Computer and Network Security (2023-2024)

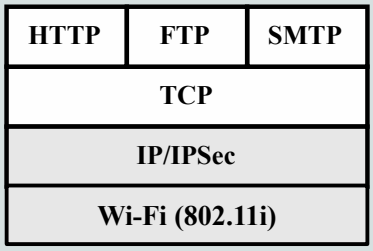
Part 6: Network and Internet Security

Jeroen Famaey
jeroen.famaey@uantwerpen.be

University of Antwerp
Faculty of Science

Security approaches for web traffic

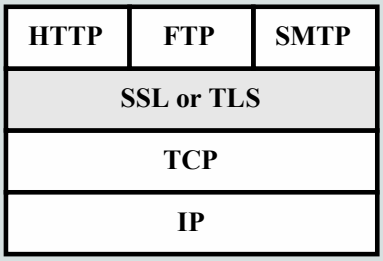
Network level



Advantages

- Transparent to users and transport layer
- Selective traffic filtering

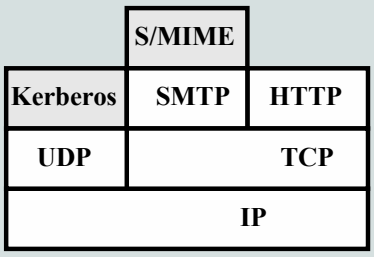
Transport level



Advantages

- Transparent to users
- Integrated into TCP or embedded into applications

Application level



Advantages

- Tailored to specific application needs
- Only requires end-point implementation

University of Antwerp
Faculty of Science

2

The World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets. As such, the security tools and approaches discussed so far in this book are relevant to the issue of Web security. A number of approaches to providing Web security are possible. The various approaches that have been considered are similar in the services they provide and, to some extent, in the mechanisms that they use, but they differ with respect to their scope of applicability and their relative location within the TCP/IP protocol stack. One way to provide Web security is to use network level solutions, such as IP security (IPsec) or Wi-Fi security (802.11i). The advantage of using these solutions is that they are transparent to end users and applications and provides a general-purpose solution. Furthermore, IPsec includes a filtering capability so that only selected traffic need incur the overhead of IPsec processing. Another relatively general-purpose solution is to implement security just above TCP. The foremost example of this approach is the Secure Sockets Layer (SSL) and the follow-on Internet standard known as Transport Layer Security (TLS). At this level, there are two implementation choices. For full generality, SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, TLS can be embedded in specific packages. For example, virtually all browsers come equipped with TLS, and most Web servers have implemented the protocol. Application-specific security services are embedded within the particular application. The figure shows examples of this architecture. The advantage of this approach is that the service can be tailored to the specific needs of a given application.

Overview

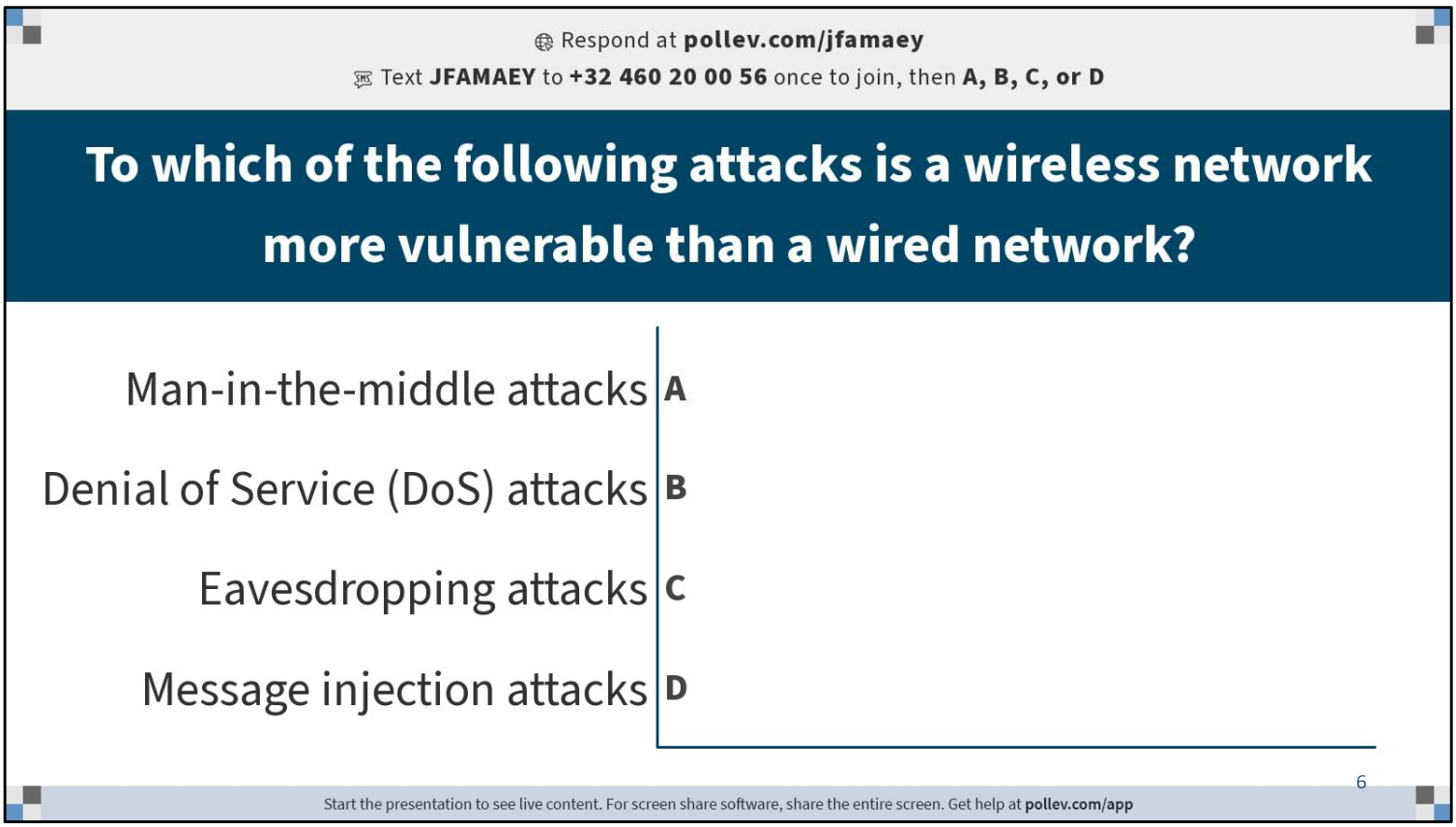
1. Network Level Security
   - Wi-Fi security
   - IP-layer security
2. Transport Level Security
   - Transport Layer Security (TLS)
   - Attacks against TLS
   - Secure Shell (SSH)
3. Application and endpoint security
   - E-mail security
   - Firewalls
   - Web application security

Network Level Security

1 Wi-Fi Security (802.11i)

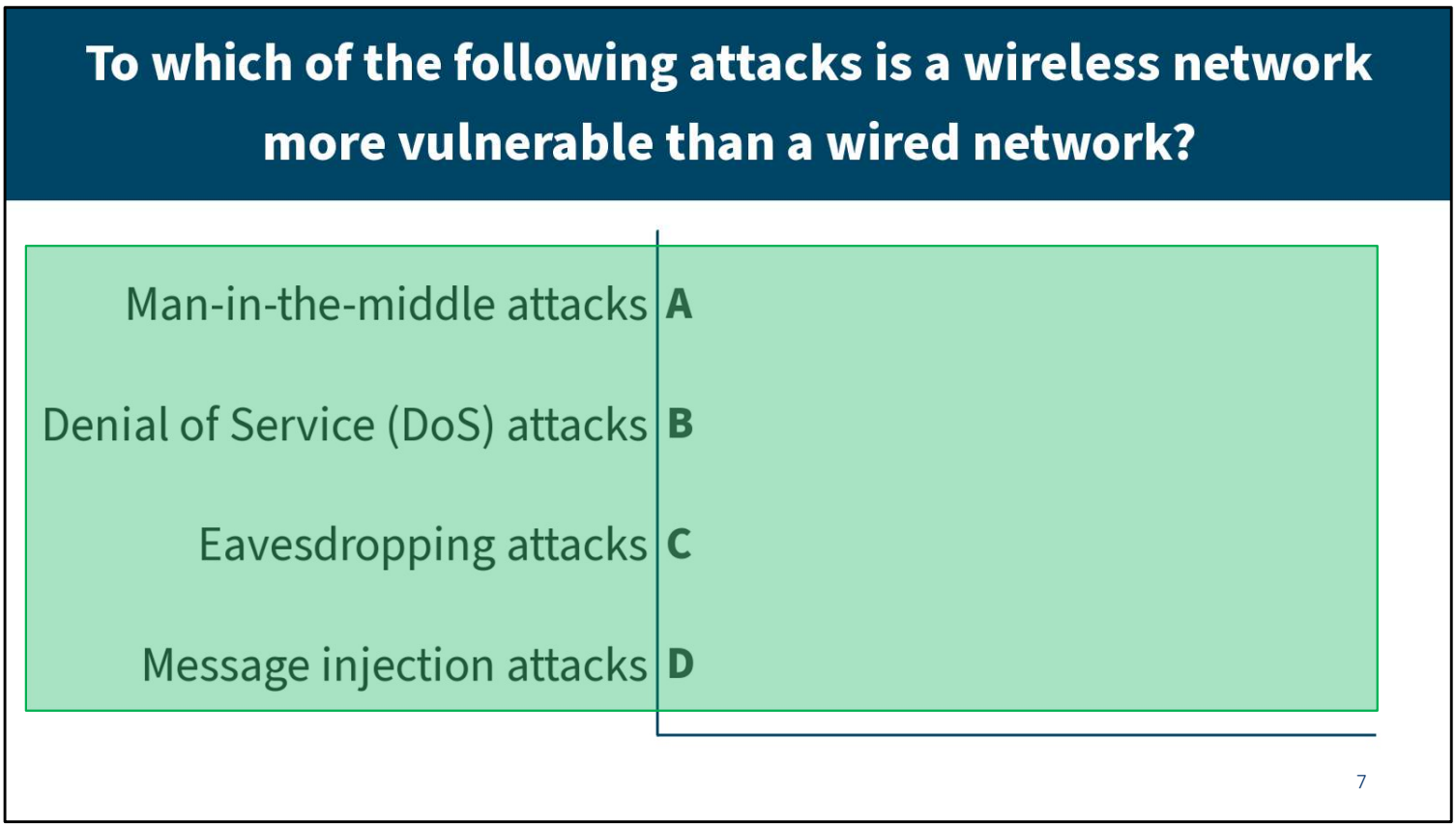
2 IP-Layer Security (IPSec)

University of Antwerp
Faculty of Science



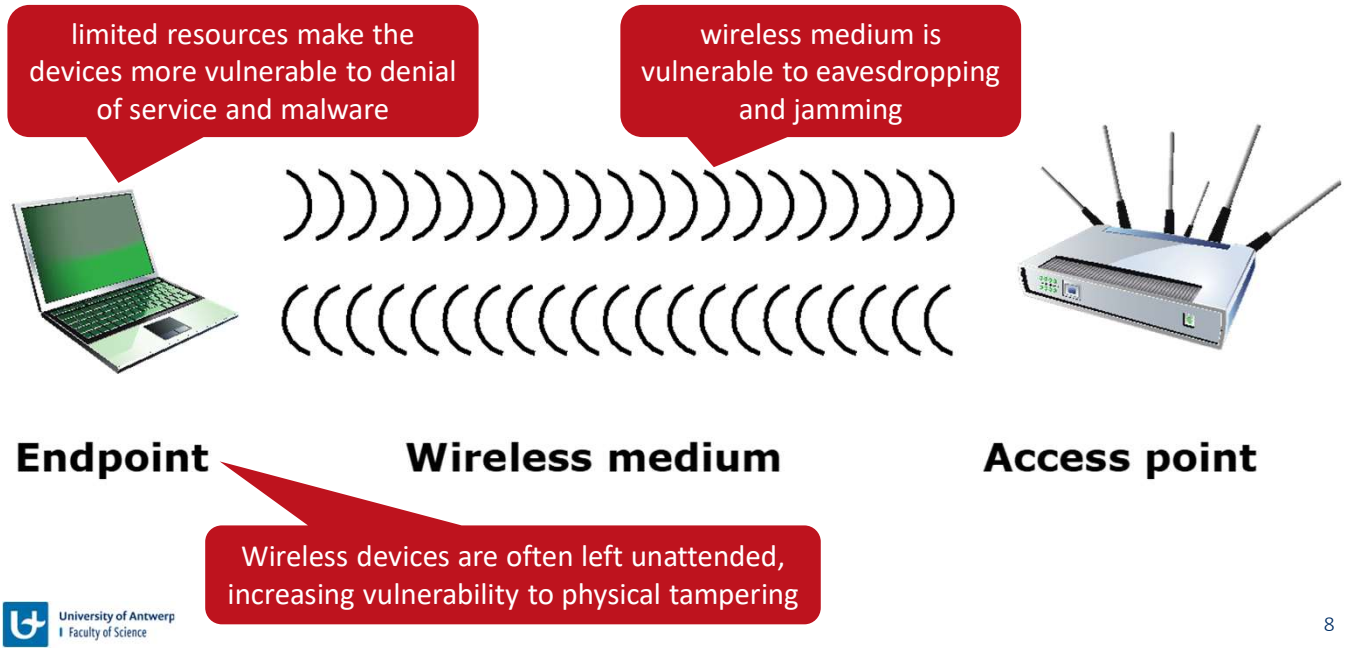
Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

To which of the following attacks is a wireless network more vulnerable than a wired network?
https://www.polleverywhere.com/multiple_choice_polls/xXljAZUneXt0B8l?state=opened&flow=Default&onscreen=persist



Poll Title: To which of the following attacks is a wireless network more vulnerable than a wired network?

Wireless security risks



University of Antwerp | Faculty of Science

8

Wireless networks, and the wireless devices that use them, introduce a host of security problems over and above those found in wired networks. Some of the key factors contributing to the higher security risk of wireless networks compared to wired networks include the following:

- **Channel:** Wireless networking typically involves broadcast communications, which is far more susceptible to eavesdropping and jamming than wired networks. Wireless networks are also more vulnerable to active attacks that exploit vulnerabilities in communications protocols.
- **Mobility:** Wireless devices are, in principal and usually in practice, far more portable and mobile than wired devices.
- **Resources:** Some wireless devices, such as smartphones and tablets, have sophisticated operating systems but limited memory and processing resources with which to counter threats, including denial of service and malware.
- **Accessibility:** Some wireless devices, such as sensors and robots, may be left unattended in remote and/or hostile locations. This greatly increases their vulnerability to physical attacks.

Security threats to wireless networks:

- **Accidental association:** Company wireless LANs or wireless access points to wired LANs in close proximity (e.g., in the same or neighboring buildings) may create overlapping transmission ranges. A user intending to connect to one LAN may unintentionally lock on to a wireless access point from a neighboring network. Although the security breach is accidental, it nevertheless exposes resources of one LAN to the accidental user.
- **Malicious association:** In this situation, a wireless device is configured to appear to be a legitimate access point, enabling the operator to steal passwords from legitimate users and then penetrate a wired network through a legitimate wireless access point.
- **Ad hoc networks:** These are peer-to-peer networks between wireless computers with no access point between them. Such networks can pose a security threat due to a lack of a

central point of control.

- **Nontraditional networks:** Nontraditional networks and links, such as personal network Bluetooth devices, barcode readers, and handheld PDAs, pose a security risk in terms of both eavesdropping and spoofing.
- **Identity theft (MAC spoofing):** This occurs when an attacker is able to eavesdrop on network traffic and identify the MAC address of a computer with network privileges.
- **Man-in-the middle attacks:** This attack involves persuading a user and an access point to believe that they are talking to each other when in fact the communication is going through an intermediate attacking device. Wireless networks are particularly vulnerable to such attacks.
- **Denial of service (DoS):** In the context of a wireless network, a DoS attack occurs when an attacker continually bombards a wireless access point or some other accessible wireless port with various protocol messages designed to consume system resources. The wireless environment lends itself to this type of attack, because it is so easy for the attacker to direct multiple wireless messages at the target.
- **Network injection:** A network injection attack targets wireless access points that are exposed to nonfiltered network traffic, such as routing protocol messages or network management messages. An example of such an attack is one in which bogus reconfiguration commands are used to affect routers and switches to degrade network performance.

IEEE 802.11i

1997 — **Wired Equivalent Protection (WEP)**
- Uses RC4 steam cipher for confidentiality and CRC-32 for integrity
- Successfully cracked in 2001 and fully deprecated in 2004

2003 — **Wi-Fi Protected Access (WPA)**
- Based on a preliminary draft of the IEEE 802.11i standard
- Uses RC4 stream cipher and Temporal Key Integrity Protocol (TKIP)

2004 — **Wi-Fi Protected Access II (WPA2) or Robust Security Network (RSN)**
- Implements the completed final IEEE 802.11i standard
- Uses strong AES encryption for confidentiality

2018 — **Wi-Fi Protected Access III (WPA3)**
- Uses stronger and up-to-date cryptography algorithms
- Replacing WPA2 pre-shared keys with more secure Simultaneous Authentication of Equals (SAE)

IEEE 802 is a committee that has developed standards for a wide range of local area networks (LANs). In 1990, the IEEE 802 Committee formed a new working group, IEEE 802.11, with a charter to develop a protocol and transmission specifications for wireless LANs (WLANs). Since that time, the demand for WLANs at different frequencies and data rates has exploded. Keeping pace with this demand, the IEEE 802.11 working group has issued an ever-expanding list of standards.

The first 802.11 standard to gain broad industry acceptance was 802.11b. Although 802.11b products are all based on the same standard, there is always a concern whether products from different vendors will successfully interoperate. To meet this concern, the Wireless Ethernet Compatibility Alliance (WECA), an industry consortium, was formed in 1999. This organization, subsequently renamed the Wi-Fi (Wireless Fidelity) Alliance, created a test suite to certify interoperability for 802.11b products.

The differences between wired and wireless LANs suggest the increased need for robust security services and mechanisms for wireless LANs. The original 802.11 specification included a set of security features for privacy and authentication that were quite weak. For privacy, 802.11 defined the **Wired Equivalent Privacy (WEP)** algorithm. The privacy portion of the 802.11 standard contained major weaknesses. Subsequent to the development of WEP, the 802.11i task group has developed a set of capabilities to address the WLAN security issues. In order to accelerate the introduction of strong security into WLANs, the Wi-Fi Alliance promulgated **Wi-Fi Protected Access (WPA)** as a Wi-Fi standard. WPA is a set of security mechanisms that eliminates most 802.11 security issues and was based on the current state of the 802.11i standard. The final form of the 802.11i standard is referred to as **Robust Security Network (RSN)**. The Wi-Fi Alliance certifies vendors in compliance with the full 802.11i specification under the WPA2 program.

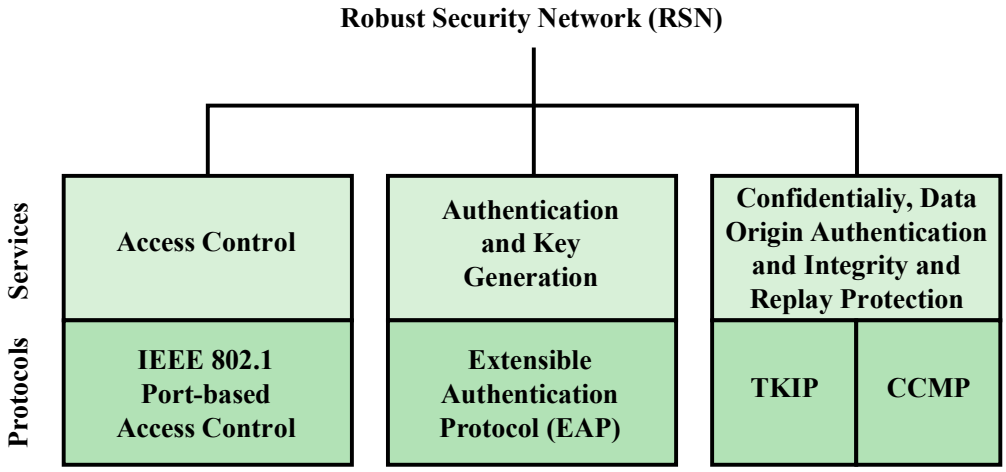
The lacking security of WEP

- Fluhrer, Mantin and Shamir (FMS) were the first to crack WEP in 2001
  - FMS is a passive attack that discovers the RC4 key by eavesdropping
  - Reconstructs the key from encrypted messages based on a weakness in the specific key derivation method of RC4 used by WEP
  - Cracking speed depends on amount of traffic, and can be as fast as minutes

- Other attacks further reduced time needed to find the key
  - KoreK: Extends FMS with additional key correlations (2004)
  - PTW: Removes certain assumptions of FMS, making more packets usable (2007)

- Network software suite for cracking Wi-Fi security
- Can be used any wireless network interface controller that supports raw monitoring mode
- Implements Fluhrer, Mantin and Shamir as well as several other known WEP and WPA2-PSK attacks

WPA2 security services and protocols



TKIP = Temporal Key Integrity Protocol
CCMP = Counter Mode with Cipher Block Chaining MAC Protocol

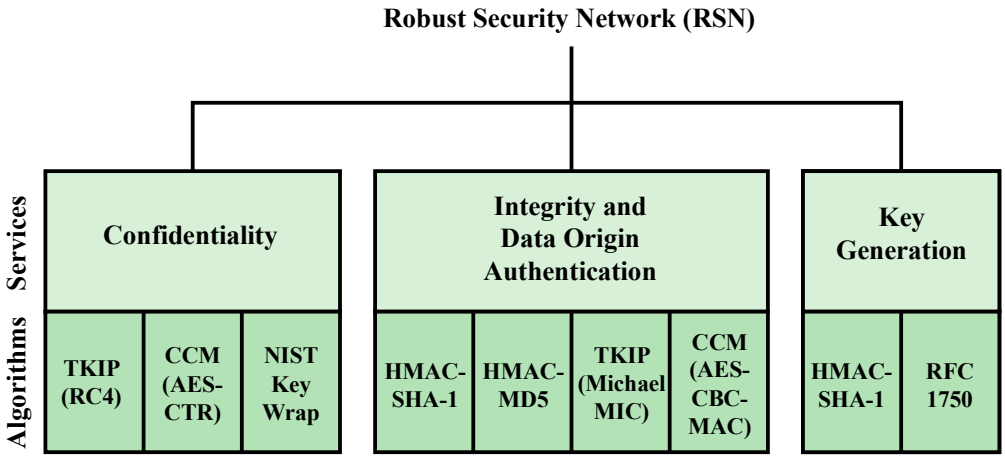
University of Antwerp | Faculty of Science

11

The 802.11i RSN security specification defines the following services:

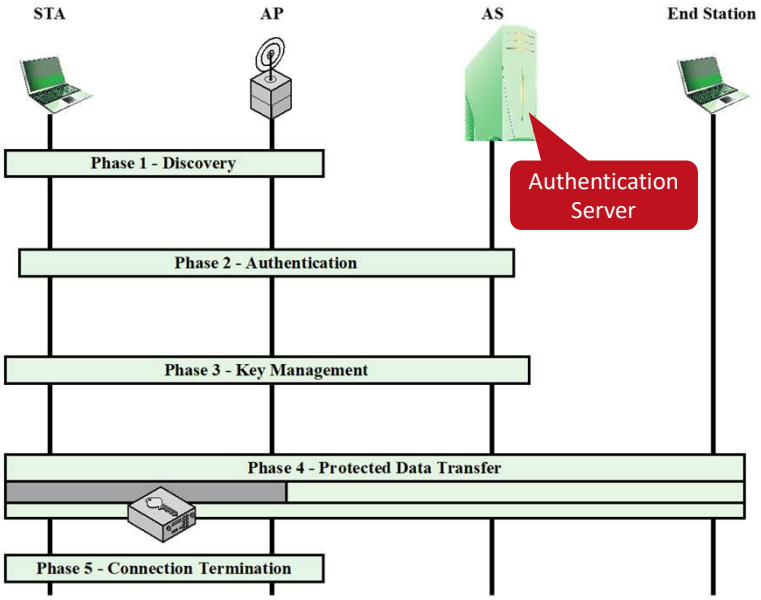
- **Authentication:** A protocol is used to define an exchange between a user and an AS that provides mutual authentication and generates temporary keys to be used between the client and the AP over the wireless link.
- **Access control:** This function enforces the use of the authentication function, routes the messages properly, and facilitates key exchange. It can work with a variety of authentication protocols.
- **Privacy with message integrity:** MAC-level data (e.g., an LLC PDU) are encrypted along with a message integrity code that ensures that the data have not been altered.

WPA2 cryptographic algorithms



TKIP = Temporal Key Integrity Protocol
CCM = Counter Mode with Cipher Block Chaining MAC

WPA2 phases



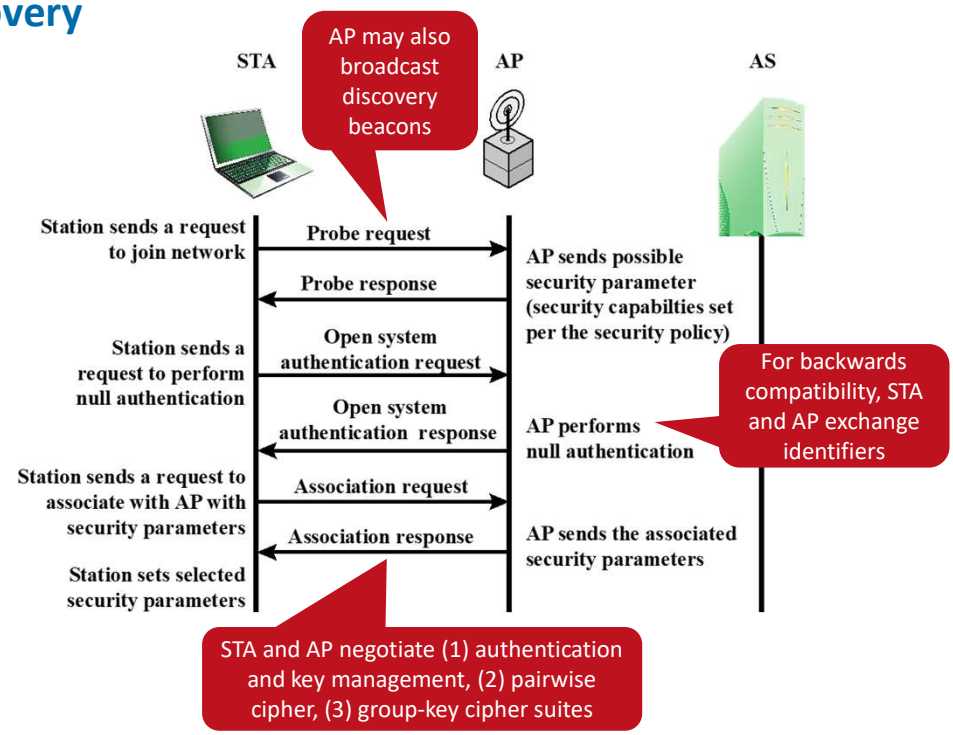
University of Antwerp
Faculty of Science

13

The figure depicts the five phases of operation for an RSN and maps them to the network components involved. One new component is the authentication server (AS). The rectangles indicate the exchange of sequences of MPDUs. The five phases are defined as follows:

1. **Discovery:** An AP uses messages called Beacons and Probe Responses to advertise its IEEE 802.11i security policy. The STA uses these to identify an AP for a WLAN with which it wishes to communicate. The STA associates with the AP, which it uses to select the cipher suite and authentication mechanism when the Beacons and Probe Responses present a choice.
2. **Authentication:** The STA and AS prove their identities to each other. The AP blocks non-authentication traffic between the STA and AS until the authentication transaction is successful. The AP does not participate in the authentication transaction other than forwarding traffic between the STA and AS.
3. **Key generation and distribution:** The AP and the STA perform several operations that cause cryptographic keys to be generated and placed on the AP and the STA. Frames are exchanged between the AP and STA only.
4. **Protected data transfer:** Frames are exchanged between the STA and the end station through the AP. As denoted by the shading and the encryption module icon, secure data transfer occurs between the STA and the AP only; security is not provided end-to-end.
5. **Connection termination:** During this phase, the secure connection is torn down and the connection is restored to the original state.

Phase 1: Discovery



University of Antwerp | Faculty of Science

14

The purpose of this phase is for a STA and an AP to recognize each other, agree on a set of security capabilities, and establish an association for future communication using those security capabilities. During this phase, the STA and AP decide on specific techniques in the following areas:

- Confidentiality and MAC protocol data unit (MPDU) integrity protocols for protecting unicast traffic (traffic only between this STA and AP)
- Authentication method
- Cryptography key management approach

The discovery phase consists of three exchanges:

- **Network and security capability discovery:** During this exchange, STAs discover the existence of a network with which to communicate. The AP either periodically broadcasts its security capabilities (not shown in figure), indicated by RSN IE (Robust Security Network Information Element), in a specific channel through the Beacon frame; or responds to a station's Probe Request through a Probe Response frame. A wireless station may discover available access points and corresponding security capabilities by either passively monitoring the Beacon frames or actively probing every channel.
- **Open system authentication:** The purpose of this frame sequence, which provides no security, is simply to maintain backward compatibility with the IEEE 802.11 state machine, as implemented in existing IEEE 802.11 hardware. In essence, the two devices (STA and AP) simply exchange identifiers.
- **Association:** The purpose of this stage is to agree on a set of security capabilities to be used. The STA then sends an Association Request frame to the AP. In this frame, the STA specifies one set of matching capabilities (one authentication and key management suite, one pairwise cipher suite, and one group-key cipher suite) from among those advertised by the AP. If there is no match in capabilities between the AP and the STA, the AP refuses the Association

Request. The STA blocks it too, in case it has associated with a rogue AP or someone is inserting frames illicitly on its channel.

Security suite negotiation

Pairwise and group-key cipher suite — Defines the confidentiality and integrity protocols together with the key length

Possible methods
- WEP with a 40 or 104-bit key (for backwards compatibility)
- Temporal key integrity protocol (TKIP)
- Counter Mode with Cipher Block Chaining MAC Protocol (CCMP)
- Vendor specific method

Authentication and key management suite — Defines mutual authentication between STA and AP and method to generate keys

Possible methods
- IEEE 802.1X and EAP: Port-based network access control
- Pre-shared key (PSK): Authentication implied if STA knows the secret key
- Vendor-specific method

University of Antwerp | Faculty of Science

15

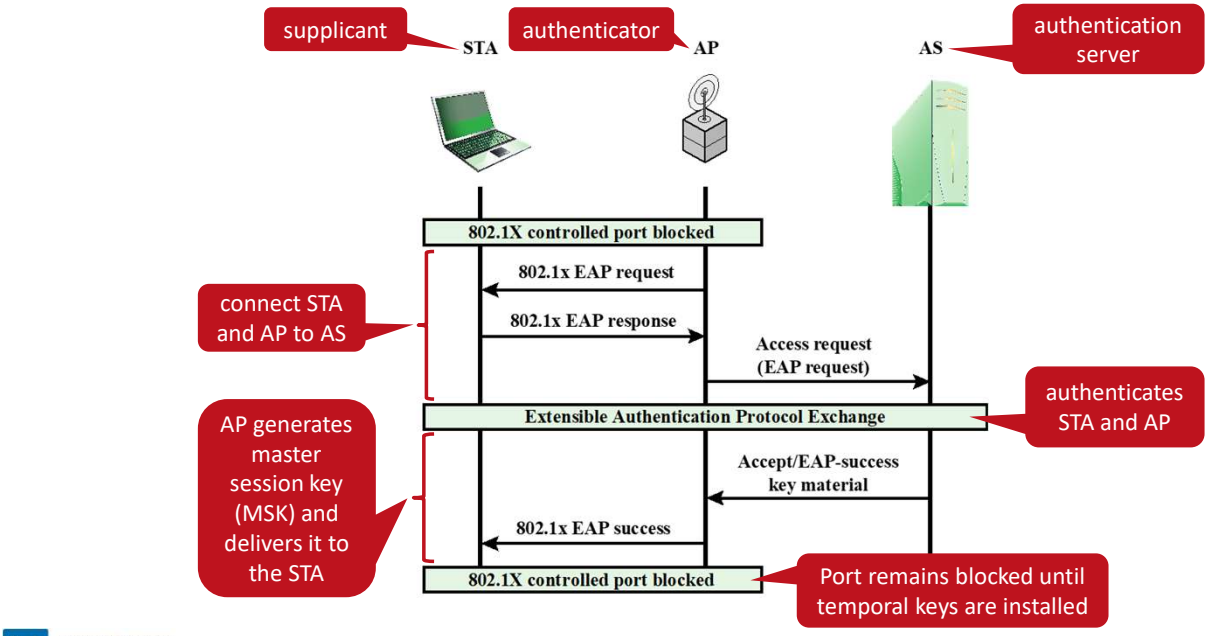
Confidentiality and integrity protocols for protecting multicast/broadcast traffic are dictated by the AP, since all STAs in a multicast group must use the same protocols and ciphers. The specification of a protocol, along with the chosen key length (if variable) is known as a *cipher suite*. The options for the confidentiality and integrity cipher suite are:

- WEP, with either a 40-bit or 104-bit key, which allows backward compatibility with older IEEE 802.11 implementations
- TKIP
- CCMP
- Vendor-specific methods

The other negotiable suite is the authentication and key management (AKM) suite, which defines (1) the means by which the AP and STA perform mutual authentication and (2) the means for deriving a root key from which other keys may be generated. The possible AKM suites are

- IEEE 802.1X
- Pre-shared key (no explicit authentication takes place and mutual authentication is implied if the STA and AP share a unique secret key)
- Vendor-specific methods

Phase 2: Authentication using EAP and 802.1X



University of Antwerp | Faculty of Science

16

Authentication is designed to allow only authorized stations to use the network and to provide the STA with assurance that it is communicating with a legitimate network. IEEE 802.11i makes use of another standard that was designed to provide access control functions for LANs. The standard is IEEE 802.1X, Port-Based Network Access Control. The authentication protocol that is used, the Extensible Authentication Protocol (EAP), is defined in the IEEE 802.1X standard. IEEE 802.1X uses the terms *supplicant*, *authenticator*, and *authentication server* (AS). In the context of an 802.11 WLAN, the first two terms correspond to the wireless station and the AP. The AS is typically a separate device on the wired side of the network (i.e., accessible over the DS) but could also reside directly on the authenticator.

Before a supplicant is authenticated by the AS using an authentication protocol, the authenticator only passes control or authentication messages between the supplicant and the AS; the 802.1X control channel is unblocked, but the 802.11 data channel is blocked. Once a supplicant is authenticated and keys are provided, the authenticator can forward data from the supplicant, subject to predefined access control limitations for the supplicant to the network. Under these circumstances, the data channel is unblocked.

We can think of authentication phase as consisting of the following three phases:

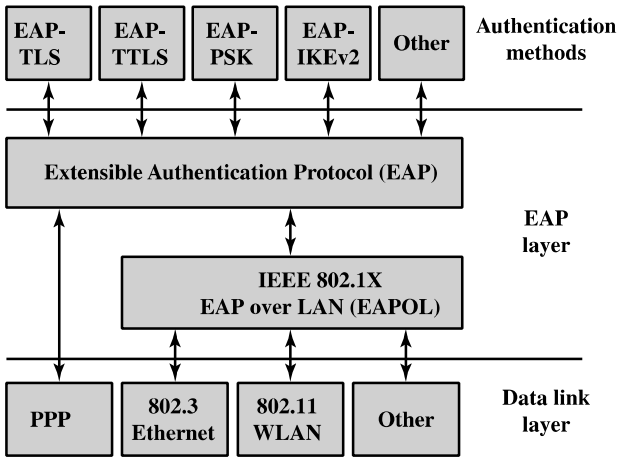
- **Connect to AS:** The STA sends a request to its AP (the one with which it has an association) for connection to the AS. The AP acknowledges this request and sends an access request to the AS.
- **EAP exchange:** This exchange authenticates the STA and AS to each other. A number of alternative exchanges are possible, as explained subsequently.
- **Secure key delivery:** Once authentication is established, the AS generates a master session key (MSK), also known as the Authentication, Authorization, and Accounting (AAA) key and sends it to the STA. As explained subsequently, all the cryptographic keys needed by the STA

for secure communication with its AP are generated from this MSK. IEEE 802.11i does not prescribe a method for secure delivery of the MSK but relies on EAP for this. Whatever method is used, it involves the transmission of an MPDU containing an encrypted MSK from the AS, via the AP, to the AS.

As mentioned, there are a number of possible EAP exchanges that can be used during the authentication phase. Typically, the message flow between STA and AP employs the EAP over LAN (EAPOL) protocol, and the message flow between the AP and AS uses the Remote Authentication Dial In User Service (RADIUS) protocol, although other options are available for both STA-to-AP and AP-to-AS exchanges.

Note that the AP controlled port is still blocked to general user traffic. Although the authentication is successful, the ports remain blocked until the temporal keys are installed in the STA and AP, which occurs during the 4-Way Handshake.

Extensible Authentication Protocol (EAP)



EAP-TLS Unencrypted TLS handshake that uses certificates for authentication

EAP-TTLS Server uses EAP-TLS and client can use any EAP authentication method

EAP-GPSK Authentication and session keys are derived from pre-shared private key

EAP-IKEv2 Internet Key Exchange protocol is used for mutual authentication

University of Antwerp | Faculty of Science

17

The Extensible Authentication Protocol (EAP), defined in RFC 3748, acts as a framework for network access and authentication protocols. EAP provides a set of protocol messages that can encapsulate various authentication methods to be used between a client and an authentication server. EAP can operate over a variety of network and link level facilities, including point-to-point links, LANs, and other networks, and can accommodate the authentication needs of the various links and networks.

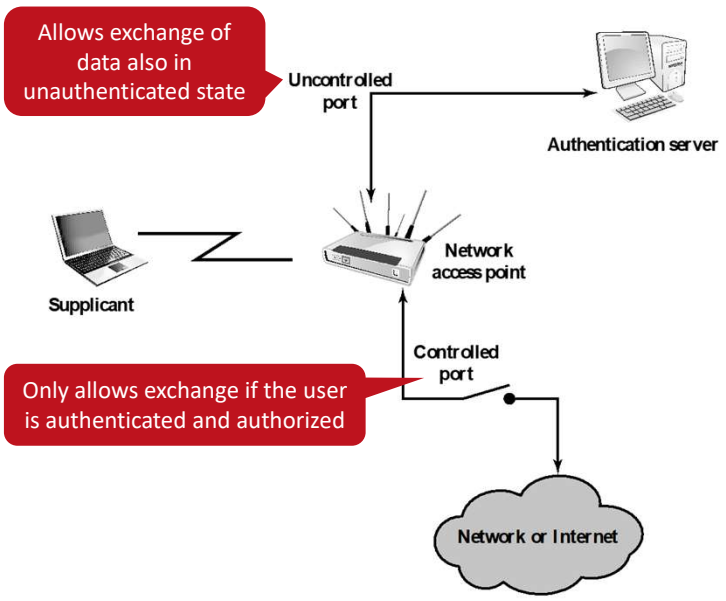
EAP supports multiple authentication methods. This is what is meant by referring to EAP as *extensible*. EAP provides a generic transport service for the exchange of authentication information between a client system and an authentication server. The basic EAP transport service is extended by using a specific authentication protocol, or method, that is installed in both the EAP client and the authentication server. The following are commonly supported EAP methods:

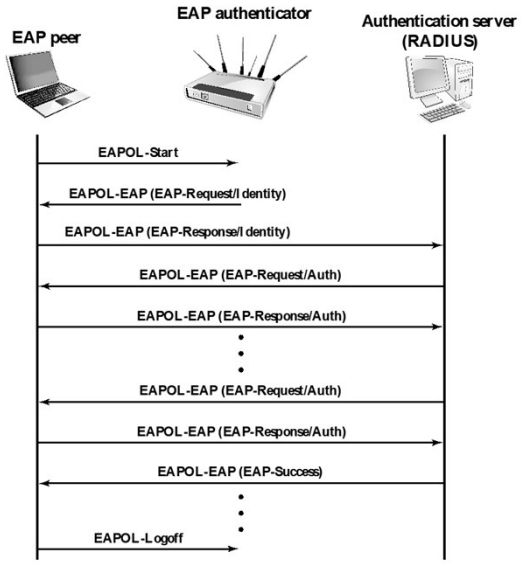
- **EAP-TLS (EAP Transport Layer Security):** EAP-TLS (RFC 5216) defines how the TLS protocol can be encapsulated in EAP messages. EAP-TLS uses the handshake protocol in TLS, not its encryption method. Client and server authenticate each other using digital certificates. Client generates a pre-master secret key by encrypting a random number with the server's public key and sends it to the server. Both client and server use the pre-master to generate the same secret key.
- **EAP-TTLS (EAP Tunneled TLS):** EAP-TTLS is like EAP-TLS, except only the server has a certificate to authenticate itself to the client first. As in EAP-TLS, a secure connection (the "tunnel") is established with secret keys, but that connection is used to continue the authentication process by authenticating the client and possibly the server again using any EAP method or legacy method such as PAP (Password Authentication Protocol) and CHAP (Challenge-Handshake Authentication Protocol). EAP-TTLS is defined in RFC 5281.
- **EAP-GPSK (EAP Generalized Pre-Shared Key):** EAP-GPSK, defined in RFC 5433, is an EAP

method for mutual authentication and session key derivation using a Pre-Shared Key (PSK). EAP-GPSK specifies an EAP method based on pre-shared keys and employs secret key-based cryptographic algorithms. Hence, this method is efficient in terms of message flows and computational costs but requires the existence of pre-shared keys between each peer and EAP server. The set up of these pairwise secret keys is part of the peer registration, and thus, must satisfy the system preconditions. It provides a protected communication channel when mutual authentication is successful for both parties to communicate over and is designed for authentication over insecure networks such as IEEE 802.11. EAP-GPSK does not require any public-key cryptography. The EAP method protocol exchange is done in a minimum of four messages.

