Advanced Network Security

Amir Mahdi Sadeghzadeh, Ph.D.

Outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec

Other than Confidentiality

- Authentication
  - Definition: The process of verifying the identity of a user, device, or system.
    - Ensures that the entity accessing the system is who they claim to be.
  - Methods: Passwords, biometrics, digital signatures, MACs, etc.
- Data Integrity
  - Definition: The assurance that data remains unaltered during transmission or storage.
  - Methods: MACs, digital signatures.
- Non-repudiation
  - Definition: The inability of a user to deny the authenticity or origin of a communication or action.
  - Methods: Digital signatures.

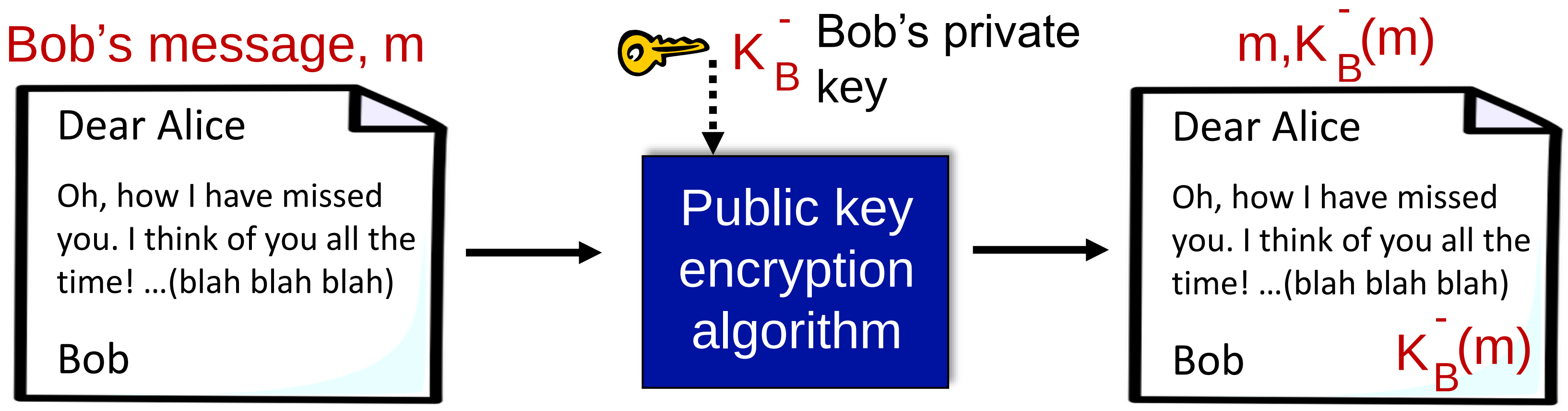
Where Does This Fit?

	Secret Key Setting	Public Key Setting
Secrecy / Confidentiality	Stream cipher Block cipher + encryption modes	Public key encryption: RSA, El Gamal, etc.
Authenticity / Integrity	MAC	Digital Signatures

Digital signatures

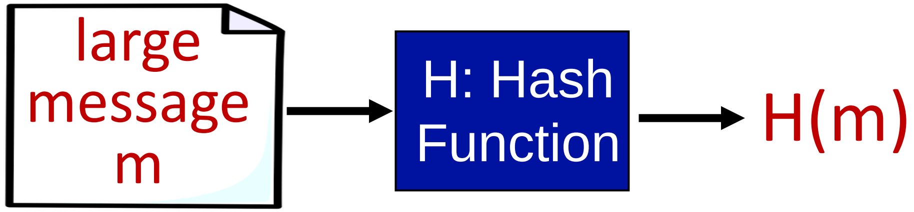
Cryptographic technique analogous to hand-written signatures:

- sender (Bob) digitally signs document: he is document owner/creator.
- *verifiable, nonforgeable:* recipient (Alice) can prove to someone that Bob, and no one else (including Alice), must have signed document
- simple digital signature for message m:
  - Bob signs m by encrypting with his private key K_B, creating "signed" message, $K_B^-(m)$



Message digests

- Computationally expensive to public-key-encrypt long messages
  - fixed-length, easy- to-compute digital “fingerprint”
  - apply hash function H to *m*, get fixed size message digest, *H(m)*
- Existential forgery
  - extra layer of security

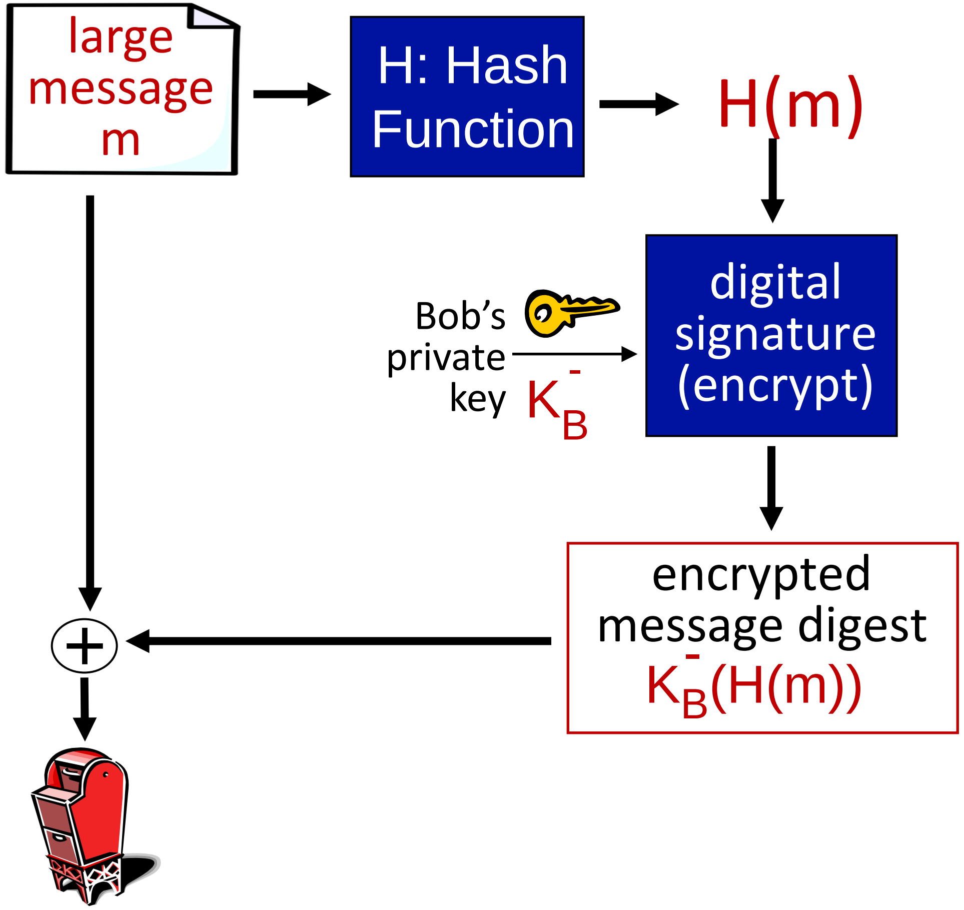


Hash function properties:

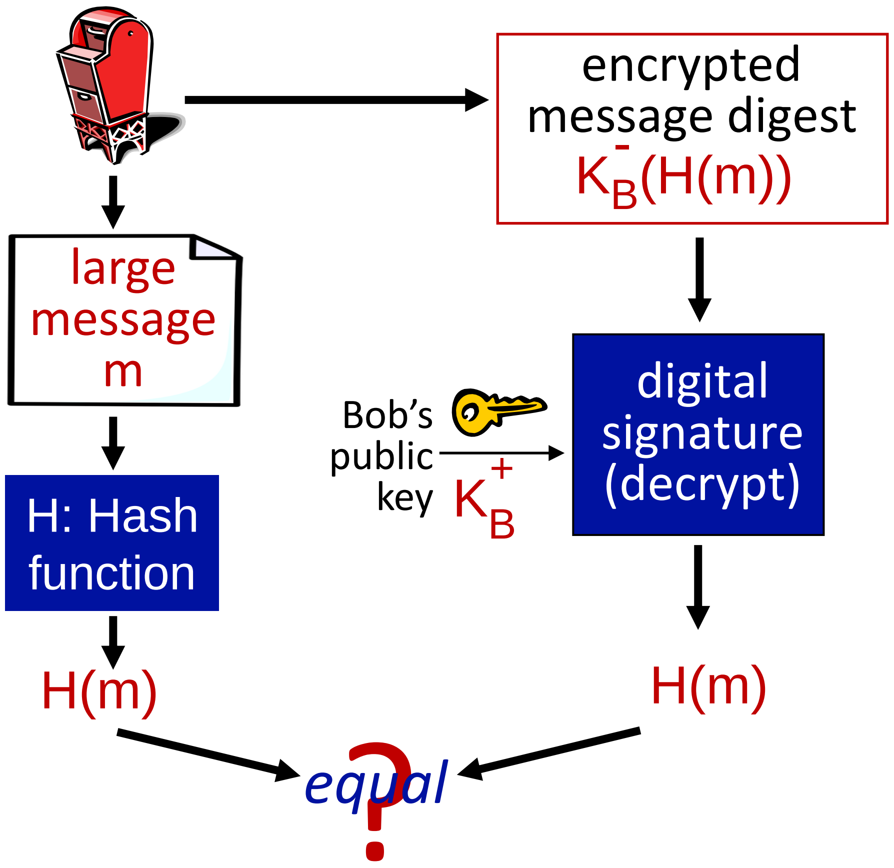
- produces fixed-size msg digest (fingerprint)
- given message digest *x*, computationally infeasible to find *m* such that $x = H(m)$

Digital signature = signed message digest

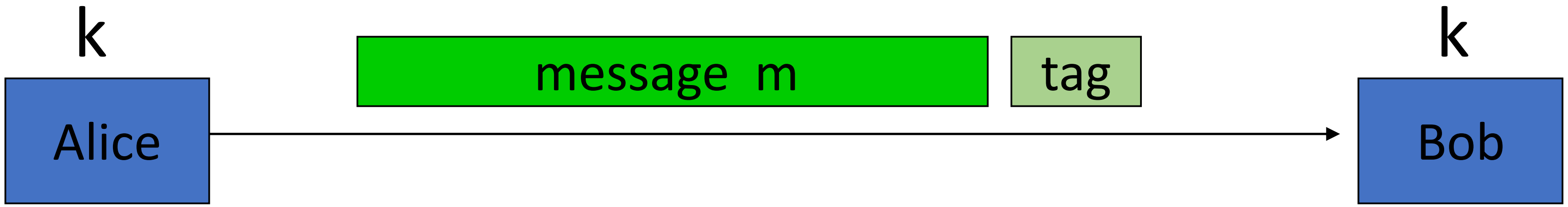
Bob sends digitally signed message:



Alice verifies signature, integrity of digitally signed message:



Message Authentication Codes (MACs)



Generate tag:

tag $\leftarrow$ S(k, m)

Verify tag:

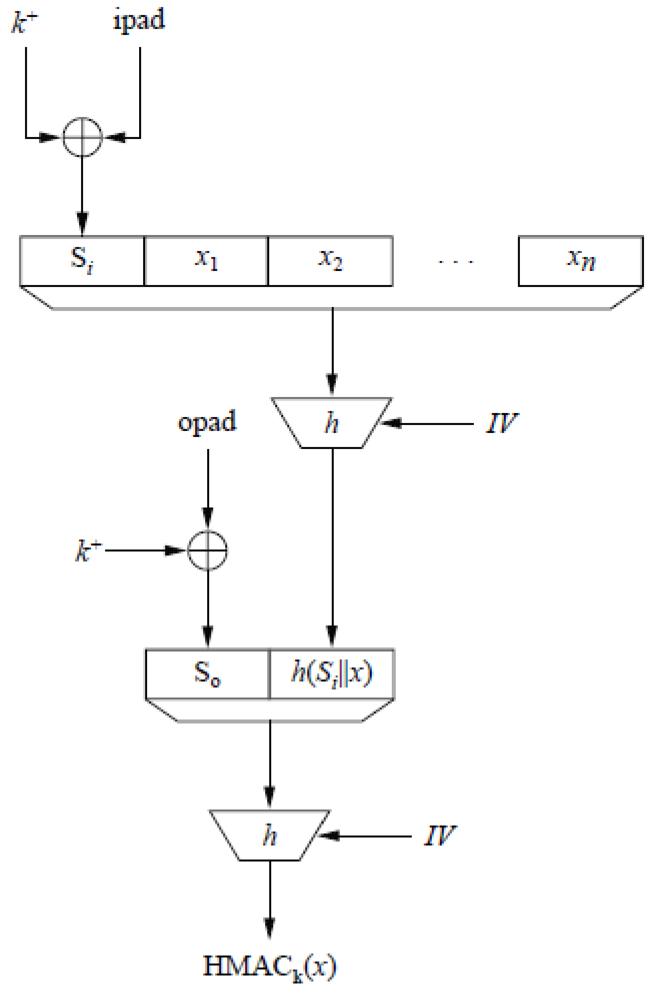
V(k, m, tag) $\stackrel{?}{=}$ \`yes'

Def: **MAC** I = (S,V) defined over (K,M,T) is a pair of algs:

- S(k,m) outputs t in T
- V(k,m,t) outputs \`yes' or \`no'

HMAC

- A hash-based message authentication code which does not show the security weakness of prefix and suffix MACs construction
  - The scheme consists of an inner and outer hash.



