BİTİRME PROJESİ

Haftalık Rapor – 26.11.2021

26 KASIM 2021

KIRIKKALE ÜNİVERSİTESİ – BİLGİSAYAR MÜHENDİSLİĞİ İÖ

AHMET MUNGAN – 160255081

İÇİNDEKİLER

ÖZET

LSB yönteminin teknik alt yapısı ve LSB algoritmasının gerçeklenmesi incelenmiştir. Bu algoritmanın incelenmesi pseudo kod ile başlayıp, python programlama ile gerçeklenmesine kadar olan tüm süreçleri kapsamaktadır. Ayrıca LSB algoritmasına dair hazır bir program kodunun tersine mühendislik ilke ve metotları ile incelenmesi yapılmıştır.

LSB DAMGALAMA YÖNTEMİNİN ALGORİTMASI

Geçen hafta itibariyle LSB damgalama yönteminin ve içerisinde barındırdığı kavramlar ve bu kavramlara karşılık gelen tanımlar araştırılmıştır. Bu yöntemin nasıl kullanılacağı, hangi şartlar altında nasıl sonuçlar verdiği incelenecektir. Bu çerçevede tanımların yapılmadığı, teknik detaylar aracılığı ile bu yöntemin uygulanabilirliği kanıtlanacaktır.

Öncelikle LSB damgalama yönteminde damgalama için kullanılan LSB'lerin sayısı arttıkça (LSB derinliğinin artması olarak da bilinir) gömülü şifrenin/mesajın gauss dağılımının sunduğu istatistiksel bilgilere göre algılanabilirliği artar [1]. Dolayısıyla stegonografi ile elde edilen nesnelerin tespit edilebilirliği artar. Bu sebeple birim ses verisi üzerine düşen damgalanacak bit sayısına ait bir sınır konulması gerekmektedir. Bu sınır, programın gauss gürültü istatistiklerini öğrendiği (makine öğrenmesi) noktayla sınırlandırılır. Sınırın belirlenmesinde laboratuvardaki belirli deneyler ile gerçekleştiği söylenebilir. Deneyler: Müzik ve ses deneyimi olan kişiler tarafından büyük bir ses veri setinin bulunduğu bir ortamda yapıldığı söylenmektedir [1]. Bu deneyler ses veri dizisi başına 16 bitlik bir damgalama yapıldığı takdirde LSB'nin dördüncü katmanında damgalama yapılabildiği ve algısal olarak fark edilemediği ortaya çıkmıştır. Dördüncü katmandan LSB yapılması algısal farkındalık yaratmamıştır fakat, ses verisinin içerisinde barındırdığı unsurlara göre[1] belirlenmiş olan dördüncü katmandan LSB için sınırları genişletmiştir. Bu gibi unsurları içeren ses verilerinin aslında daha esnek bir damgalama ortamına sahip olacağı incelenen çalışmalarda gözlemlenmiştir.

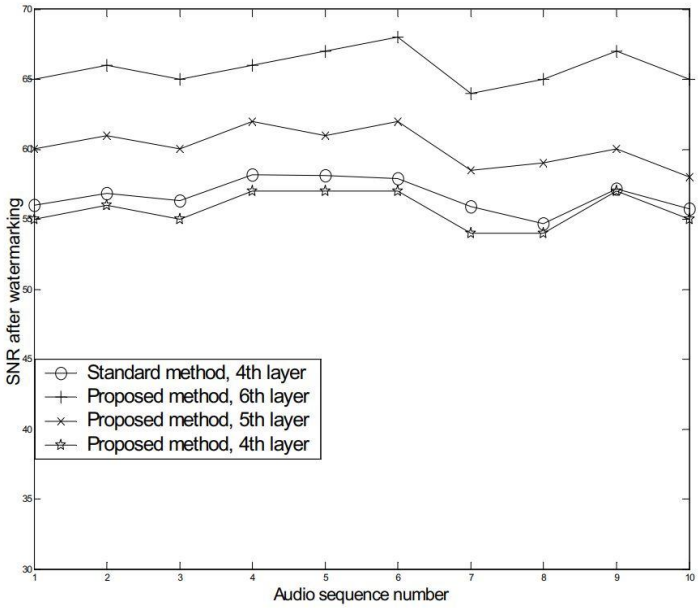
İncelenen yazılarda bulgulardan bir diğeri ise bahsi geçen dördüncü katmandan LSB yöntemini farklı bir algoritma üzerinden gerçekleştirilmesiyle SNR testlerinde daha olumlu sonuçlar alındığı gözlemlenmiştir. Bu çerçevede kullanılan pseudo kod yapısı aşağıdaki gibidir:

[1] Bu unsurlar genellikle sesin müziksel özelliklerinde, farklı tarzda müziklerin varoluşsallığından gelmektedir.

