ARŞİV FORMATLARI VE ŞİFRELEME İLE DOSYALARIN ARŞİVLEME İŞLEMLERİNİN GERÇEKLEŞTİRİLMESİ

Pamukkale Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü
Lisans Bitirme Tezi

Ramis TAŞGIN
&
İbrahim TAŞDEMİR

Danışman: Yrd. Doç. Dr. Gürhan Gündüz

Haziran, 2009
DENİZLİ

LİSANS TEZİ ONAY FORMU

Ramis TAŞGIN ve İbrahim TAŞDEMİR tarafından Yrd. Doç. Dr. Gürhan GÜNDÜZ yönetiminde hazırlanan "**Arşiv Formatları ve Şifreleme ile Dosyaların Arşivleme İşlemlerinin Gerçekleştirilmesi**" başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Lisans Tezi olarak kabul edilmiştir.

Yrd. Doç. Dr. Gürhan GÜNDÜZ
Danışman

Yrd. Doç. Dr. A. Kadir YALDIR
Jüri Üyesi

Yrd. Doç. Dr. Sezai TOKAT
Jüri Üyesi

Pamukkale Üniversitesi Bilgisayar Mühendisliği Bölümü Bölüm Kurulu'nun
..../ ..../ ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Yrd. Doç. Dr. Sezai Tokat
Bölüm Başkanı

Bu tezin tasarımı, hazırlanması, yürütülmesi, araştırılmalarının yapılması ve bulgularının analizlerinde bilimsel etiğe ve akademik kurallara özenle riayet edildiğini; bu çalışmanın doğrudan birincil ürünü olmayan bulguların, verilerin ve materyallerin bilimsel etiğe uygun olarak kaynak gösterildiğini ve alıntı yapılan çalışmalara atfedildiğini beyan ederim.

Ramis TAŞGIN İbrahim TAŞDEMİR

TEŞEKKÜR

Bu tezin tamamlanmasına değerli yorumlarıyla katkıda bulunan tez danışmanımız Yrd. Doç. Dr. Gürhan GÜNDÜZ'e; öğrenim süremiz boyunca bize her türlü desteği sağlayan tüm hocalarımıza teşekkür ederiz.

ÖZET

ARŞİV FORMATLARI VE ŞİFRELEME İLE DOSYALARIN ARŞİVLEME İŞLEMLERİNİN GERÇEKLEŞTİRİLMESİ

Ramis TAŞGIN
İbrahim TAŞDEMİR
Lisans Tezi, Bilgisayar Mühendisliği
Tez Yöneticisi: Yrd. Doç. Dr. Gürhan Gündüz

Haziran 2009, 55 Sayfa

Bu çalışmada; işletim sistemi ve kullanılan donanımdan bağımsız olarak, istenilen sayı ve büyüklükteki dosyanın, isteğe göre şifre koruma veya çok parçalı dosya seçeneklerine sahip olacak şekilde, desteklenen arşivleme formatları ile sıkıştırılarak saklanması amaçlanmıştır. Verinin hangi arşiv formatında sıkıştırılacağı kullanıcı tarafından belirlenebildiği gibi, desteklenen formatlar arasında veriyi en iyi sıkıştırma oranıyla sıkıştıran formatın bulunması ve sıkıştırma işleminin bu formatta gerçekleştirilmesi de sağlanmaktadır. Belirtilen bu özellikleri gerçekleyen yazılım hazırlanmış, çeşitli işletim sistemlerinde ve çeşitli boyutlardaki farklı dosya tipleri üzerinde test edilmiştir.

Anahtar Kelimeler: AES, BZIP2, GZIP, ZIP, LZMA, Kayıpsız Veri Sıkıştırma Algoritmaları, Veri Sıkıştırma Oranı, Java.

ABSTRACT

IMPLEMENTING FILE ARCHIVE OPERATIONS BY ARCHIVE FORMATS AND DECRYPTING

Ramis TAŞGIN
İbrahim TAŞDEMİR
Undergraduate Thesis, Computer Engineering
Supervisor: Asst. Prof. Dr. Gürhan Gündüz

June 2009, 55 Pages

The aim of this study is to compress any kind of file with supported archive formats with password protection and multi–part options without any limitation by operating system and hardware. The Archive Format which is used to compress file can be determined by user or format which has the best compression ratio for selected file is determined and is used to compress file. The software which has the features that is mentioned was designed and tested on different platforms with various file types which has different sizes.

Key Words: AES, BZIP2, GZIP, ZIP, LZMA, Lossless Data Compression Algorithms, Data Compression Ratio, Java.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

Sayfa

TABLOLAR DİZİNİ

Sayfa

SİMGE VE KISALTMALAR DİZİNİ

MTF	Move-to-front
BWT	Burrows-Wheeler transform
RLE	Run-length encoding
EOF	End of file
LZ77	Lempel-Ziv ‘77
LZW	Lembel-Ziv-Welch
RAR	Roshal archive
ASCII	American standard code for information interchange
CRC	Cyclic redundancy check
GZIP	GNU zip
TAR	Tape archive
USTAR	Uniform standard tape archive
AES	Advanced encryption standard
LZMA	Lempel-Ziv-Markov chain algorithm

1. GİRİŞ

Günümüzde, bilgisayar bilimindeki hızlı gelişmeler, insanların bilgiye erişimini ve bilgiyi saklama isteklerini arttırmış, buna paralel olarak da bilgisayar ortamında saklanan veri miktarı büyük boyutlara ulaşmıştır. Veri miktarının çok büyük boyutlara ulaşması, verinin nasıl daha verimli bir şekilde saklanacağı sorununu da beraberinde getirmiştir.

Veri miktarında meydana gelen hızlı gelişmelere rağmen, veri saklama araçlarının (hard disk, CD, DVD vb.) kapasitelerinde aynı hızda gelişim sağlanamamıştır. Bu sorun verilerin farklı biçimlerde saklanması gerekliliğini beraberinde getirmiştir. Verilerin sıkıştırılarak depolanması, bu soruna karşı bulunan en uygun çözümlerden bir tanesidir.

Veri sıkıştırma işlemi sonucunda, verilerin depolama boyutlarında belli bir miktar kazanç sağlanmaktadır. Bu kazanç verinin türüne ve kullanılan sıkıştırma yöntemine göre farklılık gösterir.

Günümüzde verileri sıkıştırmakta kullanılan pek çok arşiv formatı bulunmaktadır. Bu formatlardan bazıları (ZIP, TAR, 7-ZIP, RAR) birden fazla dosyayı aynı anda sıkıştırmaya olanak tanırken, bazıları (BZIP2, LZMA, GZIP) ise aynı anda sadece tek bir dosyayı sıkıştırmakta kullanılmaktadır.

Günümüzde yaygın olarak kullanılan arşiv formatları arasında belli bir standart yoktur. Bu yüzden bütün formatlar kullanıcıların isteklerine aynı şekilde cevap verememektedir. Örneğin bazı formatlar verilerin şifrelenerek saklanmasına olanak tanırken, bazıları bu özelliği desteklememektedir. Benzer şekilde bazı formatlar sıkıştırılmış dosyaları çok parçalı arşivler halinde saklayabilirken, bazıları bu özelliği sağlayamamaktadır. Bahsedilen bu iki özelliği aynı anda sağlayan format sayısı çok azdır.

Günümüzde kullanılmakta olan işletim sistemlerinin çoğu, yaygın olarak kullanılan arşiv formatlarının hepsini birden destekleyen uygulamalara sahip değildir. Bu da kullanıcının istediği arşivleme formatını, istediği işletim sisteminde kullanarak verilerini

sıkıştırmasına veya uygulamalar tarafından desteklenmeyen bu formatlardaki arşiv dosyalarını açmasına engel olmaktadır.

Arşivleme formatları arasında belli bir standardın olmaması sorunundan yola çıkılarak, platformdan bağımsız, kaynak kodu açık, günümüzde en çok kullanılan arşivleme formatlarından (BZIP2, ZIP, GZIP, LZMA, TAR) birini seçerek arşivleme yapılmasına olanak tanıyan bir arşivleme uygulamasının geliştirilmesine karar verilmiştir. Ayrıca Bu uygulama AES şifreleme algoritmasını kullanarak, arşivleme formatlarının dosya biçiminden farklı bir şekilde, sıkıştırılmış dosyaların şifrelenerek saklanmasına olanak vermektedir. Yine dosya biçimlerinden farklı bir şekilde arşiv dosyalarının çok parçalı arşiv dosyaları halinde saklanması da sağlanmıştır. Dosya biçiminden bağımsız olarak yapılan bu işlemler dosyanın formatını bozacağı için, şifreleme ve çoklu arşiv seçenekleri kullanılarak yapılan arşivleme işlemleri sonucunda oluşturulan dosyalar, sadece bu uygulama tarafından açılabilmektedir.

Günümüzde kullanılmakta olan pek çok arşivleme programı bulunmaktadır. Bu uygulamalardan en çok kullanılanları; ZIP formatında arşivleme yapan WinZip[1], RAR ve ZIP formatlarında arşivleme yapan WinRAR[2] ve 7-ZIP ve ZIP formatlarında arşivleme yapan 7-ZIP[3] programlarıdır. Bu uygulamalar kendi esas formatlarında (WinZIP için ZIP formatı, WinRAR için RAR formatı, 7-ZIP için 7-ZIP formatı) şifrelemeyi ve çok parçalı arşiv oluşturma özelliklerini desteklerler. Yapılan bu şifreleme ve çok parçalı arşiv oluşturma işlemi dosya biçimi dahilindedir ve bu seçenekler kullanılarak oluşturulmuş arşiv dosyaları diğer uygulamalar tarafından açılabilir.

Bu doküman içerisinde, ikinci bölümde veri sıkıştırma algoritmalarının tanımı yapılacak ve arşiv formatlarının yapısında bulunan veri sıkıştırma ve dönüşüm algoritmalarının üzerinde durulacaktır. Üçüncü bölümde ise veri arşiv formatlarının tanımı yapılıp, geliştirilen uygulama içerisinde kullanılan arşiv formatlarından bahsedilecektir. Bir sonraki bölüm olan dördüncü bölümde veri şifreleme algoritmalarından ve geliştirilen yazılımda kullanılan veri şifreleme algoritması olan AES algoritmasından bahsedilecektir. Son bölüm olan beşinci bölümde ise geliştirilen uygulama hakkında bilgi verilecektir.

[1] http://www.winzip.com/
[2] http://www.rarlab.com/
[3] http://www.7-zip.org/

2. VERİ SIKIŞTIRMA

2.1. Veri Sıkıştırma Nedir?

Bilgisayar bilimi ve bilgi teorisinde veri sıkıştırma, belli kodlama şemaları kullanarak bilgiyi kodlanmamış gösterilişinin kullanacağından daha az bit (veya diğer bilgi gösterim elemanları) kullanarak kodlama işlemidir.

2.2. Nasıl Çalışır?

Veri sıkıştırma algoritmaları iki temel bölümden oluşur: Kodlayıcı ve kod çözücü kısım. Kodlayıcı tarafından sıkıştırılarak saklanan veri, tekrar kullanılmasına gerek duyulduğu zaman kod çözücü tarafından çözülerek sıkıştırılmamış haline geri getirilir. Yani bir sıkıştırma algoritmasının kodlama metodu kullanılarak sıkıştırılan verinin, yine aynı algoritmanın kod çözücü metodu ile açılabilmesi gerekir.

Sıkıştırma işlemi hard disk boyutu ve iletim bant genişliği gibi pahalı kaynakların tüketimini azaltmaya yardımcı olur. Diğer taraftan sıkıştırılmış veri kullanılabilmek için çözülmek zorundadır ve bu fazladan işlem bazı uygulamalar için zararlı olabilir. Örnek olarak, görüntü için kullanılan bir sıkıştırma şeması, verinin çözülmesiyle aynı zamanda görüntülenebilmesini sağlamak için yeteri kadar hızlı çözümlemeyi sağlayan pahalı donanımlara ihtiyaç duyabilir. Veri sıkıştırma şemalarının tasarımını bu yüzden pek çok eleman ilgilendirir: sıkıştırma derecesi, verideki bozulma miktarı (eğer bir kayıplı sıkıştırma şeması kullanılıyorsa), veriyi sıkıştırmak ve açmak için gerekli bilgisayar kaynakları gibi.

2.3. Kayıplı ve Kayıpsız Veri Sıkıştırma

Kayıpsız sıkıştırma algoritmaları, veri içerisinde yer alan tekrarlanan kısımların azaltılmasını sağlayarak verilerin sıkıştırılmasını sağlar. Kayıpsız sıkıştırmanın mümkün olmasının en büyük nedeni, çoğu verinin istatiksel artıklık adı verilen bu tekrarlanan verilere sahip olmasıdır.

Başka bir sıkıştırma türü olan kayıplı veri sıkıştırmada, bir miktar veri kaybı kabul edilebilirdir. Genellikle bir kayıplı veri sıkıştırma insanların veriyi nasıl algılayacağı sorusuna dayanan araştırmalar tarafından yönlendirilir. Örneğin insan gözü parlaklıktaki hafif değişimlere renkteki değişimlerden daha duyarlıdır. JPEG görüntü sıkıştırma formatı bu bilgiyi kullanarak, daha az önemli verilerin bazılarını yuvarlama yoluyla kısmen çıkartarak sıkıştırma işlemini gerçekleştirir.

Kayıpsız sıkıştırma şemaları terslenebilirdir, yani esas veri tekrardan üretilebilir. Ancak kayıplı şemalarda böyle bir durum söz konusu değildir.

Belli bir kalıba uymayan bir veriyi sıkıştırırken herhangi bir sıkıştırma algoritması ister istemez başarısız olabilir. Bunun yanında sıkıştırılmış bir veriyi yeniden sıkıştırmaya kalkmak doğal olarak genişlemeye neden olur.

Kayıplı ve kayıpsız sıkıştırma karşılaştırmasının farkını göstermek için aşağıdaki örneği verebiliriz.

25.888888888

Olarak verilen ondalık sayı kayıpsız olarak şu şekilde sıkıştırılabilir:

25.[9]8

Kayıpsız olarak sıkıştırılmış yukarıdaki veri, orijinal veriden daha az yer kaplayacak şekilde ifade edilmiş oldu. Eğer biz bu veriyi kayıplı olarak sıkıştırılmak isteseydik aşağıdaki şekilde sıkıştırabilirdik.

26

Bu şekilde gerçekleştirilen kayıplı bir sıkıştırma işlemi sonucunda, veriyi daha küçük boyutlarda saklayabilmek için orijinal veri kaybedilmiş etmiş oldu.

2.4. Arşiv Formatlarında Kullanılan Kayıpsız Veri Sıkıştırma ve Dönüşüm Algoritmaları

Bu bölümde proje içerisinde kullanılan veri sıkıştırma formatlarının temelini oluşturan veri sıkıştırma ve dönüşüm algoritmalarından bahsedilecektir. Veri sıkıştırma formatları bu algoritmalar içerisinden bir veya daha fazlasını, aynen veya değiştirilmiş

olarak kullanmaktadır.

2.4.1. MTF dönüşümü

Sıkıştırma tekniklerinin performanslarını arttırmak için tasarlanmış bir kodlamadır.

MTF yönteminde her byte değeri, bir listedeki indis değerine göre kodlanır. Bir byte kodlandıktan sonra o değer listenin başına taşınır ve daha sonra sıradaki byte değerine geçilir.

Bir örnek, dönüşümün nasıl olduğunu daha iyi anlatacaktır. Byte değerleri yerine bizim 0-7 değerlerini kodladığımızı düşünün. Şu parçayı dönüştürmek istiyoruz.

524700717

Liste başlangıç olarak (0, 1, 2, 3, 4, 5, 6, 7) değerlerinden oluşur. Parçanın ilk elemanı 5, beşinci indiste bulunduğu için çıktıya 5 değerini ekleriz.

5

5 listenin başına taşınır ve liste (5, 0, 1, 2, 3, 4, 6, 7) olarak elde edilir. Sıradaki eleman 2, şimdi üçüncü indiste bulunuyor.

53

Liste (2, 5, 0, 1, 3, 4, 6, 7) şeklinde değiştirildikten sonra bütün parçalar kodlanana kadar işlem devam ettirilerek aşağıdaki sonuç elde edilir.

