

## Anahtar Bağımlı Bir Şifreleme Algoritması (IRON)

**Necati Demir, Gökhan Dalkılıç**

Dokuz Eylül Üniversitesi, Bilgisayar Mühendisliği Bölümü, 35160, İzmir  
ndemir@demir.web.tr, dalkilic@cs.deu.edu.tr

**Özet:** Bu makalede, Feistel yapısı kullanan yeni bir modern simetrik şifreleme algoritması anlatılmıştır. Bu yeni algoritmada; gizli kutular, alt anahtarlar ve döngü sayısı anahtar bağımlıdır. Bu değerler anahtarın çeşitli işlemlerden geçirilmesi ile elde edilmiştir. Yaratılan algoritma yazılım olarak geliştirilmiş, donanımsal olarak test edilmemiştir.

**Anahtar Kelimeler:** Şifreleme, Şifreleme Algoritması, Feistel Yapısı

### A Key Dependent Encryption Algorithm (IRON)

**Abstract:** In this paper, a new modern symmetric encryption algorithm, that using Feistel structure, is explained. In this new algorithm; secret boxes, sub keys and number of rounds are key dependent. These values are found by doing some operations using key. Created algorithm was created as software, wasn't tested as hardware.

**Keywords:** Encryption, Encryption Algorithm, Feistel Structure

#### 1. Giriş

İnternet üzerinde Türkçe bilgilerin çeşitliliği ve e-devlet projesi kapsamında İnternet kullanımının artması bilgi güvenliğini de bir ihtiyaç haline getirmiştir. Bilgi güvenliğinin sağlanması amacıyla yurtdışında geliştirilmiş bazı şifreleme algoritmaları kullanılmaktadır. Yaygın olarak kullanılan şifreleme algoritmaları simetrik yani tek anahtarlı algoritmalarıdır.

Simetrik anahtarlı şifreleme algoritmaları genel olarak geleneksel (Sezar, Monoalfabetik Şifreleme Sistemleri, vb) ve modern (DES, AES, vb) olarak ikiye ayrılabilir. Klasik algoritmalar karakter tabanlı şifreleme teknikleridir. Modern teknikler ise bitler üzerinde işlemler yapılarak gerçekleştirilir ve kullanım alanları daha geniştir.

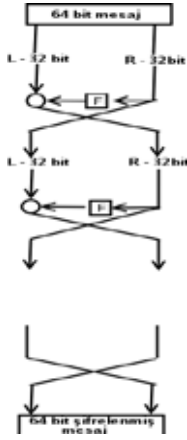
Bu makalede anlatılan şifreleme algoritması modern simetrik şifreleme algoritmaları sınıfında yer alır ve Feistel yapısını kullanarak geliştirilmiştir. Bu yapının gereği olarak, aynı

şifreleme algoritmasında aynı alt anahtarlar kullanılarak da deşifreleme yapılabilir. Deşifreleme işlemi için ayrı bir algoritmaya ihtiyaç duyulmaz sadece şifreleme sırasında kullanılan alt anahtarlar ters sıradan kullanılır. Geliştirilen şifreleme algoritmasında döngü sayısı da dâhil olmak üzere, alt anahtarlar ve gizli kutuların oluşturulması da verilen anahtara bağlıdır.

#### 2. Algoritmanın Açıklaması

Geliştirilen algoritma Feistel (Şekil 1) yapısındadır ve girdi olarak 128 bit uzunluğunda anahtar ve 64 bit uzunluğunda veri blokları kullanır. Döngü sayısı, anahtar bağımlıdır. En az 16, en fazla 32 adet olabilir.

DES, Blowfish gibi birçok algoritma Feistel yapısını kullanmaktadır [1,2]. Bu algoritmaları birbirlerinden farklı kılan F fonksiyonu ve alt anahtarların bulunuş şeklidir.



Şekil 1. Feistel Yapısı

Ayrıca,  $\pi$  sayısının kesirli kısmının on altılı sayı sistemi gösterimi de sabit değerler olarak belirlenmiştir.  $\pi$  sayısının kullanılma nedeni, basamaklarını oluşturan sayıların rastgelelik özelliği taşımasıdır. Bu sabit değerler, 256 adet 32 bitlik on altılı sayı sistemi değerlerinden oluşmaktadır. Bu sabitler, alt anahtarların ve gizli kutuların oluşturulmasında kullanılacaktır. Bu değerler; makalede  $P_0, P_1, P_2, \dots, P_{255}$  olarak gösterilecektir.

Şifreleme işlemine başlamadan önce, döngü sayısı, gizli kutular ve alt anahtarlar hesaplanmalıdır.

