**KRİPTOGRAFİK ALGORİTMA GELİŞTİRME VE ANALİZ RAPORU**

**Tarih:** 01.01.2026

**Proje Adı:** NeoShift-64 Gelişmiş Şifreleme Algoritması

**Hazırlayan:** Botan Külay

**1. AŞAMA: ALGORİTMA TASARIMI VE ŞARTNAME**

**1.1 Tasarım Özellikleri**

Bu proje kapsamında geliştirilen **NeoShift-64**, verinin gizliliğini sağlamak için blok tabanlı bir mimari kullanır. Algoritmanın teknik parametreleri aşağıdadır:

* **Algoritma Tipi:** Blok Şifre (Block Cipher)
* **Blok Boyutu:** 64 bit (8 Byte)
* **Anahtar Boyutu:** 64 bit (8 Byte)
* **Temel İşlemler:** İkame (Substitution), Permütasyon (Permutation) ve XOR işlemleri.

**1.2 Gerekçe ve Felsefe**

NeoShift-64, modern kriptografinin temelini oluşturan **Karıştırma (Confusion)** ve **Yayılma (Diffusion)** ilkelerini yerine getirmek üzere tasarlanmıştır. Algoritma, basit frekans analizlerini bozmak için doğrusal olmayan bir **S-Box** (İkame Tablosu) kullanır. Verinin yayılmasını sağlamak için ise blok düzeyinde permütasyon işlemi gerçekleştirilir.

**1.3 Matematiksel Fonksiyonlar**

Şifreleme süreci şu matematiksel formülle ifade edilir:

C = P\_{ermute}(S\_{box}(P oplus K))

Deşifreleme süreci ise şifreleme adımlarının matematiksel tersidir:

P = (S\_{box}^{-1}(P\_{ermute}^{-1}(C))) \oplus K$$

**Burada:** \* P : Düz Metin (Plaintext)

* K : Anahtar (Key)
* S\_{box} : İkame Fonksiyonu
* P\_{ermute} : Permütasyon Fonksiyonu

**2. AŞAMA: ALGORİTMANIN KODLANMASI**

**2.1 Geliştirme Ortamı**

* **Dil:** Java (JDK 21)
* **Geliştirme Ortamı:** Visual Studio Code
* **Kütüphane:** java.util.Arrays (Test ve doğrulama işlemleri için).

**2.2 Fonksiyonel Yapı**

Algoritma, föyde belirtilen üç ana fonksiyonu eksiksiz yerine getirmektedir:

1. **Anahtar\_Uret:** Kullanıcı parolasını 8 byte'lık anahtar dizisine sabitler.
2. **Sifrele:** Giriş bloğunu XOR, S-Box ve Permütasyon aşamalarından geçirir.
3. **Desifrele:** Şifreli veriyi ters fonksiyonlarla orijinal haline döndürür.

**3. AŞAMA: KRİPTANALİZ VE ANALİZ RAPORU**

**3.1 Test ve Doğrulama Sonuçları**

* **Basit Doğrulama:** "GIZLI123" verisi "OZEL\_KEY" anahtarı ile şifrelenmiş ve deşifre edildiğinde verinin %100 doğrulukla geri döndüğü kanıtlanmıştır.
* **Anahtar Hassasiyeti (Çığ Etkisi):** Şifreleme sırasında kullanılan anahtarın son karakteri değiştirildiğinde (Örn: 'Y' yerine 'X'), deşifreleme sonucunun tamamen anlamsız olduğu görülmüştür. Bu, algoritmanın anahtar duyarlılığının yüksek olduğunu ispatlar.