535740151

MTF kodlamasının geri dönüşümü kolaydır. Geri dönüşüm işlemi, MTF yöntemiyle kodlanmış listedeki her indisin o indisteki değerle değiştirilmesi ile gerçekleştirilir. Kod çözme ve kodlama arasındaki fark, kodun indisi için her değere bakmak yerine listedeki indisin direkt olarak kullanılmasıdır.

MTF dönüşümü bir mesajın entropisini düşürmek için frekansların yerel ilişki avantajını kullanır. Tüm veriler bu tür bir ilişki göstermezler ve bazı mesajlar için MTF dönüşümü entropiyi arttırabilir.

Entropi, bir elemanın kodlanması için teorik olarak gereken minimum bit sayısını verir. Tüm verinin entropisi o verinin kayıpsız bir şekilde sıkıştırılması için gereken minimum bittir. Entropi ile sıkıştırma sonucunun karşılaştırılması, sıkıştırmanın başarı oranını verir.

MTF'nin önemli bir kullanımı Burrows-Wheeler dönüşümüne dayalı sıkıştırmada yer alır. Burrows-Wheeler dönüşümü yazıdan ve diğer özel veri sınıflarından yerel frekans ilişkisini gösteren bir dizi üretiminde çok iyidir. Sıkıştırmada işlemlerinde son entropi kodlama adımından önce, Burrows-Wheeler dönüşümünü takip eden bir MTF dönüşümünün uygulanması, büyük ölçüde yarar sağlanır.

Örnek olarak Hamlet 'in monoluğunun (to be, or not to be…) orijinal metnini sıkıştırmak isteyelim. Bu mesajın entropisi 7033 bit hesaplanır. Direk MTF dönüşümü uygularsak, sonucun entropisi orijinal değerinden daha yüksek olan 7807 bit olarak hesaplanır. Çünkü bazı diller, örneğin İngilizce, genellikle yüksek bir yerel frekans bağlantısı içermez. Ancak Burrows-Wheeler dönüşümünden sonra MTF dönüşümü uygularsak, 6187 bitlik bir entropi değeri elde ederiz. Burada Burrows-Wheeler dönüşümü mesajın entropisini düşürmez; sadece byte değerlerini MTF dönüşümünde daha etkili olacak şekilde yeniden düzenler.

2.4.2. Burrows-Wheeler dönüşümü

Bir karakter dizisi BWT tarafından dönüştürüldüğünde karakterlerin değerleri değişmez. Karakter düzenini değiştirir. Eğer Orijinal dizinin sıkça karşılaşılan alt dizileri varsa, o zaman dönüştürülmüş dizide, tek bir karakterin bir satırda birden çok kez tekrarlandığı yerleri olacaktır. Burrows-Wheeler dönüşümü, MTF ve RLE ile birlikte kullanıldığında, tekrarlanan karakterleri olan dizileri sıkıştırmak daha kolaydır.

Dönüşüm, yazının tüm rotasyonlarının sıralanması ve daha sonra son sütünün alınması ile yapılır. Örneğin, "^BANANA@" yazısı "BNN^AA@A" ya çevrilir. (@ karakteri 'EOF' işaretçisidir.)

Tablo 2.1 BWT dönüşüm örneği (WEB_7)

Transformation			
Input	**All Rotations**	**Sort the Rows**	**Output**
^BANANA@	^BANANA@ @^BANANA A@^BANAN NA@^BANA ANA@^BAN NANA@^BA ANANA@^B BANANA@^	ANANA@^B ANA@^BAN A@^BANAN BANANA@^ NANA@^BA NA@^BANA ^BANANA@ @^BANANA	BNN^AA@A

BWT'nin en önemli özelliği daha kolay kodlanan bir çıktı üretmesi değil- sıradan bir sıralama da bunu yapabilir.- geri çevrilebilir olması ve son sütundan orijinal metnin üretilmesine izin vermesidir.

BWT algoritmasının neden daha kolay sıkıştırılabilir bir veri yarattığını anlamak için, içinde sıklıkla "the" kelimesi geçen uzun bir İngilizce yazıyı dönüştürelim. Bu yazının rotasyonlarını sıralarsak genellikle "he" ile başlayan rotasyonları birlikte gruplarız ve bu rotasyonun son karakteri ("he" den önceki karakter) "t" olacaktır. Yani dönüşümün sonucu birkaç "t" karakterlerini daha seyrek durumlarla karışmış olarak tutar.

Aşağıdaki pseudo kod BWT kodlama ve kod çözme örneğidir. Girdi dizisi S'nin son karakteri olarak "EOF" kullanılmıştır. EOF'nin yazı içinde bulunmadığı kabul edilir ve sıralama sırasında önemsenmez.

```
function BWT (string s)
   Bir tablo yarat, satırlar s'nin tüm rotasyonları
   Satırları alfabetik olarak sırala
   return (tablonun son sütunu)

function inverseBWT (string s)
   Boş tablo oluştur

   length(s) kere tekrarla
       Tablonun ilk sütunundan önceye s'yi bir sütun olarak ekle
              // ilk ek ilk sütunu oluşturur.
       Satırları alfabetik olarak sırala
   return ('EOF' ile biten satırı)
```

Tablo 2.2 BWT ters dönüşüm örneği (WEB_7)

Ters Dönüşüm			
Girdi			
BNN^AA@A			
1. Ekle	1. Sırala	2. Ekle	2. Sırala
B N N ^ A A @ A	A A A B N N ^ @	BA NA NA ^B AN AN @^ A@	AN AN A@ BA NA NA ^B @^
3. Ekle	3. Sırala	4. Ekle	4. Sırala
BAN NAN NA@ ^BA ANA ANA @^B A@^	ANA ANA A@^ BAN NAN NA@ ^BA @^B	BANA NANA NA@^ ^BAN ANAN ANA@ @^BA A@^B	ANAN ANA@ A@^B BANA NANA NA@^ ^BAN @^BA
5. Ekle	5. Sırala	6. Ekle	6. Sırala
BANAN NANA@ NA@^B ^BANA ANANA ANA@^ @^BAN A@^BA	ANANA ANA@^ A@^BA BANAN NANA@ NA@^B ^BANA @^BAN	BANANA NANA@^ NA@^BA ^BANAN ANANA@ ANA@^B @^BANA A@^BAN	ANANA@ ANA@^B A@^BAN BANANA NANA@^ NA@^BA ^BANAN @^BANA
7. Ekle	7. Sırala	8. Ekle	8. Sırala
BANANA@ NANA@^B NA@^BAN ^BANANA ANANA@^ ANA@^BA @^BANAN A@^BANA	ANANA@^ ANA@^BA A@^BANA BANANA@ NANA@^B NA@^BAN ^BANANA @^BANAN	BANANA@^ NANA@^BA NA@^BANA ^BANANA@ ANANA@^B ANA@^BAN @^BANANA A@^BANAN	ANANA@^B ANA@^BAN A@^BANAN BANANA@^ NANA@^BA NA@^BANA ^BANANA@ @^BANANA
Çıktı			
^BANANA@			

2.4.3. Run-length Encoding algoritması

RLE Algoritması, Orijinal dizi yerine tek bir karakter ve karakter tekrar değerini koyan en kolay veri sıkıştırma metodudur. Birden fazla tekrar barındıran verilerde en yararlı metottur. Tekrarı bulunmayan verilerde kullanımı önerilmez. Aksi takdirde veri boyutunu iki katına kadar arttırabilir.

Örnek olarak, beyaz ekran üzerine siyah yazının bulunduğu bir ekranı alalım. Boşlukta pek çok beyaz piksel tekrarı ve yazıda da pek çok siyah piksel tekrarı olacaktır. Tarama yaptığımızda siyahlar için S, beyazlar için B yazan dizi şu şekilde olsun:

BBBBBBBBBBBBSBBBBBBBBBBBBSSSBBBBBBBBBBBBBBBBBBBBBBBSBBBBBBBBBBBBBB

Eğer yukarıdaki diziye RLE uygularsak aşağıdaki çıktıyı elde ederiz:

12B1S12B3S24B1S14B

Görüldüğü üzere, RLE algoritması 67 karakterlik diziyi sadece 18 karakter ile gösterilecek şekilde sıkıştırmıştır. Tabi ki görüntüleri saklamak için kullanılan format buradaki gibi ASCII karakterler yerine ikili kodlardır, ama sistem yine aynıdır.

2.4.4. Huffman algoritması

Huffman Algoritması, verileri kayıpsız olarak sıkıştırmakta kullanılan bir entropi kodlama algoritmasıdır. Değişken uzunluklu bir kod tablosunun, verilen verideki tüm karakterler için tekrar değerine dayalı özel bir yolla, karakteri kodlamak için kullanımını ifade eder.

Huffman algoritmasında her sembolün gösterimini seçmek için özel bir metot kullanır. En çok rastlanan karakterleri için daha az rastlanan karakterlerden daha kısa bit dizileri ile ifade eden bir öntakı (prefix) kod oluşturur. (Bir sembolü ifade etmekte kullanılan bit katarı, başka bir sembolü ifade etmekte kullanılan bit katarının öntakısı olamaz.)

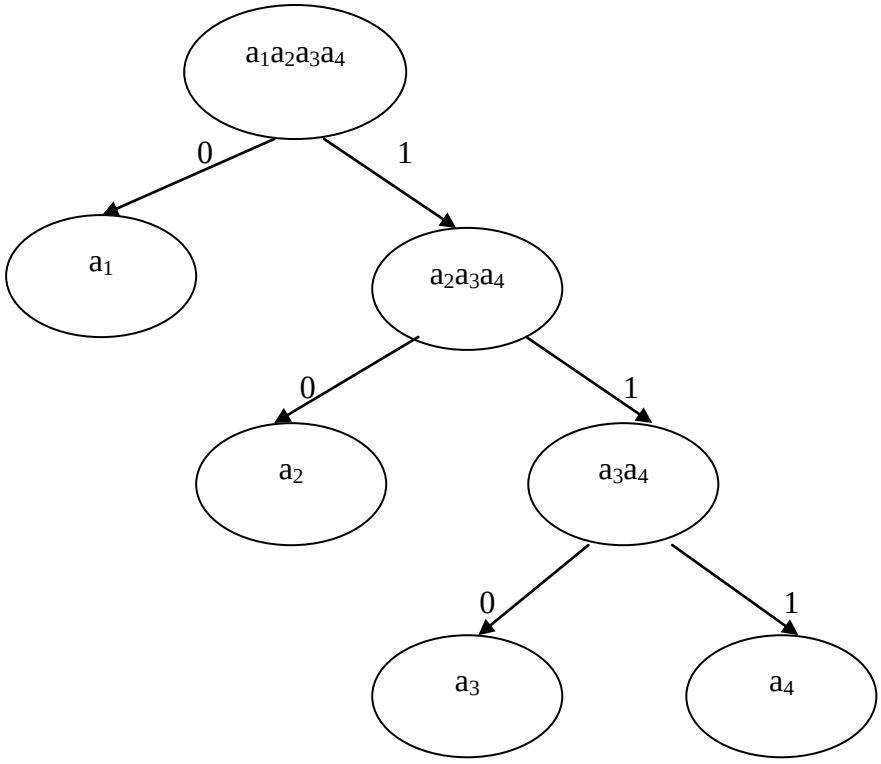
Bir kaynağın olasılıkları {0.4, 0.35, 0.2, 0.05} olan, 4 farklı sembol $\{a_1,a_2,a_3,a_4\}$ ürettiğini varsayalım. En düşük olasılığa sahip iki sembol alınarak soldan sağa bir ikili

ağaç oluşturulur, ikisinin toplam olasılığına eşit yeni bir değer elde edilir. Tek bir sembol kalıncaya kadar bu işlem tekrarlanır. Ağaç sağdan sola, farklı kollara farklı bitler atayarak tersi şekilde de okunabilir. Son Huffman kodu:

Tablo 2.3 Huffman kod örneği

Sembol	Kod
a_1	0
a_2	10
a_3	110
a_4	111

4 sembolün oluşturduğu bir sinyali göstermenin standart yolu 2 bit/sembol kullanmaktır, ama elimizdeki kaynağın entropisi 1.73 bit/semboldür. Eğer sinyali göstermek için Huffman kodu kullanılırsa, ortalama uzunluk 1.85 bit/sembole düşürülür.



Şekil 2.1 Örnek Huffman ağacı

Yukarıdaki şekilde olduğu gibi, düğümlerin bir ikili ağacını oluşturma yoluyla kolay şekilde Huffman algoritması uygulanabilir. Ağacın büyüklüğü kullanılan sembol sayısına, n, bağlıdır. Her düğüm bir yaprak veya iç düğüm olabilir. Ağaçta yer alan tüm

düğümler yapraktır ve sembolün kendisini, sembol ağırlığını ve isteğe bağlı olarak ebeveyn düğüme olan bir bağı tutar. İç düğümler sembol ağırlıklarını, iki çocuk düğüme olan bağı ve isteğe bağlı olarak da ebeveyne olan bağı tutar. Ağaç yapısında genelde bit '0' devam eden sol çocuğu ve bit '1' sağ çocuğu temsil eder. Biten bir ağacın n tane yaprak düğüm ve n-1 tane iç düğümü vardır.

En basit yapım algoritması, en düşük olasılığa sahip düğüme en yüksek önceliğin verildiği bir öncelik kuyruğu kullanır:

1. Her sembol için bir yaprak düğümü yarat ve öncelik kuyruğuna at.
2. Kuyrukta birden fazla düğüm olduğu sürece devam et:
    1. En yüksek öncelikli (en düşük olasılık) iki düğümü kuyruktan çıkar.
    2. Bu iki düğümü çocuk olarak alarak, olasılığı bu iki düğümün olasılığına eşit yeni bir iç düğüm yarat.
    3. Yeni düğümü kuyruğa ekle.
3. Kalan düğüm kök düğümdür ve ağaç tamamlanmıştır.

Eğer semboller olasılığa göre sıralanmışsa, iki kuyruk kullanarak bir Huffman ağacını yaratan bir doğrusal zaman metodu vardır. Kuyruklardan ilki başlangıç ağırlıklarını, ikincisi de birleştirilmiş ağırlıkları tutar.

1. Sembol kadar yaprakla başla.
2. Tüm yaprakları artan olasılık değeri sırasıyla ilk kuyruğa at.
3. Kuyruklarda birden fazla düğüm olduğu sürece devam et:
    1. İki kuyruktaki en düşük ağırlığa sahip iki düğümü kuyruktan çıkar.
    2. Bu iki düğümü çocuk alarak, olasılığı bu iki düğümün olasılığına eşit yeni bir iç düğüm yarat.
    3. Yeni düğümü ikinci kuyruğa ekle.
4. Kalan düğüm kök düğümdür ve ağaç tamamlanmıştır.

2.4.5. Aritmetik kodlama

Aritmetik kodlama algoritması Huffman kodlamasında olduğu gibi bir kodlama ağacı kurulması gerektirmez; ancak karakterlerin kullanım sıklığı bilinmelidir.

Aritmetik kodlamanın temeli, kodlanacak veri parçası 0 ile 1 arasında gerçel sayı aralığı ile temsil etmeye dayanır. Veri parçasındaki her karakter bu gerçel sayı aralığını daraltır. Aralık ne kadar darsa, onu temsil etmek için gerekli bit sayısı artar; genişse azalır. Kullanma sıklığı olasılığı küçük olan karakterler bu aralığı hızlı bir şekilde azaltırken, büyük olanlar daha az daraltır. Eğer veride fazlalık çoksa, sayı aralığı az daralacağı için, veri az sayıda bit ile kodlanabilir. Kodlama işlemine geçmeden önce kümedeki karakterler, olasılıkları, (kendi ve kendinden önceki karakterlerin) toplanmış olasılıkları ve her karakterin sayısal aralığından oluşan bir çizelge oluşturulur; kodlama işlemi, bu çizelgedeki değerlerin aritmetik işlemlerde kullanılmasıyla gerçeklenir. Amaç aralığın hesaplanacağı iki gerçel sayıyı bulmaktır; birine sağdaki, diğerine soldaki sayı denir.

2.4.6. LZ77 algoritması

LZ77 algoritmasında pencere olarak adlandırılan bellek alanı ve katar uyuşma yordamı kullanılır. Kodlanacak veriye ait karakterler bu pencere içine sokulur. Herhangi bir anda, kodlanacak veriye ait bir hecenin pencere içinde olup olmadığına bakılır. Eğer pencere içinde varsa, orada bulunduğu yerin adresi (a) ve uyuşma gösteren hecenin uzunluğu (l) kodlamada kullanılır. Bu iki değişken, (a,l) biçiminde gösterilir. Ve işaretçi olarak adlandırılır; çünkü uyuşma gösteren heceyi işaret eder. Örneğin işaretçi değeri (33,23) ise, 33. karakterden sonra 23 karakter uzunluğunda; (12,2) ise 12.karakterden sonra 2 karakter uzunluğunda uyuşma oldu anlamına gelir.