### 2.1 Döngü Sayısının Bulunması

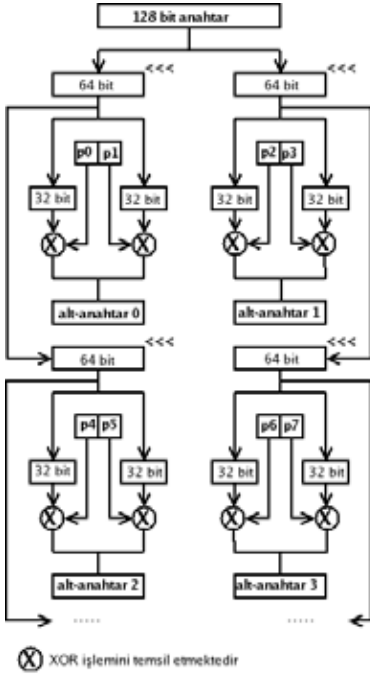
128 bit uzunluğundaki anahtar, 32 adet 4 bitlik bloklara ayrılır. Bu 4 bitlik bloklar birbirleri ile XOR işlemine sokulur. Çıkan 4 bitlik sayı onluk sayı sistemine çevrilir. Bu tamsayı değerinin çift ve 16-32 aralığında olması için tek ise 17, değilse 16 eklenir. Çıkan sonuç döngü sayısını verir.

### 2.2 Alt Anahtarların Bulunması

Alt anahtarlar hesaplanırken kullanılacak döngü sayısı, şifreleme algoritmasında kullanılan döngü sayısının yarısı kadardır çünkü alt anahtarları oluştururken kullanılan her döngüden iki adet alt anahtar çıkmaktadır. Alt anahtarların oluşturulması şu şekilde anlatılabilir (Şekil 2);

1. 128 bitlik anahtar, 64 bitlik bloklara ayrılır.
2. Sol blok bitleri, sola bir bit kaydırılır. İlk bit, sona alınır.
3. 2 nolu basamaktan çıkan 64 bitlik blok, 32 bitlik bloklara ayrılır ve bunlar  $P_0, P_1$  ile XOR işlemine sokulur.
4. 3 nolu basamaktan çıkan 32 bitlik veriler birleştirilir, böylece ortaya ilk alt anahtar çıkar.
5. Birinci basamakta elde edilen sağ bloğun bitleri, sola bir bit kaydırılır. İlk bit, sona alınır.
6. 5 numaralı basamaktan çıkan 64 bitlik blok, 32 bitlik bloklara ayrılır ve bunlar  $P_2, P_3$  ile XOR işlemine sokulur.
7. 6 numaralı basamaktan çıkan 32 bitlik veriler birleştirilir, böylece ortaya ikinci alt anahtar çıkar. Bu işlem sonunda ilk döngü tamamlanmış olur.
8. İkinci döngüde, 2. ve 5. adımlardan çıkan 64 bitlik bloklar yine, 2. ve 5. adımda olduğu gibi sola kaydırma işlemine tabi tutulur.
9. Sol 64 bitlik blok için 3. adım uygulanır ama bir farkla XOR işlemi için  $P_4$  ve  $P_5$  kullanılır. Ardından 4. adımda olduğu gibi bloklar birleştirilir.
10. Sağ 64 bitlik blok için 6. adım uygulanır ama bir farkla XOR işlemi için  $P_6$  ve  $P_7$  kullanılır. Ardından 7. adımda olduğu gibi bloklar birleştirilir.
11. Bundan sonraki adımlar da ikinci döngüde olduğu gibi devam eder ama bundan sonra kullanılacak olan sabitler  $P_8, P_9, \dots$  olarak artacaktır.

Bu işlemler esnasında 256 adet  $P$  sabitlerinden hepsi kullanılmaz. En az 32 tanesi, en fazla 64 tanesi kullanılır.  $\pi$  sayısının kesirli kısmının on altılı sayı sistemindeki değerleri, Blowfish algoritmasında da kullanılmaktadır [1,2]. Bu değerler Bailey-Borwein-Plouffe algoritması kullanılarak bulunabilir [3,4,5].



Şekil 2. Alt anahtarların oluşturulması

### 2.3 Gizli Kutuların Oluşturulması

4 adet 8x32 boyutunda gizli kutu oluşturulacaktır, her bir kutuya girdi olarak 8 bit alınır ve kutudan 32 bit çıkar. Bu işlemin gerçekleşmesi için her kutu 256 adet her biri 32 bitten oluşan veriyi barındırır.

