INTERNET OF THINGS

Lectures

School of Instrumentation,
DAVV Indore

IoT Protocols

- **Transport Layer Protocols**
  - **TCP**
  - **MPTCP**
  - **TLS/SSL**
  - **UDP**
  - DCCP
  - SCTP
  - DTLS

School of Instrumentation, DAVV Indore

TCP (Transmission Control Protocol)

- TCP was specifically designed to **provide a reliable end-to-end byte stream** over an unreliable internetwork.
- TCP was formally defined in RFC 793 in September 1981.
- Each machine supporting TCP has a TCP transport entity, either a library procedure, a user process, or most commonly part of the kernel. In all cases, it manages TCP streams and interfaces to the IP layer.
- The IP layer gives no guarantee that datagrams will be delivered properly, nor any indication of how fast datagrams may be sent. It is up to TCP to send datagrams fast enough to make use of the capacity but not cause congestion, and to time out and retransmit any datagrams that are not delivered.

School of Instrumentation, DAVV Indore

TCP Service Model

- TCP service is obtained by both the sender and the receiver creating end points, called sockets.
- Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a port.

- Port numbers below 1024 are reserved for standard services that can usually only be started by privileged users. They are called well-known ports.
- Other ports from 1024 through 49151 can be registered with IANA for use by unprivileged users, but applications can and do choose their own ports.
- For TCP service to be obtained, a connection must be explicitly established between a socket on one machine and a socket on another machine.
- A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end.

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

TCP Protocol

- A key feature of TCP, and one that dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number.
- Separate 32-bit sequence numbers are carried on packets for the sliding window position in one direction and for acknowledgements in the reverse direction.
- The sending and receiving TCP entities exchange data in the form of segments.
- A TCP segment consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes.
- It can accumulate data from several writes into one segment or can split data from one write over multiple segments.
- Two limits restrict the segment size.
  - First, each segment, including the TCP header, must fit in the 65,515-byte IP payload.
  - Second, each link has an MTU (Maximum Transfer Unit). Each segment must fit in the MTU at the sender and receiver so that it can be sent and received in a single, unfragmented packet. In practice, the MTU is generally 1500 bytes (the Ethernet payload size) and thus defines the upper bound on segment size.

School of Instrumentation, DAVV Indore

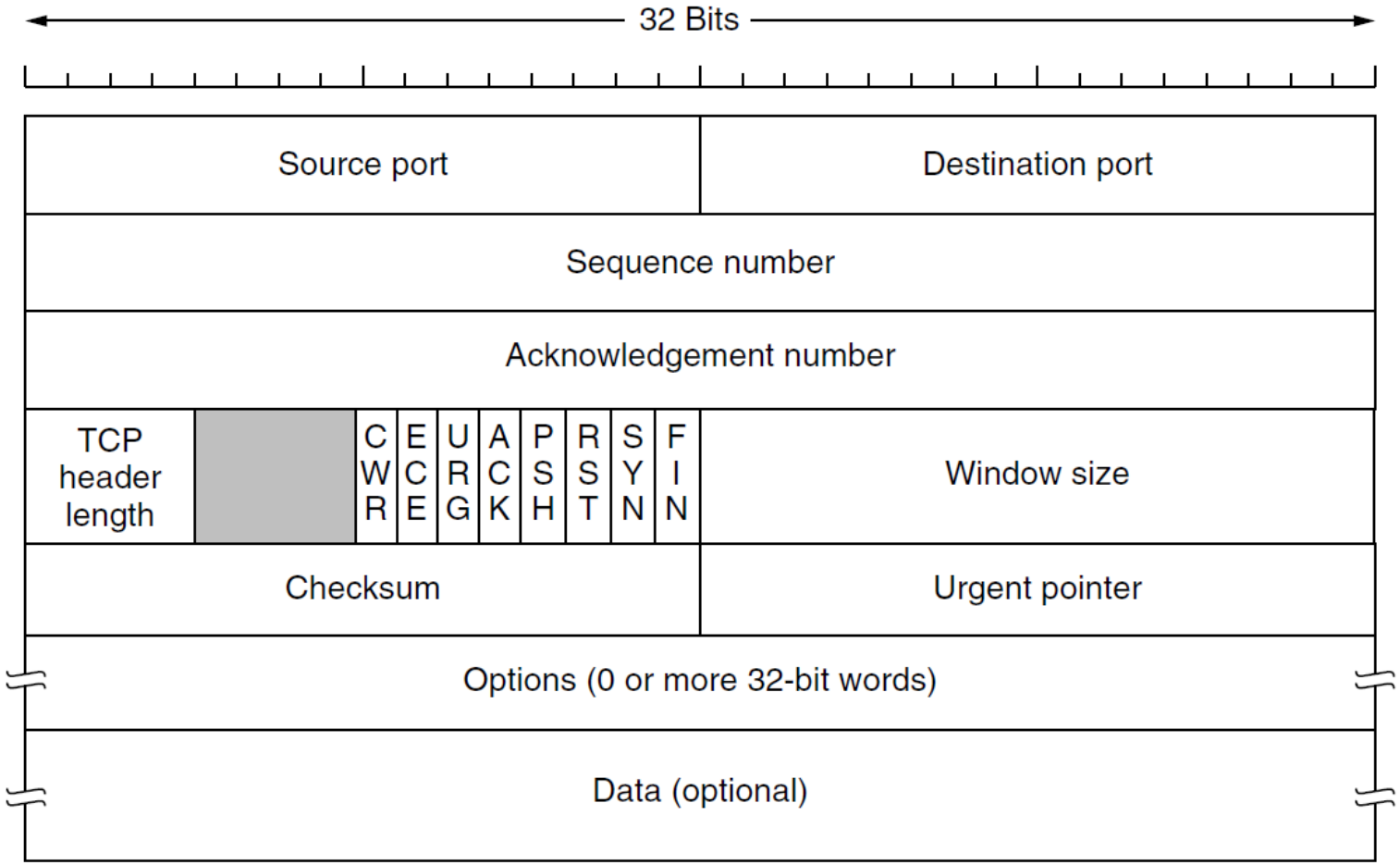
TCP Protocol

- The basic protocol used by TCP entities is the sliding window protocol with a dynamic window size. When a sender transmits a segment, it also starts a timer.
- When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, and otherwise without) bearing an acknowledgement number equal to the next sequence number it expects to receive and the remaining window size.
- If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.
- TCP must be prepared to deal with below mentioned problems and solve them in an efficient way.
  - Segments can arrive out of order, so bytes 3072–4095 can arrive but cannot be acknowledged because bytes 2048–3071 have not turned up yet.
  - Segments can also be delayed so long in transit that the sender times out and retransmits them.
  - The retransmissions may include different byte ranges than the original transmission, requiring careful administration to keep track of which bytes have been correctly received so far.

School of Instrumentation DAVV Indore

The TCP Segment Header

- Every segment begins with a fixed-format, 20-byte header. The fixed header may be followed by header options.
- After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow.
- Segments without any data are legal and are commonly used for acknowledgements and control messages.



The TCP Segment Header

- The Source port and Destination port fields identify the local end points of the connection.
- A TCP port plus its host's IP address forms a 48-bit unique end point.
- The source and destination end points together identify the connection.
- This connection identifier is called a 5 tuple because it consists of five pieces of information: the protocol (TCP), source IP and source port, and destination IP and destination port.
- Every byte on a TCP connection has its own **32-bit sequence number** because every byte of data is numbered in a TCP stream.
- During connection establishment, each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction. Sequence number should be started at random, to remove duplication problem.
- The sequence number of any other segment is the sequence number of the previous segment plus the number of bytes (real or imaginary) carried by the previous segment.
- Acknowledgement number field specifies the next in-order byte expected, not the last byte correctly received.
- It is a cumulative acknowledgement because it summarizes the received data with a single number.
- The TCP header length tells how many 32-bit words are contained in the TCP header.
- Next comes a 4-bit field that is not used.
- *CWR* and *ECE* are used to signal congestion when ECN (Explicit Congestion Notification) is used.
- ECE is set to signal an ECN-Echo to a TCP sender to tell it to slow down when the TCP receiver gets a congestion indication from the network.
- CWR is set to signal Congestion Window Reduced from the TCP sender to the TCP receiver so that it knows the sender has slowed down and can stop sending the ECN-Echo.

School of Instrumentation, DAVV Indore

The TCP Segment Header

- When an application has high priority data that should be processed immediately, for example, the sending application can put some control information in the data stream and give it to TCP along with the URGENT flag.
- This event causes TCP to stop accumulating data and transmit everything it has for that connection immediately.

- URG is set to 1 if the Urgent pointer is in use. The Urgent pointer is used to indicate a byte offset from the current sequence number at which urgent data are to be found.

School of Documentation, DAVV

The TCP Segment Header

- When an application passes data to TCP, TCP may send it immediately or buffer it (in order to collect a larger amount to send at once), at its discretion.
- However, sometimes the application really wants the data to be sent immediately.
- For example, suppose a user of an interactive game wants to send a stream of updates.
- It is essential that the updates be sent immediately, not buffered until there is a collection of them.
- To force data out, TCP has the notion of a PUSH flag that is carried on packets.
- The original intent was to let applications tell TCP implementations via the PUSH flag not to delay the transmission.

The PSH bit indicates PUSHed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received (which it might otherwise do for efficiency).

The TCP Segment Header

- The ACK bit is set to 1 to indicate that the Acknowledgement number is valid.
- If ACK is 0, the segment does not contain an acknowledgement, so the Acknowledgement number field is ignored.
- The RST bit is used to abruptly reset a connection that has become confused due to a host crash or some other reason.
- The SYN bit is used to establish connections. The connection request has SYN =1 and ACK =0 to indicate that the piggyback acknowledgement field is not in use.
- The connection reply does bear an acknowledgement, however, so it has SYN = 1 and ACK=1.
- In essence, the SYN bit is used to denote both CONNECTION REQUEST and CONNECTION ACCEPTED, with the ACK bit used to distinguish between those two possibilities.
- The FIN bit is used to release a connection. It specifies that the sender has no more data to transmit.
- However, after closing a connection, the closing process may continue to receive data indefinitely. Both SYN and FIN segments have sequence numbers and are thus guaranteed to be processed in the correct order.
- Flow control in TCP is handled using a variable-sized sliding window. The Window size field tells how many bytes may be sent starting at the byte acknowledged. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

School of Implementation, DAV Nagpur

The TCP Segment Header

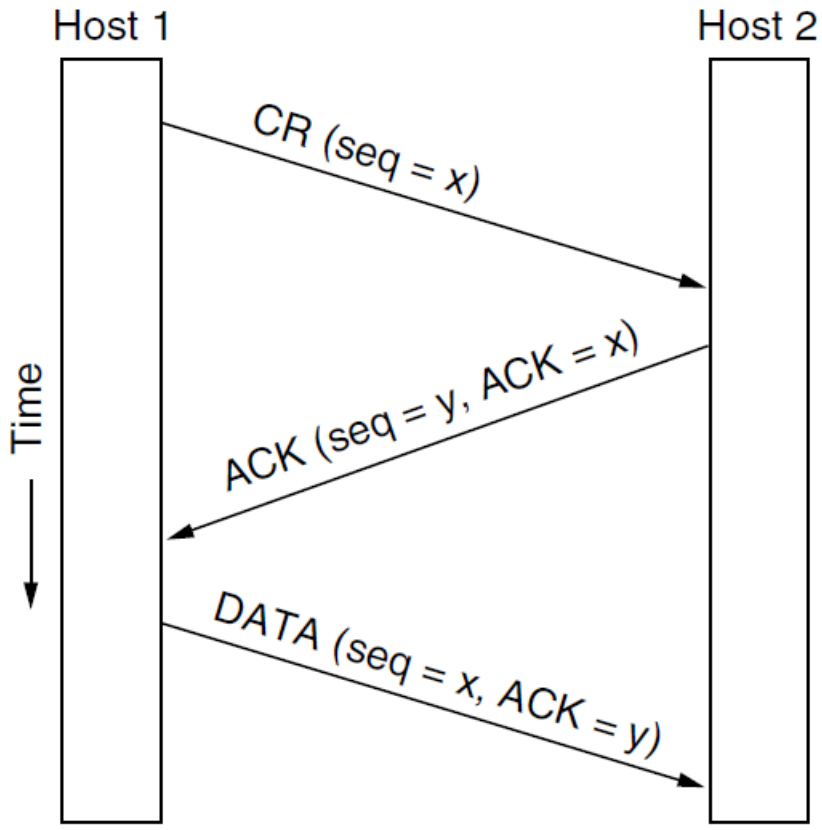
- A Checksum is also provided for extra reliability. It checksums the header, the data, and a conceptual pseudo header. The checksum is mandatory unlike UDP.
- The Options field provides a way to add extra facilities not covered by the regular header.
- Many options have been defined and several are commonly used.
- The options are of variable length, fill a multiple of 32 bits by using padding with zeros, and may extend to 40 bytes to accommodate the longest TCP header that can be specified.
- A widely used option is the one that allows each host to specify the MSS (Maximum Segment Size) it is willing to accept. Using large segments is more efficient than using small ones because the 20-byte header can be amortized over more data, but small hosts may not be able to handle big segments.

School of Information,
DAVV Indore

TCP Connection Establishment

Connections are established in TCP by means of the three-way handshake.

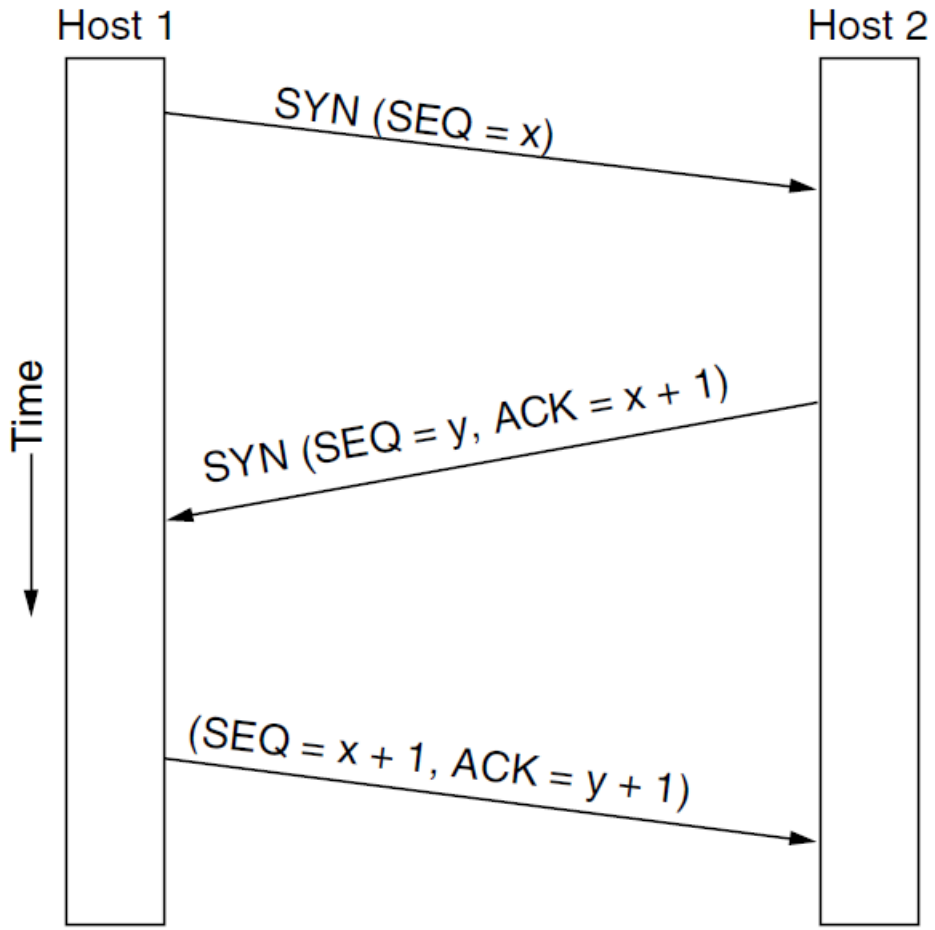
- The normal setup procedure when host 1 initiates is shown in Figure.
- Host 1 chooses a sequence number, *x*, and sends a CONNECTION REQUEST (CR) segment containing it to host 2.
- Host 2 replies with an ACK segment acknowledging *x* and announcing its own initial sequence number, *y*.
- Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data segment that it sends.



School of Instrumentation, DAVV Indore

- To establish a connection, one side, say, the server, passively waits for an incoming connection by executing the and LISTEN ACCEPT primitives in that order, either specifying a specific source or nobody in particular.

- The other side, say, the client, executes a primitive, specifying the CONNECT IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password).
- The CONNECT primitive sends a TCP segment with the SYN bit on and ACK bit off and waits for a response.
- When this segment arrives at the destination, the TCP entity there checks to see if there is a process that has done a LISTEN on the port given in the Destination port field.
- If not, it sends a reply with the RST bit on to reject the connection.
- If some process is listening to the port, that process is given the incoming TCP segment.
- It can either accept or reject the connection. If it accepts, an acknowledgement segment is sent back.
- The sequence of TCP segments sent in the normal case is shown in Figure.