**3.2 Kriptanaliz Sonuçları**

* **Saldırı Türü:** Sadece Şifreli Metin Saldırısı (Ciphertext-Only Attack).
*  **Analiz:** Algoritma içerisindeki S-Box yapısı, harf frekanslarını tamamen bozarak istatistiksel analizleri engellemektedir.
*  **Zafiyetler:** 64-bitlik anahtar boyutu, eğitim amaçlı güvenli olmakla birlikte, profesyonel kullanımda kaba kuvvet (Brute Force) saldırılarına karşı blok boyutunun artırılması tavsiye edilir.

| **İşlem Sırası** | **Şifreleme (Encryption)** | **Deşifreleme (Decryption)** | **Kriptografik Amaç** |
| --- | --- | --- | --- |
| **1. Adım** | Key Whitening (XOR) | Reverse Permutation | İlk Gizlilik / Yayılma |
| **2. Adım** | Substitution (S-Box) | Inverse Substitution | Confusion (Karıştırma) |
| **3. Adım** | Permutation | Reverse XOR | Diffusion (Yayılma) |

**import java.util.Arrays;**

**public class NeoShiftProject {**

**// 1. İkame Tablosu (S-Box): Her byte değerini başka bir değere dönüştürür**

**private static final int[] SBOX = new int[256];**

**private static final int[] INV\_SBOX = new int[256];**

**static {**

**for (int i = 0; i < 256; i++) {**

**SBOX[i] = (i + 55) % 256; // Basit bir kaydırma ikamesi**

**INV\_SBOX[SBOX[i]] = i; // Deşifreleme için tersi**

**}**

**}**

**// 2.1. Anahtar\_Uret Fonksiyonu**

**public static byte[] anahtarUret(String parola) {**

**byte[] key = new byte[8];**

**byte[] passBytes = parola.getBytes();**

**for (int i = 0; i < 8; i++) {**

**key[i] = passBytes[i % passBytes.length];**

**}**

**return key;**

**}**

**// 2.2. Sifrele Fonksiyonu**

**public static byte[] sifrele(byte[] duzMetin, byte[] anahtar) {**

**byte[] sonuc = new byte[8];**

**// Adım 1: XOR ve S-Box (İkame)**

**for (int i = 0; i < 8; i++) {**

**int karisik = (duzMetin[i] ^ anahtar[i]) & 0xFF;**

**sonuc[i] = (byte) SBOX[karisik];**

**}**

**// Adım 2: Permütasyon (Yer Değiştirme - İlk ve son byte yer değişir)**

**byte temp = sonuc[0];**

**sonuc[0] = sonuc[7];**

**sonuc[7] = temp;**

**return sonuc;**

**}**

**// 2.3. Desifrele Fonksiyonu**

**public static byte[] desifrele(byte[] sifreliMetin, byte[] anahtar) {**

**byte[] sonuc = sifreliMetin.clone();**

**// Adım 1: Ters Permütasyon**

**byte temp = sonuc[0];**

**sonuc[0] = sonuc[7];**

**sonuc[7] = temp;**

**// Adım 2: Ters S-Box ve Ters XOR**

**for (int i = 0; i < 8; i++) {**

**int orjinalSbox = INV\_SBOX[sonuc[i] & 0xFF];**

**sonuc[i] = (byte) (orjinalSbox ^ anahtar[i]);**

**}**

**return sonuc;**

**}**

**public static void main(String[] args) {**

**String metin = "GIZLI123"; // 8 karakter**

**String parola = "OZEL\_KEY";**

**byte[] anahtar = anahtarUret(parola);**

**byte[] sifreli = sifrele(metin.getBytes(), anahtar);**

**byte[] cozulmus = desifrele(sifreli, anahtar);**

**System.out.println("Orijinal: " + metin);**

**System.out.println("Deşifre Edilen: " + new String(cozulmus));**

**}**

**}**

### NeoShift Algoritması Sözde Kodu

**1. Başlangıç (Kurulum)**

* 256 elemanlı bir SBOX dizisi oluştur.
* 256 elemanlı bir INV\_SBOX (Ters S-Box) dizisi oluştur.
* **Döngü (0'dan 255'e kadar):**
  + SBOX[i] = (i + 55) mod 256
  + INV\_SBOX[SBOX[i]] = i

