CS 524 A

Introduction to Cloud Computing

Lecture 13:
Identity Management and Security in the Cloud, Part II

Agenda

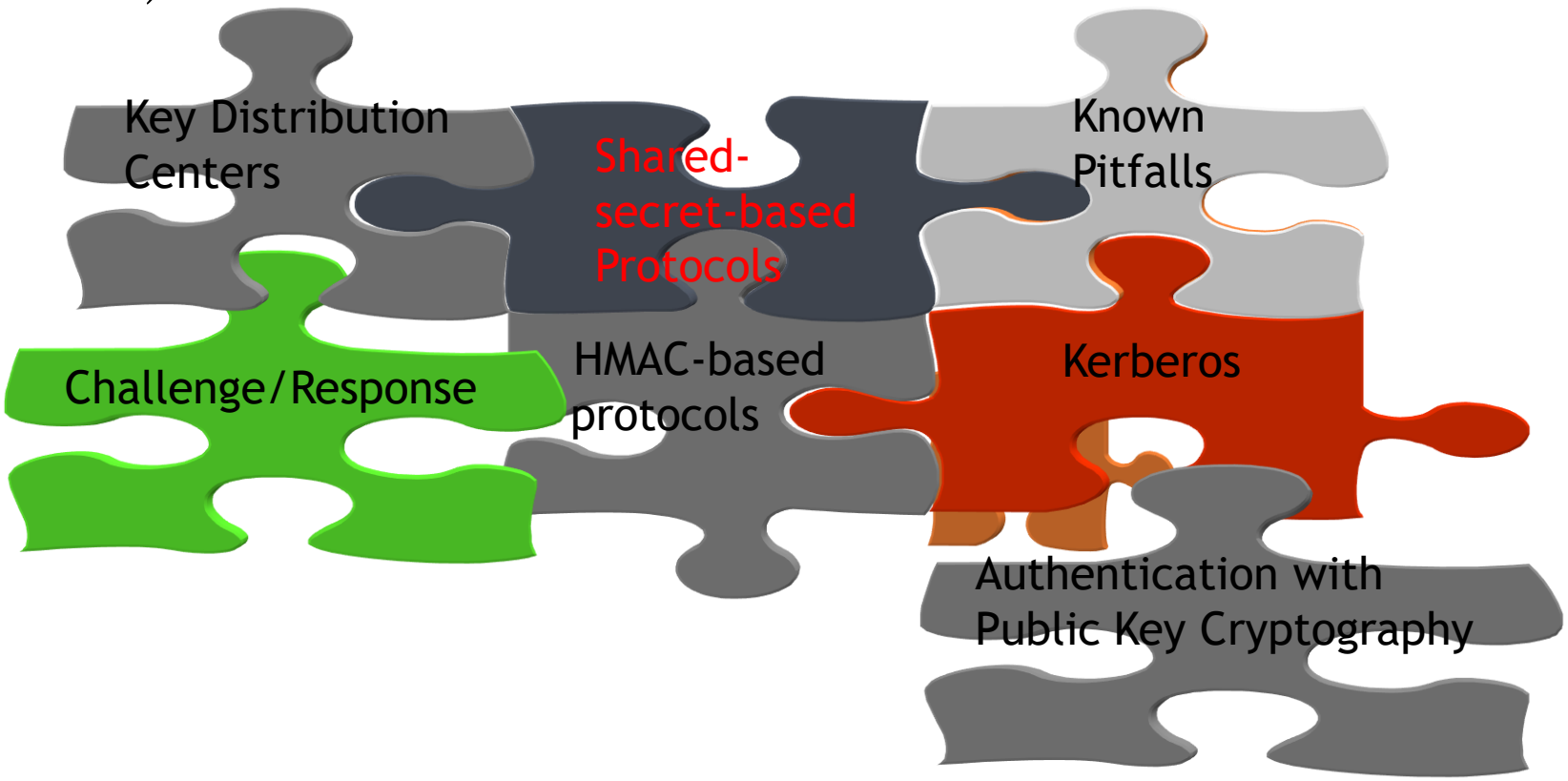
- Authentication
  - Authentication protocol principles
  - Authentication with Kerberos
  - More on authentication with PKI
  - IPsec and TLS
- Authorization (OAuth as an example)
- Identity Management in the Cloud
  - Requirements
  - OpenStack IdM (*Keystone)*
  - Use of TPM (secure boot, attestation,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

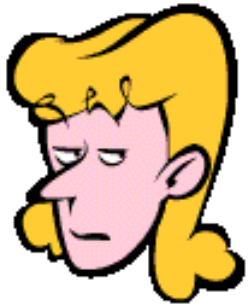
Authentication Protocols

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



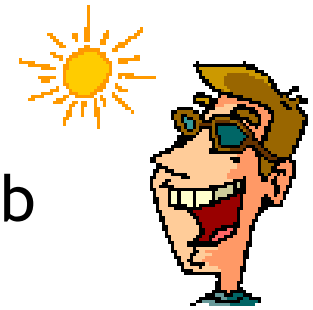
Introduction to Key Players

Alice



Trudy the Intruder

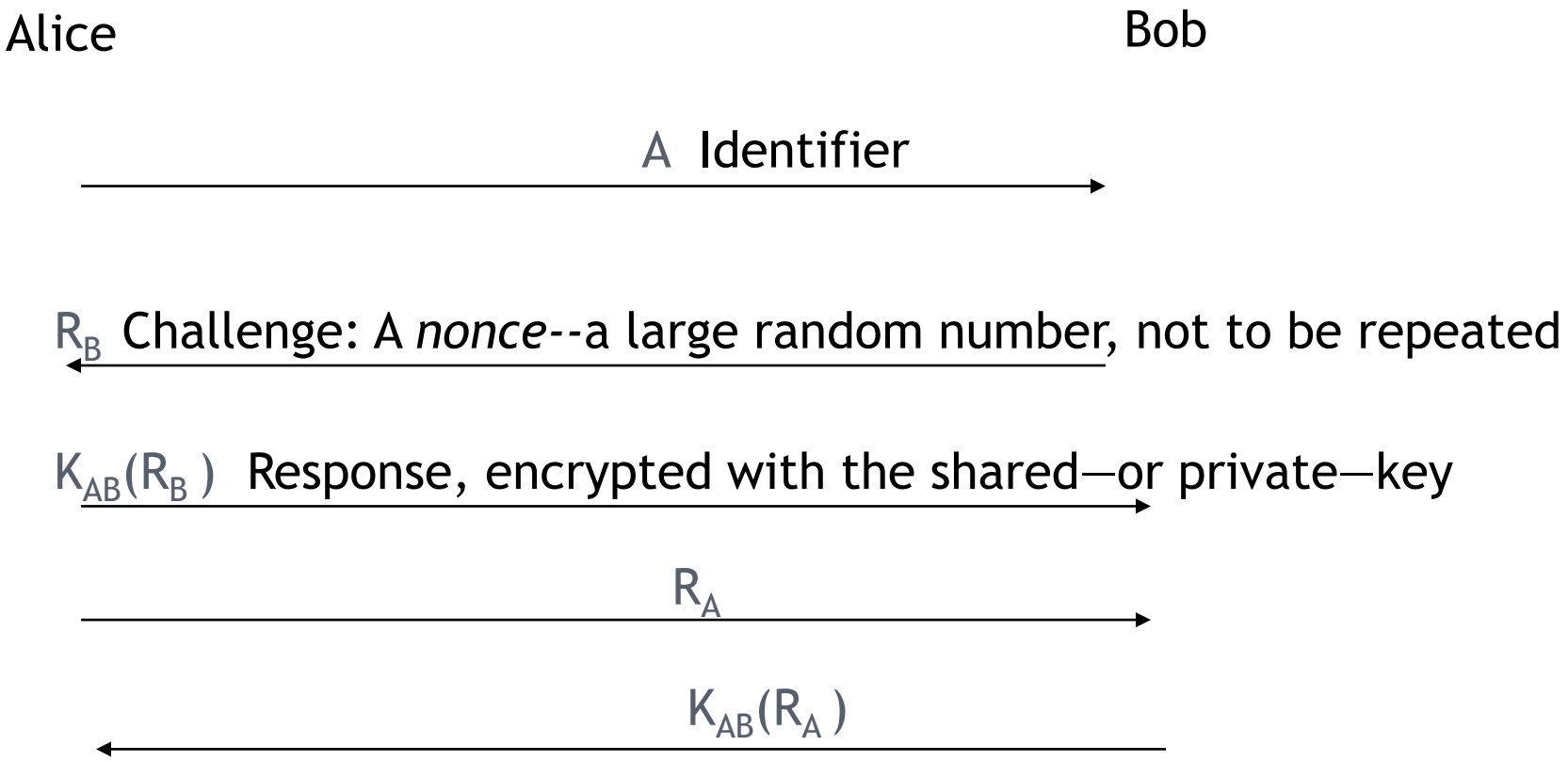
Bob



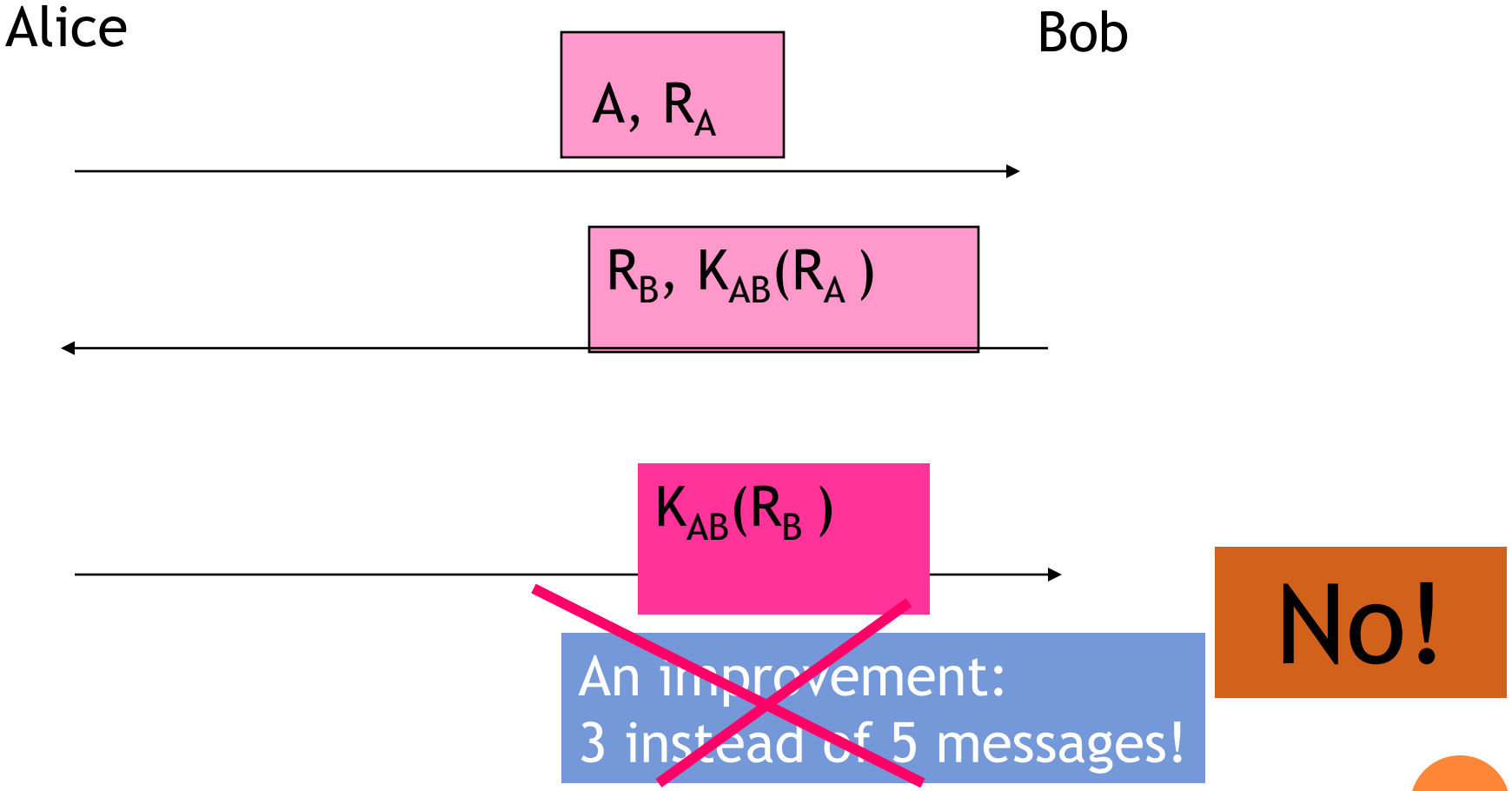
The General Model

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

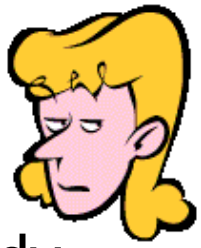
Challenge-Response Protocol (first attempt)

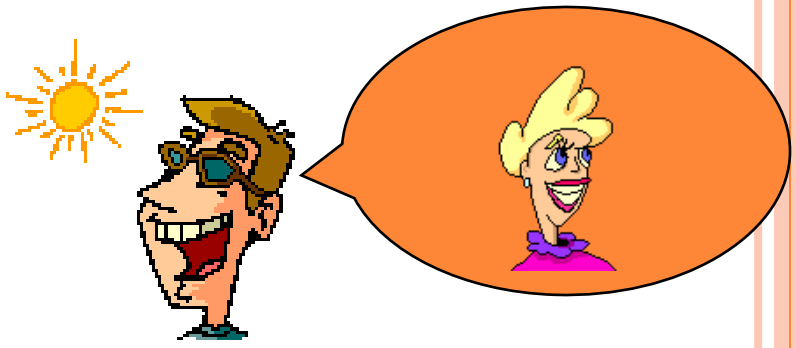


Challenge-Response Protocol (Can we do this faster?)



