h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Bruteforce password attack on FPGAs

< Insérez ici votre illustration >

(non obligatoire)

Projet de semestre présenté par

Abivarman KANDIAH

Informatique et systèmes de communication avec orientation
Systèmes informatiques embarqués

mars, 2024

Professeur-e HES responsable

Andres UPEGUI POSADA, Stéphane
Gaétan KÜNG

Mandant

ELCA Security

Légende et source de l’illustration de couverture :

TABLE DES MATIÈRES

La table des matières doit reprendre tous les niveaux de titre et sous-titre du mémoire, y compris les pages initiales (page des remerciements, énoncé du sujet, résumé, table des annexes et autres tables), ainsi que les références documentaires, etc.

h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

RÉSUMÉ

Lorsqu'un utilisateur doit s'authentifier auprès d'un service, il doit fournir un mot de passe préalablement défini. Pour des raisons de sécurité, le système stocke ce mot de passe en le passant par une fonction de hachage. Ainsi, lors de l'authentification, le système compare le hash du mot de passe entré par l'utilisateur avec le hash stocké pour vérifier son identité. Une fonction de hachage qui est souvent utilisé pour le stockage de mots de passe est le Bcrypt, qui a comme particularité d'être assez lente, rendant les mots de passe assez résistante aux attaques par bruteforce. Ce rapport a pour but de détailler la mise en oeuvre d'un système visant à attaquer les mots de passe protégés par l'algorithme bcrypt en utilisant un Field-Programmable Gate Array (FPGA). L'objectif principal est de créer une solution extensible et plus performante que les approches traditionnelles basées sur Graphics Processing Unit (GPU). Le système repose sur plusieurs cœurs de calcul parallèles sur le FPGA pour générer les hashs bcrypt. La génération des mots de passe pour le bruteforce est directement réalisée sur le FPGA à l'aide d'un système de compteur pour distribuer les tâches aux cœurs de calcul. Après validation du fonctionnement, des mesures ont été effectuées pour évaluer les performances et les ressources utilisées sur le FPGA. Pour explorer la possibilité de déléguer la génération de mots de passe à un ordinateur, une interface Peripheral Component Interconnect Express (PCIe) a été établie entre la carte FPGA et un ordinateur. Bien que le projet soit fonctionnel, il reste à mettre en place une interface complète entre l'ordinateur et la carte FPGA pour qu'elle soit utilisable lors d'une véritable attaque.

< Insérez ici votre illustration >
(obligatoire)

Candidat-e :

ABIVARMAN KANDIAH

Filière d'études : ISC

Professeur-e(s) responsable(s) :

ANDRES UPEGUI POSADA, STÉPHANE GAÉTAN KÜNG

En collaboration avec : ELCA Security

Travail de bachelor soumis à une convention de stage en entreprise : non

Travail soumis à un contrat de confidentialité : non

LISTE DE ACRONYMES

BRAM Block RAM. 12, 13, 17, 18

FPGA Field-Programmable Gate Array. v, 1, 3, 6, 7, 9

GPU Graphics Processing Unit. v

I/O Input / Output. 12

IC Integrated Circuit. 3

PC Personal Computer. 6, 7

PCIe Peripheral Component Interconnect Express. v, 1

SBOX Substitution boxes. 7, 15

VHDL Very High Speed Integrated Circuit Hardware Description Language. 3, 9

LISTE DES TABLEAUX ET ILLUSTRATIONS

Références des URL

— URL01 ce-site.ch/bla/bli/blo/blou.html
— URL03 ce-site.ch/blou/bli/bla.html
— URL04 https://commons.wikimedia.org/w/index.php?curid=906980
— URL06 ce-site.ch/monrapportdestage.pdf

LISTE DES ANNEXES

INTRODUCTION

Ce travail s'inscrit dans le cadre de mon travail de semestre, réalisé en réponse à une demande d'ELCA Security, une entreprise spécialisée dans la cyber-sécurité. Le projet présenté ici vise à explorer une approche peu commune pour attaquer des mots de passe protégés par l'algorithme bcrypt en utilisant un FPGA. Ce projet a pour objectif final de leur fournir une solution extensible et performante qui puisse être utilisable lors de leurs attaques. L'intérêt technique et scientifique de ce projet réside dans la recherche de solutions moins énergivores pour l'attaque de mots de passe, en tirant parti des capacités de traitement parallèle offertes par les FPGA.

Après une recherche approfondie des implémentations existantes du bcrypt sur FPGA, le projet a débuté par une analyse du papier [1] concernant une implémentation déjà existante. En parallèle, une page Wikipédia détaillant le fonctionnement de bcrypt [2] a été utilisée comme ressource principale pour comprendre les spécificités de cet algorithme. De plus, un papier sur une attaque MD5 [3] sur FPGA a été consulté pour enrichir la compréhension des techniques d'attaque sur des dispositifs matériels. Ces ressources documentaires ont été cruciales pour comprendre les différents concepts, orienter les choix de conception et résoudre les problèmes techniques rencontrés.

Dans le cadre de ce projet, j'ai entrepris plusieurs actions significatives. Tout d'abord, j'ai repris le code de l'implémentation existante en VHDL d'un programme d'attaque par bruteforce de mot de passe bcrypt. Après avoir étudié le papier associé et constaté des incohérences dans les testbenches, j'ai refait ces derniers pour valider le programme. Par la suite, j'ai identifié et corrigé des erreurs dans le code afin d'obtenir un programme fonctionnel.

En parallèle, j'ai pu brièvement étudier le fonctionnement du PCIe afin d'y mettre en place une simple interface entre une carte FPGA et un ordinateur.

1. Friedrich WIEMER et Ralf ZIMMERMANN. "High-speed implementation of bcrypt password search using special-purpose hardware". In : *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. ISSN: 2325-6532. Déc. 2014, p. 1-6. DOI : 10.1109/ReConFig.2014.7032529. URL : https://ieeexplore.ieee.org/document/7032529.

2. *bcrypt*. en. Page Version ID: 1210874707. Fév. 2024. URL : https://en.wikipedia.org/w/index.php?title=Bcrypt&oldid=1210874707.

