Documentation pour l'implémentation Delphi de Lynx

Thomas Bénéteau (*aka* Bacterius)

14 Juin 2010

1 Introduction

Lynx est un nouvel algorithme de chiffrement à bloc, plutôt rapide et destiné à être efficace sur la plupart des machines modernes et à être raisonnablement sûr du point de vue cryptographique. Il utilise des opérations sur 64 bits dans son key schedule (c'est-à-dire la préparation de la clef), ainsi il est regrettable de devoir déclarer que les versions de Delphi inférieures à Delphi 4 ne pourront pas compiler ce code, mais devront récupérer une implémentation d'arithmétique sur 64 bits et adapter le code en conséquence. Le chiffrement n'utilise que des opérations sur 8 bits très simples mais conçues pour rendre la récupération du message, sans en connaître la clef, une tâche ardue, voire totalement impossible sans les moyens financiers et matériels requis.

2 L'implémentation

Ce document a dû vous parvenir accompagné d'un fichier `Lynx.pas`, et possiblement avec une application exemple. Cette unité contient l'algorithme de chiffrement à proprement parler, sous la forme de ces trois routines :

- ▷ `KeySchedule` : Cette routine prend en paramètre une clef quelconque, d'une longueur maximale de 256 octets, et d'une longueur minimale de zéro octets (bien qu'il soit douteux de ne pas fournir de clef). Elle prend également un paramètre une structure `TSubkeys`, qui contiendra la véritable clef une fois transformée pour être utilisée par Lynx.
- ▷ `EncryptBlock` : Cette routine prend en entrée un bloc de 64 bits (8 octets, soit un Int64), ainsi qu'une structure `TSubkeys`, et chiffre ce bloc selon l'algorithme qui définit Lynx.
- ▷ `DecryptBlock` : Cette routine, semblable à la précédente, permet de déchiffrer un bloc.

Ces trois fonctions constituent le coeur de l'algorithme, et contiennent tout le matériel cryptographique de Lynx. Malheureusement, il n'est pas possible de les utiliser telles quelles. En effet, il faut utiliser un *mode d'opération* pour pouvoir chiffrer des données plus longues qu'un bloc afin de garantir la sécurité des données chiffrées dans toutes les situations imaginables.

Heureusement, l'unité `Lynx.pas` inclut également une classe standard, `TLynxCFBCipher`, qui permet de chiffrer n'importe quoi, et offre une routine spécialisée pour les fichiers. Le mode d'opération utilisé par cette classe est le mode CFB (Cipher FeedBack), qui est plutôt pratique, car il transforme un chiffrement à bloc, en un chiffrement à flux, ce qui élimine le besoin de "compléter" les données si elles n'ont pas une longueur multiple de 64 bits, tout en conservant le niveau de sécurité de l'algorithme. Le mode CFB est le seul mode implémenté dans `Lynx.pas` à ce jour, mais rien n'empêche d'implémenter un autre mode d'opération soi-même.

3 La classe `TLynxCFBCipher`

Cette classe contient plusieurs routines destinées à rendre le chiffrement d'un fichier le plus simple possible. Voici une liste des routines et leur description :

- ▷ `Create` : Ce constructeur permet de créer la classe, et propose de définir la clef immédiatement, sous la forme d'un buffer de longueur 256 octets maximum.
- ▷ `ResetKey` : Cette routine permet de changer la clef utilisée lors du chiffrement avec cette classe. Attention : il est illégal de changer la clef lorsqu'une opération de chiffrement/déchiffrement est en cours, sous peine de corruption des données.
- ▷ `PrepareBuffer` : Cette routine prépare une donnée au chiffrement/déchiffrement en mémorisant son pointeur. Attention : aucune copie de données n'est effectuée, seul le pointeur sur les données est mémorisé afin de pouvoir les chiffrer à tout moment.
- ▷ `PrepareFile` : Cette fonction prépare un fichier au chiffrement/déchiffrement, en le mappant en mémoire. Le fichier doit pouvoir être ouvert en lecture et écriture.
- ▷ `UnprepareBuffer` : Cette fonction ne fait que fermer un éventuel fichier préparé par `PrepareFile`. Elle ne fait rien si aucun fichier n'a été ouvert : le développeur garde la responsabilité des données passées à `PrepareBuffer`.
- ▷ `SetCallback` : Cette fonction définit un callback optionnel pour le chiffrement/déchiffrement. Ce callback est appelé à chaque bloc traité, et il est alors possible de s'en servir pour afficher une progression en temps réel de l'opération.
- ▷ `Encrypt` : Cette routine va chiffrer les données passées par `PrepareBuffer` ou `PrepareFile`, selon la clef définie dans l'instance de la classe, et en passant le callback si celui-ci a été défini. Cette fonction a un paramètre, `IV`, qui est le vecteur d'initialisation de la donnée. Il s'agit d'un paramètre cryptographique spécial détaillé dans la section suivante. Attention, il est illégal de réitérer un appel à `Encrypt` si le dernier n'a pas encore été complété.
- ▷ `Decrypt` : Cette fonction fait l'opération inverse à la fonction `Encrypt`. Les avertissements de la fonction précédente concernent également cette routine.

