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 3: Introduction - Integrity and Authenticity

Problem 3.1 Mental Model for Hash Functions

Please point out differences between hash functions and their mental model, human fingerprints.

Solution

Hash values change with the context, human fingerprints don't. While even genetically identical twins have different fingerprints, we cannot exclude the possibility that the fingerprints from different persons may be 'close'. Hash values of the same data are identical, while two fingerprints from the same finger of the same person are noisy data. AI may give you additional differences.

Problem 3.2 Iterative Structure of Hash Functions

How many calls to its internal compression function does SHA-256 need to compute the hash value of a message of 12 kbyte?

Solution

Given SHA-256 processes messages in fixed-size blocks of

$$512 \text{ bits} = 64 \text{ bytes}.$$

The message length is

$$12 \text{ kB} = 12 \times 1024 = 12{,}288 \text{ bytes}.$$

Padding SHA-256 padding consists of:

- one `1` bit,
- enough `0` bits to reach a length congruent to 448 mod 512,
- a 64-bit (8-byte) encoding of the original message length.

Thus, padding adds at least

$$1 \text{ bit} + 64 \text{ bits} = 65 \text{ bits} \approx 9 \text{ bytes}.$$

Total Length The padded message length in bytes is

$$12{,}288 + 9 = 12{,}297 \text{ bytes}.$$

Number of Blocks The number of 64-byte blocks is

$$\left\lceil \frac{12{,}297}{64} \right\rceil = \lceil 192.14 \rceil = 193.$$

Answer SHA-256 invokes its internal compression function **193 times** to compute the hash of a 12 kB message.

Problem 3.3 Attacks on Hash Functions

Why (and how) can the birthday paradox be exploited to break collision resistance of hash functions, but not second preimage resistance?

Solution

Security notions Let $H : \{0,1\}^* \to \{0,1\}^n$ be a hash function.

- *Collision resistance*: it is infeasible to find $x \neq x'$ such that $H(x) = H(x')$.
- *Second preimage resistance*: given a fixed input x, it is infeasible to find $x' \neq x$ such that $H(x') = H(x)$.

Although these notions look similar, the attacker's freedom is very different.

Birthday paradox intuition The birthday paradox states that for a random function with n-bit outputs, collisions are likely after about

$$q \approx 2^{n/2}$$

random samples, because the number of unordered pairs grows as

$$\binom{q}{2} \approx \frac{q^2}{2}.$$

Exploiting the birthday paradox for collisions To break collision resistance, an attacker may choose *both* inputs freely.

1. Choose q random messages $x_1, \ldots, x_q$.
2. Compute $H(x_1), \ldots, H(x_q)$.
3. Look for any pair (i, j) with $i \neq j$ such that

$$H(x_i) = H(x_j).$$

Since there are about $q^2/2$ pairs, the probability of at least one collision becomes significant when

$$\frac{q^2}{2} \approx 2^n \implies q \approx 2^{n/2}.$$

Thus, collision resistance can be broken generically in $O(2^{n/2})$ hash evaluations.

Why this does not work for second preimages In a second-preimage attack, the attacker is *not* free to choose both inputs.

- A specific message x is fixed in advance.
- The attacker must find $x' \neq x$ such that

$$H(x') = H(x).$$

Now there is only *one target value* $H(x)$. Each new trial x' succeeds with probability

$$\Pr[H(x') = H(x)] = 2^{-n}.$$

Complexity gap For second preimages, the expected number of trials is therefore

$$2^n,$$

not $2^{n/2}$. The quadratic amplification that powers the birthday attack disappears, because we are no longer comparing *all pairs*, but only comparing each trial to a single fixed hash value.

Core reason for the difference

	Collision resistance	Second preimage resistance
Attacker chooses both inputs?	Yes	No
Number of relevant pairs	$\Theta(q^2)$	$\Theta(q)$
Generic attack cost	$2^{n/2}$	2^n

Conclusion The birthday paradox exploits the attacker's freedom to choose many inputs and compare all pairs, which breaks collision resistance in $O(2^{n/2})$ time. Second preimage resistance fixes one input in advance, eliminating the quadratic pairing effect and forcing a brute-force search of $O(2^n)$. This asymmetry is why the birthday paradox applies to collisions, but not to second preimages.

