- Srinivas N Jay

Https = Http + SSL/TLS

Hypertext Transfer Protocol Secure (**HTTPS**) is a combination of the Hypertext Transfer Protocol with the SSL/TLS protocol to provide encryption and secure identification of the server. HTTPS connections are often used for payment transactions on the World Wide Web and for sensitive transactions in corporate information systems

Web browsers typically use HTTP to communicate with web servers, sending and receiving information without encrypting it. For sensitive transactions, such as Internet e-commerce or online access to financial accounts, the browser and server must encrypt this information.

Transport Layer Security (**TLS**) and its predecessor, **Secure Sockets Layer** (**SSL**), are cryptographic protocols that provide security for communications over networks such as the Internet. TLS and SSL encrypt the segments of network connections at the Transport Layer end-to-end.

Several versions of the protocols are in widespread use in applications like web browsing electronic mail Internet faxing instant messaging voice-over-IP (VoIP)

SSL / TLS in a Nutshell

- SSL & TLS provide a `secure TCP tunnel from client to server`:
  - Confidentiality
  - Message and connection integrity
  - Authentication of server, optionally also of client
- Original goal and main use: secure credit card number
- Implemented in almost all web clients, servers
- Many implementations, libraries, e.g. Open-SSL
- SSL: Secure Socket Layer
  - Since SSL (& TLS) operate on top of `standard` Sockets API
- TLS: Transport Layer Security
  - Since TLS (& SSL) secure TCP (the transport layer)
  - IETF standard version of SSL
  - We usually say just SSL but refer to both

Trust & Security Wish-list

- Confidentiality & authenticity of messages
- Server (site) authentication:
  - Customer needs to identify bank, merchant, etc.
- Credential validation:
  - Validate rating, certification, other credentials
  - E.g. E-bay rating of seller, BBB seal,…
- Client authentication:
  - Bank needs to identify account holder
  - Company needs to identify employee
  - Content provider needs to identify subscriber
- Non-repudiation:
  - Customer cannot dispute approving payments/orders
  - Bank cannot dispute statement, receiving orders, etc.
- Denial of service