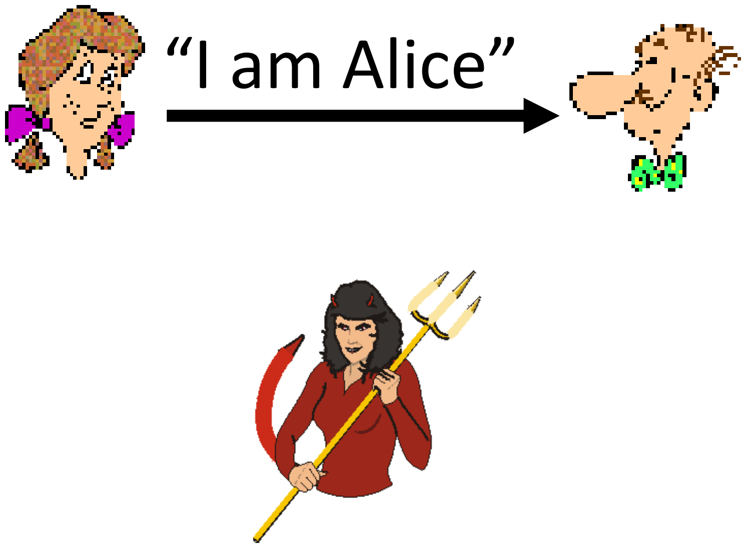
$$HMAC_k(x) = h[(k^+ \oplus opad)||h[(k^+ \oplus ipad)||x]]$$

Christof Paar , Jan Pelzl, Understanding Cryptography, 2010.

Authentication Protocol

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”

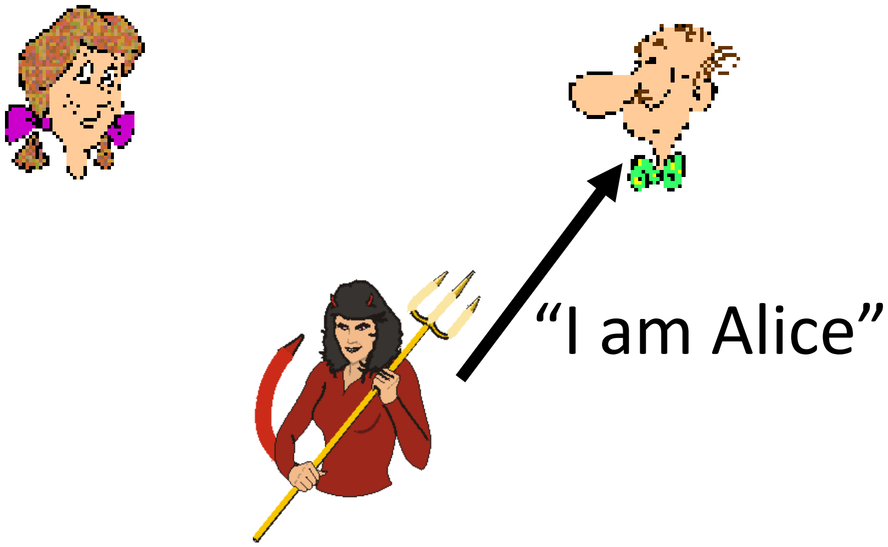


failure scenario??

Authentication Protocol

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"



in a network, Bob can not "see" Alice, so Trudy simply declares herself to be Alice

Authentication: another try

Goal: Bob wants Alice to “prove” her identity to him

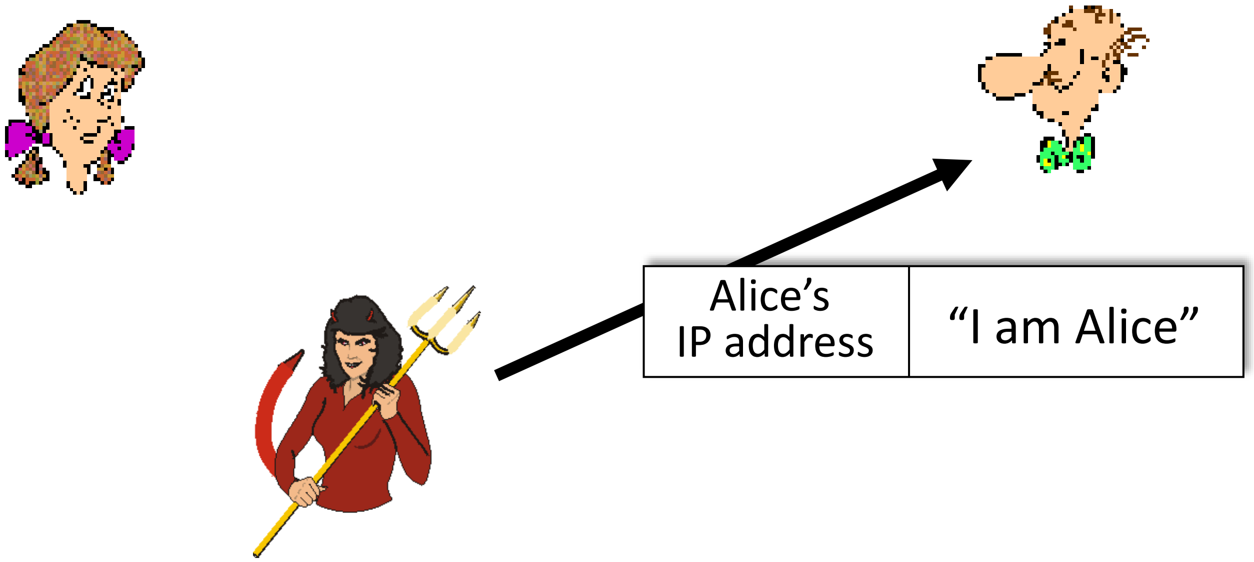
Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

failure scenario??

Authentication: another try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap2.0: Alice says “I am Alice” in an IP packet containing her source IP address

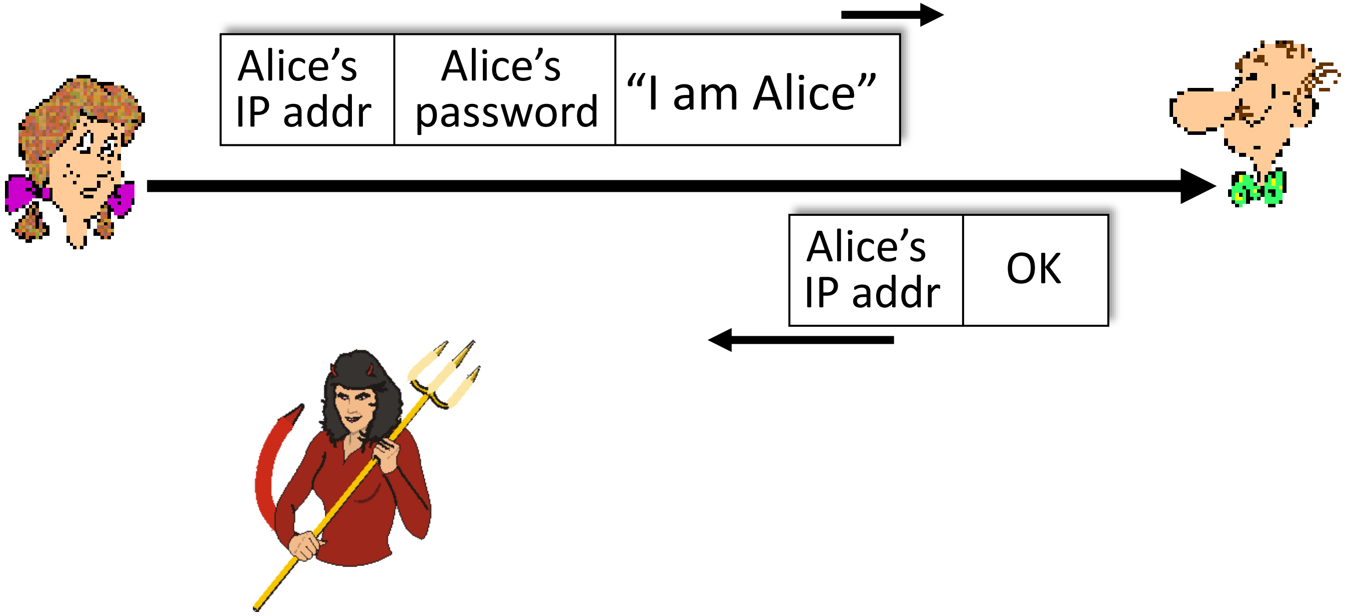


Trudy can create a packet “spoofing” Alice’s address

Authentication: a third try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.

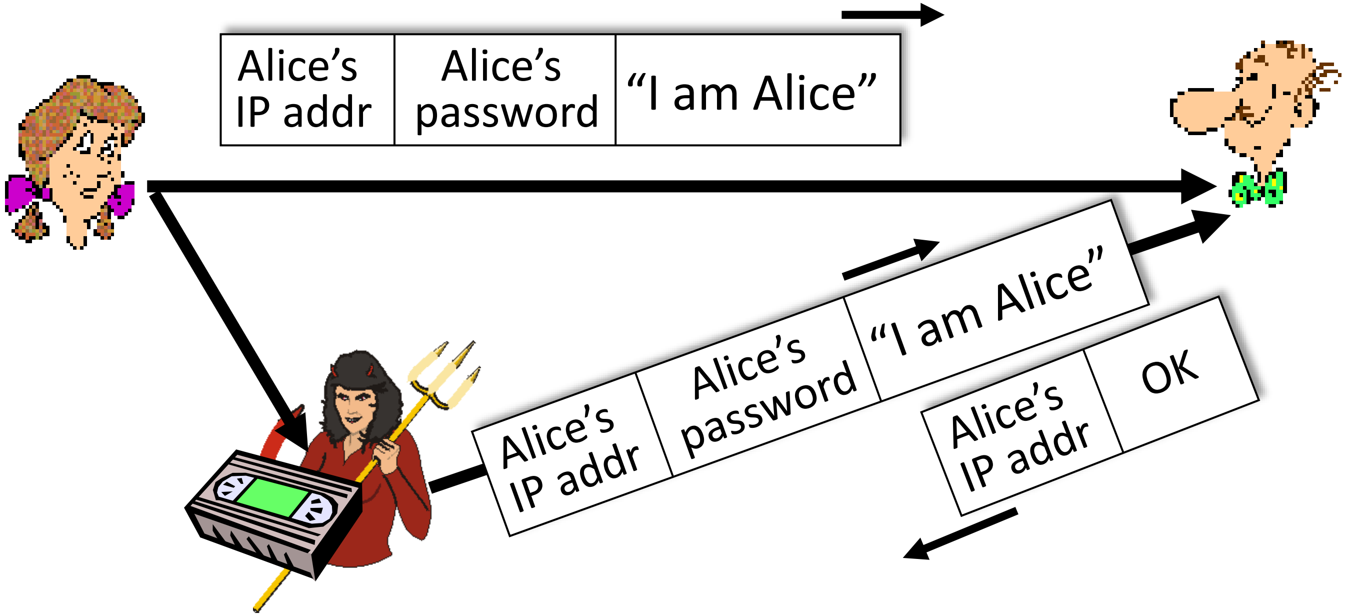


failure scenario??

Authentication: a third try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.

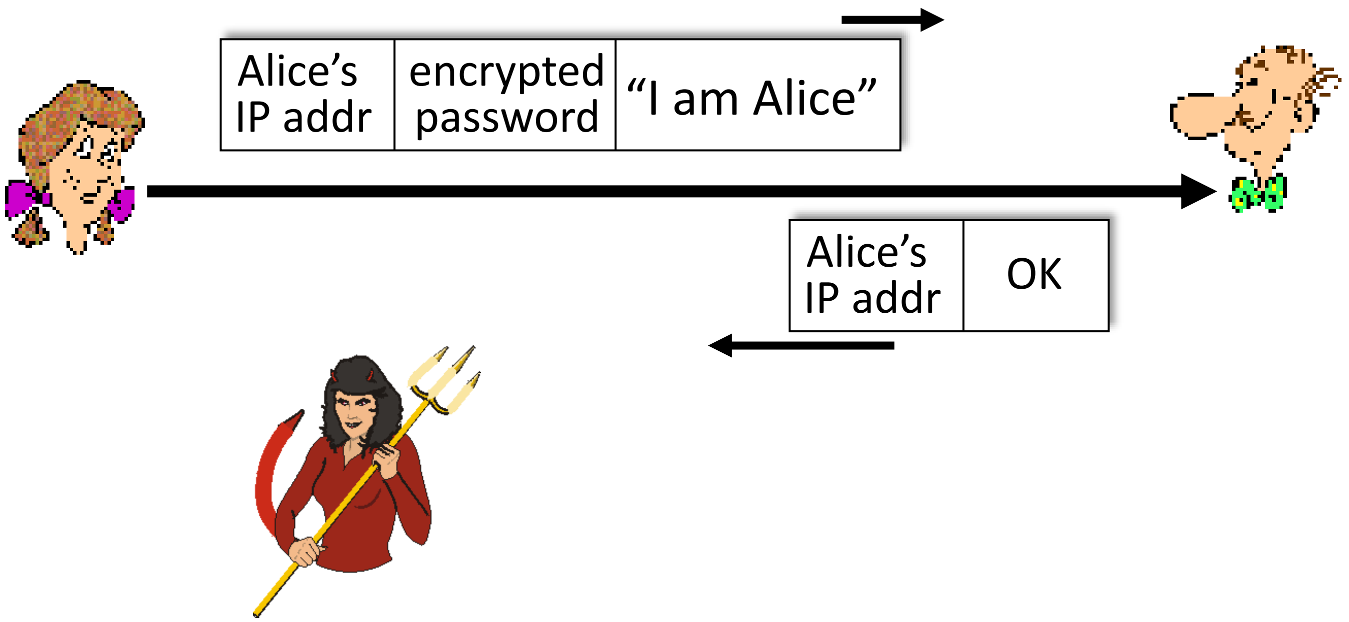


playback attack:
Trudy records Alice’s packet and later plays it back to Bob

Authentication: a modified third try

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap3.0: Alice says “I am Alice” and sends her encrypted secret password to “prove” it.