Problem 3.4 Insecure MAC Construction

On a code exchange web page, you find the following construction for a MAC function, advertised as "using military-grade encryption to secure the integrity of your data":

$$MAC_k(m) := ENC_k(Hash(m))$$

How can you break this construction if malleable encryption is used, e.g., a stream cipher?

Solution

Security notions Let $H : \{0,1\}^* \to \{0,1\}^n$ be a hash function.

- *Collision resistance*: it is infeasible to find $x \neq x'$ such that $H(x) = H(x')$.
- *Second preimage resistance*: given a fixed input x, it is infeasible to find $x' \neq x$ such that $H(x') = H(x)$.

Although these notions look similar, the attacker's freedom is very different.

Birthday paradox intuition The birthday paradox states that for a random function with n-bit outputs, collisions are likely after about

$$q \approx 2^{n/2}$$

random samples, because the number of unordered pairs grows as

$$\binom{q}{2} \approx \frac{q^2}{2}.$$

Exploiting the birthday paradox for collisions To break collision resistance, an attacker may choose *both* inputs freely.

1. Choose q random messages $x_1, \dots, x_q$.
2. Compute $H(x_1), \dots, H(x_q)$.
3. Look for any pair (i, j) with $i \neq j$ such that
$$H(x_i) = H(x_j).$$

Since there are about $q^2/2$ pairs, the probability of at least one collision becomes significant when

$$\frac{q^2}{2} \approx 2^n \implies q \approx 2^{n/2}.$$

Thus, collision resistance can be broken generically in $O(2^{n/2})$ hash evaluations.

Why this does not work for second preimages In a second-preimage attack, the attacker is *not* free to choose both inputs.

- A specific message x is fixed in advance.
- The attacker must find $x' \neq x$ such that
$$H(x') = H(x).$$

Now there is only *one target value* $H(x)$. Each new trial x' succeeds with probability

$$\Pr[H(x') = H(x)] = 2^{-n}.$$

Complexity gap For second preimages, the expected number of trials is therefore

$$2^n,$$

not $2^{n/2}$. The quadratic amplification that powers the birthday attack disappears, because we are no longer comparing *all pairs*, but only comparing each trial to a single fixed hash value.

Core reason for the difference

	Collision resistance	Second preimage resistance
Attacker chooses both inputs?	Yes	No
Number of relevant pairs	$\Theta(q^2)$	$\Theta(q)$
Generic attack cost	$2^{n/2}$	2^n

Conclusion The birthday paradox exploits the attacker's freedom to choose many inputs and compare all pairs, which breaks collision resistance in $O(2^{n/2})$ time. Second preimage resistance fixes one input in advance, eliminating the quadratic pairing effect and forcing a brute-force search of $O(2^n)$. This asymmetry is why the birthday paradox applies to collisions, but not to second preimages.

Problem 3.5 Pseudorandom Functions

Give a construction of how HMAC can generate pseudorandom byte sequences of arbitrary length. How is HMAC used in the TLS 1.1 PRF?

Solution

HMAC as a pseudorandom function Let

$$\mathrm{HMAC}_k(\cdot) : \{0,1\}^* \to \{0,1\}^n$$

be HMAC instantiated with a cryptographic hash function (e.g. SHA-256). Under standard assumptions, HMAC behaves as a pseudorandom function (PRF).
Since HMAC outputs only n bits per invocation, we need an *iterative construction* to generate pseudorandom output of arbitrary length.
PRG construction from HMAC Let k be a secret key and *seed* be a public or secret seed. Define:

$$\begin{aligned} T_1 &= \mathrm{HMAC}_k(\mathit{seed}), \\ T_2 &= \mathrm{HMAC}_k(T_1 \,\|\, \mathit{seed}), \\ T_3 &= \mathrm{HMAC}_k(T_2 \,\|\, \mathit{seed}), \\ &\vdots \end{aligned}$$

The output stream is

$$\mathrm{PRG}_k(\mathit{seed}) = T_1 \,\|\, T_2 \,\|\, T_3 \,\|\, \cdots$$

and is truncated to the desired length.
Security intuition Each block T_i is pseudorandom given the previous blocks, because:

- HMAC is a PRF keyed by k,
- chaining ensures domain separation between successive calls,
- knowledge of earlier outputs does not reveal k.

