



# CS 524 A

Introduction to Cloud Computing

Lecture 13:

Identity Management and Security in the Cloud, Part II

# AGENDA

- Authentication
  - Authentication protocol principles
  - Authentication with Kerberos
  - More on authentication with PKI
  - IPsec and TLS
- Authorization (OAuth as an example)
- Identity Management in the Cloud
  - Requirements
  - OpenStack IdM (*Keystone*)
  - Use of TPM (secure boot, attestation, etc.)
  - Use of HSM

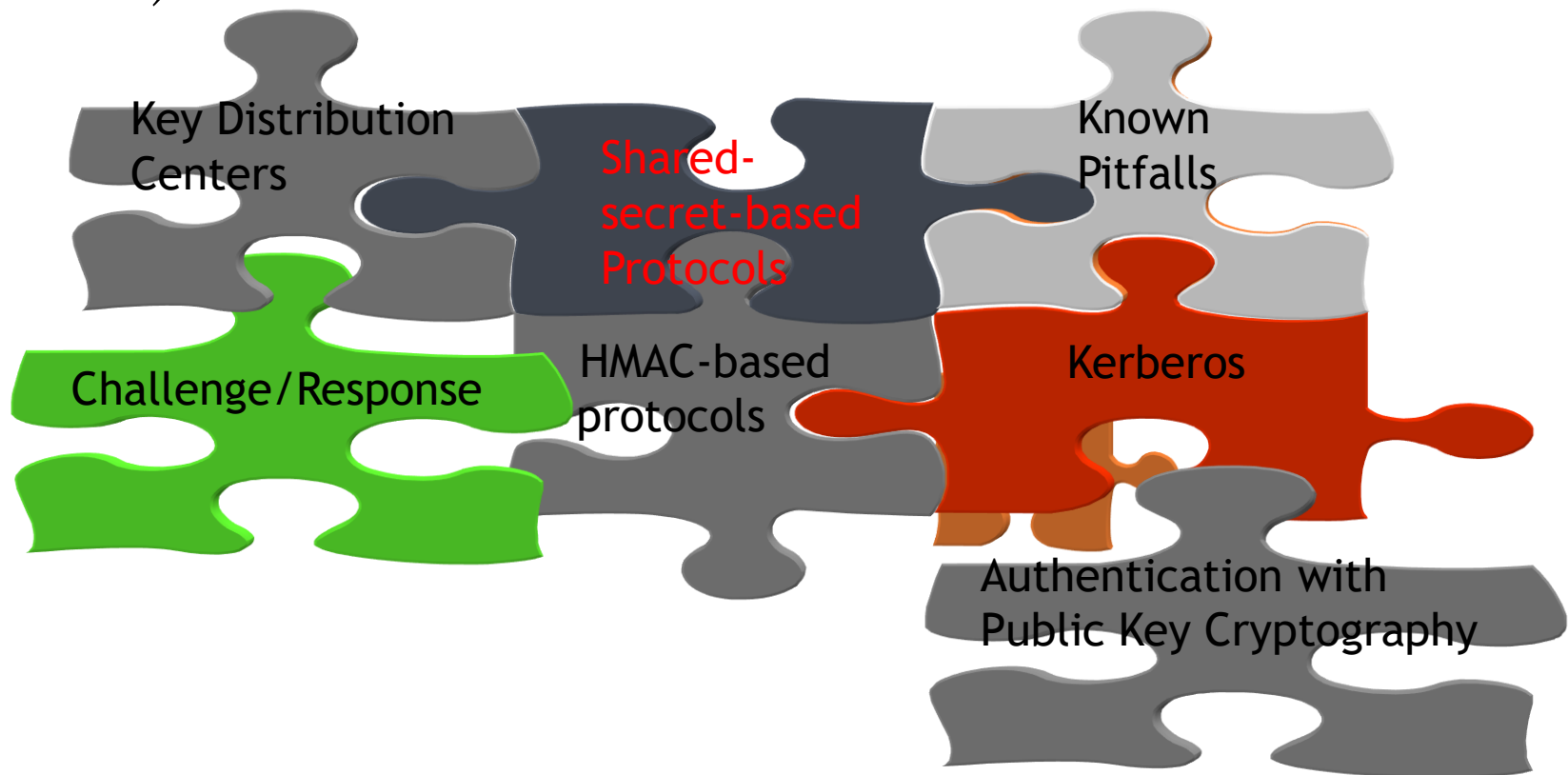


# BACK TO BASIC SECURITY PROPERTIES

- Confidentiality
  - Keeping information secret from unintended users
  - Achieved through encryption*
- Authentication
  - Confirming the identity of the presenter of the information
  - Achieved by *authentication protocols**
- Authorization
  - Determining whether a user may have access to a resource
  - Achieved through access control lists (ACLs), implementing policies, and also by *token-base authorization protocols (OAuth, OpenStack Keystone tokens, etc.)**
- Integrity
  - Ensuring that a message received was the one that was actually sent
  - Achieved by cryptographic means called *signatures**
- Non-repudiation
  - Ensuring that a party that has signed a contract cannot later deny having signed it
  - Achieved by either third-party attestation or asymmetric cryptographic means*

# AUTHENTICATION PROTOCOLS

Needed for the establishment of sessions (VoIP conversations [streams and signaling], TCP sessions, etc.)



# INTRODUCTION TO KEY PLAYERS

Alice



Bob



Trudy the Intruder



# THE GENERAL MODEL

- Alice starts by sending a message to Bob
- An exchange follows
- Trudy may intercept, modify, or replay any message

# CHALLENGE-RESPONSE PROTOCOL (FIRST ATTEMPT)

Alice

Bob

A Identifier

$R_B$  Challenge: A *nonce*--a large random number, not to be repeated

$K_{AB}(R_B)$  Response, encrypted with the shared--or private--key

$R_A$

$K_{AB}(R_A)$

# CHALLENGE-RESPONSE PROTOCOL (CAN WE DO THIS FASTER?)

Alice

Bob

$A, R_A$

$R_B, K_{AB}(R_A)$

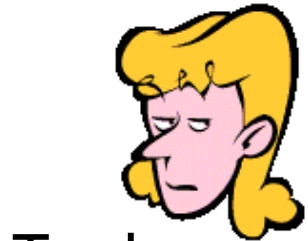
$K_{AB}(R_B)$

An improvement:  
3 instead of 5 messages!

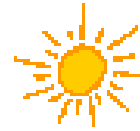
No!



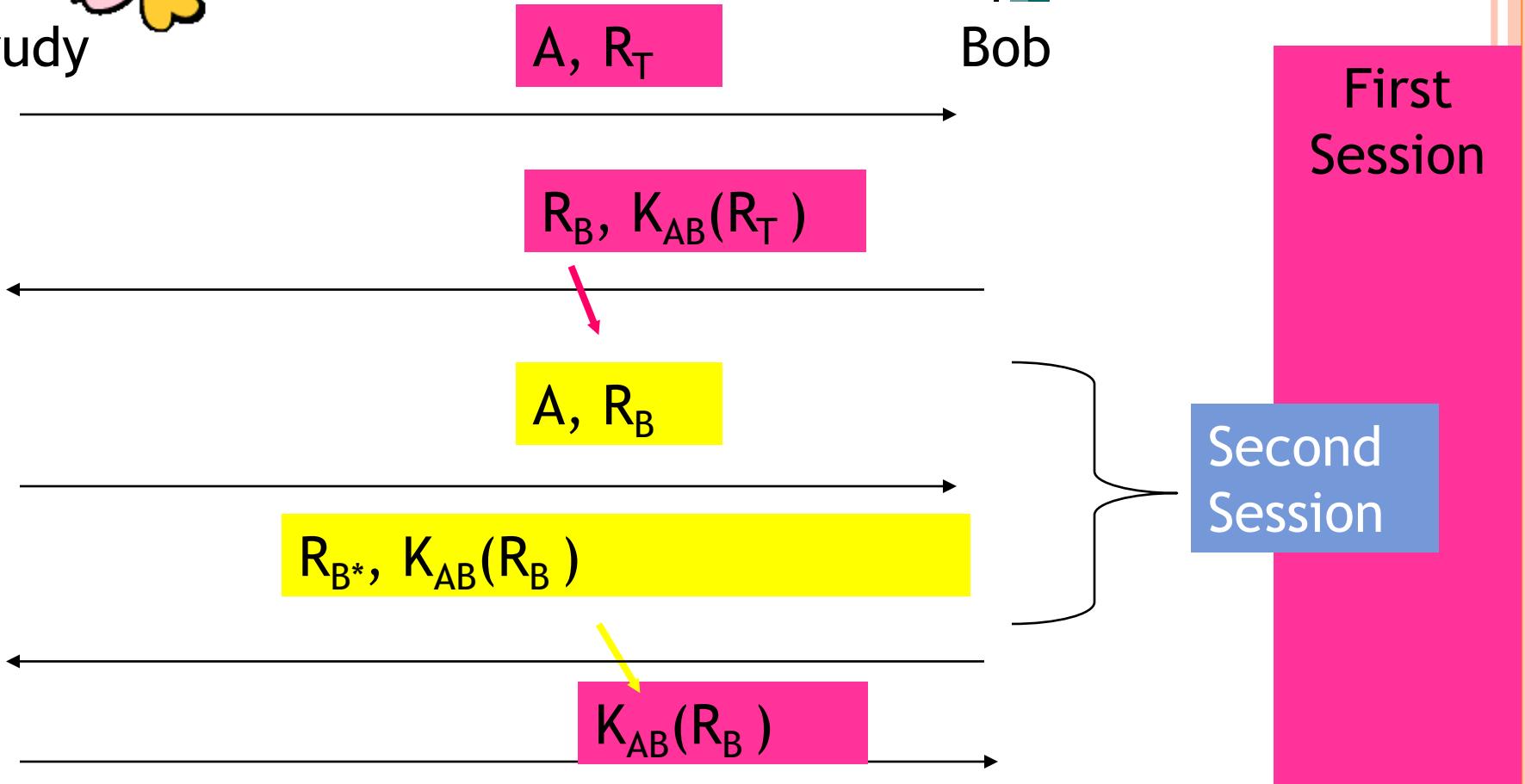
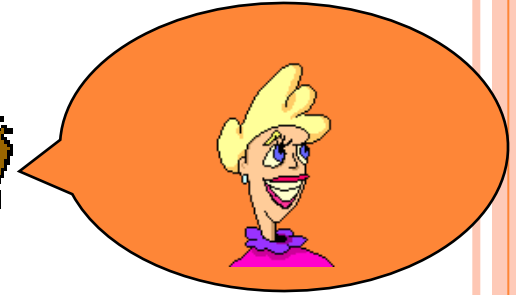
# THE REFLECTION ATTACK



Trudy



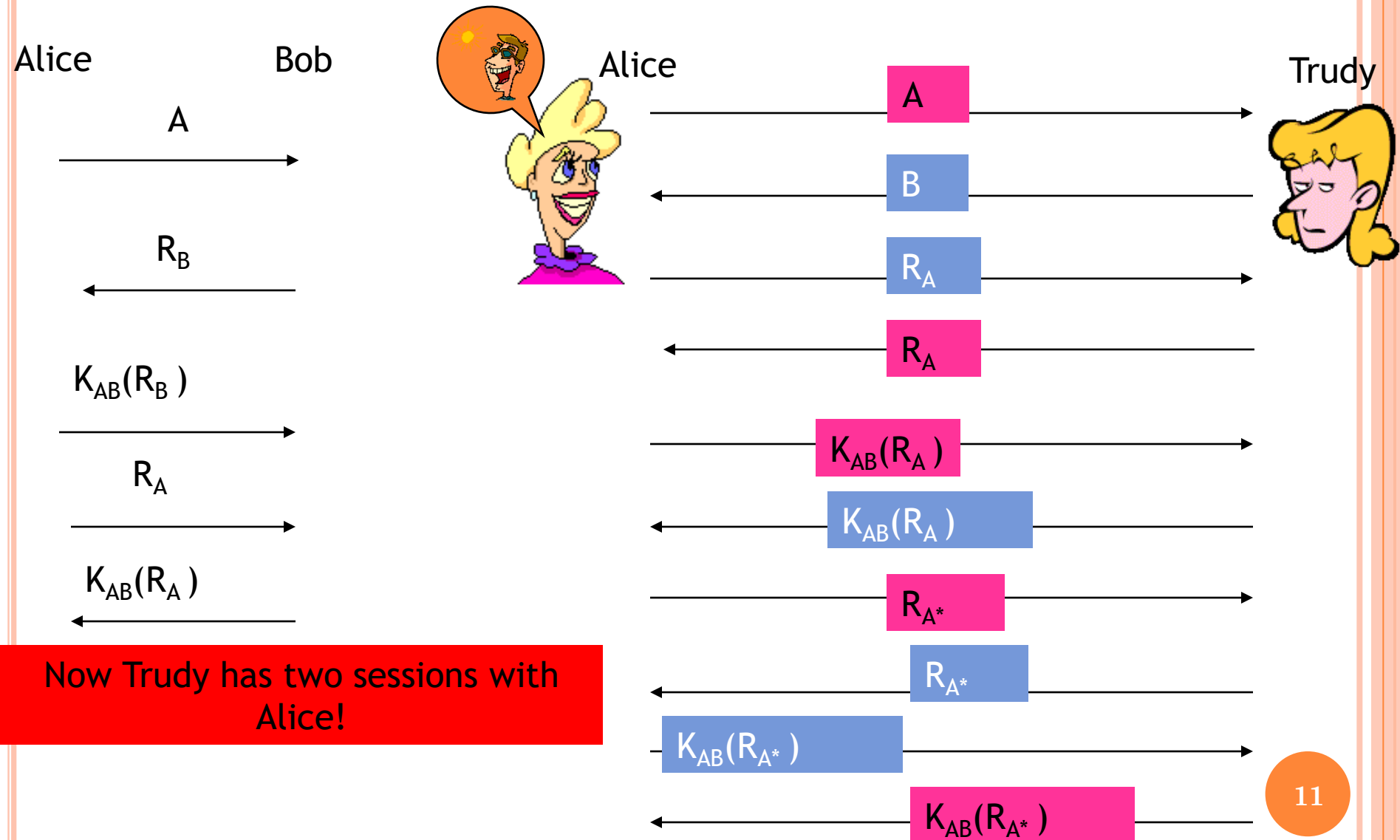
Bob



# GENERAL (TEXTBOOK) RULES

- The initiator has to prove its identity before the responder
- The initiator and responder must use different keys for proof (a need for two shared keys)
- Initiator and responder must draw challenges from different sets (e.g., odd/even)
- It must be impossible to use authentication information obtained in one session in a different one