The Reflection Attack





Trudy

Bob

A, R_T

$R_B, K_{AB}(R_T)$

A, R_B

$R_{B^*}, K_{AB}(R_B)$

$K_{AB}(R_B)$

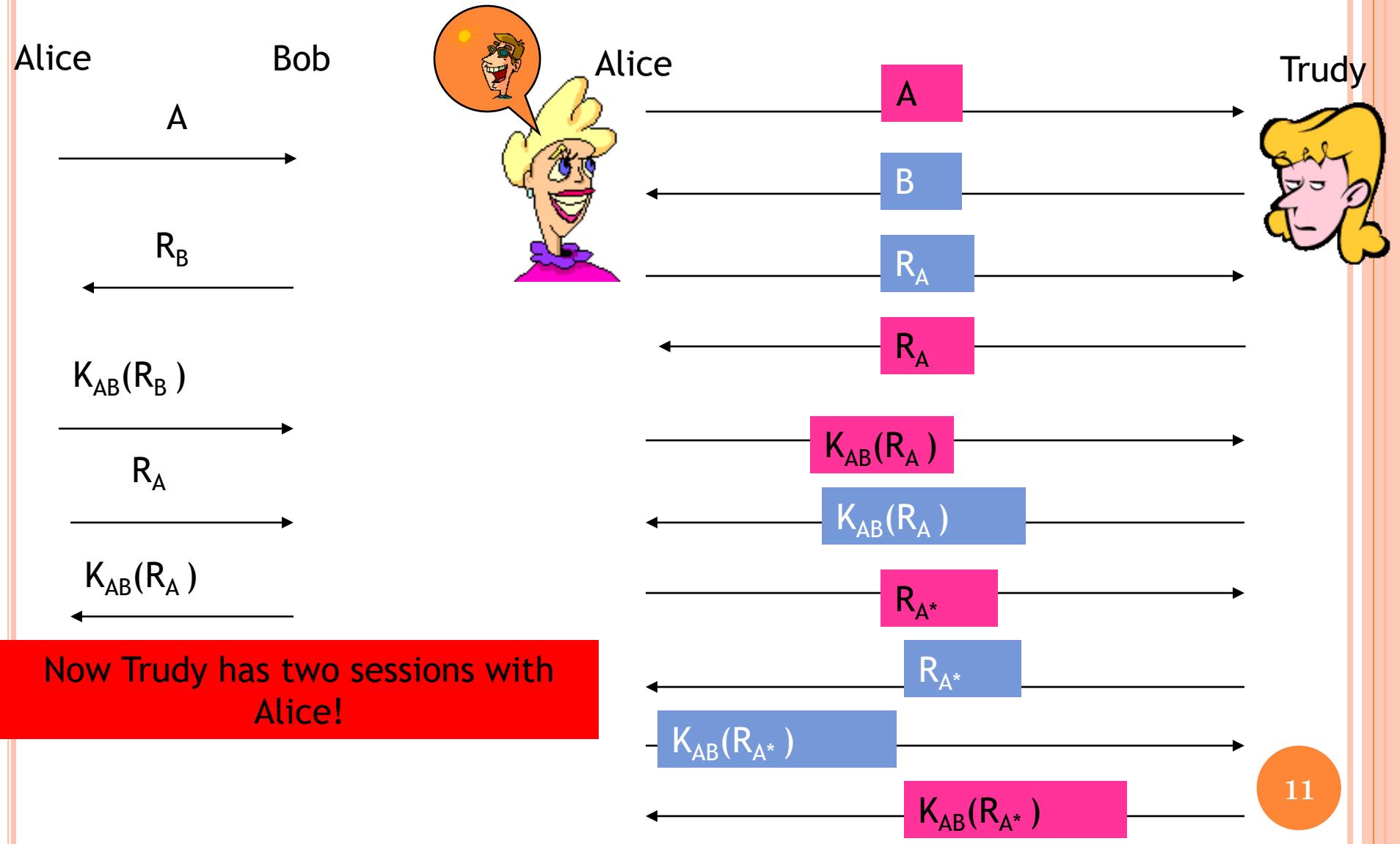
First Session

Second Session

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?



Now Trudy has two sessions with Alice!

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

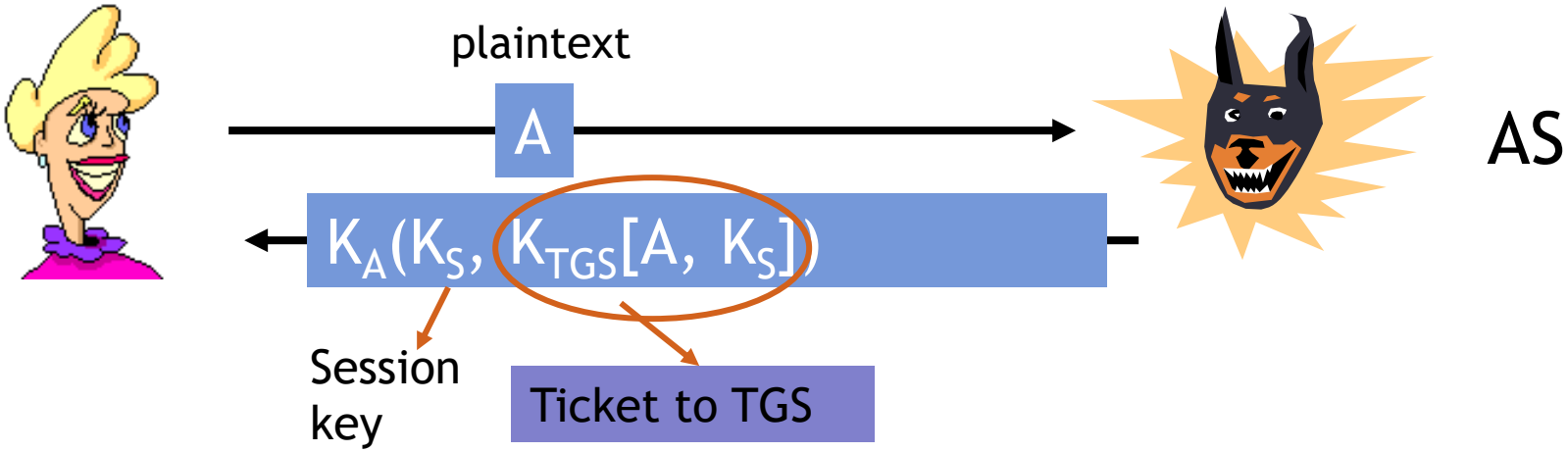
Authentication with Kerberos

- 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.)

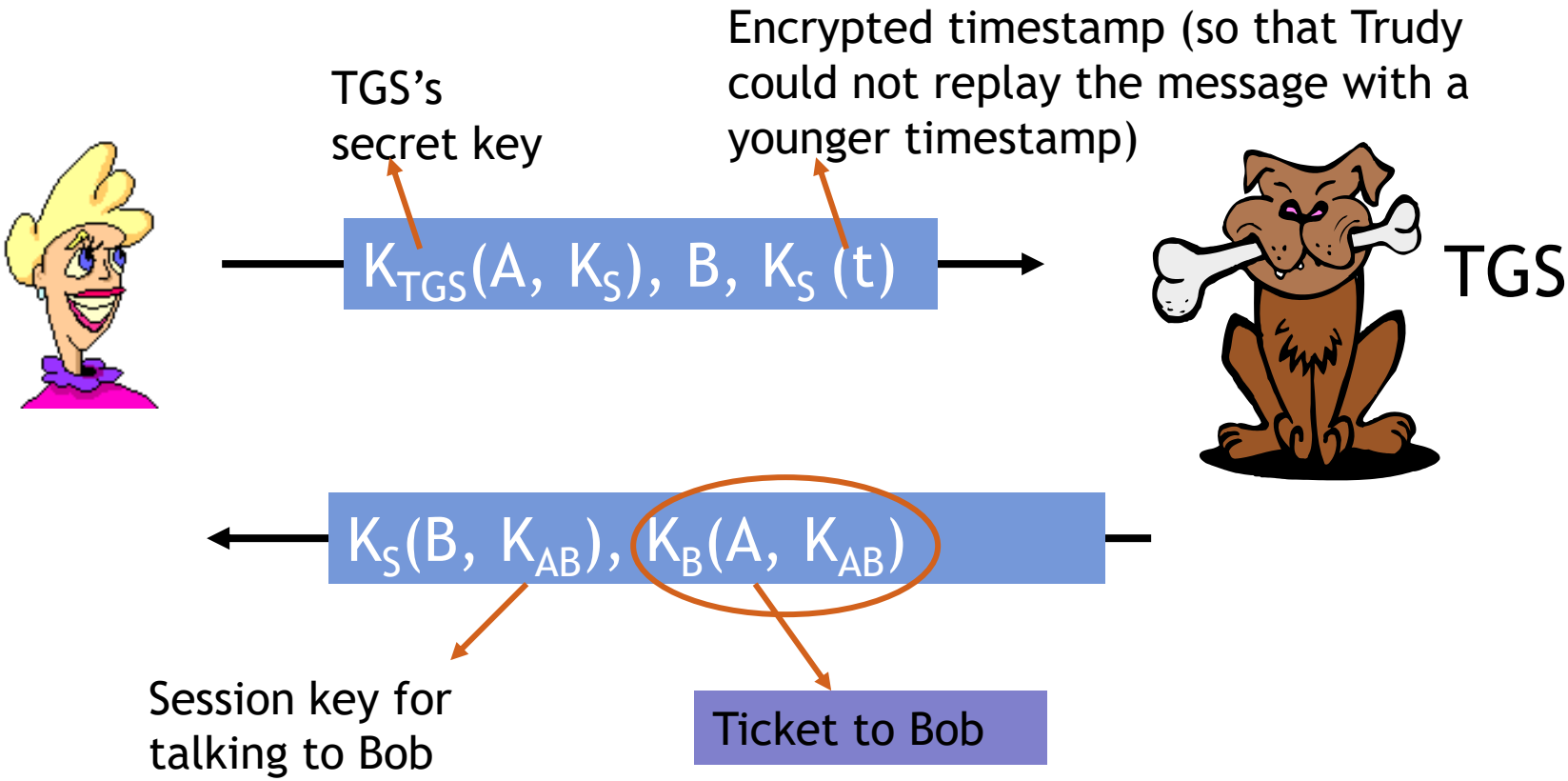
Operation of Kerberos AS



At this point,

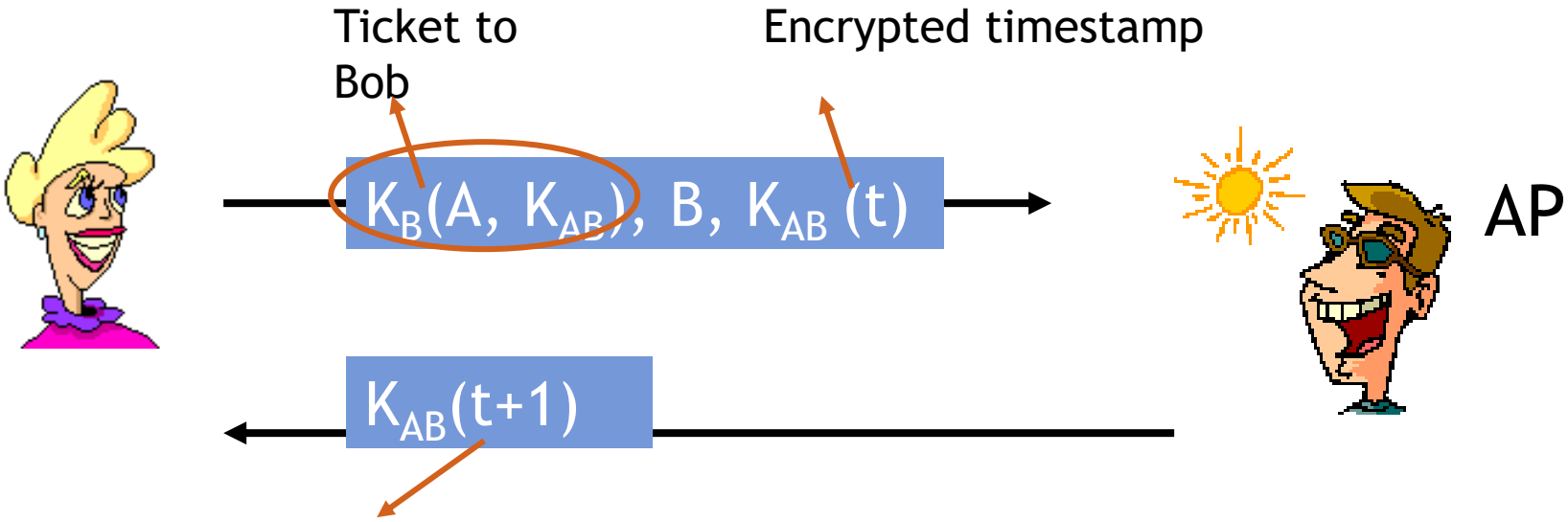
1) Alice is prompted for a password by the client, and this password is used for generating K_A, so that she obtains the session key and the ticket for TGS
2) The client ***forgets*** the password
3) Alice says she wants to use Bob's services

Operation of Kerberos TGS



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

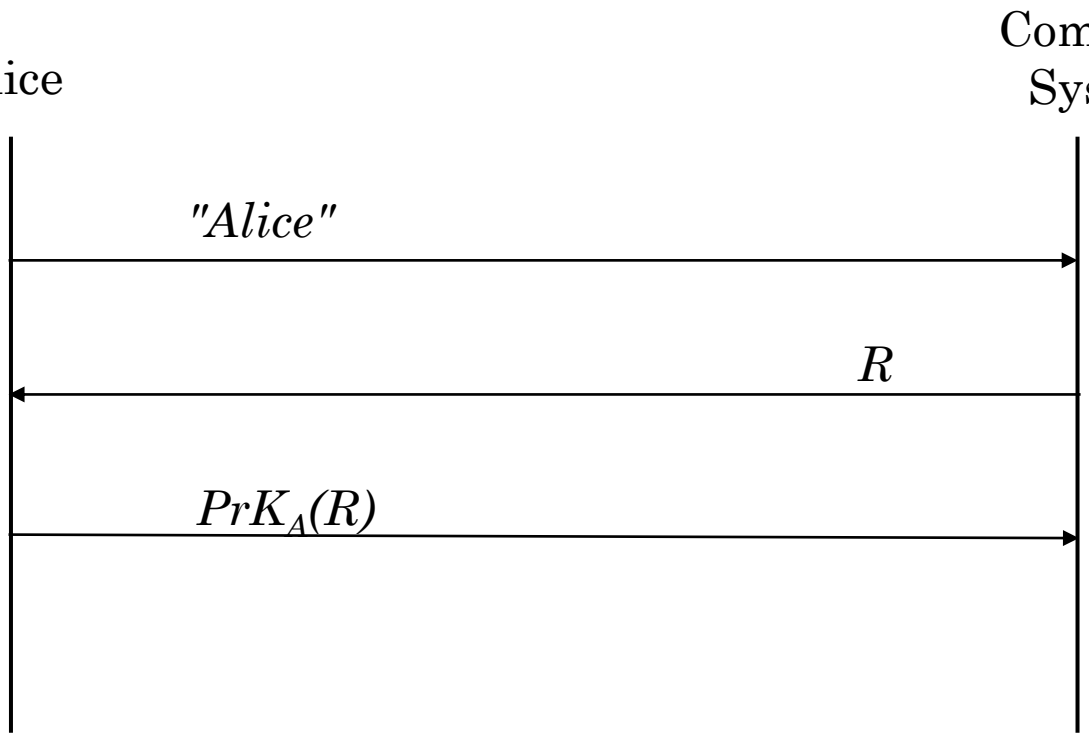
Operation of Kerberos Application Server



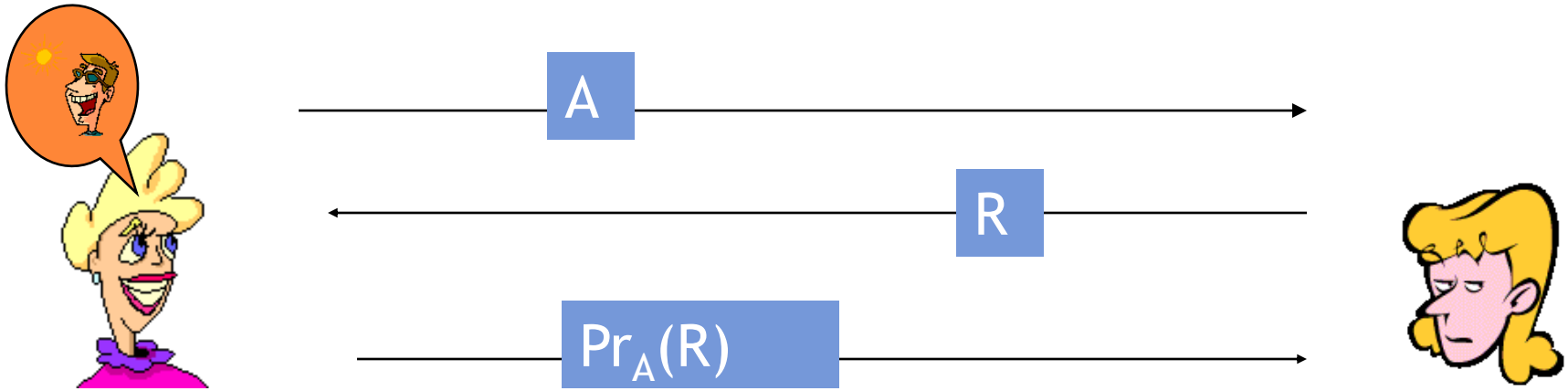
Now Alice can work with Bob, but if she needs to change to another application server, she just restarts with the request to TGS (no passwords are ever transmitted)

