



T.C.

SELÇUK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

GÖRÜNTÜ STEGANOGRAFİ MEDOTLARI VE PERFORMANSLARININ KARŞILAŞTIRILMASI

BARIŞ DEMİRCİ

YÜKSEK LİSANS TEZİ

Bilişim Teknolojileri Mühendisliği Anabilim Dalı

Mart-2016
KONYA
Her Hakkı Saklıdır

ÖZET

YÜKSEK LİSANS

GÖRÜNTÜ STEGANOGRAFİ MEDOTLARI VE PERFORMANSLARININ KARŞILAŞTIRILMASI

Barış DEMİRCİ

Selçuk Üniversitesi Fen Bilimleri Enstitüsü
Bilişim Teknolojileri Mühendisliği Anabilim Dalı

Danışman: Yrd. Doç. Dr. Kemal TÜTÜNCÜ

2016, 74 Sayfa

Jüri
Asst. Prof. Dr. Kemal TÜTÜNCÜ
Prof. Dr. Novruz ALLAHVERDİ
Doç. Dr. Fatih BAŞÇİFTÇİ

Gizliliği sağlama günümüz iletişim sistemlerindeki önemli konulardan bir tanesidir. Gizliliğin öneminin vazgeçilmez olduğu uygulamalarda; gizlenmek istenen bilgilerin üçüncü kişilerin eline geçmeden veya onların anlayamayacağı şekle getirilerek ilgili hedefe gönderilmesi temel amaçtır. Ancak herhangi bir nedenden ötürü gönderilecek olan bilgi üçüncü şahısların eline geçtiğinde gizlenen bilginin açığa çıkması istenmeyen bir durumdur.

Veri iletişimi ve depolama yeteneği alanında yaşanan gelişmeler artan dijital makine üretimiyle birleşince önlemez bir veri büyümesine yol açmıştır. Ayrıca bilgisayar, cep telefonu, tablet gibi makinelerin yaygınlaşarak her düşünceden insanca kullanılması veri güvenliği alanında ciddi tartışmalara sebebiyet vermiştir. Bankacılıktan, ulusal güvenliğe, tıptan özel yaşama mahremiyet gerektiren konularda yaşanan siber saldırılar bu tartışmaları daha da alevlendirmiş bilgisayar bilimlerinde veri güvenliği disiplini üzerindeki çalışmaları artırmıştır.

Bu çalışmada bilgisayar güvenliğinin gözde konularından birisi olan ve kötü niyetli kişilerden veri gizleme amacıyla sıklıkla başvurulan yöntemlerden birisi olan steganografi üzerinde durulmuştur. Dört temel steganografi metodu olan En Önemsiz Bite Gömme, Sözde Rasgele En Önemsiz Bite Gömme, EzStego ve F5 açıklanmış ve kaos tabanlı gömme yeni bir steganografik yöntem olarak önerilmiştir. Önerilen yöntem diğer temel metotlarla birlikte on (10) adet örnek resim üzerinde test edilmiş ve sonuçlar irdelenmiştir. Kaotik rastgele sayı üreteçlerinin tahmin edilemez ve kestirilemez indisler üretmeleri sayesinde önerilen metodun saldırılara karşı güvenli metot olduğu görülmüştür.

Anahtar Kelimeler: En Önemsiz Bite Gömme, EzStego, Kaos Tabanlı Gömme, Sözde Rasgele En Önemsiz Bite Gömme, Steganografi

ABSTRACT

MS THESIS

IMAGE STEGANOGRAPHY METHODS AND PERFORMANCE COMPARISON

Barış DEMİRCİ

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE OF SELÇUK UNIVERSITY
THE DEGREE OF MASTER OF SCIENCE
IN INFORMATION TECHNOLOGY ENGINEERING

Advisor: Asst. Prof. Dr. Kemal TÜTÜNCÜ

2016, 74 Pages

Jury
Advisor Asst. Prof. Dr. Kemal TÜTÜNCÜ
Prof. Dr. Novruz ALLAHVERDİ
Assoc. Prof. Dr. Fatih BAŞÇİFTÇİ

Providing confidentiality is one of the major issues in communication systems today. In applications where confidentiality is an indispensable importance; the basic aim is to send the concealed information without going into the hands of a third party or in the form such a way that receiver other than desired one will not be able to understand the information. Additionally, it is undesirable to see the disclosure of hidden information by third party in case of being captured.

Combination of developments in the field of data communication and storage capabilities and increased the production of digital cameras has led to unprevented growth of data. In addition, serious discussions are going on for data security due to widespread usage of machines such as computer, mobile phone and etc. by anyone with any sense. Cyber-attacks to privacy fields such as banking, national security, medical data, individual private life and etc. increased these discussions and let to increase the studies in data security discipline of computer science.

In this study, one of the commonly used and favorite topic of computer security called as steganography that aims data hiding from malicious people were focused on. Basically four steganographic methods namely least significant bit embedding, random least significant bit embedding, Ez stego and F5 were described and then chaos based embedding method was proposed as a new steganographic method. The proposed method along with other basic methods were tested on ten (10) sample images and the results were evaluated. It has been seen that proposed chaos method is robust method against attacks due to production of unpredictable and unguessable indices by chaotic random number generator.

Keywords: Chaos Based Embedding, Ez stego, Least Significant Bit Embedding, Random Least Significant Bit Embedding, Steganography

ÖNSÖZ

Yüksek Lisans ders aşamasında derslerini aldığım hocalarım: Prof. Dr. Novruz ALLAHVERDİ, Doç. Dr. Mehmet ÇUNKAŞ, Yrd. Doç. Dr. A. Alparslan ALTUN, Yrd. Doç. Dr. Rıdvan SARAÇOĞLU ve çalışmalarım sırasında beni yönlendiren, yardım ve desteğini esirgemeyen, bana her yönden danışmanlık eden saygıdeğer hocam Yrd. Doç. Dr. Kemal TÜTÜNCÜ' ye teşekkürlerimi sunarım. Ayrıca tüm öğrenim hayatım boyunca maddi, manevi desteklerini esirgemeyen aileme, bu tezin hazırlanması esnasındaki desteklerinden ve gösterdikleri özverilerden dolayı eşim ve kızıma şükranlarımı sunarım.

Barış DEMİRCİ
KONYA-2016

İÇİNDEKİLER

ÖZET ..... **iv**

ABSTRACT ..... **v**

ÖNSÖZ ..... **vi**

İÇİNDEKİLER ..... **vii**

SİMGELER VE KISALTMALAR ..... **ix**

1. GİRİŞ **1**

2. KAYNAK ARAŞTIRMASI **4**

2.1. Kriptografi 4
2.2. Sır Paylaşımı 6
2.3. Damgalama 7
2.4. Steganografi 7
2.4.1. Ses üzerinde steganografi 9
2.4.2. Video üzerinde steganografi 9
2.4.3. Network steganografi 9
2.4.4. Metin steganografi 10
2.4.5. Resim üzerinde steganografi 10

3. MATERYAL VE YÖNTEM **13**

3.1. Kullanılan Steganografi Algoritmaları 13
3.1.1. En önemsiz bite gömme 14
3.1.2. Sözde rastgele en önemsiz bite gömme 16
3.1.3. Palet tabanlı steganografi ve ez-stego algoritması 18
3.1.4. F5 algoritması 22
3.2. Önerilen Metod Kaos Tabanlı Gömme 24
3.2.1. Kaos kavramı 25
3.2.2. Geliştirilen yöntem 28
3.3. Test Metodları 29
3.3.1. Doruk işaretten gürültü oranına (peak signal to noise ratio) testi 30
3.3.2. Ki-kare testi 30
3.3.3. Kapasite testi 31
3.3.4. Histogram testi 31
3.3.5. Yapısal Benzerlik Indeksi 32
3.4. Geliştirilen Yazılım 33

4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA **39**

5. SONUÇLAR VE ÖNERİLER **47**

5.1. Sonuçlar 47
5.2. Öneriler 48

KAYNAKLAR **49**

EKLER **53**

EK-1 Önerilen algoritmanın kaynak kodları 53
EK-2 Kaotik Rastgele Sayı Üreteç Sınıfı 60
EK-3 Temel İşlemlerin tanımlandığı ve gerçekleştirildiği genel.cs sınıfı 63

ÖZGEÇMİŞ **74**

SİMGELER VE KISALTMALAR

DES:	Data Encryption Standart
CBC:	Cipher Block Chaining
CFB:	Cipher Feedback
IBM:	International Business Machines
FIPS:	Federal Information Processing Standard
RSA:	Rivest-Shamir-Adleman cryptosystem
ECC:	Elliptic curve Cryptography
IDEA:	International Data Encryption Algorithm
PGP:	Pretty Good Privacy
SHA:	Secure Hash Algorithm
NIST:	National Institute of Standards and Technology
AES:	Advanced Encryption Standard
UEKAE:	Ulusal Elektronik ve Kriptoloji Araştırma Enstitüsü
TCP:	Transmission Control Protocol
UDP:	User Datagram Protocol
ICMP:	Internet Control Message Protocol
IP:	Internet Protocol
OSI:	Open Systems Interconnection
ASCII:	American Standard Code for Information Interchange
AKD:	Ayrık kosinüs dönüşümü
FD:	Fourier dönüşümü
PSNR:	Peak signal to noise ratio
MSE:	Mean squared error
PoVs:	Pairs of values
MSSIM:	Mean Structural Similarity Index

1. GİRİŞ

Gelişen bilgisayar ve ağ teknolojileri iletişimde veri akışını ve alışverişini büyük oranda kolaylaştırmıştır. Bu kolaylaştırma işlenecek büyük veri miktarı ve bu verilerin korunmasını içeren güvenlik sorunlarını da beraberinde getirmiştir. Devletlerin en gizli istihbarat servislerinden halkla iç içe olan sanatçıların hayran kulüplerine kadar kullanılan iletişim kaynağının internet olması güvenlik ihtiyacını artırmıştır.

Bütün bu sebepler eskiden beri var olan bir takım güvenlik uygulamalarının bilgisayar bilimlerine aktarılmasına yol açmıştır. Bugün bu kavramların içeriği eskiye oranla çok daha hızlı bir şekilde dolmaktadır. Veri güvenliği kriptografi, sır paylaşımı, damgalama, steganografi gibi pek çok farklı alana ve disipline ayrılmış durumdadır.

Bu alanlardan kriptografi ve sır paylaşımın tarihi antik çağlara kadar uzanmakta iken damgalama ve steganografi nispeten çok daha yeni alanlardır ve ancak dijital bilgisayarların yaygınlaşması ve güçlenmesiyle ivme kazanabilmişlerdir. Bu yöntemlerin ortak özellikleri veri güvenliğiyle ilgilenmeleridir yani amaçları veriyi istenmeyen kişilerden korumalarıdır. Kriptografi saklamak istenilen verinin tahribi ve ancak alıcı tarafından geri oluşturulabilmesi üzerine kuruludur (Beimel, 2011). Sır paylaşımı ise bu anlamda kriptografiye yakındır. Bu disipline göre veriler belirli sayıda kişi arasında bölüştürülür ve ancak belirlenen miktarda pay bir araya gelince asıl veri elde edilir (Shamir, 1979).

Steganografi ise bu açıdan diğer yöntemlerden bambaşka bir rotada seyretmektedir. Steganografide asıl amaç veriyi bozmak değil başka ve ilgisiz bir veri içerisinde saklamaktır. Saklanacak veriye yataklık edecek veri istenmeyen kişilerin dikkatini çekmeyecek diğer bir deyişle bu kişilerin eline geçse bile fark edemeyecekleri veri türüdür. Ki bu kimi zaman bir video dosyası, kimi zaman bir resim, kimi zaman html dosyası bazense hasılat rekorları kıran bir müzisyenin şarkısı olabilir.

Bu kadar farklı alanlarda varlık göstermesine ve yaygınlaşmasına karşın steganografi disiplininin itici gücü resim steganografisidir. Bunun en başta gelen sebebi resmin gerek sosyal medya gerekse e-posta ortamıyla en rahat paylaşılabilen ve taşınabilen form olmasıdır. Video ve ses paylaşımı pek çok ofis ortamında ve ciddi kurumlarda tüzüklerle kısıtlanmıştır ve bu dosyalar ister istemez üçüncü kişilerin dikkatini çekmektedir. Ayrıca değişik video ve ses dosyalarını bilgisayar ortamında açabilmek için kodeklere (şifre çözücülere) ve ek yazılımlara gereksinim vardır. Oysa

resim bir şirketin logosundan, reklam linkine kadar pek çok alanda kullanılabilir. Ayrıca bilgisayar ortamında resim dosyasını saklamak ve açmak çok kolaydır.

Sayılan bu sebepler dolayısıyla bu tezde güncel ve önemli bir disiplin olan steganografi üzerinde durulmuştur. Mevcut steganografi yöntemleri incelenmiş steganografinin güncel ve yaygın bir kolu olan resim içerisine metin (yazı) saklama diğer bir deyişle gömme işlemi irdelenmiştir. Resim içerisinde veri saklamak amacıyla en yaygın kullanılan algoritmalardan en önemsiz bite gömme ve sözde rastgele sayı üretici ile en önemsiz bite gömme incelenmiş palet tabanlı, nicemlemeye dayalı veri gizleme disiplinlerinde yer alan algoritmalara da yer verilmiştir. Bu yöntemlerin avantaj ve dezavantajları gerek literatür araştırmasıyla gerekse de deneylerler tespit edilmeye çalışılmıştır. Pek çok algoritmanın temelde en önemsiz bite gömme algoritması üzerine kurulu olduğu müşahede edilmiştir.

Bu kapsamlı incelemeler sonucunda en önemsiz bite gömme algoritması üzerinde yapılacak iyileştirmelerin önemli sonuçlar doğurabileceği fark edilmiştir. Zira ileride değinileceği üzere bu tekniğin en zayıf noktası bitlerin sırayla gömülmesi dolayısıyla bir şüphe anında rahatlıkla tespit edilebilmesidir.

Sözde rastgele sayı üreteçleri ile üretilen indislere veri gömmesi tatbik edildiğinde bu zayıflık kısmen giderilebilmektedir. Bununla beraber bu üreteçler kolayca kırılabilen zayıf algoritmalardır. Tüm bu sebepler bu tezin konusunu daha güçlü rastgele sayı üreteçleri aramaya itmiştir. Tahmin edilemezlik konusunda kaosun ciddi bir avantaja sahip olduğu görülmüştür (Shannon, 1949).

Literatürde kaos resim kriptografisi alanında Wong ve ark. (2001) tarafından kullanılmış bir yöntem olsa da kaos tabanlı steganografinin daha az çalışılmış bir alan olduğu ayrıca fark edilmiştir. Bu da kaos tabanlı rastgele sayı üreteci kullanmak konusundaki fikirleri perçinleyen bir etken olmuştur. Kaotik rastgele sayı üretiminin avantajlarının yanında dezavantajları da vardır. Bu dezavantaj da başarımını sınamak için geliştirilen yöntemlerin daha çok istatistiki üreteçler üzerinde geliştirilmiş olmasıdır. Kaotik üreteçleri tam anlamıyla test edebilen standart bir yöntem mevcut değildir. Fakat yine de mevcut testler üzerinde kaotik üreteçlerin başarılı olduğu gözlenmektedir.

Bu doğrultuda Logistic harita adlı kaotik algoritmalara başvurularak renkli resimler içerisine veri gizleyen yeni bir steganografi metodu geliştirilmiş ve sonuçlar paylaşılmıştır. Geliştirilen metodun kapasite ve sadece renkli resimler üzerinde

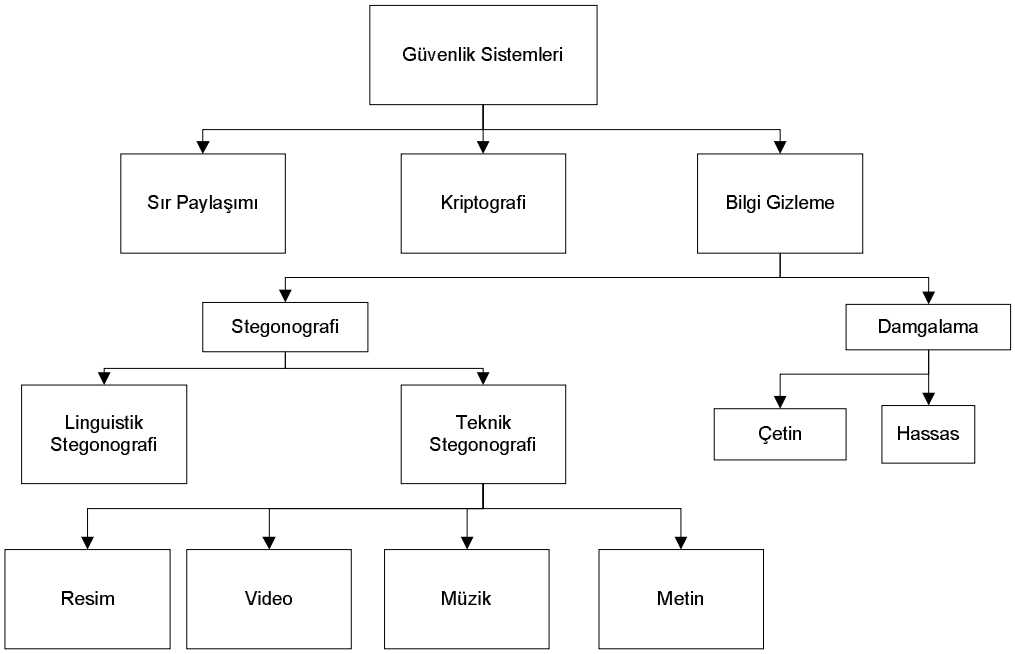
çalışabilmesiyle diğer metotlardan geride olmakla beraber kaotik tabanlı olması hasebiyle klasik metotlardan daha güvenli olduğu test edilmiştir.

Ayrıca bünyesinde literatürde sıklıkla kullanılan en önemsiz bite gömme, sözde rastgele en önemsiz bite gömme, EzStego ve F5 algoritmalarının yer aldığı bir yazılım C# programlama diliyle .NET Framework 4.0 çatısı altında Visual Studio yazılım geliştirme ortamında yazılmış, yine sıklıkla başvurulan test yöntemleri bu yazılıma dâhil edilmiştir.

Tezin 2. Bölümünde kaynak araştırmasına yer verilmiştir, 3. Bölümde yukarıda bahsedilen dört temel algoritma ile birlikte önerilen yöntem de anlatılmıştır. Geliştirilen yazılımla yapılan test sonuçları 4. Bölümün içerisinde yer almaktadır. 5. Bölümde genel sonuç ve öneriler sunulmuştur.

2. KAYNAK ARAŞTIRMASI

Bilgisayar bilimlerinde veri güvenliği kriptografi, sır paylaşımı ve steganografi olmak üzere üç temel dala ayrılmış vaziyettedir. Güvenlik sistemleri Cheddad ve ark. (2010) tarafından Şekil 2.1'deki gibi sınıflandırılmıştır. Bu sınıflandırma incelendiğinde bilgi gizlemenin kollarının yer aldığı görülmektedir. Fakat önemli bir konu olan sır paylaşımı bu ağaçta yer almamaktadır. Bu yüzden sır paylaşımı tekniği şemaya tarafımızdan ilave dilmiştir.



Şekil 2.1 Güvenlik Sistemlerinin Sınıflandırılması

2.1. Kriptografi

Kriptografi insan tarafından anlaşılabilen verinin matematik yöntemlerle bozularak anlaşılamaz hele getirilmesidir. Elbette veri bozmada kullanılan parametrelerin veriyi alması gereken kişilerce kullanılarak eski haline getirilebilmesi elzemdir. Çimen ve ark. (2008) tarafından kriptografinin kısa bir tarihçesi şu şekilde verilmiştir.

MÖ 1900 dolaylarında bir Mısırlı kâtip yazdığı kitabelerde standart dışı hiyeroglif işaretleri kullandı.

MÖ 60-50 Julius Caesar (MÖ 100-44) normal alfabedeki harflerin yerini değiştirerek oluşturduğu şifreleme yöntemini devlet haberleşmesinde kullandı. Bu

yöntem açık metindeki her harfin alfabede kendisinden 3 harf sonraki harfle değiştirilmesine dayanıyordu.

