MetFS Kitapçığı

Metin KAYA

EnderUNIX Üyesi
metin at enderunix.org
http://www.enderunix.org/metin

31 Mayıs 2008 Cumartesi EEST 02:09:03

Özgünlük Bildirisi

1. Bu çalışmada, başka kaynaklardan yapılan tüm alıntıların, ilgili kaynaklar referans gösterilerek açıkça belirtildiğini,
2. Alıntılar dışındaki bölümlerin, özellikle projenin ana konusunu oluşturan teorik çalışmaların ve yazılımın benim tarafımdan yapıldığını

bildiririm.

İstanbul, 31.05.2008
Metin KAYA

KULLANICI UZAYINDA DİNAMİK BOYUTLU VE ŞİFRELİ BİR DOSYA SİSTEMİNİN GERÇEKLENMESİ
(ÖZET)

Bilişim dünyasındaki en güncel konuların başında bilgi güvenliği bulunmaktadır. Bilginin güvenliğinden kasıt; bilgiyi ve bilgi sistemlerini yetkilendirilmemiş erişimlere, kullanıma, değiştirilmeye, görülmeye, yok edilmeye ve bozulmaya karşı korumaktır. Bilginin gizliliği, bütünlüğü ve erişilebilirliği korunması gereken 3 önemli noktadır.

Bitirme ödevinin konusu dinamik boyutlu ve şifreli bir dosya sisteminin kullanıcı uzayında gerçeklenmesidir. Veriler diske şifreli yazılacak ve bu dosya sistemi disk üzerinde sadece mevcut dosyaların/dizinlerin boyutu kadar yer kaplayacaktır. Dosya sisteminin bağlı olmadığı durumda ise verilerin tek dosya gibi görünmesi sağlanacaktır.

Kullanıcı uzayında dosya sistemi geliştirilirken kullanılacak en önemli araç olan FUSE yazılımının tercih edilmesinde şu faktörler etkili olmuştur:

- Kullanıcı uzayında kod yazmanın çekirdek uzayında kod yazmaktan daha kolay olması
- Çekirdekten bağımsız çalışabilme
- Ayrıcalıklı hakları olmayan normal kullanıcıların dosya sistemi bağlamasını sağlayabilme

Tüm bu etkenler FUSE ‘un onlarca dosya sistemi yazılımında (ZFS, NTFS-3G, WikipediaFS, YouTubeFS, mp3fs, Bluetooth File System, BitTorrent File System, EncFS, TrueCrypt, mysqlfs, GnomeVFS2, …) kullanılmasını sağlamıştır.

FUSE ‘un çalışma mantığı şöyle özetlenebilir: İlk olarak FUSE çekirdek modülü VFS ‘e kaydedilir. FUSE temel olarak sistem çağrılarını FUSE çekirdek modülünden FUSE kütüphanesine, verileri de FUSE kütüphanesinden FUSE çekirdek modülüne aktarır.

Öncelikle bu konuda yapılmış TrueCrypt ve EncFS gibi yazılımlar incelenir. Daha sonra FUSE üzerinde çalışmalar yapılır. Kullanıcının ihtiyaçlarına yanıt veren bir dosya sistemi yazıldıktan sonra şifreleme işlemleri için araştırılır ve *libgcrypt* aracılığıyla dosya sistemine şifreleme desteği eklenir.

Şifreleme konusunda çok fazla seçenek mevcuttur: blok şifreleme için DES, AES, Blowfish, Twofish, Serpent, RC2 ve akan veriyi şifreleme için RC4, A5/1, A5/2, Chameleon, FISH, Helix, ISAAC, MUGI, Panama, Phelix, Pike, SEAL, SOBER, SOBER-128 ve WAKE teknikleri mevcuttur. Verilerin özetini (hash) almak içinse HAVAL, MD2, MD4, MD5, PANAMA, RadioGratun, RIPEMD, RIPEMD-128/256, RIPEMD-160/320, SHA-0, SHA-1, SHA-256/224, SHA-512/384, Tiger(2)-192/160/128 ve WHIRLPOOL gibi tek yönlü şifreleme algoritmaları söz konusudur. Bitirme ödevinde özet almak için MD5, şifrelemek işlemleri içinse RC4 akan veri şifreleme algoritması kullanılmıştır.

Linux işletim sistemlerinin kullanıcı uzayında çalışabilen, FUSE tabanlı, dinamik boyutlu ve bağlı olmadığı durumlarda tek dosya gibi görünen MetFS isimli bir dosya sistemi yazılır. MetFS 'te, şifreleme işlemleri için *libgcrypt*, dosya sisteminin bağlı olmadığı durumlarda tek dosya gibi görünmesini sağlamak için *libtar* kütüphaneleri kullanılır. MetFS 'in performansı, üzerinde bulunduğu dosya sisteminin performansına çok yakındır (MetFS, ext3 dosya sisteminden % 10 daha yavaştır). *.mfs* dosya uzantısı MetFS dosya sistemi yazılımı için kaydedilmiştir. Bu bitirme ödevi sayesinde dosya sistemi ve kriptoloji hakkında pratik bilgi ve beceri edinildi.

Sonuç olarak yazılan dosya sistemi TrueCrypt gibi bağlı olmadığı durumlarda tek dosya şeklinde görünmekte ve EncFS gibi diskten dinamik olarak yer almaktadır. Komut satırından alınan parametrelerin esnek olmaması, dosya sistemi bağlıyken gerçek veri miktarının 2 katı kadar disk alanı işgal edilmesi ve bağlama/çözme sürelerinin veri miktarına bağlı olarak uzun sürebilmesi yazılımın eksik yönleri olarak sıralanabilir.

IMPLEMENTATION OF ENCRYPTED AND DYNAMIC SIZED FILESYSTEM IN USERSPACE
(SUMMARY)

One of the most crucial topics of computer world is security of information. Information security is focused on preventing information from unauthorized access, deleting, reading, writing, and altering.

There is a *CIA* rule which stands for Confidentiality – Integrity – Accessibility. Confidentiality means that data must be accessed, copied, used, and removed only by authorized people. Integrity means that only an authorized person can create, change, and delete any data. Accessibility is synonym to be able to service all time when the system administrator wants.

The subject of the graduation project is implementing a file system in user space that is encrypted, dynamic sized and seems as if a single file when the file system is unmounted. Encrypted data will be written to disk. Clean data and user's password must only be in RAM.

FUSE is a framework which allows implementing file systems in the user space. It consists of 2 components: the user space library (dynamically linkable) which is used by file system implementers and kernel module which binds to the VFS and redirects calls to the FUSE library.

FUSE was developed originally for Linux, but now it can run on FreeBSD, NetBSD, Mac OS X (Darwin), Windows, OpenSolaris and GNU/Hurd operating systems. FUSE library remains the same across operating systems, so it ensures file system applications need no rework.

If is there is a need to understand how FUSE works, the internals of it can be explained in that way: in short, FUSE works by passing of system calls and data from kernel module to FUSE library, and data from FUSE library to kernel module. This communication uses a device (/dev/fuse) which is opened during mounting the file system. Thus, FUSE kernel module registers itself as a character device driver. The first job is FUSE kernel module's being registered with VFS. The application implementing file system at user space will use FUSE library and provide followings:

- Specify the mount point in the current file system path to where its file system will be mounted.
- Expose methods that do all the necessary file handling such as creating directory/file, reading directory/file, writing file, etc.

FUSE library invokes mount system call for the provided mount point specifying file system type as *fuse* to ensure mounting of file system application. All file system call is passed by kernel to VFS, which passes the request to FUSE kernel module for calls at the

mount point. Moreover, FUSE kernel module passes the call on to FUSE library, which invokes appropriate method of FUSE application. [44]

Applications use FUSE by developing specific implementation for the necessary file system calls and matching the specific functions with FUSE ones. An interface for these file system calls is defined by FUSE library.

There are tens of file systems that use FUSE: ZFS, NTFS-3G, WikipediaFS, YouTubeFS, mp3fs, Bluetooth File System, BitTorrent File System, EncFS, TrueCrypt, mysqlfs, GnomeVFS2, …

While a file system is being implemented in user space, FUSE is the most useful tool for the programmer, since:

- It is easier to code in user space than kernel space.
- File system implementation with FUSE does not depend on any particular operating system.
- FUSE does not require kernel recompile.
- Via FUSE, it is possible to mount file system for non-privileged users. Mounting a FUSE file system does not require super user privilege. Hence, the user's file system runs with the user's privilege. The standalone mount utility performs the job of mounting.

On the other hand, FUSE has performance problems that redundant copy of data and many user space / kernel space switches reduce FUSE performance.

At first, TrueCrypt and EncFS, which are in the concept of the graduation project, are studied. Then, a file system was written in FUSE to provide a fully functional file system. Moreover, a research is performed in order to learn cryptographic concept of the graduation project and libgcrypt is found useful to use in MetFS.

Encryption algorithms can be overviewed in 2 sections: symmetric and asymmetric encryption algorithms.

Symmetric encryption algorithms use the same key to encrypt and decrypt data. There are lots of symmetric encryption algorithms:

- Block ciphers (input data is divided into fixed size of blocks): DES, AES, Blowfish, Twofish, Serpent, RC2
- Stream ciphers (size of input data can vary): RC4, A5/1, A5/2, Chameleon, FISH, Helix, ISAAC, MUGI, Panama, Phelix, Pike, SEAL, SOBER, SOBER-128, WAKE

On the other hand, there is a key pair in asymmetric encryption algorithms. One of them is the public key, and the other one is the private key. Public keys are visible to everyone, since it is only used for encrypting data –encrypted data cannot be decrypted via public keys. However, a private key is only visible to its owner and invisible for any other. When data is encrypted via a public key, it can only be decrypted with the associated private key. Some of asymmetric encryption algorithms are Knapsack, RSA, Diffie-Hellman, DSS, ElGamal, Paillier, Cramer-Shoup, Rabin and McEliece.

Also there is a number of one way hash functions: HAVAL, MD2, MD4, MD5, PANAMA, RadioGratun, RIPEMD, RIPEMD-128/256, RIPEMD-160/320, SHA-0, SHA-1, SHA-256/224, SHA-512/384, Tiger(2)-192/160/128, and WHIRLPOOL.

In the graduation project, MD5 is used for hashing and RC4 stream cipher is used for encryption.

MetFS is a FUSE based, encrypted, and dynamic sized file system software. Data in a MetFS file system is imported in a single file when it is unmounted. MetFS uses *libgcrypt* for cryptography, and *libtar* for importing data in a single file.

libtar is a C library for manipulating POSIX tar files. It handles adding files to a tar archive, and extracting files from a tar archive. Tar (**T**ape **AR**chive) is both a file format and the name of the program used to handle such files. The format was standardized by POSIX.1-1988 and later POSIX.1-2001. Tar is widely used in archiving data and it preserves file system related information such as user and group permissions, dates, and directory structures.

libgcrypt is a general purpose cryptographic library based on GnuPG. It provides functions for all cryptographic tools: symmetric block ciphers (AES, DES, Blowfish, CAST5, Twofish), symmetric stream cipher RC4, hash algorithms (MD4, MD5, RIPE-MD160, SHA-1, TIGER-192), MACs (HMAC for all hash algorithms), public key algorithms (RSA, ElGamal, DSA), large integer functions, random numbers and a lot of associated functions. [45]

MetFS has been written in C language for performance reasons; proved to be running on Linux and FreeBSD.

Finally, MetFS is a single file based file system like TrueCrypt and dynamic sized like EncFS. The command line parameters being not flexible, when file system is mounted the occupied disk space's being 2 times bigger than the real data size, and mount/unmount's taken long times according to size of data are disadvantages of MetFS. The graduation project made me learn lot cryptography and file system.

Features:

- MetFS partitions are visible only to the user who mounted them. No other users (including root) can view the file system contents.
- MetFS image file seems as a single file when it is unmounted.
- User password and data are never written to disk plainly.
- RC4 stream cipher algorithm is used for encryption.
- MD5 hash algorithm is preferred.
- MetFS is just slower about 10 % than *ext3* file system, and at least 46 % faster than EncFS and TrueCrypt.

- Faster and more secure string functions of OpenBSD and D.J.B are used.
- *.mfs* is official file extension of MetFS.

İÇİNDEKİLER

1 GİRİŞ

Günümüzde verilerin güvenliği kadar taşınabilirliği de çok önemlidir. Bu konuda bir kaç yazılım mevcuttur; ancak bunların önceden diskin bir kısmını ayırma, çekirdek (kernel) sürümüne bağlı olma, performans ve verilerin dosya sistemi bağlı değilken (*unmount* durumu) derli toplu olmamasından dolayı taşınabilme güçlükleri gibi çeşitli sorunları vardır.

Bu bitirme ödevinin konusu dinamik boyutlu ve şifreli bir dosya sisteminin kullanıcı uzayında gerçeklenmesidir. Veriler diske şifreli yazılacak ve bu dosya sistemi disk üzerinde sadece mevcut dosyaların/dizinlerin boyutu kadar yer kaplayacaktır. Dosya sisteminin bağlı olmadığı durumda ise tek dosya gibi görünmesi sağlanacaktır.

Öncelikle şifrelemeden genel olarak bahsedilebilir. Şifreleme Romalılardan beri var olup 2. Dünya Savaşı 'nda önemli rol oynamıştır. Son zamanlarda elektronik ticaretin yaygınlaşmasıyla şifrelemeye büyük çapta ticari ilgi oluştu.

Şifreleme teknikleri sabit diskler, disketler ve manyetik teyp gibi ortamlarda depolanan veriler ile internet üzerinden transfer edilen bilginin güvenliğini sağlamak için kullanılır.

Şifreleme işlemlerinde verinin insanlar tarafından okunamaz hale getirilmesi ve bu şifrelenmiş verinin deşifre edilip gözle okunabilir hale getirilmesi 2 önemli esastır. En basit şifreleme biçimi a=1, b=2, c=3, … olan yer değiştirme tekniğidir. Buna göre, şifreleme esnasında "abc" yerine "123" yazılır. Deşifre ederken de '1' yerine 'a', '2' yerine 'b', '3' yerine 'c', … yazılır.

Şifrelemenin temel olarak 3 işlevi vardır:

- Gizlilik: Şifrelenmiş veri taşınırken veya saklanırken şifreleme anahtarını bilmeyenlerin verileri okuyamamasıyla sağlanır.
- Kimlik doğrulama: Bilginin yetkili ve doğru kaynaktan alındığını sağlar.
- Bütünlük: Şifreleme, verilerin yetkilendirilmemiş kullanıcılar tarafından değiştirilmesini önlediğinden bütünlük sağlanır.

Donanımla birlikte kullanılan şifreleme teknikleri vardır. Savunma güçleri ve önemli ticari uygulamalar hız, güvenlik ve kurulum kolaylığı gibi nedenlerden ötürü donanımsal şifrelemeyi tercih edebilmektedir. Ayrıca; özel yapılandırılmış donanımsal şifreleme teknikleri tüm yazılımsal şifrelemelerden daha hızlıdır. Bilgisayardaki yazılımsal şifreleme tekniğinin fiziksel güvenliği yoktur. Bu yüzden fark edilmeden algoritma ile oynanması mümkündür. Fakat donanım teknikleri bu probleme karşı daha iyi koruma sunarlar. Açılması zor kutular, saldırganları uzak tutmada donanım kullanmanın sağladığı diğer bir özelliktir.

Temel olarak 3 tip şifreleme donanımı mevcuttur:

- Kendi başına çalışan şifreleme modülleri: Banka ve benzeri kuruluşlarda kullanılır.
- Haberleşme bağlantıları için adanmış şifreleme kutuları: İki site arasındaki bilgi transferini güvenli kılmak için kullanılır.
- Kişisel bilgisayarlara takılan kartlar: Bu kartlar normalde sabit diske gönderilen tüm verileri şifreler; ancak disketlere gönderilen verileri şifrelemek için de kullanılabilirler.

Yazılım teknikleri genellikle dosyaların şifrelenmesinde kullanılır. Bu tekniklerde şifreleme anahtarları kullanılır. Bu yüzden güvenli yerlerde saklanmalıdırlar. Ayrıca şifreleme işlemi sonrası şifrelenmemiş veriler ve şifreleme anahtarları başkalarının erişimine kapatılmalıdır.

Şifreleme teknikleri simetrik ve asimetrik olarak ikiye ayrılabilir.

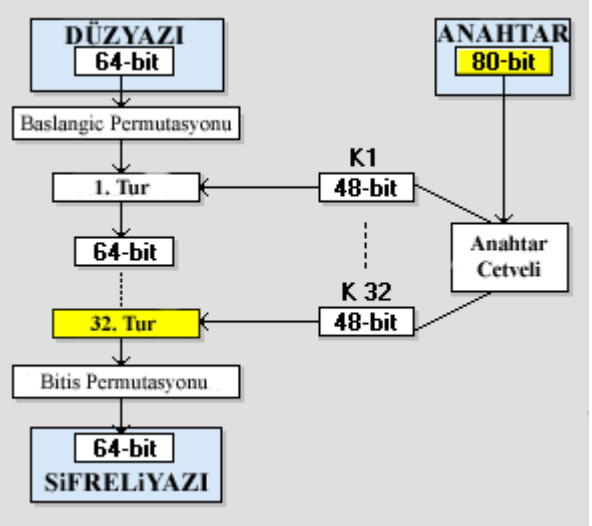
"Özel Anahtar Şifrelemesi" olarak da bilinen simetrik şifrelemede, şifreleme ve deşifre etme işlemlerinde aynı anahtar kullanılır.

