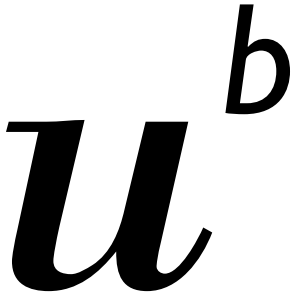


b

UNIVERSITÄT
BERN

UNIVERSITÄT BERN

Network Security

X. Transport Level Security

Prof. Dr. Torsten Braun, Institut für Informatik

Bern, 02.05.2022 – 09.05.2022

UNIVERSITÄT BERN

Transport Level Security

Table of Contents

1. Introduction

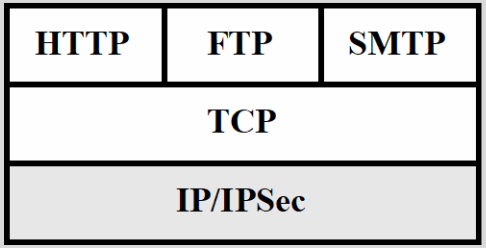
1. Threats on the Web

	Threats	Consequences	Countermeasures
Integrity	•Modification of user data •Trojan horse browser •Modification of memory •Modification of message traffic in transit	•Loss of information •Compromise of machine •Vulnerabilty to all other threats	Cryptographic checksums
Confidentiality	•Eavesdropping on the net •Theft of info from server •Theft of data from client •Info about network configuration •Info about which client talks to server	•Loss of information •Loss of privacy	Encryption, Web proxies
Denial of Service	•Killing of user threads •Flooding machine with bogus requests •Filling up disk or memory •Isolating machine by DNS attacks	•Disruptive •Annoying •Prevent user from getting work done	Difficult to prevent
Authentication	•Impersonation of legitimate users •Data forgery	•Misrepresentation of user •Belief that false information is valid	Cryptographic techniques

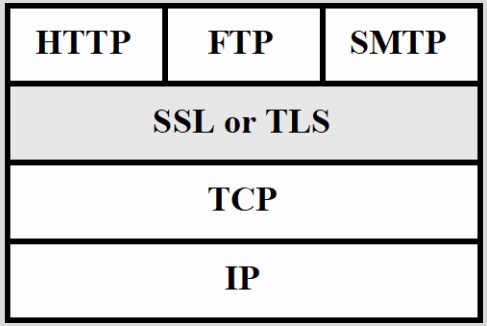
UNIVERSITÄT BERN

1. Introduction

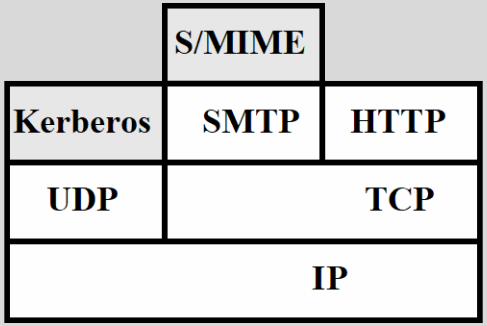
2. Web Traffic Security Approaches



(a) Network Level



(b) Transport Level

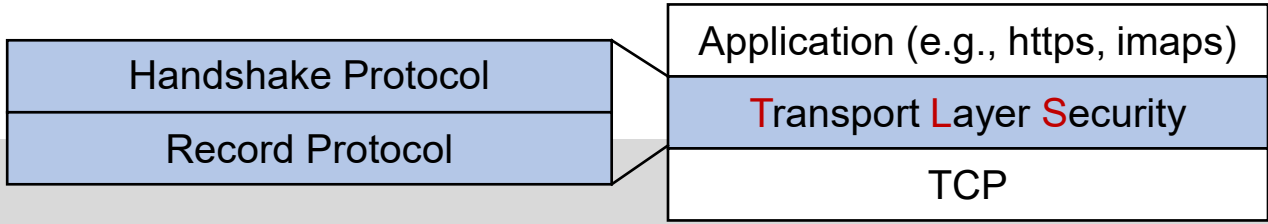


(c) Application Level

UNIVERSITÄT BERN

2. Transport Layer Security

1. Overview



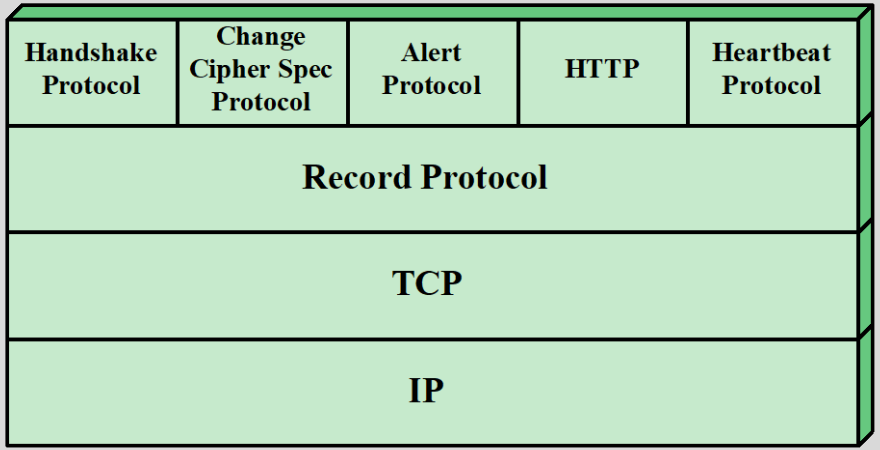
- Functions
  - Server and client authentication with public keys
  - Sessions encrypted with symmetric keys
- Handshake Protocol
  - Certificate exchange
  - Generation of a secret session key
- Record Protocol (for data exchange)
  - Fragmentation
  - Compression
  - Encryption

UNIVERSITÄT BERN

2. Transport Layer Security

2. Protocol Stack

- IETF RFC 8446
- TLS Record Protocol provides basic security services.
- Higher-layer protocols as part of TLS
  - Handshake Protocol
  - Change Cipher Spec Protocol
  - Alert Protocol
- Heartbeat Protocol is defined in a separate RFC.



UNIVERSITÄT BERN

2. Transport Layer Security

3. Connections and Session

Connections

- are transport-layer connections between two endpoints.
- are peer-to-peer relationships.
- are transient.
- Every connection is associated with one session.