3. Maruthi GILLELA, Vaclav PRENOSIL et Venkat Reddy GINJALA. "Parallelization of Brute-Force Attack on MD5 Hash Algorithm on FPGA". In : *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. ISSN: 2380-6923. Jan. 2019, p. 88-93. DOI : 10.1109/VLSID.2019.00034. URL : https://ieeexplore.ieee.org/document/8710753.

Dans ce rapport, je vais commencer par brièvement expliquer les différents notions clés de ce projet, afin de poser une base technique. Je vais par la suite vous présenter une analyse du projet, afin d'y décrire le projet de manière détaillé, expliquer le fonctionnement du Bcrypt et les recherches qui ont été faites afin de trouver une implémentation existante. Puis, je vais détailler la méthodologie de travail, en exposant les différentes étapes de mise en œuvre du projet, les simulations qui ont été faites et les différentes difficultés rencontrées. Enfin, je vais décrire les résultats obtenus lors des différents tests et mesures qui ont été faits.

CHAPITRE 0 : BASE TECHNIQUE

Ce chapitre a pour but d'introduire et expliquer les différents aspects techniques clés de ce projet de semestre. Je vais notamment expliquer brièvement ce qu'est un FPGA et le principe d'une fonction de hachage.

1.1. FPGA

Un FPGA est un Integrated Circuit (IC) dans laquelle on peut programmer et interconnecter des circuits logiques. Contrairement à un processeur, qui est limité par un certain nombre d'instructions et exécute les instructions de manière séquentielle, un FPGA permet d'exécuter de nombreux circuits logiques en parallèle.

Pour programmer un FPGA, on utilise généralement des langages de description matériel tels que le Very High Speed Integrated Circuit Hardware Description Language (VHDL) et le Verilog. Dans ce projet, j'ai personnellement travaillé avec le VHDL.

Lorsque l'on souhaite tester un programme VHDL, il est possible d'utiliser des outils de simulation afin de vérifier le fonctionnement souhaité. Il est aussi possible d'automatiser la phase de simulation à l'aide de fichier que l'on appelle testbench.

1.2. FONCTION DE HACHAGE

Une fonction de hachage est une fonction qui va prendre en entrée une donnée a taille variable et va ressortir une donnée de taille fixe.

Une des propriétés fondamentales d'une fonction de hachage est qu'il n'existe pas de fonction mathématique permettant de retrouver la donnée originale à partir d'un hash généré. Même une petite modification apportée à la donnée en entrée conduira à un hash totalement différent en sortie. Cette particularité est essentielle pour sécuriser le stockage des mots de passe, car même si des hash venait a être compromis, il est extrêmement difficile de retrouver les mots de passe originaux à partir de leurs hachages.

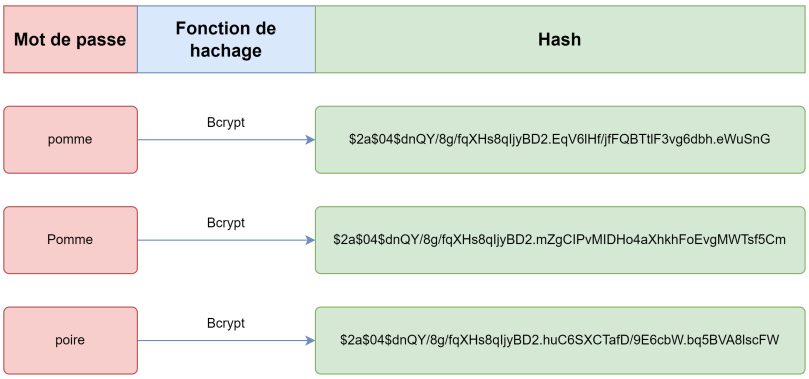

ILLUSTRATION 1.1 – Fonction de hachage. Source : réalisé par Kandiah Abivarman

a. Salt

Certaines fonctions de hachage tel que le bcrypt utilisent ce qu'on appelle un salt (sel en français), qui est une valeur générée aléatoirement qu'on va donner avec notre mot de passe. Le salt va permettre d'avoir un hash différent, même si deux personnes utilisent le même mot de passe, ajoutant ainsi une couche supplémentaire de sécurité.

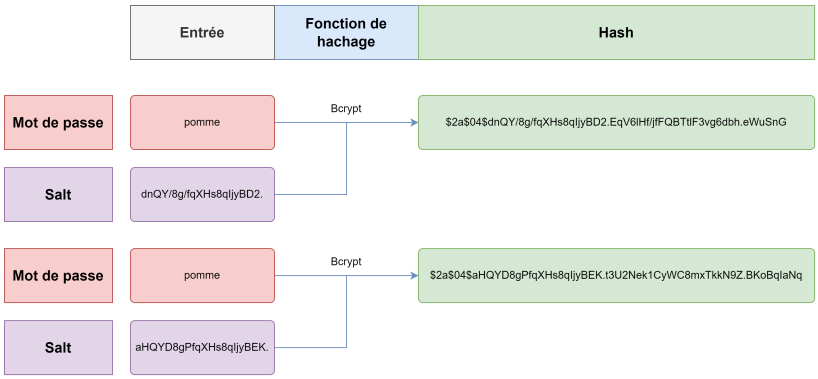

ILLUSTRATION 1.2 – Salt. Source : réalisé par Kandiah Abivarman

b. Attaque par bruteforce

L'attaque par bruteforce consiste à essayer toutes les combinaisons possibles de mots de passe afin de retrouver celui qui correspond au hash compromis. Cette méthode repose sur le fait qu'il est impossible de retrouver directement le mot de passe à partir du hash, obligeant ainsi l'attaquant à tester différentes entrées jusqu'à ce qu'il trouve celle qui génère le hash recherché.

Toutefois, cette méthode peut prendre beaucoup de temps, notamment lorsque les fonctions de hachage utilisées sont conçues pour être lentes à calculer.

CHAPITRE 1 : ANALYSE

Ce chapitre a pour but d'expliquer de manière détaillée l'objectif de ce projet de semestre. Je vais aussi faire part des différentes idées qui sont ressorties lors de nos discussions avec mes professeurs. Par la suite, je vais expliquer en quoi consiste la fonction de hachage Bcrypt, son fonctionnement et ses spécificités. Pour finir, je vais rapporter les différentes implémentations sur FPGA que j'ai pu retrouver et celui que j'ai fini par reprendre durant le projet de semestre.

