of generic types of attacks
  - note: often *threat* & *attack* mean same

- **Security attack:**
  - **passive attacks** - eavesdropping on, or monitoring of, transmissions to:
    - obtain message contents, or
    - monitor traffic flows
  - **active attacks** – modification of data stream to:
    - masquerade of one entity as some other
    - replay previous messages
    - modify messages in transit
    - denial of service

# Topic 2: Applications and Services.

- **Model for Network Security:**



- **Model for Network Security :**
  - using this model requires us to:
    - design a **suitable algorithm** for the security **transformation**
    - **generate the secret information** (keys) used by the algorithm
    - develop methods to **distribute and share** the secret information
    - specify a **protocol** enabling the principals to use the transformation and secret information for a security service

- **Key Cryptography:**
  - **substitution cipher:** substituting one thing for another
    - E.g. mono-alphabetic cipher: substitute one letter for another

**Plaintext:** abcdefghijklmnopqrstuvwxyz

**Ciphertext:** mnbvcxzasdfghjklpoiuytrewq

E.g.:

**Plaintext (m):** bob. i love you. alice

**Ciphertext (c):** nkn. s gktc wky. mgsbc

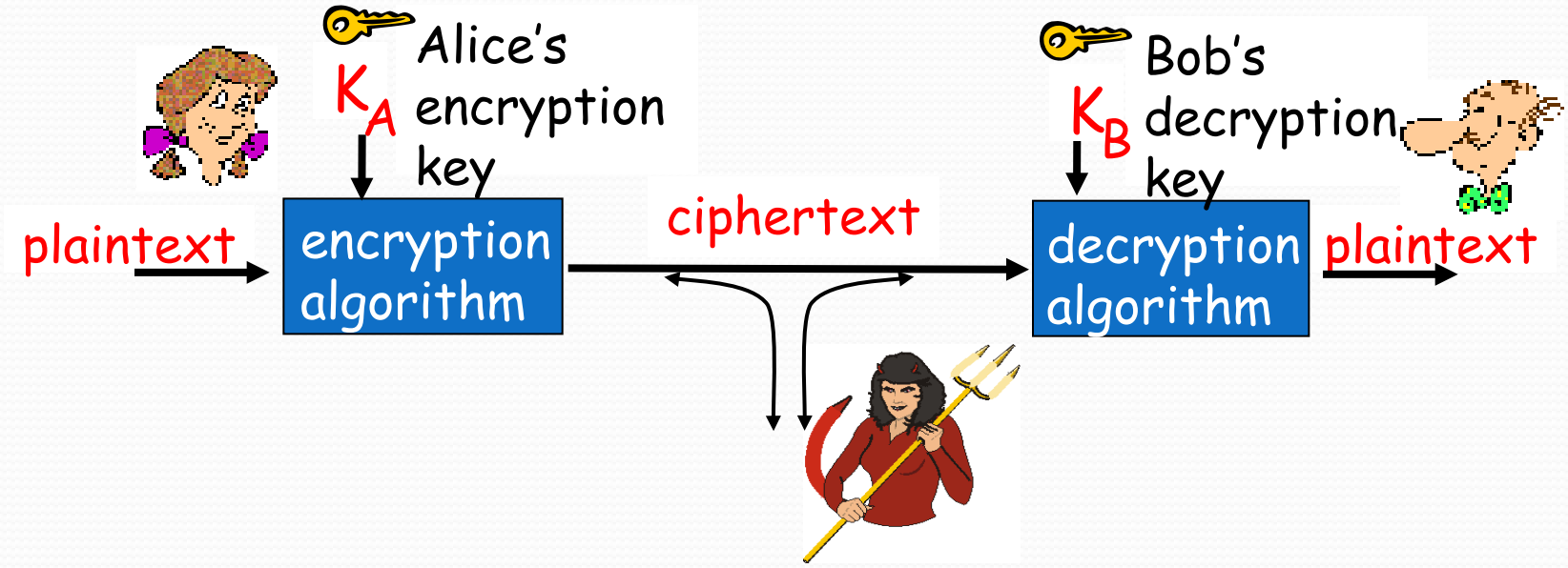
$$c = K(m)$$

**Question:** How hard to break this simple cipher?:

- ❑ brute force (how hard?)
- ❑ other?