Session

- is an association between a client and a server.
- is created by the Handshake Protocol.
- defines a set of cryptographic security parameters, which can be shared among multiple connections.
- is used to avoid expensive negotiation of new security parameters for each connection.

2. Transport Layer Security

3.1 Session State

- **Session identifier**: arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer certificate**: X.509 certificate of the peer
- **Compression method**: Algorithm used to compress data prior to encryption
- **Cipher Spec**
  - specifies bulk data encryption algorithm and hash algorithm (for MAC calculation)
  - defines cryptographic attributes such as hash size
- 48-byte **master secret** shared between client and server.
- **Resumable flag** indicates whether the session can be used to initiate new connections.

UNIVERSITÄT BERN

2. Transport Layer Security

3.2 Connection State

- **Server and Client random**
  - Byte sequences that are chosen by server and client for each connection.
- **Server (Client) write MAC secret**
  - Secret key used in MAC operations on data sent by the server (client).
- **Server (Client) write key**
  - The secret encryption key for data encrypted by the server (client) and decrypted by the client (server)
- **Initialization Vectors**
  - When a block cipher in CBC mode is used, an IV is maintained for each key.
  - It is initialized by TLS Handshake Protocol.
  - The final ciphertext block from each record is preserved for use as IV with the following record.
- **Sequence Numbers**
  - Each party maintains separate sequence numbers for transmitted and received messages for each connection.
  - When a party sends or receives a “change cipher spec message”, appropriate sequence number is set to 0.
  - Sequence numbers may not exceed $2^{64} - 1$.

2. Transport Layer Security

4.1 Record Protocol Services

Confidentiality

Handshake Protocol defines a shared secret key that is used for conventional encryption of TLS payloads.

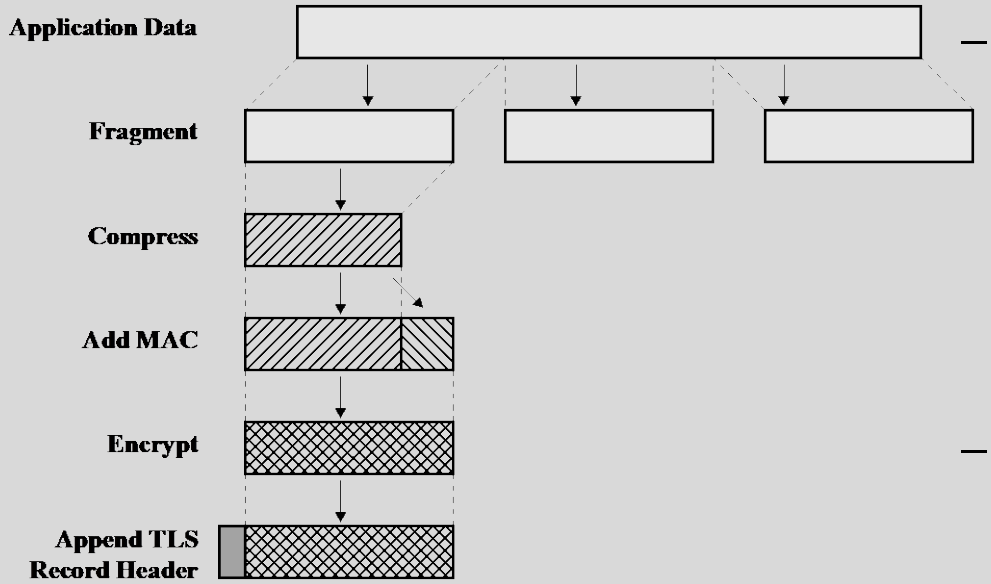
Message Integrity

Handshake Protocol also defines a shared secret key that is used to form a Message Authentication Code.

UNIVERSITÄT BERN

2. Transport Layer Security

4.2 Record Protocol Operation



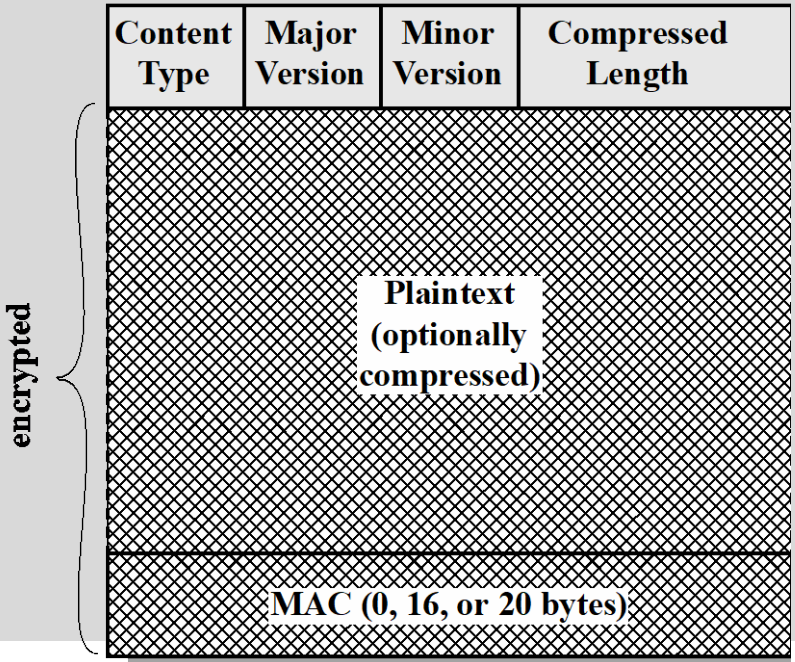
- HMAC algorithm (RFC 2104)
  - $HMAC_K(M) = H[(K^+ \oplus opad) \,||\, (K^+ \oplus ipad) \,||\, M]$
  - H: MD5 or SHA-1
  - M: message
  - K^+: secret key padded with 0's
  - ipad/opad: 36/5C repeated 64 times
- Encryption
  - Block Cipher (AES-128/256, 3DES)
  - Stream Cipher (RC4-128)

UNIVERSITÄT BERN

2. Transport Layer Security

4.3 Record Format

- **Content Type**
  - higher-layer protocol used to process the enclosed fragment.
- **Major/Minor Version Numbers**
- **Compressed Length (16 bits)**
  - length in bytes of the plaintext fragment (or compressed fragment if compression is used) < 2^{14} + 2048.

