TTM4205: Weekly Problems Fall 2023

Tjerand Silde and Jonathan Komada Eriksen

{tjerand.silde, jonathan.k.eriksen}@ntnu.no

Overview

This is the first of two assignments in the course TTM4205 Secure Cryptographic Implementations in the fall semester of 2023. More details about the course can be found at http://ttm4205.iik.ntnu.no. This assignment has to be solved *individually* (except side-channel attacks), and the solutions must be *your own*. It is, however, *allowed* to discuss the problems with other students and ask for hints or pointers from the course staff.

The assignment contains problems related to each of the main topics from the lectures. Some problems can be solved with pen and paper and/or code, and other solutions has to be documented in other ways. A portion of the problems is taken from https://cryptohack.org and ChipWhisperer-tutorials from https://github.com/newaetech/chipwhisperer-jupyter.

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. It is *allowed* to rely on external resources; however, these resources must be clearly referred to. Otherwise, it will be considered cheating; see https://i.ntnu.no/wiki/-/wiki/English/Cheating+on+exams. You are *allowed* to use a variety of tools to improve the writing quality of your solutions, e.g., Grammarly at https://grammarly.com.

All submissions must be written in LaTeX, and the source of this document is available at https://www.overleaf.com/read/gcbmcmffrhyk.

This assignment counts for at most 40 points, and each topic is marked with how many points it is worth, roughly estimating how much work is expected. Bonus problems are not expected to be solved but can give 1 additional points each to make up for missed points elsewhere in the assignment.

Submission deadline: December 1st, by email to tjerand.silde@ntnu.no.

Contents

1 Randomness (9 points)

1.1 "It is truly random, I promise!"

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

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

Question 1: Analyse it and give a high-level explanation of each code line.

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 code. What is wrong? (And what else is odd?)

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

1.2 The Next of Your Kind

Let `nextPrime` be a function that takes as input an integer x and outputs the smallest prime p such that $p \geq x$. Let an RSA-3072 key generation be implemented using a secure random generator as follows:

1. Securely sample a 128 bit entropy seed s.
2. Expand s to a random 1536-bit integer x.
3. Compute the first prime $p = \texttt{nextPrime}(x)$.
4. Compute the second prime $q = \texttt{nextPrime}(p)$.
5. Output the RSA-3072 modulus $n = p \cdot q$.

Question 1: How can you potentially break a given RSA modulus n when you know it was computed using this procedure? Are you likely to succeed?

Question 2: How can you update the procedure to generate a secure key?

1.3 This Destroyes the Schnorr Cryptosystem

Let $\mathbb{G}$ be a group of prime order p and let g be a generator for $\mathbb{G}$. Denote by pp the public parameters $(\mathbb{G}, g, p)$. 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 made

publicly available. Let H be a cryptographic hash function that outputs elements in $\mathbb{Z}_p$. The Schnorr signature of message m is computed as:

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

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

Question 1: How do you break the Schnorr signature scheme if key sk or randomness r is sampled using a low-entropy randomness source?

Question 2: How do you break the Schnorr signature scheme if randomness r is re-used to produce signatures on two different messages m and m'?

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

Question 4: What are possible ways to mitigate the above weaknesses?

1.4 ElGusto ElGamal

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

Question 1: How do you break the ElGamal encryption scheme if key sk or randomness x is sampled using a low-entropy randomness source?

Question 2: What can you 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

In this challenge, you get a bunch of public RSA keys. Can you 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 1: Go to `https://cryptohack.org`, and find the challenge **Ron was wrong, Whit is right** in the **RSA** category. Solve the challenge and give the flag.

Question 2: 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

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...

Question 1: Go to `https://cryptohack.org`, and find the challenge **No Random, No Bias** in the **Elliptic Curve** category. Solve the challenge and give the flag.

Question 2: 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

A Casino is using a homemade PRNG for shuffling their decks. Can you 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 1: Go to `https://cryptohack.org`, and find the challenge **Lo-Hi Card Game** in the **Misc** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

1.8 Bonus Problems

1.8.1 Trust Games

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

Question: Find the challenge **Trust Games** in the **Misc** category. Solve the challenge and give the flag. Write a short write-up of your solution.

1.8.2 Prime and Prejudice

Can you 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. Write a short write-up of your solution.

1.8.3 RSA vs. RNG

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. Write a short write-up of your solution.

2 Legacy Crypto (4 points)

2.1 Export Grade

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

Hint: From an earlier challenge, you know that the flag was encrypted with the following code:

```
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
```

Question 1: Go to https://cryptohack.org, and find the challenge **Export Grade** in the **Diffie-Hellman** category. Solve the challenge and give the flag.

Question 2: 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!

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

Question 1: Go to https://cryptohack.org, and find the challenge **Oh SNAP!** in the **Symmetric Ciphers** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

2.3 Bonus Problems

2.3.1 Nothing up my Sleeve

The casino is back yet again. This time using a real-world, provably secure PRNG! Surely, you 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. Write a short write-up of your solution.

2.3.2 MOVing Problems

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 Curve** category. Solve the challenge and give the flag. Give a short write-up of your solution.