- **EAP-IKEv2:** It is based on the Internet Key Exchange protocol version 2 (IKEv2). It supports mutual authentication and session key establishment using a variety of methods. EAP-IKEv2 is defined in RFC 5106.

802.1X port-based network access control





University of Antwerp | Faculty of Science

EAPOL = EAP-over-LAN (e.g., Ethernet, Wi-Fi)

18

IEEE 802.1X Port-Based Network Access Control was designed to provide access control functions for LANs. Until the AS authenticates a supplicant (using an authentication protocol), the authenticator only passes control and authentication messages between the supplicant and the AS; the 802.1X control channel is unblocked, but the 802.11 data channel is blocked. Once a supplicant is authenticated and keys are provided, the authenticator can forward data from the supplicant, subject to predefined access control limitations for the supplicant to the network. Under these circumstances, the data channel is unblocked.

802.1X uses the concepts of controlled and uncontrolled ports. Ports are logical entities defined within the authenticator and refer to physical network connections. Each logical port is mapped to one of these two types of physical ports. An uncontrolled port allows the exchange of protocol data units (PDUs) between the supplicant and the AS, regardless of the authentication state of the supplicant. A controlled port allows the exchange of PDUs between a supplicant and other systems on the network only if the current state of the supplicant authorizes such an exchange.

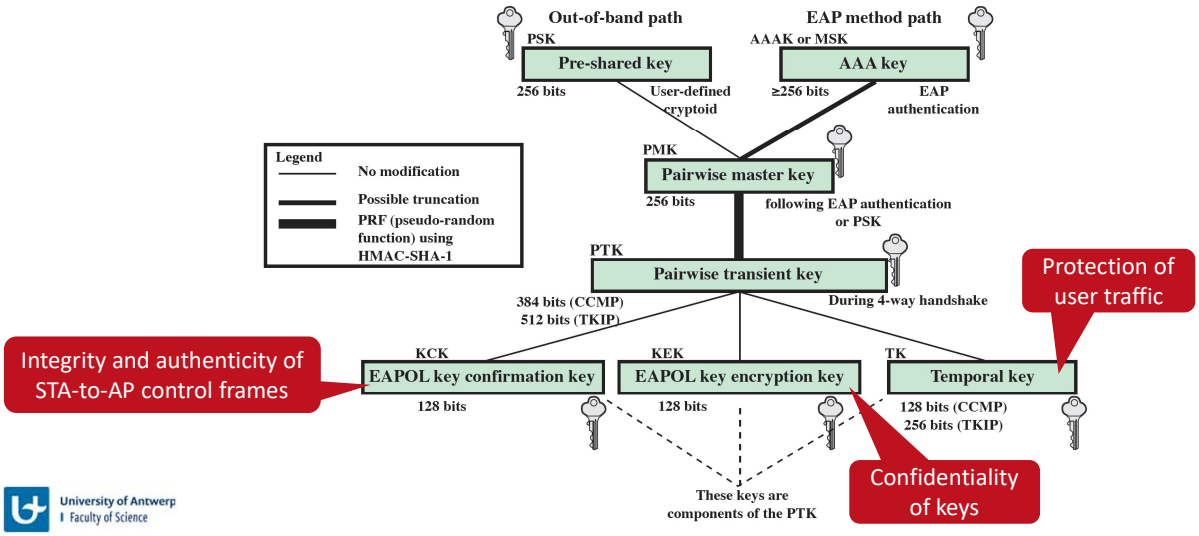
The essential element defined in 802.1X is a protocol known as EAPOL (EAP over LAN). EAPOL operates at the network layers and makes use of an IEEE 802 LAN, such as Ethernet or Wi-Fi, at the link level. EAPOL enables a supplicant to communicate with an authenticator and supports the exchange of EAP packets for authentication.

When the supplicant first connects to the LAN, it does not know the MAC address of the authenticator. Actually, it doesn't know whether there is an authenticator present at all. By sending an **EAPOL-Start** packet to a special group-multicast address reserved for IEEE 802.1X authenticators, a supplicant can determine whether an authenticator is present and let it know that the supplicant is ready. In many cases, the authenticator will already be notified that a new

device has connected from some hardware notification. For example, a hub knows that a cable is plugged in before the device sends any data. In this case the authenticator may preempt the Start message with its own message. In either case the authenticator sends an EAP-Request Identity message encapsulated in an **EAPOL-EAP** packet. The EAPOL-EAP is the EAPOL frame type used for transporting EAP packets. The authenticator uses the **EAP-Key** packet to send cryptographic keys to the supplicant once it has decided to admit it to the network. The **EAP-Logoff** packet type indicates that the supplicant wishes to be disconnected from the network.

Phase 3: Key management (pairwise keys)

Generation of pairwise keys (used for STA-AP) communication is done based on either the PSK or the MSK (also called AAA key)



University of Antwerp | Faculty of Science

19

During the key management phase, a variety of cryptographic keys are generated and distributed to STAs. There are two types of keys: pairwise keys used for communication between an STA and an AP and group keys used for multicast communication.

Pairwise keys are used for communication between a pair of devices, typically between a STA and an AP. These keys form a hierarchy beginning with a master key from which other keys are derived dynamically and used for a limited period of time.

At the top level of the hierarchy are two possibilities. A **pre-shared key (PSK)** is a secret key shared by the AP and a STA and installed in some fashion outside the scope of IEEE 802.11i. The other alternative is the **master session key (MSK)**, also known as the AAAK, which is generated using the IEEE 802.1X protocol during the authentication phase. The actual method of key generation depends on the details of the authentication protocol used. In either case (PSK or MSK), there is a unique key shared by the AP with each STA with which it communicates. All the other keys derived from this master key are also unique between an AP and a STA. Thus, each STA, at any time, has one set of keys, as depicted in the hierarchy, while the AP has one set of such keys for each of its STAs.

The **pairwise master key (PMK)** is derived from the master key. If a PSK is used, then the PSK is used as the PMK; if a MSK is used, then the PMK is derived from the MSK by truncation (if necessary). By the end of the authentication phase, marked by the 802.1X EAP Success message, both the AP and the STA have a copy of their shared PMK.

The PMK is used to generate the **pairwise transient key (PTK)**, which in fact consists of three keys to be used for communication between an STA and AP after they have been mutually authenticated. To derive the PTK, the HMAC-SHA-1 function is applied to the PMK, the MAC

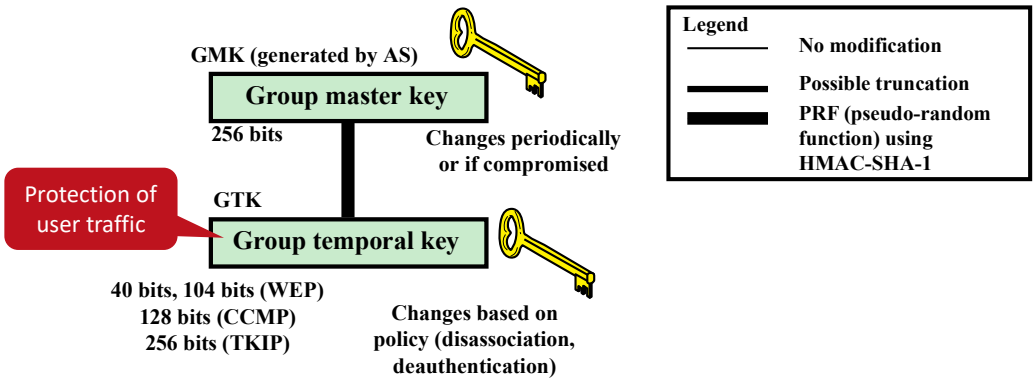
addresses of the STA and AP, and nonces generated when needed. Using the STA and AP addresses in the generation of the PTK provides protection against session hijacking and impersonation; using nonces provides additional random keying material.

The three parts of the PTK are as follows

- **EAP Over LAN (EAPOL) Key Confirmation Key (EAPOL-KCK):** Supports the integrity and data origin authenticity of STA-to-AP control frames during operational setup of an RSN. It also performs an access control function: proof-of-possession of the PMK. An entity that possesses the PMK is authorized to use the link.
- **EAPOL Key Encryption Key (EAPOL-KEK):** Protects the confidentiality of keys and other data during some RSN association procedures.
- **Temporal Key (TK):** Provides the actual protection for user traffic.

Phase 3: Key management (group keys)

- Group keys (for multicast communication) are generated by the AS
- Group temporal key (GTK) is shared by all stations in the same multicast group
- GTK changes whenever a station in the group leaves the network

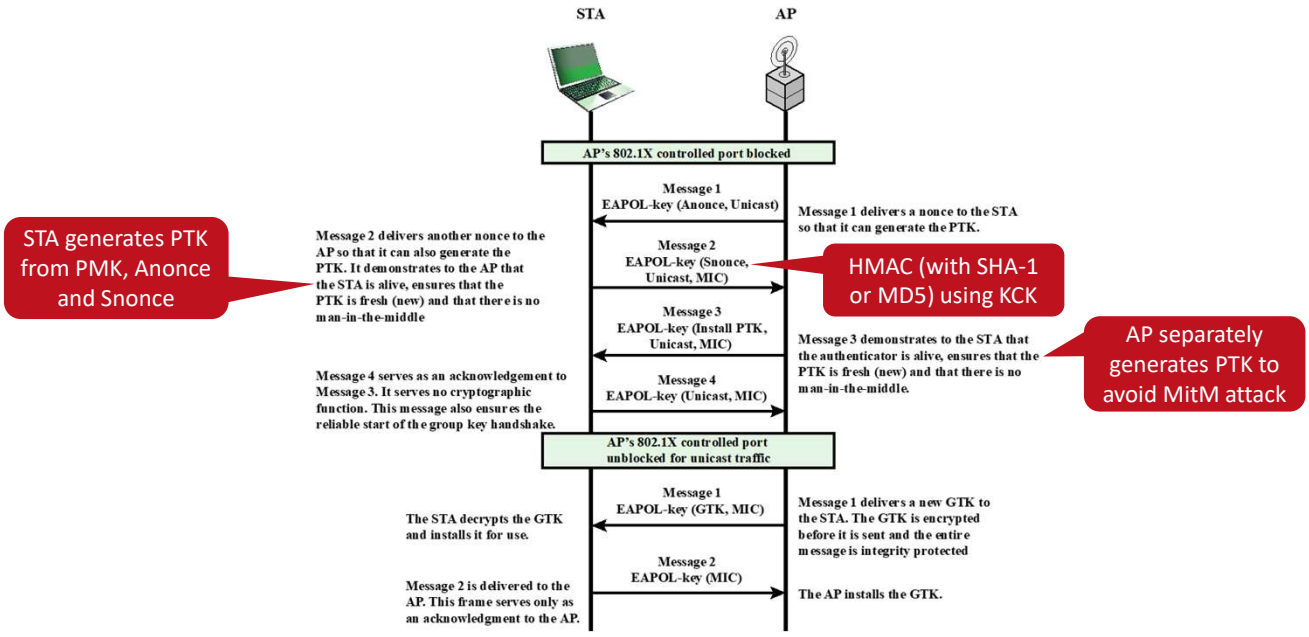


University of Antwerp
Faculty of Science

20

Group keys are used for multicast communication in which one STA sends MPDU's to multiple STAs. At the top level of the group key hierarchy is the **group master key (GMK)**. The GMK is a key-generating key used with other inputs to derive the **group temporal key (GTK)**. Unlike the PTK, which is generated using material from both AP and STA, the GTK is generated by the AP and transmitted to its associated STAs. Exactly how this GTK is generated is undefined. IEEE 802.11i, however, requires that its value is computationally indistinguishable from random. The GTK is distributed securely using the pairwise keys that are already established. The GTK is changed every time a device leaves the network.

Phase 3: 4-way handshake to exchange keys



University of Antwerp | Faculty of Science

***Note**: Message Authentication Code (MAC) is referred to as MIC in the context of 802.11, to avoid confusion with the Media Access Protocol*

21

The upper part of the figure shows the MPDU exchange for distributing **pairwise keys**. This exchange is known as the **4-way handshake**. The STA and AP use this handshake to confirm the existence of the PMK, verify the selection of the cipher suite, and derive a fresh PTK for the following data session. The four parts of the exchange are as follows:

- **AP → STA:** Message includes the MAC address of the AP and a nonce (Anonce)
- **STA → AP:** The STA generates its own nonce (Snonce) and uses both nonces and both MAC addresses, plus the PMK, to generate a PTK. The STA then sends a message containing its MAC address and Snonce, enabling the AP to generate the same PTK. This message includes a message integrity code (MIC) using HMAC-MD5 or HMAC-SHA-1-128. The key used with the MIC is EAPOL key confirmation key (KCK).
- **AP → STA:** The AP is now able to generate the PTK. The AP then sends a message to the STA, containing the same information as in the first message, but this time including a MIC.
- **STA → AP:** This is merely an acknowledgment message, again protected by a MIC.

For **group key distribution**, the AP generates a GTK and distributes it to each STA in a multicast group. The two-message exchange with each STA consists of the following:

- **AP → STA:** This message includes the GTK, encrypted either with RC4 or with AES. The key used for encryption is the EAPOL key encryption key (KEK). A MIC value is appended.
- **STA → AP:** The STA acknowledges receipt of the GTK. This message includes a MIC value.

While *MAC* is commonly used in cryptography to refer to a Message Authentication Code, the term *MIC* is used instead in connection with 802.11i because *MAC* has another standard meaning, Media Access Control, in networking.

Phase 4: Protected data transfer

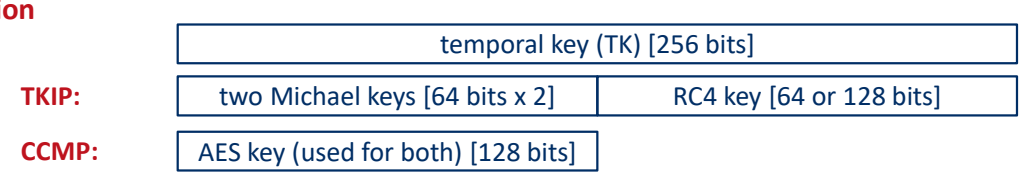
Temporal Key Integrity Protocol (TKIP)

- Hardware compatible with WEP devices, requiring only software changes
- Integrity: 64-bit MIC is generated using the Michael algorithm
- Confidentiality: MPDU+MIC are encrypted using RC4

Counter Mode-CBC MAC Protocol (CCMP)

- For newer devices, with dedicated hardware to perform the calculations
- Integrity: Provided by CBC-MAC
- Confidentiality: CTR block cipher mode combined with AES-128

Key generation



University of Antwerp | Faculty of Science

22

IEEE 802.11i defines two schemes for protecting data transmitted in 802.11 MPDUs: the Temporal Key Integrity Protocol (TKIP), and the Counter Mode-CBC MAC Protocol (CCMP).

TKIP is designed to require only software changes to devices that are implemented with the older wireless LAN security approach called Wired Equivalent Privacy (WEP). TKIP provides two services:

- **Message integrity:** TKIP adds a message integrity code (MIC) to the 802.11 MAC frame after the data field. The MIC is generated by an algorithm, called Michael, that computes a 64-bit value using as input the source and destination MAC address values and the Data field, plus key material.
- **Data confidentiality:** Data confidentiality is provided by encrypting the MPDU plus MIC value using RC4.

The 256-bit TK is employed as follows. Two 64-bit keys are used with the Michael message digest algorithm to produce a message integrity code. One key is used to protect STA-to-AP messages, and the other key is used to protect AP-to-STA messages. The remaining 128 bits are truncated to generate the RC4 key used to encrypt the transmitted data. For additional protection, a monotonically increasing TKIP sequence counter (TSC) is assigned to each frame. The TSC serves two purposes. First, the TSC is included with each MPDU and is protected by the MIC to protect against replay attacks. Second, the TSC is combined with the session TK to produce a dynamic encryption key that changes with each transmitted MPDU, thus making cryptanalysis more difficult.

CCMP is intended for newer IEEE 802.11 devices that are equipped with the hardware to support this scheme. As with TKIP, CCMP provides two services:

- **Message integrity:** CCMP uses the cipher block chaining message authentication code (CBC-

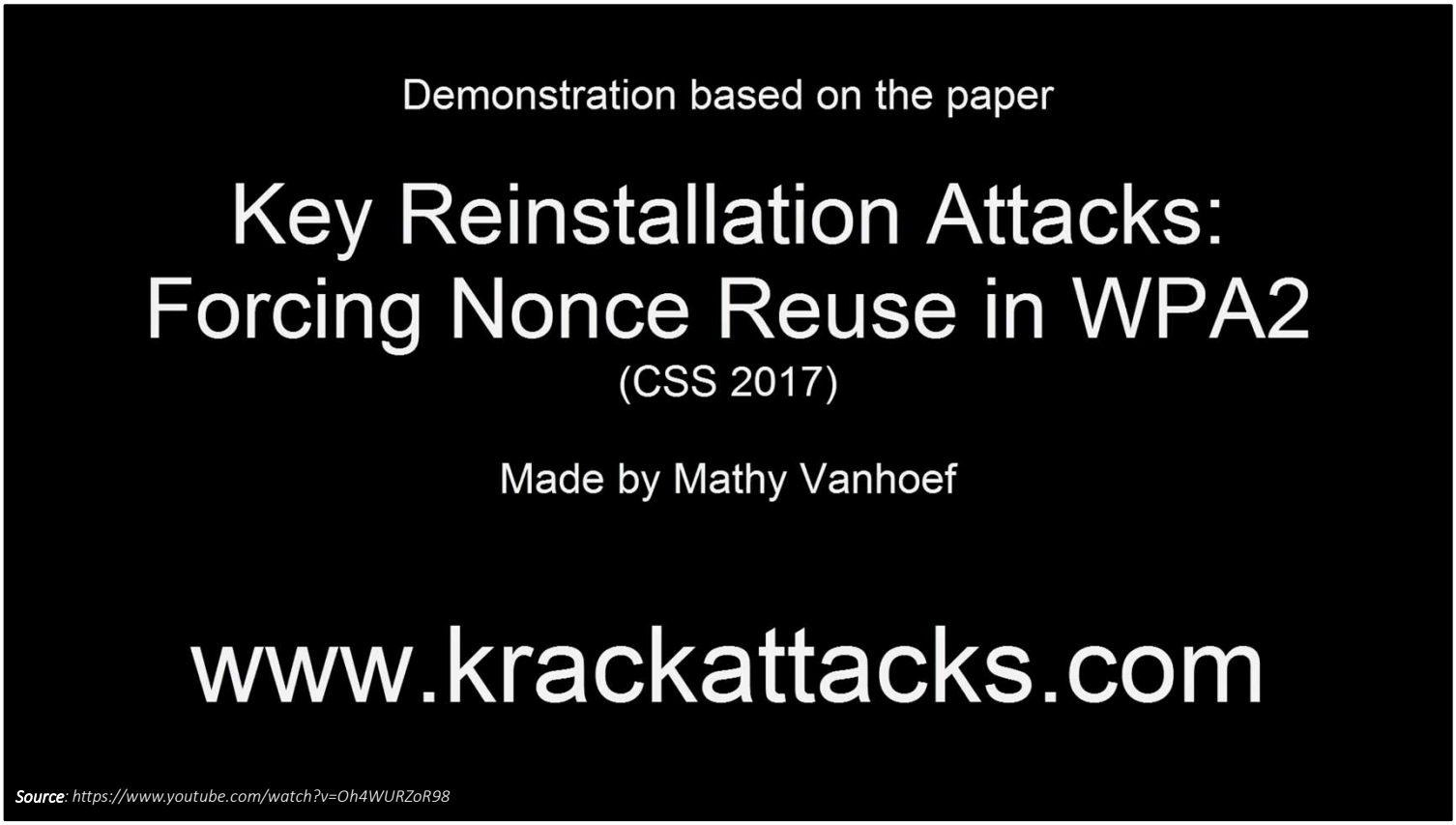
MAC).

- **Data confidentiality:** CCMP uses the CTR block cipher mode of operation with AES for encryption.

The same 128-bit AES key is used for both integrity and confidentiality. The scheme uses a 48-bit packet number to construct a nonce to prevent replay attacks.

Attacks against WPA

- **Dictionary and brute-force attacks**
  - Weak or predictable pre-shared keys (PSKs) are vulnerable
  - A truly random 20-character passphrase selected from the 95 permitted characters provides the best protection
- **Attacks against WPA-TKIP**
  - Some attacks against WEP can be applied to WPA-TKIP as well
  - Vanhoef and Piessens demonstrated an attack that can inject arbitrary packets with a payload of at most 112 bytes in 2013
  - WPA-TKIP is currently deemed insecure, but still widely used
- **Attacks against WPA2**
  - Vanhoef demonstrated the Key reinstallation attack (KRACK) in 2017, which forces key and nonce reuse by repeating messages 3 and 4 of handshake



***Source**: https://www.youtube.com/watch?v=Oh4WURZoR98*

KRACK: WPA2 Key Reinstallation Attacks

- Discovered by Prof. Mathy Vanhoef (KULeuven) in 2017
- Works against the WPA2 protocol itself and not specific implementations
- Tricks the victim into reinstalling an already-in-use-key
  - The associated nonce and replay counter are reset to initial values
  - In the Linux/Android wpa_supplicant 2.4 implementation, an all-zero encryption key is installed, instead of reinstalling the actual key, making problems worse
- If the same key is reused with the same nonce, and a message has known content, then it becomes possible to decrypt messages with the same nonce
  - e.g., decrypt TCP SYN packets to hijack TCP connections (e.g., to inject malware in HTTP)

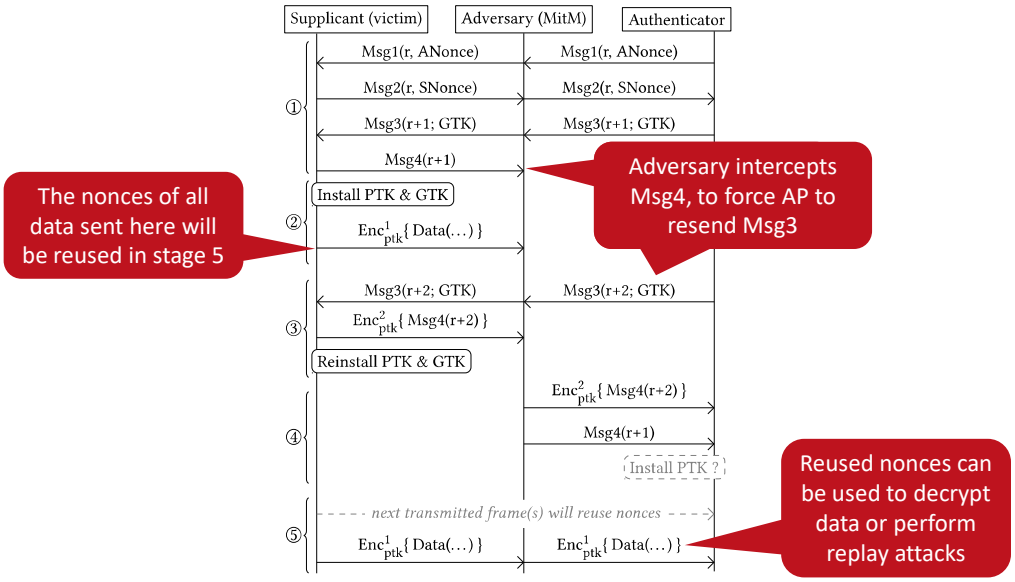
Implementation	Re. Msg3	Pt. EAPOL	Quick Pt.	Quick Ct.	4-way	Group
OS X 10.9.5	✓	✗	✗	✓	✓	✓
macOS Sierra 10.12	✓	✗	✗	✓	✓	✓
iOS 10.3.1 [c]	✗	N/A	N/A	N/A	✗	✓
`wpa_supplicant` v2.3	✓	✓	✓	✓	✓	✓
`wpa_supplicant` v2.4-5	✓	✓	✓	✓[a]	✓[a]	✓
`wpa_supplicant` v2.6	✓	✓	✓	✓[b]	✓[b]	✓
Android 6.0.1	✓	✗	✓	✓[a]	✓[a]	✓
OpenBSD 6.1 (rum)	✓	✗	✗	✗	✗	✓
OpenBSD 6.1 (iwn)	✓	✗	✗	✓	✓	✓
Windows 7 [c]	✗	N/A	N/A	N/A	✗	✓
Windows 10 [c]	✗	N/A	N/A	N/A	✗	✓
MediaTek	✓	✓	✓	✓	✓	✓

[a] Due to a bug, an all-zero TK will be installed, see Section 6.3.
[b] Only the group key is reinstalled in the 4-way handshake.
[c] Certain tests are irrelevant (not applicable) because the implementation does not accept retransmissions of message 3.

University of Antwerp
Faculty of Science

Source: https://www.krackattacks.com/

WPA2 Key Reinstallation Attack (KRACK) details



University of Antwerp | Faculty of Science

Source: https://papers.mathyvanhoef.com/ccs2017.pdf

WPA3 improvements

Simultaneous Authentication of Equals (SAE)

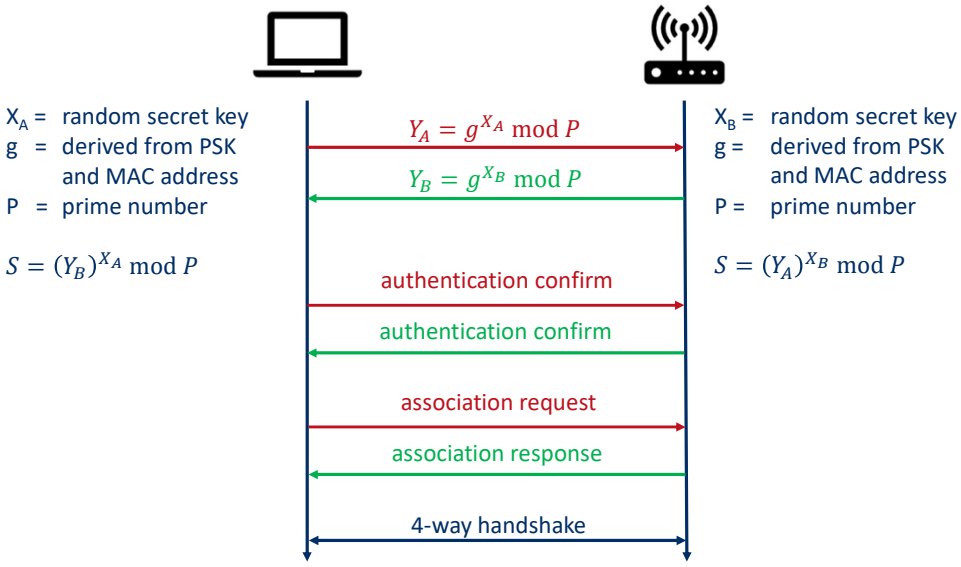
- Extends the Dragonfly handshake (RFC 7664), based on Diffie-Hellman, to derive pairwise master key (PMK) from pre-shared key (PSK)
- PMK is secure against offline dictionary attacks (in contrast to PSK)
- Offers forward secrecy (if an attacker learns the PSK, they cannot use that to decrypt old traffic, as the PMK cannot be easily determined)

Other improvements

- Wi-Fi device provisioning protocol (DPP) replaces insecure WPS to simplify configuration of devices without displays (e.g., sensors)
- Unauthenticated encryption protects open Wi-Fi networks against passive eavesdropping attacks (e.g., public hotspots)
- Increased session key sizes (i.e., AES-GCM256, HMAC-SHA384)

More details: https://www.mathyvanhoef.com/2018/03/wpa3-technical-details.html

Simultaneous authentication of equals (SAE)

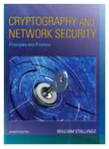


Vulnerabilities in WPA3 SAE

- Exposed by Vanhoef & Ronen in 2019
- Attack 1: Downgrade attacks
  - Forces clients to revert to WPA2 backwards compatibility mode
  - Client becomes vulnerable to known WPA2 vulnerabilities
- Attack 2: Denial of Service (DoS) attack
  - Overload CPU resources of access point
- Attack 3: Side-channel attacks
  - Timing side-channel attack: Exploits timing information in hashing algorithms
  - Caching side-channel attack: Exploit information about cache operations to speed up offline brute force attacks
- More details: https://wpa3.mathyvanhoef.com/

Summary on Wi-Fi security

- Wireless networks are more vulnerable to attacks due to their nature
- IEEE 802.11i standardizes WPA2, which overcomes the vulnerabilities in WEP
  - Uses 802.1x and EAP for authentication
  - Uses AES and HMAC for confidentiality and integrity
- WPA2 is vulnerable to certain attacks (e.g., KRACK)
- WPA3 introduces simultaneous authentication of equals (SAE)
  - SAE is based on Dragonfly handshake (based on Diffie-Hellman key exchange)
  - PMK is calculated based on PSK and MAC address, rather than using them directly
  - Not vulnerable to KRACK and offline dictionary attacks (as PMK cannot be easily guessed)
  - Provides forward secrecy, as leaked password does not lead to determining PMK
- WPA3 already has known vulnerabilities



Link with the book

- Chapter 18

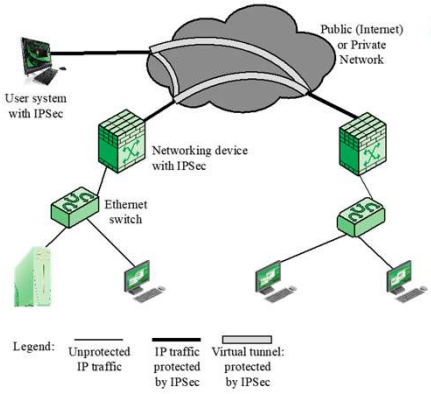
1 Wi-Fi Security (802.11i)

2 IP-Layer Security (IPSec)

University of Antwerp
Faculty of Science

IP layer security (IPsec)

- Provides end-to-end authentication, confidentiality and key management between hosts and/or networks on top of the IPv4 and IPv6 protocols
- Widely used for Virtual Private Network (VPN) security and tunneling
- Standardized by IETF in 1995



Several useful scenarios:

- Connect remote company sites over the public Internet
- Safely connect over a public (unencrypted) Wi-Fi hotspot
- Safely connect to school or company network from home

University of Antwerp | Faculty of Science

32

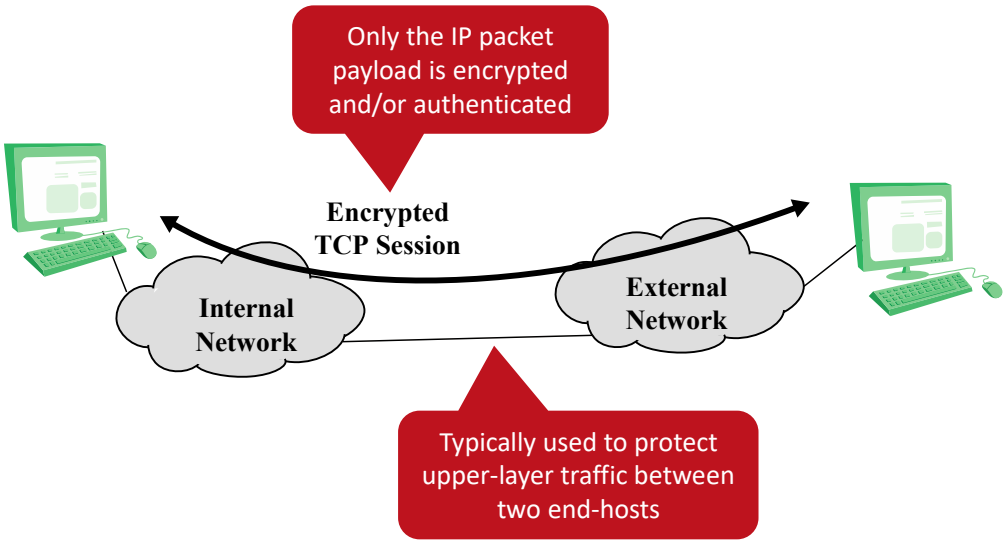
IP-level security encompasses three functional areas: authentication, confidentiality, and key management. The **authentication** mechanism assures that a received packet was, in fact, transmitted by the party identified as the source in the packet header. In addition, this mechanism assures that the packet has not been altered in transit. The **confidentiality** facility enables communicating nodes to encrypt messages to prevent eavesdropping by third parties. The **key management** facility is concerned with the secure exchange of keys.

In 1994, the Internet Architecture Board (IAB) issued a report titled “Security in the Internet Architecture” (RFC 1636). The report identified key areas for security mechanisms. Among these were the need to secure the network infrastructure from unauthorized monitoring and control of network traffic and the need to secure end-user-to-end-user traffic using authentication and encryption mechanisms. To provide security, the IAB included authentication and encryption as necessary security features in the next-generation IP, which has been issued as IPv6. Fortunately, these security capabilities were designed to be usable both with the current IPv4 and the future IPv6. This means that vendors can begin offering these features now, and many vendors now do have some IPsec capability in their products. The IPsec specification now exists as a set of Internet standards.

The figure shows a typical scenario of IPsec usage. An organization maintains LANs at dispersed locations. Non-secure IP traffic is conducted on each LAN. For traffic offsite, through some sort of private or public WAN, IPsec protocols are used. These protocols operate in networking devices, such as a router or firewall, that connect each LAN to the outside world. The IPsec networking device will typically encrypt all traffic going into the WAN and decrypt traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN. Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPsec protocols to provide security.

IPsec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services. Two protocols are used to provide security: an authentication protocol designated by the header of the protocol, Authentication Header (AH); and a combined encryption/authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP).

First mode of operation: Transport mode



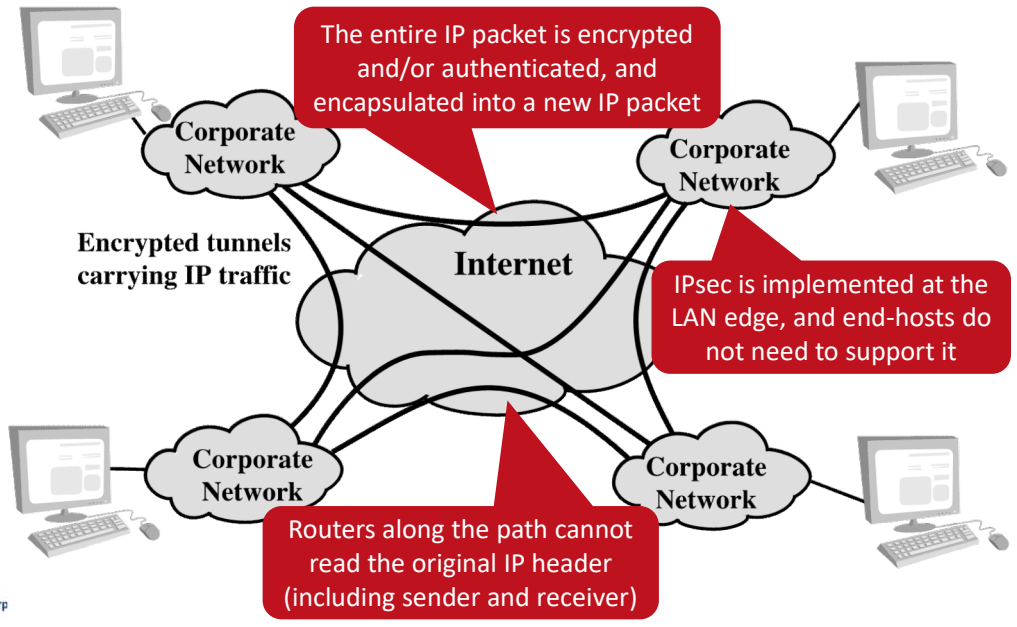
University of Antwerp | Faculty of Science

33

Transport mode provides protection primarily for upper-layer protocols. That is, transport mode protection extends to the payload of an IP packet.1 Examples include a TCP or UDP segment or an ICMP packet, all of which operate directly above IP in a host protocol stack. Typically, transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations). When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header. For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.

ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header. AH in transport mode authenticates the IP payload and selected portions of the IP header.

Second mode of operation: Tunnel mode

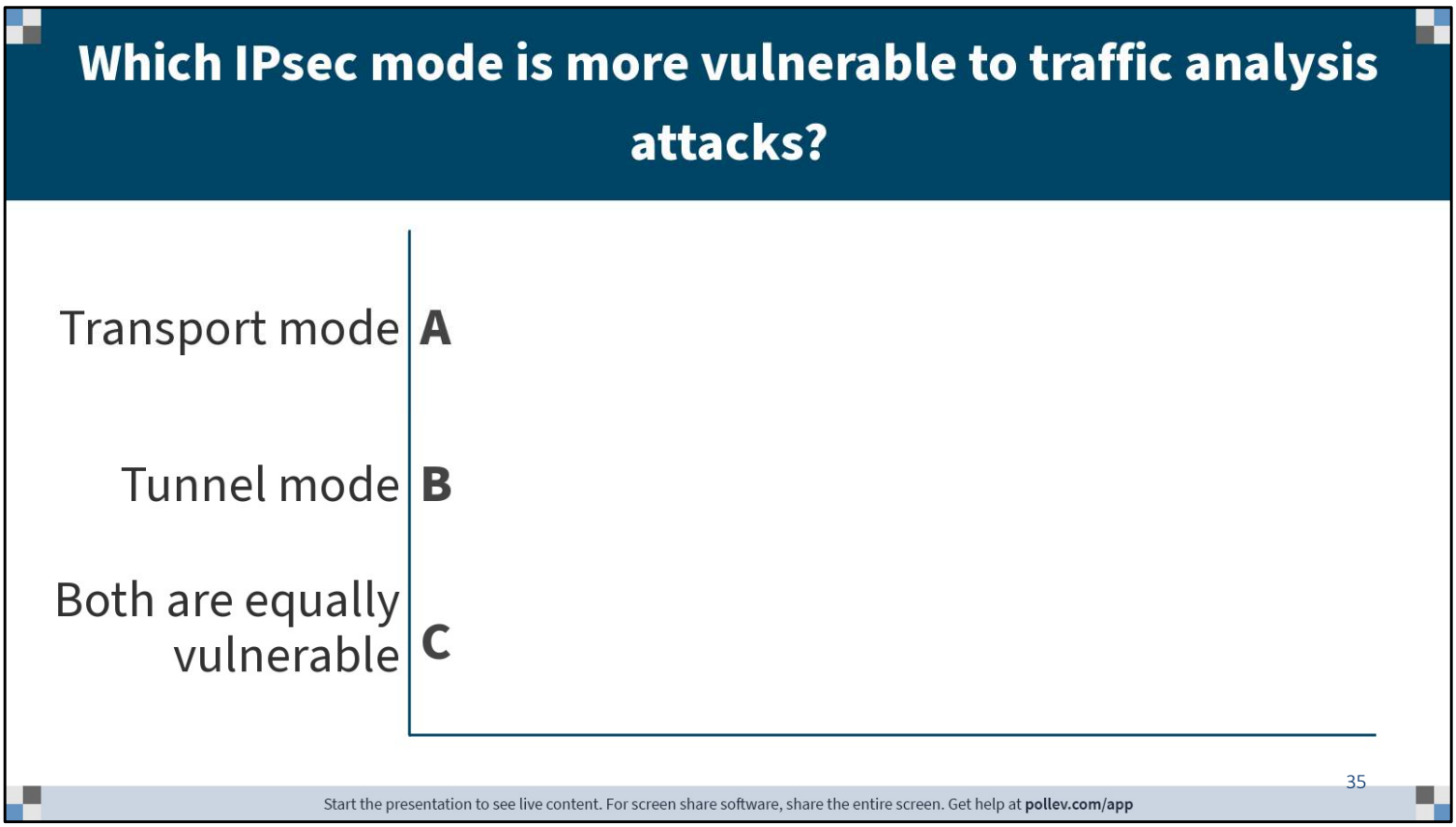


University of Antwerp | Faculty of Science

34

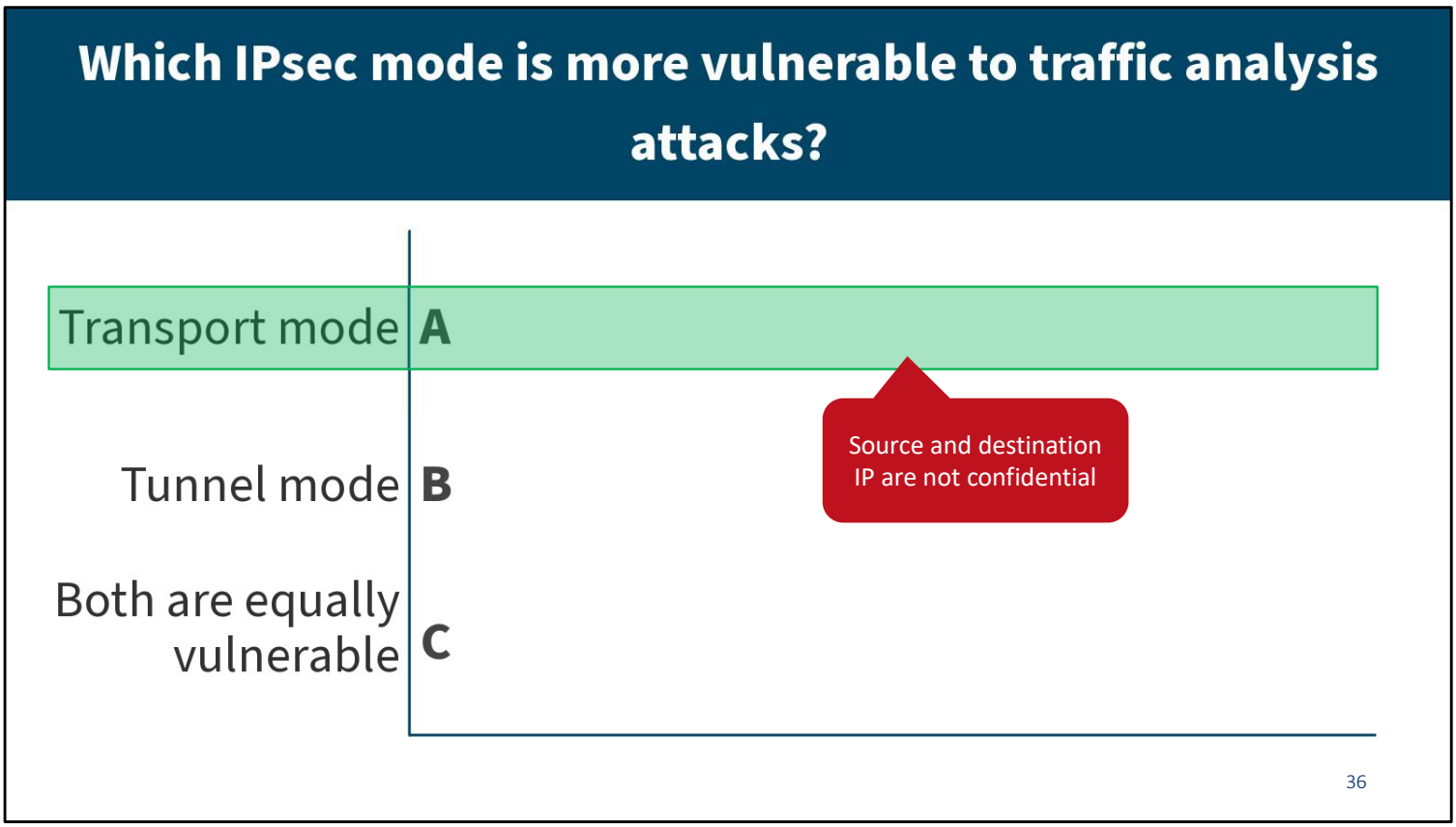
Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new outer IP packet with a new outer IP header. The entire original, inner, packet travels through a tunnel from one point of an IP network to another; no routers along the way are able to examine the inner IP header. Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security. Tunnel mode is used when one or both ends of a security association (SA) are a security gateway, such as a firewall or router that implements IPsec. With tunnel mode, a number of hosts on networks behind firewalls may engage in secure communications without implementing IPsec. The unprotected packets generated by such hosts are tunneled through external networks by tunnel mode SAs set up by the IPsec software in the firewall or secure router at the boundary of the local network.

