Emplacement du logo

Rapport de Projet : Cryptohack

Victor Bailleul, Seb

19 octobre 2025

Université Fictive de LaTeX
Année 2023-2024

Table des matières

1 Introduction

1.1 Contexte du projet

Ce rapport présente une série de challenges de cryptographie réalisés sur la plateforme CryptoHack. L'objectif est de documenter les approches méthodologiques et les solutions techniques que nous avons mis en oeuvre en binome pour résoudre les défis proposés. En préambule, il convient de situer le contexte de ce travail en présentant d'une part le rôle fondamental de la cryptographie en cybersécurité, et d'autre part l'intérêt des plateformes de type *Capture The Flag* (CTF) comme outil d'apprentissage.

1.2 Place de la cryptographie en cybersécurité

La cryptographie est une discipline scientifique qui constitue l'un des piliers de la sécurité des systèmes d'information. Son objet est de développer des techniques permettant de protéger l'information contre toute modification ou accès non autorisé. En cybersécurité, la cryptographie vise à garantir plusieurs principes de sécurité fondamentaux.

La confidentialité Ce principe assure que seules les entités autorisées puissent accéder aux données. L'outil principal pour atteindre cet objectif est le chiffrement, qui consiste à transformer une information (le texte clair) en une forme inintelligible (le texte chiffré) à l'aide d'une clé secrète.

L'intégrité Ce principe garantit que les données n'ont pas été altérées ou corrompues, que ce soit de manière accidentelle ou intentionnelle, durant leur stockage ou leur transmission. Les fonctions de hachage et les codes d'authentification de message (MAC) sont des mécanismes cryptographiques courants pour vérifier l'intégrité.

L'authenticité Ce principe permet de vérifier l'identité d'une entité (un utilisateur, un serveur, etc.). Les certificats numériques et les signatures numériques sont des exemples de techniques cryptographiques assurant l'authentification.

La non-répudiation Ce principe empêche une entité de nier avoir effectué une action, comme l'envoi d'un message ou la validation d'une transaction. La signature numérique est le mécanisme de base pour fournir cette garantie.

De la sécurisation des communications sur Internet (protocoles TLS/SSL) à la protection des données stockées, en passant par la sécurisation des transactions financières et la protection de la vie privée, les applications de la cryptographie sont omniprésentes et critiques. L'étude de ses mécanismes et de leurs implémentations est par conséquent essentielle pour tout praticien de la cybersécurité.

1.3 Les plateformes de CTF dans l'apprentissage

Les challenges de type *Capture The Flag* (CTF) sont des exercices de cybersécurité offensifs et/ou défensifs. Les participants doivent résoudre des épreuves pour trouver une

chaîne de caractères secrète, appelée *flag*, cachée dans un système, un fichier ou encore un chiffré.

Les plateformes de CTF comme *root-me*, *TryHackMe* ou encore *CryptoHack* sont des environnements d'apprentissage pratiques et contrôlés où les utilisateurs peuvent appliquer des connaissances théoriques à travers des exercices thématiques. Cette approche par la pratique favorise une compréhension approfondie des vulnérabilités et des techniques d'exploitation.

Ces plateformes couvrent un large éventail de domaines de la cybersécurité, tels que l'exploitation de binaires, la rétro-ingénierie (*reverse engineering*), l'analyse forensique (*digital forensics*), la sécurité des applications web et la cryptographie.

1.4 Présentation de CryptoHack

CryptoHack est une plateforme en ligne dédiée à l'apprentissage de la cryptographie moderne. Elle propose une série de challenges de type CTF qui permettent aux utilisateurs de se familiariser avec les principes, les algorithmes et les attaques cryptographiques. La plateforme est conçue pour être progressive, avec des défis de difficulté croissante.

Les challenges sur CryptoHack se concentrent sur l'identification et l'exploitation de failles dans des implémentations de protocoles et d'algorithmes cryptographiques largement utilisés, tels que AES, RSA ou Diffie-Hellman.

Chaque challenge est conçu pour illustrer un concept cryptographique spécifique ou une vulnérabilité connue. Les utilisateurs sont amenés à interagir avec des serveurs distants, à analyser du code source et à développer leurs propres scripts, principalement en Python, pour automatiser les attaques et récupérer les *flags*.

2 Gestion de projet

2.1 Répartition du travail

Avant de débuter notre travail sur les différents challenges de *CryptoHack*, nous avons pris le temps de planifier notre organisation, notamment en tenant compte de notre emploi du temps déjà chargé (plusieurs cours et contrôles continus à gérer en parallèle). Nous avons donc fixé une **date de début précise**, afin de disposer d'un cadre temporel clair et d'assurer une répartition équilibrée de la charge de travail.