Asimetrik şifreleme –açık anahtar şifreleme sistemi– ise şifrelemede ve şifreyi çözmede bir çift anahtar kullanır. Bir anahtar (herkesin görebildiği açık anahtar) veriyi şifrelemek, diğer anahtar (sadece anahtar sahibinin görebildiği özel anahtar) da şifrelenmiş veriyi deşifre etmek için kullanılır. Bir başka deyişle; herhangi bir kişinin bir açık anahtarla şifrelediği veriyi yalnızca bu açık anahtar sahibi açık anahtarıyla ilgilendirilmiş özel anahtarıyla deşifre edebilir. İlk açık anahtar şifreleme algoritması Knapsack olup en yaygın kullanılanı RSA 'dır. RSA; 1977 'de Ron Rivest, Adi Shamir ve Leonard Adleman tarafından MIT 'de geliştirilmiş olup açılımı, tasarımcılarının soyadlarının baş harfleridir. Diffie-Hellman, DSS, ElGamal, Paillier, Cramer-Shoup, Rabin ve McEliece ise diğer açık anahtar şifreleme algoritmalarındandır. [1] [43]

En çok kullanılan simetrik blok şifreleme Bilgi Şifreleme Standardı 'dır (DES - Data Encryption Standard). DES 'de blok boyu 64 bittir ve bu 64 bitlik düzyazılar blokları 56 bitlik anahtarlar ile 64 bitlik şifrelenmiş veri bloklarına çevrilir.

RC2, DES 'e alternatif bir metot olarak geliştirildi. Giriş ve çıkış blokları 8 sekizli uzunluğundadır. RC2 'nin düzgün çalışması için, giriş düzyazısı 8 sekizlinin bir kaç katı olmalıdır. RC2 için giriş anahtarı 1 ile 1024 bit arası bir uzunlukta olabilir. Giriş anahtarı algoritma tarafından şifrelemede kullanılacak bir anahtar yaratmak için kullanılır.

Skipjack, 64 bitlik blok, 80-bitlik anahtar ve 32 dâhili tur kullanan diğer bir blok şifreleme metodudur. DES 'in aksine Skipjack gizlidir ve herkesin kullanımına açık değildir. Skipjack sadece "Clipper" yongası ya da "Fortezza" gibi onaylanmış donanımlarla kullanılabilir.



Şekil 1: Skipjack Blok Şifreleme Metodu

AES, Belçikalı Vincent Rijmen ve Joan Daemen tarafından geliştirilmiş olup DES ve diğer zayıf algoritmaların yerini almayı hedefleyen paranoyak bir şifreleme algoritmasıdır. 1997 ‘de en iyi şifreleme algoritması yarışmasında 1. olup uluslar arası yeni şifreleme standardı olmuştur. 128, 256 ve 512 bitlik anahtarları destekleyen AES ‘in akademik ve pratik tüm saldırılara karşı dayanıklı olduğu düşünülmektedir. [36]

Bir diğer şifreleme algoritması olan Serpent ise AES ‘e alternatif olarak geliştirilmiştir. Ross Anderson, Eli Biham ve Lars Knudsen tarafından tasarlanan Serpent, 128 bitlik blok şifreleme tekniği olup son AES konferansında 59 oy alarak en başarılı 2. şifreleme algoritması seçilmiştir. Intel® Pentium 200 MHz bir bilgisayarda DES (15 Mbit/sn) ‘den 3 kat daha hızlı olan Serpent (45 Mbit/sn), kullanıcılara pratikte yüksek güvenlik standardı sunmak için geliştirilmiştir. [33]

Blowfish, 1993 ‘te Bruce Schneier tarafından DES ‘e alternatif olarak geliştirilmiş simetrik bir blok şifreleyicidir. AES ve Twofish algoritmaları karşısında pek popülerliliği bulunmamaktadır. Ancak ortaya atıldığı zamanlarda pek çok şifreleme algoritmasının lisanslı, patentli olması veya devlet sırrı olarak saklanması nedeniyle büyük yankı uyandırmıştır. Tasarımın belirgin özellikleri anahtar bağımlı S kutuları ve oldukça karmaşık anahtar çizelgesini içerir. [1]

S kutusu, doğrusal olmayan, tablosal yer değiştirme aracıdır. S kutusu giriş ve çıkış verilerinin boyutlarına göre değiştiği gibi rastgele veya algoritmik olarak tasarlanır. İlk olarak Lucifer ‘de kullanılmıştı; ancak şu an tüm şifreleme algoritmalarında kullanılmaktadır. [42]

Twofish, 128 bit blok boyutlu olup 256 bitlik anahtarları destekleyen blok şifreleyicidir. 1998 ‘de Blowfish ‘in tasarımcısı Bruce Schneier ve John Kelsey tarafından geliştirilen Twofish, AES konferansındaki şifreleme yarışmasında finale kalan 5 şifreleme tekniğinden biri olmasına rağmen bir standart olamamıştır. Twofish herhangi bir şekilde patentlenmemiş olup herkesin kullanımına açıktır.

Yukarıda sözü edilen AES konferansındaki şifreleme yarışmasının 5 finalistinin karşılaştırıldığı dokümana *http://www.schneier.com/paper-aes-comparison.pdf* adresinden erişilebilir.

Akan şifreleme farklı uzunluklardaki girişlerle çalışabilir. Yani algoritma, veriyi işlemeden önce belirli boyutta bir verinin girilmesini beklemez.

RC4, 1987 ‘de RSA Laboratories tarafından geliştirilen bir simetrik akış şifreleme metodudur. RC4, 1 ile 2048-bit arası uzunluklarda anahtarları destekler. DES ile karşılaştırıldığında RC4 beş kat daha hızlıdır. Fakat RC4 kırma girişimleri başarı ile sonuçlanmıştır. RC4 ‘ü çalıştırmada kullanılan kod DES ‘de kullanılanın onda biri kadardır. RC4 algoritması Çıkış Geribesleme Modunda (OFB) ve CFB ‘ye benzer bir şekilde çalışır. Fakat CFB ‘den farklı olarak önceki çıkış bloğunun bitlerini giriş bloğunun sağına gönderir. OFB işlerin düzyazı mesajı almamışken bile yapılabilmesini sağlar. Düzyazıyı gerçekte aldığında algoritma çıktısıyla dar veyalanır (XOR işlemi). Bu işlem şifreli yazı bloğunu yaratır. OFB hem blok hem de akış şifrelemede kullanılabilir. Bir akış şifrelemede anahtar sonraki durum fonksiyonunu etkiler. Çıktıyı yaratan fonksiyon anahtara dayalı değildir. [32]

Şifrelemenin ardından özüt (hash) konusuna değinilebilir. Bir özüt fonksiyonu kısaca giriş bilgisi olarak aldığı veriden “özüt değeri” denilen sabit uzunlukta bir katar üretir. Özüt değeri, hesaplandığı mesajın veya dokümanın kısa bir özeti niteliğindedir ve aynı zamanda dokümanın dijital parmak izidir. En yaygın kullanılan özüt fonksiyonları MD5 ve SHA-1 'dir.

MD5 (Message Digest Algorithm 5); veri bütünlüğünü test etmek için kullanılan, Ron Rivest tarafından 1991 yılında geliştirilmiş bir kriptografik özet (tek yönlü şifreleme) algoritmasıdır. Girdi verinin boyutundan bağımsız olarak 128 bitlik özetler üretir.

MD5 ‘teki her girdinin benzersiz olması mümkün değildir, çünkü üretilen "özet" sonuç olarak 128 bittir, ancak MD5 ile şifrelenebilinecek bilgiler sonsuza gider. Bu durum MD5 ‘in kırıldığı anlamına gelmez; çünkü her aynı "özetin" ispatı için deneme-yanılmadan (bruteforce) başka bir yöntem uygulanamamaktadır.

SHA (Secure Hash Algorithm) özüt fonksiyonları 5 adet olup NSA tarafından tasarlanmıştır. SHA-1, SHA-224, SHA-256, SHA-384 ve SHA-512 ‘den en yaygın kullanılanı SHA-1 ‘dir. MD5 ‘in varisi olması düşünülen SHA-1; SSL, TLS, IPsec, PGP, SSH ve S/MIME ‘de kullanılmaktadır.

AES ‘in de tasarımcısı olan Vincent Rijmen ve Paulo S. L. M. Barreto tarafından 2001 yılında gerçeklenmiş kriptografik özüt fonksiyonu olan Whirlpool, ISO (International Standardization Organization) ve IEC (International Electrotechical Commission) tarafından uluslararası standart olarak kabul edilmiştir. Whirlpool herkesin kullanımına açık olup hiçbir zaman patentlenmeyecektir.

Bitirme ödevinde yazılan dosya sistemine benzer yazılımlar hali hazırda mevcuttur: TrueCrypt [30] ve EncFS [2]. TrueCrypt; Windows XP/Vista, Mac OS X ve Linux üzerinde çalışabilen, tek bir dosya şeklinde sanal bir şifreli disk oluşturan, gerçek zamanlı ve saydam şifreleme yapabilen, USB belleklerin ve disk parçalarının tamamını AES–256, Twofish ve Serpent algoritmalarıyla şifreleyebilen, saklı bölüm desteği olan ve grafik arayüzü sunan bir yazılımdır. TrueCrypt 5. sürümünden itibaren Linux sistemlerde FUSE kullanmaktadır. TrueCrypt 'in kullanılmaya başlamadan önce kullanıcıdan kendisine sabit boyutta bir disk alanı ayırmasını istemesi bu yazılımın tek kötü tarafı olarak görülmektedir. EncFS ise Linux ve FreeBSD işletim sistemlerinde çalışabilen, kullanıcı uzayında gerçeklenmiş (FUSE [3] tabanlı) ve dinamik boyutlu bir yazılımdır. Dosya sisteminin bağlı olmadığı durumlarda tek dosya gibi görünmemesi, EncFS yapılandırma dosyasının herhangi bir şekilde zarar görmesi durumunda tüm verilerin kaybedilmesi ve dosya sisteminin taşınabilme zorluğu EncFS 'in göze çarpan eksiklikleridir.

Bitirme ödevinde sonuç olarak; Linux işletim sistemlerinin kullanıcı uzayında çalışabilen, FUSE tabanlı, dinamik boyutlu ve bağlı olmadığı durumlarda tek dosya gibi görünen MetFS [4] isimli bir dosya sistemi yazıldı. MetFS 'te, şifreleme işlemleri için *libgcrypt* [6], dosya sisteminin bağlı olmadığı durumlarda tek dosya gibi görünmesini sağlamak için *libtar* [7] kütüphaneleri kullanıldı. MetFS 'in performansı, üzerinde bulunduğu dosya sisteminin performansına çok yakındır. Bu bitirme ödevi sayesinde dosya sistemi ve kriptoloji hakkında pratik bilgi ve beceri edinildi.

2 KURAMSAL BİLGİLER

2.1 ÖZET ALMA (HASH)

Burada üzerinde konuşulacak olan kriptografik özüt fonksiyonlarıdır. Özüt fonksiyonu, giriş bilgisi olarak aldığı veriden "özüt değeri" denilen sabit uzunlukta bir katar üretir. Özüt değeri, hesaplandığı mesajın veya dokümanın kısa bir özeti niteliğindedir ve aynı zamanda dokümanın dijital parmak izidir. En yaygın kullanılan özüt fonksiyonları MD5 ve SHA-1 'dir.

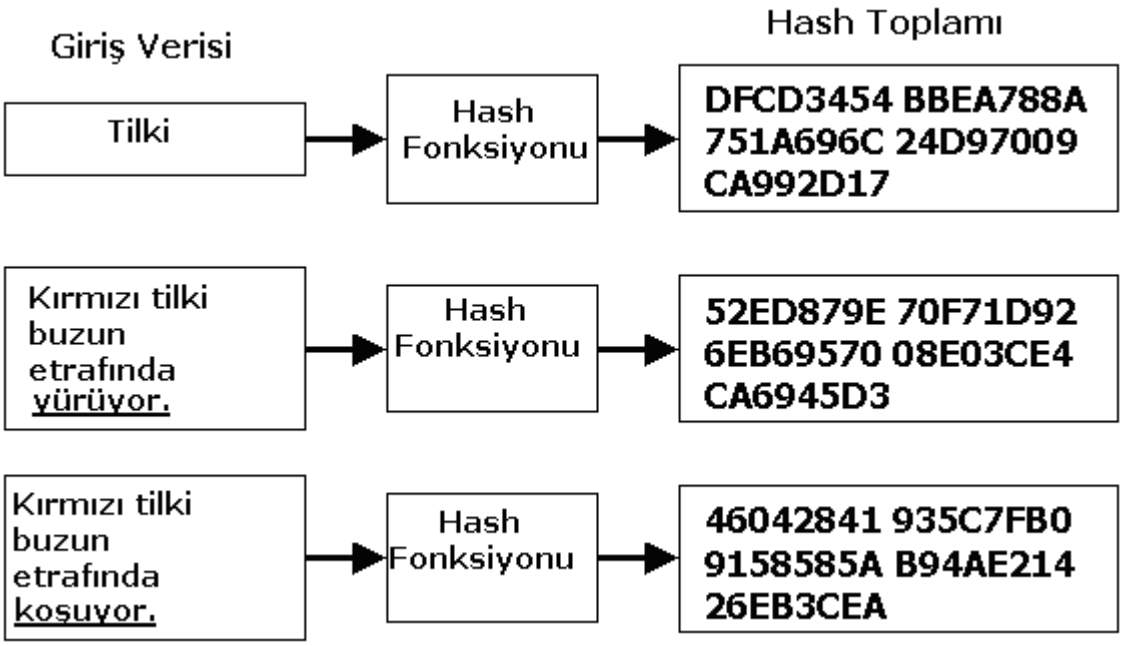
Özüt fonksiyonları etkin hesaplama yeteneği ve belirleyici (farklı giriş verilerine karşılık farklı çıkış verilerinin üretilmesi) olmanın yanında rastgele sayı üreten fonksiyonlara da benzemelidir. Bir özüt fonksiyonu farklı mesajlar için aynı çıktıyı üretiyorsa bu özüt fonksiyonunun çok güvenli olduğu söylenemez. Çünkü bu durum yüzünden yetkilendirilmemiş (unauthorized) bir mesaj, yetkinmiş gibi kullanılabilir.

Algoritma	Çıkış Uzunluğu (bit)	Blok Uzunluğu	Çakışma
HAVAL	256/224/192/160/128	1024	Evet
MD2	128	128	Neredeyse Evet
MD4	128	512	Evet
MD5	128	512	Evet
PANAMA	256	256	Evet
RadioGratun	Değişken	3/192	Hayır
RIPEMD	128	512	Evet
RIPEMD-128/256	128 /256	512	Hayır
RIPEMD-160/320	160/320	512	Hayır
SHA-0	160	512	Evet
SHA-1	160	512	Kusurlu Halinde Evet

SHA-256/224	256/224	512	Hayır
SHA-512/384	512/384	1024	Hayır
Tiger(2)-192/160/128	192/160/128	512	Hayır
WHIRLPOOL	512	512	Hayır

Tablo 1: Özüt Fonksiyonlarının Karşılaştırılması

Yukarıdaki tabloda görülen SHA fonksiyonları NSA [15] tarafından geliştirilmiştir.



Şekil 3: Örnek Bir Özüt Fonksiyonu

Bir özüt fonksiyonunun ürettiği özet (çıkış bilgisi) 2^n deneme-yanılma sonucunda bulunabilir. Buradaki n sayısı girdi olarak verilen mesajın bit olarak uzunluğudur. Eğer özüt algoritmasında çakışma (collision) varsa deneme sayısı $2^{n/2}$ 'ye iner ki bu durum doğum günü saldırısı (birthday attack [18]) olarak adlandırılır.

2.1.1 MD5

Açılımı "Message Digest Algorithm 5" olan MD5, 128 bitlik özüt değeri üreten, kısmen güvensiz, pek çok güvenlik uygulamasında kullanılan, Ron Rivest [16] tarafından 1991 'de tasarlanan, genelde on altılık sayıların sıkıştırılmasıyla oluşturulmuş 32 karakterlik katarlar üreten ve dosya bütünlüğünü test etmede de kullanılan özüt algoritmasıdır. MD5, MD4 'ün geliştirilmiş halidir ve her ne kadar MD4 'ten çok karmaşık da olsa sonuç itibariyle 128 bitlik özet üretir.

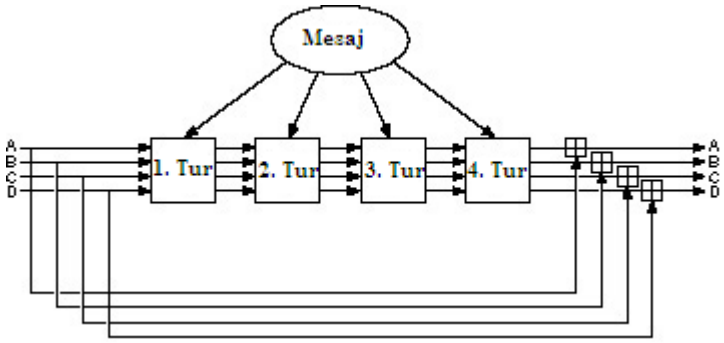
İlk değer atamalarından sonra 512 bitlik giriş metni MD5 tarafından 16 adet 32 bitlik alt bloklara ayrılır. Algoritma, çıkış olarak 4 tane 32 bitlik blok üretir.