ESP in tunnel mode encrypts and optionally authenticates the entire inner IP packet, including the inner IP header. AH in tunnel mode authenticates the entire inner IP packet and selected portions of the outer IP header.



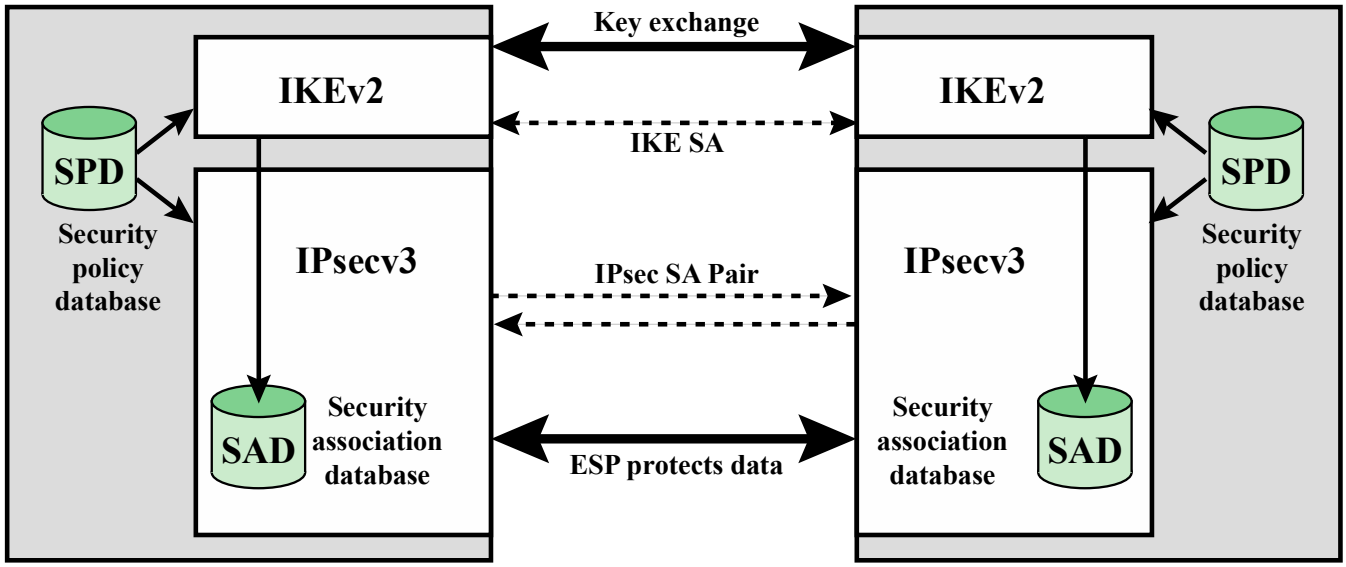
Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Which IPsec mode is more vulnerable to traffic analysis attacks?
https://www.polleverywhere.com/multiple_choice_polls/PEsDCdOeL3DMgnI?state=opened&flow=Default&onscreen=persist



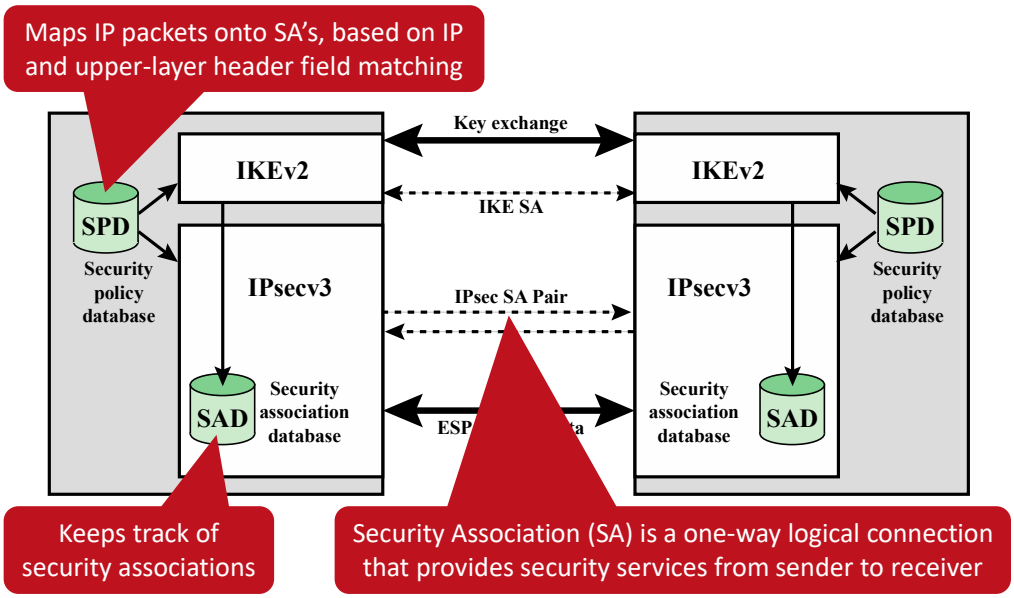
Poll Title: Which IPsec mode is more vulnerable to traffic analysis attacks?

IPsec architecture



Fundamental to the operation of IPsec is the concept of a security policy applied to each IP packet that transits from a source to a destination. IPsec policy is determined primarily by the interaction of two databases, the **security association database (SAD)** and the **security policy database (SPD)**.

IPsec architecture



University of Antwerp | Faculty of Science

38

A key concept that appears in both the authentication and confidentiality mechanisms for IP is the security association (SA). An association is a one-way logical connection between a sender and a receiver that affords security services to the traffic carried on it. If a peer relationship is needed for two-way secure exchange, then two security associations are required. In any IP packet, the security association is uniquely identified by the Destination Address in the IPv4 or IPv6 header and the Security Parameters Index (SPI), a unique 32-bit identifier of the SA, in the enclosed extension header (AH or ESP).

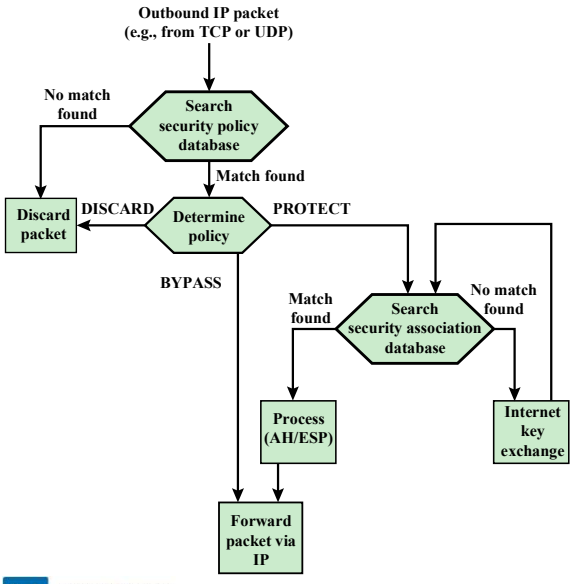
In each IPsec implementation, there is a nominal Security Association Database (SAD) that defines the parameters associated with each SA. The key management mechanism that is used to distribute keys is coupled to the authentication and privacy mechanisms only by way of the Security Parameters Index (SPI). Hence, authentication and privacy have been specified independent of any specific key management mechanism.

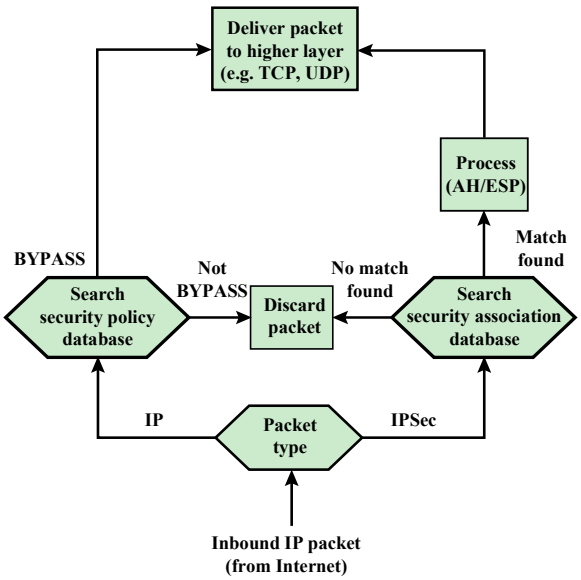
The means by which IP traffic is related to specific SAs (or no SA in the case of traffic allowed to bypass IPsec) is the nominal Security Policy Database (SPD). In its simplest form, an SPD contains entries, each of which defines a subset of IP traffic and points to an SA for that traffic. In more complex environments, there may be multiple entries that potentially relate to a single SA or multiple SAs associated with a single SPD entry. Each SPD entry is defined by a set of IP and upper-layer protocol field values, called *selectors*. In effect, these selectors are used to filter outgoing traffic in order to map it into a particular SA. Possible selectors include: remote IP address, local IP address, next layer protocol, local port, and remote port.

IPsec is executed on a packet-by-packet basis. When IPsec is implemented, each outbound IP packet is processed by the IPsec logic before transmission, and each inbound packet is processed by the IPsec logic after reception and before passing the packet contents on to the next higher

layer (e.g., TCP or UDP).

IPsec processes packets one by one





University of Antwerp
Faculty of Science

39

A block of data from a higher layer, such as TCP, is passed down to the IP layer and an IP packet is formed, consisting of an IP header and an IP body. Then the following steps occur:

1. IPsec searches the SPD for a match to this packet.
2. If no match is found, then the packet is discarded and an error message is generated.
3. If a match is found, further processing is determined by the first matching entry in the SPD. If the policy for this packet is DISCARD, then the packet is discarded. If the policy is BYPASS, then there is no further IPsec processing; the packet is forwarded to the network for transmission.
4. If the policy is PROTECT, then a search is made of the SAD for a matching entry. If no entry is found, then IKE is invoked to create an SA with the appropriate keys and an entry is made in the SA.
5. The matching entry in the SAD determines the processing for this packet. Either encryption, authentication, or both can be performed, and either transport or tunnel mode can be used. The packet is then forwarded to the network for transmission.

An incoming IP packet triggers the IPsec processing. The following steps occur:

1. IPsec determines whether this is an unsecured IP packet or one that has ESP or AH headers/trailers, by examining the IP Protocol field (IPv4) or Next Header field (IPv6).
2. If the packet is unsecured, IPsec searches the SPD for a match to this packet. If the first matching entry has a policy of BYPASS, the IP header is processed and stripped off and the packet body is delivered to the next higher layer, such as TCP. If the first matching entry has a policy of PROTECT or DISCARD, or if there is no matching entry, the packet is discarded.
3. For a secured packet, IPsec searches the SAD. If no match is found, the packet is discarded. Otherwise, IPsec applies the appropriate ESP or AH processing. Then, the IP header is processed and stripped off and the packet body is delivered to the next higher layer, such as TCP.

Encapsulating security payload (ESP)

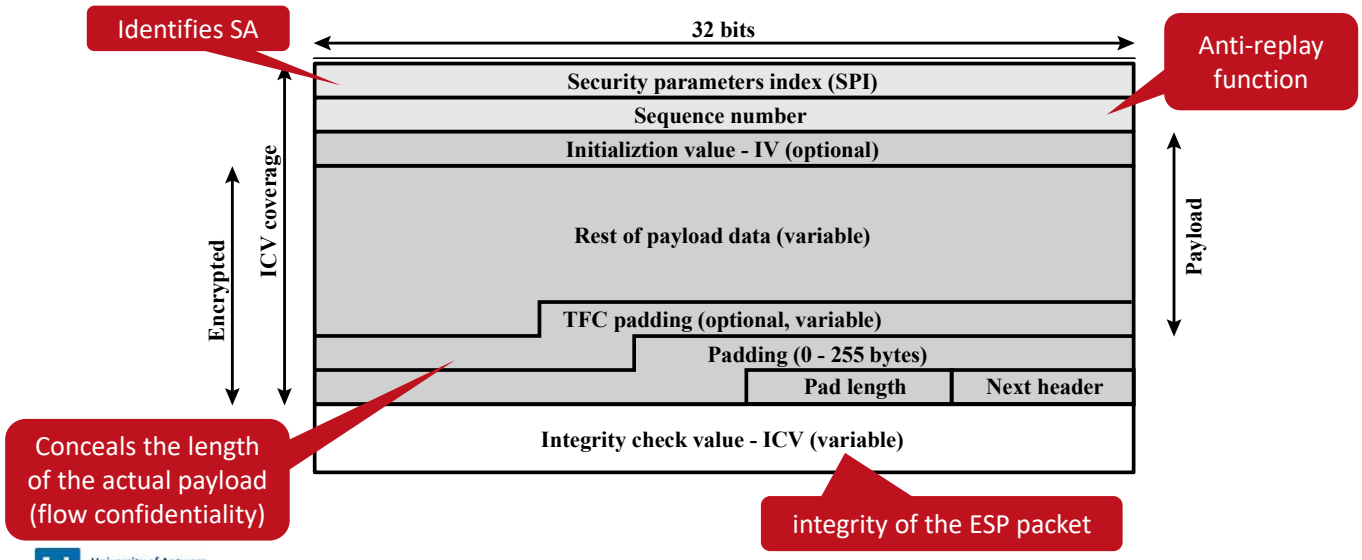
- One of two alternative security mechanisms in IPsec, which provides confidentiality and optionally authentication
- The alternative Authentication Header (AH) only provides authentication
- Can be used in combination with a wide range of encryption and authentication algorithms

Security services

- Packet confidentiality
- Data origin authentication
- Connection-less integrity
- Anti-replay service
- (limited) Traffic flow confidentiality

ESP can be used to provide confidentiality, data origin authentication, connection- less integrity, an anti-replay service (a form of partial sequence integrity), and (limited) traffic flow confidentiality. The set of services provided depends on options selected at the time of Security Association (SA) establishment and on the location of the implementation in a network topology. ESP can work with a variety of encryption and authentication algorithms.

ESP packet format



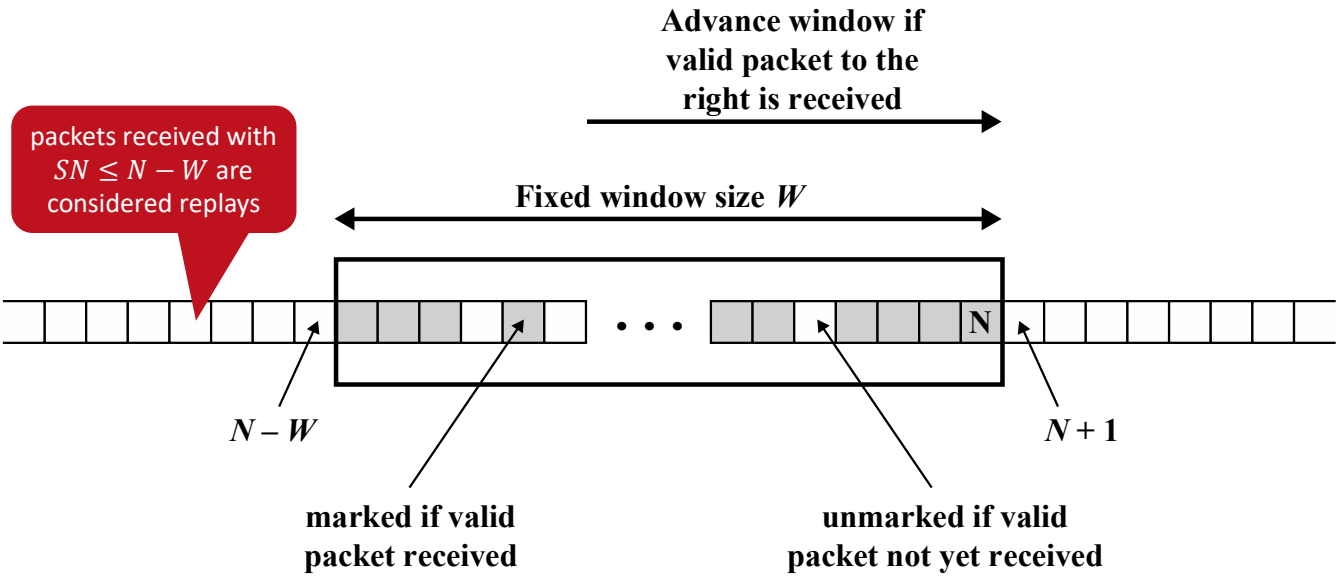
University of Antwerp | Faculty of Science

41

An ESP packet contains the following fields:

- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value; this provides an anti-replay function.
- **Initialization value or nonce (32 bits)**: Is present if this is required by the encryption or authenticated encryption algorithm used for ESP.
- **Payload Data (variable):** This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.
- **Traffic flow confidentiality (TFC) padding (variable)**: If tunnel mode is being used, then the IPsec implementation may add TFC padding after the Payload Data and before the Padding field. This padding is used to conceal the actual length of the payload to provide traffic flow confidentiality.
- **Padding (0–255 bytes):** To ensure that the data is a multiple of the encryption algorithm block size, and that the pad length and next header fields are aligned to the right of a 32-bit word.
- **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.
- **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that payload (e.g., an extension header in IPv6, or an upper-layer protocol such as TCP).
- **Integrity Check Value (variable):** A variable-length field (must be an integral number of 32-bit words) that contains the Integrity Check Value computed over the ESP packet minus the Authentication Data field.

Anti-replay function



University of Antwerp | Faculty of Science

42

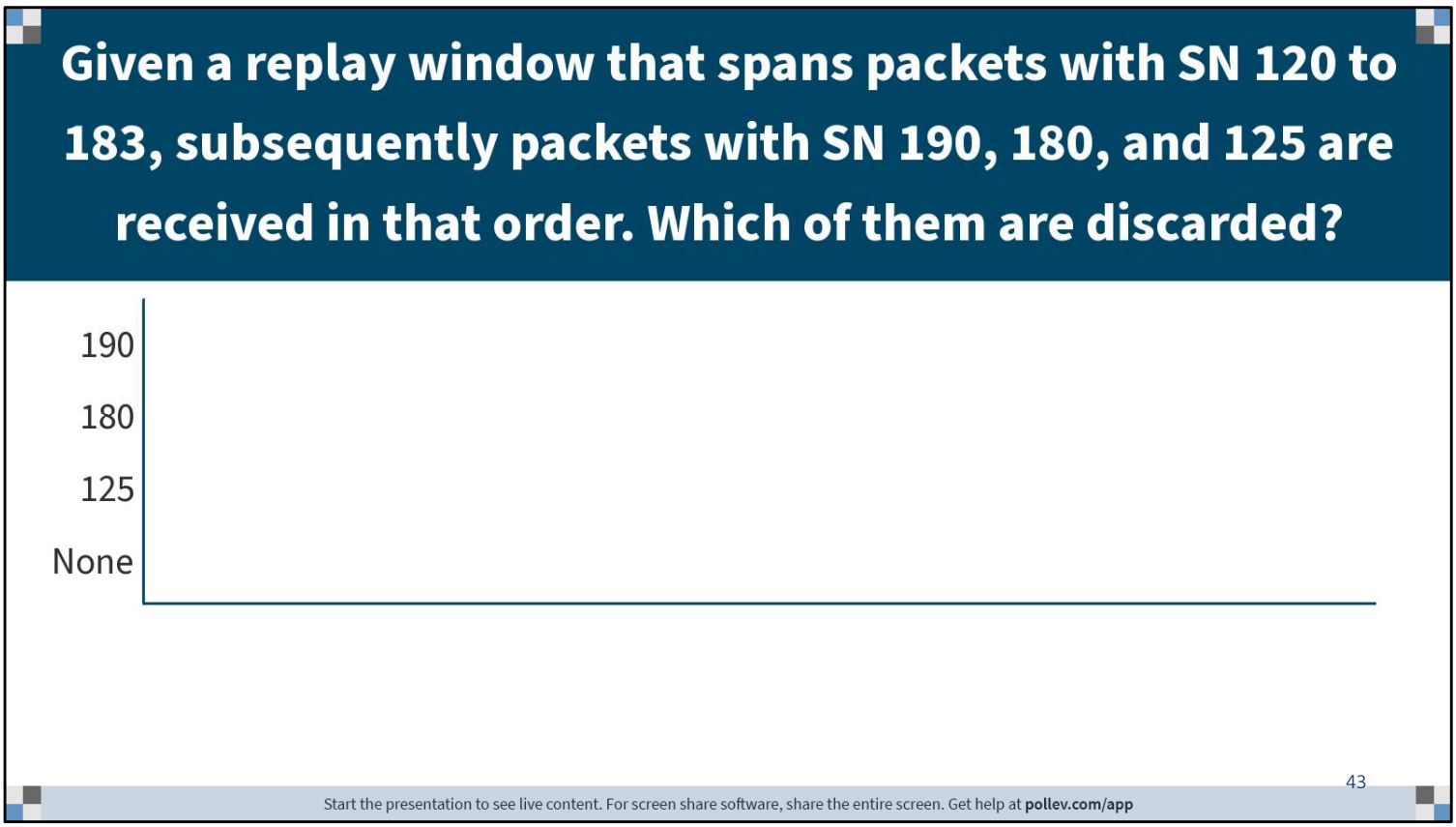
A **replay attack** is one in which an attacker obtains a copy of an authenticated packet and later transmits it to the intended destination. The receipt of duplicate, authenticated IP packets may disrupt service in some way or may have some other undesired consequence. The Sequence Number field is designed to thwart such at- tacks. First, we discuss sequence number generation by the sender, and then we look at how it is processed by the recipient.

When a new SA is established, the **sender** initializes a sequence number counter to 0. Each time that a packet is sent on this SA, the sender increments the counter and places the value in the Sequence Number field. Thus, the first value to be used is 1. If anti-replay is enabled (the default), the sender must not allow the sequence number to cycle past 2^{32} - 1 back to zero. Otherwise, there would be multiple valid packets with the same sequence number. If the limit of 2^{32} - 1 is reached, the sender should terminate this SA and negotiate a new SA with a new key.

Because IP is a connectionless, unreliable service, the protocol does not guarantee that packets will be delivered in order and does not guarantee that all packets will be delivered. Therefore, the IPsec authentication document dictates that the **receiver** should implement a window of size *W*, with a default of *W* = 64. The right edge of the window represents the highest sequence number, *N*, so far received for a valid packet. For any packet with a sequence number in the range from *N* - *W* + 1 to *N* that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked. Inbound processing proceeds as follows when a packet is received:

1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.
2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.

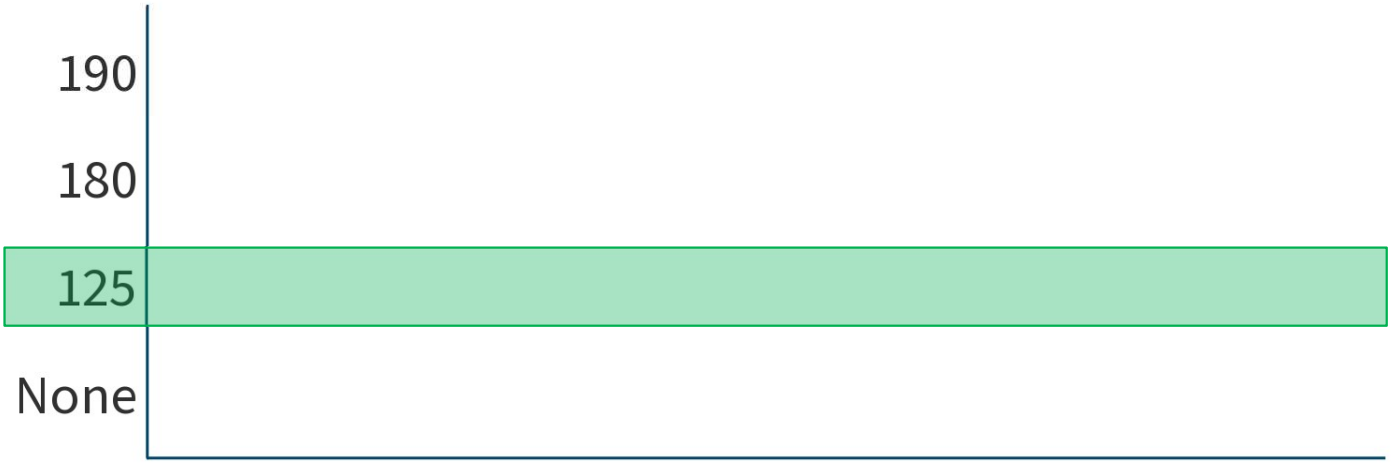
3. If the received packet is to the left of the window or if authentication fails, the packet is discarded; this is an auditable event.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

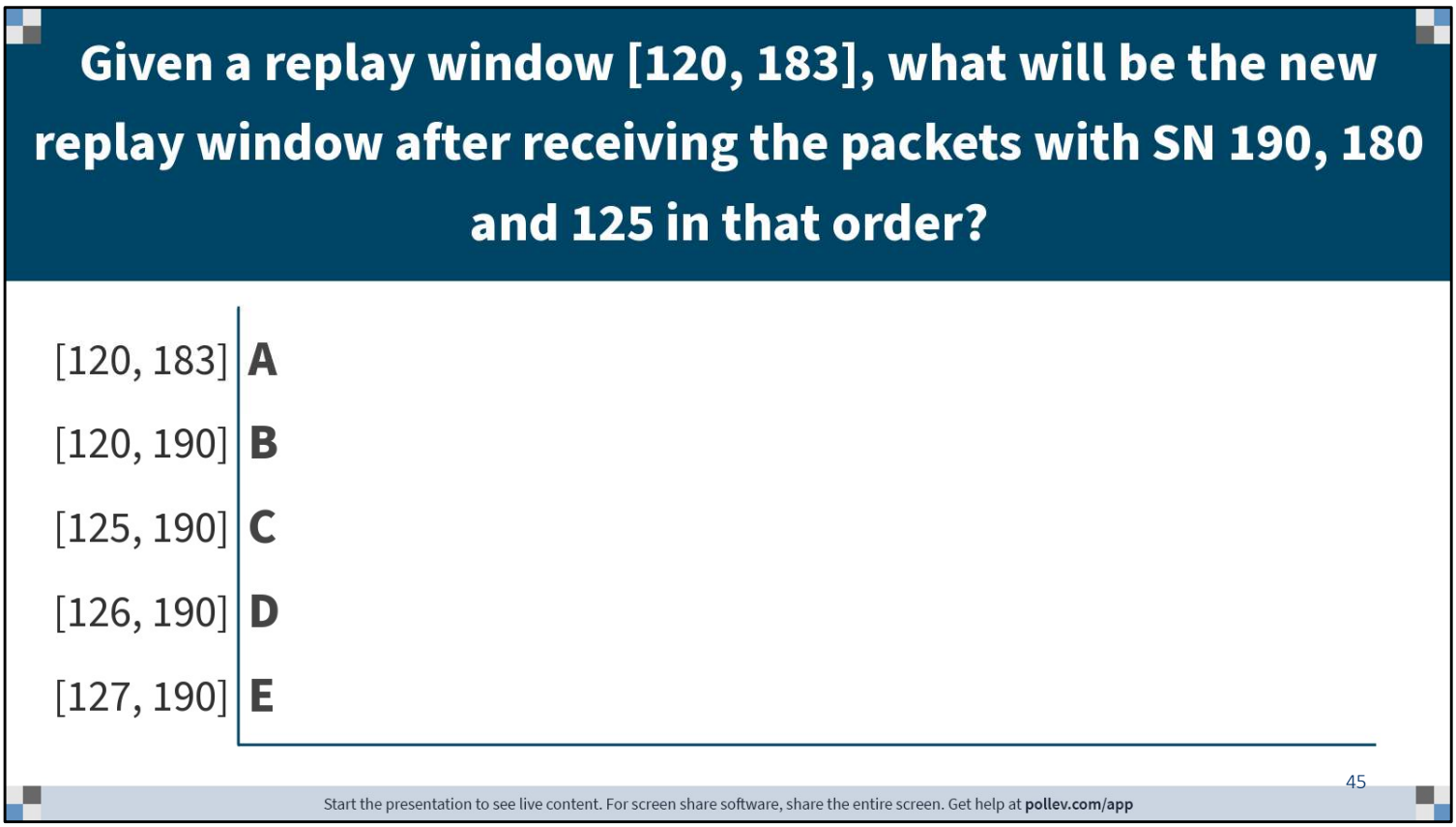
Given a replay window that spans packets with SN 120 to 183, subsequently packets with SN 190, 180, and 125 are received in that order. Which of them are discarded?
https://www.polleverywhere.com/multiple_choice_polls/FlxgZlxvhZvHWlW

Given a replay window that spans packets with SN 120 to 183, subsequently packets with SN 190, 180, and 125 are received in that order. Which of them are discarded?



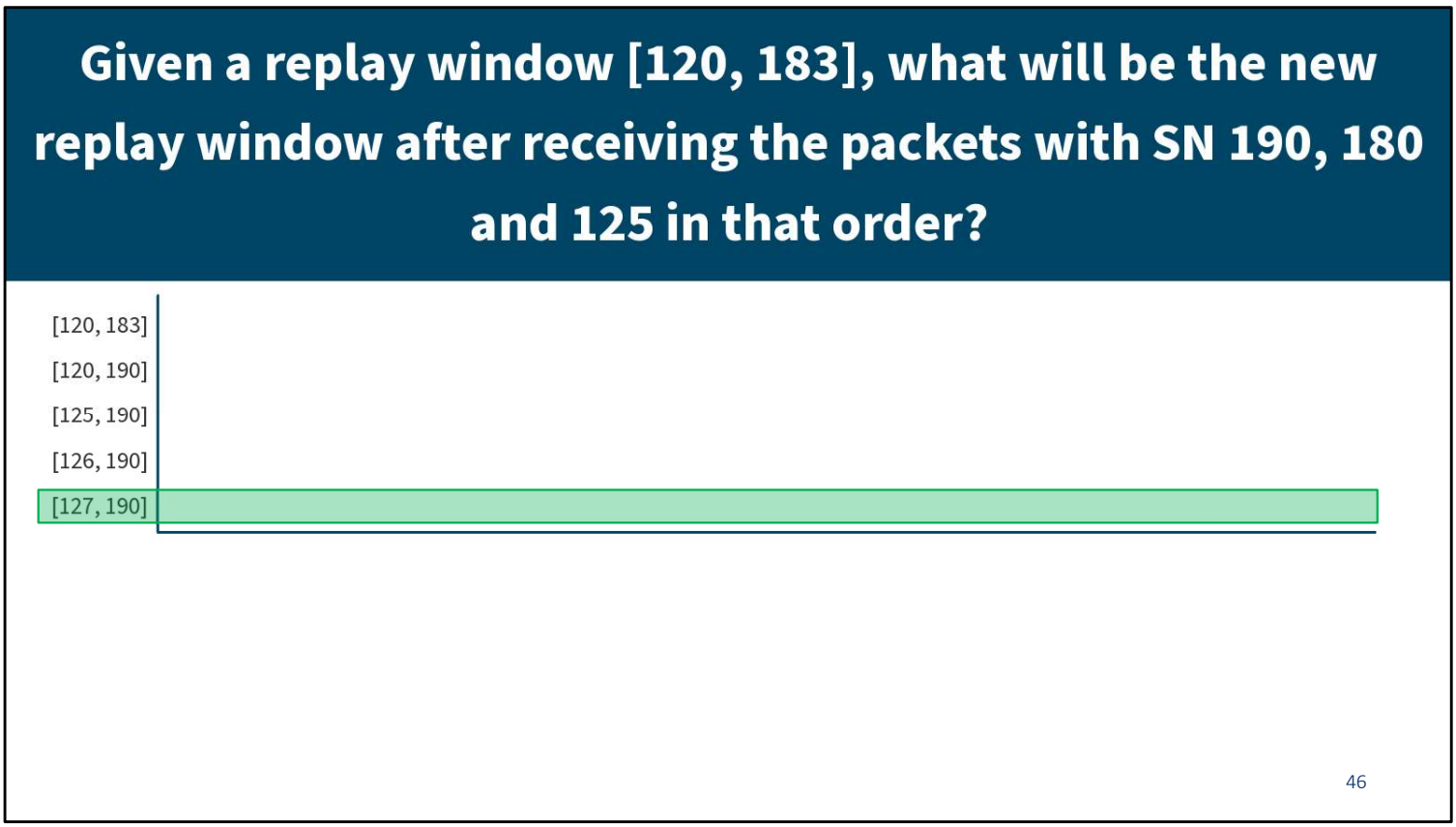
44

Poll Title: Given a replay window that spans packets with SN 120 to 183, subsequently packets with SN 190, 180, and 125 are received in that order. Which of them are discarded?



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Given a replay window [120, 183], what will be the new replay window after receiving the packets with SN 190, 180 and 125 in that order?
https://www.polleverywhere.com/multiple_choice_polls/GPZolzAJjIGTM2J

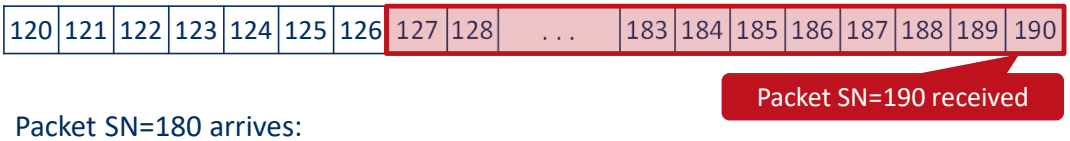


Poll Title: Given a replay window [120, 183], what will be the new replay window after receiving the packets with SN 190, 180 and 125 in that order?

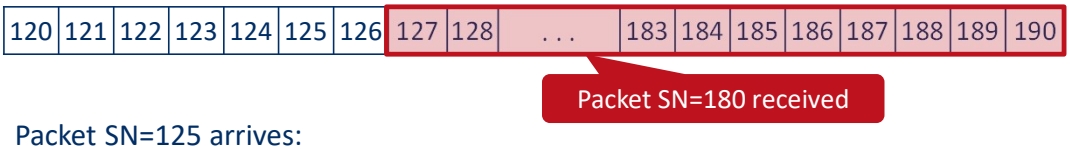
Replay window example

Initial replay window:

Packet SN=190 arrives:



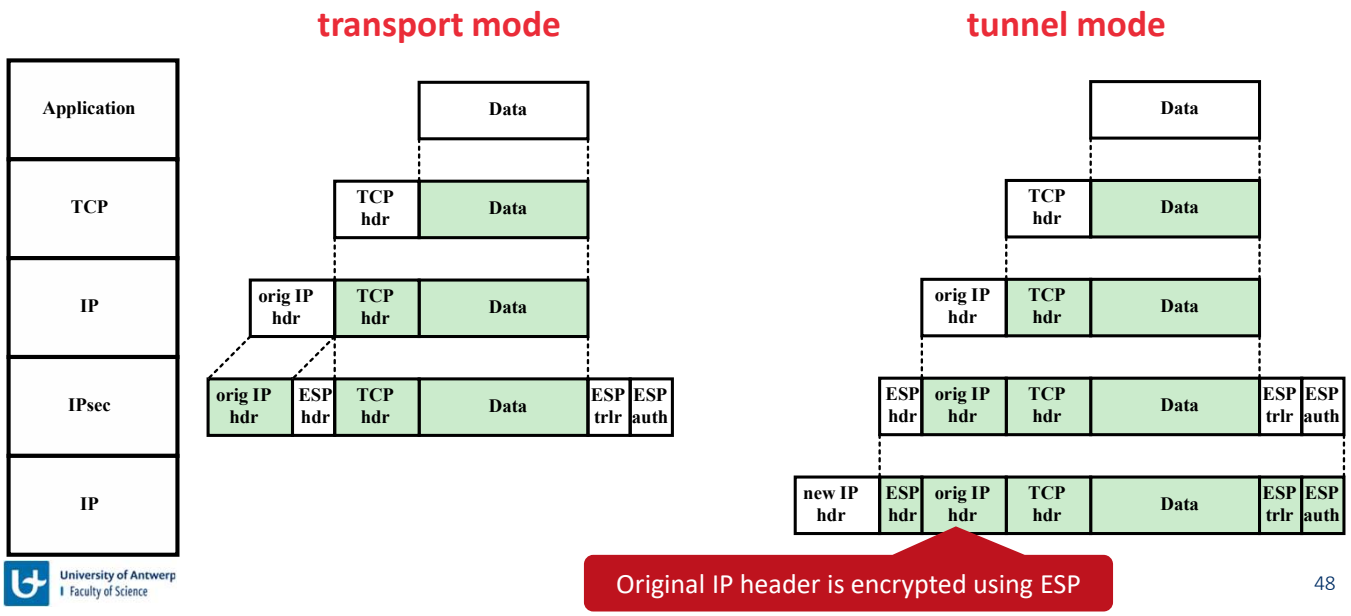
Packet SN=180 arrives:



Packet SN=125 arrives:

University of Antwerp
Faculty of Science

Packet encapsulation in transport and tunnel modes



Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP (e.g., a TCP segment). For this mode using IPv4, the ESP header is inserted into the IP packet immediately prior to the transport-layer header (e.g., TCP, UDP, ICMP), and an ESP trailer (Padding, Pad Length, and Next Header fields) is placed after the IP packet. If authentication is selected, the ESP Authentication Data field is added after the ESP trailer. The entire transport-level segment plus the ESP trailer are encrypted. Authentication covers all of the ciphertext plus the ESP header.

In the context of IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers. Therefore, the ESP header appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers. The destination options extension header could appear before or after the ESP header, depending on the semantics desired. For IPv6, encryption covers the entire transport-level segment plus the ESP trailer plus the destination options extension header if it occurs after the ESP header. Again, authentication covers the ciphertext plus the ESP header.

Tunnel mode ESP is used to encrypt an entire IP packet. For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted. This method can be used to counter traffic analysis. Because the IP header contains the destination address and possibly source routing directives and hop-by-hop option information, it is not possible simply to transmit the encrypted IP packet prefixed by the ESP header. Intermediate routers would be unable to process such a packet. Therefore, it is necessary to encapsulate the entire block (ESP header plus ciphertext plus

Authentication Data, if present) with a new IP header that will contain sufficient information for routing but not for traffic analysis.

Whereas the transport mode is suitable for protecting connections between hosts that support the ESP feature, the tunnel mode is useful in a configuration that includes a firewall or other sort of security gateway that protects a trusted network from external networks. In this latter case, encryption occurs only between an external host and the security gateway or between two security gateways. This relieves hosts on the internal network of the processing burden of encryption and simplifies the key distribution task by reducing the number of needed keys. Further, it thwarts traffic analysis based on ultimate destination.

