Université de Versailles
Saint-Quentin-en-Yvelines

Rapport de projet

Implémentation et attaque sur le chiffrement Panther

Jean-Paul Morcos Doueihy
Billal Medour
Malak Lalami
Alexandre Mihet

Date de soumission : 8 mai 2023

Table des matières

Chapitre 1

Introduction

Panther est un schéma de chiffrement authentifié léger avec une fonction éponge, publié lors de la conférence Indocrypt 2021 (cf [1]). Son fonctionnement repose sur quatre registres à décalage à rétroaction non linéaire (NFSR). Le but du projet est d'implémenter ce chiffrement ainsi qu'une attaque contre ce dernier qui permet de le casser complètement.

Cette méthode de chiffrement a été cassée par Mme Christina Boura, Mme Rachelle Heim Boissier et Mr Yann Rotella. Nous nous sommes basés sur leurs travaux de recherche pour implémenter l'attaque (cf [2]).
Ce projet a été entièrement été mené en Python.

Chapitre 2

Explication de Panther

Dans cette partie nous allons nous pencher sur les différents aspects de Panther à partir de l'article publié dans la conférence Indocrypt et de quelle manière nous les avons interprétés.

2.1 Définitions

Pour comprendre pleinement le fonctionnement du chiffrement Panther, il faut comprendre les notions suivantes.

2.1.1 Authentification

Le chiffrement Panther est un chiffrement authentifié : le chiffrement authentifié fournit à la fois la confidentialité et l'authentification, tandis que le chiffrement simple ne fournit que la confidentialité.

L'algorithme génère un texte chiffré et une étiquette d'authentification à partir de la clé, du texte en clair et des données associées fournies en entrée pendant le chiffrement. Lorsque le texte chiffré est déchiffré, l'algorithme prend la clé, l'étiquette d'authentification et les données associées, puis renvoie le texte en clair ou une erreur si l'étiquette ne correspond pas au texte chiffré.

2.1.2 Fonction éponge

Une fonction éponge est une fonction qui prend en paramètre une chaîne de n'importe quelle taille et qui retourne une chaîne de la longueur que l'on souhaite. Ses paramètres sont une permutation, ou plus généralement une transformation f opérant sur un nombre fixe b de bits, qu'on appellera largeur, et une fonction de remplissage p.

Son fonctionnement est le suivant : la chaîne d'entrée est complémentée par la fonction de complément pour aligner la taille de la chaîne d'entrée sur un nombre entier de blocs, puis découpée en blocs de r bits. Les b bits de l'état interne sont initialisés à zéros, puis le calcul se déroule en deux phases :

Absorption

Dans cette phase, les blocs de r bits sont Xorés avec les r premiers bits de l'état, en alternance avec des applications de la fonction f sur l'état. Cette phase se déroule jusqu'à ce que tous les blocs aient été traités.

Essorage

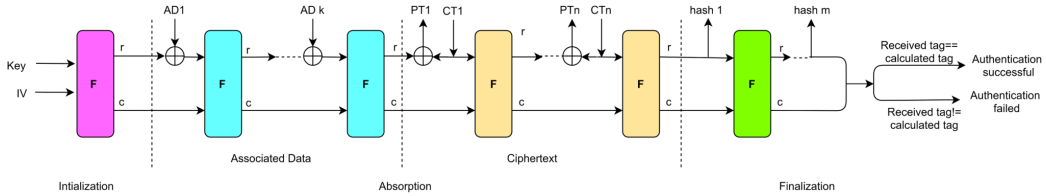
Cette phase est la construction pas à pas de la sortie. Elle consiste en la répétition des deux étapes suivantes :
1 - les r premiers bits de l'état sont retournés comme blocs de sortie.
2 - la fonction f est appliquée sur l'état. Le nombre d'itération est au choix de l'utilisateur et la taille de la chaîne de sortie en dépend.
Les derniers c bits de l'état ne sont jamais directement modifiés par les blocs d'entrée et ne sont jamais retournés pendant l'essorage.
Plus d'informations sur les fonctions éponge sur le lien wikipedia suivant : [3]