Public-Key-Based Authentication

Alice

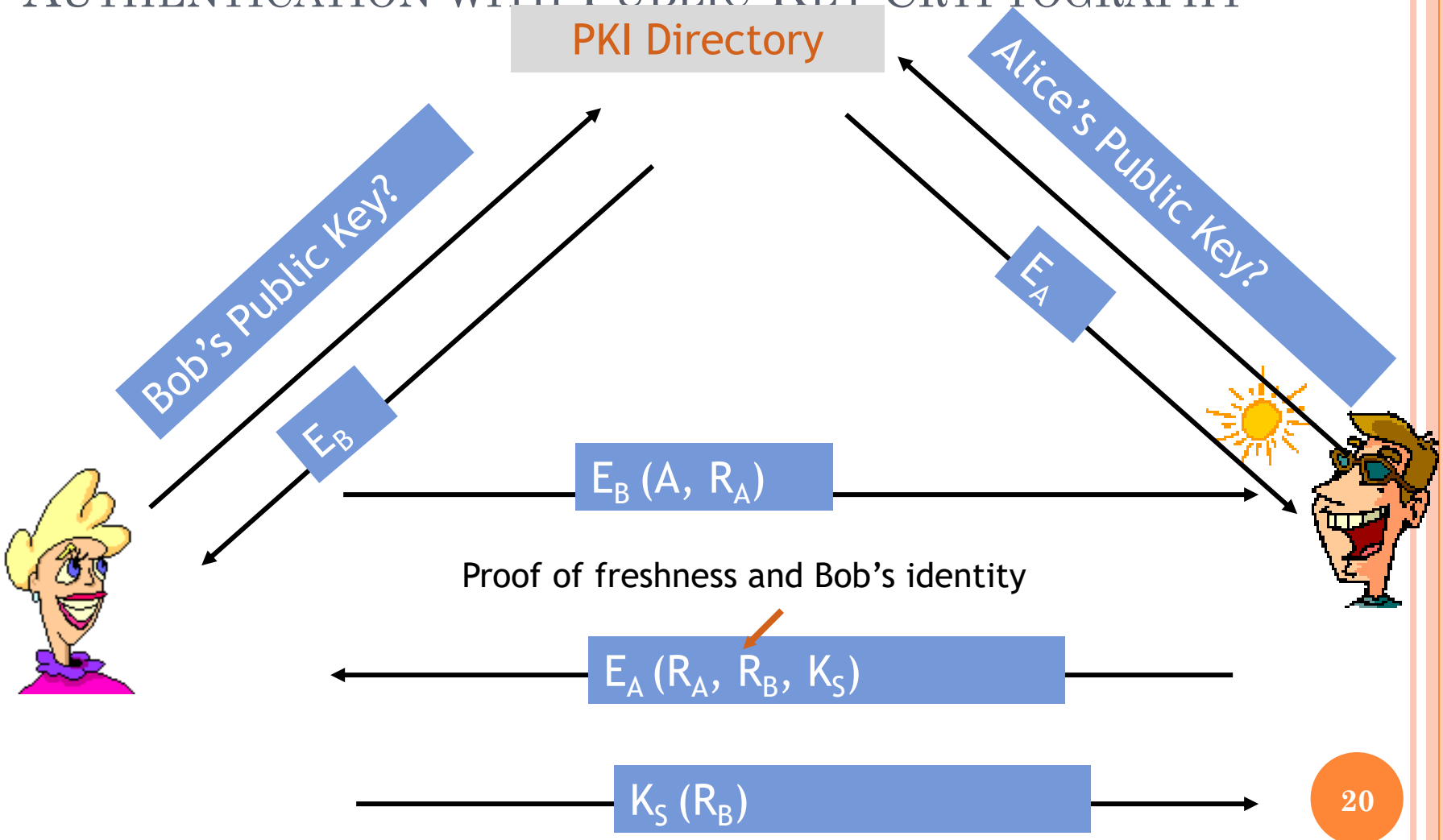


Authentication with Public-Key Cryptography: A Naïve "Solution"

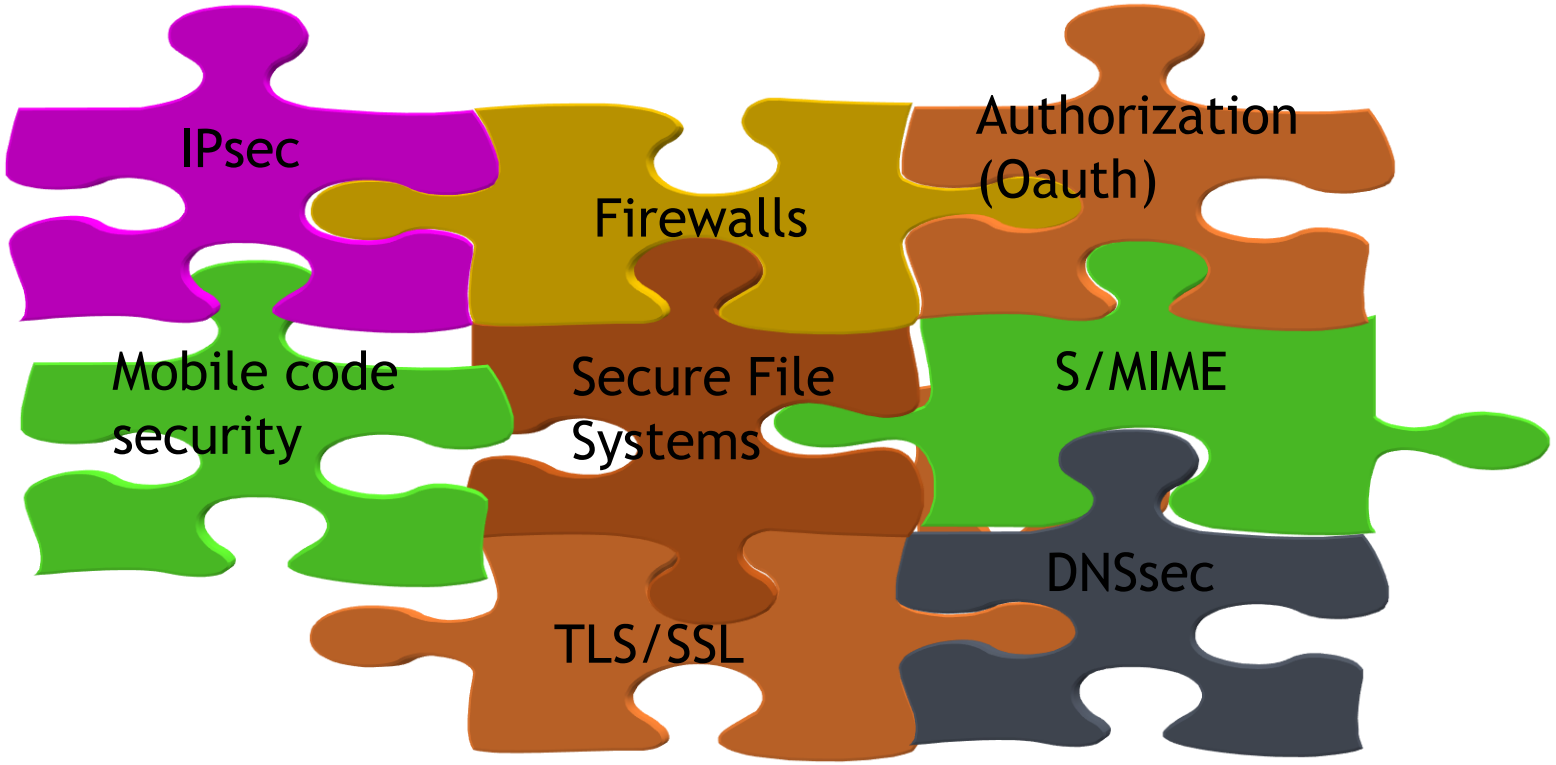


…R = "I, undersigned Alice, owe Trudy $1,000,000"

Authentication with Public-Key Cryptography

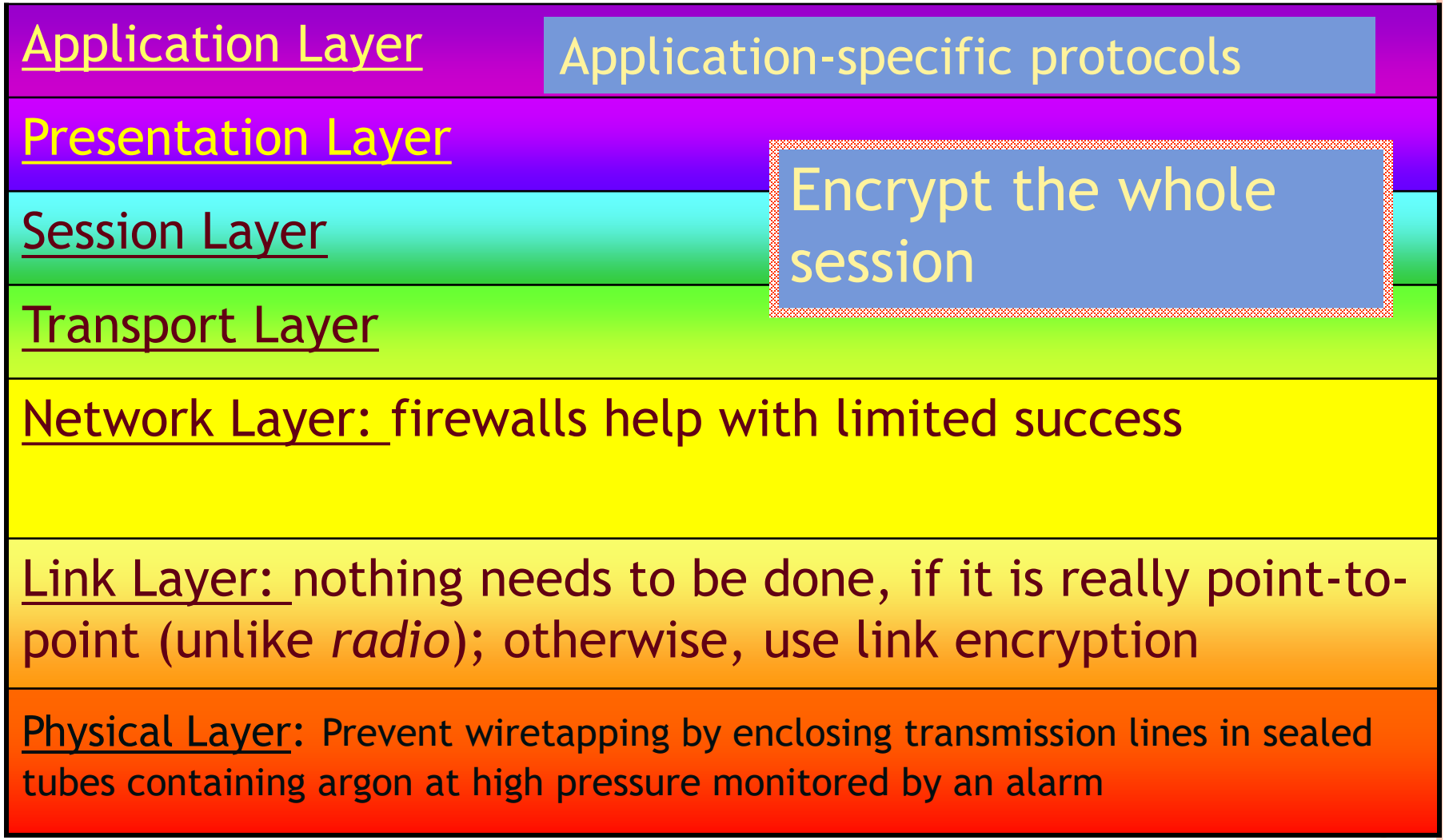


Common Security Solutions used in Cloud



We will focus on *IPsec* and *TLS*, which have become indispensible 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!

24

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

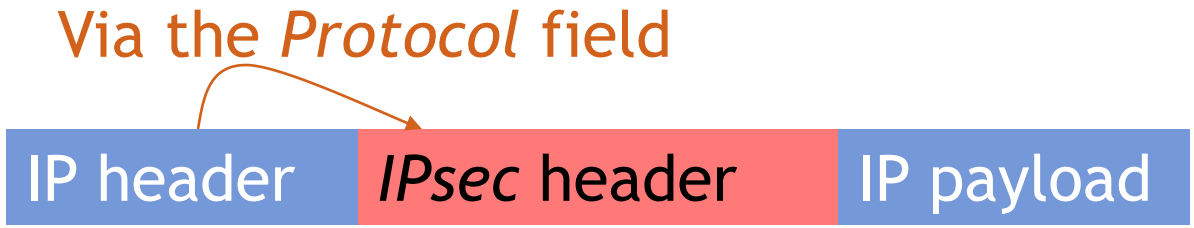
- Authenticating both ends
- Establishing the key
- Agreeing on cryptographic algorithms
- Initializing the sequence number (which will run through the life of the association)
- Establishing SPI