725-790 Abu Abd al-Rahman al-Khalil ibn Ahmad ibn Amr ibn Tammam al Farahidi al-Zadi al Yahmadi, kriptografi hakkında bir kitap yazdı (Bu kitap kayıp durumdadır). Kitabı yazmasına ilham kaynağı olan, Bizans imparatoru için Yunanca yazılmış bir şifreli metni çözmesidir. Abu Abd al-Rahman, bu metni çözmek için ele geçirdiği şifreli mesajın başındaki açık metni tahmin etme yöntemini kullanmıştır.

1000 - 1200 Gaznelilerden günümüze kalan bazı dokümanlarda şifreli metinlere rastlanmıştır. Bir tarihçinin dönemle ilgili yazdıklarına göre yüksek makamlardaki devlet görevlilerine yeni görev yerlerine giderken şahsa özel şifreleme bilgileri (belki şifreleme anahtarları) veriliyordu.

1586 Blaise de Vigenère (1523-1596) şifreleme hakkında bir kitap yazdı. İlk kez bu kitapta açık metin ve şifreli metin için otomatik anahtarlama yönteminden bahsedildi. Günümüzde bu yöntem hala DES CBC ve CFB kiplerinde kullanılmaktadır.

1623'de Sir Francis Bacon, 5-bit ikili kodlamayla karakter tipi değişikliğine dayanan stenografi buldu.

1790'da Thomas Jefferson, Strip Cipher makinesini geliştirdi. Bu makineyi temel alan M-138-A, ABD donanmasının 2.Dünya savaşında da kullandı.

1917'de Joseph Mauborgne ve Gilbert Vernam mükemmel şifreleme sistemi olan one-time pad'i buldular.

1920 ve 1930'larda FBI içki kaçakçılarının haberleşmesini çözebilmek için bir araştırma ofisi kurdu. William Frederick Friedman, Riverbank Laboratuvarlarını kurdu, ABD için kriptoanaliz yaptı, 2. Dünya savaşında Japonlar'ın Purple Machine şifreleme sistemini çözdü. II. Dünya savaşında Almanlar Arthur Scherbius tarafından icat edilmiş olan Enigma makinasını kullandılar. Bu makine Alan Turing ve ekibi tarafından çözüldü.

1970'lerde Horst Feistel (IBM) DES'in temelini oluşturan Lucifer algoritmasını geliştirdi. ·

1976'da DES (Data Encryption Standard), ABD tarafından FIPS 46 (Federal Information Processing Standard) standardı olarak açıklandı.

1976 Whitfield Diffie ve Martin Hellman Açık Anahtar sistemini anlattıkları makaleyi yayınladılar.

1978'de Ronald L. Rivest, Adi Shamir ve Leonard M. Adleman: RSA algoritmasını buldular.

1985'de Neal Koblitz ve Victor S.Miller ayrı yaptıkları çalışmalarda eliptik eğri kriptografik (ECC) sistemlerini tarif ettiler.

1990'da Xuejia Lai ve James Massey: IDEA algoritmasını buldular.

1991'de Phil Zimmerman: PGP sistemini geliştirdi ve yayınladı.

1995'de SHA-1 (Secure Hash Algorithm) özet algoritması NIST tarafından standart olarak yayınlandı. 1997'de ABD'nin NIST (National Institute of Standards and Technology) kurumu DES'in yerini alacak bir simetrik algoritma için yarışma açtı.

2001'de NIST'in yarışmasını kazanan Belçikalı Joan Daemen ve Vincent Rijmen'e ait Rijndael algoritması, AES (Advanced Encryption Standard) adıyla standart haline getirildi.

2005'de Çin'li bir ekip tarafından SHA–1 algoritmasının kırıldığı duyuruldu. Bunun üzerine Amerika, Microsoft ve Sun gibi birçok büyük firma artık kullanmayacaklarını açıkladı.

2007'de artık her programcı algoritma geliştirebilir ve bu algoritmayı programlayabilir hale gelmiştir. Fakat geliştirilen algoritmalar her seferinde kırılarak geçerlilik sağlayamamıştır. Türkiye'de UEKAE tarafından bu gelişimler takip edilmekte ve sürekli kriptoloji konusu üzerinde çalışmalar gerçekleştirilmektedir (Ulutürk, 2010).

2.2. Sır Paylaşımı

Sır paylaşımı $k \leq n$ olmak üzere verinin n sayıda kişiye paylaştırılmasıdır. Bu n parçadan hiç biri ana veriyi tek başına oluşturamaz ancak en az k tane parça bir araya geldiğinde ana veri tekrar oluşturulabilir.

Sır paylaşımı (Secret Sharing) yöntemi ilk defa Shamir (1979) ve Blakley (1979) tarafından birbirlerinden bağımsız bir şekilde ortaya atılmıştır.

İlk olarak, Thien ve Lin (2002) tarafından, gizli görüntülerin ağ üzerinden iletimi esnasında steganografi ya da Kriptografi kullanmanın getireceği problemlere vurgu yapılmıştır.

Her iki yöntem sonucunda üretilen stego görüntü veya şifreli görüntü tek bir ortamda tutulmaktadır. Ortamın ağ üzerinden iletimi esnasında tahrip olması durumunda, gizli verinin yeniden elde edilmesi olanağı yoktur.

Thien ve Lin gerek hataya karşı toleransın sağlanması için gerekse gruba güven mekanizmasının uygulanabilmesi için Shamir'in yönteminin gizli görüntülerin iletiminde kullanılabileceğine vurgu yapmıştır (Ulutaş ve ark., 2012).

2.3. Damgalama

Kriptografi ve sır paylaşımından farklı olarak damgalama Telif haklarını korumak (Copyright protection), doğrulama yapmak (Authentication), gizli haberleşme yapmak ve imge hakkında açıklayıcı bilgi saklamak için en sık kullanılan yöntemlerden biri imgenin içine bir logonun (damga, watermark) çeşitli yöntemlerle görünmeyecek şekilde eklenmesidir. Bu, damgalama (watermarking) olarak adlandırılmaktadır (Chouse ve Albayrak, 2014).

Damgalama sıklık (frekans – frequency) ve uzamsal (spatial) olmak üzere iki temel kol üzerine kurulu bir disiplindir. Uzamsal alanındaki damgalamada doğrudan piksel değerleri üzerinde işlem yapılır. Sıklık alanındaki damgalama ise Fourier, Ayrık Kosinüs ve Wavelet gibi frekans dönüşümleri marifetiyle gerçekleştirilir.

Swanson ve ark. (1996) ayrık kosinüs dönüşümüne dayalı bir yöntem önermişlerdir.

Pereira ve ark. (1999) frekans üzerindeki en temel işlem olan Ayrık Fourier dönüşümünü baz alan yeni bir yöntem üzerinde durmuşlardır.

Bir yıl sonra Lu ve ark. (2000) Wavelet dönüşümü ile bambaşka bir damlama tekniği üzerinde durmuşlardır. Gün geçtikçe artan telif hakları sorunları damgalamayı güncel bir teknik olarak gündemde tutacağa benzemektedir.

2.4. Steganografi

Steganografi Grekçe steganos 'gizli' ve graphein 'yazı' sözcüklerin birleşmesinden oluşan bir terimdir. Diğer disiplinlere göre yeni kurulmuş bir alandır. Temeli Johannes Trithemius'in 1499 tarihli steganographia eseriyle atılmış bu disiplinde amaç saklanmak istenen veriyi ilgisiz başka bir verinin içine saklayarak göndermektir. Diğer güvenlik yöntemlerine göre en önemli avantajı da burada yatmaktadır. Zira kriptografide gerçek veri bozularak gönderildiği için istenmeyen kişi veriyi gördüğünde bunun bozulmuş ve saklanmak istenen bir veri olduğunu anlayabilir steganografide ise ele geçen veri bozulmamış ama saklanan veri ile ilgisiz bir kayıttır. Burada saklanmak

istenen bir veri olduğunu anlayabilmek çok güçtür. Dolayısıyla üçüncü kişilerin dikkatini çekmez.

Steganografide metin, ses, video, dijital resim gibi farklı dosya türleri içerisine veri gizlemek mümkündür. Gizlenen veri de benzer şekilde metin, ses, video ya da dijital resim olabilir. Modern anlamda steganografinin temelleri Simmons (1984)'un ortaya attığı mahkum problemi ile atılmıştır. Steganografide içine bilgi yerleştirilecek ortama örtü verisi (cover-data) içinde gizli veriyi barındıran sonuç haline de stego nesnesi (stego object) adı verilir.

Steganografi günümüzde de güncelliğini koruyan bir disiplindir. Yakın zamanlı çalışmalara göz atacak olursak Ramalingam ve Isa (2015) rasgele anahtar kodlama fonksiyonu ile birlikte video üzerindeki kırmızı, yeşil, mavi kanallara veri gömme işlemi icra etmişlerdir.

Swain (2014) dokuz piksel farkı ve gözden geçirilmiş en önemsiz bit ile resim üzerinde steganografi uygulamışlardır.

Srinivasan ve ark. (2015) saklı sır paylaşım algoritmasına dayalı bir steganografi üzerinde durmuşlardır.

Yuan (2014) multi cover uyarlanabilir steganografi algoritması önermiştir.

Hong (2013) resmin tamamı üzerinde değil sadece kenarları içerisine veri gizleyerek başka bir bakış açısıyla olaya yaklaşmıştır.

Sedighi ve ark. (2016) istatistiki keşfedilebilirliği en aza indirmeye yönelik uyarlanabilir bir steganografi üzerinde çalışmışlardır.

Cogranne ve Fridrich (2015) sınıflandırıcı ve steganografi arasında ilginç bir yaklaşım kurmuşlardır.

Sedighi ve ark. (2015) Genelleştirilmiş Gauss cover model ile steganografi denemişlerdir.

Ker ve ark. (2013) steganografinin laboratuvardan gerçek dünya ortamına taşınması ile ilgili çalışmaları olmuştur.

Li ve ark. (2014) uzamsal resim steganografi için yeni bir maliyet fonksiyonu geliştirmişlerdir.

Ghebleh ve Kanso (2014) dijital resim steganografisi için 3 boyutlu kaotik kedi haritası ve ayrık dalgacık dönüşümünde Swelden'in kaldırma şemasını temel alan güçlü bir kaotik algoritma oluşturmuşlardır.

Aşağıda ortama göre steganografi çeşitlerinden bahsedilmiştir.

2.4.1. Ses üzerinde steganografi

Ses üzerinde steganografi en önemsiz bite gömme, eşlik kodlama, faz kodlama, yaygın spektrum ve yankı saklama olmak üzere beş temel koldan meydana gelen bir disiplindir (Jayaram ve ark., 2011).

Eşlik kodlama, sinyali ayrık örneklere böler ve iletinin her bitini eşlik bitinden gömer.

Faz kodlama başlangıç ses segmentinin fazını saklı bilgiyi temsil eden referans fazla yer değiştirir.

Yaygın spektrum en önemsiz bite gömmeye yakın bir şekilde saklı bilgiyi ses işaretinin frekans spektrumu boyunca yayar.

Yankı saklama ise saklı bilgiyi esas ses dosyası içerisine ayrık bir sinyal olacak şekilde yankı formunda gömer.

Wu ve Liu (2013), Pooyan ve Delforouzi (2007) ve Verma ve ark. (2013) bu alanda yapılmış çalışmalara örnek olarak gösterilebilir.

2.4.2. Video üzerinde steganografi

Video üzerinde steganografi de ana hatları itibariyle ses üzerindeki steganografiye benzemektedir. Bu kolda da temel dal en önemsiz bite gömmedir (Choudry ve Aakash, 2015).

Bu çalışmalara örnek olarak Chae ve Manjunath (1999)'ın çalışması gösterilebilir. Bir diğer teknik daha güçlü bir gömme yöntemi olan Yaygın Spektrum yöntemidir. Bu yöntemde döndürme ve öteleme gibi geometrik dönüşümler sonrası yaşanan veri kaybı diğer yöntemlere göre daha azdır (Cox ve ark., 1997).

Video üzerinde yapılan steganografide ayrık kosinüs dönüşümü Wang ve Izquierdo (2002) ve Wavelet dönüşümü Noda ve ark. (2004)'ne başvurarak yapılmış çalışmalar mevcuttur.

2.4.3. Network steganografi

Örtü nesnesi olarak TCP, UDP, ICMP, IP gibi ağ protokolleri gizli bilgi taşıyıcı olarak kullanılması ile network steganografi yapılmış olur. OSI ağ katmanı modelinde

örtü olarak kullanılacak kanallar bulunmaktadır. Örneğin TCP/IP alanlarındaki başlık bitinin boş kısımları veri gizleme için kullanılabilir.

2.4.4. Metin steganografi

Metin Steganografi örtü nesnesinin text olduğu Steganografi alanıdır. Metin steganografi genellikle uygulama yapılması zor bir veri gizleme yöntemidir. Metin steganografide saklanabilecek veri miktarı çok azdır. Çünkü taşıyıcı text'in içindeki boşluklar ve gereksiz alanlar az miktarda bulunmaktadır.

2.4.5. Resim üzerinde steganografi

Bu tezde değerlendirilen ve önerilen metotlar resim içerisine veri gizlemeyle ilgilidir. Dolayısıyla bu bölüm ayrıntılı olarak tetkik edilecektir. Resim üzerinde steganografi hakkındaki literatür taramasına geçmeden önce klasik bir Steganografi algoritmasının amaçlarını belirtmek yerinde olacaktır. Resim Üzerinde Steganografi hakkındaki beklentiler Hussain ve Hussain (2013) tarafından Çizelge 2.1.'deki gibi belirlenmiştir.

Çizelge 2.1 Steganografi algoritmasının kalitesini belirleyen ölçütler ve bu ölçütlerin avantajları ve dezavantajları

Ölçü	Açıklama	Avantaj	Dezavantaj
Yüksek Kapasite	Resim içerisine gömülebilecek azami veri miktarı	Yüksek	Düşük
Algısal Şeffaflık	Veri gömmesi işlemi tamamlandıktan sonra insan tarafından farkın algılanıp veri saklama işleminden şüphelenmesi	Yüksek	Düşük
Sağlamlık	Gömme işlemi gerçekleştikten sonra resimde meydana gelen küçültme, büyütme, döndürme uzaklaştırma gibi işlemler sonucunda gömülü metnin orijinal haliyle yani kayıpsız olarak korunup korunamaması.	Yüksek	Düşük
Tavlama Direnci	Resmin içerisine metin gömüldükten sonra metinde değişiklik / güncelleme yapma imkânı	Yüksek	Düşük
Hesaplama Karmaşıklığı	Resim içerisine veri gömmenin ve gömülmüş veriyi geri getirmenin hesaplama maliyeti diğer bir deyişle güçlüğü	Düşük	Yüksek

Resim üzerinde steganografi teknikleri Hussain ve Hussain (2013) tarafından dört ana başlık altında incelenmiştir. Bunlar uzamsal uzay, dönüşüm uzayı, bükülme uzayı ve son olarak süzme ve maskelemedir.

Bu ana başlıklardan en önemlisi uzamsal uzaydır. Bu başlık altında yer alan ikinci başlıklar en önemsiz bit, piksel değeri farkı, kenar tabanlı veri gömme, rastgele

piksel gömme, pikseli gizli veriye adresleme, etiketleme, piksel şiddeti tabanlı, doku tabanlı ve histogram kaydırma yöntemleridir.

Bu yöntemlerde asıl resim az değişime yani bozulmaya uğrar ve resim içerisine geniş kapsamlı veri gizlenebilir fakat bu yöntemler dayanıksızdır. Resim üzerindeki küçük bir tahribatta asıl gizlenen veriler kaybolur ayrıca pek çok steganaliz metodu bu verileri çözebilir.

Dönüşüm uzayları tekniklerinde ise daha zorlu ve karmaşık bir algoritma yapısı söz konusudur. Veri gömmek için değişik dönüşüm algoritmaları uygulanabilir. Önemli frekans dönüşümleri olan Ayrık Fourier Dönüşümü, Ayrık Kosinüs Dönüşümü, Ayrık Wavelet Dönüşümü bu başlık altında kullanılan dönüşümlere örnek gösterilebilir. Bu yöntemlerin en önemli avantajları resim üzerinde küçültme, bükme gibi işlemlerden daha az etkilenmeleridir.

Süzme, işaret işleme alanının önemli bir konusudur. En yalın haliyle işaretteki belirli frekans bileşenlerini kabul etme veya reddetme olarak tanımlanabilir. Süzme işlemi için kullanılan yapılara süzgeç (maske) adı verilir ve süzgeçler geçirdikleri işaret frekanslarına göre adlandırılır. Örneğin alçak frekansları geçiren bir süzgeç alçak geçirgen süzgeç olarak adlandırılır. Süzgeç kavramını görüntü işleme disiplini için biraz daha açacak olursak; Uzaysal bölgedeki bir görüntünün (x,y). pikseli etrafında tanımlanan ve bu pikselin komşu pikselleri arasındaki ilişkileri incelemesini sağlayan kare ya da dikdörtgen dizilere süzgeç adı verilmekte ve pratikte bu maskeler süzme işlemlerinde 3x3 piksel olarak kullanılmaktadır (Jane, 2010). Bir maskeyi görüntü üzerinde gezdirme yani süzme işlemi katlama olarak da adlandırılır. İki boyutlu tanımlanan a(n,m) ile h(n,m) fonksiyonları için katlama işlemi genel olarak (2.1) numaralı denklemle tanımlanmakta ve katlama sonucu, b(n,m) fonksiyonu olarak elde edilmektedir (Jane, 2010).

$$
\begin{aligned}
b(n,m) &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} a(k_1,k_2)h(x-k_1, y-k_2) \\
&= -\infty + \ldots + a(-1,-1)h(x+1, y+1) + a(0,-1)h(x, y+1) + a(-1,0)h(x+1, y) \\
&\quad + a(0,0)h(x, y) + a(1,0)h(x-1, y) + a(0,1)h(x, y-1) + a(1,1)h(x-1, y-1) + \ldots + \infty
\end{aligned}
\tag{2.1}
$$

Şekil 2.2'de örnek alçak geçirgen ve yüksek geçirgen süzgeçler ve kaynak resim üzerine uygulanmış biçimleri görülmektedir.

$\begin{bmatrix} 0{,}11 & 0{,}11 & 0{,}11 \\ 0{,}11 & 0{,}11 & 0{,}11 \\ 0{,}11 & 0{,}111 & 0{,}11 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$	
(a) Alçak Geçirgen Süzgeç	(b) Yüksek Geçirgen Süzgeç	
		
(c) Orijinal Görüntü	(d) Alçak Geçirgen Süzgeç Uygulanmış Hali	(e) Yüksek Geçirgen Süzgeç Uygulanmış Hali

Şekil 2.2 Alçak ve yüksek geçirgen süzgeçler ile kaynak resim ve süzgeçlerin uygulanmış biçimleri

Literatürde resim tabanlı steganografi alanında yapılmış hayli çalışma mevcuttur. Bunlar arasında Grover (1998)'ın telif amacıyla yaptığı çalışmalar önemli bir yer tutmaktadır.

Resim üzerinde steganografinin bir başka yaygın kullanım alanı da tıbbi veriler içerisine bilgi saklanmasıdır (Zagade ve Bhosale, 2014). Bu bilgiler genellikle mahrem olan hasta, adı, cinsiyeti, yaşı, hastalığı üzerinedir.

Yapay zekâ metotları da güçlü steganografik algoritmalar eklemede sıklıkla başvurulan yöntemlerdir. Kanan ve Nazeri (2014) hibrit sinir ağlarına dayalı yöntemi steganografide kullanmışlardır.

El-Emam ve Al-Zubidy (2013) ise sezgisel bir optimizasyon tekniği olan genetik algoritmaları steganografi alanında tatbik etmişlerdir.

Sajedi ve Jamzad (2010) ise tamamen farklı bir anlayış benimseyerek resim ön işleme metodu ile steganografi denemişlerdir.

3. MATERYAL VE YÖNTEM

Bu bölümde temel steganografi yöntemleri ve geliştirilen yöntem hakkında bilgi verilmiştir. Ardından yazılımda kullanılan test metotlarına değinilmiş son olarak da geliştirilen yazılım hakkında bilgi verilmiştir.

3.1. Kullanılan Steganografi Algoritmaları

Bu tez kapsamında en önemsiz bite gömme, sözde rastgele sayı üreteçine dayalı en önemsiz bite gömme, EzStego ve F5 algoritmaları ile birlikte önerilen kaosa dayalı yeni bir steganografi yöntemi kullanılmıştır. Kullanıcı tarafından resmin içine gömülmesi için metin olarak girilen veri aşağıda verilen genişletilmiş ASCII tablosuna göre ikili veriye dönüştürülerek resim içerisine gömülmüştür.

