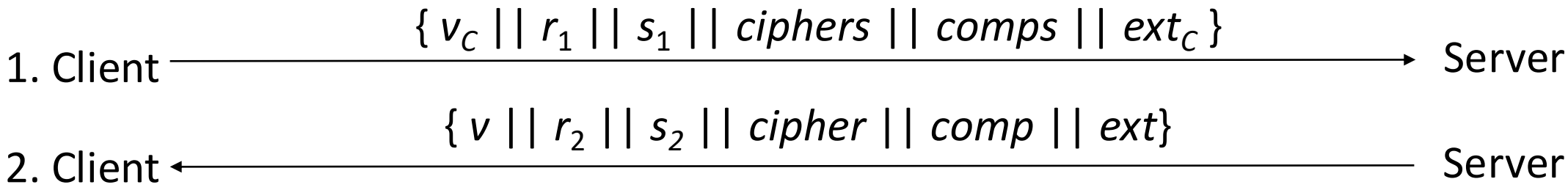
Lecture 16, May 4, 2026

Overview of Rounds

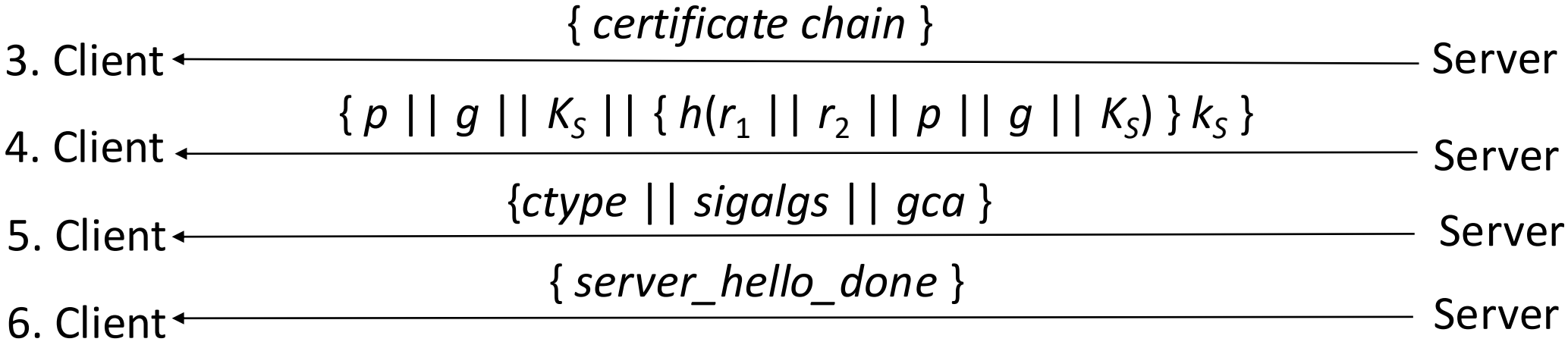
1. Create TLS connection between client, server
2. Server authenticates itself
3. Client validates server, begins key exchange
4. Acknowledgments all around

Handshake Round 1



v_C	Client's version of TLS
v	Highest version of TLS that client, server both understand
r_1, r_2	nonces (timestamp and 28 random bytes)
s_1	Current session id (empty if new session)
s_2	Current session id (if s_1 empty, new session id)
ciphers	Ciphers that client understands
comps	Compression algorithms that client understand
cipher	Cipher to be used
comp	Compression algorithm to be used
ext_C	List of extensions client supports
ext	List of extensions server supports (subset of ext_C)