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



- Tunnel mode

Useful for

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

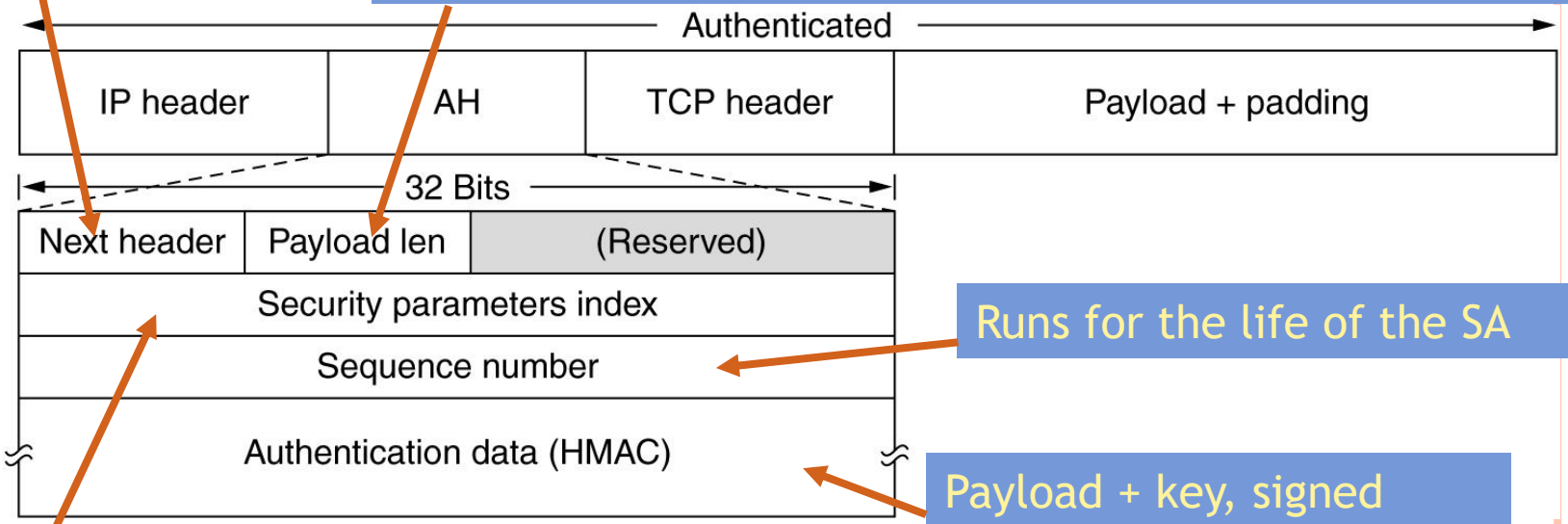
Two (Historical) Headers

- 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



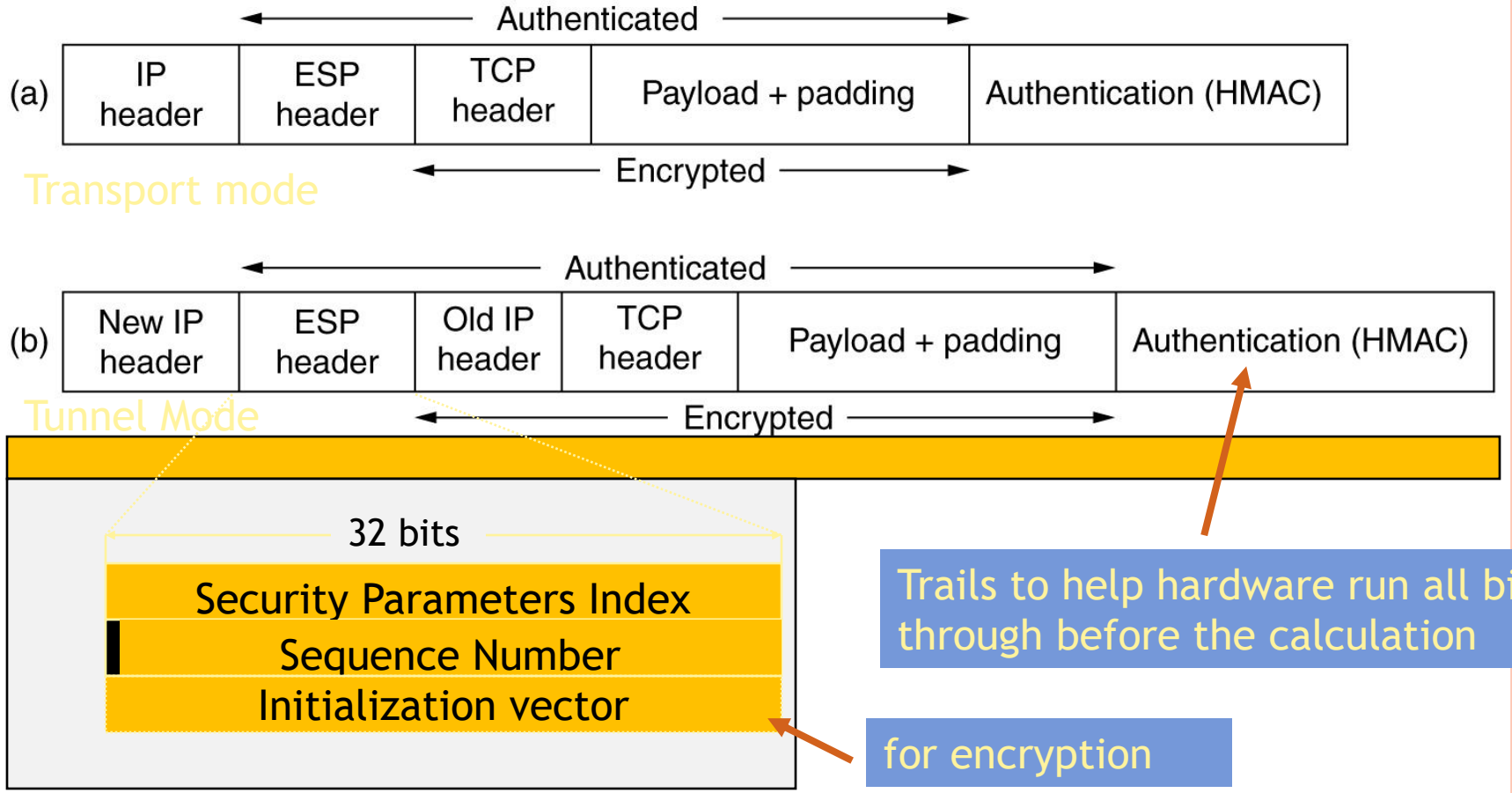
Runs for the life of the SA

Payload + key, signed

The "virtual circuit number" associated with the shared key

(After A. Tanenbaum)

Encapsulating Security Payload (ESP) Header

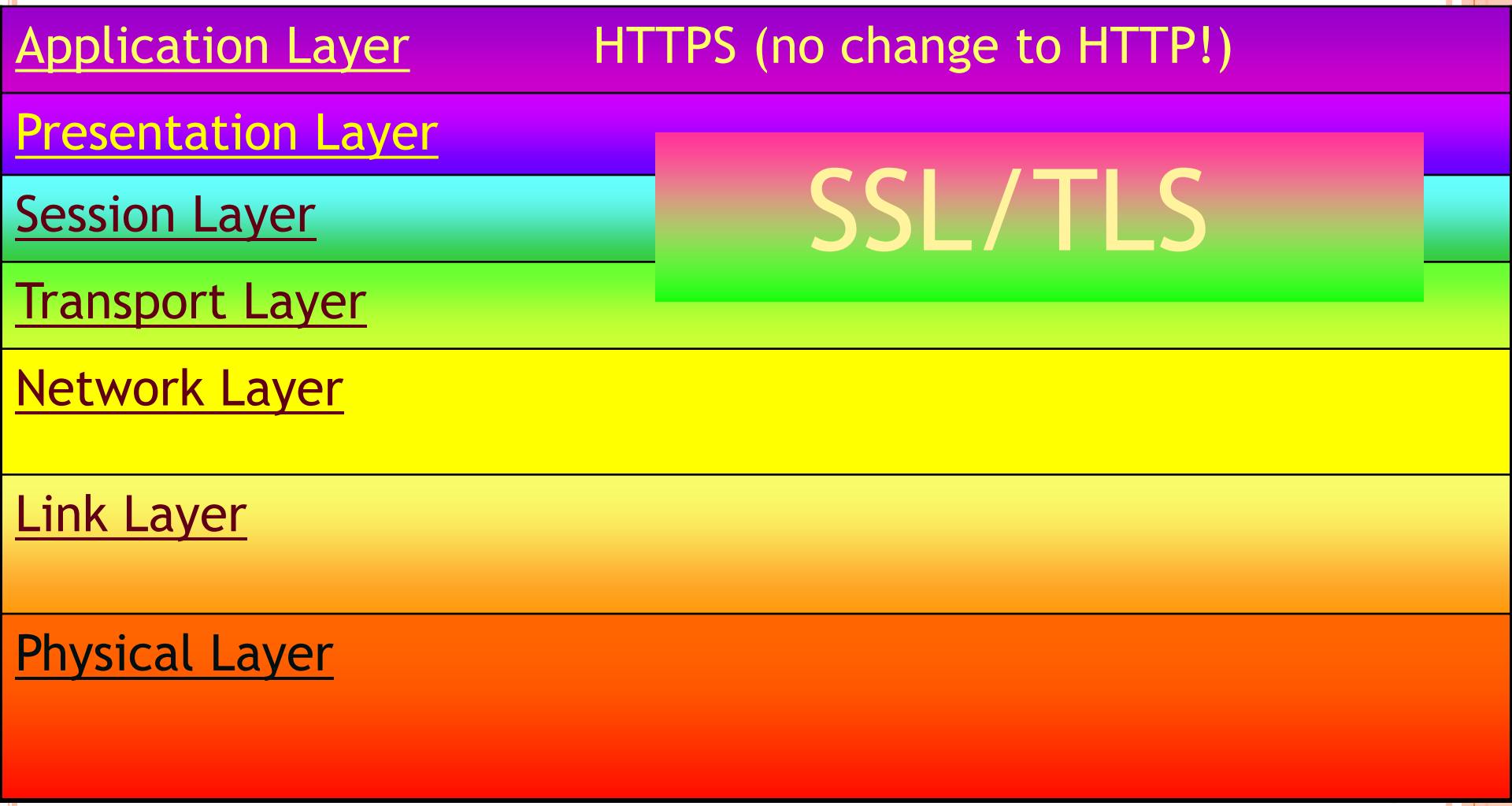


(After A. Tanenbaum)

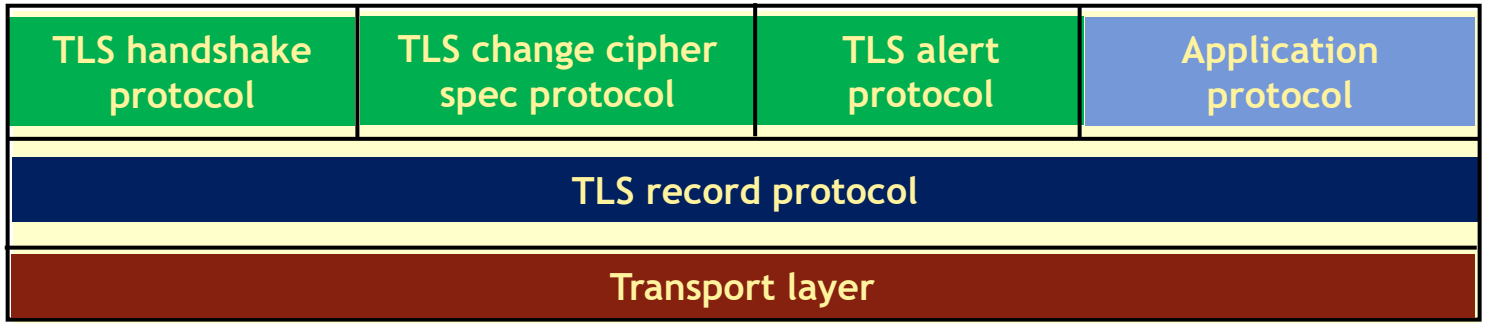
Transport Layer Security (TLS) Background

- 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)

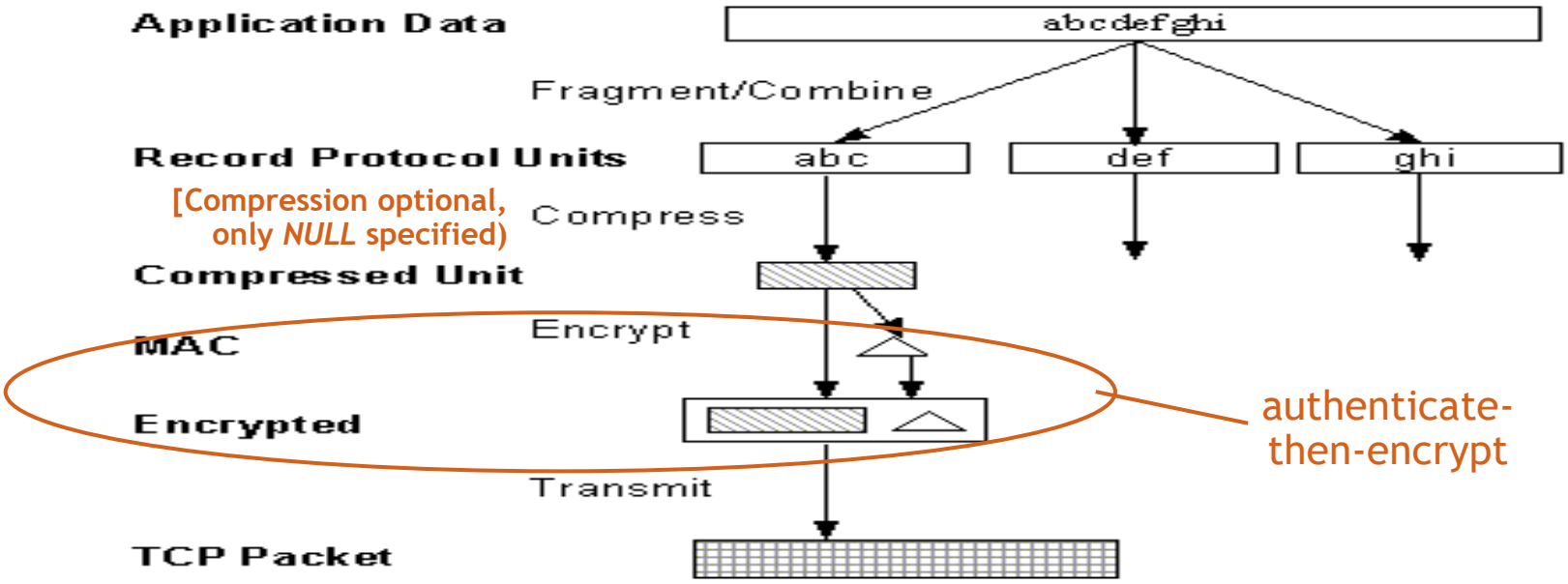
TLS *change cipher spec* protocol: Signaling to use the newly negotiated cipher suite and session keys for protecting ensuing records

TLS *alert* protocol: Signaling errors during a session

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

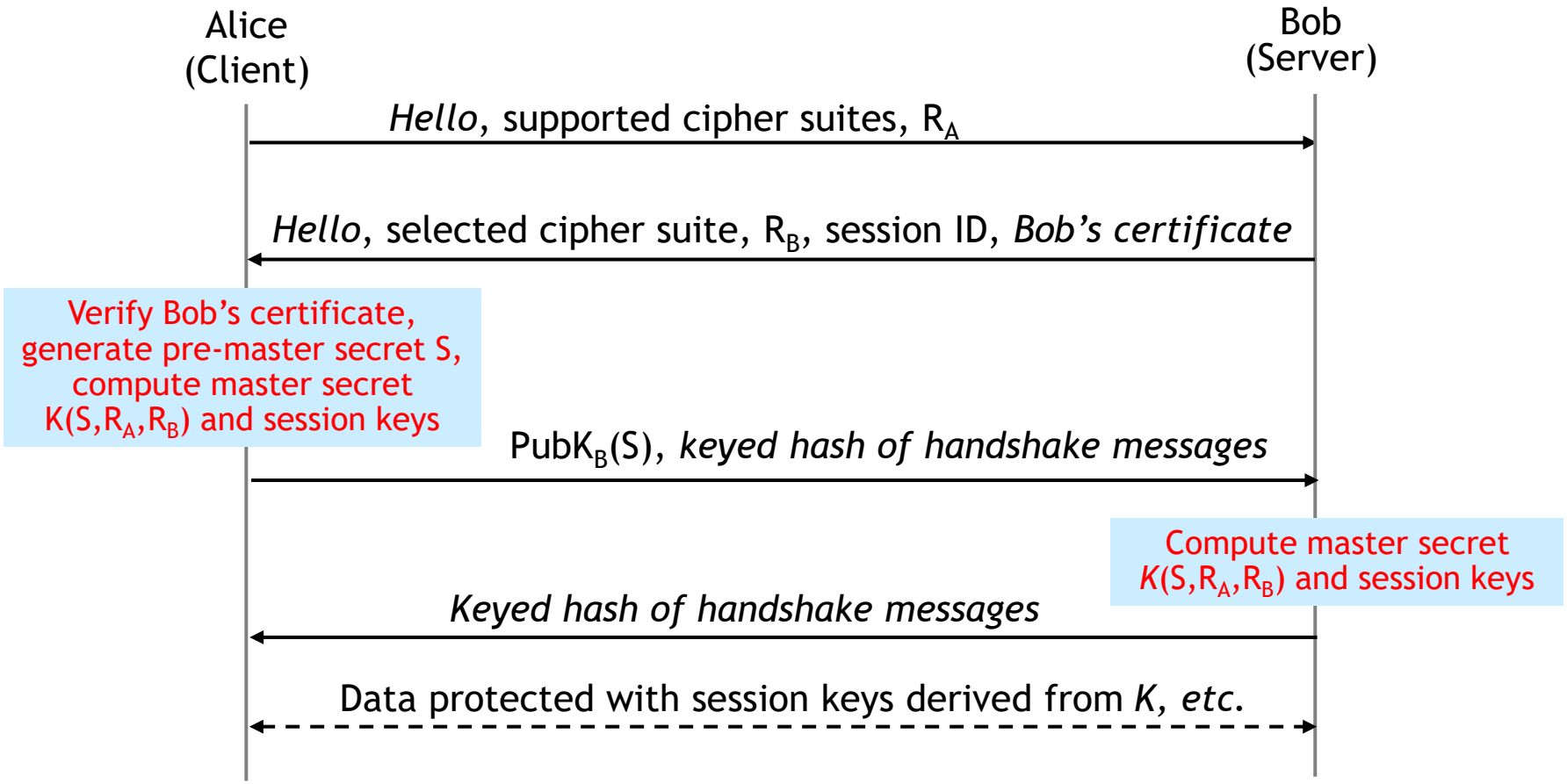
Transport layer: Reliable transport (e.g., TCP or SCTP)