- **Key Cryptography** uses keys to transform (encrypt/decrypt) the messages

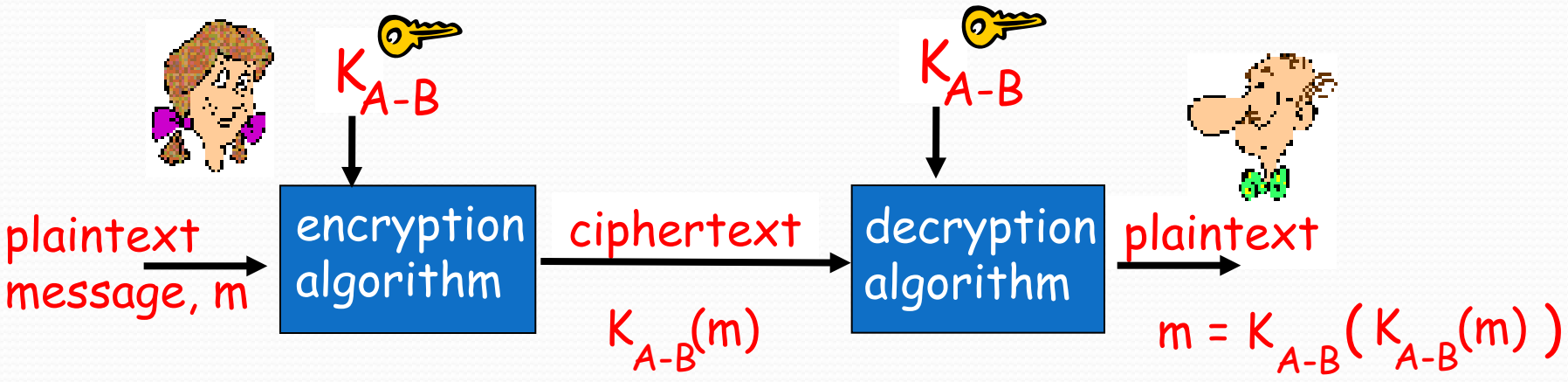
| | |
|---|---|
| Alice: | $c = K_A(m)$ |
| Bob: | $m = K_B(c)$ |



## Topic 2: Applications and Services.

- traditional **private/secret/single key** cryptography uses **one** key
  - shared by both sender and receiver
  - if this key is disclosed communications are compromised
  - also is **symmetric**, parties are equal
  - hence does not protect sender from receiver forging a message & claiming is sent by sender