Giriş metninin sonuna 1 bitlik “1” ve gerektiği kadar sıfır eklenerek metnin uzunluğunun 512 ‘nin katı olması sağlanır. 32 bitlik 4 değişkenin ilk değerleri şöyledir:

A = 0x01234567
B = 0x89abcdef
C = 0xfedcba98
D = 0x76543210

A, B, C ve D ‘ye *zincir değişkenleri* denir.

Algoritmadaki ana döngü bu noktadan itibaren çalışmaya başlar ve mesajı 512 bitlik bloklar halinde işler. Yukarıdaki 4 değişken farklı 4 değişkene (a = A, b = B, c = C ve d = D olsun) kopyalanır. Ana döngüde 4 tur (MD4 ‘te sadece 3 tur vardı) vardır. Daha sonra 4 turluk 16 işlemde a, b, c ve d değişkenlerinden üçüne doğrusal olmayan bir fonksiyon uygulanır. Bu sonuç 4. değişkene eklenip oluşan yeni sonuç bit sayısı kadar sağa döndürülür. Sonucun bu hali a, b, c ve d ‘den birine eklenir. Son olarak da sonuç a, b, c ve d ‘den birine yazılır. [1]



Şekil 4: MD5 Ana Döngüsü [1]

MD5 ‘teki doğrusal olmayan fonksiyonlar şunlardır (⊕ sembolü “dar veya”, ⊥ sembolü “ve”, ⊦ sembolü “veya” ve ¬ sembolü de “değil” için kullanılmıştır):

$F(X,Y,Z) = (X \perp Y) \vdash ((\neg X) \perp Z)$
$G(X,Y,Z) = (X \perp Z) \neg (Y (\neg Z))$
$H(X,Y,Z) = X \oplus Y \oplus Z$
$I(X,Y,Z) = Y \oplus (X \vdash (\neg Z))$

Çok açık kusurları olmasa da kriptoloji uzmanları tarafından MD5 yerine SHA* algoritmalarından birinin kullanılması tavsiye ediliyor. Çünkü 2007 yılında aralarında Arjen Lastra 'nın da bulunduğu bir grup kriptolog, MD5 özeti aynı olan dosyaların nasıl oluşturulacağına dair bir bildiri [17] yayınladı. Öte yandan 128 bitlik bir özüt değeri bir doğum günü saldırısı düzenleyebilmek için yeterince kısadır. Bunlara ek olarak 2004 Mart 'ında başlatılan MD5CRK [19] projesi de MD5 'in güvenli olmadığını ispatlamayı hedeflemişti ve 17 Ağustos 2004 'te bir IBM p690 bilgisayarda MD5 'e düzenledikleri analitik saldırının sadece 1 saatlik bir zaman diliminde başarıya ulaştığını duyurdu. 10 Mart 2006 'da ise Vlastimil Klima, bir dizüstü bilgisayarda 1 dakika içinde MD5 çakışmaları üretebilen tünel isimli algoritmasını yayınladı [20]. MD5 'in bu kadar savunmasız olmasının temelinde yatan neden MD5 algoritmasının özüt için aldığı giriş

verisinin üzerinden yalnızca 1 kez geçmesidir. Tüm bunlara karşın MD5 algoritması salt bir değerle kullanılırsa bütün saldırı algoritmaları son derece işe yaramaz hale geliyor.

MetFS projesinde MD5 'in tercih edilmesinin nedeni bir sonraki bölümde anlatılacak olan RC4 akan veri şifreleme algoritmasının 128 bit uzunluğunda şifre istemesi ve *libgcrypt* 'te bu koşulu sağlayan tek özüt fonksiyonunun MD5 olmasıdır.

2.1.2 SHA

SHA-0 ilk olarak 1993 ‘te NIST [37] tarafından Amerika Birleşik Devletleri ‘ndeki hükümet işlerinde kullanmak üzere sayısal imza standartlarına uygun olacak biçimde yayınlanmıştır.

SHA ‘da öncelikle giriş mesajı, 512 bitin katı olacak şekilde genişletilir. Bu genişletme işlemi MD5 ‘teki ile tamamen aynıdır. 32 bitlik 5 değişkenin ilk değerleri şöyle atanır (MD5 ‘te 4 değişken vardır; çünkü MD5 128 bit, SHA 160 bit özet oluşturur):

A = 0x67452301
B = 0xefcdab89
C = 0x98badcfe
D = 0x10325476
E = 0xc3d2e1f0

Algoritmadaki ana döngü bu noktadan itibaren çalışmaya başlar ve mesajı 512 bitlik bloklar halinde işler. Yukarıdaki 5 değişken farklı 5 değişkene (a = A, b = B, c = C, d = D, e = E olsun) kopyalanır. Daha sonra 4 turluk 20 işlemde (MD5 ‘te 16 işlem vardır) a, b, c, d ve e değişkenlerinden üçüne doğrusal olmayan bir fonksiyon uygulanır, kaydırmalar ve toplama işlemleri MD5 ‘teki gibi yapılır.

SHA’daki doğrusal olmayan fonksiyonlar şunlardır ($\oplus$ sembolü “dar veya”, $\perp$ sembolü “ve”, $\vdash$ sembolü “veya” ve $\neg$ sembolü de “değil” için kullanılmıştır):

$f_t(X,Y,Z) = (X \perp Y) \vdash ((\neg X) \perp Z)$, t = 0 ‘dan 19 ‘a.
$f_t(X,Y,Z) = X \oplus Y \oplus Z$, t = 20 ‘den 39 ‘a.
$f_t(X,Y,Z) = (X \perp Y) \vdash (X \perp Z) \vdash (Y \perp Z)$, t = 40 ‘tan 59 ‘a.
$ft(X,Y,Z) = X \oplus Y \oplus Z$, t = 60 ‘tan 79 ‘a.

Algoritmada kullanılan 4 sabit değer ise şöyledir:

K_t = 0x5a827999, t = 0 ile 19 arasındaki iken.
K_t = 0x6ed9eba1, t = 20 ile 39 arasındaki iken.
K_t = 0x8f1bbcdc, t = 40 ile 59 arasındaki iken.
K_t = 0xca62c1d6, t = 60 ile 79 arasındaki iken.

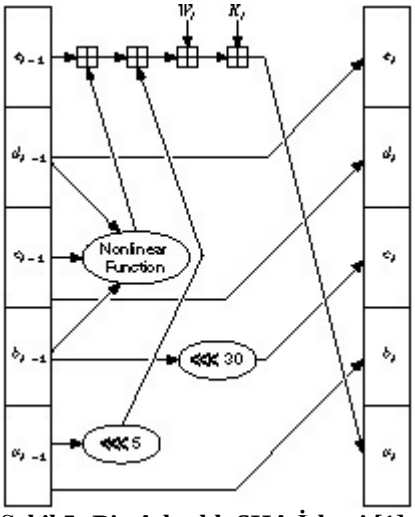
Bu sabit sayılar şöyle bulunmuştur: 0x5a827999 = $2^{1/2}$ / 4, 0x6ed9eba1 = $3^{1/2}$ / 4, 0x8f1bbcdc = $5^{1/2}$ / 4 ve 0xca62c1d6 = $10^{1/2}$ / 4

Mesaj blokları 16 tane 32 bitlik sözcükten (M_0 - M_{15}) 80 adet 32 bitlik sözcüğe (W_0 - W_{79}) aşağıdaki algoritmayla çevrilmiştir:

$W_t = M_t$, t = 0 ‘dan 15 ‘e kadar
$W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) <<< 1$, t = 16 ‘dan 79 ‘a kadar.

t, 0 ‘dan 79 ‘a kadar işlem numarasını göstermek üzere W_t, genişletilmiş mesajın t ‘ninci alt bloğunu gösterir. Ayrıca; <<< s ifadesi s bitin sola çevresel kaydırıldığını belirtir. Bu aşamada an döngü şu şekildedir: [1]

FOR t = 0 to 79
$TEMP = (a <<< 5) + f_t(b,c,d) + e + W_t + K_t$
$e = d$
$d = c$
$c = b <<< 30$
$b = a$
$a = TEMP$



Şekil 5: Bir Adımlık SHA İşlemi [1]

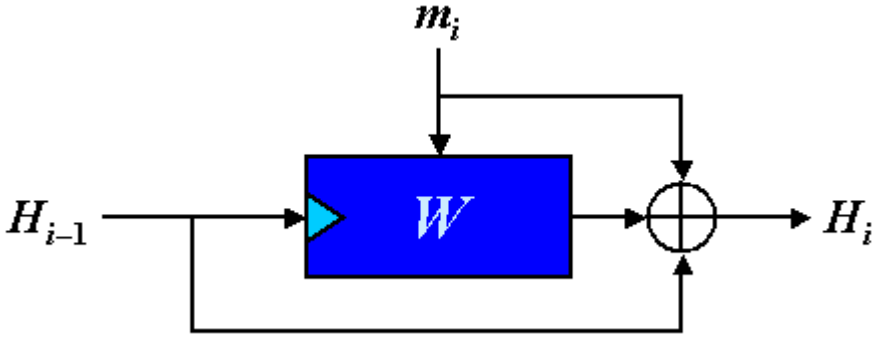
SHA-0 ‘ın çıkış uzunluğunun 160 bit ve çakışmaların olası olması nedeniyle SHA-0 algoritmasının doğum günü saldırılarına karşı 2^{80} kadar dayanıklı olduğu düşünülürdü; ancak 1998 ‘de 2 Fransız araştırmacı Florent Chabaud ve Antonie Joux SHA-0 ‘daki çakışmaların, bu algoritmanın dayanıklılığının gerçekte 2^{61} olduğunu gösterdi [38]. Daha sonra 12 Ağustos 2004 tarihinde Joux, Carribault, Lemuet ve Jalby tarafından Itanium 2 işlemcili süper bir bilgisayarla 80000 işlemci saatinde SHA-0 ‘ın dayanıklılığının 2^{51} ‘e düşürüldüğünü belirtti. Bu açıklamanın sadece 5 gün sonrasında Wang, Feng, Lai ve Yu MD5, SHA-0 ve diğer özüt algoritmalarına yaptıkları saldırının sonuçlarını açıkladı. Bu açıklamada SHA-0 ‘ın dayanıklılığının 2^{40} olduğu belirtilmiştir [39].

2.1.3 Whirlpool

Vincent Rijmen ve Paulo S. L. M. Barreto tarafından tasarlanan Whirlpool özüt algoritması 2^{256} bitten daha kısa giriş metinleri üzerinde çalışır ve 512 bitlik özet bilgisi üretir.

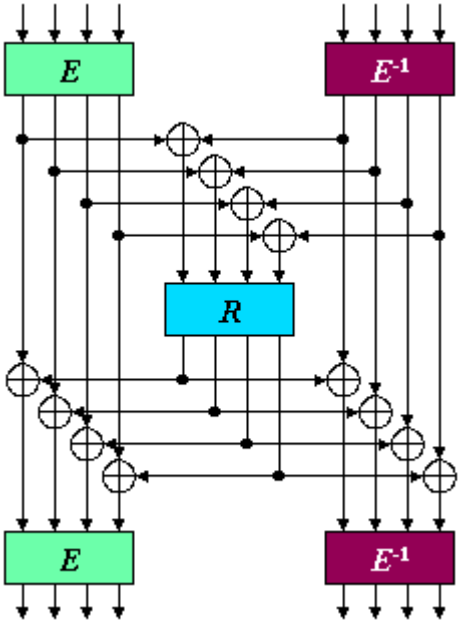
Whirlpool, Merkle-Damgard ‘ın güçlendirme ve Miyaguchi-Preneel ‘in özüt şemasını W isimli 512 bitlik adanmış blok şifreleyicide kullanır. Giriş metninin sonuna 1 bitlik “1” ve gerektiği kadar 0 eklenerek metnin uzunluğunun 512 ‘nin katı olması sağlanır. Elde edilen yeni metin, 512 bitlik bloklara bölünür: H_0, H_1, H_2, ... H_t. Whirlpool ‘un tanımı gereği H_0,

512 adet sıfırdan oluşur. H_i 'yi hesaplamak için W, m_i 'yi H_{i-1} 'i anahtar olarak kullanarak şifreler ve oluşan şifrelenmiş veriyi hem H_{i-1} hem de m_i ile dar veya işlemine sokar. Sonuç olarak üretilecek olan özet H_t 'dir. [40]



Şekil 6: Miyaguchi-Preneel Sıkıştırma Fonksiyonu

W blok şifreleyicinin iç yapısı AES algoritmasına çok benzer:



Şekil 7: W 'nin İç Yapısı

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E(u)	1	B	9	C	D	6	F	3	E	8	7	4	A	2	5	0

Tablo 2: E Mini Kutusu

u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E^{-1} (u)	F	0	D	7	B	E	5	A	9	2	C	1	3	4	8	6

Tablo 3: E^{-1} Mini Kutusu

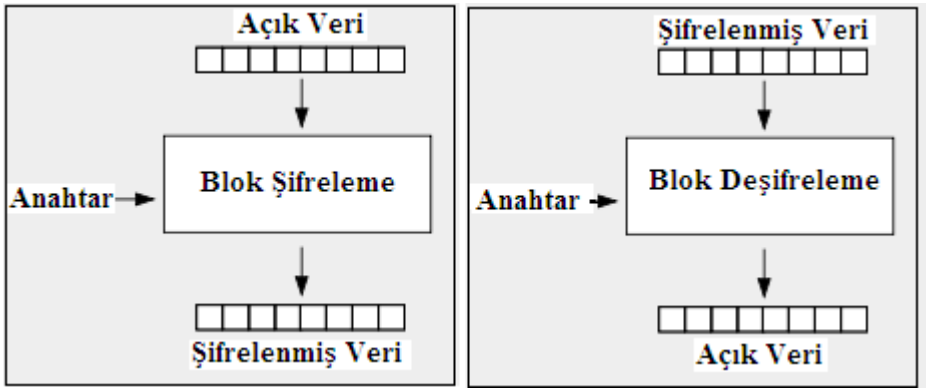
u	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
R(u)	7	C	B	D	E	4	9	F	6	3	8	A	2	5	1	0

Tablo 4: R Mini Kutusu [40]

2.2 Şifreleme

Şifreleme genel olarak simetrik ve asimetrik olmak üzere ikiye ayrılır. Simetrik şifrelemede ise blok ve akan (stream) şifreleme olarak bilinen iki tip şifreleme algoritması vardır. Blok şifrelemede veri bloklar halinde işlenir. Akan şifrelemede ise veri kullanılan tekniğe göre değişken uzunluklarda işlenir.

Blok şifreleme, şifrelenecek bir blok veriyi alır (verinin boyutu sabit ve genelde 64 bittir) ve şifreleme anahtarı ile veriyi aynı boyuttaki başka bir bloğa dönüştürür.



Şekil 8: Blok Şifreleme [31]

Blok şifrelemede 3 işlem modu yaygın olarak kullanılır:

- Elektronik Kod Kitabı (ECB - Electronic Code Book)
- Şifre Bloğu Zincirleme (CBC - Cipher Block Chaining)
- Şifre Geribesleme Modu (CFB - Cipher Feedback Mode)

ECB kullanımında düzyazı kalıpları gizli değildir. Her benzer düzyazı bloğunun benzer şifrelenmiş bloğu vardır. Düzyazı blokları kolaylıkla silinebilir, tekrar edilebilir veya yer değiştirilebilir.

Düz Yazı Blok No ile Dar Veya (XOR) İşlemi Uygula			Blok1	Blok2	Blok3 …
Şifrelenmiş Blok No:	Blok1	Blok2	Blok3	Blok4 …	

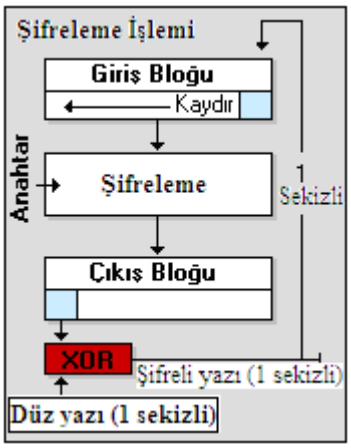
Şekil 9: ECB Blok Düzeni

CBC ‘de şifrelenmiş blok, bir düzyazı bloğu ile şifrelenmemiş düzyazı bloğunun XOR işlemi uygulanmasıyla elde edilir. Bu şekilde bütün düzyazı kalıpları bir başlangıç vektörü (Initialization Vector - IV) ile gizlenir.

CFB kullanımında şifre, sekizli üreteci olarak çalışır. Bir sekizli şifrelenmiş veri yaratmak için çıktının her bir sekizlisi düzyazının bir sekizlisi ile XOR işlemine tabi tutulur. Bu işlem sırasında IV giriş bloğuna kopyalanır. Sonra bu giriş bloğu üzerinde şifreleme yapılır ve elde edilen sonuç, çıktı için ayrılan hafızaya aktarılır. Daha sonra düzyazının ilk sekizlisi en soldaki sekizli ile dar veyalanır. Giriş bloğundaki bütün sekizliler sola kaydırılıp bir tanesi atılarak, elde edilen sekizli, giriş bloğunun en sağındaki sekizliye şifrelenmiş veri olarak gönderilir. Çıktının en soldaki sekizlisi bir sonraki düzyazı sekizlisi ile dar veyalanır.