LZ algoritmasında pencere boyu büyüdükçe sıkıştırma başarımı artar. Ancak pencere boyutunun çok fazla büyümesi işaretçi için gerekli bit sayısı arttıracağından belirli bir boydan sonra arttırmanın yararı olmaz. Pencere boyunun büyümesi kodlama zamanını artırır. Çünkü uyuşma yordamı uyuşan heceyi aramak için daha fazla çevrim gerekir. Dolayısıyla kodlama için geçen zaman büyür. Pencere içinde uyuşma gösteren hecenin etkin bir şekilde aranması ve pencere boyunun ne olacağı birçok çalışmaya kaynak olmuştur. Öyle ki, LZ77'nin türevleri olan algoritmaların hemen hemen hepsi, sıkıştırma başarımını azaltmadan bu ikisinin etkin bir şekilde gerçeklenmesi üzerinedir. Türevlerin çoğunda ikisi arasında denge kurmak amacıyla pencere sabit tutulur ve hızlı arama yapabilmek için doğrusal dizi yerine ikili ağaç, sözlük ağacı (trie tree) veya çırpı fonksiyonu kullanılır.

LZ77 algoritmasında, kodlama için kullanılan temel yordam aşağıda verildiği gibidir. Pencere olarak doğrusal dizi kullanılırsa, uyuşan hecenin bulunması için gerekli arama işlemi zaman alır, dolayısıyla, kodlama zamanı uzun sürer. Ancak çözme zamanı oldukça kısadır. LZ77 algoritması bir kez kodlanıp daha sonra defalarca çözme gerektiren uygulamalara çok uygundur. Örneğin, elektronik kitapların, paket programların yardım dosyalarının saklanması gibi.

```
while( kodlanacak veri olduğu süre ) {
     En uzun uyuşma gösteren heceye, a ve l 'yi bul:
     if (l < en küçük uyuşma uzunluğu )
            Yalnızca sıradaki ilk karakteri gönder ve
            pencereyi 1 kare kaydır.
     else
            İşaretçiyi (a,l)'e gönder ve pencereyi / kadar
            kaydır.
}
```

Yukarıdaki yordama göre veri "annebababababaannebabaanne" ise, kodlama işlemi sonucu "anneba(6,2)(8,4)(12,8)(1,4) biçiminde olur. Görüldüğü gibi kodlama işleminde uyuşma göstermeyen karakterler olduğu gibi kalırken, gösterenler işaretçiyle yer değiştirmektedir. Uygulamada, kodda bulunan işaretçileri karakterlerden ayırmak için bayrak biti veya özel desen kullanılması gerekir.

2.4.7. Deflate algoritması

Deflate algoritması, LZ77 algoritması ve Huffman kodlaması kullanan kayıpsız bir veri sıkıştırma algoritmasıdır.

Deflate algoritması ile sıkıştırılmış bir veri seri halindeki bloklardan oluşur. Her bloğun başlangıcında 3-bit uzunluğundaki başlık verisi bulunur.

- 1-bit: veri içerisindeki son blok:
    - 1: bu bloğun veri içerisindeki son blok olduğunu gösterir.
    - 0: bu bloktan sonra gelen bloklar mevcut.

- 2-bit: Bu blok için kullanılan kodlama metodu:
    - 00: Bir depolanmış/ham/hazır blok.
    - 01: Statik Huffman algoritması kullanılarak sıkıştırılmış blok.
    - 10: Huffman tablosu desteği ile sıkıştırılmış blok.
    - 11: Ayrılmış blok.

Blokların sıkıştırılmasında çoğunlukla metot 10 kullanır. Gerekli Huffman ağacını yaratmak için kullanılan komutlar blok başlığını takip eder.

Sıkıştırma iki adımda gerçekleşir:

- Kopya dizileri işaretçilerle yer değiştirmek ve eşlemek
- Sembolleri frekansı tabanlı yeni, ağırlıklı sembollerle değiştirme.

Sıkıştırma blokları içinde, eğer tekrarlanan byte katarları tespit edilirse, bu katarların veri içerisinde önceden görüldüğü yere ait bir geri başvuru eklenir. Bir önceki katarla eşleşen kodlanmış veri 3-258 byte aralığında bir uzunluğa ve 1-32768 byte aralığında bir mesafeye sahip olabilir.

İkinci adım Huffman kodlama işlemleridir. 288 sembollük bir boşluk içeren bir ağaç oluşturulur. Bu ağaca ait byte dağılımı şöyledir;

- 0–255: Sembolleri temsil eder.
- 256: Blok sonunu gösterir. Eğer son bloksa işlem bitirilir, değilse sonraki blok işlenmeye başlanır.
- 257–285: Fazladan bitlerle birleştirilmiş, 3-258 byte uzunluğundaki eşleşmeyi temsil eder.
- 286,287: Kullanılmayan bitler.

3. ARŞİV FORMATLARI

3.1. Arşiv Formatı Nedir?

Bir arşiv dosyası bir ya da birden fazla dosyayı, dosyaya ait bilgiler, dosya dizin yapısı, hata saptama ve düzeltme bilgisini içerecek şekilde arşivleyen dosyalardır. Genellikle kayıpsız sıkıştırma algoritmalarını kullanırlar. Ayrıca dosyaların tamamını veya bir parçasını şifreleyebilirler. Genellikle çok dosyadan oluşan verileri tek bir dosya haline getirerek taşınabilirliği ve dosya boyutunu düşürerek, saklamayı kolaylaştırmakta kullanılırlar.

3.2. Arşiv Formatlarının Tarihçesi

Arşiv dosyaları orijinal olarak, önemli dosyaları veri teypleri gibi alternatif veri saklama araçlarında korumak ve dosyanın aslının zarar görmesi durumunda dosyaları geri kazanmak amacıyla kullanılmışlardır. Arşiv dosyaları günümüzde genel olarak dosya transferi ile dosyaların sıkıştırılarak saklanmasında kullanılır.

3.3. Arşiv Formatı Türleri

- Dosyaları sadece arşivlemekte kullanılan dosya formatları: Birden fazla dosyayı tek bir dosya şeklinde birleştirmekte kullanılırlar.
- Dosyaları sadece sıkıştırmakta kullanılan dosya formatları: Tek bir dosyayı sıkıştırmakta kullanılırlar.
- Çok fonksiyonlu formatlar: Bu formatlar dosyaları birleştirme, sıkıştırma ve hata yakalama ve düzeltme özelliklerini desteklerler.

3.4. Projede Kullanılan Arşivleme Formatları

3.4.1. BZIP2

BZIP2**,** bir kayıpsız veri sıkıştırma formatıdır ve Julian Seward tarafından yaratılmıştır. BZIP2'nin ilk sürümü Temmuz 1996'da yayınlandı. Formatın popülaritesi ve kalıcılığı ileriki yıllarda artarak, 2000 yılının sonlarında son versiyonu yayınlandı**.** BZIP2 arşiv formatı ".bz2" dosya uzantısını kullanır.

BZIP2 pek çok dosyayı eski LZW ve ZIP sıkıştırma algoritmalarından daha etkin sıkıştırır, ama hız olarak bu algoritmalara göre 12 kat daha yavaştır.

GZIP gibi BZIP2 arşiv formatı sadece verileri sıkıştırır. RAR veya ZIP gibi aynı zamanda dosyaları arşivlemezler. Ama istenilmesi halinde, UNIX genelinde TAR gibi harici formatlar aracılığıyla arşivleme işlemini de gerçekleştirebilirler.

BZIP2, BWT algoritmasını sık sık tekrarlanan karakter kümelerinin tanımlayıcı kelimeler dizinine dönüştürülmesinde kullanır. MTF dönüşümünü de kullanır. Sıkıştırma işleminin en son adımını da Huffman kodlamasıyla bitirir. Sıkıştırılmamış veri girdi bloklarının hepsi 100 ve 900 kilobyte aralığından seçilebilir ve tüm bloklar eşit büyüklüktedir. Sıkıştırma blokları 48-bit iki sihirli (magic) sayıyla sınırlandırılırlar. Sihirli sayı olarak blok başlangıcında π'yi temsil eden ikili kodlanmış onluk sayı 0x314159265359 ve blok sonunda π'nin karekökünü temsil eden 0x177245385090 kullanılır.

BZIP2'nin atası BZIP blok sıralamadan sonra aritmetik kodlama kullanırdı. Yazılım patenti yüzünden bu algoritmadan vazgeçilerek, şu an da BZIP2 de kullanılan Huffman kodlamasına geçildi.

BZIP2 sıkıştırmada işlemini biraz yavaş gerçekleştirmesiyle bilinir**.** Kullanıcılar için zamanın önemli olduğu durumlarda GZIP gibi alternatif formatları kullanmaya iter. Bu yavaşlık sorunu asimetriktir. Yani açma işlemi daha hızlı bir şekilde gerçekleşir. Sıkıştırma için fazla miktarda CPU zamanı gerektiğinden, 2003 yılında çoklu-thread yapısını destekleyen geliştirilmiş bir sürümü yaratıldı. Bu format çoklu CPU ve çok

çekirdekli bilgisayarlarda büyük hız artışı sağladı. Ama bu özellik ana projeye henüz dahil edilmedi.

BZIP2, sıkıştırma tekniklerinin birbiri üstüne yığılmış birçok katmanını kullanır. Sıkıştırmada şu sırayı takip eder:

1) RLE katmanı:

4 ‘den 255 ‘e kadar tekrarlanan sembollerden oluşan herhangi bir parça ilk dört sembol ve 0 ile 251 arasında bir tekrar uzunluğuyla değiştirilir. Yani örnek olarak "AAAAAAABBBBCCCD" parçası şu hale getirilir: "AAAA\3BBBB\0CCCD". Sembollerin devamı işlemi geri döndürülebilir kılmak için devam uzunluğu sıfıra atansa bile her peş peşe dört sembolden sonra değiştirilir.

2) BWT katmanı:

Bu adım BZIP2 ‘nin temelinde bulunan geri dönüştürülebilir blok sıralama metodudur. Blok sıralama için, bir matris oluşturulur, i satırları tüm tamponu tutar. Matrisin satırları alfabetik olarak sıralanır. Bir 24-bit işaretçi, bloğun dönüşüm olmamış başlangıç pozisyonu için kullanılır. Pratikte, tüm matrisi işlemek gerekli değildir, yerine sıralama tampondaki her pozisyon için işaretçiler kullanarak yapılır. Çıkış tamponu matristeki son sütundur; bu tüm tamponu tutar.

3) MTF katmanı:

Yine, bu dönüşüm işlenmiş bloğun büyüklüğünü değiştirmez. Veride kullanılan sembollerin her biri bir katarın içine yerleştirilir. Bir sembol işlendiğinde, katarda işlenen sembolün konumuyla değiştirilir ve sembol katarın ucuna konur. Sonucu aniden tekrar eden semboller sıfır sembolleriyle değiştirilerek, diğer semboller yerel frekansına göre yeniden haritalanırlar.

4) 2. RLE katmanı:

Çıktıdaki uzun tekrarlanan sembol dizileri (şu anda normal olarak sıfırlar) sembol ve iki özel kodun, RUNA ve RUNB, bir birleşimi ile değiştirilir. İkili sayı olarak birden büyük tekrar uzunluğunu temsil eder. 0, 0, 0, 0, 0, 1 parçası 0, RUNB, RUNA, 1 olarak temsil edilir; RUNB ve RUNA onluk olarak 4 değerini temsil eder. RLE kodu başka bir

normal sembole erişilerek sonlandırılır. Bu RLE işlemi ilk adımdaki RLE işleminden daha karmaşıktır. Bu işlem uzun dizileri daha iyi kodlar (pratikte bu değer blok büyüklüğü ile sınırlanmıştır. Yani bu adım 900 kilobyte değerinden daha fazla işlemi kodlayamaz). Daha karışık bir örnek:

Tablo 3.1 BZIP2 algoritması ikinci RLE katmanı

RUNA	RUNB	RUNA	RUNA	RUNB	(ABAAB)
1	2	4	8	16	=31

5) Huffman kodlaması katmanı:

Bu işlem sabit büyüklükteki sembolleri (8-bit) kullanım sıklığına bağlı olarak farklı büyüklükte kod değerleriyle değiştirir. Çok tekrar eden kodlar daha az yer kaplarken (2-3 bit) , nadir kullanılan kodlar 20 bite kadar yer kaplayabilirler. Bu kodlar birbirleriyle karışmasın diye çok dikkatli seçilirler. Sıkıştırılmamış veride n farklı sembol kullanılmışsa, o zaman Huffman kodu iki RLE kodu (RUNA ve RUNB), n-1 sembol kodu ve bir veri sonu kodu barındırır. Uç örneklerde, sıkıştırılmamış veride sadece bir sembol kullanıldığı yerde, Huffman ağacında hiçbir sembol kodu olmaz ve tüm blok RUNA ve RUNB ve 2 değerinde bir veri sonu izinden oluşur.

- 0:RUNA
- 1:RUNB
- 2-257: 0-255 byte değerleri
- 258: veri sonu, işlemi bitir. (En az “2” değeri olabilir)

5) Çoklu Huffman tabloları katmanı:

Özdeş büyüklükte birkaç Huffman tablosu, bir blok ile kullanılabilir. Ancak bunları kullanmanın getirisi, fazladan tablo ekleme maliyet kazancından büyük olması gerekir. En az iki ve en fazla altı tablo olabilir, her 50 sembol işlenmeden önce en kazançlı tablo yeniden seçilebilir. Bu devamlı yeni kaynak tablosu ekleme yapmadan çok uyumlu Huffman dinamiklerine sahip olmanın avantajına sahiptir.

6) Birlik tabanda 1 kodlama katmanı:

Eğer çoklu Huffman tabloları kullanılmış ise, her tablonun seçimi bir ile altı bit arasında uzunluktaki bir listeden olur. Seçim, tabloların bir MTF listesinin içindedir. Bu

özelliğin kullanımı en fazla 1.015 civarında bir artışa neden olur, ama genellikle daha azdır. Bu ekleme tekrarlı Huffman tablolarının kullanımı ile kapatılır ve aynı Huffman tablosunun kullanımına devam etme durumu tek bir bit ile gösterilir.

7) Delta kodlama(Δ) katmanı:

Huffman kod bit büyüklükleri her kullanılan konik Huffman tablosunun yeniden yapımı için gereklidir. Her bit büyüklüğü önceki kodun bit büyüklüğünün farkı olarak kodlanır. Bir sıfır (0) biti, önceki bit büyüklüğü şimdiki kod içinde kullanılacak demek olur. Bir (1) biti ise, önceki bit okunacak ve bit büyüklüğü artma ya da azalması o değer üzerinden olacak demek olur. Genel durumda her bir tablonun her bir sembolü için tek bir bit kullanılır ve en kötü durumda- büyüklük 1 den 20 ye giden- toplamda 37 bit gereklidir. Daha önceki MTF kodlamasının sonucu olarak, kod büyüklükleri 2-3 bit (çok sıkça kullanılan kodlar) den başlar ve giderek artar. Yani delta formatı yeterince etkindir- her tam Huffman tablosu için ortalama 38 byte a ihtiyaç duyar.

8) Seyrek bit dizisi katmanı:

Blok içinde hangi sembollerin kullanıldığını ve Huffman ağacına ekleneceğini göstermesi için bir bit haritası kullanılır. İkili veri bir byte ile gösterilebilinecek tüm 256 sembolün kullanımı gibidir, ancak kodlanmamış veri sadece mevcut değerlerin küçük bir alt kümesini kullanabilir, örneğin 32 ve 126 arasındaki ASCII aralığı gibi. Eğer çoğunlukla kullanılmıyorsa 256 bit sıfır tutmak gereksiz olacaktır. Seyreklik metodunda, 256 sembol 16 kısma ayrılır ve semboller sadece eğer blok içinde kullanılmışsa 16-bit diziye eklenir. Bu 16 kısmın her biri başlangıca eklenen 16 bit dizi ile temsil edilir.

Bir BZIP2 verisi, 4-byte başlık, devamında sıfır veya daha fazla sıkıştırılmış blok, ardından işlenen tüm veri için hata kontrolünü yapan 32-bit CRC veri sonu izi içerir. Sıkıştırılmış bloklar bit tabanlıdır ve doldurma olmaz.