Codepage 1254 - Turkey Windows

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	€ 20AC	0081	‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152	008D	008E	008F
9-	0090	‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153	009D	009E	Ÿ 0178
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	2 00B2	3 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	1 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ğ 011E	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	İ 0130	Ş 015E	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ğ 011F	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ı 0131	ş 015F	ÿ 00FF

Şekil 3.1 Genişletilmiş ASCII Tablosu

3.1.1. En önemsiz bite gömme

En önemsiz bite gömme algoritması steganografi alanındaki temel yöntemdir. Bu yöntemin temel hareket noktası görüntü içerisine gömülecek metni ikili (binary) hale çevirmek ardından da ikili verinin her bir bitini kaynak resimdeki pikselleri tarayarak her bir pikselin en önemsiz bitine gömmektir. Verinin gömüleceği pikseller belirli bir algoritmaya göre değil sıralı olarak seçilir. Dolayısıyla gömülmüş veriyi ortaya çıkarma işlemi sırasında metnin kaç karakterden oluştuğu bilinemeyeceği için sonuna işaret karakteri eklenir.

Aşağıda veri gömme adımları yer almaktadır.

Adım 1: Gömülecek metni ve kaynak resmi al. Metnin sonuna işaret karakteri ekle
Adım 2: Metni bitlerine ayır (ikili veriye çevir)
Adım 3: İkili veri sonuna kadar adım 4 ve adım 5'i uygula
Adım 4: Resmin sıradaki pikselini al ve bitlerine ayır
Adım 5: Resmin sıradaki pikseline ait bitlerinin son bitini verinin sıradaki biti ile değiştir
Adım 6: İşlemi sona erdir.

Çizelge 3.1'de 6 bitlik bir verinin 6 piksel içerisine gömülmesine dair örnek yer almaktadır.

Çizelge 3.1 Örnek bit gömme işlemi

Gömülecek Bitler	1	1	0	0	1	1
Resme Ait Bitler	01000010	11111101	11100111	01100001	01101011	00101010
Sonuç	01000011	11111101	11100110	01100000	01101011	00101011

Görüldüğü üzere kaynak resim üzerindeki değişiklik sadece son bitlerde yaşanmıştır. Kaynak resmin son bitinin gömülecek verinin sıradaki bitiyle aynı olması durumunda sonuç değişmeyecektir. Renkli resimlerde kırmızı, yeşil ve maviden oluşan üç kanal olduğu için veri istenen kanala gömülebilir. Veri boyutu fazla ise sırayla üç kanala da gömülmesinde bir sakınca olmayacaktır.

8 bit ile temsil edilen bir resimde her bir piksel [0 – 255] aralığında değerler almaktadır. Dolayısıyla en önemsiz bitte yani 8. Bitte yapılacak değişiklik piksel değerinde en fazla 1 artışa ya da 1 azalmaya yol açabilir bu da insan gözünün algılayamayacağı bir değişiklik demektir.

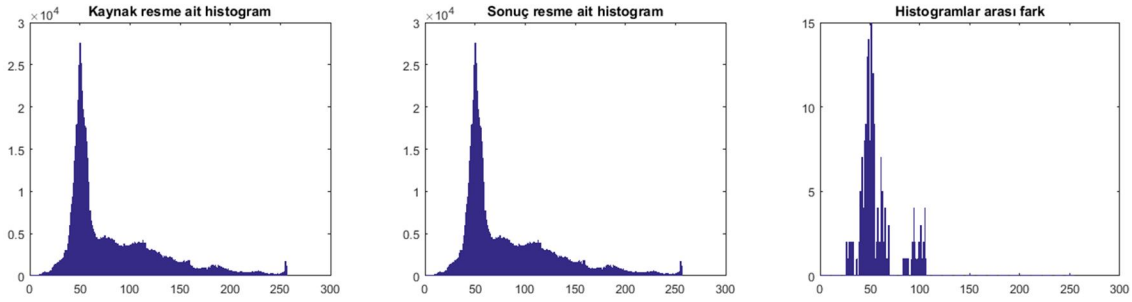
Şekil 3.2'de örnek bir resim ve en önemsiz bite gömme yöntemiyle veri gömülmüş hali yer almaktadır.



(a) Kaynak Resim (b) İçerisine veri gömülmüş resim

Şekil 3.2 En önemsiz bite gömme yöntemi uygulanmış resimler

Görüldüğü üzere iki resim arasındaki fark insan tarafından algılanamayacak kadar azdır. Ayrıca iki resim arasındaki histogram farkı da rahatlıkla anlaşılamayacak kadar azdır. Şekil 3.3.'de resimlerin gri düzeylerine ait histogramlar ve bu histogramlar arası farklar görülmektedir.



Şekil 3.3 Kaynak resim ve sonuç resim histogramları

Gömülen metni öğrenme adımları veri gömme adımlarının tersidir. Bu adımlar aşağıda verilmiştir.

Adım 1: Kaynak resmi al
Adım 2: İşaret karakterine ulaşıncaya kadar adım 3 -5'i uygula
Adım 3: Sıradaki pikselin binary değerlerini al
Adım 4: Binary değerlerinin son bitini diziye at
Adım 5: Dizinin elamanları karakter oluşturmaya yetince karakteri oku ve sonuç katarına ekle. Son karakter işaret karakteri ise döngüden çık
Adım 6: İşlemi sona erdir.

3.1.2. Sözde rastgele en önemsiz bite gömme

Klasik en önemsiz bite gömme algoritması temel ve ilk algoritma olması ve pek çok diğer algoritma açısından steganografi alanında önemli bir algoritmadır. Çıplak gözle görülemez oluşu da ayrı bir avantajıdır. Bununla birlikte metin verilerinin resim üzerine sırayla gömülmesi çok önemli bir dezavantajdır. İstenmeyen kişiler tarafından ele geçirilen resmin bitleri tek tek incelenerek resim içerisine gömülen metin rahatlıkla ortaya çıkartılabilir. Bu olumsuz durumun önüne geçmek için metnin sıralı bir şekilde değil de karışık sırayla gömülmesi fikri ortaya atılmıştır (Devi, 2013).

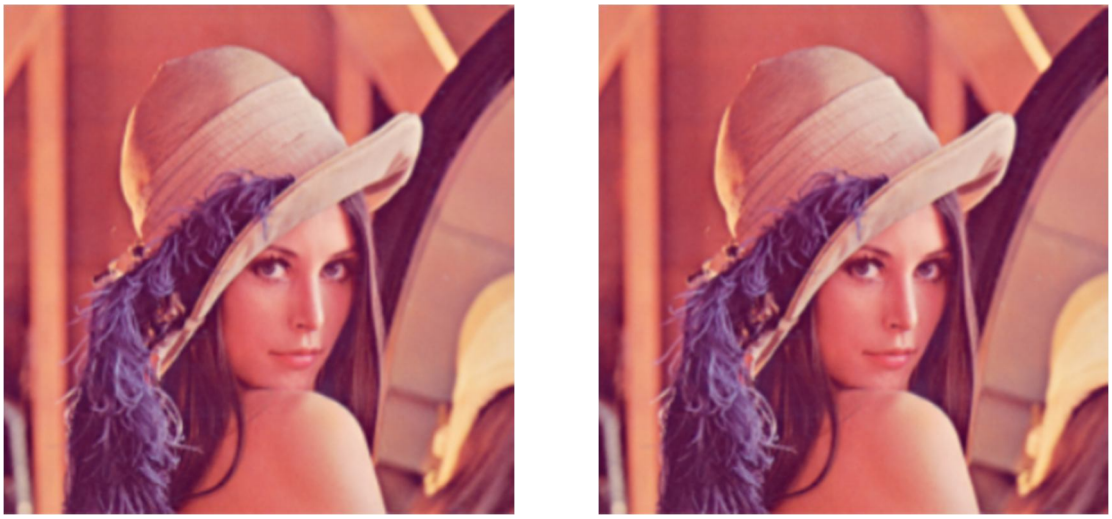
Bu yöntemde en yaygın olarak kullanılan üreteç Linear congruential generator'dir (Lehmer, 1951). Bu algoritmaya göre s_0 başlangıç değeri olarak alınarak rekürsif olarak (3.1) formülüyle üretilir.

$$s_{i+1}=as_i+b \bmod m,\ i=0,1\ldots \tag{3.1}$$

Bu yöntemlerin tam olarak rastgele oldukları iddia edilemez zira deterministik yapıları dolayısıyla yeniden hesaplanmaları mümkündür. Bu algoritmanın işlem adımları aşağıdaki gibidir.

Adım 1:	Gömülecek metni ve kaynak resmi al. S0 başlangıç değerini al.
Adım 2:	Metni bitlerine ayır (ikili veriye çevir)
Adım 3:	Metnin uzunluğu kadar elemana sahip ve bütün elemanları eşsiz olan rastgele sayı kümesini s0 değerini kullanarak üret.
Adım 4:	İkili veri sonuna kadar adım 5 ve adım 6 i uygula
Adım 5:	Rastgele sayı dizisinin sıradaki elemanına denk düşen pikseli resimden al ve bitlerine ayır.
Adım 6:	Pikselin bitini verinin sıradaki biti ile değiştir
Adım 7:	İşlemi sona erdir.

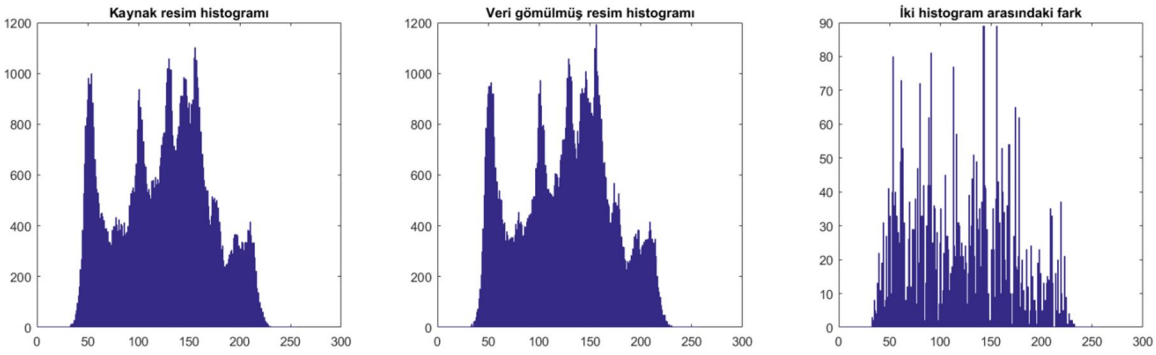
Şekil 3.4'te bu algoritmaya içerisine veri gömülmüş resim ve orijinal resim görülmektedir.



(a) Kaynak Resim (b) İçerisine veri gömülmüş resim

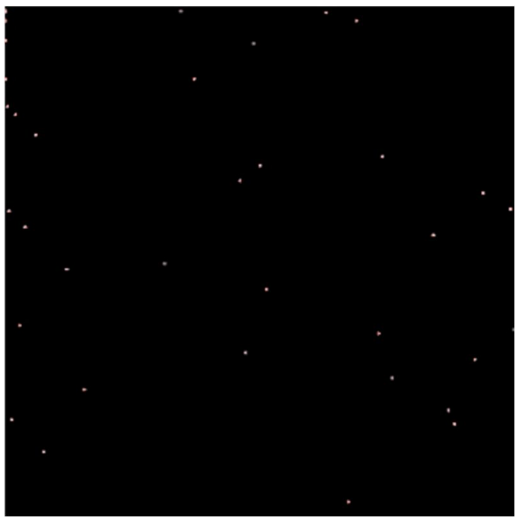
Şekil 3.4 Sözge rastgele en önemsiz bite veri gömme işlemi örnek sonuçları

Görüldüğü üzere bu iki resim arasındaki fark insan gözüyle ayırt edilemeyecek denli küçüktür. Şekil 3.5.'de yer ait histogram değerleri incelendiğinde de ciddi bir fark olmadığı görülmektedir.



Şekil 3.5 Kaynak resim ve sonuç resim histogramları ile bu histogramlar arasındaki fark

Yukarıda bu algoritmayı klasik en önemsiz bit algoritmasından ayıran temel noktanın verileri sıralı değil de rasgele dağıtılmış noktalar içerisine gömdüğü belirtilmişti. Şekil 3.6'da Şekil 3.4'teki resimler arasındaki farklı pikseller (gri düzeye dönüştürülmüş) kırmızı renkli olarak görülmektedir.

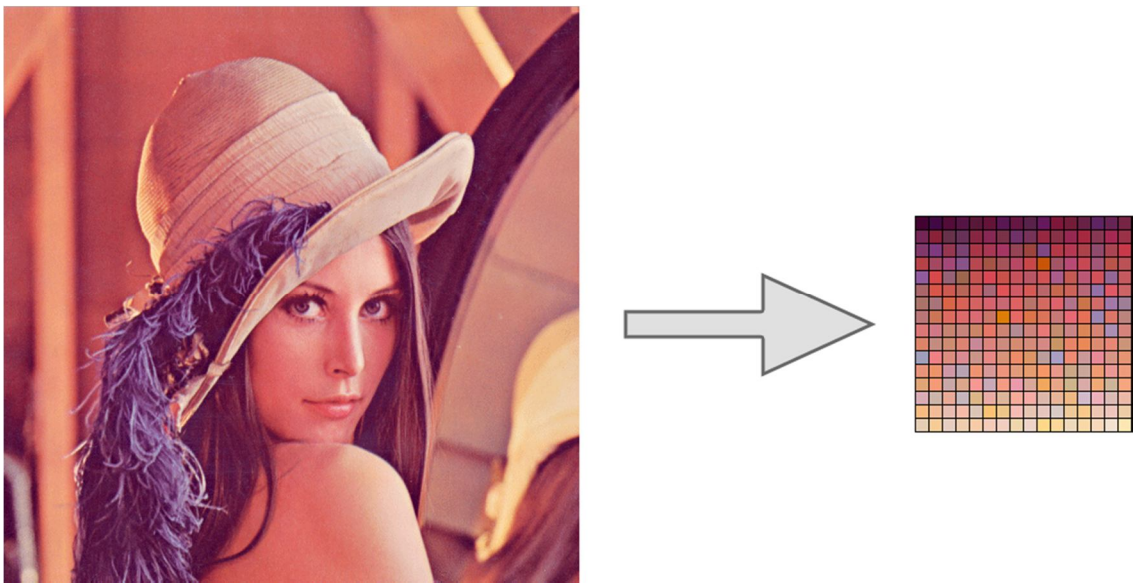


Şekil 3.6 İki resim arasında farklı noktalar

Görüldüğü üzere bu iki resim arasındaki fark insan gözüyle ayırt edilemeyecek denli küçüktür. Şekil 3.5'de histogram değerleri incelendiğinde de ciddi bir fark olmadığı görülmektedir.

3.1.3. Palet tabanlı steganografi ve ez-stego algoritması

Palet tabanlı steganografi yöntemlerini açıklamadan önce palet mantığını izah etmek zorunludur. Renkli görüntülerde her bir piksel kırmızı, yeşil ve maviden oluşan üç bileşenden meydana gelmektedir. Her bir bileşenin [0,255] aralığında değer alması yani 8 bit ile ifade edilmesi durumunda pikseli oluşturan renk 8x3 = 24 bit ile ifade edilecektir. Resimdeki piksel sayısının artması durumunda dosya boyutu aşırı şişecektir. Palet mantığı renk sayısının sınırlı olduğu durumlarda dosya boyutunu azaltmak için geliştirilmiş bir çözümdür. Bir resimde 256 farklı renk olduğunu düşünelim, bu renk bilgileri palette tutulursa her bir piksel için 24 bitlik renk değeri değil de rengin palet üzerindeki numarası saklanır böylece her bir piksel 24 bit yerine azami 8 bitlik palet numarası ile ifade edilmiş olur. Şekil 3.7. örnek bir palet tabanlı resim ve palet renkleri görülmektedir. Bu algoritmada resmin piksel sayısı arttıkça elde edilen kazanç da artacaktır.

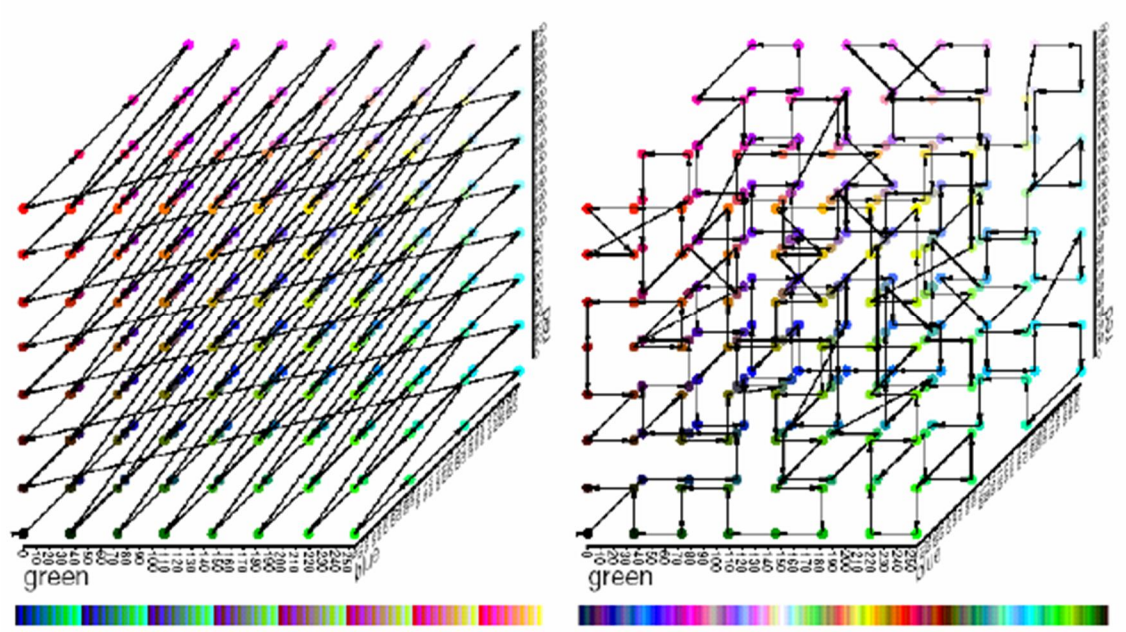


Şekil 3.7 Palet tabanlı bir resim ve palet renkleri

Algoritmanın 1. adımı renk paletini sıralamakla başlar. Paletteki her bir renk kırmızı, yeşil ve maviden oluşan üç kanaldan meydana geldiği için bunların yakınlıklarını ölçmek önemli bir noktadır. Çoğu zaman yakınlık ölçütü olarak bu değerlerin ortalaması ya da Öglid Mesafesi (3.2) alınır.

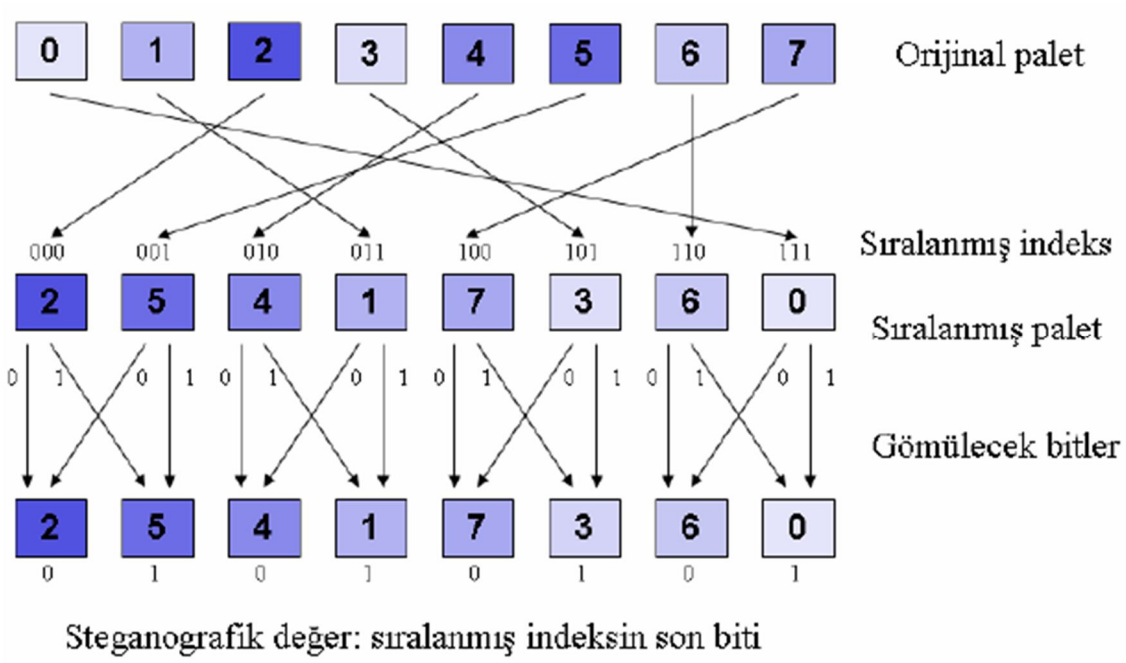
$$d=\sqrt{(R_1-R_2)^2+(G_1-G_2)^2+(B_1-B_2)^2} \tag{3.2}$$

Komşu renklerin birbirine yakın olması ve güç ayrılması arzulanan bir durumdur. Şekil 3.8.'de orijinal palet ve Ezstego ile paletin yeniden sıralanmış hali görülmektedir.