Thus, the construction yields a secure pseudorandom byte sequence of arbitrary length.

TLS 1.1 PRF overview TLS 1.1 uses HMAC to derive keying material from shared secrets. The PRF combines:

- a secret S (the master secret or keying material),
- a public label,
- a public seed.

The `P_hash` construction TLS defines an internal function P_hash based on HMAC:

$$\text{P\_hash}(S,\, seed) = \text{HMAC}_S(A_1 \,\|\, seed) \,\|\, \text{HMAC}_S(A_2 \,\|\, seed) \,\| \,\cdots$$

where

$$A_0 = seed,$$
$$A_i = \text{HMAC}_S(A_{i-1}) \quad \text{for } i \geq 1.$$

The output is truncated to the required length.

TLS 1.1 PRF definition In TLS 1.1, the PRF is defined as:

$$\text{PRF}(S, label, seed) = \text{P\_hash}(S,\, label \,\|\, seed).$$

Unlike TLS 1.0, TLS 1.1 uses *a single hash function* (chosen by the cipher suite) rather than mixing MD5 and SHA-1.

Usage in TLS The TLS 1.1 PRF is used to:

- derive the *master secret* from the premaster secret,
- expand the master secret into encryption keys, MAC keys, and IVs,
- ensure cryptographic separation via different labels.

Summary

- HMAC can generate arbitrary-length pseudorandom output via iterative chaining.
- TLS 1.1 instantiates this idea through the `P_hash` construction.
- Labels and seeds provide domain separation and prevent key reuse.

Problem 3.6 Authenticated Encryption

Is Encrypt-and-MAC INT-CTXT secure?

Solution

Setting Let $(\text{ENC}_k, \text{DEC}_k)$ be a symmetric encryption scheme and $\text{MAC}_{k'}$ be a message authentication code. The *Encrypt-and-MAC* construction is defined as:

$$c := \text{ENC}_k(m),$$
$$t := \text{MAC}_{k'}(m),$$

and the transmitted ciphertext is the pair (c, t). Verification checks the MAC on the *plaintext*.
INT-CTXT security Integrity of ciphertexts (INT-CTXT) requires that it be infeasible for an adversary to produce a *new valid ciphertext* that will be accepted by the decryption algorithm, except with negligible probability.
Formally, the adversary should not be able to create (c, t) such that:

$$\text{DEC}_k(c) = m \quad \text{and} \quad \text{MAC}_{k'}(m) = t,$$

unless (c, t) was previously output by the encryption oracle.

Why Encrypt-and-MAC is *not* INT-CTXT secure Encrypt-and-MAC generally fails to provide INT-CTXT security, even if:

- the encryption scheme is IND-CPA secure, and
- the MAC is existentially unforgeable (EUF-CMA).

The reason is that the MAC authenticates the *plaintext*, not the ciphertext.

Attack intuition Suppose the encryption scheme is *malleable* (as many IND-CPA schemes are). Then:

- an adversary can modify a ciphertext c into c',
- the modified ciphertext decrypts to a related plaintext $m' \neq m$,
- but the MAC $t = \mathrm{MAC}_{k'}(m)$ remains unchanged.

During verification:

$$\mathrm{DEC}_k(c') = m' \quad \text{and} \quad \mathrm{MAC}_{k'}(m') \neq t,$$

so the ciphertext is rejected. This looks safe at first glance.

The real problem: re-randomization attacks If the encryption scheme allows *multiple ciphertexts for the same plaintext* (e.g. randomized encryption), then an adversary can do the following:

1. Obtain a valid encryption (c, t) of some message m.
2. Re-randomize the ciphertext to get $c' \neq c$ such that
$$\mathrm{DEC}_k(c') = m.$$
3. Output the pair (c', t).

This ciphertext is:

- *new* (never output by the encryption oracle),
- *valid*, since it decrypts to m and the MAC on m verifies.

Thus, the adversary has forged a fresh valid ciphertext.

Conclusion of the attack The construction allows existential forgery of ciphertexts without breaking either the encryption or the MAC individually. Therefore, Encrypt-and-MAC does *not* achieve INT-CTXT security in general.

Comparison with secure compositions For contrast:

- **Encrypt-then-MAC** (MAC over the ciphertext) *is* INT-CTXT secure under standard assumptions.
- **MAC-then-Encrypt** is more subtle and depends on details of the encryption scheme.