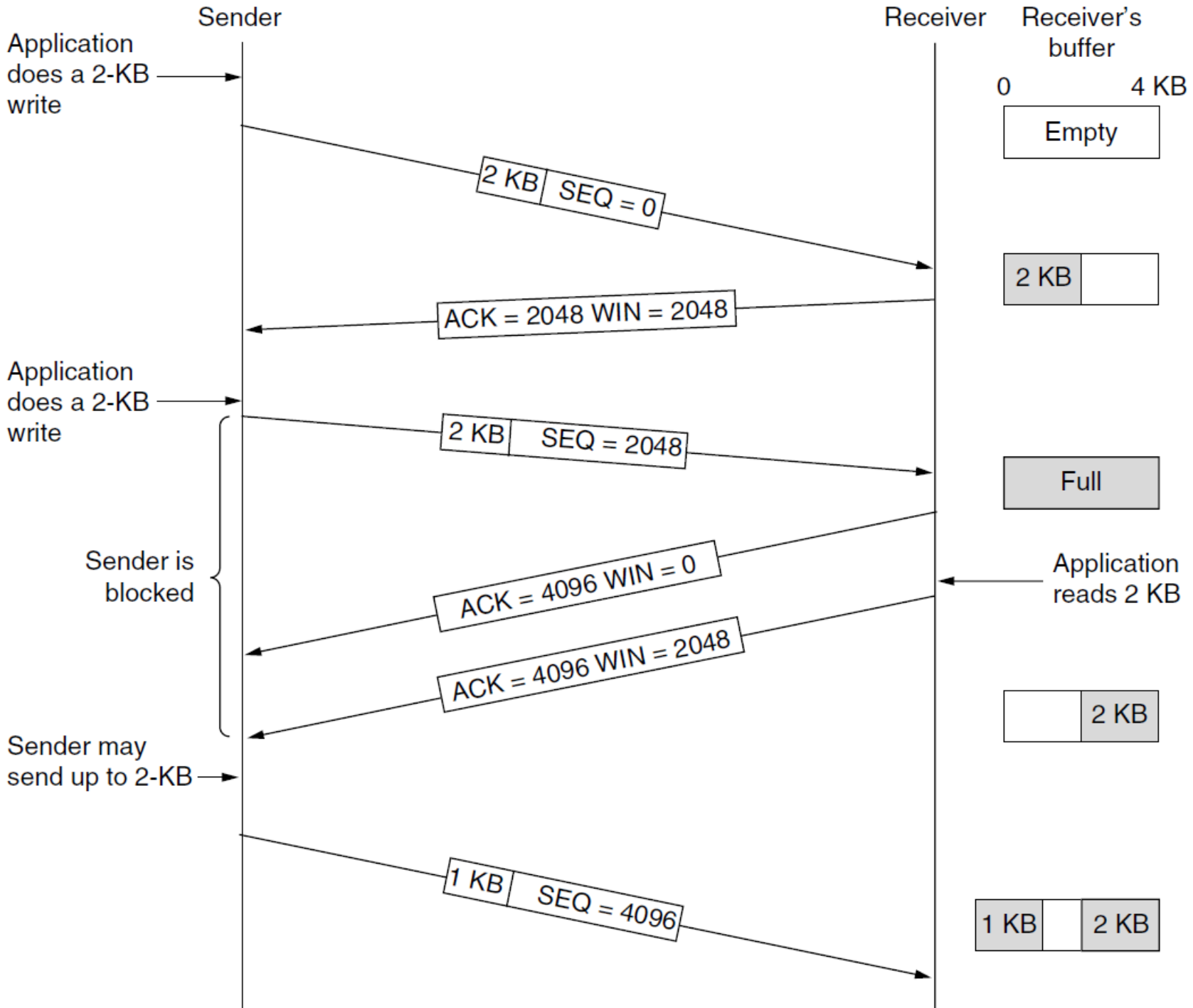
TCP Connection Release

- Although TCP connections are full duplex, to understand how connections are released it is best to think of them as a pair of simplex connections.
- Each simplex connection is released independently of its sibling.
- To release a connection, either party can send a TCP segment with the FIN bit set, which means that it has no more data to transmit.
- When the FIN is acknowledged, that direction is shut down for new data.
- Data may continue to flow indefinitely in the other direction, however when both directions have been shut down, the connection is released.
- Normally, four TCP segments are needed to release a connection: one FIN and one ACK for each direction.
- However, it is possible for the first ACK and the second FIN to be contained in the same segment, reducing the total count to three.

School of Instrumentation, DAVV Indore

TCP Sliding Window

- For example, the receiver has a 4096-byte buffer, and sender transmits a 2048-byte segment.



MPTCP (Multipath TCP)

Limitations of TCP:

- TCP connections cannot move from one IP address to another.
- Example: When a laptop switches from Ethernet to Wi-Fi it obtains another IP address. All existing TCP connections must be torn down and new connections restarted.

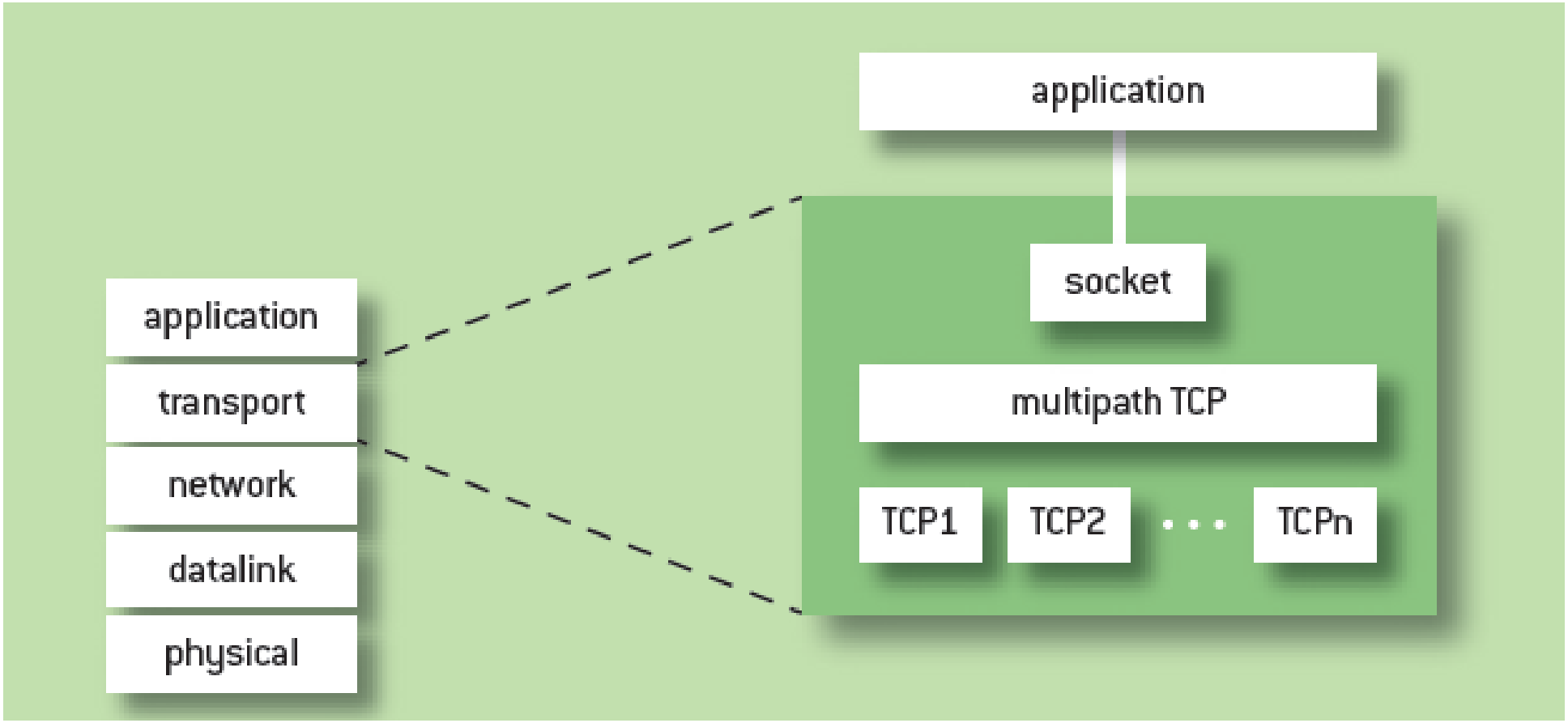
MPTCP (Multipath TCP) is a potential solution to address this problems.

More specifically, the design goals for MPTCP are:

- It should be capable of using multiple network paths for a single connection.
- It must be able to use the available network paths at least as well as regular TCP, but without starving TCP.
- It must be as usable as regular TCP for existing applications.
- Enabling MPTCP must not prevent connectivity on a path where regular TCP works.

School of Instrumentation, DAVV Indore

Multipath TCP in the Stack



THE ARCHITECTURAL PRINCIPLES

- Applications interact through the regular socket API, and MPTCP manages the underlying TCP connections (called **subflows**) that are used to carry the actual data.

- From an architectural viewpoint, MPTCP acts as a shim layer between the socket interface and one or more TCP subflows.

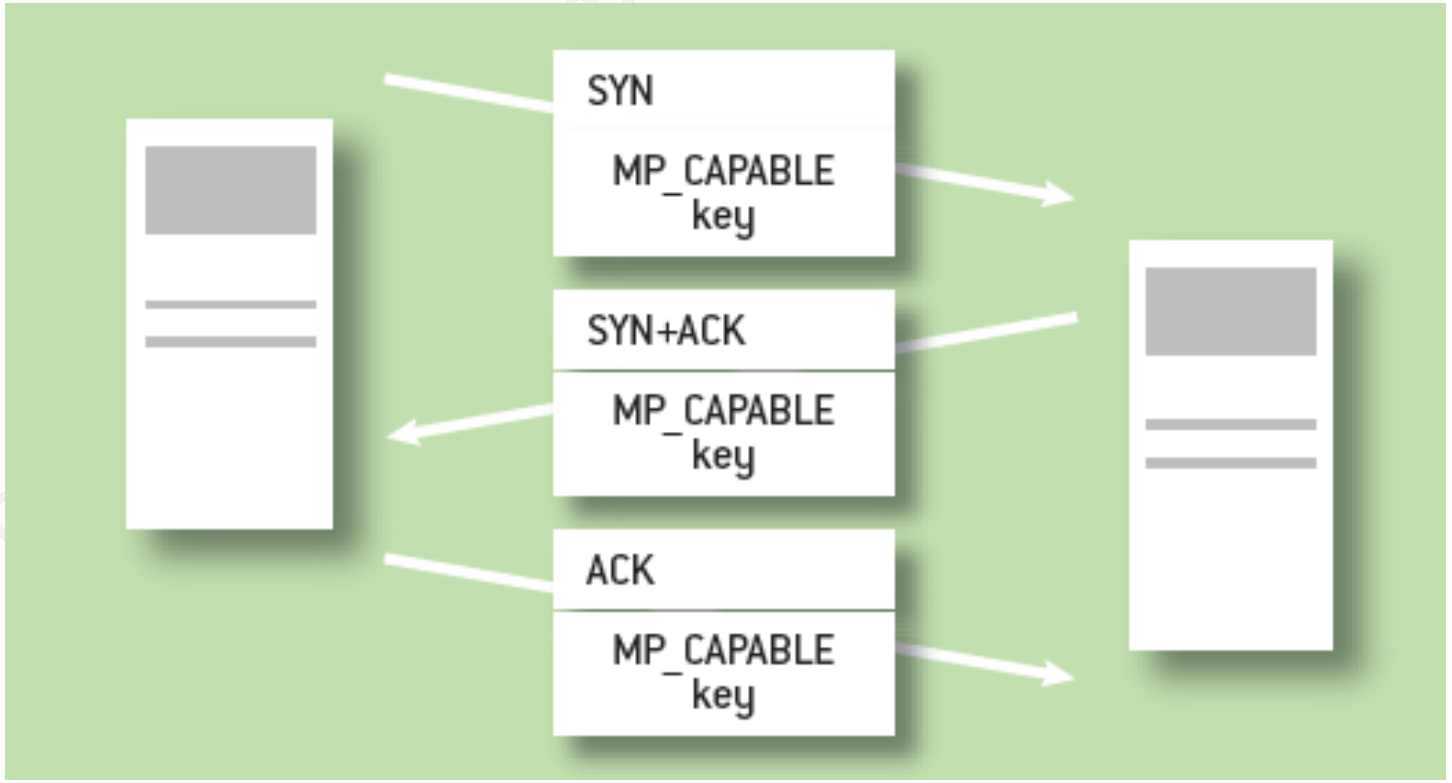
- MPTCP requires additional signaling between the end hosts. It achieves this by **using TCP options** to achieve the following goals:
  - Establish a new MPTCP connection.
  - Add subflows to an MPTCP connection.
  - Transmit data on the MPTCP connection.

School of Instrumentation
DAVV Indore

Connection Establishment

- An MPTCP connection is established by using the three-way handshake with TCP options to negotiate its usage.

- The **MP_CAPABLE** option in the **SYN segment** indicates that the client supports MPTCP. This option also contains a random key used for security purposes.
- If the server supports MPTCP, then it replies with a SYN+ACK segment that also contains the MP_CAPABLE option. This option contains a random key chosen by the server.
- The third ACK of the three-way handshake also includes the MP_CAPABLE option to confirm the utilization of MPTCP and the keys.



The three-way handshake shown in the figure creates the first TCP subflow over one interface.

Add subflows to an MPTCP connection

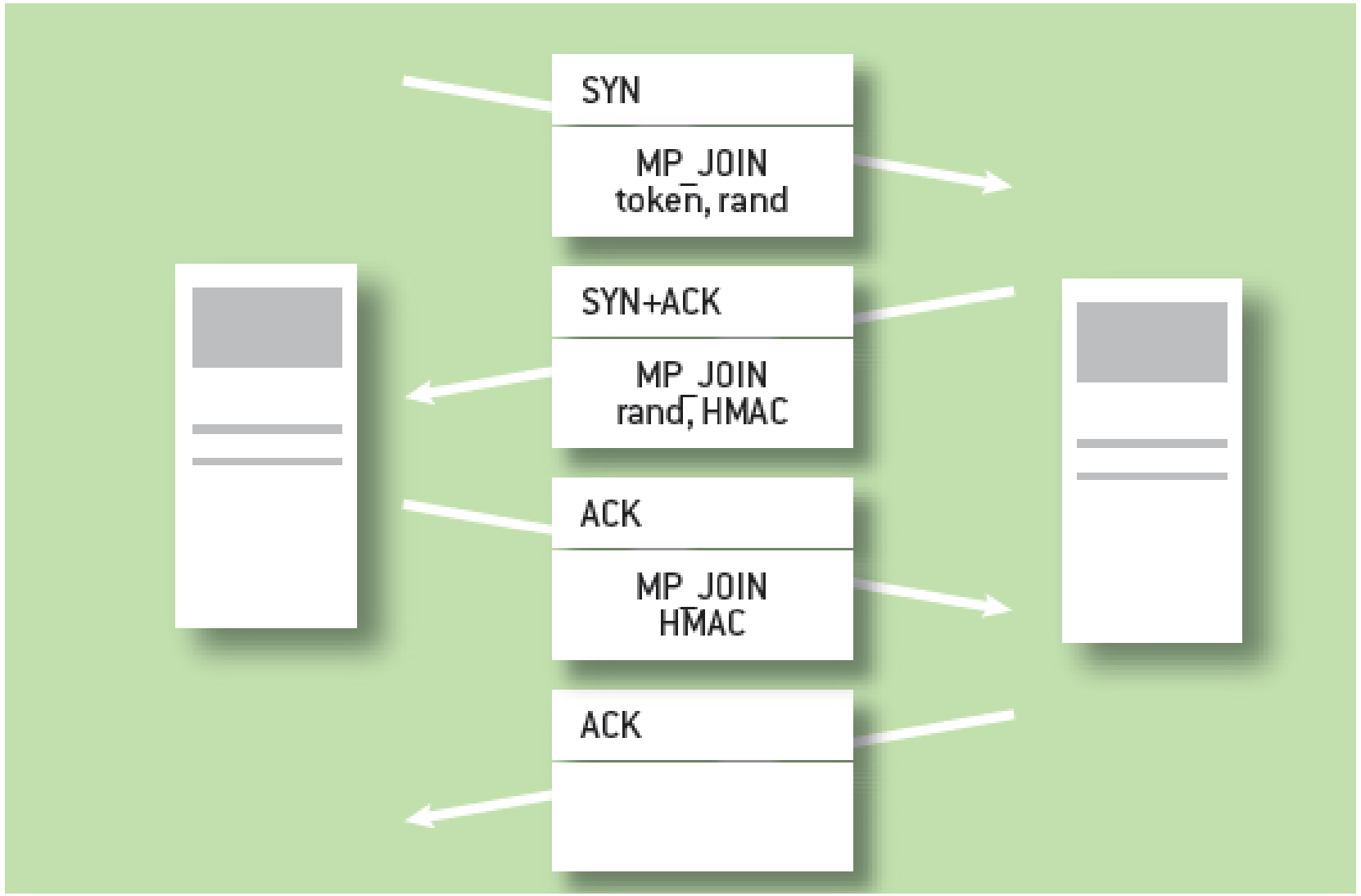
- To use another interface, **MPTCP uses a three-way handshake to establish one subflow over this interface**.
- Adding a subflow to an existing MPTCP connection requires the corresponding MPTCP connection to be uniquely identified on each end host.
- With regular TCP, a TCP connection is always identified by using the tuple **<SourceIP, DestIP, SourcePort, DestPort>**.
- Although on each host the 4-tuple is a unique local identification of each TCP connection, this identification is not globally unique (NAT is present).

- MPTCP needs to be able to link each subflow to an existing MPTCP connection.
- For this, **MPTCP assigns a locally unique token to each connection**.

- When a new subflow is added to an existing MPTCP connection, the **MP_JOIN option of the SYN segment contains the token** of the associated MPTCP connection.

