NTNU | Norwegian University of Science and Technology

TTM4205 – Weekly Problems, Fall 2026

Secure Cryptographic Implementations

Assignment

This is one out of three assignments in the course TTM4205 Secure Cryptographic Implementations during the fall semester of 2026. The sheet contains **27 problems** related to most of the main topics from the lectures, requiring both mathematical and coding skills. A selection of the problems is taken from cryptohack.org and is marked *(CryptoHack)* in the problem heading; solving those requires a free CryptoHack account. We recommend using Python or Sage to implement your solutions.

Each problem is worth 2 points, and you are free to choose which problems you solve; however, the assignment counts for at most 40 points in total. Problems marked with a star ($\star$) are more challenging than the rest, and also the ones you are likely to learn the most from. All problems require detailed answers where you describe and document what you have done to complete the task, e.g., written explanations, calculations, code, graphs, etc. We might give full or partial credit if you show that you understand a problem and made an attempt to solve it even if you are not able to solve it entirely. For the CryptoHack problems, state the flag *and* include a short write-up explaining how you solved it; the flag on its own is not enough.

Collaboration. You are *allowed* to discuss the problems with other students and to ask the course staff for hints or pointers. However, you must write your own *individual* solutions in your own words; submitting a shared or copied write-up is not permitted. Please ask any questions about the assignment on the Canvas forum.

External resources. You are *allowed* to consult papers, write-ups, and other resources online. Every resource you rely on must be *clearly* cited with a reference. Using external material without attribution is considered cheating; see i.ntnu.no/wiki/-/wiki/English/Cheating+on+exams.

Use of AI. What you hand in must be your own work: the analysis, the explanations, and the code must come from you. You are *not* allowed to use AI tools to solve the assignment or to produce your write-up. You *may* use AI tools to polish a finished submission, for example to tidy up code or improve wording and formatting. In this assignment, that is the only permitted use.

All submissions must be written in LaTeX, and we provide a mandatory template to be used at overleaf.com/read/xxnmbmnpxxfq#6ce4e3.

Submission. Submit your work in Canvas; the submission link is available from the course website at ttm4205.iik.ntnu.no.

Deadline. December 4th at 23:59.

Contents

1 Randomness **3**
1.1 "It is truly random, I promise!" . . . 3
1.2 The Next of Your Kind . . . 3
1.3 This Destroyes the Schnorr Cryptosystem . . . 3
1.4 ElGusto ElGamal . . . 4
1.5 Ron was Wrong, Whit is Right (CryptoHack) . . . 4
1.6 No Random, No Bias (CryptoHack) . . . 4
1.7 Lo-Hi Card Game (CryptoHack) . . . 5
1.8 Trust Games ⋆ (CryptoHack) . . . 5
1.9 Prime and Prejudice ⋆ (CryptoHack) . . . 5
1.10 RSA vs. RNG ⋆ (CryptoHack) . . . 5

2 Legacy Crypto **5**
2.1 Export Grade (CryptoHack) . . . 5
2.2 Oh SNAP! (CryptoHack) . . . 6
2.3 Nothing up my Sleeve ⋆ (CryptoHack) . . . 6
2.4 MOVing Problems ⋆ (CryptoHack) . . . 6

3 Padding Oracles **7**
3.1 Null or Never (CryptoHack) . . . 7
3.2 Paper Plane (CryptoHack) . . . 7
3.3 Endless Emails ⋆ (CryptoHack) . . . 7

4 Quantum-Safe Crypto **7**
4.1 Noisy Withdrawal . . . 7
4.2 No Second Thoughts . . . 8

5 Crypto API Failures **8**
5.1 Fool Me Once, Fool Me Twice . . . 8
5.2 Faulty RSA Bites the Dust . . . 9
5.3 Parts of Me, Parts of You . . . 9
5.4 Curveball (CryptoHack) . . . 10
5.5 Let's Decrypt (CryptoHack) . . . 10

6 Commitments and Zero-Knowledge **10**
6.1 Trapdoor Backdoor . . . 10
6.2 How to Steal an Election . . . 11

7 Protocol Composition **11**
7.1 Megalomaniac 1 ⋆ (CryptoHack) . . . 11

1 Randomness

1.1 "It is truly random, I promise!"