2.1. DESCRIPTION DU PROJET

L'objectif principal de ce projet est d'exploiter le parallélisme offert par les FPGA, afin de calculer les fonctions de hachage nécessitant beaucoup de temps de calculs. Le but étant d'avoir au final un système plus efficient que les solutions actuelles lors d'une attaque par bruteforce. Il est aussi nécessaire d'avoir une certaine communication entre le Personal Computer (PC) de l'attaquant et le FPGA, afin que l'attaquant puisse fournir le hash qu'il souhaite casser.

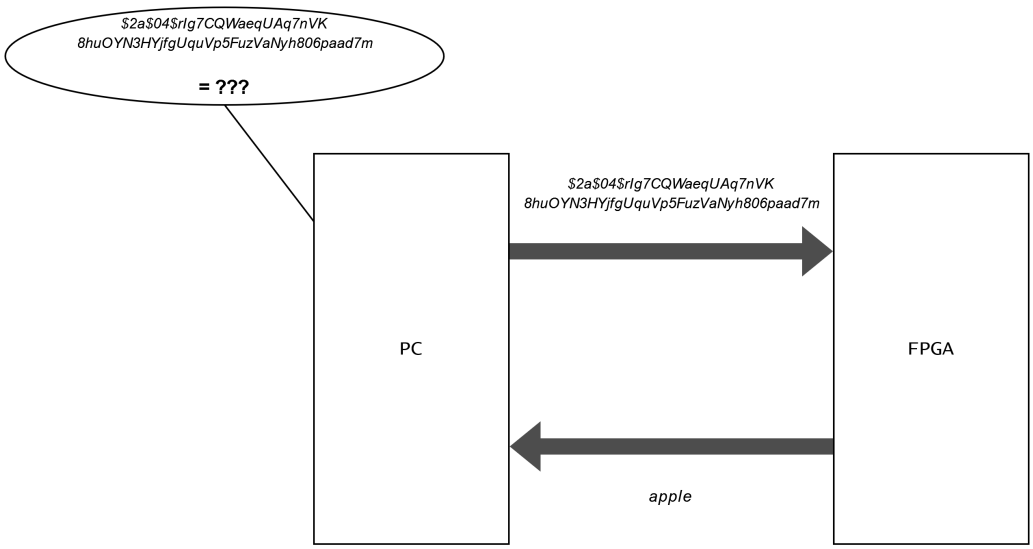

ILLUSTRATION 2.1 – Diagramme général. Source : réalisé par Kandiah Abivarman

Une des première question qu'on s'était posée avec mes professeurs responsables était la manière pour générer les mots de passe lors d'une attaque. Dans notre cas, nous avions deux possibilités, soit nous générons les mots de passe directement depuis le PC et devons les transmettre à la carte FPGA afin de procéder au hachage, soit la génération se fait directement dans le FPGA. La différence est que dans le premier cas, on peut être potentiellement limité au niveau

logiciel par le PC ou au niveau de la communication avec le FPGA. Toutefois, au niveau de la génération des mots de passe on aura plus de flexibilité permettant d'autres types d'attaques comme par exemple une attaque par dictionnaire [4].

Pour le projet de semestre, j'ai décidé de partir plutôt vers la génération sur FPGA, afin d'avoir une première solution entièrement fonctionnelle sur FPGA sans dépendance avec un PC. La génération sur PC est toutefois envisageable par la suite pour le projet de bachelor.

2.2. BCRYPT

Pour ce projet, nous avons décidé de cibler le Bcrypt car c'est une fonction de hachage qui prend du temps à être calculé.

Le Bcrypt est une fonction de hachage avec comme particularité, un paramètre supplémentaire qui est le cost (coût en français). Ce paramètre va définir le nombre d'itérations que va prendre la fonction de hachage, de ce fait plus le cost est élevé, plus le calcul va prendre du temps.

a. Algorithme

L'algorithme du Bcrypt se base sur l'algorithme de chiffrement Blowfish [5] qui est une fonction de chiffrement à clef symétrique, c'est à dire que la même clef est utilisée pour le chiffrement et le déchiffrement. L'algorithme du Bcrypt peut être diviser en deux grandes étapes.

On a une première étape qui est une phase de mise en place des clés symétriques. Dans cette étape on va créer les clés de chiffrements à partir des paramètres d'entrée de la fonction de hachage (mot de passe, salt, cost). Cette première étape est la partie la plus coûteuse de la fonction car la mise en place de la clé va prendre plus ou moins de temps en fonction du cost. Les clés de chiffrement sont composés de Subkeys qui est un tableau de 18 entiers de 32 bits et quatre Substitution boxes (SBOX) qui sont chacun des tableaux de 256 entiers de 32 bits. Avant de calculer ces clés de chiffrements, ils sont tout d'abord initialisés avec les décimales de PI.

Puis il y a la deuxième étape, où l'on va utiliser les clés de chiffrement qui ont été calculés plus tôt afin de chiffrer la phrase magique "OrpheanBeholderScryDoubt", le chiffrement va être fait 64 fois.