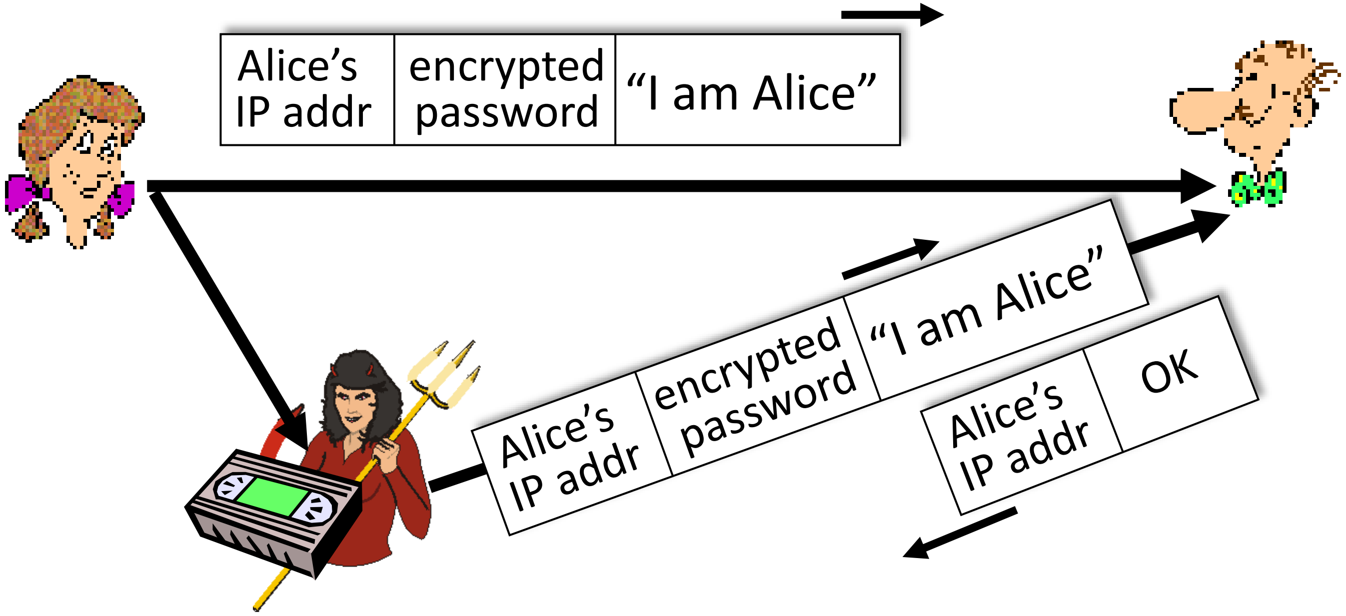


failure scenario??

Authentication: a modified third try

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap3.0: Alice says "I am Alice" and sends her encrypted secret password to "prove" it.



playback attack still works: Trudy records Alice's packet and later plays it back to Bob

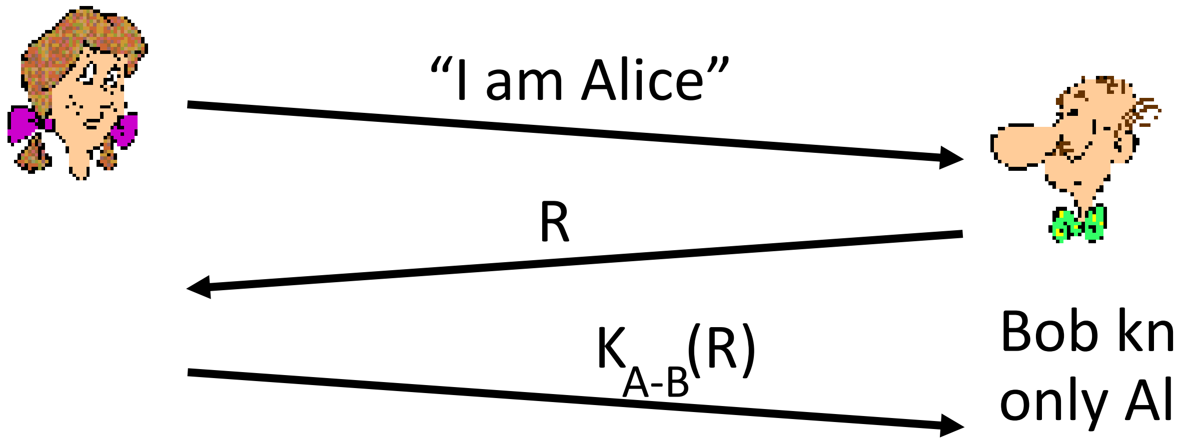
Authentication: a fourth try

Goal: avoid playback attack

nonce: number (R) used only once-in-a-lifetime

protocol ap4.0: to prove Alice "live", Bob sends Alice nonce, R

- Alice must return R, encrypted with shared secret key



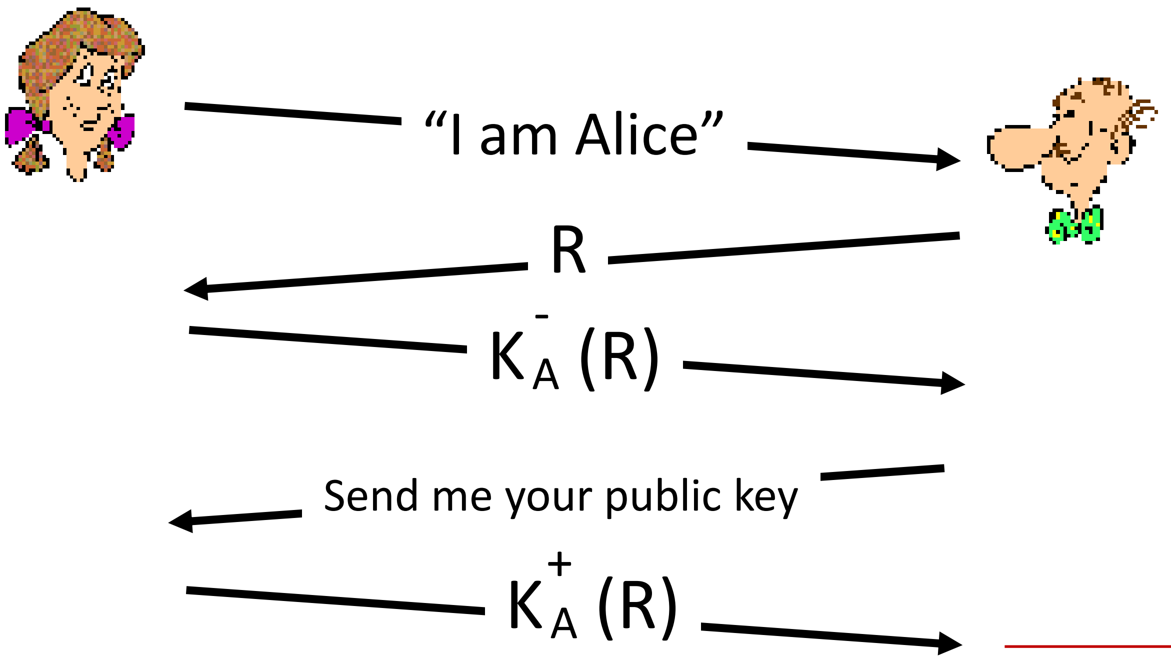
Bob know Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

Failures, drawbacks?

Authentication: ap5.0

ap4.0 requires shared symmetric key - can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



Bob computes

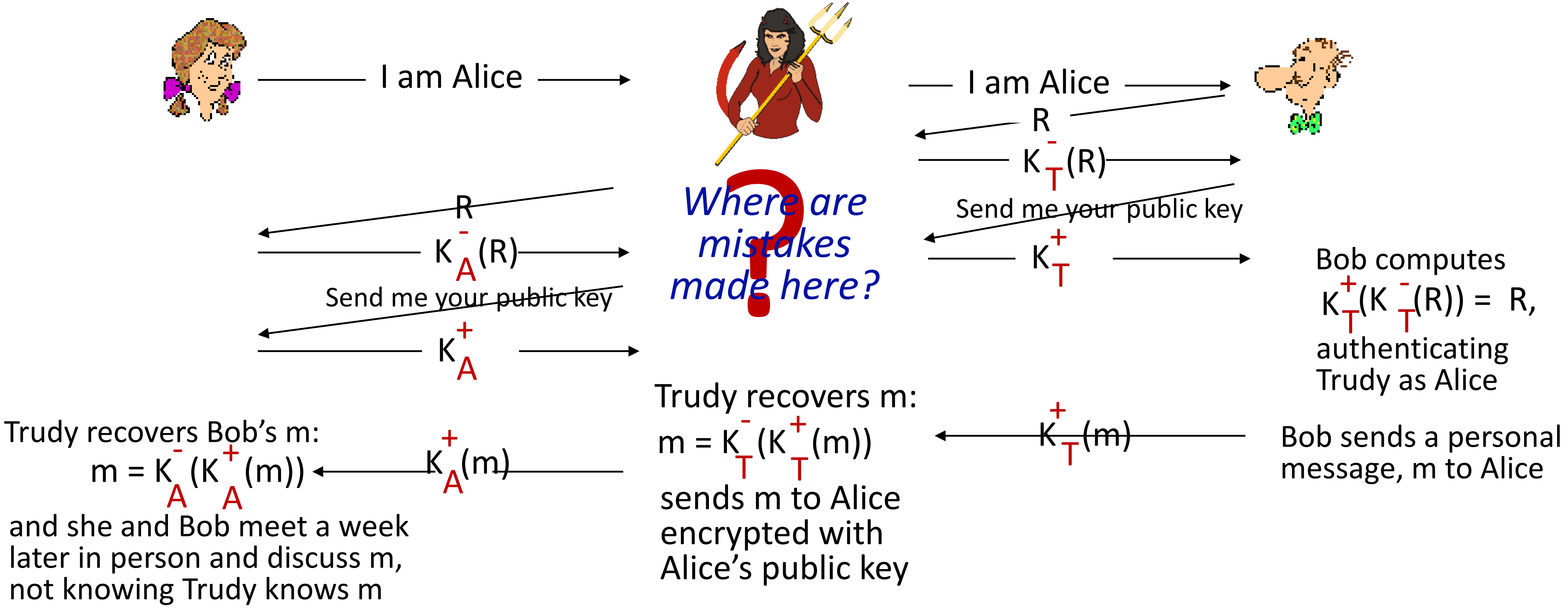
$$K_A^+(K_A^-(R)) = R$$

and knows only Alice could have the private key, that encrypted R such that

$$K_A^+(K_A^-(R)) = R$$

Authentication: ap5.0 – there’s still a flaw!

man (or woman) in the middle attack: Trudy poses as Alice (to Bob) and as Bob (to Alice)



Need for certified public keys

- motivation: Trudy plays pizza prank on Bob
  - Trudy creates e-mail order:
    Dear Pizza Store, Please deliver to me four pepperoni pizzas. Thank you, Bob
  - Trudy signs order with her private key
  - Trudy sends order to Pizza Store
  - Trudy sends to Pizza Store her public key, but says it's Bob's public key
  - Pizza Store verifies signature; then delivers four pepperoni pizzas to Bob
  - Bob doesn't even like pepperoni

Public key Certification Authorities (CA)

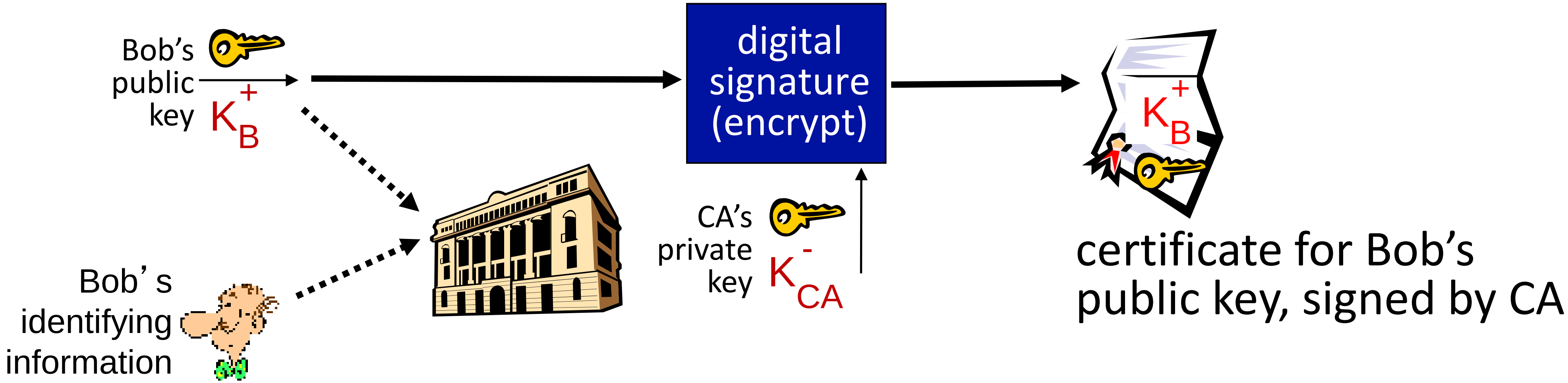
- certification authority (CA): binds public key to particular entity, E
- entity (person, website, router) registers its public key with “proof of identity” to CA
  - CA creates certificate binding identity E to E’s public key

Certification Scheme

- We can place the following requirements on Certificate scheme
  - **Any participant can read a certificate** to determine the **name** and **public key** of the certificate's owner.
  - **Any participant can verify** that the certificate originated from the certificate authority and is not counterfeit.
  - **Only the certificate authority** can **create** and **update** certificates.
  - Any participant can **verify the time validity** of the certificate.

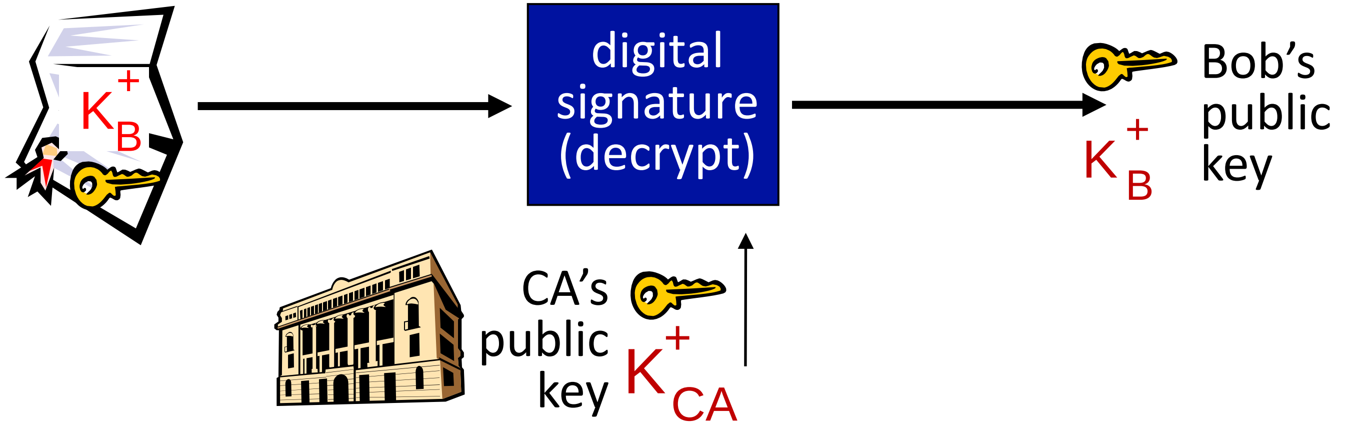
Public Key Certification

- Certificate containing E's public key digitally signed by CA: CA says "this is E's public key"



Public key Certification Authorities (CA)

- when Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere)
  - apply CA's public key to Bob's certificate, get Bob's public key



Certification Scheme

- For participant A, the authority provides a certificate of the form

$$C_A = E(PR_{auth}, [T||ID_A||PU_A])$$

where PR_{auth} is the private key used by the authority and T is a timestamp. PU_a and ID_A are the Alice's public key and ID.