Tablo 3.2. BZIP2 dosya yapısı (WEB_3)

Sihirli Bit: 16 bit	Dosyanın bzip2 formatında sıkıştırıldığını gösteren “BZ” karakterleri
Sürüm: 8 bit	Bzip2 için ‘h’, bzip1 için ‘0’
Kullanılan blok boyutu: 8 bit	100 kb ile 900 kb arasında bir değer ifade eden 1-9 arası bir sayı
Sıkıştırılmış veriye ait sihirli bit: 48 bit	0x314159265359 (pi)
Crc: 32 bit	Bu blok için kontrol toplamı
Rastgelelenmiş: 1 bit	0=>normal, 1=>rastgelelenmiş
OrigPtr: 24 bit	BWT dönüştürme için başlangıç pointerı
Huffman da kullanılan harita sayısı: 16 bit	16 bytelık bitmaplerin sayısı. Mevcut ya da mevcut değil.
Huffman da kullanılan bitmaplerin sayısı: 0-256 bit	Kullanılan sembollerin bit haritası.
Huffman grupları: 3 bit	2-6 farklı Huffman tablosu kullanılır.
Kullanılan seçiciler: 15 bit	Huffman tablolarının kaç kez değiştirildiği
Seçici liste: 1-6 bit	MTF lenmis Huffman tabloların sıfır sonlandırmalı bit uzunluğu (0-62)
Başlangıç Huffman uzunluğu: 5 bit	0-20 Huffman deltaları için başlangıç bit uzunluğu
Delta bit uzunluğu: 1-40 bit	0=>sonraki sembol; 1=>uzunluğu güncelle
İçerik: 2-∞ bit	Huffman kodlanmış veri, blok sonuna kadar
Dosya sonu sihirli biti: 48 bit	0x177245385090 karekök (pi)
CRC: 32 bit	Tüm veri için kontrol toplamı
Doldurma: 0-7 bit	Tam byte hizalama

3.4.2. GZIP

GZIP, Jean-Loup Gilly ve Mark Adler tarafından yaratılmış bir formattır. GZIP 0.1 versiyonu 1992 yılında, 1.0 versiyonu da 1993 de yayınlanmıştır. GZIP arşiv formatı ".gz" dosya uzantısını kullanır.

GZIP, LZ77 ve Huffman kodlamasının bir kombinasyonu olan Deflate algoritmasına dayanır. GZIP dosya formatı aşağıdaki verileri içerir:

- Dosyanın GZIP formatıyla sıkıştırıldığını belirtmekte kullanılan sihirli numara (magic number) verisi ile sürüm ve zaman bilgisini içeren 10 byte uzunluğundaki bir başlık verisi.
- Opsiyonel ekstra başlık alanları. (Örneğin, orijinal dosya adını belirmekte kullanılmak üzere.)
- Deflate algoritması ile sıkıştırılmış bir gövde.
- 8-byte büyüklüğünde, CRC-32 hata kontrol verisini ve orijinal sıkıştırılmamış veri uzunluğunu tutan bir alt başlık.

Tablo 3.3 GZIP dosya yapısı (WEB_2)

ID1	ID2	CM	FLG	MTIME	XFL	OS	Sıkıştırılmış veri	CRC32	ISIZE

GZIP ile sıkıştırılan her bir dosya yukarıdaki dosya yapısına sahiptir. Bu yapı içerisinde yer alan terimler aşağıda açıklanmıştır.

- ID1 ve ID2 alanları dosyanın GZIP formatı ile sıkıştırıldığını gösteren başlık alanlarıdır. Bu alanlardan ID1 ondalık olarak 49 (0x31), ID2 de yine ondalık olarak 139 (0x8b) değerini alır.
- CM alanı GZIP dosyasının sıkıştırılmasında kullanılan sıkıştırma metodunu gösterir. Bu alanın değeri genelde, Deflate sıkıştırma algoritmasının kullanıldığını gösteren 8 değeri olur.

- FLG alanı dosya içerisinde yer alan ekstra alanları belirlemekte kullanılır. Olması planlanan ekstra alanlar OS alanı ile sıkıştırılmış verinin bulunduğu alan arasında yer alır.

- MTIME alanı sıkıştırılmış veri üzerinde en son ne zaman değişiklik yapıldığını gösterir.

- XFL alanı sıkıştırma metodu ile ilgili ekstra özellikleri belirlemekte kullanılır. Deflate sıkıştırma algoritması için bu alan şu şekildedir:

    - XFL = 2 ise sıkıştırıcı maksimum sıkıştırmayı gerçekleştirir, fakat yavaş olarak çalışır.

    - XFL = 4 ise sıkıştırıcı en hızlı sıkıştırma algoritmasını kullanır.

- OS alanı sıkıştırma işleminin gerçekleştirildiği dosya sistemini tutar. Bu alan sıkıştırılmış yazı dosyalarının sonundaki EOF karakterini bulmaya yardımcı olur.

- Sıkıştırılmış veri alanı, adından da anlaşılacağı üzere sıkıştırılmış olan verinin bulunduğu bölümdür.

- CRC32 alanı, sıkıştırılmamış veriye ait, daha sonra hata kontrolünde kullanılmak üzere hesaplanan CRC32 değerinin yazıldığı bölümdür.

- ISIZE alanı, verinin sıkıştırılmamış halinin boyutunun 2 ‘nin kuvvetleri tutulduğu bölümdür.

Dosya formatı birden fazla veriyi sıralamaya izin vermesine rağmen, BZIP2 sıkıştırma formatında olduğu gibi, sadece tek dosyayı sıkıştırmak için kullanılır. Eğer dosya gruplarından oluşan veriler sıkıştırılmak isteniyorsa, bu dosya grupları tek bir tar arşivine toplanıp, daha sonra GZIP arşiv formatıyla sıkıştırılması gerçekleştirilir. Bu işlem sonucunda “.tar.gz” veya “.tgz” dosya uzantılarından birine sahip tarball olarak adlandırılan dosyalar oluşturulur.

GZIP, yine Deflate algoritmasını kullanan ZIP formatı ile karıştırılmamalıdır. ZIP formatı çoklu dosyaları herhangi bir harici arşivleme işlemine gerek duymadan sıkıştırabilen bir formattır. Ancak ZIP formatı, genelde tarball kullanan GZIP

formatından daha kötü sıkıştırma yapar. Bunun nedeni GZIP'in kullandığı tarballın aksine dosyaları ayrı ayrı sıkıştırması ve dosyalar arası artıklığın avantajını kullanamamasıdır.

3.4.3. TAR

TAR arşiv formatı, UNIX işletim sisteminin ilk günlerinde yaratıldı ve POSIX.1-1988 ve PSIX.1-2001 tarafından standart haline geldi. Başlangıçta teyp yedekleme amacıyla ardışık G/Ç araçlarının direk yazılabilmesi için tasarlandı. Şu anda genel olarak kullanıcı ve grup izinleri, tarih ve klasör yapısı gibi dosya sistemi bilgilerini koruyarak dağıtım veya arşivleme için pek çok dosyayı tek bir dosya içinde toplamak için kullanılıyor.

Birden fazla dosyayı aynı anda sıkıştırma desteği olmayan pek çok sıkıştırma formatı TAR arşivini kullanarak sıkıştırma işlemini gerçekleştirirler. Bu formatlar arasında en bilindik olanlar GZIP, BZIP2 ve LZMA'dır.

TAR formatı kullanarak arşivlenmiş dosyalar ".tar" dosya uzantısını kullanır. Eğer TAR formatı yukarıda bahsedilen sıkıştırma formatları ile birlikte kullanılmışsa aşağıdaki dosya uzantılardan birisini kullanılır.

- ".tgz" veya ".tar.gz".
- ".tbz", ".tbz2", ".tb2" veya ".tar.bz2".
- ".tar.lzma".

TAR arşivi içinde yer alan her bir dosya 512 byte boyutundaki başlık kaydı ardından gelir. Dosya verileri değiştirilmeden yazılır. Yalnız sıkıştırılan dosyanın boyutu 512 'nin katlarına yuvarlanır ve fazladan boşluk sıfırlarla doldurulur. Bir arşivin sonu içeriği sıfır ile doldurulmuş kayıt ile gösterilir.

Pek çok tarihi teyp sürücüleri değişken uzunlukta veri bloklarını okur ve yazar, bloklar arasında teypte anlamlı boşluklar bırakır (teyp fiziksel olarak harekete başlasın ve dursun diye). Bazı teyp sürücüleri ve ham diskler sadece sabit uzunluklu veri bloklarını destekler. Ayrıca dosya sistemi veya ağ gibi bir ortama yazarken tek bir büyük bloğu yazmak pek çok küçük bloğu yazmaktan daha az zaman alır. Bu yüzden

TAR veriyi 512 byte boyutundaki bloklar halinde yazar. Kullanıcı blok başına düşen kayıt sayısını gösteren, normal değeri 20 olan ve 10 kilobyte büyüklüğünde bloklar üreten blok faktörünü özelleştirebilir.

Dosya başlık bloğu, dosyayla ilgili bilgileri tutar. Farklı byte düzenine sahip yapılarda taşınabilirliği garanti altına almak için başlık bloğundaki bilgiler ASCII olarak kodlanmışlardır.

Orijinal UNIX TAR formatı tarafından tanımlanan alanlar aşağıdaki tabloda gösterilmiştir. Bağ gösterici/dosya tipi tablosu bazı modern uzantıları içerir. Bir alan kullanılmadığı zaman boş byte değerleri ile doldurulur. Başlık boyutu eğer 512 byte dan küçük ise 512 byte a tamamlanmak için boş byte değerleri ile doldurulur.

Tablo 3.4 Orijinal UNIX TAR dosya yapısı (WEB_4)

Alanın Konumu	**Alan Boyutu**	**Alan**
0	100	Dosya Adı
100	8	Dosya Modu
108	8	Kullanıcının Sayısal ID
116	8	Grubun Sayısal ID
124	12	Byte Olarak Dosya Boyutu
136	12	Sayısal Unix Zaman Formatında Son Güncelleme Zamanı
148	8	Başlık Bloğu İçin Kontrol Toplamı
156	1	Bağ Gösterici (Dosya Tipi)
157	100	Bağlanmış Dosyanın Adı

Tablo 3.5 TAR dosya yapısına ait bağ gösterici alanı. (WEB_4)

Değer	Anlamı
'0'	Normal Dosya
(ASCII NUL)	Normal Dosya (şu an kullanılmıyor)
'1'	Kuvvetli Bağ
'2'	Sembolik Bağ
'3'	Karakter Özel
'4'	Blok Özel
'5'	Dizin
'6'	FIFO

Günümüzde kullanılan TAR programlarının birçoğu, POSIX(IEEE P1003.1) standartları grubu tarafından tanımlanmış, genişletilmiş başlık tanımlarına sahip yeni USTAR formatında arşivleri yazar ve okur. Eski programlar fazladan bilgiyi görmezden gelirken, yeni programlar "ustar" katarının varlığını kontrol ederek, yeni formatın kullanılıp kullanılmadığına bakar. USTAR formatı daha uzun dosya isimlerine ve her dosya için fazladan bilgi depolamaya izin verir.

Tablo 3.6 USTAR UNIX TAR dosya yapısı (WEB_4)

Alan Konumu	Alan Boyutu	Alan
0	156	Eski formattaki gibi
156	1	Tip Bayrağı
157	100	Eski formattaki gibi
257	6	USTAR Belirteci "ustar"
263	2	USTAR Versiyonu "00"
265	32	Kullanıcı Adı
297	32	Grup Adı
329	8	Aygıt Ana Sayısı
337	8	Aygıt İkincil Sayısı
345	155	Dosya Adı Öntakı

3.4.4. ZIP

ZIP dosya formatı bir veri sıkıştırma ve arşivleme formatıdır. Bir ya da birden fazla sıkıştırılmış dosya içeren bir ZIP dosyası, dosyanın boyutunu düşürmekte kullanılır. ZIP dosya formatı birden fazla sıkıştırma algoritmasına izin vermesine rağmen 2009 yılı itibariyle sadece Deflate algoritması yaygın olarak kullanılmakta ve desteklenmektedir.

Orijinal dosya formatı 1989 yılında Phil Katz tarafından daha eski bir kayıpsız sıkıştırma formatı olan ARC den geliştirilerek, PKZIP[1] adlı arşivleme programı için yaratıldı.

ZIP dosya formatı yaratıldığı zamandan itibaren birçok arşiv uygulaması ve birçok işletim sistemi tarafından desteklenmektedir. Microsoft şirketi 1998 yılından sonraki bütün Windows işletim sistemlerinde, Apple şirketi de Mac OS X 10.3 den sonraki bütün işletim sistemlerinde yerleşik ZIP desteği sunmaktadır.

ZIP dosyaları genellikle ".zip" veya ".ZIP" dosya uzantılarını kullanır. ZIP dosya formatı ayrıca pek çok program tarafından farklı isimlerle de kullanılmaktadır. Bu kullanımlara Java RAR (JAR) formatı, Mozilla Firefox eklentilerinin uzantıları olan ".xpi" formatı, Winamp ve Windows Media Player arayüz dosyalarının kullandıkları formatlar, OpenOffice dosya formatlarından olan ".odt" formatı örnek olarak gösterilebilir.

ZIP her dosyayı ayrı ayrı sıkıştıran basit bir arşiv formatıdır. Bu da teoride farklı dosyalar için farklı sıkıştırma algoritmaları kullanarak daha iyi sıkıştırma yapmayı sağlayabilir. Ayrıca dosyaların ayrı ayrı sıkıştırılması, sıkıştırılmış dosya içerisindeki diğer dosyaları okumaya gerek duymadan, istenilen dosyanın elde edilmesini sağlar.

ZIP formatının özelliklerden birisi de dosyaların ister sıkıştırılmadan, isterse de farklı sıkıştırma algoritmaları kullanarak saklanabilmesidir. Sıkıştırma algoritması olarak genellikle Katz'ın Deflate algoritması kullanılır. Fakat sıkıştırılmak istenen dosyalar zaten sıkıştırılmış veya sıkıştırılmaya dirençli ise dosya verisi sıkıştırılmadan saklanır.

[1] http://www.pkware.com/

ZIP, ciddi derecede kusurlu olduğu bilinen basit bir şifre tabanlı simetrik şifreleme sistemini destekler. Bu sistem, known – plaintext adı verilen saldırılara karşı zayıftır. Known – plaintext saldırılarında saldırganın elinde şifrelenmemiş veriye (plaintext) ve bu verinin şifrelenmiş haline (ciphertext) ait örnekler bulunur ve saldırgan bu örnekleri şifrelenmiş veriye ait anahtarları ve kod bloklarını elde etmekte kullanır. ZIP dosya formatının 5.2 versiyonundan itibaren gelen özellikleri arasında yeni sıkıştırma metotlarının yanı sıra yeni şifreleme (AES) metotları da bulunmaktadır.

ZIP formatının özelliklerinden bir tanesi de arşiv dosyalarını bölerek birden fazla dosya halinde saklamaya olanak tanımasıdır. Başlarda büyük boyutlu ZIP dosyalarını birden fazla 1.44 megabyte lık disketlerde saklayabilmek üzere geliştirilen bu özellik, şimdilerde ZIP dosyalarını parçalar halinde e-mail olarak göndermekte ya da taşınabilir medyalarda depolamakta kullanılmaktadır.

ZIP dosya formatı, rastgele bir şekilde depolanmış dosya ve klasörler içerir. ZIP dosyası içerisinde yer alan dosyaların konumu ZIP dosyasının sonunda yer alan merkezi dizin (central directory) kısmında yer alır. ZIP dosyası içerisindeki dosyalar ve dizinler dosya girdileri ile ifade edilir.

Her bir dosya girdisi bir yerel başlık (local header) ve bu yerel başlık bilgisinin yanında yer alan yorum, dosya boyutu ve dosya adı bilgileriyle ifade edilir. Yerel başlık bilgisini opsiyonel olmak üzere ekstra veri alanları izleyebilir. Son olarak bu alanları sıkıştırılmış ya da hem sıkıştırılıp hem de şifrelenmiş olan verinin bulunduğu alan izler.

Ekstra veri alanları ZIP formatının esnekliğini ve genişletilebilirliğini sağlayan anahtarlardır. Teoride birçok yeni eklentinin bu ekstra veri alanları sayesinde formata dahil edilmesi öngörülmektedir.