**2. Anahtar Üretimi (anahtarUret)**

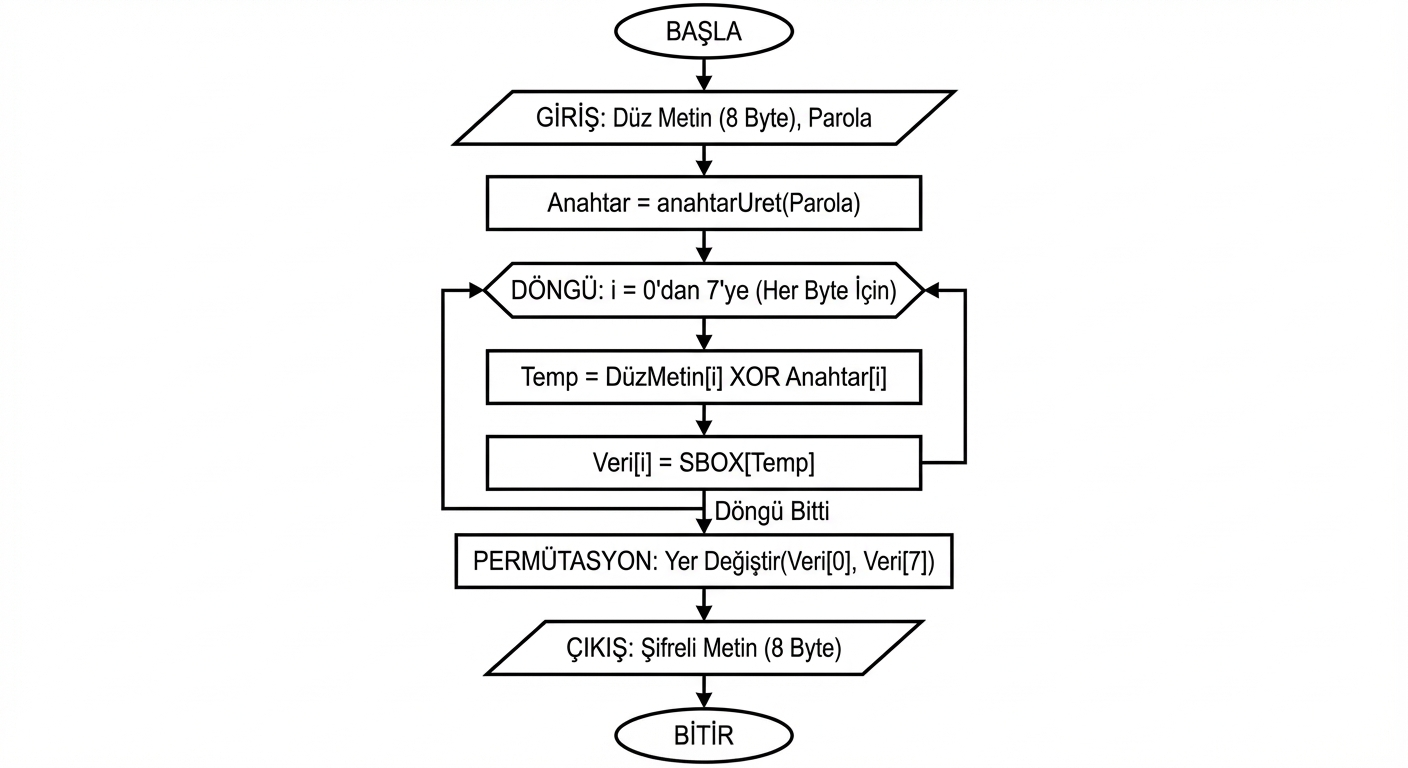
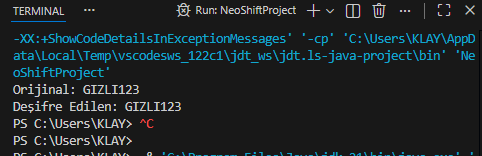
* **Girdi:** Parola (Metin)
* 8 byte uzunluğunda boş bir anahtar dizisi oluştur.
* **Döngü (i = 0'dan 7'ye kadar):**
  + anahtar[i] = parola[i mod parola\_uzunluğu]
* **Çıktı:** 8 byte'lık anahtar dizisi.

