

Applied Cryptography

Week #10 Extra

Bernardo Portela and Rogério Reis

2025/2026

Important

- Your answers must **always** be accompanied by a justification. Presenting the final result (e.g. the result of a calculation) without the rationale that laid to said result will result in a grade of 0.
- Submit your answers via e-mail to *bernardo.portela@fc.up.pt*, with adequate identification of the group and its members.

Q1: Man-in-the-Middle

Implement a prototype that demonstrates how a Man-in-the-Middle attack can occur in a standard unauthenticated Diffie-Hellman key exchange. As usual, this happens between usual suspects *Alice* and *Bob*. In appendix, you can find four files:

- `config_alice` and `config_bob` are configuration files that are used by Alice and Bob to know who to talk to.
- `alice.py` establishes a connection with Bob (hopefully), and performs the Diffie-Hellman key exchange.
- `bob.py` mirrors the behavior of Alice.

The goal of the work is to design the man-in-the-middle adversary code: `mitm.py`. Your code must convince Alice and Bob to instead talk to him, and perform a key exchange with him. **Your attack must not change the source code of Alice or Bob.** Your attack is successful if Alice and Bob are **not** agreeing on the same secret, and the secrets they have agreed to are both known to the adversary.

Suggestion: Start by analysing the code for Alice and Bob, what are they using to communicate? How can we subvert this mechanism to be more... convenient?

To facilitate communication, this code uses *pwntools* (reference). It is not mandatory to use this, but the library considerably facilitates communication.

Q2: ECC

The following is a naive attempt at an elliptic curve signature scheme. Consider a global elliptic curve, prime p and generator G . The scheme works as follows.

- Alice picks a private signing key sk_A and forms the public verifying key by computing $pk_A \leftarrow sk_A \cdot G$
- To sign message m , Alice picks a random value k , and computes the signature $\sigma \leftarrow m - k \cdot sk_A \cdot G$. It then sends to Bob the tuple (m, k, σ)
- To verify the signature, Bob checks that $m = \sigma + k \cdot pk_A$. If this is true, the signature is validated.

Question - P1: Show that the scheme works, i.e. show that, for correctly signed messages, the verification algorithm works accordingly.

Question - P2: Show that this scheme is vulnerable, by describing a simple technique for forging a signature on an arbitrary message, without knowledge of the secret key sk_A . *Hint:* consider what computations can one do using simply pk_A

Q3: ElGamal

ElGamal is a public-key encryption scheme. Its reasoning is similar to that of classical Diffie-Hellman, using g^x as the public key, and having the encryption encapsulate the message with g^y .

The algorithms are presented below, assuming operations in the group \mathbb{Z}_q , with generator g . Encryption assumes that m is an element of \mathbb{Z}_q , which can be achieved by having a reversible mapping function from the message domain to the group domain. s^{-1} means the inverse of s .

Algorithm Gen(): $x \leftarrow \$\{1, \dots, (q-1)\}$ $X \leftarrow g^x$ Return (X, x)	Algorithm Enc(X, m): $y \leftarrow \$\{1, \dots, (q-1)\}$ $s \leftarrow X^y$ $Y \leftarrow g^y, c \leftarrow m \cdot s$ Return (Y, c)	Algorithm Dec($x, (Y, c)$): $s \leftarrow Y^x$ $m \leftarrow c \cdot s^{-1}$ Return m
--	---	--

Figure 1: ElGamal encryption scheme

Question - P1: Describe why decryption works – show how m is recovered

Question - P2: Explain how the hardness of the discrete logarithm ensures confidentiality

Question - P3: ElGamal is malleable – show how it can be done. Consider an adversary that can request the encryption of message m , receiving ciphertext c , and show how can he present another ciphertext c' that will decrypt in a related way.