Merkezi dizin, sıkıştırılmış veri içerisinde yer alan her dosya girdisi için dosya adının, yerel başlıkların bağıl konum (relative offset) bilgilerinin yer aldığı dosya başlıklarını (file header) içerir.

Merkezi dizin içerisindeki her dosya girdisi, her girdi için farklı olacak şekilde 4 byte boyutundaki bir imza ile işaretlenir. ZIP dosya ayrıştırıcıları genellikle bir ZIP dosyasını incelerken tahsis edilen imzalara bakar.

Merkezi dizindeki dosya girdilerinin sırası ile arşivdeki dosya girdilerinin sırasının birbirleriyle aynı olması gerekmez. Çünkü format sıralı değildir.

ZIP dosya formatında arşiv sonunu gösteren EOF (End of File) izleri yoktur. Bunun yerine her bir alana ait olan imza değerine bakılır.

Tablo 3.7 ZIP dosya yapısı (WEB_1)

Yerel dosya başlığı 1
Dosya verisi 1
Veri tanımlayıcısı 1
. . .
Yerel dosya başlığı n
Dosya verisi n
Veri tanımlayıcısı n
Arşiv deşifreleme başlığı
Arşiv ekstra veri kaydı
Merkezi dizin
Merkezi dizin sonu

3.4.5. LZMA

LZMA (Lempel-Ziv-Markov chain-Algorithm) veri sıkıştırma işlemi için kullanılan bir formattır. 1998 den bu yana geliştirilmekte olan algoritma, 7z sıkıştırma formatının da temelini oluşturmaktadır.

LZMA formatı GZIP ve BZIP2 sıkıştırma formatlarında olduğu gibi, sadece dizin halinde olmayan tek bir dosyayı sıkıştırmakta kullanılır. İçerisinde birden fazla dosya bulunan dizin şeklindeki dosyalar sıkıştırılmak isteniyorsa TAR arşiv formatı ile birlikte kullanılarak sıkıştırma işlemi gerçekleştirilir.

Küçük kod boyutu, düşük miktarda RAM kullanımı, yüksek sıkıştırma oranı ve açık kaynak kodlu olması LZMA formatının, gömülü uygulamalarda yaygın olarak kullanılmasına yol açar.

LZMA algoritması, LZ77 ye benzer bir sözlük sıkıştırma şeması kullanır. Ayrıca yüksek bir sıkıştırma oranı (genelde BZIP2 den daha fazla) ve değişken bir sözlük boyutu (4 GB e kadar) gibi avantajlar sağlar.

LZMA algoritma olarak, bir aralık kodlama (range encoding) ile desteklenmiş, gelişmiş bir LZ77 algoritması ve bir DEFLATE sıkıştırma algoritması kullanır.

LZMA, Deflate sıkıştırma algoritmasını, GZIP ve ZIP formatlarından daha etkili bir şekilde kullanır. Normal Deflate algoritmasının 32 KB olan sözlük boyutu LZMA algoritmasında 32 MB olarak kullanılır. Ayrıca LZMA tarafından kullanılan Deflate algoritmasında, LZ77 algoritması tarafından oluşturulan geri dönüş referansları bulunduktan sonra, elde edilen kaynak veri Huffman algoritması yerine aritmetik kodlama kullanılarak sıkıştırılır.

LZMA ile sıkıştırılmış bir dosyaya ait dosya yapısı aşağıda gösterilmiştir. Gösterilen alanlardan bazıları da kendi içlerinde alt alanlara ayrılmaktadır. LZMA dosya formatı içerisinde bulunan alanlar aşağıda açıklanmıştır.

Tablo 3.8 LZMA dosya yapısı (WEB_5)

Başlık Sihirli Byteları
Veri Bayrakları
CRC32
Blok başlığı
Sıkıştırılmış veri
Blok alt başlığı

- LZMA verisinin ilk 6 bytelık kısmına başlık sihirli byteları (header magic bytes) adı verilir. Bu veri dosyanın bir LZMA dosyası olduğunu belirlemekte kullanılır. Bu byte değerleri sırasıyla 0xFF, ‘L’, ‘Z’, ’M’, ’A’, 0x00 şeklindedir.
- Veri bayrakları alanı, sıkıştırılmış veri ile ilgili bazı özelliklerin belirlemekte kullanılan veri alanıdır.
- CRC32 alanı sıkıştırılmış veri hakkında hata kontrolü bilgisinin tutulduğu alandır. 32 bit büyüklüğündedir. Sıkıştırılmış verinin açılması esnasında hesaplanan CRC32 değeri dosyada önceden kayıtlı olan CRC32 değeri ile aynı değilse veri hatalı demektir.

- Blok başlığı alanı, blok bayrakları, sıkıştırılmış boyut, sıkıştırılmamış boyut, filtre bayraklarının listesi, CRC32 ve başlık doldurma bölümlerinden oluşur.

    - Blok bayrakları bölümü; kullanılan filtre bayraklarının sayısı, sıkıştırılmış boyut ve sıkıştırılmamış boyut alanlarının mevcut olup olmadığı bilgisi, başlık doldurma verisinin boyutu gibi başlık verisi ile ilgili özelliklerin tutulduğu bölümdür.
    - Sıkıştırılmış boyut ve sıkıştırılmamış boyut bölümleri, isimlerinde de anlaşılacağı üzere veriye ait sıkıştırılmış ve sıkıştırılmamış dosya boyut büyüklerinin yazıldığı bölümdür.
    - CRC32 bölümü başlık ile ilgili CRC32 değerini tutarak, eğer başlık verisinde herhangi bir hata oluşursa bunu saptamakta kullanılır.

- Blok alt başlığı bölümü isminden de anlaşılacağı üzere veri bloğunun sonlandığı bölümdür. Kontrol, veri alt başlığı ve alt başlık doldurma bölümlerinden oluşur.

    - Kontrol alanı CRC32 gibi hata kontrolü yapmakta kullanılır. Sıkıştırılmamış veriden hesaplanan bu kontrol verisi kaydedilmiş veri ile uyuşmazsa kod çözücü hata verir.
    - Veri alt başlığı alanı kendi içerisinde sıkıştırılmamış boyut, tersine boyut, veri bayrakları ve alt başlık sihirli byte değerlerinden oluşur.

        - Sıkıştırılmamış boyut alanı, blok başlığında yer alan sıkıştırılmamış boyut alanının bulunmaması durumunda mevcuttur ve verinin sıkıştırılmamış boyutunu tutar.
        - Tersine boyut alanı; blok başlığı, sıkıştırılmış veri, kontrol ve sıkıştırılmamış boyut alanlarının toplam boyutunu tutar.
        - Veri bayrakları alanı, verinin başında yer alan başlık sihirli byteları kısmından hemen sonra gelen veri bayrakları kısmının bir kopyasıdır.
        - Alt başlık alanı dosya yapısının sonunu gösterir. Tıpkı başlık sihirli byte larında olduğu gibi veri sonunda da dosyanın bir LZMA dosyası olduğunu gösteren bir tanımlayıcı bulunur. 2 byte büyüklüğündeki bu sihirli byte değerleri sırasıyla ‘Y’ ve ‘Z’ dir.

- Alt başlık doldurma bölümü, blok veya verinin boyutunun bir sayının katları, 512 gibi, olması istendiği durumlarda verinin sonuna boş byte değerleri eklemekte kullanılır.

4. VERİ ŞİFRELEME

4.1. Genel Bilgi

Veri şifreleme, başkaları tarafından erişilmesi istenmeyen verilerin, çeşitli veri şifreleme algoritmaları aracılığıyla şifrelenmesi işlemidir. Veri şifreleme matematik ve bilgisayar bilimlerinin bir dalı olarak ele alınmaktadır. Günümüzde veri şifreleme, ATM kartlarının güvenliği, bilgisayar şifreleri, elektronik alışveriş gibi veri güvenliğinin çok önemli olduğu alanlarda kullanılmaktadır.

Veri şifreleme algoritmalarında amaç, şifresiz metinleri (plaintext) belirli algoritmalar aracılığıyla şifrelenmiş verilere (ciphertext) dönüştürmektir. Şifreleme işlemi şifreleme algoritması ve bir anahtar ile kontrol edilir. Anahtar adı verilen bu gizli parametreler verinin içeriğini şifrelemekte kullanılır. Eğer değişken değerli bu anahtarlar olmasaydı, şifreleme işlemi birçok amaç için uygunluğunu yitirir ve kolay bir şekilde kırılabilir bir hale gelirdi.

4.2. Veri Şifrelemenin Projede Kullanım Amacı

Veri şifreleme işlemi daha önce de bahsedildiği üzere verileri şifreleyerek, istenmeyen kişiler tarafından erişimini engellemekte kullanılır. Buna amaca paralel olarak, veri şifreleme, proje içerisinde sıkıştırılmış verileri şifreleyerek istenmeyen kişiler tarafından erişilmesini engellemek amacıyla kullanılmıştır.

ZIP dosya formatı daha önceki bölümlerde bahsedildiği üzere kendi dosya formatı içerisinde şifreleme desteğine sahiptir. Ancak proje içerisinde kullanılan ZIP formatı haricindeki diğer formatlardan hiçbiri şifreleme desteğine sahip değildir. Bu yüzden proje kapsamında geliştirilen program tarafından desteklenen bütün formatlar kullanılarak sıkıştırılmış verileri şifrelemekte kullanılmak üzere, arşiv dosya biçimleri tarafından desteklenmeyen, harici bir şifreleme algoritmasının kullanılmasına karar verilmiştir. Harici bir şifreleme algoritması kullanılacağı için şifreleme kullanılarak

oluşturulan dosya arşivleri, diğer arşiv programları tarafından açılamayacaktır.

Şifreleme işlemi, verinin sıkıştırma işleminin tamamlanmasından sonra sıkıştırılmış verinin şifrelenmesi şeklinde gerçekleştirilir.

Şifreleme işlemini gerçekleştirmek üzere seçilen şifreleme yöntemi AES (Advanced Encryption Standart) algoritmasıdır. Bu algoritmanın seçilmesinin nedeni günümüzde en yaygın olarak kullanılan ve en güvenli şifreleme yöntemi olmasıdır.

4.3. AES

AES, A.B.D. yönetimi tarafından benimsenmiş bir şifreleme standardıdır. AES-128, AES-192 ve AES-256 olmak üzere 3 farklı blok şifreleme metodunu destekler. Her bir AES şifreleme metodu 128 bir blok boyutuna ve kullanılan metoda göre 128, 192 ve 256 bitlik anahtar boyutlarından birisine sahiptir.

AES, National Institute of Standards and Technology (NIST) tarafından 5 yıl süren ve içerisinde 15 tasarımın yer aldığı ve sonuç olarak Rijndael tasarımının en uygun tasarım seçildiği bir standart yarışması süreci sonucunda U.S. FIPS PUB 197 (FIPS 197) adıyla 26 Kasım 2001 de duyuruldu. Bir standart olarak 26 Mart 2002 tarihinde etkin hale geldi. AES, 2009 itibariyle simetrik anahtarlı şifreleme yöntemini kullanılan en yaygın algoritmalardan birisidir.

Rijndael şifrelemesi Belçikalı iki şifrelemeci, Joan Daemen ve Vincent Rijmen tarafından geliştirilmiştir ve AES seçme sürecine gönderilmiştir. Rijndael ismi iki yaratıcının soyadlarının birleşmesinden oluşturulmuştur.

4.3.1 Şifreleme tanımı

AES hem yazılım hem de donanım bazında hızlı, uygulaması kolay ve az miktarda belleğe ihtiyaç duyan bir algoritmadır. AES in sabit 128 bit blok boyutu ve 128, 192 veya 256 bit anahtar boyutu vardır,

Bir byte verinin 8 bite eşit olduğu farz edilerek, sabit 128 blok boyutu 128÷8=16 byte eder. AES işlemleri, durum matrisi adı verilen 4×4 boyutundaki bir dizide işlem yapar. AES hesaplamalarından çoğu özel bir sonlu alanda (finite field) gerçekleştirilir.

4.3.2. Şifreleme algoritması

AES şifreleme metodu, girdi olarak aldığı veriyi şifrelenmiş veri haline dönüştüren belli sayıdaki dönüşüm döngü basamaklarından oluşur. Biri de şifreleme anahtarıyla ilgili olan bu basamaklardan her biri belirli işlemleri içerir.

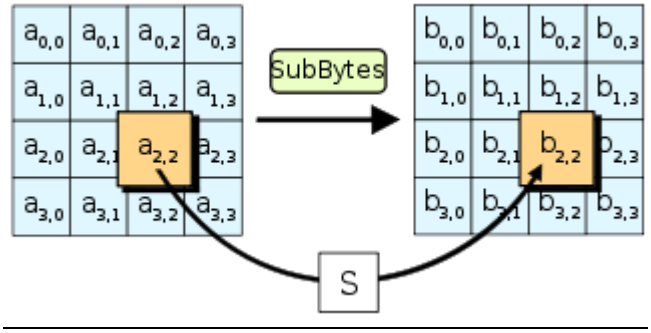
AES şifreleme algoritması şu şekilde gerçekleşir:

- Başlangıç basamağı
    - Basamak anahtarının eklenmesi
- Ara Basamaklar
    - Byte değerlerinin değiştirilmesi
    - Satırların kaydırılması
    - Sütunların karıştırılması
    - Basamak anahtarının eklenmesi
- Son Basamak
    - Byte değerlerinin değiştirilmesi
    - Satırların kaydırılması
    - Basamak anahtarının eklenmesi

1) Byte değerlerinin değiştirilmesi (SubBytes) adımı

Byte değerlerinin değiştirilmesi adımında dizi içerisinde bulunan her byte değeri, 8 bitlik bir Rijndael S-box adı verilen değiştirme kutusu (substitution box) kullanılarak değiştirilir. Bu işlem şifrelemeye non–lineerlik sağlayarak şifrenin kırılmasını zorlaştırır.

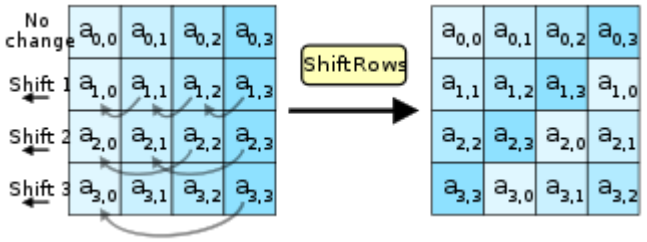
Değiştirme kutusu yöntemleri AES gibi blok şifreleme algoritmalarında anahtar ile şifrelenmiş veri arasındaki ilişkiyi belirsizleştirmekte kullanılır. AES algoritmasının kullandığı Rijndael S-box ise AES hesaplamalarının yapıldığı sonlu durum uzayının çarpımsal tersi ve affine dönüşümünün (affine transformation) birleştirilmesiyle oluşturulur. Böylelikle basit cebirsel saldırıların engellenmesi amaçlanmıştır. Burada kullanılan affine dönüşümü iki vektör uzayı arasında gerçekleştirilen bir lineer dönüşüm işlemidir.



Şekil 4.1 AES algoritması, byte değerlerinin değiştirilmesi adımı

2) Satırların kaydırılması (ShiftBytes) adımı

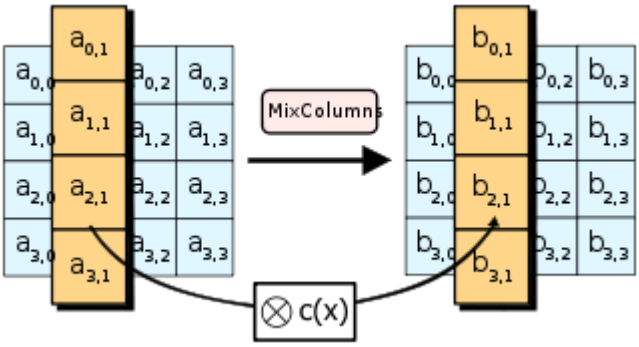
Bu adım da durum matrisinin satırları üzerinde işlem yapılır. Yapılan işlem her satırdaki byte değerlerinin belli bir konum değerine göre dairesel olarak kaydırılmasıdır. Durum matrisinin ilk satırında hiçbir değişiklik yapılmaz. İkinci satırdaki tüm byte değerleri birer byte sola kaydırılır. Üçüncü ve dördüncü satırlarda bulunan byte değerleri da benzer şekilde ikişer ve üçer byte sola kaydırılır.