Internet Key Exchange (IKE)

Goal: Determination and distribution of secret keys for confidentiality and integrity

Key exchange methods

- Manual: Sysadmin manually configures each system with the keys
- Automated: On-demand creation of keys for SAs

Default automated key exchange method is ISAKMP/Oakley

- Oakley Key Determination Protocol: Key exchange protocol based on Diffie-Hellman
- Internet Security Association and Key Management Protocol (ISAKMP): Protocol for the negotiation of security attributes

The key management portion of IPsec involves the determination and distribution of secret keys. A typical requirement is four keys for communication between two applications: transmit and receive pairs for both integrity and confidentiality. The IPsec Architecture document mandates support for two types of key management:

- **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems. This is practical for small, relatively static environments.
- **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.

The default automated key management protocol for IPsec is referred to as ISAKMP/Oakley and consists of the following elements:

- **Oakley Key Determination Protocol:** Oakley is a key exchange protocol based on the Diffie–Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.
- **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

ISAKMP by itself does not dictate a specific key exchange algorithm; rather, ISAKMP consists of a set of message types that enable the use of a variety of key exchange algorithms. Oakley is the specific key exchange algorithm mandated for use with the initial version of ISAKMP. In IKEv2, the terms Oakley and ISAKMP are no longer used, and there are significant differences from the use of Oakley and ISAKMP in IKEv1. Nevertheless, the basic functionality is the same.

IPsec cryptographic algorithms

- IPsec can be used in combination with a wide variety of algorithms
- RFC 4308 and RFC 6379 define two alternative possible suites

RFC 4308

	VPN-A	VPN-B
ESP encryption	3DES-CBC	AES-CBC (128-bit key)
ESP integrity	HMAC-SHA1-96	AES-XCBC-MAC-96
IKE encryption	3DES-CBC	AES-CBC (128-bit key)
IKE PRF	HMAC-SHA1	AES-XCBC-PRF-128
IKE Integrity	HMAC-SHA1-96	AES-XCBC-MAC-96
IKE DH group	1024-bit MODP	2048-bit MODP

RFC 6379 (optional)

	GCM-128	GCM-256	GMAC-128	GMAC-256
ESP encryption/ Integrity	AES-GCM (128-bit key)	AES-GCM (256-bit key)	Null	Null
ESP integrity	Null	Null	AES-GMAC (128-bit key)	AES-GMAC (256-bit key)
IKE encryption	AES-CBC (128-bit key)	AES-CBC (256-bit key)	AES-CBC (128-bit key)	AES-CBC (256-bit key)
IKE PRF	HMAC-SHA-256	HMAC-SHA-384	HMAC-SHA-256	HMAC-SHA-384
IKE Integrity	HMAC-SHA-256-128	HMAC-SHA-384-192	HMAC-SHA-256-128	HMAC-SHA-384-192
IKE DH group	256-bit random ECP	384-bit random ECP	256-bit random ECP	384-bit random ECP

University of Antwerp | Faculty of Science

50

The IPsecv3 and IKEv2 protocols rely on a variety of types of cryptographic algorithms. As we have seen in this book, there are many cryptographic algorithms of each type, each with a variety of parameters, such as key size. To promote interoperability, two RFCs define recommended suites of cryptographic algorithms and parameters for various applications.

RFC 4308 defines two cryptographic suites for establishing virtual private networks. Suite VPN-A matches the commonly used corporate VPN security used in older IKEv1 implementations at the time of the issuance of IKEv2 in 2005. Suite VPN-B provides stronger security and is recommended for new VPNs that implement IPsecv3 and IKEv2.

The top table lists the algorithms and parameters for the two suites. There are several points to note about these two suites. Note that for symmetric cryptography, VPN-A relies on 3DES and HMAC, while VPN-B relies exclusively on AES. Three types of secret-key algorithms are used:

- **Encryption:** For encryption, the cipher block chaining (CBC) mode is used.
- **Message authentication:** For message authentication, VPN-A relies on HMAC with SHA-1 with the output truncated to 96 bits. VPN-B relies on a variant of CMAC with the output truncated to 96 bits.
- **Pseudorandom function:** IKEv2 generates pseudorandom bits by repeated use of the MAC used for message authentication.

RFC 6379 defines four optional cryptographic suites that are compatible with the United States National Security Agency's (NSA) Suite B specifications. In 2005, the NSA issued Suite B, which defined the algorithms and strengths needed to protect both sensitive but unclassified (SBU) and classified information for use in its Cryptographic Modernization program. The four suites defined in RFC 6379 provide choices for ESP and IKE. The four suites are differentiated by the choice of cryptographic algorithm strengths and a choice of whether ESP is to provide both

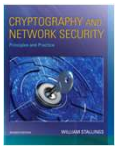
confidentiality and integrity or integrity only. All of the suites offer greater protection than the two VPN suites defined in RFC 4308. The bottom table lists the algorithms and parameters for the two suites. As with RFC 4308, three categories of secret key algorithms are listed:

- **Encryption:** For ESP, authenticated encryption is provided using the GCM mode with either 128-bit or 256-bit AES keys. For IKE encryption, CBC is used, as it was for the VPN suites.
- **Message authentication:** For ESP, if only authentication is required, then GMAC is used. GMAC is simply the authentication portion of GMC. For IKE, message authentication is provided using HMAC with one of the SHA-3 hash functions.
- **Pseudorandom function:** As with the VPN suites, IKEv2 in these suites generates pseudorandom bits by repeated use of the MAC used for message authentication.

For the Diffie–Hellman algorithm, the use of elliptic curve groups modulo a prime is specified. For authentication, elliptic curve digital signatures are listed. The original IKEv2 documents used RSA-based digital signatures. Equivalent or greater strength can be achieved using ECC with fewer key bits.

Summary on IP-layer security (IPSec)

- Provides end-to-end authentication and confidentiality between hosts
- Two modes of operation
  - Transport mode for direct host-to-host communication
  - Tunnel mode to connect different networks, transparent to the end-hosts
- Encapsulating security payload (ESP) offers both confidentiality and authentication
  - Replay window offers anti-replay functionality
  - Traffic flow confidentiality (TFC) padding offers flow confidentiality against passive eavesdropping attacks
- Makes use of Internet Key Exchange (IKE) for the creation and distribution of secret keys between hosts
  - Can be used with a wide range of cryptographic algorithms



Link with the book

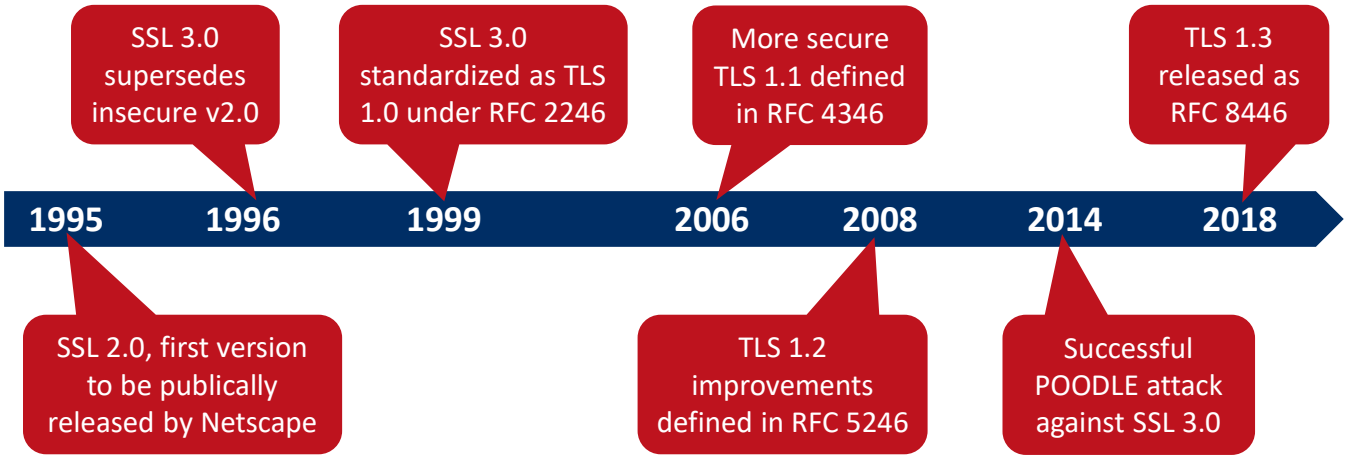
- Chapter 20

Transport Level Security

1. Transport-Layer Security (TLS)
2. Attacks against TLS
3. Secure Shell (SSH)

University of Antwerp
Faculty of Science

Secure Socket Layer (SSL) and TLS history



University of Antwerp | Faculty of Science

54

One of the most widely used security services is **Transport Layer Security (TSL)**; the current version is Version 1.2, defined in RFC 5246. TLS is an Internet standard that evolved from a commercial protocol known as **Secure Sockets Layer (SSL)**. Although SSL implementations are still around, it has been deprecated by IETF and is disabled by most corporations offering TLS software. TLS is a general- purpose service implemented as a set of protocols that rely on TCP. At this level, there are two implementation choices. For full generality, TLS could be provided as part of the underlying protocol suite and therefore be transparent to applications. Alternatively, TLS can be embedded in specific packages. For example, most browsers come equipped with TLS, and most Web servers have implemented the protocol.

TLS 1.3 (RFC 8446) is based on TLS 1.2, but includes any improvements to increase security (e.g., removing backwards compatibility for insecure cryptographic algorithms, mandating forward secrecy, improving latency with single round-trip time handshake, etc.).

TLS connections and sessions

Session

Association between client and server, defining the cryptographic security parameters, can be shared among multiple connections

Session Id	Peer certificate	Compression method	Encryption algorithm	Hash algorithm	Master secret	Resumable flag

Connection

Peer-to-peer connectivity at the transport layer, associated with a single session

random	MAC write secret	write key	Initialization vectors (IV)	Message sequence numbers

different for client and server

University of Antwerp | Faculty of Science

55

Two important TLS concepts are the TLS session and the TLS connection, which are defined in the specification as follows:

- **Connection:** A connection is a transport (in the OSI layering model definition) that provides a suitable type of service. For TLS, such connections are peer-to-peer relationships. The connections are transient. Every connection is associated with one session.
- **Session:** A TLS session is an association between a client and a server. Sessions are created by the Handshake Protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

Between any pair of parties (applications such as HTTP on client and server), there may be multiple secure connections. In theory, there may also be multiple simultaneous sessions between parties, but this feature is not used in practice. There are a number of states associated with each session. Once a session is established, there is a current operating state for both read and write (i.e., receive and send). In addition, during the Handshake Protocol, pending read and write states are created. Upon successful conclusion of the Handshake Protocol, the pending states become the current states.

A session state is defined by the following parameters:

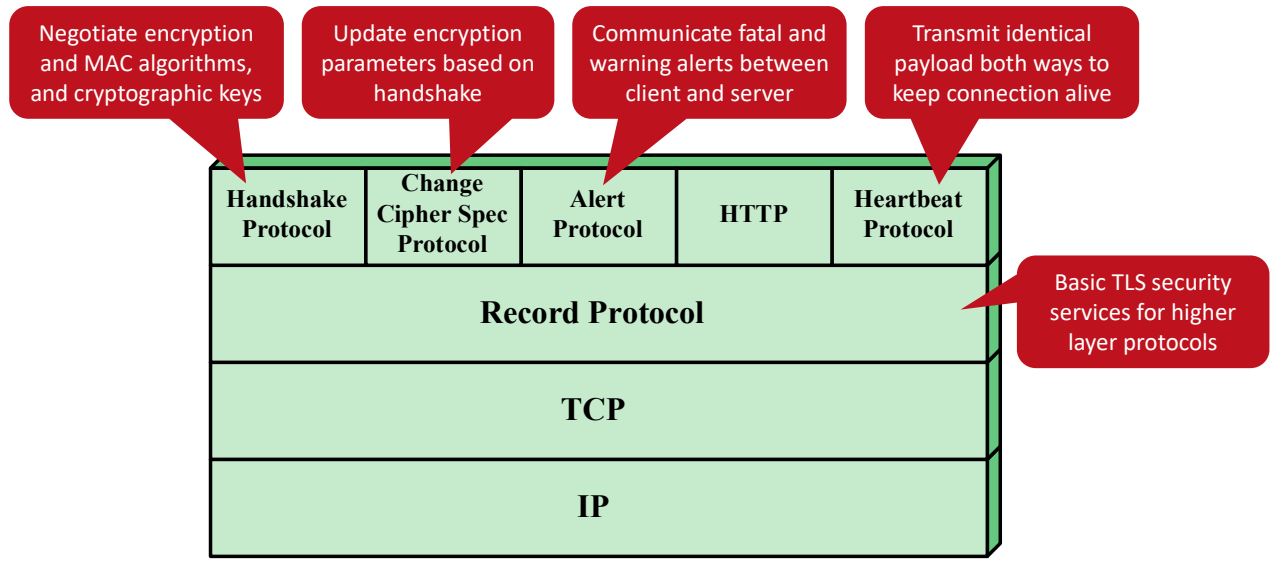
- **Session identifier:** An arbitrary byte sequence chosen by the server to identify an active or resumable session state.
- **Peer certificate:** An X509.v3 certificate of the peer. This element of the state may be null.

- **Compression method:** The algorithm used to compress data prior to encryption.
- **Cipher spec:** Specifies the bulk data encryption algorithm (such as null, AES, etc.) and a hash algorithm (such as MD5 or SHA-1) used for MAC calculation. It also defines cryptographic attributes such as the hash_size.
- **Master secret:** 48-byte secret shared between the client and server.
- **Is resumable:** A flag indicating whether the session can be used to initiate new connections.

A connection state is defined by the following parameters:

- **Server and client random:** Byte sequences that are chosen by the server and client for each connection.
- **Server write MAC secret:** The secret key used in MAC operations on data sent by the server.
- **Client write MAC secret:** The symmetric key used in MAC operations on data sent by the client.
- **Server write key:** The symmetric encryption key for data encrypted by the server and decrypted by the client.
- **Client write key:** The symmetric encryption key for data encrypted by the client and decrypted by the server.
- **Initialization vectors:** When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key. This field is first initialized by the TLS Handshake Protocol. Thereafter, the final ciphertext block from each record is preserved for use as the 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,” the appropriate sequence number is set to zero. Sequence numbers may not exceed $2^{64} - 1$.

TLS protocol stack

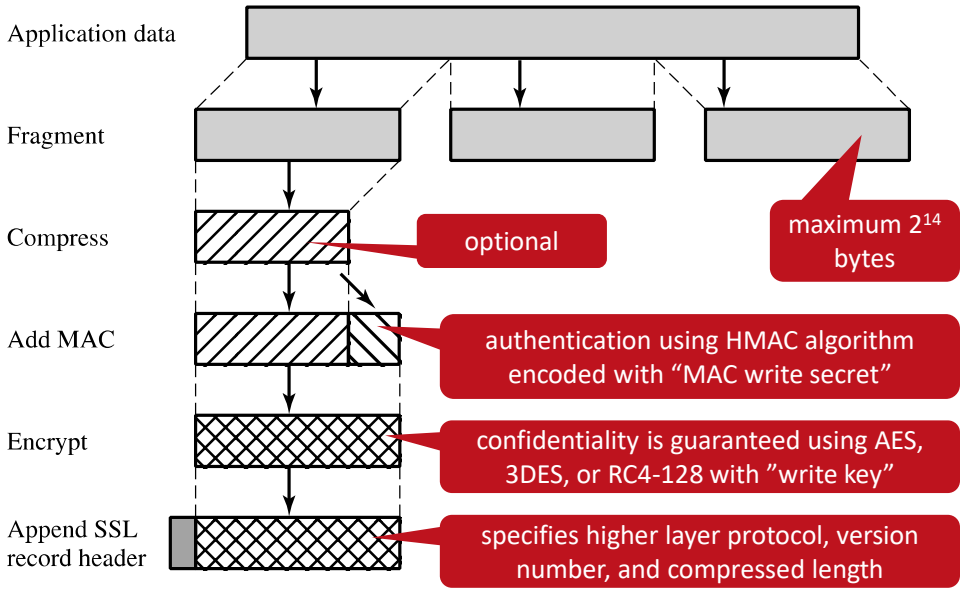


University of Antwerp | Faculty of Science

56

TLS is designed to make use of TCP to provide a reliable end-to-end secure service. TLS is not a single protocol but rather two layers of protocols. The TLS Record Protocol provides basic security services to various higher-layer protocols. In particular, the **Hypertext Transfer Protocol (HTTP)**, which provides the transfer service for Web client/server interaction, can operate on top of TLS. Three higher-layer protocols are defined as part of TLS: the Handshake Protocol; the Change Cipher Spec Protocol; and the Alert Protocol. These TLS-specific protocols are used in the management of TLS exchanges and are examined later. A fourth protocol, the Heartbeat Protocol, is defined in a separate RFC and is also discussed subsequently.

TLS record protocol



University of Antwerp | Faculty of Science

57

The TLS Record Protocol provides two services for TLS connections:

- **Confidentiality:** The Handshake Protocol defines a shared secret key that is used for conventional encryption of TLS payloads.
- **Message Integrity:** The Handshake Protocol also defines a shared secret key that is used to form a message authentication code (MAC).

The figure indicates the overall operation of the TLS Record Protocol. The Record Protocol takes an application message to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, adds a header, and transmits the resulting unit in a TCP segment. Received data are decrypted, verified, decompressed, and reassembled before being delivered to higher-level users.

The first step is **fragmentation**. Each upper-layer message is fragmented into blocks of 2^{14} bytes (16,384 bytes) or less. Next, **compression** is optionally applied. Compression must be lossless and may not increase the content length by more than 1024 bytes. In TLSv2, no compression algorithm is specified, so the default com- pression algorithm is null. The next step in processing is to compute a **message authentication code** over the compressed data. TLS makes use of the HMAC algorithm defined in RFC 2104.

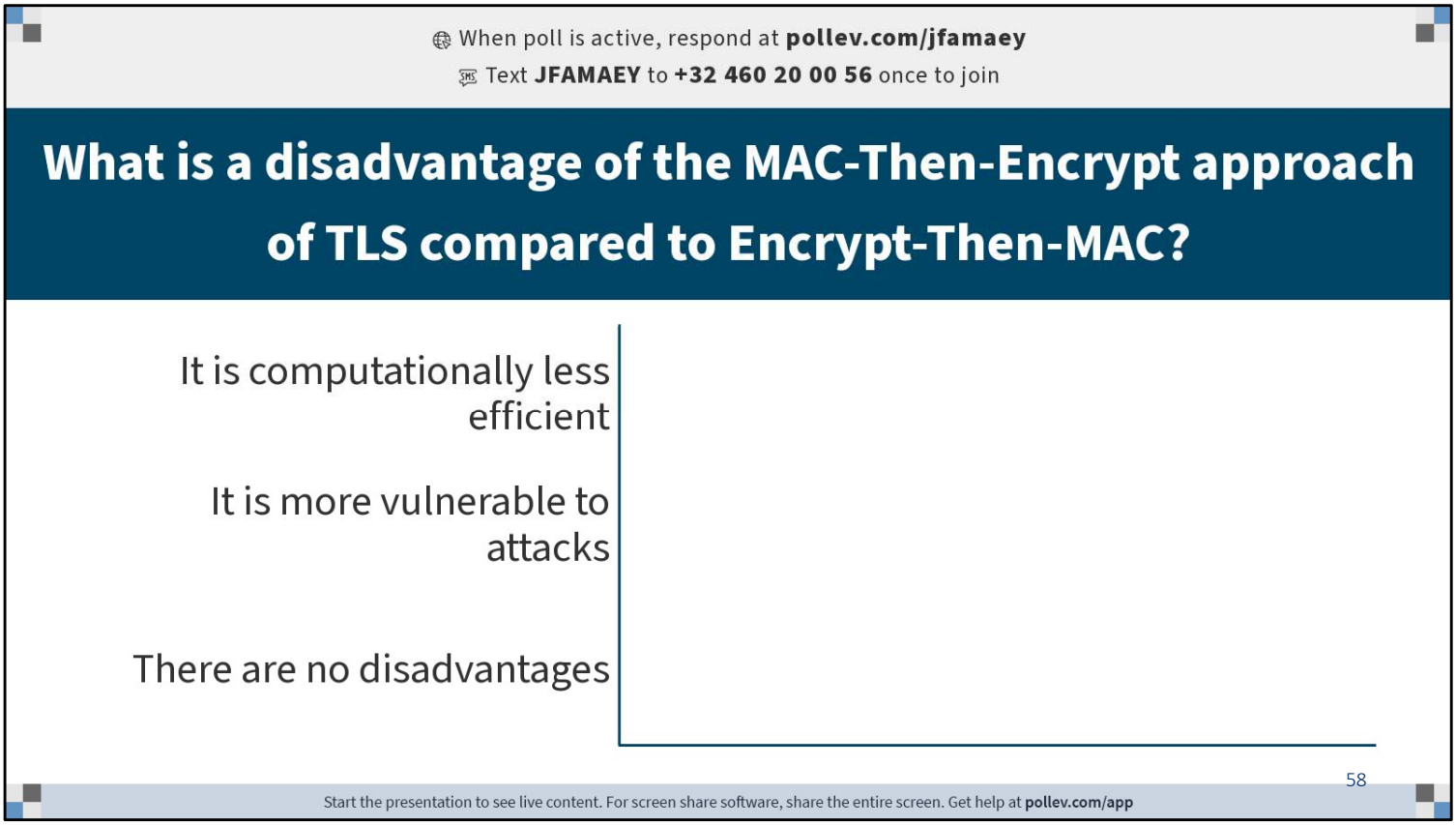
Next, the compressed message plus the MAC are **encrypted** using symmetric encryption. Encryption may not increase the content length by more than 1024 bytes, so that the total length may not exceed 214 + 2048. Several encryption algorithms are permitted: AES-128, AES-256, 3DES, and RC4-128. For stream encryption with RC4, the compressed message plus the MAC are encrypted. Note that the MAC is computed before encryption

takes place and that the MAC is then encrypted along with the plaintext or compressed plaintext.

For block encryption, padding may be added after the MAC prior to encryption. The padding is in the form of a number of padding bytes followed by a one-byte indication of the length of the padding. The padding can be any amount that results in a total that is a multiple of the cipher's block length, up to a maximum of 255 bytes. For example, if the cipher block length is 16 bytes (e.g., AES) and if the plaintext (or compressed text if compression is used) plus MAC plus padding length byte is 79 bytes long, then the padding length (in bytes) can be 1, 17, 33, and so on, up to 161. At a padding length of 161, the total length is 79 + 161 = 240. A variable padding length may be used to frustrate attacks based on an analysis of the lengths of exchanged messages.

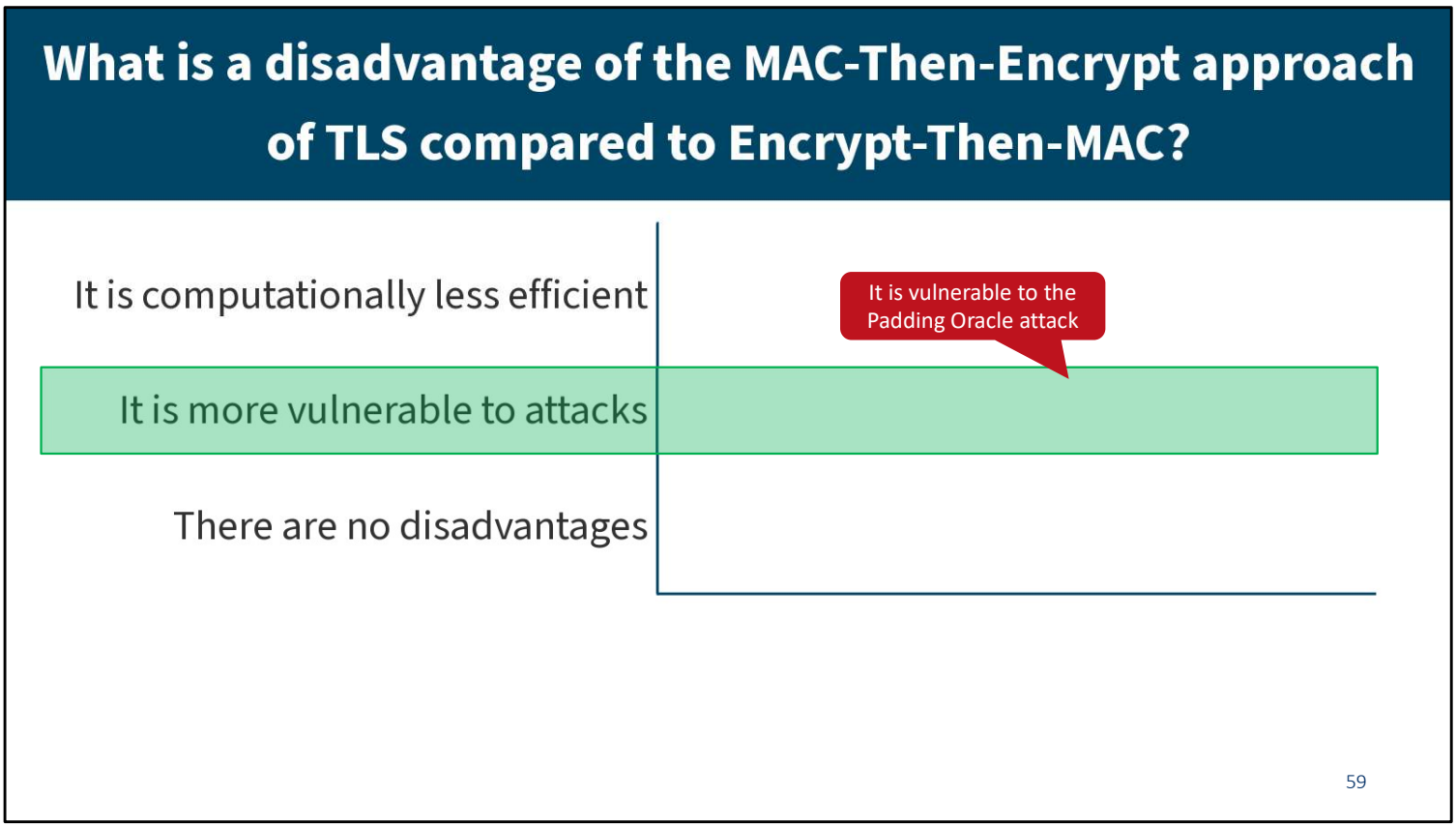
The final step of TLS Record Protocol processing is to prepend a header consisting of the following fields:

- **Content Type (8 bits)**: The higher-layer protocol used to process the enclosed fragment.
- **Major Version (8 bits):** Indicates major version of TLS in use. For TLSv2, the value is 3.
- **Minor Version (8 bits):** Indicates minor version in use. For TLSv2, the value is 1.
- **Compressed Length (16 bits):** The length in bytes of the plaintext fragment (or compressed fragment if compression is used). The maximum value is 214 + 2048.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What is a disadvantage of the MAC-Then-Encrypt approach of TLS compared to Encrypt-Then-MAC?
https://www.polleverywhere.com/multiple_choice_polls/0DkF1QnrivWKJVY?state=opened&flow=Default&onscreen=persist

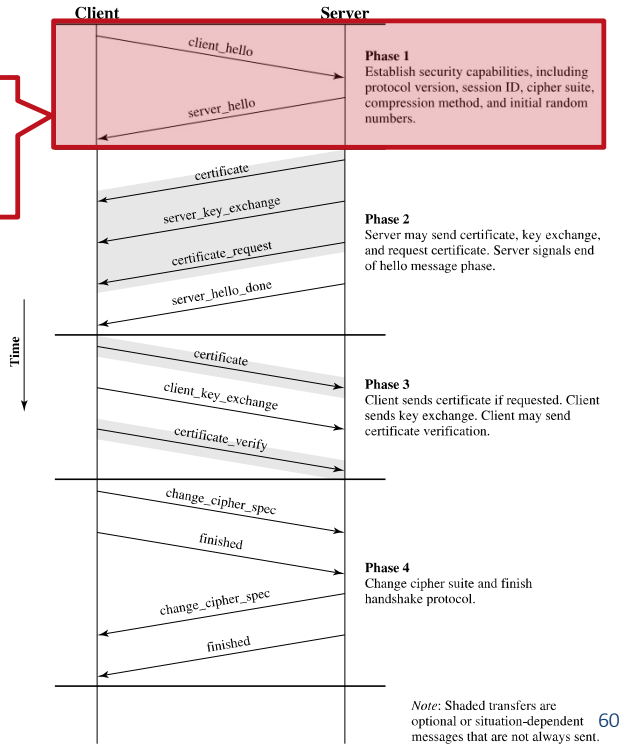


Poll Title: What is a disadvantage of the MAC-Then-Encrypt approach of TLS compared to Encrypt-Then-MAC?

Handshake protocol phases

Phase 1: Establish capabilities

Client and server exchange nonce, key material, supported TLS/SSL versions, and supported cryptographic algorithms



University of Antwerp
Faculty of Science

60

Phase 1 initiates a logical connection and establishes the security capabilities that will be associated with it. The exchange is initiated by the client, which sends a **client_hello message** with the following parameters:

- **Version:** The highest TLS version understood by the client.
- **Random:** A client-generated random structure consisting of a 32-bit timestamp and 28 bytes generated by a secure random number generator. These values serve as nonces and are used during key exchange to prevent replay attacks.
- **Session ID:** A variable-length session identifier. A nonzero 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:** This is a list that contains the combinations of cryptographic algorithms supported by the client, in decreasing order of preference. Each element of the list (each cipher suite) defines both a key exchange algorithm and a CipherSpec; these are discussed subsequently.
- **Compression Method:** This is a list of the compression methods the client supports.

After sending the client_hello message, the client waits for the **server_ hello message**, which contains the same parameters as the client_hello message. For the server_hello message, the following conventions apply. The Version field contains the lowest of the version suggested by the client and the highest supported by the server. The Random field is generated by the server and is independent of the client's Random field. If the SessionID field of the client was nonzero, the same value is used by the server; otherwise the server's SessionID field contains the value for a new session. The

CipherSuite field contains the single cipher suite selected by the server from those proposed by the client. The Compression field contains the compression method selected by the server from those proposed by the client.

The first element of the Ciphersuite parameter is the key exchange method (i.e., the means by which the cryptographic keys for conventional encryption and MAC are exchanged). The following key exchange methods are supported:

- **RSA:** The secret key is encrypted with the receiver's RSA public key. A public-key certificate for the receiver's key must be made available.
- **Fixed Diffie–Hellman:** This is a Diffie–Hellman key exchange in which the server's certificate contains the Diffie–Hellman public parameters signed by the certificate authority (CA). That is, the public-key certificate contains the Diffie–Hellman public-key parameters. The client provides its Diffie–Hellman public-key parameters either in a certificate, if client authentication is required, or in a key exchange message. This method results in a fixed secret key between two peers based on the Diffie–Hellman calculation using the fixed public keys.
- **Ephemeral Diffie-Hellman:** This technique is used to create ephemeral (temporary, one-time) secret keys. In this case, the Diffie–Hellman public keys are exchanged and signed using the sender's private RSA or DSS key. The receiver can use the corresponding public key to verify the signature. Certificates are used to authenticate the public keys. This would appear to be the most secure of the three Diffie–Hellman options because it results in a temporary, authenticated key.
- **Anonymous Diffie–Hellman:** The base Diffie–Hellman algorithm is used with no authentication. That is, each side sends its public Diffie–Hellman parameters to the other with no authentication. This approach is vulnerable to man-in-the-middle attacks, in which the attacker conducts anonymous Diffie– Hellman with both parties.

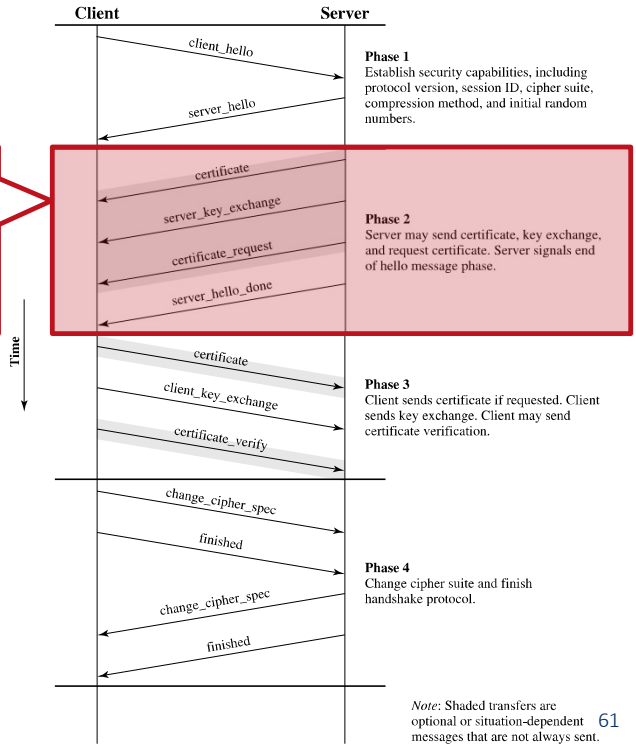
Following the definition of a key exchange method is the CipherSpec, which includes the following fields:

- **CipherAlgorithm:** Any of the algorithms mentioned earlier: RC4, RC2, DES, 3DES, DES40, or IDEA
- **MACAlgorithm:** MD5 or SHA-1
- **CipherType:** Stream or Block
- **IsExportable:** True or False
- **HashSize:** 0, 16 (for MD5), or 20 (for SHA-1) bytes
- **Key Material:** A sequence of bytes that contain data used in generating the write keys
- **IV Size:** The size of the Initialization Value for Cipher Block Chaining (CBC) encryption ß

Handshake protocol phases

Phase 2: Server authentication

Depending on selected public key exchange method (RSA, Diffie-Hellman), server sends certificate, public key, and/or client certificate request



The server begins this phase by sending its certificate if it needs to be authenticated; the message contains one or a chain of X.509 certificates. The **certificate message** is required for any agreed-on key exchange method except anonymous Diffie–Hellman. Note that if fixed Diffie–Hellman is used, this certificate message functions as the server's key exchange message because it contains the server's public Diffie–Hellman parameters.

Next, a **server_key_exchange message** may be sent if it is required. It is not required in two instances: (1) The server has sent a certificate with fixed Diffie–Hellman parameters; or (2) RSA key exchange is to be used. The server_key_exchange message is needed for the following:

- **Anonymous Diffie–Hellman:** The message content consists of the two global Diffie–Hellman values (a prime number and a primitive root of that number) plus the server's public Diffie–Hellman key.
- **Ephemeral Diffie–Hellman:** The message content includes the three Diffie– Hellman parameters provided for anonymous Diffie–Hellman plus a signature of those parameters.
- **RSA key exchange (in which the server is using RSA but has a signature-only RSA key):** Accordingly, the client cannot simply send a secret key encrypted with the server's public key. Instead, the server must create a temporary RSA public/private key pair and use the server_key_exchange message to send the public key. The message content includes the two parameters of the temporary RSA public key (exponent and modulus) plus a signature of those parameters.

Some further details about the signatures are warranted. As usual, a signature is created by taking the hash of a message and encrypting it with the sender's private key. In this case, the hash is defined as:

hash(ClientHello.random || ServerHello.random || ServerParams)

So the hash covers not only the Diffie–Hellman or RSA parameters but also the two nonces from the initial hello messages. This ensures against replay attacks and misrepresentation. In the case of a DSS signature, the hash is performed using the SHA-1 algorithm. In the case of an RSA signature, both an MD5 and an SHA-1 hash are calculated, and the concatenation of the two hashes (36 bytes) is encrypted with the server's private key.

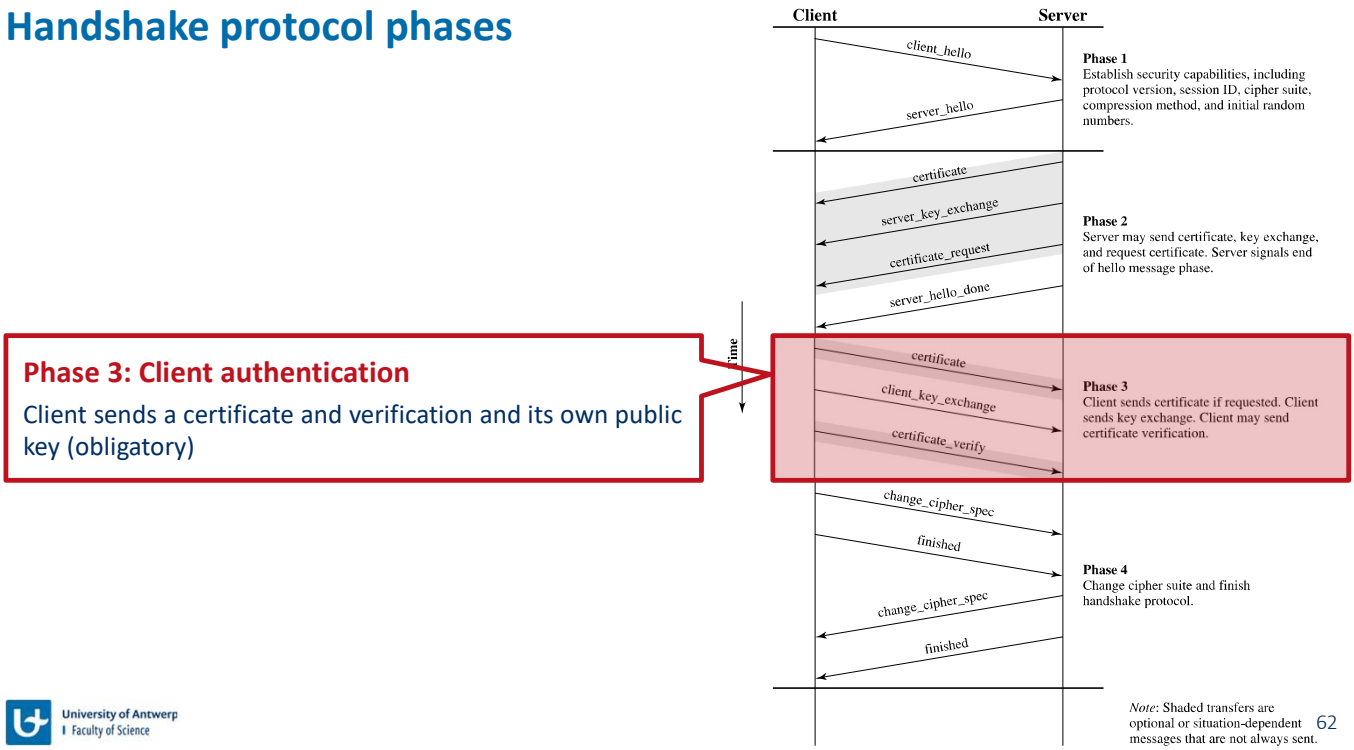
Next, a nonanonymous server (server not using anonymous Diffie–Hellman) can request a certificate from the client. The **certificate_request message** includes two parameters: certificate_type and certificate_authorities. The certificate type in- dicates the public-key algorithm and its use:

- RSA, signature only
- DSS, signature only
- RSA for fixed Diffie–Hellman; in this case the signature is used only for authentication, by sending a certificate signed with RSA
- DSS for fixed Diffie–Hellman; again, used only for authentication

The second parameter in the certificate_request message is a list of the distinguished names of acceptable certificate authorities.

The final message in phase 2, and one that is always required, is the **server_done message**, which is sent by the server to indicate the end of the server hello and associated messages. After sending this message, the server will wait for a client response. This message has no parameters.

Handshake protocol phases



Phase 3: Client authentication

Client sends a certificate and verification and its own public key (obligatory)

Upon receipt of the server_done message, the client should verify that the server provided a valid certificate (if required) and check that the server_hello parameters are acceptable. If all is satisfactory, the client sends one or more messages back to the server.

If the server has requested a certificate, the client begins this phase by sending a **certificate message**. If no suitable certificate is available, the client sends a no_certificate alert instead.

Next is the **client_key_exchange message**, which must be sent in this phase. The content of the message depends on the type of key exchange, as follows:

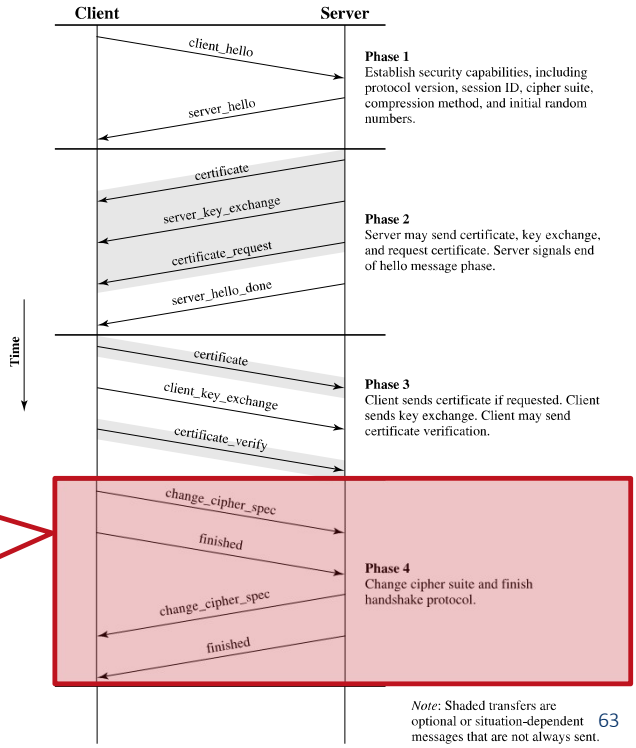
- **RSA:** The client generates a 48-byte *pre-master secret* and encrypts with the public key from the server's certificate or temporary RSA key from a server_key_exchange message. Its use to compute a *master secret* is explained later.
- **Ephemeral or Anonymous Diffie–Hellman:** The client's public Diffie–Hellman parameters are sent.
- **Fixed Diffie–Hellman:** The client's public Diffie–Hellman parameters were sent in a certificate message, so the content of this message is null.