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

	Object 1	*Object 2*	*Object 3*	*Object 4*
Subject 1	read, write, execute	read, write, execute	read	read
Subject 2	read, execute	read, execute	null	null
Subject 3	read, execute	read	read, write	write

Access Control Matrix

Capability Lists

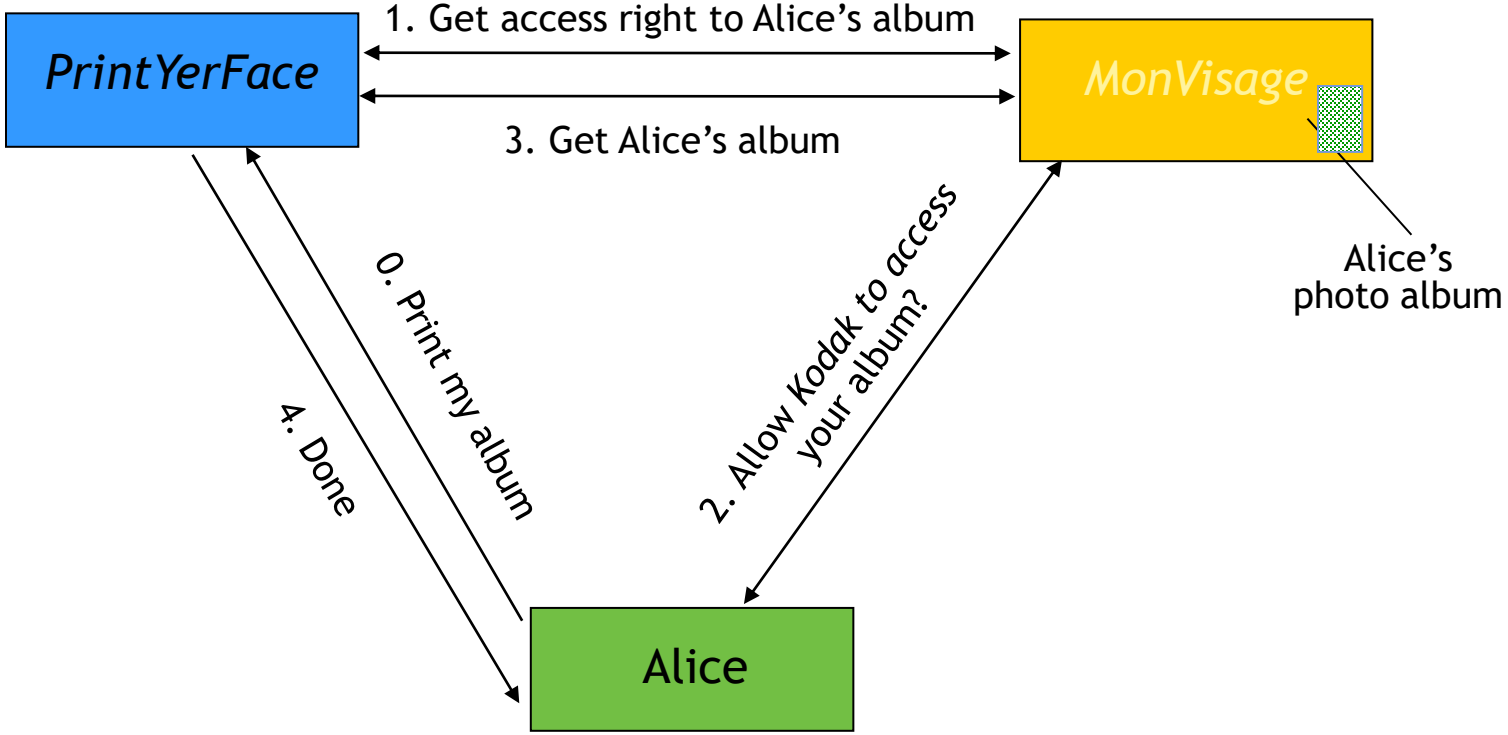
Subject 1 →	Object 1: read, write, execute; Object 2: read, write, execute; Object 3: read; Object 4: read
Subject 2 →	Object 1: read, execute; Object 2: read, execute
Subject 3 →	Object 1: read, execute; Object 2: read; Object 3: read, write; Object 4: write

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

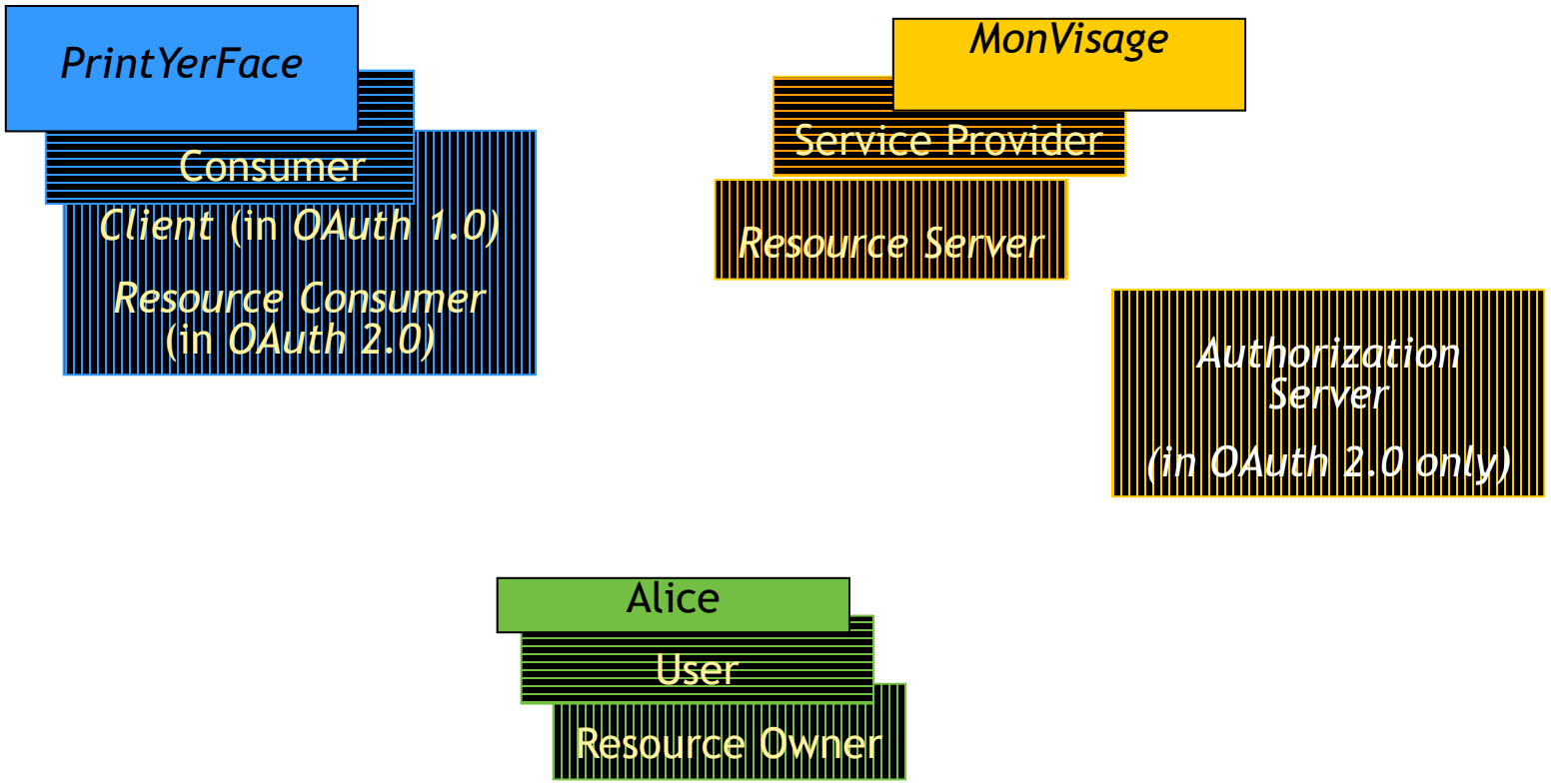
40

Open Authorization (*OAuth*) Protocol

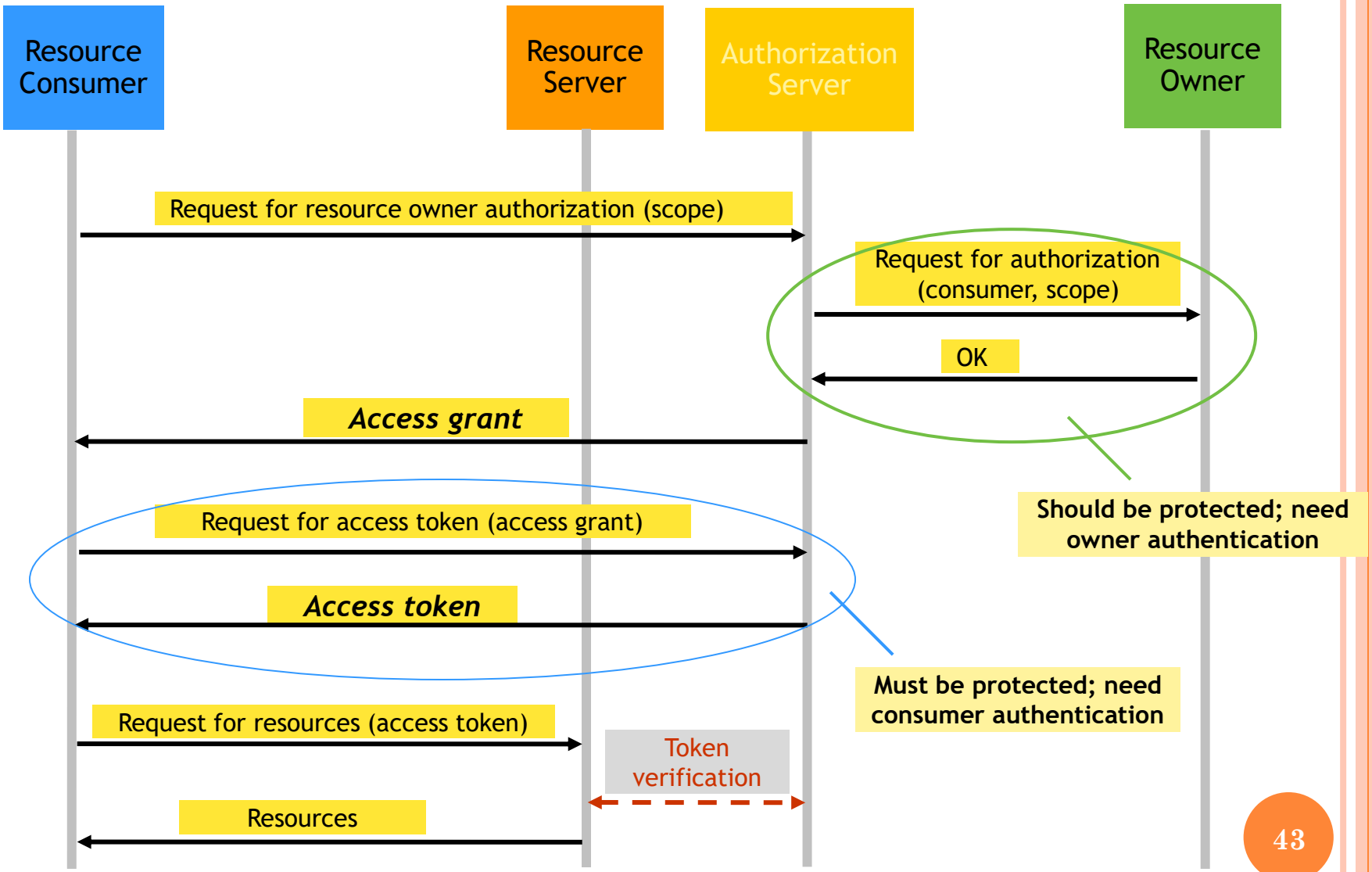


Alice grants *PrintYerFace* access to her album at *MonVisage* without revealing her credentials

OAUTH TERMINOLOGY



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)

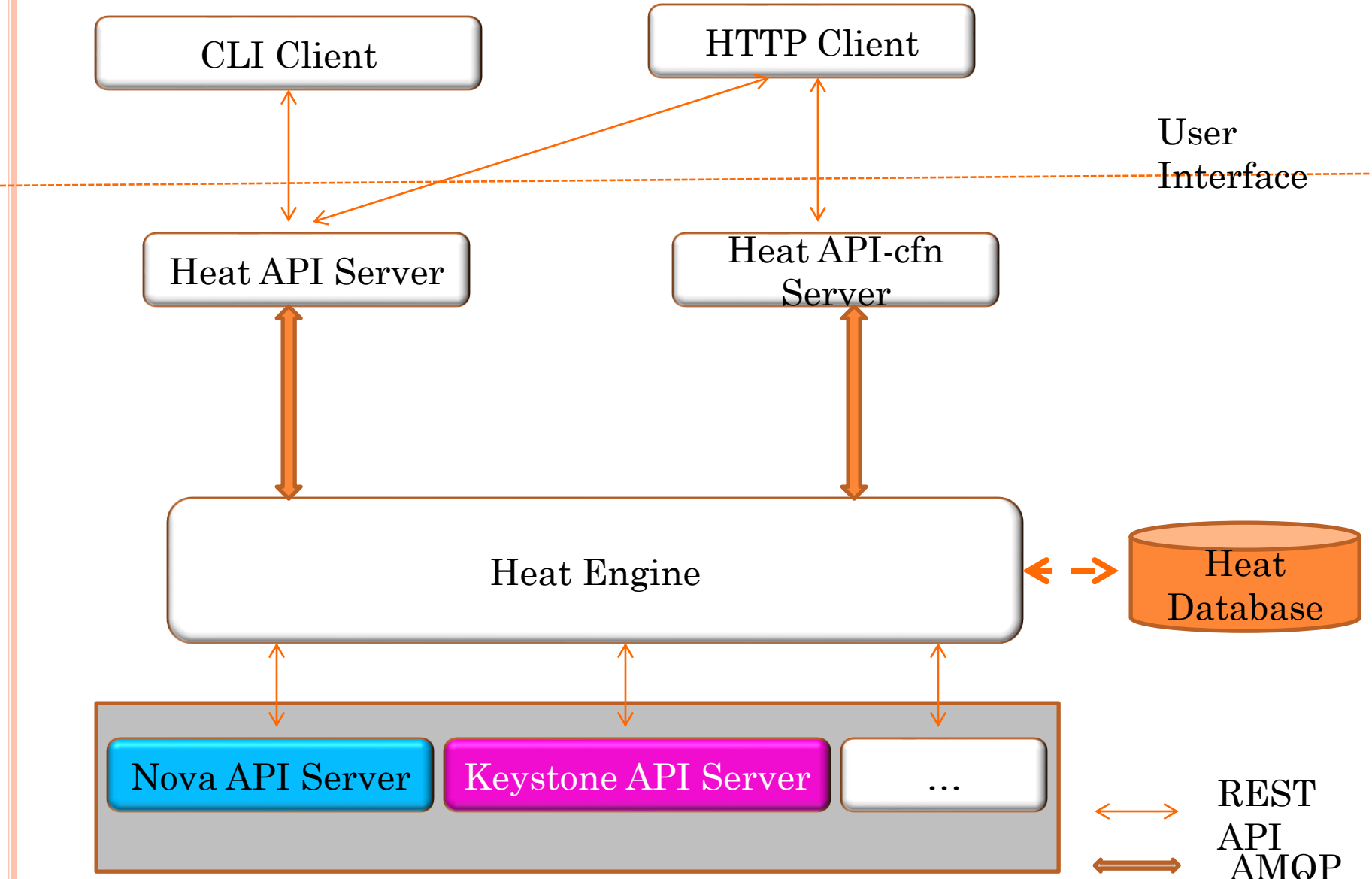
Cloud Data Storage Considerations

- 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!