Final answer

$$\boxed{\text{Encrypt-and-MAC is \textit{not} INT-CTXT secure in general.}}$$

The failure stems from authenticating the plaintext instead of the ciphertext, which allows new valid ciphertexts to be generated without forging a MAC.

Problem 3.7 Digital Signatures: PKCS#1

Why is the PKCS#1 padding for encryption probabilistic but deterministic for digital signatures?

Solution

Different security goals PKCS#1 defines two padding schemes with different purposes:

- **RSAES-PKCS1-v1_5** for *encryption*,
- **RSASSA-PKCS1-v1_5** for *digital signatures*.

The difference in padding randomness follows directly from the different security goals of encryption and signatures.

Encryption: need for semantic security Public-key encryption aims to achieve *semantic security* (IND-CPA): an attacker should not learn whether two encryptions correspond to the same plaintext.
If encryption padding were deterministic, then for a public key pk,

$$\mathrm{Enc}_{pk}(m)$$

would always produce the same ciphertext for the same message m, immediately leaking equality information.

Probabilistic padding in encryption PKCS#1 encryption padding therefore includes random bytes:

$$\texttt{EM} = 00 \,\|\, 02 \,\|\, R \,\|\, 00 \,\|\, M,$$

where R is a string of nonzero random bytes.
Randomness ensures that:

- encryptions of the same message differ each time,
- attackers cannot test guesses by recomputing ciphertexts,
- the scheme resists chosen-plaintext attacks.

Thus, probabilistic padding is essential for encryption security.

Signatures: need for reproducibility Digital signatures aim to provide:

- *authenticity*,
- *integrity*,
- *non-repudiation*.

For a signature to be verifiable by anyone, verification must be deterministic: given (m, σ), the verifier must be able to recompute exactly what was signed.

Deterministic padding in signatures PKCS#1 signature padding is deterministic:

$$\texttt{EM} = 00 \,\|\, 01 \,\|\, FF \ldots FF \,\|\, 00 \,\|\, \mathrm{Hash}(M).$$

No randomness is used, so:

- the same message always yields the same padded value,
- verifiers do not need extra information,
- signatures are easy to validate and compare.

Why randomness is unnecessary for signatures For signatures, revealing that the same message was signed twice is not a security problem. In fact, determinism is often desirable:

$$\sigma = \mathrm{Sign}_{sk}(m)$$

should be reproducible and uniquely bound to m.
Security against forgery (EUF-CMA) does not require randomness in the padding itself; it relies on the hardness of the underlying problem (e.g. RSA) and

Problem 3.8 Digital Signatures: RSA

In the ElGamal signature scheme, all signers may use the same modulus p. Why isn't this possible for RSA, even if all signers would use different private exponents d?

Solution

ElGamal signature scheme ElGamal signatures work over a large prime p and a generator g of $\mathbb{Z}_p^*$. Each signer chooses a *private key* x and computes the *public key* $y = g^x \bmod p$.
The important observation is:

- The modulus p is *public and shared.*
- Each signer has a unique private key x (and corresponding y).
- The security of ElGamal depends on the discrete logarithm problem in $\mathbb{Z}_p^*$, not on unique factorization.

Thus, multiple signers can safely share the same p without compromising security.

RSA key generation In RSA, a key pair (e, d, N) is generated as follows:

$$N = pq, \quad \varphi(N) = (p-1)(q-1), \quad ed \equiv 1 \pmod{\varphi(N)}.$$

Crucially:

- The modulus N must remain secret in terms of its factorization.
- The private exponent d is computed relative to $\varphi(N)$.

Problem with sharing N in RSA If two users share the same modulus N but choose different exponents d_1, d_2:

$$e_1 d_1 \equiv 1 \pmod{\varphi(N)}, \quad e_2 d_2 \equiv 1 \pmod{\varphi(N)},$$

then an attacker can exploit *the shared modulus attack*:

$$\text{Given } N, e_1, e_2 \text{ and ciphertexts } c_1 = m^{e_1}, c_2 = m^{e_2},$$

one can compute m using the extended Euclidean algorithm on (e_1, e_2) because the same $\varphi(N)$ applies to both:

$$s_1 e_1 + s_2 e_2 = 1 \implies m = c_1^{s_1} c_2^{s_2} \pmod{N}.$$

Even without ciphertexts, having multiple d_i for the same N can leak information about $\varphi(N)$ or allow recovery of other private keys.