# BUT WAS THE FIRST ATTEMPT REALLY FAULTLESS?



# A FEW CONCLUSIONS

- The authentication protocols are very hard to design correctly
- There is a method of designing protocols of this kind that are *provably* correct: R. Bird et al., *Systematic Design of a Family of Attack-Resistant Authentication Protocols*, IEEE Journal on Selected Areas in Communications, vol. 11, pp. 679-693, June 1993

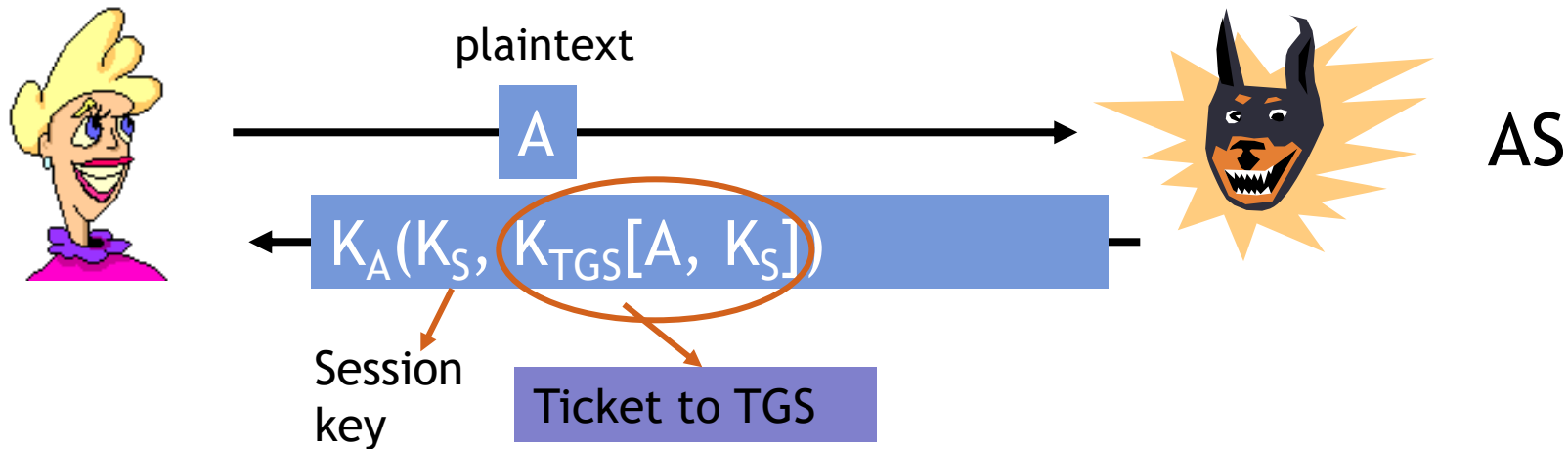
# AUTHENTICATION WITH KERBEROS

- Kerberos was originally developed by MIT based on a variant of *Needham-Shroeder* protocol and then standardized in the IETF
  - RFC 4120 specifies Kerberos V5
- Kerberos assumes that all clocks are synchronized
- Kerberos modifies the KDC model

# THE KERBEROS MODEL: THREE SERVERS

- Authentication Server (AS)
  - Authenticates users during the *login* session
  - Shares a secret (password) with every user
- Ticket-Granting Server (TGS)
  - Issues proof-of-identity tickets, which convince other servers that the owners of the tickets are who they claim to be
- The Application Server (AP)
  - Does the real work (performs services such as banking transactions, telephone calls, etc.)

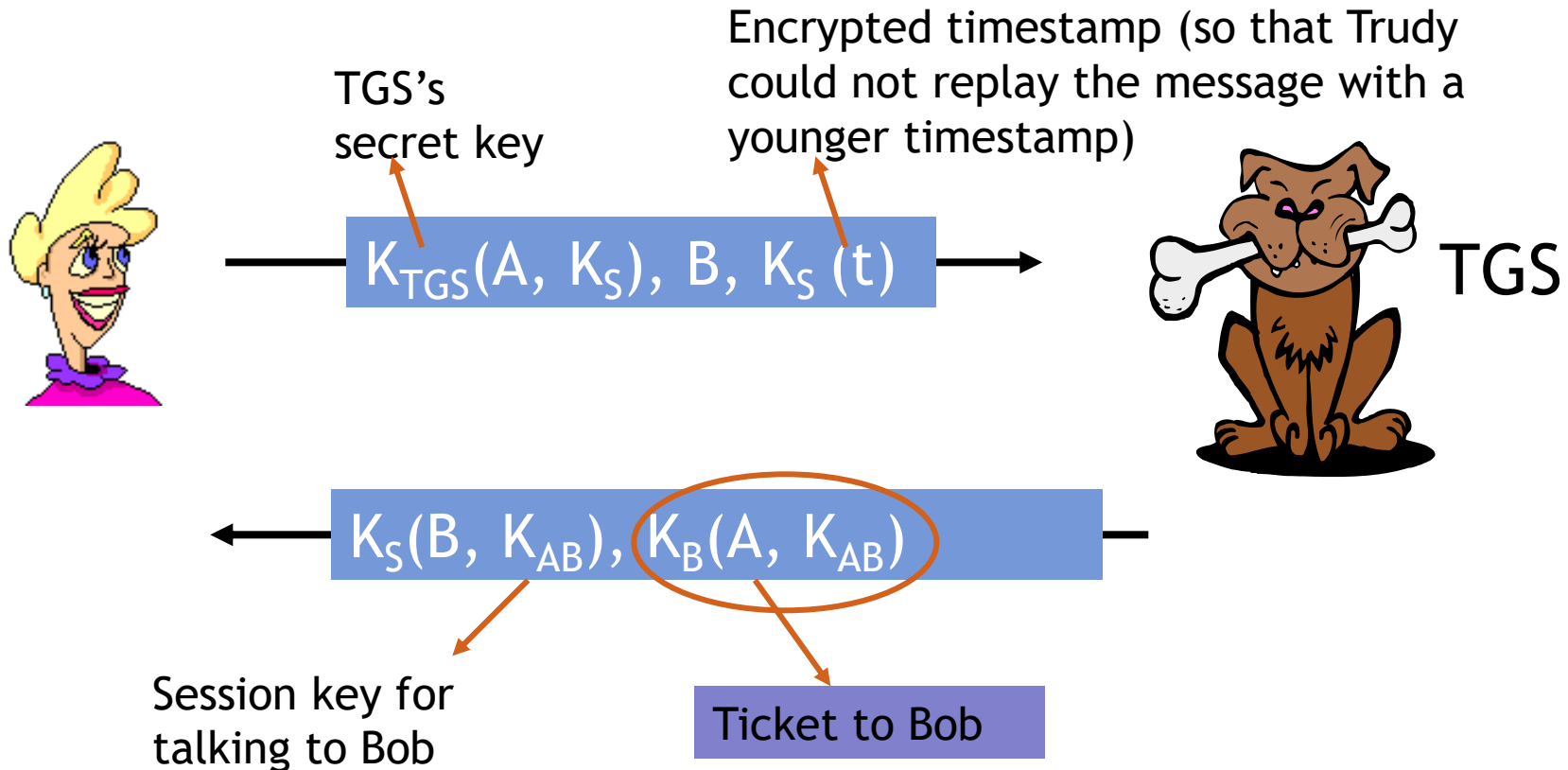
# OPERATION OF KERBEROS AS



At this point,

- 1) Alice is prompted for a password by the client, and this password is used for generating  $K_A$ , so that she obtains the session key and the ticket for TGS
- 2) The client *forgets* the password
- 3) Alice says she wants to use Bob's services