Une fois cette date établie, nous avons défini des **objectifs initiaux** afin d'avoir une vision claire de notre progression. Dans un premier temps, nous avons convenu de nous concentrer entièrement sur le module *General*, dans le but de nous familiariser avec la plateforme *CryptoHack* et de comprendre le format des exercices proposés. Cette phase de découverte avait également pour but d'identifier les outils nécessaires à la résolution des énigmes (Python, bibliothèques de cryptographie, etc.) et d'évaluer le niveau de difficulté des différents types de challenges.

Pour structurer notre travail, nous avons mis en place une **répartition des tâches** claire. L'un de nous s'est concentré sur les modules *Encoding* et *XOR*, tandis que l'autre s'est chargé des parties *Mathematics* et *Data Formats*. Cette division nous a permis de progresser en parallèle tout en couvrant un large spectre de thématiques cryptographiques. Nous avons également fixé une première **deadline commune** pour la fin de cette phase, afin de pouvoir ensuite faire un point global sur l'avancement du projet.

2.2 Gestion d'un dépôt GitHub

Dès le début du projet, nous avons créé un **dépôt GitHub partagé**, servant de point central pour le suivi et la gestion de notre travail. Ce dépôt nous a permis d'organiser nos scripts de résolution, nos notes et les différents fichiers associés aux challenges. L'utilisation de GitHub s'est révélée particulièrement utile pour la **collaboration asynchrone**, notamment lorsque nos disponibilités ne coïncidaient pas.

Chaque membre disposait d'un accès complet au dépôt et pouvait le mettre à jour dès qu'il estimait qu'une contribution était suffisamment stable ou pertinente. Les commits étaient accompagnés de messages explicites décrivant les modifications apportées, ce qui facilitait la compréhension de l'évolution du projet.

2.3 Méthodologie GitHub

Afin d'assurer une organisation cohérente et efficace, nous avons adopté une **méthodologie de travail simple mais structurée** basée sur les bonnes pratiques Git.

Chaque fois qu'un challenge était résolu, le code correspondant était ajouté dans un dossier thématique (par exemple, *Encoding/*, *XOR/*, *Mathematics/*, etc.).

Nous avons également convenu d'un rythme de mise à jour régulier du dépôt : chacun pouvait pousser ses modifications après vérification, en veillant à ne pas écraser le travail de l'autre. Lorsque cela s'avérait nécessaire, nous communiquions directement pour fusionner ou réorganiser certaines branches, garantissant ainsi une cohérence globale dans la structure du projet.

Une fois les premiers modules terminés, nous avons commencé la rédaction du rapport sur LaTeX. Là encore, nous avons établi une **structure commune** afin d'uniformiser notre style et notre présentation. Nous avons fixé une nouvelle deadline pour la finalisation de cette partie, ce qui nous a permis de maintenir un rythme constant.

Par la suite, nous avons décidé d'ajouter deux modules supplémentaires à nos objectifs : *Diffie-Hellman* et *RSA*. Pour cette nouvelle étape, chacun s'est vu attribuer un module et a évalué, en fonction de son temps disponible, quels challenges étaient les plus pertinents à traiter. Cette approche flexible mais organisée nous a permis de continuer à avancer efficacement, tout en tenant compte de nos contraintes personnelles et académiques.

— En somme, cette gestion de projet s'est appuyée sur une **communication régulière**, une **planification claire** et une **utilisation rigoureuse de GitHub**. Cela nous a permis de progresser de manière structurée, d'assurer la cohérence de nos contributions et de produire un travail collectif de qualité.

3 General

3.1 Encoding

Cette première sous-partie de la catégorie General aborde les différentes méthodes de représentation de l'information, essentielles au transport et à l'échange de données. La maîtrise des conversions entre des formats comme le binaire, l'hexadécimal ou le Base64 constitue un prérequis indispensable pour aborder des défis cryptographiques plus complexes. Il est fondamental de bien distinguer l'encodage du chiffrement : le premier est une transformation de format, publique et réversible, qui ne vise pas à garantir la confidentialité, contrairement au second.

Nous avons décidé de présenter le dernier challenge de cette partie, nommé *Encoding challenge*.

3.1.1 Objectifs

L'objectif de ce challenge consiste à développer un script pour automatiser l'interaction avec un serveur distant de CryptoHack. Le processus implique la réception de données encodées selon diverses méthodes (Base64, hexadécimal, ROT13, BigInt, et UTF-8), leur décodage approprié, puis le renvoi de la valeur décodée au serveur.