4. TO DO : Add wiki
5. TO DO : Add wiki

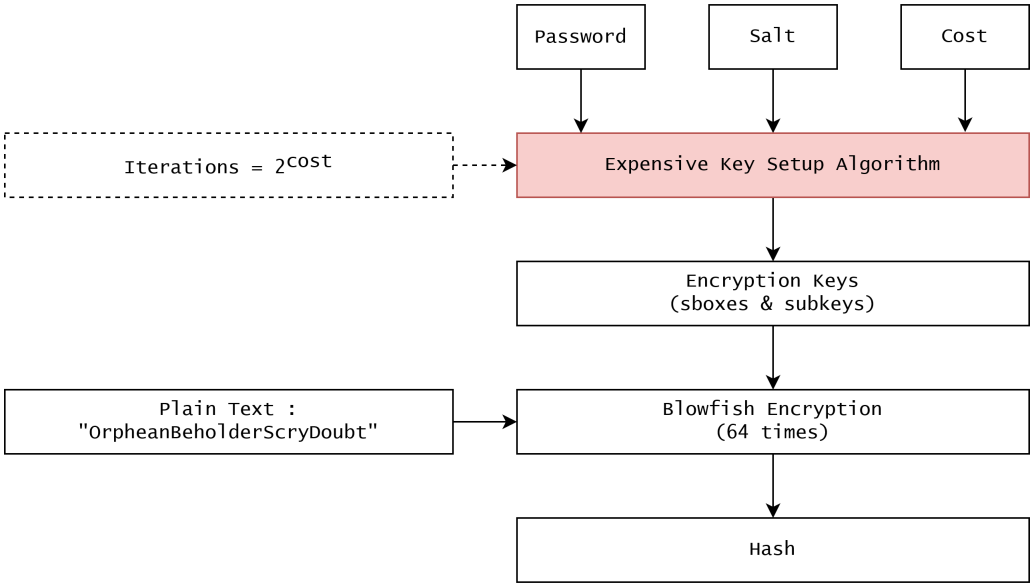

ILLUSTRATION 2.2 – Algorithme Bcrypt. Source : réalisé par Kandiah Abivarman

b. Format du Hash

Le hash généré par la fonction Bcrypt est généralement stocker sous une forme particulière.

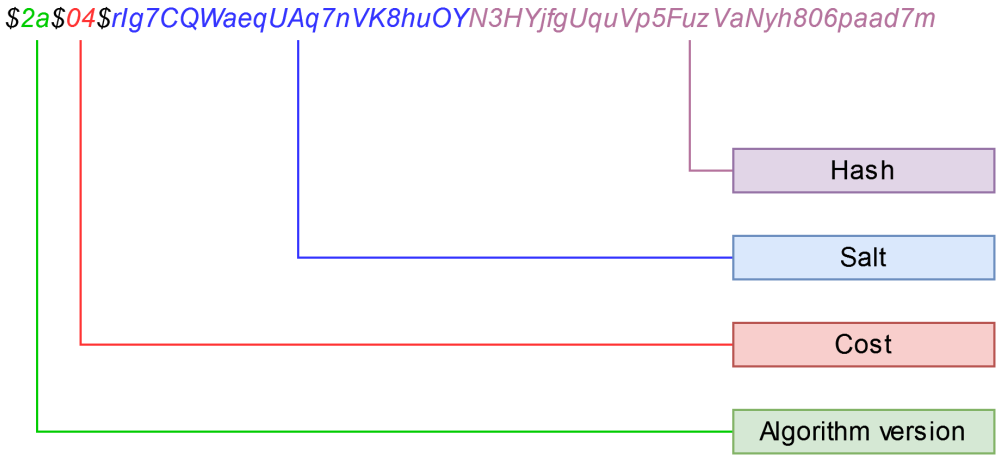

ILLUSTRATION 2.3 – Format du hash Bcrypt. Source : réalisé par Kandiah Abivarman

On va avoir un premier champs qui contient la version de l'algorithme, un deuxième qui contient le cost de la fonction, un troisième avec le salt et le quatrième avec le hash généré. Le salt et le hash sont en base 64, mais il faut faire attention car c'est une base 64 différente de la

norme RFC 4648 [6] qui est couramment utilisé.

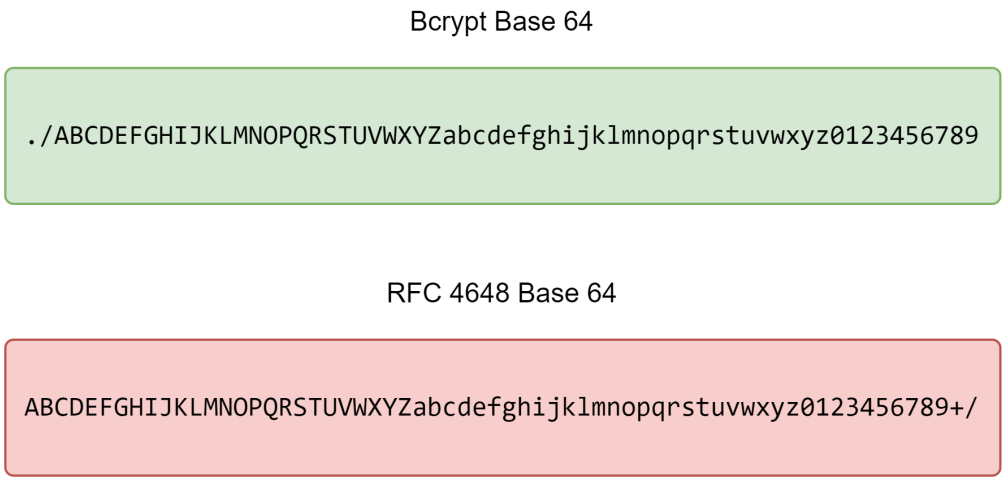

ILLUSTRATION 2.4 – Différence Base 64. Source : réalisé par Kandiah Abivarman

2.3. IMPLÉMENTATIONS EXISTANTES

Afin d'éviter de réinventer la roue, la première tâche que j'ai entrepris est de chercher afin de voir s'il n'existe pas déjà des implémentations existantes sur FPGA.

D'après mes recherches, j'ai retrouvé seulement deux implémentations du Bcrypt sur FPGA. La première se situe dans le répertoire github de JohnTheRipper [7] qui est un logiciel de cyber-sécurité destiné au craquage de mot de passe, l'implémentation a été faite en Verilog et spécifiquement pour la Ztex 1.15y qui est une carte FPGA assez ancienne et difficilement retrouvable. La deuxième est une implémentation faite en VHDL que j'ai aussi retrouvé dans un répertoire github, accompagné d'un papier [8] décrivant un travail de recherche effectué sur l'attaque de mot de passe sur FPGA. Pour ma part, ne connaissant seulement le VHDL et ne comprenant pas réellement la structure de code du premier et par manque d'informations, j'ai préféré reprendre le code du deuxième.