Şekil 3.8 (a) Palette renklerin sıralanışı (b) EzStego kullanılarak sıralanmış palet

Sonraki aşamada resim piksel piksel taranır ve her bir piksel değerinin yeni sıralanmış palet üzerindeki yeri tespit edilir. Tespit edilen yer değerlerinden ara matris elde edilir. Daha sonra klasik en önemsiz bit algoritması ile birlikte ara matris değeri üzerinde veri gömme işlemi icra edilir. Şekil 3.9.'da bu süreç özetlenmiştir.



Şekil 3.9 Gömme Fonksiyonu

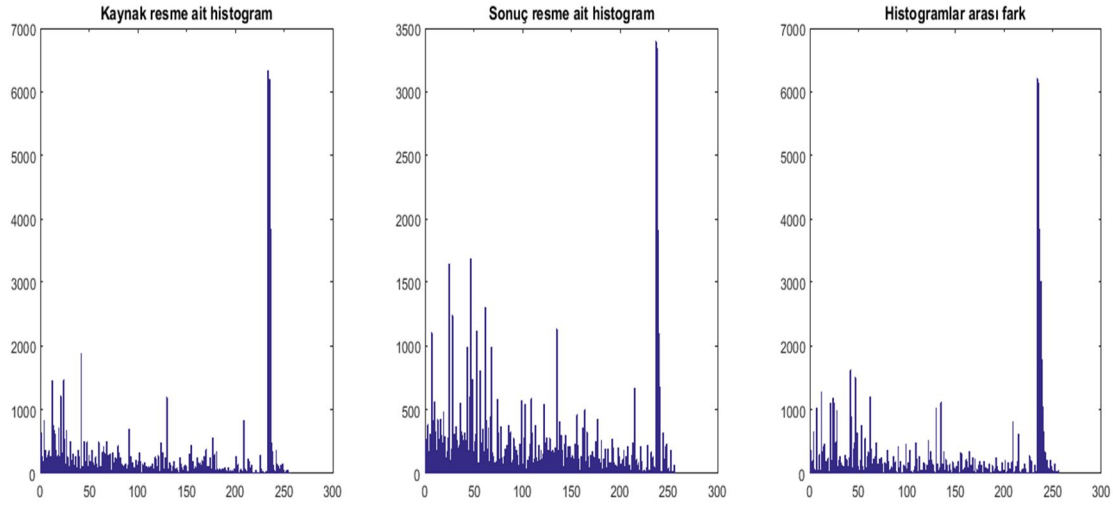
Son olarak elde edilen yeni matris taranır. Matrisin her bir değerinin orijinal paletteki karşılığı bulunur ve sonuç resmine o renk değeri işlenir.

Algoritmanın veriyi tekrar elde etme bir diğer deyişle çözme işlemi de benzer bir yapıyla yapılır. Resim paleti gömme işlemi sırasındaki sıralama algoritması ile sıralanır. Resimdeki her bir piksel taranır ve pikselin sıralı palet üzerindeki yeri tespit edilir. Sıralardan oluşan ara matris elde edilir. Sıralı matris klasik en önemsiz bite gömme algoritmasının veri çözme tekniğiyle analiz edilerek resmin içerisine gömülmüş olan metin ortaya çıkarılır.



Şekil 3.10 Örnek resim ve Ezstego algoritması ile veri gömülmüş hali

Şekildeki resimlerden de görüleceği üzere bu algoritmada gömülen metin uzunluğu arttıkça sonuç insan gözü tarafından rahatlıkla algılanabilmektedir. Bunun sebebi verinin en önemiz bite gömülmek yerine renk sıralarını değiştirmesi yani rengin değerini 1 azaltmak yerine tamamen değiştirmesidir. Şekil 3.11.'de Şekil 3.10.'daki resimlere ait histogram değerleri ve bunlar arası farklar görülmektedir.

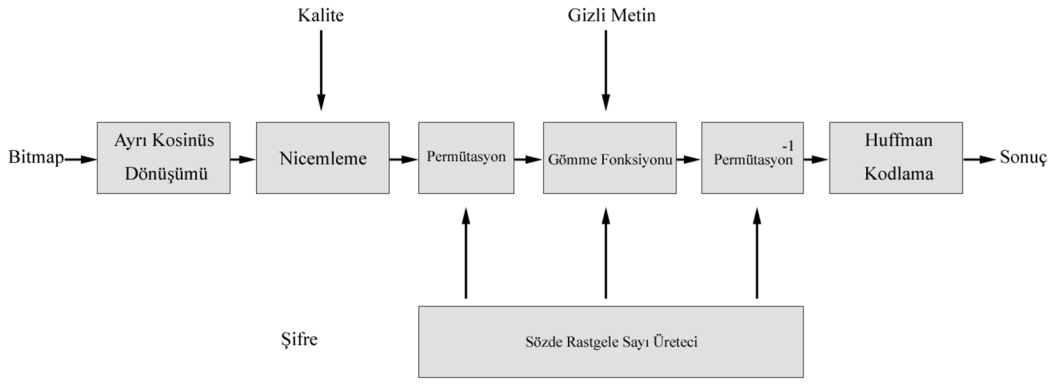


Şekil 3.11 Kaynak resme ait histogram, sonuç resme ait histogram ve arasındaki farklar**.**

Histogramlar arası farklar gözle belli olacak kadar barizdir. Palet tabanlı yöntemlerin bir diğer dezavantajı da sadece palet tabanlı işlem yapan gif, png gibi dosya türleri için uygulanabilir olması diğer dosya biçimlerine izin vermemesidir.

3.1.4. F5 algoritması

F5 Algoritması Westfeld (2001) tarafından ayrık kosinüs dönüşümü temeli üzerine kurulmuş bir steganografi yöntemidir. Ayrık kosinüs dönüşümü (AKD) tıpkı Fourier dönüşümü (FD) gibi frekans dönüşümüdür. Ondan en önemli farkı ise hem kosinüs hem de sinüs bilgileri yerine sadece kosinüs bileşeni içermesidir. Dolayısıyla karmaşık değil gerçek bileşenler taşır. AKD yüksek iniltiye sahip bilgiyi, iniltisi düşük bilgiye dönüştürür. Algoritmanın yapısı Şekil 3.12.'de şematize edilmiştir.



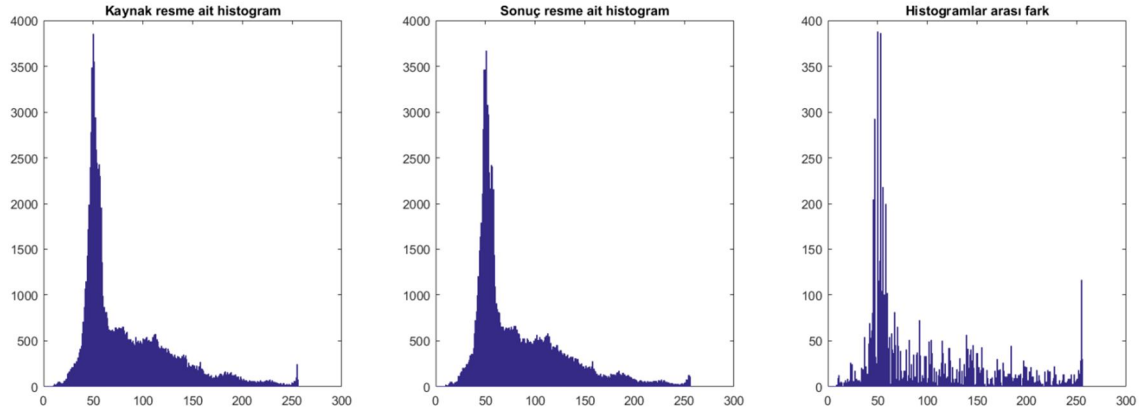
Şekil 3.12 F5 Algoritmasının Genel Yapısı

Şekil 3.13.'de örnek bir resim ve bu resmin içerisine F5 algoritmasıyla %80 sıkıştırma oranıyla veri gömülmüş hali görülmektedir. F5 algoritması, sıkıştırma oranı yüksek tutulduğu zaman sonucu insan gözü tarafından kolay algılanan bir algoritma değildir.



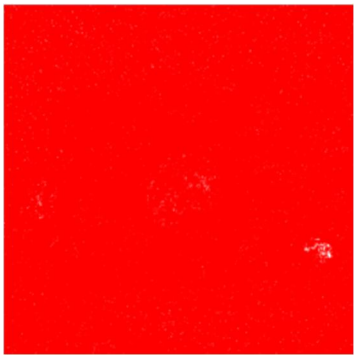
Şekil 3.13 (a) Kaynak Resim (b) İçerisine veri gömülmüş resim

Bununla birlikte Şekil 3.14.'de görüldüğü üzere bu algoritma tarafından üretilmiş sonuç resmin histogramı ile kaynak resmin histogramı arasında ciddi farklar vardır.



Şekil 3.14 Kaynak resim ve sonuç resim histogramları ve bu histogramlar arasındaki fark

Şekil 3.15'de Şekil 3.13'de yer alan resimler arasındaki farklar, aynı pikseller beyaz farklılar kırmızı olmak üzere gösterilmiştir.



Şekil 3.15 Şekil 3.13.'de yer alan resimler arasındaki farklar

Bu farklılıkların sıkıştırma oranından kaynaklanıp kaynaklanmadığını ölçmek için hiçbir veri gömme işlemi yapılmadan sadece F5 algoritması ile aynı resmin %100 ve %80 oranında sıkıştırılmış halleri ve aralarındaki farklar Şekil 3.16'da görülmektedir.



Şekil 3.16 (a) 100% ve (b) %80 kalite ile kaydedilmiş resimler (c) aralarındaki fark

Şekil 3.16. ve Şekil 3.15. karşılaştırıldığında görüldüğü üzere veri gömüldüğü zaman bozulma daha fazla yaşanmaktadır fakat herhangi bir farklılığın gömülmüş veriden mi yoksa sıkıştırma oranından mı kaynaklandığı tam olarak çözülemez.

3.2. Önerilen Metod Kaos Tabanlı Gömme

Klasik en önemsiz bite gömme algoritması insan gözüyle algılanamayacak derecede fark meydana getirse de metin bitlerini sırayla yerleştirdiği için güvensiz kabul edilir. Sözde rasgele sayı üreteçleri en önemsiz bite veri gömme algoritması bu açıdan daha güvenilir olsa da bu algoritmalar kaba kuvvet yöntemlerine karşı dayanıksız ve kolayca çözülebilir niteliktedir. Çünkü bu üreteçler aslında sahte rastgele sayı ürettikleri bilinmektedir (L'Ecuyer, 2012). Üreteçlerde kullanılan yöntemler bilindiği için bu

sayılar öngörülebilir. Sayılan sebeplerden ötürü bu tezde indislerin kaotik rastgele sayı üreteçlerine göre belirlendiği, en önemsiz bite gömme anlayışına dayalı yeni bir algoritma önerilmiştir. Kaotik rastgele sayı üreteçleri gelişigüzel ve tahmin edilemez şekilde sayı üretmektedirler. Bu algoritmayı tanıtmadan önce kaos kavramı ve kaotik rastgele sayı üreteçleri üzerinde durulacaktır.

3.2.1. Kaos kavramı

Kaos teorisi başlangıç koşullarına büyük ölçüde duyarlı olan dinamik sistemlerin davranışlarını inceleyen bir çalışma alanıdır. Kaotik sistemlerin zamana bağlı değişimleri uzun vadede tahmin edilememektedir. Ancak bu değişim deterministiktir ve başlangıç koşullarıyla belirlenir. Kaotik sistemlerin bu belirlenebilir doğal yapıları, bu sistemleri tahmin edilebilir yapmamaktadır. Ayrık kaotik bir sistem denklem (3.3)'deki gibi iteratif bir f fonksiyonuyla (kaotik harita) ve I durum uzayıyla tanımlanır (Keleş, 2012).

$$x_{n+1}=f(x_n) \tag{3.3}$$

Bu çalışmada rastgele sayı üretmek amacıyla denklem (3.4) ile ifade edilen bir boyutlu kaotik harita fark denklemi kullanılmıştır.

$$x_{n+1}=f(x_n,r),n=0,1,2,3\ldots \tag{3.4}$$

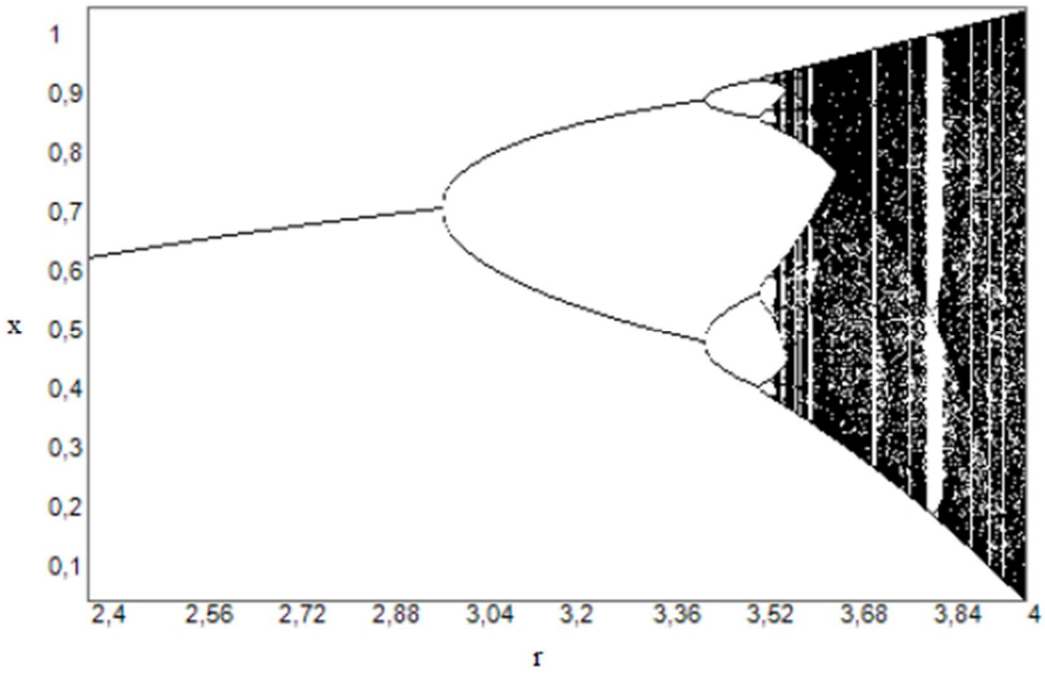
Bu formülde r sistem kontrol parametresi x ise durum değişkenini temsil etmektedir. Sistemin x_{n+1} değeri x_n değerine bağımlıdır. Logistic harita (3.5) belirli şartlarda değişkenlik gösteren 1 boyutlu bir polinomal haritadır.

$$x_{n+1}=r\, x_n(1-x_n) \tag{3.5}$$

Kaotik yapılı bir dinamik sistem, topolojik sistem, topolojik geçişgenlik, periyodik noktaların yoğunluğu ve başlangıç koşullarına duyarlılık niteliklerine sahiptir. Literatürde pek çok kaotik harita vardır. Bunlar arasında tek boyutlulara Tent map (Collet ve Eckmann, 2009), Circle Map (Devaney ve Eckmann, 1987), Complex quadratic map (Aho ve Sloane, 1973) ve iki boyutlu haritalar Hénon map (Hénon,

1976), Bakers Map (Balazs ve Voros, 1987), Arnold's cat map (Arnol′d ve Avez, 1968) örnek olarak verilebilir.

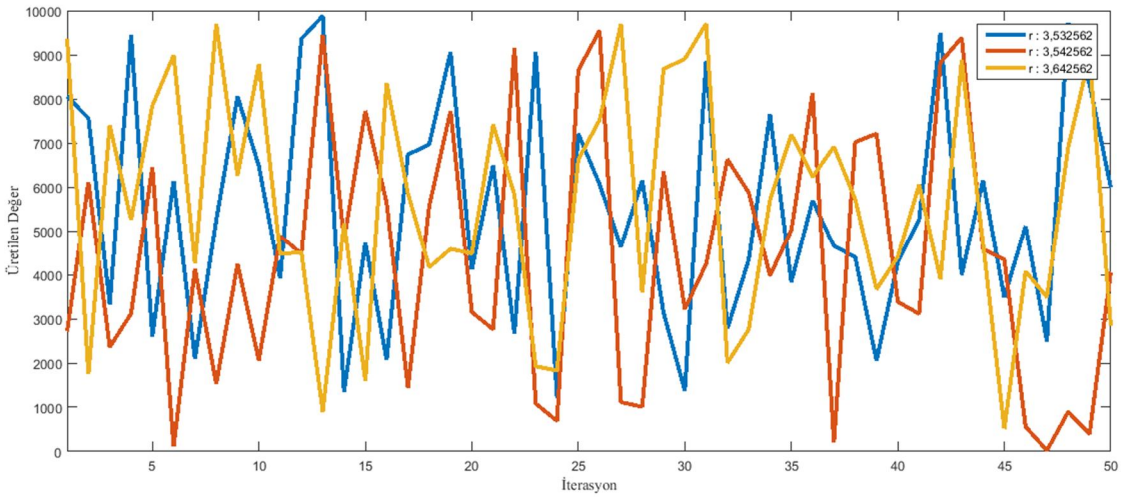
Dallanma, dinamik sistemlerde parametrelerin değişimiyle beraber sistem davranışında meydana gelen değişmeyle ölçülür. Bazı aralıklarda [0 – 3] dallanma kararlı bir yapı gösterebilir. Bazı aralıklarda [3,5 – 4.0] ise kararsız bir yapı gösterir ve olağan seyrinden çıkarak tahmin edilemez davranışlar gösterir. Bu aralıklar veri güvenliği açısından en değerli aralıklardır. Şekil 3.17.'de logistic haritanın dallanma diyagramı görülmektedir.



.

Şekil 3.17 Logistic haritanın dallanma grafiği

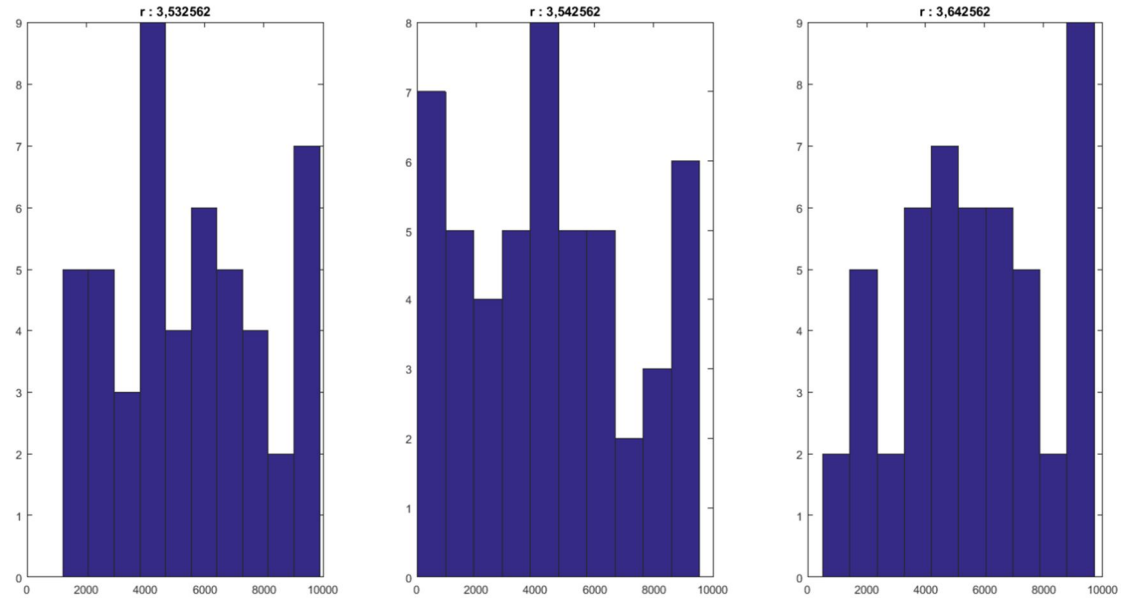
Dallanmanın tahmin edilemez ve küçük değişimlerden büyük oranda etkilenebilir olması veri güvenliğini sağlamada önemli avantajları beraberinde getirmektedir. Şekil 3.18.'de yüzde birlik ve onda birlik farklılık gösteren 3 ayrı r parametresine göre üretilen ellişer sayının grafikleri yer almaktadır.