CFB ile akan veri şifrelemesi de yapılabilir.

Akan veri şifrelemesinde giriş verisinin sabit bir uzunluğu yoktur.

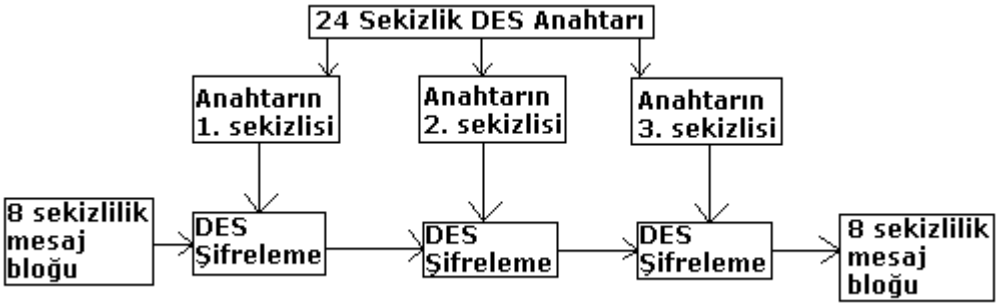


Şekil 10: Akan Şifreleme

2.2.1 DES Blok Şifreleme

DES 'te düzyazı bazı permutasyon ve yedekleme işlemlerine tabi tutulur. Sonra, güvenli bir şifreli yazı bloğu yaratmak için çıktılar XOR işlemiyle orijinal düzyazı ile birleştirilir. Bu şifreleme serisi 16 kere tekrarlanır. Her seferinde farklı anahtar grupları kullanılır.

DES orta seviye kaynakları olan saldırganlara karşı sınırlı bir koruma sağlar. DES 'in 56 bitlik kısa anahtar boyutu yeni işlemcilerin gücü karşısında artık yetersiz kalmaktadır ve 3-7 saatte kırılabilir. DES 'in zayıflıkları yüzünden daha güçlü ve etkili şifreleme metotları geliştiriliyor.



Şekil 11: DES Şifreleme Algoritması

Triple-DES (Üçlü DES), DES 'in daha çok güvenlik sağlayan bir türüdür. Bu metotta şifreleme anahtarının uzunluğu DES 'in 3 kez kullanılmasıyla sağlanmıştır. Triple-DES 'in DES 'e oranla 2 kat daha fazla güvenlik sağladığı düşünülür. Bu düşüncenin kaynağında 112 bitlik kod vardır. Kodlamanın 2 katına çıkması doğal olarak şifreleme süresini artırmıştır.

Triple-DES tarafından kullanılan teknik EDE (encrypt – decrypt – encrypt) olarak bilinmektedir. Düzyazı, Triple-DES anahtarının ilk 8 sekizlisi ile şifrelenir. Sonra şifreleme anahtarının ortadaki 8 sekizlisi ile deşifre edilir. Son olarak anahtarın son 8 sekizlisi ile şifrelenerek 8 sekizlilik bir blok elde edilir. Eğer tüm anahtarlar aynı ise Triple-DES ile DES aynıdır. Eğer anahtarlar birbirinden farklı ise şifrelemenin çok sağlam olduğunu söylemek mümkündür.

DESX yine DES ‘e dayalı ve daha fazla özelliğe sahip diğer bir algoritmadır. Düzyazı girişini gizlemek için ekstra bir anahtar kullanır.
Uluslararası Veri Şifreleme Algoritması (International Data Encryption Algorithm - IDEA) 64-bit bloklar üzerinde 128-bit anahtar kullanan bir simetrik şifreleme metodudur. IDEA patentli bir algoritmadır. [32]

2.2.2 AES Blok Şifreleme

Rijndael olarak da bilinen AES, 2002 yılında standart haline getirildi. AES, uzunluğu 128 bit olan giriş mesajını uzunluğu 128, 192 veya 256 bit olan anahtarla şifreler. Algoritma esas olarak şifrelenecek açık metin ile sonuçta oluşturulacak şifreli yazı arasında anahtar aracılığıyla yapılan çeşitli dönüşümlerden oluşan turlardan ibarettir.

Öncelikle anahtar, Rijndael Anahtar Tablosu (Rijndael ‘s Key Schedule [41]) ile genişletilip *AddRoundKey* turu gerçeklenir. Bundan sonra 4 tur atılır:

- *SubBytes*: Her sekizlinin tablodaki başka bir sekizli ile yer değiştirdiği doğrusal olmayan adımdır.
- *ShiftRows*: Mevcut durumu yansıtan tablodaki her satır belli sayıda çevrel olarak kaydırılır.
- *MixColumns*: Her kolondaki 4 sekizlinin karıştırıldığı adımdır.
- *AddRoundKey*: Durum tablosunun her sekizlisi tur şifreleme anahtarının, anahtar tablosu ile türetilmiş haliyle birleştirilir.

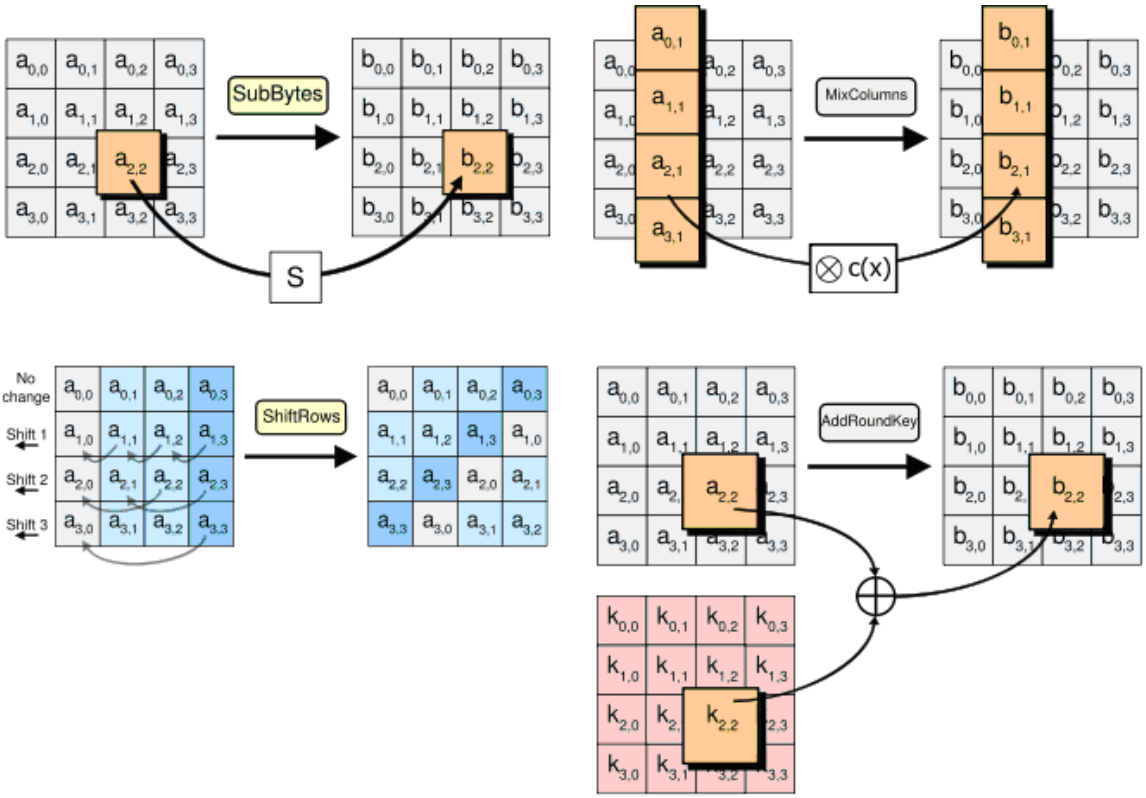
Son tur ise şu adımlardan oluşur (*MixColumn* adımı yok):
- *SubBytes*
- *ShiftRows*
- *AddRoundKey*

SubBytes Adımı: Bu adımda dizideki her sekizli Rijndael S kutusu olarak bilinen 8 bitlik yer değiştirme kutusuyla güncellenir. Bu işlem sayesinde algoritma doğrusal olmama özelliğini kazanır. Buradaki S kutusu doğrusallık özelliğini iyi giderdiği bilinen 2^8 (soyut cebirdeki sonlu alan) ‘in ters çarpanlarından (1/x) türetilmiştir. Algoritmayı cebirsel saldırılardan korumak amacıyla birinci dereceden dönüşümler ($x \rightarrow A.x + B$) kullanılmıştır.

ShiftRows Adımı: Bu adımda durum tablosunun her satırındaki bitler dairesel olarak belli bir sayıda kaydırılır. 128 ve 192 bitlik giriş verisi için kaydırma miktarı aynıdır. 256 bitlik verilerde durum tablosunun ilk satırı kaydırılmaz; 2., 3. ve 4. satırlar ise sırasıyla 1, 2 ve 3 sekizli kadar kaydırılır.
MixColumns Adımı: Bu adımda durum tablosunun her sütunundaki 4 sekizli, tersinir bir doğrusal dönüşümle birleştirilir. *MixColumns* fonksiyonu giriş parametresi olarak 4 sekizli alır ve çıkış olarak 4 sekizli uzunluğunda veri üretir. Giriş parametresindeki her sekizli çıkıştaki verinin tamamını etkiler. *MixColumns*, *ShiftRows* ile birlikte şifrelemenin yayılmasını sağlar. Her sütun 2^8 ‘in bir polinomu gibi işlem görür. Her sütun daha sonra $c(x) = 3x^3 + x^2 + x + 2$ sabit polinomuyla çarpılıp $x^4 + 1$ ile modu alınır.

AddRoundKey Adımı: Bu adımda alt anahtar ile durum bağlanmıştır. Her turda ana anahtardan Rijndael Anahtar Tablosu yardımıyla bir alt anahtar üretilir. Alt anahtar ile

durumun boyutu aynıdır. Alt anahtar ile durum tablosunun her sekizlisi dar veya işlemine tabi tutulur.



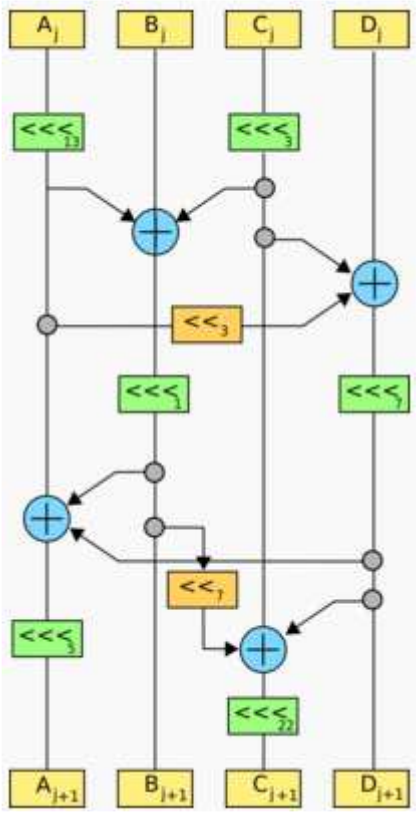
Şekil 12: *SubBytes, ShiftRows, MixColumns* ve *AddRows* Adımları

2006 yılında AES ‘e karşı başarılı olan tek saldırı gerçekleşti: yan kanal saldırısı. Bu saldırı, şifreleme sisteminin algoritmasından çok bağımlı olduğu fiziksel yapıdan kaynaklandığı için NSA, AES algoritmasının ABD verilerini korumada hala yeterince güvenli olduğunu belirtti. Ayrıca bu saldırı 128 bit için 7. tura, 192 bit için 8. tura ve 256 bit için 9. tura kadar gelmiştir. Buna karşın AES 128 bit için 10, 192 bit için 12 ve 256 bit için 14 turluk şifreleme yapmaktadır.

2002 yılında Nicolas Courtois ve Josef Pieprzyk AES ‘e karşı başarılı olabilecek teorik bir saldırı duyurdu: XSL Saldırısı. Kriptoloji uzmanları XSL saldırısını tasarlayanların matematiksel bir hata yaptığını, AES ‘in cebirsel alt yapısının çok sağlam olduğunu ve XSL saldırısı karşısında dayanıklı olduğunu belirtmektedirler. Sonuç olarak XSL saldırıları söylentiden öteye geçemedi ve AES hala en güvenli blok şifreleyici konumundadır. [36]

2.2.3 Serpent Blok Şifreleme

Diğer blok şifreleyicilerde olduğu gibi Serpent de 128 bitlik giriş verisi ile 128, 192 ve 256 bitlik anahtarları desteklemektedir. Serpent, 32 bitlik 4 sözcüğü 32 tur boyunca yer değiştirme işlemine tabi tutar. Serpent ‘in tasarımında 32 biti paralel olarak işlemek vardır. Bu sayede her turda 8 tane 4 bitlik veri ile 4 bitlik S kutusu paralel işlenir.



Şekil 13: Serpent – Doğrusal Karıştırma Adımı

Serpent, güvenliğe AES şifreleme yarışmasının diğer finalistlerinden çok daha fazla dikkat eder. Mevcut saldırılara karşı 16 turluk şifreleme yeterliyken, Serpent ‘te gelecekteki saldırılara önlem olarak 32 turluk şifreleme yapılmaktadır.

Serpent blok şifreleme algoritması patentlenmemiş olup herkes tarafından herhangi bir ücret ödemeden hem yazılımlarda hem de donanımlarda kullanılabilir.

Çok etkili XSL saldırılarının Serpent ‘i mağlup edebileceği düşünülse de XSL saldırılarının deneme-yanılma saldırılarından çok daha maliyetli olması nedeniyle bu yönden saldırılacağı pek düşünülmemektedir. [35]

2.2.4 Blowfish Blok Şifreleme

Büyük ölçekli mikroişlemcilerde kullanmak üzere tasarlanan algoritma:

- Patentlenmemiştir: Herhangi bir ücret ödemeden kullanılabilir.
- Sade tasarlanmıştır: Yalnızca toplama, dar veya ve 32 bitlik işlemlerde tablodan bakma işlemleri yürütülür. Bu nedenle gerçekleme hatalarına karşı dirençlidir.
- Az yer kaplar: 5 KB bellek alanı kullanır.
- Anahtar boyutu değişkendir: 448 bite kadar.

5 KB ‘lık bellek gereksinimi, Blowfish ‘in akıllı kartlarda ve gömülü sistemlerde kullanılmasını engellemiştir. Buna karşın 32 bitlik mimarilerde Blowfish, DES ‘e oranla çok daha hızlıdır.

Blowfish ‘te veriler 64 bitlik bloklar halinde şifrelenir. Algoritma 2 kısımdan oluşur: anahtarın genişletilmesi ve şifreleme. Anahtar genişletmede en fazla 468 bit uzunluğunda olan anahtar çok sayıda alt anahtarlarla 4168 bite genişletilir.

Veriler, basit bir fonksiyonun 16 kez kullanılmasıyla şifrelenir. Her turda anahtara bağımlı permutasyon ve veriye bağımlı yer değiştirme işlemleri yürütülür.

