MASTER 2 CYBERSÉCURITÉ ET INFORMATIQUE LÉGALE

Projet ICP :
Implémentation **ARIA** LibreSSL 2.9.0

Présenté par : Chris Allard et Corentin Aulagnier

Juin 2019

Table des matières

1 Gestion du projet

1.1 Organisation de l'équipe

Afin de pouvoir travailler dans les meilleures conditions, nous avons mis en place un dépôt privé GitHub pour partager notre avancement sur le projet et travailler de manière indépendante. La première étape est d'analyser, ensemble, le fonctionnement du chiffrement symétrique Aria pour garantir que nous partagions la même compréhension du projet.

Nous avons choisi de réaliser le code de la primitive nous-mêmes, sans se baser sur une implémentation existante, car ça les avantages suivants :

- nous avons une maîtrise sur le fonctionnement de notre primitive ;
- cela permet de garantir notre compréhension du fonctionnement d'Aria ;
- l'intégration sera plus rapide ;
- l'optimisation sur notre plateforme est facilitée.

A l'inverse, le temps de développement est rallongé, mais nous sommes parti du postula que le bénéfice de ce choix s'en ferai ressentir plus tard.

1.2 Organisation du projet

Nous avons découpé le projet en quartes tâches :

1. implémentation fonctionnelle, en mode autonome, du chiffrement Aria, pour toutes les tailles de clé, 128, 192 et 256 bits et les deux modes de chiffrements CBC et ECB ;
2. intégration dans LibreSSL[1];
3. optimisation du code avec de l'utilisation du langage assembleur x86 ;
4. ajouter d'autres modes de chiffrements et l'intégration dans SSL/TLS.

Lors du démarrage de ce projet, il n'est pas possible de connaître à l'avance les difficultés que nous pouvions rencontrer. C'est pourquoi nous avons choisi ce découpage afin d'avoir dans un premier temps la primitive totalement hors de LibreSSL mais fonctionnelle. Les améliorations pouvant être rajouté ultérieurement sans altérer le fonctionnement principal du programme.

En réalité, nous nous sommes arrêté à la fin de la partie 2 car l'intégration dans LibreSSL a pris plus de temps que prévu.

1.3 Homogénéité du code et conformité

Le travail en équipe sur un code commun peut rendre la lecture de ce dernier difficile car chaque personne implémente, à sa manière, les différentes fonctions et ajoute son empreinte dans

1 Site Web: https://www.libressl.org/

le rendu final. Nous avons donc imposé l'utilisation d'un programme externe pour effectuer la mise en forme du code selon des règles pré-établies et validées par l'équipe. Chaque compilation, à travers un makefile, exécutera une mise en conformité du code.

- Clang-format[2] : il s'agit d'un programme permettant la mise en forme du code en utilisant un fichier de configuration décrivant les différentes règles à suivre (nommage des variables, constantes, fonctions, indentation, etc.).
- Clang-tidy[3] : programme permettant de diagnostiquer et avertir le développeur en cas d'erreur de programmation (possible fuites mémoire lors d'une allocation de mémoire via la fonction *malloc*, mauvaise utilisation d'un pointeur, oublie d'un appel de fonction *free*, variable non initialisée).

2 Site Web https://clang.llvm.org/docs/ClangFormat.html

3 Site Web https://clang.llvm.org/extra/clang-tidy/

2 Chiffrement par bloc Aria

2.1 Introduction

Aria (Academia, Research Institute and Agency) est un algorithme de chiffrement symétrique développé en 2003 par une équipe de chercheurs sud Coréens. En 2004 il est adopté comme standard par l'agence Coréenne des technologies après avoir augmenter le nombre de ronde dans l'algorithme, passant respectivement de 10/12/14 rondes (selon la taille de la clé) à 12/14/16. Depuis 2011, ce chiffrement est supporté par le protocole de communication TLS (Transport Layer Security). Il s'agit d'un chiffrement par bloc de 128bits utilisant trois tailles de clé différentes : 128, 192 et 256 bits.

