# Solutions to Selected Problems

Guide to Internet Cryptography

Companion Material

February 9, 2026

## Preface

This document provides solutions to selected problems from the book *Guide to Internet Cryptography: Security Protocols and Real-World Attack Implications*. The material is intended for educational use in courses and self-study.

**Book website:** https://link.springer.com/book/10.1007/978-3-031-19439-9

## 1 Chapter 10: Transport Layer Security

**Problem 10.1 Activation of TLS**

(a) You are the administrator of your company's web server shop.bestproduct.com. For maximal availability, product information can be accessed via HTTPS and HTTP. However, the customer login page at shop.bestproduct.com/login should only be accessible through TLS to protect your customer's passwords against eavesdropping. What should your server do if a web browser tries to access the URL http://shop.bestproduct.com/login?
(b) Your boss tells you that his emails contain confidential information, so his email client should use TLS encryption to send and receive emails. Suppose your company's email server supports TLS with well-known ports. Which ports do you have to configure for TLS to be used? Are your company's trade secrets protected once TLS is activated?

**Solution**

(a) It should send an 30x HTTP redirection error code, redirecting the browser to the different https://shop.bestproduct.com/login - the 's' in 'https' now forces the browser to perform a TLS Handshake.
(b) The email client of your boss should be cofigured to retrieve emails from port 993 for IMAP-over-TLS, and to send emails using port 465 for SMTP-over-TLS. Still, company secrets are not fully protected by using TLS, as this configuration only affects the last TCP hop from the email servers *in* your company to the email client. If the email isn't protected by S/MIME or OpenPGP encryption, the content of the email can still be read during transmission on non-TLS links, and during storage at SMTP servers.

**Problem 10.2 TLS Record Layer**

(a) Suppose your browser will send an HTTP request of length 500 bytes, and this request is sent in a single TLS record. No compression is applied to the plaintext, AES-256 in CBC mode is used for encryption, and HMAC-SHA1 for the MAC computation. How

many and which padding bytes must be added?
(b) The server `moneytransfer.com` accepts HTTP requests for money transfers but validates them through TLS before they are executed. After receiving the request, a mutually authenticated TLS connection is established, and the server sends the transfer request '`$ 1000 to attacker@evil.xyz`' in a single TLS record to the client. To confirm the transfer, the client has to send back the same text '`$ 1000 to attacker@evil.xyz`' in a single TLS record. Pentester Paul now claims that he has found a vulnerability: A man-in-the-middle attacker could simply send an HTTP request, intercept the encrypted TLS record from the server and send this record back to the server. Since no changes are made to the TLS record, the MAC will be correct, and the server will accept this record as confirmation. Is Paul right?
(c) A TLS client sends three TLS records $c_1, c_2, c_3$ to a server. A man-in-the-middle attacker deletes $c_2$ and updates all TCP sequence numbers such that the deletion is not noticed at the TCP level. Will the attack be detected? Please explain.

**Solution**

(a) In TLS with CBC mode, the plaintext is structured as

$$\text{plaintext} \;=\; \text{data} \,\|\, \text{MAC} \,\|\, \text{padding} \,\|\, \text{padding\_length}.$$

We are given:

- HTTP request length: 500 bytes
- MAC: HMAC-SHA1 $\Rightarrow$ 20 bytes
- Block cipher: AES $\Rightarrow$ block size 16 bytes

**Step 1: Length before padding**

$$500 + 20 = 520 \text{ bytes}.$$

**Step 2: Padding requirements** In TLS, the total length (including the final padding_length byte) must be a multiple of the block size. Therefore, we need to find the smallest $p \geq 1$ such that

$$520 + p \equiv 0 \pmod{16}.$$

We compute:

$$520 \bmod 16 = 8.$$

Thus,

$$p = 16 - 8 = 8.$$

**Step 3: Padding bytes** TLS padding consists of:

- $p - 1 = 7$ padding bytes, and
- 1 padding_length byte.

All padding bytes (including the padding_length byte) have the same value, namely

$$p - 1 = 7 = \texttt{0x07}.$$

**Final Answer**

8 padding bytes are added: `0x07 0x07 0x07 0x07 0x07 0x07 0x07 0x07`

This makes the total plaintext length

$$520 + 8 = 528 \text{ bytes},$$

which is a multiple of the AES block size.

(b) No, Paul is not right.
Although TLS uses symmetric cryptography, it establishes *separate keys and state for each direction* of the connection. In particular, after the TLS handshake:

- one set of keys is used for records sent from server → client,
- a different set of keys is used for records sent from client → server.