- **Symmetric Key Cryptography:**



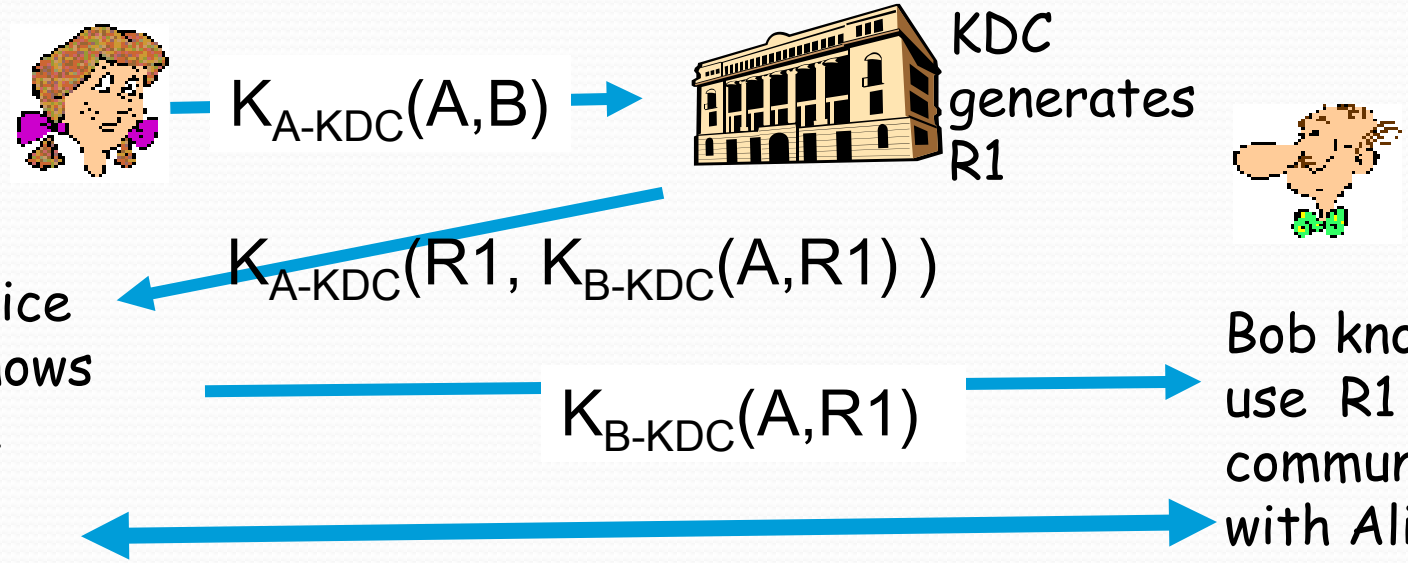

**symmetric key crypto:** Bob and Alice share know same (symmetric) key: $K_{A-B}$

- e.g., key is knowing substitution pattern in mono alphabetic substitution cipher
- **Question:** how do Bob and Alice agree on key value?

  **Solution:** trusted key distribution center (KDC) acting as intermediary between entities

# Key Distribution Center (KDC)

***Question:*** How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?




Alice and Bob communicate: using R1 as ***session key*** for shared symmetric encryption

- **Symmetric Key Cryptography: transformation**

## DES: Data Encryption Standard

- US encryption standard [NIST 1993]
- 56-bit symmetric key, 64-bit plaintext input
- How secure is DES?
  - **DES Challenge**: 56-bit-key-encrypted phrase ("Strong cryptography makes the world a safer place") decrypted (brute force) in 4 months
  - no known "backdoor" decryption approach

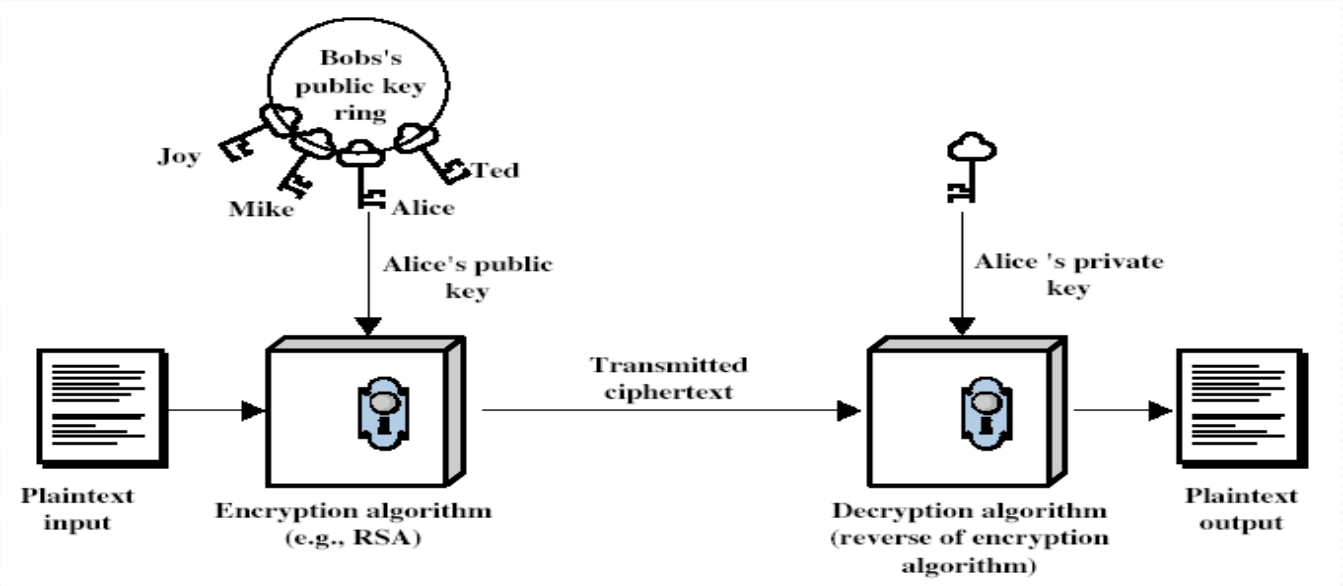
- **Symmetric Key Cryptography: transformation**