Le papier venant avec le code source a été très instructif, j'ai pu notamment comprendre les différents choix qui ont été pris dans le code source. Malheureusement, tout n'a pas été documenté et le répertoire n'a pas été mis en place correctement. En effet, certains partie du

6. TO DO : Add wiki
7. TO DO : Add john repo
8. TO DO : Add weimer repo

code contenait pas mal d'erreur, des fichiers de tests étaient incomplets et des fichiers source semble avoir été retravaillé en aval et au final la plupart des fichiers de tests qui ont été fournis n'était plus utilisables.

CHAPITRE 2 : CONCEPTION

Je vais maintenant expliquer en détail les travaux qui ont été fait autour du code source que j'ai repris.

3.1. BCRYPT SUR FPGA

Après lecture du code source, j'ai pu déduire l'architecture suivante :

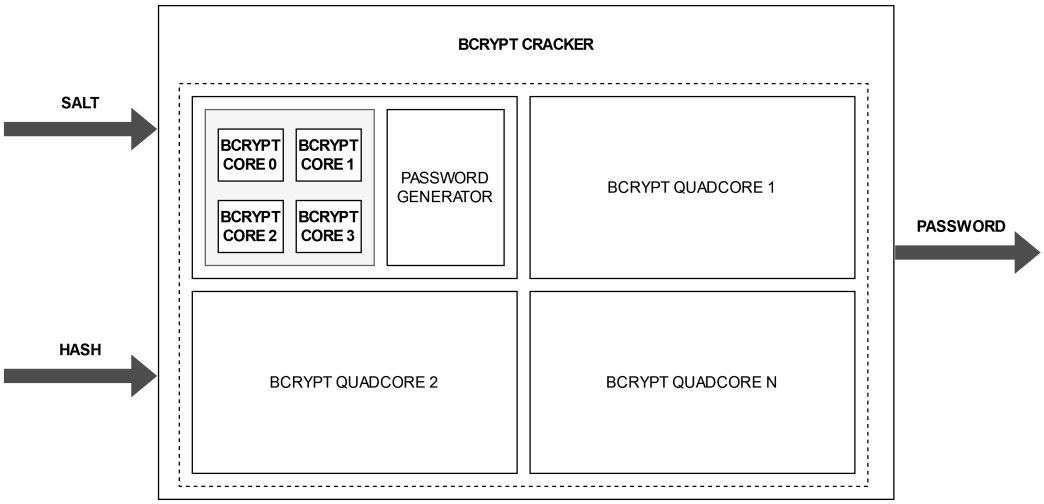

ILLUSTRATION 3.1 – Architecture Bcrypt sur FPGA. Source : réalisé par Kandiah Abivarman

L'architecture contient 4 modules clés, il y a d'abord le bcrypt core qui est le coeur de calcul qui va s'occuper de faire la fonction de hachage bcrypt. Ensuite, le password generator qui va s'occuper de générer les différents mots de passe pour l'attaque par bruteforce. Puis, le bcrypt quadcore qui va s'occuper d'instancier quatre bcrypt core et un générateur de mots de passe pour alimenter les coeurs en mot de passe. Enfin, le bcrypt cracker va instancier le nombre souhaité de quadcore, s'occuper de la gestion des différents coeurs et retransmettre le mot de passe lorsque il est retrouvé.

a. Bcrypt Core

Le bcrypt core est la partie qui m'intéresse le plus dans ce code source car c'est le module qui s'occupe de faire le hachage et c'était ce que je cherchais initialement parmi les implémentations déjà existante.

J'ai donc entamé le projet en testant le module avec de la simulation en utilisant le testbench

fourni, mais je me suis vite rendu compte que le testbench fourni ne fonctionnait pas. En effet, le testbench fourni semble avoir été fait pour une ancienne version du bcrypt core avec une interface totalement différente, rendant le fichier de test obsolète.

Afin de pouvoir implémenter moi même le testbench de ce module, je suis passé par une première phase ou j'ai analysé le code afin de comprendre l'interface du bcrypt core.

J'ai pu notamment identifier les Input / Output (I/O) qui permettant le contrôle du module, les I/O de la fonction de hachage (mot de passe, salt et hash) et les I/O qui vont permettre l'initialisation de la mémoire pour les clés de chiffrement. Le cost de la fonction de hachage est lui fixé par une constante situé dans un fichier à part regroupant d'autres constantes du système.

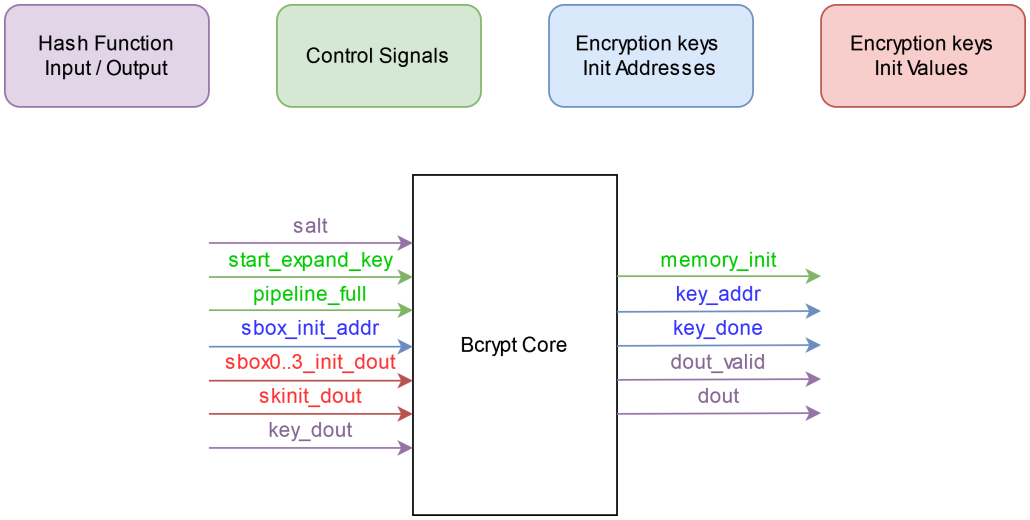

ILLUSTRATION 3.2 – Interface du Bcrypt core. Source : réalisé par Kandiah Abivarman

Après identification des entrée et sorties, j'ai examiné les différents processus et instanciations qui ont lieu dans le module bcrypt core. Il y a tout d'abord plusieurs Block RAM (BRAM) qui sont utilisés pour le stockage des clés de chiffrement, un module qui s'occupe du chiffrement blowfish, une machine d'état pour gérer les différents étapes de la fonction de hachage et différents compteurs nécessaire à l'adressage mémoire et à la machine d'état.