3 Side-Channel Attacks (10 points)

In this section, you will work through several detailed tutorials for conducting side-channel attacks (power analysis and fault injection) against modern schemes like AES and RSA using the ChipWhisperer platform. You will perform the experiments in pairs, but you must write down your answers individually. We will provide each pair with a ChipWhisperer Level 1 Kit; see https://www.newae.com/products/NAE-SCAPACK-L1.

Instructions for ChipWhisperer installation, set up, and use; see https://github.com/tjesi/TTM4205/blob/main/Chipwhisperer-installation-set-up-and-use.ipynb.

It is not unusual for errors to happen when conducting real-world experiments. For common errors and possible solutions, see https://github.c

om/tjesi/TTM4205/blob/main/Common-errors-and-possible-solutions-with-chipwhisperer.ipynb.

In each of the following tutorials, you should provide a short write-up, e.g., including a screenshot and/or a few lines of code and/or a paragraph describing what you have completed. The tutorials are available at https://github.com/newaetech/chipwhisperer-jupyter/tree/master/courses.

3.1 Introduction to Power Analysis

Complete the following tutorials in the folder "sca101": Setup and Lab 2-1 to 4-3 (not Lab 5-1 and Lab 6-4). (You can also use a ChipWhisperer Nano.)

3.2 More Advanced Power Analysis

Complete the following tutorials in the folder "sca201": Lab 1-1.

3.3 Introduction to Fault Injection

Complete the following tutorials in the folder "fault101": Lab 1-1 to 1-4.

3.4 More Advanced Fault Injection

Complete the following tutorials in the folder "fault201": Lab 1-3 and 2-1.

3.5 Bonus Problems

3.5.1 CPA in Practice and Jittery Trigging

Complete the following tutorials in the folder "sca101" Lab 5-1 and 6-4.

3.5.2 AES256 Bootloader Attack and Reverse Engineering

Complete the following tutorials in the folder "sca201" Lab 3-1.

3.5.3 Voltage Glitching

Complete the following tutorials in the folder "fault101" Lab 2-1 to 2-3.

4 Protocols APIs (7 points)

4.1 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., $r = H(\mathsf{sk}, m)$.

Let `Sign` be an API where a client inputs an identity id, a message m and a public key pk_{id} and the server uses the secret key sk_{id} corresponding to id (e.g. stored in a hardware security module) and output a signature σ.

Question 1: How can a malicious client use this API to break the scheme?

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

4.2 It is All About Sharing

In many applications, we want to distribute the trust by sharing parts of a key with many parties so that a threshold (subset) of the parties must work together to apply the key, e.g. a threshold signature on a message or a threshold decryption of a ciphertext.

Let n be the total number of parties, and let t be the threshold size. This means that t parties must work together, but if an adversary corrupts less than t parties, then the secret is still secure.

A fundamental building block in these schemes is the Shamir secret sharing scheme. Let p be a prime and s be a secret element in $\mathbb{Z}_p$ known to a trusted dealer. Let `Share` be an API. The protocol works as follows:

1. The dealer samples $t-1$ elements a_i uniformly at random from $\mathbb{Z}_p$.
2. The dealer define the polynomial $f(x) = s + a_0 x + \cdots + a_{t-1}x^{t-1}$.
3. On input an integer $id \in \mathbb{N}$ to `Share`, the dealer checks that $id \neq 0$.
4. The dealer returns the value $f(id)$ from the `Share` API to the client.

Here, it is essential that we do not accept $id = 0$ since $f(0) = s$. If t parties work together, they can reconstruct s by Lagrange Interpolation.

Question 1: How can a malicious party use this API to break the scheme?

Question 2: How can the trusted dealer protect against such attacks?

4.3 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 the clients.

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

Let 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 Reminder Theorem in the following way:

1. Compute $d_p \equiv d \mod p$ and $d_q \equiv d \mod q$.
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 a more efficient computation 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: $\mu(m) \stackrel{?}{\equiv} \sigma^e \mod n$.

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 these above RSA issues?

4.4 Parts of Me, Parts of You

Let $E : y^2 = x^3 + a \cdot x + b$ be an elliptic curve over a finite field $\mathbb{F}_p$ of prime order p where $a, b \in \mathbb{F}_p$ and that the elliptic curve group $E(\mathbb{F}_p)$ 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. Denote the number of points in $E(\mathbb{F}_p)$ by η and note that η does not necessarily have to be a prime number.

Give two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(\mathbb{F}_p)$, 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 $R = (x_3, y_3)$.

The *double-and-add* (resp. *square-and-multiply* for multiplicative notation) algorithm can then be used to efficiently compute the scalar multiplication $Q = s \cdot P$ for $s \in \mathbb{Z}_p$ and $P \in E(\mathbb{F}_p)$ in at most $2 \log_2 \eta$ additions. The discrete logarithm problem says it is hard to find s given P and Q.