## AES: Advanced Encryption Standard

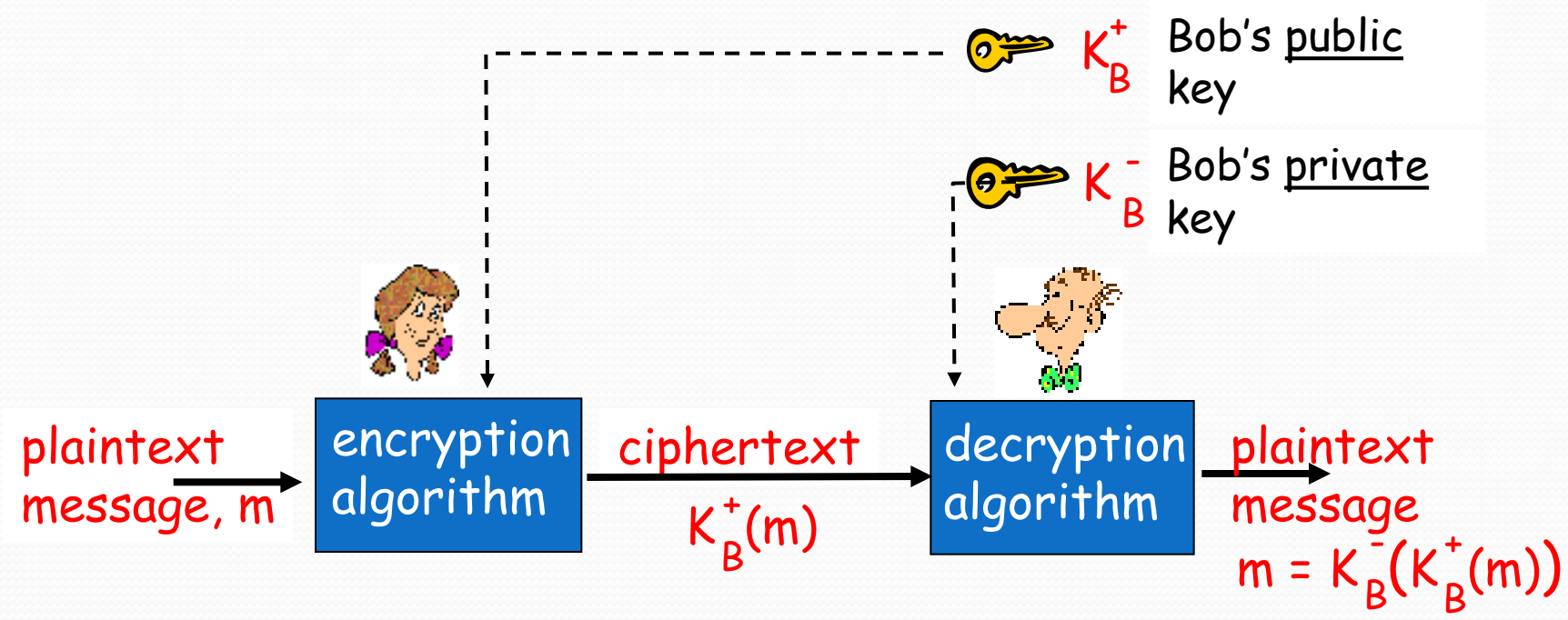
- new (Nov. 2001) symmetric-key NIST standard, replacing DES
- processes data in 128 bit blocks
- 128, 192, or 256 bit keys
- brute force decryption (try each key) taking 1 sec on DES, takes 149 trillion years for AES