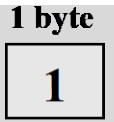


UNIVERSITÄT BERN

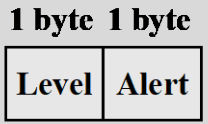
2. Transport Layer Security

5. Specific Protocols

- Change Cipher Spec Protocol
  - to copy pending state into current state and to update the cipher suite to be used on a connection
- Alert Protocol
  - to convey TLS-related alerts to the peer entity.
  - Level: warning or fatal (immediate connection termination)
  - Code to indicate alert



(a) Change Cipher Spec Protocol



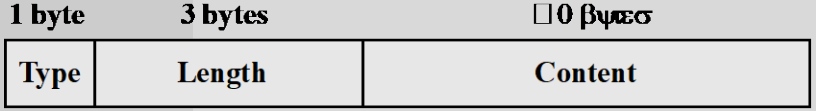
(b) Alert Protocol

UNIVERSITÄT BERN

2. Transport Layer Security

6.1 Handshake Protocol Messages

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

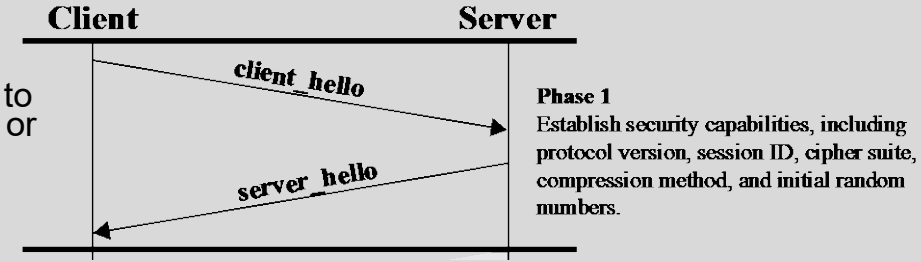


2. Transport Layer Security

6.2.1 Handshake Protocol Operation: Establish Security Capabilities (Phase 1)

client_hello message / server_hello message

- Version: highest version understood by client; lowest suggested by client and highest supported by server
- Random: A client(server)-generated random structure consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These serve as nonces during key exchange to prevent replay attacks.
- Session ID: A variable-length session identifier.
  - A non-zero value indicates that the client wishes to update the parameters of an existing connection or to create a new connection on this session.
  - A zero value indicates that the client wishes to establish a new connection on a new session.



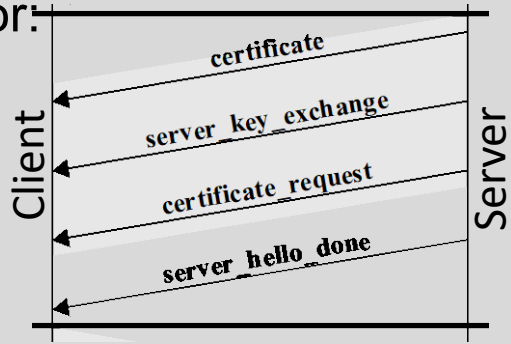
- CipherSuite: a list that contains the cryptographic algorithms supported by the client, each element defines a key exchange algorithm, and a Cipher Spec. Server selects Cipher Suite from that list.
- Compression Method: a list of compression methods the client supports. Server selects a method.

2. Transport Layer Security

6.2.2 Handshake Protocol Operation: Server Authentication and Key Exchange (Phase 2)

server_key_exchange message needed for parameter exchange, not needed if the server has sent a certificate with fixed DH parameters or RSA key exchange is to be used, but required for:

- Anonymous DH
  - 2 global DH values + server's public DH key
- Ephemeral DH
  - 3 DH parameters + signature of those
- RSA key exchange (if server has a signature-only RSA key)
  - Server creates temporary public/private RSA key pair.
  - **`server_key_exchange`** to send public key



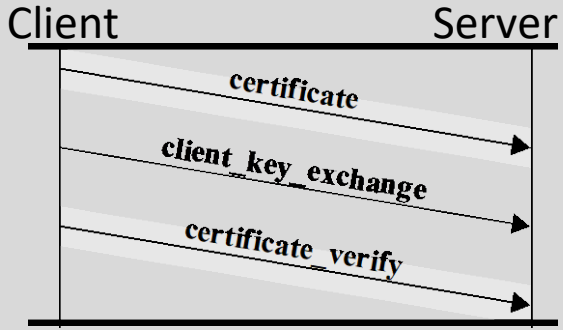
UNIVERSITÄT BERN

2. Transport Layer Security

6.2.3 Handshake Protocol Operation: Client Authentication and Key Exchange (Phase 3)

client_key_exchange

- RSA: 48-byte pre-master secret, encrypted using public server key or key from 2nd phase
- Ephemeral/Anonymous DH parameters

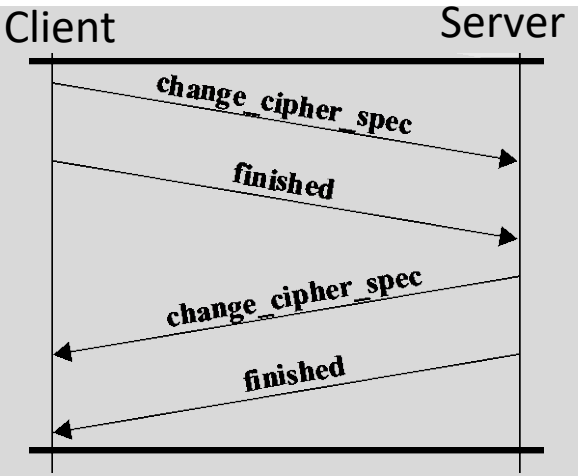


UNIVERSITÄT BERN

2. Transport Layer Security

6.2.4 Handshake Protocol Operation: Finish (Phase 4)

- Client
  - sends **`change_cipher_spec`** message
  - copies pending into current Cipher Spec
  - sends **`finished`** message to verify that key exchange and authentication processes were successful.
- Server
  - sends own **`change_cipher_spec`** message,
  - transfers pending to current Cipher Spec
  - sends its **`finished`** message.
- **`finished`** message
  - **`PRF (master_secret, finished_label, MD5/SHA-1 (handshake_messages))`**