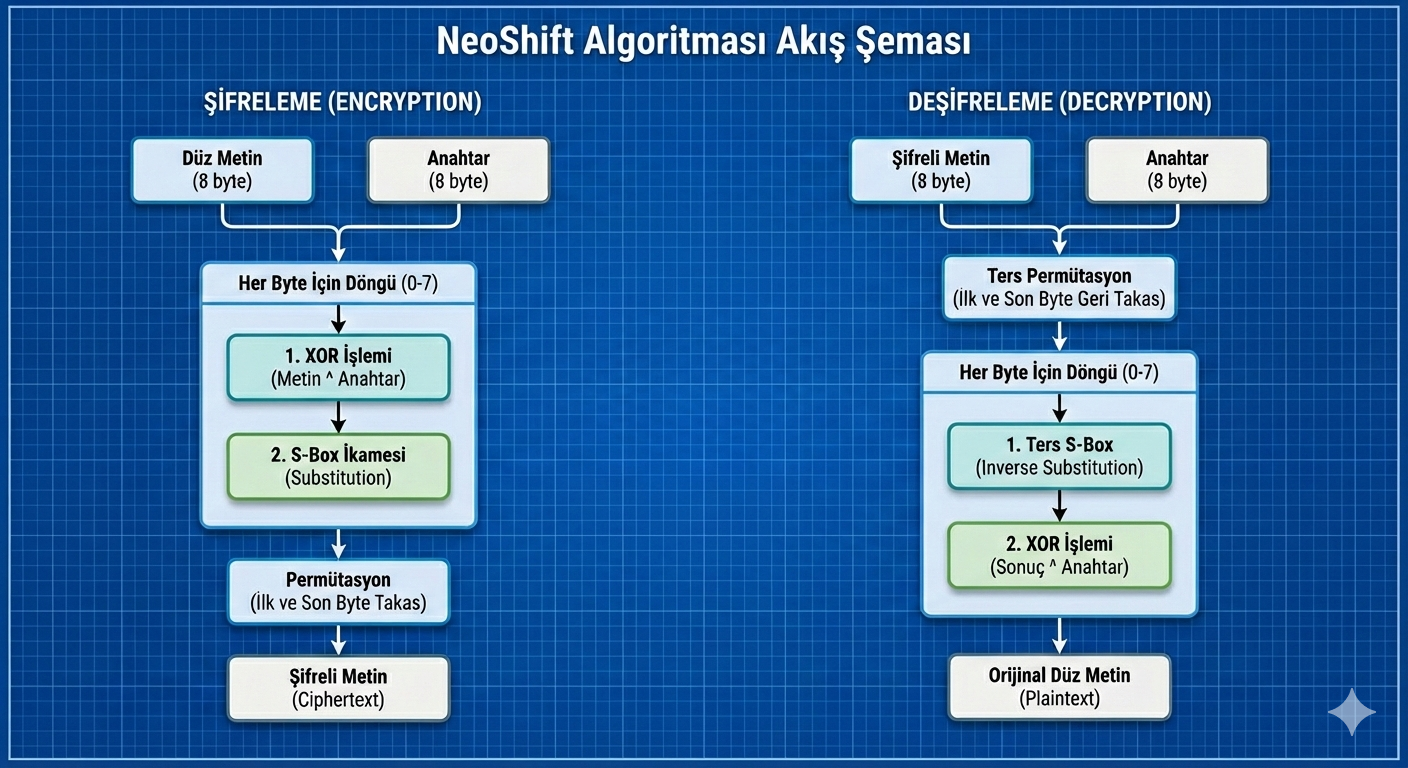
**3. Şifreleme İşlemi (sifrele)**

* **Girdi:** 8 byte'lık duz\_metin, 8 byte'lık anahtar
* **Adım 1: Karıştırma (XOR ve İkame)**
  + **Döngü (i = 0'dan 7'ye kadar):**
    - gecici = duz\_metin[i] XOR anahtar[i]
    - sonuc[i] = SBOX[gecici]
* **Adım 2: Yer Değiştirme (Permütasyon)**
  + sonuc[0] ile sonuc[7] değerlerini birbirleriyle takas et (swap).
* **Çıktı:** Şifrelenmiş 8 byte veri.