2.2 Chiffrement

2.2.1 Structure

Panther fonctionne de manière itérative et se compose d'un état de 328 bits divisé en une partie extérieure de taux r = 64 bits et une partie intérieure de capacité c = 264 bits. Tous les algorithmes décrits dans l'article [1] fonctionnent sur l'état interne de 328 bits. Les 328 bits sont divisés en registres P, Q, R, S. L'état est mis à jour en itérant une fonction F composée de quatre NFSRs inter-connectés de tailles 19, 20, 21 et 22 respectivement. Cette fonction est itérée 92 fois pour la phase d'initialisation et de finalisation et seulement 4 fois après l'absorption des données associées (AD), au niveau des blocs de texte en clair et après l'extraction d'un bloc du tag.
Voici un schéma de Panther tiré de l'article [1]

Voici une légende du schéma précédent :



Notation	Description
Key	Clé secrète de taille 128 bits
IV	Vecteur d'initialisation de taille 128 bits
PT	Texte clair de taille arbitraire
CT	Texte chiffré de taille arbitraire
AD	Données associées
F	Fonction de mise à jour de l'état
État	L'état interne de la fonction éponge de taille 328 bits
$\oplus$	XOR

2.2.2 Phases de Panther

Comme le montre le schéma précédent, nous pouvons remarquer qu'il y a plusieurs phases distinctes durant un chiffrement avec Panther, nous allons les développer dans cette partie :

Phase d'initialisation

Les entrées sont la clé et l'IV, tous deux de 128 bits. La clé est chargée en premier, puis l'IV. Les bits restants contiennent 64 bits de la complémentaire de la clé, de sept 1, et d'un 0 à la fin. En utilisant la fonction de mise à jour de l'état F, l'état est mis à jour 92 fois.

Phase d'absorption

La phase d'absorption partitionne à la fois les données associées et le texte en clair en morceaux de r bits. Les bits d'entrée sont XORés avec les bits d'état et entrelacés avec la fonction de mise à jour d'état F. Les données associées sont organisées en k blocs de 64 bits. Chacun de ces k blocs est mis à jour quatre fois avec la fonction F. Le texte en clair est divisé en n blocs de 64 bits chacun après le traitement des k blocs. Nous obtiendrons le texte chiffré tout en traitant le texte en clair.

Phase de traitement du texte chiffré

Dans le processus de décryptage, nous effectuons le traitement du texte chiffré comme indiqué dans l'algorithme 5 de l'article [1]. Le texte chiffré reçu est divisé en blocs de 64 bits et donné à cette phase en entrée. Les bits d'état sont XORés avec le texte chiffré. À cette étape, nous récupérons le texte clair.

Phase de finalisation

Ici, la partie sr de l'état interne de la fonction éponge est donnée sous forme de blocs de sortie qui sont entrelacés avec quatre applications de la fonction F. La longueur d'entrée de hachage de l'utilisateur détermine le nombre de blocs de sortie. La balise d'authentification est le résultat de cette phase.

2.3 Attaque

Le chiffrement Panther se veux être un système léger, avoir un bon compromis entre la rapidité , la sûreté et le besoin mémoire. Cependant, comme le démontre l'article [2], il y a une faille majeure dans Panther : il est possible de retrouver la clé a partir uniquement d'un seul couple clair/chiffré et avec des ressources informatiques minimales. Cette faille majeure vient du fait qu'il y a un faible nombre d'itérations de la fonction de tour pendant la phase d'absorption.

2.3.1 Explication de l'attaque

A la fin de la phase d'initialisation nous ne sommes pas censé connaître le "state" étant donné que la fonction F a été appliquée 92 fois et la clé a été mixée avec le vecteur d'initialisation (IV).

Cependant on connaît le texte chiffré, et a chaque fois qu'on applique la fonction F (4 fois en tout), les bits sont tous décalés d'un cran vers la droite.

Au début le "state" était composé par exemple de P18 jusqu'à P0 et on ne pouvait pas savoir ce que c'était. Ensuite on applique la fonction F 4 fois et les 4 premiers bits du message vont être Xoré avec P18, P17, P16, P15 et nous allons obtenir le texte chiffré c0, c1, c2, c3.

c0 par exemple est le résultat du Xor entre P18 et le bit m0 du message. Ensuite si on ré-applique une fois supplémentaire la fonction F 4 fois, nous allons avoir c4, c5, c6, c7 par exemple et les c0, c1, c2, c3 sont décalés vers la droite (vers l'inner state) et du coup le state sera composé de c0 à p4.