Good in SSL/TLS

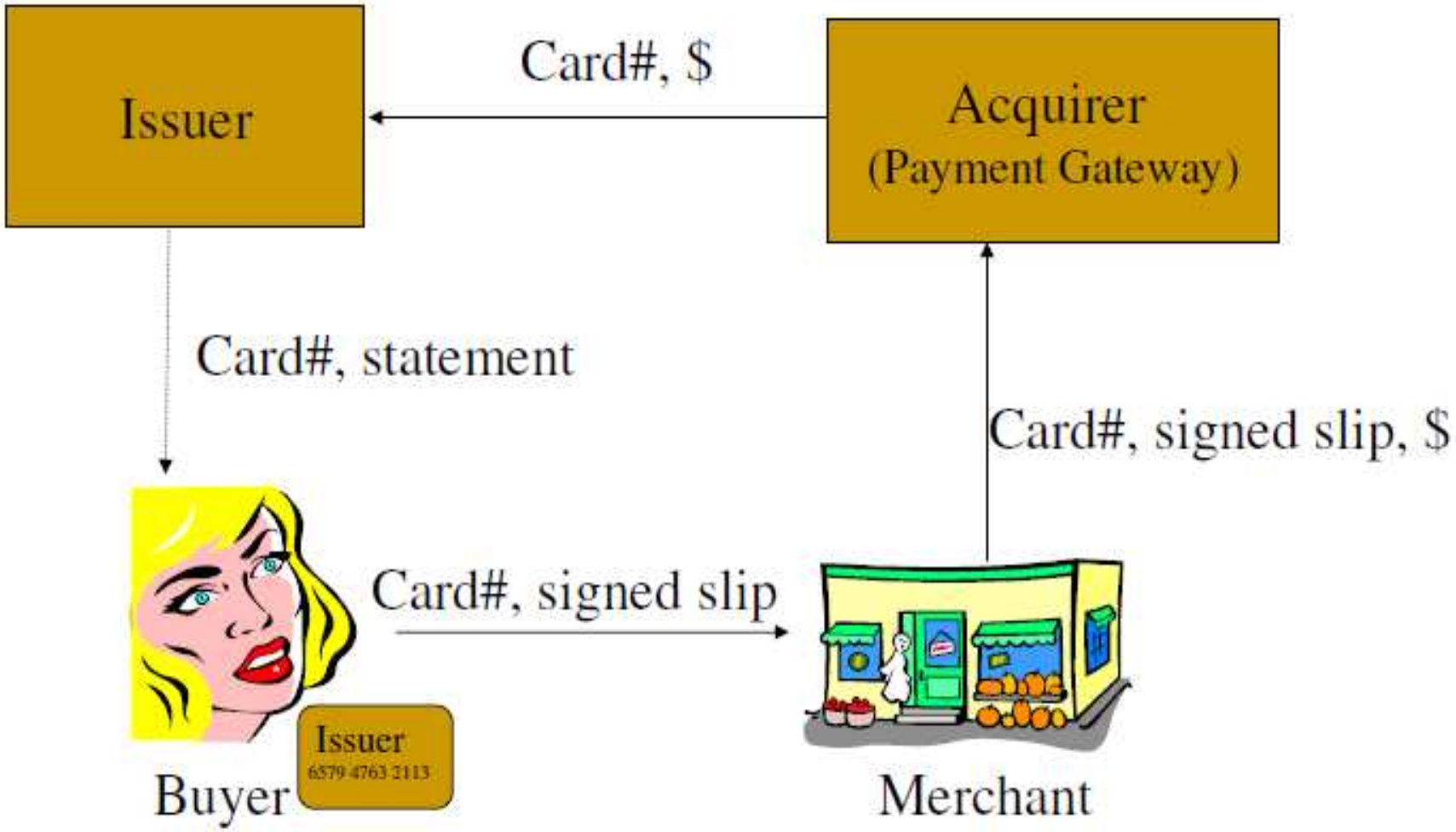
Done

Should be used,

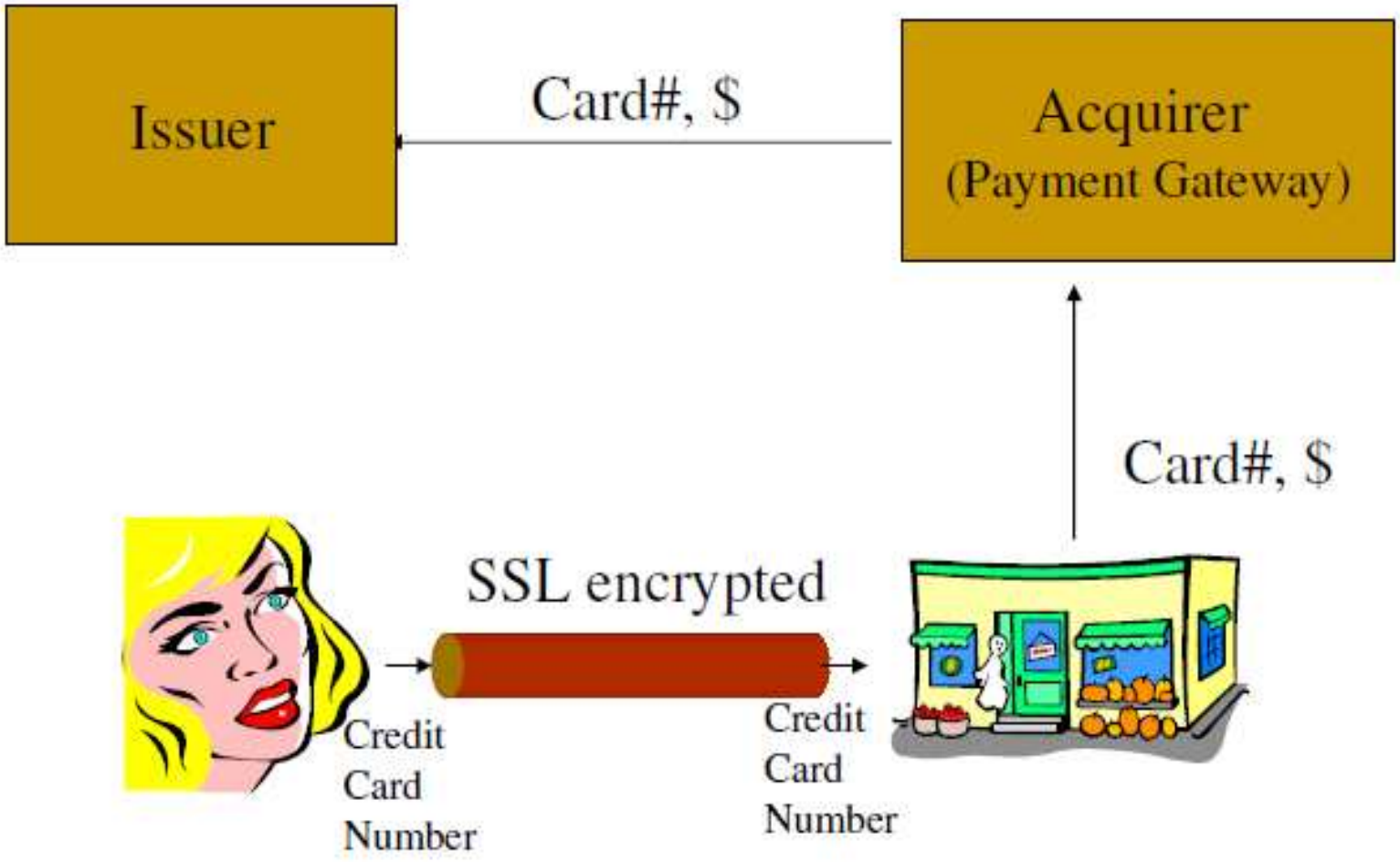
Done

Not in SSL

Credit Card Payments



SSL Credit Card Payments

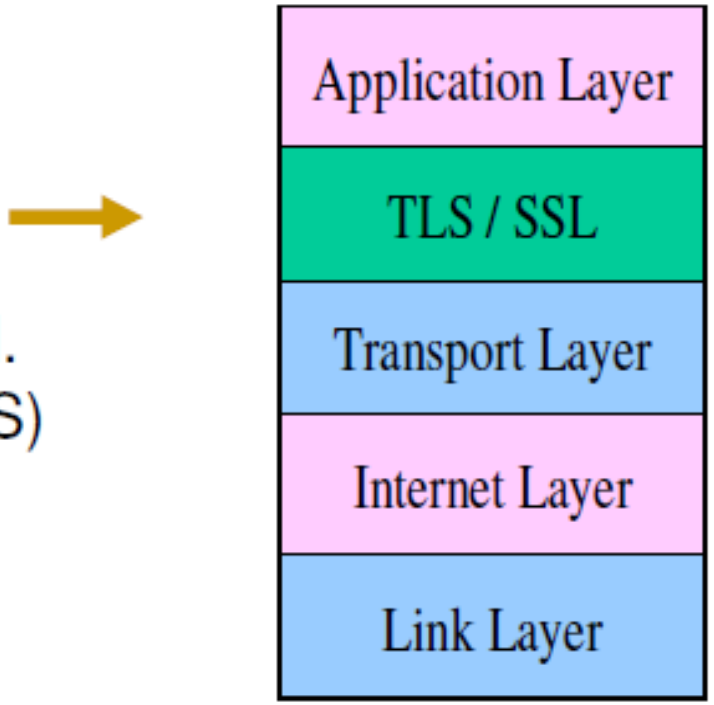


SSL Based Credit Card Payments

- Use SSL to securely transfer credit card numbers
- Trivial deployment (merchant decision).
- No client software required (SSL is in browser).
- Built on top of the existing credit card infrastructure.
- By far, the most widely used payment method.