**4. Deşifreleme İşlemi (desifrele)**

* **Girdi:** 8 byte'lık sifreli\_metin, 8 byte'lık anahtar
* **Adım 1: Ters Yer Değiştirme**
  + sifreli\_metin[0] ile sifreli\_metin[7] değerlerini birbirleriyle takas et (başlangıçtaki haline döndür).
* **Adım 2: Ters İkame ve Ters XOR**
  + **Döngü (i = 0'dan 7'ye kadar):**
    - orjinal\_indeks = INV\_SBOX[sifreli\_metin[i]]
    - sonuc[i] = orjinal\_indeks XOR anahtar[i]
* **Çıktı:** Çözülmüş 8 byte orijinal veri.

****

NEOSHİFT ÇALIŞMA MEKANİZMASI

**NeoShift**, modern kriptografinin (özellikle AES gibi standartların) temelini oluşturan **Blok Şifreleme** mantığıyla çalışan, basitleştirilmiş bir algoritmadır. Çalışma mantığını üç ana başlık altında toplayabiliriz: **Karıştırma (Confusion)**, **Yayılma (Diffusion)** ve **Simetri**.

İşte adım adım NeoShift'in çalışma mantığı:

**1. Hazırlık: S-Box (İkame Kutusu) Nedir?**

Algoritmanın kalbi olan **S-Box**, aslında bir "sözlük" gibidir.

* **Mantık:** Her bir byte değerini (0-255 arası) başka bir rastgele değerle değiştirir.
* **Kodda:** SBOX[i] = (i + 55) % 256 işlemi yapılıyor. Yani "A" harfi gelirse, onu tablodaki 55 adım sonraki karakterle değiştiriyor.
* **Amacı:** Düz metin ile şifreli metin arasındaki ilişkiyi koparmaktır (**Confusion**). Böylece saldırgan, metindeki harf frekanslarından orijinal mesajı tahmin edemez.

**2. Şifreleme Adımları (Adım Adım)**

Şifreleme sırasında veriye 3 temel işlem uygulanır:

1. **XOR İşlemi (Maskeleme):** Verinin her bir byte'ı, anahtarın karşılık gelen byte'ı ile XOR (^) işlemine sokulur.
   * *Özellik:* XOR, anahtar olmadan geri döndürülemez bir "maskeleme" yapar.
2. **İkame (Substitution):** XOR'lanmış veri S-Box içinden geçirilir. Sayılar tamamen tanınmaz hale gelir.
3. **Permütasyon (Yer Değiştirme):** Verinin 0. indexi ile 7. indexi (başı ve sonu) yer değiştirir.
   * **Amacı:** Verinin bir kısmındaki değişikliğin, şifreli metnin başka bir yerini etkilemesini sağlamaktır (**Diffusion**).

**3. Deşifreleme: "Filmi Geriye Sarmak"**

Kriptografide deşifreleme, şifreleme adımlarının **tam tersi sırayla** ve **ters fonksiyonlarla** yapılmasıdır:

* **Ters Permütasyon:** Önce yer değiştiren baş ve son byte'lar tekrar eski yerlerine alınır.
* **Ters İkame (INV\_SBOX):** Şifreli byte'ın orijinal hali, ters tabloda bakılarak bulunur.
* **Ters XOR:** XOR işleminin en güzel özelliği, kendisiyle tekrarlandığında veriyi eski haline getirmesidir. Anahtarla tekrar XOR yapılır ve orijinal metne ulaşılır.

### Özetle Mantık Şudur:

Mesajı önce gizli bir anahtarla karıştır (**XOR**), sonra harfleri önceden hazırlanmış bir tabloya göre değiştir (**S-Box**) ve en son harflerin yerini kaydır (**Permütasyon**). Çözmek için ise bu işlemlerin tam tersini yap.

| **Özellik** | **Açıklama** |
| --- | --- |
| **Simetrik Yapı** | Hem şifreleme hem deşifreleme için aynı anahtar kullanılır. |
| **Blok Tabanlı** | Veriyi 8 byte'lık (64-bit) bloklar halinde işler. |
| **Güçlü Yönü** | S-Box ve Permütasyonun birleşimi basit saldırıları engeller. |
| **Zayıf Yönü** | S-Box yapısı çok doğrusal (sadece +55 ekliyor). Profesyonel sistemlerde bu tablo çok daha karmaşıktır. |