# Topic 2: Applications and Services.

- **public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a **private-key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
  - is **asymmetric** because
    - those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures



- Public Key Cryptography:



- **Public Key Cryptography:**

Requirements:

1. need $K_B^+(\cdot)$ and $K_B^-(\cdot)$ such that

$$K_B^-(K_B^+(m)) = m$$

2. given public key $K_B^+(\cdot)$, it should be impossible to compute private key $K_B^-(\cdot)$

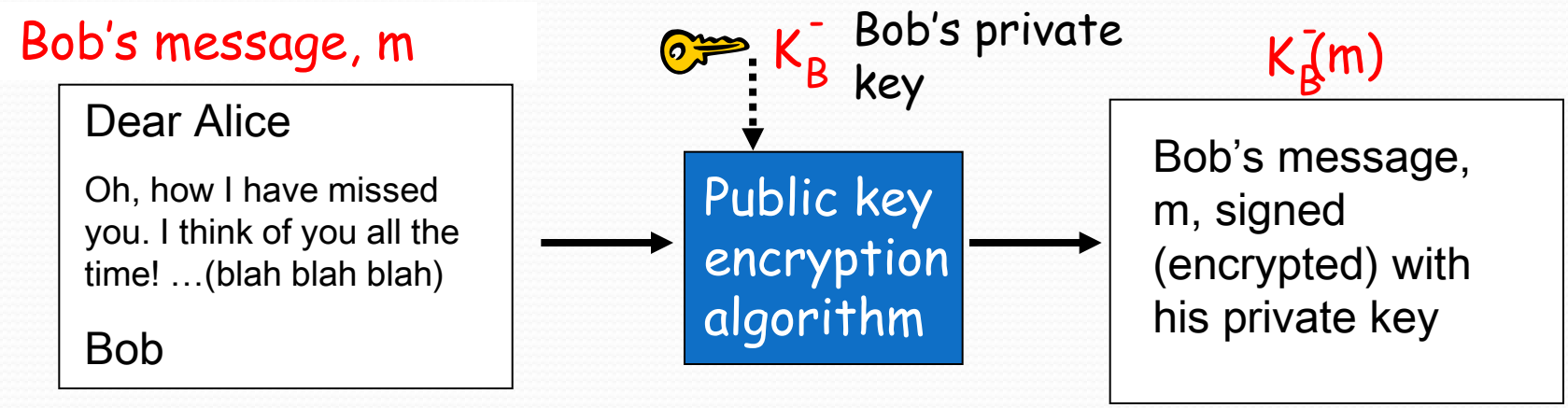
**RSA:** Rivest, Shamir, Adelson algorithm

$$K_B^-(K_B^+(m)) = m = K_B^+(K_B^-(m))$$

- **Why Public Key Cryptography ?**
  - developed to address two key issues:
    - **key distribution** – how to have secure communications in general without having to trust a KDC with your key
    - **digital signatures** – how to verify a message comes intact from the claimed sender
  - public invention due to Whitfield Diffie & Martin Hellman at Stanford Uni in 1976
    - known earlier in classified community

# Public Key Cryptography: Digital Signature

- Bob signs **m** by encrypting with his private key $K_B^-$, creating "signed" message, $K_B^-(m)$



- **Public Key Cryptography: Digital Signature**
  - Suppose Alice receives msg m, digital signature $K_B^-(m)$
  - Alice verifies **m** signed by Bob by applying Bob's public key $K_B^+$ to $K_B^-(m)$ then checks $K_B^+(K_B^-(m)) = m$.
  - If $K_B^+(K_B^-(m)) = m$, whoever signed m must have used Bob's private key.