Adding Security in Transport Layer (SSL / TLS)

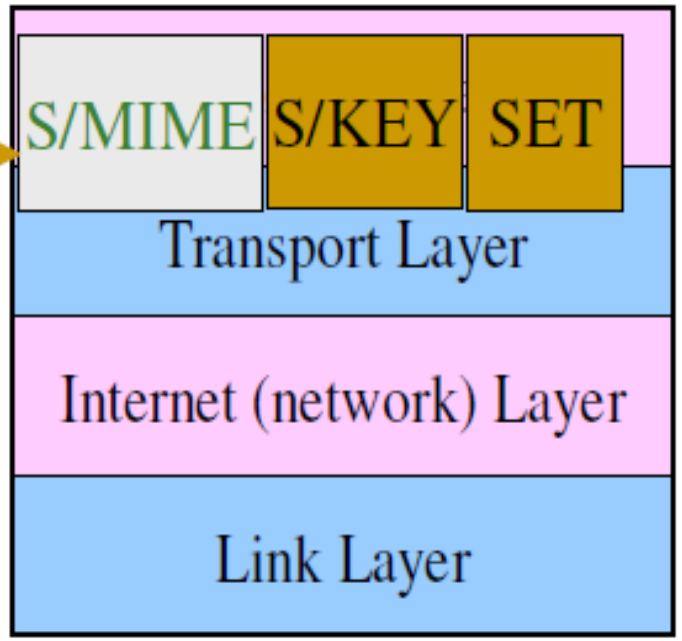
- SSL: Secure Socket Layer (Sockets is TCP/IP API)
- TLS: Transport Layer Security (IETF standard SSL)
  - When we say `SSL`, we refer also to TLS
- Pros:
  - Easy to implement and use
  - Deployed in most browsers, servers, …
- Cons:
  - Protects only if used by appl.
  - Vulnerable to Clogging (DOS)
    - Over TCP
  - Only end to end
  - Headers exposed



Adding Security

Alternative 1: Add to Each Application

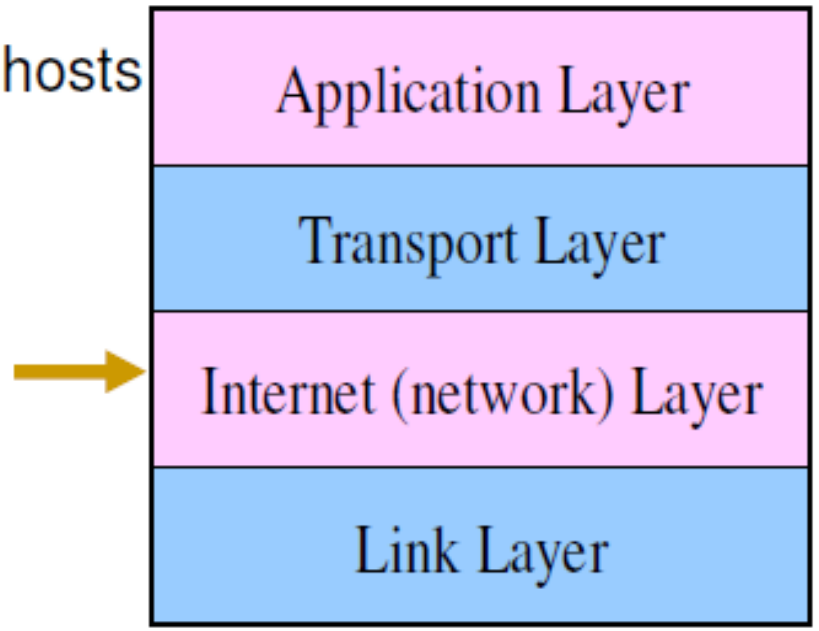
- Pros: easy, independent; awareness of semantics
- Cons:
  - Change each app, computer… hard, wasteful, error-prone, must trust all computers
  - No protection for headers
- Examples:
  - S/Key (login)
  - Payment protocols, e.g. SET (credit card payments)
  - Tools: XML security, Kerberos, …
  - Secure E-mail (S/MIME,PGP,…)



Adding Security
Alternative 2: IP Security

Pros:

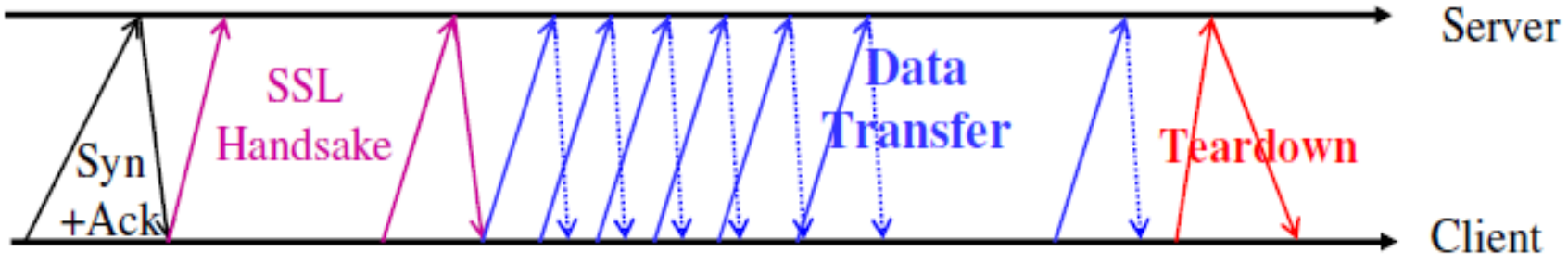
- Protect all applications, data (IP header, addresses)
- No change to applications
- Gateway can protect many hosts
- Anti-clogging mechanisms
- Implemented by operating systems, Routers, …
- Standard



- Cons:
  - Implementation, interoperability, availability
  - Application awareness/control is difficult

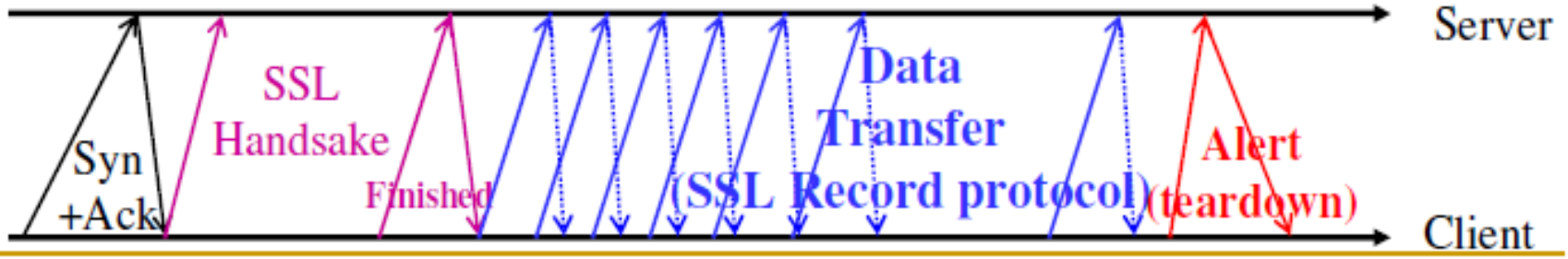
SSL Operation Phases (high level)

- TCP Connection
- Handshake
  - Negotiate (agree on) algorithms, methods
  - Authenticate server and optionally client
  - Establish keys
- Data transfer
- SSL Secure Teardown (why is this necessary?)



SSL Operation Phases

- Client uses SSL API to open connection
- SSL Handshake protocol:
  - For efficiency – resume `session` if possible
  - If not (session not kept, new connection, override)
    - Establish session - algorithms and master keys
  - Establish connection (keys, etc.)
- Data transfer (SSL Record protocol)
- Teardown – use Alert protocol:
  - By application closing connection
  - Or due to error (by handshake or record protocols)