Şekil 3.18 Parametrelerdeki yüzde bir ve onda birlik değişimlerle üretilen sayıların kıyaslanması

Görüldüğü üzere çok küçük parametre değişimleri bile üretilen sayılar üzerinde ciddi sapmalara yol açmakta ve tahmin edilmeyi güçleştirmektedir.

Şekil 3.19.'da ise Şekil 3.18.'deki sayıların histogramları gösterilmektedir. Üretilen sayıların histogramları incelendiğinde ise farklı dağılım gösterdikleri ve birbirlerine benzemedikleri dikkatlerden kaçmayacaktır.



Şekil 3.19 Parametrelerdeki yüzde bir ve onda birlik değişimlerle üretilen sayıların histogramları

3.2.2. Geliştirilen yöntem

Geliştirilen yöntem iki temel esasa dayanmaktadır. Bunlardan ilki verinin gömüleceği bitlerin kaotik olarak belirlenmesi diğeri de verinin hangi renk kanalına gömüldüğünün tam olarak belirlenememesidir. Bu işlemleri icra edecek algoritma aşağıda görülmektedir.

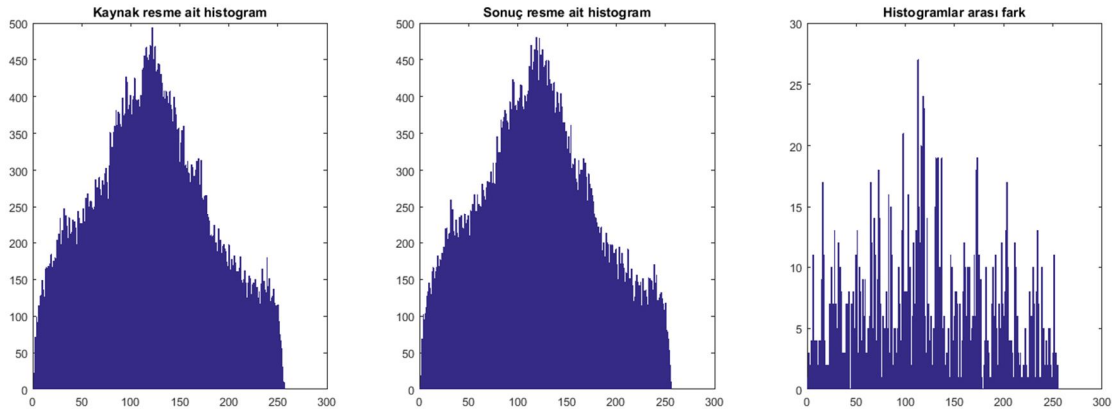
Adım 1: Gömülecek metni ve kaynak resmi al. Kullanıcıdan r ve x parametrelerini girmesini iste.
Adım 2: Metni bitlerine ayır (ikili veriye çevir)
Adım 3: Metindeki bitlerin sayısı kadar kaotik rasgele sayı üret. Bu sayıları diziye aktar (dizi).
Adım 4: Birden resimdeki piksellerin sayısı kadar ardışık sayıyı dinamik diziye aktar (sıra)
Adım 5: Bütün bitler gömülene kadar Adım 6-8'i uygula
Adım 6: Kaotik dizinin sıradaki elemanını al (k). Sıra dizisinde bu indisteki elemanı indis = sıra(k) indis olarak kabul et ve sıra dizisinden çıkar. İndis numaralı piksel üzerinde veri gömülecek olarak belirle.
Adım 7: Eğer indis 3'e bölünüyorsa mavi kanala, 2 ye bölünüyorsa yeşil kanala, ikisine de bölünmüyorsa kırmızı kanala veriyi göm.
Adım 8: Veri gömülen kanal haricindeki kanalların son piksellerini boz. (0 ise 1, 1 ise 0 yap)
Adım 9: İşlemi sonlandır



Şekil 3.20 (a) Kaynak Resim (b) İçerisine veri gömülmüş resim (c) İki resim arasındaki farklı bölgeler.

Şekil 3.20.'de örnek bir resim üzerinde veri gömme işlemi sonucu oluşan fark görülmektedir.

Bu resimler ait histogramlar ve histogram değerleri arasındaki fark Şekil 3.21.'de yer almaktadır.



Şekil 3.21 Resimlere ait histogramlar ve bu histogram değerleri arasındaki fark

Gömülen veririn ortaya çıkarmak için uygulanması gereken algoritma aşağıda yer almaktadır.

Adım 1:	İçerisine veri gömüş resmi al. Kullanıcıdan a ve x parametrelerini girmesini iste. Metin uzunluğunu kullanıcıdan al.
Adım 2:	Metindeki bitlerin sayısı kadar kaotik rasgele sayı üret. Bu sayıları diziye aktar (dizi).
Adım 3:	Birden resimdeki piksellerin sayısı kadar ardışık sayıyı dinamik diziye aktar (sıra)
Adım 4:	Bütün bitler elde edilene kadar Adım 5-6'yı uygula
Adım 5:	Kaotik dizinin sıradaki elemanını al (k). Sıra dizisinde bu indisteki elemanı indis = sıra(k) indis olarak kabul et ve sıra dizisinden çıkar. indis numaralı piksel üzerinde veri gömülecek olarak belirle.
Adım 6:	Eğer indis 3'e bölünüyorsa mavi kanaldan, 2 ye bölünüyorsa yeşil kanaldan, ikisine de bölünmüyorsa kırmızı kanaldan biti al ve bit dizisine ekle.
Adım 7:	Bit dizisini seçilen kodlama yöntemine göre metne dönüştür
Adım 8:	İşlemi sonlandır.

3.3. Test Metodları

Steganografi alanında veriyi kaynak resim içerisine gömmek kadar resmin istenmeyen kişilerin eline geçtiğinde anlaşılamaması da önemlidir. Bu yüzden geliştirilen algoritmanın sağlamlığını test etmek amacıyla pek çok yöntem geliştirilmiştir. Bu yöntemleri içeren disipline genel olarak steganaliz adı verilmektedir. Bu bölümde bu tezdeki algoritmaları test etmek için kullanılan beş adet steganaliz yöntemi hakkında bilgi verilmektedir.

3.3.1. Doruk işaretten gürültü oranına (peak signal to noise ratio) testi

Bu test iki sayı dizi arasındaki farkı ölçmede sıklıkla kullanılan ortalama karesel hataya dayalı bir testtir. M x N boyutunda kaynak resim x ve veri gömülmüş resim y olmak üzere ortalama karesel hata (3.6) formülüyle hesaplanır.

$$\boldsymbol{MSE} = \left(\frac{\mathbf{1}}{\boldsymbol{M \times N}}\right) \sum_{\boldsymbol{i=1}}^{\boldsymbol{M}} \sum_{\boldsymbol{j=1}}^{\boldsymbol{N}} (\boldsymbol{x_{ij} - y_{ij}})^{\mathbf{2}} \tag{3.6}$$

Buna göre kaynak resim ve içerisine veri gömülmüş resim piksel piksel taranır ve aynı konumdaki iki piksel arasındaki farkın karesi alınarak toplanır. Karelerin toplanmasının sebebi pozitif ve negatif hataların toplandığında sıfır vermesini engellemektir. Bu formülden hareketle PSNR (3.7) formülüyle hesaplanır.

$$\text{PSNR}=10 \times \log\left(\frac{255^2}{\text{MSE}}\right) (\text{dB}) \tag{3.7}$$

3.3.2. Ki-kare testi

Ki Kare testi literatürde X^2 testi olarak da bilinir. Bu algoritmanın temeli en önemsiz bit içinde değer çiftlerinin istatistiki çözümlemesine dayanır (Çimen ve ark., 2008). Westfeld ve Pfitzmann tarafından 1999 yılında önerilen bu yöntem en sık kullanılan stegenaliz yöntemlerinden birisidir. Bu yöntemin adımları aşağıda verilmiştir (Şahin, 2007).

Adım 1: k kategoriler ve gözlemlerden oluşan rasgele bir örnekleme olduğunu varsayılmakta. Her gözlem sadece ve sadece bir kategoriye düşmektedir. Şüpheli bilginin PoVs'lerinin tek değerlerine önem verilmektedir.

Adım 2: Düz bir şekilde dağılmış bir mesajın gizlenmesinden sonra, i kategoride teorik olarak beklenen frekansı böyledir:

$$n_i^* = \frac{\left|\left\{\text{renk}\middle|(\text{renk})'\text{in sıralanmış indeksi} \in \{2i,2i+1\}\right\}\right|}{2} \tag{3.8}$$

Adım 3: Rasgele örneklemede, ölçülen vuku bulma frekansı aşağıdaki gibidir.

$$n_i = \left|\{\text{renk}\middle|(\text{renk})'\text{in sıralanmış indeksi}=2i\}\right| \tag{3.9}$$

Adım 4: χ 2 istatistiği ise k-1 bağımsızlık dereceleri ile şu şekilde hesaplanır:

$$\chi^2_{k-1} = \sum_{i=1}^{k} \frac{(n_i - n_i^*)^2}{n_i^*} \tag{3.10}$$

Adım 5: n ve *i n dağılımları eşit olduğu durumda, p mesaj gömme olasılığıdır. Bu olasılık yoğunluk fonksiyonun integrali alınarak hesaplanmaktadır (Γ , Euler'in gamma fonksiyonudur):

$$p=1- \frac{1}{2^{\frac{k-1}{2}}\Gamma\left(\frac{k-1}{2}\right)} \int_0^{\gamma^2_{k-1}} e^{-\frac{x^2}{2}} x^{\frac{k-1}{2}-1} dx \tag{3.11}$$

3.3.3. Kapasite testi

Kapasite testi, bir resmin içerisine azami kaç bit veri gömülebileceğini belirleyen testtir. Doğrudan güvenilirlik testi olmayıp çoğu zaman verim ölçmede kullanılır. Kesin bir formüle sahip olmayıp her bir metot için değişkenlik arz eder. En temel yöntem olarak en önemsiz bite gömme metodunda kapasite piksel sayısı kadardır. Eğer resim gri düzeyli değil de renkli ise her renk kanalına ayrı ayrı veri gömülebileceği için kapasite üç katına çıkabilir. Sözde rastgele sayı üreteçleri ile yapılan veri gömme işleminde de durum farklı olmayacaktır. Ezstego yönteminde resim renkli resim olduğu halde palet tabanlıdır dolayısıyla ancak piksel sayısı kadar gömme işlemi uygulanabilir. F5 algoritması için kapasite JPEG resimler için %13'tür. Önerilen algoritma ise renkli resimler üzerinde geçerli olmasına karşın üç kanalı da kullandığı için piksel sayısı kadar bit içerebilir.

3.3.4. Histogram testi

Histogram bir veri dizisindeki her bir elemanın sıklığını gösteren ölçüttür. İstatistiğin çok sık kullanılan bu yöntemi bilgisayar görmesi, görüntü işleme disiplinlerinde olduğu gibi dijital resim güvenliği alanında da sıklıkla başvurulan bir yöntemdir. Histogramı elde etmek için başlangıç olarak her bir elemanı sıfır olan 256 elemanlı bir dizi tanımlanır (resmin kaç bit ile ifade edildiğine göre bu sayı değişebilir). Sayısal resmin her bir pikseli taranır dizinin piksel değerine karşılık düşen elemanın değeri bir artırılır. Şekil 3.22'de örnek bir resim ve resmin gri düzeyine ait histogram grafiği yer almaktadır.



Şekil 3.22 (a) örnek resim (b) resmin gri düzeyli hali (c) gri düzeyli resme ait histogram grafiği

Histogramda herhangi bir renk değerine sahip kaç pikselin bulunduğu yer aldığı için bu piksel değerlerindeki bir değişiklik histogram farkıyla rahatlıkla anlaşılabilir.

3.3.5. Yapısal Benzerlik İndeksi

Yapısal benzerlik testi iki resim arasındaki benzerliği ölçen temel metotlardan birisidir. Benzer şekilde farklı istatistiksel yöntemlerle geliştirilen bu yaklaşım 2004 yılında Wang ve arkadaşları tarafından Ortalama Yapısal Benzerlik (Mean Structural Similarity Index: M–SSIM) adıyla ortaya konulmuştur. Bu yaklaşımda ise [0…1] değer aralığında bir MSSIM değeri üretilir (Yalman, 2010). Yapısal benzerlik indeksi; $\mu_x, \mu_y, \sigma_x, \sigma_y$ ve σ_{xy} sırasıyla yerel ortalamalar, standart sapmalar ve çapraz kovaryanslar olmak üzere ve

$$l(x,y)=\frac{2\mu_x\mu_y+c_1}{\mu_x^2+\mu_y^2+c_1} \tag{3.12}$$

$$c(x,y)=\frac{2\,\sigma_x\sigma_y+c_2}{\sigma_x^2+\sigma_y^2+c_2} \tag{3.13}$$

$$s(x,y)=\frac{\sigma_{xy}+c_3}{\sigma_x\sigma_y+c_3} \tag{3.14}$$

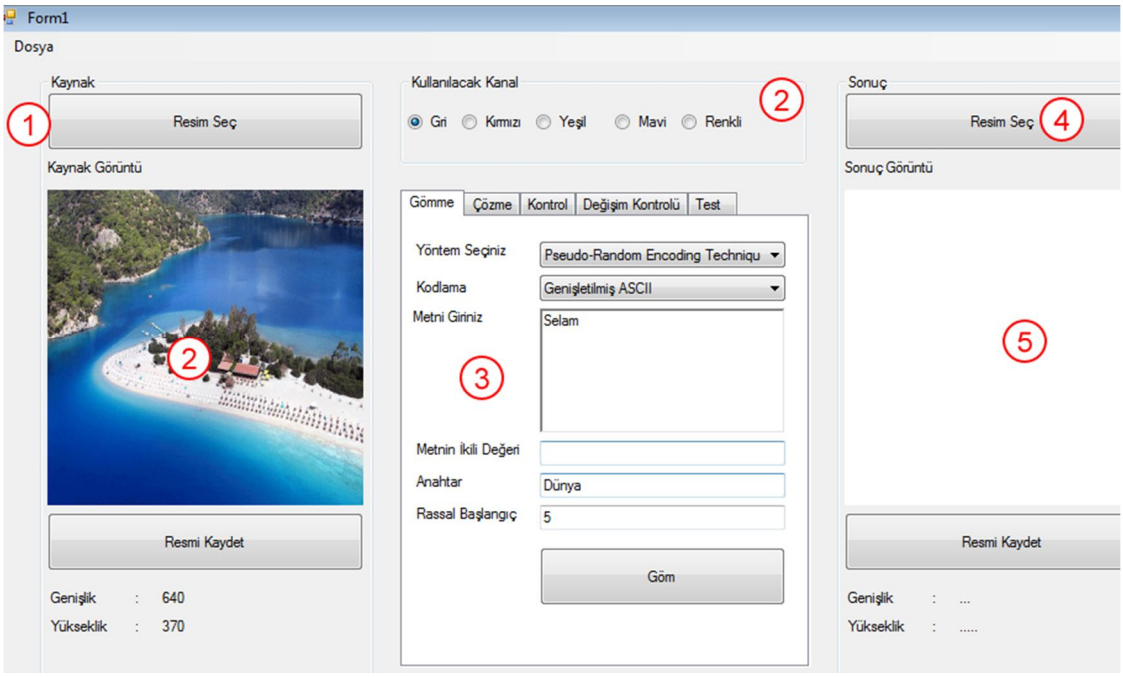
olmak üzere;

$$SSIM(x,y)=[\,l(x,y)\,]^{\alpha}+[\,c(x,y)\,]^{\beta}+s(x,y)^{\gamma} \tag{3.15}$$

formülüyle hesaplanır.

3.4. Geliştirilen Yazılım

Tez bünyesinde hem literatürde mevcut yöntemleri hem de geliştirilen algoritmayı uygulamak amacıyla C# programlama diliyle .Net Framework 4.0 derleyicisiyle Windows masaüstü uygulaması geliştirilmiştir. Şekil 3.23'de yazılımın ara yüzü ve temel bölgeleri görülmektedir.



Şekil 3.23 Veri saklama ara yüzü

Birinci bölge resim seçme bölgesidir. Burada içerisine veri gömülmesi gereken resim dosyadan seçilir. Sadece resim formatı taşıyan dosyalar seçilebilir.

İkinci bölge kanal seçme bölgesidir. Kullanıcı, burada veri gömme işlemi hangi kanala yapmak istiyorsa o kanalı seçer. Ezstego, F5 gibi yöntemlerde kanal seçeneği otomatik olarak devre dışı kalır. Çünkü Ezstego yönteminde resim renkli de olsa veri gömme işlemi paletten elde edilen matris üzerinde icra edilmektedir. Benzer şekil F5 algoritmasında da sıkıştırma işlemi kanaldan bağımsız olacağı için farklı kanallara gerek kalmamaktadır.

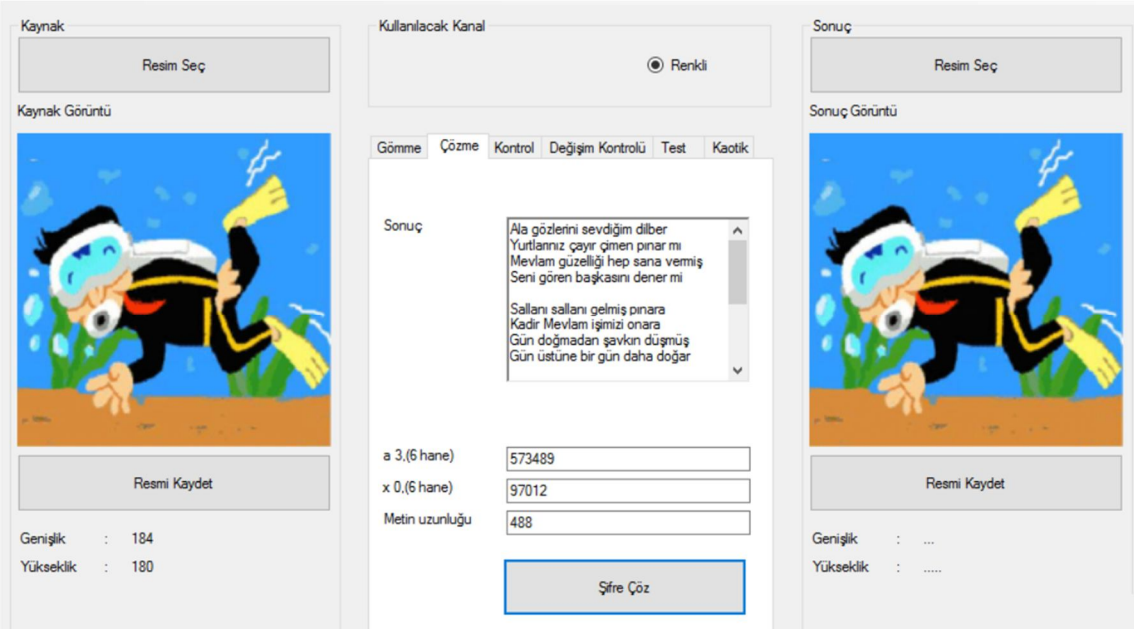
Üçüncü bölge parametre ekranıdır. Kullanıcı gizlemek istediği metni 'Metni Giriniz' yazan kutucuk içerisine girer ardından veri kodlama formatını seçer.