Handshake Round 2



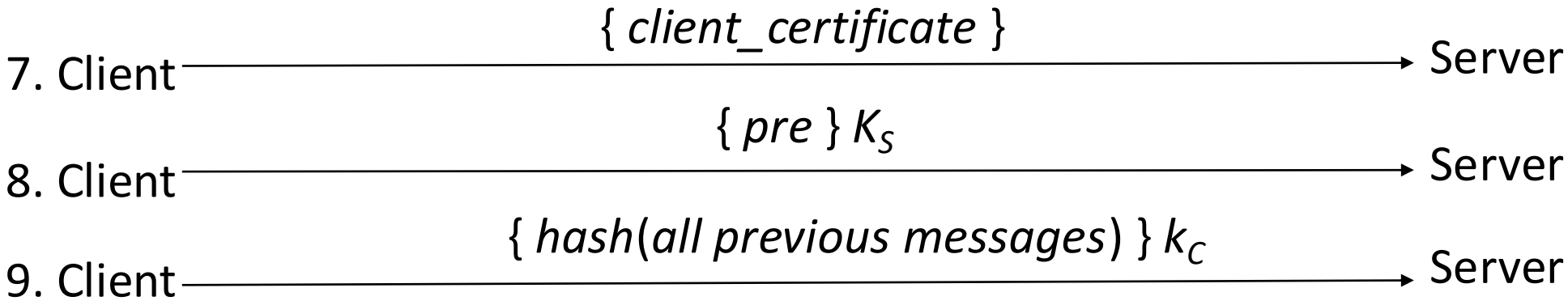
If server not going to authenticate itself, only last message sent

Second step is for Diffie-Hellman with RSA certificate

Third step omitted if server does not need client certificate

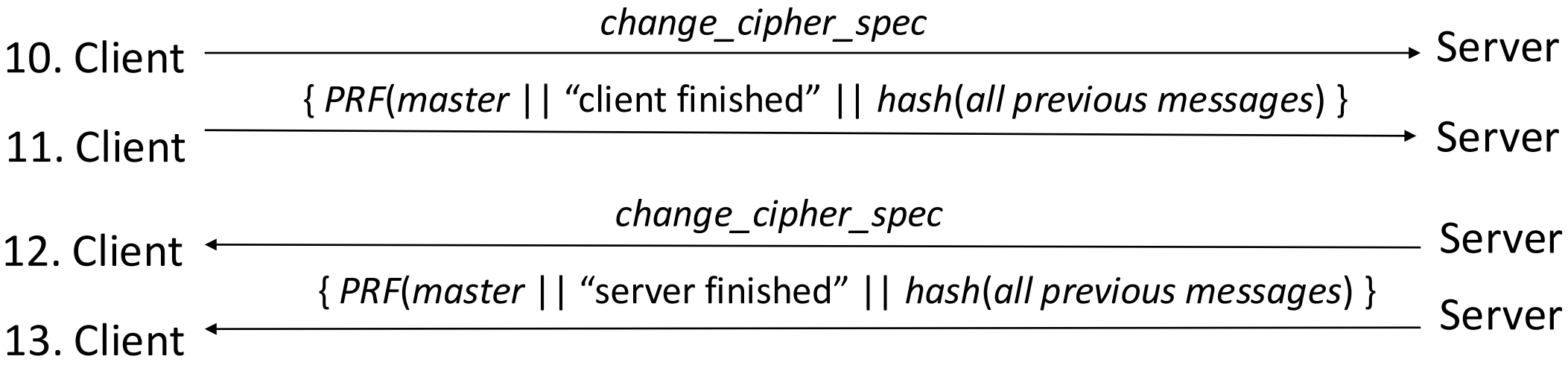
K_S, k_S	Server's Diffie-Hellman public, private keys
ctype	Certificate type accepted (by cryptosystem)
sigalgs	List of hash, signature algorithm pairs server can use
gca	Acceptable certification authorities

Handshake Round 3



pre	Premaster secret
K_S	Server's public key
k_C	Client's private key

Handshake Round 4



change_cipher_spec Begin using cipher specified

TLS Change Cipher Spec Protocol

- Send single byte
- In handshake, new parameters considered “pending” until this byte received
  - Old parameters in use, so cannot just switch to new ones

TLS Alert Protocol

- Closure alert
  - Sender will send no more messages
  - Pending data delivered; new messages ignored
- Error alerts
  - Warning: connection remains open
  - Fatal error: connection torn down as soon as sent or received

TLS Heartbeat Extension

- Message has 4 fields
  - Value indicating message is request
  - Length of data in message
  - Data of given length
  - Random data
- Message sent to peer; peer replies with similar message
  - If second field is too large (> 214 bytes), ignore message
  - Reply message has same data peer sent, new random data
- When peer sends this for the first time, it sends nothing more until a response is received