Why ElGamal is safe In ElGamal, even if p is shared:

- The discrete logarithm problem is independent for each public key y_i.
- No algebraic relation between different x_i values helps an attacker.

Conclusion RSA moduli must be generated independently for each signer because sharing N with different private exponents d enables attacks that can recover d or break confidentiality. ElGamal relies on discrete logs and supports shared p without compromising the independence of each signer's key.

Problem 3.9 Digital Signatures: ElGamal

In one step, the ElGamal signature algorithm uses the fact that in the group $\mathbb{Z}_p^*$, group elements are numbers. How must this step be modified when adapting ElGamal to an elliptic curve $EC(a, b)$?

Solution

ElGamal signature over $\mathbb{Z}_p^*$ Recall the standard ElGamal signature steps over a multiplicative group $\mathbb{Z}_p^*$ with generator g:
1. Choose a private key $x \in \{1, \dots, p-2\}$ and compute the public key $y = g^x \bmod p$. 2. To sign a message m:

1. Pick a random k coprime to $p - 1$.
2. Compute $r = g^k \bmod p$. $\leftarrow$ group element is a number mod p
3. Compute $s = k^{-1}(H(m) - xr) \bmod (p - 1)$.
4. Signature is (r, s).

The step $r = g^k \bmod p$ relies on the fact that group elements are **numbers**, so we can multiply, exponentiate, and reduce modulo p.

Adapting to an elliptic curve $E(\mathbb{F}_q)$ On an elliptic curve over a finite field $\mathbb{F}_q$, the group is $(E(\mathbb{F}_q), +)$ with:
- Addition of points instead of multiplication, - Scalar multiplication $k \cdot G = G + \cdots + G$ (repeated addition) instead of exponentiation.

Modification of the problematic step The numeric exponentiation step $r = g^k \bmod p$ is replaced by **elliptic curve scalar multiplication**:

$$R = k \cdot G$$

where:
- $G \in E(\mathbb{F}_q)$ is a fixed base point of large prime order n, - $k \in \{1, \dots, n-1\}$ is a randomly chosen ephemeral key, - R is a point on the curve, not a number modulo p.

Extracting a scalar from the point For the signature, we need a scalar r modulo n. Typically, we take:

$$r = x_R \bmod n$$

where x_R is the x-coordinate of the point $R = (x_R, y_R)$.

Other computations - The computation of s also changes to use modular arithmetic modulo the order n of G:

$$s = k^{-1}(H(m) + x \cdot r) \bmod n$$

where x is the private key.
- The signature is (r, s), similar to the original scheme.

Summary of the key adaptation

$$\text{Numeric exponentiation } g^k \bmod p \quad \longrightarrow \quad \text{Elliptic curve scalar multiplication } R = k \cdot G$$

- The group operation switches from multiplicative to additive. - A numeric value r is derived from the x-coordinate of the resulting point. - All modular arithmetic is performed modulo the order of the base point G, not modulo $p - 1$.

Problem 3.10 Digital Signatures: RSA

Why can't RSA be adapted to elliptic curves?

Solution

RSA overview RSA relies on the following key ideas:

- Select two large primes p and q and compute $N = pq$.
- Define the totient $\varphi(N) = (p - 1)(q - 1)$.
- The public exponent e and private exponent d satisfy

$$ed \equiv 1 \pmod{\varphi(N)}.$$

- Encryption/signing is done via exponentiation modulo N:

$$c = m^e \bmod N, \quad m = c^d \bmod N.$$

The security of RSA rests on the *difficulty of factoring* N to recover $\varphi(N)$ and d.

Why elliptic curves are incompatible with RSA Elliptic curve groups $E(\mathbb{F}_q)$ have very different algebraic structure:

- The group is $(E(\mathbb{F}_q), +)$, an *additive abelian group.*
- There is no natural analog of integer multiplication modulo a composite number $N = pq$.
- There is no analog of the Euler totient $\varphi(N)$ in EC groups that allows computing a modular inverse in the same way.
- "Exponentiation modulo N" in RSA is essential for encryption/signing. In ECs, the analogous operation is *scalar multiplication*, which is already one-way under the discrete logarithm assumption.