Şekil 4.2 AES algoritması, satırların kaydırılması adımı

3) Sütunların karıştırılması (MixColumns) adımı

Bu adımda durum matrisinin her sütununa ait dörder byte değeri bir lineer dönüşüm fonksiyonuna sokularak yeni bir sütunun üretilmesinde kullanılır. Bu lineer dönüşüm fonksiyonu dörder byte değeri girdi olarak alarak dörder byte çıktı üretir.

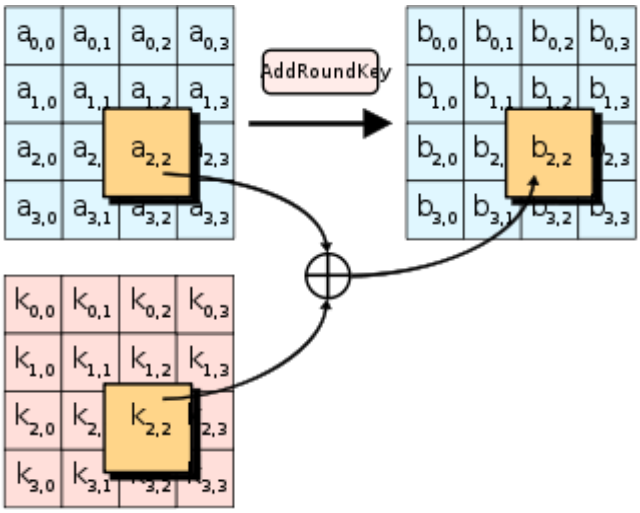


Şekil 4.3 AES algoritması, sütunların karıştırılması adımı

4) Basamak anahtarının eklenmesi (AddRoundKey) adımı

Basamak anahtarının eklenmesi adımı, alt anahtarın (subkey) durum matrisi ile birleştirilmesi şeklinde gerçekleştirilir. Her basamakta yeniden oluşturulan alt anahtar, ana anahtarın (main key) Rijndael anahtar çizelgesi (Rijndael's key schedule) algoritmasına uygulanması sonucunda elde edilir. Her alt anahtar, durum matrisi ile aynı boyutta olan bir matristir. Alt anahtar matrisinde yer alan her bir byte, durum matrisinde aynı indiste yer alan byte değeri ile bit bazında XOR işlemine tabi tutularak yeni durum matrisi elde edilir.

Rijndael'in anahtar çizelgesi, kendisine gönderilen anahtar değerini belirli işlemler sonucunda, AES şifreleme basamaklarında kullanılmak üzere alt anahtarlara dönüştüren bir algoritmadır.



Şekil 4.4 AES algoritması, basamak anahtarının eklenmesi adımı

5. GELİŞTİRİLEN UYGULAMA

Proje kapsamında geliştirilen uygulama aracılığıyla arşivleme formatları arasında bir standart oluşturulmaya çalışılmıştır. Bu amaçla program tarafından desteklenen bütün arşivleme formatlarının şifreleme ve çok parçalı dosyalar halinde arşivleme özelliklerini desteklenmesi sağlanmıştır. Bu sayede kullanıcının istediği arşivleme formatını kullanarak, şifreleme ve çok parçalı dosya seçenekleri ile arşivler oluşturması sağlanmıştır. Ancak yapılan bu şifreleme ve arşiv parçalama işlemleri dosya formatından bağımsız olarak gerçekleştirilmektedir. Yani dosya formatları bu işlemleri desteklememektedir. Bu yüzden proje kapsamında geliştirilen uygulama tarafından şifreleme ve çok parçalı dosya seçenekleri kullanılarak oluşturulan arşiv dosyaları diğer arşiv programları tarafından desteklenmez. Bu seçenekler kullanılmadan oluşturulan arşiv dosyalarında ise böyle bir sorun yoktur.

Projenin gerçekleştirilmesinde Java programlama dili kullanılmıştır. Bunun en büyük nedeni Java programlama dilinin platformdan bağımsız bir programlama dili olmasıdır. Bu sayede geliştirilen yazılımın bütün platformlarda sorunsuz bir şekilde çalışması sağlanmıştır.

5.1. Arşivleme Formatlarının Gerçekleştirilmesi

Proje kapsamında geliştirilen uygulama tarafından desteklenen arşivleme formatlarının gerçeklenmesinde Java program diline ait gömülü kütüphanelerin yanı sıra harici kütüphanelerden de faydalanılmıştır. Java gömülü kütüphaneleri ile ZIP ve GZIP arşiv formatları, Harici kütüphaneler ile de BZIP2, LZMA ve TAR formatlarının desteklenmesi sağlanmıştır.

5.1.1. BZIP2

BZIP2 formatının proje içerisindeki gerçekleştirimi Apache Software Foundation[1] lisanslı BZIP2 Java harici kütüphanesi kullanılarak sağlanmıştır. Arşiv oluşturma ve

[1] http://www.apache.org/

arşivden çıkarma işlemleri program içerisindeki BZip2 paketi içinde yer alan Compress ve Decompress sınıfları aracılığıyla gerçekleştirilmektedir.

5.1.1.1. Arşiv oluşturma

BZIP2 arşivinin oluşturulması program içerisindeki Bzip2 paketi altında yer alan Compress sınıfı içerisinde yapılmaktadır. Sıkıştırma işleminde External.org.apache.tools.bzip2 paketi içerisinde yer alan CBZip2OutputStream sınıfı ve gerek duyulması durumunda, program içerisindeki Tar paketi altında yer alan CreateArchive sınıfı kullanılmaktadır.

Verilerin sıkıştırılması işlemi sınıf içerisinde yer alan compressFile ve tarAndCompressFile metotlarının içerisinde gerçekleştirilir. Sıkıştırılması istenen dosya, tek bir dosya ise compressFile metodu kullanılır. İlk olarak sıkıştırılacak dosya BufferedInputStream sınıfı kullanılarak okunur. Verinin sıkıştırılmasında CBZip2OutputStream sınıfı kullanılır. BufferedInputStream deki bütün veriler okununcaya kadar devam eden bir döngü aracılığıyla, her seferinde BUFFERSIZE kadar veri BufferedInputStream den okunup, CBZip2OutputStreame yazılarak sıkıştırma işlemi gerçekleştirilir. Eğer sıkıştırılacak olan dosya bir dizin ise tarAndCompressFile metoduna kullanılır. Sıkıştırılacak veriyi girdi olarak alan CBZip2OutputStream sınıfına ait bir nesne oluşturulur. Son olarak da sıkıştırma işlemini gerçekleştirecek olan Tar.CreateArchive sınıfının bir nesnesi oluşturularak sıkıştırma işlemi başlatılır.

5.1.1.2 Arşiv açma

BZIP2 arşivi açma işlemi program içerisindeki Bzip2 paketi altında yer alan Decompress.java sınıfında gerçekleştirilir. Arşiv açma işleminde External.org.apache.tools.bzip2 paketi altında yer alan CBZip2InputStream ve gerek duyulması durumunda, Tar paketi altında yer alan ExtractArchive sınıflarına ait metotlar kullanılır.

Sıkıştırılmış verileri açma işlemi Decompress sınıfı içerisinde bulunan iki farklı metot ile gerçekleştirilir. Eğer ".tar.bz2" uzantılı dosya açılacak ise decompressAndUntarFile metodu kullanılır. Sıkıştırılmış olan dosyayı girdi olarak alan CBZip2InputStream sınıfına ait bir nesne oluşturulur. Sonra sıkıştırılmış dosyayı açma işlemini gerçekleştirecek olan Tar.ExtractArchive sınıfına ait bir nesne oluşturularak

dosya açma işlemi başlatılır. Açılması istenen arşiv dosyasının uzantısı ".bz2" ise decompressFile metodu kullanılır. İlk olarak açma işleminin gerçekleştirileceği dizinde aynı isimde başka bir dosyanın olup olmadığı kontrol edilir. Bu kontrol gerçekleştirildikten sonra sıkıştırılmış olan dosyayı girdi olarak alan CBZip2InputStream sınıfına ait bir nesne oluşturulur. Açılmış dosyayı yazmak için de BufferedOutputStream sınıfına ait bir nesne oluşturulur. CBZip2InputStream içerisindeki tüm veriler okununcaya kadar devam eden bir döngü aracılığıyla, her seferinde BUFFERSIZE kadar veri CBzip2InputStream den okunup, BufferedOutputStream nesnesi aracılığıyla, hedef dosyaya yazılarak dosya açma işlemi gerçekleştirilir.

5.1.2. GZIP

GZIP formatının proje içerisindeki gerçekleştirimi Java programlama dilinin gömülü kütüphanesi olan java.util.zip kullanılarak gerçekleştirilmiştir. Arşiv oluşturma ve arşivden çıkarma işlemleri, program içerisinde yer alan GZip paketi içerisindeki Compress ve Decompress sınıfları kullanılarak gerçekleştirilir.

5.1.2.1. Arşiv oluşturma

GZIP arşivinin oluşturulması program içerisinde, GZip paketi altında yer alan Compress sınıfı içerisinde yapılmaktadır. Sıkıştırma işleminde java.util.zip kütüphanesi içerisinde yer alan GZIPOutputStream sınıfı ve gerek duyulması durumunda program içerisindeki Tar paketi altında yer alan CreateArchive sınıfı kullanılmıştır.

Verilerin sıkıştırılması işlemi sınıf içerisinde yer alan compressFile ve tarAndCompressFile metotları tarafından gerçekleştirilir. Sıkıştırılacak olan dosya eğer bir dizin ise tarAndCompressFile metodu kullanılır. Sıkıştırılacak veriyi girdi olarak alan GZIPOutputStream sınıfına ait bir nesne oluşturulur. Sonra sıkıştırma işlemini gerçekleştirecek olan Tar.CreateArchive sınıfına ait bir nesne oluşturularak sıkıştırma işlemi başlatılır. Sıkıştırılması istenen dosya, tek bir dosyadan oluşuyor ise ilk olarak sıkıştırılacak dosya BufferedInputStream nesnesi tarafından okunur. Verinin sıkıştırılmasında GZIPOutputStream kullanılır. BufferedInputStream deki bütün veriler okununcaya kadar devam eden bir döngü aracılığıyla, her seferinde BUFFERSIZE kadar veri BufferedInputStream den okunup, GZIPOutputStream nesnesinin gösterdiği hedef dosyaya yazılarak sıkıştırma işlemi gerçekleştirilir.

5.1.2.2. Arşiv açma

GZIP Arşivi açma işlemi, program içerisinde yer alan GZip paketi altındaki Decompress sınıfında gerçekleştirilir. Arşiv açma işleminde java.util.zip kütüphanesinde yer alan GZIPInputStream sınıfı ve gerek duyulması durumunda program içerisinde yer alan Tar paketi altında bulunan ExtractArchive sınıfı kullanılır.

Sıkıştırılmış verileri açma işlemi iki farklı metot ile gerçekleştirilir. Eğer açılacak olan dosya ".tar.gz" uzantılı ise decompressAndUntarFile metodu kullanılır. Sıkıştırılmış olan dosyayı girdi olarak alan GZIPInputStream sınıfına ait bir nesne oluşturulur. Son olarak da sıkıştırılmış dosyayı açma işlemini gerçekleştirecek olan Tar.ExtractArchive sınıfına ait bir nesne oluşturularak dosya açma işlemi başlatılır. Açılması istenen sıkıştırılmış dosya ".gz" uzantılı ise decompressFile metodu kullanılır. İlk olarak açma işleminin gerçekleştirileceği dizinde aynı isimde başka bir dosya olup olmadığı kontrol edilir. Bu kontrol başarılı bir şekilde gerçekleştirilirse, sıkıştırılmış olan dosyayı girdi olarak alan GZIPInputStream sınıfına ait bir nesne oluşturulur. Açılmış dosyayı yazmak için de BufferedOutputStream sınıfına ait bir nesne oluşturulur. GZIPInputStream içerisindeki tüm veriler okununcaya kadar devam eden bir döngü aracılığıyla, her seferinde BUFFERSIZE kadar veri GZIPInputStream den okunup, BufferedOutputStream nesnesinin gösterdiği hedef dosyaya yazılarak, dosya açma işlemi gerçekleştirilir.

5.1.3. TAR

Proje içerisindeki TAR formatı desteği, Timothy Gerard Endres tarafından yazılmış harici bir TAR kütüphanesi kullanılarak sağlanmıştır. TAR arşivi oluşturma ve arşivden çıkarma işlemleri kaynak kod içerisinde yer alan TAR paketi altında bulunan CreateArchive ve ExtractArchive sınıflarında gerçekleştirilmektedir.

5.1.3.1. Arşiv oluşturma

TAR arşivinin oluşturulması program içerisindeki Tar paketi altında yer alan CreateArchive adlı sınıf içerisinde yapılmaktadır. Arşivleme işleminde, kaynak kod içerisinde yer alan External.publicDomain.tar paketi altında yer alan TarEntry ve TarOutputStream sınıfları kullanılmıştır.

Verilerin arşivlenmesi işlemi sınıf içerisinde yer alan CreateArchive metodunun içerisinde gerçekleştirilir. Arşivlenecek olan dosyalar archiveFile metoduna gider.

Sıkıştırılacak olan dosya bir dizin ise dosyanın içerdiği her bir alt dosya için TarEntry nesneleri oluşturarak, bu girdiler TarOutputStream nesnesinin putNextEntry metodu aracılığıyla sıkıştırılır. Sıkıştırılacak olan tek bir dosya ise bir BufferedInputStream nesnesi oluşturularak, bir döngü aracılığıyla BufferedInputStream deki bütün veriler okunarak TarOutputStream aracılığıyla arşivlenir.

5.1.3.2. Arşiv açma

TAR arşivi açma işlemi kaynak kod içerisindeki Tar paketi altında yer alan ExtractArchive sınıfında gerçekleştirilir. Arşiv açma işleminde External.publicDomain.tar paketi altında yer alan TarEntry ve TarInputStream sınıfları kullanılmıştır.

Arşiv açma işlemi sınıf içerisinde yer alan extractFile metodunun içerisinde gerçekleştirilir. Oluşturulan TarInputStream nesnesi içerisinde bulunan TarEntry nesnelerine bir döngü aracılığıyla teker teker erişilip, her bir TarEntry'nin, TarInputStream nesnesinin bir metodu olan copyEntryContents aracılığıyla bir FileOutputStream nesnesi vasıtasıyla hedef dosyaya yazılmasıyla dosya çıkarma işlemi gerçekleştirilir.

5.1.4. LZMA

LZMA formatının proje içindeki gerçekleştirimi Java programlama diline ait gömülü bir LZMA kütüphanesi olmamasından dolayı harici bir kütüphane yardımı ile sağlanmıştır. Harici kütüphane olarak LZMA algoritmasının yaratıcısı olan Igor Pavlov tarafından geliştirilen LZMA yazılım geliştirme kiti kullanılmıştır. Proje içerisinde LZMA formatında arşivleme ve arşivden çıkarma işlemleri iki farklı sınıf tarafından gerçekleştirilmektedir.

5.1.4.1. Arşiv Oluşturma

LZMA arşivinin oluşturulması proje içerisindeki Lzma paketi altında yer alan Compress sınıfı içerisinde gerçekleştirilmektedir. Verilerin sıkıştırma işlemi External.SevenZip.Compression.LZMA paketi içerisinde yer alan Encoder sınıfı tarafından yapılmaktadır.

Dosyaların sıkıştırılması sınıf içerisinde bulunan compressFile, tarAndCompressFile isimli iki farklı metot ile gerçekleştirilmektedir. Eğer arşivlenecek olan dosya dizin

değilse, yani tek bir dosya ise, compressFile, dizin ise tarAndCompresFile metodu kullanılmaktadır.

compressFile metodu Encoder sınıfını kullanarak dizin olmayan tek bir dosyayı sıkıştırmakta kullanılır. Sıkıştırma işlemi Encoder sınıfının Code methodu kullanarak gerçekleştirilir. Sıkıştırılacak olan dosyayı girdi olarak alan BufferedInputStream sınıfına ait bir nesne ve sıkıştırılmış verinin yazılacağı BufferedOutputStream sınıfına ait bir nesne oluşturularak, daha önce oluşturulmuş olan Encoder sınıfına ait nesnenin Code metoduna bu nesneler parametre olarak gönderilerek sıkıştırma işlemi gerçekleştirilir.

tarAndCompressFile metodu sıkıştırılmak istenen bir dizininin ".tar.lzma" formatında sıkıştırılmasında kullanılır. Daha öncede bahsedildiği üzere LZMA formatı dizin halinde olmayan sadece tek bir dosya sıkıştırabilmektedir. Bir dizini sıkıştırmak için ise, ilk olarak dizinin tar olarak arşivlenerek tek bir dosya haline getirilmesi, daha sonra da LZMA formatı kullanılarak sıkıştırılması gerekmektedir.