TLS Application Data Protocol

- Passes data from application to TLS Record Protocol layer

Differences Between TLSv2 and SSLv3

- SSLv3 master secret computed differently

  master = MD5(*premaster* || SHA('A' || *premaster* || r_1 || r_2) ||
  MD5(*premaster* || SHA('BB' || *premaster* || r_1 || r_2) ||
  MD5(*premaster* || SHA('CCC' || *premaster* || r_1 || r_2)

- SSLv3 key block also computed differently

  key_block = MD5(*master* || SHA('A' || *master* || r_1 || r_2) ||
  MD5(*master* || SHA('BB' || *master* || r_1 || r_2) ||
  MD5(*master* || SHA('CCC' || *master* || r_1 || r_2) || . . .

Differences Between TLSv2 and SSLv3

SSLv3 MAC for each block computed differently:

hash(*MAC_ws* || *opad* ||

hash(*MAC_ws* || *ipad* || *seq* || *SSL_comp* || *SSL_len* || *block*))

- *hash*: hash function used
- *MAC__ws*, *seq*, *SSL_comp*, *SSL_len*, *block*: as for TLS (with obvious changes)
- *ipad, opad*: as for HMAC

Differences Between TLSv2 and SSLv3

- Verification message (9, above) is different:

9’. Client —{ *hash*(*master* || *opad* || *hash*(*all previous messages* || *master* || *ipad*)) }→ Server

- Messages after change cipher spec (11, 13 above) are also different:

11’. Client —{ *hash*(*master* || *opad* || *hash*(*all previous messages* || 0x434C4E54 || *master* || *ipad*)) }→ Server

13’. Client —{ *hash*(*master* || *opad* || *hash*(*all previous messages* || 0x53525652 || *master* || *ipad*)) }→ Server

Differences Between TLSv2 and SSLv3

- Different sets of ciphers
  - SSL allows use of RC4, but its use is deprecated
  - SSL allows set of ciphers for the Fortezza cryptographic token used by the U.S. Department of Defense

Problems with SSL

- POODLE attack focuses on padding of messages
  - In SSL, all but the last byte of the padding are random and so cannot be checked
- How padding works (assume block size of *b*):
  - Message ends in a full block: add additional block of padding, and last byte is the number of bytes of random padding (*b* – 1)
  - Message ends in part of a block: add random bytes out to last byte, set that to number of random bytes (so if block is *b* – 1 bytes, one padding byte added and it is 0)

The POODLE Attack

- Peer receives incoming ciphertext message c_1, …, c_n
- Peer decrypts it to m_1, …, m_n: $m_i = D_k(c_i) \oplus c_{i-1}$, where c_0 is initialization vector
  - Validates by removing padding, computes and checks MAC over remaining bytes
- Attacker replaces c_n with some earlier block, say c_j, $j \neq n$
  - If last byte of c_j is same as c_n, message accepted as valid; otherwise, rejected
- So attacker arranges for HTTP messages to end with known number of padding bytes
  - Then server should accept changed message in at least 1 out of 256 tries

Example POODLE Attack

- Here's HTTP request (somewhat simplified):

GET / HT TP/1.1\r\n Cookie: abcdefgh \r\n\r\nxxxx MAC •••••••7

- Attacker cannot see plaintext
- Run Javascript in browser that duplicates cookie block and overwrites last block
  - It's enciphered using (for example) 3DES-CBC
- You see enciphered block
  - If it is accepted, then plaintext block xor'ed with previous ciphertext block ends in 7

SSL, TLS, and POODLE

- POODLE serious enough that SSL is strongly deprecated in favor of TLS
- TLS not vulnerable, as all padding bytes set to length of padding
  - And TLS implementations must check this padding (all of it) for validity before accepting messages

Authentication Basics

- Authentication: binding of identity to subject
  - Identity is that of external entity (my identity, Matt, *etc*.)
  - Subject is computer entity (process, *etc*.)