Nous connaissons donc une partie du "state" étant donné que l'on connaît c0 à c3. Il suffit de ré-appliquer la fonction F 4 fois pour avoir c8, c9, c10, c11. Ainsi, l'inner state sera c4,..., c7, c0,..., p8.

Nous voyons que nous connaissons mieux l'inner state. Nous ré-appliquons la fonction F 4 fois (5 fois en tout) et à la fin nous connaîtrons tout l'inner state et il suffira d'implémenter une fonction inverse à F pour retrouver le message clair.

Chapitre 3

Implémentation de Panther

3.1 Implémentation du chiffrement

Dans cette partie du rapport de projet, nous allons présenter la méthodologie que nous avons suivie pour implémenter Panther en Python en nous basant sur l'article [1]. Nous discuterons également des difficultés que nous avons rencontrées et des solutions que nous avons trouvées pour les surmonter.

3.1.1 Préambule

Avant de commencer, bien qu'il n'y ai pas d'implémentation officielle de l'attaque (vérifiée par les créateurs de Panther), dans l'article [2] il y est mentionné un lien github où les concepteurs de l'attaque ont implémenté l'attaque entière sur Panther en C.

Nous avons implémenté toutes les fonctions relatives au chiffrement Panther d'après l'article [1] et nous avons comparé leur fonctionnement au fonctionnement du chiffrement dans le github de l'attaque. A l'aide d'un vecteur de test fourni par Mme Rachelle Heim Boissier, nous avons pu vérifier que notre implémentation du chiffrement fonctionne de la même manière que celui sur le github.

3.1.2 Structures de données utilisé

Étant donné que l'entièreté du projet a été mené en Python, nous avons utilisé des structures de données natives de ce langage.

Par exemple, nous avons choisi d'implémenter la clé, le vecteur d'initialisation, le texte clair, le texte chiffré et les données associés comme des chaînes de caractères.

L'état des registres à un moment donné est stocké dans une variable "state" qui est une chaînes de caractères également. Concernant la boite-S, nous utilisons un dictionnaire : un caractère est associé à un autre caractère, ce qui permet de faire le changement facilement.

Il est bon de mentionner que l'entièreté de notre code a été implémenté avec de la programmation fonctionnelle, il n'y a donc pas de classes mais uniquement des fonctions.

3.1.3 Fonction F

La fonction F est la fonction permettant de mettre à jour le "state", elle a besoin de fonctions préalablement définies selon l'article [1]. Voici de quelles étapes la fonction F est constituée.

1) On divise l'état en 4 registres de tailles différentes (P, Q, R, S).

2) Pour chaque registre, on convertit chaque bloc de 4 bits en un entier binaire.

3) On calcule quatre polynômes de rétroaction (fp, fq, fr, fs) à partir de bits sélectionnés dans les registres P, Q, R, S. Cette étape consiste en des opérations logiques (ou exclusif) et des multiplications de champ fini.

4) On calcule quatre polynômes d'interconnexion (gp, gq, gr, gs) à partir de bits sélectionnés dans les registres P, Q, R, S. Cette étape consiste en des opérations logiques (ou exclusif).

5) On crée quatre chaînes de 4 bits (l1, l2, l3, l4) en combinant des bits des polynômes de rétroaction et d'interconnexion.

6) On effectue une multiplication de matrice (multiplication de Toeplitz) pour transformer les chaînes l1, l2, l3, l4 en quatre nouvelles chaînes de 4 bits (d1, d2, d3, d4).

7) On applique une boîte de substitution (S-box) à chaque élément de d1, d2, d3, d4, puis on effectue une multiplication de matrice (multiplication de Toeplitz) pour obtenir quatre nouvelles chaînes de 4 bits (t1, t2, t3, t4).

8) On décale les registres P, Q, R, S de 1 bit vers la gauche.

Les 8 étapes sont répétées n fois, où n est le deuxième argument de la fonction. La fonction renvoie la nouvelle valeur de l'état après n itérations.