Pour valider le challenge et obtenir le flag, il est impératif d'exécuter cette séquence de réception, décodage et renvoi avec succès cent fois consécutives. Cette contrainte requiert une solution entièrement automatisée, capable de gérer dynamiquement les différents types d'encodage rencontrés.

3.1.2 Méthode

Le challenge met à disposition un script partiel qui présente la manière d'envoyer et recevoir des données avec le serveur, ainsi que le script exécuté côté serveur pour vérifier les valeurs qui lui ont été transmises. Ces scripts nous ont permis de comprendre le format des données transmises.

La communication avec le serveur s'effectue via l'échange d'objets au format JSON. Pour chaque itération du challenge, le serveur envoie une requête structurée de la manière suivante :

```
{
    "type": "type_d_encodage",
    "encoded": "donnees_encodees"
}
```

Notre script doit alors analyser cette requête, appliquer la méthode de décodage appropriée, et renvoyer la solution au serveur sous le format JSON attendu :

```
{
    "decoded": "donnees_decodees"
}
```

Le script côté serveur vérifie alors la validité des données décodées, puis si cela est valide, crée un nouveau challenge. Si notre script résout cent challenges, la prochaine requête au serveur permettra d'afficher le *flag* dans la sortie standard.

3.1.3 Résultat

Pour résoudre le challenge, nous nous sommes donc appuyés sur les scripts python du challenge afin de développer un script de résolution (voir ...).

FIGURE 1 – Ceci est le schéma explicatif de notre méthode.

Après cent requêtes valides, le *flag* *crypto{3nc0d3_d3c0d3_3nc0d3}* a été obtenu, permettant la validation du challenge.

3.2 Xor (Lemur)

La deuxième sous-partie de la catégorie *General* est consacrée à l'opération XOR (ou exclusif), un concept fondamental en cryptographie constituant l'une des briques de base de nombreux algorithmes de chiffrement. Sa simplicité de mise en œuvre et ses propriétés mathématiques uniques en font un outil puissant pour manipuler l'information.

Comprendre le fonctionnement du XOR est une étape cruciale, car il se situe à la frontière entre l'opération logique et le chiffrement. L'une de ses propriétés essentielles est sa réversibilité : appliquer deux fois la même clé XOR à une donnée permet de retrouver la donnée originale ($A \oplus K \oplus K = A$). Cette caractéristique est au cœur de son utilisation dans des chiffrements à flux comme le One-Time Pad.

Nous avons décidé de présenter le challenge le plus représentatif de cette partie, nommé *Lemur XOR*.

3.2.1 Objectifs

Le but de ce challenge est de retrouver le *flag*, à partir de deux images fournies : `lemur.png` et `flag.png`. L'énoncé nous apprend que ces deux images ont été chiffrées avec l'opération XOR en utilisant la même clé secrète.

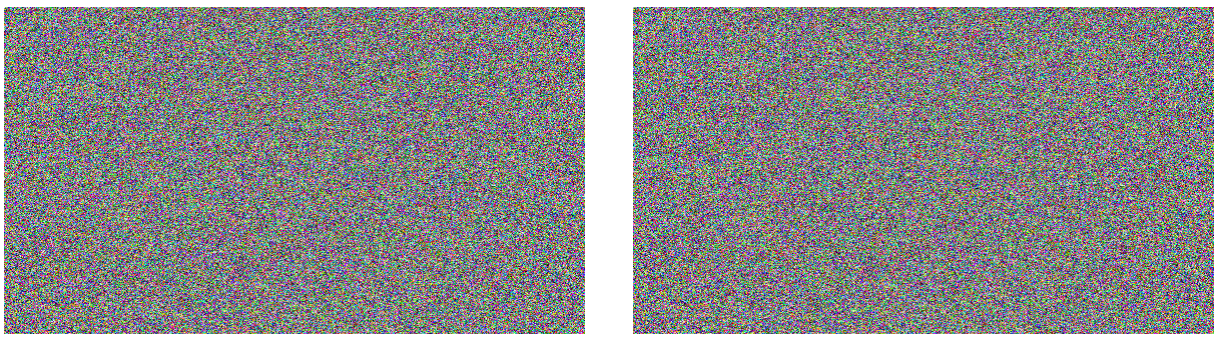

FIGURE 2 – Ceci est la légende commune pour les deux images.