- Anyone can verify the certificate as follows

$$D(PU_{auth}, \mathrm{C_A}) = \mathrm{D}(PU_{auth}, E(PR_{auth}, [T||ID_A\ ||PU_A])\ ) = (T||ID_A\ ||PU_A)$$

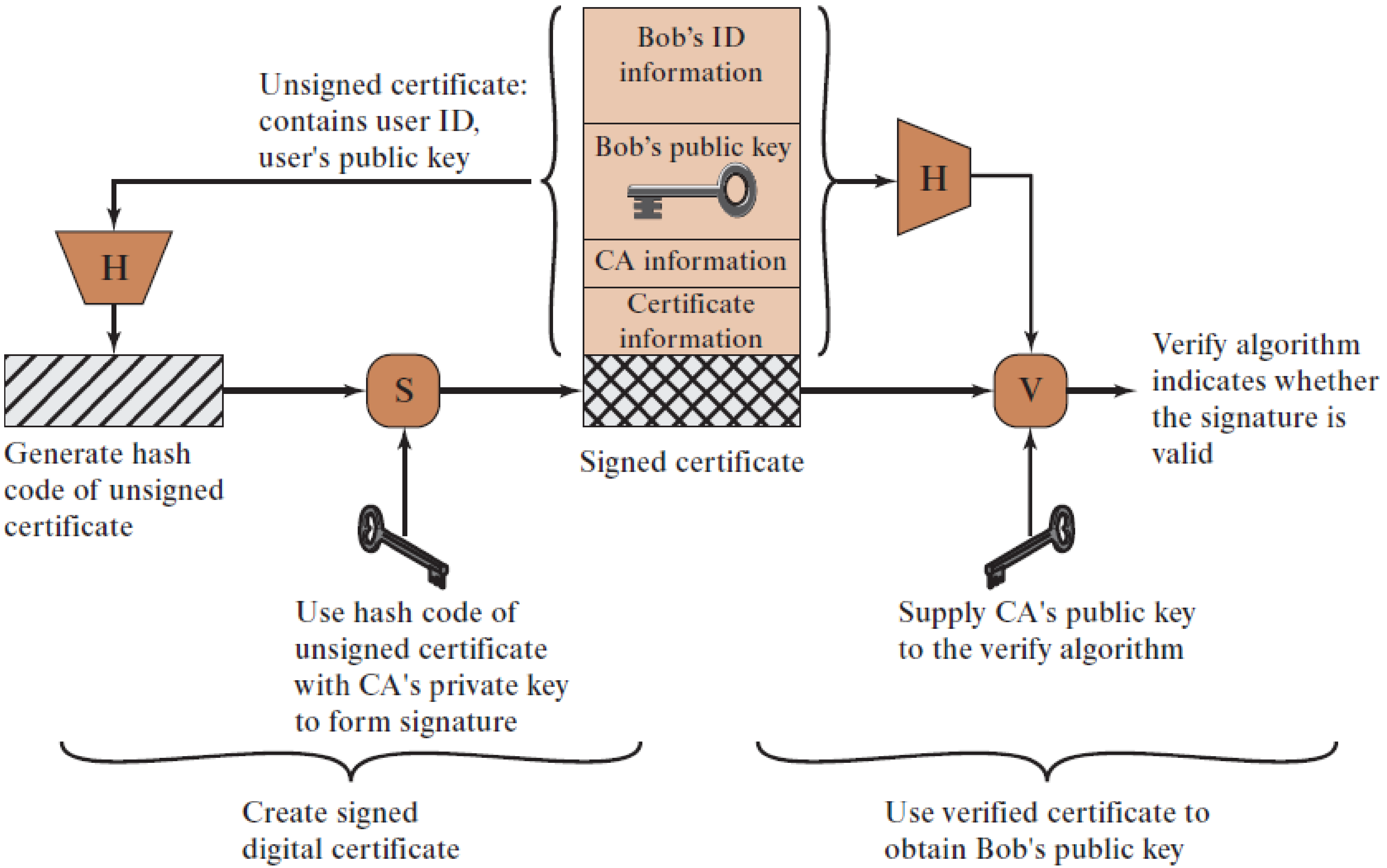
where PU_{auth} is the public key used by the authority

X.509 Certificates

- X.509 is **the Internationally accepted standard** for how to construct a public key certificate, and is becoming widely used.
  - It has gone through several versions.

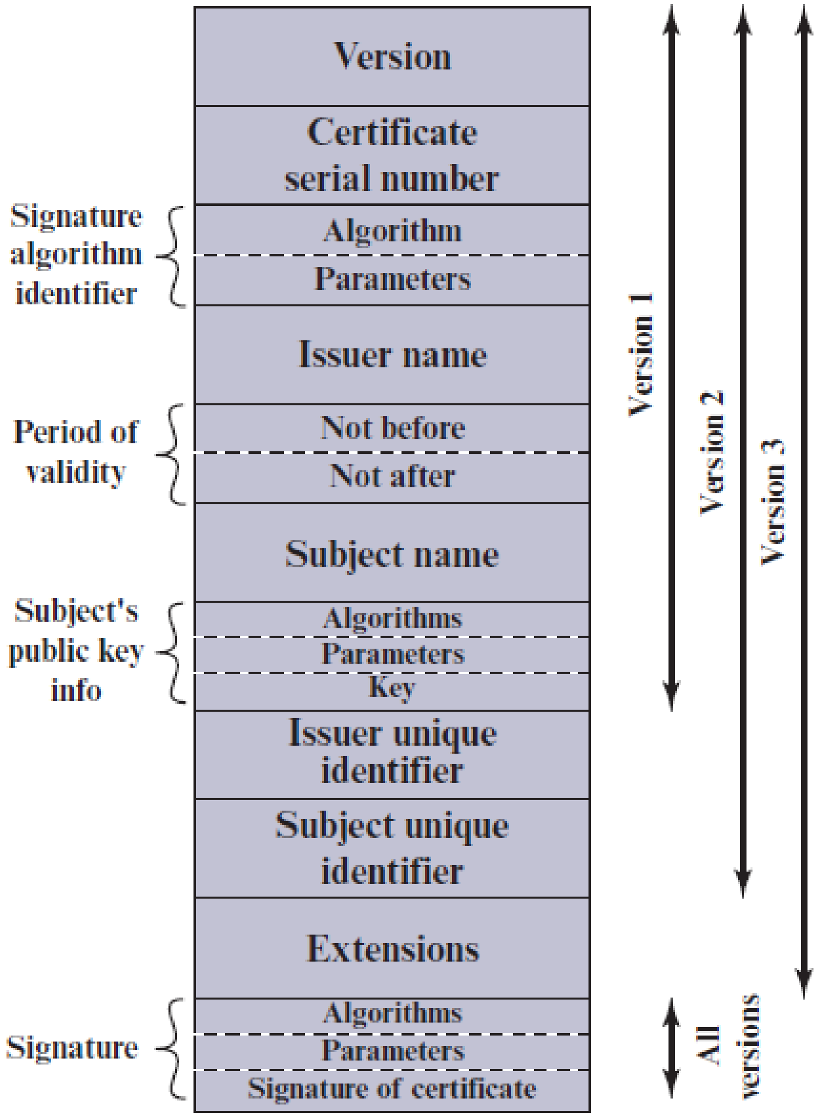
- Defines framework for authentication services:
  - Defines that public keys stored as certificates in a public directory.
  - Certificates are issued and signed by an entity called certification authority (CA).
  - Used by numerous applications: SSL, IPSec, SET
  - Example: see certificates accepted by your browser

X.509 Certificates



X.509 Certificates

- issued by a Certification Authority (CA), containing:
  - version V (1, 2, or 3)
  - serial number SN (unique within CA) identifying certificate
  - signature algorithm identifier
  - issuer name
  - period of validity (from - to dates)
  - subject name (name of owner)
  - subject public-key info Ap (algorithm, parameters, key)
  - issuer unique identifier
  - subject unique identifier
  - extension fields
  - signature (of hash of all fields in certificate)



(a) X.509 certificate

Figure 15.11 X.509 Formats

Verifying Identity

- A CA verifies some site's identity. How?
- It's easy to verify the existence of a corporation—but how do you verify who speaks for it?
- For publicly traded companies, top executives are generally a matter of public record—do the CEO or CTO have to show up in person?

Many Different Ways!

- From the Baseline Requirements document from the CA Browser Forum, verification can be done by:
  - Documents from or communication with a government agency;
  - An in-person visit by the CA;
  - A reliable third-party database;
  - An “attestation letter” from a lawyer or accountant, etc.;
  - More. . .
- These procedures are manual, annoying, and complex—can we automate them?

Automated Web CAs

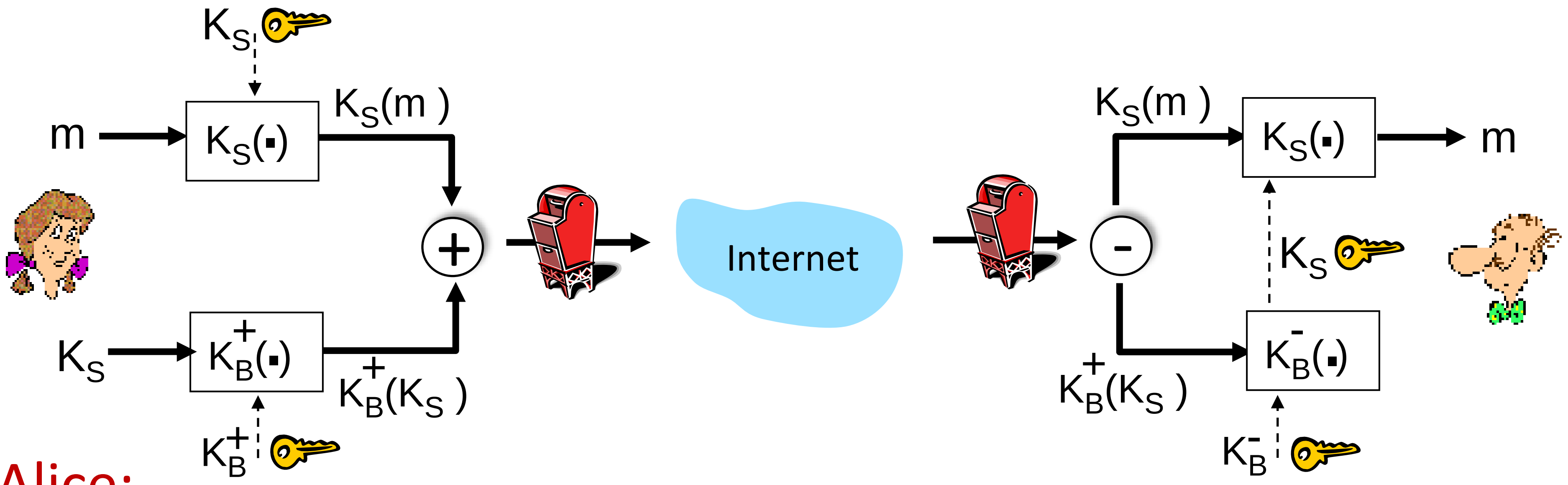
- Web site certificates should be issued to the owner of the web site, e.g., the party that controls the site
- Demonstrate control of the web site by doing something only the site owner can do
- Example: the CA sends the site a random number; the site puts that number into a specific URL

Outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec

Secure e-mail: confidentiality

Alice wants to send *confidential* e-mail, m, to Bob.

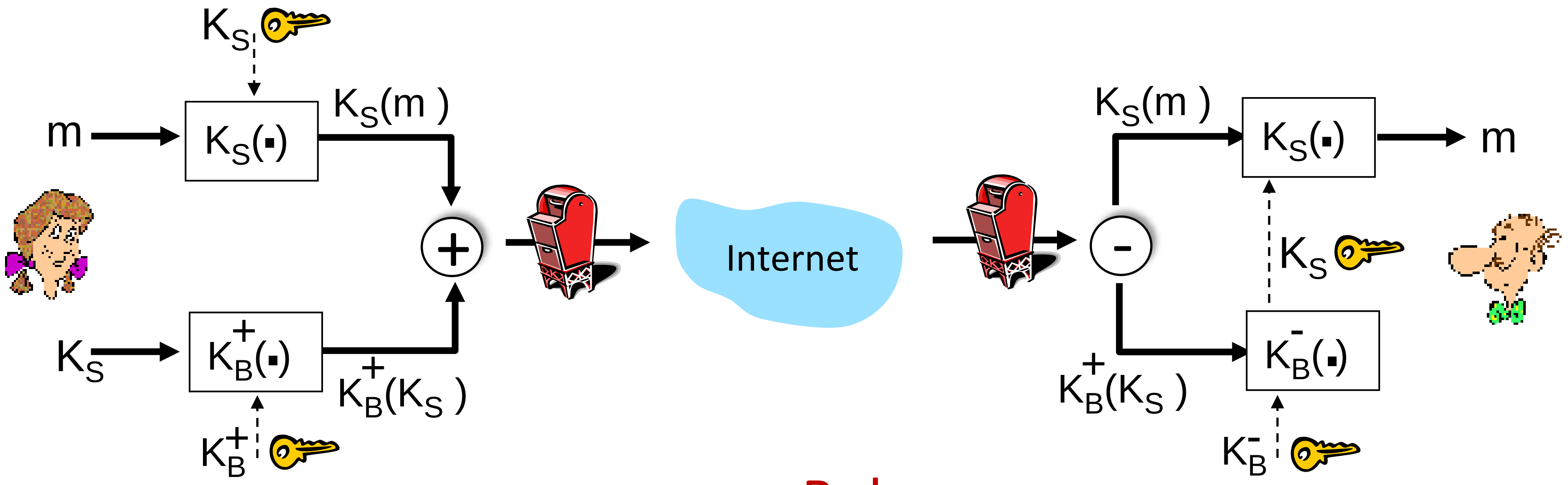


Alice:

- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K^+_B(K_S)$ to Bob

Secure e-mail: confidentiality (more)

Alice wants to send *confidential* e-mail, m, to Bob.