2.1.1 Avantages

- arithmétique simple pouvant être adapté sur des environnements avec peu de ressources en calculs ;
- efficient sur des implémentations hardware et même sur des architectures 8-bit ;
- chiffrement éprouvé par différents groupes de recherches sur sa sécurité depuis 2004.

2.1.2 Inconvénients

- utilisation moins rependu que son homologue AES.

2.2 Fonctionnement

L'algorithme, comme pour AES, utilise un réseau de permutation-substitution dont les opérations sont semblable pour le chiffrement tant que pour le déchiffrement. La seul différence est que les sous clés sont inversé dans un cas par rapport à l'autre. Selon la taille de la clé, l'algorithme à un nombre de ronde déterminé. A chaque ronde nous effectuons les opérations suivante :

- Une addition du bloc de 128 bits courant avec une sous clé, propre à la ronde actuelle, généré au début de l'algorithme. L'opération est un XOR bit à bit entre le bloc courant et de clé ;
- Une substituions du bloc de 128 bits à travers une S-box de 128 bits. 2 S-box et leur inverses sont utilisées par le chiffrement selon si le numéro de la ronde actuelle est paire ou impaire ;

- Une diffusion dans une fonction linéaire semblable à une multiplication de matrice 16 x 16 bits

Les sous clés, généré au début de l'algorithme sont obtenus à travers l'utilisation d'un réseaux de Feistel.

2.3 Attaque connues sur l'algorithme

Aujourd'hui, bien qu'AES soit devenu un standard utilisé massivement, Aria reste une alternative envisageable et n'a pas encore fait preuve d'une implémentation d'attaques connues. Des chercheurs ont démontrés[4] qu'une attaque boomerang, version améliorée de la cryptanalyse différentielle, est possible sur 5 rondes pour une taille de clé égal à 128 bits.

Des expérimentations d'attaques Meet-in-the-Middle sur Aria ont été faites en changeant le nombre de round. Cependant, il n'y a pas, pour l'heure, d'attaque permettant de casser ce chiffrement.

(La complexité des textes choisis et des textes chiffrés sont de l'ordre de 2^{57}. La complexité en temps est de $2^{115.5}$. Pour une taille de clé de 192 bits, l'attaque est portée sur 6 rondes et a la même complexité des données que pour l'attaque à 5 rondes. La complexité en temps est cependant de l'ordre de $2^{171.2}$.) on enlève

2.4 Implémentation de l'algorithme

Le papier fourni par l'équipe ayant mis au point le chiffrement Aria est excellemment bien rédigé et permet de comprendre facilement les différentes étapes et opérations utilisées pendant le chiffrement/déchiffrement. Chaque fonction est un chapitre du document et nous avons suivi cette méthodologie pour effectuer le découpage de notre programme en fonctions. Le traitement des données se fait sur des *unsigned char*, étant la plus petite taille de variable dont sa taille, 8 bits, est standardisé. La seule opération ne respectant pas ce principe est la rotation circulaire dont nous convertissons les 128 bits de blocs (16 éléments de 8bits) en deux variables de 64 bits afin de pouvoir effectuer un décalage d'au moins 31 bits facilement.

Nous avons découvert, dans le document de référence pour tester notre implémentation, une erreur d'un bit dans le jeu de donnée. Nous avons fait remonter cette information à l'équipe en charge de maintenir ce document, et notre contribution a été entendu prise en compte.

2.5 Performance

La comparaison de notre implémentation avec celle disponible dans OpenSSL n'est pas possible. L'intégration dans le programme speed.c n'a pas été faite et ne semble pas être structuré de la même sorte qu'avec LibreSSL. Néanmoins, notre primitive se rapproche de

4 https://eprint.iacr.org/2009/334.pdf

Camellia. Nous avons donc comparé les performances des deux implémentations de ces primitives disponibles dans LiberSSL. Voici les résultats que nous avons eu :

Nombre de bloc traités en 3 secondes

	Taille de bloc				
	16	**64**	**256**	**1024**	**8192**
Aria-128	1301158	318991	82092	19754	2312
Camellia-128	10693185	5218567	1714588	462788	58703
Aria-192	975849	275534	69580	17529	2275
Camellia-192	9768067	4245318	1356328	359356	46119
Aria-256	1030230	239047	60590	15788	1986
Camellia-256	9911135	4350292	1322229	359629	46026

Notre implémentation actuelle est loin d'atteindre les performances de Camellia dont le traitement est optimisé en assembleur. Cela nous laisse entendre que de multiples améliorations sont encore possibles sur notre primitive.