UNIVERSITÄT BERN

2. Transport Layer Security

7.1 Master Secret Creation

1. Creation of a shared master secret
   - shared master secret: one-time 48-byte value generated for a session by secure key exchange
   - Stages
     1. Exchange of **`pre_master_secret`**
        - RSA: client generates **`pre_master_secret`**, encrypts using server's public RSA key, sends to server
        - DH: client and server generate DH public key for **`pre_master_secret`**
     2. Calculation of master_secret by both parties
        - **`master_secret = PRF(pre_master_secret, "master secret", ClientHello.Random||ServerHello.Random)`**
2. Generation of cryptographic parameters from master secret

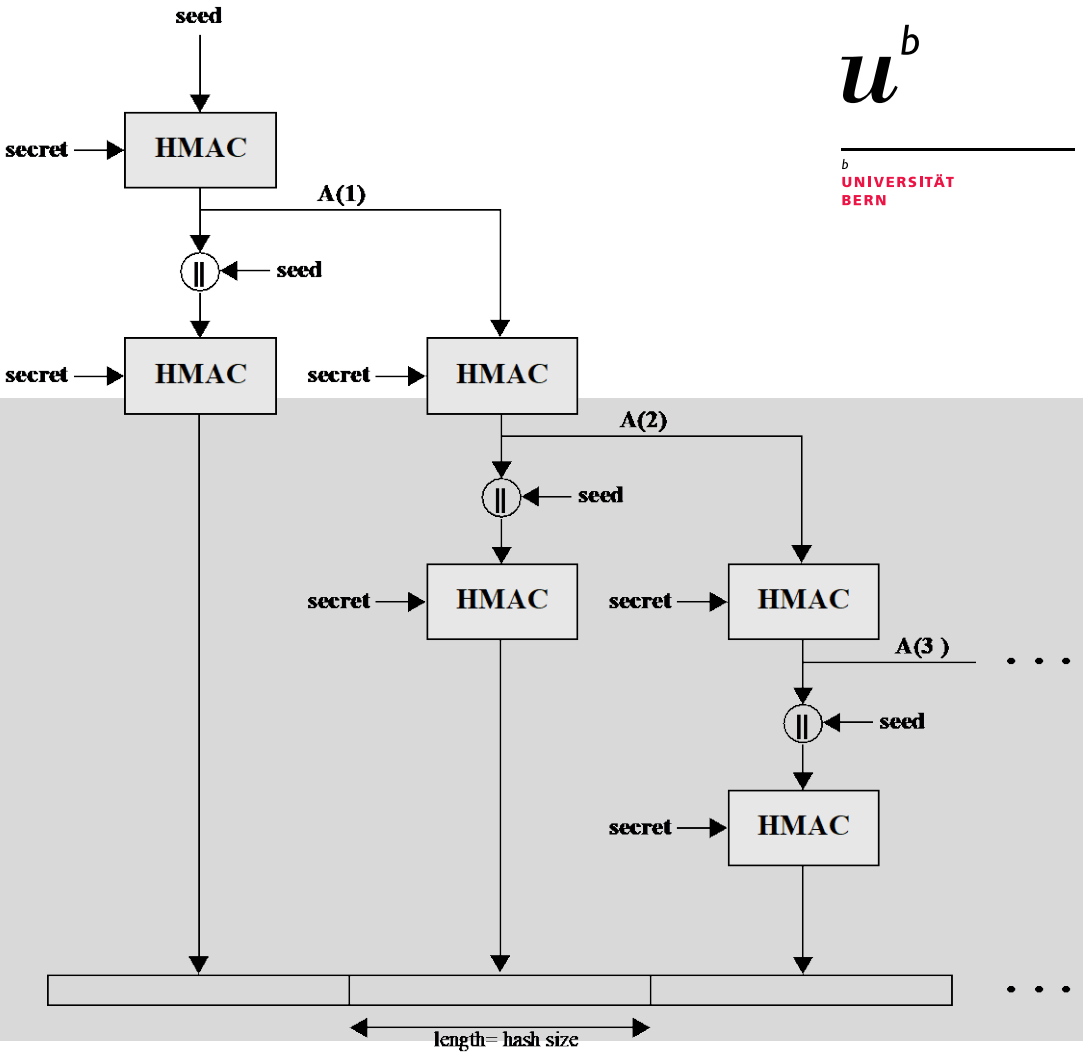
```
key_block == PRF(SecurityParameters.master_secret, "key expansion",
SecurityParameters.server_random || SecurityParameters.client_random)
```

2. TLS

7.2 Pseudo-Random Function

A(0) = seed

A(i) = HMAC_hash(secret, A(i−1))



UNIVERSITÄT BERN

2. Transport Layer Security

7.3 Generation of Cryptographic Parameters

CipherSpecs

- client write MAC secret
- server write MAC secret
- client write key
- server write key
- client write IV
- server write IV

Parameters are generated from master secret in that order by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters.

```
key_block =
MD5(master_secret   ||  SHA('A'   || master_secret||
 ServerHello.random ||  ClientHello.random))      ||
MD5(master_secret   ||  SHA('BB'  || master_secret||
 ServerHello.random ||  ClientHello.random))      ||
MD5(master_secret   ||  SHA('CCC' || master_secret||
 ServerHello.random ||  ClientHello.random))      ||
...
```

2. Transport Layer Security

8. Attacks

- Attacks on Handshake protocol
  - Example: compromising the handshake protocol based on exploiting the formatting and implementation of the RSA encryption scheme
- Attacks on Record and application data protocols
  - Example: Browser Exploit Against SSL/TLS leverages chosen-plaintext attack.
- Attacks on PKI: Checking the validity of X.509 certificates is an activity subject to a variety of attacks
  - Example: it was demonstrated that commonly used libraries for SSL/TLS suffer from vulnerable certificate validation implementations.

UNIVERSITÄT BERN

2. Transport Layer Security

9. TLSv1.3

TLSv1.3 removes support for several options and functions

- Compression
- Ciphers that do not offer authenticated encryption
- Static RSA and DH key exchange
- …

TLSv1.3 (RFC 8446)

- uses DH or Elliptic Curve DH for key exchange and does not permit RSA.
- encrypts all handshake messages after **`server_hello`**.
- allows for a “1 round trip time” handshake by changing the order of messages sent with establishing a secure connection.