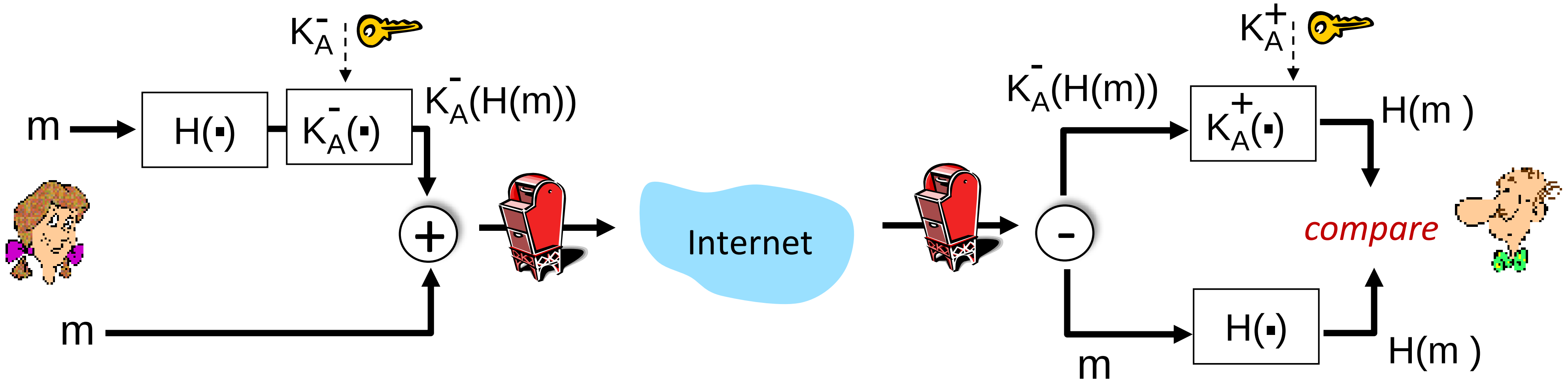


Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure e-mail: integrity, authentication

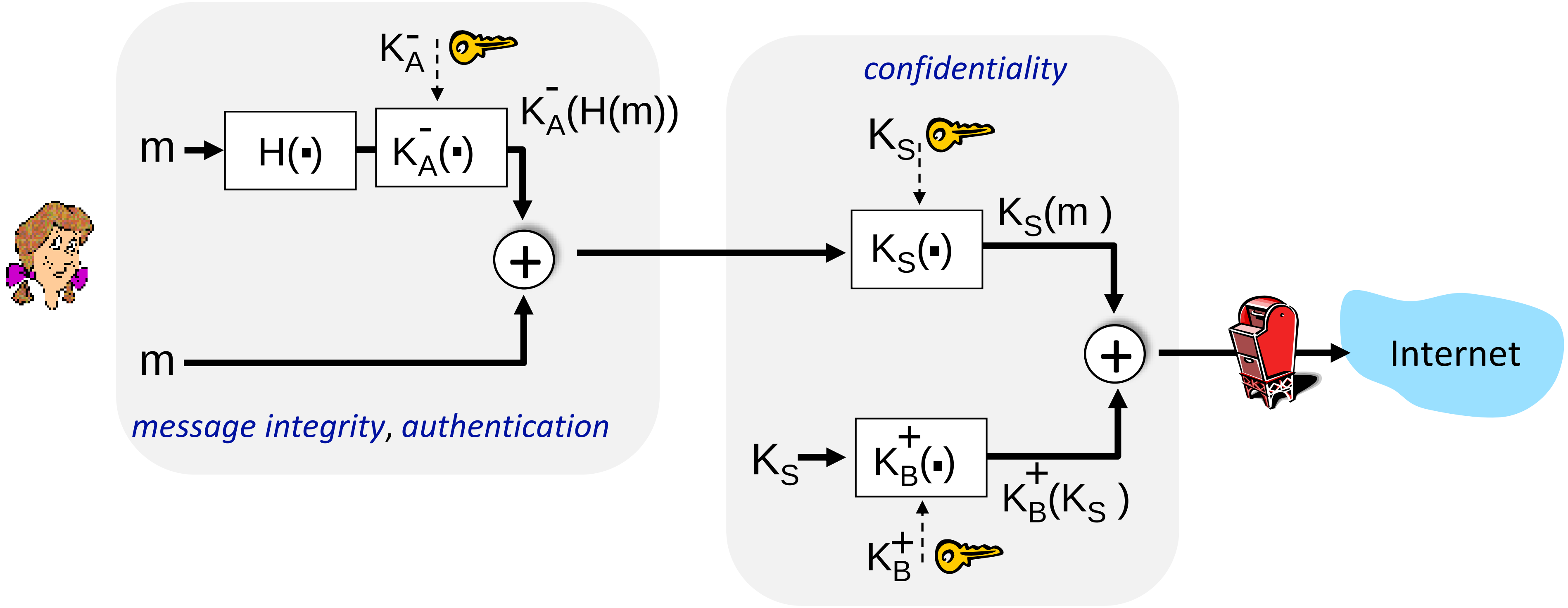
Alice wants to send m to Bob, with *message integrity*, *authentication*



- Alice digitally signs hash of her message with her private key, providing integrity and authentication
- sends both message (in the clear) and digital signature

Secure e-mail: confidentiality, integrity, authen.

Alice sends m to Bob, with *confidentiality, message integrity, authentication*



Alice uses three keys: her private key, Bob's public key, new symmetric key

What are Bob's complementary actions?

Outline

- What is network security?
- Principles of cryptography
- Authentication, message integrity
- Securing e-mail
- Securing TCP connections: TLS
- Network layer security: IPsec

Transport-layer security (TLS)

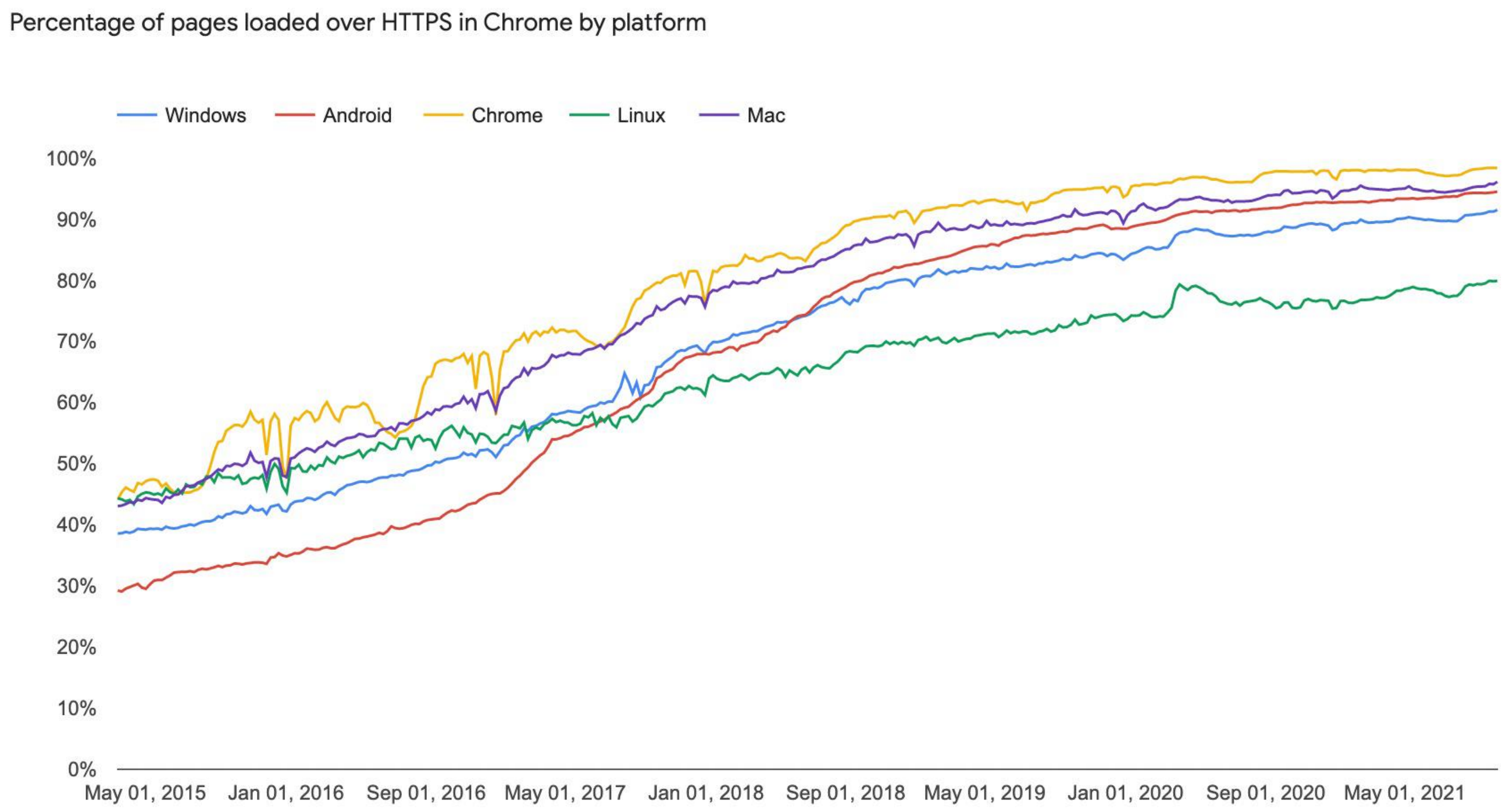
- widely deployed security protocol above the transport layer
  - supported by almost all browsers, web servers: https (port 443)
- provides:
  - confidentiality: via *symmetric encryption*
  - integrity: via *cryptographic hashing (MAC)*
  - authentication: via *public key cryptography*

all techniques we have studied!

Importance of TLS

- Originally designed for secure e-commerce over http, now used much more widely.
  - Retail customer access to online banking facilities.
  - Access to Gmail, Facebook, Yahoo, etc.
  - >70% of web traffic is now encrypted using TLS.
- TLS has become the de facto secure protocol of choice.
  - Used by hundreds of millions/billions of people and devices every day.