5.1.4.2. Arşiv açma

LZMA arşivlerini açmak için projede Lzma paketi içerisindeki Decompress sınıfı kullanılmaktadır. Arşiv açma işlemi external.SevenZip.Compression.LZMA paketi altında yer alan Decoder sınıfı kullanılarak gerçekleştirilmektedir.

Sıkıştırılmış dosyaları açmak için sınıf içerisinde iki adet metot tanımlanmıştır. Bu metotlardan decompressFile metodu ".lzma" uzantısına sahip dosyaları, decompressAndUntarFile metodu ise ".tar.lzma" uzantısına sahip dosyaları açmakta kullanılmaktadır.

decompressFile metodu Decoder sınıfına ait Code metodunu kullanarak sıkıştırılmış dosyaları açma işlemini gerçekleştirmektedir. Code metodu, sıkıştırılmış dosyayı okunmakta kullanılacak BufferedInputStream sınıfına ait bir nesneyi ve açılacak olan dosyayı yazmakta kullanacak BufferedOutputStream sınıfına ait bir nesneyi parametre olarak almaktadır.

decompressAndUntarFile metodu ".tar.lzma" dosya uzantısına sahip sıkıştırılmış dosyaları açmakta kullanılır. Arşivden çıkarma işlemi, dosyanın ilk olarak LZMA formatından dışarıya çıkarılması, sonra da tar arşivinin açılması şeklinde gerçekleşir.

5.1.5. ZIP

ZIP formatının proje içerisindeki gerçekleştirimi Java programlama dilinin yerleşik kütüphanesi olan java.util.zip kullanılarak gerçekleştirilmiştir. Arşiv oluşturma ve arşivden çıkarma işlemleri iki farklı sınıf kullanılarak gerçekleştirilir.

5.1.5.1. Arşiv oluşturma

ZIP arşivinin oluşturulması kaynak kodu içerisindeki Zip paketi altında yer alan Compress sınıfı içerisinde yapılmaktadır. Sıkıştırma işleminde java.util.zip kütüphanesi içerisinde yer alan ZipEntry ve ZipOutputStream sınıfları kullanılmıştır.

Verilerin sıkıştırılması işlemi sınıf içerisinde yer alan compressFile metodu kullanılarak gerçekleştirilir. Sıkıştırılacak olan dosya eğer bir dizin ise, dizin içerisinde yer alan her dosya için birer adet ZipEntry girdisi oluşturulur. Daha sonra ZipOutputStream sınıfından oluşturulmuş olan nesnenin putNextEntry metodu yardımıyla, veri pozisyonu yazılacak olan girdi için ayarlanır. Bu işlemden sonra BufferedInputStream sınıfından oluşturulmuş bir nesne aracılığıyla, sıkıştırılacak olan dosyadan veriler okunarak ZipOutputStream nesnesinin write metodu kullanılarak sıkıştırma işlemi gerçekleştirilir.

5.1.5.2. Arşiv açma

ZIP Arşivi açma işlemi kaynak kodu içerisindeki Zip paketi altında yer alan Decompress sınıfında gerçekleştirilir. Arşiv açma işleminde java.util.zip kütüphanesinde yer alan ZipEntry ve ZıpInputStream sınıfları kullanılır.

Sıkıştırılmış verileri açma işlemi sınıf içerisinde yer alan decompressFile metodu kullanılarak gerçekleştirilir. ZıpInputStream sınıfından oluşturulan nesneye ait getNextEntry metodu ile sıkıştırılmış dosya içerisinde dosyalara teker teker erişilerek, o anki erişilen ZipEntryi okumak üzere ZipInputStream içerisindeki veri pozisyonu ayarlanır. Daha sonra BufferedOutputStream sınıfına ait bir nesne kullanılarak, sıkıştırılmış veri ZipInputStream nesnesinin read metoduyla okunup dosya hedef dizinine yazılır.

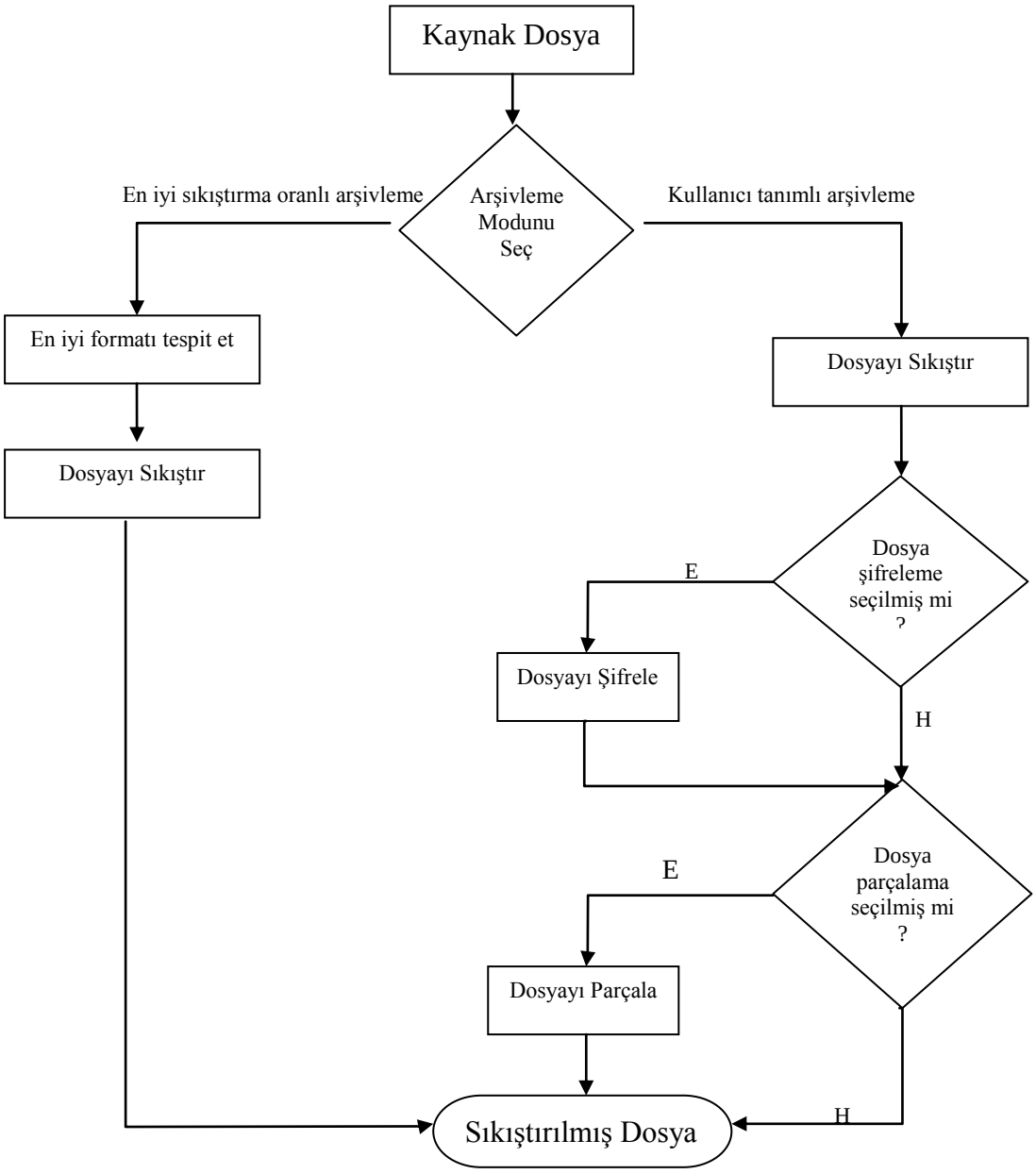
5.2. Şifreleme İşleminin Gerçekleştirilmesi

Arşivlenmiş dosyaların AES şifreleme algoritması kullanılarak şifrelenmesi işlemi, program içerisindeki Common paketi altında yer alan Encrypter sınıfına ait encrypt ve decrypt metotlarında gerçekleştirilir. Şifreleme ve şifrelenmiş verileri şifrelerinin çözülmesi işlemleri Java programlama dilinin gömülü kütüphaneleri olan java.crypto ve java.security kütüphanelerine ait sınıflar kullanılarak gerçekleştirilmektedir.

5.3. Akış Diyagramı

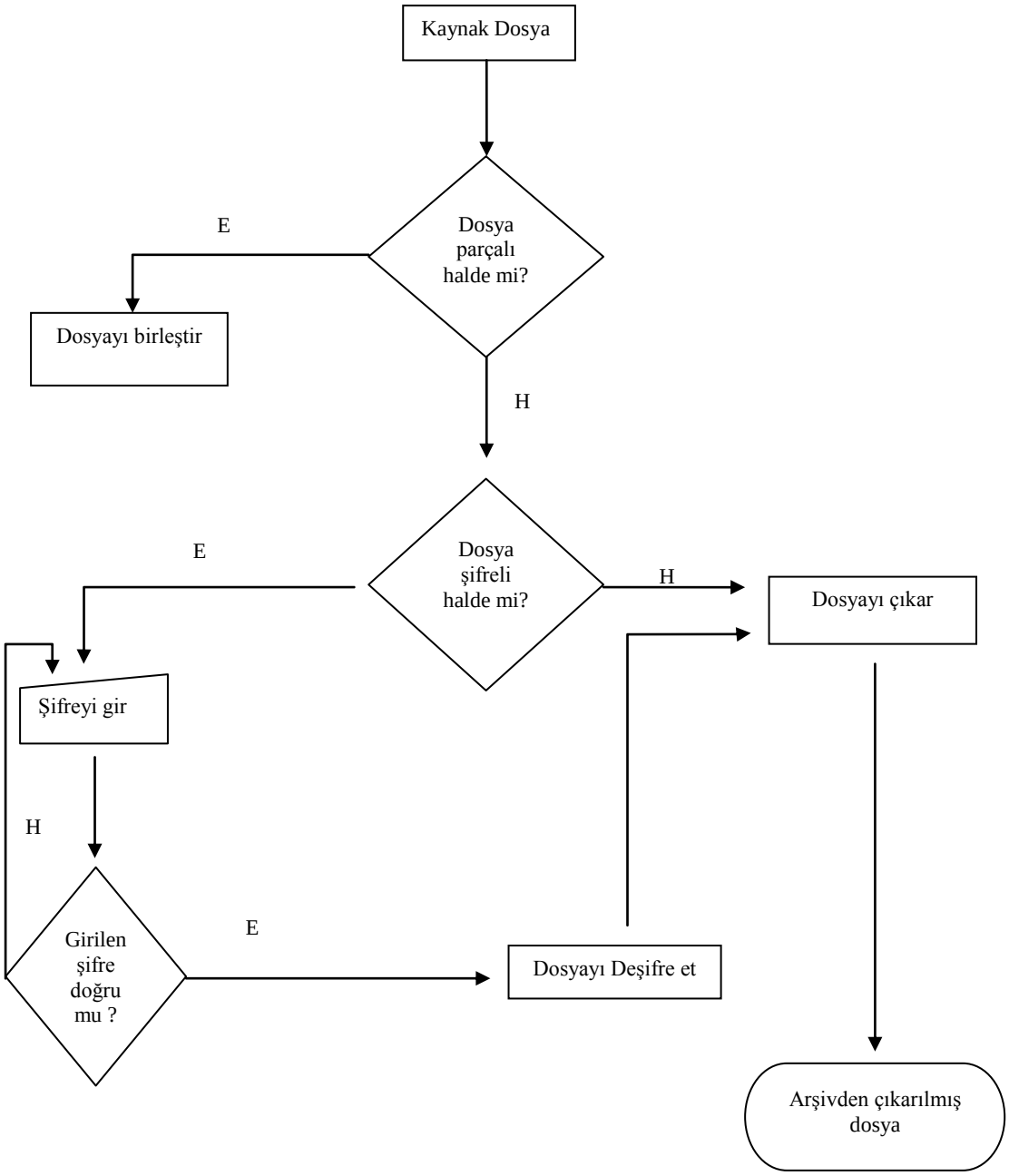
Şekil 5.1'de dosya sıkıştırma işleminin genel işleyişi görülmektedir. Arşivleme işlemine başlamak için ilk olarak arşivlenmesi istenen dosya seçilir. Daha sonra arşivlenecek dosyanın hangi dizine kaydedileceği belirlenir. Dosyanın yazılacağı dizin belirlendikten sonra arşivleme işlemine ait özelliklerin belirlenmesine geçilir. İlk olarak kullanıcı kontrollü arşivlemenin mi yoksa en iyi sıkıştırma oranlı arşivlemenin mi seçileceği belirlenir. Eğer kullanıcı kontrollü sıkıştırma seçilir ise dosyanın sıkıştırılacağı arşivleme formatı, eğer seçilmiş ise şifreleme ve çok parçalı arşiv dosyası özellikleri belirlenerek sıkıştırma işlemi gerçekleştirilir. En iyi sıkıştırma oranlı arşivleme seçilmiş ise şifreleme ve çok dosyalı arşivleme özellikleri seçilemez.

Kullanıcı tanımlı arşivleme de ilk olarak arşivlenmek üzere seçilmiş dosyanın belirlenen formatta sıkıştırılması gerçekleştirilir. Daha sonra eğer şifreleme ve çok parçalı dosya seçenekleri aktif edilmiş ise, sırasıyla ilk olarak şifreleme daha sonra da dosya parçalama işlemleri gerçekleştirilir. En iyi sıkıştırma oranlı arşivleme de arşivlenmek üzere seçilmiş dosya uygun olan bütün formatlarda sıkıştırılarak, en küçük boyut da sıkıştırma işlemini gerçekleştiren arşiv formatı tespit edilir ve dosya bu formatta saklanır.



Şekil 5.1 Dosya arşivleme akış diyagramı

Şekil 5.2'de sıkıştırılmış dosyaları açma işleminin genel işleyişi görülmektedir. İlk olarak arşivlenmiş dosyanın çok parçalı bir arşiv dosyası olup olmadığı kontrol edilir. Daha sonra arşivin şifrelenmiş olup olmadığına bakılır. Son olarak da arşiv dosyasının formatı belirlenerek arşiv açma işlemi gerçekleştirilir.



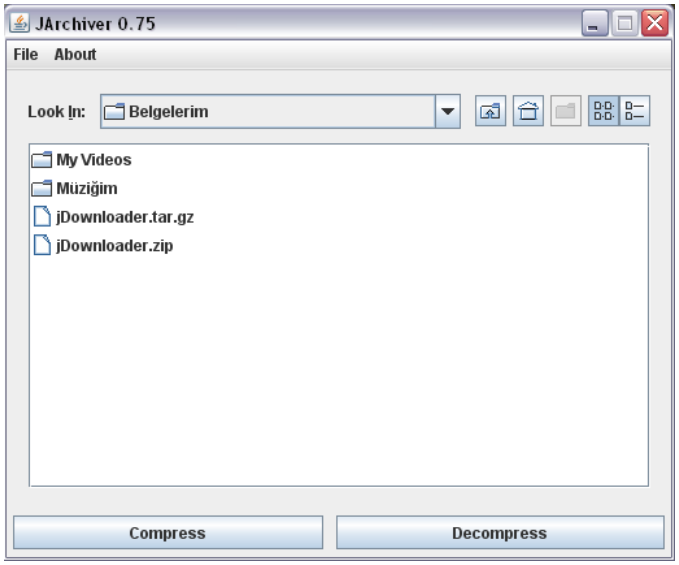
Şekil 5.2 Arşivlenmiş dosyayı çıkarma akış diyagramı

5.4. Uygulamanın Kullanıcı Arayüzü

Uygulamanın kullanıcı arayüzü Java programlama dilinin java.swing ve java.awt kütüphaneleri altında yer alan sınıflar aracılığıyla yaratılmıştır. Bu arayüzleri oluşturan sınıflar kaynak kodu içerisinde yer alan Gui paketi altında yer almaktadır.