Finally, in this phase, the client may send a **certificate_verify message** to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability (i.e., all certificates except those containing fixed Diffie–Hellman parameters). This message signs a hash code based on the preceding messages, defined as

```
CertificateVerify.signature.md5_hash
        MD5(handshake_messages);
Certificate.signature.sha_hash
        SHA(handshake_messages);
```

where handshake_messages refers to all Handshake Protocol messages sent or received starting at client_hello but not including this message. If the user's private key is DSS, then it is used to encrypt the SHA-1 hash. If the user's private key is RSA, it is used to encrypt the concatenation of the MD5 and SHA-1 hashes. In either case, the purpose is to verify the client's ownership of the private key for the client certificate. Even if someone is misusing the client's certificate, he or she would be unable to send this message.

Handshake protocol phases



Phase 4: Finish

- Part of change cipher spec protocol
- Client and server send a message encrypted with the new algorithms and keys for verification

University of Antwerp
Faculty of Science

63

Phase 4 completes the setting up of a secure connection. The client sends a **change_cipher_spec message** and copies the pending CipherSpec into the current CipherSpec. Note that this message is not considered part of the Handshake Protocol but is sent using the Change Cipher Spec Protocol. The client then immediately sends the **finished message** under the new algorithms, keys, and secrets. The finished message verifies that the key exchange and authentication processes
were successful. The content of the finished message is:

PRF(master_secret, finished_label, MD5(handshake_messages) || SHA-1 (handshake_messages))

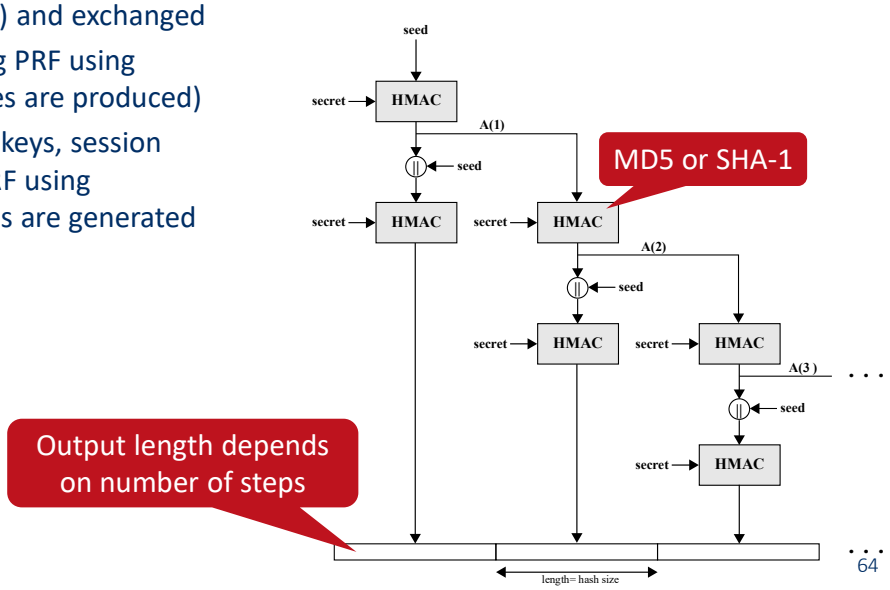
where finished_label is the string “client finished” for the client and “server finished” for the server.

In response to these two messages, the server sends its own change_cipher_spec message, transfers the pending to the current CipherSpec, and sends its finished message. At this point, the handshake is complete and the client and server may begin to exchange application-layer data.

Creation of symmetric keys

- pre_master_secret is generated by client (RSA) or client and server (Diffie-Hellman) and exchanged
- master_secret is calculated using PRF using pre_master_secret (until 48 bytes are produced)
- cryptographic parameters (MAC keys, session keys, IVs) are generated using PRF using master_secret until enough bytes are generated

TLS Pseudorandom function (PRF)



The shared master secret is a one-time 48-byte value (384 bits) generated for this session by means of secure key exchange. The creation is in two stages. First, a pre_master_secret is exchanged. Second, the master_ secret is calculated by both parties. For pre_master_secret exchange, there are two possibilities.

- **RSA:** A 48-byte pre_master_secret is generated by the client, encrypted with the server's public RSA key, and sent to the server. The server decrypts the ciphertext using its private key to recover the pre_master_secret.
- **Diffie–Hellman:** Both client and server generate a Diffie–Hellman public key. After these are exchanged, each side performs the Diffie–Hellman calculation to create the shared pre_master_secret.

Both sides now compute the master_secret as

master_secret = PRF(pre_master_secret, "master secret", ClientHello.random || ServerHello.random)

where ClientHello.random and ServerHello.random are the two nonce values exchanged in the initial hello messages.

The algorithm is performed until 48 bytes of pseudorandom output are produced. The calculation of the key block material (MAC secret keys, session encryption keys, and IVs) is defined as:

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

until enough output has been generated.

CipherSpecs require a client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV, which are generated from the master secret in that order. These parameters are generated from the master secret by hashing the master secret into a sequence of secure bytes of sufficient length for all needed parameters.

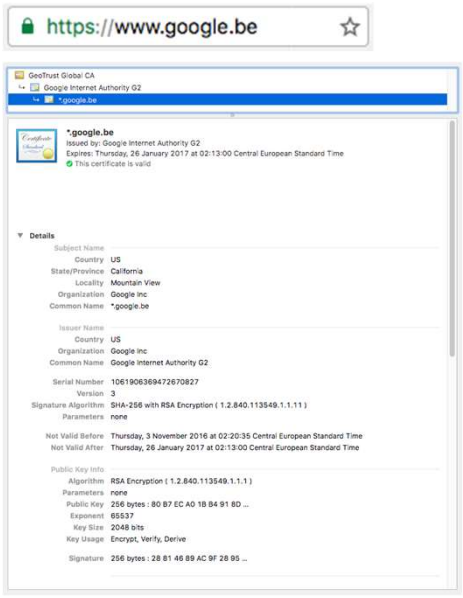
The generation of the key material from the master secret uses the same format for generation of the master secret from the pre-master secret as

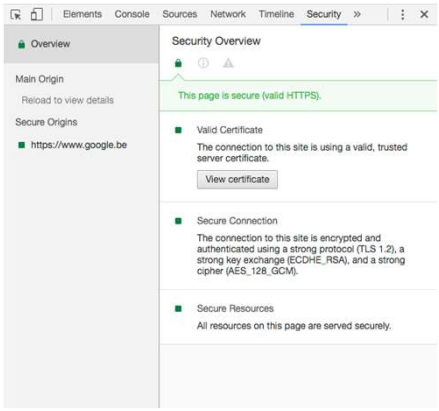
key_block = MD5(master_secret || SHA('A' || master_secret || ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('BB' || master_secret || ServerHello.random || ClientHello.random)) ||
MD5(master_secret || SHA('CCC' || master_secret || ServerHello.random || ClientHello.random)) || …

until enough output has been generated. The result of this algorithmic structure is a pseudorandom function. We can view the master_secret as the pseudorandom seed value to the function. The client and server random numbers can be viewed as salt values to complicate cryptanalysis.

TLS makes use of a pseudorandom function referred to as PRF to expand secrets into blocks of data for purposes of key generation or validation. The objective is to make use of a relatively small, shared secret value but to generate longer blocks of data in a way that is secure from the kinds of attacks made on hash functions and MACs. The data expansion function makes use of the HMAC algorithm with either MD5 or SHA-1 as the underlying hash function.

HTTPS: HTTP over SSL/TLS





HTTPS encrypts the following:

- URL of requested document
- Contents of the document
- Contents of browser forms
- Cookies
- Contents of the HTTP header

University of Antwerp
Faculty of Science

65

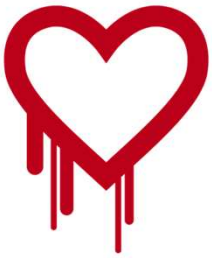
- Built into modern browsers
- Needs to be supported by the web server
- Standardized as RFC 2018 as HTTP over TLS
- Encrypts the following information
    - URL of the requested document
    - Contents of the document
    - Contents of browser forms
    - Cookies
    - Contents of the HTTP header

1. Transport-Layer Security (TLS)
2. Attacks against TLS
3. Secure Shell (SSH)

University of Antwerp
Faculty of Science

Heartbleed: The Internet's worst vulnerability to date

- Caused by a minor programming error of the heartbeat implementation in OpenSSL 1.0.1
- Introduced March 2012, made public in April 2014
- At the time of discovery 17% (500.000) of the Internet's secure web servers were vulnerable to attacks



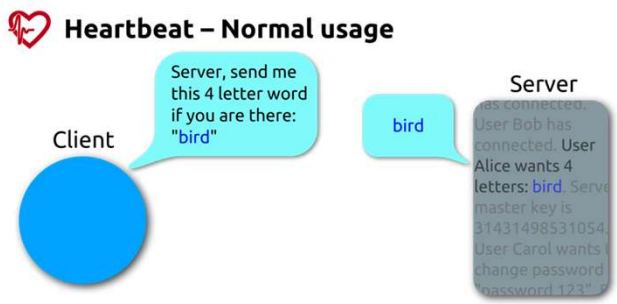
Faulty line of code in OpenSSL 1.0.1:

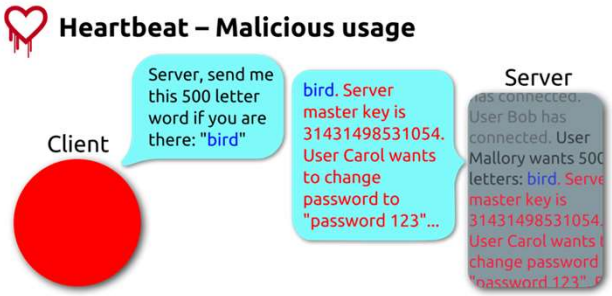
```
memcpy(bp, pl, payloadsize);
```

General attack principle

- If pl array is smaller than what payloadsize makes believe, old data in memory allotted to bp array is not overwritten
- heartbeat protocol sends back contents of bp array
- Attacker receives whatever was previously stored in memory allotted to bp

Heartbleed: The Internet's worst vulnerability to date





The heartbleed fix

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
       return 0; // silently discard

hbtype = *p++;
n2s(p, payload);

if (1 + 2 + payload + 16 > s->s3->rrec.length)
       return 0; //silently discard per RFC 6520 sec. 4

pl = p;
```

protect against zero-length heartbeats

make sure heartbeat length is correct

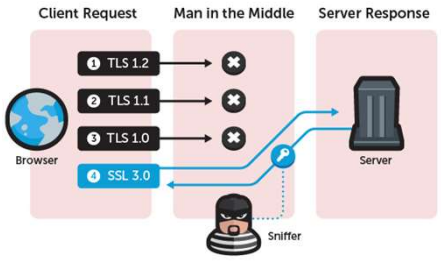
POODLE: Padding Oracle On Downgraded Legacy Encryption

Two stage attack

- Trick the server into reverting to SSL 3.0 compatibility mode
- Perform a Padding Oracle man-in-the-middle attack

Step 1: trick the server into reverting to SSL 3.0

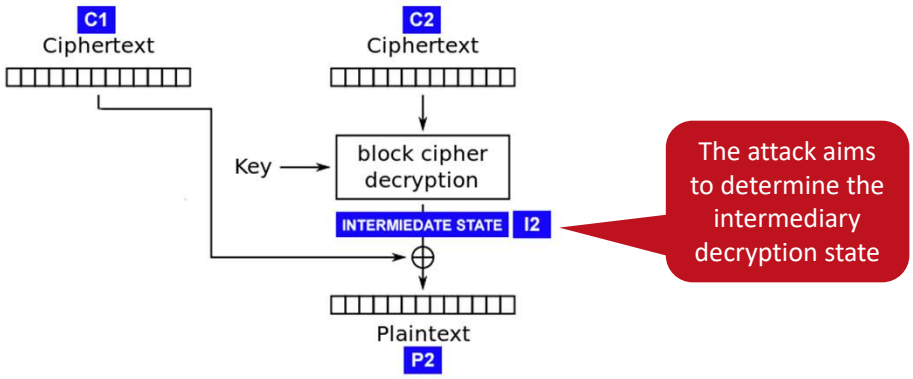
- Assume an attacker controls the network between the client and server
- Attacker can interfere during TLS handshake when client offers TLS 1.0 or later
- Attacker only allows successful handshake when client offers SSL 3.0 downgrade



More information: https://www.openssl.org/~bodo/ssl-poodle.pdf

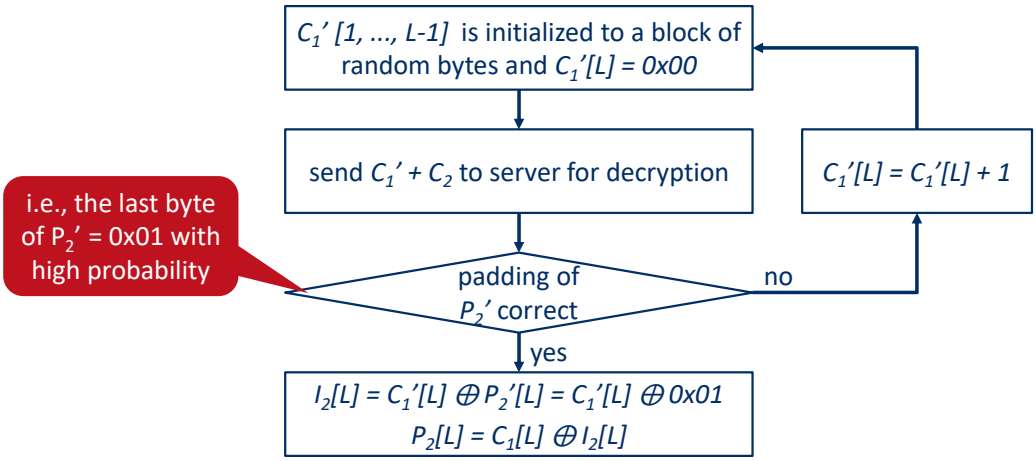
Padding oracle attacks

- In Cipher Block Chaining (CBC) mode, each block of plaintext is XORed with the previous ciphertext block before encryption
- Knowing whether a given ciphertext produces plaintext with valid padding is enough to break CBC encryption
- In PKCS7 padding, each padded byte equals the length of the padding (e.g., a 13-byte block is padded with 3 0x03 bytes to form a valid full 16-byte block)



Padding oracle attacks: Finding one byte

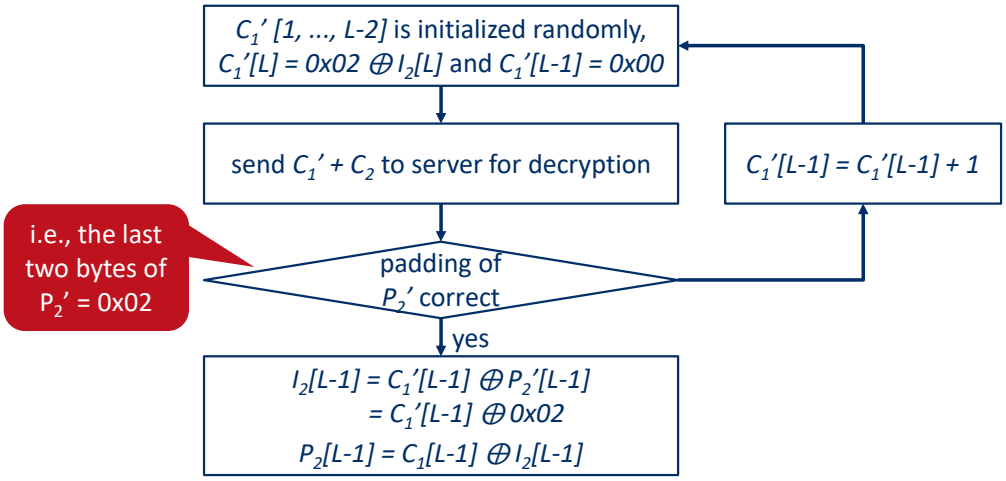
Given a block length L, the last byte of the intermediary state I_2 can be determined as follows:



Remember that we can pass in any ciphertext, and the server will tell us whether it decrypts to plaintext with valid padding or not. That's it. We exploit this by passing in C1' + C2, where C1' is a sneakily chosen ciphertext block, C2 is the ciphertext block we are trying to decrypt, and C1' + C2 is the concatenation of the two. We call the decrypted plaintext block produced P'2.

To begin with, we choose C1'[1..15] to be random bytes, and C1'[16] to be 00. We pass C1' + C2 to the server. If the server says we have produced a plaintext with valid padding, then we can be pretty sure that P2'[16] must be 01 (as this would give us valid padding). Of course, if the server comes back and tells us that our padding is invalid, then we just set C1'[16] to 01, then 02, and so on, until we hit the jackpot.

Padding oracle attacks: finding the previous byte

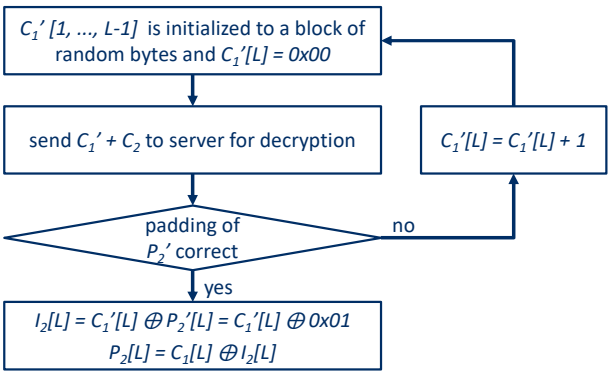


We found the last byte by fiddling with C1' until we produced something with valid padding, and in doing so were able to infer that the final byte of P'2 was 0x01. We then used the fact that we knew P2'[16] and C1'[16] to find I2[16]. We continue on this theme to find the rest of the bytes of I2, and therefore decrypt the ciphertext block. We now choose C1'[1..14] to be random bytes, C1'[15] to be the byte 00, and C1'[16] to be a byte chosen so as to make P2'[16] == 02.

So we can be sure that P2' will end in a 0x02, and therefore the only way for P2' to have valid padding is if P2[15] is also 0x02! We fiddle with C1'[15] until the server does its things and tells us we have passed it a ciphertext that decrypts to a plaintext with valid padding.

Exercise: Padding Oracle Attack

Exercise: Given an L-byte cipher block C_1, with byte $C_1[L] = 0x24$. Assume a POODLE attack gives valid padding for C_2 with $C'_1[L] = 0x94$, determine $P_2[L]$.



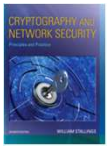
Solution: Padding Oracle Attack

Solution

$I_2[L] = C_1'[L] \oplus P_2'[L] = 0 \times 94 \oplus 0 \times 01 = 0 \times 95$
$P_2[L] = I_2[L] \oplus C_1[L] = 0 \times 95 \oplus 0 \times 24 = 0 \times B1$

Summary on Transport Layer Security

- Originally developed as SSL (1995), renamed to TLS in 1999
  - Provides confidentiality, integrity and authentication over TCP
  - Offers different methods for compression, encryption, and authentication
- TLS defines different protocols
  - Handshake: Negotiate cryptographic algorithms and keys
  - Change cipher spec: Update cryptographic parameters
  - Alert: Communicate warnings and errors between client and server
  - Heartbeat: Keep connection alive by reflecting payload
  - Record: Basic TLS security services offered to higher level protocols
- Widely used for WWW security through HTTP over SSH (**HTTPS**)
- Several high-impact **exploits** have been found (e.g., Heartbleed, POODLE)



Link with the book

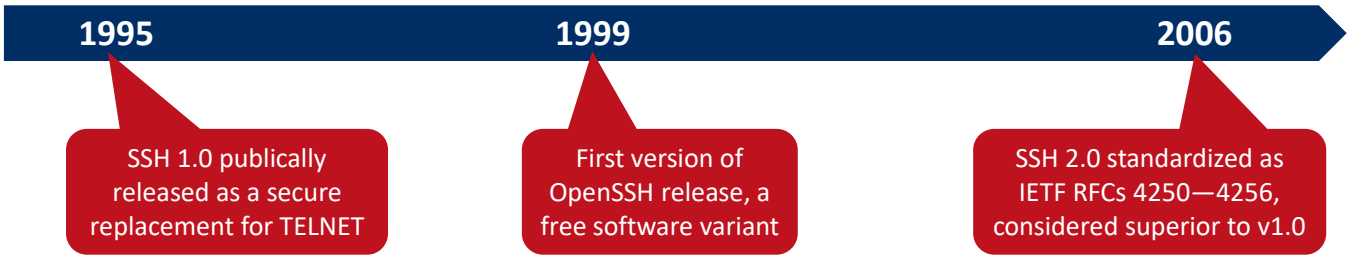
- Chapter 17 (17.2 – 17.3)

1. Transport-Layer Security (TLS)
2. Attacks against TLS
3. Secure Shell (SSH)

University of Antwerp
Faculty of Science

Secure Shell (SSH) in a nutshell

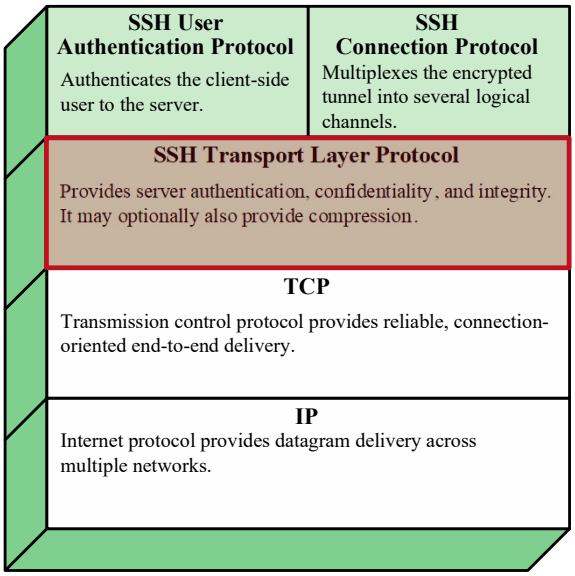
- Replacement of non-secure remote logon schemes, such as TELNET
- Provides client-server confidentiality, integrity and authentication
- Capabilities include
  - Remote logon
  - Remote command execution
  - File transfer
  - Email
  - …



Secure Shell (SSH) is a protocol for secure network communications designed to be relatively simple and inexpensive to implement. The initial version, SSH1 was focused on providing a secure remote logon facility to replace TELNET and other remote logon schemes that provided no security. SSH also provides a more general client/server capability and can be used for such network functions as file transfer and email. A new version, SSH2, fixes a number of security flaws in the original scheme. SSH2 is documented as a proposed standard in IETF RFCs 4250 through 4256.

SSH client and server applications are widely available for most operating systems. It has become the method of choice for remote login and X tunneling and is rapidly becoming one of the most pervasive applications for encryption technology outside of embedded systems.

SSH protocol stack



SSH is organized as three protocols that typically run on top of TCP:

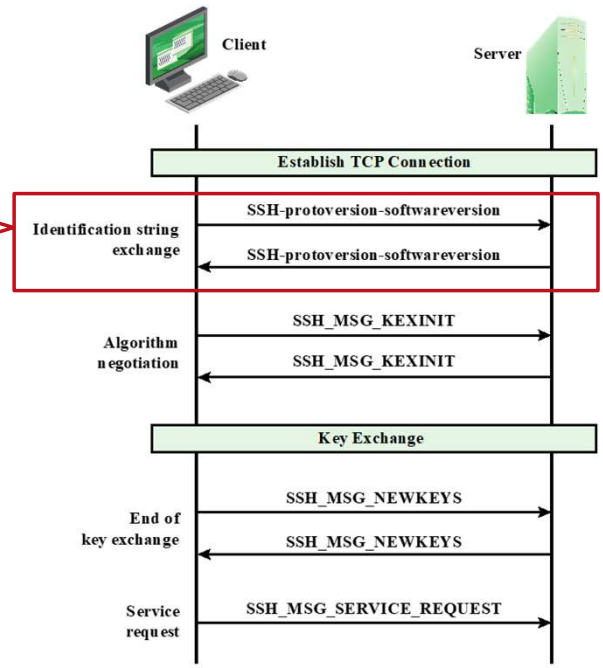
- **Transport Layer Protocol:** Provides server authentication, data confidentiality, and data integrity with forward secrecy (i.e., if a key is compromised during one session, the knowledge does not affect the security of earlier sessions). The transport layer may optionally provide compression.
- **User Authentication Protocol:** Authenticates the user to the server.
- **Connection Protocol:** Multiplexes multiple logical communications channels over a single, underlying SSH connection.

Transport layer protocol: handshake procedure

Identification string exchange

SSH-protoversion.softwareversion
SP comments CR LF

Example:
SSH-2.0 billsSSH_3.6.3q3<CR><LF>



University of Antwerp
Faculty of Science

80

The first step, the **identification string exchange**, begins with the client sending a packet with an identification string of the form:

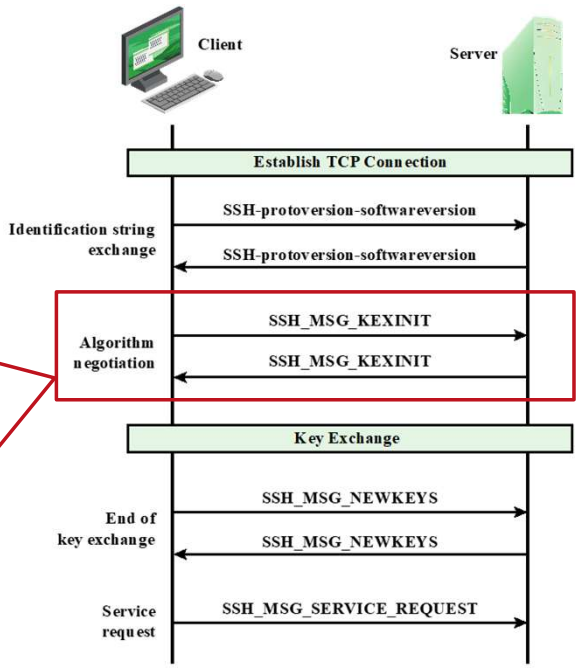
SSH-protoversion-softwareversion SP comments CR LF

where SP,CR, and LF are space character, carriage return, and line feed, respectively. An example of a valid string is SSH-2.0-billsSSH_3.6.3q3<CR><LF>. The server responds with its own identification string. These strings are used in the Diffie–Hellman key exchange.

Transport layer protocol: handshake procedure

Algorithm negotiation

- Each side sends list of supported algorithms sorted by preference
- First algorithm on client list also supported by server is chosen
- Cipher algorithms
  - 3DES-CBC (required)
  - AES128-CBC (recommended)
  - Blowfish, Twofish, AES, Serpent, …
- MAC algorithms
  - HMAC-SHA1 (required)
  - HMAC-SHA1-96 (recommended)
  - HMAC-MD5, HMAC-MD5-96
- Compression algorithms
  - None (required)
  - ZLIB



University of Antwerp
Faculty of Science

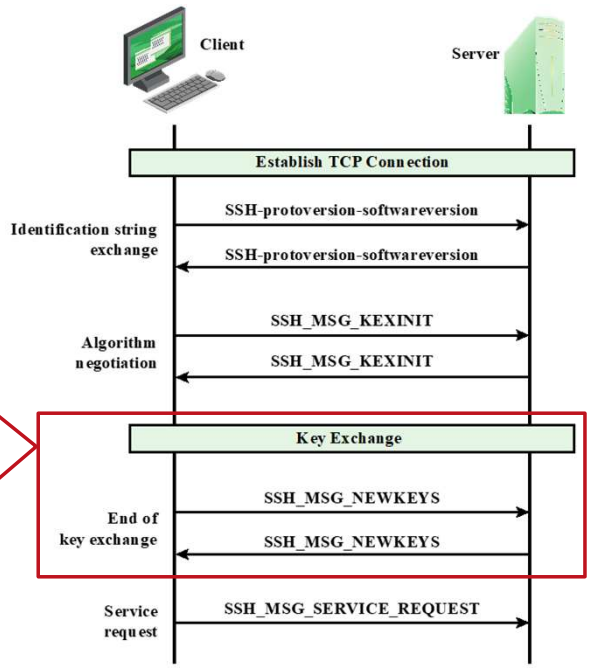
81

Next comes **algorithm negotiation**. Each side sends an SSH_MSG_KEXINIT containing lists of supported algorithms in the order of preference to the sender. There is one list for each type of cryptographic algorithm. The algorithms include key exchange, encryption, MAC algorithm, and compression algorithm. For each category, the algorithm chosen is the first algorithm on the client's list that is also supported by the server.

Transport layer protocol: handshake procedure

Key exchange

- Authenticate the server and securely agree on master key K
- Based on Diffie-Hellman
- K is generated based on random numbers x and y, generated by client and server respectively
- Server signs K with its private key, providing authentication
- K is subsequently used to generate symmetric encryption and MAC keys



University of Antwerp | Faculty of Science

82

The next step is **key exchange**. The specification allows for alternative methods of key exchange, but at present, only two versions of Diffie–Hellman key exchange are specified. Both versions are defined in RFC 2409 and require only one packet in each direction. The following steps are involved in the exchange. In this, C is the client; S is the server; p is a large safe prime; g is a generator for a subgroup of GF(p); q is the order of the subgroup; V_S is S's identification string; V_C is C's identification string; K_S is S's public host key; I_C is C's SSH_MSG_KEXINIT message and I_S is S's SSH_MSG_KEXINIT message that have been exchanged before this part begins. The values of p, g, and q are known to both client and server as a result of the algorithm selection negotiation. The hash function hash() is also decided during algorithm negotiation:

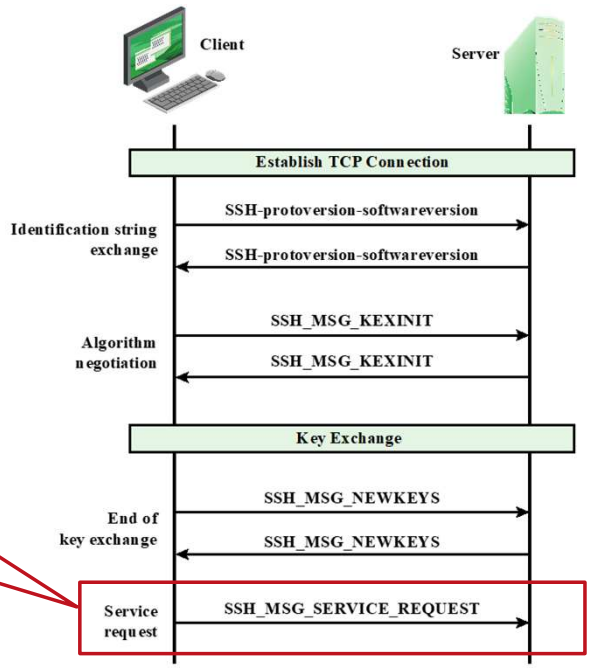
1. C generates a random number $x (1 < x < q)$ and computes $e = g^x \bmod p$. C sends e to S.
2. S generates a random number $y (0 < y < q)$ and computes $f = g^y \bmod p$. S receives e. It computes $K = e^y \bmod p$, H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K), and signature s on H with its private host key. S sends (K_S || f || s) to C. The signing operation may involve a second hashing operation.
3. C verifies that K_S really is the host key for S (e.g., using certificates or a local database). C is also allowed to accept the key without verification; however, doing so will render the protocol insecure against active attacks (but may be desirable for practical reasons in the short term in many environments). C then computes $K = f^x \bmod p$, H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K),and verifies the signature s on H.

As a result of these steps, the two sides now share a master key K. In addition, the

server has been authenticated to the client, because the server has used its private key to sign its half of the Diffie-Hellman exchange. Finally, the hash value *H* serves as a session identifier for this connection. Once computed, the session identifier is not changed, even if the key exchange is performed again for this connection to obtain fresh keys.

The **end of key exchange** is signaled by the exchange of SSH_MSG_NEWKEYS packets. At this point, both sides may start using the keys generated from *K*.

Transport layer protocol: handshake procedure



Service request

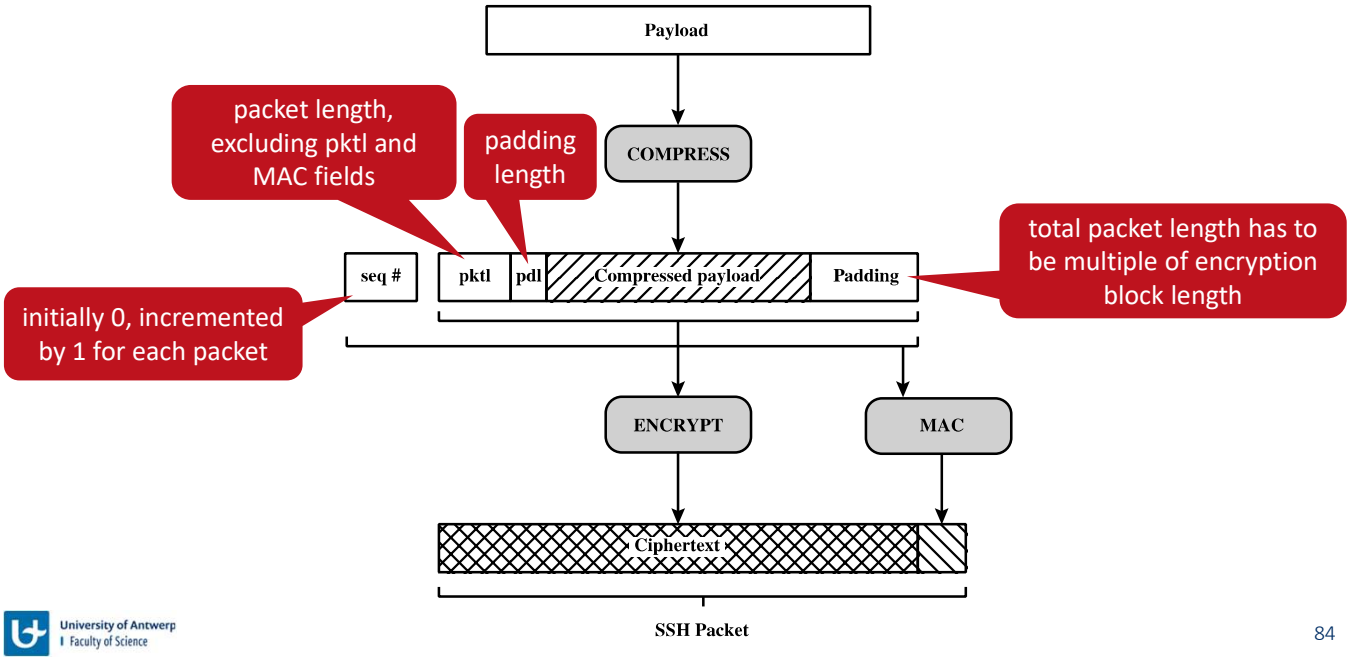
Client requests either User Authentication or Connection Protocol

University of Antwerp
Faculty of Science

83

The final step is **service request**. The client sends an SSH_MSG_SERVICE_ REQUEST packet to request either the User Authentication or the Connection Protocol. Subsequent to this, all data is exchanged as the payload of an SSH Transport Layer packet, protected by encryption and MAC.

Transport layer protocol: packet format



University of Antwerp | Faculty of Science

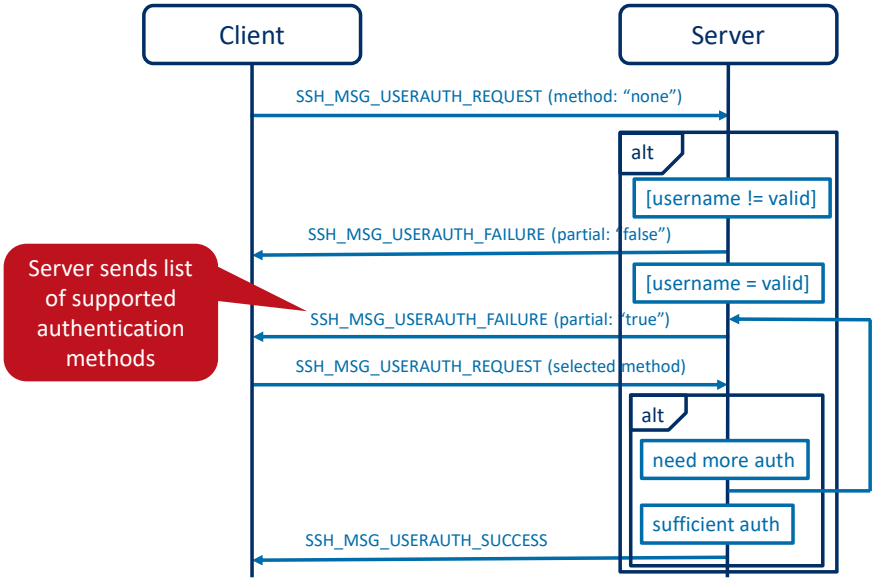
84

Once the connection is established, the client and server exchange data, referred to as packets, in the data field of a TCP segment. Each packet is in the following format:

- **Packet length:** Length of the packet in bytes, not including the packet length and MAC fields.
- **Padding length:** Length of the random padding field.
- **Payload:** Useful contents of the packet. Prior to algorithm negotiation, this field is uncompressed. If compression is negotiated, then in subsequent packets, this field is compressed.
- **Random padding:** Once an encryption algorithm has been negotiated, this field is added. It contains random bytes of padding so that the total length of the packet (excluding the MAC field) is a multiple of the cipher block size, or 8 bytes for a stream cipher.
- **Message authentication code (MAC):** If message authentication has been negotiated, this field contains the MAC value. The MAC value is computed over the entire packet plus a sequence number, excluding the MAC field. The sequence number is an implicit 32-bit packet sequence that is initialized to zero for the first packet and incremented for every packet. The sequence number is not included in the packet sent over the TCP connection.

Once an encryption algorithm has been negotiated, the entire packet (excluding the MAC field) is encrypted after the MAC value is calculated.

User authentication protocol: message exchange



The User Authentication Protocol provides the means by which the client is authenticated to the server. The message exchange involves the following steps.

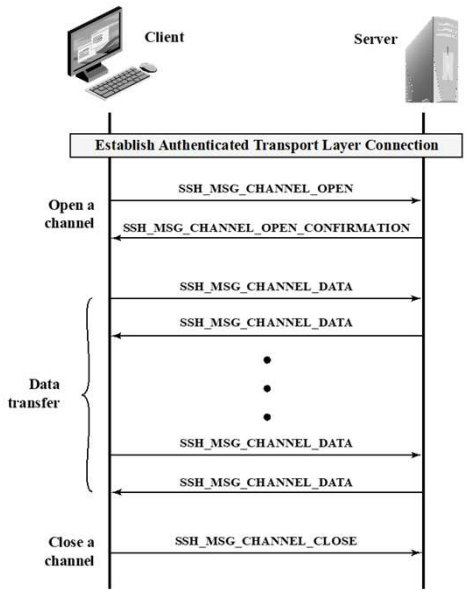
1. The client sends an SSH_MSG_USERAUTH_REQUEST with a requested method of none.
2. The server checks to determine if the username is valid. If not, the server returns SSH_MSG_USERAUTH_FAILURE with the partial success value of false. If the username is valid, the server proceeds to step 3.
3. The server returns SSH_MSG_USERAUTH_FAILURE with a list of one or more authentication methods to be used.
4. The client selects one of the acceptable authentication methods and sends an SSH_MSG_USERAUTH_REQUEST with that method name and the required method-specific fields. At this point, there may be a sequence of exchanges to perform the method.
5. If the authentication succeeds and more authentication methods are required, the server proceeds to step 3, using a partial success value of true. If the 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 an SSH_MSG_USERAUTH_SUCCESS message, and the Authentication Protocol is over.

Connection protocol

Multiplexes logical channels over the secure authenticated SSH connection (also called tunnel)

Channel types

- session: remote execution of a program
- x11: remote GUI forwarding using the X Window System
- port forwarding: Convert insecure TCP connection to secure SSH connections



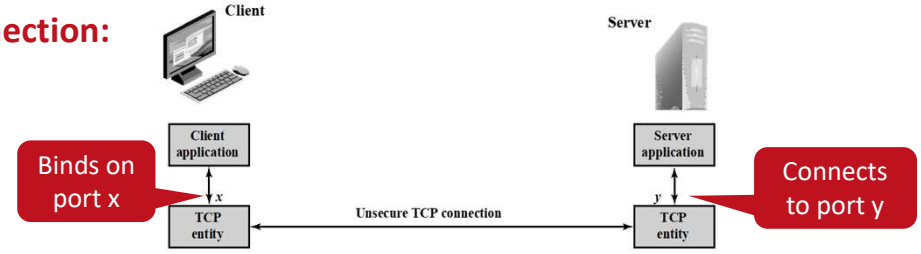
The SSH Connection Protocol runs on top of the SSH Transport Layer Protocol and assumes that a secure authentication connection is in use. That secure authentication connection, referred to as a **tunnel,** is used by the Connection Protocol to multiplex a number of logical channels. All types of communication using SSH, such as a terminal session, are supported using separate channels. Either side may open a channel. For each channel, each side associates a unique channel number, which need not be the same on both ends. 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. The life of a channel progresses through three stages: opening a channel, data transfer, and closing a channel.

Four channel types are recognized in the SSH Connection Protocol specification:

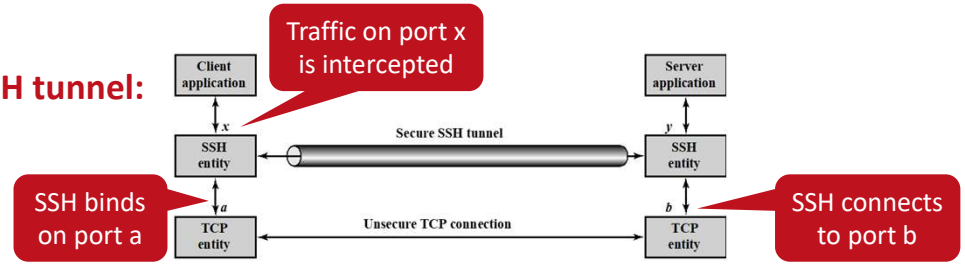
- **session:** The remote execution of a program. The program may be a shell, an application such as file transfer or email, a system command, or some built-in subsystem. Once a session channel is opened, subsequent requests are used to start the remote program.
- **x11:** This refers to the X Window System, a computer software system and network protocol that provides a graphical user interface (GUI) for networked computers. X allows applications to run on a network server but to be displayed on a desktop machine.
- **forwarded-tcpip:** This is remote port forwarding.
- **direct-tcpip:** This is local port forwarding.

SSH port forwarding

Insecure TCP connection:



Secure TCP via SSH tunnel:



University of Antwerp | Faculty of Science

87

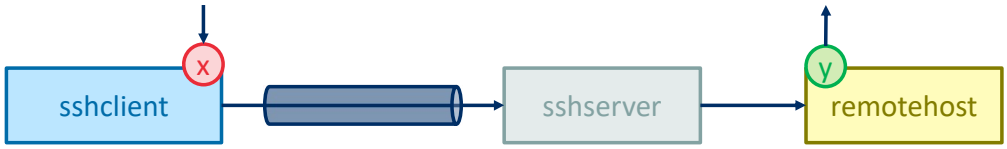
One of the most useful features of SSH is port forwarding. In essence, port forwarding provides the ability to convert any insecure TCP connection into a secure SSH connection. This is also referred to as SSH tunneling. We need to know what a port is in this context. A **port** is an identifier of a user of TCP. So, any application that runs on top of TCP has a port number. Incoming TCP traffic is delivered to the appropriate application on the basis of the port number. An application may employ multiple port numbers. For example, for the Simple Mail Transfer Protocol (SMTP), the server side generally listens on port 25, so an incoming SMTP request uses TCP and addresses the data to destination port 25. TCP recognizes that this is the SMTP server address and routes the data to the SMTP server application.

The figure illustrates the basic concept behind port forwarding. We have a client application that is identified by port number *x* and a server application identified by port number *y*. At some point, the client application invokes the local TCP entity and requests a connection to the remote server on port *y*. The local TCP entity negotiates a TCP connection with the remote TCP entity, such that the connection links local port *x* to remote port *y*.

To secure this connection, SSH is configured so that the SSH Transport Layer Protocol establishes a TCP connection between the SSH client and server entities, with TCP port numbers *a* and *b*, respectively. A secure SSH tunnel is established over this TCP connection. Traffic from the client at port *x* is redirected to the local SSH entity and travels through the tunnel where the remote SSH entity delivers the data to the server application on port *y*. Traffic in the other direction is similarly redirected.

Local SSH port forwarding

Traffic is intercepted on the local SSH host (client) and forwarded to a given host and port via the remote SSH host (server)



OpenSSH command: `ssh -L x:remotehost:y user@sshserver`

Example

Securely send email to SMTP server smtpserv:25 by setting up an SSH tunnel to the SSH server running at smtpserv. Email traffic sent to localhost:2020 will be securely sent over SSH to remote SMTP server smtpserv:25.