Establishing Identity

- One or more of the following
  - What entity knows (*eg.* password)
  - What entity has (*eg.* badge, smart card)
  - What entity is (*eg.* fingerprints, retinal characteristics)
  - Where entity is (*eg*. In front of a particular terminal)

Authentication System

- (*A*, *C*, *F*, *L*, *S*)
  - *A* information that proves identity
  - *C* information stored on computer and used to validate authentication information
  - *F* complementation function; for $f \in F$, $f : A \rightarrow C$
  - *L* functions that prove identity; for $l \in L$, $l : A \times C \rightarrow$ { true, false }
    - *l* is lowercase “L”
  - *S* functions enabling entity to create, alter information in *A* or *C*

Example

- Password system, with passwords stored on line in clear text
  - *A* set of strings making up passwords
  - *C* = *A*
  - *F* singleton set of identity function { *I* }
  - *L* single equality test function { *eq* }
  - *S* function to set/change password

Passwords

- Sequence of characters
  - Examples: 10 digits, a string of letters, *etc*.
  - Generated randomly, by user, by computer with user input
- Sequence of words
  - Examples: pass-phrases
- Algorithms
  - Examples: challenge-response, one-time passwords

Storage

- Store as cleartext
  - If password file compromised, *all* passwords revealed
- Encipher file
  - Need to have decipherment, encipherment keys in memory
  - Reduces to previous problem
- Store one-way hash of password
  - If file read, attacker must still guess passwords or invert the hash

Example

- UNIX system original hash function
  - Hashes password into 11 char string using one of 4096 hash functions
- As authentication system:
  - *A* = { strings of 8 chars or less }
  - *C* = { 2 char hash id || 11 char hash }
  - *F* = { 4096 versions of modified DES }
  - *L* = { *login*, *su*, … }
  - *S* = { *passwd*, *nispasswd*, *passwd+*, … }

Anatomy of Attacking

- Goal: find $a \in A$ such that:
  - For some $f \in F$, $f(a) = c \in C$
  - c is associated with entity
- Two ways to determine whether a meets these requirements:
  - Direct approach: as above
  - Indirect approach: as $l(a)$ succeeds iff $f(a) = c \in C$ for some c associated with an entity, compute $l(a)$

Preventing Attacks

- Hide one of *a*, *f*, or *c*
  - Prevents obvious attack from above
  - Example: UNIX/Linux shadow password files hides *c*'s
- Block access to all $l \in L$ or result of $l(a)$
  - Prevents attacker from knowing if guess succeeded
  - Example: preventing *any* logins to an account from a network
    - Prevents knowing results of *l* (or accessing *l*)

Approaches: Password Selection

- Random selection
  - Any password from *A* equally likely to be selected
- Pronounceable passwords
- User selection of passwords

Random Passwords

- Choose characters randomly from a set of possible characters; may also choose length randomly from a set of possible lengths
- Expected time to guess password maximized when selection of characters in the set, lengths in the set, are equiprobable
- In practice, several factors to be considered:
  - If password too short, likely to be guessed
  - Some other classes of passwords need to be eliminated, such as repeated patterns (“aaaaa”), known patterns (“qwerty”)
  - But if too much is excluded, space of possible passwords becomes small enough to search exhaustively

Generating Random Passwords

- Random (pseudorandom) number generator period critical!
- Example: PDP-11 randomly generated passwords of length 8, and composed of capital letters and digits
  - Number of possible passwords = $(26 + 10)^8 = 36^8 = 2.8 \times 10^{12}$
  - Took 0.00156 to test a password, so would take about 140 years to try all
- Attacker noticed the pseudorandom number generator on PDP-11, with word size of 16 bits, had period of $2^{16} – 1$
  - Number of possible passwords = $2^{16} – 1 = 65{,}535 = 6.5 \times 10^4$
  - Took 0.00156 to test a password, so would take about 102 seconds to try all
- When launched, found all passwords in under 41 seconds

Remembering Random Passwords

- Humans can repeat with perfect accuracy 8 meaningful items
  - Like digits, letters, words
- Write them down
  - Put them in a place where others are unlikely to get to them
  - Purse or wallet is good; keyboard or monitor is not
- Write obscured versions of passwords
  - Let $p \in P$ be password; choose invertible transformation algorithm t: $P \rightarrow A$
  - Write down $t^{-1}(p)$ but not t
  - Now user must memorize t, not each individual password
- Use a password manager (password wallet)
  - Now must remember password to unlock the other passwords