Furthermore, each TLS record is protected by a MAC that is computed over

- the plaintext,
- the record type,
- the TLS version,
- the record length,
- and an implicit *sequence number* that is maintained separately for each direction.

The encrypted record sent by the server was:

- encrypted with the *server-to-client* encryption key, and
- authenticated with the *server-to-client* MAC key and sequence number.

If a man-in-the-middle attacker replays this record back to the server:

- the server will attempt to verify it using the *client-to-server* keys,
- and the expected client-side sequence number.

As a result, MAC verification will fail, even though the ciphertext itself was not modified.

**Conclusion.** TLS provides implicit protection against reflection and replay attacks by using direction-specific keys and sequence numbers. A TLS record sent by the server cannot be replayed as a valid client record. Therefore, the attack described by Paul does not work.

(c) Yes, the attack will be detected.
In TLS, each record is protected by a MAC (or AEAD tag) that includes an *implicit record sequence number*. This sequence number:

- starts at 0 after the handshake,
- is incremented by 1 for each TLS record,
- is maintained independently of TCP sequence numbers.

Let the client send records $c_1, c_2, c_3$ with TLS sequence numbers

$$0,\ 1,\ 2.$$

If the attacker deletes $c_2$:

- the server receives $c_1$ (expected sequence number 0) and accepts it,
- the next record it receives is $c_3$, which was created with sequence number 2.

However, the server will verify $c_3$ using sequence number 1, since it expects the next TLS record in order. Because the sequence number is part of the MAC input, the computed MAC will not match the received one.

**Conclusion.** Even though TCP-level manipulation hides the deletion from TCP, TLS detects the attack at the record layer. The missing record causes a sequence number mismatch, leading to MAC verification failure. Therefore, the deletion attack is detected.

### Problem 10.3 TLS Handshake 1 (section 10.3)

(a) Why must cryptographic algorithms be negotiated in TLS? E.g., in instant messaging protocols, clients simply *know* which algorithms to use.

(b) The mental model for the TLS-RSA handshake depicted with the letterbox in Figure 10.5 is incomplete. Please answer the following question, and discuss what could be added to the letterbox: Why can't a MITM attacker simply send her/his letterbox to the client?

(c) Why do we need the two Finished messages? The key exchange is already completed with ClientKeyExchange!

### Solution

(a) Cryptographic algorithms must be negotiated in TLS because TLS is designed to be a general-purpose, long-lived, and interoperable security protocol used by many different clients and servers across the Internet.
First, different parties may support different sets of cryptographic algorithms. A client and a server developed at different times, or with different security capabilities (e.g., hardware acceleration, legal restrictions, or legacy systems), cannot assume a single common algorithm. Negotiation ensures that they can agree on an algorithm that both support.
Second, cryptographic algorithms age and become insecure over time. Algorithm negotiation allows TLS to deprecate weak algorithms (such as RC4 or SHA-1) and introduce stronger ones without redesigning the protocol or breaking compatibility with older implementations.
Third, TLS must support multiple security properties and performance trade-offs. Different use cases may prefer different algorithms (e.g., faster vs. more secure, or RSA vs. ECDSA certificates). Negotiation enables flexibility while still maintaining security.
In contrast, instant messaging protocols often operate in a closed ecosystem where clients are tightly controlled and updated simultaneously. In such settings, it is feasible to predefine a fixed set of algorithms. TLS, however, operates in an open and heterogeneous environment, making algorithm negotiation essential.

(b) The letterbox represents the X.509 certificate of the server. The difference between TLS certificates and the depicted letterbox is that the certificates are bound to specific domains. This binding is more or less strict (less strict, e.g., for wildcard certificates containing `*.example.com`, covering all subdomains), but this binding to domain names prohobits the exchange of certificates. In the mental model, this could be represented by some kind of certification written on the letterbox. E.g., "This letterbox belongs to `example.com`, and this is certified by our trusted organization".

(c) The `Finished` messages in the TLS 1.2 handshake are essential even though the key exchange is completed with `ClientKeyExchange`, because they provide *authentication of the entire handshake* and *key confirmation*.

**Handshake integrity.** The `Finished` message contains a MAC (the *verify_data*) computed over *all previous handshake messages* using keys derived from the negotiated master secret. This ensures that:

- both parties have the same view of the handshake transcript, and
- no attacker has modified, inserted, or removed any handshake message (including algorithm negotiation and certificates).

Without the `Finished` messages, a man-in-the-middle could tamper with the handshake while still allowing the key exchange to complete.