```
> ssh -L 2020:smtpserv:25 user@smtpserv
```

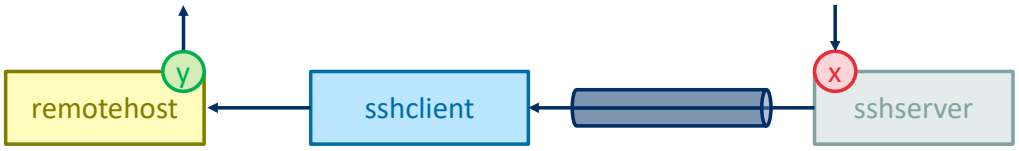
University of Antwerp | Faculty of Science

88

SSH supports two types of port forwarding: local forwarding and remote forwarding. **Local forwarding** allows the client to set up a "hijacker" process. This will 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, the SSH server sends the incoming traffic to the destination port dictated by the client application.

Remote SSH port forwarding

Traffic is intercepted on the remote host (server) and forwarded to the given host and port via the local host (client)



OpenSSH command: `ssh -R x:remotehost:y user@sshserver`

Example

Connect to the work intranet through a firewall at workintra:8080 from homehost, by setting up remote SSH port forwarding from work to a home SSH server. Browsing at home to localhost:9000 shows the intranet website running on workintra:8080

```
> ssh -R 9000:workintra:8080 user@homehost
```

University of Antwerp | Faculty of Science

89

With **remote forwarding**, the user's SSH client acts on the server's behalf. The client receives traffic with a given destination port number, places the traffic on the correct port and sends it to the destination the user chooses. A typical example of remote forwarding is the following. You wish to access a server at work from your home computer. Because the work server is behind a firewall, it will not accept an SSH request from your home computer. However, from work you can set up an SSH tunnel using remote forwarding. This involves the following steps.

1. From the work computer, set up an SSH connection to your home computer. The firewall will allow this, because it is a protected outgoing connection.
2. Configure the SSH server to listen on a local port, say 9000, and to deliver data across the SSH connection addressed to remote port, say 8080.
3. You can now go to your home computer and configure SSH to accept traffic on port 9000.
4. You now have an SSH tunnel that can be used for remote logon to the work server.

When poll is active, respond at **pollev.com/jfamaey**

Text **JFAMAEY** to **+32 460 20 00 56** once to join

A user wants to connect to a database running on dbhost port 5432 that only allows connections from localhost for security reasons, which OpenSSH command(s) allow this?

ssh -L 9000 : dbhost : 5432 user@dbhost

ssh -L 9000 : localhost : 5432 user@dbhost

ssh -R 9000 : dbhost : 5432 user@remotehost

ssh -R 9000 : localhost : 5432 user@remotehost

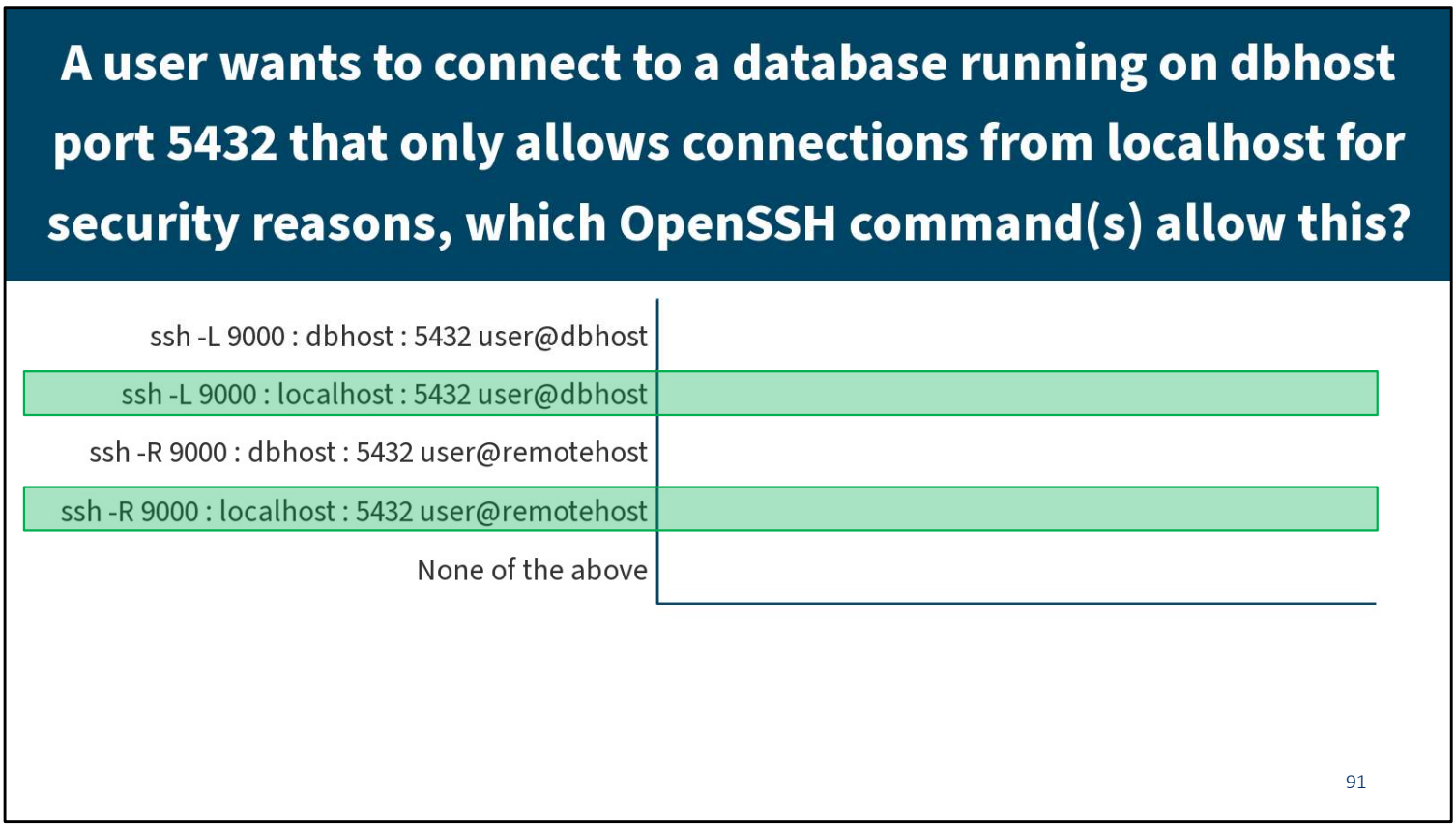
None of the above

90

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

A user wants to connect to a database running on dbhost port 5432 that only allows connections from localhost for security reasons, which OpenSSH command(s) allow this?
https://www.polleverywhere.com/multiple_choice_polls/vZJH3xUR59Wao7B?state=opened&flow=Default&onscreen=persist



Poll Title: A user wants to connect to a database running on dbhost port 5432 that only allows connections from localhost for security reasons, which OpenSSH command(s) allow this?

Summary on Secure Shell

- Provides **confidentiality**, **integrity**, compression and **authentication** for remote login
  - Key exchange based on Diffie-Hellman
  - Supports multiple encryption algorithms (e.g., AES, 3DES)
  - Supports different version of HMAC for integrity
- Supports remote **command execution**, remote **X11 GUI**, and **port forwarding**
- Two types of port forwarding
  - Local: Intercept local traffic on a specific TCP port via SSH client and forward to specific port on remote server via SSH server
  - Remote: Intercept remote traffic on specific TCP port via SSH client and forward to specific port on local computer via SSH server

Link with the book

- Chapter 17 (17.4)

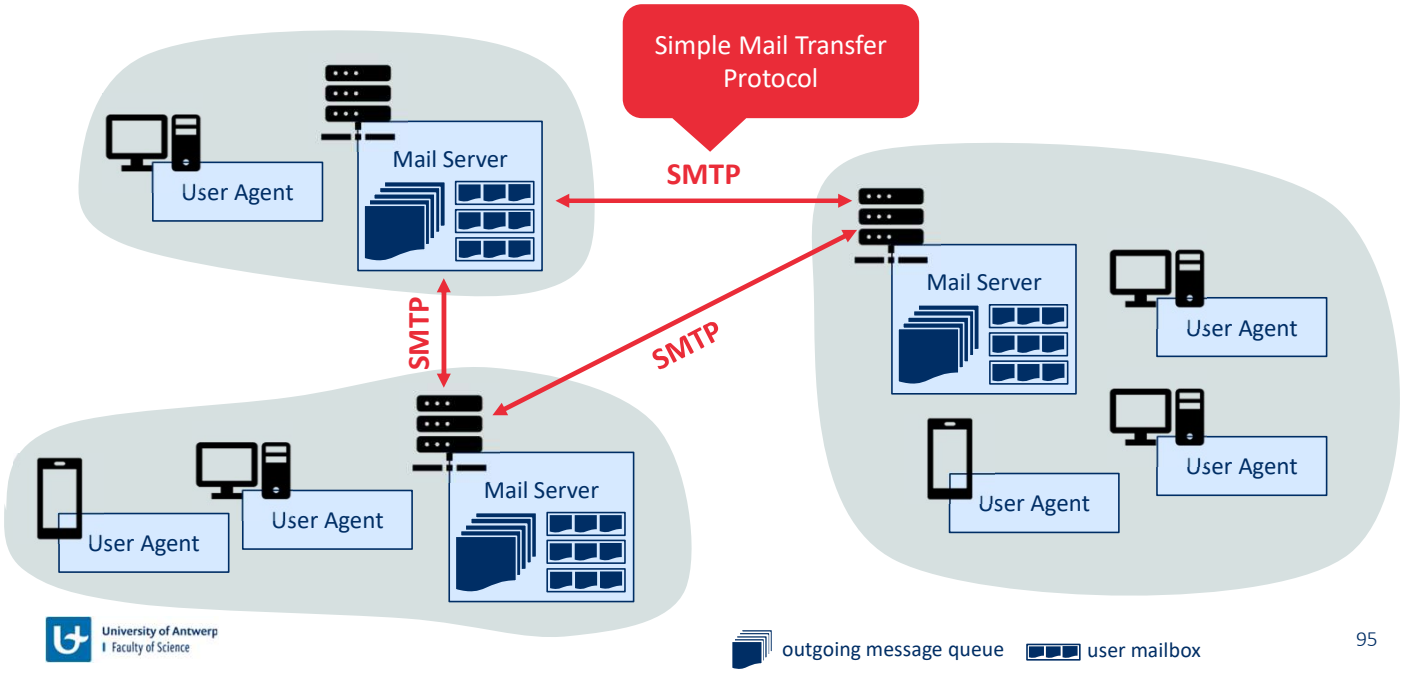
Application and Endpoint Security

University of Antwerp
Faculty of Science

1. Email
2. Firewalls
3. Web application security

University of Antwerp
Faculty of Science

High-level overview of Internet email



The figure presents a high-level view of the Internet mail system. We see from this diagram that it has three major components: **user agents**, **mail servers**, and the **Simple Mail Transfer Protocol (SMTP)**. We now describe each of these components in the context of a sender, Alice, sending an e-mail message to a recipient, Bob. User agents allow users to read, reply to, forward, save, and compose messages. Examples of user agents for e-mail include Microsoft Outlook, Apple Mail, Web-based Gmail, the Gmail App running in a smartphone, and so on. When Alice is finished composing her message, her user agent sends the message to her mail server, where the message is placed in the mail server's outgoing message queue. When Bob wants to read a message, his user agent retrieves the message from his mailbox in his mail server.

Mail servers form the core of the e-mail infrastructure. Each recipient, such as Bob, has a **mailbox** located in one of the mail servers. Bob's mailbox manages and maintains the messages that have been sent to him. A typical message starts its journey in the sender's user agent, then travels to the sender's mail server, and then travels to the recipient's mail server, where it is deposited in the recipient's mailbox. When Bob wants to access the messages in his mailbox, the mail server containing his mailbox authenticates Bob (with his username and password). Alice's mail server must also deal with failures in Bob's mail server. If Alice's server cannot deliver mail to Bob's server, Alice's server holds the message in a **message queue** and attempts to transfer the message later. Reattempts are often done every 30 minutes or so; if there is no success after several days, the server removes the message and notifies the sender (Alice) with an e-mail message.

SMTP is the principal application-layer protocol for Internet electronic mail. It uses the reliable data transfer service of TCP to transfer mail from the sender's mail server to the recipient's mail server. As with most application-layer protocols, SMTP has two sides: a client side, which

executes on the sender's mail server, and a server side, which executes on the recipient's mail server. Both the client and server sides of SMTP run on every mail server. When a mail server sends mail to other mail servers, it acts as an SMTP client. When a mail server receives mail from other mail servers, it acts as an SMTP server.

Multipurpose Internet Mail Extensions (MIME) – RFCs 2045 and 2046

Type	Subtypes	Description
Text	Plain, enriched	Unformatted for formatted text
Multipart	Mixed, parallel, alternative, digest	Content of multiple types transmitted together
Message	Rfc822, partial, external-body	RFC822 compliant, large fragmented message, or pointer to external object
Image	Jpeg, gif	Image in JPEG or GIF format
Video	Mpeg	Video in MPEG format
Audio	Basic	Audio encoded at 8 kHz
Application	Postscript, octet-stream	Postscript file or general binary data

Example

```
From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple boundary"

This is the preamble. It is to be ignored, though it is a handy
place for mail composers to include an explanatory note to non-
MIME conformant readers.
–simple boundary

This is implicitly typed plain ASCII text. It does NOT end with
a linebreak.
–simple boundary
Content-type: text/plain; charset=us-ascii

This is explicitly typed plain ASCII text. It DOES end with a
linebreak.