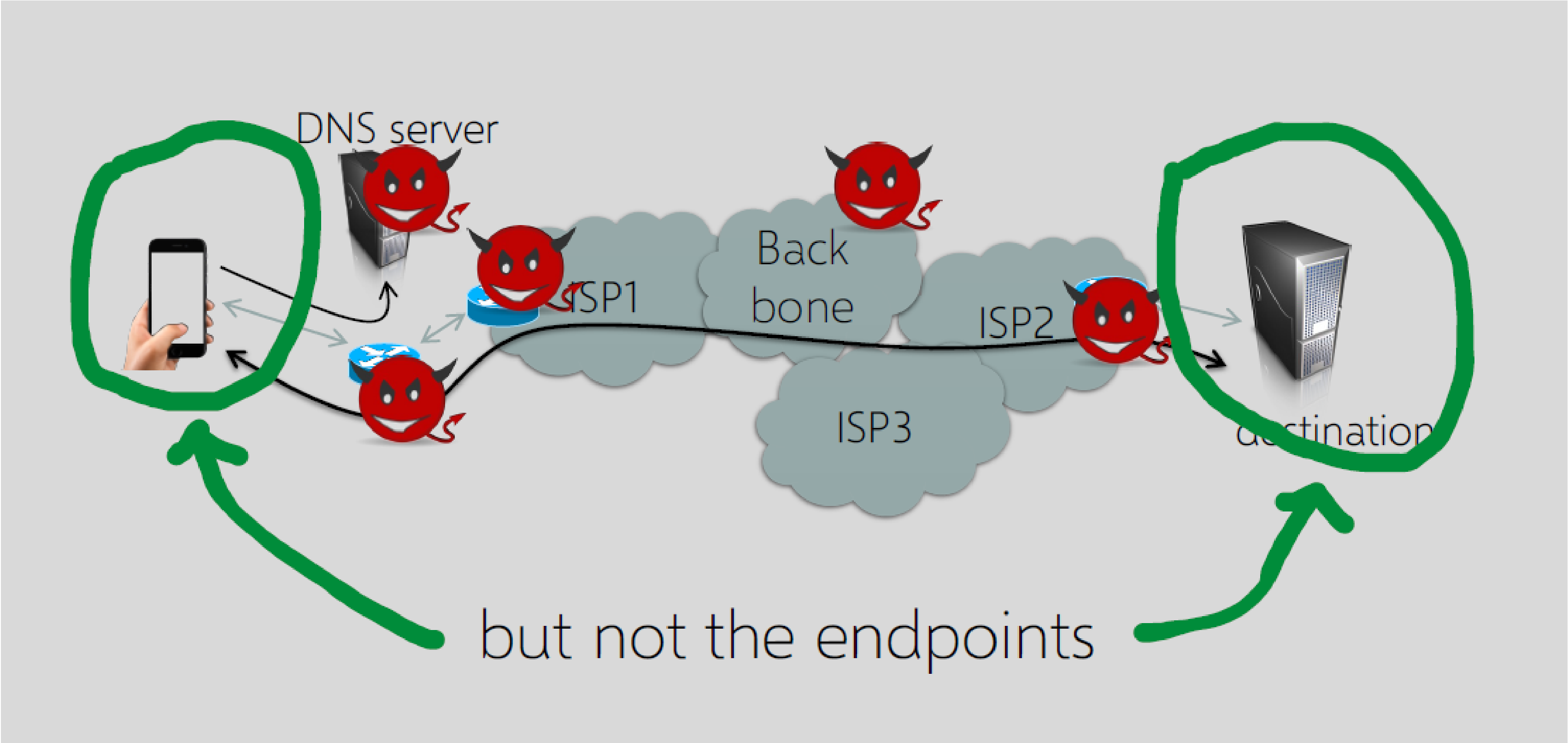
HTTPS Adoption by Websites



https://transparencyreport.google.com/https/overview

TLS Threat Model

- End-to-end secure communications in
- the presence of a network attacker



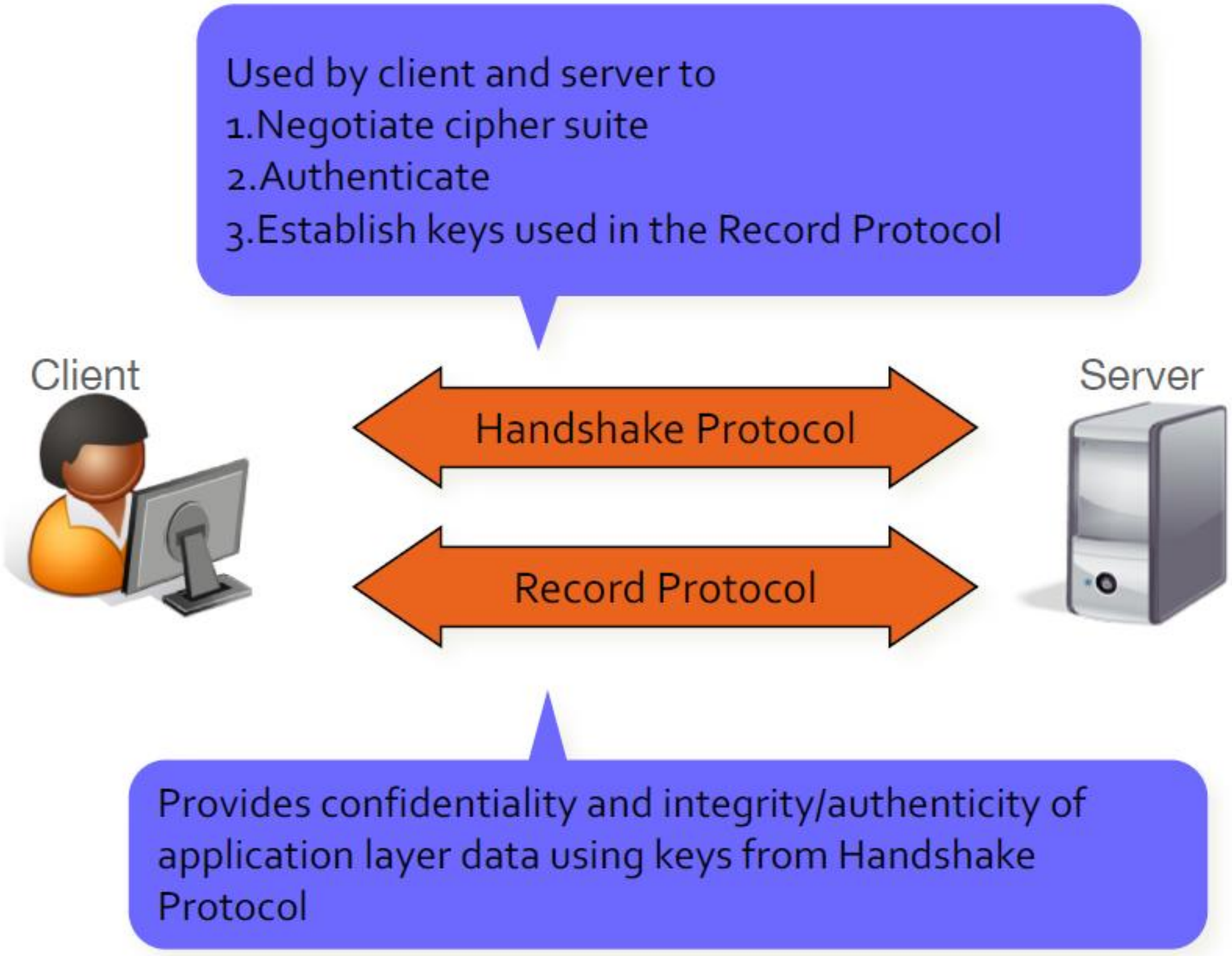
SSL/TLS Overview

- SSL = Secure Sockets Layer.
  - Developed by Netscape in mid 1990s.
  - SSLv1 broken at birth.
  - SSLv2 seriously flawed in various ways.
  - SSLv3 now considered broken.
- TLS = Transport Layer Security.
  - IETF-standardised version of SSL.
  - TLS 1.0 in RFC 2246 (1999).
  - TLS 1.1 in RFC 4346 (2006).
  - TLS 1.2 in RFC 5246 (2008).
  - TLS 1.3 in RFC 8446 (2018).

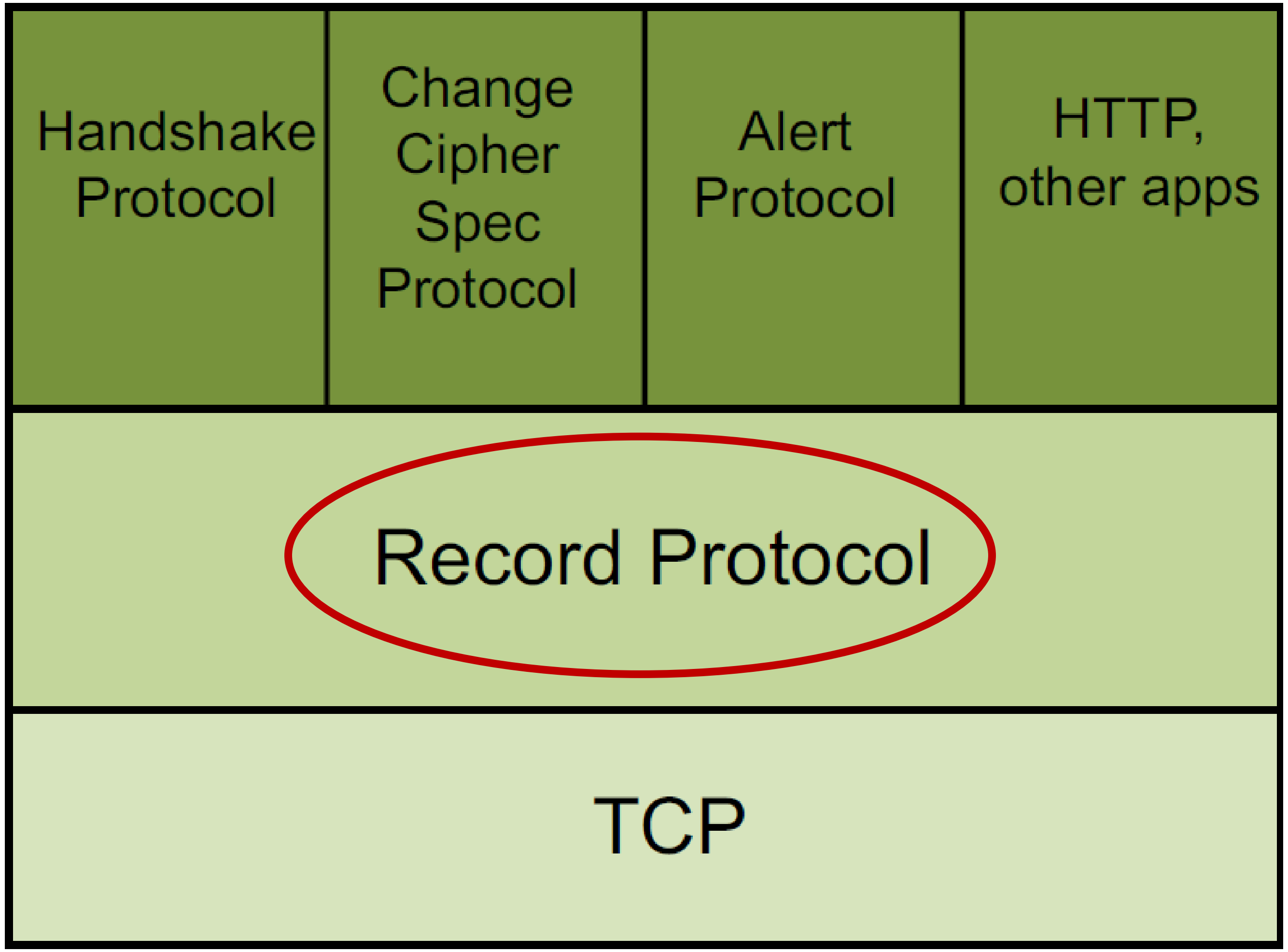
Transport-layer security: what's needed?

- let's *build* a toy TLS protocol, *t-tls,* to see what's needed!
- we've seen the "pieces" already:
  - handshake: Alice, Bob use their certificates, private keys to authenticate each other, exchange or create shared secret
  - key derivation: Alice, Bob use shared secret to derive set of keys
  - data transfer: stream data transfer: data as a series of records
    - not just one-time transactions
  - connection closure: special messages to securely close connection

Highly Simplified View of TLS



TLS Protocol Architecture



TLS: encrypting data

- recall: TCP provides data *byte stream* abstraction
- Q: can we encrypt data in-stream as written into TCP socket?
  - *A:* where would MAC go? If at end, no message integrity until all data received and connection closed!
  - *solution:* break stream in series of "records (fragments)"
    - each client-to-server record carries a MAC, created using M_c
    - receiver can act on each record as it arrives
- t-tls record encrypted using symmetric key, $K_{c,}$ passed to TCP:

$K_c($ | *length* | *data* | *MAC* | $)$

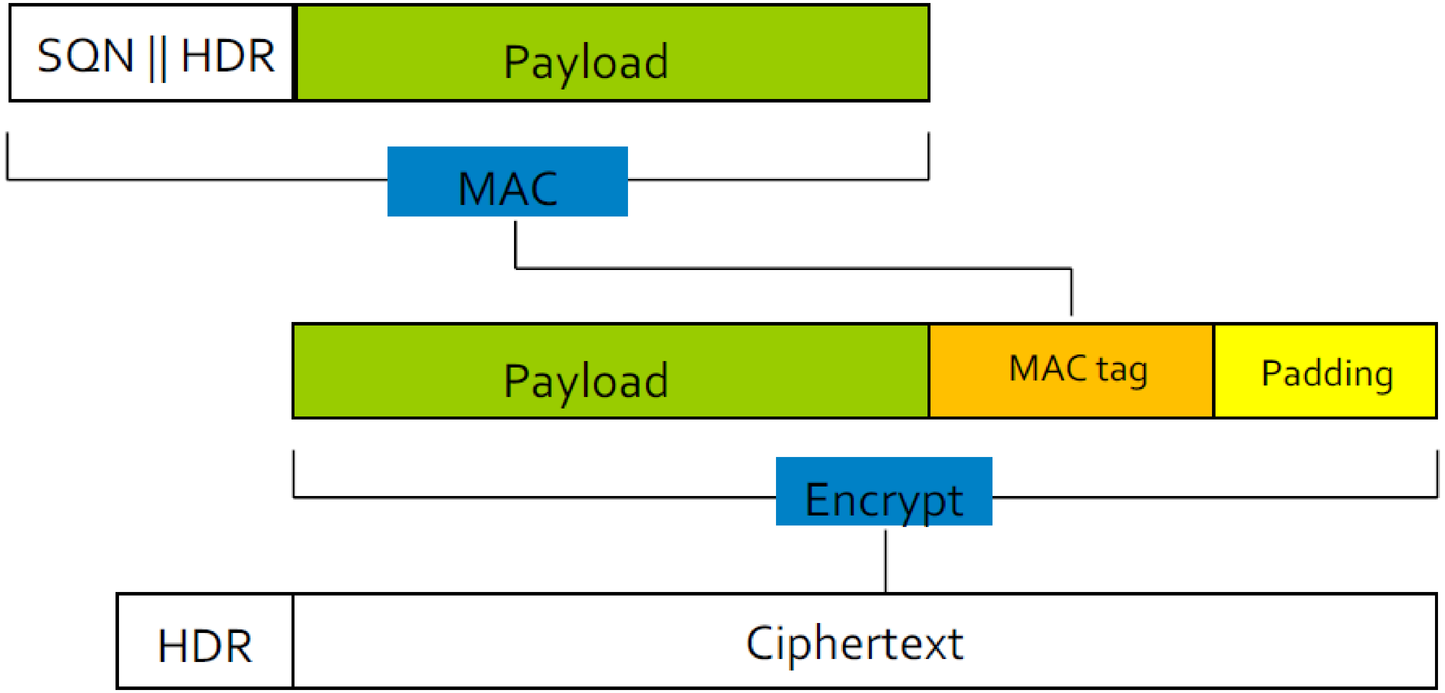
TLS: encrypting data (more)

- possible attacks on data stream?
  - *re-ordering:* man-in middle intercepts TCP segments and reorders (manipulating sequence #s in unencrypted TCP header)
  - *replay*
- solutions:
  - use TLS sequence numbers (data, TLS-seq-# incorporated into MAC)
  - use nonce