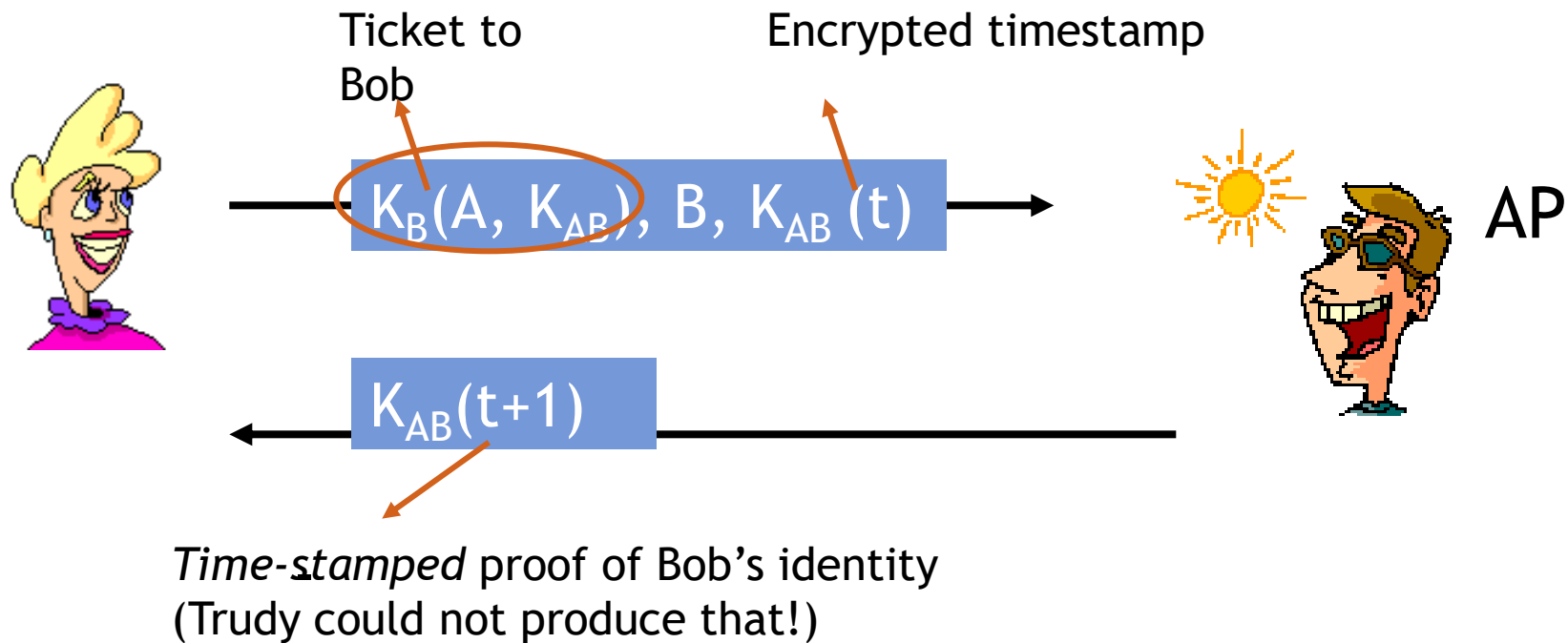
# OPERATION OF KERBEROS TGS



Now Alice can start talking to the real-work server—Bob

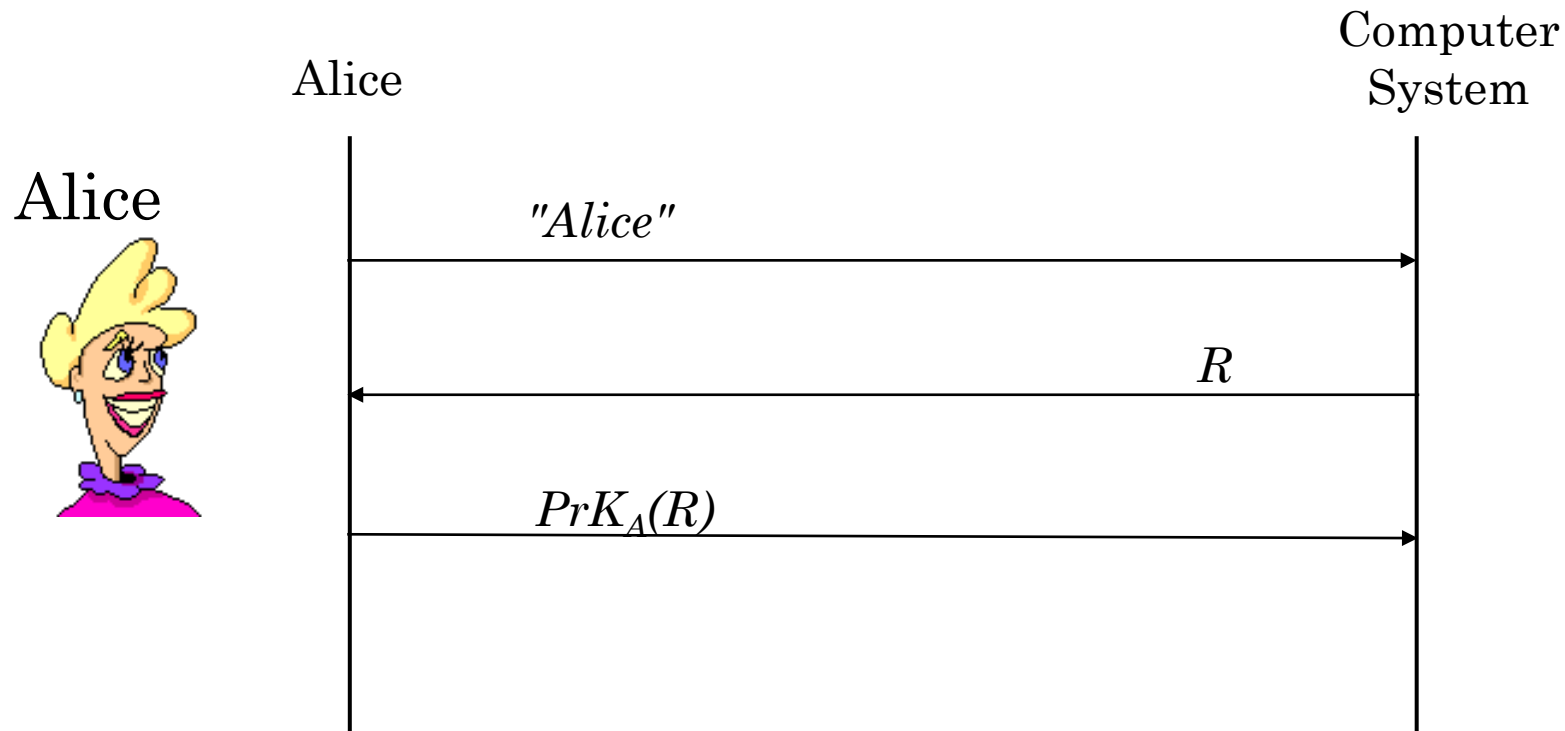


## OPERATION OF KERBEROS APPLICATION SERVER

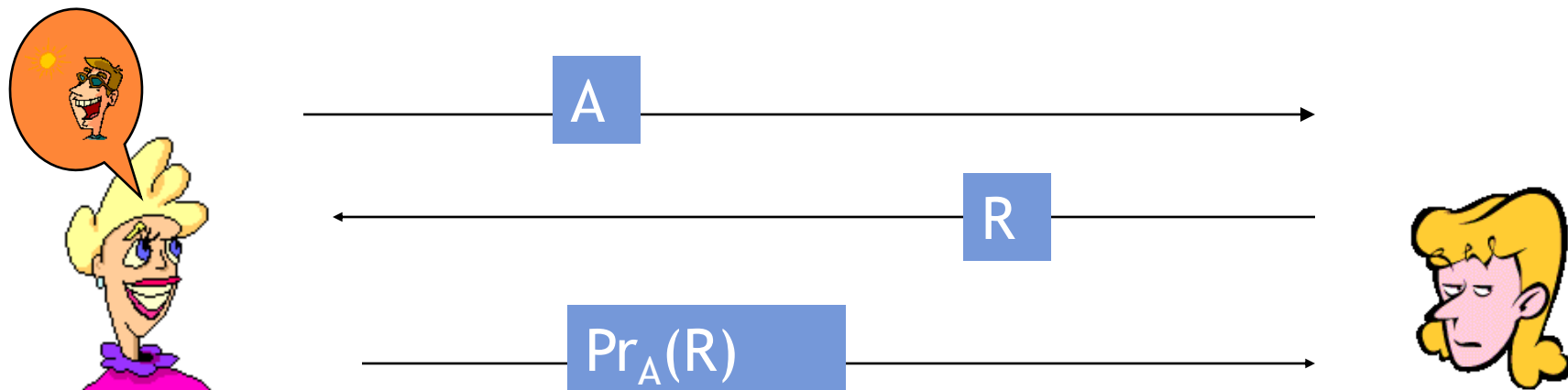


Now Alice can work with Bob, but if she needs to change to another application server, she just restarts with the request to TGS (no passwords are ever transmitted)

# PUBLIC-KEY-BASED AUTHENTICATION



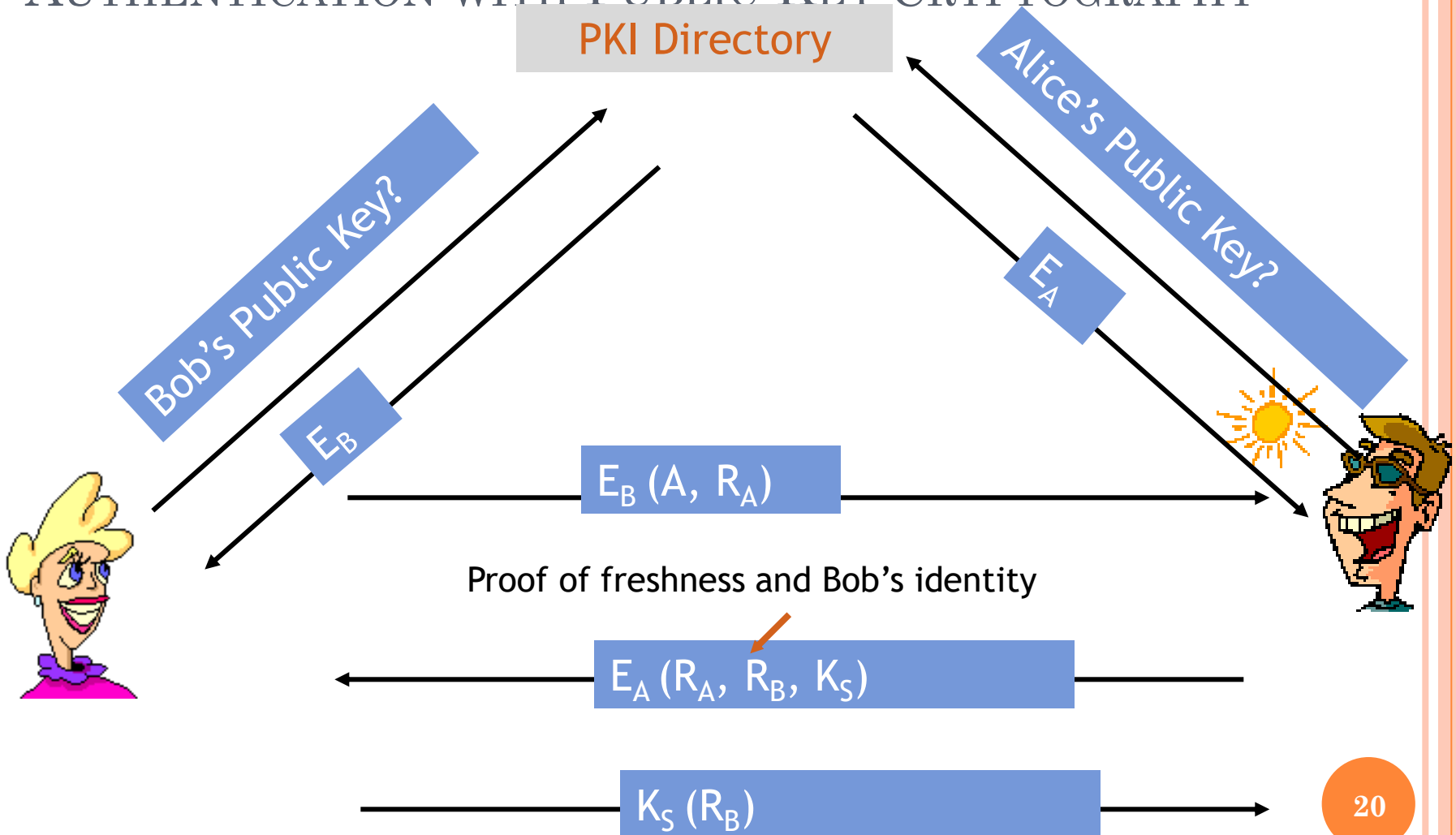
# AUTHENTICATION WITH PUBLIC-KEY CRYPTOGRAPHY: A NAÏVE “SOLUTION”



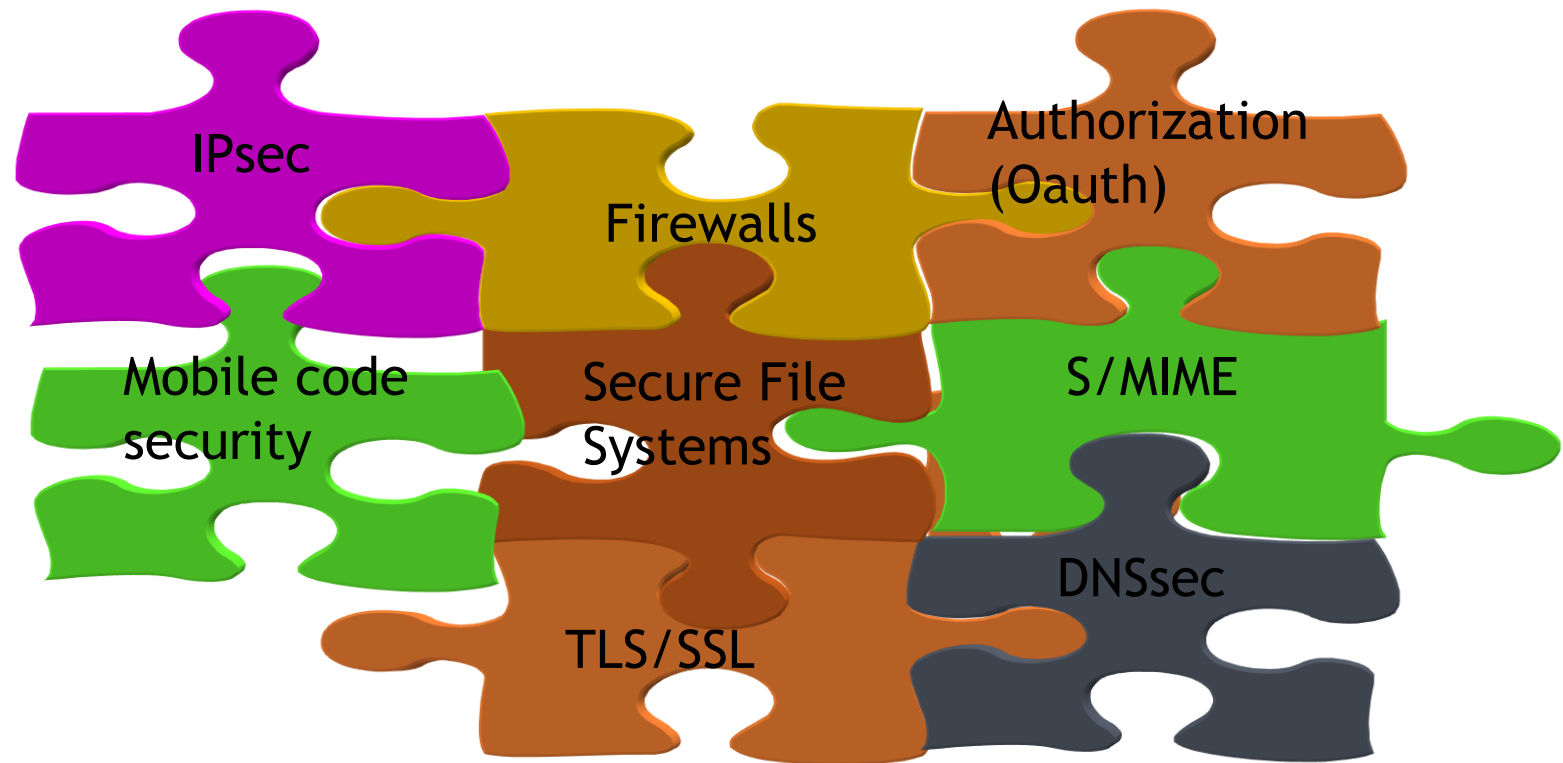
... $R$  = “I, undersigned Alice, owe Trudy \$1,000,000”

# AUTHENTICATION WITH PUBLIC-KEY CRYPTOGRAPHY

PKI Directory

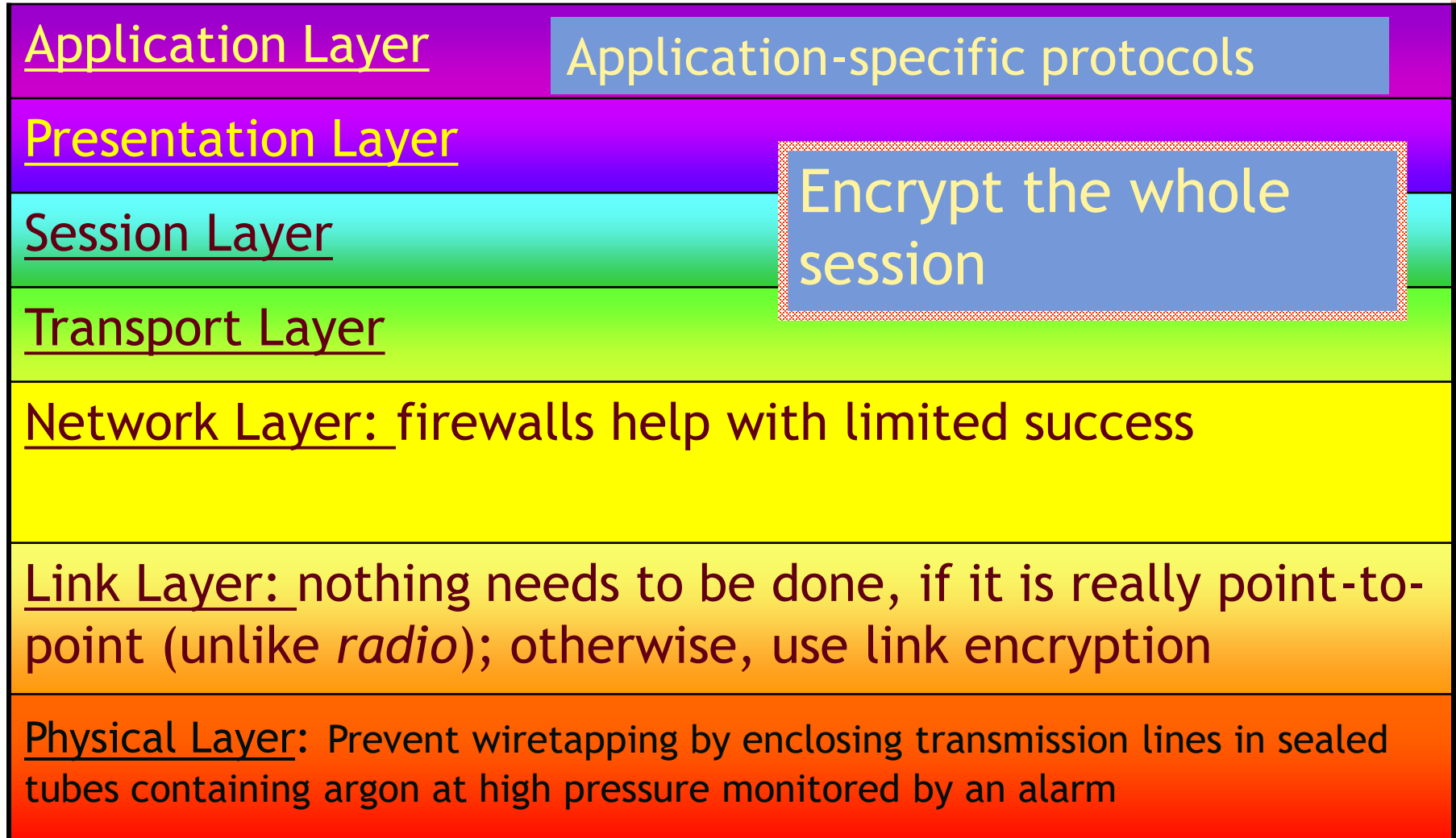


# COMMON SECURITY SOLUTIONS USED IN CLOUD



We will focus on *IPsec* and *TLS*, which have become indispensable for authentication and *Oauth*, which is considered for authorization

# NETWORK SECURITY IN THE PROTOCOL STACK



# TWO VIEWS IN THE INTERNET CAMP

Security must be end-to-end, and for this reason alone must be implemented in the Application Layer (which will make plaintext unavailable to operating systems)

Problem: Then all applications must be re-written and... how many people really understand security to rewrite them?