Le principe de résolution repose sur le fait qu'appliquer deux fois un XOR avec la même clé annule l'opération. En effectuant un XOR entre les deux images chiffrées que

nous possédons, la clé secrète commune s'élimine, ne laissant que le résultat du XOR entre les deux images originales. C'est sur cette image combinée que le *flag* devrait devenir visible.

3.2.2 Méthode

Pour résoudre ce challenge, nous avons utilisé un script en Python avec la bibliothèque de manipulation d'images **Pillow**. Nous avons commencé par charger les deux images, `lemur.png` et `flag.png`. Conformément aux instructions, nous avons appliqué l'opération XOR pixel par pixel sur les valeurs de couleur RVB, et non sur les fichiers entiers.

Nous avons donc parcouru les deux images simultanément et calculé la nouvelle valeur de chaque pixel en effectuant un XOR sur ses composantes rouge, verte et bleue. Nous avons ensuite utilisé ces nouveaux pixels pour construire une image de sortie de mêmes dimensions que nous avons sauvegardée.

3.2.3 Résultat

FIGURE 3 – Ceci est le schéma explicatif de notre méthode.

3.3 Data formats

Le challenge s'appuie sur le principe de *Certificate Transparency* (CT), une mesure de sécurité imposée aux Autorités de Certification (CA) pour garantir la transparence dans la délivrance des certificats TLS.

Un certificat TLS (souvent appelé certificat SSL ou certificat numérique) est un document électronique qui remplit deux fonctions principales : il authentifie l'identité d'un site web ou d'un domaine auprès des clients (navigateurs, applications), et il permet d'établir une connexion chiffrée (TLS) entre le client et le serveur, garantissant la confidentialité et l'intégrité des données échangées.

Un *CT log* est une base de données publique, append-only (où l'on ne peut qu'ajouter des entrées) dans laquelle les certificats émis par les CA sont enregistrés. Aujourd'hui, les principales CA doivent publier (ou « soumettre ») chaque certificat qu'elles émettent dans au moins deux logs CT publics pour qu'il soit accepté et reconnu comme valide par les navigateurs modernes.

Ces logs sont audités et surveillés : chacun peut vérifier les entrées, détecter des certificats inattendus, ou vérifier la cohérence de la structure interne (par exemple via un arbre de Merkle) pour s'assurer qu'on ne cache pas d'entrées.

3.3.1 Objectifs

L'objectif de ce challenge est double. Premièrement, il s'agit de retrouver le sous-domaine de cryptohack.org qui utilise la même clé publique que celle fournie dans le fichier transparency.pem dans son certificat TLS. Deuxièmement, en visitant ce sous-domaine, il faut obtenir le flag. À travers ce challenge, les objectifs pédagogiques sont d'abord de comprendre le fonctionnement d'un certificat TLS et sa structure (clé publique, signature, chaîne de confiance, etc.), ensuite de découvrir le système des Certificate Transparency logs, qui sont des bases de données publiques des certificats valides, puis d'apprendre à faire correspondre une clé publique à un certificat et enfin d'utiliser des outils d'investigation SSL/TLS et de recherche de certificats.

3.3.2 Méthode

Le fichier fourni est au format PEM (Privacy-Enhanced Mail), un format standard pour les clés cryptographiques qui utilise l'encodage Base64. La clé publique fournie était la suivante :