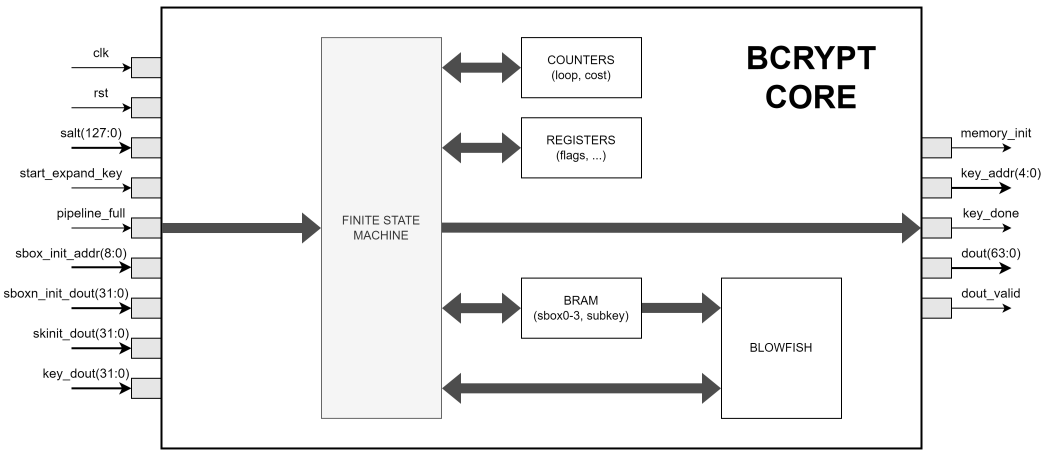

ILLUSTRATION 3.3 – Schéma du Bcrypt core simplifié. Source : réalisé par Kandiah Abivarman

La machine d'état contient 18 états, mais je vais la simplifier pour l'explication. Tout d'abord le module va attendre un signal du module parent afin de démarrer, après réception du signal, la phase d'initialisation de la mémoire est lancée. Cette étape consiste à initialiser les clés de chiffrement, pour ce faire il faut fournir au module l'adresse mémoire ou l'on souhaite écrire dans les BRAM et les données que l'on souhaite écrire dans notre cas les différents décimales de PI. Après la phase d'initialisation de mémoire, le module va de nouveau attendre un signal en entrée afin de procéder au calculs des clés de chiffrement. Cette étape consiste en 7 états dans la machine d'état et va reboucler un certain nombre de fois en fonction du cost. Après les calculs des clés de chiffrement, vient le chiffrement du mot magique qui va nous donner notre hash. Le port de sortie pour le hash fait une taille de 64 bits mais un hash fait 192 bits, de ce fait le module ressort le hash en trois morceaux. La processus de chiffrement est donc séparé en trois étapes pour chaque morceau du hash, chaque étapes consiste en réalité à 2 états, un premier état de préparation et ensuite un état de calcul.

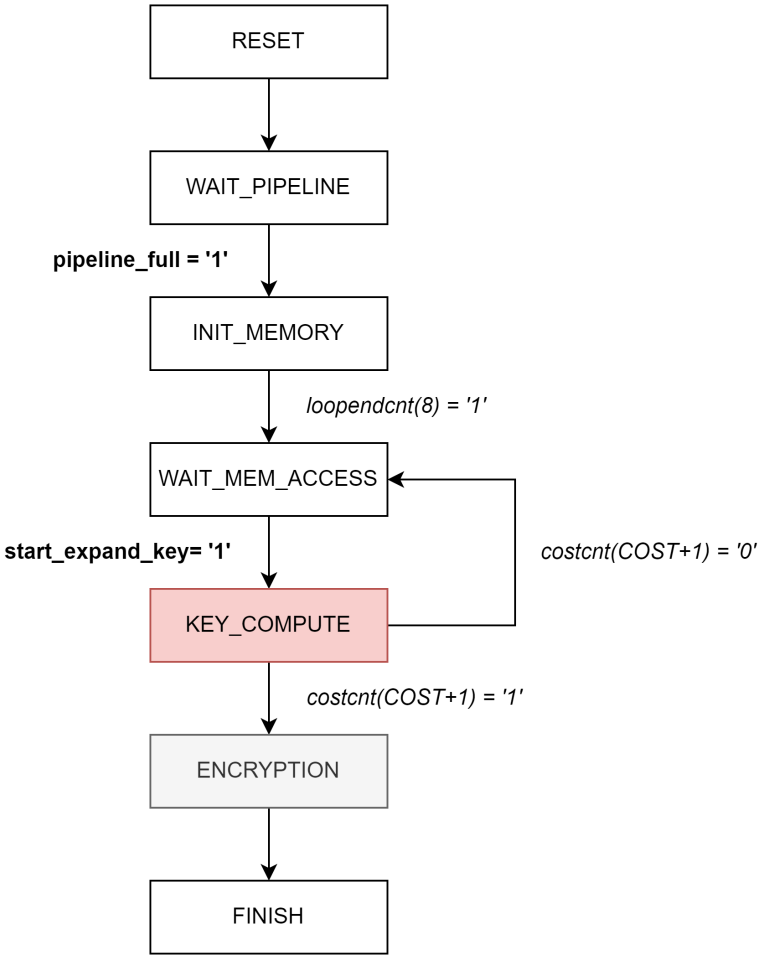

ILLUSTRATION 3.4 – Machine d'état du Bcrypt core simplifié. Source : réalisé par Kandiah Abivarman

Par la suite j'ai fais des simulations en testant des valeurs dans les différentes entrée afin d'essayer de comprendre les timings attendu par le modules. Après ces analyses, j'ai pu comprendre ce que le module attendait en entrée et les différents timings de ceux-ci.