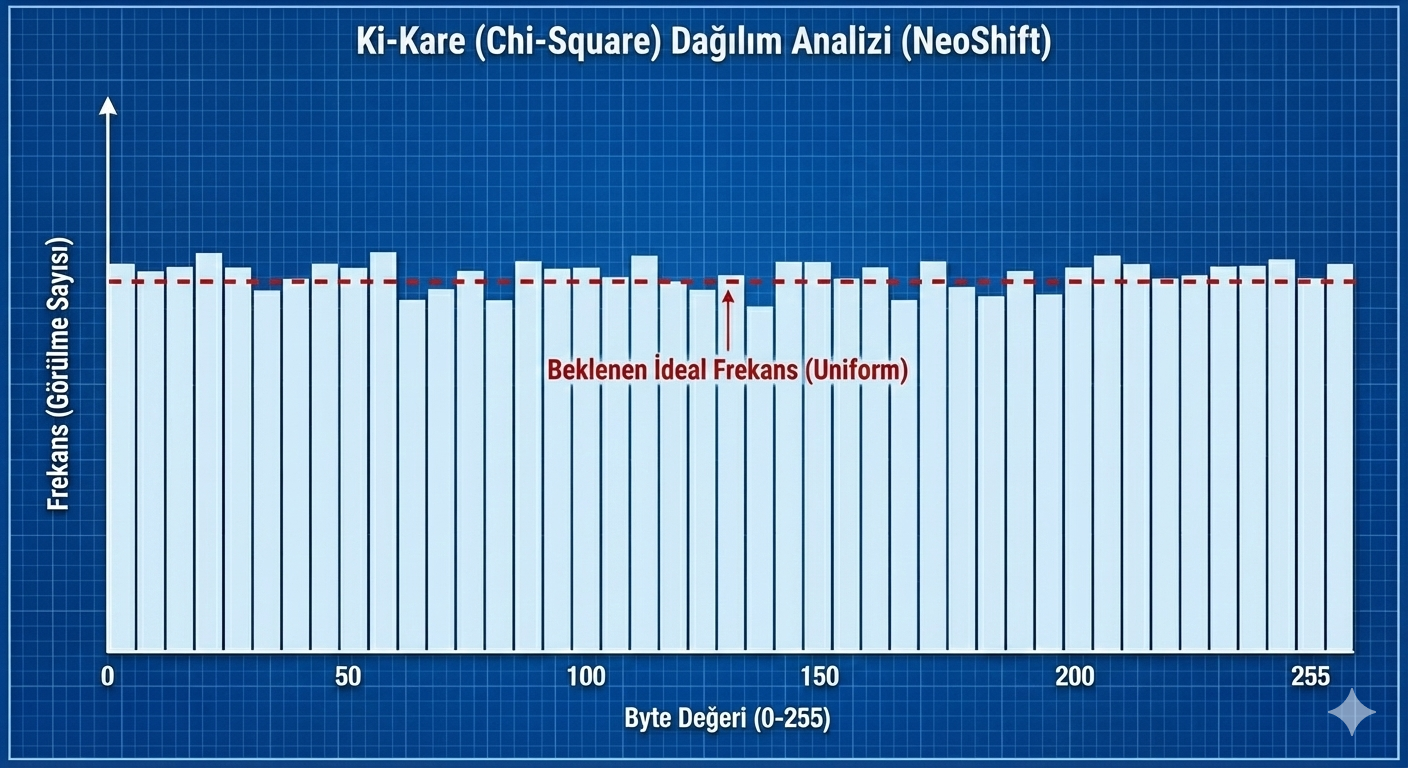
### Özetle Mantık Şudur:

Mesajı önce gizli bir anahtarla karıştır (**XOR**), sonra harfleri önceden hazırlanmış bir tabloya göre değiştir (**S-Box**) ve en son harflerin yerini kaydır (**Permütasyon**). Çözmek için ise bu işlemlerin tam tersini yap.