Gizli kutular, 128 bitlik anahtar ve  $P$  sabitleri kullanılarak bulunur. Kullanılan işlemler toplama, çarpma ve XOR işlemleridir. Toplama ve çarpma işlemleri  $2^{32}+1$  moduna göre yapılmaktadır. Gizli kutuların elemanları şu şekilde gösterilmiştir;

$$S_{0,0}, S_{0,1}, S_{0,2}, S_{0,3}, \dots, S_{0,255}$$

$$S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3}, \dots, S_{1,255}$$

$$S_{2,0}, S_{2,1}, S_{2,2}, S_{2,3}, \dots, S_{2,255}$$

$$S_{3,0}, S_{3,1}, S_{3,2}, S_{3,3}, \dots, S_{3,255}$$

Alt anahtarlar ise şu şekilde gösterilmiştir;

$$SK_0, SK_1, SK_2, \dots$$

Gizli kutuların oluşturulması aşağıda anlatılmıştır;

İlk gizli kutunun bulunması;

1. 128 bitlik anahtar 4 adet 32 bitlik bloklara ayrılır;  $k_0, k_1, k_2, k_3$
2. İlk gizli kutunun ilk elemanını oluşturmak için 32 bitlik anahtar blokları ve  $P_0$  toplanır. Böylece ilk gizli kutunun ilk elemanı bulunmuş olur.
3.  $SK_0$  alt anahtarının ilk 32 biti ile ikinci 32 biti çarpılır.
4. Bir önceki çıkan gizli kutu elemanı ile  $P_1$  çarpılır.
5. 3 nolu adımdan ve 4 nolu adımdan çıkan sonuçlar toplanır.
6. Bu şekilde 255 kere döngü devam eder, her eleman bir önceki elemanı kullanarak bulunur. Her döngüde  $P$  indisi bir artar. Son alt anahtar kullanıldığı zaman, alt anahtar tekrar  $SK_0$ 'dan itibaren kullanılmaya başlanır.

İkinci gizli kutunun bulunması;

1. Bir önceki gizli kutunun son elemanı ile  $k_1$  ve  $P_0$  XOR işlemine sokulur. Çıkan sonuç ikinci gizli kutunun ilk elemanıdır.
2. İlk kutu oluşturulurken kullanılan 2. adımdan devam edilir.

Üçüncü ve dördüncü gizli kutunun oluşturulması;

Üçüncü ve dördüncü gizli kutuların oluşturulmasında tek fark, ilk elemanlardır. Üçüncü gizli kutuda ilk eleman, ikinci kutunun son elemanı,  $k_2$  ve  $P_0$  XOR işlemine sokularak bulunur. Dördüncü gizli kutuda ilk eleman, üçüncü kutunun son elemanı,  $k_3$  ve  $P_0$  XOR işlemine sokularak bulunur.

İlk gizli kutunun ilk elemanı oluşturulurken XOR işlemi yerine toplama işlemi kullanılmamasının sebebi şudur;

Anahtarın 32 bitlik bloklarının birbirlerinin aynı olduğu durumlarda XOR işlemi kullanılırsa, oluşacak gizli kutular da aynıdır. Örneğin XOR kullanılacak olsaydı “aaaaaaaaaaaa” ile “6666666666666666” anahtarlarının oluşturacağı gizli kutular aynı olacaktır çünkü anahtarın 32 bitlik bloklarının XOR sonucu hep aynı olacaktır. Ama toplama işlemi yapılarak bir çıkış etkisi meydana gelmektedir, bu sayede gizli kutular oluşturulurken kullanılan zincirleme reaksiyon ile birbirlerinden bağımsız gizli kutular meydana gelmektedir.

## 2.4 F Fonksiyonu

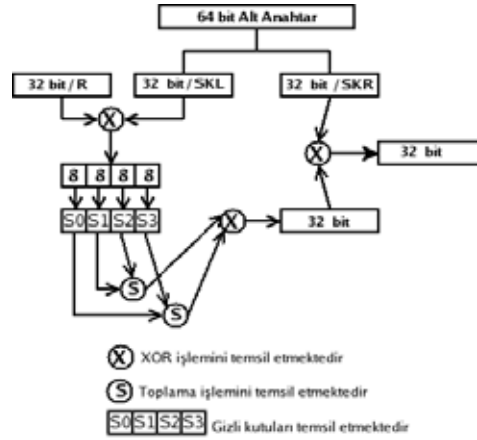
F fonksiyonu Feistel yapısının önemli basamaklarından biridir. Genellikle bu fonksiyon girdi olarak, bir önceki döngüden gelen verinin sağ bloğu ve alt anahtar olmak üzere iki parametre alır.