Intel published a cryptography library with the following C++ code snippet:

```
1 static void rand32u(std::vector<Ipp32u>& addr) {
2   std::random_device dev;
3   std::mt19937 rng(dev());
4   std::uniform_int_distribution<std::mt19937::result_type> dist(0, UINT_MAX);
5   for (auto& x : addr)
6     x = (dist(rng) << 16) + dist(rng);
7 }
```

Question 1. Give a high-level explanation of each line of code.

Question 2. This code was used to generate cryptographic keys, which are supposed to be of 128 bits entropy. However, there are two *catastrophic failures* in this snippet. What is wrong?

Question 3. Describe (in words and/or pseudocode) how to fix this.

1.2 The Next of Your Kind

Let $\mathsf{nextPrime}$ take as input an integer x and output the smallest prime p such that $p > x$. Let the RSA-3072 key generation procedure be implemented as follows using a secure true random number generator (TRNG):

1. Securely sample a 1536 bit integer x using a TRNG.
2. Compute the first prime $p = \mathsf{nextPrime}(x)$.
3. Compute the second prime $q = \mathsf{nextPrime}(p)$.
4. Output the RSA-3072 modulus $n = p \cdot q$.

Question 1. Describe an efficient attack against a RSA-3072 modulus n computed using the above procedure.

Question 2. How can we update the procedure to generate a key securely?

1.3 This Destroyes the Schnorr Cryptosystem

Let $\mathbb{G}_p$ be a group of prime order p and let g be a generator for $\mathbb{G}_p$. Denote by pp the public parameters $(\mathbb{G}_p, g, p)$. Let the secret key $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$ be sampled uniformly at random, and let the public key be $\mathsf{pk} = g^{\mathsf{sk}}$, where pk is publicly available. Let H be a cryptographic hash function that outputs elements in $\mathbb{Z}_p$. The Schnorr signature of a message m is computed as:

1. Sample uniformly random $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute commitment $R = g^r$.
2. Compute the output hashed challenge $c = \mathsf{H}(\mathsf{pp}, \mathsf{pk}, m, R)$.
3. Compute the response $z = r - c \cdot \mathsf{sk} \mod p$. Output $\sigma = (c, z)$.

To verify the signature, one computes $R' = g^z \cdot \mathsf{pk}^c$ and checks if challenge $c \stackrel{?}{=} \mathsf{H}(\mathsf{pp}, \mathsf{pk}, m, R')$. If correct, one accepts and otherwise rejects.

We consider the signature scheme to be broken if an adversary is able to extract the secret key or forge signatures without knowing the secret key.

Question 1. How can we break the Schnorr signature scheme if the key sk is sampled using a low-entropy randomness source? How can we break it if the randomness r is sampled using a low-entropy randomness source?

Question 2. How can we break the Schnorr signature scheme if randomness r is re-used to produce signatures on different messages m and m'?

Question 3. How can we create a valid Schnorr signature without knowing sk if a weak hash function H outputs easily predictable challenges c?

1.4 ElGusto ElGamal

Let $\mathbb{G}_p$ be a group of prime order p and let g be a generator for $\mathbb{G}_p$. Let the secret key $\mathsf{sk} \leftarrow\!\!\$\, \mathbb{Z}_p$ be sampled uniformly at random and let the public key be computed as $\mathsf{pk} = g^{\mathsf{sk}}$, where pk is publicly available. The ElGamal encryption and decryption of a message $m \in \mathbb{G}_p$ is computed as:

Enc: Sample a random $x \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute $X = g^x$ and $Y = \mathsf{pk}^x \cdot m$.

Dec: Decrypt the ciphertext (X, Y) to get the messages as $m = Y \cdot X^{-\mathsf{sk}}$.

We denote $\mathsf{ctx} = (X, Y)$ and consider the encryption scheme to be broken if an adversary is able to extract the secret key or can learn something about the encrypted messages (breaking the IND-CPA security of ElGamal).

Question 1. How can we break the ElGamal encryption scheme if the key sk is sampled using a low-entropy randomness source? How can we break it if the randomness x is sampled using a low-entropy randomness source?

Question 2. What can we learn from two ElGamal ciphertexts if the same randomness x is used to encrypt two different messages m and m'?