3.2 Implémentation de l'attaque

L'attaque a tout d'abord nécessité l'implémentation de fonctions inverses de certaines fonctions précédemment définies comme par exemple l'inverse de la fonction de substitution, l'inverse de la matrice Toeplitz etc...

3.2.1 Attaque par couple clair/chiffré connu

L'objectif de cette attaque est de retrouver la clé secrète qui a été utilisée pour chiffrer un message donné en clair et son correspondant chiffré.

Ce code effectue une attaque de récupération de clé en travaillant en arrière à partir de de la sixième itération du chiffrement et en XORant les bits avec le message en clair correspondant, puis en inversant la fonction de chiffrement utilisée pour transformer l'état interne. L'attaque utilise ensuite l'état interne obtenu après cinq itérations pour retrouver la clé secrète en effectuant
92 itérations supplémentaires.

1. Le code commence par récupérer le "state" complet à partir du texte chiffré en concaténant les six parties de chaque bloc de 64 bits du texte chiffré dans un ordre spécifique. Le "state" est une représentation interne de l'état de la fonction de chiffrement à un moment donné.
2. Ensuite, le code divise le texte clair en blocs de 64 bits et les stocke dans une liste.
3. L'attaque commence en partant du chiffrement du sixième bloc et en travaillant en arrière. Elle effectue une boucle de cinq itérations, chacune impliquant les opérations suivantes :
   a. Les 64 bits de l'état externe "sr" sont XORés avec les 64 bits du message en clair correspondant.
   b. Les bits de l'état externe "sr" sont ensuite transformés à l'aide d'une fonction appelée Inv_f, qui est l'inverse de la fonction de chiffrement utilisée pour chiffrer le message en clair.
4. Après avoir effectué l'attaque par force brute sur tous les blocs de texte clair, le code effectue un XOR entre les parties extérieures du "state" et le premier bloc de texte clair. Le "state" est ensuite mis à jour avec le nouveau "sr" obtenu.
5. Enfin, le "state" est inversé 92 fois à l'aide de la fonction inverse de chiffrement et les 128 premiers bits du résultat sont retournés comme clé de chiffrement récupérée.

3.2.2 Attaque par chiffré seul connu

L'objectif de cette attaque est de retrouver une partie du texte clair en ne connaissant que le texte chiffré.

1. Le code commence par récupérer le "state" complet à partir du texte chiffré en concaténant les six parties de chaque bloc de 64 bits du

texte chiffré dans un ordre spécifique. Le "state" est une représentation interne de l'état de la fonction de chiffrement à un moment donné.

2. Il divise le texte chiffré en blocs de 64 bits et stocke ces blocs dans une liste.
3. Il procède à une boucle sur tous les blocs de texte chiffré supérieurs au sixième bloc et effectue les opérations suivantes :

   a. Applique la fonction F à l'état actuel 4 fois (ce qui est une caractéristique de l'algorithme utilisé dans ce code).

   b. Récupère l'état de registre de décalage (SR) de l'état actuel.

   c. Effectue une opération XOR entre le SR et le bloc de texte chiffré actuel, puis stocke le résultat dans la liste des blocs de texte clair récupérés.

   d. Met à jour l'état avec le SR modifié.
4. Il retourne la liste des blocs de texte clair récupérés.

Remarque : Cette attaque ne récupère que les blocs de texte clair après le sixième bloc. Pour récupérer l'ensemble du texte clair, il faut connaître les six premiers blocs de texte clair ou utiliser une autre attaque pour les récupérer.

Chapitre 4

Remerciements

Nous tenons à remercier Mme Christina Boura pour l'encadrement du projet, les entretiens explicatifs et le temps pris pour les réponses à toutes les questions que nous avons eues.
Également, nous tenions à remercier Mme Rachelle Heim Boissier qui nous a fourni un vecteur de test, ce qui nous a permis de tester notre code.

Bibliographie

[1] K. V. L. Bhargavi , C. Srinivasan , and K. V. Lakshmy, Panther : A Sponge Based Lightweight Authenticated Encryption Scheme, Indocrypt, 2021.

[2] C. Boura, R. Heim Boissier, Y. Rotella, Breaking Panther, Université Paris-Saclay, UVSQ, CNRS, 2022, https://eprint.iacr.org/2022/111.pdf.

[3] Fonctions éponge (Wikipedia).