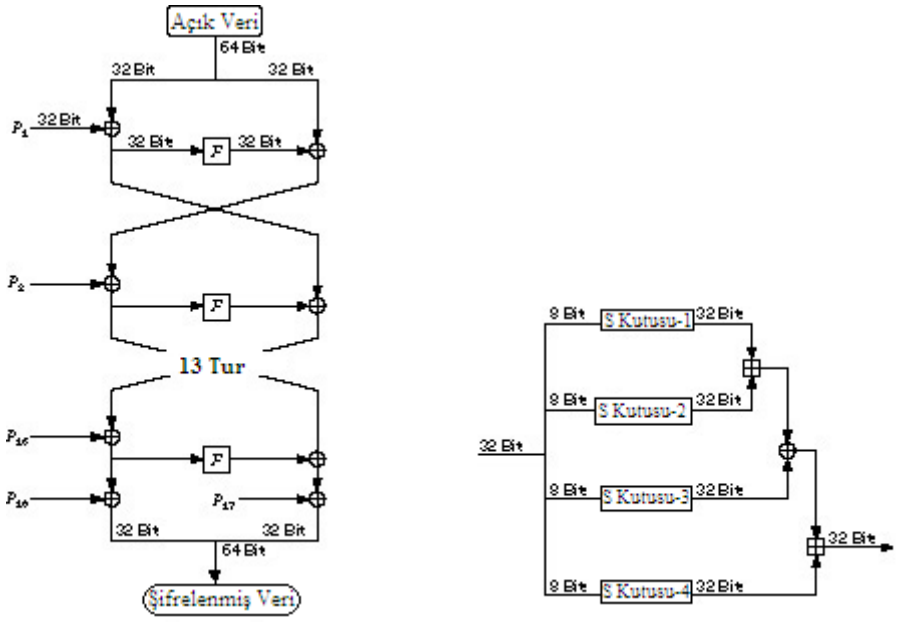
Blowfish ‘te 32 bitlik 18 adet alt anahtar bulunmaktadır: P_1, P_2,..., P_{18}

4 S kutusunun her birinde 256 kayıt vardır:

$S_{1,0}$, $S_{1,1}$,..., $S_{1,255}$
$S_{2,0}$, $S_{2,1}$,..., $S_{2,255}$
$S_{3,0}$, $S_{3,1}$,..., $S_{3,255}$
$S_{4,0}$, $S_{4,1}$,..., $S_{4,255}$

Blowfish 16 turdan oluşur ve giriş verisi 64 bittir (x olsun). Buna göre şifreleme işlemi şöyle yapılır:

x ‘i 32 bitlik 2 parçaya böl: x_L, x_R
For i = 1 to 16:
$x_L = x_L \oplus P_i$
$x_R = F(x_L) \oplus x_R$
x_L ve x_R ‘yi takas et
x_L ve x_R ‘yi takas et (son takası geri al)
$x_R = x_R \oplus P_{17}$
$x_L = x_L \oplus P_{18}$
x_L ve x_R ‘yi geri birleştir



Şekil 14: Serpent Algoritması ve F Fonksiyonu

F fonksiyonu şöyle tanımlanabilir:

x_L ‘yi 8 bitlik 4 parçaya böl: a, b, c ve d
$F(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$

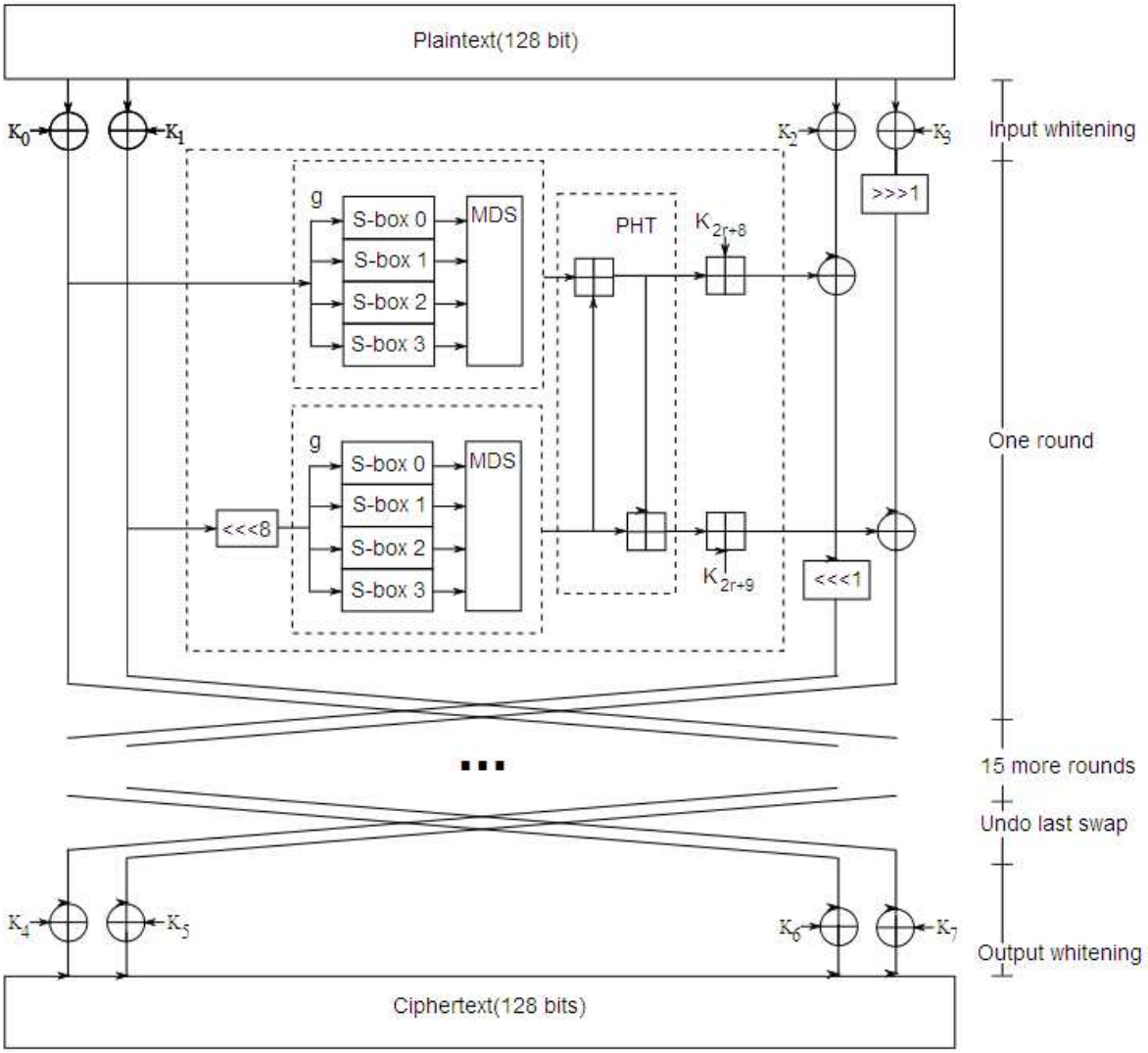
Deşifre işlemi şifreleme işlemiyle tamamen aynıdır; yalnızca P_1, P_2,..., P_{18} ters sırada kullanılır. [1]

En hızlı blok şifreleyicilerden biri olan Blowfish ‘e karşı yapılan saldırılardan hiç biri başarılı olamamıştır. Microsoft® Windows, Macintosh®, Nautilus ve PGPfone ‘da Blowfish kullanılmaktadır. [1]

2.2.5 Twofish Blok Şifreleme

Twofish, kriptolojide 256 bit şifre destekleyen, veriyi 128 bitlik bloklar halinde işleyen simetrik blok şifreleyici demektir.

Twofish algoritmasında girişte alınan veri 32 bitlik 4 sözcüğe bölünür. Asıl şifreleme anahtarı genişletilip 4 alt anahtara bölünür ve her alt anahtarla bir sözcük dar veya işlemine sokulur. Bu işlemin 16 kez yapılmasının ardından gerçekleştirilen sağa/sola döndürme işlemleri ve takasın ardından şifrelenmiş veri oluşturulur. Şekil 15 ‘te Twofish algoritması ayrıntılı olarak görülmektedir.



Şekil 15: Twofish Algoritması [42]

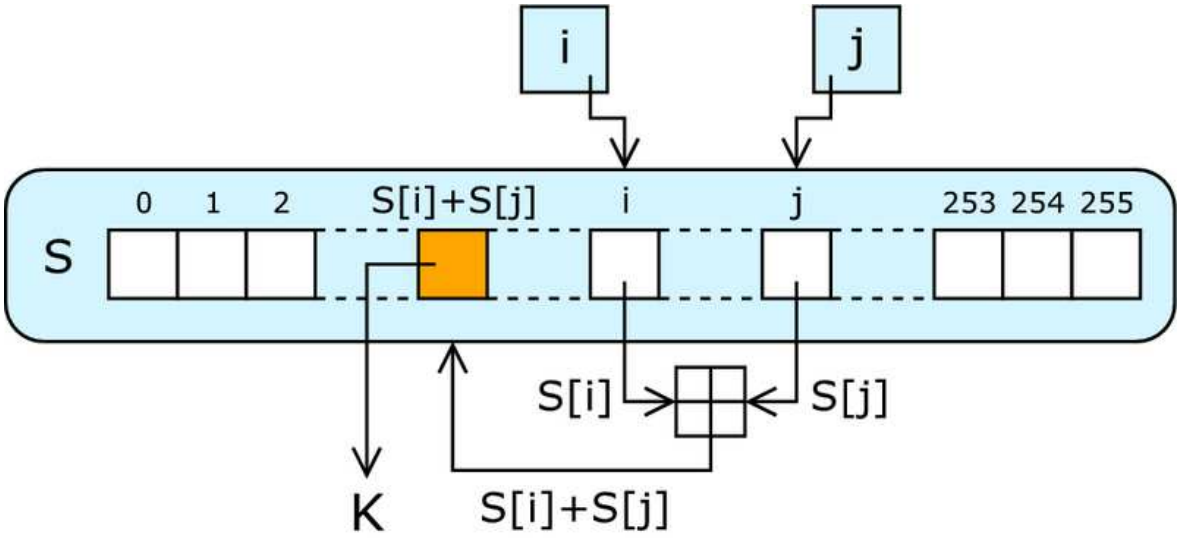
2.2.6 Akan Veri Şifreleme Algoritmaları ve RC4

Kriptolojide açık veriyi rastgele bir şifreleme verisiyle, genelde dar veya işlemiyle, karıştıran simetrik anahtar şifreleyicisine akan veri şifreleyicisi denir. Buna karşın blok şifreleyiciler büyük bloklar üzerinde sabit ve değişmeyen dönüşümler yapar. Akan veri şifreleyicisi, blok şifreleyicisinden daha hızlıdır ve daha düşük donanıma ihtiyaç duyar. Öte yandan yanlış kullanıldığında, özellikle aynı başlangıç durumu 2. kez kullanıldığında, büyük güvenlik açıkları verebilir.

En yaygın kullanılan akan veri şifreleme algoritması RC4 olup A5/1, A5/2, Chameleon, FISH, Helix, ISAAC, MUGI, Panama, Phelix, Pike, SEAL, SOBER, SOBER-128 ve WAKE gibi çeşitli akan veri şifreleme algoritmaları mevcuttur.

Yine Ron Rivest tarafından RSA Security ‘de 1987 yılında geliştirilen RC4 ‘ün birden fazla açılımı mevcuttur: Ron’s Code 4, Rivest Cipher 4. RC4 başlangıçta ticari bir sırdı; ancak 1994 Eylül ‘ünde Cypherpunks e-posta listesine isimsiz bir kişi tarafından detaylı açıklaması ve kaynak kodu yazıldı. Çok kısa bir süre sonra da *sci.crypt* haber grubunda da açıklandı. Buna rağmen RC4 ticari bir marka haline getirildi ve marka sorunlarıyla uğraşmamak için RC4 ‘e ARCFOUR ve ARC4 gibi isimler takıldı. *Alleged RC4*, yani “iddia edilen RC4” tabirinin nedeni ise RSA firmasının şimdiye dek RC4 algoritmasının ne olduğunu resmi olarak açıklamamış olmasıdır.

Şekil 3 ‘te örnek bir S katarı için RC4 algoritmasının nasıl çalıştığı gösterilmiştir. [21]



Şekil 16: RC4 Çalışma Şeması

RC4 algoritması OFB ‘de çalışır ve anahtar, giriş verisinden bağımsızdır. 8*8 ‘lik S kutuları (S_0, S_1,..., S_{255}) bulunan algoritmada ilk değerleri 0 olan *i* ve *j* değişkenleri vardır.

RC4 ‘te rastgele bir sekizli şöyle oluşturulur:

$i = (i + 1) \bmod 256$
$j = (j + S_i) \bmod 256$
S_i ve S_j ‘yi takas et
$t = (S_i + S_j) \bmod 256$
$K = S_t$

Son olarak oluşturulan K sekizlisi ile açık veri dar veyalanırsa şifrelenmiş veri, şifrelenmiş veriyle dar veyalanırsa da açık veri elde edilir. RC4 ‘teki şifreleme işlemi DES ‘ten 10 kat daha hızlıdır.

S kutularının ilk değerlerini atamak oldukça kolaydır: $S_0 = 0$, $S_1 = 1$,..., $S_{255} = 255$. Daha sonra bir 256 bitlik dizi anahtarla doldurulur. Dizinin dolması için anahtar gerektiği kadar tekrar edilir: K_0, K_1,..., K_{255}. j değişkeni sıfırlanır ve şu işlemler yapılır:

for $i = 0$ to 255:
$j = (j + S_i + K_i) \bmod 256$
S_i ile S_j ‘yi takas et

RC4 akan veri şifreleme algoritmasının tamamının bu kadar olması ne kadar sade tasarlandığının göstergesidir.

RC4; BitTorrent, Kerberos, Microsoft® Point-to-Point Encryption, Oracle Secure SQL, Lotus Notes, Apple – AOCE, RDC, RDP, SASL, SSH, SSL, TLS, WPA ve WEP gibi yerlerde kullanılan en popüler akan veri şifreleme algoritmasıdır. Sıra dışı performansı ve hem donanım hem yazılım açısından çok sade gerçeklenebilirliği RC4 ‘ün arkasındaki itici güçtür. Ancak; aynı anahtar veri 2. kez kullanıldığında veya çıkış verisinin başlangıç kısmı atılmazsa, RC4 son derece savunmasız kalır.

2.3 FUSE

Açılımı “Filesystem in Userspace” olan FUSE yazılımı, kullanıcı uzayında tam fonksiyonlu dosya sistemi yazmaya olanak sağlayan bir anaçatı yazılımdır. Sade kütüphanesi, çekirdeği tekrar derlemeye ve yamalamaya gerek duymayan basit kurulumu, güvenli ve kararlı kodu, *root* hakkı olmayan kullanıcılar tarafından da kullanılabilmesi ve iyi dokümantasyonu FUSE 'un çok önemli avantajları arasında sayılabilir. Öte yandan Linux (ilk geliştirildiği sistem), FreeBSD, NetBSD, Mac OS X (Darwin), Windows, OpenSolaris ve GNU/Hurd işletim sistemlerinde çalışabilmesi de diğer bir avantajıdır.

FUSE esasında AVFS 'i (A Virtual Filesystem) [12] desteklemek amacıyla geliştirilmiş bir projedir. Fakat daha sonra 100 satırdan az bir kodla dosya sistemi gerçeklenmesini sağlayan onlarca dosya sistemi yazılımının [13] üzerine bina edildiği ayrı bir proje haline gelmiştir.

FUSE 2 bileşenden oluşur:

- Dosya sistemi yazanlar tarafından kullanılan kullanıcı uzayındaki kütüphane (dinamik bağlanabilir)
- VFS ‘e bağlanan ve çağrıları FUSE kitaplığına yönlendiren çekirdek modülü

Yeni bir dosya sistemi yazılırken bu dosya sisteminin amaçları doğrultusunda gerekli tüm metotlar gerçeklenmelidir. Örneğin; verilerin diske şifrelenerek yazılması isteniyorsa diske veri yazan sistem çağrısı uygun şekilde değiştirilmelidir. FUSE tam bu noktada işin içine girerek programcının çekirdek uzayında büyük miktarda zor kodu yazmaktan kurtulmasını sağlayan arayüz fonksiyonlarını sunar. Dosya sistemi yazarken ihtiyaç duyulabilecek tüm sistem çağrıları FUSE kaynak kodunda yer alan *include/fuse.h* dosyasındaki *struct fuse_operations* veri yapısı tarafından sağlanmıştır: *mknode, mkdir, rename, chmod, chown, open, read, write, create, release, opendir, rmdir, …* Programcı, kendi dosya

sistemine özgü dosya sistemi çağrılarını yazdıktan sonra sadece bir *fuse_operations* veri yapısında kendi sistem çağrılarını belirtmelidir. Şekil 17 ‘te görüldüğü üzere, mevcut dosya sistemindeki *getattr()* dosya sistemi çağrısına karşılık *metfs_getattr()*, *access()* dosya sistemi çağrısına karşılık *metfs_access()*, *readlink()* dosya sistemi çağrısına karşılık *metfs_readlink()* ve *opendir()* dosya sistemi çağrısına karşılık *metfs_opendir()* fonksiyonu yazılmıştır.

```
struct fuse_operations metfs_oper = {
      .getattr   =  metfs_getattr,
      .access    =  metfs_access,
      .readlink  =  metfs_readlink,
      .opendir   =  metfs_opendir,
      …
};
```

Şekil 17: Yeni Yazılan Dosya Sistemi Çağrılarının FUSE Fonksiyonlarıyla Eşleştirilmesi

Eğer bir dosya sistemi çağrısının değiştirilmesi gerekmiyorsa FUSE kaynak kodunun *example/fusexmp_fh.c* dosyasındaki ilgili fonksiyon olduğu gibi kullanılabilir.