Security must be implemented in the Network Layer without users ever approaching it!

Problem: Even though this view has prevailed, a truly network-layer implementation proved to be impossible, and Internet principles had to be violated.

# IP SECURITY PROTOCOL (*IPSEC*)

*IPsec* is a framework for multiple

- Services
  - confidentiality, integrity, protection from replay—among the major ones
- Algorithms
  - to make it algorithm-independent (and there is a *Null algorithm*)
- Granularities
  - from a single TCP connection to an aggregate

*IPsec* is... connection-oriented!



# SECURITY ASSOCIATION (SA)

- SA is a *simplex* connection identified by Security Parameters Index (SPI) carried by all packets
- SA is needed because
  - a key must be used for some period of time—the duration of the connection
  - the set up time is amortized among many packets

# ESTABLISHING AN SA

This involves

- Authenticating both ends
- Establishing the key
- Agreeing on cryptographic algorithms
- Initializing the sequence number (which will run through the life of the association)
- Establishing SPI

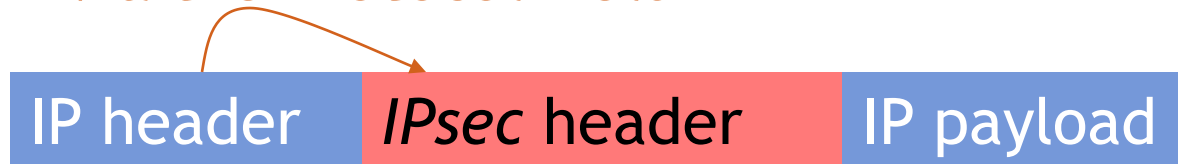
## TWO PARTS OF *IPSEC*

- The *Internet Security Association and Key Management Protocol (ISAKMP)* deals with establishing symmetric keys
  - The main protocol is called *Internet Key Exchange (IKE)*. It has problems, and it is being replaced by *IKE2*
- The other part deals with the headers defined for the two modes of *IPsec* operation
  - Transport mode and
  - Tunnel mode

# TRANSPORT- AND TUNNEL MODES

- Transport mode

Via the *Protocol* field



- Tunnel mode



Useful for

- terminating at other than end-user locations (e.g., firewalls)
- aggregation to prevent *traffic analysis*

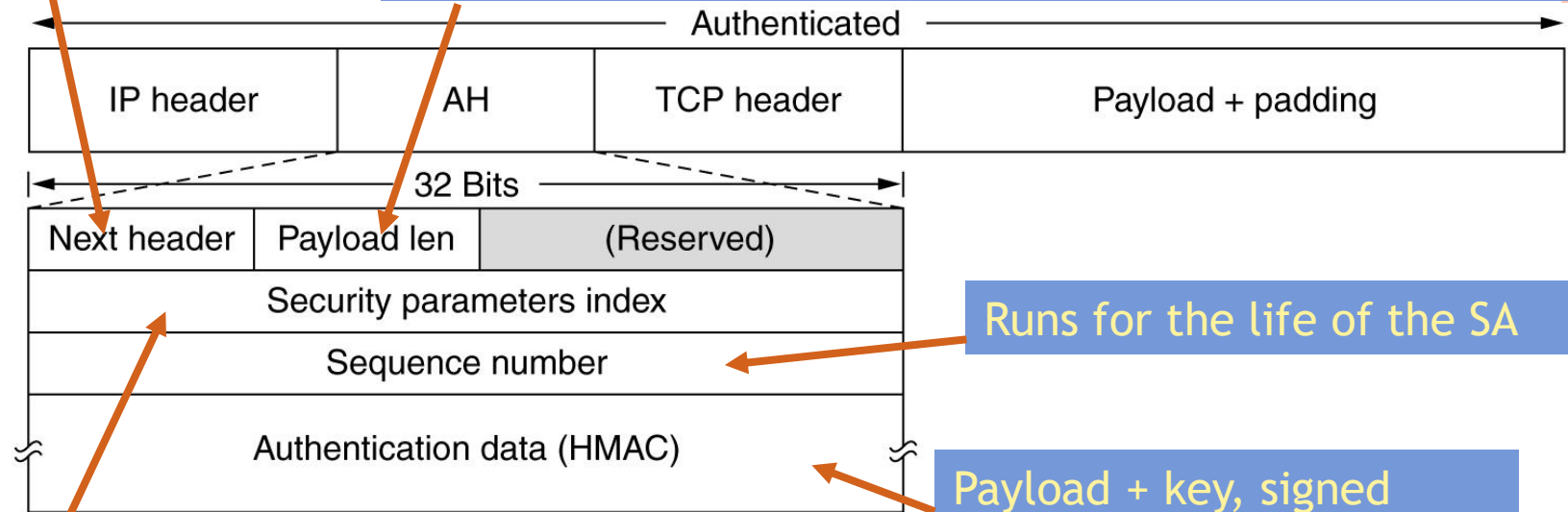
## TWO (HISTORICAL) HEADERS

- The *Authentication Header (AH)* deals only with integrity checking but *not* confidentiality; hashed message authentication code (HMAC) covers only immutable IP fields (not TTL)
- The *Encapsulating Security Payload (ESP)* supports both HMAC integrity and full confidentiality. In a way, it makes AH superfluous
- Both, of course, provide authentication

# AUTHENTICATION HEADER (AH) (IPv4 TRANSPORT MODE)

Stores the value that *IP Protocol* field had

Number of 32-bit words in AH minus 2



Runs for the life of the SA

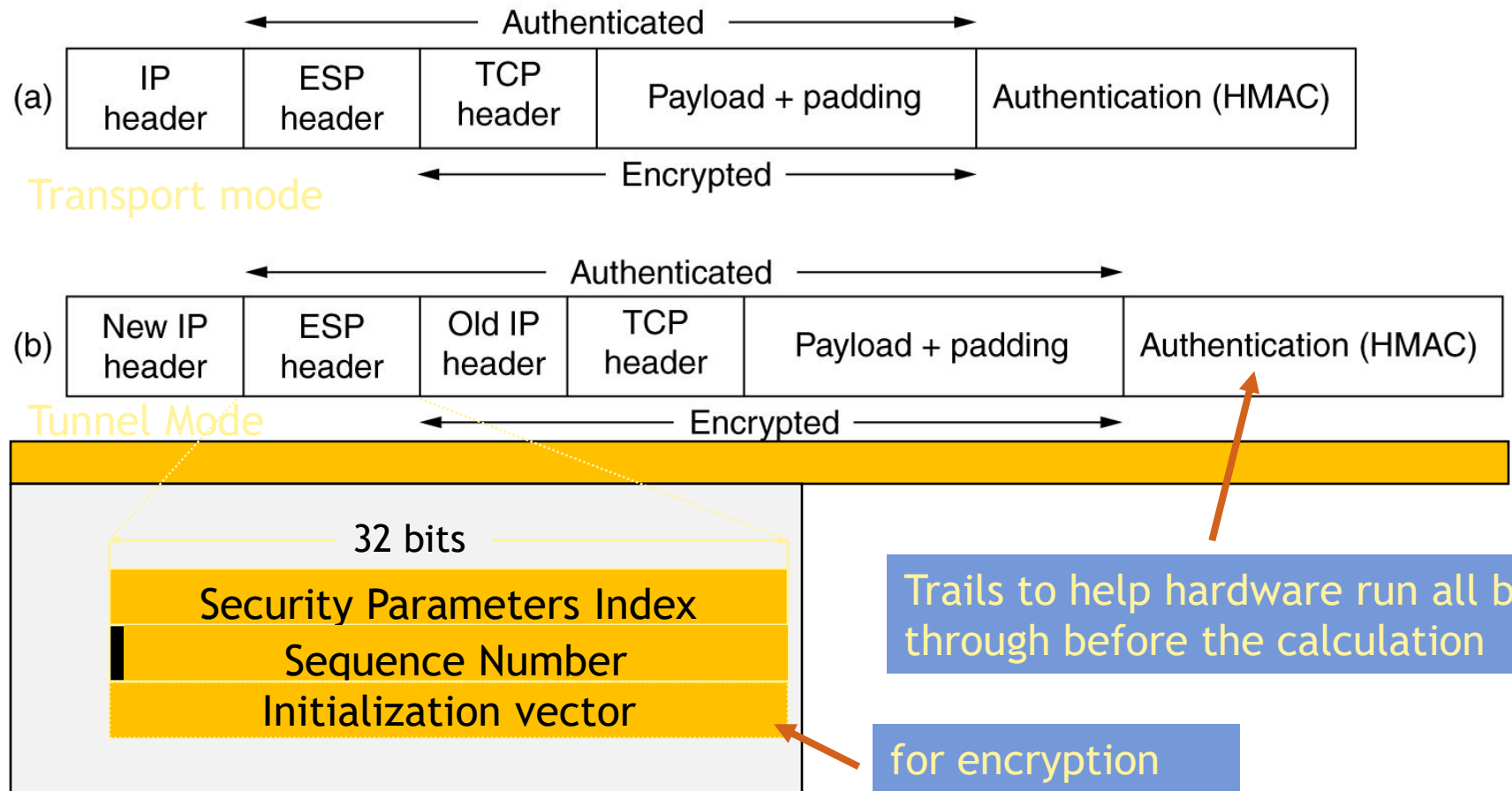
Payload + key, signed

The “virtual circuit number” associated with the shared key

(After A. Tanenbaum)

30

# ENCAPSULATING SECURITY PAYLOAD (ESP) HEADER



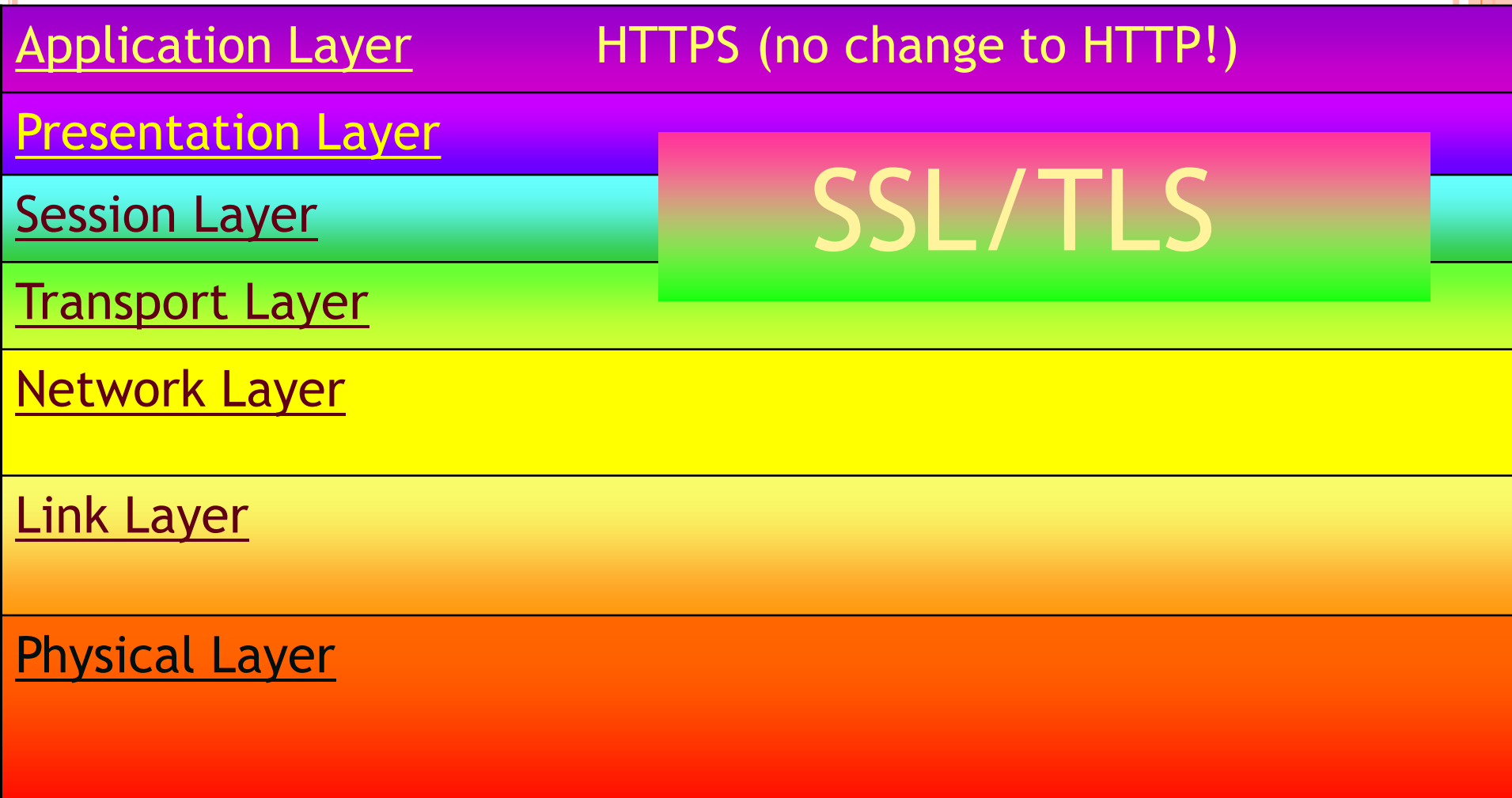
(After A. Tanenbaum)