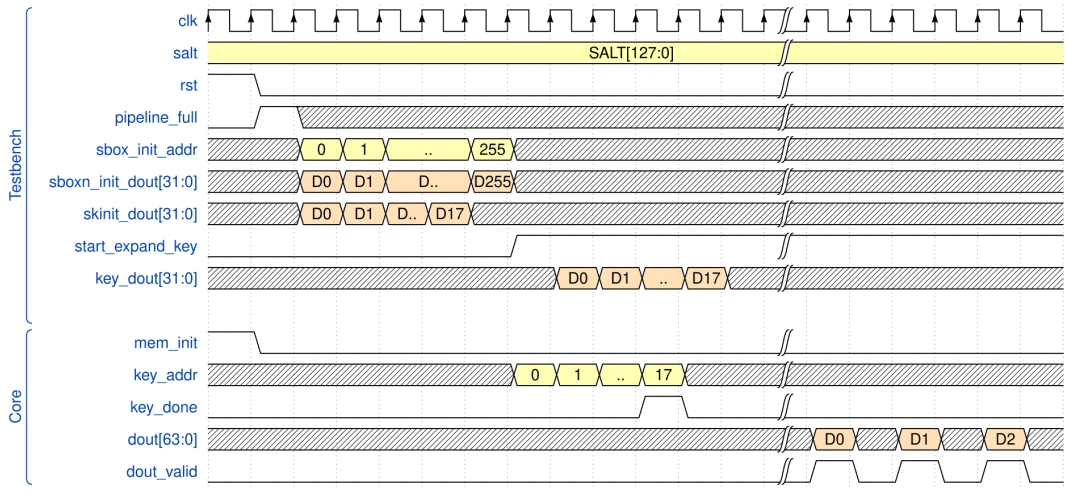

ILLUSTRATION 3.5 – Timing du bcrypt. Source : réalisé par Kandiah Abivarman

Le port *pipeline_full* va permettre de démarrer l'initialisation de la mémoire dans le module. Lors de l'initialisation de la mémoire, il faut fournir au module les adresses mémoires où l'on souhaite écrire avec *sbox_init_addr* et on va fournir les valeurs initiales des SBOX sur *sboxn_-init_dout* et des subkeys sur *skinit_dout*. Suite à cela, on va utiliser le port *start_expand_key* afin de démarrer le calcul des clés de chiffrement. Pour le calcul des clés, on va donner à *key_dout* la bonne partie du mot de passe en fonction de l'adresse mémoire fourni par *key_addr*, puis le port *key_done* va signaler la fin des calculs des clés. Pour finir, les différents morceaux du hash sont données par *dout* et le port *dout_valid* va avertir lorsque les différents morceaux sont prêts.

Après analyses des différents timings attendu, j'ai pu ré-implémenter le testbench du module bcrypt core est valider son bon fonctionnement.

b. Password Generator

Le password generator à un rôle assez important, car c'est le module qui va s'occuper de générer les différents de mots de passe à tester pour l'attaque.

J'ai donc repris la même démarche que pour le module précèdent afin de comprendre le fonctionnement du bloc. C'est à partir de ce module que j'ai commencé à rencontrer des difficultés notamment dû à des constantes qui ont été fixés avec des valeurs sans aucun sens et sans explications. Après quelques temps passés avec le simulateur, j'ai réussi à comprendre le comment marche le module et comment utiliser les constantes et les fixer pour avoir le fonctionnement souhaité.

Dans ce module les mots de passe sont générer à l'aide de compteur, il y a un compteur par caractère que l'on souhaite générer pour le mot de passe. Chaque valeur de compteur vont être converti en valeur ASCII afin de retrouver les caractères que l'on utilise dans nos mots de passe. La conversion est assez simple, nous avons dans l'ordre l'alphabet en minuscule, l'alphabet en majuscule et les chiffres.

Table de conversion	
0x00	NULL
0x01	'a'
0x02	'b'
0x1B	'A'
0x1C	'B'
0x35	'0'
0x36	'1'

ILLUSTRATION 3.6 – Table de conversion. Source : réalisé par Kandiah Abivarman

Enfin les différents caractères générer sont concaténés afin d'avoir au final un mot de passe. La fonction de hachage Bcrypt a besoin en entrée un mot de passe de 72 bytes, donc lorsque un mot de passe plus petit est utilisé, on va répéter en boucle le mot de passe en boucle jusqu'à atteindre les 72 bytes. Il est aussi nécessaire de délimiter chaque répétition par un caractère null. Ce module prend compte ce détail est s'occuper de remplir les 72 bytes comme il se doit

lorsque le mot de passe généré est plus petit.

Un point à retenir est que lorsque l'on instancie le module, on peut définir le compteur initial et la taille du mot de passe initial. Par exemple, si on initialise le compteur à zéro et la taille du mot de passe à un, nous aurons comme premier mot de passe le caractère 'a' puis 'b' et ainsi de suite. Donc lors de l'instanciation, il est nécessaire de initialiser les compteurs intelligemment afin d'avoir l'attaque la plus optimale.

c. Bcrypt Quadcore

Le bcrypt quadcore est le bloc qui va s'occuper d'instancier les deux modules dont j'ai parlé précédemment, c'est aussi dans ce module que j'ai rencontré de nombreux erreurs que j'ai dû corriger afin d'avoir un programme fonctionnel.

Ce module contient le générateur de mot de passe, une BRAM pour stocker les mots de passe généré, quatres bcrypt core, une machine d'état et des compteurs.

Le système va tout d'abord avoir un premier état de initialisation dans laquelle quatre mots de passe vont être généré pour chaque bcrypt core. Après la génération, chaque bcrypt core va calculer le hash en fonction de son mot de passe. Lorsque les calculs sont finis, les hash vont être comparer à l'hash que l'on souhaite casser, lorsque un hash correspond, le module va ressortir le mot de passe correspondant.