**Key confirmation.** Sending a `Finished` message proves that the sender:

- derived the same master secret, and
- possesses the correct session keys.

Thus, the `Finished` messages confirm that both sides are actually sharing the same cryptographic keys.

**Why two `Finished` messages?** Each party must verify that the *other* party derived the same keys and saw the same handshake transcript. Therefore:

- the client sends a `Finished` to authenticate itself to the server,
- the server sends a `Finished` to authenticate itself to the client.

**Conclusion.** Although `ClientKeyExchange` establishes the key material, the two `Finished` messages are necessary to ensure handshake integrity, mutual key confirmation, and protection against active attacks.

### Problem 10.4 TLS ciphersuites

Specify for each of the following ciphersuites: (a) whether the Perfect Forward Secrecy security property is achieved when using this ciphersuite; and (b) which handshake messages contribute to the computation of the PremasterSecret.

**Solution**

| Ciphersuite | (a) | (b) |
|---|---|---|
| TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 | YES | ServerKeyExchange, ClientKeyExchange |
| TLS_DHE_ECDSA_WITH_AES_128_GCM_SHA256 | YES | ServerKeyExchange, ClientKeyExchange |
| TLS_RSA_WITH_THE_EDE_CBC_SHA | NO | ClientKeyExchange |
| TLS_DH_RSA_WITH_NULL_SHA256 | NO | (Server) Certificate, ClientKeyExchange |
| TLS_ECDH_DSS_WITH_3DES_EDE_CBC_SHA | NO | (Server) Certificate, ClientKeyExchange |

**Problem 10.5 TLS Handshake 2 (section 10.5)**

A few ideas are circulating on social media platforms for potential man-in-the-middle attacks on the TLS handshake. Evaluate the statements for their correctness in each case. Explain why or under which (realistic) conditions the attacks work or do not work and, if applicable, at which stage the attacker's manipulations would be noticed in the handshake.

(a) To be able to read the application data exchanged between client and server in a TLS connection where only the server authenticates itself, the man-in-the-middle simply forwards the ClientHello message but then replaces the ClientKeyExchange message with his own. Then he knows the PremasterSecret and can calculate the Finished messages correctly.

(b) A secret service has retrieved the private key of a webmail TLS server and now wants to read the plaintext emails exchanged between client and server. The user's web browser only allows ciphersuites with Perfect Forward Secrecy. There are two ways to authenticate to the webmail service as a user: by password or by TLS client authentication. Which of these possibilities is safe in the given scenario, and why? If the intelligence service only wants to impersonate the server, can it do so in either case?

(c) An attacker forces the client to choose a TLS-RSA cipher suite by manipulating the ServerHello message. He has discovered a Bleichenbacher oracle in the server's implementation and uses it to impersonate the server to the client.

(d) The attacker manipulates the traffic in such a way that the client's request to the IANA server (where the 2-byte values of the ciphersuite encoding are converted to the long form) for the ciphersuite selected by the server is answered by a forged response with a weak ciphersuite (e.g., with only 40 bits of key length in the record layer).

**Solution**

(a) This works, but cannot be extended to a man-in-the-middle attack: With server-only authentication, TLS clients remain anonymous. So naturally the attacker can replace the (anonymous) client with himself.
What is missing is a TLS cnnection from the attacker (in the role of the server) to the original client, successfully impersonating the target server. But the server authenticates itself with a trusted certificate and the ability to perform private-key operations (decryption, DHKE, signing) related to the public key in the certificate. In reference [46],

a formal proof is given that this is impossible.

(b) Only knowing the private key of a server is different from obtaining full control over this server. So we assume that the secret service sets up a new server *impersonating* the original webmail server, and redirecting all traffic from the target user to this fake server. If the target user only connects to this fake server, he will immediately notice that his mailbox is missing on the server. If he ignores this fact, he can only send messages, but not receive any. Since the client browser is configured to only use ciphersuites with PFS, all emails exchange in TLS sessions which where established prior to the leakage of the private key remain confidential.
If the secret service also wants to access the emails received by the target user, the fake server must act as a man-in-the-middle on the application layer. In the context of webmail this means that the fake server acts as a HTTP proxy. To be able to do so, the fake server must also impersonate the target client to the original server.
If the client authenticates via password, the first authetnication to the fake server will reveal this password, and the impersonification succeeds. If, on the other hand, the client uses TLS client authentication (or Passkeys as a newer alternative), then the fake server will not learn the private key of the client, and will therefore be unable to impersonate the client.
As for the intelligence service to be able to impersonate the server: This is possible, and has been described above.