# TRANSPORT LAYER SECURITY (TLS) BACKGROUND

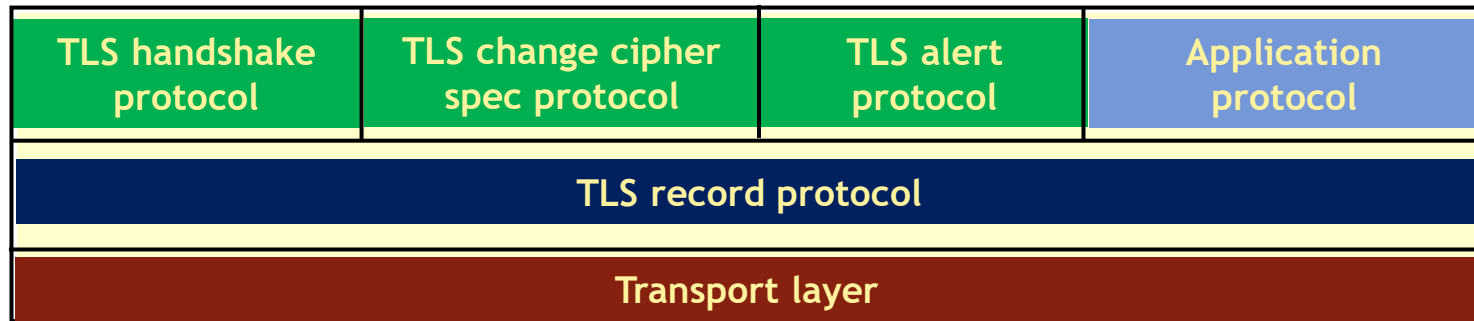
- TLS has been designed to
  - protect a *client-server* session
  - run over TCP (or, actually, any reliable transport protocol)
  - be used by communicating process at a *user-level* (rather than *system-level* so as to avoid operating system changes)
- It has been developed by the IETF based on the *Netscape* Secure Socket Layer (SSL) v3
- The latest version is TLS v1.2 (RFC 5246), which supersedes the previous two versions v1.0 (RFC 2246) and v1.1 (RFC 4346)



# POSITION OF THE SSL/TLS IN THE OSI REFERENCE ARCHITECTURE



# TLS ARCHITECTURE



## Application protocol

**TLS *handshake* protocol:** Signaling for session initiation, including authentication and negotiation of session parameters (e.g., random numbers and the cipher suite)

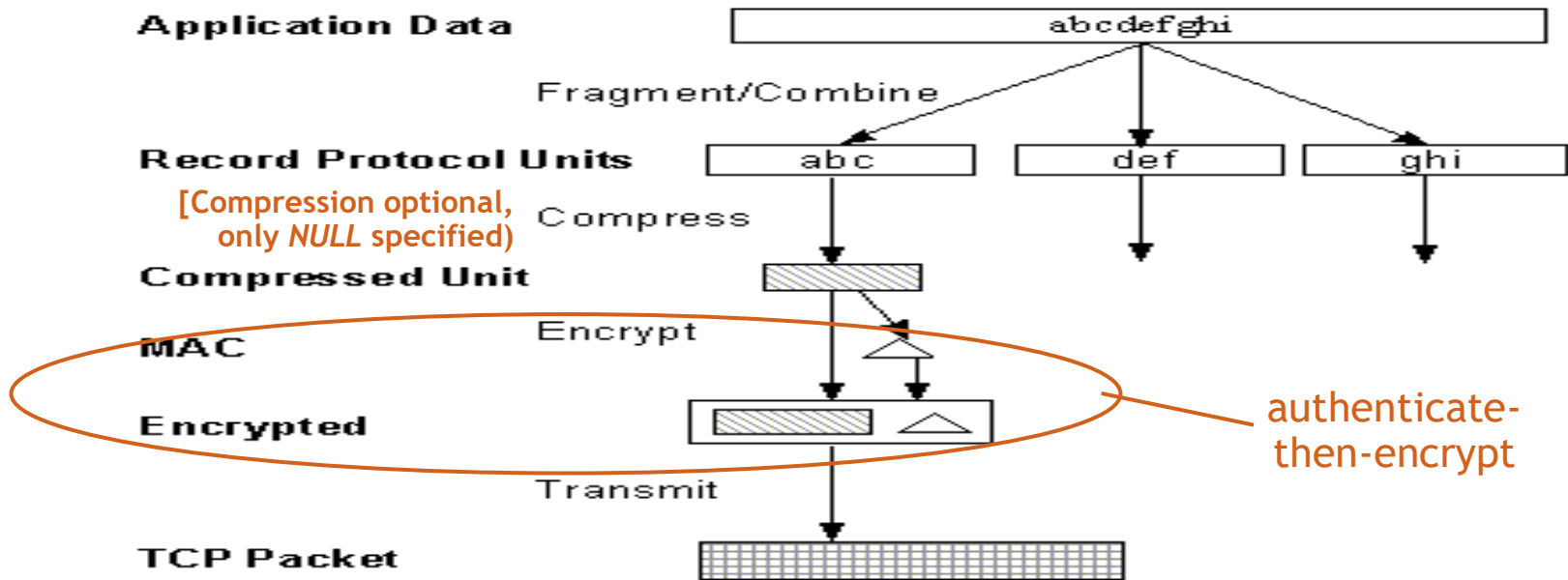
**TLS *change cipher spec* protocol:** Signaling to use the newly negotiated cipher suite and session keys for protecting ensuing records

**TLS *alert* protocol:** Signaling errors during a session

**TLS *record* protocol:** Processing application data for transmission, including digest computation and encryption

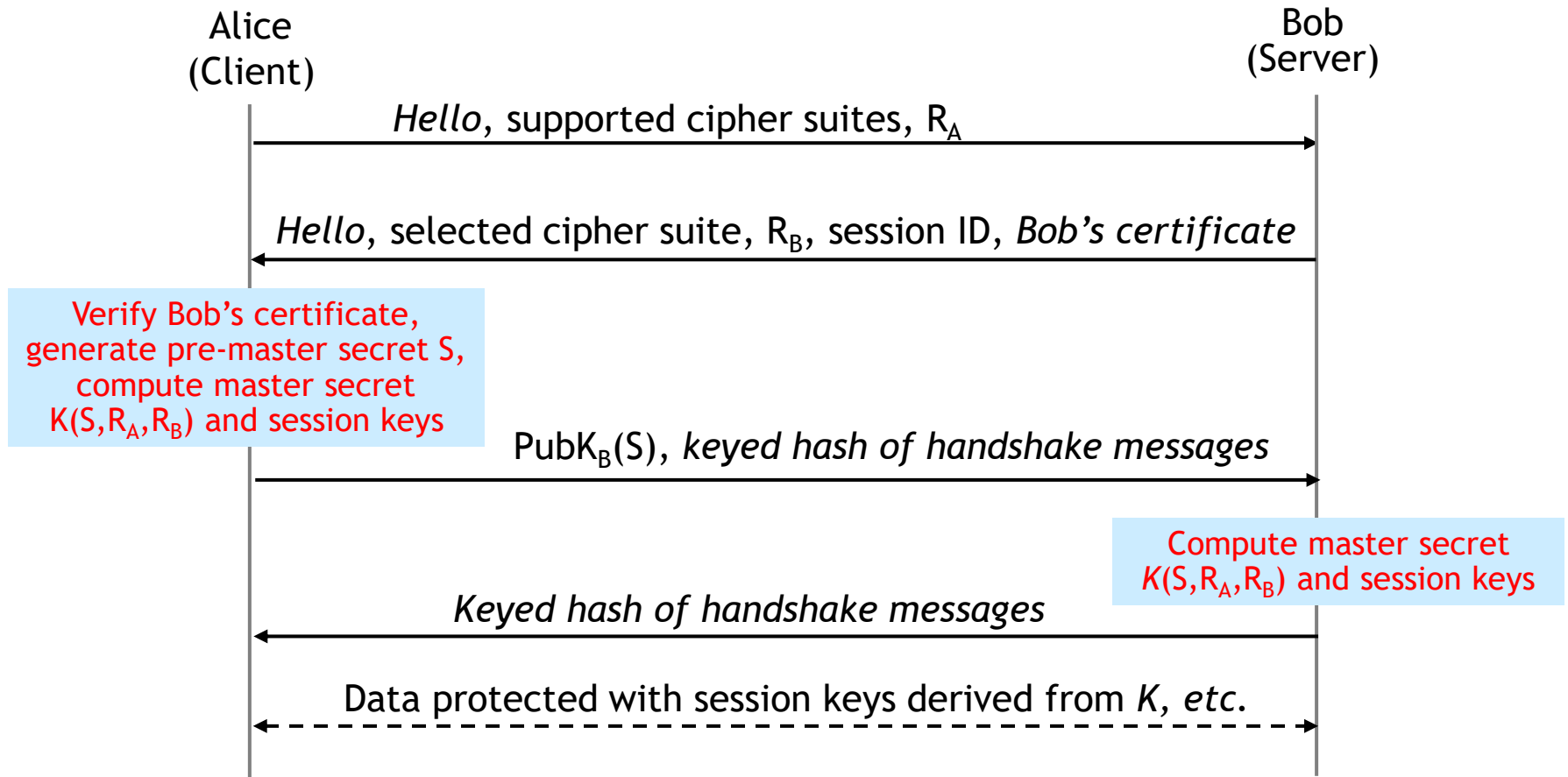
**Transport layer:** Reliable transport (e.g., TCP or SCTP)

# TLS RECORD PROTOCOL



Session keys and algorithms for compression, encryption, and MAC are negotiated during the TLS handshake

# OUTLINE OF A TYPICAL TLS SESSION SETUP



Only Bob is authenticated (based on the certificate).  
Alice needs to be authenticated at the application layer.

A cipher suite (covering key exchange, encryption and MAC) is negotiated.

# AUTHORIZATION PROTOCOLS



	<i>Object 1</i>	<i>Object 2</i>	<i>Object 3</i>	<i>Object 4</i>
<i>Subject 1</i>	read, write, execute	read, write, execute	read	read
<i>Subject 2</i>	read, execute	read, execute	null	null
<i>Subject 3</i>	read, execute	read	read, write	write

## ACCESS CONTROL MATRIX



# CAPABILITY LISTS

*Subject 1* → Object 1: read, write, execute; Object 2: read, write, execute; Object 3: read; Object 4: read

*Subject 2* → Object 1: read, execute; Object 2: read, execute

*Subject 3* → Object 1: read, execute; Object 2: read; Object 3: read, write; Object 4: write

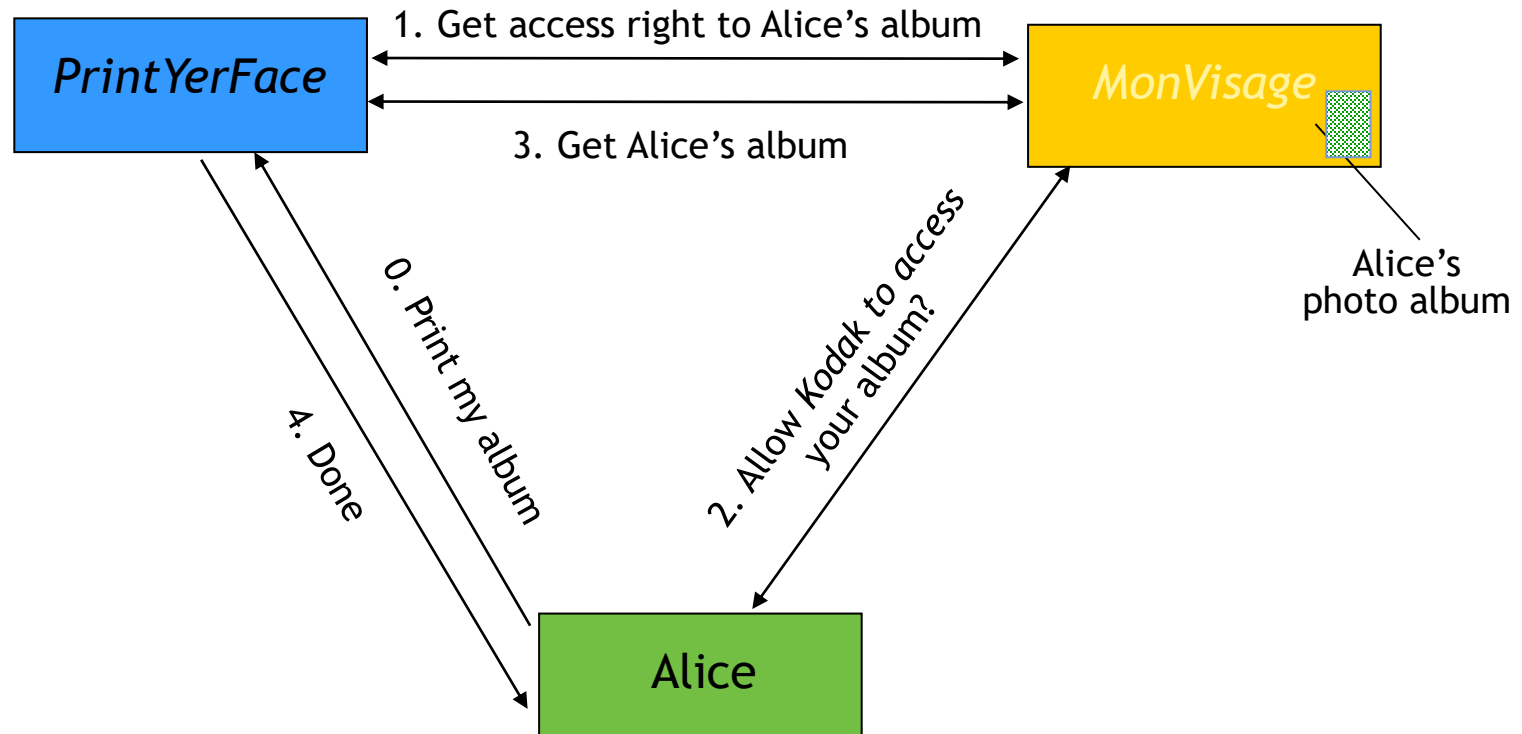


# *OAuth* DEVELOPMENT HISTORY

- The community effort (<http://oauth.net/>) was inspired by OpenID and started in late 2006
- The IETF formed a working group (<http://datatracker.ietf.org/wg/oauth/>) to standardize *OAuth* in May 2009
  - It has published RFC 5849 (*Informational*) based on *OAuth* Core 1.0 Revision A
  - Current focus is on development of *OAuth* 2.0 taking into account RFC 5849 and the Web Resource Authorization Protocol (WRAP)
  - The effort has an expanding support base, including telecom operators and over-the-top service providers