Nous avons fait le choix de ne pas comparer notre implémentation avec celle d'AES présente dans LibreSSL car ça ne serait pas équitable. En effet, depuis 2008, une implémentation hardware est intégré directement dans les processeurs avec les jeux d'instructions permettant d'accélérer le temps de traitements des commandes d'AES.

L'université de Oulu, en Finlande, a publier un article[5] sur les différentes performances possibles avec des algorithmes de chiffrement de type blocs, sur des implémentations x86-64.

Dans notre cas, il est possible d'effectuer des optimisations assembleurs qui profitent des architectures courantes de processeurs comme les registres permettant d'effectuer des opérations sur 128bits, soit la taille des blocs de traitement d'Aria. Intel propose une libraire[6] permettant l'utilisation direct de ces registres. Le compilateur peut aussi optimiser et utiliser ces registres avec les options suivantes : -msse2 -mavx.

Nos recherches nous ont permis aussi de découvrir le travail de Michael Abrash, spécialiste dans l'optimisation assembleur, dont le livre « Graphics programming Black Book », disponible aujourd'hui gratuitement au format numérique[7], permet d'introduire certaines notions sur la manière de penser un programme pour accélérer son exécution.

5 http://axh.mbnet.fi/mastersthesis/thesis_final_sRGB_PDFA2b.pdf

6 https://software.intel.com/sites/landingpage/IntrinsicsGuide/#

7 http://floppsie.comp.glam.ac.uk/download/pdf/abrash-black-book.pdf

3 Intégration dans LibreSSL

3.1 Introduction

3.1.1 OpenSSL

Dans un premier temps il convient de parler d'OpenSSL. Il s'agit d'une librairie implémentant les protocoles SSL et TLS utilisés pour des connections dans des applications de type client/serveur sécurisées. Il offre aussi des outils pour :

- La création de clé RSA, DSA ;
- La création de certificats x509 ;
- Le calcul d'empreintes (MD5, SHA, etc.) ;
- Le chiffrement et déchiffrement.

Le code est open-source, toujours en cours de développement et largement utilisé dans le monde des entreprises. Sa popularité et son support pour de multiple architectures systèmes (Linux, MacOS, BSD ou Microsoft) en font un choix logique comme libraire cryptographique.

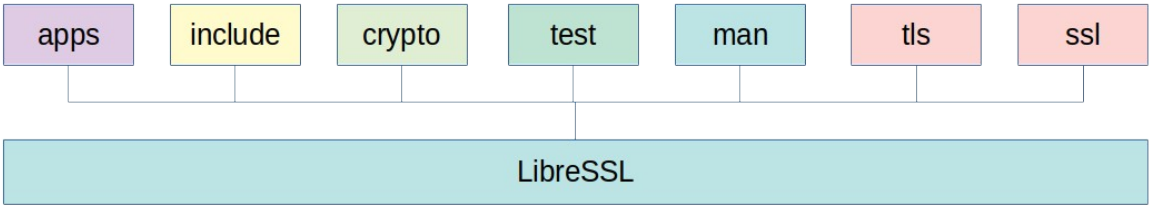
3.1.2 D'OpenSSL à LibreSSL

En 2014, une faille majeure, nommé Heartbleed, fût découverte dans la bibliothèque OpenSSL permettant d'accéder à la mémoire utilisée par ce dernier et d'en extraire des informations sensibles (mot de passe, clé de chiffrement, etc.). La popularité du logiciel a fait que de nombreux sites internet étaient impactés par ce problème, surtout que la technique utilisée pour exploiter la faille est accessible par les moins initiés. Parti de ce constat, le projet a été forké par les développeurs d'OpenBSD pour créer LibreSSL. Les objectifs sont simples :

- Compatibilité sur l'utilisation des commandes avec OpenSSL ;
- Seuls les chiffrements non obsolètes doivent faire partie de la librairie ;
- Suppression de la fonctionnalité responsable dans la faille Heartbleed ;
- Suppression de codes non nécessaire au fonctionnement de la librairie ;
- Audit du code actuellement implémenté ;
- Ajout d'options à la compilation pour prévenir de nouvelles failles.