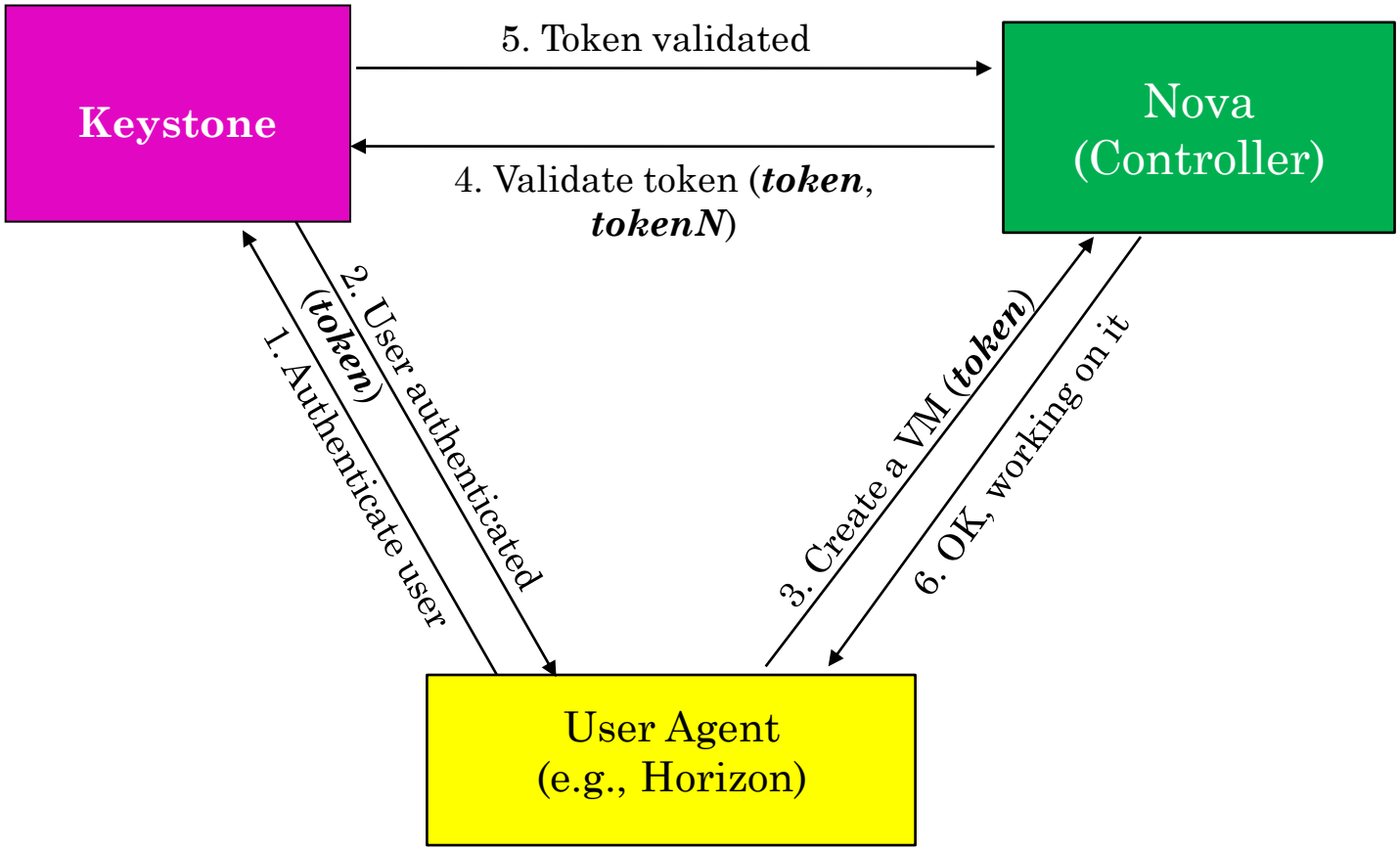
Identity Management in OpenStack

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

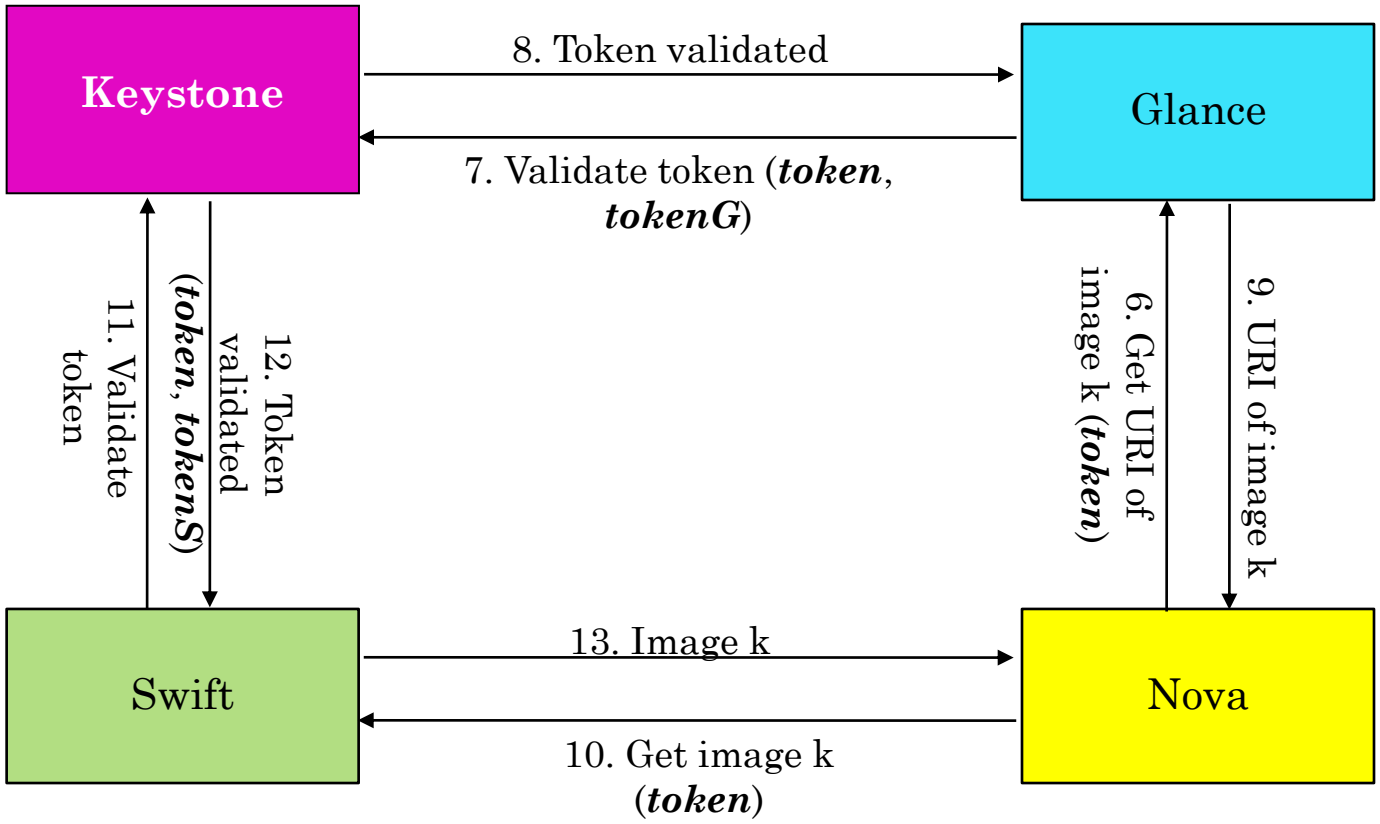


The *Heat* computing architecture

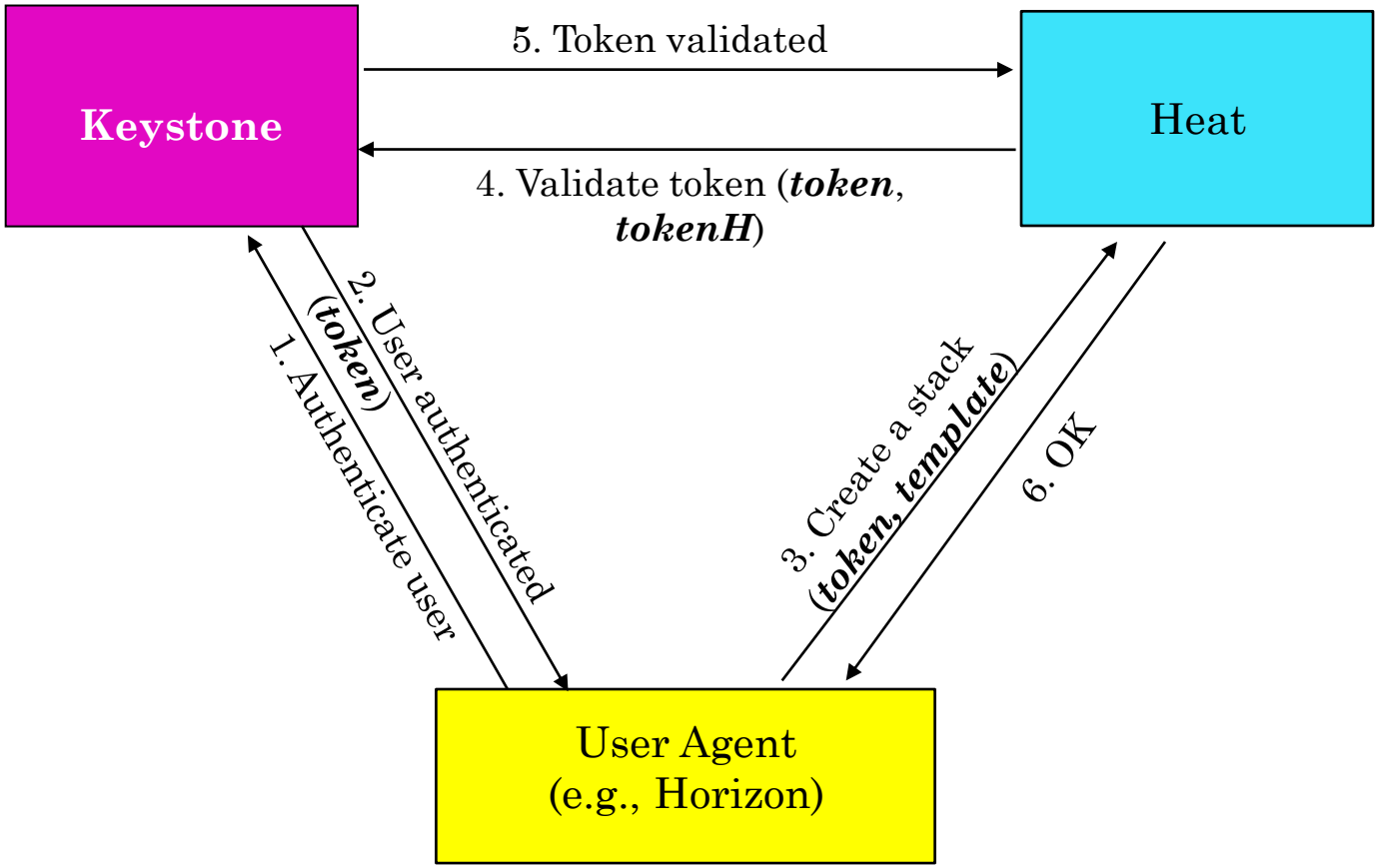
A Simplified Workflow for VM Provisioning in OpenStack