–simple boundary–
This is the epilogue. It is also to be ignored.
```

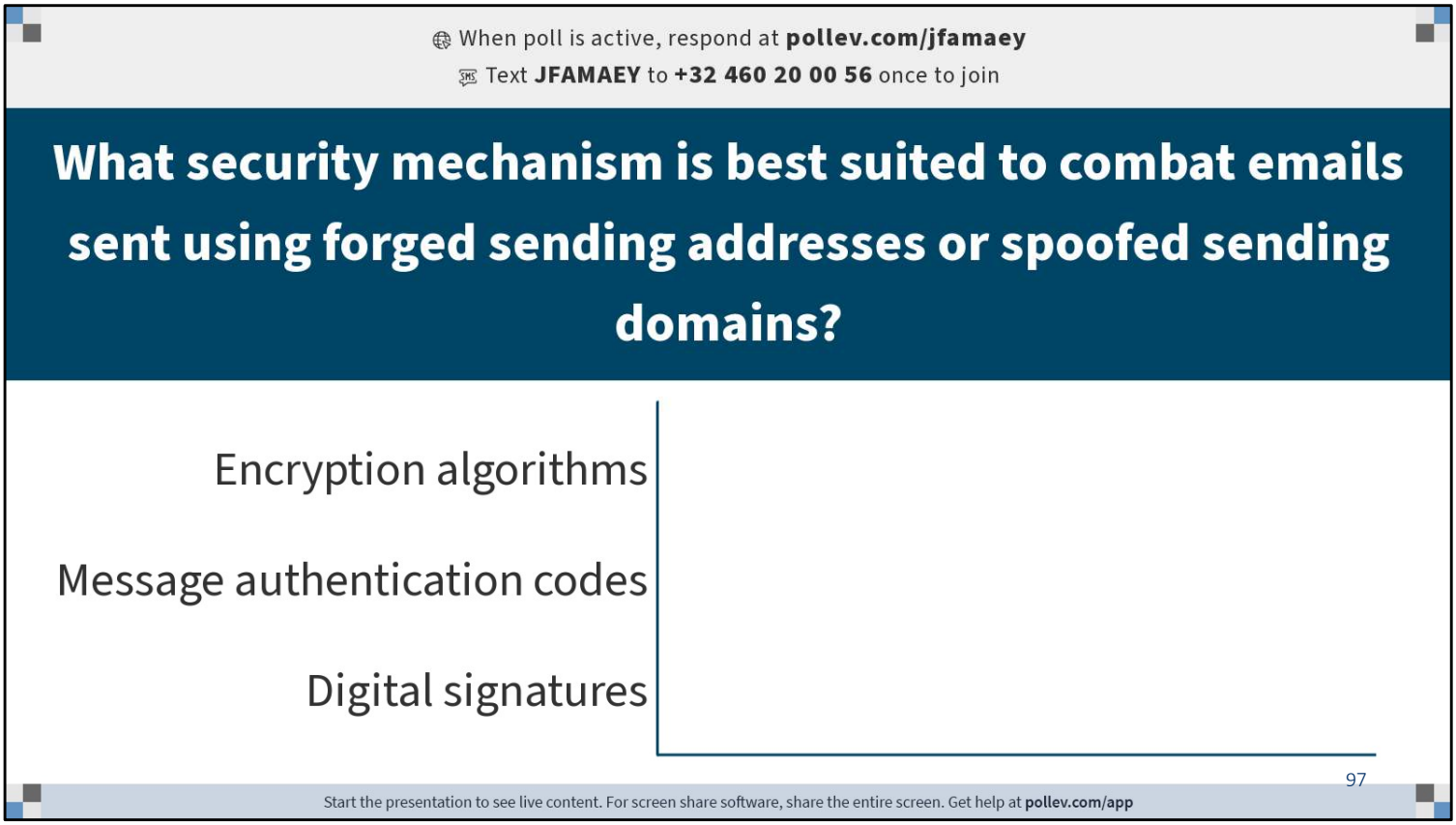
University of Antwerp | Faculty of Science

96

The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment. The table lists the content types specified in RFCs 2045 and 2046. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

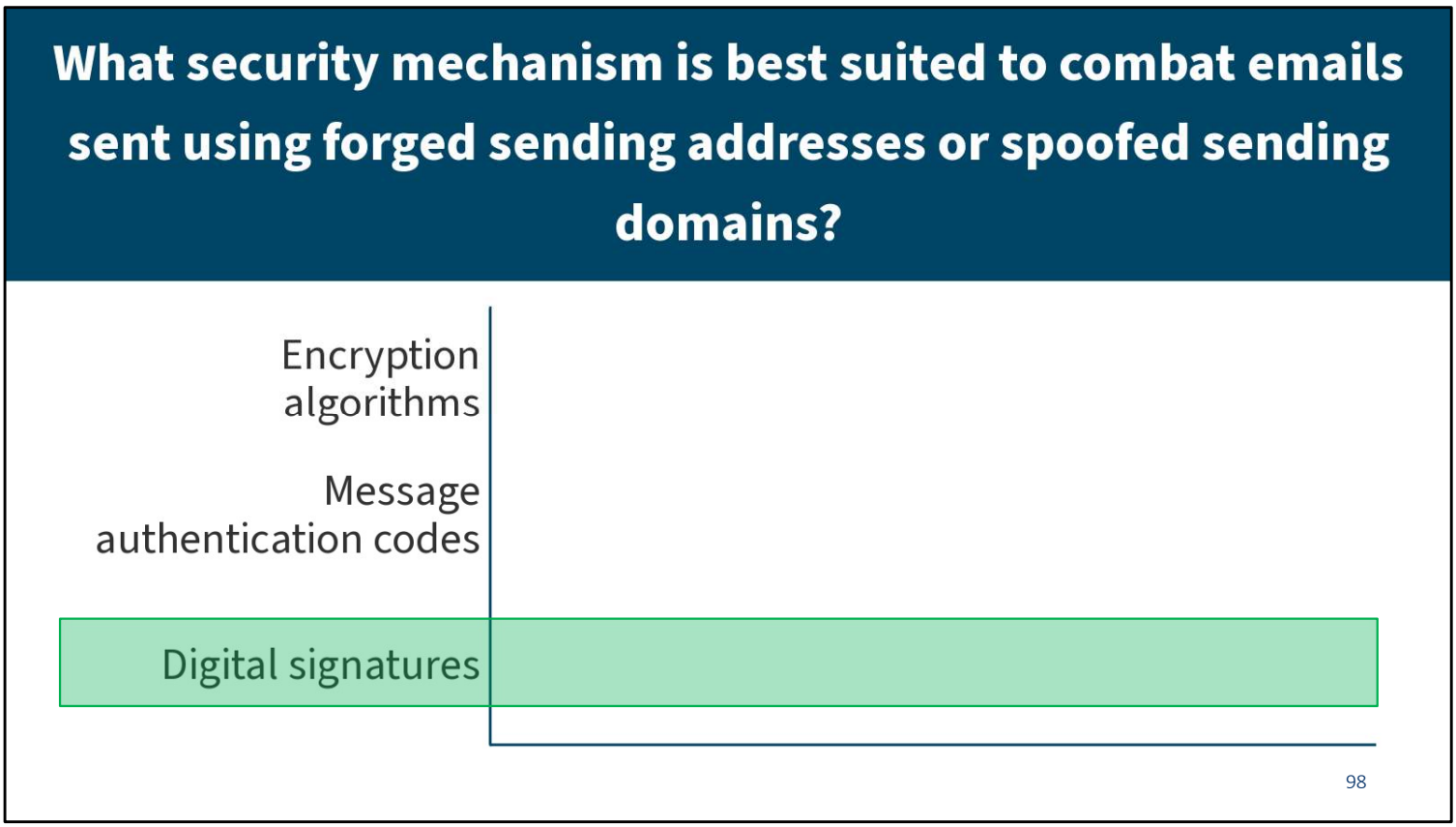
For the **text type** of body, no special software is required to get the full meaning of the text aside from support of the indicated character set. The primary subtype is *plain text*, which is simply a string of ASCII characters or ISO 8859 characters. The *enriched* subtype allows greater formatting flexibility.

The **multipart type** indicates that the body contains multiple, independent parts. The Content-Type header field includes a parameter (called boundary) that defines the delimiter between body parts. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What security mechanism is best suited to combat emails sent using forged sending addresses or spoofed sending domains?
https://www.polleverywhere.com/multiple_choice_polls/CIgmlumKqrTSHok?state=opened&flow=Default&onscreen=persist



Poll Title: What security mechanism is best suited to combat emails sent using forged sending addresses or spoofed sending domains?

Email security threats

Threat	Mitigation
Email sent by unauthorized mail server in enterprise (e.g. malware botnet)	Deployment of domain-based authentication techniques. Use of digital signatures over email.
Email message sent using spoofed or unregistered sending domain	
Email message sent using forged sending address or email address (i.e. phishing, spear phishing)	
Email modified in transit	Use of TLS to encrypt email transfer between server. Use of end-to-end email encryption.
Disclosure of sensitive information (e.g. PII) via monitoring and capturing of email traffic	
Unsolicited Bulk Email (UBE), i.e., spam	Techniques to address UBE (e.g., spam filters)
DoS/DDoS attack against an enterprises' email servers	Multiple mail servers, use of cloud-based email providers.

For both organizations and individuals, email is both pervasive and especially vulnerable to a wide range of security threats. In general terms, email security threats can be classified as follows:

- **Authenticity-related threats:** Could result in unauthorized access to an enterprise's email system.
- **Integrity-related threats:** Could result in unauthorized modification of email content.
- **Confidentiality-related threats:** Could result in unauthorized disclosure of sensitive information.
- **Availability-related threats:** Could prevent end users from being able to send or receive email.

A useful list of specific email threats, together with approaches to mitigation, is provided in NIST SP 800-177 (*Trustworthy Email*, September 2015) and is shown in the table.

Secure MIME (S/MIME)

- Compatible with MIME (RFC 2045—2049)
- Standardized in many RFCs (e.g., 5750, 5751, 2634)

Functions	Default Algorithm
Digital signature	RSA with SHA-256
Message encryption	AES-128 with CBC
Private key exchange	RSA
Public key exchange	X.509 certificates
Compression	No compression
E-mail compatibility	Radix-64 conversion

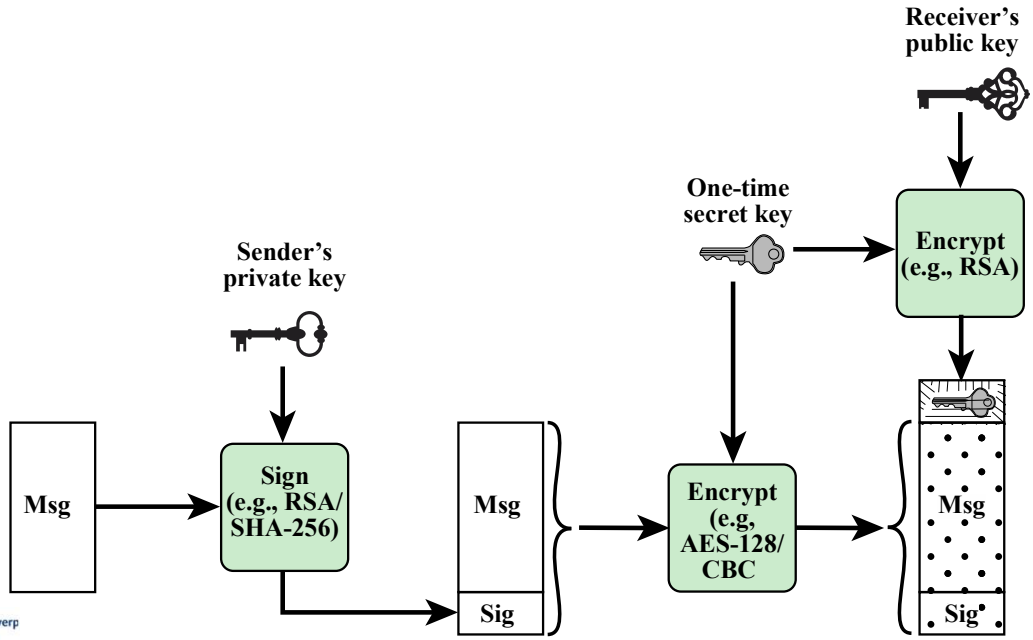
University of Antwerp | Faculty of Science

100

Secure/Multipurpose Internet Mail Extension (S/MIME) is a security enhancement to the MIME Internet email format standard based on technology from RSA Data Security. S/MIME is a complex capability that is defined in a number of documents. The most important documents relevant to S/MIME include the following:

- **RFC 5750, S/MIME Version 3.2 Certificate Handling:** Specifies conventions for X.509 certificate usage by (S/MIME) v3.2.
- **RFC 5751, S/MIME Version 3.2 Message Specification:** The principal defining document for S/MIME message creation and processing.
- **RFC 4134, Examples of S/MIME Messages:** Gives examples of message bodies formatted using S/MIME.
- **RFC 2634, Enhanced Security Services for S/MIME:** Describes four optional security service extensions for S/MIME.
- **RFC 5652, Cryptographic Message Syntax (CMS):** Describes the Crypto- graphic Message Syntax (CMS). This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.
- **RFC 3370, CMS Algorithms:** Describes the conventions for using several cryptographic algorithms with the CMS.
- **RFC 5752, Multiple Signatures in CMS:** Describes the use of multiple, parallel signatures for a message.
- **RFC 1847, Security Multiparts for MIME—Multipart/Signed and Multipart/ Encrypted:** Defines a framework within which security services may be applied to MIME body parts. The use of a digital signature is relevant to S/MIME, as explained subsequently.

Simplified S/MIME message sending

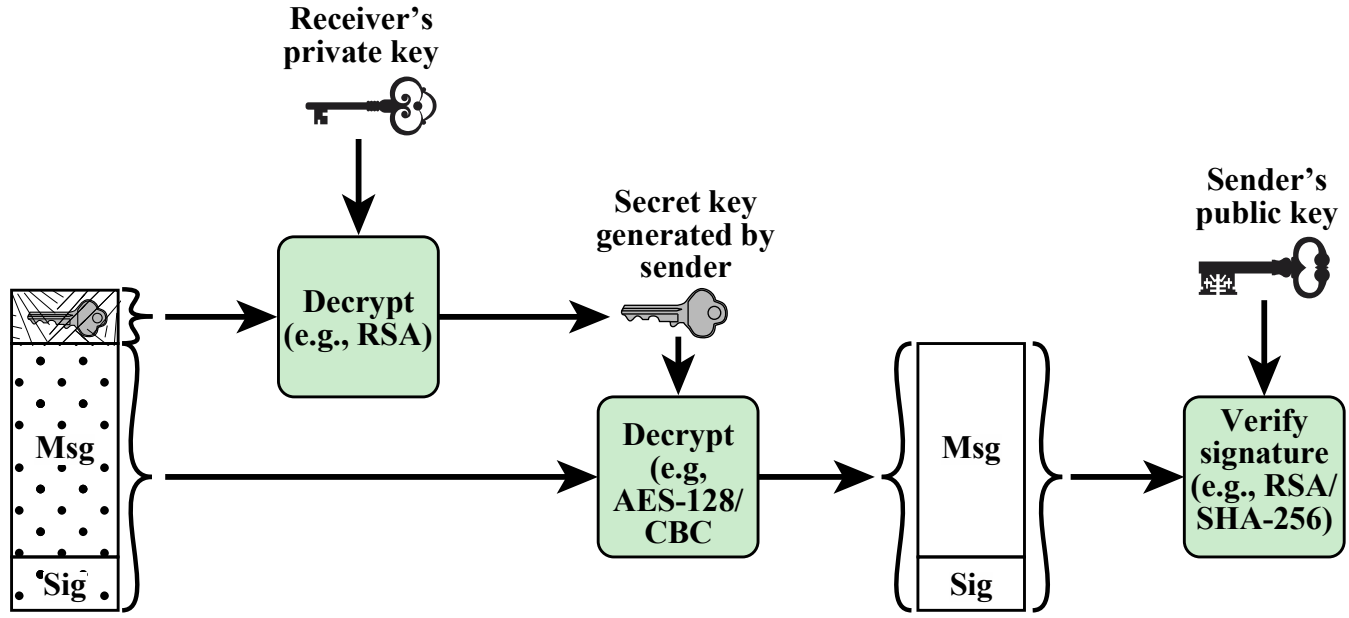


Both confidentiality and authentication may be used for the same message. The figure shows a sequence in which a signature is generated for the plaintext message and appended to the message. Then the plaintext message and signature are encrypted as a single block using symmetric encryption and the symmetric encryption key is encrypted using public-key encryption.

S/MIME allows the signing and message encryption operations to be per- formed in either order. If signing is done first, the identity of the signer is hidden by the encryption. Plus, it is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature.

If encryption is done first, it is possible to verify a signature without exposing the message content. This can be useful in a context in which automatic signature verification is desired, as no private key material is required to verify a signature. However, in this case the recipient cannot determine any relationship between the signer and the unencrypted content of the message.

Simplified S/MIME message receiving



Pretty Good Privacy (PGP)

- Originally developed by Phil Zimmerman
- Provides confidentiality and authentication for e-mail and file storage applications
- Standardized as RFC 3156

Functions	Default Algorithms
Digital signature	DSS or RSA with SHA-256
Message encryption	CAST, IDEA, or 3DES
Private key exchange	RSA or Diffie-Hellman
Public key exchange	Web of Trust
Compression	ZIP
E-mail compatibility	Radix-64 conversion

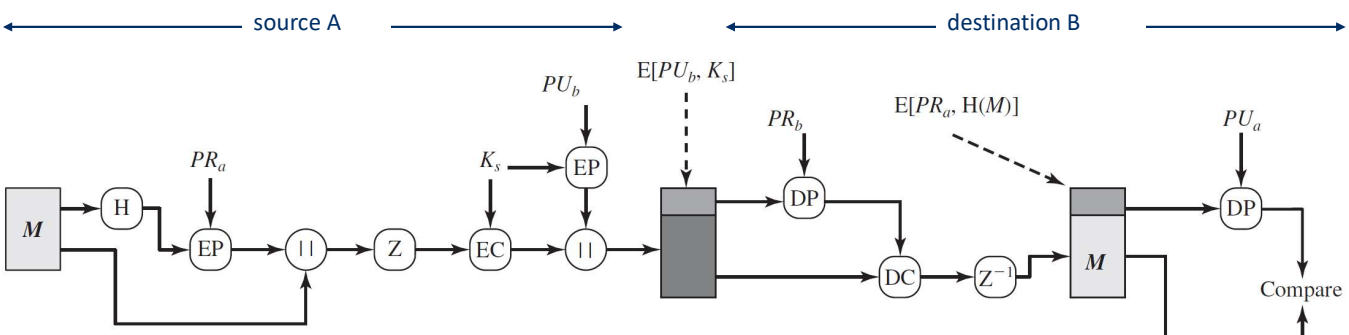
Phil Zimmerman

An alternative email security protocol is Pretty Good Privacy (PGP), which has essentially the same functionality as S/MIME. PGP was created by Phil Zimmerman and implemented as a product first released in 1991. It was made available free of charge and became quite popular for personal use. The initial PGP protocol was proprietary and used some encryption algorithms with intellectual property restrictions. In 1996, version 5.x of PGP was defined in IETF RFC 1991, *PGP Message Exchange Formats*. Subsequently, OpenPGP was developed as a new standard protocol based on PGP version 5.x. OpenPGP is defined in RFC 4880 (*OpenPGP Message Format*, November 2007) and RFC 3156 (*MIME Security with OpenPGP*, August 2001).

The actual operation of PGP, as opposed to the management of keys, consists of four services: authentication, confidentiality, compression, and e-mail compatibility.

PGP: Authentication and confidentiality

- Authentication and confidentiality can be provided independently
- Digital signature can be attached to message or sent separately



H = hash function
Z = compression
PR = private key

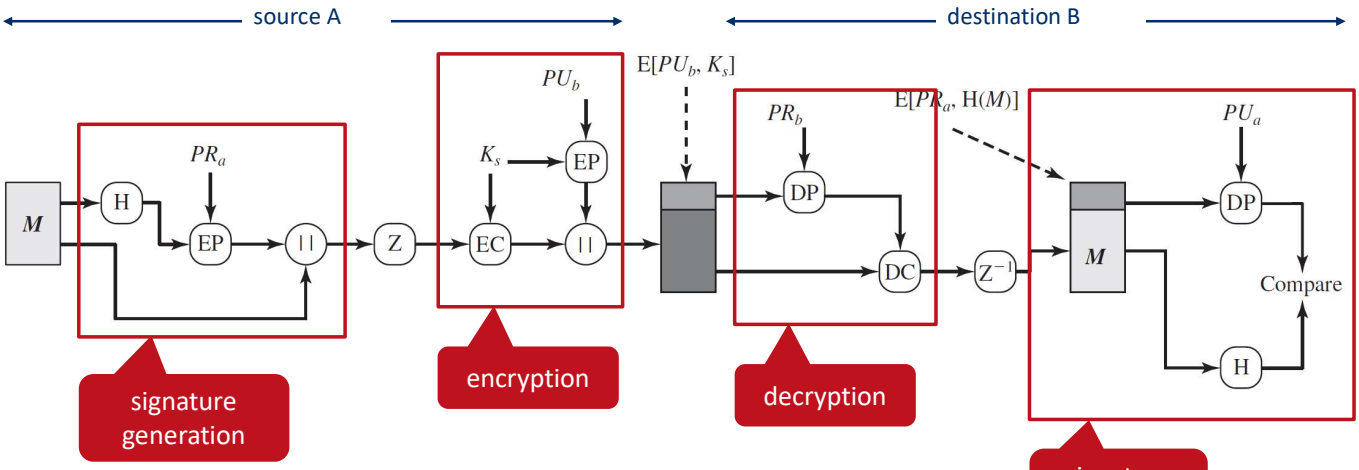
EP = public-key encryption
DP = public-key decryption
PU = public key

EC = private-key encryption
DC = private-key decryption
K_s = session key

Both authentication and confidentiality services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal). This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature. In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and finally encrypts the session key with the recipient's public key.

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

PGP: Authentication and confidentiality



H = hash function
Z = compression
PR = private key

EP = public-key encryption
DP = public-key decryption
PU = public key

EC = private-key encryption
DC = private-key decryption
K_s = session key

University of Antwerp
Faculty of Science

105

Both authentication and confidentiality services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal). This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message. Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature. In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and finally encrypts the session key with the recipient's public key.

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

PGP compression

- Why perform **compression after** signature generation?
  - Then only the uncompressed message needs to be stored to validate the signature afterwards
  - Dynamically regenerating the compressed message is not feasible as the ZIP algorithm is non-deterministic
- Why perform **compression before** encryption?
  - Stronger cryptographic security against frequency attacks and due to reduced redundancy in compressed message

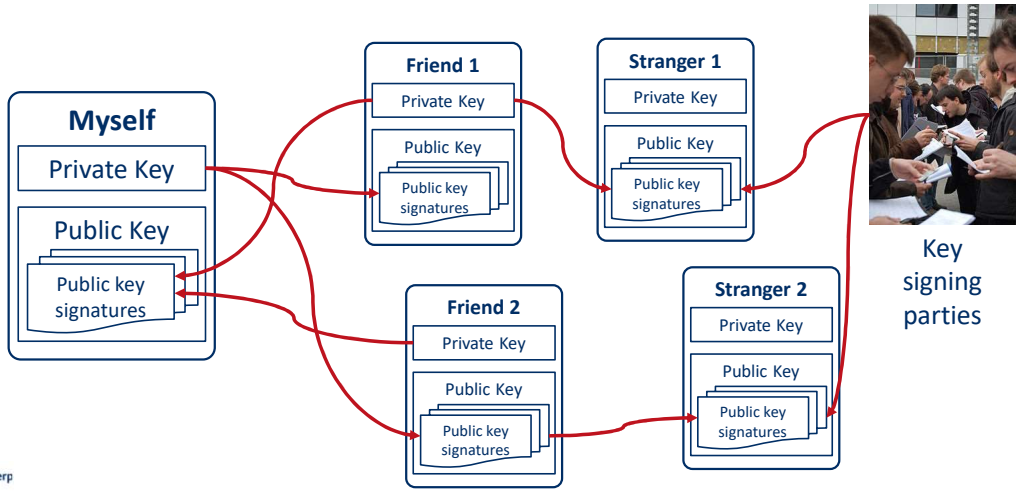
The signature is generated before compression for two reasons:

1. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.
2. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.

Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

PGP uses Web of Trust to validate public keys

- It relies on self-signed certificates, that can in turn be signed by your circle of trusted friends
- This creates a web of people that trust each other (in)directly



Differences between S/MIME and PGP

Key certification

- S/MIME uses X.509 certificates and CAs to certify public keys
- In PGP, users generate their own public-private key pairs, trust is gained by signing them by someone already trusted by the recipient (Web of Trust)

Key distribution

- S/MIME attaches the sender's public key and the certificate to each message
- PGP puts the responsibility of distributing keys with the user, which often post them to TLS-protected websites (e.g., https://pgp.mit.edu/)

Due to greater confidence in the CA system over a Web of Trust, NIST recommends the use of S/MIME over PGP

University of Antwerp | Faculty of Science

There are two significant differences between S/MIME and OpenPGP:

- **Key Certification:** S/MIME uses X.509 certificates that are issued by Certificate Authorities (or local agencies that have been delegated authority by a CA to issue certificates). In OpenPGP, users generate their own OpenPGP public and private keys and then solicit signatures for their public keys from individu- als or organizations to which they are known. Whereas X.509 certificates are trusted if there is a valid PKIX chain to a trusted root, an OpenPGP public key is trusted if it is signed by another OpenPGP public key that is trusted by the recipient. This is called the *Web-of-Trust*.
- **Key Distribution:** OpenPGP does not include the sender's public key with each message, so it is necessary for recipients of OpenPGP messages to separately obtain the sender's public key in order to verify the message. Many organizations post OpenPGP keys on TLS-protected websites: People who wish to verify digital signatures or send these organizations encrypted mail need to manually download these keys and add them to their OpenPGP clients. Keys may also be registered with the OpenPGP public key servers, which are servers that maintain a database of PGP public keys organized by email address. Anyone may post a public key to the OpenPGP key servers, and that public key may contain any email address. There is no vetting of OpenPGP keys, so users must use the Web-of-Trust to decide whether to trust a given public key.

NIST 800-177 recommends the use of S/MIME rather than PGP because of the greater confidence in the CA system of verifying public keys.

Summary on email security

- Two widely used IETF standards with similar functionality and structure that provide message authentication, compression, and/or confidentiality
  - Signature is based on message digest (hash) in combination with public key encryption
  - Compression is done after signature generation, but before encryption
  - Private session key is used to encrypt message using symmetric encryption
  - Session key is encrypted using receiver's public key and attached to the message
- Secure MIME (S/MIME): most popular for professional usage
  - Uses certificates and certification authorities for public key distribution
- Pretty Good Privacy (PGP): most popular among private users
  - Uses "web of trust" for public key distribution

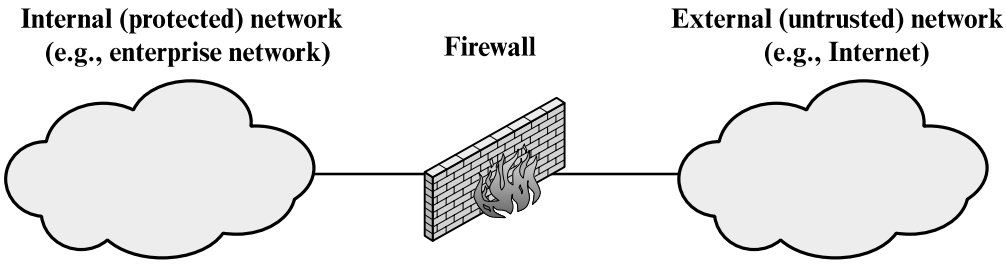
Link with the book

- Chapter 19 (19.1 – 19.5)

1. Email
2. Firewalls
3. Web application security

University of Antwerp
Faculty of Science

General firewall model



Design goals

1. All traffic in and out must pass through the firewall
2. Only authorized traffic is allowed to pass through
3. The firewall is immune to penetration

Traffic filtering properties

source and destination IP address, port numbers, application protocol, user identity, network activity (DoS, scanning, …), etc.

Design goals of a firewall:

1. All traffic from inside to outside, and vice versa, must pass through the firewall. This is achieved by physically blocking all access to the local network except via the firewall. Various configurations are possible.
2. Only authorized traffic, as defined by the local security policy, will be allowed to pass. Various types of firewalls are used, which implement various types of security policies.
3. The firewall itself is immune to penetration. This implies the use of a hardened system with a secured operating system. Trusted computer systems are suitable for hosting a firewall and often required in government applications.

SP 800-41-1 (*Guidelines on Firewalls and Firewall Policy*, September 2009) lists a range of characteristics that a firewall access policy could use to filter traffic, including:

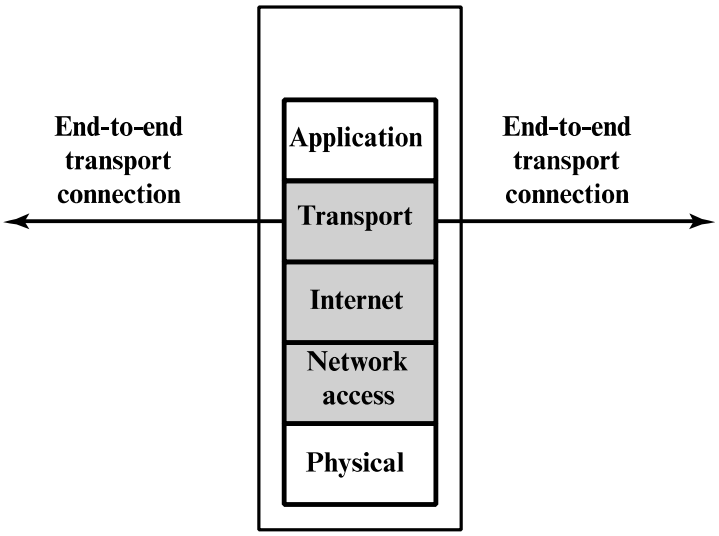
- **IP Address and Protocol Values:** Controls access based on the source or destination addresses and port numbers, direction of flow being inbound or outbound, and other network and transport layer characteristics. This type of filtering is used by packet filter and stateful inspection firewalls. It is typically used to limit access to specific services.
- **Application Protocol:** Controls access on the basis of authorized application protocol data. This type of filtering is used by an application-level gateway that relays and monitors the exchange of information for specific application protocols, for example, checking SMTP e-mail for spam, or HTPP Web requests to authorized sites only.
- **User Identity:** Controls access based on the users identity, typically for inside users who identify themselves using some form of secure authentication technology, such as IPSec.
- **Network Activity:** Controls access based on considerations such as the time or request, for example, only in business hours; rate of requests, for example, to detect scanning attempts; or other activity patterns.

Firewall types

Packet filtering	Network access rules are applied to each packet, separately which is then forwarded or discarded
Stateful inspection	Keeps track of outbound TCP connections, so only currently active TCP ports have to be opened
Application proxy	Acts as a termination point for TCP connections, relaying traffic after user authentication
Circuit-level proxy	Similar to an application proxy, but does not examine content after connection establishment

A firewall can monitor network traffic at a number of levels, from low-level network packets either individually or as part of a flow, to all traffic within a transport connection, up to inspecting details of application protocols. The choice of which level is appropriate is determined by the desired firewall access policy. It can operate as a positive filter, allowing to pass only packets that meet specific criteria, or as a negative filter, rejecting any packet that meets certain criteria. The criteria implement the access policy for the firewall, that we discussed in the previous section. Depending on the type of firewall, it may examine one or more protocol headers in each packet, the payload of each packet, or the pattern generated by a sequence of packets.

Packet filtering firewall



Two types of default behavior

- Whitelisting: packets are discarded by default
- Blacklisting: packets are forwarded by default

University of Antwerp | Faculty of Science

113

A packet filtering firewall applies a set of rules to each incoming and outgoing IP packet and then forwards or discards the packet. The firewall is typically configured to filter packets going in both directions (from and to the internal network). Filtering rules are based on information contained in a network packet:

- **Source IP address:** The IP address of the system that originated the IP packet (e.g., 192.178.1.1).
- **Destination IP address:** The IP address of the system the IP packet is trying to reach (e.g., 192.168.1.2).
- **Source and destination transport-level address:** The transport-level (e.g., TCP or UDP) port number, which defines applications such as SNMP or TELNET.
- **IP protocol field:** Defines the transport protocol.
- **Interface:** For a firewall with three or more ports, which interface of the firewall the packet came from or which interface of the firewall the packet is destined for.

The packet filter is typically set up as a list of rules based on matches to fields in the IP or TCP header. If there is a match to one of the rules, that rule is invoked to determine whether to forward or discard the packet. If there is no match to any rule, then a default action is taken. Two default policies are possible:

- **Default** = **discard:** That which is not expressly permitted is prohibited (whitelisting).
- **Default** = **forward:** That which is not expressly prohibited is permitted (blacklisting).

The default discard policy is more conservative. Initially, everything is blocked, and services must be added on a case-by-case basis. This policy is more visible to users, who are more likely to see the firewall as a hindrance. The default forward policy increases ease of use for end users but provides reduced security; the security administrator must, in essence, react to each new security threat as it becomes known.

Packet filtering example

Only allow in- and outbound SMTP traffic (port 25)

Rule	Direction	Source Address	Destination Address	Protocol	Destination Port	Action
A	In	External	Internal	TCP	25	Permit
B	Out	Internal	External	TCP	> 1023	Permit
C	Out	Internal	External	TCP	25	Permit
D	In	External	Internal	TCP	> 1023	Permit
E	Either	Any	Any	Any	Any	Deny

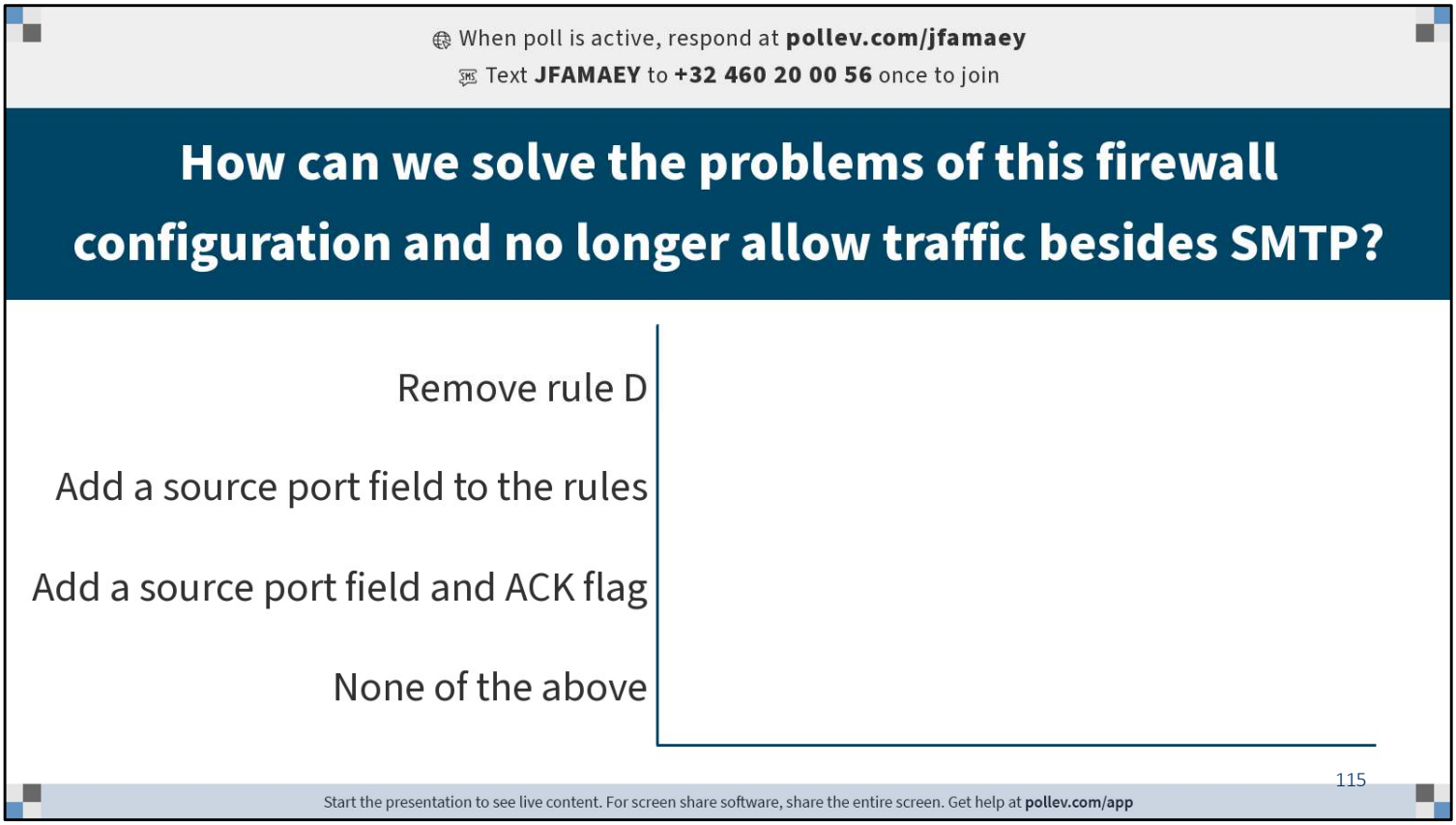
This configuration has some problems

Rule D allows any incoming external TCP traffic on ports above 1023

This is a simplified example of a ruleset for SMTP traffic. The goal is to allow inbound and outbound e-mail traffic but to block all other traffic. The rules are applied top to bottom to each packet:

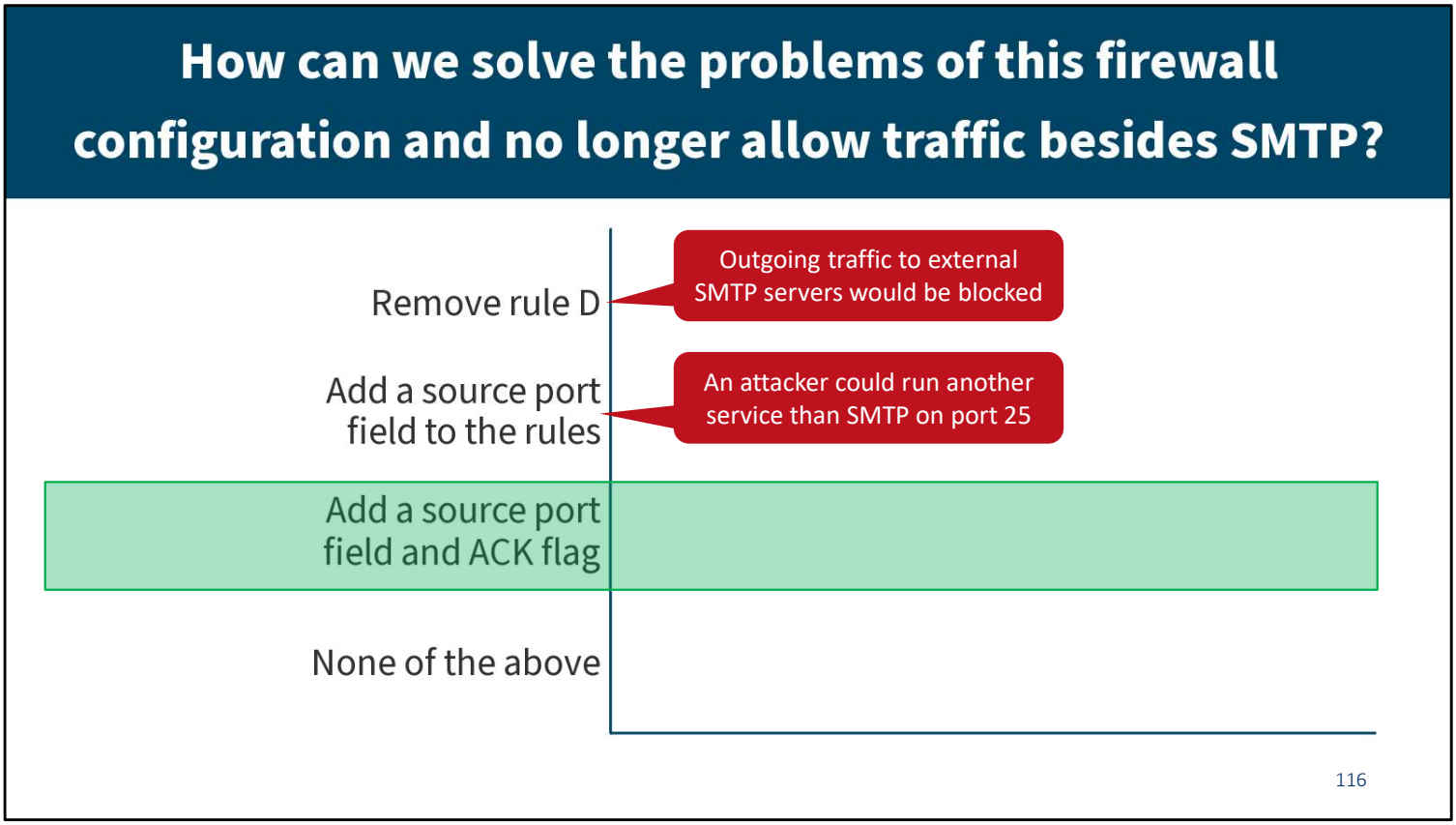
A. Inbound mail from an external source is allowed (port 25 is for SMTP incoming).
B. This rule is intended to allow a response to an inbound SMTP connection.
C. Outbound mail to an external source is allowed.
D. This rule is intended to allow a response to an inbound SMTP connection.
E. This is an explicit statement of the default policy. All rulesets include this rule implicitly as the last rule.

There are several problems with this ruleset. Rule D allows external traffic to any destination port above 1023. As an example of an exploit of this rule, an external attacker can open a connection from the attacker's port 5150 to an internal Web proxy server on port 8080. This is supposed to be forbidden and could allow an attack on the server.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

How can we solve the problems of this firewall configuration and no longer allow traffic besides SMTP?
https://www.polleverywhere.com/multiple_choice_polls/wjBa2pfwapwTh0z?state=opened&flow=Default&onscreen=persist



Poll Title: How can we solve the problems of this firewall configuration and no longer allow traffic besides SMTP?

Packet filtering example

Only allow in- and outbound SMTP traffic (port 25)

Rule	Direction	Source Address	Destination Address	Protocol	Destination Port	Action
A	In	External	Internal	TCP	25	Permit
B	Out	Internal	External	TCP	> 1023	Permit
C	Out	Internal	External	TCP	25	Permit
D	In	External	Internal	TCP	> 1023	Permit
E	Either	Any	Any	Any	Any	Deny

This configuration has some problems

Rule D allows any incoming external TCP traffic on ports above 1023

Solution

Rule	Direction	Source Address	Source Port	Dest Address	Protocol	Dest Port	Flag	Action
D	In	External	25	Internal	TCP	> 1023	ACK	Permit

University of Antwerp | Faculty of Science

117

To counter this attack, the firewall ruleset can be configured with a source port field for each row. For rules B and D, the source port is set to 25; for rules A and C, the source port is set to > 1023.

But a vulnerability remains. Rules C and D are intended to specify that any inside host can send mail to the outside. A TCP packet with a destination port of 25 is routed to the SMTP server on the destination machine. The problem with this rule is that the use of port 25 for SMTP receipt is only a default; an outside machine could be configured to have some other application linked to port 25. As the revised rule D is written, an attacker could gain access to internal machines by sending packets with a TCP source port number of 25. To counter this threat, we can add an ACK flag field to each row. For rule D, the field would indicate that the ACK flag must be set on the incoming packet.

The rule takes advantage of a feature of TCP connections. Once a connection is set up, the ACK flag of a TCP segment is set to acknowledge segments sent from the other side. Thus, this rule allows incoming packets with a source port number of 25 that include the ACK flag in the TCP segment.

Question: Packet filtering

Question: Given the packet filter ruleset below, which of the listed packets (1 to 4) are permitted by the firewall?

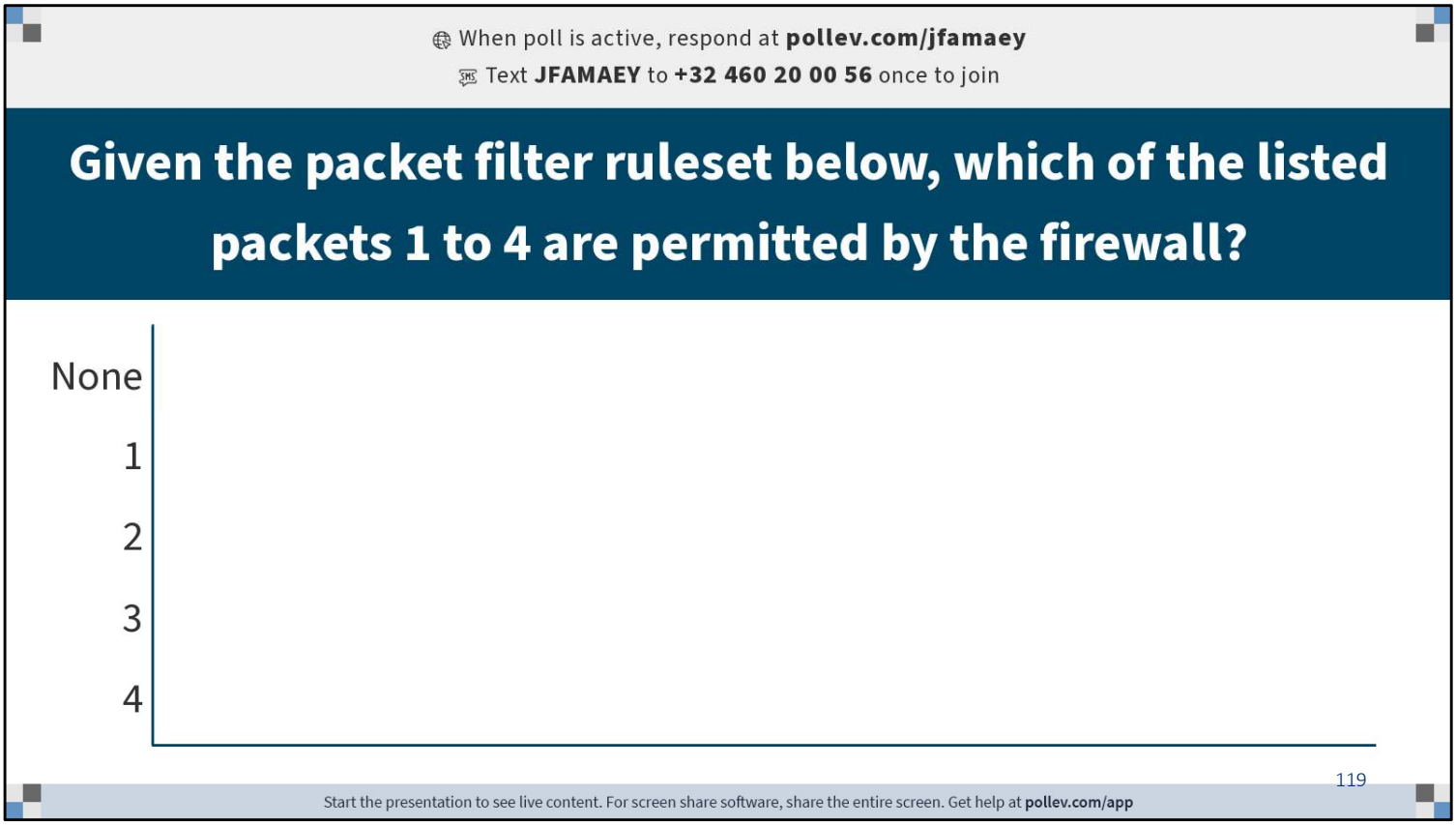
Packet filter ruleset

Rule	Direction	Src Addr	Dest Addr	Protocol	Dest Port	Action
A	In	External	Internal	TCP	25	Permit
B	Out	Internal	External	TCP	>1023	Permit
C	Out	Internal	External	TCP	25	Permit
D	In	External	Internal	TCP	>1023	Permit
E	Either	Any	Any	Any	Any	Deny

172.16.1.1 = internal address
192.168.3.4 = external address

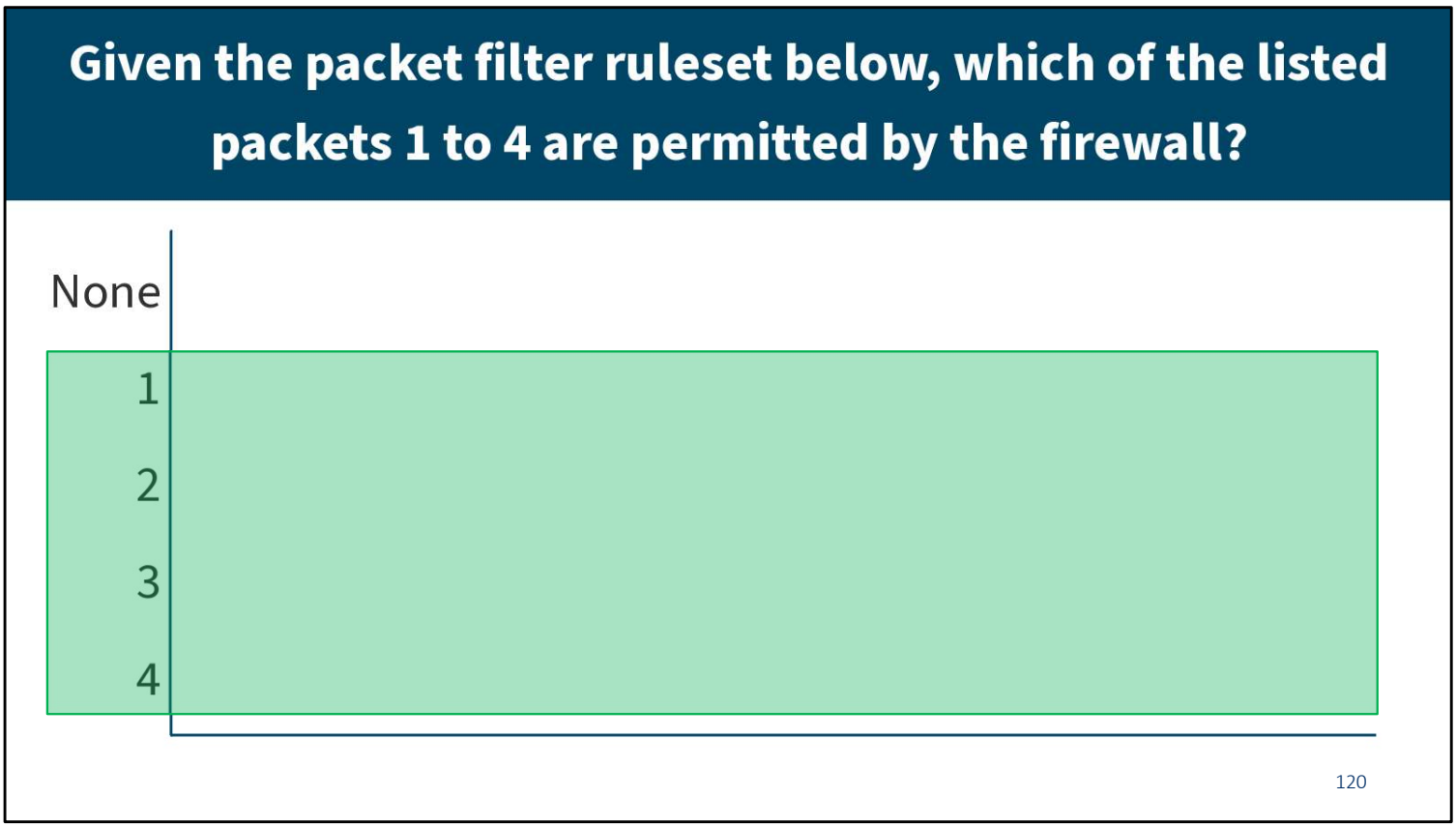
Packets

Packet	Direction	Src Addr	Dest Addr	Protocol	Dest Port	Action
1	In	192.168.3.4	172.16.1.1	TCP	25	?
2	Out	172.16.1.1	192.168.3.4	TCP	1234	?
3	Out	172.16.1.1	192.168.3.4	TCP	25	?
4	In	192.168.3.4	172.16.1.1	TCP	1357	?



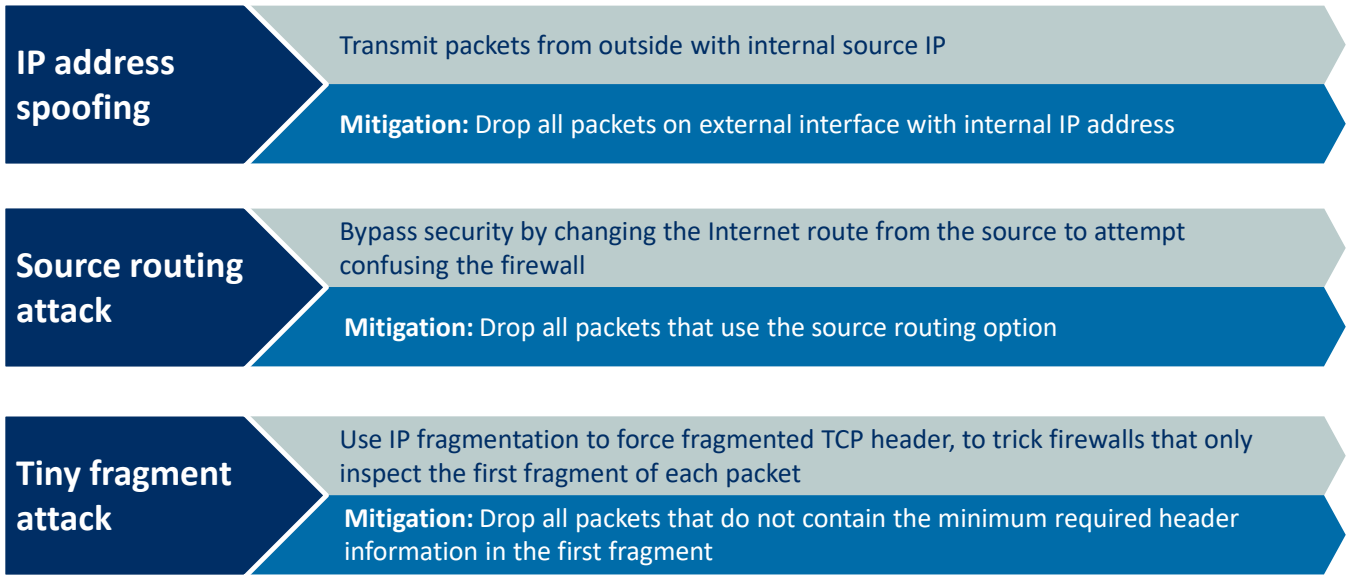
Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

Given the packet filter ruleset below, which of the listed packets 1 to 4 are permitted by the firewall?
https://www.polleverywhere.com/multiple_choice_polls/ZdTSLad4WK7YEM5?state=opened&flow=Default&onscreen=persist



Poll Title: Given the packet filter ruleset below, which of the listed packets 1 to 4 are permitted by the firewall?

Attacks against packet filtering firewalls



Some of the attacks that can be made on packet filtering firewalls and the appropriate countermeasures are the following:

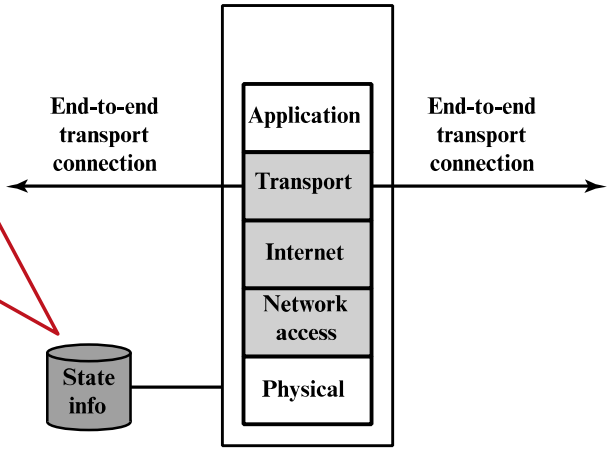
- **IP address spoofing:** The intruder transmits packets from the outside with a source IP address field containing an address of an internal host. The attacker hopes that the use of a spoofed address will allow penetration of systems that employ simple source address security, in which packets from specific trusted internal hosts are accepted. The countermeasure is to discard packets with an inside source address if the packet arrives on an external interface. In fact, this countermeasure is often implemented at the router external to the firewall.
- **Source routing attacks:** The source station specifies the route that a packet should take as it crosses the Internet, in the hopes that this will bypass security measures that do not analyze the source routing information. The countermeasure is to discard all packets that use this option.
- **Tiny fragment attacks:** The intruder uses the IP fragmentation option to create extremely small fragments and force the TCP header information into a separate packet fragment. This attack is designed to circumvent filtering rules that depend on TCP header information. Typically, a packet filter will make a filtering decision on the first fragment of a packet. All subsequent fragments of that packet are filtered out solely on the basis that they are part of the packet whose first fragment was rejected. The attacker hopes that the filtering firewall examines only the first fragment and that the remaining fragments are passed through. A tiny fragment attack can be defeated by enforcing a rule that the first fragment of a packet must contain a predefined minimum amount of the transport header. If the first fragment is rejected, the filter can remember the packet and discard all subsequent fragments.

Stateful inspection firewall

Directory of outbound TCP connections

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.22.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
2122.22.123.32	2112	192.168.1.6	80	Established
210.922.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.21.22.12	1046	192.168.1.6	80	Established

Incoming traffic to ports > 1023 can now be blocked if there is no entry associated with it in the directory



University of Antwerp | Faculty of Science

122

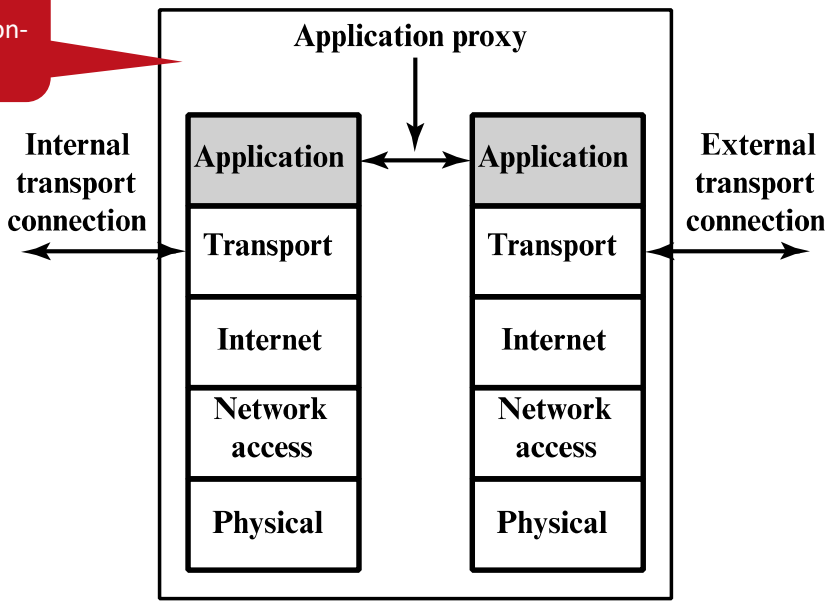
A traditional packet filter makes filtering decisions on an individual packet basis and does not take into consideration any higher-layer context. In general, when an application that uses TCP creates a session with a remote host, it creates a TCP connection in which the TCP port number for the remote (server) application is a number less than 1024 and the TCP port number for the local (client) application is a number between 1024 and 65535. The numbers less than 1024 are the "well-known" port numbers and are assigned permanently to particular applications (e.g., 25 for server SMTP). The numbers between 1024 and 65535 are generated dynamically and have temporary significance only for the liftime of a TCP connection.

A simple packet filtering firewall must permit inbound network traffic on all these high-numbered ports for TCP-based traffic to occur. This creates a vulnerability that can be exploited by unauthorized users. A **stateful inspection packet firewall** tightens up the rules for TCP traffic by creating a directory of outbound TCP connections. There is an entry for each currently established connection. The packet filter will now allow incoming traffic to high-numbered ports only for those packets that fit the profile of one of the entries in this directory.

A stateful packet inspection firewall reviews the same packet information as a packet filtering firewall, but also records information about TCP connections. Some stateful firewalls also keep track of TCP sequence numbers to prevent attacks that depend on the sequence number, such as session hijacking. Some even inspect limited amounts of application data for some well-known protocols like FTP, IM and SIPS commands, in order to identify and track related connections.

Application proxy firewall

The proxy terminates all TCP connections and checks application-layer payloads of packets



University of Antwerp | Faculty of Science

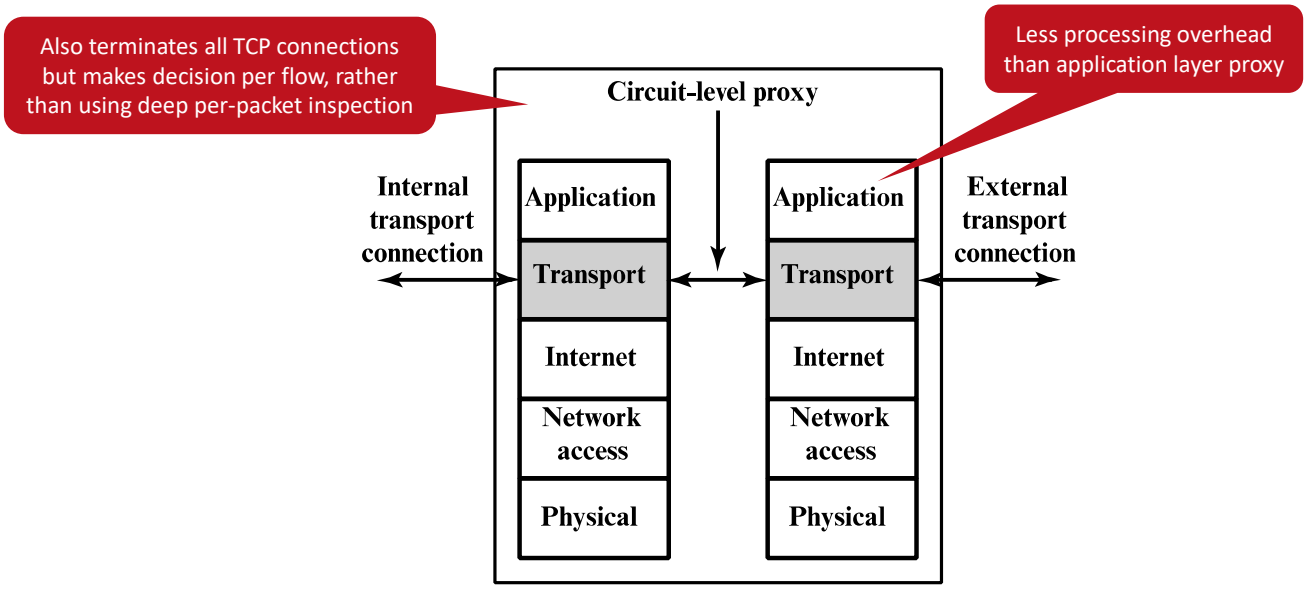
123

An application-level gateway, also called an **application proxy**, acts as a relay of application-level traffic. The user contacts the gateway using a TCP/IP application, such as Telnet or FTP, and the gateway asks the user for the name of the remote host to be accessed. When the user responds and provides a valid user ID and authentication information, the gateway contacts the application on the remote host and relays TCP segments containing the application data between the two end-points. If the gateway does not implement the proxy code for a specific application, the service is not supported and cannot be forwarded across the firewall. Further, the gateway can be configured to support only specific features of an application that the network administrator considers acceptable while denying all other features.

Application-level gateways tend to be more secure than packet filters. Rather than trying to deal with the numerous possible combinations that are to be allowed and forbidden at the TCP and IP level, the application-level gateway need only scrutinize a few allowable applications. In addition, it is easy to log and audit all incoming traffic at the application level.

A prime **disadvantage** of this type of gateway is the additional processing overhead on each connection. In effect, there are two spliced connections between the end users, with the gateway at the splice point, and the gateway must examine and forward all traffic in both directions.

Circuit-level proxy firewall



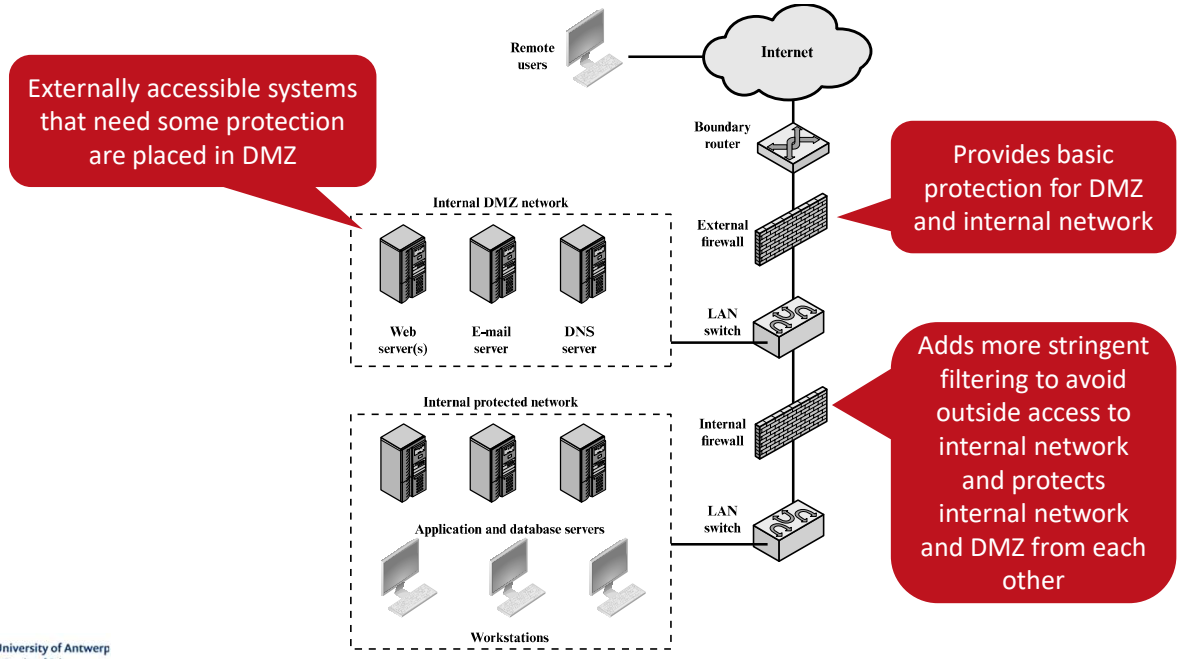
University of Antwerp | Faculty of Science

124

A fourth type of firewall is the circuit-level gateway or **circuit-level proxy**. This can be a stand-alone system or it can be a specialized function performed by an application-level gateway for certain applications. As with an application gateway, a circuit-level gateway does not permit an end-to-end TCP connection; rather, the gateway sets up two TCP connections, one between itself and a TCP user on an inner host and one between itself and a TCP user on an outside host. Once the two connections are established, the gateway typically relays TCP segments from one connection to the other without examining the contents. The security function consists of determining which connections will be allowed.

A typical use of circuit-level gateways is a situation in which the system administrator trusts the internal users. The gateway can be configured to support application-level or proxy service on inbound connections and circuit-level functions for outbound connections. In this configuration, the gateway can incur the processing overhead of examining incoming application data for forbidden functions but does not incur that overhead on outgoing data.

Firewall placement and demilitarized zone (DMZ)



An **external firewall** is placed at the edge of a local or enterprise network, just inside the boundary router that connects to the Internet or some wide area network (WAN). One or more **internal firewalls** protect the bulk of the enterprise network. Between these two types of firewalls are one or more networked devices in a region referred to as a **DMZ** (demilitarized zone) network. Systems that are externally accessible but need some protections are usually located on DMZ networks. Typically, the systems in the DMZ require or foster external connectivity, such as a corporate Web site, an e-mail server, or a DNS (domain name system) server.

The external firewall provides a measure of access control and protection for the DMZ systems consistent with their need for external connectivity. The external firewall also provides a basic level of protection for the remainder of the enterprise network. In this type of configuration, internal firewalls serve three purposes:

1. The internal firewall adds more stringent filtering capability, compared to the external firewall, in order to protect enterprise servers and workstations from external attack.
2. The internal firewall provides two-way protection with respect to the DMZ. First, the internal firewall protects the remainder of the network from attacks launched from DMZ systems. Such attacks might originate from worms, root-kits, bots, or other malware lodged in a DMZ system. Second, an internal firewall can protect the DMZ systems from attack from the internal protected network.
3. Multiple internal firewalls can be used to protect portions of the internal network from each other. For example, firewalls can be configured so that internal servers are protected from internal workstations and vice versa. A common practice is to place the DMZ on a different network interface on the external firewall from that used to access the internal networks.

Distributed Denial of Service (DDoS) attacks

Consumes the target's resources so that it cannot provide service

- Denial of Service (DoS): originates from a single user
- Distributed DoS (DDoS): originates from multiple users, usually a botnet

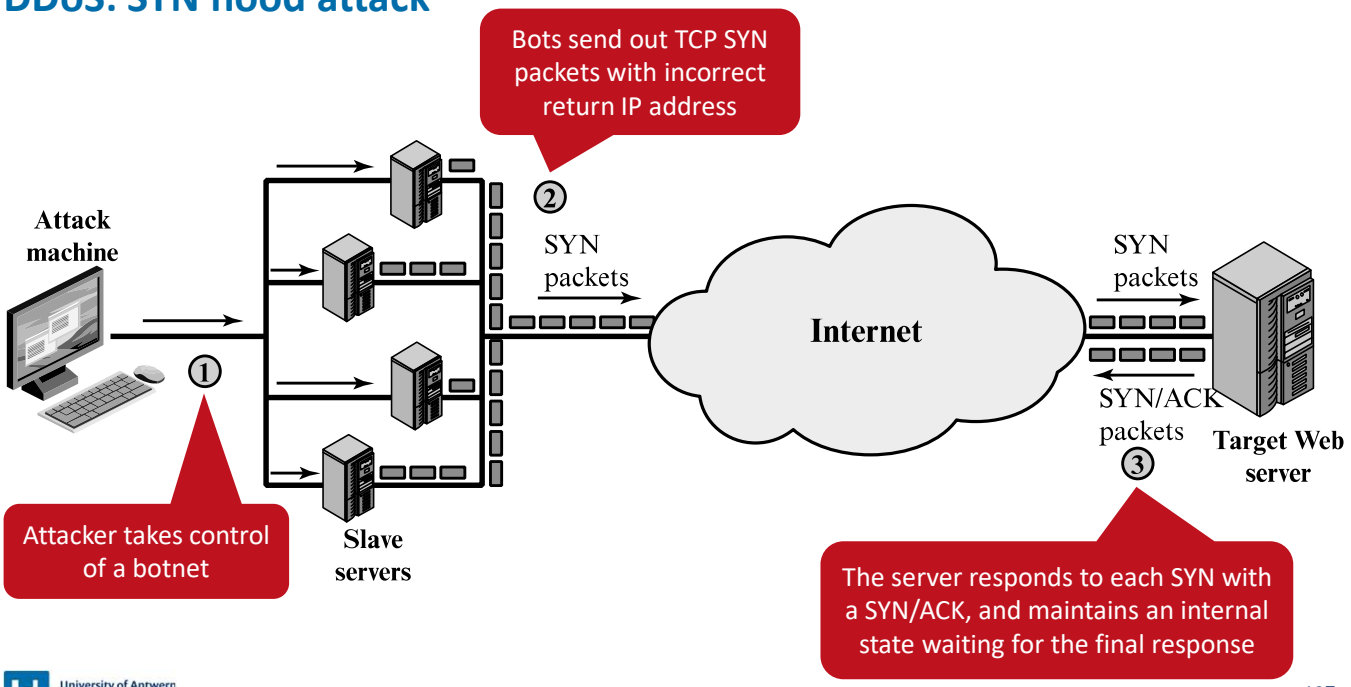
Types of resources targeted

- Internal host resources
- Data transmission resources

A denial-of-service (DoS) attack is an attempt to prevent legitimate users of a service from using that service. When this attack comes from a single host or network node, then it is simply referred to as a DoS attack. A more serious threat is posed by a DDoS attack. DDoS attacks make computer systems inaccessible by flooding servers, networks, or even end-user systems with useless traffic so that legitimate users can no longer gain access to those resources. In a typical DDoS attack, a large number of compromised hosts are amassed to send useless packets.

A DDoS attack attempts to consume the target's resources so that it cannot provide service. One way to classify DDoS attacks is in terms of the type of resource that is consumed. Broadly speaking, the resource consumed is either an internal host resource on the target system or data transmission capacity in the local network to which the target is attacked.

DDoS: SYN flood attack

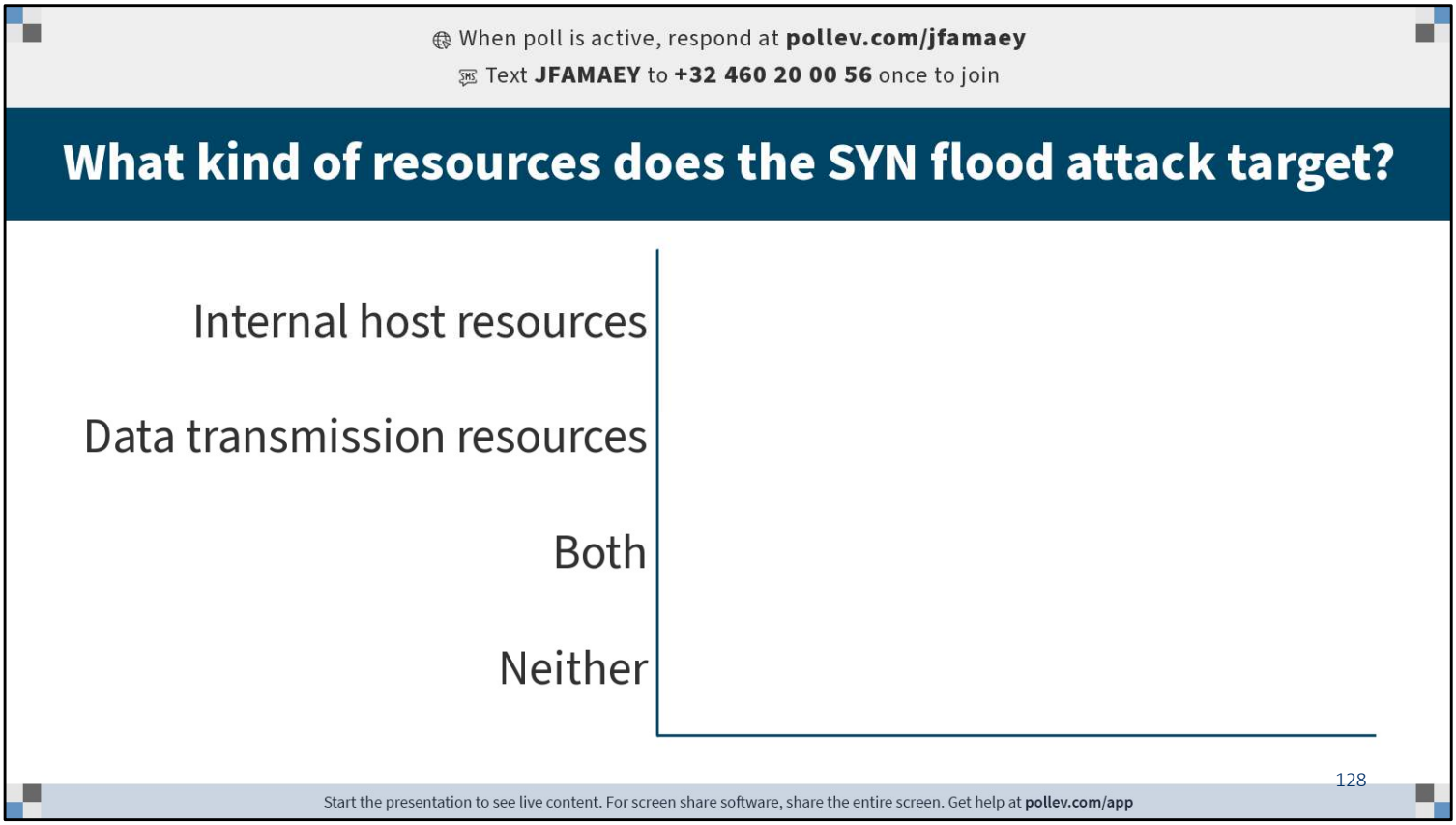


University of Antwerp | Faculty of Science

127

A simple example of an **internal resource attack** is the SYN flood attack. The following steps are involved:

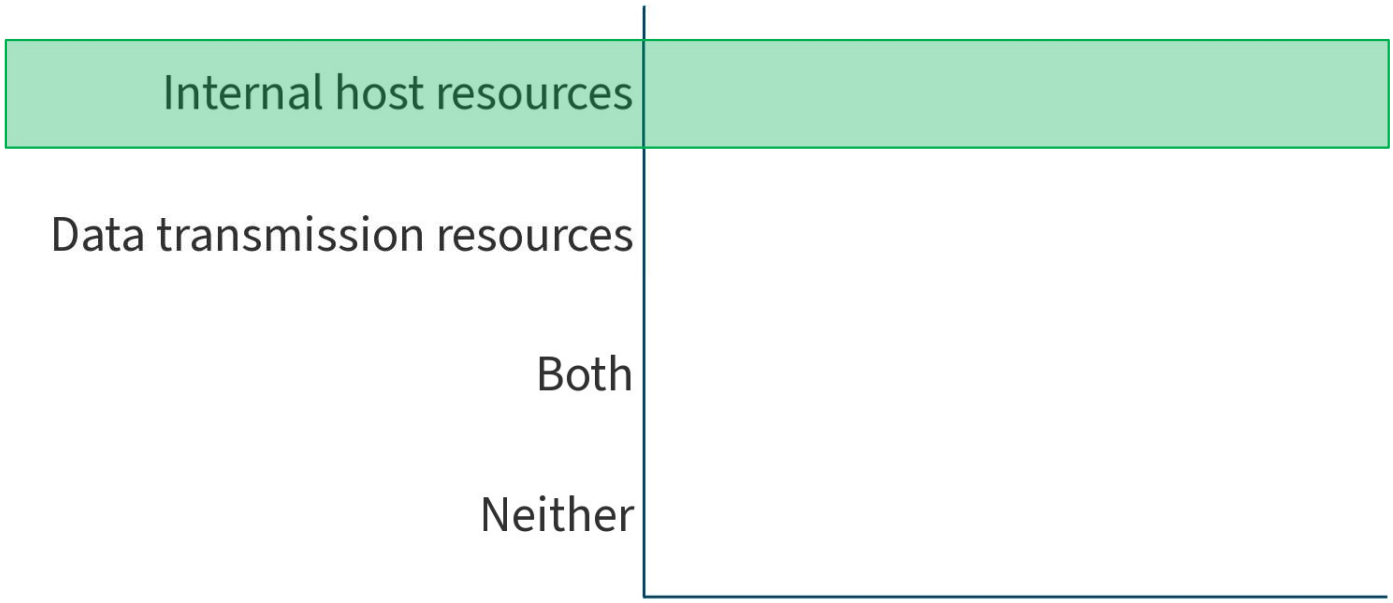
1. The attacker takes control of multiple hosts over the Internet, instructing them to contact the target Web server.
2. The slave hosts begin sending TCP/IP SYN (synchronize/initialization) packets, with erroneous return IP address information, to the target.
3. Each SYN packet is a request to open a TCP connection. For each such packet, the Web server responds with a SYN/ACK (synchronize/acknowledge) packet, trying to establish a TCP connection with a TCP entity at a spurious IP address. The Web server maintains a data structure for each SYN request waiting for a response back and becomes bogged down as more traffic floods in. The result is that legitimate connections are denied while the victim machine is waiting to complete bogus “half-open” connections.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What kind of resources does the SYN flood attack target?
https://www.polleverywhere.com/multiple_choice_polls/DjcXq5PWg6j62XT?state=opened&flow=Default&onscreen=persist

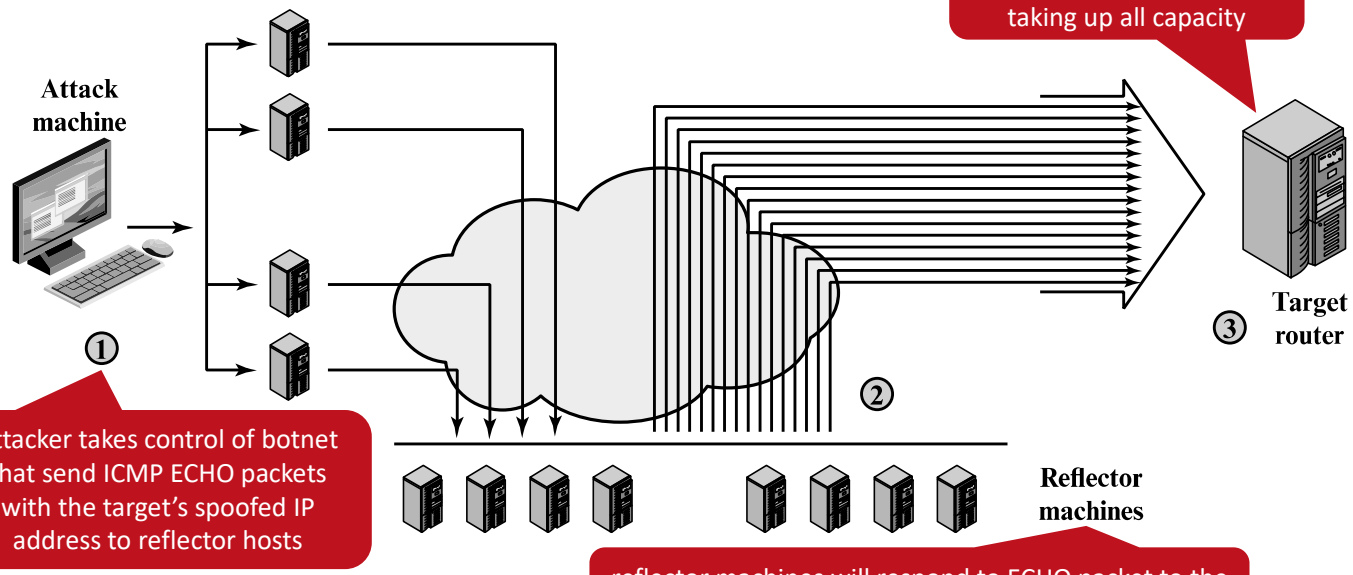
What kind of resources does the SYN flood attack target?



129

Poll Title: What kind of resources does the SYN flood attack target?

DDoS: ICMP attack

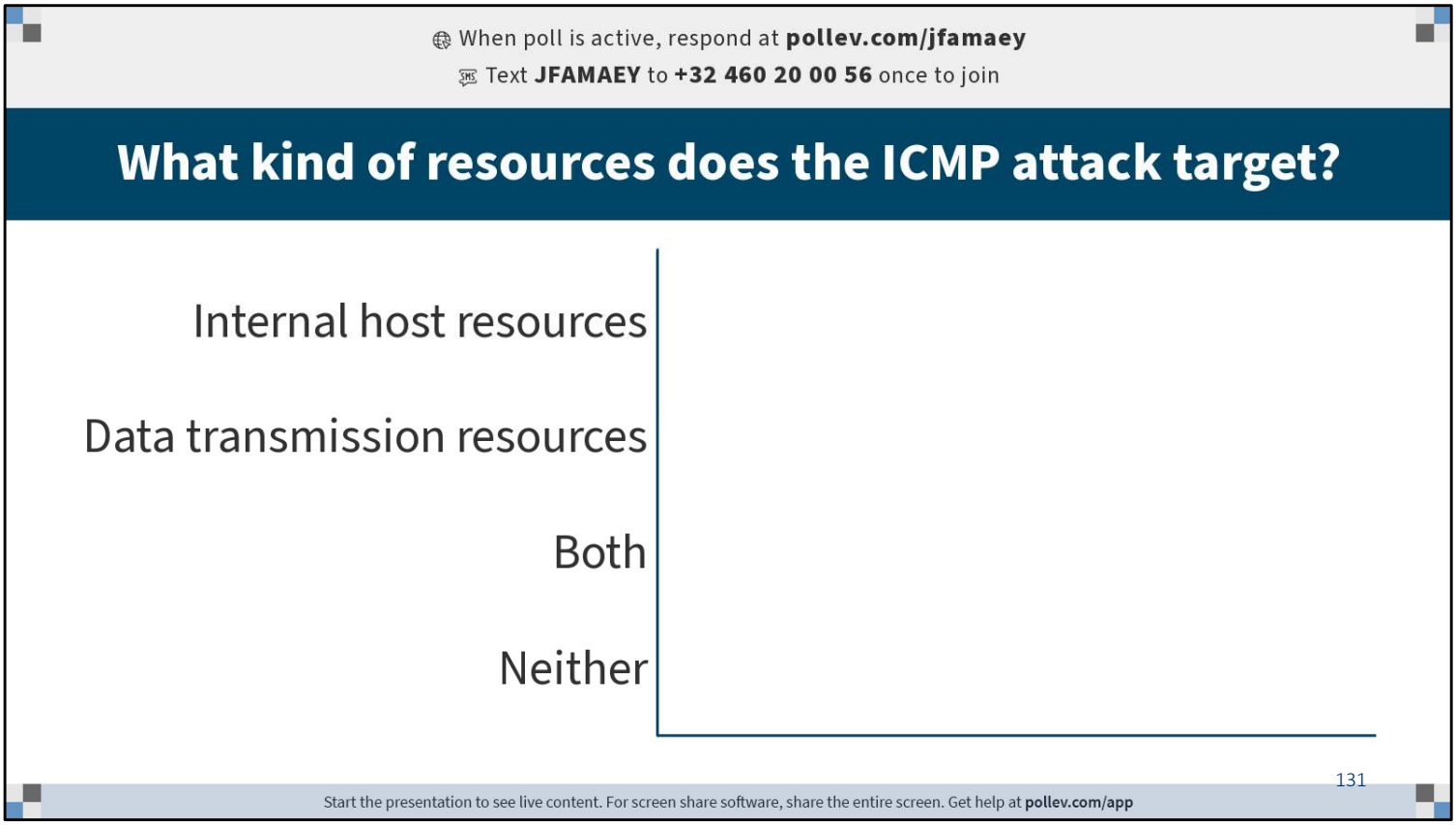


University of Antwerp | Faculty of Science

130

The ICMP attack is an example of an **attack that consumes data transmission resources**. The following steps are involved:

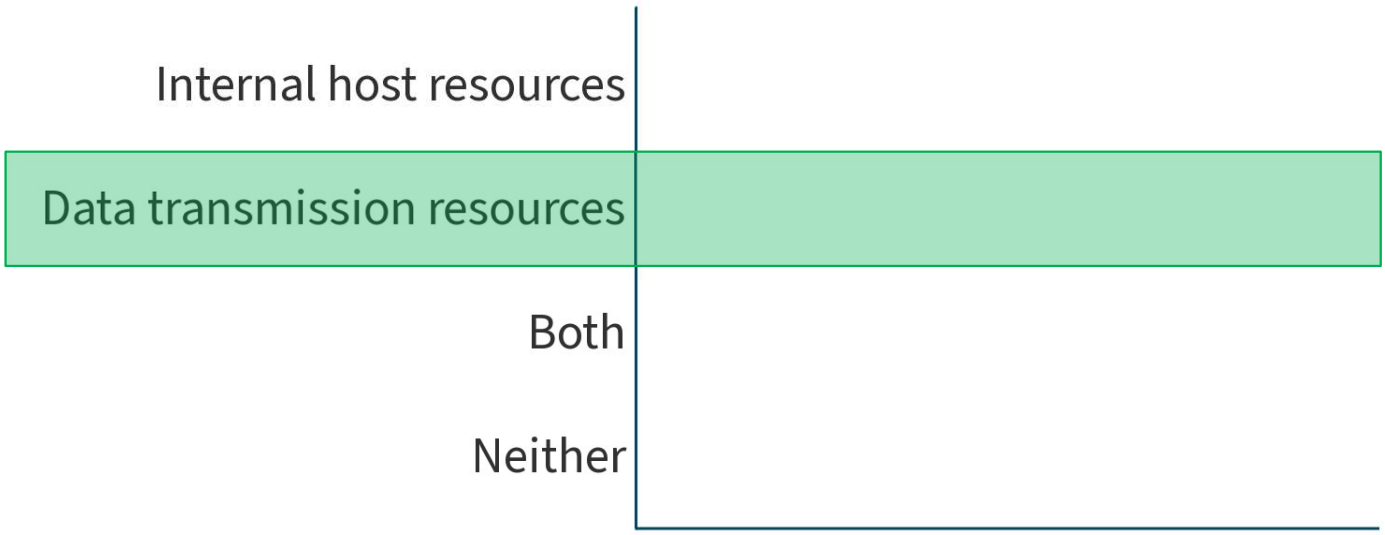
1. The attacker takes control of multiple hosts over the Internet, instructing them to send ICMP ECHO packets with the target's spoofed IP address to a group of hosts that act as reflectors, as described subsequently.
2. Nodes at the bounce site receive multiple spoofed requests and respond by sending echo reply packets to the target site.
3. The target's router is flooded with packets from the bounce site, leaving no data transmission capacity for legitimate traffic.



Poll Title: Do not modify the notes in this section to avoid tampering with the Poll Everywhere activity.
More info at polleverywhere.com/support

What kind of resources does the ICMP attack target?
https://www.polleverywhere.com/multiple_choice_polls/BpCi1aHwtQOvE0R?state=opened&flow=Default&onscreen=persist

What kind of resources does the ICMP attack target?



132

Poll Title: What kind of resources does the ICMP attack target?

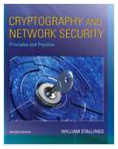
Summary on firewalls

Firewalls

- Four types: packet filter, stateful inspection, application level and circuit filter
- Stateful firewall avoids having to allow all incoming TCP traffic to ports > 1023
- Application-level firewall performs deep packet inspection (most flexible, but resource intensive)
- Circuit filter firewall performs deep packet inspection only when a new flow is established

Distributed Denial of Service Attacks

- SYN flood attack overloads the internal system resources of a target server
- ICMP attack overloads the data transmission resources of a target router



Link with the book

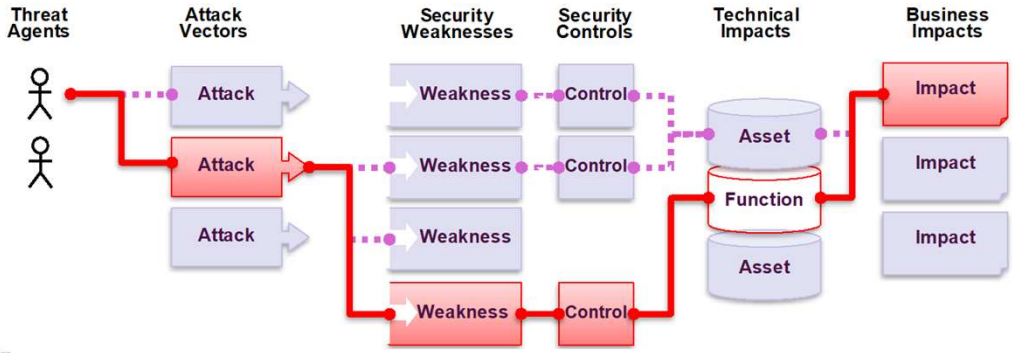
- Chapter 21 (Section 21.11)
- Chapter 23

1. Email
2. Firewalls
3. Web application security

University of Antwerp
Faculty of Science

Web application security

- Firewalls consider web traffic “harmless” and let it pass
- Secure network protocols (e.g., HTTPS) prevent eavesdropping but do not protect from vulnerabilities in applications themselves
- The threat agent finds an attack that exploits a weakness, providing them with control over functions or assets in the application



Most important web application security risks (1/2)

1. **Broken Access Control**
   Failure to properly control access of authenticated users to resources, leading to unauthorized data disclosure/modification/destruction or users performing unintended actions.
2. **Cryptographic failures**
   Failure to protect the confidentiality of data in transit or rest, leading to unintended exposure of confidential/sensitive data.
3. **Injection**
   Failure to validate user-supplied data, leading to unintended/unauthorized execution of code as part of valid queries (e.g., SQL injection, cross-site scripting).
4. **Insecure design**
   Weaknesses in the inherit control design of the application, complementing other risks, which are related to insecure implementation.
5. **Security misconfiguration**
   A common issue, often resulting from leaving default configurations, improperly configured permissions, verbose error messages containing sensitive information, etc.

Source: *https://owasp.org/Top10/ (updated in 2021)*

1. **Access control** enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.
2. The first thing is to determine the **protection needs of data** in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS).
3. Some of the more common **injections** are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections. Automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs is strongly encouraged. Organizations can include static (SAST), dynamic (DAST), and interactive (IAST) application security testing tools into the CI/CD pipeline to identify introduced injection flaws before production deployment.
4. **Insecure design** is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

5. **Security misconfiguration** can take many forms, such as:
    - Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
    - Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
    - Default accounts and their passwords are still enabled and unchanged.
    - Error handling reveals stack traces or other overly informative error messages to users.
    - For upgraded systems, the latest security features are disabled or not configured securely.
    - The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
    - The server does not send security headers or directives, or they are not set to secure values.
    - The software is out of date or vulnerable

Most important web application security risks (2/2)

6. **Vulnerable and outdated components**
   Old versions of libraries with known vulnerabilities that are not updated provide easy access to attackers and may result in data loss or takeovers.
7. **Identification and authentication failures**
   Incorrect implementation of authentication and session management functions allow attackers to compromise passwords or assume users' identities.
8. **Software and data integrity failures**
   Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations (e.g., in external plugins or auto-update functions)
9. **Security logging and monitoring failures**
   Makes it much harder to detect and understand attacks, for future prevention, which may lead to longer detection times or exposure.
10. **Server-side request forgery**
    Occurs when fetching a remote resource without validating the user-supplied URL, which allows sending a crafted request to an unexpected destination.

Source: *https://owasp.org/Top10/*

University of Antwerp | Faculty of Science

137

6. **Vulnerable and outdated components** can contain a variety of security bugs. Your software may be vulnerable:
   - If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies. If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
   - If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
   - If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
   - If software developers do not test the compatibility of updated, upgraded, or patched libraries.
7. Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be **authentication weaknesses** if the application permits brute-force or automated password guessing attacks, weak passwords, unsafe password-recovery, weakly hashed passwords, or exposes/mishandles session identifiers.
8. **Software and data integrity failures** relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and

applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

9. **Security logging and monitoring** is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Logging, detection, monitoring, and active response include auditable events, warnings/errors, application logs, penetration testing, and active attack logs.
10. **Server-side request forgery** (SSRF) flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL). As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

SQL injection

- Untrusted data sent to interpreter as part of a valid query
- Normal operations:


```
POST /login?u='jfamaey'&p='password'
```

```
SELECT * FROM Users WHERE Name = 'jfamaey' AND Pass = 'password'
```

Injecting malicious code in SQL queries

- This query will return all entries in the "Users" database table
- Can be used to retrieve a list of username/password pairs or even to log in without knowing a valid username/password combination


```
SELECT * FROM Users WHERE Name = 'jfamaey' or '1=1'
        AND Pass = 'password' or '1=1'
```

SQL injection based on Batched SQL statements

- Most databases support batched SQL statements, grouping multiple statements separated by semicolons


```
SELECT * FROM Users WHERE Name = ''; DROP TABLE Users; SELECT *
FROM Users WHERE Name = '' AND Pass = 'password'
```

SQL injection example

https://www.hacksplaining.com/exercises/sql-injection

SQL injection prevention mechanisms

- Parameterize your SQL statement
- Encode and validate input
- Prevent untrusted input from being interpreted as part of query

Parameterization example:

```
txtNam = getRequestString("CustomerName");
txtAdd = getRequestString("Address");
txtCit = getRequestString("City");
txtSQL = "INSERT INTO Customers
            (CustomerName,Address,City) Values(@0,@1,@2)";
db.Execute(txtSQL,txtNam,txtAdd,txtCit);
```

SQL engine checks parameters and ensures that they are correct, as well as treated literally

XML External Entity (XXE)

- Poorly configured XML parsers are vulnerable
- Exploits the parser's desire to blindly process external DTD entities
- Allows various attacks, such as accessing local files, server-side request forgery (SSRF), remote file includes and XML DoS attacks

Prevention:

- Easiest way is to completely disable external entities

Example (accessing local files):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY>
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >
]>
```

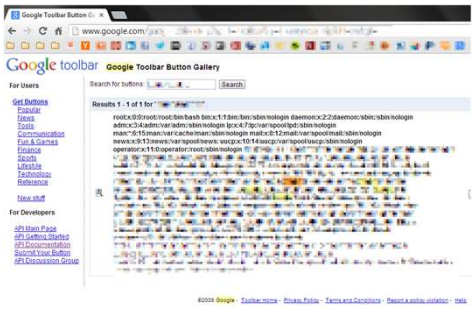

An *XML External Entity* attack is a type of attack against an application that parses XML input. This attack occurs when **XML input containing a reference to an external entity is processed by a weakly configured XML parser**. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

The XML 1.0 standard defines the structure of an XML document. The standard defines a concept called an entity, which is a storage unit of some type. There are a few different types of entities, external general/parameter parsed entity often shortened to external entity, that can access local or remote content via a declared system identifier. The system identifier is assumed to be a URI that can be dereferenced (accessed) by the XML processor when processing the entity. The XML processor then replaces occurrences of the named external entity with the contents dereferenced by the system identifier. If the system identifier contains tainted data and the XML processor dereferences this tainted data, the XML processor may disclose confidential information normally not accessible by the application. Similar attack vectors apply the usage of external DTDs, external stylesheets, external schemas, etc. which, when included, allow similar external resource inclusion style attacks.

Source: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

XXE example: Google Toolbar button gallery

- Google service that allows developers to create toolbar buttons by uploading XML files containing meta data (e.g., styling information)
- When searching for buttons, the title and description of a button are printed out to the browser by the web application
- Could be used to create a “button” XML file that prints out local server files (e.g., /etc/hosts, /etc/passwd) when searching for them



Source: *https://blog.detectify.com/2014/04/11/how-we-got-read-access-on-googles-production-servers/*

Cross-Site Scripting (XSS)

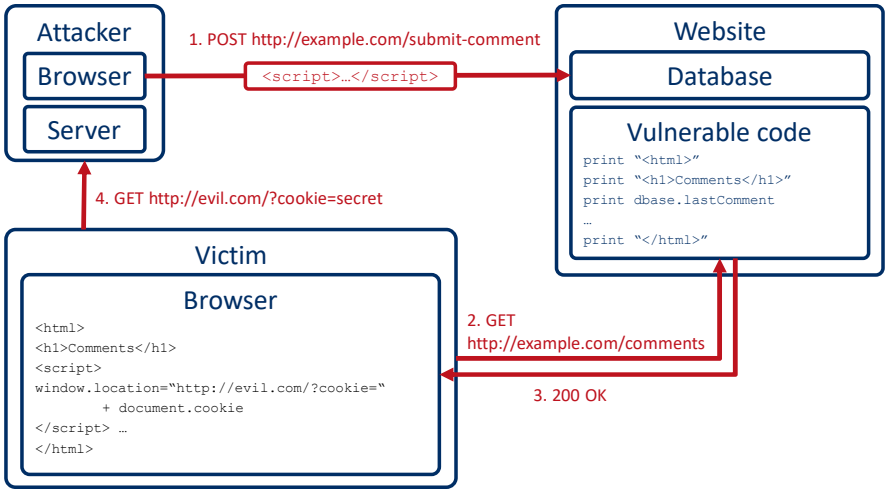
- Attacker injects malicious scripts in otherwise trusted web pages
- Can occur in any web application that uses input from a user within the output it generates without validating or encoding it

Two types of XSS attacks

- Stored XSS attacks: The injected script is permanently stored on the target servers (e.g., in a database, message forum, comment field)
- Reflected XSS attacks: The injected script is reflected off the server (e.g., in an error message, search result) and is delivered to the victim using some other way (e.g., in an email message or via another website)

Stored XSS attack example

An attacker tries to steal a victim's identity by obtaining their session cookie



University of Antwerp
Faculty of Science

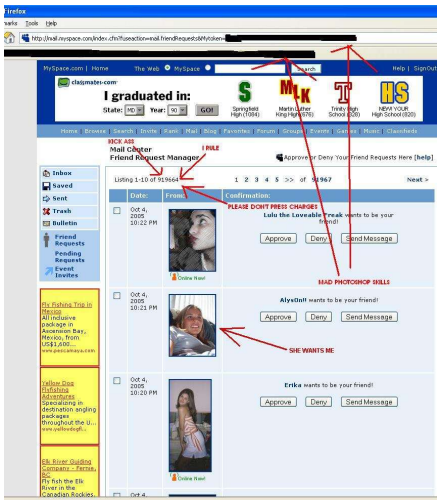
Source: *https://www.acunetix.com/websitesecurity/cross-site-scripting/*

Stored XSS case study: MySpace Samy worm

Source: *https://www.youtube.com/watch?v=DtnuaHl378M*

Stored XSS attack case study: MySpace Samy Worm

Full technical explanation available: https://samy.pl/myspace/tech.html



Stored XSS example

https://www.hacksplaining.com/exercises/xss-stored

Reflected XSS attack example

- Assume code for an error page that handles requests for non-existing pages (i.e., the classic 404 error page)

Error page code

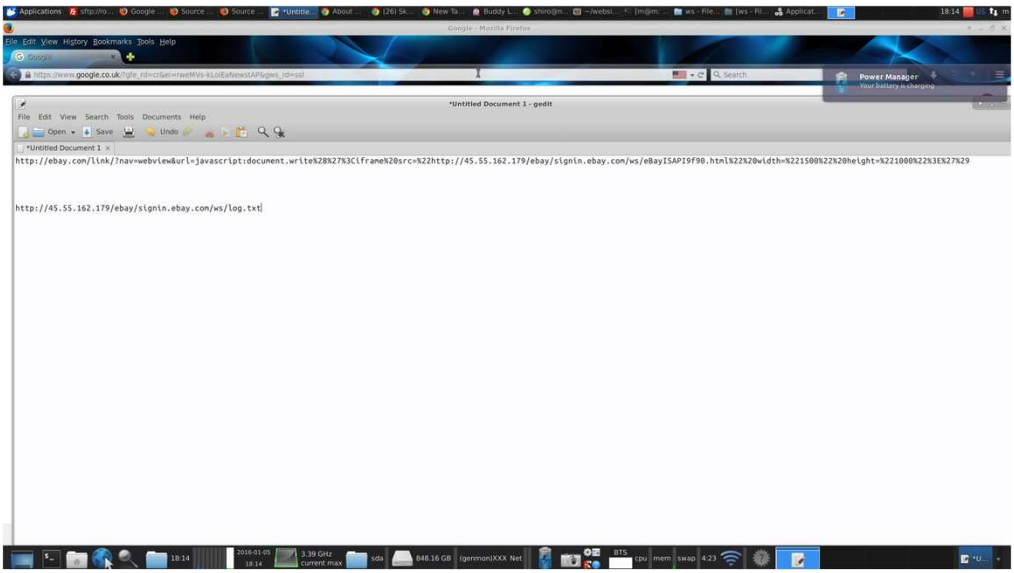
```
<html> <body>
  <? php print "Not found: " . urldecode($_SERVER["REQUEST_URI"]); ?>
</body> </html>
```

- The url "http://testsite.test/file_which_not_exist" will generate the output:
  > Not found: /file_which_not_exist

- An attacker may send a URL via email that executes JavaScript on the victim's local system:
  http://testsite.test/<script>alert("TEST");</script>

Source: *https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)*

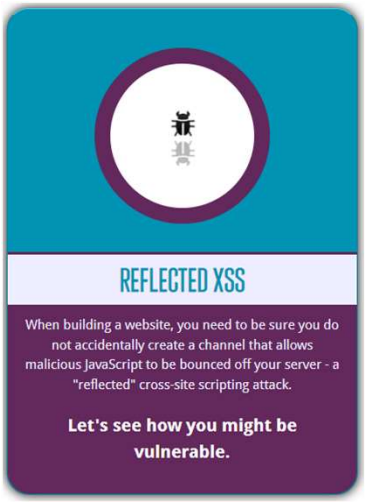
Reflected XSS attack example



University of Antwerp
Faculty of Science

Source: *https://www.youtube.com/watch?v=WuZ61NWbK_4*

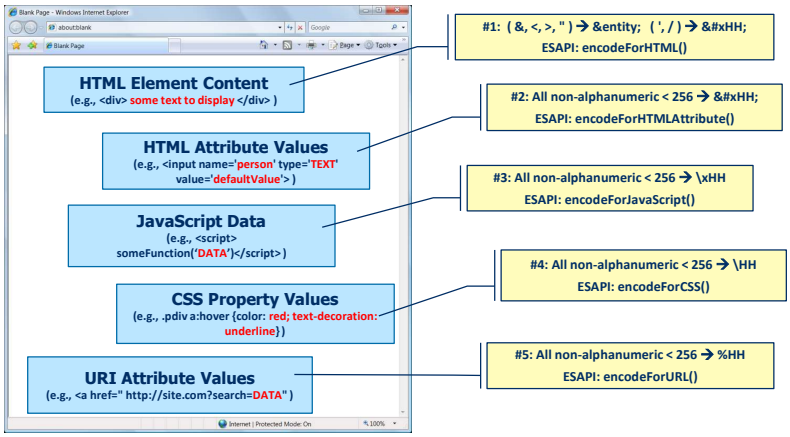
Reflected XSS example



https://www.hacksplaining.com/exercises/xss-reflected

XSS attack prevention

- Output encoding
- Input validation
- Content security policy
- Use security frameworks (e.g., OWASP ESAPI, AntiSamy)



Web application security summary

- **Vulnerabilities in web applications** may be exploited by attackers to take control over assets or functionality
- **OWASP Top 10** web application vulnerabilities can be used by developers to reduce the chance of exploitable vulnerabilities
- Common web application security issues to watch out for
  - SQL injection: Allows attackers to access or change data stored in database
  - XML external entities (XXE): Allows attackers to trick XML parser to parse external files
  - Cross-Site Scripting (XSS): Allows attacker to execute malicious scripts in victim's browser

End of Part 6

University of Antwerp
Faculty of Science