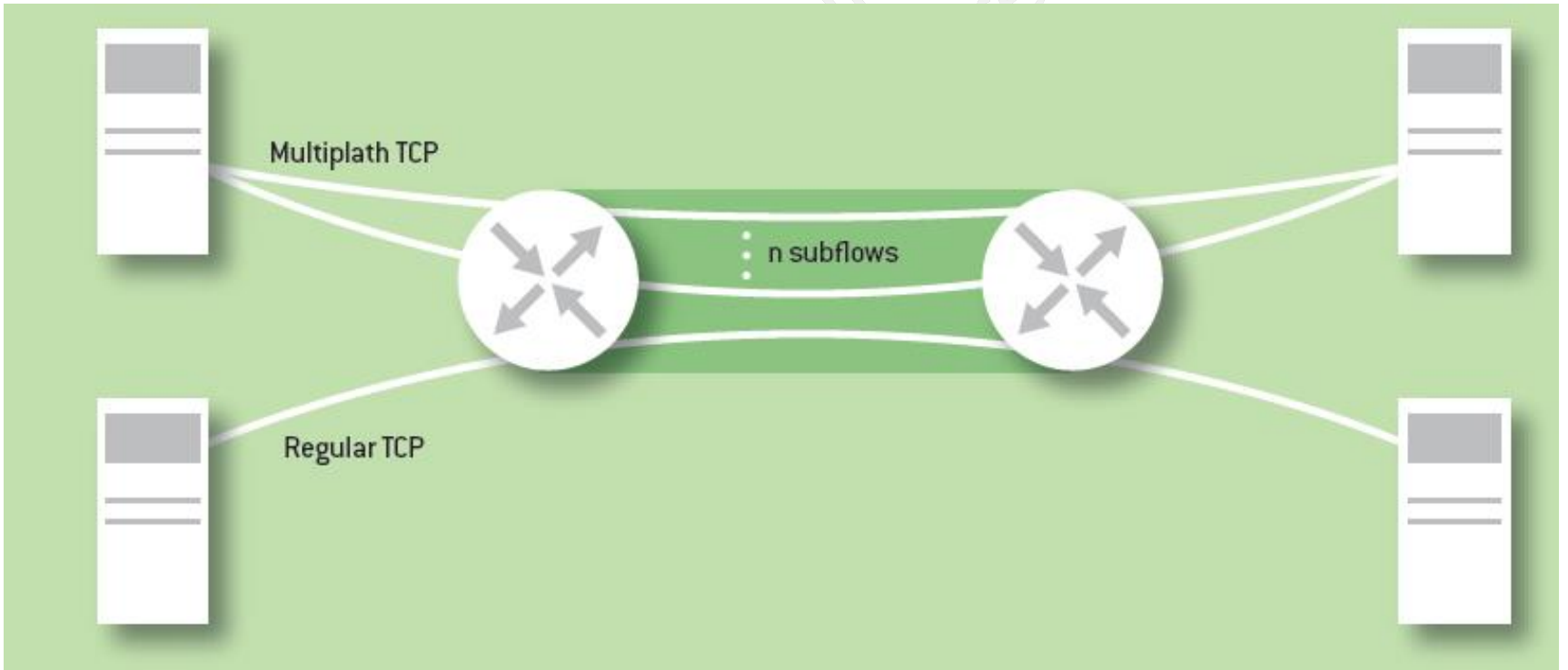
- To reduce the length of the MP_CAPABLE option and avoid using all the limited TCP options space (40 bytes) in the SYN segment, MPTCP **derives the token as the result of a truncated hash of the key**.

Add subflows to an MPTCP connection



Data transmission on the MPTCP connection:

- Now that the subflows have been established, MPTCP can use them to exchange data.
- Each host can send data over any of the established subflows.
- Furthermore, data transmitted over one subflow can be retransmitted on another to recover from losses.
- This is achieved by using **two levels of sequence numbers**. The **regular TCP sequence number** ensures that data is received in order over each subflow and allows losses to be detected.
- MPTCP uses the **data sequence number** to reorder the data received over different subflows before passing it to the application.



USE CASES

- **MPTCP ON SMARTPHONES:**

- Smartphones are equipped with Wi-Fi and 3G/4G interfaces, but they typically use only one interface at a time.
- Still, users expect their TCP connections to survive when their smartphone switches from one wireless network to another.
- With regular TCP, switching networks implies changing the local IP address and leads to a termination of all established TCP connections.
- With MPTCP, the situation could change because it enables seamless handovers from Wi-Fi to 3G/4G and vice versa.

School of Instrumentation
DAVV Indore

Different types of handovers are possible:

• ***make-before-break* handover:** can be used if the smartphone can predict that one interface will disappear soon (e.g., because of a decreasing radio signal). In this case, a new subflow will be initiated on the second interface and the data will switch to this interface.

• ***break-before-make* handover**: With this, MPTCP can react to a failure on one interface by enabling another interface and starting a subflow on it. Once the subflow has been created, the data that was lost as a result of the failure of the first interface can be retransmitted on the new subflow, and the connection continues without interruption.

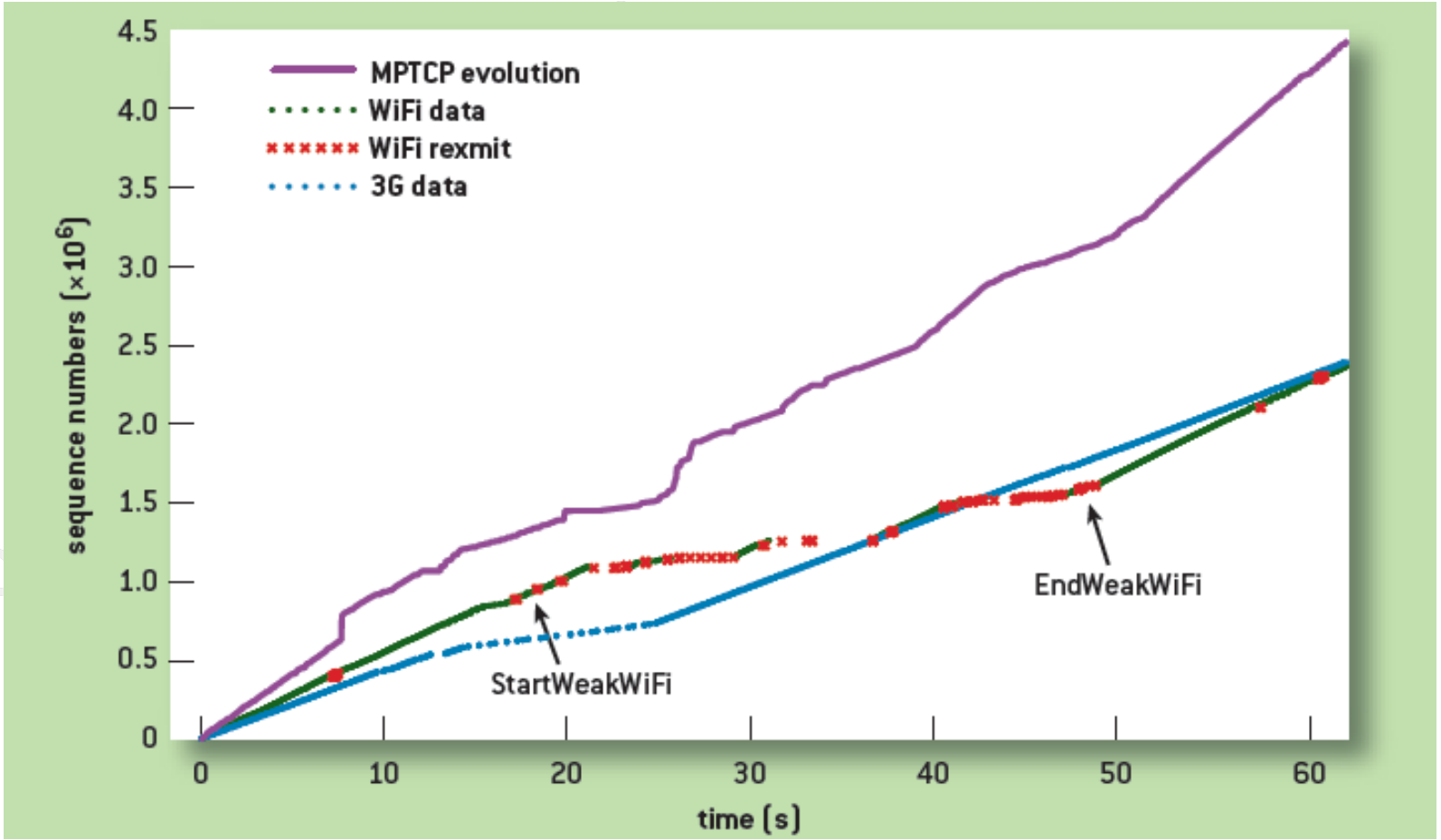
• **The third handover mode uses two or more interfaces simultaneously**. With MPTCP, data can be transmitted over both interfaces to speed up the data transfer. From an energy viewpoint, enabling two radio interfaces is more costly than enabling a single one; however, the phone's display often consumes much more energy than the radio interfaces. When the user looks at the screen (e.g., while waiting for a Web page), increasing the download speed by combining two interfaces could reduce the display usage and thus the energy consumption.

School of Instrumentation, DAVV Indore

An analysis of the operation of MPTCP in real Wi-Fi and 3G networks.

A user starts a download inside a building over Wi-Fi and then moves outside. The Wi-Fi connectivity is lost as the user moves, but thanks to MPTCP the data transfer is not affected.

- Figure shows a typical MPTCP trace collected by using both 3G and Wi-Fi. The x-axis shows the time since the start of the data transfer, while the y-axis displays the evolution of the sequence numbers of the outgoing packets.
- For this transfer, MPTCP was configured to use both Wi-Fi and 3G simultaneously to maximize performance.
- With MPTCP, the packets that are lost over the Wi-Fi interface are automatically retransmitted over the 3G subflow, and the MPTCP connection continues without interruption.



3G/WiFi Handover with Multipath TCP

MPTCP IN THE DATA CENTER:

- Another important use case for MPTCP lies in data centers.
- Today, most servers are equipped with several high-speed interfaces, which can be combined to **achieve either higher performance or better resilience to failures**.
- **With MPTCP, it is possible both to improve performance and to achieve high availability**.
- A typical MPTCP-aware server design would use two interfaces attached to different switches.
- Each interface has its own MAC and IP addresses. Once an MPTCP connection has been initiated over one interface, MPTCP will announce the other available IP addresses to the remote host by using the **ADD_ADDR** option, and subflows will be established over these interfaces.
- Once these subflows have been established, the MPTCP congestion-control scheme will dynamically spread the load over the available interfaces.
- If one interface or any intermediate switch fails, MPTCP automatically changes to the remaining paths.

School of Implementation
DAW Models

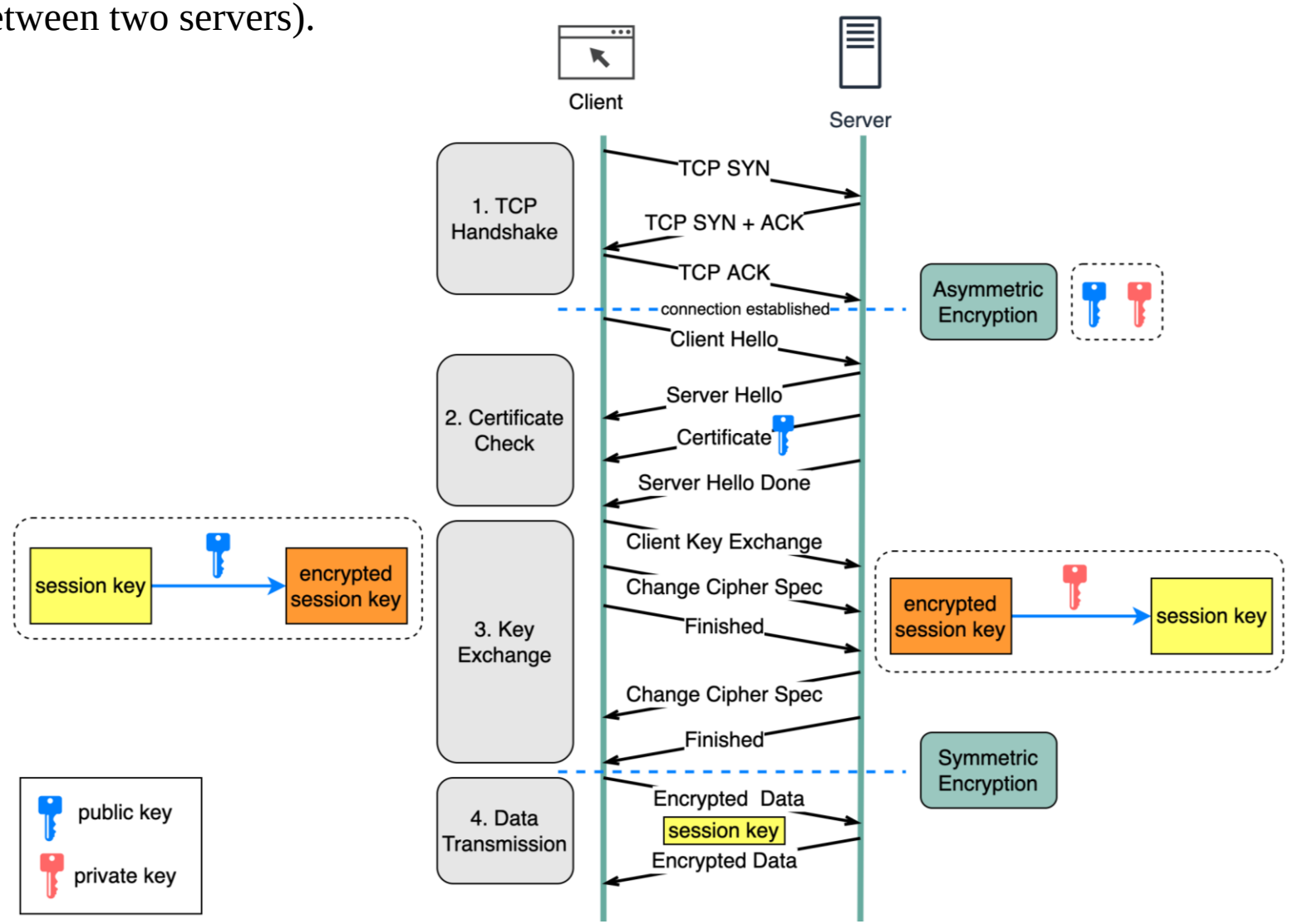
TLS/SSL

- **SSL (SECURE SOCKETS LAYER)** is standard technology for securing an internet connection by encrypting data sent between a website and a browser (or between two servers).

- **TLS (Transport Layer Security)** is an updated, more secure version of SSL.

- SSL/TLS prevents hackers from seeing or stealing any information transferred, including personal or financial data.

- SSL/TLS is **used by https** protocol to send data in encrypted from.
- https appears in the URL when a website is secured by an SSL/TLS certificate.



TLS Handshake

It creates a protected tunnel between client and server to protect data transfer.

School of Instrumentation, DAVV Indore

Digital Certificate

Before continuing with the TLS handshake, let us first look at digital certificate

- **Public key certificate** or **digital certificate** is an electronic document used to prove the validity of a public key.

- The **certificate includes the public key and information about it, information about the identity of its owner (e.g. server), and the digital signature of an entity that has verified the certificate's contents** (called the issuer).

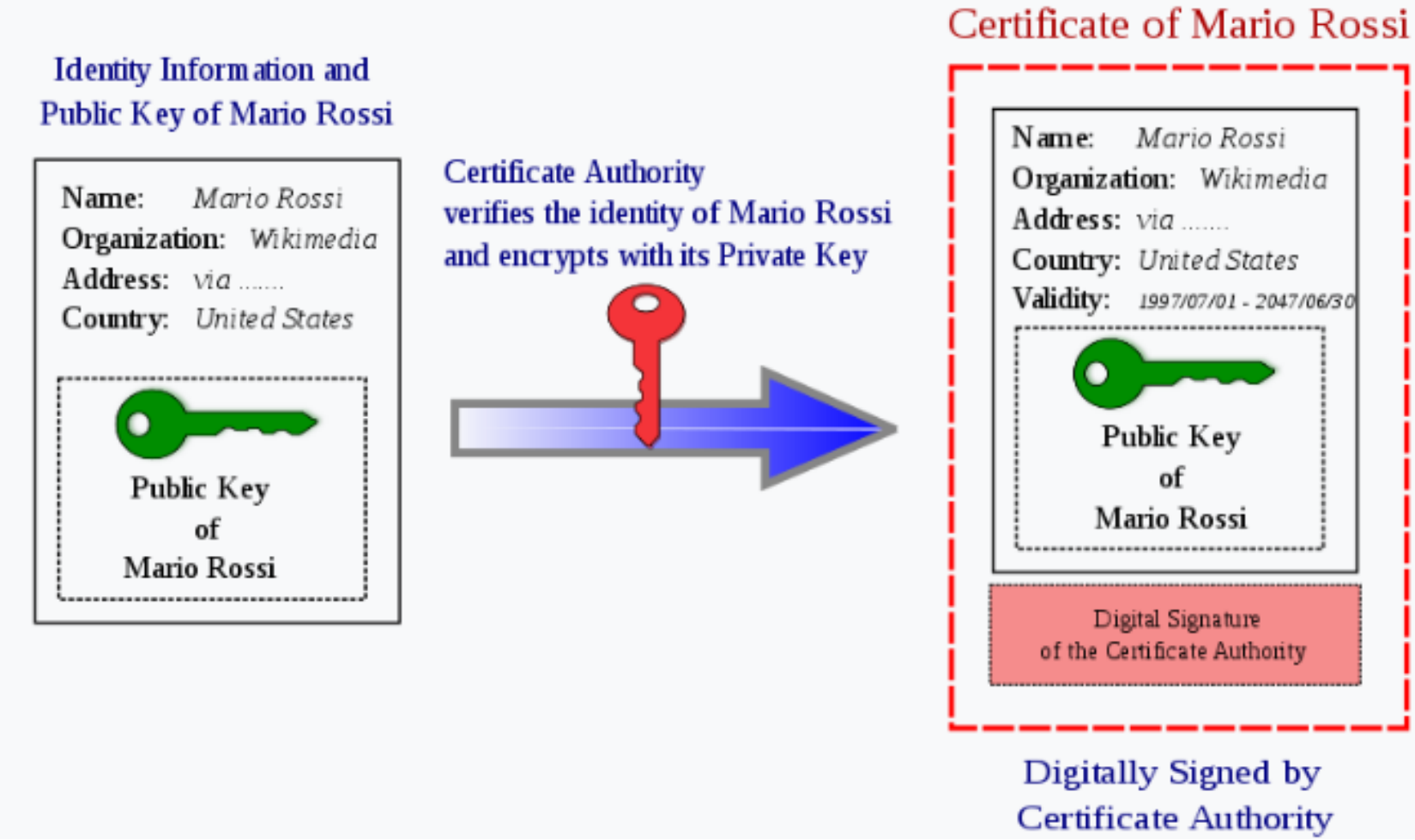
- The certificate issuer is a **certificate authority (CA),** usually a company, that charges customers a fee to issue certificates for them.

- A **CA** is responsible for signing certificates and acts as a trusted third party.
- A **CA** processes requests from people or organizations requesting certificates (called subscribers), verifies the information, and potentially signs an end-entity certificate based on that information.