5.4.1. Ana menü

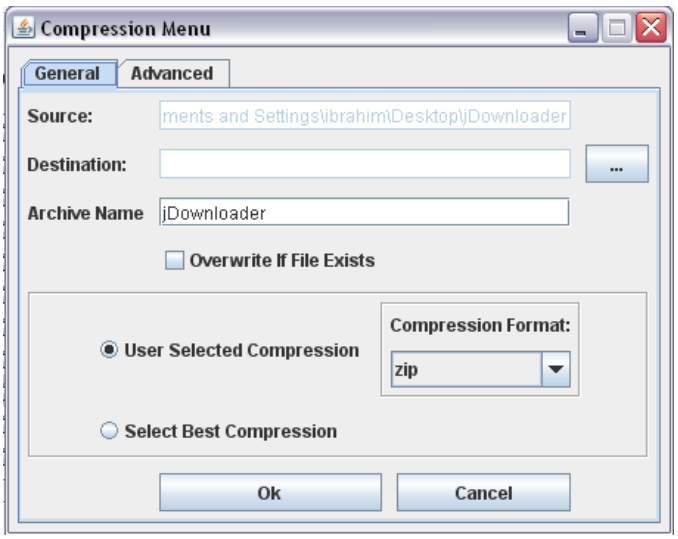
Uygulama çalıştırıldığı zaman kullanıcının karşısına gelen ilk arayüzdür. Kullanıcı arşivlemek veya arşivden açmak istediği dosyayı seçip, "Compress" veya "Decompress" kontrollerinden birisini kullanarak istediği işlemi gerçekleştirir.



Şekil 5.3. Program ana menüsü

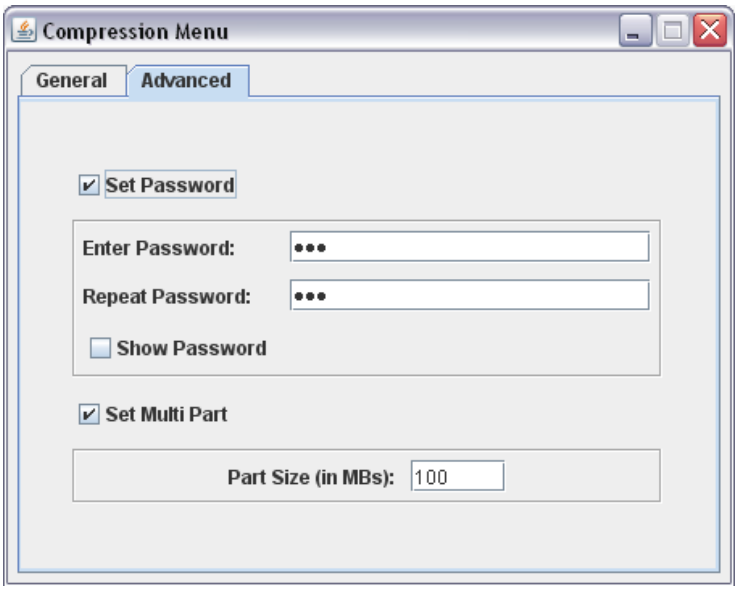
5.4.2. Dosya sıkıştırma menüsü

Ana menü arayüzü kullanılarak belirlenen sıkıştırılacak dosyanın, nereye, hangi isimde sıkıştırılacağının belirlendiği arayüzdür. Ayrıca dosyanın arşivlenmesinde, en iyi sıkıştırma oranlı arşivlemenin mi ya da kullanıcının belirlediği formatın mı kullanılacağı yine bu arayüzde belirlenir.



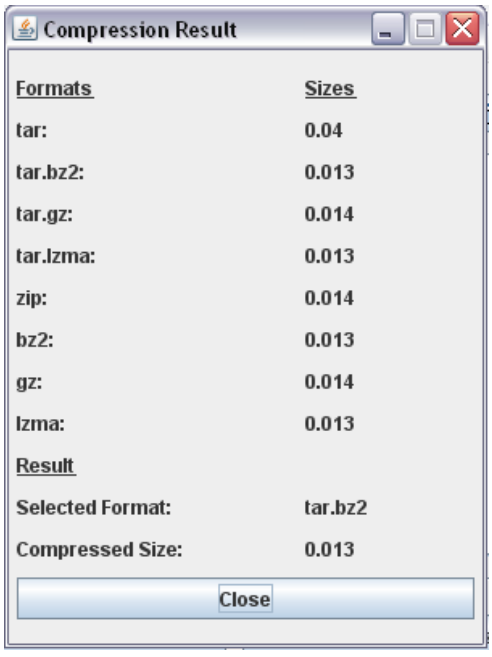
Şekil 5.4 Dosya sıkıştırma menüsü - 1

Arşivleme yöntemi olarak kullanıcının belirlediği formatın kullanılması seçilmiş ise, arşivlenecek dosyanın şifrelenmesi ve çok parçalı arşiv dosyaları halinde kaydedilmesi özellikleri de bu arayüz kullanılarak belirlenir.



Şekil 5.5 Dosya sıkıştırma menüsü - 2

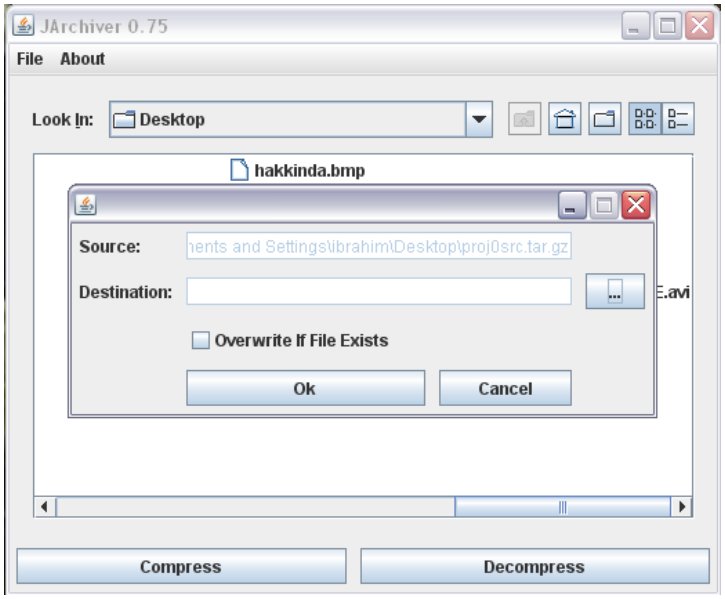
En iyi sıkıştırma oranına sahip arşivleme yöntemi seçilir ise sıkıştırılacak olan dosya uygun olan bütün formatlar ile sıkıştırılarak, dosyayı en küçük boyutta sıkıştıran format bulunur ve dosya o formatta saklanır.



Şekil 5.6. En iyi sıkıştırma oranına dayalı arşivleme sonucu

5.4.3. Dosya açma menüsü

Ana menü kullanılarak seçilen arşiv dosyasının, hangi dizine çıkarılacağı ve çıkarılma işleminde eğer hedef dizinde aynı isimde başka bir dosya var ise üzerine yazma işleminin gerçekleştirilip gerçekleştirilmeyeceği bu arayüz kullanılarak belirlenir.



Şekil 5.7 Arşivlenmiş dosyayı çıkarma menüsü

5.5. Uygulamanın Diğer Arşivleme Uygulamalarıyla Karşılaştırılması

Proje kapsamında geliştirilen arşivleme uygulaması WinZip, WinRar ve 7-zip gibi uygulamalar ile karşılaştırılıp avantajlı ve dezavantajlı yönleri saptanmaya çalışılmıştır.

Proje kapsamında geliştirilen uygulamanın en büyük avantajı, sıkıştırılması istenen dosyaların, format farkı gözetilmeksizin, şifrelenerek ve birden fazla parçadan oluşan dosyalar halinde arşivlenebilmesidir. Yukarıda bahsedilen uygulamalar kendi özelleştikleri arşiv formatında bu işlemleri gerçekleştiriyor olsalar da, destekledikleri diğer formatlarda bu desteği vermemektedirler. Örneğin, WinRar uygulaması RAR formatında oluşturulan arşiv dosyaları için şifreleme ve çok parçalı dosyalar halinde arşivleme işlemini gerçekleştirirken, desteklediği diğer bir format olan ZIP de bu işlemleri gerçekleştirmemektedir. Bu bakımdan proje kapsamında gerçekleştirilen uygulama bu konuda diğer arşiv uygulamalarına göre bariz bir avantaja sahiptir.

Geliştirilen uygulamanın diğer arşiv programlarına göre en büyük dezavantajı, henüz onlar kadar stabil bir şekilde çalışmaması ve gelişmiş seçeneklere sahip olmamasıdır. Uygulamanın ilerleyen sürümlerinde bu dezavantajların önüne geçilmesi planlanmakatadır.

SONUÇ VE ÖNERİLER

Gerçekleştirilen bu tez çalışması neticesine tamamen açık kaynak kodlu, birden fazla formatı kullanarak sıkıştırma ve arşivleme işlemini gerçekleştirebilen, arşivlenecek dosyaların isteğe bağlı olacak şekilde şifrelenip, çok parçalı dosyalar şeklinde saklanabildiği bir yazılım meydana getirilmiştir.

Projenin gerçekleştirilmesindeki ana amaç olan, verilerin sıkıştırılarak depolama birimleri üzerinde daha az yer kaplamasını sağlayacak şekilde sıkıştırılması başarıyla gerçekleştirilmiştir.

Gerçekleştirilen proje sayesinde yaygın olarak kullanılan arşivleme formatları olan ZIP, GZIP, BZIP2, LZMA ve TAR ile dosya sıkıştırma ve başka bir arşiv programı tarafından veya geliştirilen uygulama tarafından oluşturulan bu formatlardaki arşiv dosyalarının açılması sağlanmıştır. Projenin gerçekleştirilmesinde kullanılan Java programlama dili sayesinde platform kısıtlaması da ortadan kaldırılmıştır. Ayrıca bütün formatların, dosya yapısından bağımsız olarak, AES şifreleme algoritması ile şifrelenip güvenli bir şekilde saklanması sağlanmıştır. Oluşturulan arşivin boyutunun çok büyük olması ve depolama birimine tek parça halinde sığmaması durumunda, arşivin bölünmesi sağlanarak birden fazla dosya şeklinde sıkıştırılması sağlanmıştır.

Projenin kapsamında hazırlanan programın çalıştırılması esnasında bazı sorunlarla karşılaşılmıştır. Bu sorunlar ve bunların ileride nasıl çözülmesinin planlandığı aşağıda belirtilmiştir:

- BZIP2 formatı kullanılarak yapılan arşivleme işleminde, diğer formatlara oranla gözle görülür bir yavaşlık hissedilmektedir. Bu yavaşlığın kaynağı BZIP2 dosya formatının kullandığı algoritmadan kaynaklanmaktadır. Bu yüzden program tarafından kullanılan harici kütüphane optimize edilmeye çalışılarak bu yavaşlığın en aza indirilmesi planlanmaktadır.

- Uzantısı TAR olan dosya formatları ve çok parçalı şekilde arşivlenen dosyaların uzantıları silindiği zaman dosya türü tespit edilememektedir. İleride bu tür dosyalar için tür belirleme işlemi, diğer dosya türlerinde olduğu gibi, dosya verisi içerisindeki başlık byte değerlerine bakılarak yapılması planlanmaktadır.

- Gerçekleştirilen programda arşivleme, şifreleme ve dosya parçalama işlemleri birbirlerinden bağımsız olarak gerçekleştirilmektedir. Yani bir dosya sıkıştırılacağı zaman ilk olarak arşivlenip, daha sonra sırayla şifreleme ve dosyayı parçalama işlemleri gerçekleştirilmektedir. Bu işlemlerin ayrı ayrı yapılması büyük oranda zaman kaybına neden olmaktadır. İleride bu işlemlerin üçünün birden senkronize olarak yapılması planlanmaktadır.

- Dosya isimleri sadece ASCII karakter kodlamasını destekler. Bu yüzden dosya isimleri Türkçe karakterler içermemesi gerekir. Aksi takdirde dosya isimlerinde bozulmalar meydan gelecektir. İleride dosya isimlerinin ASCII karakter seti yerine UTF-8 karakter seti ile kodlanması planlanmaktadır.

Belirtilen sorunlara rağmen proje kapsamında, birden fazla işletim sisteminde sorunsuz bir şekilde çalışan, en yaygın arşivleme formatlarını ve güvenli bir şifreleme yöntemini destekleyen bir arşivleme programı yaratılmıştır.

KAYNAKLAR

Daemen, J. and Rijmen, V. (2002) The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography), Springer Press, Berlin, 235s.

Harold, E. (2006) Java I/O, O'Reilly Media, Sebastopol, 555s.

Pu, I. M. (2006) Fundamental Data Compression, Butterworth-Heinemann Press, Oxford, 241s.

Salomon, D., Motta G. and Bryant D. (2006) Data Compression: The Complete Reference, Springer Press, 1092s.

Sayood, K. (2005) Introduction to Data Compression, Morgan Kaufmann, San Francisco, 655s.

Wayner, P. (2005) Compression Algorithms for Real Programmers, Elsevier Press, Amsterdam, 235s.

WEB_1. (2007). .ZIP Application Note. http://www.pkware.com/support/zip-application-note (01.06.2009).

WEB_2 (2003). GZIP File Format Specification Version 4.3. http://www.gzip.org/zlib/rfc-gzip.html (30.05.2009).

WEB_3. (2007). BZIP2. http://www.bzip.org (29.05.2009).

WEB_4. (2009). TAR – GNU Project. http://www.gnu.org/software/tar (02.06.2009).

WEB_5 (2009). 7-ZIP. http://www.7-zip.org (01.06.2009).

WEB_6 (2009). The Apache Ant Project. http://ant.apache.org (03.03.2009).

WEB_7 (2009). Burrows-Wheeler transform. http://en.wikipedia.org/wiki/BWT (01.06.2009).

ÖZGEÇMİŞLER

Kişisel bilgiler	
Ad / Soyad	**Ramis Taşgın**
Adres	Yunus Emre Mahallesi, 6465 sokak, Emsan Evleri B1 Blok Kapı no. 7, Kınıklı / DENİZLİ
Telefon	05358586155
e-posta	rtasgin@gmail.com
Uyruk	T.C.
Doğum tarihi	16/01/1987
Eğitim ve öğretim	
Tarihler	Eylül 2005 – Devam ediyor.
Eğitim ve öğretim veren kurumun adı	Pamukkale Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü
Tarihler	Eylül 2000 – Haziran 2005
Eğitim ve öğretim veren kurumun adı	Kemer Yabancı Dil Ağırlıklı Lisesi
Staj	
Tarihler	Haziran 2007 – Temmuz 2007
Eğitim ve öğretim veren kurumun adı ve türü	Antalya Türk Telekom Müdürlüğü, Bilgi İşlem Daire Başkanlığı
Kişisel beceri ve yeterlilikler	
Bilinen yabancı diller	İngilizce
Okuma kabiliyeti	İyi
Yazma kabiliyeti	İyi
Konuşma kabiliyeti	Orta
Mesleki Kabiliyet ve Beceriler	**Programlama Dilleri** C, C#, Java **İşletim Sistemleri** Windows, Linux **Web** Php, Html **Veritabanı** MS SQL Server 2005, MySQL.

Kişisel bilgiler	
Ad / Soyad	**İbrahim TAŞDEMİR**
Adres	Yunus Emre Mahallesi, 6465 sokak, Emsan Evleri B1 Blok Kapı no. 7 Kınıklı / DENİZLİ
Telefon	05365637118
e-posta	tasdemiribrahim@mynet.com
Uyruk	T.C.
Doğum tarihi	8/12/1986
Eğitim ve öğretim	
Tarihler	Eylül 2005 – Devam ediyor.
Eğitim ve öğretim veren kurumun adı	Pamukkale Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü
Tarihler	Eylül 2000 – Haziran 2005
Eğitim ve öğretim veren kurumun adı	Boğazlıyan Anadolu Lisesi
Staj	
Tarihler	Temmuz 2008 – Ağustos 2008
Eğitim ve öğretim veren kurumun adı ve türü	Kayseri Türk Telekom ADSL/DATA Bölümü
Kişisel beceri ve yeterlilikler	
Bilinen yabancı diller	İngilizce
Okuma kabiliyeti	İyi
Yazma kabiliyeti	İyi
Konuşma kabiliyeti	Orta
Mesleki Kabiliyet ve Beceriler	**Programlama Dilleri** C, C#, Java **İşletim Sistemleri** Windows, Linux **Web** Php, Html **Veritabanı** MS SQL Server 2005, MySQL.