1.5 Ron was Wrong, Whit is Right (CryptoHack)

In this challenge, we get a bunch of public RSA keys. Can we decrypt any of the messages?

Hint: There is seemingly little wrong with the challenge generation file. However, a quick Google search might provide useful.

Question. Go to cryptohack.org, and find the challenge **Ron was wrong, Whit is right** in the **RSA** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

1.6 No Random, No Bias (CryptoHack)

Try your hands on a famous attack we've covered in the lectures! Except, this time, we're using deterministic signatures, to be sure there's no bias in the randomness. Just don't use the solution to steal Bitcoins and become a cyber-criminal... or do, I don't really care...

Hint (ROT13): Abgvpr gung gur bhgchg fvmr bs FUN-1 vf 160 ovgf. Guhf, gur fbyhgvba gb guvf punyyratr jnf pbirerq va gur RPQFN yrpgher.

Question. Go to cryptohack.org, and find the challenge **No Random, No Bias** in the **Elliptic Curves** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

1.7 Lo-Hi Card Game (CryptoHack)

A Casino is using a homemade PRNG for shuffling their decks. Can we use this to pull off a great heist?

Hint: This challenge (and many future challenges) features interaction with a remote server. If you are using Python to solve this challenge, a good option for easy socket interaction is using the library pwntools.

Question. Go to cryptohack.org, and find the challenge **Lo-Hi Card Game** in the **Misc** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

1.8 Trust Games ⋆ (CryptoHack)

Given that we stole all their money, it would only be fair if we would test their new PRNG pro bono.

Question. Find the challenge **Trust Games** in the **Misc** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

1.9 Prime and Prejudice ⋆ (CryptoHack)

Can we construct a composite number that the Miller-Rabin test marks as a prime?

Question. Find the challenge **Prime and Prejudice** in the **Mathematics** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

1.10 RSA vs. RNG ⋆ (CryptoHack)

Try to break this poorly generated RSA key.

Question. Find the challenge **RSA vs. RNG** in the **Misc** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

2 Legacy Crypto

2.1 Export Grade (CryptoHack)

Alice and Bob are both supporting lots of parameters for nice, backward compatibility! You've MITM'd their communication. Can we use this to recover the flag?

Hint 1: We know that the flag was encrypted with the following code:

```python
import hashlib, os
from Crypto.Cipher import AES

def encrypt_flag(FLAG, shared_secret: int):
    # Derive AES key from shared secret
    sha1 = hashlib.sha1()
    sha1.update(str(shared_secret).encode('ascii'))
    key = sha1.digest()[:16]
    # Encrypt flag
    iv = os.urandom(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(FLAG)
    # Prepare data to send
    data = {}
    data['iv'] = iv.hex()
    data['encrypted_flag'] = ciphertext.hex()
    return data
```

Hint 2 (ROT13): Fntrzngu pna fbyir qvfpergr ybtf va snveyl ovt svryqf irel dhvpxyl. Hfr ‘$S = TS(c)$‘ gb vafgnagvngr n svavgr svryq, gura ‘$ybt(S(l), S(t))$‘ gb pbzchgr k fhpu gung $t^k = l$

Question. Go to cryptohack.org, and find the challenge **Export Grade** in the **Diffie-Hellman** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

2.2 Oh SNAP! (CryptoHack)

Can we break a classic cipher used for years, famously breaking one of Kerckhoffs' principles?

Hint (ROT13): Ybbx sbe vzcyrzragngvbaf bs gur ***Syhuere, Znagva naq Funzve nggnpx*** *bayvar.*

Question. Go to cryptohack.org, and find the challenge **Oh SNAP!** in the **Symmetric Ciphers** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

2.3 Nothing up my Sleeve ⋆ (CryptoHack)

The casino is back yet again. This time using a real-world, provably secure PRNG! Surely, we can't swindle them yet again??

Question. Find the challenge **Nothing up my Sleeve** in the **Misc** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

2.4 MOVing Problems ⋆ (CryptoHack)

The famous MOV-attack prevents the use of supersingular elliptic curves in discrete-log based systems. Luckily, this curve is ordinary, so there should be no problems.

Question. Find the challenge **MOVing Problems** in the **Elliptic Curves** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

3 Padding Oracles

3.1 Null or Never (CryptoHack)

RSA is padded here precisely to avoid textbook attacks. Can we still break it?