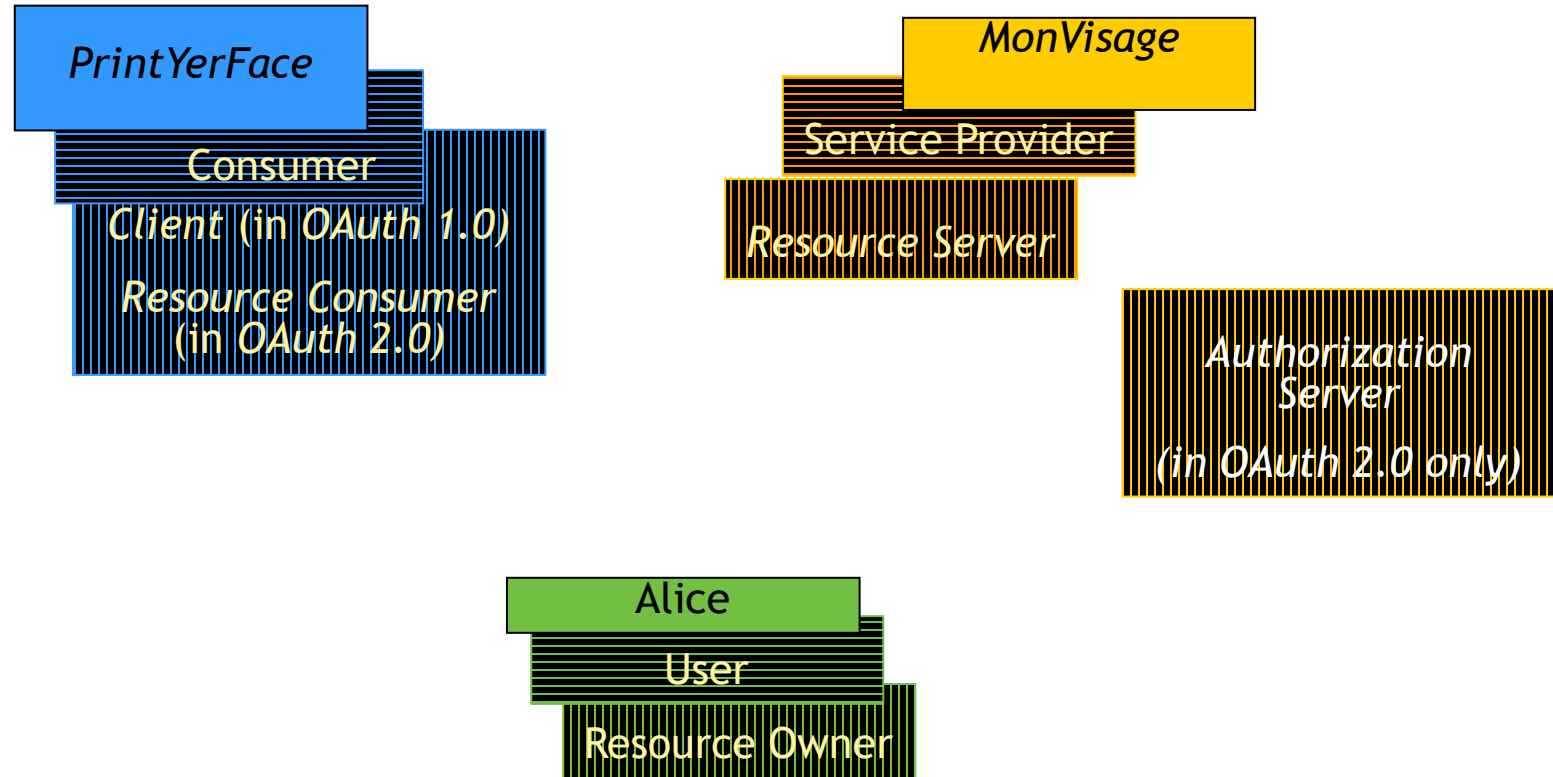


# OPEN AUTHORIZATION (*OAUTH*) PROTOCOL



Alice grants *PrintYerFace* access to her album at *MonVisage* without revealing her credentials

# *OAUTH* TERMINOLOGY

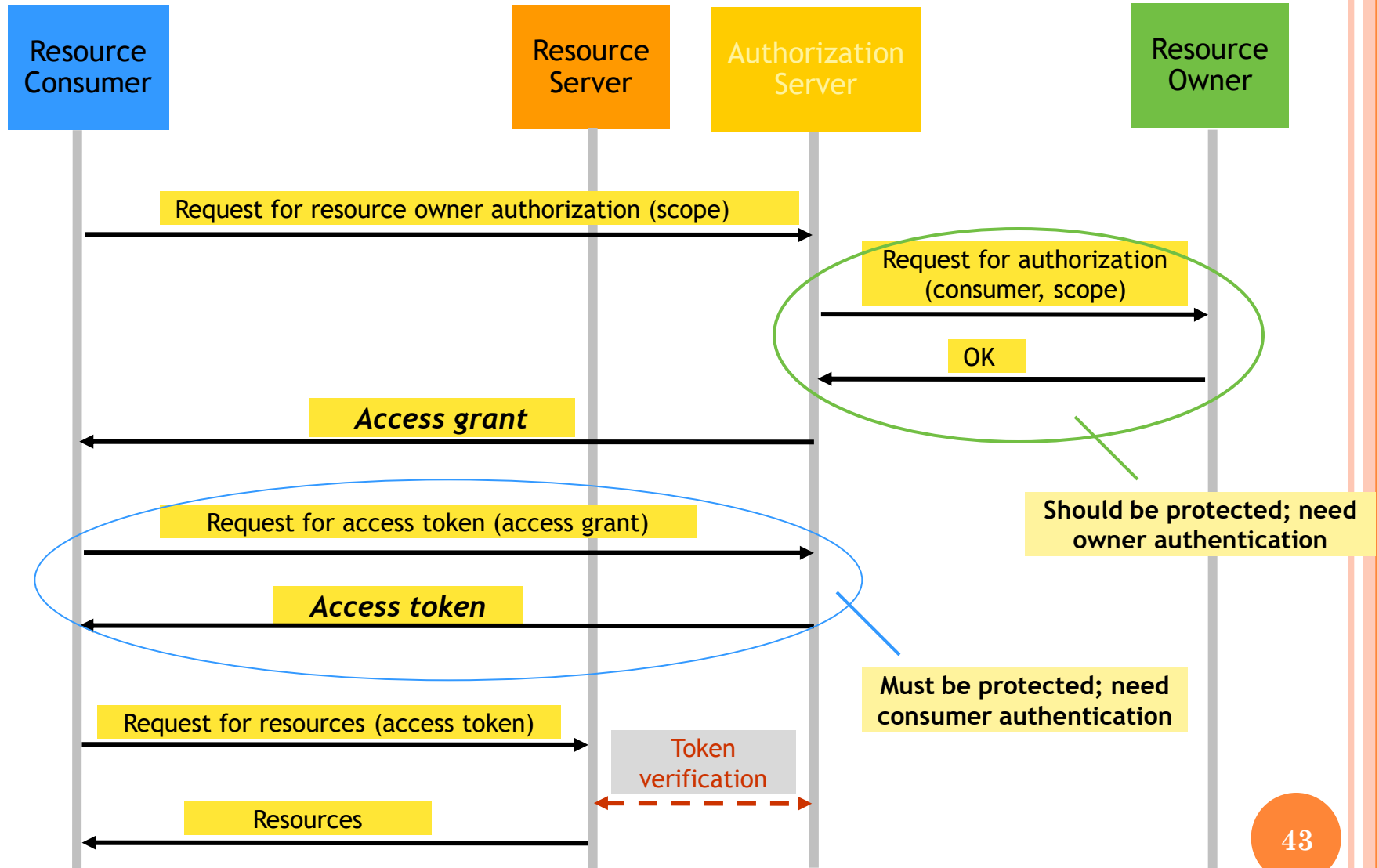


community term



IETF term

# *OAUTH* 2.0 CONCEPTUAL PROTOCOL VIEW



# IDENTITY MANAGEMENT IN THE CLOUD



# PROTECTION OF *DATA AT REST*

- In SaaS and PaaS, protection for data is expected to be performed by the Cloud *provider*
- In IaaS, protection of data is also an essential responsibility of the Cloud *consumer*
- Issues to consider for the decision on where to maintain encryption keys:
  - access control (authorization)
  - availability of the keys,
  - compliance (whether it is necessary to ***escrow*** the keys with the provider)

**Case study: Amazon's S3 storage supports encryption options and provides an option for key escrowing**

# PROTECTION OF *DATA IN MOTION*

- Access to data
  - End-user authentication (including key establishment)
  - End-user authorization for accessing specific records
- Data transfer among VMs
- Data transfer among data centers
  - Secure tunnels (possibly provided by the *Network as a Service* provider)

# CLOUD DATA STORAGE CONSIDERATIONS

- Can an enterprise store *all* its data in the Cloud?
  - Who (the enterprise or Cloud provider) is responsible for the private or business-sensitive information?
  - How to ensure compliance with the regulations?
  - Does the Cloud Provider guarantee data security “*at rest*” and “*in motion*”?
  - Who (enterprise or Cloud provider) is responsible for key distribution and management?
  - Who is responsible for scheduling and carrying back-ups?
- What happens if the Cloud provider goes out of business?

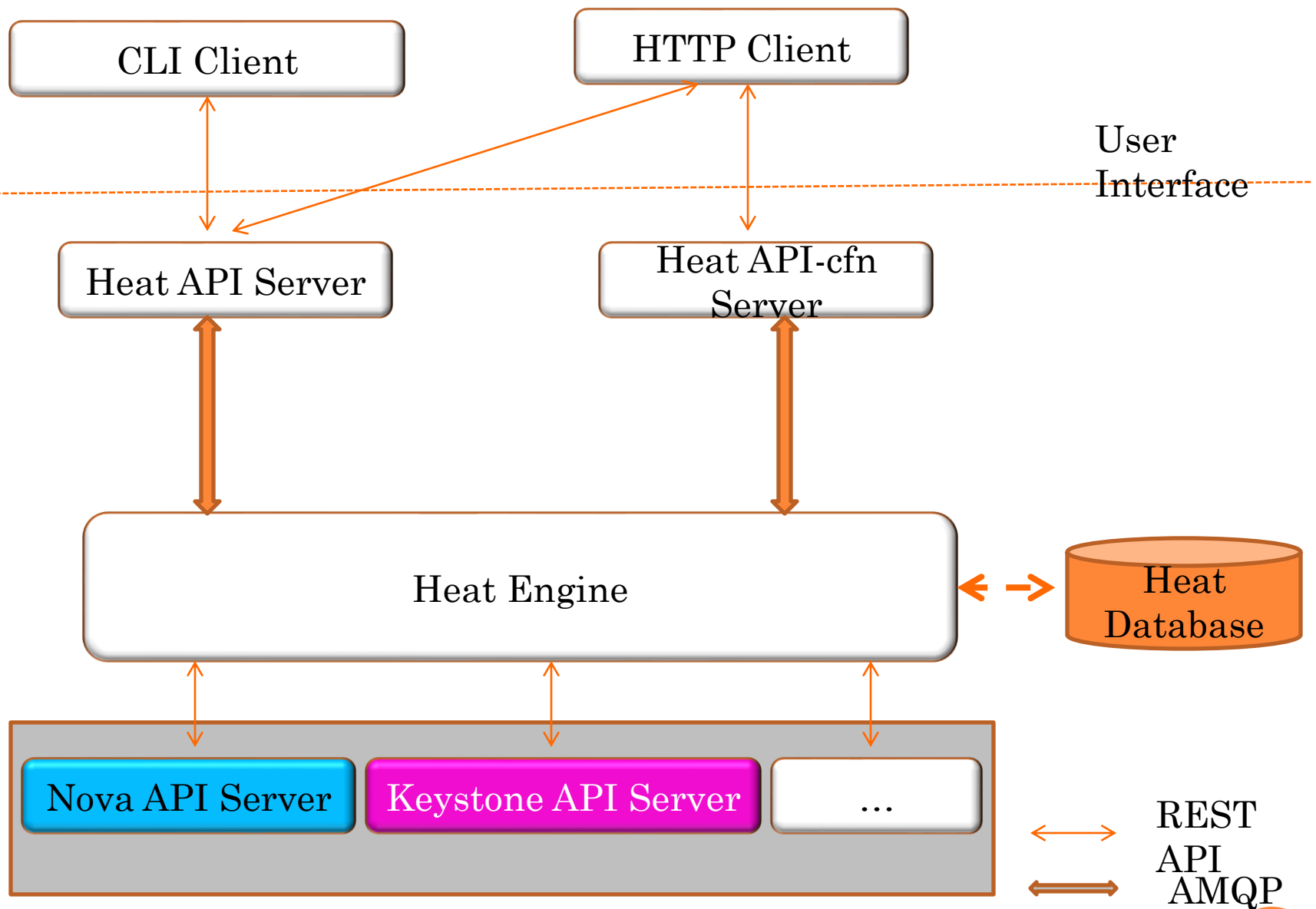
The data storage problem should never have a single *all-or-nothing* solution!