UNIVERSITÄT BERN

3. Datagram Transport Layer Security

1. Overview

- DTLS is TLS with added features to deal with the unreliable nature of UDP communications
- as close as possible to TLS design
- IETF RFC 6347
- important for Constrained Application Protocol in IoT

Major changes

- Reliable transmission of handshakes using retransmission at the beginning of the communication to support authentication and key exchange.
- Explicit numbering of packets (records) to enable HMAC calculation in case of packet loss.
- Records must fit into a single datagram
- optional replay detection for single packets

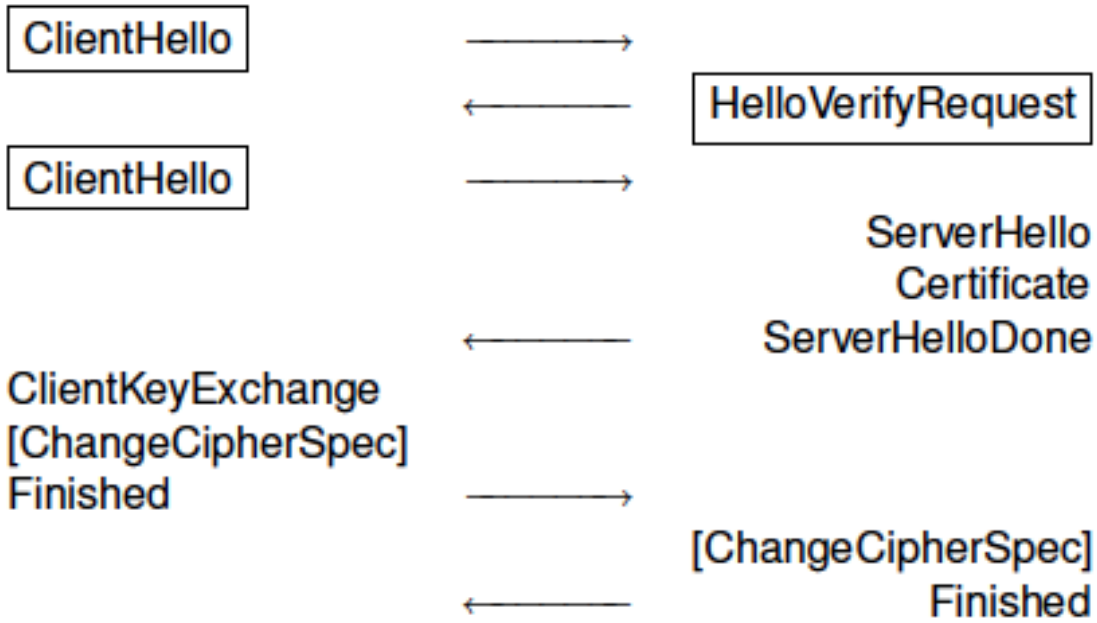
UNIVERSITÄT BERN

3. Datagram Transport Layer Security

2. RSA Handshake

Cookies for denial of service attack prevention:

Before the handshake begins, the client must replay a “cookie” provided by server in order to demonstrate that it is capable of receiving packets at its claimed IP address.



4. Hyper Text Transfer Protocol Secure

1. Overview

- secure version of HTTP encrypts all communications between browser and web server
- Data sent using HTTPS provides 3 important areas of protection:
  1. Encryption
  2. Data integrity
  3. Authentication

UNIVERSITÄT BERN

4. Hyper Text Transfer Protocol Secure

2. Connection Initiation

- Agent acting as HTTP client also acts as TLS client.
- Client initiates a connection to server on an appropriate port and sends TLS `client hello` to begin TLS handshake.
- After finishing TLS handshake, client may initiate first HTTP request.
- All HTTP data is sent as TLS application data.

Connection levels in HTTPS

1. HTTP
   - HTTP client requests a connection to HTTP server by sending a connection request to the next lowest layer.
2. TLS
   - A session is established between TLS client and TLS server.
3. TCP
   - A TLS request to establish a connection begins with the establishment of a TCP connection between TCP client and server

UNIVERSITÄT BERN

4. Hyper Text Transfer Protocol Secure

3. Connection Closure

- An HTTP client or server indicates closing of a connection by including the following line in an HTTP record: **`Connection:close`**
- Closure of HTTPS connection requires that TLS closes the connection with the peer TLS entity, which will involve closing the underlying TCP connection.
- At TLS level, the proper way to close a connection is for each side to use Alert protocol to send a **`close_notify`** alert.
- HTTP clients also must be able to cope with situations when underlying TCP connection is terminated without **`close_notify`** alert and **`Connection:close`** indicator.
- Such situation could result from
  - programming error on the server or
  - communication error causing TCP connection to drop.
- Unannounced TCP closure could be evidence of some attack
  → HTTPS client should issue some sort of security warning when this occurs.

5. Secure Shell

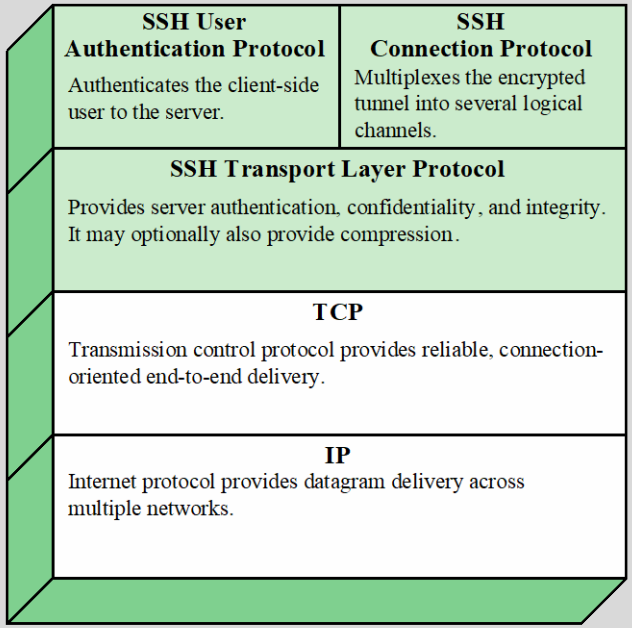
1. Overview

- Initial version to provide secure remote logon facility to replace TELNET and other remote logon schemes
- SSH also provides a more general client/server capability and can be used for such applications as file transfer and e-mail.
- SSH client and server applications are widely available for most operating systems.
- IETF RFCs 4250-4256

UNIVERSITÄT BERN

5. Secure Shell

2. Protocol Stack



- Transport Layer Protocol
  - provides server authentication, data confidentiality, and data integrity with forward secrecy, i.e., if a key is compromised during one session, the knowledge does not affect the security of earlier sessions.
  - may optionally provide compression.
- User Authentication Protocol
  - authenticates client/user to server.
- Connection Protocol
  - multiplexes multiple logical communication channels over a single, underlying SSH connection.