School of Instrumentation, DAVV Indore

Digital Certificate

The procedure of obtaining a Public key certificate



TLS Certificate

Before continuing with the TLS handshake, let us look at TLS certificate

- An SSL/TLS certificate is a digital object that **allows systems to verify the identity** & subsequently **establish an encrypted network connection to another system using the SSL/TLS protocol**.
- SSL/TLS certificates thus act as digital identity cards to secure network communications, establish the identity of websites over the Internet as well as resources on private networks.
- SSL/TLS certificates establish trust among website users. **Businesses install SSL/TLS certificates on web servers** to create SSL/TLS-secured websites.
- **Browsers validate the SSL/TLS certificate of any website to start and maintain secure connections** with the website server.
- SSL/TLS technology **helps ensure the encryption of all communication between your browser and the website**.

School of Instrumentation, DAVV Indore

Key principles in SSL/TLS certificate technology:

Public key

- The browser and webserver communicate by encrypting and decrypting information using public and private key pairs.
- The public key is a cryptographic key that the web server gives the browser in the SSL/TLS certificate. The browser uses the key to encrypt the information before sending it to the web server.

Private key

- Only the web server has the private key. A file that is encrypted by the private key can only be decrypted by the public key, and vice versa.
- If the public key can only decrypt the file that has been encrypted by the private key, being able to decrypt that file assures that the intended receiver and sender are who they claim to be.

Authentication

- The server sends the public key in the SSL/TLS certificate to the browser. The browser verifies the certificate from a trusted third party. Hence, it can verify that the web server is who it claims to be.

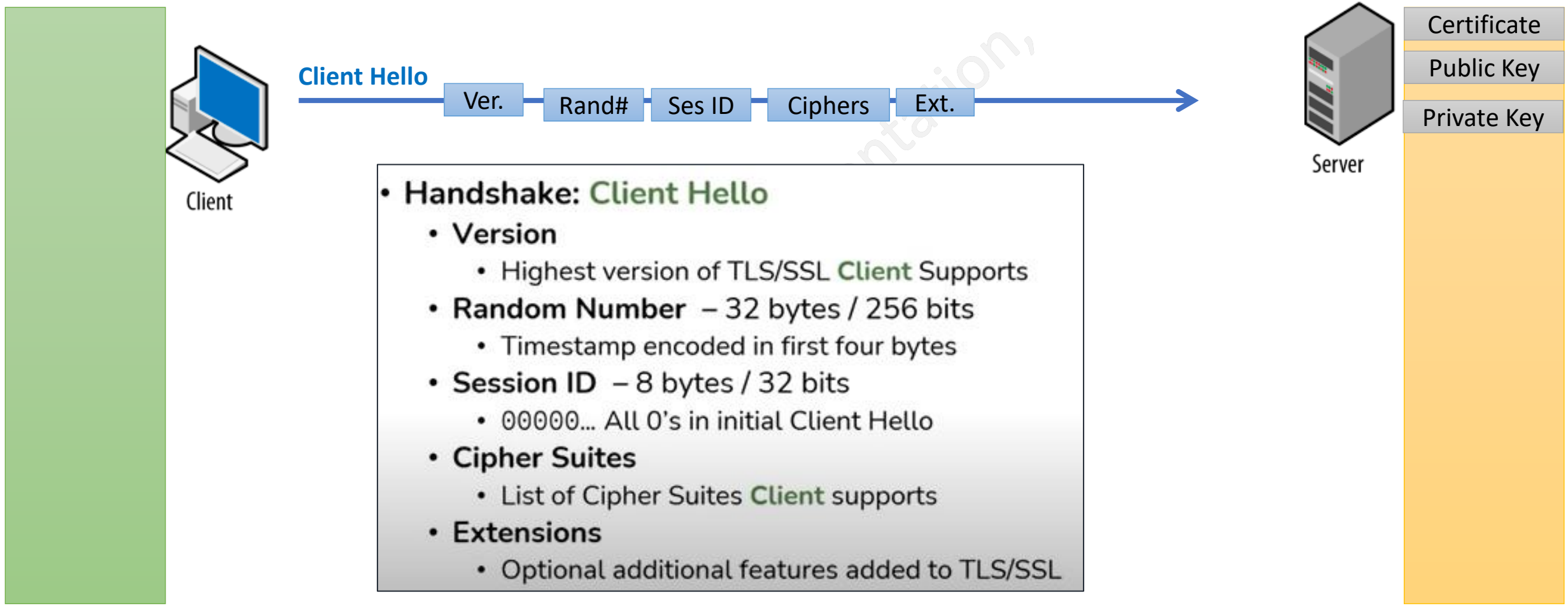
Digital signature

- A digital signature is a number unique to every SSL/TLS certificate. It is nothing but the signing of digital certificate by the CA using his private key (A CA has its own public ley- private key pair; and the public key is also shared with the web browser).
- The recipient/client/web browser already has the CA's public key (obtained in advanced from CA) which is used to validate that the certificate is legitimate.

School of Instrumentation, DAVV Indore

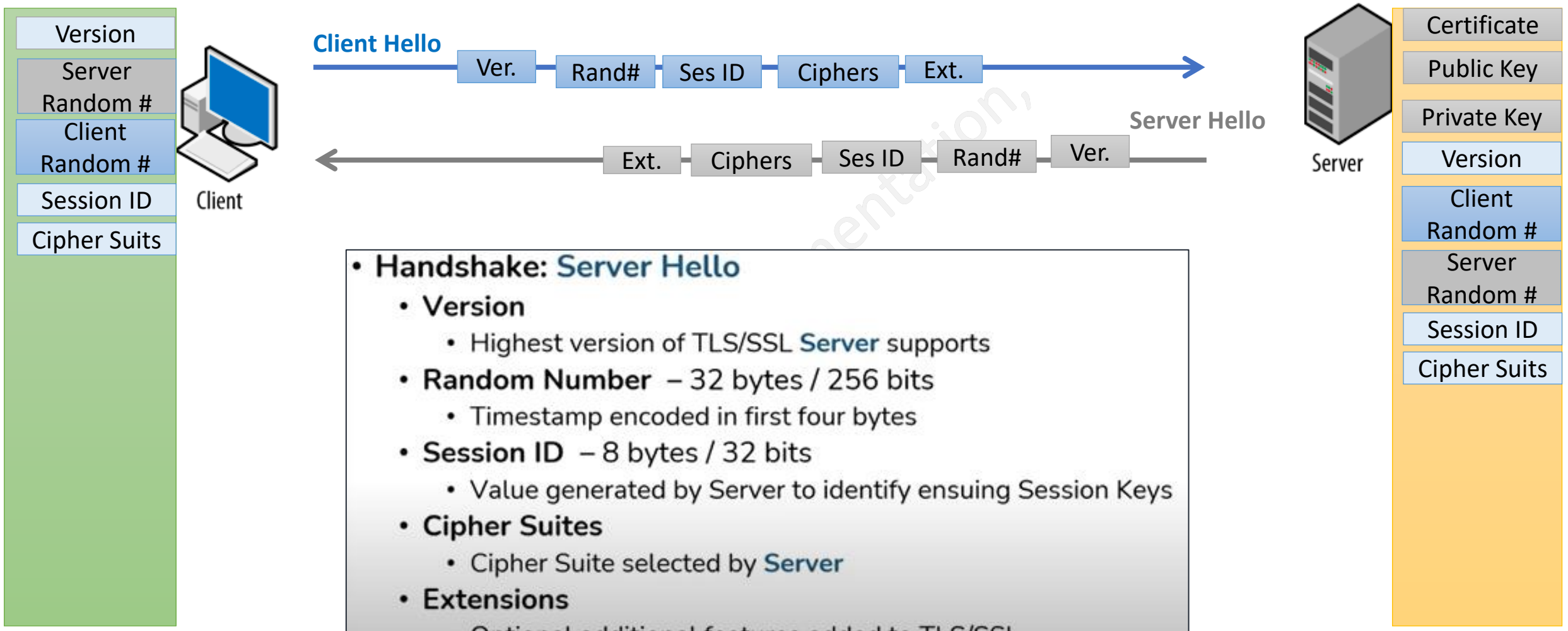
TLS Handshake cont..

It creates a protected tunnel between client and server to protect data transfer.



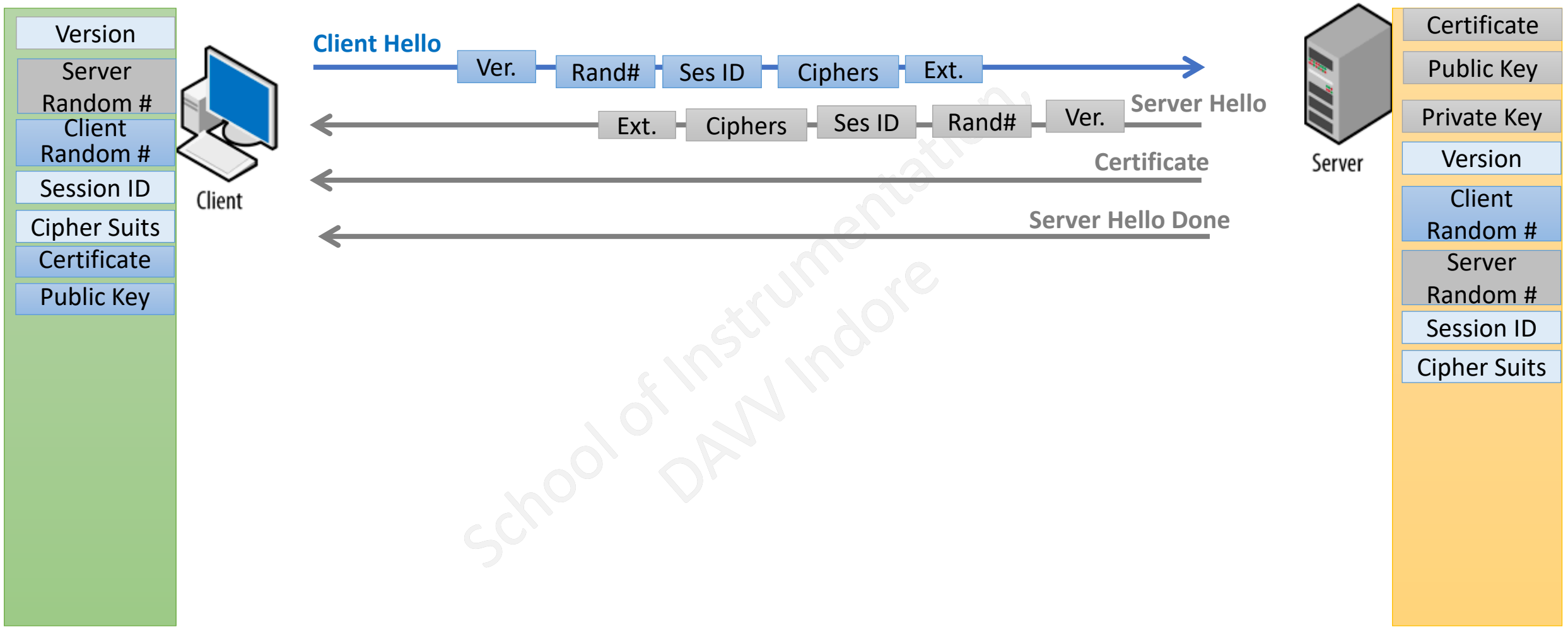
- A cipher suite is a collection of algorithms that create keys to encrypt information between a browser and a server.
- It includes a key exchange algorithm, a validation algorithm, a bulk encryption algorithm, and a MAC algorithm.

TLS Handshake cont..



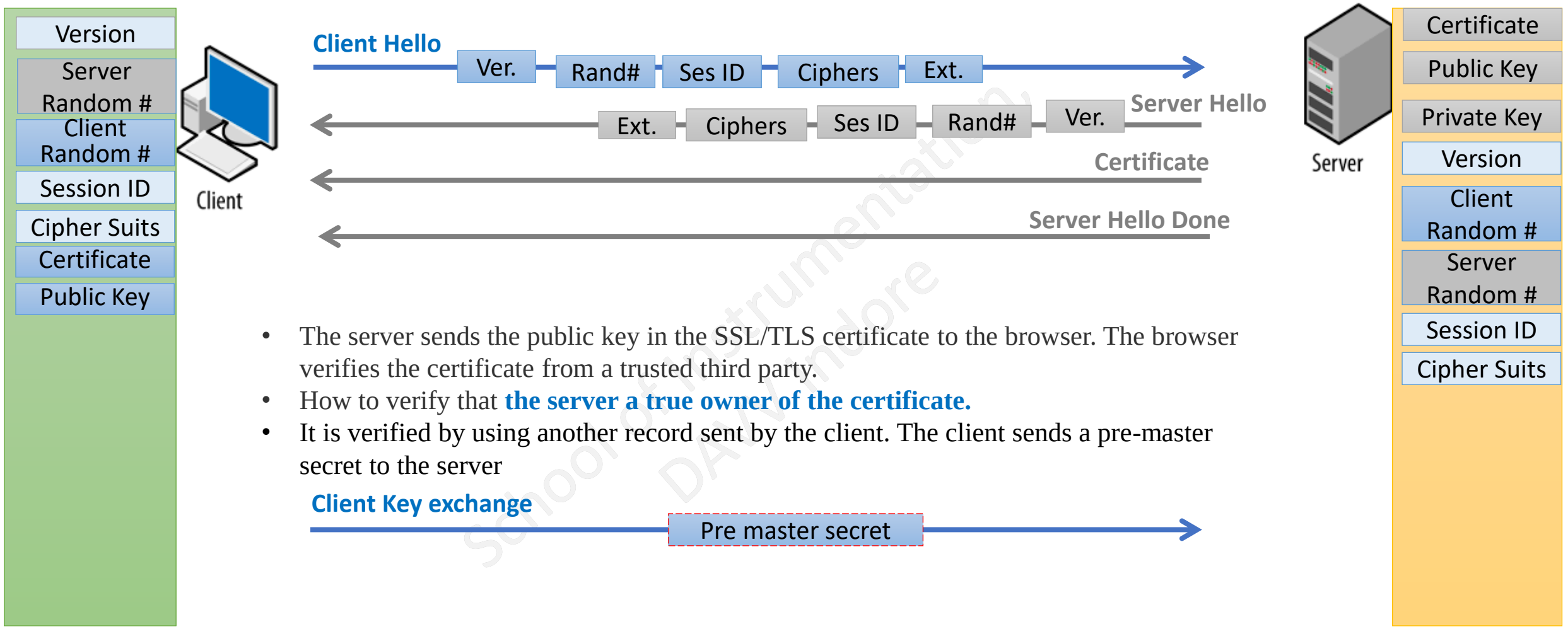
- **Handshake: Server Hello**
  - **Version**
    - Highest version of TLS/SSL Server supports
  - **Random Number** – 32 bytes / 256 bits
    - Timestamp encoded in first four bytes
  - **Session ID** – 8 bytes / 32 bits
    - Value generated by Server to identify ensuing Session Keys
  - **Cipher Suites**
    - Cipher Suite selected by Server
  - **Extensions**
    - Optional additional features added to TLS/SSL

TLS Handshake



School of Instrumentation, DAVV Indore

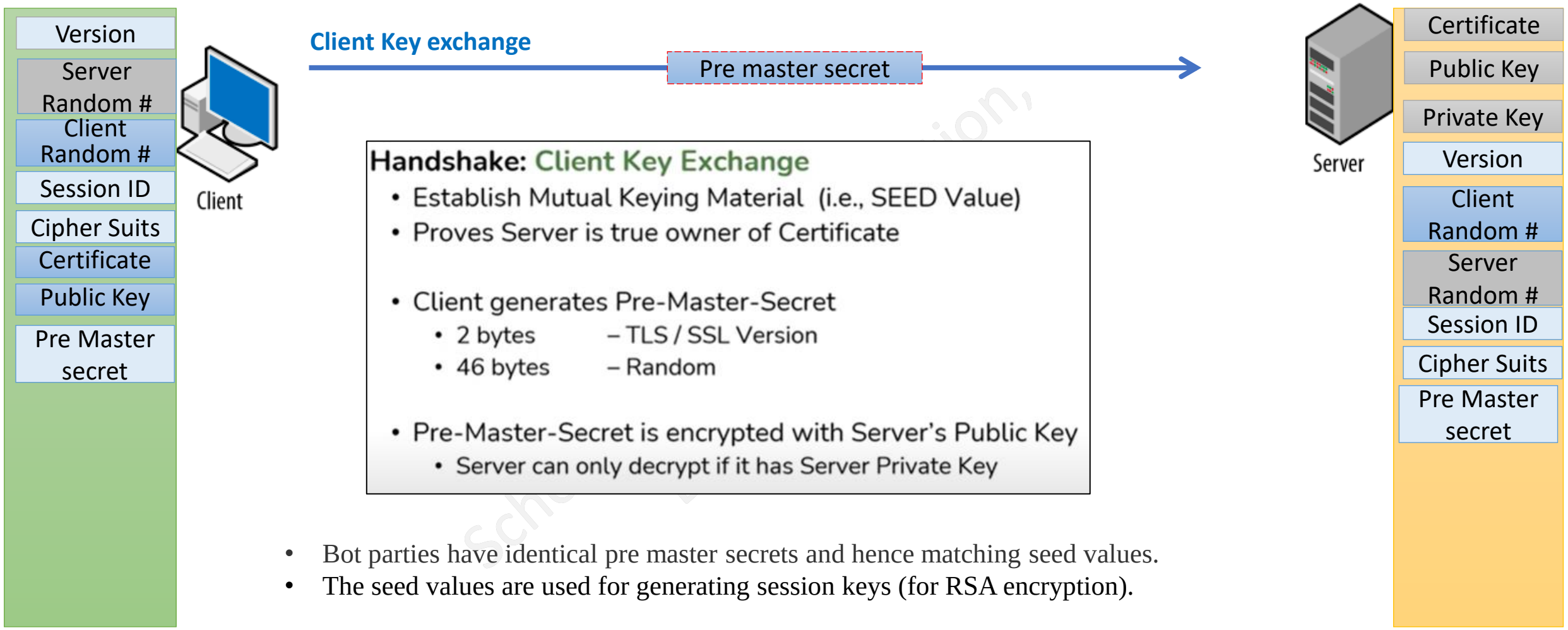
TLS Handshake cont..