# IDENTITY MANAGEMENT IN OPENSTACK

- *Keystone* is the *gatekeeper*
- *Keystone*
  - provides centralized authentication and authorization service
  - supports the constructs of domain, project, group, and role
  - generates and validates *tokens*

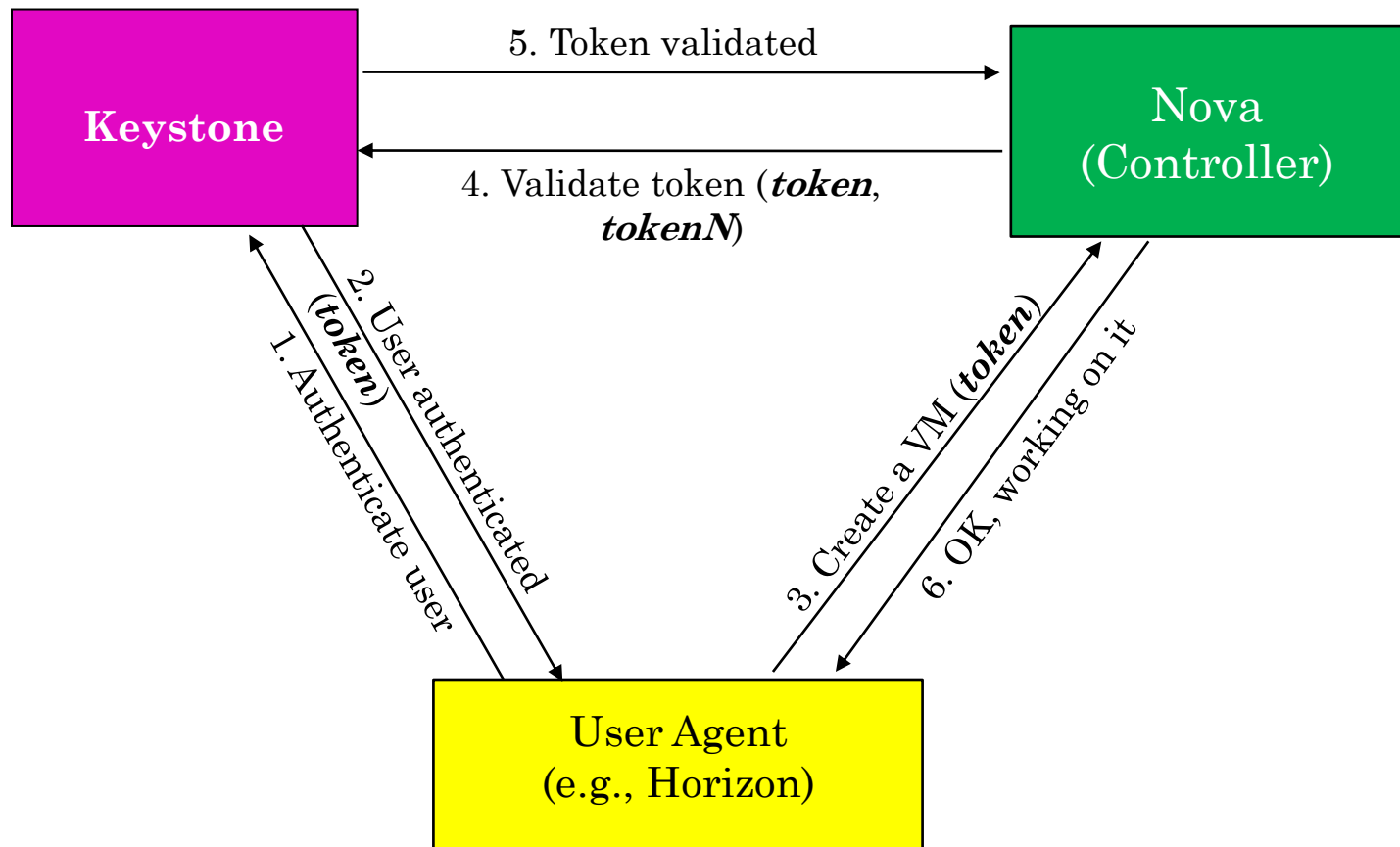




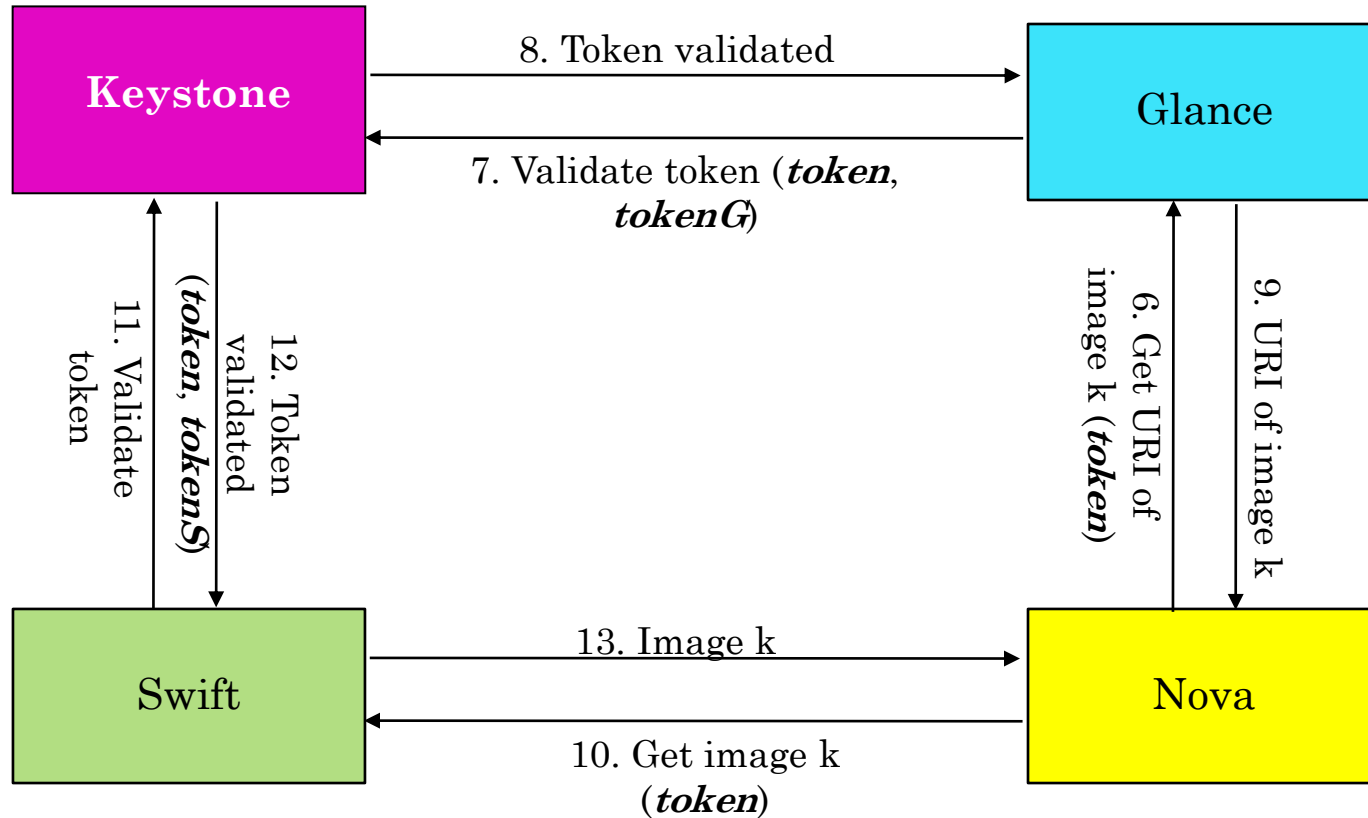


The *Heat* computing architecture

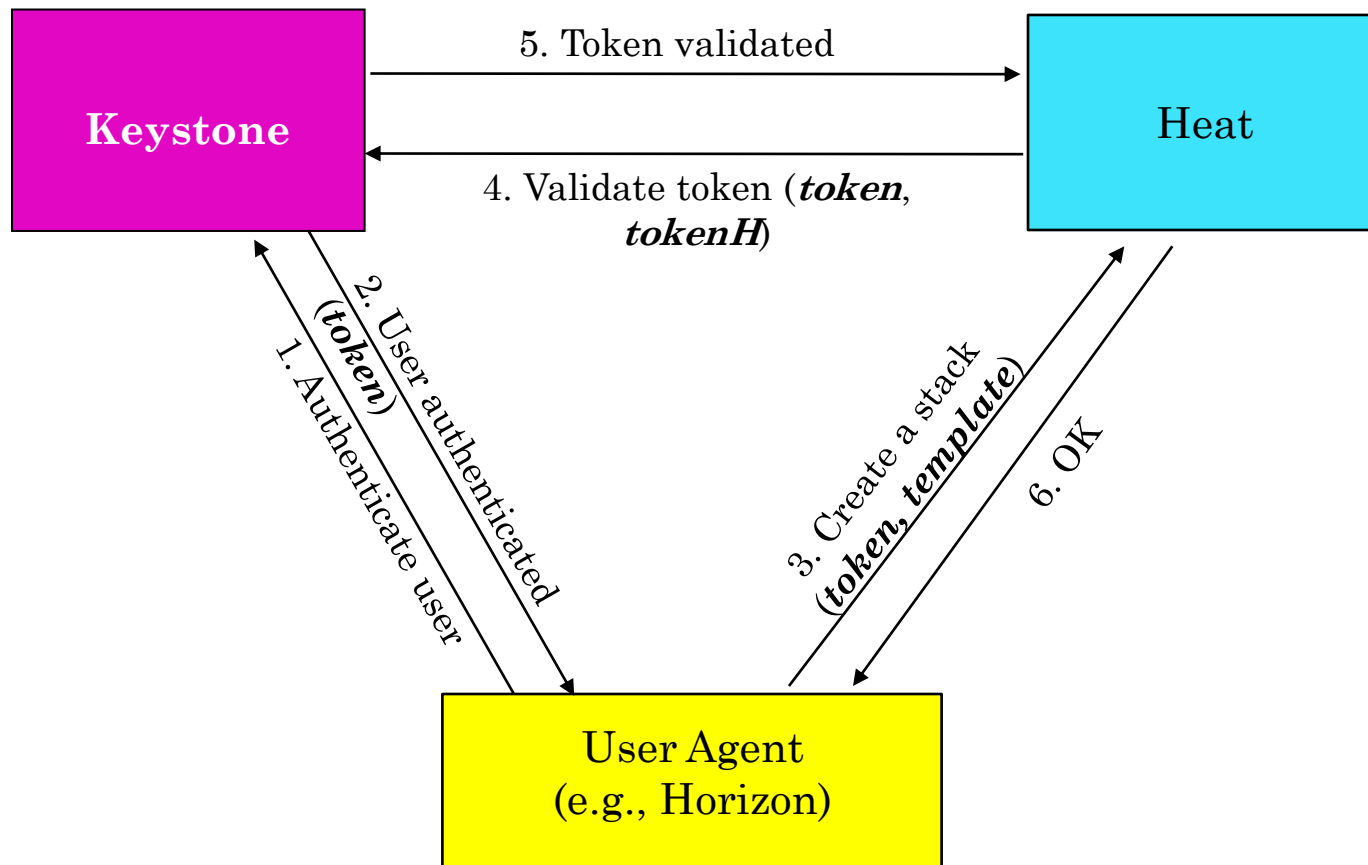
# A SIMPLIFIED WORKFLOW FOR VM PROVISIONING IN OPENSTACK



## ADDITIONAL STEPS FOR VM PROVISIONING



## A SIMPLIFIED WORKFLOW FOR AUTO-SCALING

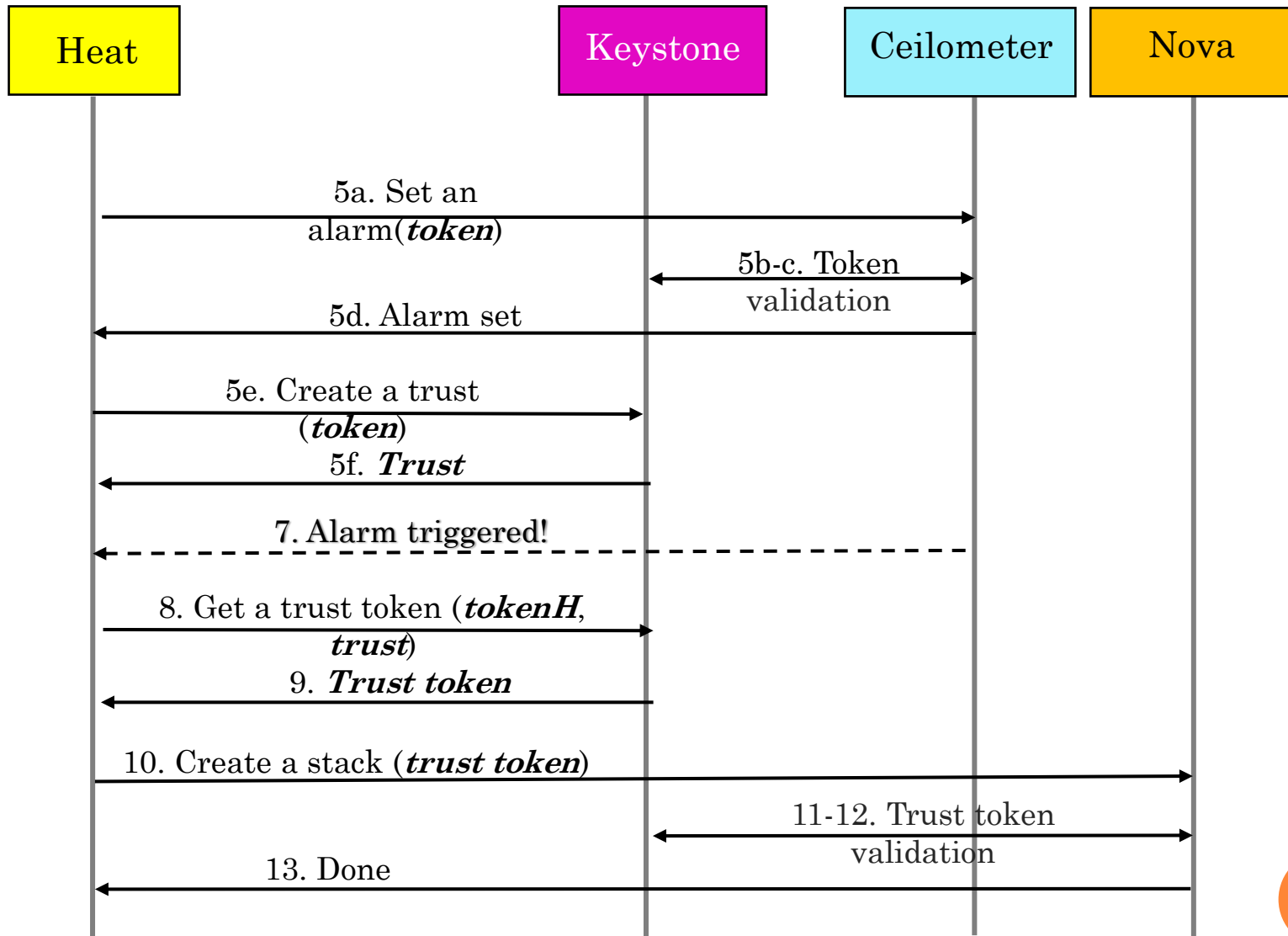


## AN EXAMPLE TOKEN (UNSIGNED)

```
{
  "token": {
    "expires_at": "2015-05-27T22:52:58.852167Z",
    "issued_at": "2015-05-27T21:52:58.852167Z",
    "methods": ["password"],
    "domain": {
      "id": "3a5140aecd974bf08041328b53a62458",
      "name": "Wonderland"
    },
    "roles": [{
      "id": "9fe2ff9ee4384b1894a90878d3e92bab",
      "name": "admin"
    }],
    "user": {
      "domain": {
        "id": "3a5140aecd974bf08041328b53a62458",
        "name": "Wonderland"
      },
      "id": "3ec3164f750146be97f21559ee4d9c51",
      "name": "Alice"
    }
  }
}
```



## ADDITIONAL STEPS FOR AUTO-SCALING

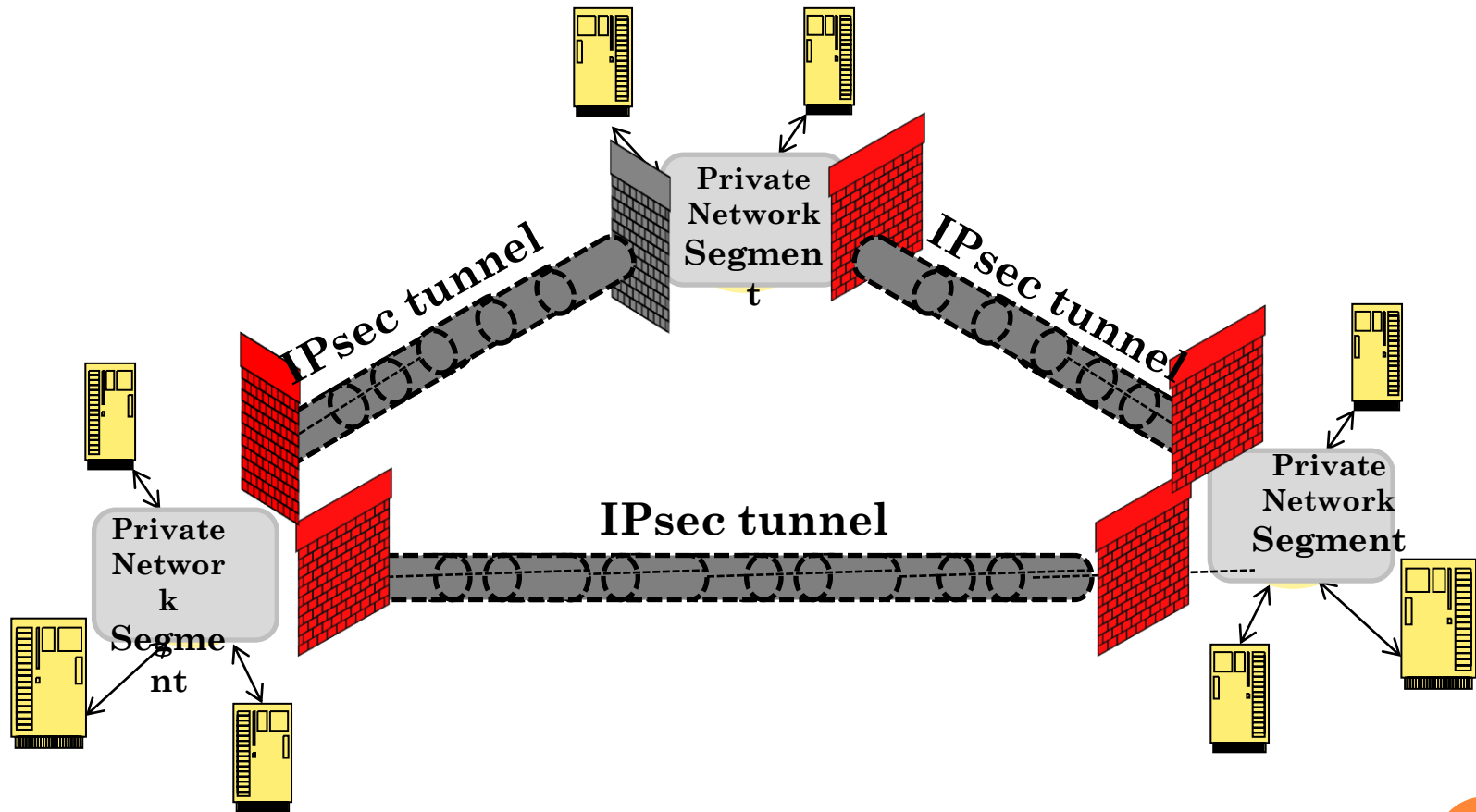


# AN EXAMPLE TRUST

```
{
  "trust": {
    "expires_at": "2016-05-27T21:52:58.852167Z",
    "id": "c703057be878458588961ce9a0ce686b",
    "impersonation": true,
    "project_id": "fb49a0ecd60c4d2092643b4cfe272106",
    "remaining_uses": null,
    "roles": [{
      "id": "9fe2f79ee4384b1894a9083bd3e92bab",
      "name": "admin"}
    ],
    "trustee_user_id": "29beb2f1567642eb810b042b6719ea88",
    "trustor_user_id": "3ec3164f750146be97f21559ee4d9c51"