5. Secure Shell

3. Transport Layer Protocol

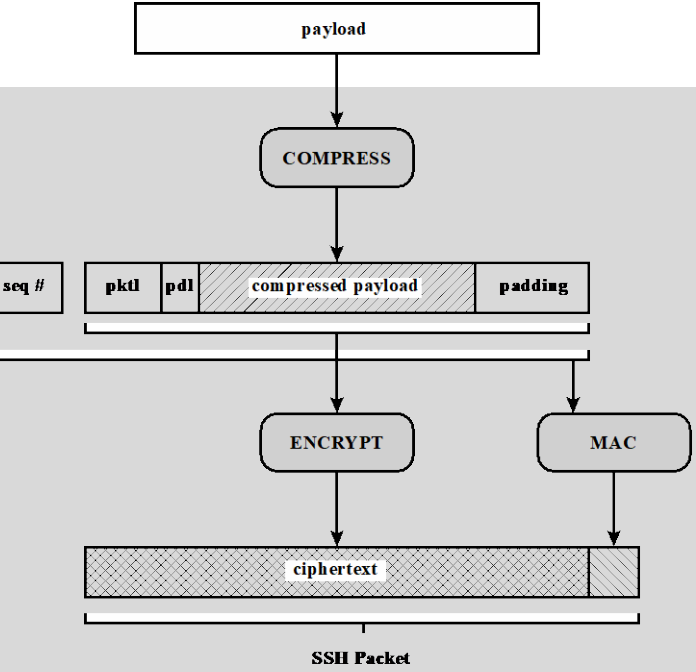
- Server authentication occurs at transport layer, based on the server possessing a public/private key pair.
- A server may have multiple host keys using multiple different asymmetric encryption algorithms.
- Multiple hosts may share the same host key.
- Server host key is used during key exchange to authenticate the identity of the host.

RFC 4251 dictates 2 alternative trust models

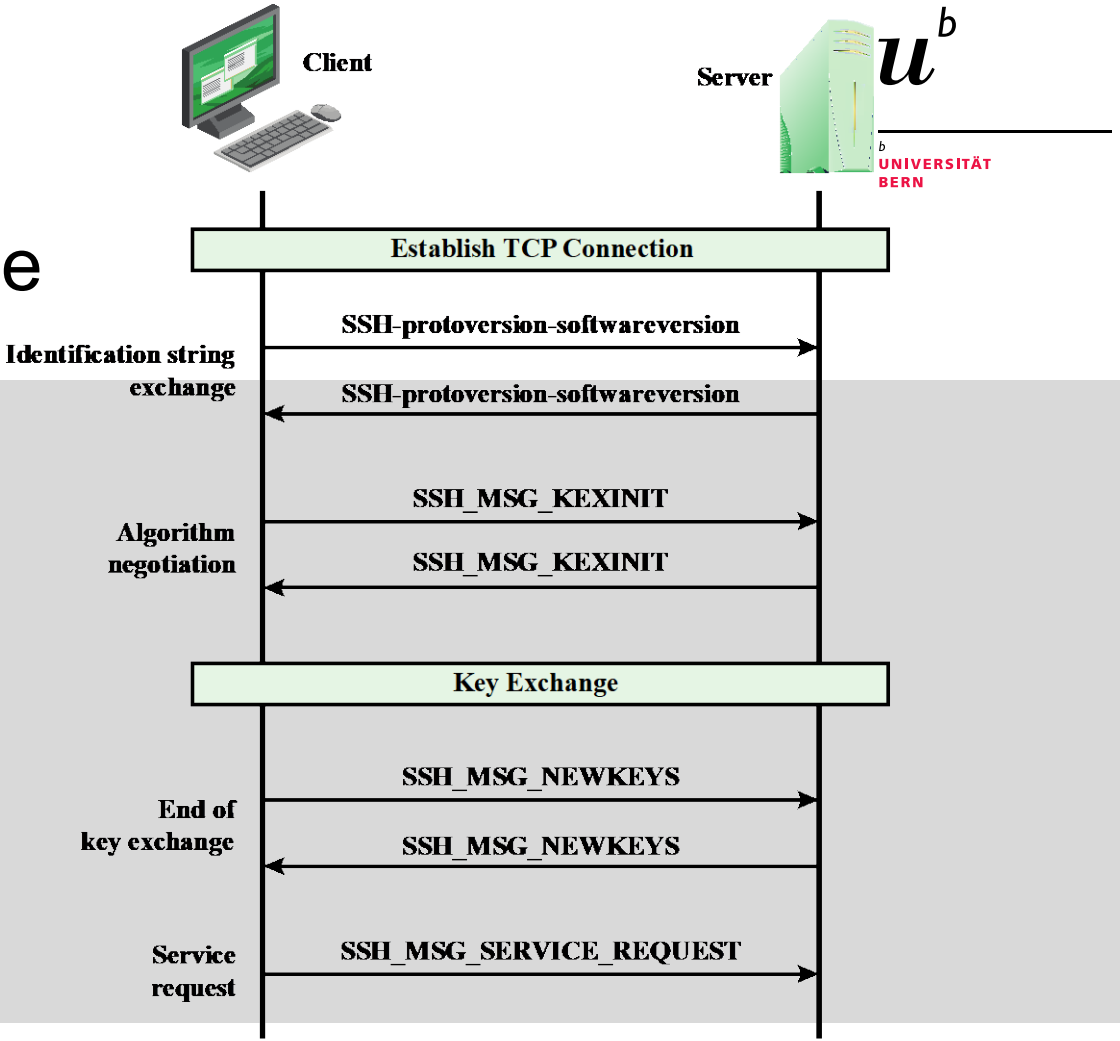
1. The client has a local database that associates each host name with the corresponding public host key
2. The host name-to-key association is certified by a trusted certification authority (CA); the client only knows the CA root key and can verify the validity of all host keys certified by accepted CAs

5. Secure Shell

3.1 Packet Exchange



pktl = packet length
pdl = padding length



5. Secure Shell

3.2 Cryptographic Algorithms

Cipher	
3des-cbc*	Three-key 3DES in CBC mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	AES in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC mode