Key points of incompatibility

1. RSA relies on **integer factorization and arithmetic modulo** N to define a trapdoor function. EC groups are not based on integer factorization, and there is no composite modulus N to factor.
2. In RSA, the security trapdoor comes from knowing $\varphi(N)$. On elliptic curves, the order of a point or the curve is usually public, so there is no hidden trapdoor that allows "inverting" scalar multiplication in the same algebraic way as RSA inversion.
3. Scalar multiplication on elliptic curves is already a one-way function (based on the discrete logarithm problem), making RSA-style trapdoor exponentiation unnecessary.

Conclusion

RSA cannot be adapted to elliptic curves because EC groups lack the structure of *modular integer multipl*

Elliptic curves are naturally used for schemes like ElGamal, DSA, or ECDSA, which are based on the *discrete logarithm problem*, not on integer factorization. RSA's trapdoor mechanism does not translate to the additive group structure of elliptic curves.

Problem 3.11 Digital Signatures: DSA

Why does $k^{-1} \equiv k^{q-2} \pmod q$ hold?

Solution

Setting Let q be a prime and k an integer such that $k \not\equiv 0 \pmod q$. We want the multiplicative inverse of k modulo q, i.e., k^{-1} such that:

$$k \cdot k^{-1} \equiv 1 \pmod q.$$

Fermat's little theorem Fermat's little theorem states that for any integer k not divisible by the prime q:

$$k^{q-1} \equiv 1 \pmod q.$$

Derivation of the inverse formula Divide both sides of $k^{q-1} \equiv 1 \pmod q$ by k (or equivalently multiply by k^{-1}):

$$k^{q-1} = k \cdot k^{q-2} \equiv 1 \pmod q.$$

Comparing with the definition of the modular inverse:

$$k \cdot k^{q-2} \equiv 1 \pmod q \implies k^{-1} \equiv k^{q-2} \pmod q.$$

Intuition - The formula works because in the multiplicative group $(\mathbb{Z}_q^*, \cdot)$, the order of any nonzero element is a divisor of $q-1$. - Raising k to the power $q-2$ effectively "cancels" k modulo q, producing the multiplicative identity 1.
Example Let $q = 7$, $k = 3$:

$$k^{q-2} = 3^{7-2} = 3^5 = 243 \equiv 5 \pmod 7.$$

Check:

$$3 \cdot 5 = 15 \equiv 1 \pmod 7.$$

Thus, $3^{-1} \equiv 3^5 \equiv 5 \pmod 7$, confirming the formula.
Summary

$$\boxed{k^{-1} \equiv k^{q-2} \pmod q \text{ for prime } q \text{ and } k \not\equiv 0 \pmod q}$$

- This is a direct consequence of Fermat's little theorem. - It is widely used in cryptography to compute inverses efficiently in prime fields.

Problem 3.12 Digital Signatures: Comparison

If all three signature schemes use a modulus of length 2048 bit, how long are (a) RSA signatures, (b) ElGamal signatures, and (c) DSA signatures?

Solution

Assumptions Let the modulus size be 2048 bits for all three schemes. We compute the signature lengths:

(a) RSA signatures An RSA signature is:

$$\sigma = m^d \bmod N$$

where N is the modulus. Its length equals the modulus size:

$$\text{Length of RSA signature} = 2048 \text{ bits.}$$

(b) ElGamal signatures An ElGamal signature is a pair (r, s) with:

$$r = g^k \bmod p, \quad s = k^{-1}(H(m) - xr) \bmod (p-1).$$

- Both r and s are integers modulo p (or $p-1$), - Each is roughly 2048 bits, so the total length is:

$$\text{Length of ElGamal signature} \approx 2048 + 2048 = 4096 \text{ bits.}$$

(c) DSA signatures A DSA signature is a pair (r, s) with:

$$r, s \in \mathbb{Z}_q$$

where q is a smaller prime (typically 256 bits for a 2048-bit modulus p).
- Both r and s are modulo q, - Therefore the total signature length is:

$$\text{Length of DSA signature} \approx 256 + 256 = 512 \text{ bits.}$$

Summary Table

Signature Scheme	Signature Length (bits)
RSA	2048
ElGamal	4096
DSA (with 256-bit q)	512

Intuition - RSA: single integer modulo N.
- ElGamal: two integers modulo p: roughly twice the modulus size.
- DSA: two integers modulo small q: much shorter than the modulus.