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuYj06m5q4M8SsEQwKX+5
NPs2lyB2k7geZw4rP68eUZmqODeqxDjv5mlLY2nz/RJsPdks4J+y5t96KAyo3S5g
mDqEOMG7JgoJ9KU+4HPQFzP9C8Gy+hisChdo9eF6UeWGTioazFDIdRUK+gZm81c1
iPEhOBIYu3Cau32LRtv+L9vzqre0Ollf7oeHqcbcMBIKL6MpsJMG+neJPnICI36B
ZZEMu6v6f8zIKuB7VUHAbDdQ6tsBzLpXz7XPBUeKPa1Fk8d22EI99peHwWt0RuJP
0QsJnsa4oj6C6lE+c5+vVHa6jVsZkpl2PuXZ05a69xORZ4oq+nwzK8O/St1hbNBX
sQIDAQAB
-----END PUBLIC KEY-----
```

Première étape On commence par comprendre comment relier une empreinte SHA-256 à un certificat TLS complet et comment extraire la clé publique d'un certificat. Un certificat TLS X.509 contient une clé publique encapsulée et la représentation DER est la base sur laquelle on calcule l'empreinte SHA-256 utilisée par certains services (par exemple `crt.sh` avec le paramètre `spkisha256`). Les formats PEM (texte Base64) et DER (binaire) sont simplement deux représentations interchangeables de la même information.

Deuxième étape La conception du script suit plusieurs objectifs : générer l'empreinte SHA-256 de la clé publique fournie, interroger les CT logs (par exemple via `crt.sh`) pour retrouver les certificats correspondants, télécharger le certificat complet (PEM) identifié dans les logs, extraire le nom de domaine du certificat, puis accéder au sous-domaine identifié pour récupérer le flag.

3.3.3 Résultat (Revoir présentation des resutlats)

L'exécution du script a produit les résultats suivants :
— **Empreinte (SHA-256) :**

`29ab37df0a4e4d252f0cf12ad854bede59038fdd9cd652cbc5c222edd26d77d2`

— **Sous-domaine identifié :**
`thetransparencyflagishere.cryptohack.org`

— **Flag obtenu :**
`crypto{thx_redpwn_for_inspiration}`

4 Diffie-Hellman

4.1 Introduction à l'échange de clés Diffie-Hellman

Cette catégorie aborde le protocole d'échange de clés de **Diffie-Hellman**, un mécanisme de cryptographie asymétrique. Proposé en 1976, il a pour objectif de résoudre le problème de la distribution de clés sur un canal de communication non sécurisé.

Le principe de ce protocole repose sur l'utilisation de fonctions mathématiques à sens unique, dont le calcul est aisé dans une direction mais calculatoirement difficile à inverser. Spécifiquement, Diffie-Hellman s'appuie sur la difficulté du problème du **logarithme discret** dans un groupe fini. Ce procédé permet à deux interlocuteurs d'établir une clé secrète partagée sans transmission préalable de celle-ci, y compris en présence d'un adversaire observant la communication.

Les challenges de cette section visent à analyser les fondements mathématiques de ce protocole, ainsi que les vulnérabilités pouvant résulter d'une implémentation incorrecte ou d'un choix de paramètres inadéquat.

4.2 L'attaque de l'homme du milieu (Man-in-the-Middle)

Cette sous-partie examine une vulnérabilité du protocole Diffie-Hellman : l'attaque de l'homme du milieu (Man-in-the-Middle ou MitM). Le protocole de base, dans sa forme originelle, ne fournit aucun mécanisme d'authentification des interlocuteurs. Il permet de s'assurer que la clé partagée est secrète, mais pas de vérifier l'identité de la personne avec qui on la partage.

Une attaque MitM exploite cette absence d'authentification. Un adversaire se positionne entre les deux communicants, intercepte leurs messages publics et établit une session Diffie-Hellman distincte avec chacun d'eux. Chaque interlocuteur génère alors une clé secrète partagée avec l'attaquant, tout en croyant communiquer directement l'un avec l'autre.

L'adversaire peut alors déchiffrer les messages, les lire, les modifier, puis les rechiffrer avec la clé de l'autre session avant de les transmettre au destinataire final. Les challenges de cette section illustrent comment cette interception est mise en œuvre et comment elle compromet la confidentialité et l'intégrité de l'échange.

4.2.1 Objectifs

L'objectif de ce challenge est de compromettre la confidentialité d'un échange sécurisé par le protocole Diffie-Hellman. Le scénario repose sur une attaque de l'homme du milieu (Man-in-the-Middle) durant la phase de négociation des paramètres cryptographiques.

La vulnérabilité exploitée est la capacité pour un attaquant d'influencer le choix des paramètres de groupe utilisés par les deux correspondants. En les contraignant à utiliser un groupe de petite taille (64 bits), le problème mathématique du logarithme discret, sur lequel repose la sécurité du protocole, devient calculatoirement soluble. La résolution de ce problème permet de retrouver une clé privée, et par conséquent la clé secrète partagée utilisée pour chiffrer la communication.