TLS Record Protocol

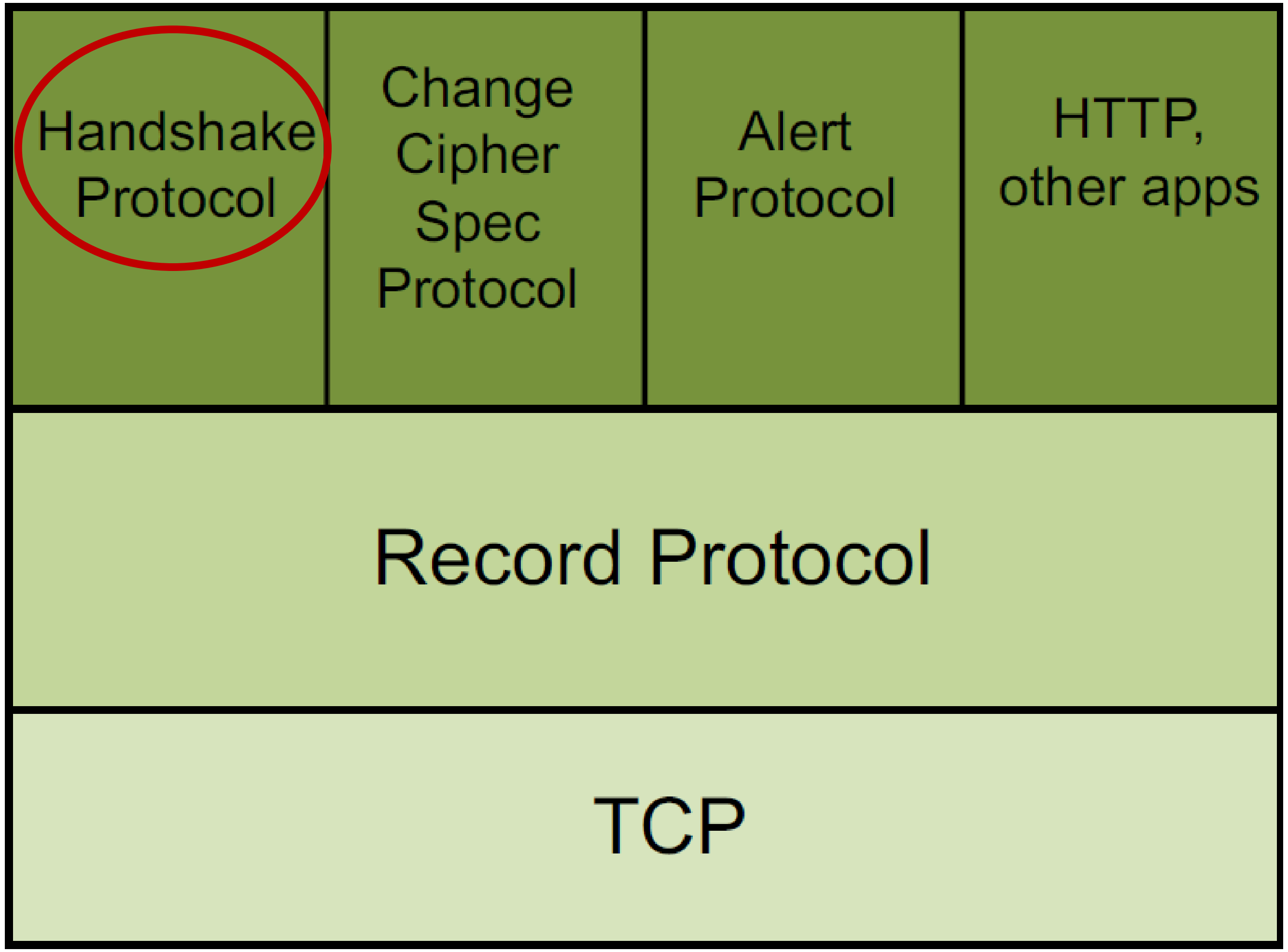
- TLS Record Protocol provides:
  - Data origin authentication, integrity using a MAC.
  - Confidentiality using a symmetric encryption algorithm.
  - Anti-replay using sequence numbers protected by the MAC.
  - Optional compression.

TLS Record Protocol Operation

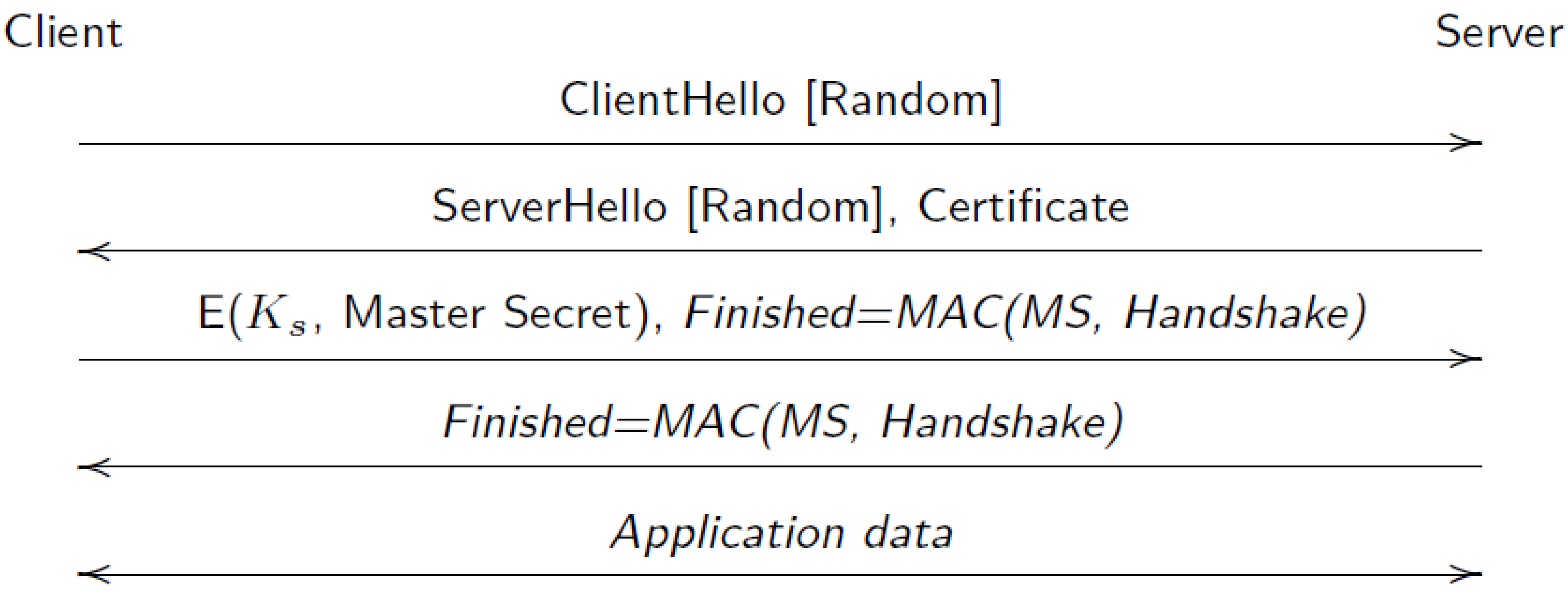


MAC	HMAC-MD5, HMAC-SHA1, HMAC-SHA256,…
Encrypt	CBC-AES128, CBC-AES256, CBC-3DES, RC4-128,…

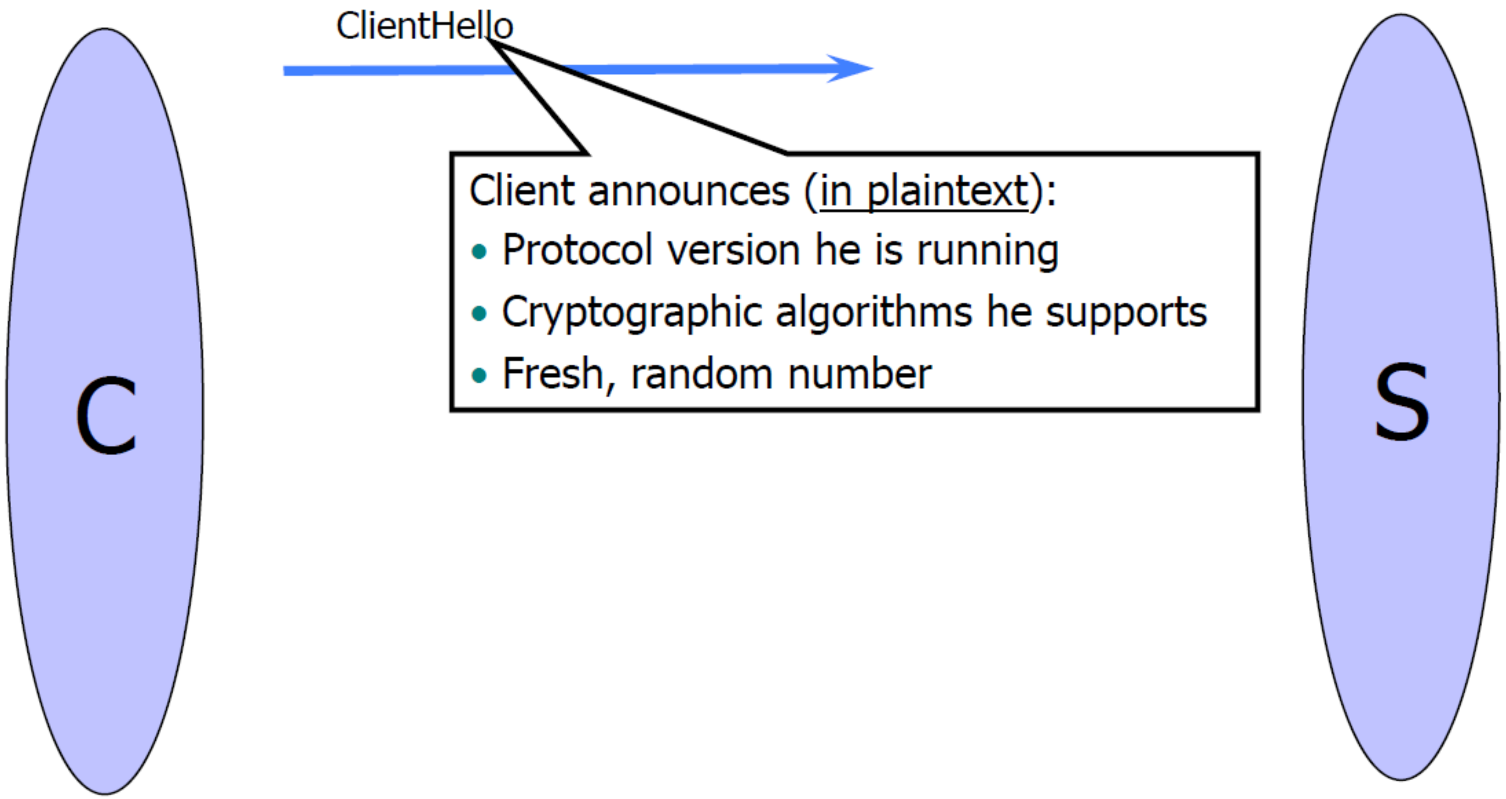
TLS Protocol Architecture



TLS 1.2: RSA Handshake Skeleton



ClientHello



ClientHello

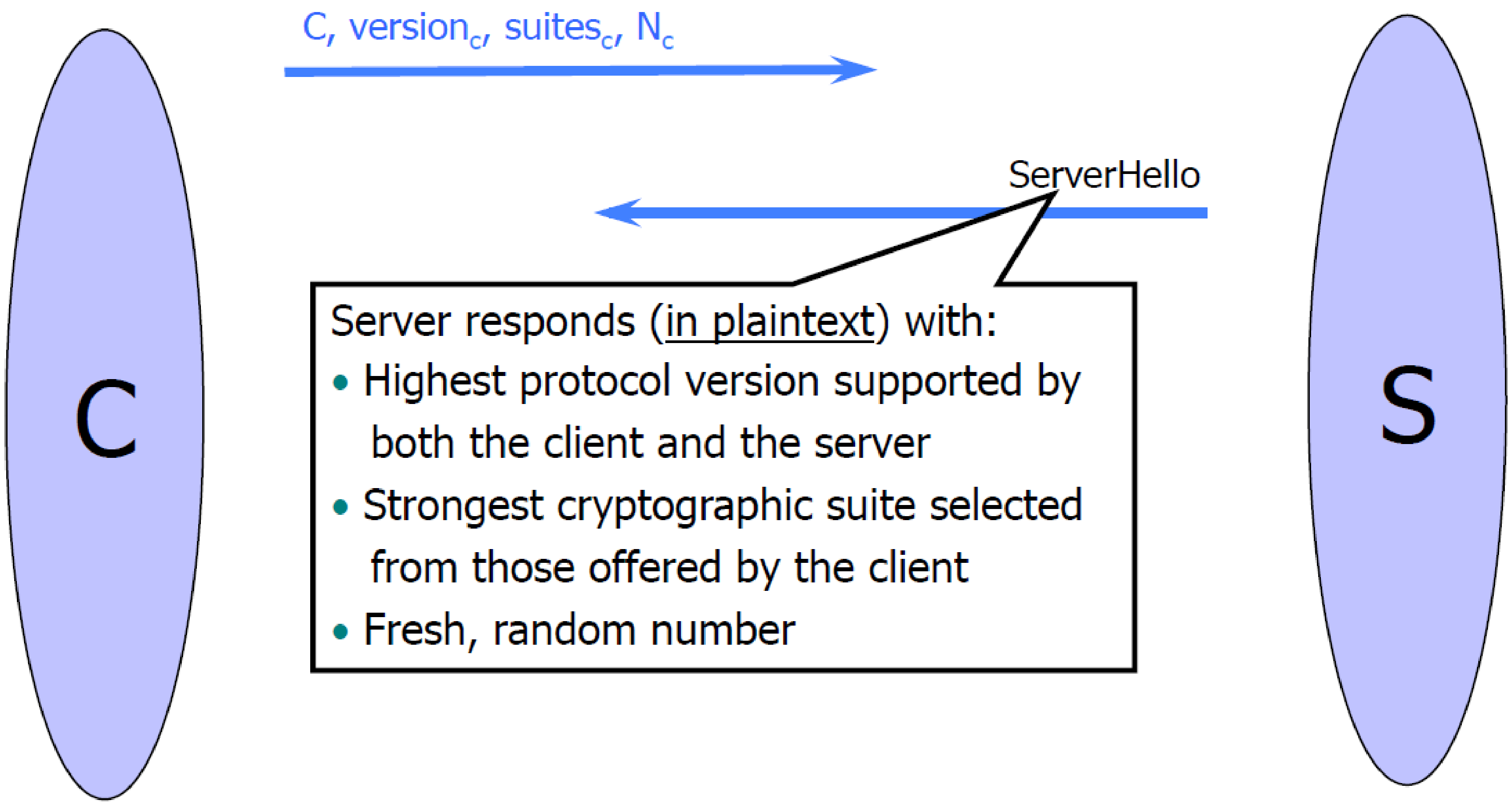
```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites;
    CompressionMethod compression_methods;
} ClientHello
```