- The server sends the public key in the SSL/TLS certificate to the browser. The browser verifies the certificate from a trusted third party.
- How to verify that **the server a true owner of the certificate.**
- It is verified by using another record sent by the client. The client sends a pre-master secret to the server

School of Instrumentation, DAVV Indore

TLS Handshake cont..



Handshake: Client Key Exchange

- Establish Mutual Keying Material (i.e., SEED Value)
- Proves Server is true owner of Certificate

- Client generates Pre-Master-Secret
  - 2 bytes – TLS / SSL Version
  - 46 bytes – Random

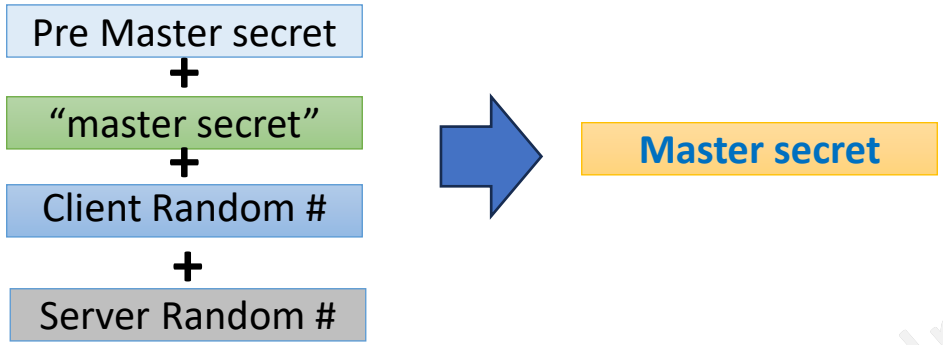
- Pre-Master-Secret is encrypted with Server's Public Key
  - Server can only decrypt if it has Server Private Key

- Bot parties have identical pre master secrets and hence matching seed values.
- The seed values are used for generating session keys (for RSA encryption).

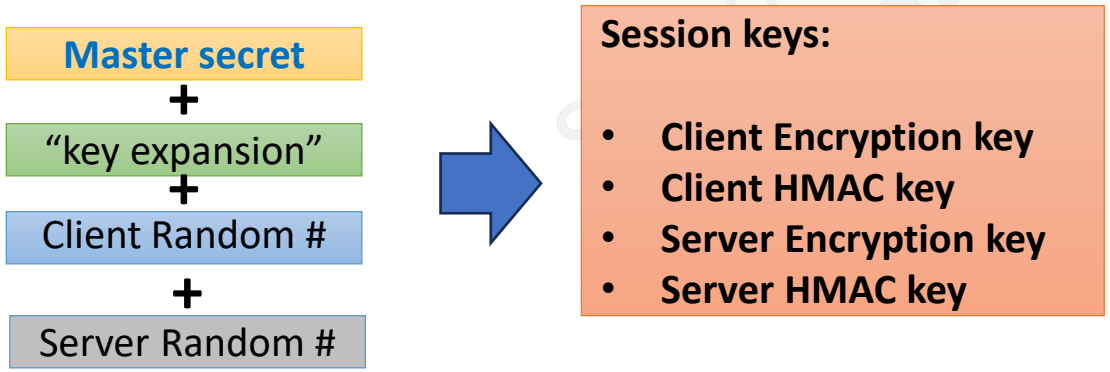
TLS Handshake cont..

Generating session keys from SEED

First we derive **Master secret** from Pre master secret



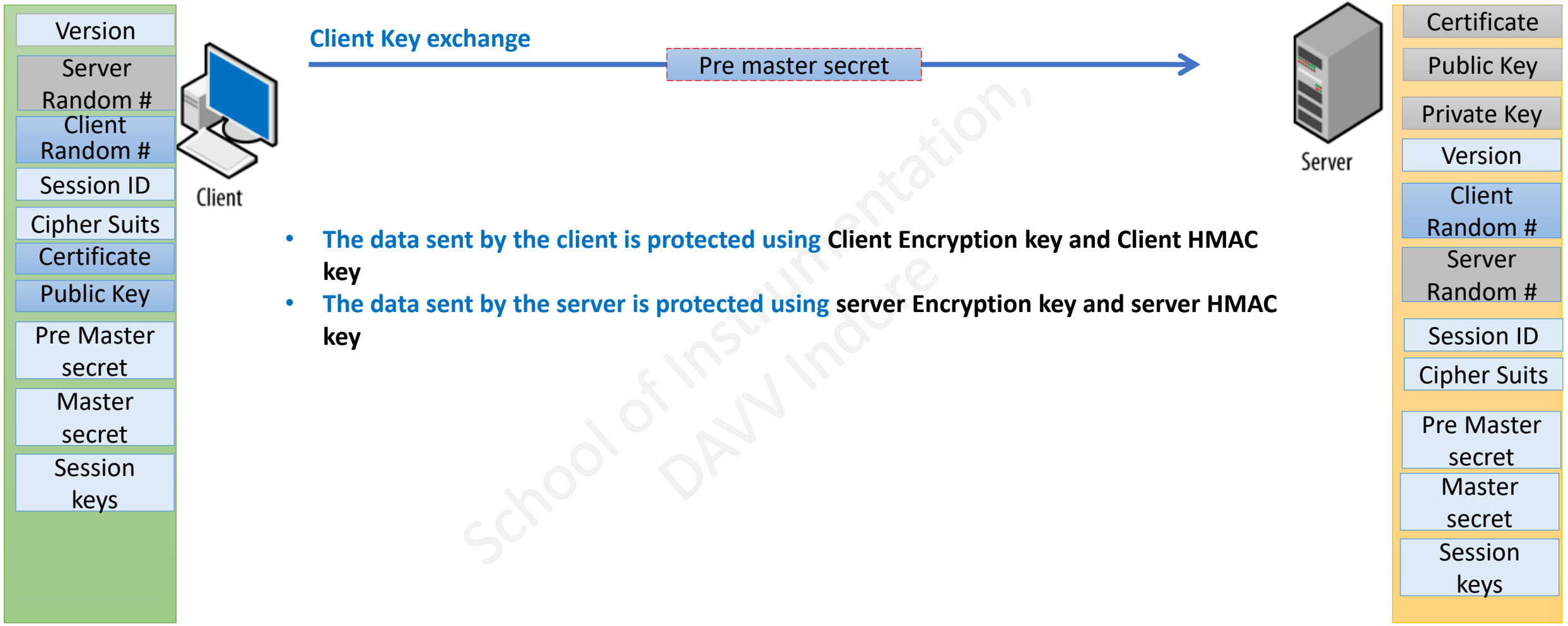
Master secret is used to generate session keys as follows:



- ✓ **Encryption ensures confidentiality**
- ✓ **HMAC ensures data integrity and authentication**

of Instrumentation, IIITDM Kancheepuram, Indore

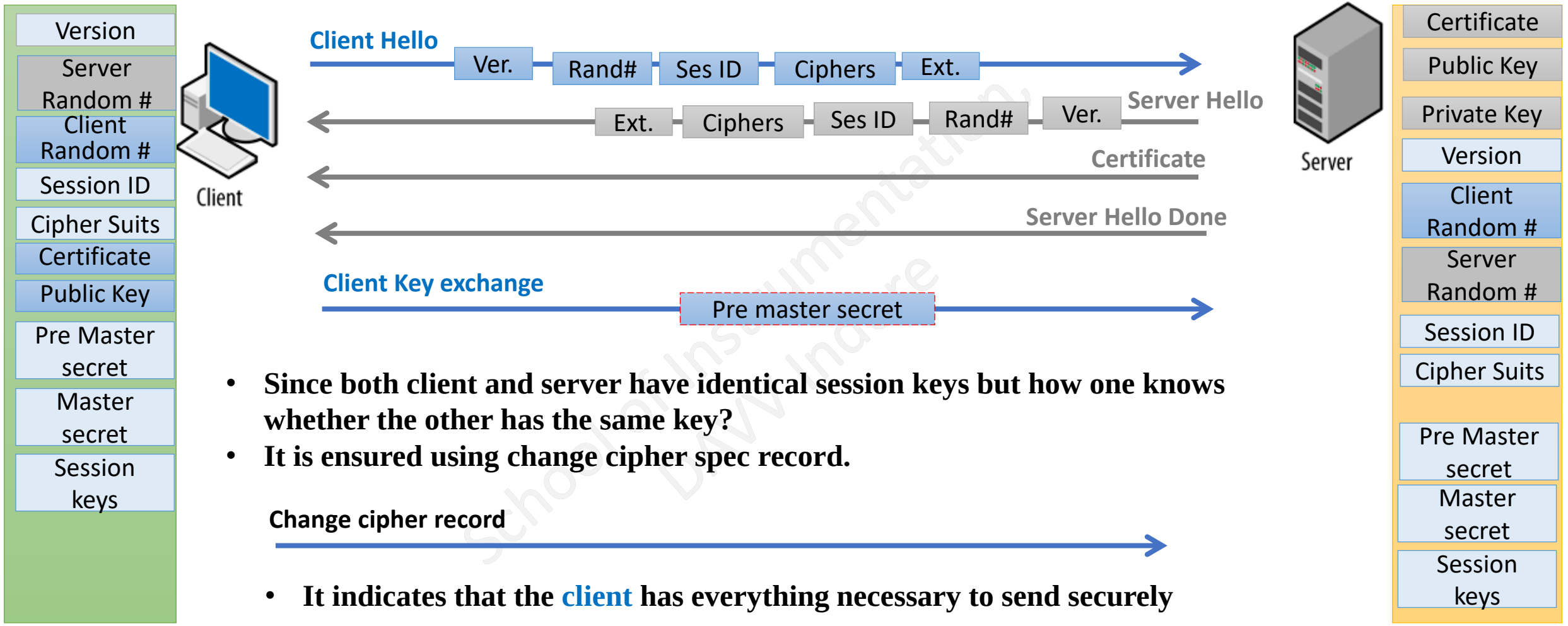
TLS Handshake cont..



- **The data sent by the client is protected using Client Encryption key and Client HMAC key**
- **The data sent by the server is protected using server Encryption key and server HMAC key**

School of Instrumentation, DAVV Indore

TLS Handshake cont..

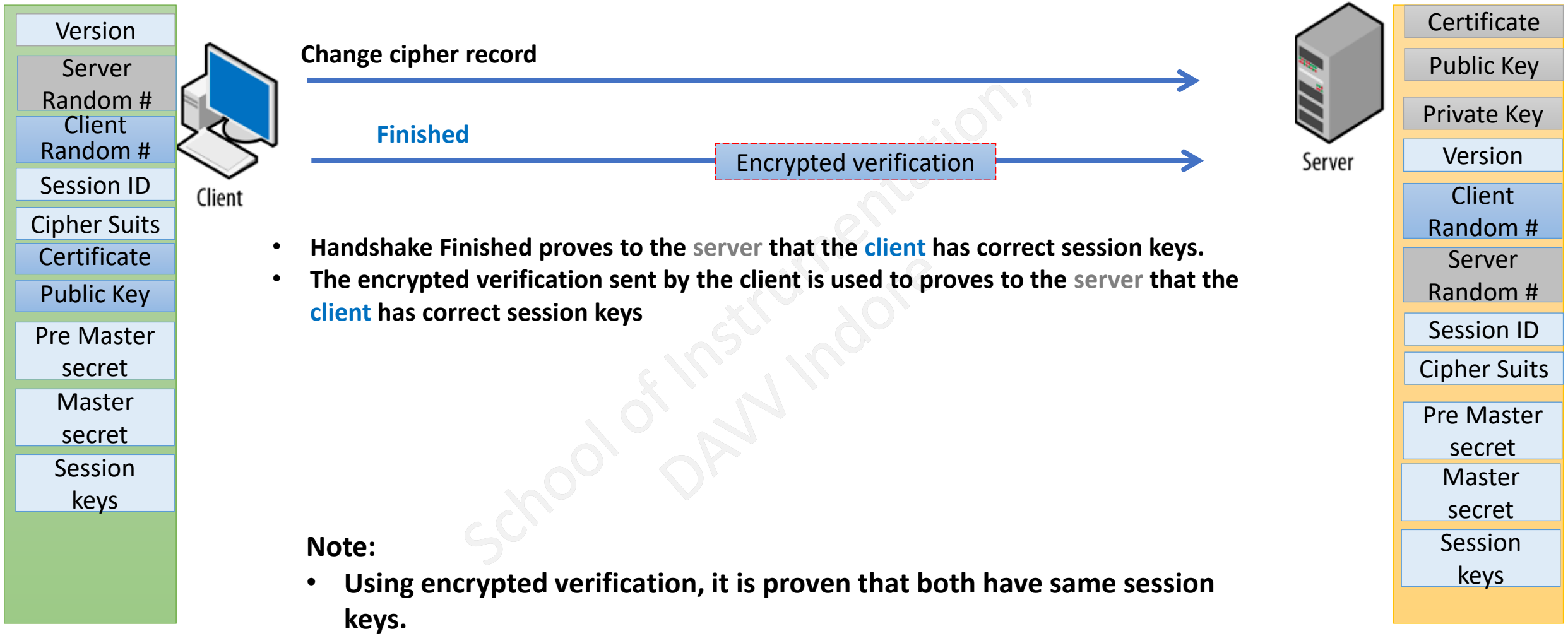


- **Since both client and server have identical session keys but how one knows whether the other has the same key?**
- **It is ensured using change cipher spec record.**

Change cipher record

- **It indicates that the client has everything necessary to send securely**

TLS Handshake cont..



- **Handshake Finished proves to the server that the client has correct session keys.**
- **The encrypted verification sent by the client is used to proves to the server that the client has correct session keys**

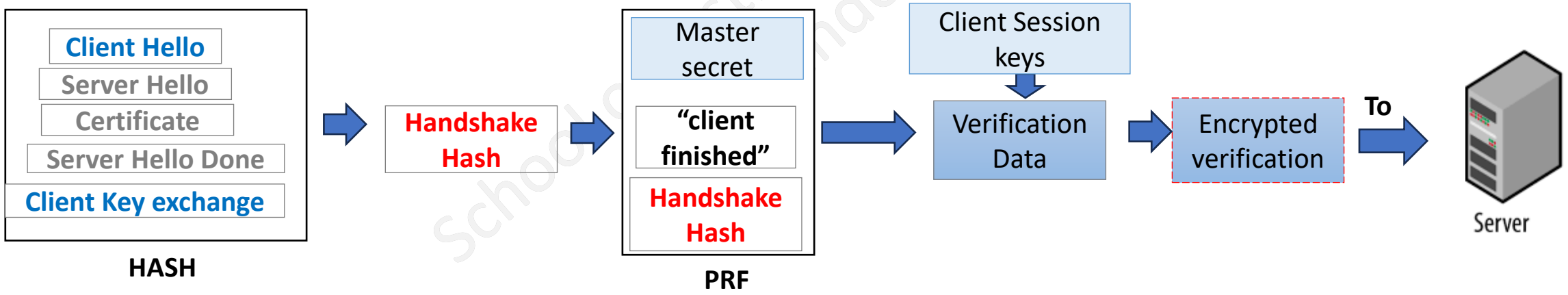
School of Instrumentation, DAVV Indore

Note:

- **Using encrypted verification, it is proven that both have same session keys.**
- **How encrypted verification is produced and how it verifies that both have same session keys?**

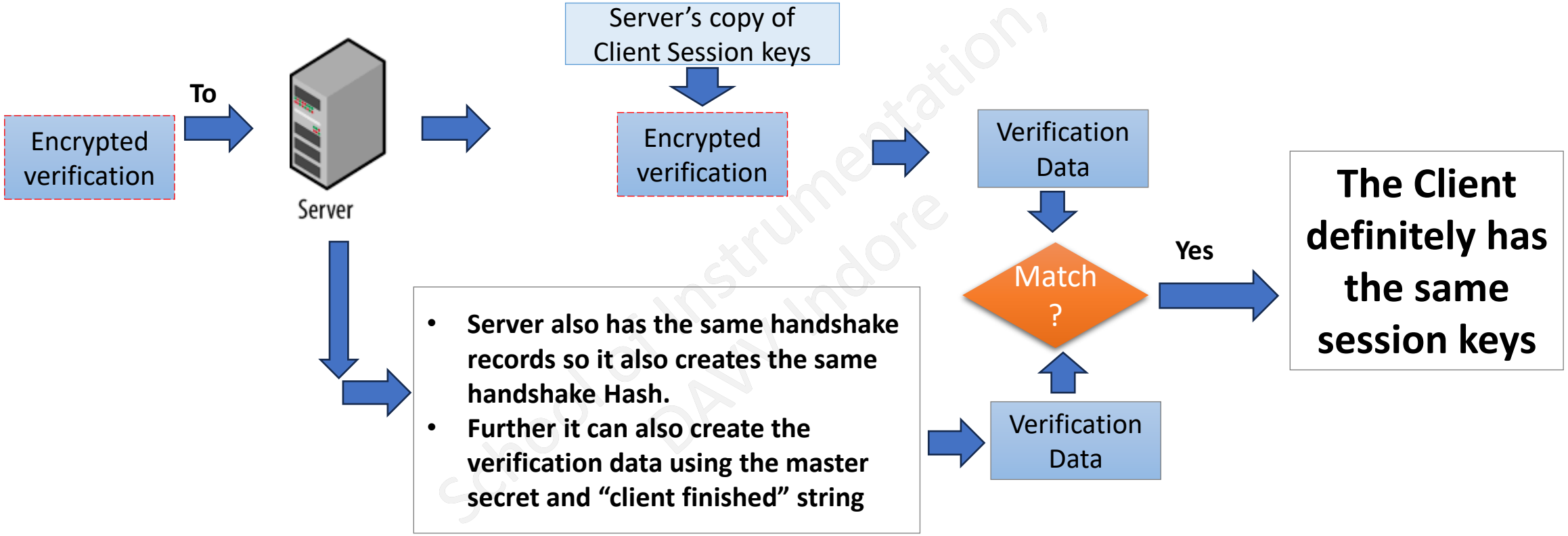
TLS Handshake cont..