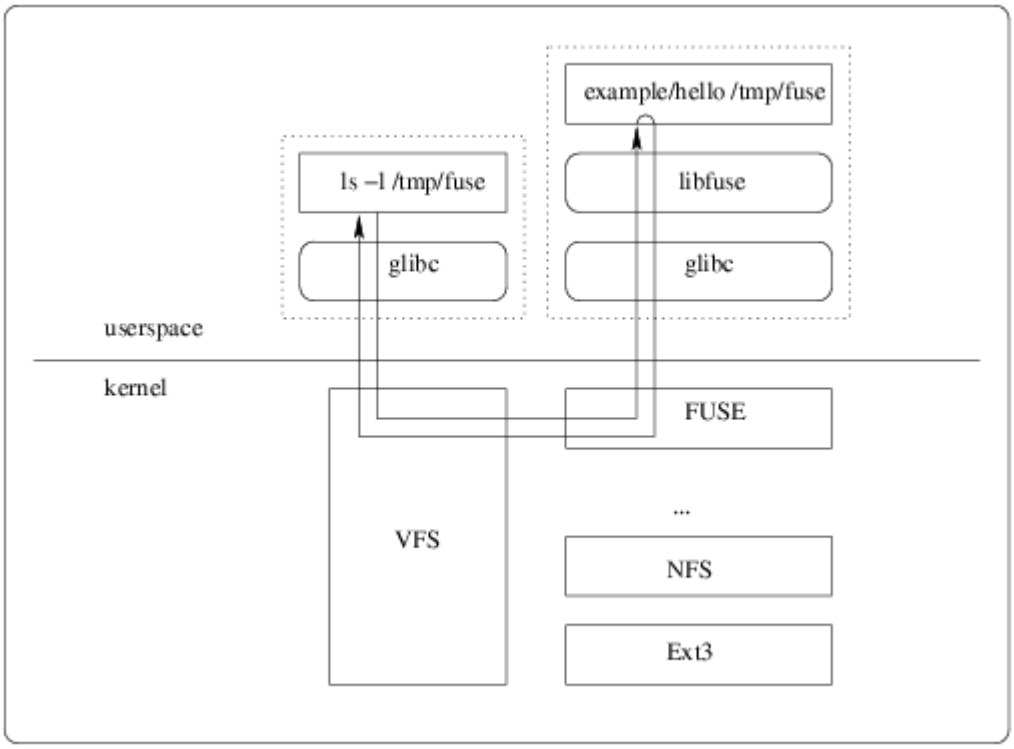
2.3.1 FUSE ‘un Çalışma Biçimi

İlk olarak FUSE çekirdek modülü VFS ‘e kaydedilir. Kullanıcı uzayında gerçeklenen uygulamalar FUSE kullanacaksa aşağıdakileri yapmalıdır:

- Mevcut dosya sisteminde FUSE kullanan dosya sisteminin bağlanacağı bir bağlama noktası belirtilmelidir.
- Dosya ve dizinleri yönetebilmek için gerekli tüm işlemler tanımlanmalıdır: dosya/dizin oluşturma, dosya/dizin silme, dosya/dizin okuma, dosya yazma,…
- FUSE bağlama fonksiyonu belirtilen bağlama noktası için çalıştırılır.
- Tanımlanan bağlama noktasıyla ilgili tüm dosya sistemi çağrıları çekirdek tarafından VFS ‘e gönderilir.
- FUSE çekirdek modülü, dosya sistemi çağrılarını FUSE kütüphanesine yönlendirir.

FUSE temel olarak sistem çağrılarını FUSE çekirdek modülünden FUSE kütüphanesine, verileri de FUSE kütüphanesinden FUSE çekirdek modülüne aktarır. Bu aktarma işlemlerini bağlama sırasında açtığı */dev/fuse* karakter aygıtı üzerinden gerçekleştirir.

Şekil 2 'de FUSE 'un nasıl çalıştığı resmedilmiştir. Bu şekilde bağlanma noktası */tmp/fuse* ve çalışan FUSE programı *example/hello* 'dur. FUSE çekirdek modülü ve FUSE kütüphanesi */dev/fuse* 'un açılmasıyla elde edilen özel bir dosya tanımlayıcısı (file descriptor) ile iletişim kurarlar. */dev/fuse* aygıtı birden fazla sayıda açılabilir ve her açılmasında elde edilen dosya tanımlayıcısı, bu dosya tanımlayıcısı ile bağlanan dosya sisteminin birbiriyle eşleştirilmesi için *mount* sistem çağrısına verilir. Bu mekanizma ile – FUSE ‘un işletim sistemi ile kullanıcı uzayı arasında çalışması– dosya sistemi uygulamalarını çok fazla işi yeniden yapmaktan kurtarmıştır.



Şekil 18: FUSE Çalışma Şeması [3]

2.3.2 Neden FUSE?

Avantajlar:

- Kullanıcı uzayında kod yazmak çekirdek uzayında kod yazmaktan daha esnek ve kolaydır.
- Dosya sistemi kodu işletim sisteminden bağımsız hale gelir.
- Çekirdeğin yeniden derlenmesine gerek yoktur.
- Ayrıcalıklı hakları olmayan normal kullanıcıların dosya sistemi bağlamasını olanaklı kılar. Bu sayede dosya sistemi, kendisini bağlayan kullanıcının haklarıyla çalışır.

Dezavantaj:

- Verilerin gereksiz kopyası ve kullanıcı uzayı – çekirdek uzayı arasında çok fazla geçiş yapılması performansı düşürür.

2.3.3 FUSE Kullanan Dosya Sistemleri

FUSE tabanlı yaklaşık 100 adet dosya sistemi yazılımı bulunmaktadır. Bunlara aşağıdaki kategorilerde örnekler verilebilir [13]:

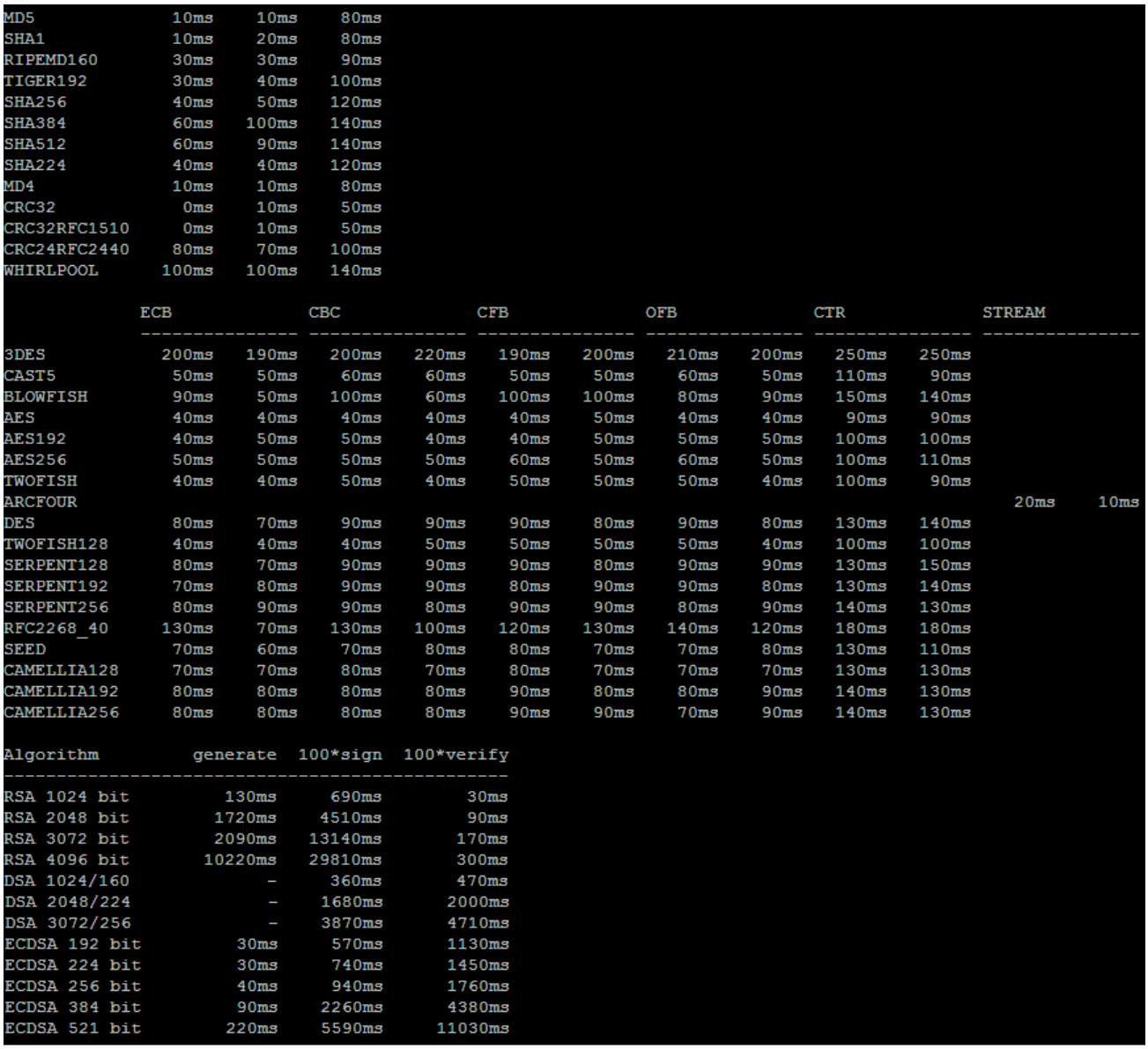
- Genel Amaçlı Dosya Sistemi Yazılımları:
  - GnomeVFS2
  - Mountlo
  - httpfs
  - ClamFS
  - vmware-mount
- Arşivleme Amaçlı Dosya Sistemi Yazılımları:

    - rarfs
    - avfs
    - fuse.gunzip
- Sıkıştırma Amaçlı Dosya Sistemi Yazılımları:
    - fuseiso
    - compFUSEd
    - FuseCompress
    - fusecloop
- Veritabanı Dosya Sistemi Yazılımları:
    - ReIFS
    - Fuse::DBI
    - fuse-dbfs
    - mysqlfs
    - TagFS
    - DBToy
    - VaultFS
- Şifreli Dosya Sistemi Yazılımları:
    - EncFS
    - TrueCrypt
    - PhoneBook
    - CryptoFS
    - MinorFS
    - MetFS
- Çoklu Ortam Dosya Sistemi Yazılımları:
    - YouTubeFS
    - gphoto2-fuse-fs
    - Siefs
    - djmount
    - FUSEpod
    - playlistfs
    - mp3fs
    - DVDfs
    - fusedaap
    - Bluetooth File System
    - BitTorrent File System
    - MythTVfs
- Donanım Dosya Sistemi Yazılımları:
    - OWFS
    - k8055fs
- Takip Amaçlı Dosya Sistemi Yazılımları:
    - LoggedFS
    - POWfs
- Ağ Dosya Sistemi Yazılımları:
    - SMB for FUSE
    - Fusedav
    - Gmailfs
    - SshFS
    - SMBNetFS
    - WikipediaFS
    - CurlFtpFS

    - GridFtpFS
    - BlogFS
    - imapfs
    - Simple TCP Fuse FS
    - FuseFTP
- Suni Dosya Sistemi Yazılımları:
    - ZFS
    - NTFS-3G
    - ext2fuse
    - FatFuse
    - FUR
- Birleştirme Amaçlı Dosya Sistemi Yazılımları:
    - FunionFS
    - unionfs-fuse
- Sürümleme Amaçlı Dosya Sistemi Yazılımları:
    - CopyFS
    - CvsFS
    - gitfs
    - SvnFs

2.4 Libgcrypt

libgcrypt, GnuPG kodunu esas alan genel amaçlı kriptoloji kütüphanesidir. Simetrik şifreleyiciler (AES, DES, Blowfish, Twofish, RC4), özet alma algoritmaları (CRC32, CRC32RFC1510, CRC24RFC2440, MD4, MD5, RIPE-MD160, SHA-1, SHA224, SHA256, SHA384, SHA512, TIGER-192, WHIRLPOOL), MAC ‘ler (bütün özüt fonksiyonları için HMAC), açık anahtar algoritmaları (DSA, ElGamal, RSA), büyük tam sayı fonksiyonları, rastgele sayı üreten fonksiyonlar ve yardımcı pek çok aracı içeren geniş bir kaynaktır.



MD5	10ms	10ms	80ms
SHA1	10ms	20ms	80ms
RIPEMD160	30ms	30ms	90ms
TIGER192	30ms	40ms	100ms
SHA256	40ms	50ms	120ms
SHA384	60ms	100ms	140ms
SHA512	60ms	90ms	140ms
SHA224	40ms	40ms	120ms
MD4	10ms	10ms	80ms
CRC32	0ms	10ms	50ms
CRC32RFC1510	0ms	10ms	50ms
CRC24RFC2440	80ms	70ms	100ms
WHIRLPOOL	100ms	100ms	140ms

	ECB		CBC		CFB		OFB		CTR		STREAM	
3DES	200ms	190ms	200ms	220ms	190ms	200ms	210ms	200ms	250ms	250ms		
CAST5	50ms	50ms	60ms	60ms	50ms	50ms	60ms	50ms	110ms	90ms		
BLOWFISH	90ms	50ms	100ms	60ms	100ms	100ms	80ms	90ms	150ms	140ms		
AES	40ms	40ms	40ms	40ms	40ms	50ms	40ms	40ms	90ms	90ms		
AES192	40ms	50ms	50ms	40ms	40ms	50ms	50ms	50ms	100ms	100ms		
AES256	50ms	50ms	50ms	50ms	60ms	50ms	60ms	50ms	100ms	110ms		
TWOFISH	40ms	40ms	50ms	40ms	50ms	50ms	50ms	40ms	100ms	90ms		
ARCFOUR											20ms	10ms
DES	80ms	70ms	90ms	90ms	90ms	80ms	90ms	80ms	130ms	140ms		
TWOFISH128	40ms	40ms	40ms	50ms	50ms	50ms	50ms	40ms	100ms	100ms		
SERPENT128	80ms	70ms	90ms	90ms	90ms	80ms	90ms	90ms	130ms	150ms		
SERPENT192	70ms	80ms	90ms	90ms	80ms	90ms	90ms	80ms	130ms	140ms		
SERPENT256	80ms	90ms	90ms	80ms	90ms	90ms	80ms	90ms	140ms	130ms		
RFC2268_40	130ms	70ms	130ms	100ms	120ms	130ms	140ms	120ms	180ms	180ms		
SEED	70ms	60ms	70ms	80ms	80ms	70ms	70ms	80ms	130ms	110ms		
CAMELLIA128	70ms	70ms	80ms	70ms	80ms	70ms	70ms	70ms	130ms	130ms		
CAMELLIA192	80ms	80ms	80ms	80ms	90ms	80ms	80ms	90ms	140ms	130ms		
CAMELLIA256	80ms	80ms	80ms	80ms	90ms	90ms	70ms	90ms	140ms	130ms		

Algorithm	generate	100*sign	100*verify
RSA 1024 bit	130ms	690ms	30ms
RSA 2048 bit	1720ms	4510ms	90ms
RSA 3072 bit	2090ms	13140ms	170ms
RSA 4096 bit	10220ms	29810ms	300ms
DSA 1024/160	-	360ms	470ms
DSA 2048/224	-	1680ms	2000ms
DSA 3072/256	-	3870ms	4710ms
ECDSA 192 bit	30ms	570ms	1130ms
ECDSA 224 bit	30ms	740ms	1450ms
ECDSA 256 bit	40ms	940ms	1760ms
ECDSA 384 bit	90ms	2260ms	4380ms
ECDSA 521 bit	220ms	5590ms	11030ms

Şekil 19: Libgcrypt ile Yapılmış Bir Karşılaştırma

2.5 Libtar

Tar (**T**ape **AR**chive), arşiv dosya biçimidir. Bu dosyalar UNIX/Linux işletim sistemlerinde tar komutu ile üretilir. Tar dosya tipi önce POSIX.1-1988 ve sonra da POSIX.1-2001 ile standartlaştırılmıştır. Bir grup dosyayı, kullanıcı ve grup izinleri, tarih ve dizin yapısı gibi bir takım dosya sistemi bilgilerini koruyarak tek ve daha büyük bir dosya içinde saklamak amacı ile kullanılır. Tar özellikle teyp sürücüler gibi yedekleme amacı ile kullanılan sıralı erişimli (sequential access) aygıtlarda kullanılmak üzere geliştirildi. Günümüzde özellikle arşivleme aracı olarak kullanılmaktadır. [23]

Libtar; POSIX ve GNU tar dosyalarını yönetmek için bir C programlama dilinde yazılmış bir kütüphanedir. Tar arşivine dosya ekleme ve tar arşivinden dosya çıkarma özelliklerine sahiptir. Tar arşivinden tek bir dosyayı çıkarabildiği gibi tüm arşivi tek seferde açabilir. Kullanıcının kendi tar arşivlerini okumasına/yazmasına yarayan fonksiyonlar sunar.

2.6 OpenBSD Katar Fonksiyonları

MetFS kaynak kodunda OpenBSD ‘den alınan strdup(), strcat(), strcpy() ve strstr() gibi fonksiyonlar kullanılmıştır. Bu fonksiyonlar diğer açık kaynak kodlu işletim sistemlerinin katar fonksiyonlarından daha hızlı ve daha güvenli çalışmaktadır. strcpy() ve strcat() fonksiyonlarının daha güvenli oldukları düşünülen strncpy() ve strncat() fonksiyonlarında çeşitli sorunlar vardır. Fonksiyonların aldıkları katar uzunluğunun alışılanın dışında ve sezgisel olması bu sorunların birincisi olarak gösterilebilir. Diğer sorun ise strncpy() fonksiyonunun kopyalama işleminden sonra hedef kataın kalan kısmını sıfırlarla doldurarak performansı düşürmesidir. En önemli nokta ise katarların NULL karakteriyle sonlandırılmasıdır ki bu durumu yalnızca OpenBSD ‘nin strlcpy() ve strlcat() fonksiyonları garanti eder. [24]

strlcpy() ve strlcat() fonksiyonları, hedef kataın bir NULL karakteriyle sonlanmasını garanti etmelerinin dışında strncpy() ve strncat() fonksiyonlarıyla tamamen aynıdır. Öte yandan hem Apache grubu [25] hem de NCurses [26] geliştiricileri strncpy() fonksiyonunu bu fonksiyonun kodun performansını çok düşürdüğünü öne sürerek kullanmayı bıraktılar [27]. Aşağıdaki tabloda OpenBSD ‘nin katar fonksiyonları ile yapılan testin sonuçları görülmektedir (testte “this is just a string” cümlesi 1024 sekizli uzunluğundaki tampona OpenBSD-2.5 üzerinde 1000 kez yazdırılmıştır).