  }
}
```



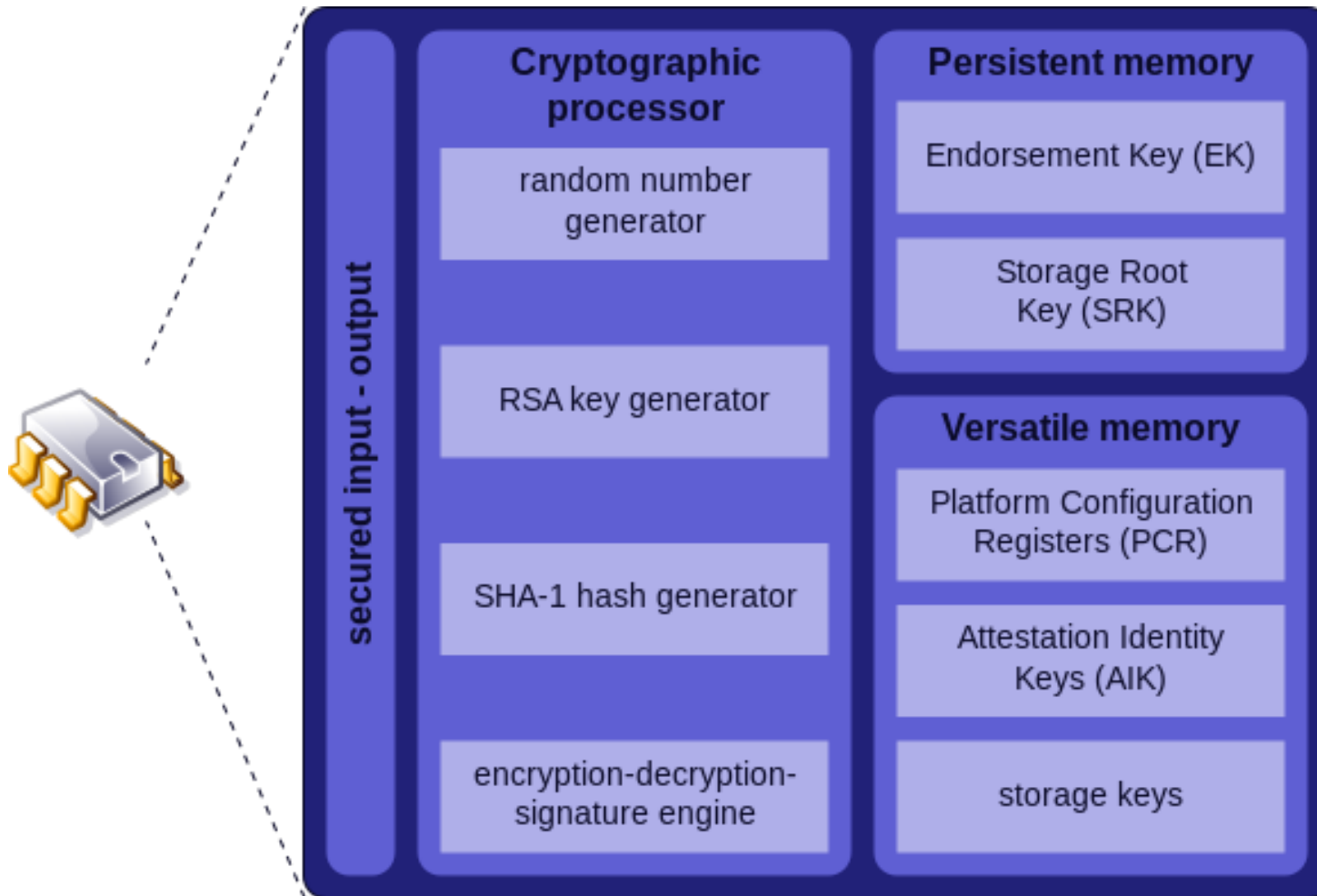
# Layer 3 VPN with firewalls





# TRUSTED PLATFORM MODULE (TPM)

- A standard for a secure cryptoprocessor
  - Specified by the Trusted Computing Group (TCG).
  - Standardized as ISO/IEC 11889
- Major capabilities:
  - ***Remote attestation*** – via an unforgeable hash key summary of the hardware and software configuration to allow a third party to verify that the software has not been changed.
  - ***Binding*** – encrypts data using TPM bind key, a unique RSA key descended from a storage key
  - ***Sealing*** – binding, to which a required state of the TPM to be at so as to unseal the data is specified



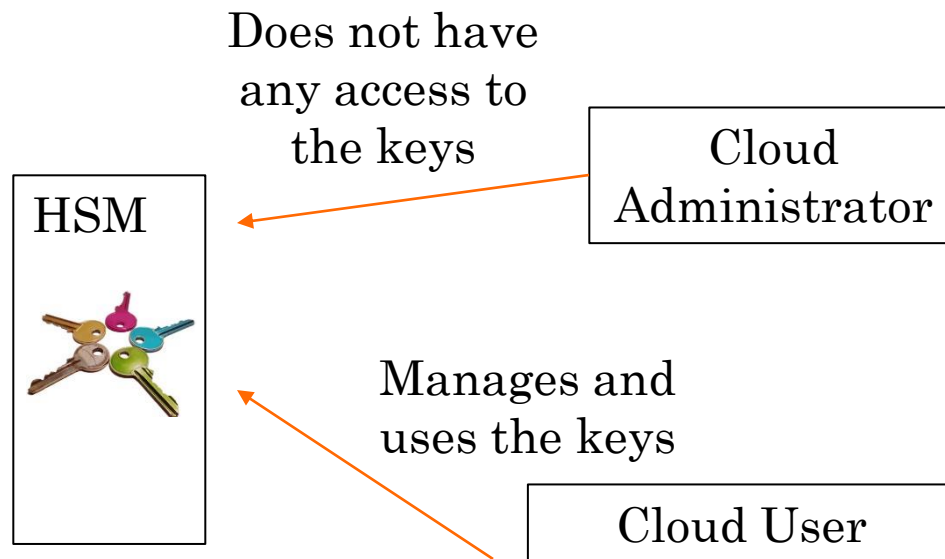
Source: Eusebius (*Guillaume Piolle*,  
<https://commons.wikimedia.org/w/index.php?curid=4809738>

# KEY APPLICATIONS OF TPM IN THE CLOUD

- **Ensuring platform integrity:** TPM forms a *root of trust*. The BIOS, hypervisors, and the OS build on this. The Intel Trusted Execution Technology (TXT) creates a *chain of trust* to attest remotely that a computer has a specified hardware setup and is using specified software..
- **Disk encryption**
- **Password protection**
- **Digital rights management**
- **Enforcement of software licenses**
- **Storing the root certificates**

# HARDWARE SECURITY MODULES

- A physical computing device that stores and manages cryptographic keys and also performs cryptoprocessing. Thus no key escrowing is needed!
- Unlike TPM, HSMs are plug-in devices—often pooled externally.



# THE LAST HOMEWORK

- Read and understand  
<https://aws.amazon.com/cloudhsm/details/>
- Close all Amazon instances that you have created lest you be charged!