- **How encrypted verification is produced and how it verifies that both client and server have same session keys?**
  - Client calculates Hash of all handshake records seen so far and combine with other values to create verification data
  - The verification data is encrypted with the client session keys
  - Server verifies with the server's copy of the client session keys



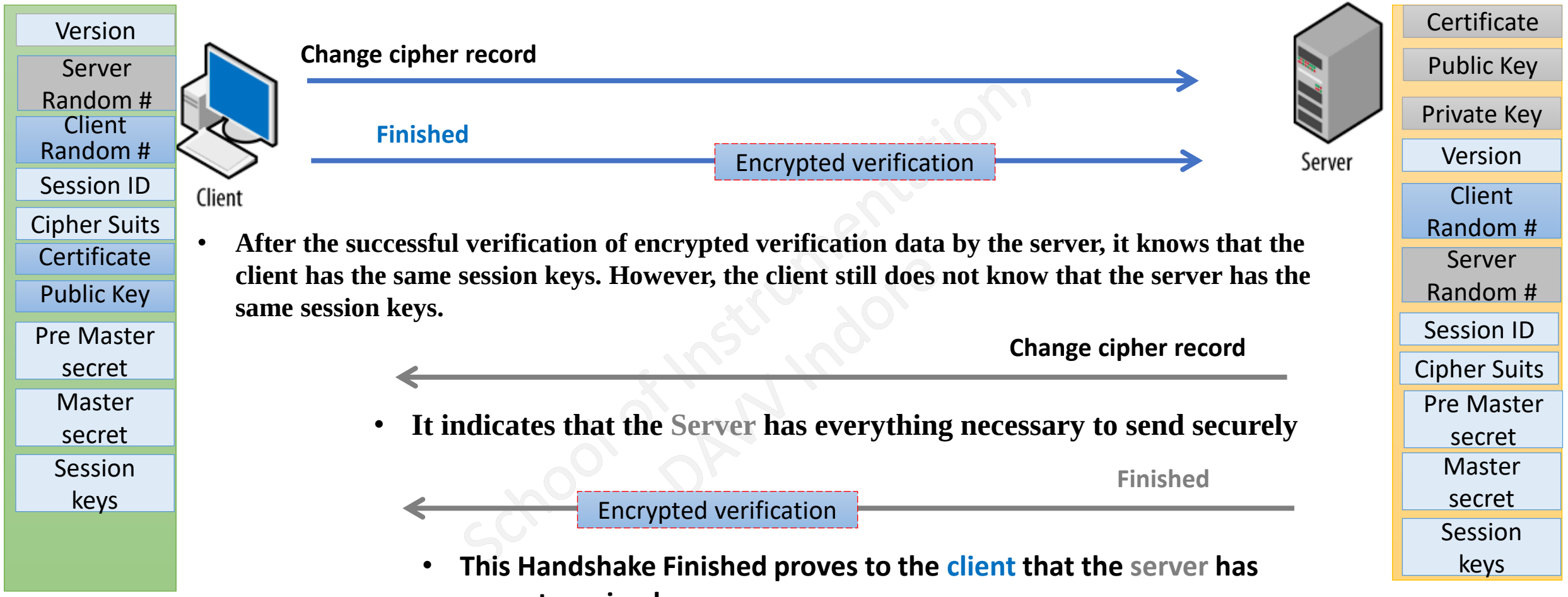
TLS Handshake cont..

- **How encrypted verification is produced and how it verifies that both client and server have same session keys?**



- **If there is any tampering with the record then this match will not happen.**
- **After the successful verification of encrypted verification data by the server, it knows that the client has the same session keys.**

TLS Handshake cont..



- **After the successful verification of encrypted verification data by the server, it knows that the client has the same session keys. However, the client still does not know that the server has the same session keys.**

Change cipher record

- **It indicates that the Server has everything necessary to send securely**

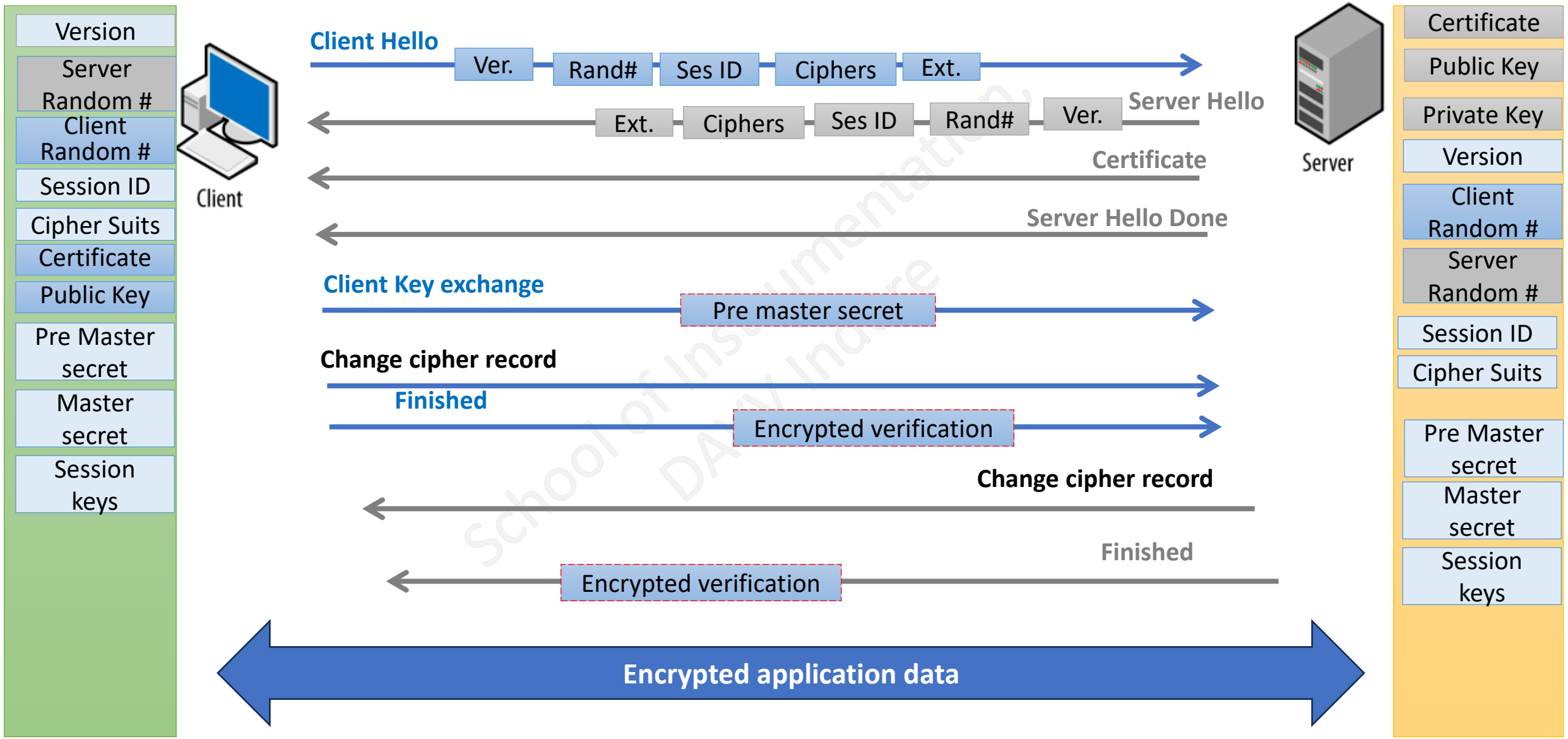
Finished

Encrypted verification

- **This Handshake Finished proves to the client that the server has correct session keys.**

- Server calculates Hash of all handshake records seen so far and combine with other values to create verification data.
- The verification data is encrypted with the server session keys
- Client verifies with the client's copy of the server session keys

TLS Handshake cont..



Differences between SSL and TLS?

SSL is an older technology that contains some security flaws.
Transport Layer Security (TLS) is the upgraded version of SSL that fixes existing SSL vulnerabilities. TLS authenticates more efficiently and continues to support encrypted communication channels.

An SSL handshake was an explicit connection, while a TLS handshake is an implicit one. The SSL handshake process had more steps than the TLS process. By removing additional steps and reducing the total number of cipher suites, TLS has sped up the process.

The SSL protocol uses the MD5 algorithm—which is now outdated—for MAC generation. TLS uses Hash-Based Message Authentication Code (HMAC) for more complex cryptography and security.

Several key generation and encryption algorithms in TLS were upgraded from SSL due to security concerns.

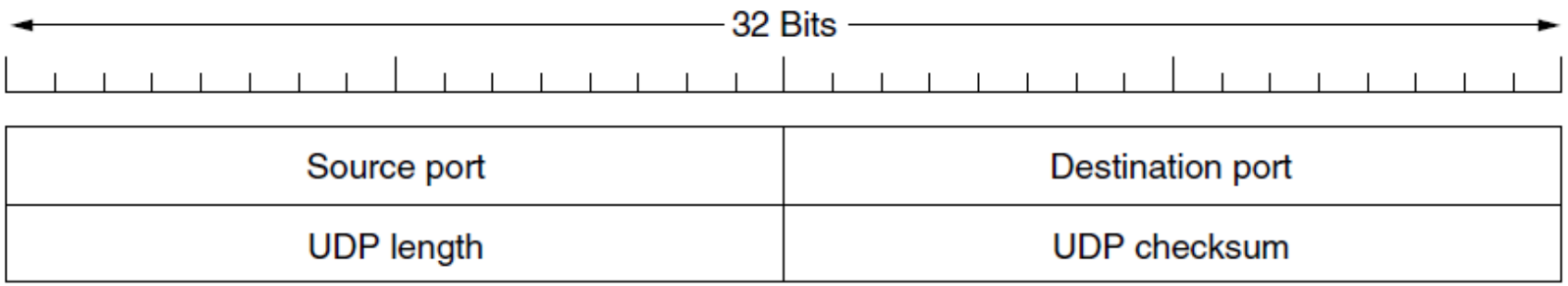
- At present, all SSL certificates are no longer in use. TLS certificates are the industry standard.
- However, the industry continues to use the term *SSL* to refer to TLS certificates.

UDP (User Datagram Protocol)

- UDP is a connectionless protocol.
- It does almost nothing beyond sending packets between applications, letting applications build their own protocols on top as needed.
- UDP provides a way for applications to send encapsulated IP datagrams without having to establish a connection.
- UDP is described in RFC 768.

School of Instrumentation,
DAVV Indore

UDP Header



- UDP transmits segments consisting of an 8-byte header followed by the payload.

- The two ports serve to identify the endpoints within the source and destination machines.
- When a UDP packet arrives, its payload is handed to the process attached to the destination port.

- The UDP length field includes the 8-byte header and the data.
- The minimum length is 8 bytes, to cover the header.
- The maximum length is 65,515 bytes, which is lower than the largest number that will fit in 16 bits because of the size limit on IP packets.

- An **optional Checksum** is also provided **for extra reliability**.
- It checksums the header, the data, and a conceptual IP pseudo header.
- When performing this computation, the Checksum field is set to zero and the data field is padded out with an additional zero byte if its length is an odd number.
- The checksum algorithm is simply to add up all the 16-bit words in one’s complement and to take the one’s complement of the sum.
- As a consequence, when the receiver performs the calculation on the entire segment, including the Checksum field, the result should be 0.
- Turning it off is foolish unless the quality of the data does not matter.

School of Instrumentation
DAVV Indore

- It **does not do flow control, congestion control, or retransmission upon receipt of a bad segment**. All of that is up to the user processes.
- One area where it is especially useful is in client-server situations. Often, the client sends a short request to the server and expects a short reply back. If either the request or the reply is lost, the client can just time out and try again.
- An application that uses UDP this way is DNS (Domain Name System), a program that needs to look up the IP address of some host name, for example, www.cs.berkeley.edu, can send a UDP packet containing the host name to a DNS server.
- The server replies with a UDP packet containing the host's IP address.
- **No setup is needed in advance and no release is needed afterward**. Just two messages go over the network.

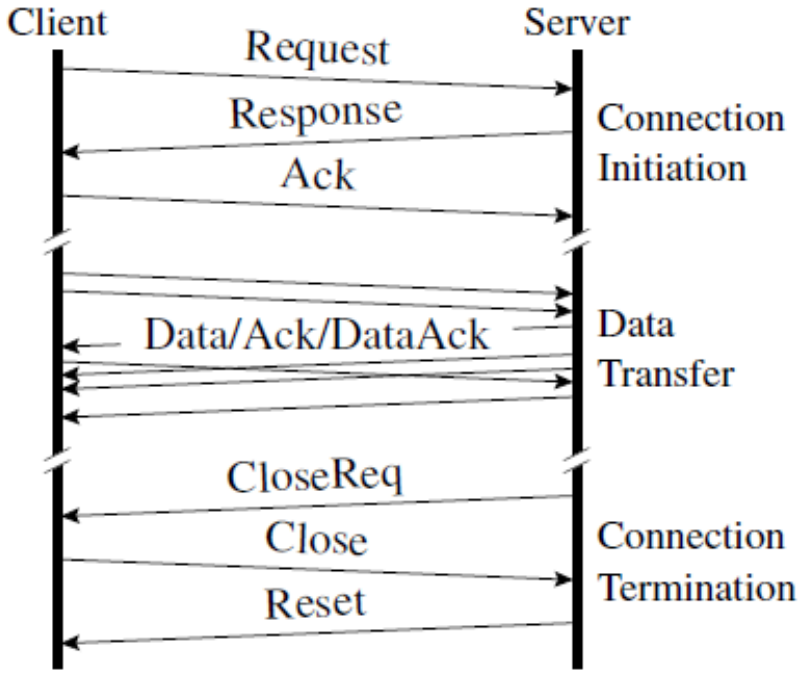
School of Instrumentation, DAVV Indore

DCCP: Datagram Congestion Control Protocol

- Fast-growing Internet applications like streaming media and telephony prefer timeliness to reliability, making TCP a poor fit.
- Unfortunately, UDP, the natural alternative, lacks congestion control.

- **DCCP is a unicast, connection-oriented protocol with bidirectional data flow.**
- **It is a message-oriented transport layer protocol.**

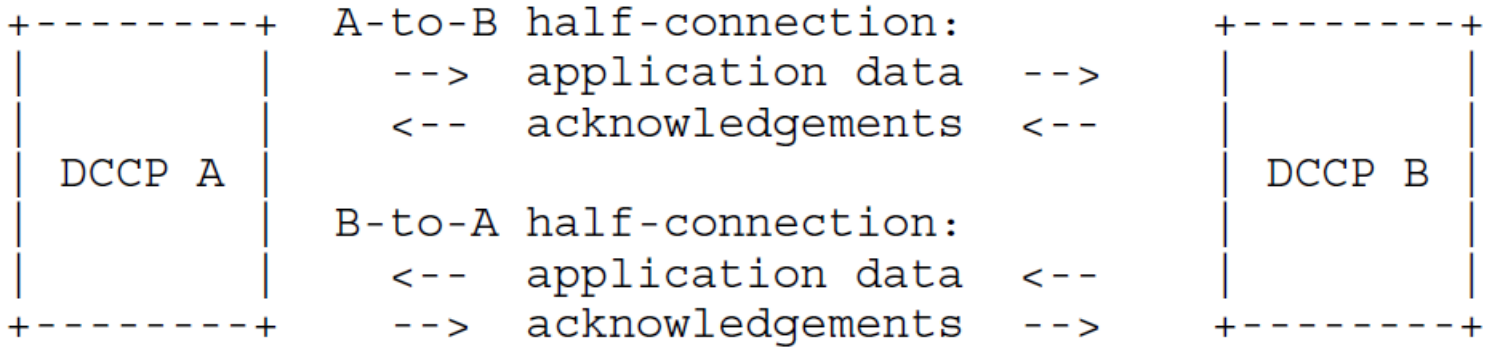
DCCP packet exchange overview:



School of Instrumentation, DAVV Ir

DCCP connection

- Each DCCP connection runs between two hosts, which we often name DCCP A and DCCP B.
- Each connection is actively initiated by one of the hosts, which we call the client; the other, initially passive host is called the server.
- The term "DCCP endpoint" is used to refer to either of the two hosts explicitly named by the connection (the client and the server).
- DCCP connections are bidirectional: **data may pass from either endpoint to the other**. This means that data and acknowledgements may flow in both directions simultaneously.
- Logically, however, a DCCP connection consists of two separate unidirectional connections, called half-connections.
- Each half-connection consists of the application data sent by one endpoint and the corresponding acknowledgements sent by the other endpoint.



Note: In the context of a single half-connection, the terms "HC-Sender" and "HC-Receiver" denote the endpoints sending application data and acknowledgements, respectively.

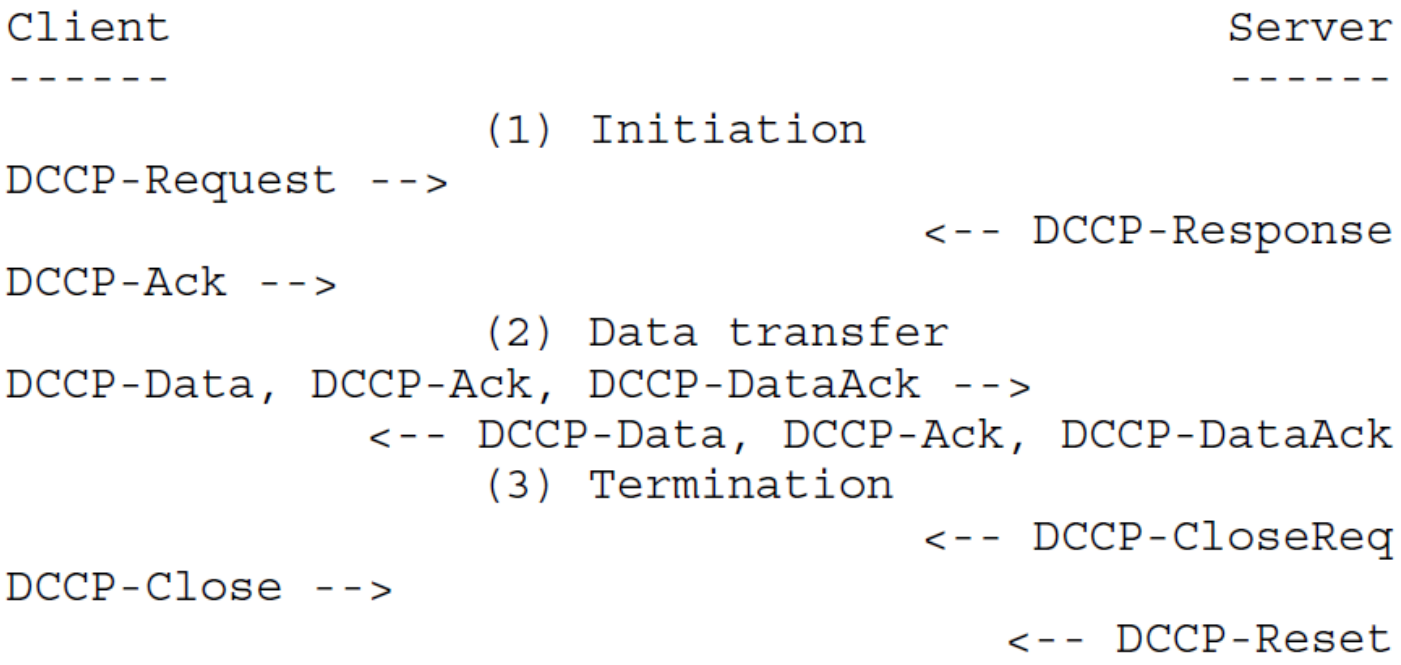
DCCP Features

- **A DCCP feature is a connection attribute** on whose value the two endpoints agree.
- Many properties of a DCCP connection are controlled by features, including the congestion control mechanisms in use on the two half-connections.
- The endpoints achieve agreement through the exchange of feature negotiation options in DCCP headers.
- **DCCP features are identified by a feature number and an endpoint**.
- The notation "F/X" represents the feature with feature number F located at DCCP endpoint X.