Highest version of the protocol supported by the client

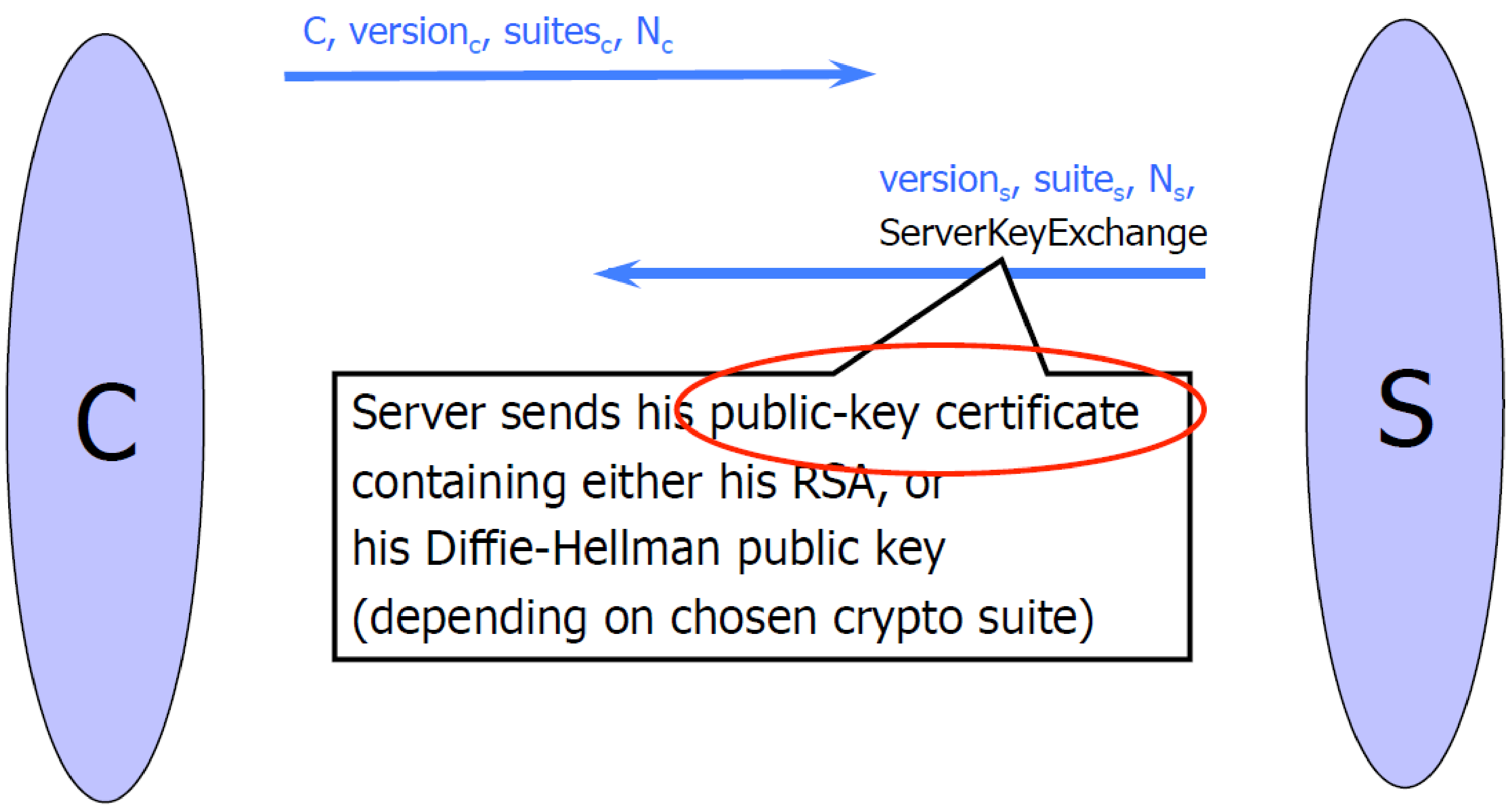
Session id (if the client wants to resume an old session)

Set of cryptographic algorithms supported by the client (e.g., RSA or Diffie-Hellman)

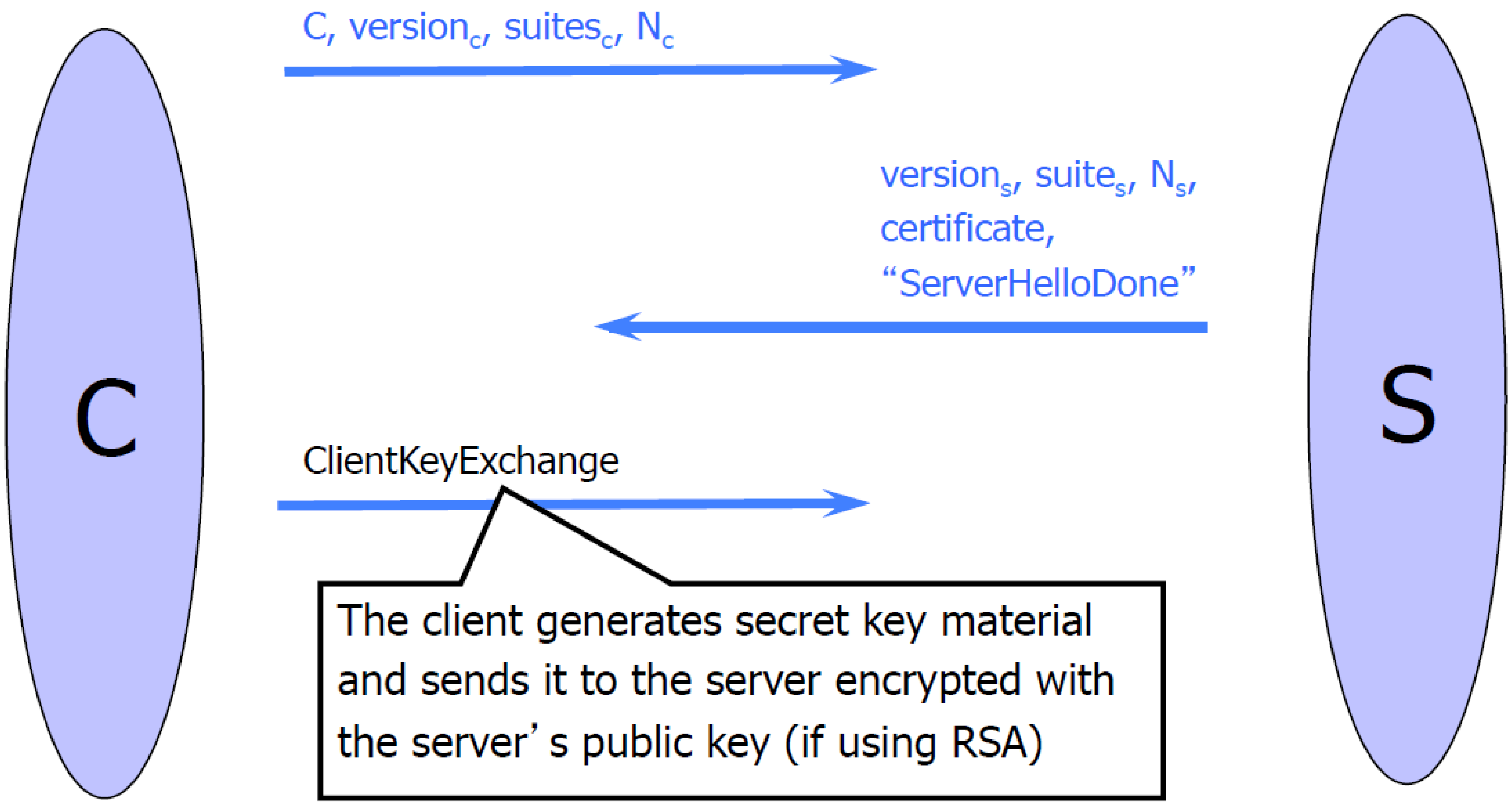
ServerHello



ServerHello



ClientKeyExchange



ClientKeyExchange

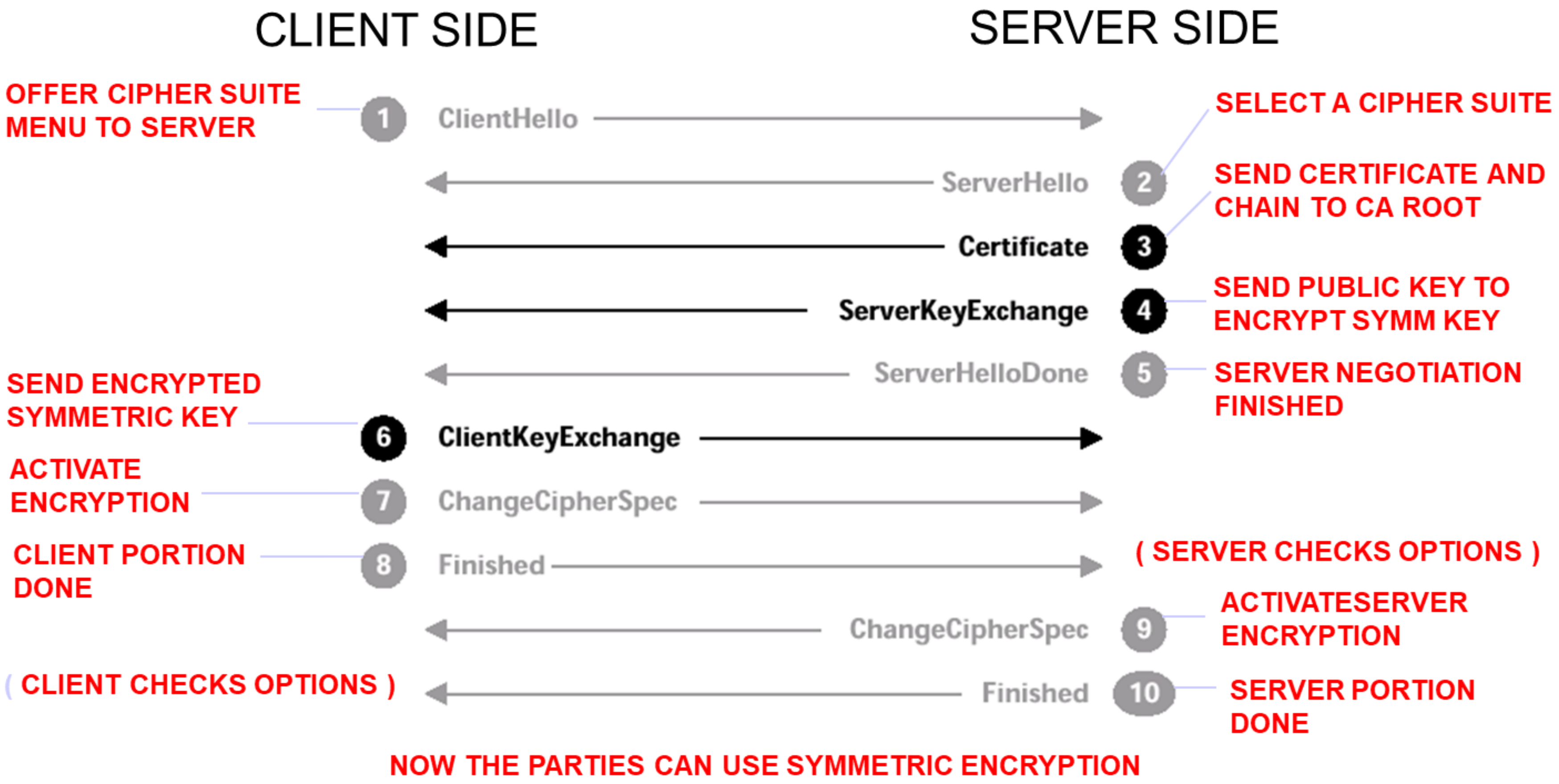
```
struct {
   select (KeyExchangeAlgorithm) {
      case rsa: EncryptedPreMasterSecret;
      case diffie_hellman: ClientDiffieHellmanPublic;
   } exchange_keys
} ClientKeyExchange
```

Where do random bits come from?

```
struct {
   ProtocolVersion client_version;
   opaque random[46];
} PreMasterSecret
```

Random bits from which symmetric keys will be derived (by hashing them with nonces)

TLS Handshake



SOURCE: THOMAS, *SSL AND TLS ESSENTIALS*

TLS: cryptographic keys

- considered bad to use same key for more than one cryptographic function
  - different keys for message authentication code (MAC) and encryption
- four keys:
  - K_c : encryption key for data sent from client to server
  - M_c : MAC key for data sent from client to server
  - K_s : encryption key for data sent from server to client
  - M_s : MAC key for data sent from server to client
- keys derived from key derivation function (KDF)
  - takes master secret and (possibly) some additional random data to create new keys

TLS 1.3 Objectives

- Clean up: Remove unused or unsafe features
- Security: Improve security by using modern security analysis techniques
- Privacy: Encrypt more of the protocol
- Performance: Our target is a 1-RTT handshake for naive clients; 0-RTT handshake for repeat connections
- Continuity: Maintain existing important use cases

TLS: 1.3 cipher suite

- "cipher suite": algorithms that can be used for key generation, encryption, MAC, digital signature
- TLS: 1.3 (2018): more limited cipher suite choice than TLS 1.2 (2008)
  - only 5 choices, rather than 37 choices
  - *requires* Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined encryption and authentication algorithm ("authenticated encryption") for data rather than serial encryption, authentication
    - 4 based on AES
  - HMAC uses SHA (256 or 284) cryptographic hash function

References

- Kurose, James F., and Keith W. Ross. "Computer networking: A top-down approach edition." Addision Wesley (2007), chapter 8.
- Cryptography and Network Security: Principles and Practice, William Stallings, Pearson, 2022.
- COMS W4182 — Computer Security II (Spring ‘22), Prof. Steven M. Bellovin, Columbia University.
- CS 5435 - Fall 2022 “Security and Privacy Concepts in the Wild” Vitaly Shmatikov, Cornell University.
- Introducing TLS, Information Security Group, Kenny Paterson.