Problem 3.13 Digital Signatures: Mathematics

Let p be a prime number. Is there a subgroup of order d in $\mathbb{Z}_n^*$ for any divisor d of $p-1$? How can you construct such a subgroup?

Solution

Setting Let p be a prime number. Consider the multiplicative group:

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}, \quad |\mathbb{Z}_p^*| = p-1.$$

We ask: for any divisor $d \mid (p-1)$, is there a subgroup of order d in $\mathbb{Z}_p^*$, and how can we construct it?

—

Existence of subgroups $\mathbb{Z}_p^*$ is a *cyclic group* of order $p-1$. A fundamental result from group theory:

Every divisor of the order of a cyclic group corresponds to a unique subgroup of that order.

Since $d \mid (p-1)$, there exists a unique subgroup of $\mathbb{Z}_p^*$ of order d.
Yes, a subgroup of order d always exists.

—

Construction of a subgroup of order d 1. Find a *generator* g of $\mathbb{Z}_p^*$, i.e., an element of order $p-1$. (This is always possible because $\mathbb{Z}_p^*$ is cyclic.)
2. Compute the element

$$h = g^{(p-1)/d} \in \mathbb{Z}_p^*.$$

3. The subgroup of order d is

$$\langle h \rangle = \{h^0 = 1,\, h^1,\, h^2,\, \ldots,\, h^{d-1}\}.$$

- Check: $h^d = g^{(p-1)/d \cdot d} = g^{p-1} \equiv 1 \pmod{p}$, so the order of h is exactly d.

—

Summary

$$\begin{cases} \text{Cyclic group } \mathbb{Z}_p^* \text{ of order } p-1 \\ \forall d \mid (p-1), \exists \text{ a subgroup of order } d \\ \text{Construction: choose generator } g,\ h = g^{(p-1)/d},\ \langle h \rangle \end{cases}$$

This is the standard way to construct subgroups of any order dividing $p-1$ in a prime field.

Problem 3.14 Security Goal: Integrity and Authenticity

EUF-CMA allows the adversary to acquire valid signatures for any chosen message m. A slightly weaker and more realistic security goal would be EUF-KP, where the adversary only learns a list of valid message-signature pairs (m, σ) but cannot choose the messages. Is the Textbook RSA signature EUF-KP secure?

Solution

Setting - **Textbook RSA signature:** for a message $m \in \mathbb{Z}_N^*$, the signature is

$$\sigma = m^d \bmod N,$$

where (N, e, d) is the RSA key pair. Verification is

$$\sigma^e \equiv m \pmod{N}.$$

- **EUF-CMA (Existential Unforgeability under Chosen-Message Attack):** the adversary can request signatures on messages of its choice.
- **EUF-KP (Existential Unforgeability under Known-Message Attack):** the adversary only sees some (m_i, σ_i) pairs, cannot choose messages.

We ask: is Textbook RSA EUF-KP secure?

—

Attack intuition Textbook RSA is *multiplicatively homomorphic*:

$$(m_1^d \bmod N) \cdot (m_2^d \bmod N) = (m_1 m_2)^d \bmod N.$$

Suppose the adversary sees two valid signatures:

$$\sigma_1 = m_1^d \bmod N, \quad \sigma_2 = m_2^d \bmod N.$$

Then the adversary can compute

$$\sigma = \sigma_1 \cdot \sigma_2 \bmod N = (m_1 m_2)^d \bmod N$$

which is a valid signature for $m = m_1 m_2 \bmod N$, a message not previously signed. This is an existential forgery, even though the adversary never chose the messages.

—

Conclusion Textbook RSA is **not EUF-KP secure**, because:
- Seeing valid (m, σ) pairs allows the adversary to forge new signatures using the multiplicative property. - The attack does *not require chosen messages*, so it works even under the weaker EUF-KP model.

$$\boxed{\text{Textbook RSA signatures are } \textit{not} \text{ secure under EUF-KP.}}$$

Remark - To make RSA signatures secure, one must *randomize* the message before signing, e.g., using **RSA-PSS**. - Randomized or hash-based padding breaks the multiplicative structure, preventing this type of forgery.