Pseudo 1

```
if kesit örneği a >= 0
   if bit 0 olarak gömülü ise
     if a-1 = 0 then a-1a-2...a0 = 11...1
     if a-1 = 1 then a-1a-2...a0 = 00...0 and
       if a+1 = 0 then a+1 = 1
         else if a+2 = 0 then a+2 = 1
         ...
         else if a15 = 0 then a15 = 1
   else if bit 1 olarak gömülü ise
     if a-1 = 1 then a-1a-2...a0 = 00...0
     if a-1 = 0 then a-1a-2...a0 = 11...1 and
       if a-1 = 1 then a+1 = 0
         else if a+2 = 1 then a+2 = 0
         ...
         else if a15 = 1 then a15 = 0
if kesit örneği a < 0
     if bit 0 olarak gömülü ise
     if a-1 = 0 then a-1a-2...a0 = 11...1
     if a-1 = 1 then a-1a-2...a0 = 00...0 and
       if a+1 = 1 then a+1 = 0
         else if a+2 = 1 then a+2 = 0
         ...
         else if a15 = 1 then a15 = 0
   else if bit 1 olarak gömülü ise
     if a-1 = 1 then a-1a-2...a0 = 00...0
     if a-1 = 0 then a-1a-2...a0 = 11...1 and
       if a-1 = 1 then a+1 = 0
         else if a+2 = 1 then a+2 = 0
         ...
         else if a15 = 1 then a15 = 0
```

Pseudo 1'de bu metodun incelendiği kaynaktan detay kısımları elimine edilerek ve türkçeleştirip anlamlı hale getirilerek sözde kod yazılmıştır [1].



Şekil 1 [1]

Şekil 1'de [1] ise Pseudo 1'deki gibi yazılan ve çalışan algoritmanın SNR çıktısı verilmiştir. Standart metotun dışına çıkıldığı Pseudo 1'deki algoritma ile belirlenmiş olsa da, aslında çok ufak farklar olduğu Şekil 1'de görülmektedir. Dolayısıyla incelenen dökümanlarda verilen sözde kodların, klasik dördüncü katmandan LSB yönteminden daha verimli olduğu söylenebilir.

LSB ALGORİTMASINA AİT PROGRAM KODUNUN TERSİNE MÜHENDİSLİK YARDIMIYLA GERÇEKLENMESİ

Bu programın temel mantığı geçen haftalarda ve bu raporun algoritmanın teknik alt yapısının anlatıldığı birince bölümden yola çıkarak oluşturulmuştur. LSB algoritması için dördüncü katmandan uygulamalar programa dökülebilir fakat programın birinci katmandan LSB algoritmasına uyması bir sakınca teşkil etmemektedir. Eğer birinci katmandan LSB algoritması çalıştırılabilir (programlanabilir) bir biçimde yazılırsa; dördüncü katmandan LSB algoritması farklı kural setleri ile itere edilmiş bir mantığa dayandığı için aralarında büyük farklar yoktur. Bu farkı belirleyen veri setindeki değişimler olabilir. Eğer bir veri seti üzerinde birinci katmandan LSB algoritması yavaş çalışıyorsa, kural setleri ve 4 sefer itere edilmiş şekilde dördüncü katmandan çalıştırmak algoritmanın çok daha yavaş çalışmasına neden olabilir. Bu sebeple; veri setinden bağımsız, bahsi geçen tekniklerin ön planda tutulduğu, kural setleri ve dörtlü iterelerin bulunmadığı safi bir program kodunun varlığı gerekmektedir.

Bu programın yazımı için python kullanılarak, hazır modüller aracılığı ile import ve export işlemleri ile veri seti alınabilir. Ayrıca bitsel işlemleri python üzerinde yapmak gerekmektedir [2]. Fakat literatür araştırması yapılarak LSB algoritmasının python dili üzerinde gerçekleyen birçok kaynağın varlığından söz edilebilir. Fakat LSB algoritmasının direkt bir şekilde kodlanmasına nazaran, kodlamasına tersine yaklaşılacaktır. Bu sayede gereksinimler gibi tespiti kimi zaman zorlaşan durumların aşılması için kolaylık sağlayacağı kesindir.