4.2.2 Méthode

Notre attaque s'est déroulée en plusieurs étapes. Premièrement, nous avons activement intercepté la communication initiale d'Alice, qui proposait une liste de groupes cryptographiques. Nous avons altéré ce message en ne conservant que le groupe le plus faible, caractérisé par un module de 64 bits, avant de le transmettre à Bob. L'acceptation de cette unique option par Bob a été relayée à Alice, établissant ainsi un accord sur un canal de communication affaibli.

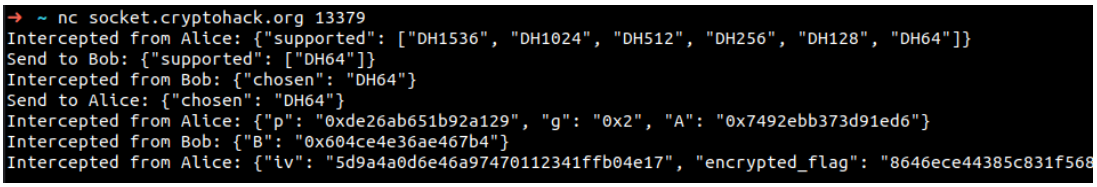

FIGURE 4 – Ceci est le schéma explicatif de notre méthode.

Une fois cet accord forcé, notre rôle est devenu passif. Nous avons collecté les paramètres publics de l'échange : le module `p`, le générateur `g`, et les clés publiques d'Alice (`A`) et de Bob (`B`). La faible taille du module `p` a alors permis de résoudre le problème du logarithme discret. À l'aide d'un script Python, nous avons calculé la clé privée `a` d'Alice à partir des valeurs publiques `p`, `g` et `A`.

La connaissance de cette clé privée nous a permis de reconstituer la clé secrète partagée ($s = B^a \pmod{p}$). Cette dernière a servi à dériver la clé de session AES, avec laquelle nous avons déchiffré le message final pour obtenir le *flag*.

4.2.3 Résultat

L'attaque par manipulation des paramètres a réussi. En forçant l'usage d'un groupe faible, nous avons pu calculer la clé privée, reconstituer la clé de session et déchiffrer le message, révélant le *flag* suivant :

`crypto{d0wn6r4d35_4r3_d4n63r0u5}`

4.3 Théorie des groupes

Cette sous-partie aborde les structures mathématiques qui sous-tendent de nombreux protocoles de cryptographie asymétrique : les groupes. En algèbre abstraite, un **groupe** est un ensemble d'éléments muni d'une opération binaire qui satisfait à des axiomes spécifiques (fermeture, associativité, existence d'un élément neutre et d'un inverse pour chaque élément).

La sécurité de protocoles comme Diffie-Hellman ne repose pas sur les nombres en tant que tels, mais sur les propriétés structurelles de ces groupes mathématiques. Des concepts

comme l'ordre d'un groupe, l'ordre d'un élément, et la notion de **générateur** d'un **groupe cyclique** sont des composantes directes de l'implémentation et de l'analyse de sécurité de ces systèmes.

Les challenges de cette section ont pour objectif d'étudier ces propriétés. Ils illustrent comment les caractéristiques d'un groupe, ou le choix de ses paramètres, peuvent influencer la robustesse d'un schéma cryptographique.

4.3.1 Objectifs

L'objectif de ce challenge est de calculer une clé secrète partagée dans une implémentation du protocole Diffie-Hellman utilisant un groupe additif. Contrairement à l'implémentation classique qui utilise un groupe multiplicatif d'entiers modulo un nombre premier, ce challenge transpose le problème dans une structure où l'opération de groupe est l'addition.

Le principe de sécurité reste fondé sur la difficulté d'inverser une fonction à sens unique. Ici, l'opération équivalente à l'exponentiation est la multiplication scalaire. Le but est de résoudre l'équivalent du problème du logarithme discret dans ce contexte additif afin de retrouver une clé privée, puis de reconstituer la clé secrète partagée, qui constitue le *flag*.

4.3.2 Méthode

Le challenge nous fournit les paramètres publics d'un échange Diffie-Hellman additif : un module premier `p`, un générateur `g`, ainsi que les clés publiques d'Alice (`A`) et de Bob (`B`). La relation qui lie la clé privée `a` à la clé publique `A` n'est plus $A = g^a \pmod{p}$, mais $A = a \cdot g \pmod{p}$.

Le problème du logarithme discret se traduit ici par la résolution de l'équation $A \equiv a \cdot g \pmod{p}$ pour trouver l'inconnue `a`. Cette équation est une congruence linéaire qui se résout efficacement en calculant l'inverse modulaire de `g` modulo `p`. Nous avons donc déterminé la clé privée d'Alice en calculant $a = A \cdot g^{-1} \pmod{p}$.

Une fois la clé privée `a` obtenue, nous avons pu calculer la clé secrète partagée en appliquant l'opération du groupe avec la clé publique de Bob. L'opération étant la multiplication scalaire, la clé secrète `s` est obtenue par la formule $s = a \cdot B \pmod{p}$.

4.3.3 Résultat

L'analyse de la structure de groupe additive a permis d'identifier la méthode de résolution appropriée. Le calcul de l'inverse modulaire nous a donné accès à la clé privée, menant directement à la reconstitution de la clé secrète partagée. Le *flag* obtenu est la valeur numérique de cette clé secrète :