Additional Steps for VM Provisioning



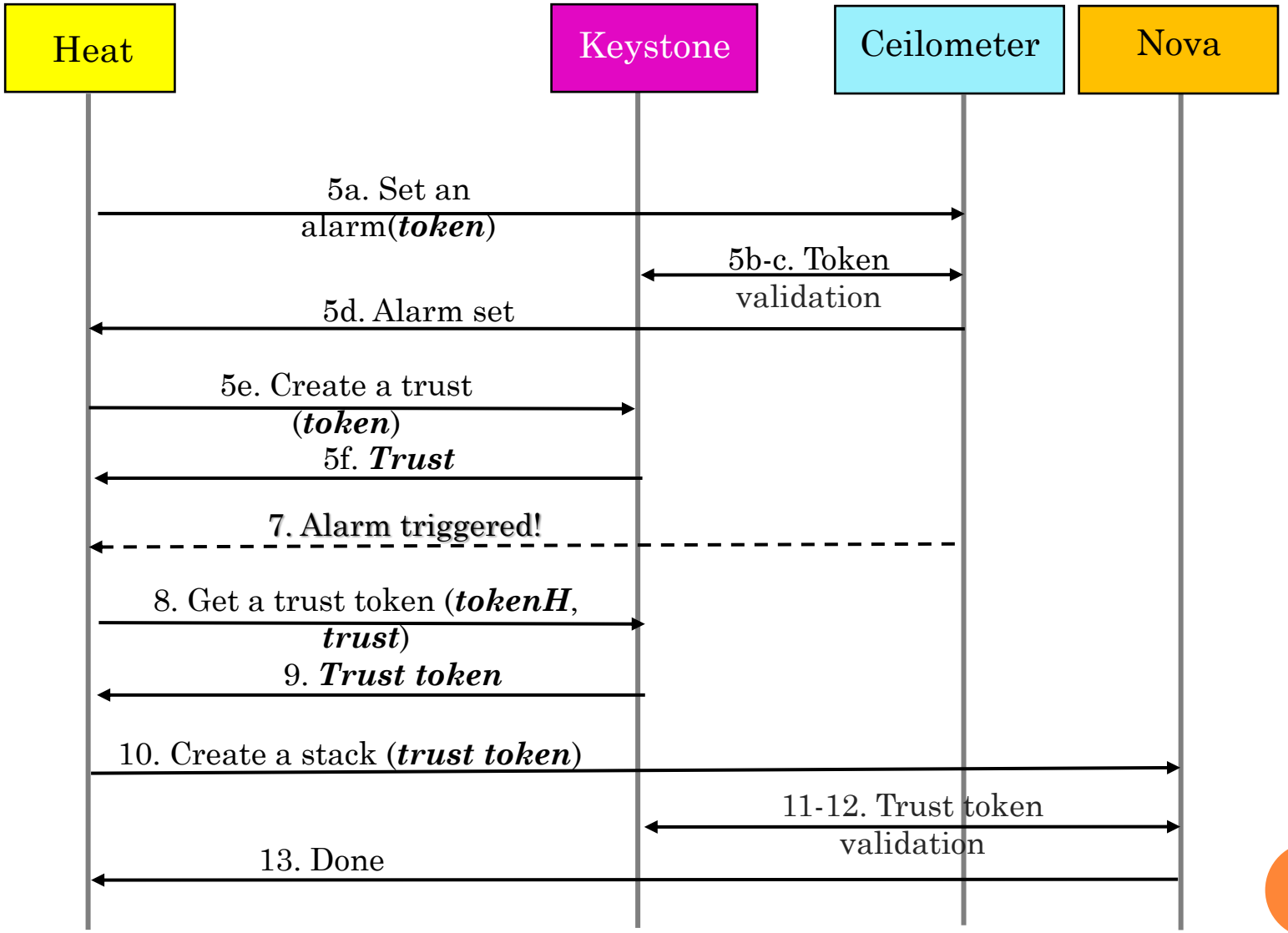
A Simplified Workflow for Auto-Scaling



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"
        }
    }
}
```

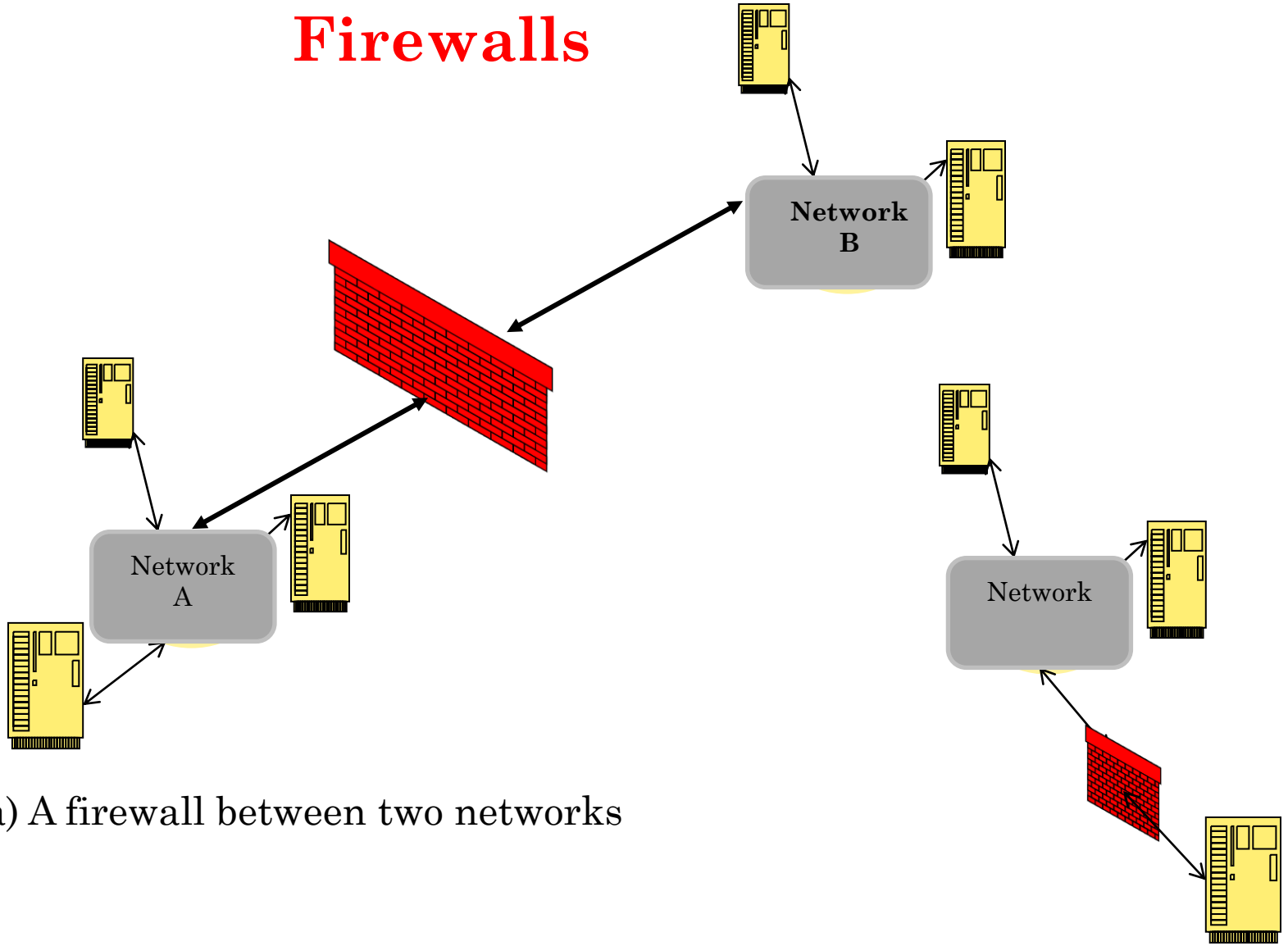
Additional Steps for Auto-Scaling



An Example Trust

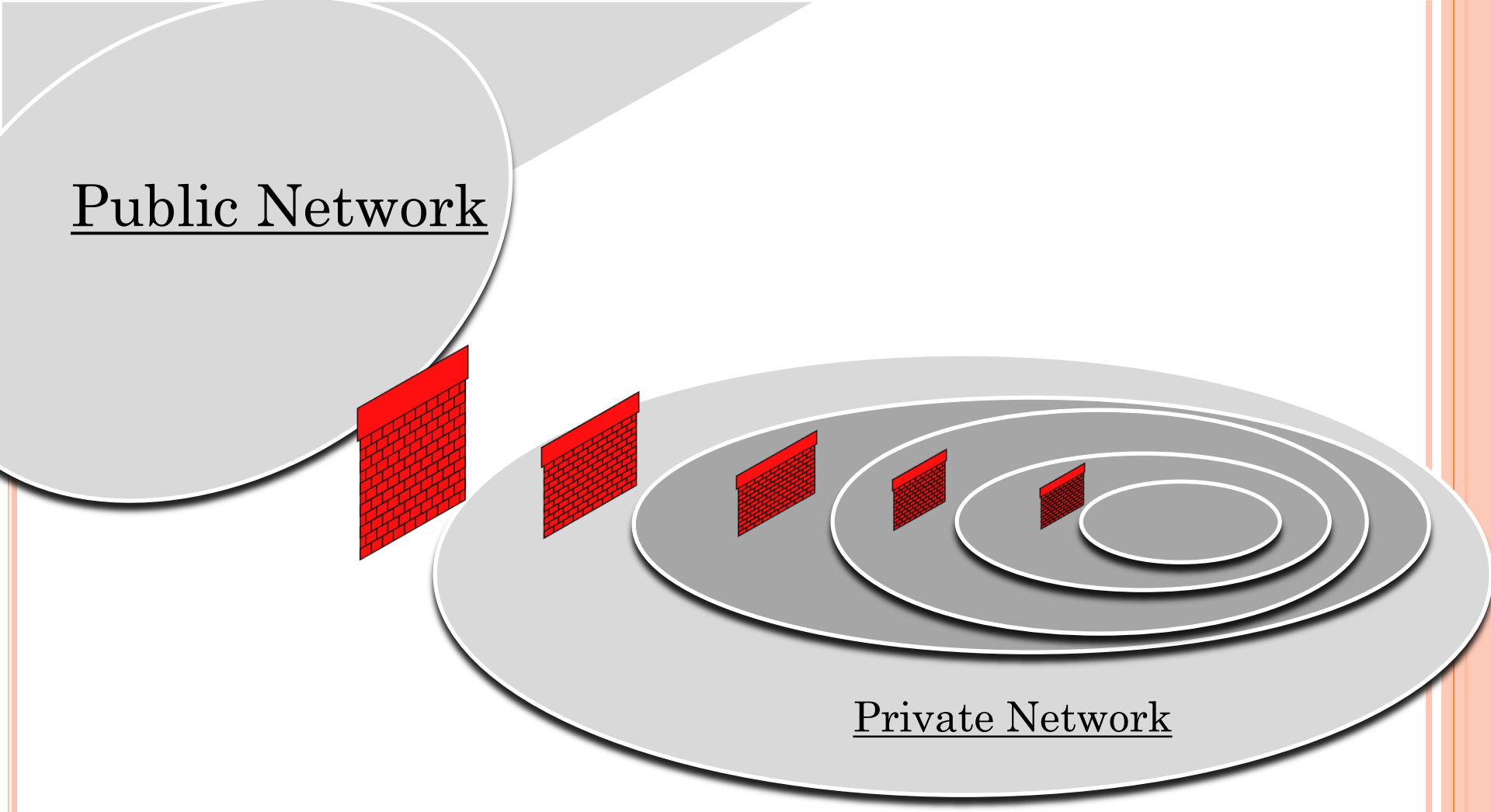

```
{
   "trust": {
          "expires_at": "2016-05-27T21:52:58.852167Z",
          "id": "c703057be878458588961ce9a0ce686b",
          "impersonation": true,
          "project_id": "fb49a0ecd60c4d2092643b4cfe272106",
          "remaining_uses": null,
          "roles": [{
          "id": "9fe2f79ee4384b1894a9083bd3e92bab",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