İşlemci Mimarisi	Fonksiyon	Zaman (sn)
Motorola 68K	strcpy	0.137
Motorola 68K	strncpy	0.464
Motorola 68K	strlcpy	0.14
Alpha	strcpy	0.018
Alpha	strncpy	0.10
Alpha	strlcpy	0.02

Şekil 20: OpenBSD ve Diğer Platformların Katar Fonksiyonlarının Karşılaştırılması

Daniel J. Bernstein ‘ın [28] yazdığı veritabanı yazılımı olan CDB ‘nin [29] str_len() fonksiyonu da MetFS projesinde kullanılmıştır. str_len() ‘i klasik katar uzunluğunu ölçme fonksiyonlarından ayıran en önemli özellik her adımda tek değil, dört karakter saymaya çalışmasıdır. Bu deneysel bir veri olup her durumda strlen() fonksiyonundan daha hızlı çalışacağı kesin değildir; ancak genelde daha hızlı çalıştığı görülmüştür.

3 TASARIM, GERÇEKLEME VE TEST

İlk aşamada FUSE, TrueCrypt ve EncFS 'in ne oldukları, nasıl çalıştıkları ve ne işe yaradıkları gibi konularda bilgi edinmek gereklidir. Bu nedenle bahsedilen yazılımların dokümanları okunup yazılımlar kurulur. Bazı testler ve denemeler yapılır, çalışma prensipleri dikkatle incelenir ve kaynak kodlar taranır.

Bir sonraki aşamada basit ama kontrolün elde tutulabileceği FUSE tabanlı bir dosya sisteminin yazılması gereklidir. Bu nedenle FUSE ile yazılmış dosya sistemi örnekleri (*fuse-2.7.2/examples* dizininde yer alan *hello.c*, *fusexmp.c*, *fusexmp_fh.c*) incelenir. Bu kodlar detaylı inceleme modunda (FUSE 'a verilen -d parametresi aracılığıyla) çalıştırılarak dosya sistemi üzerinde verilen *ls*, *chown*, *chmod*, *mkdir*, v.s. gibi komutlara FUSE 'un hangi fonksiyonları çalıştırarak yanıt verdiği gözlenir. Örneğin; aşağıdaki çıktıda *ls* komutu çalıştırıldığında FUSE 'un yaptığı işler görülmektedir (sırasıyla *GETATTR*, *OPENDIR*, *GETATTR*, *READDIR* ve *RELEASEDIR* fonksiyonları çalışır).

```
[root@enderunix fuse-2.7.2]# example/fusexmp_fh /root/bitirme/mount/ -
omodules=subdir,subdir=/root/bitirme/enc/ -d
unique: 1, opcode: INIT (26), nodeid: 0, insize: 56
INIT: 7.8
flags=0x00000003
max_readahead=0x00020000
   INIT: 7.8
   flags=0x00000001
   max_readahead=0x00020000
   max_write=0x00020000
   unique: 1, error: 0 (Success), outsize: 40
unique: 2, opcode: GETATTR (3), nodeid: 1, insize: 40
   unique: 2, error: 0 (Success), outsize: 112
unique: 3, opcode: GETATTR (3), nodeid: 1, insize: 40
   unique: 3, error: 0 (Success), outsize: 112
unique: 4, opcode: OPENDIR (27), nodeid: 1, insize: 48
   unique: 4, error: 0 (Success), outsize: 32
unique: 5, opcode: GETATTR (3), nodeid: 1, insize: 40
   unique: 5, error: 0 (Success), outsize: 112
unique: 6, opcode: READDIR (28), nodeid: 1, insize: 64
   unique: 6, error: 0 (Success), outsize: 80
unique: 7, opcode: READDIR (28), nodeid: 1, insize: 64
   unique: 7, error: 0 (Success), outsize: 16
unique: 8, opcode: RELEASEDIR (29), nodeid: 1, insize: 64
   unique: 8, error: 0 (Success), outsize: 16
^C
[root@enderunix fuse-2.7.2]#
```

Şekil 21: ls Komutunun Yaptığı Dosya Sistemi Çağrıları

fusexmp_fh.c kaynak kodunun açıklama satırlarında kodun “gcc -Wall `pkg-config fuse --cflags --libs` -lulockmgr fusexmp_fh.c -o fusexmp_fh” komutuyla derlenebileceği yazılıdır. Ancak *fusexmp_fh.c* kodu belirtilen komutla derlendiğinde *chmod* komutunun ardından “Transport endpoint not connected” ve “Software caused connection abort” hataları alınmaktadır. Bu nedenle *fusexmp_fh.c* örneğini tek başına nasıl derlenebileceğini bulmak için *fusexmp_fh.c* üzerinde kodun çalışma şeklini değiştirmeyecek küçük bir değişiklik yapılır ve FUSE'un *examples* dizininde *make* komutu çalıştırılır:

```
[root@enderunix example]# make
if gcc -DHAVE_CONFIG_H -I. -I. -I../include  -I../include -
D_FILE_OFFSET_BITS=64 -D_REENTRANT   -Wall -W -Wno-sign-compare -
Wstrict-prototypes -Wmissing-declarations -Wwrite-strings -g -O2 -fno-
strict-aliasing -MT fusexmp_fh.o -MD -MP -MF ".deps/fusexmp_fh.Tpo" -c -
o fusexmp_fh.o fusexmp_fh.c; \
        then mv -f ".deps/fusexmp_fh.Tpo" ".deps/fusexmp_fh.Po"; else rm
-f ".deps/fusexmp_fh.Tpo"; exit 1; fi
/bin/sh ../libtool --tag=CC --mode=link gcc  -Wall -W -Wno-sign-compare
-Wstrict-prototypes -Wmissing-declarations -Wwrite-strings -g -O2 -fno-
strict-aliasing   -o fusexmp_fh  fusexmp_fh.o ../lib/libfuse.la
../lib/libulockmgr.la -pthread -lrt -ldl
libtool: link: gcc -Wall -W -Wno-sign-compare -Wstrict-prototypes -
Wmissing-declarations -Wwrite-strings -g -O2 -fno-strict-aliasing -o
.libs/fusexmp_fh fusexmp_fh.o -pthread  ../lib/.libs/libfuse.so -pthread
../lib/.libs/libulockmgr.so -lrt -ldl -Wl,-rpath -Wl,/usr/local/lib
libtool: link: creating fusexmp_fh
[root@enderunix example]#
```

Şekil 22: *fusexmp_fh.c* Kodunun FUSE Tarafından Derlenmesi

Yukarıdaki komut çıktısında da görüldüğü üzere *fusexmp_fh.c* kaynak kodunun *libtool* [8] ile derlenmesi gerekiyor (*libtool* paketi sistemde yoksa ayrıca kurulmalıdır). Yukarıdaki komutlar sadeleştirince *fusexmp_fh.c* kaynak kodu şu şekilde derlenip sorunsuz çalıştırılabilir:

```
[root@enderunix example]# cc -D_FILE_OFFSET_BITS=64 -D_REENTRANT -Wall -
O2 -g -ansi -fno-strict-aliasing  -c -o fusexmp_fh.o fusexmp_fh.c
[mk@enderunix example]$ libtool --tag=CC --mode=link cc -
D_FILE_OFFSET_BITS=64 -D_REENTRANT -Wall -O2 -g -ansi -fno-strict-
aliasing -pthread -O2 -lrt -lfuse -lulockmgr fusexmp_fh.o -o fusexmp_fh
```

Şekil 23: *fusexmp_fh.c* Kodunun Tek Başına Derlenmesi

Daha sonra *fusexmp_fh.c* düzenlenerek kullanıcı uzayında ilk dosya sistemi yazılır. Bu aşamanın ardından sırada temel kriptoloji kavramlarını öğrenmek vardır.

Yapılan araştırmalarda ilk olarak *ccrypt* [9] yazılımı bulunur ve üzerinde çalışılır; fakat *ccrypt* kullanmaktan bu yazılımın kriptoloji fonksiyonlarını içeren bir kütüphane sunmamasından ötürü vazgeçilir. Turgut Bey 'in önerisi üzerine önce *gpgme* [10] ve daha sonra da *beecrypt* [5] yazılımları incelenir; ancak her ikisi de projeye uygun paketler değildir (*gpgme* ile şifreleme sonucunda oluşan verinin uzunluğu değişkendir ve veri açık bir şekilde diske yazılmaktadır; *beecrypt* 'in ise kullanımı zor, dokümantasyonu kötü ve kodu rahat okunabilir değildir). *gpgme* hakkında incelemeler yapılırken yine aynı projenin bir dalı olan *libgcrypt* dikkati çeker. *libgcrypt* bitirme çalışması için çok uygundur. Çünkü kriptoloji fonksiyonlarını kitaplık fonksiyonları gibi kullanabilmeyi sağlar ve çok iyi bir dokümantasyona sahiptir. *http://www.gnupg.org/documentation/manuals/gcrypt/* bağlantısındaki dokümandan hareketle *libgcrypt* ile çeşitli örnek kodlar yazılıp bu şifreleme süreçlerini dosya sistemiyle birleştirilmesi için çalışmalara başlanır.

Bu süreç yaklaşık 2 hafta gibi uzun sayılabilecek bir zaman alır. Çünkü FUSE 'un diske bir defada en fazla 4096 sekizli (byte) yazabildiğini ve diskten veri okuma esnasında ise ilk seferde 4096, okunacak veri varsa 2. seferde 8192, hala okunacak veri varsa 16384 (bu sayı her defasında 2 katına çıkarak artıyor: 32768, 65536, ...) sekizli okuduğu ancak yapılan ayrıntılı testlerde öğrenilebilir. FUSE 'un yazarken sabit, okurken değişken miktarda veri okuması şifrelemeyi bir hayli zorlaştırmaktadır. Çünkü şifrelerken yazılan

veri miktarı ile deşifre ederken okunan veri miktarının aynı olması gereklidir. Bu nedenle FUSE 'un tek seferde en fazla okuyacağı veri miktarı 4096 sekizli olarak sınırlandırılır.

Öncelikle basit olması nedeniyle blok şifreleme kullanılır. Ancak blok şifreleme 16 sekizliler halinde yapılır ve bu durum veri miktarının 16 'nın katı olmasını gerektirir. Bu nedenle akan veri şifrelemesi tercih edilir. Bu teknikte veri miktarı önemli değildir; çünkü açık veri ile şifreli verinin boyutu aynıdır.

Kullanıcıdan şifre alan süreci yazmak için yapılan araştırmalar sonucunda EncFS 'te olduğu gibi OpenBSD 'nin *readpassphrase* kodu kullanıldı.

Bu noktada MetFS, bir dosya sisteminden beklenen özelliklere sahiptir ve sırada dosya sisteminin bağlı olmadığı durumda tek dosya gibi görünmesini sağlamak var. FAT dosya sistemi sade olması açısından incelenir. Bu konu üzerinde yaklaşık 1 hafta çalışılmasına rağmen dosya sistemlerinde olduğu gibi yer ayırma tablosu (allocation table) gibi yapılar gerçeklenemez. Bu nedenle *libtar* kullanmaya karar verilir. Bunun üzerine *libtar* kaynak kodları incelenip sıkıştırma (*metfs_create_file*) ve açma (*metfs_extract_file*) fonksiyonları yazılır.

Saklı bölüm (*hidden partition*) üzerine araştırma yapılır; ancak varılan sonuç kullanıcı uzayında bu konunun gerçeklenemeyeceğini gösterir.

Bu aşamadan sonra gerekli testler yapılır. *libtar* 'ın tek dosya haline getirilmiş dosya sistemini açabildiği ve *file* komutunun bahsedilen tek dosyayı “POSIX tar archive” olarak tanıdığı görülür. Her ne kadar açılan dosyaların içeriği şifrelenmiş de olsa bu durumun düzeltilmesi gerekir. *libtar* 'ın sıkıştırdığı dosyalar incelediğinde arşiv dosyasının başına arşivin kök (root) dizini, dosya izinleri, içerdiği dosyalarının listesi ve dosya boyutları gibi verileri yazdığı açıkça görülebilir. Bu nedenle ilk 1024 sekizlinin şifrelenerek *libtar* 'ın arşiv dosyasını açması engellenir. Daha sonra yapılan testlerde *libtar* 'ın arşiv dosyalarını açamadığı ve *file* komutunun tek dosyayı *data* dosyası olarak tanıdığı görülür.

MetFS kodundaki pek çok kısmın üzerinde çalıştırıldığı işletim sistemine bağımlı olması nedeniyle *GNU AutoTools* ile bir *configure* betiği yazılmasına karar verildi. Bu amaçla *http://www.enderunix.org/docs/autotools/* adresindeki makale okunarak *Makefile.am*, *Makefile.in*, *configure.in* ve *config.h.in* dosyaları yazıldı. Bu dosyalar yazıldıktan sonra *aclocal && autoheader; automake –add-missing* ve *autoconf* komutlarıyla *configure* dosyası oluşturuldu. Bu adımdan sonra projenin kod stili *style(9)* standardına uygun hale getirilir.

MetFS ‘in örnek kullanımı Şekil 24 ‘te görülmektedir:

```
$ metfs ~metin/mount ~metin/enc                                    (1)
Enter your metfs file name (if you don't have, just hit ENTER):
New    MetFS Password:
Verify MetFS Password:
Enter the file name that will contain your encrypted data [max. 1024
characters]: ~metin/test.mfs
$ cp ~metin/gizli_dosyam ~metin/mount                              (2)
$ fusermount -u ~metin/mount/                                      (3)
$ file ~metin/test.mfs                                             (4)
/home/metin/test.mfs: data
$ metfs ~metin/mount ~metin/enc                                    (5)
```

```
Enter your metfs file name (if you don't have, just hit ENTER):
/home/metin/test.mfs
Enter MetFS Password:
Password OK...
$ ls ~metin/mount
gizli_dosyam
$ fusermount -u ~metin/mount/
```

Şekil 24: MetFS Örnek Kullanımı

Komutların açıklamaları:
1- Bağlanma dizini (verilerin açık hali buradadır) *~metin/mount*, çalışma dizini (şifreli veriler burada tutulacaktır) *~metin/enc* olarak bir MetFS bölümü bağlanmıştır. Burada kullanıcıdan programı ilk defa kullanıyorsa sadece ENTER 'a basması istenmiştir. Ardından verilere erişim şifresi istenmektedir. Son olarak da MetFS bölümü çözüldüğünde kullanıcının verilerinin saklanacağı tek dosyanın adı istenmektedir.
2- Bu aşamada *~metin/mount* dizininde normal olarak -veriler açık haldedir- çalışabilir.
3- MetFS bölümü sistemden çözülmüştür. "*fusermount -u*" ile MetFS bölümünü normal kullanıcılar dahi çözme hakkına sahip olur ama sadece ve sadece MetFS bölümünü bağlayan kullanıcı bu bölümü çözebilir.
4- MetFS tek dosyası normal veri dosyası olarak görülür.
5- Bu kez kullanıcı daha önce oluşturduğu MetFS dosyasını bağlamaktadır. Burada MetFS dosyasının tam dizin adresi verilmelidir.

4 DENEYSEL SONUÇLAR

Sonuç olarak ortaya çıkan yazılımda TrueCrypt 'in sabit disk alanı kullanma, EncFS 'in dosya sistemini çözdükten sonra kullanıcı verilerinin ayrı dosya ve dizinlerde bulunması sorunu yoktur. MetFS hem EncFS gibi dinamik boyutludur -sabit bir disk alanı kullanmaz, veri ekledikçe boyutu artar, sildikçe boyutu azalır- hem de sisteme bağlı olmadığı durumda TrueCrypt gibi tek dosya halinde durur.

Intel® Mobile Celeron 1.5 GHz işlemcili, i386 mimarili ve 128 MB bellekli Fedora Core 8 (2.6.23.9-85 çekirdek) bir sistemde 1.3 GB büyüklüğündeki bir dosya sabit diskteki ext3 dosya sisteminde bir dizinden yine ext3 dosya sistemindeki başka bir dizine 2 dk 37 sn 'de, MetFS dosya sistemine 2 dk 54 sn 'de, EncFS dosya sistemine 4 dk 18 sn 'de, TrueCrypt bölümüne ise 15 dk 20 sn 'de kopyalandı. Ve bu dosya MetFS bölümünde 1.2 GB, EncFS 'te 1.3 GB yer tutmaktadır (TrueCrypt 'te önceden ayrılan sabit disk alanı kullanılmaktadır). Şekil 25 ‘te bu karşılaştırma için çalıştırılan komutlar görülmektedir. Bu şekildeki “real” alanı işlemin başlangıcından sonuna kadar geçen toplam süreyi, “user” alanı programın kendisinin ve çağırdığı kütüphane alt programlarının aldığı süreyi ve “sys” alanı da programın doğrudan veya dolaylı olarak yaptığı sistem çağrılarının aldığı toplam süreyi gösterir. “user” ve “sys” alanlarının toplamı, bu sürecin işlemci zamanını (CPU ‘da geçen süre) gösterir.

```
[root@enderunix OS]# du -h FedoraCore8.iso
1.3G    FedoraCore8.iso
[root@enderunix OS]# time cp FedoraCore8.iso /root/
real    2m36.589s
user    0m0.278s
sys     0m21.339s
[root@enderunix OS]# time cp FedoraCore8.iso /root/metfs-mount/
real    2m54.353s
user    0m0.247s
sys     0m30.018s
[root@enderunix OS]# time cp FedoraCore8.iso /encfs/crypt
real    4m17.799s
user    0m0.280s
sys     0m11.486s
[root@enderunix OS]# time cp FedoraCore8.iso /media/truecrypt1/
real    15m20.412s
user    0m0.274s
sys     1m47.323s
```

Şekil 25: MetFS, EncFS ve TrueCrypt ile Yapılan Performans Testi

Bu durumda MetFS 'in EncFS ve TrueCrypt ‘ten en az % 46 daha hızlı; ext3 dosya sisteminden ise yaklaşık % 10 daha yavaş olduğu görülmektedir.