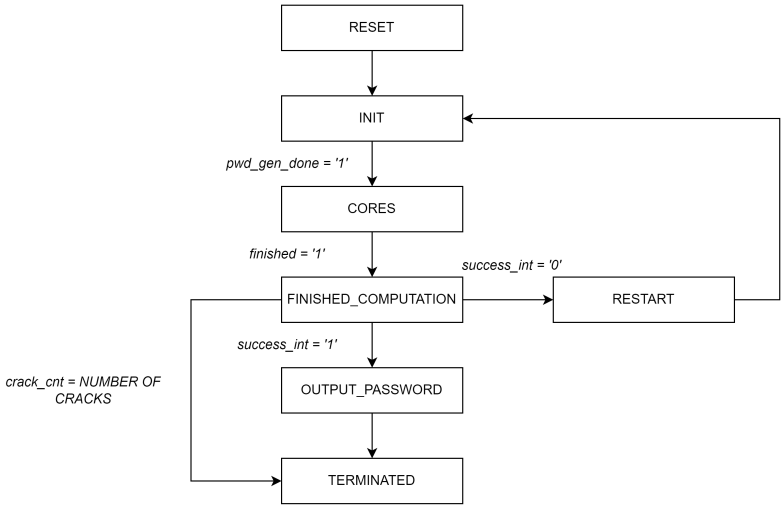

ILLUSTRATION 3.7 – Machine d'état du Bcrypt quadcore simplifié. Source : réalisé par Kandiah Abivarman

d. Bcrypt Cracker

Ce module va s'occuper d'initialiser deux BRAM avec l'état initial des clés de chiffrement qui vont permettre d'initialiser les BRAM pésentes dans les blocs bcrypt core. C'est aussi dans le bcrypt cracker qu'il y a l'instanciation des quadcore, le nombre souhaité de quadcore et le nombre d'essai est réglable dans le code.

Ce module est au final la partie qui va s'occuper du craquage de mot de passe, elle va prendre en entrée le salt et le hash du mot de passe que l'on souhaite retrouver et va ressortir le mot de passe lorsque il est retrouvé.

Afin de bien vérifier le fonctionnement de ce module et du bcrypt quadcore, j'ai utilisé le testbench fourni et j'ai ajouté la vérification de la sortie qui manquait au fichier de test.

Pour ce faire, j'ai regardé au niveau de la simulation afin d'étudier le comportement des sorties du module en fonction des entrées.

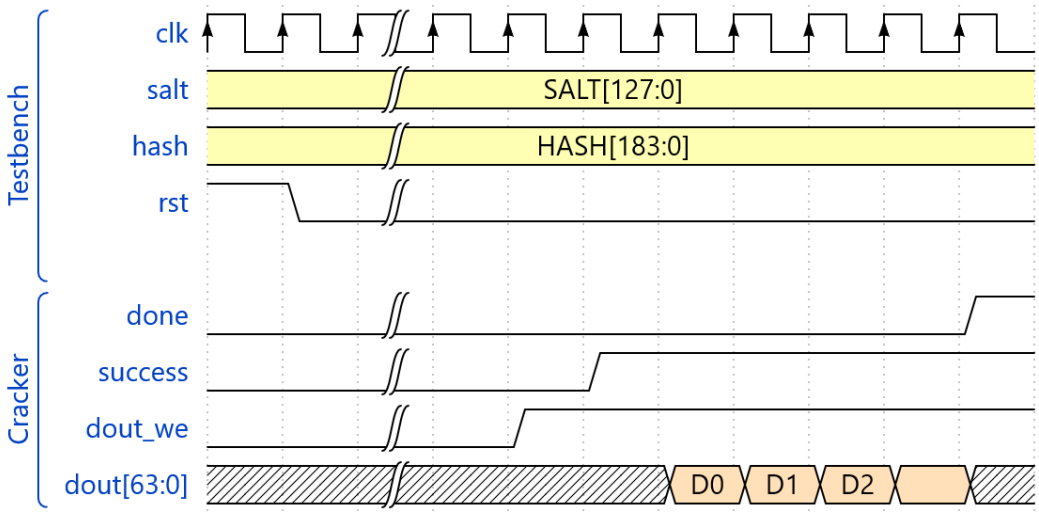

ILLUSTRATION 3.8 – Timing du bcrypt cracker. Source : réalisé par Kandiah Abivarman

TO DO TALK ABOUT THE ENCOUNTERED PROBLEMS

3.2. INTERFACE PCIE SUR FPGA

CHAPITRE 3 : RÉSULTATS

4.1. BCRYPT CRACKER

a. Validation

b. Mesures

4.2. INTERFACE PCIE

a. Validation

CONCLUSION

Votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte, votre texte.

ANNEXES

Imprimer idéalement cette page sur une page de couleur. Chaque annexe doit commencer sur une nouvelle page et doit être numérotée : Annexe 1 puis Annexe 2, etc.

ANNEXE 1

ANNEXE 2

ANNEXE 3

RÉFÉRENCES DOCUMENTAIRES

bcrypt. en. Page Version ID: 1210874707. Fév. 2024. URL : https://en.wikipedia.org/w/index.php?title=Bcrypt&oldid=1210874707.

GILLELA, Maruthi, Vaclav PRENOSIL et Venkat Reddy GINJALA. "Parallelization of Brute-Force Attack on MD5 Hash Algorithm on FPGA". In : *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*. ISSN: 2380-6923. Jan. 2019, p. 88-93. DOI : 10.1109/VLSID.2019.00034. URL : https://ieeexplore.ieee.org/document/8710753.

WIEMER, Friedrich et Ralf ZIMMERMANN. "High-speed implementation of bcrypt password search using special-purpose hardware". In : *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*. ISSN: 2325-6532. Déc. 2014, p. 1-6. DOI : 10.1109/ReConFig.2014.7032529. URL : https://ieeexplore.ieee.org/document/7032529.

Sites Web consultés – Code repris d'ailleurs – Notices techniques – Articles de presse – Ouvrage imprimés – Ouvrages électroniques – Chapitre dans un ouvrage imprimé – Rapports imprimés – Travaux universitaires – Articles de revues imprimés – Articles de périodiques électroniques – Communication dans un congrès. Pour chacun de ces types de document, les mise en forme sont dans le document « Méthode de citation et de rédaction d'une bibliographie ».

Afin de gagner du temps, pensez à utiliser le logiciel de gestion bibliographique Zotero (et/ou BibTeX si vous utilisez LaTeX) pour la mise en forme et l'édition automatique de vos références à la norme ISO690.