Firewalls

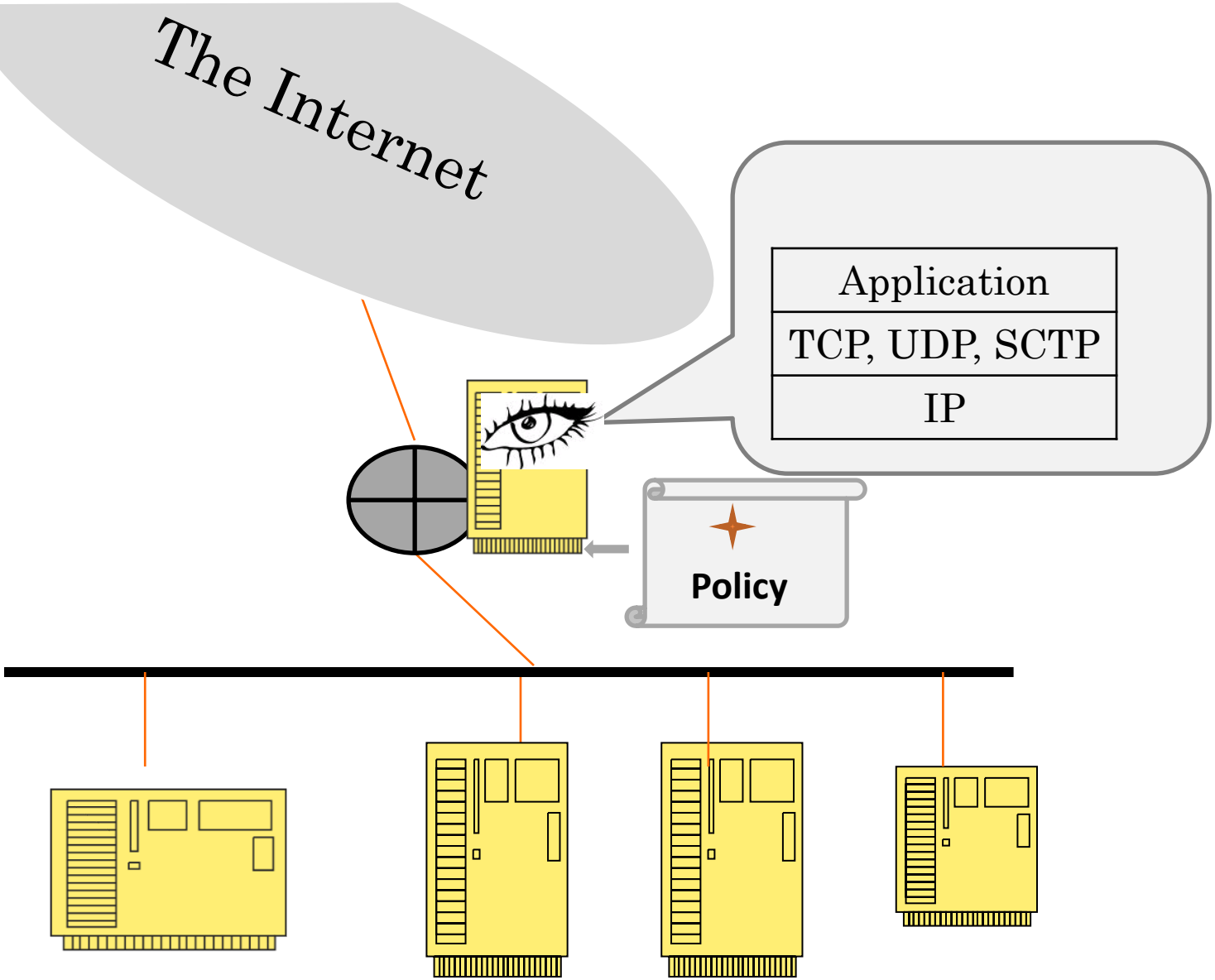


a) A firewall between two networks

b) A firewall protecting a single host

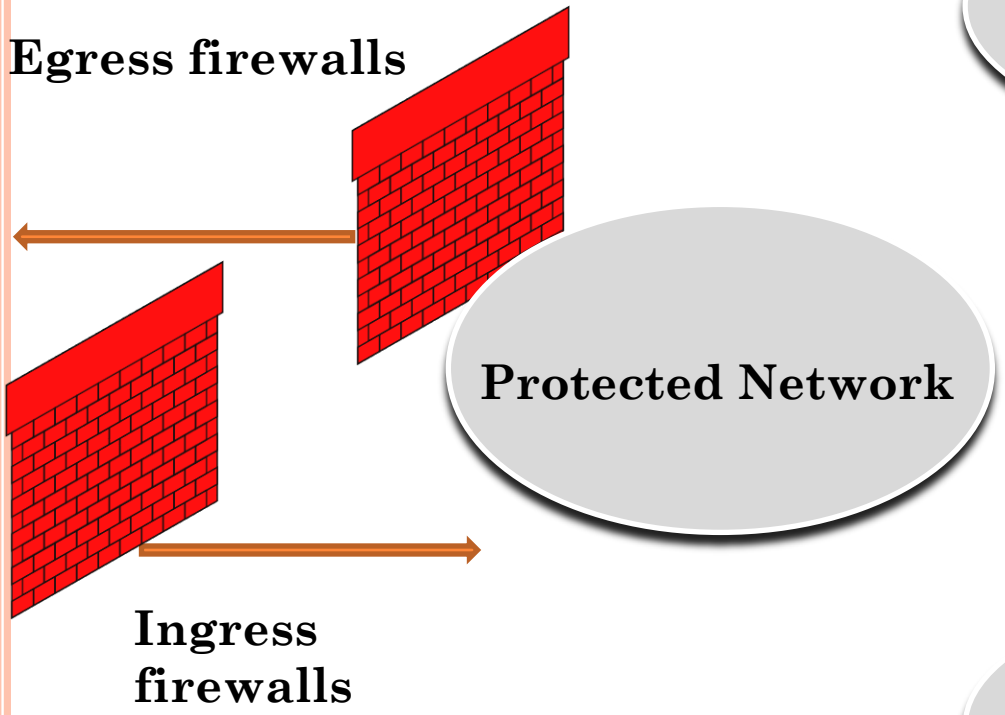


Interconnecting networks with different security postures

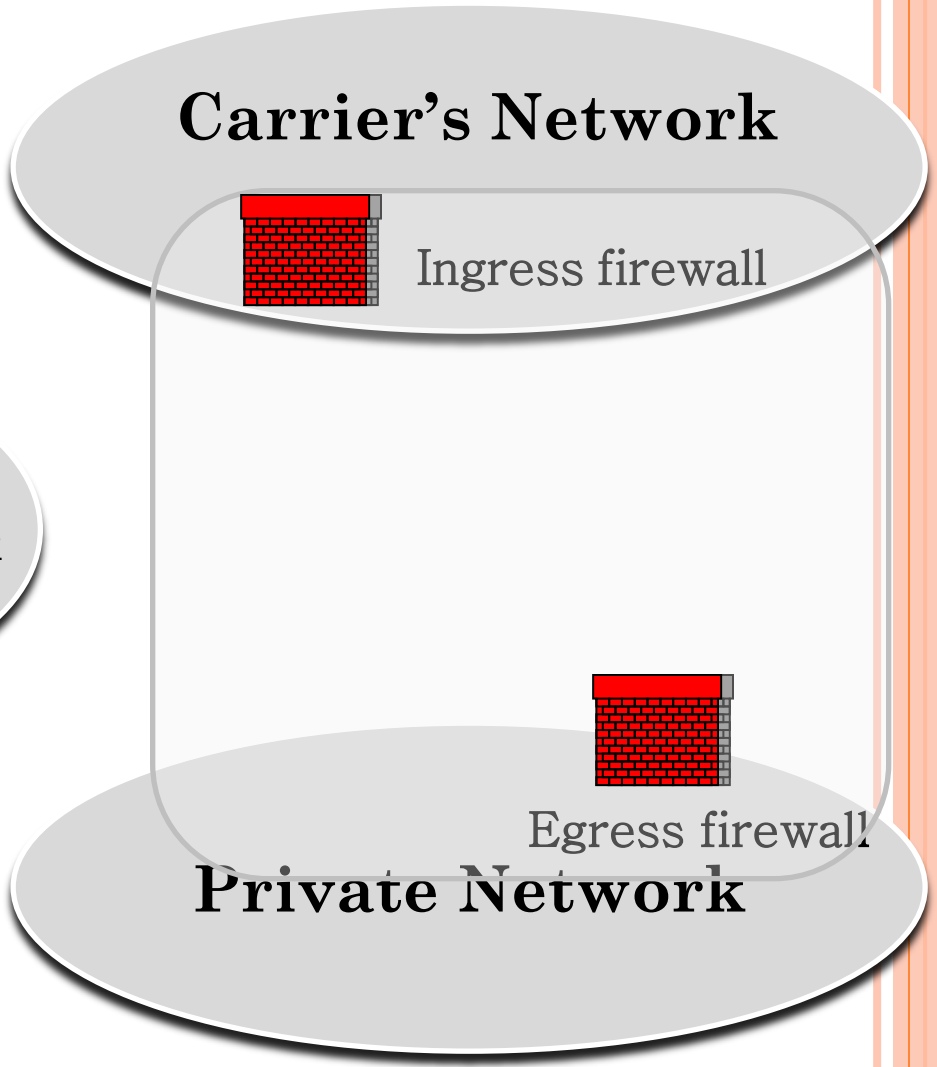


The application gateway

Ingress and egress firewalls

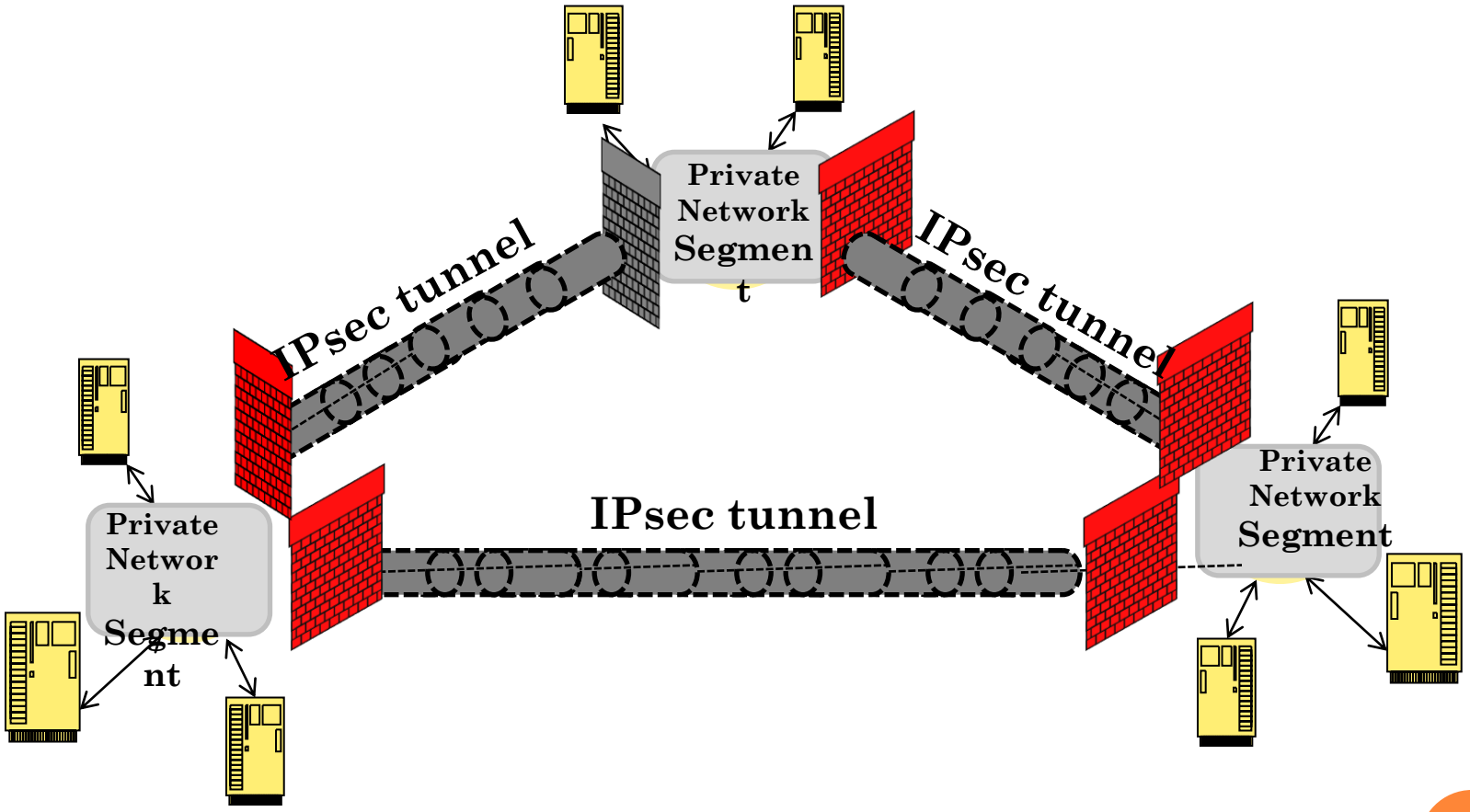


a) Interfaces



b) Split CPE

Layer 3 VPN with firewalls



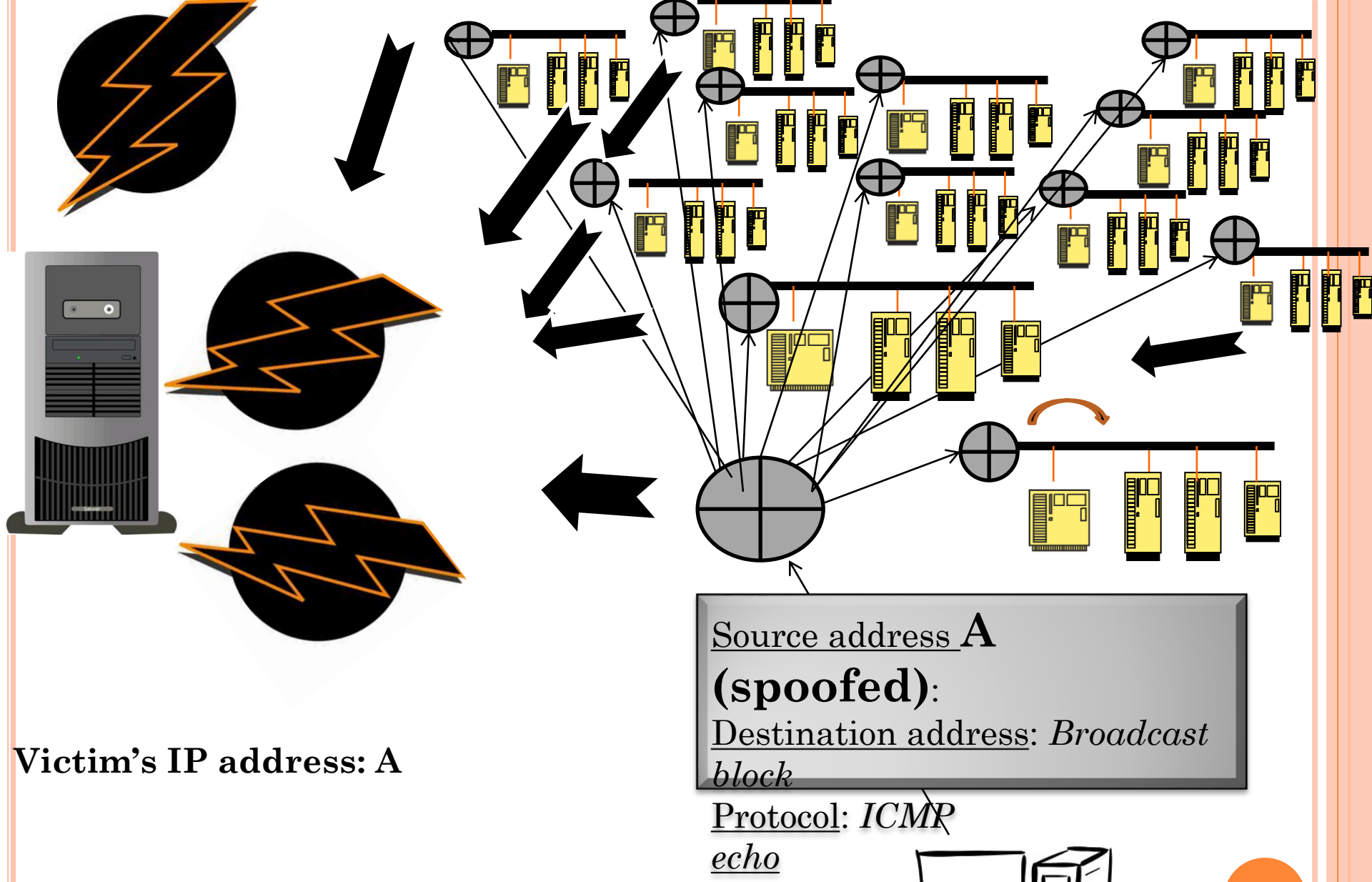
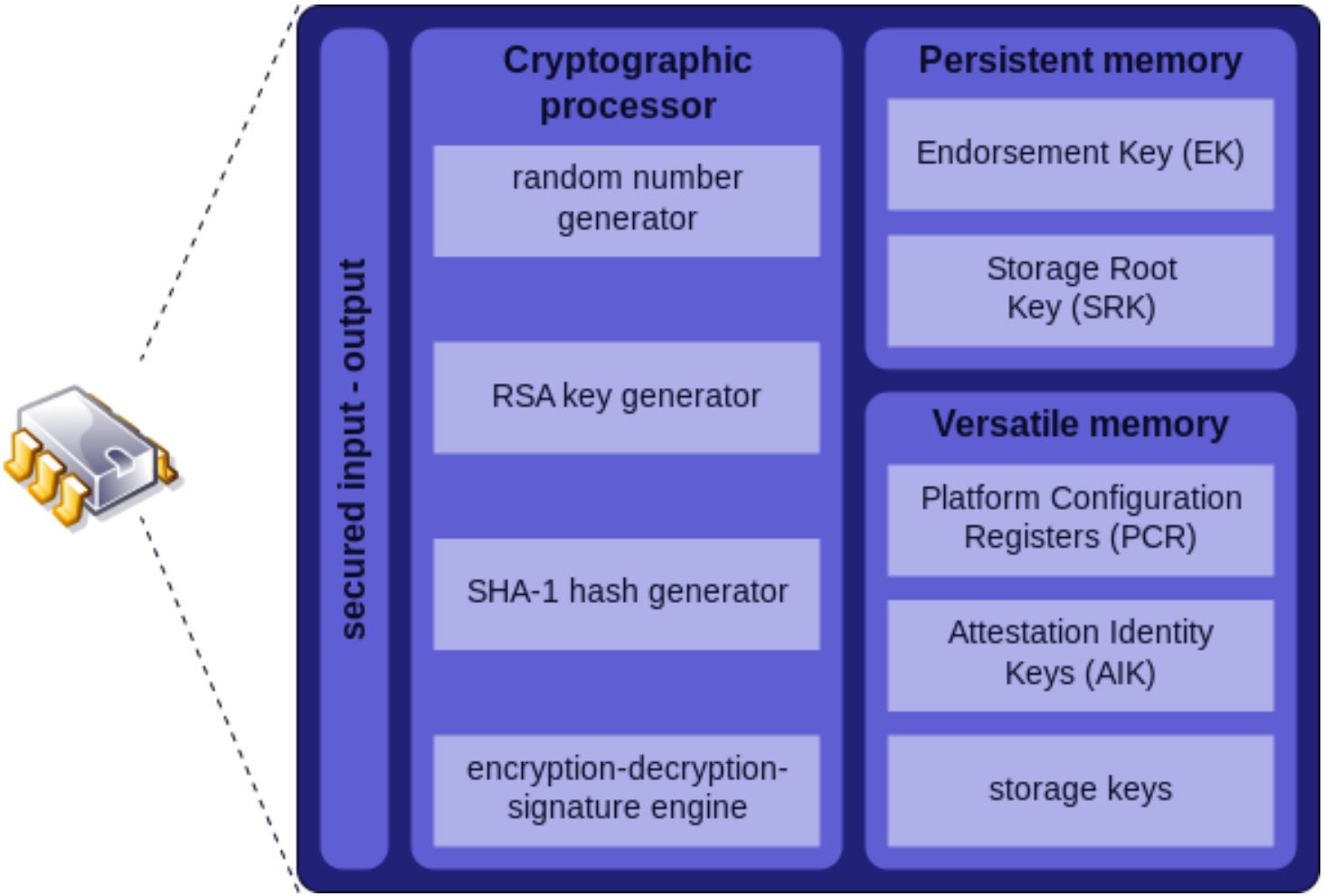


Figure 5.16: A *Smurf* attack

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 allows 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



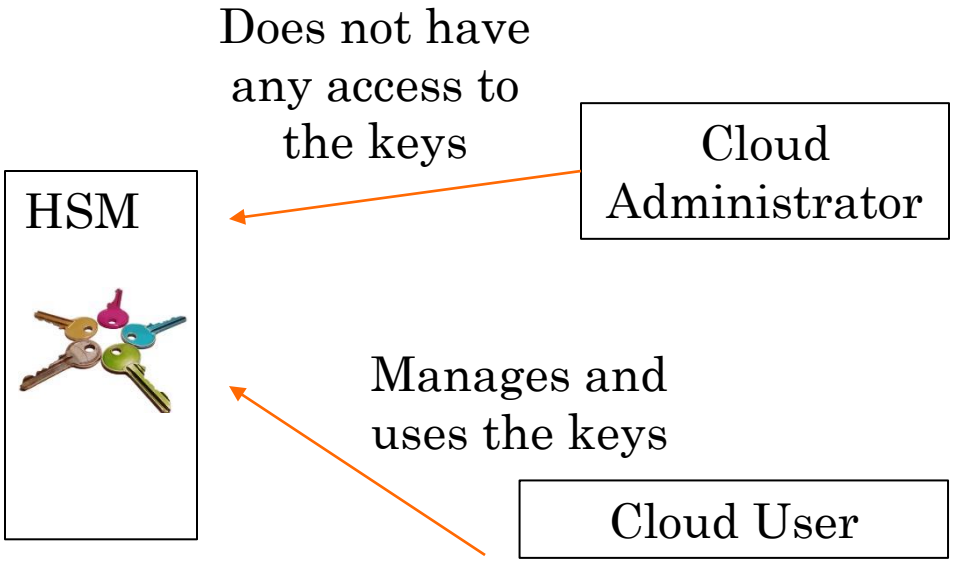
Source: Eusebius (***Guillaume Piolle*** , https://commons.wikimedia.org/w/index.php?curid=4809738

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 trsut* 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!