Verinin ikili veriye dönüştürülmesi için varsayılan kodlama biçimi Türkçe karakterleri de kapsayan Genişletilmiş ASCII kodlamasıdır. Metin ikili veriye çevrildikten sonra ikili değeri Metnin ikili değeri kutucuğunda yer alır. Kullanıcı göm

butonuna bastığında istediği metin resim içerisine gömülmüş olur. Yazılım ayrıca .NET platformunun destek verdiği ASCII, UTF32, UTF7, UTF8 ve Unicode kodlama standartlarını da veri gömmede kullanabilmektedir. Ancak bunlardan bazıları Türkçe karakterleri içermediği için bazıları da metin harici verileri de kapsayacak denli geniş olup kapasiteyi daraltacağı için tercih edilmemiştir.

Dördüncü bölge sonuç bölgesi için resim seçme alanıdır. Sadece istenen resim seçilip test edilebilsin diye geliştirilmiş bir alandır.

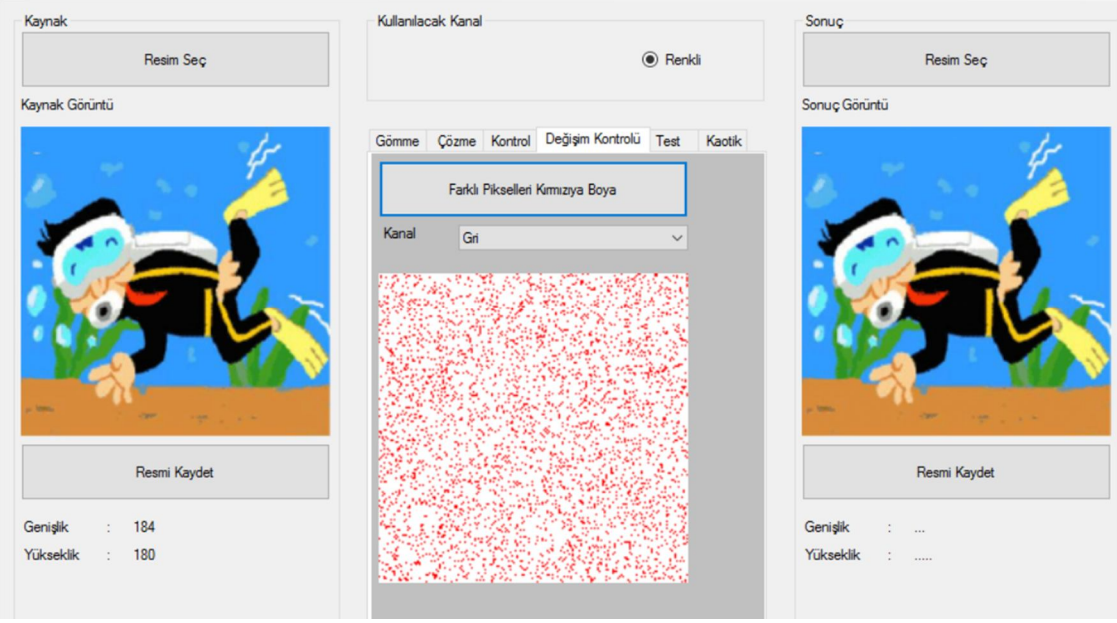
Beşinci bölge sonuç resmin alanıdır. İçerisine veri saklanmış resim bu alanda gösterilir. Yazılımda yer alan başka bir modül de gizli veriyi ortaya çıkarma başka bir deyişle veri çözme modülüdür. Bu modül Şekil 3.24'de görülmektedir.



Şekil 3.24 Sır Çözme Ara Yüzü

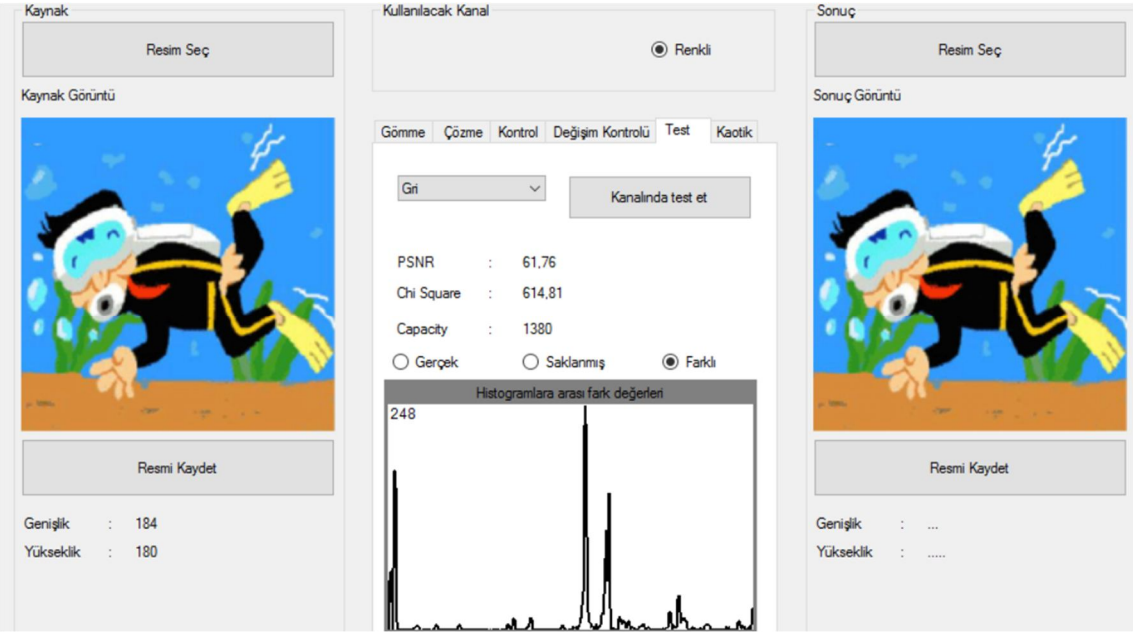
Burada gömme modülündeki kodlama tekniği ve kodlama biçimine göre saklanmış verinin ortaya çıkarılması gerçekleşir. Steganografi algoritmasına göre gerekli olan parametreler buraya girildikten sonra 'Şifre Çöz' butonuna tıklanır ve Sonuç Görüntü bölümünde yer alan resim esas alınarak gömülmüş olan metin ortaya çıkarılır.

Yazılımın bundan sonraki ara yüzleri kontrol amacıyla üretilmiş olan ara yüzlerdir. Şekil 3.25'de bu ara yüzlerden Değişim Kontrolü ara yüzü yer almaktadır.



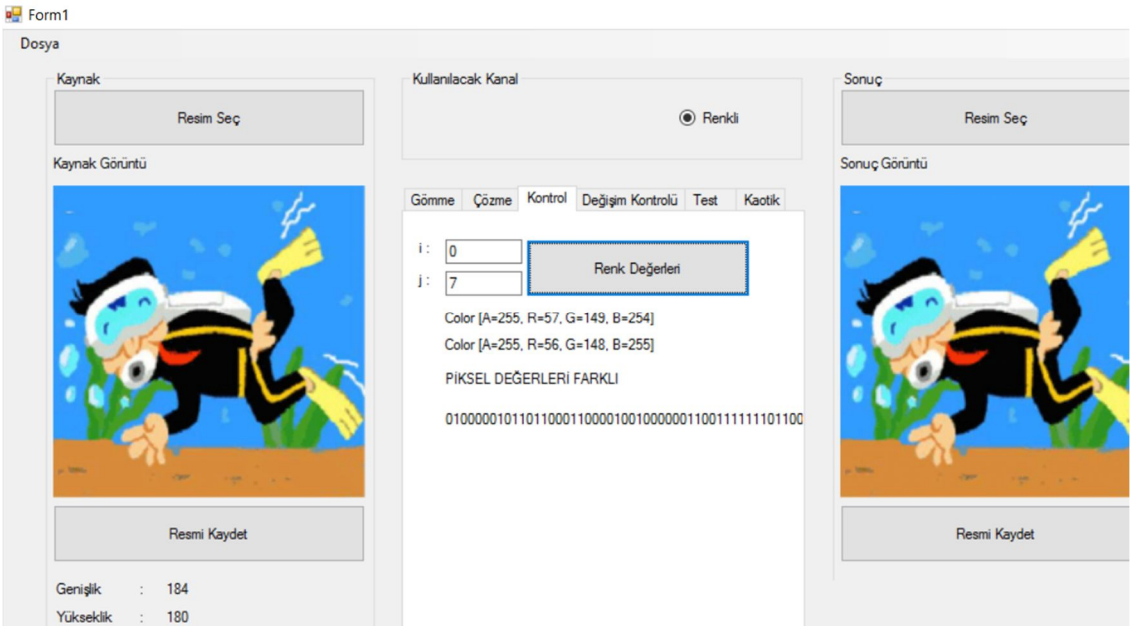
Şekil 3.25 Değişim kontrolü ara yüzü

Bu ara yüz kaynak resim ve içerisinde veri gizli olan resim yani sonuç resim arasındaki farklı pikselleri göstermek için tasarlanmış bir ara yüzdür. Aşağıdaki kanal seçme kontrolünden hangi renk kanalına göre karşılaştırma yapılmak isteniyorsa seçilir. Ardından da Farklı Pikselleri kırmızıya boya butonuna tıklanır. Yazılım seçilen kanala göre iki resimdeki pikselleri tek tek karşılaştıracak ve fark bulursa kırmızı bulamazsa beyaz nokta ile gösterecektir. Yazılımın bir diğer ara yüzü Test ara yüzüdür. Bu ara yüzde 3. Bölüm'de bahsedilen test algoritmalarına göre kaynak ve sonuç resimler karşılaştırılmaktadır. Bu ara yüz Şekil 3.26'da yer almaktadır.



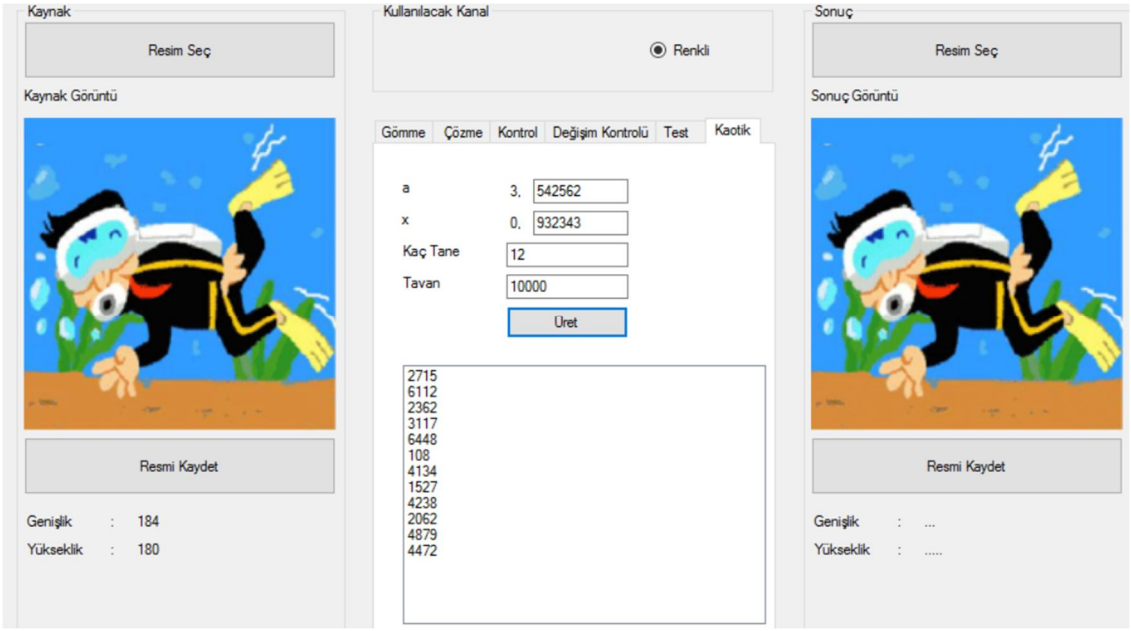
Şekil 3.26 Test ara yüzü

Bu ara yüzde bir önceki ara yüzde olduğu gibi önce hangi kanala göre test edilmek isteniyorsa o kanal seçilir. Sonrada Kanalında Test Et butonu tıklanır ve test sonuçları ekranda yer alır. Sayfa altındaki grafikte ise resmin seçilen kanala ait histogram değerleri yer almaktadır. İster kaynak isterse veri saklanmış resme ait histogram değerleri görülebilir. Farklı butonuna tıklanırsa iki histogram değeri arası farkın mutlak değeri grafik ekranında yer alır. Grafiğin sol üstünde ise grafikte yer alan en büyük değer yer alır. Yazılımın bir diğer ara yüzü ise yine test amacıyla geliştirilmiş olan Kontrol ara yüzüdür. Bu ara yüz Şekil 3.27'de görülmektedir.



Şekil 3.27 Kontrol Ara yüzü

Kontrol ara yüzünde kullanıcı kontrol etmek istediği piksel indislerini kutucuklara yazmakta ardından Renk Değerleri butonuna tıklamaktadır. Buton altında yer alan etiketleri sırasıyla kaynak resimdeki ve sonuç resimdeki ilgili piksele ait renk değerleri (kırmızı, yeşil ve mavi) ile doldurmaktadır. Eğer renk değerleri aynı ise değer aynı farklı ise PİKSEL DEĞERLERİ FARKLI ikazı gelmektedir. Yazılımın son ara yüzü ise Logistic haritayı test için geliştirilen Kaotik ara yüzüdür. Kaotik ara yüzü Şekil 3.28'de yer almaktadır.



Şekil 3.28 Kaotik ara yüzü

Bu ara yüzde Logistic harita parametreleri girilir. Kaç tane sayı üretilmek isteniyorsa ve bu sayıların azami değerleri tavan kutucuğuna girilerek üret butonuna tıklanır. İstenen miktarda rastgele sayı üretilerek listede yer alır.

4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA

Geliştirilen algoritmanın diğer metotlarla farkını ölçmek için 10 adet resim kullanılmıştır. Her bir resim palet tabanlı test yöntemi için (Ezstego) 256 renk değerine sahip .gif formatına, F5 algoritması için de gif'den jpg formatına dönüştürülmüştür. Diğer yöntemler renkli resimler üzerinde işlem yapabildiği için o yöntemlerde yapılan testler jpg formatındaki resimler üzerinde test edilmiştir.

Kaotik üreteçleri tam anlamıyla test edebilen standart bir yöntem mevcut değildir. Geliştirilen yöntemlerin daha çok istatistiki üreteçler üzerinde test için uygun olduğu bilinmektedir. Fakat yine de mevcut testler üzerinde geliştirmiş olduğumuz kaos tabanlı gömme algoritmasının testleri diğer dört metot ile birlikte gerçekleştirilecektir. Şekil 4.1.'de test amacıyla kullanılan 10 adet resim yer almaktadır.



1.Pullar



2.Fethiye



3.Kaplan



4.Köpek



5.Lena



6.Fener



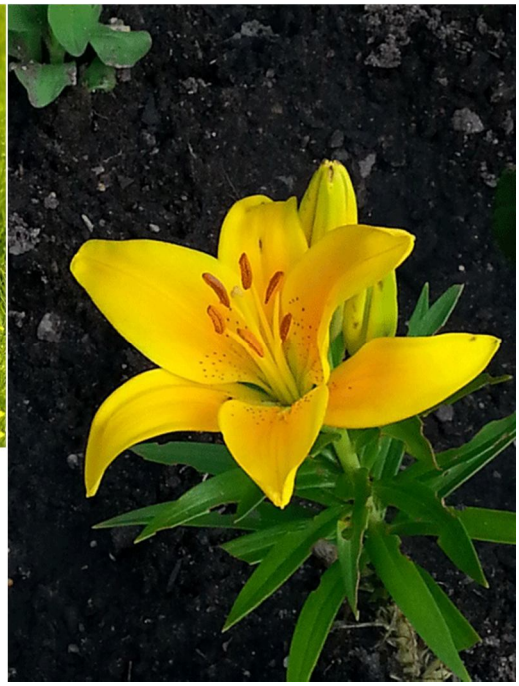
7. Soğan



8.Armut



9. Ceylan



10. Çiçek

Şekil 4.1. Test Amacıyla Kullanılan Resimler.

Resimlere saklamak için sözleri Karacaoğlan'a ait olan aşağıdaki şiir seçilmiştir.

Ala gözlerini sevdiğim dilber
Yurtlarınız çayır çimen pınar mı
Mevlam güzelliği hep sana vermiş
Seni gören başkasını dener mi
Sallanı sallanı gelmiş pınara
Kadir Mevlam işimizi onara
Gün doğmadan şavkın düşmüş
Gün üstüne bir gün daha doğar
Kırmızı gülden rengin almışsın
Güzellikte kemalini bulmuşsun
Sallanı sallanı suya gelmişsin
Güzel senin ziyaretin pınar mı
Karac'oğlan der ki ermediler mi
Tomurcuk güllerin dermediler mi
Seni sevdiğine vermediler mi
Aşıkın ağlatan güzel güler mi

Çizelge 4.1. Testte kullanılan yöntemlere ait parametreler

Yöntem Adı	**Parametreleri**
En Önemsiz bite gömme	YOK
Sözde Rasgele En Önemsiz Bite Gömme	Rastsal Başlangıç: 3
Ez Stego	YOK
F5	Kalite: 80
Önerilen Yöntem	a: 3,573489 x: 0,970120

Bu bölümün bundan sonraki kısmında yukarıdaki parametreler girilerek yapılan deneylerin ardından elde edilen sonuçlar yer almaktadır.

Çizelge 4.2. Doruk İşaretten Gürültü Oranına Testi Sonuçları

Resim No	**Resim Boyu**	**En Önemsiz Bit**	**Sözde Rastgele**	**EzStego**	**F5**	**Geliştirilen Yöntem**
1	518 x 391	74,52	72,36	63,96	36	68,97
2	640 x 370	75,25	73,81	61,68	31,77	69,88
3	650 x 644	77,78	76,33	59,04	37,28	72,33
4	1632 x 1224	84,15	82,7	71,74	38,29	78,83
5	512 x 512	75,8	74,02	56,59	36,44	70,14
6	480 x 640	76,17	74,66	64,13	34,99	70,92
7	198 x 135	65,77	64,43	61,89	33,33	60,15
8	732 x 486	76,97	75,38	59,13	36,44	71,34
9	600 x 450	75,75	74,63	67,48	32,01	70,21
10	500 x 667	76,38	75,57	65,24	36,17	70,97

Doruk İşaretten Gürültü Oranına Testi Sonuçları testi itibariyle yöntemlerin birbirlerine yakın sonuçlar verildiği gözlemlenmiştir. F5 algoritması sonuçlarının diğer yöntemlere nazaran daha iyi sonuçlar verdiği müşahede edilmiştir. Yukarıda da belirtildiği üzere bu test ortalama kare hatasına dayalı bir testtir.

$$MSE=\left(\frac{1}{N \times M}\right) \sum_{i=1}^{M} \sum_{j=1}^{N} (x_{ij}-y_{ij})^2 \qquad (4.1)$$

Bu test formülden de anlaşılacağı üzere doğrudan iki resimdeki bütün piksellerin farklarının toplamlarıyla doğru orantılıdır. Dolayısıyla birbirlerine yakın resimler için düşük, birbirlerinden uzak resimler için yüksek değerler üretmektedir. Buradan hareketle en düşük değeri üreten algoritmaların rakiplerine nazaran daha başarılı oldukları iddia edilebilir. Diğer bir ifadeyle insanın test etme mantalitesine en yakın bir biçimde farklılıkları algılamaya çalışmaktadır. Yani insan tarafından gürültü olarak anlaşılabilecek görüntüler -ki EzStego algoritması sonucunda bolca ortaya çıkmaktadır-,

kolayca ayırt edilebilmektedir. Resme gömülen veri arttıkça yani değiştirilen bitler artıkça bu test sonuçlarının da artacağı muhakkaktır. Diğer bir deyişle gömülen bit sayısını artmasına rağmen bu testin sonuçları daha düşük bir ivme ile artıyorsa başarıdan söz edilebilir. Bu itibarla geliştirilen yöntem kendisine benzer mantaliteye sahip sözde rastgele ve en önemsiz bite gömme algoritmalarına kıyasla daha iyi sonuçlar vermiştir.