Question. Go to cryptohack.org, and find the challenge **Null or Never** in the **RSA** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

3.2 Paper Plane (CryptoHack)

Can we solve the following, using what we have learned so far?

Question. Go to cryptohack.org, and find the challenge **Paper Plane** in the **Symmetric Ciphers** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

3.3 Endless Emails ⋆ (CryptoHack)

We close with a classic illustration of what goes wrong when RSA is used with no padding at all.

Question. Go to cryptohack.org, and find the challenge **Endless Emails** in the **RSA** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

4 Quantum-Safe Crypto

4.1 Noisy Withdrawal

Let q be an odd prime, n a dimension, and β a small noise bound. Matrices and vectors are written in bold, so that $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the inner product of two vectors while m and β are plain integers. Consider the following toy scheme for encrypting a single bit, a simplified version of ML-KEM (CRYSTALS-Kyber):

KGen: Sample $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times n}$ uniformly at random and short vectors $\mathbf{s}, \mathbf{e} \leftarrow\!\!\$\ \{-\beta, \ldots, \beta\}^n$. Compute $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \mod q$ and output $\mathsf{pk} = (\mathbf{A}, \mathbf{b})$ and $\mathsf{sk} = \mathbf{s}$.

Enc: To encrypt $m \in \{0, 1\}$, sample a short $\mathbf{r} \leftarrow\!\!\$\ \{-\beta, \ldots, \beta\}^n$ together with fresh noise $\mathbf{e}_1 \leftarrow\!\!\$\ \{-\beta, \ldots, \beta\}^n$ and $e_2 \leftarrow\!\!\$\ \{-\beta, \ldots, \beta\}$, and output the ciphertext

$$\mathsf{ctx} = (\mathbf{u}, v) = \left(\mathbf{A}^{\mathsf{T}}\mathbf{r} + \mathbf{e}_1,\ \langle \mathbf{b}, \mathbf{r} \rangle + e_2 + m \cdot \left\lfloor \frac{q}{2} \right\rceil \right) \mod q.$$

Dec: Compute $w \equiv v - \langle \mathbf{u}, \mathbf{s} \rangle \mod q$ and let w denote its representative in $(-q/2, q/2]$. Output $m = 0$ if $|w| < q/4$, and $m = 1$ otherwise.

The vectors $\mathbf{r}, \mathbf{e}_1$ and the integer e_2 are sampled afresh for every encryption, whereas the key material $\mathbf{s}, \mathbf{e}$ is fixed once and for all. The security of the scheme relies on the Learning With Errors (LWE) assumption: given $\mathbf{A}$ and $\mathbf{A}\mathbf{s} + \mathbf{e}$ with a *small* error $\mathbf{e}$, it is hard to recover $\mathbf{s}$, and the pair is moreover indistinguishable from a uniformly random one.

Question 1. Assume that the implementation by mistake omits the encryption noise, so that $\mathbf{e}_1 = \mathbf{0}$ and $e_2 = 0$, while the public key is generated correctly. Show that an adversary who

observes ciphertexts can then decrypt them without knowing the secret key. What consequences would it have if instead $\mathbf{e} = \mathbf{0}$ in KGen?

Question 2. Show that decryption from w is correct and derive the resulting condition on β, n and q. Discuss the impact of adjusting β on correctness and security of the encryption scheme.

Question 3. Assume that the implementation by mistake caches the randomness $\mathbf{r}$ and re-uses it to encrypt two messages m and m' under the same public key, while the noise is still sampled afresh. What can an adversary learn from the two ciphertexts?

4.2 No Second Thoughts

Lattice signatures such as ML-DSA (CRYSTALS-Dilithium) follow the *Fiat–Shamir with aborts* paradigm: the signer masks the secret with fresh randomness and restarts whenever the masked value falls outside a fixed range. Consider the following toy version, where q is a prime, $\mathbf{A}$ is a $k \times \ell$ matrix with $\ell > k$[1], the parameters satisfy $\tau < \gamma < q$, and $\|\mathbf{x}\|_\infty$ denotes the largest absolute value of the entries of a vector $\mathbf{x}$:

KGen: Sample $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{k\times\ell}$ and a binary secret $\mathbf{s} \leftarrow\!\!\$\ \{0,1\}^\ell$, and compute $\mathbf{t} = \mathbf{A}\mathbf{s} \mod q$. Output $\mathsf{pk} = (\mathbf{A}, \mathbf{t})$ and $\mathsf{sk} = \mathbf{s}$.

