Solutions to Selected Problems

Guide to Internet Cryptography

Companion Material

February 6, 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 4: Introduction - Cryptographic Protocols

Problem 4.1 Passwords: Salt

When a *salt* is used for hashing, wouldn't it be better to keep the *salt* secret? Hint: As described in this chapter, an individual salt is used for each user.

Solution

No, keeping the salt secret does not provide a meaningful security benefit.
A salt is not intended to be a secret value; its purpose is to ensure that identical passwords result in different hash values and to prevent the use of precomputed attacks such as rainbow tables. Since an individual salt is used for each user, an attacker must attack each password hash separately, even if multiple users share the same password.
Keeping the salt secret does not significantly increase security because, if an attacker gains access to the password database, they will typically also gain access to the salts. The security of password hashing should therefore not rely on the secrecy of the salt, but on the computational cost of the hashing function (e.g., using slow, adaptive hash functions such as `bcrypt`, `scrypt`, or `Argon2`).
Relying on a secret salt would violate Kerckhoffs' principle, which states that a system should remain secure even if everything about the system, except the secret key, is public. In password hashing, the password itself is the only secret; the salt can safely be stored alongside the hash.

Problem 4.2 Passwords: Rainbow Tables

Your fellow student asks: "Why don't we use a single chain in a rainbow table? This would save storage since we only have to store the first password and the last hash value." Is she right?

Solution

No, using a single chain in a rainbow table would not be practical or effective.
Rainbow tables rely on many chains to cover a large portion of the password space. Each chain represents a sequence of alternating hash and reduction functions. If only a single chain were used, it would cover only a very small subset of all possible passwords, making the probability of successfully reversing a given hash extremely low.
Additionally, long single chains suffer heavily from *collisions and merges*. When two chains reach the same intermediate value, they merge and follow the same path thereafter. In a single long chain, this would drastically reduce coverage, since many possible starting passwords would map to the same chain segments, wasting computation and offering no additional benefit.
Using many shorter chains mitigates this problem: merges only affect small portions of the table, and overall coverage of the password space is much higher. Although storing more chains requires more storage, it is a necessary trade-off to make rainbow tables effective.
Therefore, while a single chain would indeed save storage, it would render the rainbow table largely useless in practice.

Problem 4.3 Authentication: OTP

If the internal clocks of A and B may differ up to $25s$, and if the round-trip-time of the network is $12s$: Is it enough to check only two MAC values in Figure 4.3 to avoid False Negatives?

Solution

No, checking only two MAC values is not sufficient to avoid false negatives.
In RFC 6238 (TOTP), the MAC is computed over a time counter

$$T = \left\lfloor \frac{t - T_0}{X} \right\rfloor,$$

where X is the time step (typically 30s). To tolerate clock drift and network delay, the verifier must accept MACs computed for multiple adjacent time steps.
Here, the clocks of A and B may differ by up to 25s, and the round-trip time of the network is 12s, implying a one-way delay of up to 6s. In the worst case, the total time offset between the sender's MAC computation and the verifier's check is therefore

$$25\text{s} + 6\text{s} = 31\text{s}.$$

Since this exceeds a single time step of 30s, the MAC generated by A may correspond to a time counter that differs by more than ± 1 step from the verifier's current counter. Checking only two MAC values (typically the current and one adjacent time step) may therefore reject a valid MAC.
To avoid false negatives, the verifier must check a larger window of time steps (e.g., at least ± 2 steps), ensuring that all possible valid MACs within the maximum clock skew and network delay are accepted.

Problem 4.4 Authentication: Challenge-and-Response

Consider the following Challenge-and-Response variant: In Figure 4.4, B sends $c = ENC_{k_{AB}}(chall)$ and checks if $res = chall$. Compare the two variants assuming that B uses weak randomness to generate $chall$.

Solution

We compare the original challenge–response variant (where B sends *chall* in the clear) with the modified variant in which

$$c = ENC_{k_{AB}}(chall)$$

is sent and B checks whether $res = chall$.
Assume that B uses *weak randomness* to generate *chall*.

Original variant (unencrypted challenge). If *chall* is sent in the clear and drawn from a small or predictable space, an attacker can:

- predict future challenges, or
- record previous $(chall, res)$ pairs and replay a valid response if a recorded challenge is asked again by B.

Thus, weak randomness enhances the probability for successful replay attacks. But a replay attack is only possible when the response (the encryption or MAC od the challenge under the given key) was previously recorded. So the success probability is very low at the beginning of the use of this specific key, and increases only lienearly with the lifetime of the key.