Notez que la libération d'une instance de `TLynxCFBCipher` par la méthode `Free` cause l'appel automatique de la fonction `UnprepareBuffer`, pour des raisons évidentes de sécurité.

4 Notes additionnelles

Cette section détaille plusieurs points abordés précédemment, comme divers paramètres mentionnés sans explication dans les sections précédentes :

4.1 Le vecteur d'initialisation

Un vecteur d'initialisation est très important, pour rendre chaque donnée chiffrée unique. Il s'agit d'une composante cryptographique majeure, et qui ne doit pas être négligée. Vous devez, à chaque donnée différente, passer un vecteur d'initialisation différent et raisonnablement aléatoire. Ce dernier *doit* être connu du destinataire pour pouvoir déchiffrer les données chiffrées, mais peut être diffusé librement (il n'est pas requis que le vecteur d'initialisation soit gardé secret). Il n'est pas nécessaire d'obtenir un aléa de haute qualité pour l'IV, il faut juste que les nombres utilisés soient uniformément distribués. Dans Lynx, le vecteur d'initialisation est sur 64 bits.

4.2 Le callback

Le callback, de type `TLynxCallback`, prend la forme d'une fonction admettant deux paramètres sur 32 bits, qui représentent respectivement le bloc qui vient d'être traité, sur le nombre de blocs au total. Ainsi, ces deux paramètres permettent de visualiser la progression de l'opération. Attention toutefois à ne traiter qu'une faible fraction des callbacks reçus : tous les traiter n'apportera que peu d'informations supplémentaires et ralentira énormément le traitement. Le callback renvoie une valeur booléenne : si celle-ci est définie à `False`, le chiffrement/déchiffrement continue, si elle est définie à `True`, l'opération s'interrompt immédiatement. Attention : les données étant chiffrées "au vol", un fichier partiellement chiffré peut se révéler relativement difficile à récupérer.

4.3 Limitation de l'implémentation

L'implémentation bas niveau est complète et fonctionnera dans tous les cas, toutefois le mode CFB ne prend pas en charge les fichiers plus grands que 4 Go (en théorie moins dû à des restrictions matérielles).

4.4 Choisir la clef pour les fonctions `KeySchedule` ou `ResetKey`

Il est très important de ne pas donner n'importe quoi à ces fonctions : si la clef est faible, les données peuvent être déchiffrées par n'importe-qui. La méthode la plus standard de procéder est de prendre une clef quelconque (une chaîne de caractères entrés par l'utilisateur, comme un mot de passe), de la passer dans une fonction de hachage telle que MD5 ou SHA, et passer le résultat de cette fonction aux routines `KeySchedule` ou `ResetKey`. Les utilisateurs plus expérimentés pourront éventuellement trouver des solutions alternatives dépendant de l'utilisation qu'ils souhaitent faire de cette implémentation.

4.5 Fonctions non documentées

Vous aurez probablement remarqué la présence de deux routines non documentées, à savoir `EncryptBlock` et `DecryptBlock`. Ce sont simplement les routines de bas niveau pour le mode d'opération CFB, qui sont appelées de façon interne par la classe `TLynxCFBCipher`. Il est bien sûr possible de les appeller soi-même pour une utilisation quelconque.

4.6 Autres

Remarquez que le mode d'opération CFB ne requiert que le chiffrement des données. Les routines de déchiffrement standard sont tout de même fournies pour éviter de devoir les déduire soi-même, mais ne sont pas utilisées par `TLynxCFBCipher`.

L'exemple éventuellement fourni avec cette unité permet de chiffrer/déchiffrer des fichiers, avec un exemple de l'utilisation correcte du callback optionnel, avec une gestion sûre des clefs en fournissant non pas le mot de passe mais son empreinte MD5 à Lynx.

5 Remerciements

Je tiens à remercier en particulier Yann Guibet, qui m'a soutenu et aidé tout au long de la construction de cet algorithme. Enormément d'autres personnes m'ont aidé pour l'implémentation, mais il serait difficile de toutes les mentionner, partiellement par le fait qu'elles m'ont surtout aidé indirectement par l'intermédiaire de pseudonymes virtuels sur des forums informatiques divers.