Sign: To sign a message m, sample a mask $\mathbf{y} \leftarrow\!\!\$\ \{-\gamma, \ldots, \gamma\}^\ell$, compute $\mathbf{w} = \mathbf{A}\mathbf{y} \mod q$ and the challenge $c = \mathsf{H}(\mathbf{t} \,\|\, \mathbf{w} \,\|\, m) \in \{1, \ldots, \tau\}$, and set $\mathbf{z} = \mathbf{y} + c \cdot \mathbf{s}$ over the integers. If $\|\mathbf{z}\|_\infty > \gamma - \tau$, discard everything and restart with a fresh $\mathbf{y}$; otherwise output $\sigma = (c, \mathbf{z})$.

Vf: Accept $\sigma = (c, \mathbf{z})$ if $\|\mathbf{z}\|_\infty \le \gamma - \tau$ and $c = \mathsf{H}(\mathbf{t} \,\|\, \mathbf{A}\mathbf{z} - c \cdot \mathbf{t} \mod q \,\|\, m)$.

Note that the entries of $\mathbf{z}$ are small integers and are transmitted as such, rather than reduced modulo q. Verification is correct because $\mathbf{A}\mathbf{z} - c \cdot \mathbf{t} = \mathbf{A}(\mathbf{y} + c\mathbf{s}) - c\mathbf{A}\mathbf{s} = \mathbf{A}\mathbf{y} = \mathbf{w}$, so the verifier recomputes the same $\mathbf{w}$ as the signer. Since $\mathbf{s}$ is binary and $c \le \tau$, every coordinate of $c \cdot \mathbf{s}$ lies in $\{0, c\}$ and is at most τ, which is where the acceptance bound $\gamma - \tau$ comes from.

Question 1. An implementation drops the restart and simply outputs the first $(c, \mathbf{z})$ it computes. Show that the published signatures then reveal coordinates of the binary secret $\mathbf{s}$. What does this mean for the security of the scheme in general, e.g., if the secret is sampled from a wider distribution than binary?

Question 2. Since $\mathbf{w} = \mathbf{A}\mathbf{z} - c \cdot \mathbf{t}$ can be recomputed from a signature, an adversary can search a collection of signatures for two of them, on different messages, that were produced from the same $\mathbf{w}$. Explain how such a pair can occur and how it breaks the signature scheme.

5 Crypto API Failures

5.1 Fool Me Once, Fool Me Twice

Let the Schnorr signature scheme be defined as earlier but with a slight change: the randomness r is not sampled randomly but deterministically computed as the hash of the secret key and the message, i.e., $r = \mathsf{H}(\mathsf{sk}, m)$.

Let Sign be an API where a client inputs an identity id, a message m and a public key $\mathsf{pk}_{\mathsf{id}}$ and the server uses the secret key $\mathsf{sk}_{\mathsf{id}}$ corresponding to id (e.g. stored in a hardware security module) and output a signature σ on m.

[1] The matrix has to be wide: for $\ell \le k$ the system $\mathbf{t} = \mathbf{A}\mathbf{s}$ is over-determined and Gaussian elimination returns $\mathbf{s}$ outright, whereas for $\ell > k$ it has many solutions and finding a *short* one is the inhomogeneous SIS problem, which is believed to be hard. Real schemes such as ML-DSA instead publish $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ for two short vectors, which is an LWE sample.

Question 1. How can malicious clients use this API to extract secret keys?

Question 2. How can we protect this signing API against such attacks?

5.2 Faulty RSA Bites the Dust

Let (n, e) be a public RSA signature verification key and (n, e') a public RSA encryption key for the same user, where $n = p \cdot q$ for secret prime numbers p, q and corresponding secret signing key d and decryption key d'.

Assume that the signing API Sign is implemented in a faulty way so that the signing key d leaks to malicious clients.

Question 1. How can the knowledge of the signing key d be used to decrypt messages encrypted using the public encryption key (n, e')?