School of Instrumentation, DAVV Indore

DCCP Packet Types

- **Ten packet types implement DCCP's protocol functions.**
- For example, every new connection attempt begins with a DCCP-Request packet sent by the client.
- In this way a DCCP-Request packet resembles a TCP SYN, but since DCCP-Request is a packet type there is no way to send an unexpected flag combination, such as TCP's SYN+FIN+ACK+RST.
- Eight packet types occur during the progress of a typical connection. The two remaining packet types are used to resynchronize after bursts of loss.
- Note the three-way handshakes during initiation and termination.



- Every DCCP packet starts with a fixed-size generic header.
- Particular packet types include additional fixed-size header data; for example, DCCP-Acks include an Acknowledgement Number.
- DCCP options and any application data follow the fixed-size header.

DCCP-Request
- Sent by the client to initiate a connection (the first part of the three-way initiation handshake).

DCCP-Response
- Sent by the server in response to a DCCP-Request (the second part of the three-way initiation handshake).

DCCP-Data
- Used to transmit application data.

DCCP-Ack
- Used to transmit pure acknowledgements.

DCCP-DataAck
- Used to transmit application data with piggybacked acknowledgement information.

DCCP-CloseReq
- Sent by the server to request that the client close the connection.

DCCP-Close
- Used by the client or the server to close the connection; elicits a DCCP-Reset in response.

DCCP-Reset
- Used to terminate the connection, either normally or abnormally.

DCCP-Sync, DCCP-SyncAck
- Used to resynchronize sequence numbers after large bursts of loss.

School of Instrumentation, DAVV Indore

Packet Sequencing

- Each DCCP packet carries a sequence number so that losses can be detected and reported. Unlike TCP sequence numbers, which are byte based, **DCCP sequence numbers increment by one per packet**.
- **Every DCCP packet increments the sequence number, whether or not it contains application data.**
- DCCP-Ack pure acknowledgements increment the sequence number.

```
DCCP A                                              DCCP B
------                                              ------
DCCP-Data(seqno 1) -->
DCCP-Data(seqno 2) -->
                        <-- DCCP-Ack(seqno 10, ackno 2)
DCCP-DataAck(seqno 3, ackno 10) -->
                                  <-- DCCP-Data(seqno 11)
```

- DCCP B's second packet above uses sequence number 11, since sequence number 10 was used for an acknowledgement.
- This lets endpoints detect all packet loss, including acknowledgement loss. It also means that endpoints can get out of sync after long bursts of loss.
- The DCCP-Sync and DCCPSyncAck packet types are used to recover.

Congestion Control Mechanisms

- DCCP connections are congestion controlled, but unlike in TCP, **DCCP applications have a choice of congestion control mechanism.**
- In fact, the two half-connections can be governed by different mechanisms.
- Mechanisms are denoted by one-byte congestion control identifiers, or **CCIDs**.
- **The endpoints negotiate their CCIDs during connection initiation**.
- Each CCID describes how the HC-Sender limits data packet rates, how the HC-Receiver sends congestion feedback via acknowledgements, and so forth.
- **CCIDs 2 and 3 are currently defined;**
- CCIDs 0, 1, and 4-255 are reserved. Other CCIDs may be defined in the future.

- **CCID 2** provides TCP-like Congestion Control, which is similar to that of TCP.
  - The sender maintains a congestion window and sends packets until that window is full. Packets are acknowledged by the receiver.
  - Dropped packets and ECN indicate congestion; the response to congestion is to halve the congestion window.
  - Acknowledgements in CCID 2 contain the sequence numbers of all received packets within some window, similar to a selective acknowledgement (SACK).

School of Instrumentation, DAVV Indore

Congestion Control Mechanisms

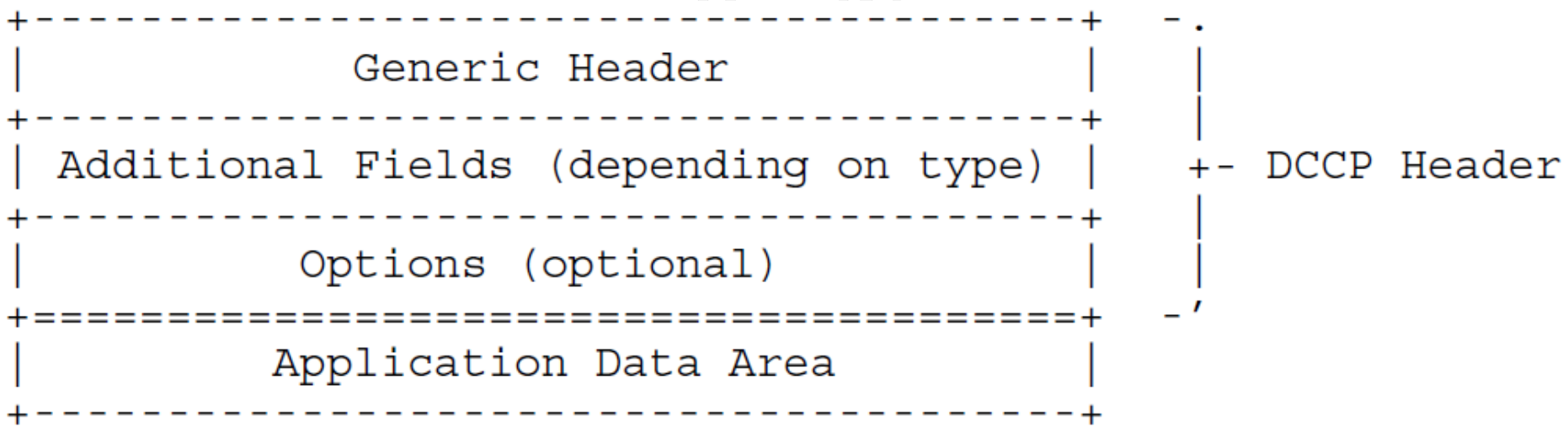
- **CCID 3** provides **TCP-Friendly Rate Control (TFRC)**, an equation-based form of congestion control intended to respond to congestion more smoothly than CCID 2.

  - Instead of a congestion window, a **TFRC sender uses or maintains a sending/transmit rate**.
  - The receiver sends feedback to the sender roughly once per round-trip time reporting the loss event rate it is currently observing.
  - The sender uses this loss event rate to determine its sending rate; if no feedback is received for several round-trip times, the sender halves its rate.
  - Thus, the **sender updates its transmit rate using the receiver's estimate of the packet loss and mark rate.**

School of Instrumentation, DAVV Indore

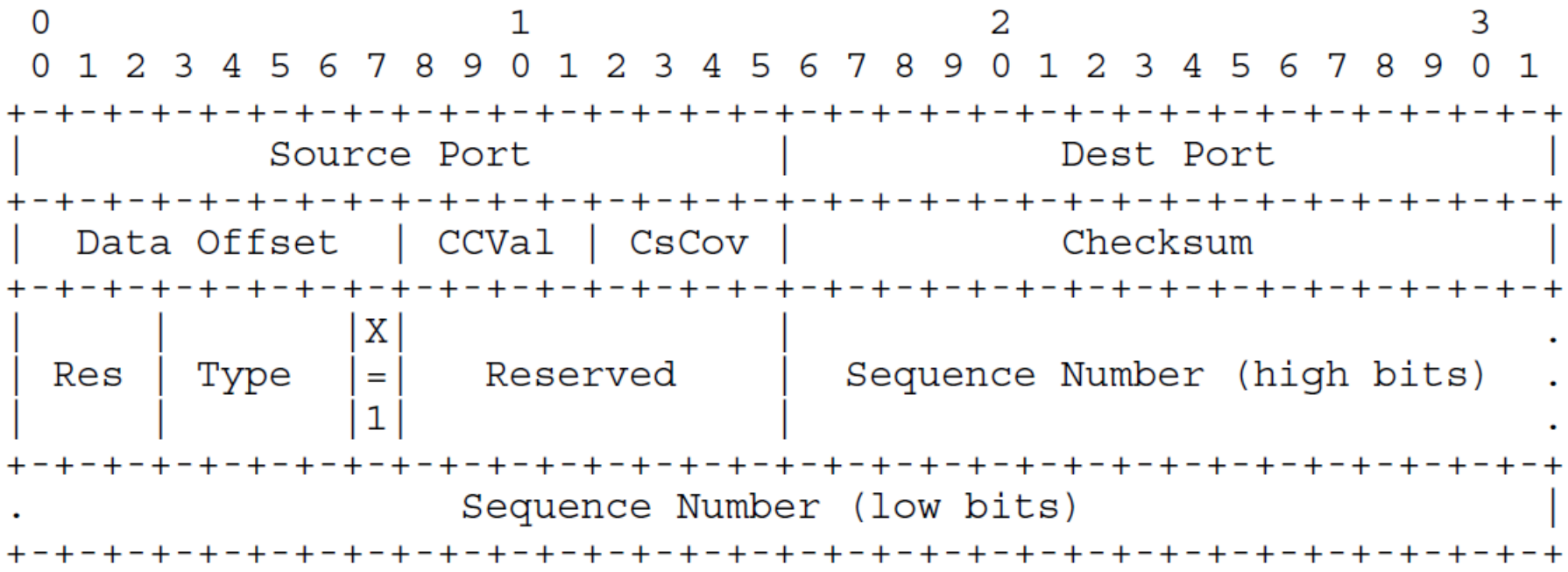
Packet Formats

- The **DCCP header can be from 12 to 1020 bytes long**. The initial part of the header has the same semantics for all currently defined packet types.
- Following this comes any additional fixed-length fields required by the packet type, and then a variable-length list of options.
- The application data area follows the header. In some packet types, this area contains data for the application; in other packet types, its contents are ignored.



Generic Header

- The DCCP generic header takes different forms depending on the value of X, the Extended Sequence Numbers bit.
- If X is one, the Sequence Number field is 48 bits long, and the generic header takes 16 bytes, as follows.



- If X is zero, only the low 24 bits of the Sequence Number are transmitted, and the generic header is 12 bytes long.

Source and Destination Ports: 16 bits each

- These fields identify the connection, similar to the corresponding fields in TCP and UDP. The Source Port represents the relevant
- port on the endpoint that sent this packet, and the Destination Port represents the relevant port on the other endpoint.
- When initiating a connection, the client SHOULD choose its Source Port randomly to reduce the likelihood of attack.
- DCCP APIs should treat port numbers similarly to TCP and UDP port numbers.
- For example, machines that distinguish between "privileged" and "unprivileged" ports for TCP and UDP should do the same for DCCP.

Data Offset: 8 bits

- The offset from the start of the packet's DCCP header to the start of its application data area, in 32-bit words.
- The receiver MUST ignore packets whose Data Offset is smaller than the minimum-sized header for the given Type or larger than the DCCP packet itself.

CCVal: 4 bits

- Used by the HC-Sender CCID. For example, the A-to-B CCID's sender, which is active at DCCP A, may send 4 bits of information per packet to its receiver by encoding that information in CCVal.
- The sender MUST set CCVal to zero unless its HC-Sender CCID specifies otherwise, and the receiver MUST ignore the CCVal field unless its HC-Receiver CCID specifies otherwise.

School of Instrumentation
DAVV Indore

Checksum Coverage (CsCov): 4 bits

- Checksum Coverage determines the parts of the packet that are covered by the Checksum field.
- This always includes the DCCP header and options, but some or all of the application data may be excluded.

Checksum: 16 bits

- The Internet checksum of the packet's DCCP header (including options), a network-layer pseudoheader, and, depending on Checksum Coverage, all, some, or none of the application data.

Reserved (Res): 3 bits

- Senders MUST set this field to all zeroes on generated packets, and receivers MUST ignore its value.

Type: 4 bits

- The Type field specifies the type of the packet.

Extended Sequence Numbers (X): 1 bit

- Set to one to indicate the use of an extended generic header with 48-bit Sequence and Acknowledgement Numbers.
- DCCP-Data, DCCPDataAck, and DCCP-Ack packets may set X to zero or one.
- All DCCP-Request, DCCP-Response, DCCP-CloseReq, DCCP-Close, DCCPReset, DCCP-Sync, and DCCP-SyncAck packets MUST set X to one; endpoints MUST ignore any such packets with X set to zero.

School of Instrumentation, DAVV Indore

Sequence Number: 48 or 24 bits

- Identifies the packet uniquely in the sequence of all packets the source sent on this connection. Sequence Number increases by one with every packet sent, including packets such as DCCP-Ack that carry no application data.
- When X=0, only the low 24 bits of the Acknowledgement Number are transmitted.

Reserved: 16 or 8 bits

- Senders MUST set this field to all zeroes on generated packets, and receivers MUST ignore its value.

Acknowledgement Number: 48 or 24 bits

- Generally contains GSR, the Greatest Sequence Number Received on any acknowledgeable packet so far.
- A packet is acknowledgeable if and only if its header was successfully processed by the receiver; Options such as Ack Vector combine with the Acknowledgement Number to provide precise information about which packets have arrived.

School of Instrumentation, DAVV Indore

SCTP: Stream Control Transmission Protocol

The limitations that users have wished to bypass include the following and this becomes the motivation of SCTP:

- **TCP provides both reliable data transfer and strict order-of- transmission delivery of data**. Some applications need reliable transfer without sequence maintenance, while others would be satisfied with partial ordering of the data. In both of these cases, the head-of-line blocking offered by TCP causes unnecessary delay.

- The stream-oriented nature of TCP is often an inconvenience. Applications must add their own record marking to delineate their messages, and must make explicit use of the push facility to ensure that a complete message is transferred in a reasonable time.

- The limited scope of TCP sockets complicates the task of providing highly-available data transfer capability using multi-homed hosts.

- TCP is relatively vulnerable to denial-of-service attacks, such as SYN attacks.

School of Instrumentation
DAVV Indore

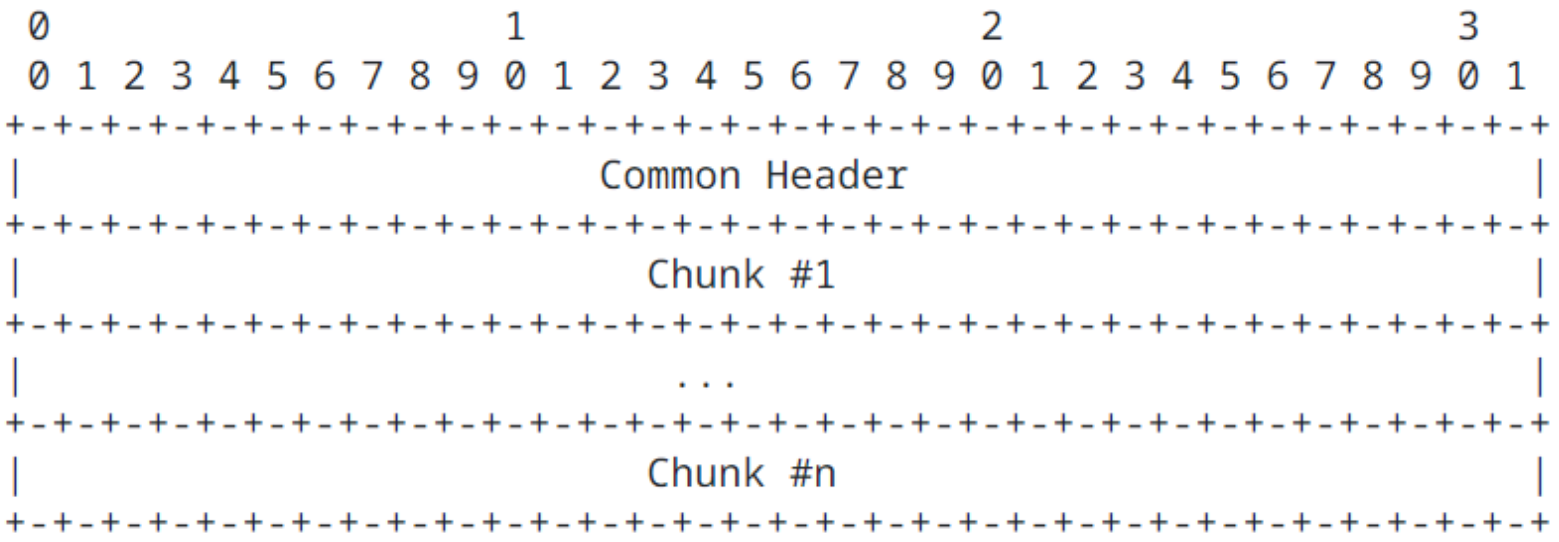
Architectural View of SCTP

- SCTP is viewed as a **layer between the SCTP user application ("SCTP user" for short) and a connectionless packet network service such as IP**.
- **SCTP is connection-oriented in nature**, but the SCTP association is a broader concept than the TCP connection.
- SCTP provides the means for each SCTP endpoint to provide the other endpoint (during association startup) with a list of transport addresses (i.e., **multiple IP addresses in combination with an SCTP port**) through which that endpoint can be reached and from which it will originate SCTP packets.