MAC algorithm	
hmac-sha1*	HMAC-SHA1; digest length = key length = 20
hmac-sha1-96**	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20
hmac-md5	HMAC-MD5; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-MD5; digest length = 12; key length = 16

Compression algorithm	
none*	No compression
zlib	Defined in RFC 1950 and RFC 1951

* required
** recommended

UNIVERSITÄT BERN

5. Secure Shell

3.3 Key Generation

The keys used for encryption and MAC are generated from

- shared secret key *K* resulting from DH key exchange
- hash value from key exchange *H*
- session identifier, which is equal to *H* unless there has been a subsequent key exchange after the initial key exchange

Initial IV client to server: $\mathrm{HASH}(K \| H \| \text{“A”} \| \text{session\_id})$
Initial IV server to client: $\mathrm{HASH}(K \| H \| \text{“B”} \| \text{session\_id})$
Encryption key client to server: $\mathrm{HASH}(K \| H \| \text{“C”} \| \text{session\_id})$
Encryption key server to client: $\mathrm{HASH}(K \| H \| \text{“}D\text{”} \| \text{session\_id})$
Integrity key client to server: $\mathrm{HASH}(K \| H \| \text{“}E\text{”} \| \text{session\_id})$
Integrity key server to client: $\mathrm{HASH}(K \| H \| \text{“}F\text{”} \| \text{session\_id})$

5. Secure Shell

4.1 User Authentication Protocol: Message Exchange

1. Client sends an **`SSH_MSG_USERAUTH_REQUEST`** with a requested method of none
2. Server checks if user name is valid. If not, the server returns **`SSH_MSG_USERAUTH_FAILURE`** with the partial success value of false, otherwise continue with 3.)
3. Server returns **`SSH_MSG_USERAUTH_FAILURE`** with a list of one or more authentication methods to be used.
4. Client selects 1 authentication method and sends **`SSH_MSG_USERAUTH_REQUEST`**. There may be a sequence of exchanges to perform the authentication method.
5. If authentication succeeds and more authentication methods are required, server proceeds to step 3, using a partial success value of true. If authentication fails, the server proceeds to step 3, using a partial success value of false
6. When all required authentication methods succeed, the server sends **`SSH_MSG_USERAUTH_SUCCESS`** message; Authentication Protocol is over.

5. Secure Shell

4.2 Authentication Methods

- Public key
  - The client sends a message to the server that contains the client's public key, with the message signed by the client's private key.
  - Server checks whether the supplied key is acceptable for authentication and, if so, whether signature is correct.
- Password
  - Client sends a message containing a plaintext password, which is protected by encryption by the Transport Layer Protocol.
- Host-based
  - Authentication is performed on the client's host rather than the client itself by having the client sending a signature created with the private key of the client host.
  - Rather than directly verifying the user's identity, the SSH server verifies the identity of the client host.

5. Secure Shell

5.1 Connection Protocol

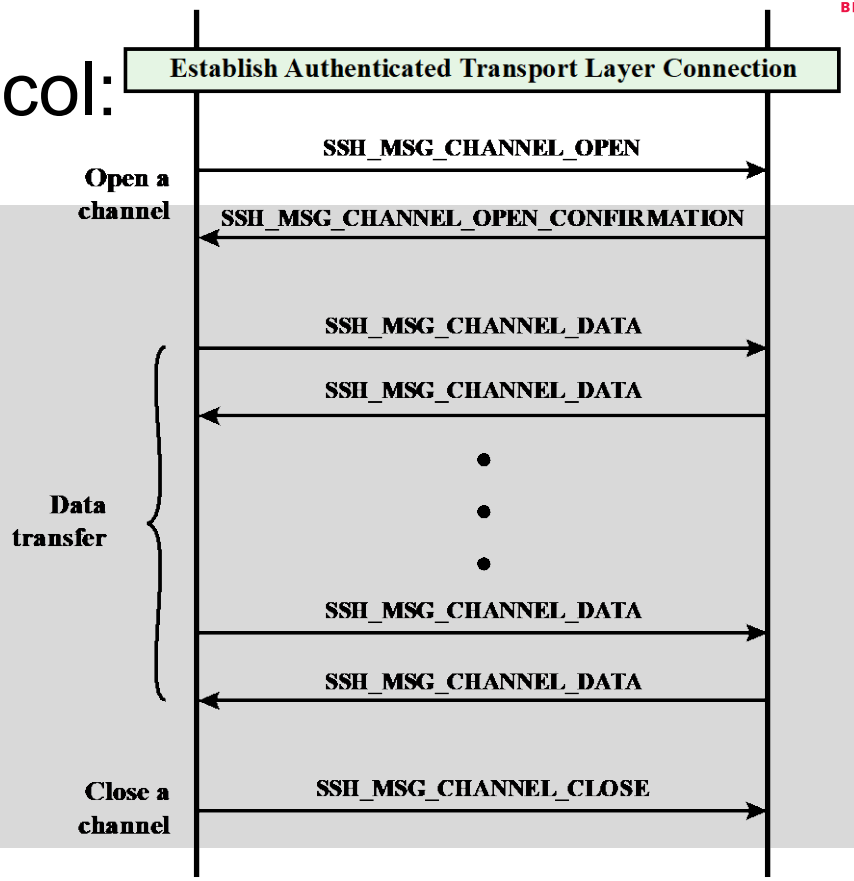
- SSH Connection Protocol runs on top of SSH Transport Layer Protocol and assumes that a secure authentication connection is in use.
- The secure authentication connection, referred to as a tunnel, is used by the Connection Protocol to multiplex several logical channels.