Encrypted-challenge variant. In the modified scheme, the challenge is encrypted under the shared secret key k_{AB}. A passive attacker nevertheless learns *chall*, but that's not important.
As a result:

- an attacker can start impersonification attacks without recording an encryption dictionary,
- by simply sampling a random challenge $chall'$ and sending it back to B.

Conclusion. The encrypted-challenge variant is strictly weaker when B uses weak randomness. While both variants rely on a secret key, the variantcritically depends on the unpredictability of *chall*, whereas the original version additionally requires an encryption/MAC disctionary.

Problem 4.5 Key exchange

Suppose the private key of party B was leaked to an adversary who did record all previously exchanged messages between A and B. Which methods described in section 4.3 will protect the confidentiality of the exchanged data even in this scenario?

Solution

This question asks which mechanisms provide *confidentiality after key compromise*, i.e. *forward secrecy*.
Assume that an adversary has:

- recorded all past communication between A and B, and
- later obtains the long-term private key of B.

We analyze each method.

Diffie–Hellman Key Exchange (DHKE). If DHKE is used with *ephemeral* private values (DHE or ECDHE), the session key is derived from temporary secrets that are erased after the session. Even if B's long-term private key is later compromised, past session keys cannot be recovered.

$$\Rightarrow \text{DHKE provides forward secrecy and protects past confidentiality.}$$

RSA Encryption. In RSA encryption, the session key (or plaintext) is encrypted directly using B's public key. If the corresponding private key is later leaked, the adversary can decrypt all previously recorded ciphertexts.

$$\Rightarrow \text{RSA encryption does } \textit{not} \text{ provide forward secrecy.}$$

ElGamal KEM. ElGamal KEM derives a session key using fresh randomness for each encapsulation. B recovers this shared secret from the ciphertext with the help of his private key. If B's private key is later compromised, an attacker can thus derive all shared secrets from the recorded ciphertexts.

$$\Rightarrow \text{ElGamal KEM doen not provide forward secrecy.}$$

Conclusion. The method that protect the confidentiality of past communication after leakage of B's private key is:

$$\boxed{\text{DHKE}}$$

while RSA encryption and ElGamal KEM do not.

Problem 4.6 Authenticated key exchange

Please explain why the signed Diffie-Hellman protocol from Figure 4.8 does not use any variant of challenge-and-response (or certificate/verify). Suppose the ephemeral value a has been leaked to an adversary. Can she then impersonate A? To which other parties?

Solution

Since digital signatures are used, the correct comparison would be with the certificate/verify protocol. In this protocol, the digital signatures cover data provided by the other party: The signature of A covers the challenge chosen by B. In the signed Diffie-Hellman protocol, the digital signature of A only covers a value chosen by A, and similarly for B.
So it the ephemeral value a of A is leaked, an attacker can impersonate A by always replaying (α, σ_A), without knowing the long-lived private key of A.

Problem 4.7 Authenticated key exchange

Can you find the following basic protocols in the complex TLS handshake depicted in Figure 10.14? (a) DHKE (b) Certificate/Verify (c) Challenge-and-Response.

Solution

(a) DHKE is contained in SKE and CKE.
(b) Certificate/Verify is contained CH and SKE (the value r_C is included in the signature contained in SKE), and optionally in SH and CV (the value r_S is contained in the transcript of the protocol, and this transcript is signed; the signature is included in CV).
(c) Challenge-and-Response is contained in CH and FIN_S, and in SH and FIN_C. In both cases, a challenge r_C (r_S, resp.) is contained in CH (SH, resp.), and this challenge is input to a MAC function together with the rest of the transcript, under the master secret as a key. These MACs FIN_S (FIN_C, resp.) are included in the two Finished messages.

Problem 4.8 Formal models

How would you model a governmental agency which has control over several network nodes and may request long-lived keys through court orders?

Solution

I would provide my formal model attacker with more capabilities; if my protocol is secure against this stronger attacker, it will also be secure against the (weaker) government agency.
In particular, I would model that the formal attacker controls all of the Internet, not only some nodes. And I would model that the formal attacker can request long-lived keys from any protocol party.
This trivially enables the adversary to learn some secret messages, or to forge some MACs or signatures. The same trivially holds for the government agency: If the agency learns the private key of party B, the agency can impersonate B in the future, and maybe read old or new messages. If only such trivial attacks are possible, I woul still believe my system is as secure as it can be.