Kodun tersine incelenmesi veya tersine mühendislik yöntemleri ile incelenmesi demek, birçok programcının pratikte kullandığı bir yöntemdir. Ön oluşturmasız (pre-ungenerate) olarak bilinen, son yıllarda IDE'lerin[2] de desteklediği bir yöntemdir. Örnek vermek gerekirse, bir IDE'de programlama dilleri için ayrılmış anahtar kelimeler (reserved keywords) kullanılmadan herhangi bir kelime yazılıp bunun bir metot olduğunu belirten parantezlerin açılıp kapanması yapılmış olsun. IDE, yazılan

[2] Visual Studio, Apache NetBeans, IntelliJ IDEA ve türevlerine bağımlı uygulama uzantıları bu IDE'ler için örnek olarak verilebilir.

kelimenin ana programa dahil edilen kütüphanelerde olup olmadığına bakmaksızın düzenleme seçeneklerinde bu metotun oluşturulması önerilir. Dolayısıyla bir alt programın yerine getirmesi gereken görevlerden önce, bu alt program ana programda tanımlanır ve daha sonradan bu alt program oluşturulur.

Bahsi geçen yönteme göre incelenirse:

Code 1

```
>>> lsbDamgala()
>>> lsbDesifrele()
```

Code 1'deki gibi 2 temel fonksiyon üzerinde algoritma çalışacaktır. Bu fonksiyonların içeriği doldurulacaktır. İhtiyaç halinde algoritmaya yönelik üretimler olacaktır.

Code 2

```
>>> lsbDamgala(ses, mesaj, cikti)
>>> lsbDesifrele(cikti)
```

Code 2'de bu fonksiyonların temelde barındırması gereken parametreler belirlenmiştir.

Code 3

```
>>> def lsbDamgala(ses, mesaj, cikti):
>>>     mesaj = str(mesaj)
>>>     damga = struct.unpack("%dB" % len(mesaj), mesaj)
>>>     damgaBoyutu = len(damga)
```

Code 3'te damgalama fonksiyonunun içeriği kodlanmaya başlanmıştır. Code 3'te dikkati çeken yapısal bir düzenlemeye gidildiğinden haricen eklenmesi gereken struct modülü eklenmesi gereklidir. İşte tam olarak tersine kod yazımı yani kodun yerine göre üretimi bunu içermektedir. Dolayısıyla struct modülünü bilmeyen bir kişi tersine bir geliştirme yapamaz.

Code 4

```
>>> import struct
```

Bahsi geçen gereksinimlerin import edilmesi Code 4'teki gibi diğer tüm gereksinimler için de yapılmıştır.

Code 5

```
>>> def lsbDamgala(ses, mesaj, cikti):
>>>     varsayilanBit = 32
>>>     mesaj = str(mesaj)
>>>     damga = struct.unpack("%dB" % len(mesaj), mesaj)
>>>     damgaBoyutu = len(damga)
>>>     damgaBitleri = bitleriDamgala(damgaBoyutu, varsayilanBit)
```

Code 5'te fonksiyonun değişkenlerine eklemeler yapıldı ve `bitleriDamgalama` isminde yeni bir fonksiyonun varlığından söz edilebilir.

Code 6

```
>>> def bitleriDamgala(damgaBoyutu, nBit = 8):
>>>     damgaBitleri = []
>>>     for byte in damgaBoyutu:
>>>         for i in range(0, nBit):
>>>             damgaBitleri.append((byte & (2 ** i)) >> i)
>>>     return damgaBitleri
```

Code 6'da bitleriDamgala fonksiyonunun içeriği verilmiştir.

Code 7

```
>>> def lsbDamgala(ses, mesaj, cikti):
>>>     varsayilanBit = 32
>>>     mesaj = str(mesaj)
>>>     damga = struct.unpack("%dB" % len(mesaj), mesaj)
>>>     damgaBoyutu = len(damga)
>>>     damgaBitleri = bitleriDamgala((damgaBoyutu, ), varsayilanB
it)
>>>     damgaBitleri.extend(bitleriDamgala(damga))
>>>     sesFiltresi = wave.open(ses, 'rb')
>>>     (kanallar, kesitUzunlugu, cerceveOrani, nCerceve, tip, isi
m) = sesFiltresi.getparams()
>>>     cerceveler = sesFiltresi.readframes(nCerceve * kanallar)
>>>     kesitler = struct.unpack_from("%dh" % nCerceve * kanallar,
cerceveler)
>>>     if len(kesitler) < len(damgaBitleri):
>>>         print("Damga verisi kesitten daha büyük olamaz.")
>>>     sifreliKesitler = []
>>>     damgaKonumu = 0
>>>     for kesit in kesitler:
>>>         sifreliKesitler = kesit
>>>         if damgaKonumu < len(damgaBitleri):
```

```
>>>             sifreliBit = damgaBitleri[damgaKonumu]
>>>             if sifreliBit == 1:
>>>                 sifreliKesit = kesit | sifreliBit
>>>             else:
>>>                 sifreliKesit = kesit
>>>                 if kesit & 1 != 0:
>>>                     sifreliKesit = kesit - 1
>>>             damgaKonumu += 1
>>>         sifreliKesitler.append(sifreliKesit)
>>>     sifreliSes = wave.open(cikti, 'wb')
>>>     sifreliSes.setparams((kanallar, kesitUzunlugu, cerceveOran
i, nCerceve, tip, isim))
>>>     sifreliSes.writeframes(struct.pack("%dh" % len(sifreliKesi
tler), *sifreliKesitler))
```

Süreç yukarıdaki tersine kod geliştiriciliği ile devam ettiği takdirde, damgalama fonksiyonu Code 7'de verilmiştir.