Öte yandan, kullanıcı izinlerini kıyaslanacak olursa; MetFS de EncFS gibi yalnızca dosya sistemini bağlayan kullanıcının bu bölüme erişmesine izin verir. *root* kullanıcısı “*su normal_kullanıcı*” komutunu verdikten sonra bile MetFS bölümüne erişemez (çünkü FUSE sadece kullanıcı kimlik numarasını -uid- değil, etkin kullanıcı kimlik numarasını -euid- da göz önünde bulundurur). Normal kullanıcıların bir MetFS bölümünü bağlayabilmesi ve çözebilmesi için bu kullanıcının *fuse* grubuna eklenmesi gerekir. TrueCrypt 'te böyle bir izin durumu söz konusu değildir.

MetFS ve EncFS kullanıcı uzayında gerçeklenmiş yazılımlardır ve çekirdekle doğrudan ilgili değillerdir. Buna karşın TrueCrypt 'in 5.0 ‘dan önceki sürümlerinde çekirdek modülleri vardır ve çekirdek güncellemelerinde sorun çıkartabilmektedir. 5.0 sürümünden itibaren ise Linux platformlarda FUSE kütüphanesinin kullanılmasıyla bu sorun giderilmiştir.

Bu sonuçlardan yola çıkarak MetFS 'in diğer güvenlik etkin dosya sistemi yazılımlarına kıyasla daha yüksek performans sunduğunu, daha az disk alanı kullandığını, ayrıcalıklı hakları olmayan normal kullanıcıların verilerini bile *root* (sistemdeki en yetkili kullanıcı) kullanıcısının erişimine kapatarak verilerin tutarlılığını, bütünlüğünü koruduğunu ve kullanıcı uzayında çalışmasından dolayı daha güvenli olduğunu söylemek mümkündür.

5 SONUÇ ve ÖNERİLER

4. kısımda MetFS 'in diğer dosya sistemi yazılımlarından üstün taraflarından bahsedildi. Bu bölümde ise MetFS 'in kötü yanları şöyle sıralanabilir:

- Dosya sistemini bağlama ve çözme süresi EncFS ve TrueCrypt 'e nazaran veri miktarına bağlı olarak uzun zaman alabilir.
- Dosya sistemi bağlıyken kullanılan disk alanı gerçek veri miktarının yaklaşık 2 katıdır. Bunun nedeni kullanıcının herhangi bir sorun yaşaması halinde en azından eski verilerini kurtarabilmesine olanak sağlamak amacıyla eski tek dosyanın yenisi oluşturulana dek silinmemesidir. Buna karşın dosya sistemi bağlı değilken kullanılan disk alanı gerçek veri miktarından daha azdır.
- Tüm seçeneklerin komut satırından parametre olarak alınamaması programın esnekliğini azaltmaktadır. Bu durum biraz da FUSE 'un komut satırından alınan parametreleri kullanmasından kaynaklanmaktadır.

MetFS projesi başkalarının da kullanımına açılmış olup ilerde JAVA ile bir arayüz yazılması planlanmaktadır. Bu nedenle *.mfs* dosya uzantısı MetFS için kaydedilmiştir.

7 KAYNAKLAR

[1] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd Edition, ISBN: 0471128457, John Wiley & Sons, Inc., 1996.

[2] Valient Gough, EncFS Yazılımının Resmi Web Sayfası, *http://www.arg0.net/encfs*, incelenme tarihi: 28.05.2008

[3] Miklos Szeredi, FUSE Yazılımının Resmi Web Sayfası, *http://fuse.sourceforge.net*, incelenme tarihi: 28.05.2008

[4] Metin Kaya, MetFS Yazılımının Resmi Web Sayfası, *http://EnderUNIX.org/metfs*, 12.04.2008

[5] Bob Deblier, BeeCrypt Yazılımının Resmi Web Sayfası, *http://beecrypt.sourceforge.net*, incelenme tarihi: 28.05.2008

[6] Werner Koch, Libgcrypt Yazılımının Resmi Web Sayfası, *http://www.gnupg.org/related_software/libraries.en.html#lib-libgcrypt*, incelenme tarihi: 28.05.2008

[7] Mark D. Roth, Libtar Yazılımının Resmi Web Sayfası, *http://www.feep.net/libtar/*, incelenme tarihi: 28.05.2008

[8] G. V. Vaughan, B. Friesenhahn, P. O'Gorman ve R. Wildenhues; Libtool Yazılımının Resmi Web Sayfası, *http://www.gnu.org/software/libtool/*, incelenme tarihi: 28.05.2008

[9] Peter Selinger, Ccrypt Yazılımının Resmi Web Sayfası, *http://ccrypt.sourceforge.net/*, incelenme tarihi: 28.05.2008

[10] Werner Koch, Gpgme Yazılımının Resmi Web Sayfası, *http://www.gnupg.org/gpgme.html*, incelenme tarihi: 28.05.2008

[11] Style(9) Web Sayfası, *http://www.freebsd.org/cgi/man.cgi?query=style&sektion=9*, incelenme tarihi: 28.05.2008

[12] Miklos Szeredi ve Ralf Hoffmann, *AVFS Yazılımının Resmi Web Sayfası*, *http://avf.sourceforge.net/*, incelenme tarihi: 28.05.2008

[13] Forest Bond, *FUSE Kullanan Dosya Sistemi Yazılımları*, *http://fuse.sourceforge.net/wiki/index.php/FileSystems*, 10.05.2008

[14] Wikipedia, *Kriptografik özüt fonksiyonu*, *http://en.wikipedia.org/wiki/Cryptographic_hash_function*, 12.05.2008

[15] NSA Resmi Sayfası, *http://www.nsa.gov/*, incelenme tarihi: 28.05.2008

[16] Ron Rivest 'in Kişisel Web Sayfası, *http://people.csail.mit.edu/rivest/*, incelenme tarihi: 28.05.2008

[17] Benne de Weger, *Software Integrity Checksum and Code Signing Vulne*rability, *http://www.win.tue.nl/hashclash/SoftIntCodeSign/*, incelenme tarihi: 28.05.2008

[18] Wikipedia, *Birthday attack*, *http://en.wikipedia.org/wiki/Birthday_attack*, 21.05.2008

[19] Wikipedia, *MD5CRK Projesi*, *http://en.wikipedia.org/wiki/MD5CRK*, 07.02.2008

[20] Vlastimil Klima 'nın Tünel Algoritması, *http://eprint.iacr.org/2006/105*, incelenme tarihi: 28.05.2008

[21] Wikipedia, *RC4 Stream Cipher*, *http://en.wikipedia.org/wiki/Stream_cipher*, 03.04.2008

[22] Ron Rivest, *MD5 RFC Kılavuzu*, *http://www.ietf.org/rfc/rfc1321.txt*, incelenme tarihi: 28.05.2008

[23] Wikipedia, *POSIX Tar Dosya Biçimi*, *http://en.wikipedia.org/wiki/Tar_(file_format)*, 24.05.2008

[24] Todd C. Miller ve Theo de Raadt, *strlcpy and strlcat - consistent, safe, string copy and concatenation*, *http://www.gratisoft.us/todd/papers/strlcpy.html*, incelenme tarihi: 28.05.2008

[25] Apache Grubu , *http://httpd.apache.org/*, incelenme tarihi: 28.05.2008
[26] Thomas Dickey, Ncurses Resmi Web Sayfası, *http://www.gnu.org/software/ncurses/*, incelenme tarihi: 28.05.2008
[27] Apache-1.3 Performans İyileştirmesi, *http://httpd.apache.org/docs/1.3/new_features_1_3.html*, incelenme tarihi: 28.05.2008
[28] D.J.B. 'nin Resmi Web Sayfası, *http://cr.yp.to/djb.html*, incelenme tarihi: 28.05.2008
[29] D. J. Bernstein, CDB Resmi Web Sayfası, *http://cr.yp.to/cdb.html*, incelenme tarihi: 28.05.2008
[30] TrueCrypt Yazılımının Resmi Web Sayfası, *http://www.truecrypt.org*, incelenme tarihi: 28.05.2008
[31] Wikipedia, *Block Cipher*, *http://en.wikipedia.org/wiki/Block_cipher*, 02.12.2007
[32] Ertan Kurt, *Kriptografi – Bölüm 1 (Simetrik Kriptografi)*, *http://www.olympos.org/belgeler/bxa/kriptografi-bolum-1-simetrik-kriptografi-5549.html*, 20.05.2001.
[33] Serpent Home Page, *http://www.cl.cam.ac.uk/~rja14/serpent.html*, incelenme tarihi: 29.05.2008
[34] R. Anderson, E. Biham ve L. Knudsen, *Serpent: A Proposal for the Advanced Encryption Standard, http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf*, incelenme tarihi: 29.05.2008
[35] Wikipedia, *Serpent Cipher*, *http://en.wikipedia.org/wiki/Serpent_(cipher)*, 06.05.2008
[36] Wikipedia, *AES*, *http://en.wikipedia.org/wiki/Advanced_Encryption_Standard*, 28.05.2008
[37] NIST Resmi Web Sayfası, *http://www.nist.gov/*, incelenme tarihi: 30.05.2008
[38] F. Chabaud ve A. Joux, *Differential Collisions in SHA-0*, *http://fchabaud.free.fr/English/Publications/sha.pdf*, 19.06.2000
[39] Harlan Yu, *Hash Attacks*, *http://www.freedom-to-tinker.com/archives/000664.html*, incelenme tarihi: 30.05.2008
[40] P. S. L. M. Barreto, *The Whirlpool Hash Function*, *http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html*, 28.10.2006
[41] Wikipedia, *Rijndael Key Schedule*, *http://en.wikipedia.org/wiki/Rijndael_key_schedule*, 13.05.2008
[42] B. Schneier, J. Kelsey, D. Whiting, D. Wagner ve C. Hall, *Twofish: A 128-Bit Block Cipher*, *http://www.schneier.com/paper-twofish-paper.pdf*, 15.05.1998
[43] Wikipedia, *Public Key Cryptography*, *http://en.wikipedia.org/wiki/Public_key_cryptography*, 26.05.2008
[44] S. K. T. Subramanya, *FUSE on OpenSolaris*, *http://opensolaris.org/os/project/fuse/files/Fuse-OpenSolaris.pdf*, 07.04.2007
[45] Werner Koch, *Libgcrypt*, *http://directory.fsf.org/project/libgcrypt/*, 12.09.2006

EK 1 – MetFS Kurulumu

FUSE paketi "yum -y install fuse-devel" (bu komut sadece Fedora Core, Mandrake Mandriva, CentOS gibi Red Hat türevi Linux dağıtımlarında geçerlidir) komutuyla veya *http://sourceforge.net/project/showfiles.php?group_id=121684&package_id=132802* bağlantısından FUSE kaynak kodu indirildikten sonra aşağıdaki komutlar çalıştırılarak kurulabilir:

```
[root@enderunix bitirme]# tar -zxvf fuse-2.7.2.tar.gz
[root@enderunix bitirme]# cd fuse-2.7.2
[root@enderunix fuse-2.7.2]# ./configure
[root@enderunix fuse-2.7.2]# make && make install
```

Libgcrypt paketi "yum -y install libgcrypt-devel" (bu komut sadece Fedora Core, Mandrake Mandriva, CentOS gibi Red Hat türevi Linux dağıtımlarında geçerlidir) komutuyla veya *ftp://ftp.gnupg.org/gcrypt/libgcrypt/libgcrypt-1.4.0.tar.bz2* bağlantısından libgcrypt kaynak kodu indirildikten sonra aşağıdaki komutlar verilerek kurulabilir:

```
[root@enderunix bitirme]# tar -jxvf libgcrypt-1.4.0.tar.bz2
[root@enderunix bitirme]# cd libgcrypt-1.4.0
[root@enderunix libgcrypt-1.4.0]# ./configure
[root@enderunix libgcrypt-1.4.0]# make && make install
```

Libtar paketinin kaynak kodundan kurulması önemle tavsiye edilir. Çünkü; bu kitaplıktan çağrılan fonksiyonların çalışabilmesi için gerekli olan libtar.a arşivine ancak libtar paketi kaynak kodundan kurularak erişilebilir. *ftp://ftp.feep.net/pub/software/libtar/libtar-1.2.11.tar.gz* bağlantısından libtar kaynak kodu indirildikten sonra aşağıdaki komutlarla kurulum yapılabilir:

```
[root@enderunix bitirme]# tar -zxvf libtar-1.2.11.tar.gz
[root@enderunix bitirme]# cd libtar-1.2.11
[root@enderunix libtar-1.2.11]# ./configure
[root@enderunix libtar-1.2.11]# make && make install
```

Bu aşamada MetFS kurulabilir:

```
[root@enderunix bitirme]# tar -zxvf metfs-1.0.tar.gz
[root@enderunix bitirme]# cd metfs-1.0
[root@enderunix metfs-1.0]# ./configure
[root@enderunix metfs-1.0]# make && make install strip
```

EK 2 – MetFS Installation

This is a quick installation guide.

Prerequisites: MetFS depends on *FUSE*, *libgcrypt*, and *libtar*. Thus, these packages must be installed in order to install MetFS.

NOTE: It is highly recommended to install each package from its tarball.

At first, install *FUSE*:
1st way: Issue “yum -y install fuse-devel” or “pkg_add -r fusefs-libs” in Red Hat or FreeBSD respectively.
2nd way: The source code of FUSE should be downloaded from *http://sourceforge.net/project/showfiles.php?group_id=121684&package_id=132802*. Then these commands must be given:

```
# tar -zxvf fuse-2.7.2.tar.gz
# cd fuse-2.7.2
# ./configure
# make && make install
```

Secondly, install *libgcrypt*:
1st way: Issue “yum -y install libgcrypt-devel” or “pkg_add -r libgcrypt” in Red Hat or FreeBSD respectively.
2nd way:

```
# wget ftp://ftp.gnupg.org/gcrypt/libgcrypt/libgcrypt-1.4.0.tar.bz2
# tar -jxvf libgcrypt-1.4.0.tar.bz2
# cd libgcrypt-1.4.0
# ./configure
# make && make install
```

Finally, install *libtar*:

```
# wget ftp://ftp.feep.net/pub/software/libtar/libtar-1.2.11.tar.gz
# tar -zxvf libtar-1.2.11.tar.gz
# cd libtar-1.2.11
# ./configure
# make && make install
```

NOTE: On some FreeBSD, Ubuntu, and Debian systems, also the *libtool* and *libgpg-error* packages must be installed manually.

Now, MetFS can be installed smoothly:

```
# wget http://www.EnderUNIX.org/metfs/metfs-1.0.tar.gz
# tar -zxvf metfs-1.0.tar.gz
# cd metfs-1.0
# ./configure
# make && make install strip
```

By default, all FUSE based file systems are visible only to the user who mounted them. No other users (including root) can view the file system contents (although, root makes "su normal_user"; root cannot access the MetFS partition of a non-privileged user). In order to

enable this feature, please add the non-privileged user to the group "fuse"
in the "/etc/group" file.

Sample Usage

First parameter of MetFS is the mount point that data can be written/read in a clear way.
Second parameter of MetFS is the working directory that data is encrypted.

If MetFS is run with "-d" parameter, then it will run as a daemon and prints log/debug messages:

```
$ metfs ~metin/mount -d ~metin/enc
```

Parameters of MetFS must be in this sequence:

```
$ metfs mountpoint -d (optional) working_directory
```

Here is a sample session:

```
$ metfs ~metin/mount ~metin/enc /* metfs mount_directory working directory */
Enter your metfs file name (if you don't have, just hit ENTER):
/* For the first time, just hit ENTER. */
New    MetFS Password:      /* Type your password. It won't be echoed.
*/
Verify MetFS Password:      /* Retype your password.
*/
Enter the file name that will contain your encrypted data [max. 1024
characters]: ~metin/test.metfs
/* Please provide full path to MetFS file. */

$ cp ~metin/my_secret_file ~metin/mount
/* Copy your secret files to the MetFS partition. */

$ echo "İstanbul Teknik Üniversitesi" > ~metin/mount/test.txt

$ cat ~metin/mount/test.txt
İstanbul Teknik Üniversitesi

$ cat ~metin/enc/test.txt
Áõ®´ÛÕ1Ç¡NÑ:±_ÃÛ#ýwÆr~sBÀ,õ½b%

$ fusermount -u ~metin/mount/
/* Then unmount your MetFS partition via fusermount. */

$ file ~metin/test.metfs
/* The MetFS single file seems as a random data file. */
/home/metin/test.metfs: data

$ metfs ~metin/mount ~metin/enc
Enter your metfs file name (if you don't have, just hit ENTER):
/home/metin/test.metfs
/* full path of the file must be given*/
Enter MetFS Password:
Password OK...

$ ls ~metin/mount       /* files and directories can be seen. */
my_secret_file

$ fusermount -u ~metin/mount/
```