Assume now that the leakage in Sign be fixed so that d is stored securely. Let μ be a secure padding function. The RSA signature is often computed using the Chinese Remainder Theorem in the following way:

1. Compute $d_p \equiv d \mod (p-1)$ and $d_q \equiv d \mod (q-1)$.
2. Compute a such that $a \equiv 1 \mod p$ and $a \equiv 0 \mod q$.
3. Compute b such that $b \equiv 0 \mod p$ and $b \equiv 1 \mod q$.
4. Compute $\sigma_p \equiv \mu(m)^{d_p} \mod p$ and $\sigma_q \equiv \mu(m)^{d_q} \mod q$.
5. Output the signature $\sigma = a \cdot \sigma_p + b \cdot \sigma_q \mod n$.

This is more efficient than computing $\mu(m)^d \mod n$ directly since p and q are much smaller than n and (d_p, d_q, a, b) can be pre-computed and stored for later use. We can verify the signature as following: $\mu(m) \stackrel{?}{\equiv} \sigma^e \mod n$.

Question 2. Assume that there is a bug in the implementation so that $\sigma_p \equiv \mu(m)^{d_p} \mod p$ but $\sigma_q \not\equiv \mu(m)^{d_q} \mod q$. Show how the faulty signature σ, where $\mu(m) \not\equiv \sigma^e \mod n$, can be used to factor n.

Question 3. What are possible ways of avoiding the above RSA issues?

5.3 Parts of Me, Parts of You

Let $E_{a,b} : y^2 = x^3 + a \cdot x + b$ be an elliptic curve over a finite field $\mathbb{F}_p$ of prime order p where $a, b \in \mathbb{F}_p$ and the elliptic curve group $E_{a,b}(\mathbb{F}_p)$ consists of the point at infinity $\mathcal{O}$ and all pairs $(x, y) \in \mathbb{F}_p \times \mathbb{F}_p$ that satisfy the curve equation of $E_{a,b}$. Denote the number of points in $E_{a,b}(\mathbb{F}_p)$ by $\eta_{a,b}$ and note that $\eta_{a,b}$ does not necessarily have to be a prime number.

Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E_{a,b}(\mathbb{F}_p)$, then we compute the sum $P + Q$ in the following way:

1. If $P = \mathcal{O}$, output Q. If $Q = \mathcal{O}$, output P.
2. If $x_1 = x_2$ and $y_2 = -y_1$, then output $\mathcal{O}$.

3. Otherwise, let $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = -y_1 - \lambda \cdot (x_3 - x_1)$, where

$$\lambda = \begin{cases} \frac{3x_1^2 + a}{2y_1} & \text{if } P = Q \\ \frac{y_1 - y_2}{x_1 - x_2} & \text{otherwise,} \end{cases}$$

and output $R = (x_3, y_3)$.

Let q be a large prime such that $\eta_{a,b} = q \cdot h$, then $E_{a,b}(\mathbb{F}_p)$ has a subgroup $\mathbb{G}_q$ of order q such that $q \cdot P = \mathcal{O}$ if P is in $\mathbb{G}_q$. Here, h is called the co-factor, and is often a very small value compared to q.

The *double-and-add* algorithm can be used to efficiently compute the scalar multiplication $Q = s \cdot P$ for $s \in \mathbb{Z}_q$ and $P \in \mathbb{G}_q$ in at most $2 \log_2 q$ additions. The discrete logarithm problem says it is hard to find s given P and Q.

Note that for each choice of a and b we get a new elliptic curve group of different size $\eta_{a,b}$ where $\eta_{a,b}$ is roughly of the size p. We can efficiently compute $\eta_{a,b}$ using Schoof's algorithm.

Let sMult be an API that takes a point P as input and outputs a point $Q = s \cdot P$ for a fixed and secret value s of high entropy.

Question 1. Assume that sMult checks that P is on the curve $E_{a,b}$ but forget to check if it is the in the correct subgroup $\mathbb{G}_q$. How can a malicious client learn something about s in a API query?

Question 2. Describe an attack that completely breaks the sMult API when it is not checking if P is on the curve $E_{a,b}$ and explain why it works.

5.4 Curveball (CryptoHack)

Can we prove that we own a public ECDSA key, by giving the corresponding private key? This attack is so stupid, it could not have occurred in the real world, right? RIGHT??!