Aujourd'hui le projet est toujours maintenu et continue d'évoluer en parallèle au projet « BoringSSL » par Google dont la collaboration pour le partage de correction de bugs est toujours d'actualité.

3.2 Architecture de LibreSSL



L'architecture de LibreSSL de sept sous-dossiers :

- apps : le dossier contenant l'ensemble des sources du noyau de LibreSSL, y compris l'interface utilisateur.
- include : le code source permettant de faire le lien entre le noyau de LibreSSL et les implémentations des primitives.
- crypto : les codes sources des différentes primitives cryptographiques proposées par LibreSSL.
- man : la documentation pour l'utilisation de LibreSSL.
- ssl et tls : code source pour utiliser les primitives avec les protocoles de chiffrements SSL/TLS.
- test : code source permettant de tester les implémentations des primitives cryptographiques de LibreSSL.

Chaque sous-répertoire intègre son propre Makefile qui doit être modifié en conséquence si l'on souhaite prendre en compte notre nouveau chiffrement.

3.3 Intégration d'Aria

Du fait qu'il s'agisse d'un clone de la libraire OpenSSL, certaines documentations d'OpenSSL sont encore compatibles avec l'architecture de LibreSSL. Sur la page Wikipédia officielle d'OpenSSL, il existe un guide[8] pour l'intégration d'un algorithme de chiffrement symétrique avec comme exemple une itération d'Aria.

De plus, le fonctionnement du chiffrement Aria peux s'apparenter aux chiffrements AES ou Camellia. Il est alors possible de comparer leur intégration et fonctionnement avec la notre. Pour cela, la commande suivante nous à permis dans un premier temps d'obtenir une liste de fichier à étudier pour comprendre le fonctionne de LibreSSL :

grep -rnw « ./libressl-2.9.0/ » -e « camellia »

8 https://wiki.openssl.org/index.php/How_to_Integrate_a_Symmetric_Cipher

3.3.1 Crypto/aria :

Il s'agit du répertoire où sera stocké toute la partie logique de notre chiffrement (fonctions de génération des clés intermédiaire, opérations logiques pour le chiffrement/déchiffrement).

Les traitements à effectuer pour les différents modes d'opérations sur le chiffrement, ECB ou CBC dans notre cas, sont déjà gérés par LibreSSL et ne font donc pas l'objet d'une adaptation spécifique pour nos algorithmes.

Liste des fichiers :

- aria.c : contient toutes les fonctions propres aux chiffrements et déchiffrements ;
- aria.h : toutes les entêtes des fonctions principales appelées par LibreSSL ;
- aria_locl.h : entêtes des sous-fonctions utilisées pour le fonctionnement interne à l'algorithme ;
- aria_ecb.c et aria_cbc.c : redirige aux différentes fonctions de LibreSSL s'occupant de ces différents mode d'opérations avant de faire appel à nos fonctions.

3.3.2 Crypto/evp/e_aria.c :

C'est ici que nous allons faire la transition entre les opérations et gestion propre à LibreSSL sur les différentes données (clés secrètes fourni par l'utilisateur, fichier en entrée, fichier en sortie) et nos différentes fonctions.

Pour chaque mode d'opération et taille de clé, nous déclarons une structure de type *EVP_CIPHER* qui contient les informations suivantes :

- Le numéro unique, interne à LibreSSL, sur le chiffrement utilisé, le mode opératoire et la taille de la clé ;
- Le nom de la fonction à appeler pour initialiser la clé ;
- Les différentes informations sur la taille de la clé secrète, la taille du bloc de traitement ainsi que la taille du vecteur d'initialisation ;
- Le nom de la fonction à appeler pour réaliser l'opération de chiffrement/déchiffrement.