Code 8

```
>>> def lsbDesifrele(cikti):
>>>     varsayilanBit = 32
>>>     damgaliSes = wave.open(cikti, 'rb')
>>>     (kanallar, kesitUzunlugu, cerceveOrani, nCerceve, tip, isi
m) = damgaliSes.getparams()
>>>     cerceveler = damgaliSes.readframes(nCerceve * kanallar)
>>>     kesitler = struct.unpack_from("%dh" % nCerceve * kanallar,
cerceveler)
>>>     damgaByte = 0
>>>     for (kesit, i) in zip(kesitler[0:varsayilanBit], range(0,
varsayilanBit)):
>>>         damgaByte += (kesit & 1) * (2 ** i)
>>>     damgaVerisi = []
>>>     for n in range(0, damgaByte):
>>>         damgaByteKesitleri = kesitler[varsayilanBit + (n * 8)
: varsayilanBit + ((n + 1) * 8)]
>>>         damgaByte = 0
>>>         for (kesit, i) in zip(damgaByte, range(0, 8)):
>>>             damgaByte += (kesit & 1) * (2 ** i)
>>>         damgaVerisi.append(damgaByte)
>>>     return damgaVerisi
```

Code 7'de damgalanan ses Code 8'de deşifrelemesi yapılmıştır.

Code 9

```
>>> def damgaStr(liste):
>>>     return "".join([chr(x) for x in liste])
```

Code 9'de damganın string halini oluşturan bir fonksiyon yazılmıştır.

Code 10

```
>>> def gomuDosyasi(ses, gizliDosya, cikti):
>>>     dosya = open(gizliDosya)
>>>     gizliDosya = dosya.read()
>>>     lsbDamgala(ses, gizliDosya, cikti)
```

Gömü dosyası için dosya işlemleri Code 10'da verilmiştir.

Code 11

```
>>> def desifreDosyasi(sifreliSinyal, gizliVeriKonumu):
>>>     damga = lsbDesifrele(sifreliSinyal)
>>>     damgaStr = damgaStr(damga)
>>>     dosya = open(gizliVeriKonumu, "W")
>>>     dosya.write(damgaStr)
```

Code 11'da deşifre için gerekli bilgilerin barındırıldığı dosya işlemlerini gerçekleştiren fonksiyon verilmiştir.

Code 12

```
>>> ses = ".../sesler/ses.mp3"
>>> output = "x.wav"
>>> if len(sys.argv) > 1:
>>>     mesaj = sys.argv[1]
>>>     if len(sys.argv) > 2:
>>>         ses = sys.argv[2]
>>>         if len(sys.argv) > 3:
>>>             cikti = sys.argv[3]
```

Code 2'de main fonksiyonunda temel fonksiyonlar üzerinden oluşan yapıya ek olarak Code 12'deki parametreler verilmiştir.

Bu kısımda unutulmaması gereken bir nokta vardır. Bu program kodu veri setinin içeriği dikkate alınmadan genel-geçer bir algoritmaya dayanır. Dolayısıyla tüm veri setleri için geçerli olmayacağı gibi, veri setlerinden de ayrı değerlendirmemek gerekmektedir. Bu kapsamda veri setinde farklı bir depolama (graf tabanlı, lineer tabanlı vs.) tekniği kullanılmış ise, bu duruma uygun şekilde algoritma spesifike edilmesi gerekir. Bu durumdan doğacak sonuçlarda ise veri setine bağımlı bir şekilde program kodunun değişkenlik göstermesi beklenir.

REFERANS VE KAYNAKÇA

[1] Cvejic, Nedeljko, and Tapio Seppanen. "Increasing robustness of LSB audio steganography using a novel embedding method." International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.. Vol. 2. IEEE, 2004.

[2] Python bitwise operators, Python Wiki. Link için tıklayınız.

[3] Python struct documents. Link için tıklayınız.

[4] Python sys documents. Link için tıklayınız.

[5] Python wave documents. Link için tıklayınız.

EKLER

Raporun tersine mühendislik kısmında yazılan kodun github linki için tıklayınız.