  **Alice thus verifies that:**

  - ü Bob signed m.
  - ü No one else signed m.
  - ü Bob signed m and not m'.

  **Non-repudiation:**

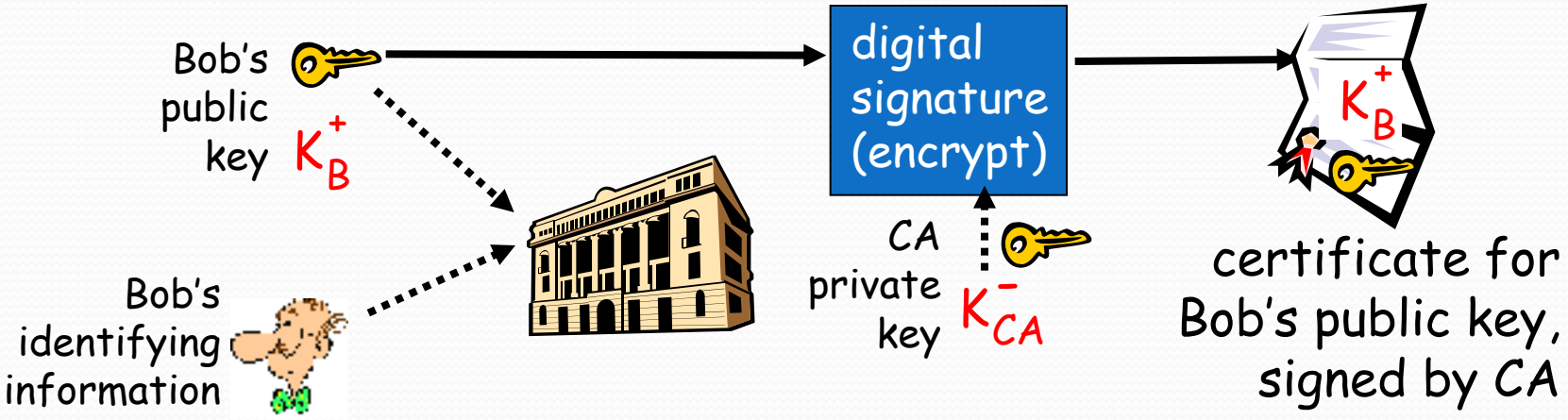
  - ✓ Alice can take m, and signature $K_B(m)$ to court and prove that Bob signed m.

## Topic 2: Applications and Services.

- **Distribution of Public Keys:** can be considered as using one of:
  - **Public announcement:** users distribute public keys to recipients or broadcast to community at large, but anyone can create a key claiming to be someone else and broadcast it
  - **Publicly available directory:** can obtain greater security by registering keys with a public directory, still vulnerable to tampering or forgery
  - **Public-key authority:** improve security by tightening control over distribution of keys from directory. It has properties of directory and requires users to know public key for the directory
  - **Public-key certificates:** certificates allow key exchange without real-time access to public-key authority. A certificate binds **identity** to **public key** usually with other info such as period of validity, rights of use etc with all contents **signed** by a trusted Public-Key or **Certificate Authority (CA)**

# Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- E (person, router) registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



- **Homework**
  - Look information on Web Services architecture: SOAP and WSDL
  - Look information on RESTful architecture
  - Look information on Web Services: XML versus JSON
  - Look information on public keys and signature with public keys
  - Look information on KDC and CA