L'implémentation du fichier e_aria.c se base en grande parti sur le travail fourni par LibreSSL dans le fichier evp_locl.h où sont décrit les différentes structures utilisées par chaque mode d'opération.

3.3.3 Crypto/objects/obj_dat.h

Ce fichier contient une grande table qui liste tous les différents objets utilisés par LibreSSL ainsi que les modes de chiffrements proposées et leur numéro d'identification sur le standard ASN1.

Celui-ci si est généré par des scripts sous le langage Perl, qui ne sont pas disponibles nativement dans l'archive proposée par LibreSSL. Il est toujours possible de les récupérer sous le projet OpenSSL et reste compatibles.

3.3.4 Apps :

Nous avons ajouté les options et les descriptions pour permettre l'utilisation des deux modes de chiffrements avec les trois tailles de blocs proposées. Dans les fichiers apps/openssl/[gendsa.c, genrsa.c, openssl.c, pkcs12.c, smime.c, speed.c] nous avons ajouter les modifications pour l'utilisation de :

- aria-128-cbc, aria-192-cbc, aria-256-cbc ;
- aria-128-ecb, aria-192-ecb, aria-256-ecb.

Nous avons profité aussi pour corriger un warning présent lors de la compilation du fichier ocspcheck où la valeur de retour lors de l'appel à la fonction *ftruncate* n'était pas prise en compte.

3.3.5 Include :

Nous avons modifié l'interface permettant l'utilisation d'Aria par LibreSSL. Cela correspond aux fichiers include/openssl/[aria.h, evp.h, Makefile.am, obj_dat.h, opensslv.h]. A revoir je pense

3.3.6 Man :

Nous avons intégré un fichier *manpage* pour notre chiffrement. De base, le manpage installé par LibreSSL ne liste pas les différents chiffrements disponibles. L'installation de notre manpage se fait donc manuellement, comme les autres disponibles qui décrivent les différents le fonctionnement interne des différentes fonctions proposées par LibreSSL et qui ne sont pas installés nativement.

3.3.7 Test :

Pour être sûr que notre algorithme fonctionne à chaque compilation, il éxiste dans LibreSSL un programme nommé evptest qui prend en entrée un fichier texte dans lequel on renseigne nos jeux de tests.

Ce dernier est exécuté lors de l'utilisation de la commande *make check*. Si une erreur se trouve dans notre algorithme, alors le jeu de test sort en erreur. Structure du jeu de test dans le fichier evptests.txt :

#cipher:key:iv:plaintext:ciphertext:0/1(decrypt/encrypt)

#digest:::input:output

3.4 Améliorations possibles

3.4.1 De nouveaux modes de chiffrement

Il est possible d'ajouter, pour Aria, d'autres modes d'opérations comme : CFB, OFB par exemple. Cependant, par manque de temps, nous n'avons pas pu effectuer ce travail.

3.4.2 Intégration dans les protocoles SSL/TLS

De plus, l'intégration d'Aria dans les protocoles de communications SSL/TLS peut être envisagé. Comme l'ANSSI conseil l'utilisation d'Aria comme solution de secours, cela ne semble pas aberrant.

4 Conclusion

Ce projet a permis de comprendre, de manière concrète, le fonctionnement d'un chiffrement symétrique par bloc. La difficulté de se dernier réside dans l'intégration avec la librairie LibreSSL dont la documentation est moins fourni que le projet d'origine OpenSSL. Loin d'être parfaite, notre implémentation permets d'être facilement accessible, grâce à sa structure comparable à celle de l'article rédigé par le laboratoire ayant mis au point ce chiffrement. Par ailleurs, à la suite d'une erreur constaté dans une des constantes présente dans les jeux de tests, nous avons contacté l'équipe d'Aria, nous remerciant en retour.