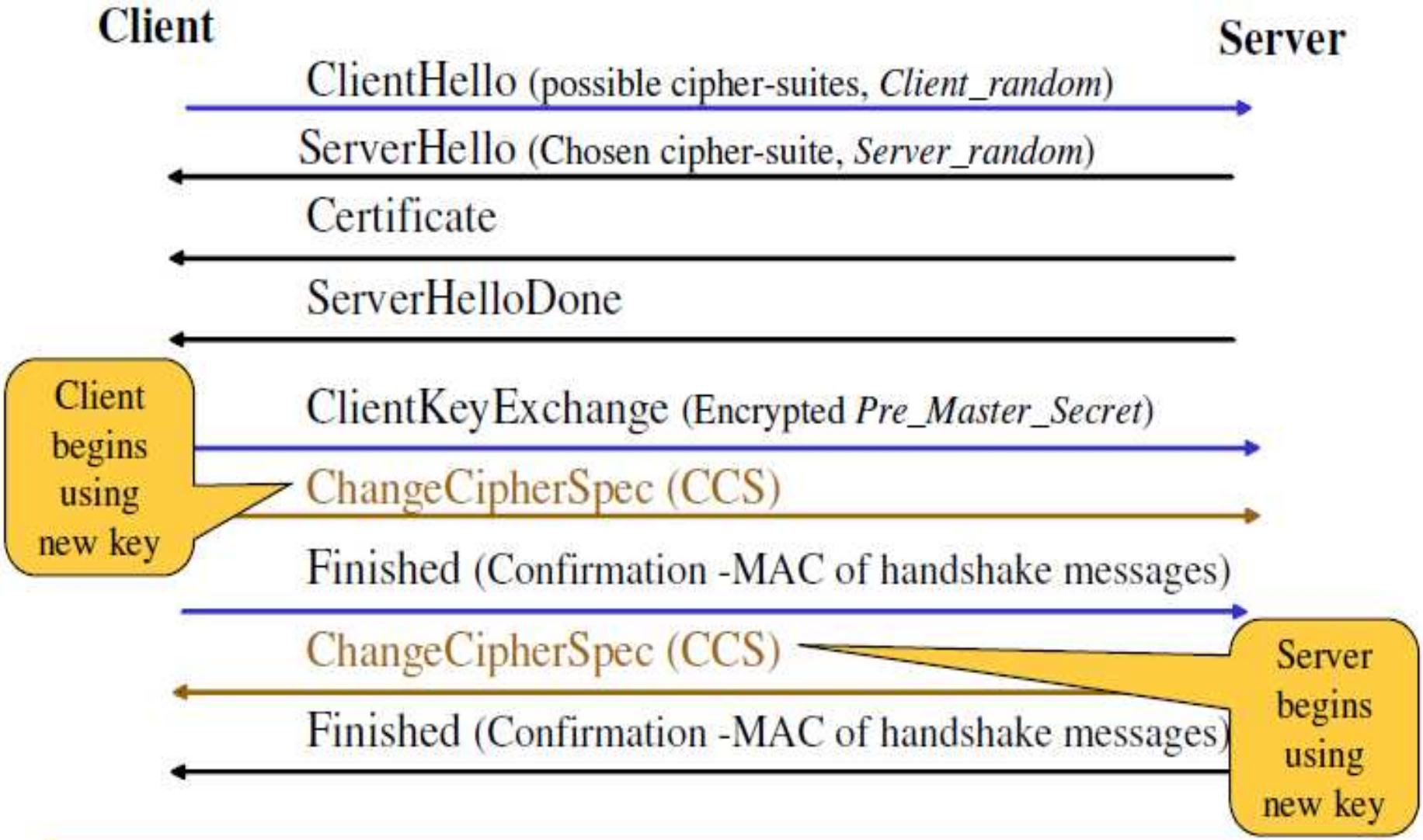
SSL Sessions and Connections

- Connection:
  - TCP/IP connection – send/receive secure messages
  - Reliable: ensures Delivery, Matching, FIFO
  - Independent, different keys for each connection
- SSL Session:
  - May span multiple connections for efficiency
  - Agree on algorithms and options
    - Client specifies possibilities, server chooses or rejects
  - Use public keys to Establish shared *MasterSecret* key
  - Server sets `session_id` so connection can resume (use existing session, for efficiency)
    - Client, server may discard session
    - Recommended (in RFC): keep session at most 24 hours