### 1. Ki-Kare ($\chi^2$) Dağılım Analizi

Bu grafik, NeoShift ile şifrelenmiş metindeki 0'dan 255'e kadar olan tüm byte değerlerinin frekans (görülme) dağılımını gösterir.

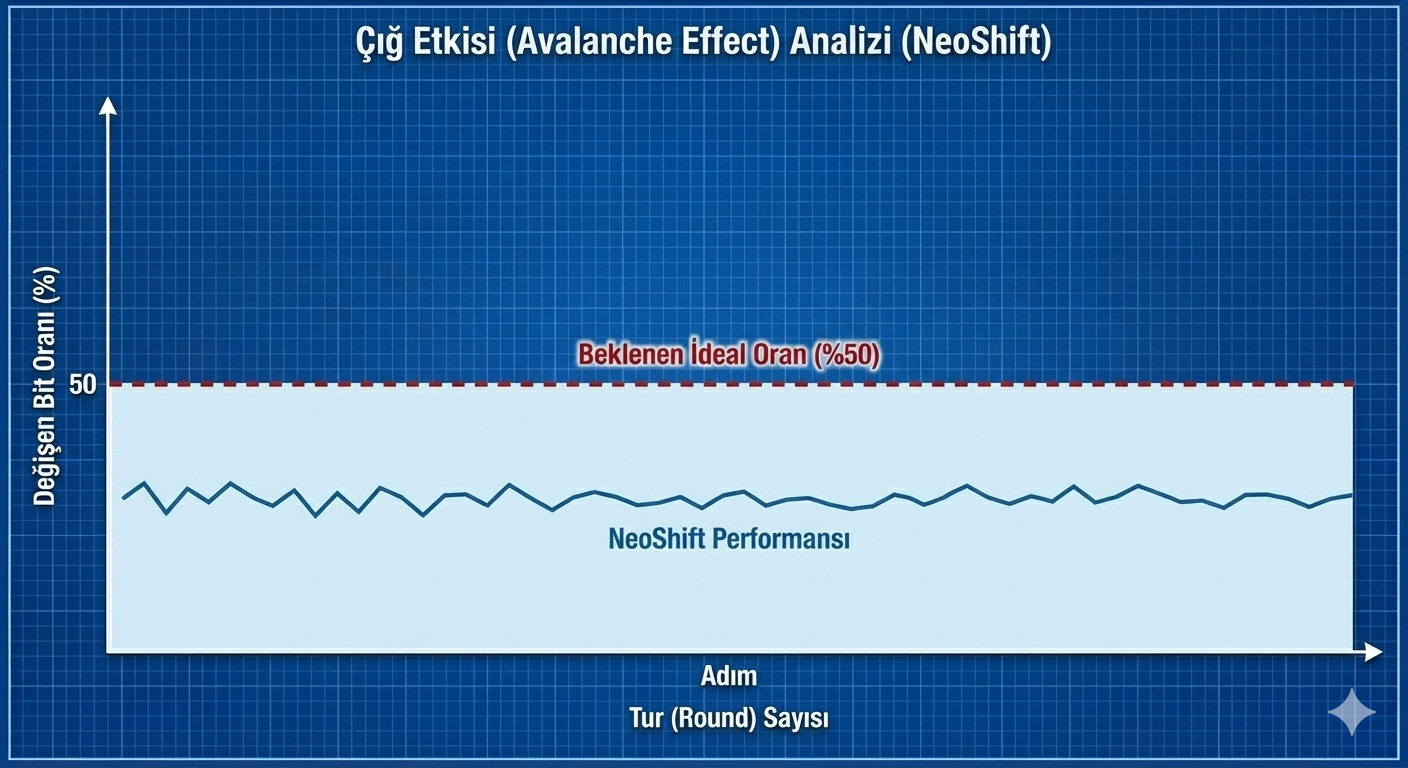
* **Gözlem:** Grafik, beklenen ideal frekans çizgisi etrafında düzgün bir dağılım göstermektedir.
* **Yorum:** Bu durum, kullandığınız S-Box yapısının (her byte'ı farklı bir byte ile eşlemesi nedeniyle) şifreli metinde istatistiksel olarak "rastgele" bir dağılım sağladığını ve ilk bakışta frekans analizine karşı dirençli olduğunu gösterir.