(c) This attack will not succeed because the hash value of all exchanged handshake messages is used in the computation and verification of the MACs contained in the two Finished messages. If an attacker removes the non-RSA ciphersuites from the list of ciphersuites in the ClientHello message, the transcript changes, and client and server will compute different hash values to create and verify a given MAC. As a consequence, the TLS server will abort the handshake because it cannot verify the MAC contained in the ClientFinished message.

(d) This question contains complete nonsense. The meaning of the two-byte encodings of ciphersuites is hardcoded to both TLS client and server, and there is no request to any server. If there is no request, then the non-existent answer to this request cannot be forged.

**Problem 10.6 TLS Handshake 3 (Figure 10.14)**

In Figure 10.14, several basic authentication and key exchange protocols are combined. Identify at least four challenge-and-response protocols and two key exchange protocols.

**Solution**

The four challenge-and-response protocols (unilateral) are:

- Challenge $r_C$ sent in CH, response (MAC) sent in $FIN_S$.
- Challenge $r_S$ sent in SH, response (MAC) sent in $FIN_C$.
- Challenge $r_C$ sent in CH, response (signature) sent in SKE.
- Challenge $r_S$ sent in SH, response (signature) sent in CV (optional).

The two key exchange protocols are:

- DHKE in SKE and CKE.
- Symmetric key exchange with nonces $r_C, r_S$ contained in CH and SH, and $ms$ used as the shared symmetric key.

**Problem 10.7 TLS Session Resumption**

Why is the order of ClientFinished and ServerFinished reversed in TLS session resumption?

**Solution**

Because no DHKE is required, and the client can derive the keying material already after receiving the SH message where the server acknowledges its posession of the references MasterSecret $ms$. By doing so, the handshake only takes 1.5 RTT instead of 2 RTT.

**Problem 10.8 TLS Renegotiation**

Does the Ray-Dispensa attack also work if the client always requests to use TLS session resumption?

**Solution**

No. TLS 1.2 renegotiation always requires a full handshake, not session resumption. Renegotiation happens on an existing connection to establish new keys or change security parameters, while session resumption is only for reconnecting with a new TCP connection using cached session parameters.

**Problem 10.9 TLS Extensions**

(a) SNI: Why can't the server just use the `Host` HTTP header, which contains the domain name of the requested resource and is mandatory for every HTTP request, to determine the correct server certificate?

(b) ALPN: The authors of [13] propose to use the ALPN extension to prevent cross-protocol attacks over TLS. In a cross-protocol attack, an attacker redirects HTTP requests to FTP or SMTP servers and redirects back their answers to the web browser. Why would this extension prevent such attacks?

**Solution**

(a) Because the `Host` header is only sent *after* the TLS handshake is completed.

(b) Because the FTPS or SMTPS server would be informed during the handshake that the client intends to talk HTTPS afterward, and could deny the connection.

**Problem 10.10 HTTP Headers Affecting TLS**

Due to a hardware crash, your company loses all private keys associated with your TLS server certificates. How can you recover from this crash if
(a) you have configured HSTS on all your web servers?
(b) you have configured HPKP on all your web servers?

**Solution**

(a) HSTS: Just configure new, valid certificates on your servers.
(b) HPKP: If you also lost the private keys to your 'next' TLS certificates, then you can't do anything: You have blocked all browsers that have pinned your certificates from accepting any other certificate.

**Problem 10.11 DTLS**

(a) In TLS, whenever MAC validation fails, this is a critical error; the TLS connection must be terminated, and all key material must be deleted. In DTLS, a failed MAC validation is *not* critical. Which type of network error would justify this modified behavior?
(b) If a TLS Renegotiation is completed, old key material can be deleted. In DTLS, this shouldn't be done. Please explain why.

**Solution**

(a) RFC 6347 names Denial-of-Service attacks as the reason for this behaviour:

> Unlike TLS, DTLS is resilient in the face of invalid records (e.g., invalid formatting, length, MAC, etc.). In general, invalid records SHOULD be silently discarded, thus preserving the association; however, an error MAY be logged for diagnostic purposes. Implementations which choose to generate an alert instead, MUST generate fatal level alerts to avoid attacks where the attacker repeatedly probes the implementation to see how it responds to various types of error. Note that if DTLS is run over UDP, then any implementation which does this will be extremely susceptible to denial-of-service (DoS) attacks because UDP forgery is so easy. Thus, this practice is NOT RECOMMENDED for such transports.

(b) In DTLS, encrypted records can arrive out-of-order. Therefore, old key generations should be kept for a certain time period to be able to decrypt such records from the previous epoch.