Bu algorithma da girdiler bu şekilde alınmıştır, bunlar  $R$  ve  $SK$  olarak temsil edilecektir.

F fonksiyonu içindeki toplama işlemi  $2^{32}+1$  moduna göre hesaplanmıştır (Şekil 3).

Bu fonksiyon şu şekilde anlatılabilir;

1. Alt anahtar ( $SK$ ) sol ve sağ olmak üzere 32 bitlik bloklara ayrılır;  $SKL$ ,  $SKR$ .
2.  $SKL$ ,  $R$  ile XOR işlemine sokulur.
3. 2 nolu adımdan çıkan 32 bitlik blok 4 adet 8 bitlik bloğa ayrılır.
4. Her bir blok sırayla  $S_0$ ,  $S_1$ ,  $S_2$  ve  $S_3$  gizli kutularına girer ve 4 adet 32 bitlik blok çıkar;  $A_0, A_1, A_2, A_3$ .
5.  $A_0$  ve  $A_3$  toplanır.
6.  $A_1$  ve  $A_2$  toplanır.
7. 5 ve 6. adımlardan çıkan 32 bitlik bloklar XOR işlemine sokulur.
8.  $SKR$ , 7. adımdan çıkan 32 bitlik blok ile XOR işlemine sokulur. Çıkan 32 bitlik blok F fonksiyonun çıktısıdır.



Şekil 3. F Fonksiyonu

## 3. Algoritmanın Avantajları

Alt anahtar oluşturma, gizli kutuların bulunması ve döngü sayısının anahtara bağlı olması algoritmanın lineer ve diferansiyel ataklara karşı dayanıklılığı arttırmaktadır.

Bu algorithma diğer modern simetrik anahtarlı şifreleme algoritmalarından farklı olarak döngü sayısı da anahtar bağımlıdır. RC5 algoritmasında döngü sayısı bir seçenek olarak sunulmuştur ama anahtara bağımlı değildir. Geliştirilen bu algorithma minimum döngü sayısı olarak Feistel yapısında güvenli olarak kabul edilen 16 alınmıştır ve anahtara bağımlı olarak 16 ve 32 arasında değişebilir. Bu sayede saldırgan, döngü sayısını da bilmediği için, yapacağı atak bir adım daha zorlaşacaktır.

Gizli kutuların sabit değerlerden değil de, değişken değerlerden oluşması da diferansiyel atakları zorlaştırmaktadır. DES algoritmasında bu algorithma anlatılan gizli kutularla aynı görevi yapan değiştirme kutuları (*substitution boxes*) sabit iken, Blowfish algoritmasında da bu algorithmadaki gibi değişkendir.

Algorithma işlemler DES algoritmasında olduğu gibi tek tek bitler üzerinden yapılmamaktadır. Bu da algoritmanın kodlanmasını kolaylaştırmaktadır.

#### **4. Kısıtlamalar**

Geliştirilen algoritma donanımlar için tasarlanmadığı için donanıma uyumu zor olabilir.

Döngü sayısının değişken olması bazı durumlarda işlem yükünü arttırabilir.

Algoritma girdi olarak sadece 128 bit uzunluğunda anahtar almaktadır. Blowfish, AES, RC5 algoritmaları değişken anahtar uzunluklarını kabul ederken; DES algoritması sabit uzunlukta anahtar kabul etmektedir [1]. Ama burada göz ardı edilmemesi gereken nokta, DES algoritmasının 56 bit uzunluğunda anahtar kullanmasıdır.

#### **5. Sonuç**

Geliştirilen algoritma simetrik modern şifreleme algoritmasıdır ve 128 bitlik anahtar ile 64 bitlik veri blokları üzerinde kullanılabilir. Algoritma içerisinde yapılan işlemler anahtar bağımlıdır, bu da anahtarın bilinmemesi halinde açık metine ulaşmayı engellemektedir.

Bu algoritma Linux işletim sistemi üzerinde kodlanarak test edilmiştir ve Pardus işletim sistemine kolaylıkla uygulanabilir.

#### **6. Kaynaklar**

[1] Stallings W., Cryptography and Network Security: Principles and Practice, Prentice Hall, New Jersey, 1998.

[2] Schneier B., Applied Cryptography, Wiley, New York, 1996.

[3] Yuen P.K., Practical Cryptology and Web Security, Addison Wesley, Harlow, 2006.

[4] BBP Formula, <http://mathworld.wolfram.com/BBPFormula.html>, Wolfram Research Inc.

[5] Extract Hex Digits of Pi Program Listing in C++, <http://www.geocities.com/hjsmith/Pi/PiQPCpp.html>, Harry J. Smith.