SSL Handshake Protocol

- Agree on *cipher suite*: algorithms and options:
  - Symmetric and Asymmetric Encryption
  - Signature and MAC
  - Compression
  - Options: client authentication, export (weak) versions,…
- Exchange random values
- Check for session resumption.
- Send certificate(s)
- Establish shared keys.
- Authenticate server
- Optionally authenticate client
- Confirm synchronization with peer

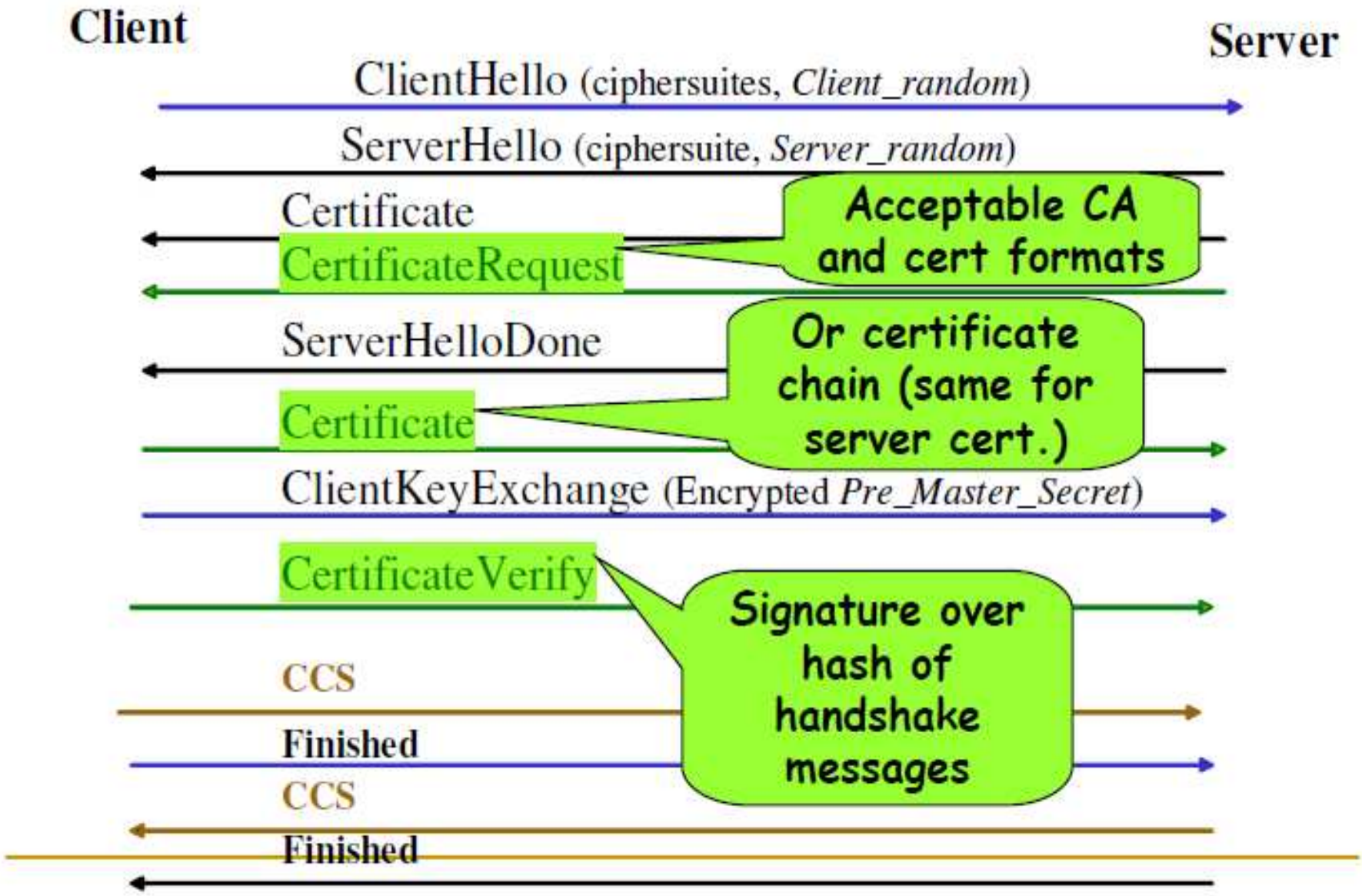
SSL Typical Handshake Messages



SSL Client Authentication

- Usually, only the server has a certificate
  - Client can authenticate the server
  - Client sends some identification info (e.g. username, password) to server using the SSL tunnel – after it is established
- SSL also supports authentication with client certificates
  - Server requires certificate from client
  - Server signals acceptable Certificate Authorities (CAs) and certificate formats, options etc.
  - Client returns appropriate certificate (chain)
  - Client authenticates by signing using certified public key
- Client authentication using certificates is used mostly within organizations, communities

Client Authentication Handshake



Crypto in SSL & TLS: Key Derivation

- Key derivation in SSL, TLS:
  - *Key block* (block of connection keys)
- Critical for security
- Design based on hash functions
  - Why not on block ciphers e.g. AES? Not available when SSL designed; DES was already too weak, no other standard and free cipher
- Which hash function to use?
  - Two main candidates: MD5 and SHA1
  - SSLv2: use MD5; SSLv3 and TLS: use both!
- How to use the hash functions?
  - Different design for TLS and SSL
  - SSL design: intuitive
  - TLS design: Cryptanalysis-tolerant PRF

SSL Services

- Server Authentication (mandatory)
- Client Authentication (optional - if required by server)
- Secure connection:
  - Confidentiality (Encryption) – optional, possibly weak (export)
  - Message Authentication
  - Reliability: prevent re-ordering, truncating etc.
- Efficiency: allow resumption of SSL session in new connection (no need to re-do handshake)

Denial of Service Attacks on SSL

- SSL and TLS are vulnerable to DOS attacks:
  - Operate over TCP → subject to DOS attacks on TCP, e.g. SYN-flooding