`crypto{cycl1c_6r0up_und3r_4dd1710n?}`

5 RSA

5.1 Introduction au cryptosystème RSA

Cette catégorie aborde le cryptosystème à clé publique **RSA**, décrit pour la première fois en 1977. C'est l'un des systèmes de chiffrement asymétrique les plus connus et les plus utilisés dans le monde.

RSA repose sur un principe fondamental : la difficulté de factoriser de grands nombres composés en leurs facteurs premiers. Cette propriété mathématique permet de garantir la sécurité du système, à condition que les paramètres soient choisis et implémentés correctement.

Le cryptosystème RSA possède deux usages principaux. Le premier est le **chiffrement**, où un utilisateur publie une clé publique permettant à d'autres de lui envoyer des messages chiffrés, qu'il peut ensuite déchiffrer grâce à sa clé privée. Le second est la **signature numérique**, qui permet à un utilisateur de signer un message avec sa clé privée, afin que quiconque puisse vérifier l'authenticité et l'intégrité du message à l'aide de la clé publique correspondante.

Les challenges de cette section ont pour objectif d'explorer les fondements du cryptosystème RSA, tout en mettant en évidence les erreurs courantes et les failles d'implémentation qui ont, dans certains cas, conduit à des attaques réelles causant des pertes financières considérables.

5.2 RSA Multi-Factor Attack

Le challenge `"ManyPrime"` propose un fichier contenant un module RSA n, un exposant public e et un ciphertext ct. L'énoncé contient deux indices explicites : l'auteur indique qu'il utilisera "over 30" facteurs premiers et renvoie vers la méthode des courbes elliptiques (ECM). Ces indications orientent la stratégie d'attaque : extraire un grand nombre de facteurs premiers relativement petits grace à l'ECM, et retrouver la clef privée pour déchiffrer le message.

5.2.1 Objectifs

L'objectif est donc de récupérer le flag contenu dans le ciphertext. Concrètement, il faut factoriser **n** en l'ensemble de ses facteurs premiers. Puis calculer $d = e^{-1} \pmod{\varphi(n)}$ où $\varphi(n) = \prod_i (p_i - 1)$ et finalement déchiffrer ct pour obtenir le plaintext (le flag).

5.2.2 Méthode

Principe de l'attaque La vulnérabilité exploitée dans ce scénario réside dans une conception cryptographique déficiente : la construction d'un module RSA comme produit d'un grand nombre de facteurs premiers de petite taille. Cette approche, bien que contre-intuitive, affaiblit considérablement la résistance du module aux algorithmes de factorisation modernes. En effet, les méthodes de factorisation telles qu'ECM (Elliptic Curve Method) deviennent particulièrement efficaces lorsque le plus grand facteur premier n'excède pas une certaine taille critique.

Mise en œuvre pratique La première étape consiste à configurer l'environnement de calcul en initialisant un environnement Python virtuel et en installant les bibliothèques spécialisées nécessaires. La bibliothèque `primefac` fournit l'implémentation de l'algorithme ECM, tandis que `pycryptodome` offre les primitives cryptographiques essentielles pour les opérations arithmétiques modulaires.

Le processus de factorisation proprement dit s'articule autour d'une boucle itérative. Initialement, une variable temporaire `current` reçoit la valeur du module n à factoriser. À chaque itération, tant que `current` demeure composite, l'algorithme ECM est invoqué via `primefac.ecm()` pour en extraire un facteur premier p. Ce facteur est immédiatement

ajouté à une liste de facteurs reconnus, tandis que `current` est mis à jour par division entière par p. Une vérification systématique de la primalité du résidu guide la progression du processus.

Reconstruction et déchiffrement Une fois la factorisation complétée, une phase de validation est entreprise. L'intégrité de la décomposition est vérifiée en recalculant le produit de tous les facteurs identifiés, qui doit restituer exactement la valeur originale n. La fonction indicatrice d'Euler est alors calculée selon la relation $\varphi(n) = \prod(p_i - 1)$ pour l'ensemble des facteurs premiers p_i.

L'exposant privé d est déterminé comme solution de l'équation $e \cdot d \equiv 1 \pmod{\varphi(n)}$ via l'algorithme d'Euclide étendu. Finalement, le message chiffré ct est déchiffré par exponentiation modulaire selon la transformation $pt = ct^d \pmod{n}$, restaurant ainsi le message original.

Cette méthodologie démontre l'importance cruciale d'utiliser des modules RSA composés de deux facteurs premiers de grande taille en pratique cryptographique.