Let q is a large prime and $\eta = q \cdot h$, then $E(\mathbb{F}_p)$ has a subgroup $\mathbb{G}_q$ of order q such that $q \cdot P = \mathcal{O}$ if P is in $\mathbb{G}_q$. h is called the co-factor. Note that for each choice of a, b we get a new elliptic curve of different size $\eta_{a,b}$. Assume that we have an API called `sMult` that takes a point P as input and outputs a point $Q = s \cdot P$ for a fixed and secret value s with 256 bits of entropy.

Question 1: Assume that `sMult` forget to check if $q \cdot P = \mathcal{O}$, and that h is a relatively small number or is a product of several small prime numbers. What kind of information about s can we learn from Q?

Question 2: Assume that `sMult` forget to check if P in on the curve E. How can a client adaptively learn something about s in each API query?

Question 3: How can we protect `sMult` from leaking information about s?

4.5 Curveball

Can you prove that you 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 1: Go to `https://cryptohack.org`, and find the challenge **Curveball** in the **Elliptic Curve** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

4.6 Let's Decrypt

In this challenge, you 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 1: Go to `https://cryptohack.org`, and find the challenge **Let's Decrypt** in the **RSA** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

4.7 Bonus Problems

4.7.1 Implementing `sMult` attack

Implement an attack that breaks the `sMult` API in "Parts of Me, Parts of You" when it does not check if the point P is on the curve. Explain your attack and provide the code.

5 Padding Oracles (4 points)

5.1 Endless Emails

Here is a classic example you might have seen before in earlier courses of what can go wrong when using RSA without padding.

Question 1: Go to https://cryptohack.org, and find the challenge **Endless Emails** in the **RSA** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

5.2 Null or Never

This time, we padded RSA precisely to avoid something similar to the previous attack. Can you still break it?

Question 1: Go to https://cryptohack.org, and find the challenge **Null or Never** in the **RSA** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

5.3 Bonus Problems

5.3.1 Paper Plane

Can you solve this, using what you have learned?

Question: Go to https://cryptohack.org and find the challenge **Paper Plane** in the **Symmetric Crypto** category. Solve the challenge and give the flag. Write a short write-up of your solution.

6 Commitments and Zero-Knowledge (4 points)

6.1 Trapdoor Backdoor

Let $\mathbb{G}$ be a group of prime order p and let g and h be independent generators for $\mathbb{G}$. 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 $\mathbb{G}$. A commitment is *binding* if it is hard to find to 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.

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

Question 3: How can we ensure that g and h are independently sampled?

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 the correct decryption of a ciphertext (X, Y) is m.

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 $\mathsf{pp} = (\mathcal{G}, g, p)$:

1. Compute $T = X^{\mathsf{sk}}$ and the encrypted message as $m = Y \cdot T^{-1}$.
2. Sample random $r \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute $R = g^r$ and $S = T^r$.
3. Compute the hashed challenge as $c = H(\mathsf{pp}, \mathsf{pk}, X, Y, R, S, T)$.
4. Compute the response $z = r - c \cdot \mathsf{sk}$. Output proof (T, c, z).

To verify the proof one computes $R' = g^z \cdot \mathsf{pk}^c$ and $S' = X^z \cdot T^c$ and checks if $c = H(\mathsf{pp}, \mathsf{pk}, X, Y, R, S, T)$. 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 encrypted message.

Question 1: Assume $c = H(\mathsf{pp}, \mathsf{pk}, X, Y, R, S)$. How can a malicious prover create an accepting proof that the given ciphertext decrypts to a random message $m' \neq m$?

Question 2: Assume $c = H(\mathsf{pp}, X, Y, R, S, T)$. How can a malicious prover change the public key and create an accepting proof that the given ciphertext decrypts to a chosen message $m' \neq m$ with respect to the new public key?

Question 3: Assume $c = H(\mathsf{pp}, \mathsf{pk}, X, R, S, T)$. How can a malicious prover change the ciphertext and create an accepting proof that it decrypts to a chosen message $m' \neq m$?

Question 4: Assume $c = H(\mathsf{pp}, \mathsf{pk}, Y, R, S, T)$. How can a malicious prover change the ciphertext and create an accepting proof that it decrypts to a chosen message $m' \neq m$?

7 Protocol Composition (2 points)

7.1 Megalomaniac 1:

The first challenge, introducing a simplified Mega vulnerability.

Question 1: Go to https://cryptohack.org, and find the challenge **Megalomaniac 1** in the **Crypto on the web** category. Solve the challenge and give the flag.

Question 2: How did you solve the challenge? Provide a short write-up, including the main mathematical concepts, and some relevant code snippets.

7.2 Bonus Problems

7.2.1 Megalomaniac 2:

The second challenge is also related to the Mega vulnerability.

Question: Go to https://cryptohack.org and find the challenge **Megalomaniac 2** in the **Crypto on the web** category. Solve the challenge and give the flag. Write a short write-up of your solution.

7.2.2 Megalomaniac 3:

The final challenge, forcing you to limit your number of queries to steal all the data!

Question: Go to https://cryptohack.org and find the challenge **Megalomaniac 3** in the **Crypto on the web** category. Solve the challenge and give the flag. Write a short write-up of your solution.