  - Record protocols aborts connection on any abnormality – easy to `kill` connections
  - Server decrypts *PreMasterKey* from client → clog server by sending many requests (minimal work for attacker!)
- Solutions (not in standard SSL/TLS!):
  - Server encrypts Pre-Master-Secret using Client's public key (this reduces server's work)
  - Or: when server is under attack (or high load), clients must `solve a puzzle` before server decrypts *PreMasterKey*
  - Server sends puzzle, e.g. find x given $h(x)$, $x[1\ldots80]$.

SSL Payments - Problems

- Hackers steal credit card info from Merchant sites
- Hackers use merchants to test guesses of card#s
- Hackers use site-spoofing to collect cards
  - Spoofing
  - Fake, unauthorized merchant site to collect card #s
- Bad merchants abuse – double/incorrect charges, expose
- Customers abuse their right to dispute (no signature)
- Very high dispute rates
  - Especially for online content and services
- Very expensive
  - Banks and credit associations – dispute resolution costs (~50$)
  - Merchants – high fees, liability for refunds;

Non-Repudiation for e-Commerce

- Normal contracts, checks, obligations are physically signed for *non-repudiation*
- Resolves dispute on authorization, approval
- How to ensure non-repudiation for e-commerce?
  - Digital signature by customer
  - Not done by SSL/TLS
    - Client authentication signature is not over transactions
    - Better done by application (which understand semantics)
  - Some products sign (even in some browsers) but… can't really trust PC environment – virus could sign!
  - Solution: signature feature for mobile phone
  - Existing spec signs simple text, not widely used
  - See CACM, May 03 for flexible signing by mobile…

Conclusion

- SSL / TLS is the most widely deployed security protocol, standard
  - Easy to implement, deploy and use; widely available
  - Flexible, supports many scenarios and policies
  - Mature cryptographic design
- But SSL is not always the best tool…
  - Use IP-Sec e.g. for anti-clogging, broader protection
  - Use application security, e.g. s/mime, for non-repudiation, store-and-forward communication (not online)
- Beware of host-spoofing and web-spoofing
  - Many browsers allow hard-to-detect spoofing
  - Many users will not detect simple spoofing (similar URL)