5.2.3 Résultat

Le script extrait plusieurs facteurs premiers et les affiche sur le terminal. Les valeurs obtenues sont présentées dans le tableau ci- dessous :

$p_1 = 17281246625998849649$	$p_2 = 9389357739583927789$	$p_3 = 10638241655447339831$
$p_4 = 16656402470578844539$	$p_5 = 14100640260554622013$	$p_6 = 9303850685953812323$
$p_7 = 17174065872156629921$	$p_8 = 14963354250199553339$	...

Table 1 – Facteurs premiers extraits par ECM

Après les calculs de vérification et de déchiffrement, on obtient le message en clair suivant :

`crypto{700_m4ny_5m4ll_f4c70r5}`

5.3 Cryptanalyse RSA avec exposant faible

Cette sous-partie présente l'analyse du challenge « Vote for Pedro » qui illustre une vulnérabilité cryptographique classique dans l'implémentation du chiffrement RSA. Le challenge met en avant les dangers liés à l'utilisation d'un exposant public faible, particulièrement lorsque combiné avec des messages de petite taille et l'absence de mécanisme de padding approprié. La résolution de ce challenge démontre comment il est possible de forger des signatures sans disposer de la clé privée, compromettant ainsi l'intégrité du système d'authentification.

5.3.1 Objectifs

L'objectif principal de ce challenge consiste à obtenir un flag en votant pour Pedro avec une signature valide émise par Alice. Le serveur vérifie la signature du vote en utilisant la clé publique d'Alice et retourne le flag uniquement si le message déchiffré correspond exactement à la chaîne « VOTE FOR PEDRO ». La difficulté réside dans l'impossibilité d'accéder à la clé privée d'Alice pour signer le message légitimement.

5.3.2 Méthode

Le challenge fournit deux éléments essentiels : le script du serveur et la clé publique d'Alice. L'analyse du script serveur révèle le mécanisme de vérification des signatures. Le serveur reçoit un vote signé sous forme hexadécimale, applique la vérification RSA en calculant le vote à la puissance de l'exposant public modulo N, puis convertit le résultat en bytes. Le message est ensuite extrait en prenant la partie suivant le dernier octet nul avant d'être comparé avec la chaîne attendue.

La clé publique d'Alice présente une particularité cruciale : l'exposant public $e = 3$. Cette valeur faible, combinée avec l'absence de padding robuste, crée une vulnérabilité exploitable en permettant une attaque par extraction de racine cubique. En effet, lorsque le message original est inférieur à la racine cubique de N, l'opération de signature peut être inversée sans recours au modulo, rendant possible la forge de signature.

Principe mathématique Le système vérifie la signature via l'équation RSA standard $\text{vote}^e \pmod N$. Avec $e = 3$ et un message M suffisamment petit ($M < \sqrt[3]{N}$), l'opération modulo devient inutile, simplifiant l'équation en $\text{vote}^3 = M$. La signature s'obtient donc par calcul direct de $\sqrt[3]{M}$.

Le message « VOTE FOR PEDRO » n'étant pas un cube parfait, on cherche un entier S tel que S^3 partage les mêmes bits de poids faible que le message original. Cette condition est suffisante car le serveur extrait le message après le dernier octet nul, ignorant les bits de poids fort.

Algorithme de résolution L'algorithme `cube_root_2_pow` calcule la racine cubique bit par bit. Initialisé avec $s = c \pmod 8$ pour capturer les trois bits de poids faible, il procède par raffinement successif. Pour chaque position k, il ajuste le bit correspondant pour minimiser la différence entre s^3 et c modulo 2^k, garantissant la convergence vers la solution optimale.

Implémentation technique L'implémentation établit une connexion socket, convertit le message en entier via `bytes_to_long`, et calcule la racine cubique avec une précision adaptée. La signature résultante est convertie en hexadécimal et transmise dans un payload JSON conforme au protocole attendu par le serveur.

5.3.3 Résultat

L'exécution du script de résolution produit le résultat suivant après connexion au serveur :

```
Place your vote. Pedro offers a reward to anyone who votes for him!
{"flag": "crypto{y0ur_v0t3_i5_my_v0t3}"}
```

6 Conclusion

Annexes

A Script de résolution du challenge Encoding

Voici le code Python complet utilisé pour le challenge "Encoding".

```
import pwn
import json

# ... (le reste de votre code) ...
```

B Script de résolution du challenge Lemur XOR

Voici le code utilisé pour le challenge "Lemur XOR".

```
from PIL import Image

# ... (le reste de votre code) ...
```