Çizelge 4.3 Ki-Kare Testi Deney Sonuçları

Resim No	Resim Boyu	En Önemsiz Bit	Sözde Rastgele	EzStego	F5	Geliştirilen Yöntem
1	518 x 391	1,49	5,28	182,77	-	7,64
2	640 x 370	1,49	3,27	87,9	-	7,94
3	650 x 644	0,74	2,3	33,53	-	3,98
4	1632 x 1224	0,17	0,41	5,54	-	0,84
5	512 x 512	1,32	2,3	43,59	-	3,95
6	480 x 640	2,34	4,12	92,33	-	8,18
7	198 x 135	8,55	28,26	341,26	-	58,13
8	732 x 486	1,1	2,21	18,97	-	4,32
9	600 x 450	0,83	2,3	27,76	-	6,9
10	500 x 667	1	2,12	117,93	-	5,66

Ki kare testi açısından değerlendirecek olursak, literatürde F5 algoritmasının ki kare testine karşı dayanıklı olduğu belirtilmiştir (Westfeld, 2001). Dolayısıyla en başarılı algoritmanın F5 algoritması olduğu görülebilir. EzStego algoritmasının bu alanda diğerlerinden verimsiz olduğu görülmektedir. En önemsiz bite gömme, sözde rastgele en önemsiz bite gömme ve önerilen yöntem F5 ve Ezstego metotlarına kıyasla daha başarılıdır. Literatürde F5 algoritmasının Ki kare testine karşı dayanıklı olduğu kaydedildiği için bu algoritma sonuçlarının test edilmesine gerek duyulmamıştır. Bu testte daha düşük sonuç veren algoritmaların daha güvenli sonuçlar ürettiği öne sürülebilir.

Çizelge 4.4 Kapasite Testi Sonuçları

Resim No	Resim Boyu	En Önemsiz Bit	Sözde Rastgele	EzStego	F5	Geliştirilen Yöntem
1	518 x 391	**75951**	**75951**	25317	9874	25317
2	640 x 370	**88800**	**88800**	29600	11544	29600
3	650 x 644	**156975**	**156975**	52325	20407	52325
4	1632 x 1224	**749088**	**749088**	249696	97381	249696
5	512 x 512	**98304**	**98304**	32768	12780	32768
6	480 x 640	**115200**	**115200**	38400	14976	38400
7	198 x 135	**10023**	**10023**	3341	1303	3341
8	732 x 486	**133407**	**133407**	44469	17343	44469
9	600 x 450	**101250**	**101250**	33750	13163	33750
10	500 x 667	**125062**	**125062**	41687	16258	41687

Kapasite testi sonucunda daha iyi sonuç veren yani daha yüksek kapasiteye sahip sonuçlar kalın rakamlarla gösterilmiştir. Bu sonuçları incelendiği zaman görülmektedir ki en önemsiz bite gömme ve sözde rastgele en önemsiz bite gömme algoritmaları diğer yöntemlere göre daha başarılıdır. O ikisini önerilen algoritma ve EzStego algoritması takip etmektedir. F5 algoritması ise bu açıdan en verimsiz algoritma kabul edilebilir. Tabloda verilen değerler ilgili yöntemde resim içerisine gömülebilecek azami bit sayısını vermektedir. Dolayısıyla daha yüksek değer taşıyan test sonuçlarına sahip algoritmaların diğerlerine nazaran daha iyi oldukları söylenebilir. Elbette daha çok ya da az gömmek veri güvenliği açısından değer taşımaz. Sadece taşınabilecek saklanan verinin çokluğu açısından performans ölçütüdür. Bu bakış açısından bakıldığında geliştirilen yöntemin düşük kapasiteye sahip olması olumsuz bir taraf olsa da doğrudan güvenlikle ilgili olmadığı için ciddi bir dezavantaj taşımaz.

Ayrıca içerisine veri gömülen ham resmin ve sonuç resmin istenmeyen kişilerin eline geçtiğini düşünelim. İki resim arası farkları tespit edilmesinin ardından metnin ortaya çıkarılması için gereken hesap karmaşıklıkları çizelge 4.5.'de verilmiştir. Kötü niyetli kişilerin veriyi elde etmek için daha fazla algoritma karmaşıklığıyla meşgul olmasının veri gömme açısından bir avantaj olduğu açıktır.

Çizelge 4.5 Yöntemler ve hesap karmaşıklıkları

Yöntem Adı	**Karmaşıklık**
En Önemsiz Bit	O(n)
Söz Rastgele En Önemsiz Bit	$O(n^2)$
Ez-Stego	$O(n^2)$
F5	$O(n^2)$
Geliştirilen Yöntem	$O(3*n^2)$

Tablo verileri incelendiğinde en önemsiz bite gömme yönteminin veri hırsızları açısından en dayanıksız yöntem olduğu görülmektedir. Geriye kalanlardan Geliştirilen yöntem haricindeki bütün yöntemlerin ise benzer algoritma karmaşıklığına sahip olduğu görülmektedir. Önerilen yöntemin ise sadece bir kanala veri gömüp kalan iki kanalı da anlamsız bir şekilde bozması sonucu diğer yöntemlerden önde olduğu görülmektedir.

Son olarak algoritmaları yapısal benzerlik indeksi açısından kıyaslanmıştır. Kıyaslama sonuçları Çizelge 4.6.'da yer almaktadır.

Çizelge 4.6 Yapısal Benzerlik İndeksi Testi

Resim No	**Resim Boyu**	**En Önemsiz Bit**	**Sözde Rastgele**	**EzStego**	**F5**	**Geliştirilen Yöntem**
1	518 x 391	**0,999**	**0,99999**	0,99976	0,98424	0,99999
2	640 x 370	**0,999**	**0,99999**	0,99825	0,88851	0,99996
3	650 x 644	**0,999**	**0,99999**	**0,999**	0,99853	**0,999**
4	1632 x 1224	**0,999**	**0,99999**	**0,999**	0,99896	**0,999**
5	512 x 512	**0,999**	**0,99999**	**0,999**	0,98832	0,99999
6	480 x 640	**0,999**	**0,99999**	**0,999**	0,98737	0,99999
7	198 x 135	**0,99997**	**0,99991**	0,96209	0,90177	0,9997
8	732 x 486	**0,999**	**0,99999**	**0,999**	0,98567	0,99999
9	600 x 450	**0,999**	**0,99999**	**0,999**	0,99357	0,99999
10	500 x 667	**0,999**	**0,99999**	**0,999**	0,98891	0,99999

İki resim birbirine benzedikçe yani diğer bir deyişle aradaki farklar azalınca yapısal benzerlik testi sonuçları 1'e yaklaşmaktadır. Test sonuçları incelendiğinde en önemsiz bite gömme algoritmalar ve geliştirilen yöntem bire yakın sonuçlar üretmektedir. Ez stego algoritması ise bu yöntemler kadar olmasa da bire yaklaşan güzel

sonuçlar üretebilmektedir. F5 algoritmasının ise çoğu zaman düşük değerler üreterek bu testten başarısız olarak ayrıldığı görülmektedir.

5. SONUÇLAR VE ÖNERİLER

5.1. Sonuçlar

Geliştirilen yöntem C# programlama diliyle .NET Framework 4.0 ortamında kodlanmış ve 10 adet örnek resim üzerinde test edilmiştir. Ayrıca literatürde farklı konumlar işgal eden dört adet yöntem kullanılarak ayrı resim seti üzerinde test işlemi gerçekleştirilmiş ve sonuçlar kıyaslamalı olarak sunulmuştur.

Kaotik rastgele sayı üreteçlerinin tahmin edilemez ve kestirilemez indisler üretmeleri sayesinde güçlü steganografik teknikler üretiminde kullanılabileceği görülmüştür.

Yukarıdaki test sonuçları dikkate alınarak temel dört yöntem ve önerilen beşinci yöntem ile birlikte yöntemlerin avantaj ve dezavantajları Çizelge 5.1'de verilmiştir.

Çizelge 5.1 Yöntemlerin Avantaj ve Dezavantajları

Yöntem	**Avantajları**	**Dezavantajları**
En Önemsiz Bite Gömme	Gözle tam olarak fark edilmez. Uygulaması kolaydır.	Sıralı veri gizleme yaptığı için rahatlıkla çözülebilir.
Sözde Rasgele En Önemsiz Bite Gömme	Gözle tam olarak fark edilmez. Rastgele gömme veri yaptığı için en önemsiz bite gömmeye nazaran daha güvenlidir.	Rastgele sayı üreteci güvensizdir.
EzStego	Palet tabanlı gizleme işlemi gerçekleştirdiği için doğrudan piksellere bakılarak çözülemez.	Sadece palet tabanlı resimleri destekleyen gif, png gibi formatlar üzerinde çalışır. Gizlenen veri boyutu arttığında sonuç üzerinde gözle görülür değişiklikler yaşanır.
F5 Algoritması	X2 testine karşı dayanaklıdır.	Sadece jpeg gibi sıkıştırılabilen formatlar üzerinde uygulanabilir.
Önerilen Yöntem	Kaotik sayı üreteçleri kullandığı için kolay kolay kırılamaz. Üç kanala da veri saklandığı için verinin hangi kanallar üzerinde olduğu tahmin edilemez.	Yavaş çalışır. Üç renk kanalından sadece birine veri gömdüğü halde diğer ikisini de kullandığı için gömülecek veri kapasitesi düşer.

5.2. Öneriler

Steganografi algoritmaları veri güvenliği alanında sıklıkla başvurulan algoritmalardır. Bu algoritmaların merkezinde en önemsiz bite gömme algoritması mevcuttur. İşbu tezde rastgele indislerin kaotik algoritmalarla üretilmesine dayalı yeni bir algoritma önerilmiştir.

Yeni algoritma renkli resimler üzerinde uygulanabilmektedir.

Yeni algoritma renk kanallarından sadece biri üzerinde veri gizleme yapmakta ve bu kanalı rastgele seçmektedir. Veri gizlenen kanallar haricindeki kanallarda da tahribat yapmakta böylelikle tespit istenmeyen kişilerin eline geçen resimde metni elde etmeyi daha da güçleştirmektedir.

Üç kanaldan sadece biri üzerinde veri gizleme işlemi yöntemin kapasitesini azaltmaktadır.

Kaotik algoritmaların gelişimiyle birlikte yeni algoritmalar steganografi alanında icra edilerek verim ve güvenilirlik artırılabilir. Ayrıca önerilen kaos tabanlı algoritmanın kapasitesinin arttırılması amacıyla resim içerisine gizlenecek olan metnin farklı sıkıştırma algoritmaları ile sıkıştırılarak hibrit sistemler üzerinde çalışılabilir.

KAYNAKLAR

Aho, A. V. ve Sloane, N. J., 1973, Some doubly exponential sequences, *Fibonacci Quart*, 11 (4), 429-437.

Arnol′d, V. I. ve Avez, A., 1968, Ergodic problems of classical mechanics, Benjamin, p.

Balazs, N. ve Voros, A., 1987, The quantized baker's transformation, *EPL (Europhysics Letters)*, 4 (10), 1089.

Beimel, A., 2011, Secret-Sharing Schemes: A Survey, In: Coding and Cryptology: Third International Workshop, IWCC 2011, Qingdao, China, May 30-June 3, 2011. Proceedings, Eds: Chee, Y. M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H. ve Xing, C., *Berlin, Heidelberg*: Springer Berlin Heidelberg, p. 11-46.

Blakley, G. R., 1979, Safeguarding cryptographic keys, *Proceedings of the 1979 AFIPS National Computer Conference*, Monval, NJ, USA, 313--317.

Chae, J. J. ve Manjunath, B., 1999, Data hiding in video, *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, 311-315.

Cheddad, A., Condell, J., Curran, K. ve Mc Kevitt, P., 2010, Digital image steganography: Survey and analysis of current methods, *Signal Processing*, 90 (3), 727-752.

Choudry, K. N. ve Aakash, W., 2015, A Survey Paper on Video Steganography *International Journal of Computer Science and Information Technologies*, 6 (3), 2335-2338.

Chouse, C. ve Albayrak, S., 2014, Dayanıklı sayısal resim damgalama.

Cogranne, R. ve Fridrich, J., 2015, Modeling and extending the ensemble classifier for steganalysis of digital images using hypothesis testing theory, *Information Forensics and Security, IEEE Transactions on*, 10 (12), 2627-2642.

Collet, P. ve Eckmann, J.-P., 2009, Iterated maps on the interval as dynamical systems, Springer Science & Business Media, p.

Cox, I. J., Kilian, J., Leighton, F. T. ve Shamoon, T., 1997, Secure spread spectrum watermarking for multimedia, *Image Processing, IEEE Transactions on*, 6 (12), 1673-1687.

Çimen, C., Akleylek, S. ve Akyıldız, E., 2008, Şifrelerin matematiği: kriptografi, ODTÜ, p. 17-22.

Devaney, R. L. ve Eckmann, J.-P., 1987, An Introduction to Chaotic Dynamical Systems, *Physics Today*, 40, 108-111.

Devi, K. J., 2013, A Sesure Image Steganography Using LSB Technique and Pseudo Random Encoding Technique, *National Institute of Technology-Rourkela*.

El-Emam, N. N. ve Al-Zubidy, R. A. S., 2013, New steganography algorithm to conceal a large amount of secret message using hybrid adaptive neural networks with modified adaptive genetic algorithm, *Journal of Systems and Software*, 86 (6), 1465-1481.

Ghebleh, M. ve Kanso, A., 2014, A robust chaotic algorithm for digital image steganography, *Communications in Nonlinear Science and Numerical Simulation*, 19 (6), 1898-1907.

Grover, D., 1998, Steganography for identifying ownership of copyright, *Computer Law & Security Review*, 14 (2), 121-122.

Hénon, M., 1976, A two-dimensional mapping with a strange attractor, *Communications in Mathematical Physics*, 50 (1), 69-77.

Hong, W., 2013, Adaptive image data hiding in edges using patched reference table and pair-wise embedding technique, *Information Sciences*, 221, 473-489.

Hussain, M. ve Hussain, M., 2013, A survey of image steganography techniques.

Jane, O., 2010, Görüntü pekiştirmede kullanılan uyarlanır bulanık maskeleme tekniği için en uygun süzgeç ve parametre seçimi analizi, *Ankara Üniversitesi*, Ankara.

Jayaram, P., Ranganatha, H. ve Anupama, H., 2011, Information hiding using audio steganography–a survey, *The International Journal of Multimedia & Its Applications (IJMA) Vol*, 3, 86-96.

Kanan, H. R. ve Nazeri, B., 2014, A novel image steganography scheme with high embedding capacity and tunable visual image quality based on a genetic algorithm, *Expert Systems with Applications*, 41 (14), 6123-6130.

Keleş, C., 2012, Kaotik haritalar kullanarak görüntü şifreleme, *Karadeniz Teknik Üniversitesi*, Trabzon.

Ker, A. D., Bas, P., Böhme, R., Cogranne, R., Craver, S., Filler, T., Fridrich, J. ve Pevný, T., 2013, Moving steganography and steganalysis from the laboratory into the real world, *Proceedings of the first ACM workshop on Information hiding and multimedia security*, 45-58.

L'Ecuyer, P., 2012, Random number generation, Springer, p.

Lehmer, D. H., 1951, Mathematical methods in large-scale computing units, *Proc. 2nd Symp. on Large-Scale Digital Calculating Machinery*, 141-146.

Li, B., Wang, M., Huang, J. ve Li, X., 2014, A new cost function for spatial image steganography, *Image Processing (ICIP), 2014 IEEE International Conference on*, 4206-4210.

Lu, C.-S., Huang, S.-K., Sze, C.-J. ve Liao, H.-Y. M., 2000, Cocktail watermarking for digital image protection, *Multimedia, IEEE Transactions on*, 2 (4), 209-224.

Noda, H., Furuta, T., Niimi, M. ve Kawaguchi, E., 2004, Application of BPCS steganography to wavelet compressed video, *Image Processing, 2004. ICIP'04. 2004 International Conference on*, 2147-2150.

Pereira, S., Ruanaidh, J. J., Deguillaume, F., Csurka, G. ve Pun, T., 1999, Template based recovery of Fourier-based watermarks using log-polar and log-log maps, *Multimedia Computing and Systems, 1999. IEEE International Conference on*, 870-874.

Pooyan, M. ve Delforouzi, A., 2007, LSB-based audio steganography method based on lifting wavelet transform, *Signal Processing and Information Technology, 2007 IEEE International Symposium on*, 600-603.

Ramalingam, M. ve Isa, N. A. M., 2015, A steganography approach over video images to improve security, *Indian Journal of Science and Technology*, 8 (1), 79.

Sajedi, H. ve Jamzad, M., 2010, BSS: Boosted steganography scheme with cover image preprocessing, *Expert Systems with Applications*, 37 (12), 7703-7710.

Sedighi, V., Fridrich, J. ve Cogranne, R., 2015, Content-adaptive pentary steganography using the multivariate generalized Gaussian cover model, *IS&T/SPIE Electronic Imaging*, 94090H-94090H-94013.

Sedighi, V., Cogranne, R. ve Fridrich, J., 2016, Content-Adaptive Steganography by Minimizing Statistical Detectability, *Information Forensics and Security, IEEE Transactions on*, 11 (2), 221-234.

Shamir, A., 1979, How to share a secret, *Commun. ACM*, 22 (11), 612-613.

Shannon, C. E., 1949, Communication Theory of Secrecy Systems*, *Bell System Technical Journal*, 28 (4), 656-715.

Simmons, G. J., 1984, The prisoners' problem and the subliminal channel, *Advances in Cryptology*, 51-67.

Srinivasan, B., Arunkumar, S. ve Rajesh, K., 2015, A novel approach for color image, steganography using nubasi and randomized, secret sharing algorithm, *Indian Journal of Science and Technology*, 8 (S7), 228-235.

Swain, G., 2014, Digital image steganography using nine-pixel differencing and modified LSB substitution, *Indian Journal of Science and Technology*, 7 (9), 1444-1450.

Swanson, M. D., Zhu, B. ve Tewfik, A. H., 1996, Transparent robust image watermarking, *Image Processing, 1996. Proceedings., International Conference on*, 211-214.

Şahin, A., 2007, Görüntü steganografide kullanılan yeni metodlar ve bu metodların güvenilirlikleri.

Thien, C.-C. ve Lin, J.-C., 2002, Secret image sharing, *Computers & Graphics*, 26 (5), 765-770.

Ulutaş, G., Ulutaş, M. ve Nabiyev, V. V., 2012, A new casccaded secret image sharing scheme, *Signal Processing and Communications Applications Conference (SIU), 2012 20th*, 1-4.

Ulutürk, A., 2010, Gelişmiş şifreleme standardı, *Gazi Üniversitesi*, Ankara, 4-7.

Verma, T. G., Hasan, Z. ve Verma, D. G., 2013, A Unique Approach for Data Hiding Using Audio Steganography, *International Journal of Modern Engineering Research (IJMER) www. ijmer. com*, 3 (4).

Wang, Y. ve Izquierdo, E., 2002, High-capacity data hiding in MPEG-2 compressed video, *Proceeding of the 9th International Workshop on Systems, Signals and Image Processing, Manchester, UK*, 212-218.

Westfeld, A., 2001, F5—A Steganographic Algorithm, *Information Hiding: 4th International Workshop, IH 2001, Pittsburgh, PA, USA, April 25-27, 2001. Proceedings*, 289.

Wong, W.-k., Lee, L.-p. ve Wong, K.-w., 2001, A Modified Chaotic Cryptographic Method, In: Communications and Multimedia Security Issues of the New Century: IFIP TC6 / TC11 Fifth Joint Working Conference on Communications and Multimedia Security (CMS'01) May 21–22, 2001, Darmstadt, Germany,

Eds: Steinmetz, R., Dittman, J. ve Steinebach, M., *Boston, MA*: Springer US, p. 123-126.

Wu, M. ve Liu, B., 2013, Multimedia data hiding, Springer Science & Business Media, p.

Yalman, Y., 2010, Sayısal görüntüler için histogram temelli veri gizleme yöntemi ve uygulama yazılımı, *Kocaeli Üniversitesi*, Kocaeli.

Yuan, H.-D., 2014, Secret sharing with multi-cover adaptive steganography, *Information Sciences*, 254, 197-212.

Zagade, S. ve Bhosale, S., 2014, Secret Data Hiding in Images by using DWT Technique's.