School of Instrumentation,
DAVV Indore

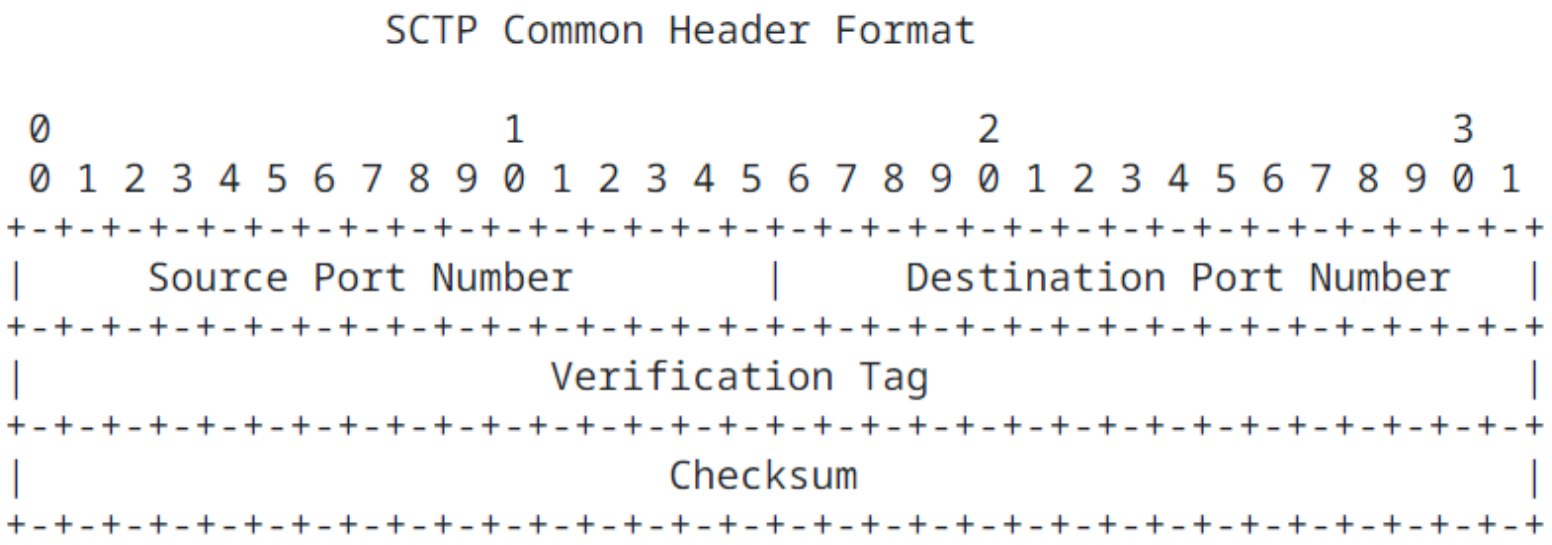
SCTP Packet Format

An SCTP packet is composed of a common header and chunks. A chunk contains either control information or user data.



- Multiple chunks can be bundled into one SCTP packet up to the MTU size, except for the INIT, INIT ACK, and SHUTDOWN COMPLETE chunks. These chunks MUST NOT be bundled with any other chunk in a packet.
- If a user data message doesn't fit into one SCTP packet it can be fragmented into multiple chunks

SCTP Common Header



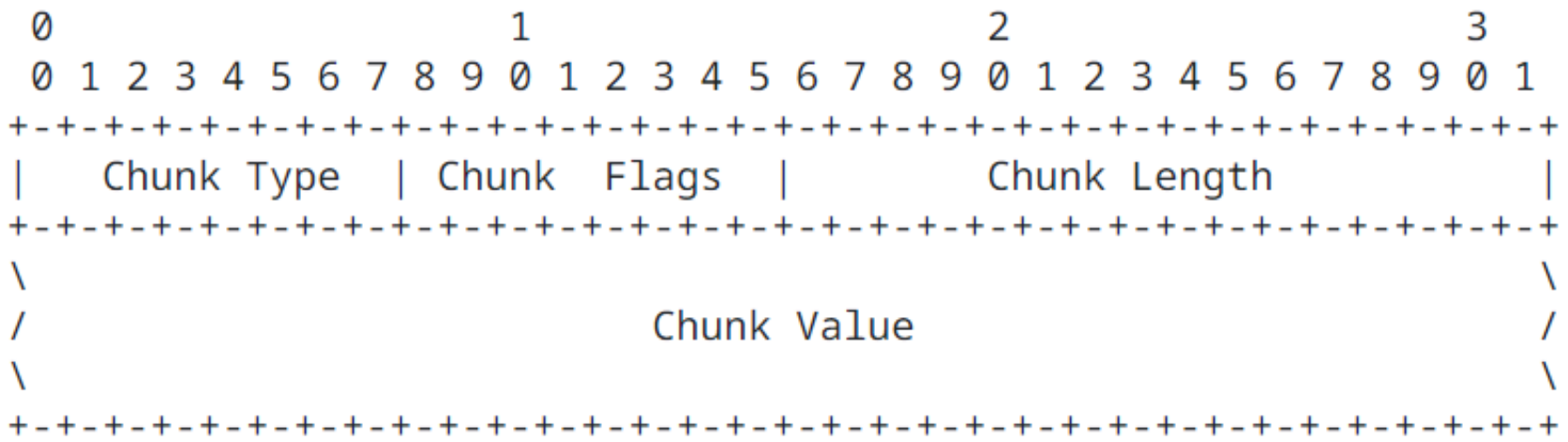
Source Port Number: 16 bits (unsigned integer)
This is the SCTP sender's port number. It can be used by the receiver in combination with the source IP address, the SCTP destination port, and possibly the destination IP address to identify the association to which this packet belongs.

Destination Port Number: 16 bits (unsigned integer)
This is the SCTP port number to which this packet is destined.
The receiving host will use this port number to de-multiplex the SCTP packet to the correct receiving endpoint/application.

Verification Tag: 32 bits (unsigned integer)
The receiver of this packet uses the Verification Tag to validate the sender of this SCTP packet.
On transmit, the value of this Verification Tag MUST be set to the value of the Initiate Tag received from the peer endpoint during the association initialization.

Checksum: 32 bits (unsigned integer)
This field contains the checksum of this SCTP packet.

Chunk Field Descriptions
The figure below illustrates the field format for the chunks to be transmitted in the SCTP packet. Each chunk is formatted with a Chunk Type field, a chunk-specific Flag field, a Chunk Length field, and a Value field.



Chunk Type: 8 bits (unsigned integer)

This field identifies the type of information contained in the Chunk Value field. It takes a value from 0 to 254. The value of 255 is reserved for future use as an extension field.

ID Value	Chunk Type
0	- Payload Data (DATA)
1	- Initiation (INIT)
2	- Initiation Acknowledgement (INIT ACK)
3	- Selective Acknowledgement (SACK)
4	- Heartbeat Request (HEARTBEAT)
5	- Heartbeat Acknowledgement (HEARTBEAT ACK)
6	- Abort (ABORT)
7	- Shutdown (SHUTDOWN)
8	- Shutdown Acknowledgement (SHUTDOWN ACK)
9	- Operation Error (ERROR)
10	- State Cookie (COOKIE ECHO)
11	- Cookie Acknowledgement (COOKIE ACK)
12	- Reserved for Explicit Congestion Notification Echo (ECNE)
13	- Reserved for Congestion Window Reduced (CWR)
14	- Shutdown Complete (SHUTDOWN COMPLETE)
15 to 62	- available
63	- reserved for IETF-defined Chunk Extensions
64 to 126	- available
127	- reserved for IETF-defined Chunk Extensions
128 to 190	- available
191	- reserved for IETF-defined Chunk Extensions
192 to 254	- available
255	- reserved for IETF-defined Chunk Extensions

Heartbeat Request (HEARTBEAT): An endpoint should send this chunk to its peer endpoint to probe the reachability of a particular destination transport address defined in the present association.

Chunk Flags: 8 bits

The usage of these bits depends on the Chunk type as given by the Chunk Type field. Unless otherwise specified, they are set to 0 on transmit and are ignored on receipt.

Chunk Length: 16 bits (unsigned integer)

- This value represents the size of the chunk in bytes, including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields.
- Therefore, if the Chunk Value field is zero-length, the Length field will be set to 4. The Chunk Length field does not count any chunk padding.
- Chunks (including Type, Length, and Value fields) are padded out by the sender with all zero bytes to be a multiple of 4 bytes long.

Chunk Value: variable length

- The Chunk Value field contains the actual information to be transferred in the chunk.
- The usage and format of this field is dependent on the Chunk Type.
- The total length of a chunk (including Type, Length, and Value fields) MUST be a multiple of 4 bytes.
- If the length of the chunk is not a multiple of 4 bytes, the sender MUST pad the chunk with all zero bytes, and this padding is not included in the Chunk Length field.
- The receiver MUST ignore the padding bytes.

School of Instrumentation, DAVV Indore

DTLS: Datagram Transport Layer Security

- The DTLS protocol provides communications privacy for datagram protocols.
- The protocol allows client/server applications to communicate while preventing eavesdropping, tampering, or message forgery.
- The DTLS protocol is based on the Transport Layer Security (TLS) protocol and provides equivalent security guarantees.

Why to use DTLS over TLS:

- TLS must run over a reliable transport channel -- typically TCP. Therefore, it cannot be used to secure unreliable datagram traffic.
- An increasing number of application layer protocols have been designed that use UDP transport.
- The requirement for datagram semantics automatically prohibits use of TLS.

- DTLS is deliberately designed to be as similar to TLS as possible, both to minimize new security invention and to maximize the amount of code and infrastructure reuse.

School of Instrumentation, DAVV Indore

Overview of DTLS

- The basic design philosophy of DTLS is to **construct "TLS over datagram transport**". The reason that TLS cannot be used directly in datagram environments is simply that packets may be lost or reordered.
- TLS has no internal facilities to handle this kind of unreliability; therefore, TLS implementations break when rehosted on datagram transport.
- The purpose of DTLS is to make only the minimal changes to TLS required to fix this problem.

Unreliability creates problems for TLS at two levels:

1. TLS does not allow independent decryption of individual records. Because the integrity check depends on the sequence number, if record N is not received, then the integrity check on record N+1 will be based on the wrong sequence number and thus will fail.
2. The TLS handshake layer assumes that handshake messages are delivered reliably and breaks if those messages are lost.

School of Instrumentation DAVV Indore

Approaches used by DTLS to solve these problems:

1. Loss-Insensitive Messaging:

- In TLS's traffic encryption layer (called the TLS Record Layer), records are not independent.
- There are two kinds of inter-record dependency:

1) Cryptographic context (stream cipher key stream) is retained between records.

1) Anti-replay and message reordering protection are provided by a MAC that includes a sequence number, but the sequence numbers are implicit in the records.

- DTLS solves the first problem by banning stream ciphers.
- DTLS solves the second problem by adding explicit sequence numbers.

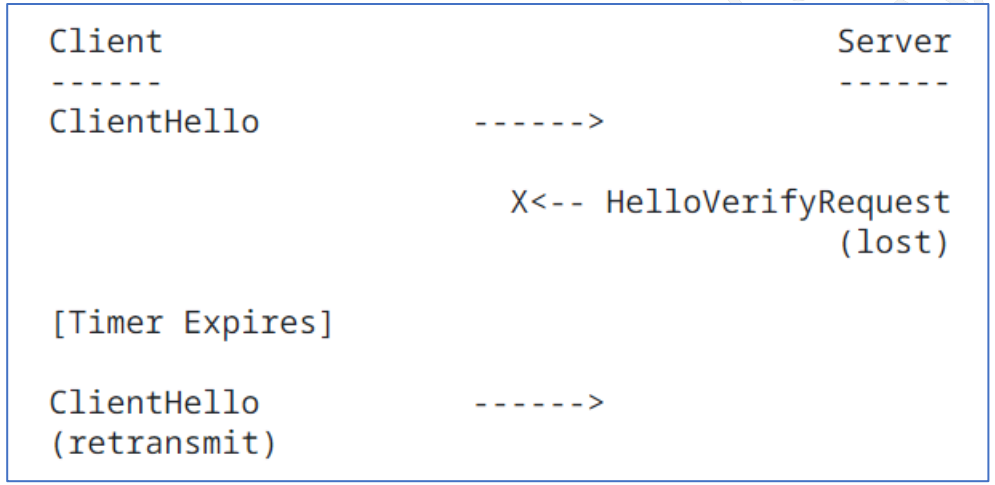
School of Instrumentation, DAVV Indore

2. Providing Reliability for Handshake

- The TLS handshake is a lockstep cryptographic handshake.
- Messages must be transmitted and received in a defined order; any other order is an error. Clearly, this is incompatible with reordering and message loss.
- TLS handshake messages are potentially larger than any given datagram, thus creating the problem of IP fragmentation.
- DTLS must provide fixes for both of these problems.

➢ Packet Loss

- DTLS uses a simple retransmission timer to handle packet loss.



➢ Reordering

- In DTLS, each handshake message is assigned a specific sequence number within that handshake.
- When a peer receives a handshake message, it can quickly determine whether that message is the next message it expects. If it is, then it processes it. If not, it queues it for future handling once all previous messages have been received.

3. Replay Detection

- DTLS optionally supports record replay detection, by maintaining a bitmap window of received records.
- Records that are too old to fit in the window and records that have previously been received are silently discarded.
- The replay detection feature is optional, since packet duplication is not always malicious, but can also occur due to routing errors.
- Applications may conceivably detect duplicate packets and accordingly modify their data transmission strategy.

School of Instrumentation, DAVV Indore

The DTLS Handshake Protocol

DTLS uses all of the same handshake messages and flows as TLS, with three principal changes:

1. A stateless cookie exchange has been added to prevent denial-of-service attacks.
2. Modifications to the handshake header to handle message loss, reordering, and DTLS message fragmentation (in order to avoid IP fragmentation).
3. Retransmission timers to handle message loss.

School of Instrumentation,
DAVV Indore

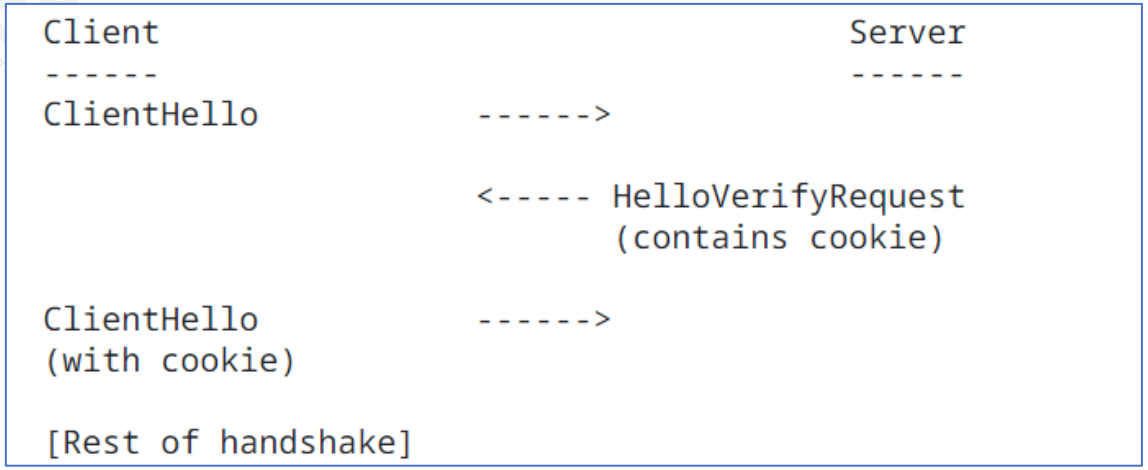
Denial-of-Service Countermeasures

- Datagram security protocols are extremely susceptible to a variety of DoS attacks.
- Two attacks are of particular concern:

1. An attacker can consume excessive resources on the server by transmitting a series of handshake initiation requests, causing the server to allocate state and potentially to perform expensive cryptographic operations.

2. An attacker can use the server as an amplifier by sending connection initiation messages with a forged source of the victim. The server then sends its next message (in DTLS, a Certificate message, which can be quite large) to the victim machine, thus flooding it.

- In order to counter both of these attacks, DTLS borrows the stateless cookie technique. When the client sends its ClientHello message to the server, the server MAY respond with a HelloVerifyRequest message. This message contains a stateless cookie.
- The client MUST retransmit the ClientHello with the cookie added. The server then verifies the cookie and proceeds with the handshake only if it is valid. This mechanism forces the attacker/client to be able to receive the cookie, which makes DoS attacks with spoofed IP addresses difficult.



School of Instrumentation, DAVV, Indore

References

- Andrew Tanenbaum (Author), David Wetherall, “**Computer Networks**”.
- "**Using multipath TCP and opportunistic routing in IoT network**," 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), India, 2022, pp. 94-104, doi: 10.1109/ICSSIT53264.2022.9716336.
- **Multipath TCP: Decoupled from IP, TCP is at last able to support multihomed hosts**, Volume 12Issue 2pp 40–51https://doi.org/10.1145/2578508.2591369.
- Ji, R.; Cao, Y.; Fan, X.; Jiang, Y.; Lei, G.; Ma, Y., **Multipath TCP-Based IoT Communication Evaluation: From the Perspective of Multipath Management with Machine Learning**. Sensors 2020, 20, 6573. https://doi.org/10.3390/s20226573
- **Datagram Congestion Control Protocol (DCCP),** https://datatracker.ietf.org/doc/html/rfc4340.
- **Designing DCCP: congestion control without reliability**, ACM SIGCOMM Computer Communication ReviewVolume 36Issue 4pp 27–38https://doi.org/10.1145/1151659.1159918.
- Datagram Transport Layer Security Version 1.2, https://datatracker.ietf.org/doc/html/rfc6347#page-15.
- Stream Control Transmission Protocol, https://datatracker.ietf.org/doc/html/rfc4960

School of Instrumentation DAVV Indore