Channel mechanism

- All types of communication using SSH are supported using separate channels
- Either side may open a channel
- For each channel, each side associates a unique channel number
- Channels are flow controlled using a window mechanism
- No data may be sent to a channel until a message is received to indicate that window space is available

5. Secure Shell

5.2 Connection Protocol: Message Exchange



5. Secure Shell

5.3 Channel Types

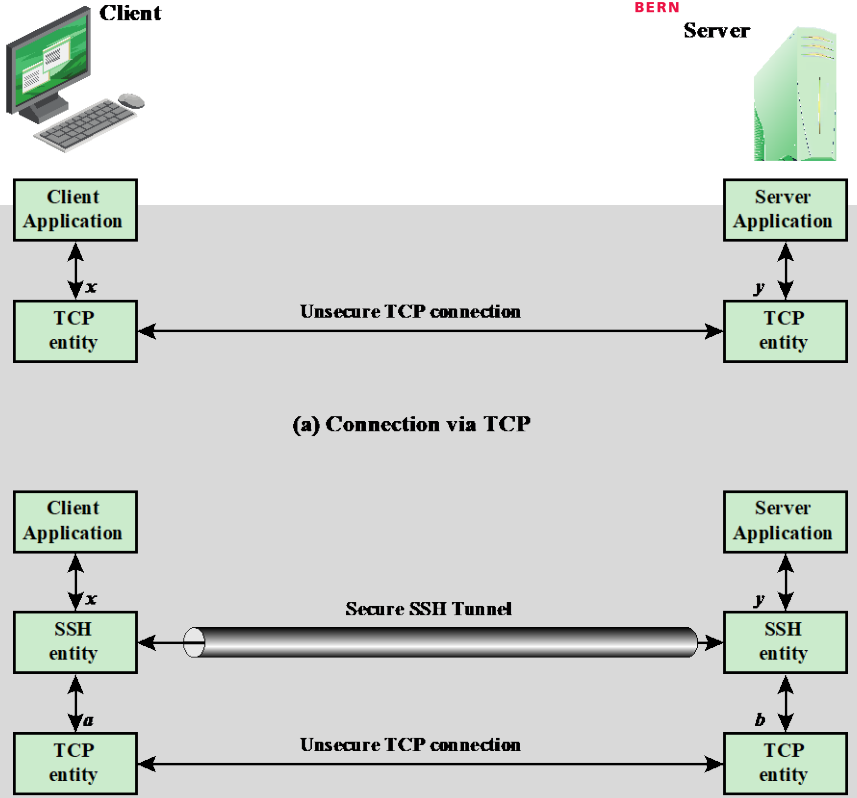
- Session
  - remote execution of a program.
  - Program may be a shell, an application such as file transfer or e-mail, a system command, or some built-in subsystem.
  - Once a session channel is opened, subsequent requests are used to start the remote program.
- X11
  - refers to the X Window System, a network protocol that provides a graphical user interface (GUI) for networked computers.
  - allows applications to run on a network server but to be displayed on a desktop machine
- forwarded-tcpip
  - remote port forwarding
- direct-tcpip
  - local port forwarding

UNIVERSITÄT BERN

5. Secure Shell

5.4 Port Forwarding

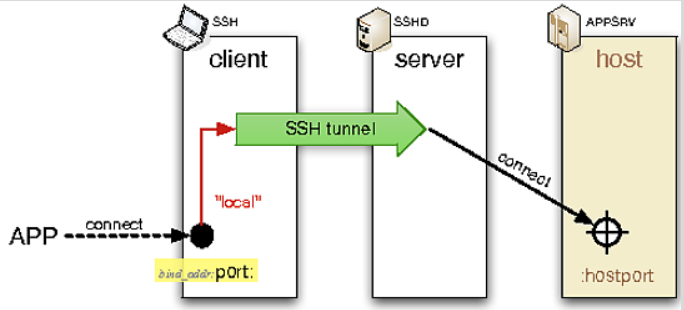
- provides the ability to convert any insecure TCP connection into a secure SSH connection (also referred to as SSH tunneling)
- Incoming TCP traffic is delivered to the appropriate application based on port number.
- Types of port forwarding
  - Local
  - Remote



5. Secure Shell

5.4.1 Local Port Forwarding

- allows the client to set up a “hijacker” process to intercept selected application-level traffic and redirect it from an unsecured TCP connection to a secure SSH tunnel.
- SSH is configured to listen on selected ports. SSH grabs all traffic using a selected port and sends it through an SSH tunnel.
- On the other end, SSH server sends incoming traffic to destination port dictated by the client application.

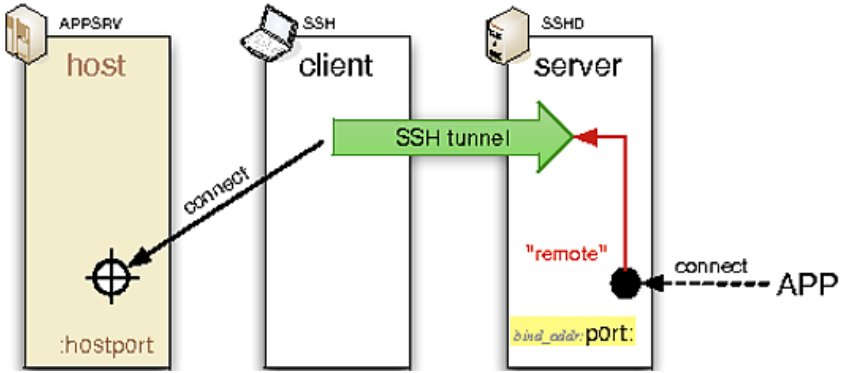


Example: POP using port 110

1. SSH client sets up a connection to the remote server.
2. Select unused local port number, e.g., 9999; configure SSH to accept traffic from this port destined for port 110 on server.
3. SSH client informs SSH server to create a connection to the destination, in this case mail server port 110.
4. Client takes any bits sent to local port 9999 and sends them to server inside the encrypted SSH session. SSH server decrypts incoming bits and sends plaintext to port 110.
5. In the other direction, SSH server takes any bits received on port 110 and sends them inside the SSH session back to client, who decrypts and sends them to the process connected to port 9999.

5. Secure Shell

5.4.2 Remote Port Forwarding



User's SSH client acts on server's behalf.

- receives traffic with a given destination port
- places traffic on the correct port and
- sends it to the destination the user chooses.

Example application: access to work computer from home computer

- Work computer behind a firewall will not accept SSH request from home computer.
- However, from work computer one can set up SSH tunnel using remote port forwarding.

Example

- From work computer, set up outgoing SSH connection to home computer.
- Configure SSH server to listen on a local port, e.g., 22, and to deliver data across the SSH connection addressed to remote port, e.g., 2222.
- Configure SSH at home computer to accept traffic on port 2222.
- SSH tunnel can now be used for remote logon to work server

Thanks a lot

for your Attentation

Prof. Dr. Torsten Braun, Institut für Informatik

Bern, 02.05.2022 – 09.05.2022