Recommended listening while solving: Aretha Franklin - Chain of Fools

Question. Go to cryptohack.org, and find the challenge **Curveball** in the **Elliptic Curves** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

5.5 Let's Decrypt (CryptoHack)

In this challenge, we should make a given RSA signature verify a different message. Luckily, the RSA signing operation is a bijection from $(\mathbb{Z}/n\mathbb{Z})^\times$ to itself, so this should be impossible.

Question. Go to cryptohack.org, and find the challenge **Let's Decrypt** in the **RSA** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

6 Commitments and Zero-Knowledge

6.1 Trapdoor Backdoor

Let $\mathbb{G}_p$ be a group of prime order p and let g and h be independent generators for $\mathbb{G}_p$. Let m be a message and w be uniform randomness, both elements in $\mathbb{Z}_p$. A Pedersen commitment is computed as $\mathsf{com} = g^m h^w$.

A commitment is *hiding* if it is hard to decide if com is a commitment to m or if com is sampled uniformly at random from $\mathbb{G}_p$. A commitment is *binding* if it is hard to find two valid openings (m, w) and (m', w') such that $\mathsf{com} = g^m h^w = g^{m'} h^{w'}$ and $m \neq m'$.

Question 1. Provide some simple or high-level arguments to explain why the Pedersen commitment is both hiding and binding. For each of the two properties, state whether it holds *unconditionally* or only *computationally*, and explain what that difference means for an adversary with unbounded computing power.

Question 2. Let $g = h^t$. Show how the knowledge of t can break binding.

Question 3. How can we ensure that g and h are independently sampled so that no one knows a value t such that $g = h^t$?

6.2 How to Steal an Election

Let the ElGamal encryption scheme be defined as earlier. A prover has the secret key sk corresponding to the public key pk and wants to prove in zero-knowledge that m is the correct decryption of a ciphertext $\mathsf{ctx} = (X, Y)$.

This can be used in an electronic voting scheme where we want to prove that the tally is correct without making the decryption key publicly available.

The decryption proof is computed as follows, where $\mathsf{pp} = (\mathbb{G}_p, g, p, \mathsf{pk})$ is public information (the public key pk is fixed but the ciphertext ctx is not):

1. Compute $T = X^{\mathsf{sk}}$, such that the decrypted message is $m = Y \cdot T^{-1}$.
2. Sample a uniformly random $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $R = g^r$ and $S = X^r$.
3. Compute the hashed challenge as $c = \mathsf{H}(\mathsf{pp}, X, Y, R, S, T)$.
4. Compute the response $z = r - c \cdot \mathsf{sk} \mod p$. Define proof $\pi = (T, c, z)$.
5. The prover outputs ctx and π for anyone to verify the correctness.

To verify the proof π one computes $R' = g^z \cdot \mathsf{pk}^c$ and $S' = X^z \cdot T^c$ and checks if $c = \mathsf{H}(\mathsf{pp}, X, Y, R', S', T)$. If correct, one outputs $m = Y \cdot T^{-1}$ and otherwise rejects. The proof is similar to Schnorr signatures, proving that $\log_g \mathsf{pk} = \log_X T$, and then $m = Y \cdot T^{-1}$ is the correctly decrypted message.

In this problem we assume that the malicious prover knows the secret key sk, but want to provide a seemingly valid proof that the ElGamal ciphertext $\mathsf{ctx} = (X, Y)$ decrypts to a different message m' than the real plaintext m.

Question 1. Assume $c = \mathsf{H}(\mathsf{pp}, Y, R, S, T)$, where the ciphertext component X is not included. How can a malicious prover change the ciphertext and create an accepting proof that it decrypts to a chosen message $m' \neq m$?

Question 2. Assume that $c = \mathsf{H}(\mathsf{pp}, X, Y, R, S)$, where the decryption component T is not included. How can a malicious prover create an accepting proof where the given ciphertext decrypts to a random message $m' \neq m$?

7 Protocol Composition

7.1 Megalomaniac 1 ⋆ (CryptoHack)

The first challenge, introducing a simplified Mega vulnerability.

Question. Go to cryptohack.org, and find the challenge **Megalomaniac 1** in the **Crypto on the Web** category. Solve the challenge and give the flag. How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.