EKLER

EK-1 Önerilen algoritmanın kaynak kodları

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Stenografi.algoritmalar
{
    /// <summary>
    ///  Algoritma üç amaşadan meydana gelir.
    ///  1. aşama metnin karıştırılmasıdır. Söz gelimi
    ///   'ayak' metni gömülecekse bu 'kaya', 'aayk', 'kyaa' gibi harfler karıştırılarak yapılabilir.
    ///   2. aşama da metnin rgb içerisine gömüleceği yerin kaotik olarak belirlenmesidir.
    ///    3. Aşamada metindeki karakterler kırmızı, yeşil ya da mavi kanallarından birisi içerisine gömülür.
    /// </summary>
    public class barisDemirci : gommeAlgoritmasi
    {
        private resim sourceImage;
        private resim destinationImage;
        private Logistic harita;
        private void varsayilan()
        {
            this.a = 0;
            this.x = 0;
            this.harita = new Logistic();
            this.turu = algoritmaTurleri.barisDemirci;
        }

        public resim _sourceImage
        {
            get
            {
                return sourceImage;
            }
```

```
            set
            {
                sourceImage = value;
            }
        }
        public resim _destinationImage
        {
            get
            {
                return destinationImage;
            }
            set
            {
                destinationImage = value;
            }
        }
        private double a, x;

        public double _a
        {
            get
            {
                return a;
            }
            set
            {
                a = value;
            }
        }
        public double _x
        {
            get
            {
                return x;
            }
            set
            {
                x = value;
            }
        }
```

```
        private void soyle(object ifade)
        {
            System.Windows.Forms.MessageBox.Show(ifade.ToString());
        }

        private void bitiBoz(ref goruntu res, int indis)
        {
            Int32[] ham = res.ikiliBit(indis, this.bit);
            int hangisi = ham[ham.Length - 1];

            Int32 enOnemsiz = 0;
            if (hangisi == 0)
                enOnemsiz = 1;
            ham[this.bit - 1] = enOnemsiz;
            res.ikiliBitAta(indis, ham);
        }
        public override void gom()
        {
            if (this._metin == null)
                throw new Exception("Gömülecek metni giriniz");

            if (this.a == 0)
                throw new Exception("a kat sayısını giriniz");

            if (this.x == 0)
                throw new Exception("x kat sayısını giriniz");

            if (sourceImage == null)
                throw new Exception("Kaynak görüntüyü giriniz");

            gomulecek = this.metninIkiliDegeri();
            int adet = gomulecek.Length;
            int resUz = sourceImage._gen * sourceImage._yuk;
            string ifade = String.Format("Adet : { 0 }   Uuzluk : ", adet,
resUz);
            int[] indisler = Logistic.uret(a, x, adet, resUz);

            goruntu kirmizi = sourceImage._kirmizi;
            goruntu yesil = sourceImage._yesil;
            goruntu mavi = sourceImage._mavi;
```

```
                for (int i = 0; i < gomulecek.Length; i++)
                {
                    int hangi = 1;

                    if (indisler[i] % 2 == 0)
                        hangi = 2;

                    if (indisler[i] % 3 == 0)
                        hangi = 3;

                    Int32[] ham;
                    int indis = indisler[i];

                    switch (hangi)
                    {
                        case 1:
                            ham = kirmizi.ikiliBit(indis, this.bit);
                            bitiBoz(ref yesil, indis);
                            bitiBoz(ref mavi, indis);

                            break;
                        case 2:
                            ham = yesil.ikiliBit(indis, this.bit);
                            bitiBoz(ref kirmizi, indis);
                            bitiBoz(ref mavi, indis);
                            break;
                        case 3:
                            ham = mavi.ikiliBit(indis, this.bit);
                            bitiBoz(ref kirmizi, indis);
                            bitiBoz(ref yesil, indis);
                            break;
                        default: ham = null;
                            break;
                    }
                    Int32 enOnemsiz = gomulecek[i];
                    ham[this.bit - 1] = enOnemsiz;

                    switch (hangi)
                    {
                        case 1:
                            kirmizi.ikiliBitAta(indis, ham);
```

```
                                break;
                            case 2:
                                yesil.ikiliBitAta(indis, ham);
                                break;
                            case 3:
                                mavi.ikiliBitAta(indis, ham);
                                break;
                        }
                    }
                    destinationImage = sourceImage;
                }
                private int metinUzunlugu;

                public int _metinUzunlugu
                {
                    get
                    {
                        return metinUzunlugu;
                    }
                    set
                    {
                        metinUzunlugu = value;
                    }
                }
                public override string coz()
                {
                    if (this.a == 0)
                        throw new Exception("a kat sayısını giriniz");

                    if (this.x == 0)
                        throw new Exception("x kat sayısını giriniz");

                    if (destinationImage == null)
                        throw new Exception("Kaynak görüntüyü giriniz");


                    int adet = metinUzunlugu * this.bit;
                    int resUz = destinationImage._gen * destinationImage._yuk;
                    string ifade = String.Format("Adet : {0}    Uuzluk : 1", adet,
resUz);
                    int[] indisler = Logistic.uret(a, x, adet, resUz);
```

```
goruntu kirmizi = destinationImage._kirmizi;
goruntu yesil = destinationImage._yesil;
goruntu mavi = destinationImage._mavi;
string deger = "";
int sayac = 0;
for (int i = 0; i < indisler.Length; i++)
{
    int indis = indisler[i];
    int hangi = 1;

    if (indisler[i] % 2 == 0)
        hangi = 2;

    if (indisler[i] % 3 == 0)
        hangi = 3;

    Int32[] siradaki;
    switch (hangi)
    {
        case 1:
            siradaki=kirmizi.ikiliBit(indis, this.bit); break;
        case 2:
            siradaki = yesil.ikiliBit(indis, this.bit); break;
        case 3:
            siradaki = mavi.ikiliBit(indis, this.bit); break;
        default:
            siradaki = null; break;
    }

    deger += siradaki[this.bit - 1].ToString();
    sayac++;
}
Int32[] bir = new int[adet];
for (int i = 0; i < bir.Length; i++)
    bir[i] = Convert.ToInt16(deger[i].ToString());


string katar = "";
for (int i = 0; i < bir.Length; i++)
    katar += bir[i].ToString();
```

```
                return          genel.cozulmusIkiliSayilariMetneDonustur(katar,
this.bit, this.kodlama);
            }
            public barisDemirci()
            {
                varsayilan();
            }
        }
    }
```

EK-2 Kaotik Rastgele Sayı Üreteç Sınıfı

```
public  class Logistic
{
    public static int[] uret(double a, double x, int adet, int tavan)
    {
        List<int> indisler = new List<int>();
        for (int i = 0; i < tavan; i++)
            indisler.Add(i);

        int[] deger = new int[adet];
        for (int i = 0; i < deger.Length; i++)
        {
            deger = Logistic.Quantization(a, x, adet, tavan );

        }
        int[] sonuc = new int[adet];
        for (int i = 0; i < deger.Length; i++ )
        {
            int bu = deger[i];
            sonuc[i] = indisler[bu];
            indisler.RemoveAt(bu);
        }

        return sonuc;
    }
    public static double LogisticHarita(double k, double x0)
    {
        x0 = k * x0 * (1 - x0);
        return x0;
    }

    public static int[] Quantization(double k, double x0, int adet,
int tavan)
    {
        int[] deger = new int[adet ];
        for (int i = 0; i < adet ; i++)
        {
            x0 = LogisticHarita(k, x0);
            deger[i] = intEldeEt(BitHesapla(x0)) % tavan;
```

```
            tavan--;
        }
        return deger;
    }


    public static double OndalikKisim(double a)
    {
        int b = 0;
        double c = 0;
        if (a >= 1)
        {
            b = Convert.ToInt32(Math.Floor(a));
            c = a - b;
        }
        else
        {
            c = a;
        }
        return c;
    }

    public static int intEldeEt(string str)
    {
        int deger = Convert.ToInt32(str, 2);
        return deger;
    }

    public static string BitHesapla(double a)
    {
        string str = "";
        int us = 9 * 3;

        for (int i = 0; i < us; i++)
        {
            a = a * 2;
            if (a < 1)
            {
                str += "0";
            }
            else
```

```
                        if (a > 1)
                        {
                            str += "1";
                        }
                        else
                            if (a == 1)
                            {
                                str += "1";
                            }
                    if (a == 1)
                        break;
                    a = OndalikKisim(a);
                }

                if (str.Length < us)
                    for (int j = 0; j < us - str.Length; j++)
                        str = '0' + str;
                return str;
            }

        }
```

EK-3 Temel İşlemlerin tanımlandığı ve gerçekleştirildiği genel.cs sınıfı

```
        /// <summary>
        ///  2liden 10luya çevirme rasgele sayı üretme gibi
        ///  fonksiyonların olduğu başvuru sınıfıdır.
        /// </summary>
        public class genel
        {

            public static int xor(int bir ,int iki)
            {

                if (bir == 0 && iki == 0)
                    return 0;

                if (bir == 0 && iki ==1 )
                    return 1;

                if (bir == 1 && iki == 0)
                    return 1;

                if (bir == 1 && iki == 1)
                    return 0;

                throw new Exception("Tanımlanmamış bit değeri");
            }

            public static double mse(goruntu x, goruntu y)
            {
                double sonuc = 0;

                for (int i = 0; i < x._boy; i++)
                    sonuc += Math.Pow(x[i] - y[i], 2.0);

                sonuc = sonuc / x._boy;
                return sonuc;
            }

            public static double Imperceptibility(goruntu x, goruntu y)
            {
                double sonuc = 0;
```

```
                for (int i = 0; i < x._boy; i++)
                    sonuc += Math.Pow(x[i] - y[i], 2.0);
                sonuc = sonuc / x._boy;

                return 10.0 * Math.Log( (255.0 * 255.0)  / sonuc,2 );
            }

            public static double psnr(goruntu x, goruntu y)
            {
                double r = 255;
                return 10.0 * Math.Log(r * r / mse(x, y), 10.0);
            }

            public static Int32 []xorlaBirlestir(int []bir, int []iki)
            {
                if (bir.Length != iki.Length)
                    throw  new  Exception("Birleştirilecek  dizilerin  uzunluğu
eşit olmalıdır");

                Int32[] sonuc = new int[bir.Length];

                for (int i = 0; i < sonuc.Length; i++                    )
                {
                    sonuc[i] = xor(bir[i], iki[i]);
                }

                    return sonuc;
            }

            public static Int32 ikilidenOna(Int32[] dizi)
            {
                string ifade = "";
                for (int i = 0; i < dizi.Length; i++)
                    ifade += dizi[i].ToString();

                return ikilidenOna(ifade);
            }
            public static Int32 ikilidenOna(string bin)
            {
                Int32  l = Convert.ToInt32(bin, 2);
```

```
            return Convert.ToInt32(1);
        }

        public static Int32 onludanIkiliye(string veri)
        {
            string sonuc = string.Empty;
            int rem = 0;


            int num = int.Parse(veri);
            while (num > 0)
            {
                rem = num % 2;
                num = num / 2;
                sonuc = rem.ToString() + sonuc;
            }
            return Convert.ToInt32(sonuc);
        }
        public static Int32[] onludanIkiliyeDizi(string veri, int kacBit)
        {
            string result = string.Empty;
            int rem = 0;
                    int num = int.Parse(veri);
            while (num > 0)
            {
                rem = num % 2;
                num = num / 2;
                result = rem.ToString() + result;
            }
            return diziyeAktar(result, kacBit);
        }

        public static Int32[] onludanIkiliyeDizi(Int32 num, int kacBit)
        {
            string result = string.Empty;
            Int32 rem = 0;


            while (num > 0)
            {
                rem = num % 2;
```

```
                    num = num / 2;
                    result = rem.ToString() + result;
                }
                return diziyeAktar(result, kacBit);
            }
            public static Int32[] diziyeAktar(string ifade, Int32 basamak)
            {
                Int32[] sonuc = new Int32[basamak];

                int sayac = sonuc.Length - 1;
                for (int i = ifade.Length - 1; i >= 0; i--)
                {
                    sonuc[sayac] = Convert.ToInt32(ifade[i].ToString());
                    sayac--;
                }
                return sonuc;
            }

             public static goruntu griDuzeyGoruntuAl(Bitmap res)
            {

                goruntu sonuc = new goruntu(res.Width, res.Height);
                for (int i = 0; i < res.Width; i++)
                    for (int j = 0; j < res.Height; j++)
                    {
                        Color renk = res.GetPixel(i, j);
                        float gri = ((float)renk.R + (float)renk.G +
(float)renk.B) / 3.0f;
                        sonuc[i, j] = (int)gri;
                    }

                return sonuc;

            }


            /// <summary>
            /// Rasgele sayı üreteci
            /// </summary>
            public static Random rasgele;
```

```
        public static List<Int32 > benzemezRasgeleSayilarUret(int enKucuk,
int enBuyuk, int kacTane)
        {
            if (rasgele == null)
                rasgele = new Random();

            List<Int32> kaynak = new List<Int32>();
            for (int i = enKucuk; i <= enBuyuk; i++)
                kaynak.Add(i);

            List<Int32> sonuc = new List<Int32>();
            for (int i = 0; i < kacTane; i++)
            {
                int kim = rasgele.Next(0, kaynak.Count - 1);

                sonuc.Add(kaynak[kim]);
                kaynak.RemoveAt(kim);
            }
            kaynak.Clear();
            return sonuc;
        }
        public static string indislerdenKatarUret(List<Int32 > dizi)
        {

            string ifade = "";
            if (dizi.Count == 0)
                return ifade;

            if (dizi.Count == 1)
                return dizi[0].ToString();

            for (int i = 0; i < dizi.Count - 1; i++)
                ifade += dizi[i].ToString() + ", ";

            ifade += dizi[dizi.Count - 1].ToString();
            return ifade;
        }

        public  static  string  cozulmusIkiliSayilariMetneDonustur(string
cozulmus, int biti, System.Text.Encoding kodlamasi)
        {
```

```
            int harfSayisi = cozulmus.Length / biti;

            byte[] katar = new byte[harfSayisi];

            for (int i = 0; i < harfSayisi; i++)
            {
                int bas = i * biti;
                int son = bas + biti;
                byte[] bitler = new byte[biti];

                string siradaki = "";
                for (int k = 0; k < biti; k++)
                {
                    bitler[k] =(byte)  cozulmus[k + bas];

                    siradaki += cozulmus[k + bas];
                }


                katar[i] = (byte) genel.ikilidenOna(siradaki);
            }
            return  kodlamasi .GetString(katar );

        }
        public static int obeb(int bir, int iki)
        {
            if (bir == iki)
                return bir;

            int kucuk = Math.Min(bir, iki);
            int buyuk = Math.Max(bir, iki);


            if (kucuk == 1)
                return 1;
            if (buyuk <= 2)
                return 1;

            int sonuc = 1;
```

```
            for(int i=2; i<= kucuk; i++)
            {
                if (kucuk % i == 0 && buyuk % i == 0)
                    sonuc = i;
            }
            return sonuc;
        }
          public static bool asalmi(int aday)
        {

            if (aday <= 2)
                return true;

            if (aday % 2 == 0)
                return false;
            int kok = Convert.ToInt32( Math.Sqrt(aday))+ 1;
            for (int i = 3; (i * i) <= kok; i += 2)
            {
                if ((aday % i) == 0)
                {
                    return false;
                }
            }
            return true;
        }

        public static  List<Int32> asalCarpanlari(Int32 sayi)
        {
            List<Int32> sonuc = new List<Int32>();

            while (sayi % 2 == 0)
            {
                int yer = sonuc.IndexOf(2);
                if(yer == -1 )
                sonuc.Add(2);
                sayi /= 2;
            }

            Int32 carpan = 3;
            while (carpan * carpan <= sayi)
```

```
                {
                    if (sayi % carpan == 0)
                    {

                        int yer = sonuc.IndexOf(carpan);

                        if(yer == -1 )
                        sonuc.Add(carpan);
                        sayi /= carpan;
                    }
                    else
                    {
                        carpan += 2;
                    }
                }

                if (sayi > 1)
                {
                    int yer = sonuc.IndexOf(sayi);
                    if (yer == -1)
                    sonuc.Add(sayi);
                }
                return sonuc;
            }

            public static ColorPalette GetColorPalette(uint nColors)
            {


                PixelFormat bitscolordepth = PixelFormat.Format1bppIndexed;
                ColorPalette palette;
                Bitmap bitmap;

                if (nColors > 2)
                    bitscolordepth = PixelFormat.Format4bppIndexed;
                if (nColors > 16)
                    bitscolordepth = PixelFormat.Format8bppIndexed;

                bitmap = new Bitmap(1, 1, bitscolordepth);

                palette = bitmap.Palette;
```

```
                bitmap.Dispose();
                return palette;
            }

            public static void SaveGIFWithNewColorTable( Image image,
string filename, uint nColors, bool fTransparent, List<Color> renkler )
            {
      .
                if (nColors > 256)
                    nColors = 256;
                if (nColors < 2)
                    nColors = 2;
                int Width = image.Width;
                int Height = image.Height;
                Bitmap bitmap = new Bitmap(Width, Height,
PixelFormat.Format8bppIndexed);
                ColorPalette pal = GetColorPalette(nColors);
                for (uint i = 0; i < nColors; i++)
                {
                  uint Alpha = 0xFF;
                  uint Intensity = i * 0xFF / (nColors - 1);
                    if (i == 0 && fTransparent)
                        Alpha = 0;
                    pal.Entries[i] = Color.FromArgb((int)Alpha,
                                                    (int)Intensity,
                                                    (int)Intensity,
                                                    (int)Intensity);


                                }
                bitmap.Palette = pal;
                Bitmap BmpCopy = new Bitmap(Width, Height);
                {
                    Graphics g = Graphics.FromImage(BmpCopy);
                    g.PageUnit = GraphicsUnit.Pixel;
                    g.DrawImage(image, 0, 0, Width, Height);
                    g.Dispose();
                }
                BitmapData bitmapData;
                Rectangle rect = new Rectangle(0, 0, Width, Height);
                bitmapData = bitmap.LockBits(
                    rect,
```

```
                    ImageLockMode.WriteOnly,
                    PixelFormat.Format8bppIndexed);
                IntPtr pixels = bitmapData.Scan0;

                unsafe
                {
                    byte* pBits;
                    if (bitmapData.Stride > 0)
                        pBits = (byte*)pixels.ToPointer();
                    else
                        pBits = (byte*)pixels.ToPointer() + bitmapData.Stride
* (Height - 1);
                    uint stride = (uint)Math.Abs(bitmapData.Stride);

                    for (uint row = 0; row < Height; ++row)
                    {
                        for (uint col = 0; col < Width; ++col)
                        {
                            Color pixel;
                            byte* p8bppPixel = pBits + row * stride + col;

                            pixel = BmpCopy.GetPixel((int)col, (int)row);
                            double luminance = (pixel.R * 0.299) +
                                (pixel.G * 0.587) +
                                (pixel.B * 0.114);


                            *p8bppPixel = (byte)(luminance * (nColors - 1) /
255 + 0.5);

                        }
                    }
                }

                bitmap.UnlockBits(bitmapData);


                bitmap.Save(filename, ImageFormat.Gif);

                BmpCopy.Dispose();
                bitmap.Dispose();
```

```
            }

            public static int[] histogram(goruntu kimin, int kacliHistogram)
            {
                int[] sonuc = new int[kacliHistogram];

                for (int i = 0; i < kimin._gen; i++)
                    for (int j = 0; j < kimin._yuk; j++)
                    {
                        int deger = kimin[i, j];
                        sonuc[deger]++;
                    }
                return sonuc;
            }
            public static int[] histogram(goruntu kimin)
            {
                return histogram(kimin, 256);
            }
            public static double kiKare(Int32[] gercek, Int32[] beklenen)
            {
                double sonuc = 0;
                for (int i = 0; i < gercek.Length; i++)
                {
                    if (beklenen[i] == 0)
                        continue;
                    sonuc += (double)((gercek[i] - beklenen[i]) * (gercek[i] -
beklenen[i])) / ((double)beklenen[i]);
                }
                return sonuc;
            }
        }
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı	**:**	Barış DEMİRCİ
Uyruğu	**:**	TC
Doğum Yeri ve Tarihi	**:**	Konya – 1986
Telefon	**:**	541 486 85 40
Faks	**:**	
e-mail	**:**	barisdemirci@live.com

EĞİTİM

Derece		**Adı, İlçe, İl**	**Bitirme Yılı**
Lise	:	Karatay Anadolu Lisesi, Konya	2004
Üniversite	:	Selçuk Üniversitesi, Konya	2008
		İstanbul Üniversitesi, İstanbul	2015
Yüksek Lisans	:	Selçuk Üniversitesi, Konya	2016
Doktora	:		

İŞ DENEYİMLERİ

Yıl	**Kurum**	**Görevi**
2008	Meb	Bilişim Teknolojileri Öğretmeni

UZMANLIK ALANI

Kriptografi, Steganografi

YABANCI DİLLER

İngilizce

BELİRTMEK İSTEĞİNİZ DİĞER ÖZELLİKLER

YAYINLAR