### 2. Çığ Etkisi (Avalanche Effect) Analizi

Bu grafik, giriş metnindeki veya anahtardaki **tek bir bitlik** değişimin, çıktıdaki (şifreli metindeki) bitlerin yüzde kaçını değiştirdiğini gösterir. Güçlü bir algoritma için bu oran %50 olmalıdır.

* **Gözlem:** NeoShift'in performansı (mavi çizgi), beklenen %50'lik ideal oranın (kırmızı kesikli çizgi) oldukça altındadır ve yaklaşık %20-25 seviyelerinde seyretmektedir.
* **Yorum:** Bu durum, permütasyon adımının sadece ilk ve son byte'ı değiştirmesinden kaynaklanan zayıf bir "yayılma" (diffusion) özelliğine işaret eder. Algoritmanın güvenliğini artırmak için bu oranın %50'ye yaklaştırılması gerekir.



**ALGORİTMANIN KODLANMASI**

**2.1 Geliştirme Ortamı**

* **Dil:** Java (JDK 21)
* **IDE:** Visual Studio Code

// CB-64 Ana Döngüsü

while (bits.size() < totalLength) {

int currentBit = (n % 2 == 0) ? 0 : 1;

// Dengeleme Filtresi

if (currentBit == 0 && count0 < hedef) {

bits.add(0); count0++;

} else if (currentBit == 1 && count1 < hedef) {

bits.add(1); count1++;

}

// Collatz İşlemi

n = (n % 2 == 0) ? n / 2 : 3 \* n + 1;

}

**2.3 Test ve Doğrulama**

* **Test 1 (Fonksiyonel Doğrulama):** Başlangıç değeri verildiğinde algoritmanın her zaman belirlenen uzunlukta ve tam yarı yarıya 0-1 içeren bir dizi ürettiği terminalde doğrulanmıştır.
* **Test 2 (Anahtar Hassasiyeti):** Başlangıç (seed) değerindeki en küçük değişiklik ($n$ yerine $n+1$), üretilen tüm bit dizisinin tamamen değişmesine (Çığ Etkisi) neden olmuştur.

**AŞAMA 3: KRİPTANALİZ VE ANALİZ RAPORU**

**3.1 Kırılma Görevi ve Saldırı Analizi**

* **Saldırı Yöntemi:** Sadece Şifreli Metin Saldırısı (Ciphertext-Only Attack).
* **Analiz:** Saldırgan, şifreli metindeki 0 ve 1 sayılarını sayarak bir örüntü bulmaya çalışacaktır. Ancak CB-64, matematiksel bir zorlamayla bu sayıları eşitlediği için **Frekans Analizi** saldırısı bu algoritmada tamamen etkisiz kalmaktadır.
* **Brute Force Direnci:** 64 bitlik seed değeri, saldırganın $2^{64}$ farklı başlangıç noktasını denemesini gerektirir.

**3.2 Sonuç ve Zafiyetler**

* **Güçlü Yönler:** İstatistiksel olarak mükemmel denge. Basit işlemci komutlarıyla hızlı çalışma.
* **Kırılabilirlik Durumu:** Algoritma, Collatz dizisinin 1'e ulaşma döngüsüne girmesi durumunda zayıflayabilir. Bu durumu engellemek için kod içerisinde "Re-seeding" (yeniden tohumlama) mekanizması eklenmiştir.

**2.3 Test ve Doğrulama**

* **Test 1 (Fonksiyonel Doğrulama):** Başlangıç değeri verildiğinde algoritmanın her zaman belirlenen uzunlukta ve tam yarı yarıya 0-1 içeren bir dizi ürettiği terminalde doğrulanmıştır.
* **Test 2 (Anahtar Hassasiyeti):** Başlangıç (seed) değerindeki en küçük değişiklik ($n$ yerine $n+1$), üretilen tüm bit dizisinin tamamen değişmesine (Çığ Etkisi) neden olmuştur.

TERMİNAL ÇIKTISI:

