Anahtarlı Boole Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteçleri için Gelişmiş Saldırı Yöntemi

Bu tez Bilgi Güvenliği Mühendisliği'nde
Tezli Yüksek Lisans Programının bir koşulu olarak

İlker Yasin YILDIZ

tarafından

Fen Bilimleri Fakültesi'ne
sunulmuştur.

İSTANBUL
ŞEHİR
ÜNİVERSİTESİ

Bu tezi okuduk, kapsam ve nitelik açısından Bilgi Güvenliği Mühendisliği alanında Yüksek Lisans derecesi için tümüyle uygun olduğu görüşüne vardık.

ONAYLAYANLAR:

Prof. Dr. Ensar Gül
(Tez Danışmanı) .....

Doç. Dr. Orhun Kara
(Tez Eş-danışmanı) .....

Prof. Dr. Erkay Savaş .....

Doç. Dr. Alptekin Küpçü .....

Dr. Öğretim Üyesi Erdinç Öztürk .....

Bu tez İstanbul Şehir Üniversitesi, Fen Bilimleri Fakültesi tarafından belirlenen tüm koşullara uygundur.

ONAY TARİHİ: 23.07.2019

MÜHÜR/İMZA:

Yazarlık Beyanı

Ben, İlker Yasin YILDIZ, başlığı, 'Anahtarlı Boole Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteçleri için Gelişmiş Saldırı Yöntemi ' olan tezin ve içinde sunulan bilgilerin şahsıma ait olduğunu beyan ederim. Ayrıca:

- Bu çalışmanın bütünü veya esası bu üniversitede Yüksek Lisans derecesi elde etmek üzere çalıştığım süre içinde gerçekleştirilmiştir.
- Daha önce bu tezin herhangi bir kısmı başka bir derece veya yeterlik almak üzere bu üniversiteye veya başka bir kuruma sunulduysa bu açık biçimde ifade edilmiştir.
- Başkalarının yayımlanmış çalışmalarına başvurduğum durumlarda bu çalışmalara açık biçimde atıfta bulundum.
- Başkalarının çalışmalarından alıntıladığımda kaynağı her zaman belirttim. Tezin bu alıntılar dışında kalan kısmı tümüyle benim kendi çalışmamdır.
- Esaslı yardım aldığım bütün kaynaklara teşekkür ettim.
- Tezde başkalarıyla birlikte gerçekleştirilen çalışmalar varsa onların katkısını ve kendi yaptıklarımı tam olarak açıkladım.

İmza: 

Tarih: 23.07.2019

"Engineers like to solve problems. If there are no problems handily available, they will create their own problems."

Scott Adams

An Improved Attack on Keystream Generators with Boolean Keyed Feedback Function

İlker Yasin Yıldız

Abstract

Ultra-lightweight stream ciphers are highly optimized variation of stream ciphers for miniscule hardwares with limited power and calculation resources such as RFID product tags used in retail marketing and Wireless Sensor Network components that are indispensable part of modern SCADA systems.

In FSE 2015, Armknecht and Mikhalev presented a unique ultra-lightweight stream cipher design approach defined as *Keystream Generators with Keyed Update Function (KSG with KUF)* along with a concrete cipher Sprout [1]. This design approach used by recent stream ciphers such as Fruit [2] and Plantlet [3], promises to make use of secret key during state updates in order to maintain security level as well as shorten internal state size to reduce hardware area in conjunction with power consumption. In 2018, definition of KSG with KUF is narrowed by Kara and Esgin [4], with new definition *Keystream Generators with Boolean Keyed Feedback Function (KSG with Boolean KFF)*, on which a generic scope trade-off attack is also mounted. This attack relies on guess capacity definition given in the same article, to eliminate wrong states during exhaustive search operation.

In this thesis, we examined this generic Kara and Esgin attack in-depth and accelerated by a factor up to about 60 times. In order to accomplish this speedup, a new guess capacity definition and sieving method are introduced in addition to the improved algorithm which contributes efficiency of the attack in both performance and stability. Improvements are validated with intense performance tests comprising nearly twenty sample feedback functions, including Sprout, with diverse existence of guess capacities.

Keywords: stream cipher, keyed update, ultra-lightweight, cryptanalysis, sprout, guess and determine

Anahtarlı Boole Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteçleri için Gelişmiş Saldırı Yöntemi

İlker Yasin Yıldız

Öz

Ultra-hafifsiklet dizi şifreleme algoritmaları, perakende sektöründe kullanılan RFID ürün etiketleri ve modern SCADA sistemlerinin vazgeçilmez parçası olan Kablosuz Sensör Ağı bileşenleri gibi kısıtlı kaynaklara sahip küçük boyutlu cihazlar için özelleştirilmiş dizi şifreleme algoritmalarıdır.

FSE 2015 etkinliğinde Armknecht ve Mikhalev ultra-hafifsiklet dizi şifreleme örneği Sprout [1] ile birlikte *Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteci (AGF-KAÜ)* adını verdikleri yeni bir tasarım yöntemi sundu. İç durumların gizli anahtar kullanılarak güncellenmesini öngören bu özgün yöntem, Sprout baz alınarak geliştirilen Plantlet [3] ve Fruit [2] gibi güncel algoritmalarda da kullanılmıştır. AGF-KAÜ tanımı 2018 yılında Kara ve Esgin tarafından daraltılarak *Anahtarlı Boole Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteci (ABGBF-KAÜ)* olarak yeniden tanımlanmış ve bu sınıfa uyumlu tüm algoritmaları etkileyen genel ölçekli bir saldırı geliştirilmiştir [4]. Bu saldırı, aynı çalışmada tanımı yapılan *Tahmin Kapasitesi* ve *Ortalama Tahmin Kapasitesi* kavramları kullanılarak tahmin edilebilirliği yüksek iç durumların tespiti üzerine kuruludur.

Çalışmamızda bu saldırı incelenerek birden fazla yöntemle geliştirilmiş ve 60 kata kadar hızlandırılmıştır. İlk geliştirmemizde yeni bir *Ortalama Tahmin Kapasitesi* tanımı yapılmış ve yeni algoritma buna bağlı olarak tasarlanmıştır. Yeni *Ortalama Tahmin Kapasitesi* tanımını kullanan algoritma, geri besleme değerinin tahmin edilmesinde fayda sağlamayan iç durumlara ait tahmin kapasitelerini atlayarak saldırıyı etkin hale getirmektedir. Bununla birlikte ikinci geliştirme olarak saldırının eliminasyon süreci etaplara bölünerek yanlış iç durumların daha kısa sürede elenmesi sağlanmıştır. Geliştirmelerimiz, tahmin kapasiteleri değişiklik gösteren, Sprout dahil yirmiye yakın örnek ile yoğun bir test sürecine tabi tutulmuş ve geçerliliği onaylanmıştır.

Anahtar Sözcükler: dizi şifreleme, anahtarlı güncelleme, ultra-hafifsiklet, kriptanaliz, sprout, tahmin et ve belirle

Eğitim hayatımın en büyük destekçisi, rahmetli dedem

Bekir YILDIZ'a adanmıştır

Teşekkür

- Bu tezi hazırlayabilmem için gerekli matematiksel temelin sağlanmasında büyük emeği geçen kıymetli danışman hocam Doç. Dr. Orhun KARA'ya en derin teşekkürlerimi iletiyor ve minnettarlığımı sunuyorum.
- Öğrenim sürem boyunca desteklerini esirgemeyen kıymetli danışman hocam Prof. Dr. Ensar Gül'e teşekkürlerimi sunuyorum.
- Tez çalışmamı inceleyip daha iyi olması için görüşlerini bildiren Prof. Dr. Erkay Savaş'a teşekkürlerimi iletiyorum.
- Tez çalışmamı inceleyip daha iyi olması için görüşlerini bildiren Doç. Dr. Alptekin Küpçü'ye teşekkürlerimi iletiyorum.
- Tez çalışmamı inceleyip daha iyi olması için görüşlerini bildiren Dr. Öğretim Üyesi Erdinç Öztürk'e teşekkürlerimi iletiyorum.
- Bu yüksek lisans programında öğrenim görmem için gerekli şartları ve imkânları sunan TÜBİTAK BİLGEM ve yönetim kadrosuna teşekkürlerimi iletiyorum.
- Tezimi hazırlamamda manevi desteğini esirgemeyen ve beni cesaretlendiren kıymetli arkadaşım Merve DOĞRU'ya en derin sevgilerimi ve teşekkürlerimi sunuyorum.
- Öğrenim sürem boyunca maddi ve manevi desteklerini esirgemeyen annem Feruzan YILDIZ ve babam Necdet YILDIZ'a en derin sevgi ve minnettarlığımı sunuyorum.
- Tezi hazırladığım süre zarfında şahsım için en çok endişelenen ve manevi desteğini esirgemeyen babaannem Şürkiye YILDIZ'a en içten sevgi ve minnettarlığımı iletiyorum.

İçindekiler

Şekil Listesi

Tablo Listesi

Kısaltmalar

KSG with **KUF**	**K**ey**s**tream **G**enerator with **K**eyed **U**pdate **F**unction
KSG With Boolean **KFF**	**K**ey**s**tream **G**enerator with Boolean **K**eyed **F**eedback **F**unction
LFSR	**L**inear **F**eedback **S**hift **R**egister
NLFSR	**N**on-**L**inear **F**eedback **S**hift **R**egister
RKF	**R**ound **K**ey **F**unction
KE	**K**ara ve **E**sgin
RFID	**R**adio **F**requency **ID**entification
EEPROM	**E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory
WSN	**W**ireless **S**ensor **N**etworks
IV	**I**nitialization **V**ector
OTP	**O**ne **T**ime **P**ad

Sözlükçe

LFSR	Lineer Geri Beslemeli Ötelemeli Saklayıcı
NFSR/NLFSR	Lineer Olmayan Geri Beslemeli Ötelemeli Saklayıcı
Stream Cipher	Dizi Şifreleme
Asynchronous Stream Cipher	Eşzamanlı Olmayan Dizi Şifreleme Algoritması
Synchronous Stream Cipher	Eşzamanlı Dizi Şifreleme Algoritması
Block Cipher	Blok Şifreleme
Key	Anahtar
Internal State	İç Durum
Initial Internal State	İlk İç Durum
Throughput	Çıktı Hızı
Clock	Saat Vuruşu
Clocking	İşletme (Mantıksal Devrenin İşletilmesi)
Operational Clock Frequency	Operasyonel Saat Frekansı
Round Key Function	Anahtar Çevrim Fonksiyonu
Initialization Vector	İlklendirme Vektörü
One Time Pad	Tek Seferlik Şifre
Wireless Sensor Networks	Kablosuz Sensör Ağları
Radio Frequency Identification	Radyo Frekanslı Kimlik Belirleme
Binary Additive Encryption	İkili Toplamalı Şifreleme
Propagation Delay	Yayılım Gecikmesi
Keyed Update Function	Anahtarlı Güncelleme Fonksiyonu
Boolean Keyed Feedback Function	Anahtarlı Boole Geri Besleme Fonksiyonu

Bölüm 1

Giriş

Bu bölümde tez çalışmamızla ilgili temel bilgileri içeren bir giriş yapılmış ve konumuza ilişkin çalışmalardan bahsedilmiştir. Daha sonra, tezde yapılan yeni çalışma ve literatüre katkılarımız anlatılmıştır. Son olarak, 8 bölümden oluşan tez çalışmamızdaki her bir bölümün içeriğine kısaca değinilmiştir.

1.1 Motivasyon

Yükselen teknoloji ile birlikte haberleşme, bilgi depolama ve arşivleme yöntemleri zaman içerisinde gelişirken, bilginin mahremiyeti ve gizliliğinin korunmasına dair gereksinim de aynı oranda artmıştır. Kriptografi, son derece kritik önem arz eden bu gizlilik ve mahremiyetin korunması için matematiksel yöntemler kullanılarak bilginin yetkisiz üçüncü şahısların hiçbir mana veremeyeceği karmaşık bir hâle getirilmesi konusunda yöntemler geliştiren bir bilim dalıdır. Bu matematiksel yöntemler şifreleme algoritması veya şifreleyici olarak adlandırılır. Kriptografinin başlıca kullanım alanları arasında bilgisayar ağları, internet güvenliği, baz istasyonları, sayısal depolama, akıllı kart teknolojileri, uydu haberleşmesi ve sayısal yayıncılık gibi alanlar bulunmaktadır.

Şifreleme işleminde anahtar adı verilen gizli ek bilgi kullanılmaktadır. Şifreleme ve şifre çözme işleminde aynı anahtar kullanılıyorsa buna Simetrik Şifreleme; farklı anahtarlar kullanılıyorsa Asimetrik Şifreleme denir. Simetrik Şifreleme iki türdür: Blok Şifreleme ve Dizi Şifreleme. Blok Şifreleme, bilginin eşit parçalara bölünerek anahtarla ayrı ayrı şifrelenmesi üzerine kuruludur. Tez konumuzla ilgili Dizi Şifreleme ise işleyeceği bilgiyle

aynı büyüklükte anahtar üreterek bir bütün hâlde şifreler. Bu tek parça anahtara Tek Kullanımlık Şifre (One Time Pad) adı verilir. Kayan Anahtar Üreteçleri olarak da bilinen dizi şifreleme algoritmaları günümüzde ulaşım ve kimlik kartlarında, perakendecilik sektöründe kullanılan ürün etiketlerinde, kablosuz sensör ağlarının (WSN) güvenliğinde, baz istasyonlarında, radyo frekanslı kimlik belirleme (RFID) cihazlarında kullanılmaktadır.

Ultra-hafifsiklet dizi şifreleme algoritmaları, hesapsal kabiliyeti zayıf ve güç gereksinimi çok düşük RFID ve WSN cihazları için özelleştirilmiş, gerçeklemelerinin kapladığı donanım alanı olarak mantıksal kapı eşdeğeri son derece az (örneğin 1000 GE'den daha düşük) algoritmalardır. Bununla birlikte, bir dizi şifreleme algoritmasının geleneksel Zaman-Bellek-Veri ödünleşim (TMD trade-off) saldırılarına karşı dirençli olması için, iç durumun anahtar uzunluğundan en az iki kat büyük olması gerekmektedir [9–12]. Tüm bu kısıtlar göz önünde bulundurulduğunda güvenli bir ultra-hafifsiklet dizi şifreleme algoritması tasarlamak çok zorlu bir iş olarak karşımıza çıkmaktadır. ECRYPT organizasyonu tarafından düzenlenen, 2004'ten 2008'e kadar süren eSTREAM projesi kapsamında kazananlar olarak Trivium [5], Mickey v2 [13] ve Grain v1 [14] gibi hafifsiklet dizi şifreleme algoritmaları tanıtılmıştı. Diğer taraftan, hafifsiklet blok şifreleme türünde Present [15], Prince [16], Midori [17], LED [18], Piccolo [19], SIMON ve SPECK Ailesi [20], Simeck [21], KATAN ve KTANTAN Ailesi [22], Lblock [23], ITUbee [24] gibi çok sayıda örnek bulunmaktadır. Bu sebeple literatüre ultra-hafifsiklet dizi şifreleme algoritmaları için yapılacak katkılar yüksek önem arz etmektedir.

1.2 İlişkin Çalışmalar

2015 yılında İstanbul'da düzenlenen FSE etkinliğinde Armknecht ve Mikhalev'in sunduğu *Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteci* (AGF-KAÜ) yaklaşımı, anahtarın iki katı uzunluğunda iç durum kullanmadan da güvenli ultra-hafifsiklet algoritma tasarlanması iddiasıyla açıklığı kapatacak bir adım olmuştur [1].

Bu yaklaşımda, alışılagelmiş kayan anahtar üreteçlerinden farklı olarak iç durum güncelleme fonksiyonu anahtarı da girdi olarak kabul etmektedir. Eğer anahtar yongaya kazınmış halde sabitlenirse ve cihazın hayat döngüsü içinde hiç değiştirilmez ise, bu durumda anahtarı yonganın devre alanında neredeyse hiç yer kaplamayacak şekilde saklamak mümkün olabilir [1]. Diğer taraftan bu yaklaşımda iç durum boyu için ödünleşim

(trade-off) saldırılarına karşı dayanıklılık açısından bir kısıtlama söz konusu değildir. Dolayısıyla AGF-KAÜ'ler ile ultra-hafifsiklet dizi şifreleme algoritmaları tasarımının yolu açılmış oldu.

Türün vücuda bürünmüş ilk örneği olan Sprout algoritması kısa süre içerisinde Lallemand ve Naya-Plasencia [25], Zhang ve Gong [26] ve Kara&Esgin [27] saldırılarının hedefi olmasına karşın ana fikrin uygulanabilirliği ve genel yaklaşıma ait güvenlik analizleri günümüzde bu alanda ilgi çeken araştırma konuları arasındadır.

Sprout algoritmasındaki açıklık 2 yıl sonra düzeltilerek 2017 yılında Plantlet ismiyle yeniden sunuldu [3]. Plantlet iki yıldır literatürde olmasına rağmen henüz ona yapılmış başarılı bir saldırı yoktur. AGF-KAÜ türüne diğer bir örnek ise Ghafari vd. tarafından tasarlanan Fruit algoritmasıdır [2]. Tam çevrim kriptanalizi [28] yayınlandıktan sonra düzeltilmiş sürümü Fruit-80 olarak yeniden tasarlanmıştır [29]. Kara ve Esgin tarafından geliştirilmiş genel saldırı yöntemi [4] hariç, şu ana kadar düzenlenen saldırıların tümü belirli bir algoritma hedef alınarak gerçekleştirilmiştir.

Anahtarlı Boolean Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteci (ABGBF-KAÜ) sınıfı basitçe, her saat vuruşu başına 1 bit çıktı üreten ve geri besleme değerinin yalnızca 1 biti anahtardan etkilenen geri besleme fonksiyonuna sahip dizi şifreleme algoritmalarıdır. Bu kayan anahtar üreteci ailesinin net tanımı Bölüm 4'te bulunan Tanım 2'de verilmiştir. Tanım son derece ayrıntılı ve karmaşık gözükse de oldukça doğal bir tanımdır. Nitekim literatürde var olan tüm örnekler (Sprout [1], Plantlet [3], Fruit [2] vb. gibi) bu tanıma uymaktadır ve ABGBF-KAÜ sınıfına girmektedir.

Kara ve Esgin'in makalesinde [4] yer alan bir diğer tanım olan **Tahmin Kapasitesi** ise, belirli bir iç durumla üretilecek geri besleme değerinin anahtarı bilmeden doğru tahmin edilme olasılığı olarak geçmektedir [4]. Bu olasılık 0.5 ile 1 arasında bir değerdir. 0.5 değeri geri beslemeyi tahmin etmenin hiçbir avantaj sağlamadığını ifade ederken, 1 değeri anahtarı bilmeden geri besleme değerinin kesin olarak hesaplanabilmesi anlamına gelmektedir. Aynı çalışmada tanımlanan **Ortalama Tahmin Kapasitesi** ise olası tüm iç durumlara ait tahmin kapasitelerinin aritmetik ortalamasıdır. Ortalama Tahmin Kapasitesi yeterli düzeyde (0.5'ten büyük) olan algoritmalara, geri besleme ve çıktı fonksiyonları hakkında ayrıntılar bilinmese dahi saldırı uygulanabilmektedir [4]. Bu saldırının en temel özelliği genel bir saldırı olması ve ortalama tahmin kapasitesi 0.5'den büyük olan

her türlü ABGBF-KAÜ'ye, saldırının çıktı fonksiyonuna ya da güncelleme fonksiyonuna bakılmaksızın uygulanabilir olmasıdır.

ABGBF-KAÜ'lere Yapılan Genel Saldırı Algoritması

Saldırı algoritması, Ortalama Tahmin Kapasitesi'ne bağlı olarak hesaplanan *Sonlandırma Değeri* uzunluğunda bir kayan anahtar çıktısı alarak iç durum uzayını taramaya başlar. Aynı zamanda her bir iç durum için bir cezalandırma/uyuşmazlık sayacı (mismatch count) tutar. İç durumların her biri bağımsız olarak şifreleyiciden geçirilerek her saat vuruşu için Tahmin Kapasitesi hesaplanır.

Genel olarak saldırının çalışma prensibi test edilen iç durumun ileri ya da geriye doğru (hangisi uygunsa) bir yandan tüm geri besleme değerlerini, ya *Determine(Belirleme)* ya da *Check & Guess (Kontrol et ve sonra tahmin et)* yöntemi ile oluşturmaya, diğer yandan bu geri besleme değerlerinden kaçının beklenen değerden farklı olduğunu saymaya, yani her bir iç durum için uyuşmazlık sayacı tutmaya dayalıdır. Ortalama tahmin kapasitesinin oldukça yüksek olduğu ABGBF-KAÜ'lerde doğru iç durumun uyuşmazlık sayacının diğer bütün rastgele ve yanlış iç durumlarınkinden çok daha düşük olması beklenir. Bu özellik saldırı algoritmasında ayıraç olarak kullanılır ve belli bir eşik değerini geçen sayaçlara sahip bütün iç durumlar "yanlış iç durum" olarak değerlendirilerek elenir. Saldırıda tek bir iç durum (doğru iç durum) kalıncaya kadar iç durumlarda ne kadar geri besleme değeri hesaplanacağı ve eşik değerinin ne olacağı teorik olarak hesaplanmıştır. Saldırının ayrıntıları için [4] No'lu kaynağa bakılabilir.

Kayan anahtar bitleri kullanılarak geri besleme değerinin belirlenip belirlenemediğine bakılır. Eğer belirleme mümkün değilse çıktı bitine göre geri besleme değerini bilmeksizin iç durumun doğruluğunu kontrol etmek mümkündür. Saldırı algoritması bu kontrolün sonucuna göre iki farklı işlemden birini yapar. Belirleme işlemi mümkün ise *Determine* alt yordamı, mümkün değil ise *Check&Guess* alt yordamı tetiklenir.

Determine basitçe, belirleme için gereken matematiksel denklemi çözerek elde edilen geri besleme değeri ile iç durumu şifreleyicide 1 saat vuruşu ilerletir. Önerilen geri besleme değeri varsa belirlenen geri besleme değeri ile aynı olmadığı durumda uyuşmazlık sayacı o iç durum için 1 artırılır. Uyuşmazlık sayacı, ortalama tahmin kapasitesine bağlı olarak hesaplanan eşik değerinden düşükse mevcut iç durum sonraki iterasyona aktarılır, aksi halde o iç durum elenir.

Check&Guess alt yordamına ise geri besleme değerinin belirlenemediği durumlarda girilir. Bu alt yordam iç durumun iki kopyasını alarak bir kopyayı '0', diğer kopyayı da '1' değerini geri besleme kullanarak 1 saat vuruşu işletir. Hemen akabinde önerilen değer varsa, ondan farklı bitle işletileni 1 puan cezalandırır. İki iç durum kopyasının da uyuşmazlık sayacı kontrol edilerek eşik değeri aşan iç durum varsa elenir. Elenmediği durumda kopyalanan iç durumlar sonraki iterasyona devreder. Saldırı algoritması iç durumları en fazla *Sonlandırma Değeri* kadar işletip geriye 1 tane iç durum kaldığında sonlanacak şekilde çalışmaya devam eder.

Gerçek bir iç durumun uyuşmazlık sayacının herhangi bir iç durumun uyuşmazlık sayacından çok daha az olması beklenmektedir. Gerçek durumun beklenen uyuşmazlık sayacı, ortalama tahmin kapasitesi ile belirlenir. Yanlış durumlar için ise bu sayaç değerinin sonlandırma değerinin yarısı kadar olması beklenir. Dolayısıyla uyuşmazlık sayaç değerinin, hesaplanan eşik değerini aştığı iç durumların elenmesi neticesinde gerçek iç durumun yakalanması beklenmektedir. Eşik değeri saldırıdan beklenen başarı oranı, algoritmaya giren iç durum sayısı ve ortalama tahmin kapasitesine göre hesaplanır.

1.3 Katkılarımız

Tez çalışmamızda ABGBF-KAÜ türüne yapılan KE saldırısı [4] üzerinde çalışılmış ve saldırı algoritması iki farklı geliştirme yöntemi ile iyileştirilmiştir. Geliştirilmiş algoritma, mevcut KE saldırısı ile ayrıntılı bir test ve geçerleme sürecinden geçirilerek farklı ABGBF-KAÜ'lerle karşılaştırılmış ve ortalama olarak yaklaşık 15 kat hızlanma gözlemlenmiştir (Bkz.: Şekil 7.7 ve 7.8).

Yeni Saldırı Algoritması

Kara ve Esgin'in algoritmasını baz alarak geliştirdiğimiz yeni algoritmada iyileştirmeye yönelik iki önemli geliştirme yapılmıştır.

İyileştirme No:1 adını verdiğimiz ilk iyileştirme için Ortalama Tahmin Kapasitesi tanımında değişikliğe gidilmiştir. Mevcut Kara ve Esgin tanımına göre Ortalama Tahmin Kapasitesi, bütün iç durumların tahmin kapasitelerinden hesaplanan doğal bir ortalama değerdir. Yeni tanımladığımız ortalama tahmin kapasitesinde ise yalnızca 0.5'ten büyük iç durumlar kullanılarak ortalama hesaplanmaktadır ve saldırıda tahmin kapasitesi 0.5

olan durumlar dikkate alınmamaktadır. Bu sebeple ortalama tahmin kapasitesi her zaman için mevcut ortalama tahmin kapasitesi formülü ile hesaplanan değere eşit veya ondan daha büyük bir değer olmaktadır.

Tanımın geliştirilmesinin yanı sıra algoritma gövdesinde önemli bir değişiklik yapılmıştır. Ortalama Tahmin Kapasitesi formülünün ana mantığına paralel olarak yalnızca tahmin kapasitesi 0.5'ten büyük durumları, yani önerilen değer var olduğu zamanları saymak üzere her bir iç durum için bağımsız olarak, *önerilen geri besleme değeri* sayacı oluşturulmuştur. Önerilen değer sayacının sonlandırma değerini geçtiği iç durum doğru iç durum olarak kabul edilerek algoritma sonunda sergilenecek şekilde saklanmıştır. Bununla birlikte önerilen değer olmadığı durumlarda iç durumlar cezalandırılmamış, algoritma bu iç durumları Determine ve Check&Guess alt yordamları ile işletmeye devam etmiştir. İyileştirme No:1'de sonuç olarak eski ve yeni ortalama tahmin kapasitesi değerinin farklı ve aynı olduğu 19 adet geri besleme fonksiyonu ile yaptığımız testlerde 22 kata varan hızlanma oranları kaydedilmiştir (Bkz.: Şekil 7.1 ve 7.2).

İyileştirme No:3 olarak isimlendirdiğimiz ikinci önemli iyileştirme ise algoritmanın çalışmasını küçük safhalara bölüp aşama aşama çalıştırarak daha hızlı ve efektif durum eleme üzerine kuruludur. KE saldırısında tüm iç durumlar tek bir hamlede test edilmektedir. Gerçek iç durum kalıncaya ve geri kalan bütün iç durumlar eleninceye kadar test algoritması çalıştırılmaktadır. Ortalama tahmin kapasitesinin 0.5 değerine yakın olduğu durumlarda sonlandırma değeri son derece yüksek olmakta ve bu da iç durumların elenmesi için yüksek eşik değerlerine kadar işletilmelerini gerektirmektedir. Yeni eleme yöntemi ile eşik değeri çok daha küçük tutulmuş ve iç durumların hepsi eleninceye kadar testi çalıştırmak yerine küçük eşik değeri ile belirlenmiş bir oranda iç durum elenmiştir. Ardından geri kalan iç durumlar küçük eşik değeri ile tekrar teste tabi tutulmuş ve bu işlem bütün iç durumlar eleninceye kadar tekrar edilmiştir. Bu şekilde parçalı eleme yöntemi ile eleme yapıldığında iç durumların [4]'de verilen saldırıya göre çok daha hızlı elendiği ve algoritmanın önemli ölçüde hızlandığı gözlenmiştir (Bkz.: Şekil 7.3 ve 7.4).

KE saldırısında 1000 saat vuruşu işletilmesi öngörülen bir geri besleme fonksiyonu örneği için, aşamalandırılan algoritmada 100 saat vuruşluk 10 parçaya bölündüğünde iç durumların elenme sınırı olan eşik değeri de buna bağlı olarak yeniden hesaplanarak küçültülür. İlk eleme aşamasında doğrudan tüm elemeler gerçekleşmeden, kalan iç durumlarla ikinci eleme aşaması başlatılır. Bu şekilde tek iç durum kalana kadar eleme işlemi tekrarlanır.

Testlerimizde yanlış iç durumların büyük bir kısmı daha ilk aşamada elendiği için algoritmada [4]'deki saldırıya kıyasla 14 kata varan hızlanmalar gerçekleşmiştir (Bkz.: Tablo 7.4, Şekil 7.3 ve 7.4).

Tasarımın son etabı olarak iyileştirmeler birleştirilerek tek bir algoritma haline getirilmiştir. İyileştirme No:1'in düzensiz saat vuruşlarıyla gerçekleştirdiği iç durum işletimi, İyileştirme No:3'ün düzenli saat vuruşu takip ederek işlettiği eleme yöntemine sorunsuz şekilde entegre edilebilmesi için her bir iç durumun işletilirken hangi saat vuruşunda kaldığı saklanmıştır. İlk eleme aşamasından kalan iç durumlar yeni eleme aşamasına geçtiğinde saat vuruşunun en son kaldığı değer görülene kadar işlem yapılmadan bekletilmiş, ardından sırası gelen iç durum saldırı algoritmasına dahil edilerek rutin eleme işlemleri aynı şekilde tek iç durum kalana kadar devam ettirilmiştir. Testlerimizde bütünleşik algoritmada 66 kata kadar hızlanma gözlemlenmiştir (Bkz.: Şekil 7.7 ve 7.8).

Birbirinden farklı, Sprout'u temsil eden geri besleme fonksiyonunun da içinde bulunduğu 19 adet örnek üzerinde yürüttüğümüz testlerde KE [4] algoritmasına kıyasla ortalama hızlanma; binde bir hata payı dikkate alındığında (%99.9 başarımla) 15,17 kat, yüzde bir hata payı dikkate alındığında ise (%99 başarımla) 14,36 kat olarak kaydedilmiştir. Test sonuçları için Bölüm 7 incelenebilir. Algoritma yapıları ve performans testleri C++ programlama diliyle kodladığımız benzetim yazılımı kullanılarak gerçekleştirilmiştir. Şeffaflık açısından benzetim yazılımının kaynak kodları Ek-B'de açıklamalarıyla birlikte verilmiştir.

1.4 Tezin Bölümleri (Ana Hatları)

Tez çalışmamızı oluşturan 8 adet bölüm aşağıdaki gibidir. Literatüre katkılarımızı içeren ve yeni bilgilerin olduğu bölümler '*' simgesi ile işaretlenmiştir. Özellikle tezdeki katkılarımızın ana kısmını oluşturan bölümler saldırı algoritmasındaki geliştirmelerin anlatıldığı Bölüm 6 ve deneylerle gelişmelerin doğrulandığı Bölüm 7'dir.

- **Bölüm-1 Giriş:** Tez içeriği ve yapısı hakkında özet bilgi veren, literatüre olan katkılarımızın anlatıldığı, asgari düzeyde teknik bilgi gerektiren giriş bölümüdür.
- **Bölüm-2 Temel Kavramlar:** Kriptolojinin günümüze kadarki gelişiminin özetlendiği, temel kavramlardan bahsedilen bölümdür.

- **Bölüm-3 Dizi Şifreleme:** Bu bölümde kriptografik algoritmaların tez çalışmamızla ilgili ana kolu olan *Dizi Şifreleme Algoritmaları*'nın tanımı yapılmış ve bu tür algoritmaların iyi anlaşılması için gerekli temel kavramlar anlatılmıştır. Devamında ise A5/1 [30], Trivium [5] ve Espresso [31] gibi türün somut örnekleri incelenmiş ve bölümde anlatılan bilgilerin pekişmesi sağlanmıştır.

- **Bölüm-4 Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteçleri:** Kriptografik algoritmaların sınıflandırıldığı ağaçta bir seviye daha aşağıya inilerek tez çalışmamızın konusuyla en yakın bağlantılı türün tarifinin yapıldığı bölümdür. Ayrıca, ilerleyen bölümlerde kullanılacak matematiksel tanımlar verilmiş ve türün ilk somut örneği olan Sprout [1] algoritması ayrıntılı şekilde incelenmiştir.

- **Bölüm-5 ABGBF-KAÜ Ailesine Yönelik Genel Kapsamlı Saldırı:** Tez konumuz olan kriptografik algoritma türüne, yani *Anahtarlı Boolean Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteçleri*'e yönelik ilk genel kapsamlı saldırı olan KE [4] saldırısının ayrıntılı şekilde incelendiği bölümdür.

- ***Bölüm-6 Geliştirilmiş Saldırı Algoritması:** Tasarladığımız yeni algoritma kapsamında İyileştirme No:1 ve İyileştirme No:3'ün anlatıldığı ve sözde kodlar (pseudo-code) ile açıklandığı bölümdür.

- ***Bölüm-7 Geliştirilmiş Algoritmanın Performans Analizi:** Tasarladığımız yeni algoritma ile orijinal algoritma olan KE [4] saldırısının performans ölçümlerinin sergilendiği bölümdür. Bu bölümde öncelikle test sistemi ve test senaryosu belirtilmiş ve 19 adet örnek geri besleme fonksiyonu tanımlanmıştır. Daha sonra bu 19 farklı örnek için testler gerçekleştirilmiş; KE algoritması ve yeni algoritmanın kıyaslandığı tablolar ve çubuk grafikler verilmiştir.

- ***Bölüm-8 Sonuç:** Yeni algoritma tasarımında yapılanlar özetlenmiş, çıkarılan sonuç ve istatistikler verilmiştir. Daha sonra gelişime açık noktalar belirtilmiş ve çalışmaya devam edilmesi durumunda ele alınması gereken konular vurgulanmıştır.

- **Ek-A** Belirli bir test durumu için KE algoritmasının bellek kullanım raporu verilmiştir.

- ***Ek-B Benzetim Yazılımı Kaynak Kodları:** Bölüm 7'daki testlerin gerçekleştirilmesinde kullanılan, tez çalışması kapsamında C++ programlama dilinde

yazılmış benzetim uygulamasına ait kaynak kodlar açıklamalarıyla birlikte verilmiştir.

Bölüm 2

Temel Kavramlar

2.1 Kriptografinin Kısa Geçmişi

2.1.1 İletişim Yöntemlerinin Gelişimi

İnsanoğlu uzun mesafe iletişim söz konusu olduğunda tarih boyunca pek çok yöntem geliştirmiştir. Ateş, duman, davul gibi kalıcılık niteliği bulunmayan ilkel haberleşme yöntemleri, yerini taş ve duvar yazıtları gibi zamana meydan okuyan eserlere bırakmış; Eski Mısır'da M.Ö. 3000 yılında Papirus'ın icat edilmesiyle başlayan süreçle birlikte iletişim daha kolay, ucuz, taşınabilir ve çoğaltılabilir niteliğe bürünmüştür.

Samuel Morse'un 1844 yılında ilk telgraf mesajını göndermesi, Alexander G. Bell'in 1875'te kendi icadı olan *Telefon* ile yaptığı ilk konuşma ile daha ileri boyuta taşınmış; tüm dünyaya yayılan telefon ağları üzerine kurulu devrimsel bir teknoloji olan İnternet'in icadı ile taçlandırılarak günümüze gelmiştir. Günümüzde, hücresel telefon ağları (GSM, 4.5G), radyo frekanslı kimlik belirleme (RFID), kablosuz bilgisayar ağları, uydu haberleşmesi gibi teknolojilerin yaygınlaşarak internet bankacılığı, e-ticaret, kamu ve askeri içerikli verilerin transferi gibi mahremiyet derecesinin yüksek olduğu alanlarda yer bulmuş ve kriptografiye duyulan ihtiyacı tarihte hiç olmadığı kadar artırmıştır.

2.1.2 Kriptografi Nedir?

Kriptografi, sayısal haberleşmede ve depolama ortamlarında veri güvenliğini sağlamak ve mahremiyeti korumak amacıyla geliştirilmiş matematiksel öğreti ve teknikler bütünüdür. Kriptografide gönderilen veya depolanan verinin, gönderilmesi veya depolanması sırasında tamamen tanınmaz ve anlamsız biçimde olması, ihtiyaç halinde yetkili taraf veya taraflar tarafından tekrar okunabilir özgün haline kayıpsız olarak geri döndürülebilmesi amaçlanır.

Bu temel operasyon, açık hâlde olan veriyi oluşturan karakterlere ait sayısal karşılıkların (ASCII veya UTF-8) matematiksel formüllerden geçirilmesiyle gerçekleşir [32][33]. Açık veri şifrelenirken *anahtar* adı verilen sayısal ifade bu matematiksel fonksiyonlarda kullanılarak verinin mahremiyeti formülün mahremiyetinden koparılır. Böylece formüller bilinse dahi bu gizli anahtar bilgisi olmadığı sürece şifre çözülemez.

2.2 Kriptografik Algoritmaların Sınıflandırılması

2.2.1 Antik Dönem Teknikleri

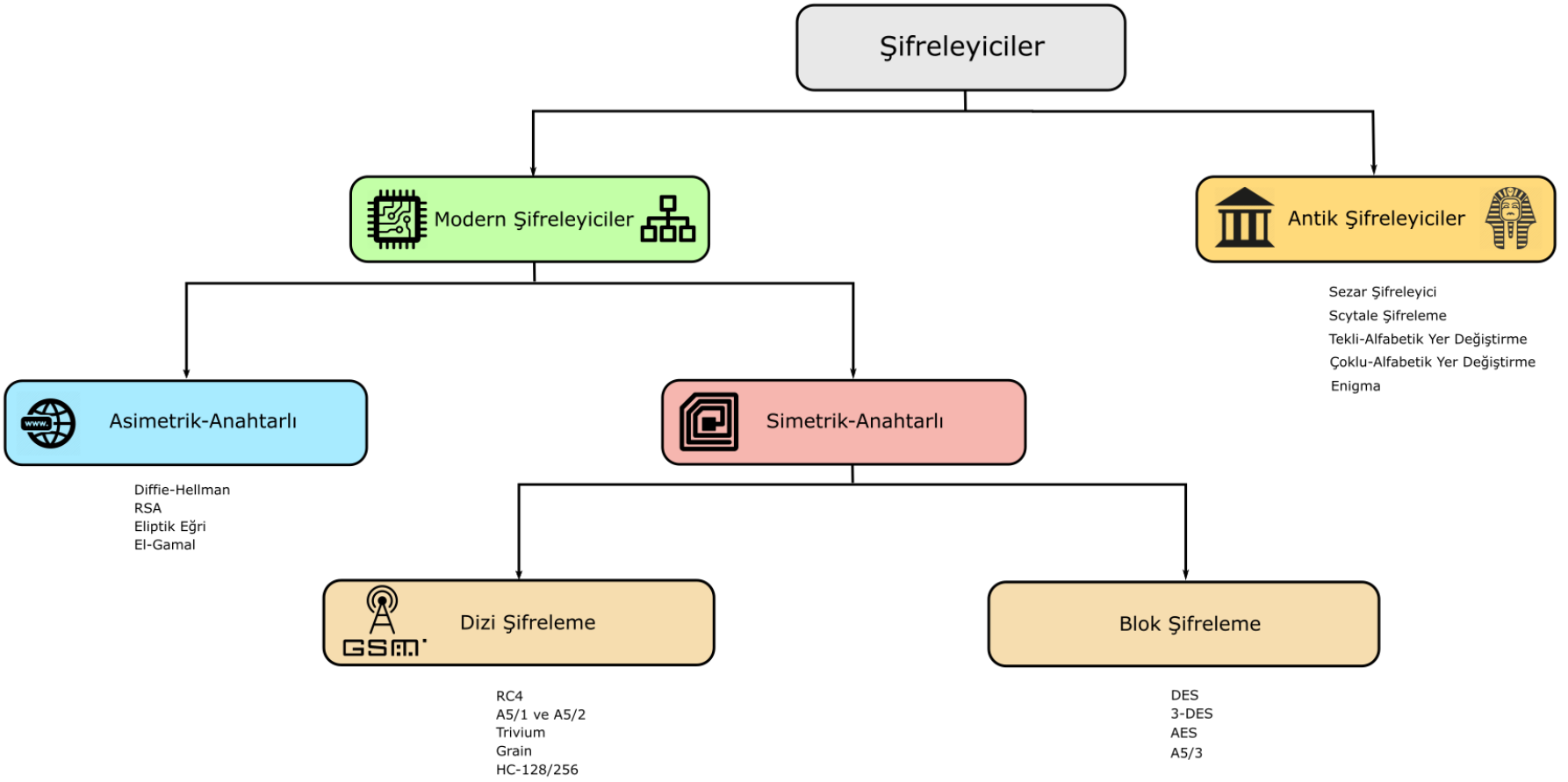
İletişimde mahremiyet ihtiyacı ortaya çıktığında tekli-alfabetik ve çoklu-alfabetik harf yerleştirme ve kaydırma teknikleri yaygın olarak kullanılmaktaydı. M.Ö. 500-600 yıllarında Ortadoğu'da kullanılan Atbash, aynı dönemde Antik Yunan'da Sparta ordusunun kullandığı Scytale silindiri, Jül Sezar döneminde kullanılan Sezar harf kaydırması gibi çok eski örneklerin yanı sıra II. Dünya Savaşı'nda Alman ordusunun kullandığı *Enigma* cihazı da dahil olmak üzere bilgisayar tabanlı olmayan tüm kriptografik yöntemleri *Antik Şifreleyiciler* başlığı altında topladık.

Antik Şifreleme yöntemlerinin ortak özelliği şifrelenmiş metnin yine alfabeyle sınırla bir uzayda karşılığının olmasıdır. Antik Şifreleyicilerde atomik nüfuz alanının alfabeyle sınırlı olması şifreleyici ne kadar güvenli olursa olsun kaba kuvvet(brute-force) saldırılarına karşı savunmasız bırakmıştır. Zira tekli-alfabetik şifreleyicilerin kriptanalizi alfabedeki harf sayısı kadar deneme yapmaktan ibarettir.

2.2.2 Elektronik Dünyaya Geçiş

Bilgisayar ve sayısal iletişim yöntemlerinin gelişmesinden sonra bu dezavantaj ortadan kalkmıştır. Zira artık alfabedeki harflerde değil 8 Bit ASCII ve 16 bit UTF-8 standardına sahip karakterlerle çalışılmaya başlanmıştır [32][33]. Örneğin UTF-8 kodlamasında 16'lık tabandaki $(0041)_{16}$ sayısı alfabedeki 'A' harfini, $(003F)_{16}$ sayısı ise '?' sembolünü temsil etmektedir. Bu da temsil uzayını alfabedeki harf sayısıyla sınırlı değil, ASCII için 2^8, UTF-8 için 2^{16} farklı sayıyla belirlendiğini gösterir. Olası kombinasyonların her bir karakter için 25'ten $2^8 = 256$ ve $2^{16} = 65536$'ya yükselmesi şifreleme algoritmalarına olan saldırıları harf değiştirme tabanlı şifreleme yöntemlerine kıyasla zorlaştırmıştır.

Kriptografik Algoritmalar, başka bir deyişle Modern Şifreleyiciler; Simetrik-Anahtarlı ve Asimetrik-Anahtarlı olmak üzere iki grup altında incelenmiştir. Simetrik-Anahtarlı şifreleyiciler de kendi içerisinde ikiye ayrılmaktadır: Blok Şifreleyiciler ve Dizi Şifreleyiciler. Dizi Şifreleyiciler açık metni, metin kadar uzunlukta tek bir anahtarla şifreleyen algoritmalardır. Blok Şifreleyiciler ise açık metni blok boyutuna uygun parçalara bölerek parçalar üzerinde çalışır. Dizi Şifreleyiciler düşük kaynaklı donanım tasarımlarına daha yatkın yapıları nedeniyle tercih sebebidir. 2001'de standart olarak kabul edilen ve günümüzde güvenilirliğini koruyan AES, akıllı kart uygulamalarında, kablosuz ağ güvenliğinde, sabit disklerin şifrelenmesinde ve IPTV içeriklerinin şifrelenmesine kadar birçok alanda aktif olarak kullanılmaktadır [34].



Şekil 2.1: Kriptografik Algoritmaların sınıflandırma diyagramı

Bölüm 3

Dizi Şifreleme

Bu bölümde ana konumuzla bağlantılı olan Dizi Şifreleme algoritmalarının anlaşılması için gereken temel altyapı bilgisine, bu tür algoritmaların günlük hayattaki kullanım alanlarına ve A5/1[30], Trivium[5] ve Espresso[31] algoritmalarına ait özet anlatımlara yer verilmiştir.

3.1 Giriş & Kullanım Alanları

Dizi şifreleme algoritmaları, her bir saat döngüsünde şifreleme parametrelerini değiştirmek için iç durum saklayıcıları (internal state registers) adı verilen bir geçici bellek alanı kullanarak Tek Seferlik Şifre(One-Time Pad) kavramını elektronik ortamda gerçekleyen, başka bir deyişle kayan anahtar oluşturan algoritmalardır. İleride daha detaylı bahsedeceğimiz Tek Seferlik Şifre, kusursuz şekilde rasgelelik ihtiva eden, açık metinle aynı veya ondan daha uzun olan bir şifreleme anahtarıdır.

Dizi şifreleme algoritmalarının ortak noktası kayan anahtarı üretip açık metinle mantıksal XOR işlemine tabi tutmaktır. $\mathcal{K}$ dizi şifreleme algoritmasının oluşturacağı kayan anahtarı, k_1 to k_n'e kadar olan değerler de bu kayan anahtara ait bitler, P ve C ise sırasıyla açık metni ve şifrelenmiş metni temsil etsin. Dizi şifreleme algoritması $\mathcal{K}$ kayan anahtarını oluşturduktan sonra bunu P açık metni ile XOR'lar ve C şifreli metnini elde eder.

$$c_1 = p_1 \oplus k_1$$

$$c_2 = p_2 \oplus k_2$$

$$c_3 = p_3 \oplus k_3$$

$$...$$

$$c_n = p_n \oplus k_n$$

$$\mathcal{C} = \mathcal{P} \oplus \mathcal{K}$$

3.1.1 GSM (2G), UMTS(3G) ve LTE(4G) Güvenliği

A5/1 ve A5/2 algoritmaları baz istasyonu ve cep telefonları arasında havadan iletilen sinyallerin şifrelenmesi için kullanılan 3 LFSR'dan oluşan ve düzensiz işletilen iç duruma sahip en bilinen dizi şifreleme algoritmalarındandır. 1994'ten bu yana çeşitli guess-and-determine(tahmin et ve belirle), cebirsel ve istatistiksel saldırı yöntemlerinin hedefi olmuştur [30][35][36].

A5/1 ve A5/2 dizi şifreleme algoritması olmasına karşın, Mutsubishi'nin patentli ürünü MYSTY1 algoritmasından türetilmiş olan A5/3, diğer adıyla KASUMI [37], 128-bit anahtar boyutuna sahip, 2G ve 3G hücresel veri iletişiminin korunmasında kullanılan bir blok şifreleme algoritmasıdır. Günümüzde 4. Nesil (4G, LTE) veri ağında, 32-bit word uzunluğu, 128-bit anahtar ve 128-bit ilklendirme değeri barındıran SNOW-3G [38] algoritması kullanılmaktadır. 5. nesil (5G) iletişimi ise 4G'ye benzer şekilde SNOW-3G [38], AES-CTR [39] ve ZUC [37] algoritmaları kullanılmaktadır.

3.1.2 Kablosuz Ağ Güvenliği (WEP and WPA)

Ron Rivest tarafından 1987 yılında tasarlanan RC4 (Rivest Cipher 4) algoritması özellikle İnternet trafiğinin korunmasında SSL/TLS, kablosuz ağ iletişiminin korunması noktasında ise WEP(1999) ve WPA(2003) gibi yaygınlaşmış algoritmalar tarafından kullanılmış; fakat daha sonra istismar edilebilir yapısal sorunları sebebiyle terk edilmek zorunda kalınmıştır [40][41].

3.1.3 RFID Uygulamaları

Radyo Frekanslı Kimlik Belirleme anlamına gelen RFID, ulaşım ve kimlik kartlarından, perakendecilik sektöründe kullanılan ürün etiketlerine, ücretli yol ve köprü geçişlerine kadar birçok alanda yaygın olarak kullanılan bir teknolojidir. Aktif, Yarı-Pasif ve Pasif olmak üzere üç farklı RFID etiket türü vardır.

Aktif RFID etiketleri devre, radyo sinyali üreteci ve bir pile sahiptir. En pahalı donanıma sahip olan Aktif RFID etiketleri okuyucu cihaz ile 30 metreye kadar uzaktan iletişim kurabilmektedir. Yarı-Pasif cihazlar kullanılmadığı süre boyunca uyku modunda kalarak pilden tasarruf ederek, yalnızca iletişim kuracakları zaman okuyucu cihaz tarafından uyandırılarak daha fazla güç tüketecek olan işlemleri yapmaya ve veri aktarımına başlarlar.

Üçüncü ve son tür olarak Pasif RFID etiketleri ise herhangi bir güç kaynağı taşımaz. Üzerinde sarmal şekilde tasarlanmış olan anten, okuyucu cihazın oluşturduğu manyetik alana girdiğinde RFID anteni transformatör vazifesi görerek devreye yeterli gücü sağlamaya başlar. Pasif RFID etiketleri yalnızca birkaç santimetre etki alanına sahiptir. Okuyucuya çok yakın tutulması gerekir. RFID'de hem dizi şifreleme hem de blok şifreleme örnekleri görülmesine karşın dizi şifreleme en zayıf donanıma sahip olan Pasif RFID uygulamalarında kullanılır.

WG serisi, Grain [14], Trivium [5], Mickey [13] bu tür dizi şifreleme algoritmalarına örnek olarak verilebilir. Bu alanda kullanılan güncel algoritmalar arasında David vd.'nin tasarladığı A2U2 [42] ve Luo vd.'nin tasarladığı WG-7 [43] bulunmaktadır.

3.1.4 Kablosuz Sensör Ağları (WSN)

Kablosuz sensörler bir ya da daha fazla sensöre ev sahipliği yapan, pil, sayısal devre ve iletişim biriminden oluşan, görevi kaydettiği veriyi eşdeğeri olan cihazlara ve veri merkezine aktarmak olan küçük boyutlu otomatik ölçüm cihazlardır. Ölçülebilen büyüklüklerden bazıları sıcaklık, nem oranı, duman, titreşim, yanıcı gaz, su seviyesi, asitlik ve bazlık dengesidir.

18. yüzyılın sonuna doğru sanayi devrimi olduktan sonra fabrikaların etkin yönetimi ve insan gücünün düşürülmesi amacıyla insansız ölçüm yapma teknolojisinin ihtiyacı

hissedilmiştir. 15 Ocak 1919'da Amerika Birleşik Devletleri'nin Boston eyaletinde devasa boyuttaki bir şeker pekmezi tankı infilak ederek 21 kişinin ölümü ve yüzlerce kişinin yaralanmasına sebep oldu [44]. Şayet tanktaki karbondioksit seviyesi, iç basınç ve sıcaklık yeterince sık kontrol edilmiş olsaydı önlemi hızlı bir şekilde alınabilir ve felaketin önüne geçilebilirdi.

Kablosuz Sensör Ağları (Wireless Sensor Networks, WSN) küçük boyutlu, zayıf donanımı ve düşük güç tüketimi olan ölçüm cihazlarından oluşan bir ağ türüdür. WSN düğümlerinin nihai amacı çok uzun süre bakım gerektirmeden çalışarak ölçüm görevini gerçekleştirmektir. Askeri uygulamaların yanı sıra ağır sanayi endüstrisinde, hatta hayvan belgeselleri ve araştırmalarında bilime katkı sağlama amacıyla kullanılmaktadır.

Kablosuz sensörler bir kaplanın yavrularına takılarak aylarca bu yavruların gelişiminin veri merkezinden takip edilmesi, orman ve kırsal alanlarda hakimiyet bölgelerinin ayırt edilmesi ve görsellenmesi sağlanabilir [45, 46]. WSN teknolojisi kullanılarak göçmen kuşların göç güzergah haritasının çıkarılması, ilgili çeşitli akademik çalışmalar yayınlanmıştır [47][48][49]. WSN, bu gibi mobil uygulamaların yanı sıra bir ağaca sabitlenmiş veya yapay bir taşın içerisine monte edilmiş fotoğraf kapanı gibi hareket etme amacı gütmeyen uygulamalarda da kullanılmaktadır [50].

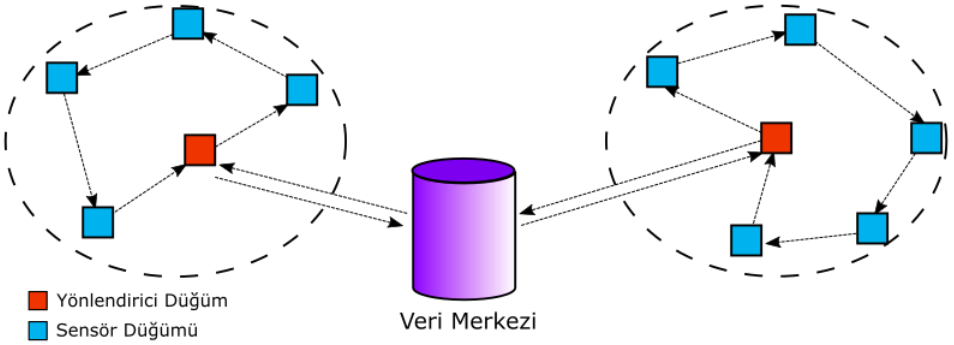
3.1.5 ZigBee Protokolü

Bu bölümde WSN ürünlerinde kullanılan protokollerden bahsedilecektir. Silicon Labs Ember ® EM351/EM357 kablosuz sensör serisi AES-CCM (Cipher Block Chaining Message Authentication Code) şifrelemesini barındıran ZigBee protokolünü kullanmaktadır. IETF RFC3610 Bluetooth standardı ile uyumluluk arz eden bu protokol AES-CTR (AES Sayaç Modu) ve CBC-MAC protokollerinin birleşimi olan AES-CCM şifreleme algoritmasını kullanır. ZigBee için ayrıntılı bilgi, MWRI firmasının yayınladığı donanım belgesinde mevcuttur [51].

Bu algoritma, şifrelemenin temel mantığına uygun olarak, gönderilecek veri büyüklüğünde kayan anahtar üreterek veri ile XOR'lar. Diğer dizi şifreleme yöntemlerinden farklı olarak tüm veri paketlerine 4 bayt'lık bir MIC(Message Integrity Check, Mesaj Bütünlük Kontrolü) alanı ilave eder. Tablo 3.1'den görüldüğü üzere WSN donanımları oldukça zayıf özelliklere sahip donanımlardır.

Tablo 3.1: Silicon Labs Ember ® EM351/EM357 [6] ve NXP Semiconductors JN5148-001 [7] donanım özellikleri karşılaştırması

Özellik	**EM351/EM357**	**NXP JN5148-001**
CPU	ARM® Cortex® 64 Mhz 32-Bit Processor with FPU	Ayarlanabilir 4-32 Mhz 32-bit RISC CPU
Bellek	32kb / 64Kb	128Kb
Depolama	512Kb(Flash)	128Kb(ROM)
Bağlantı	Bluetooth® Low Energy Mode (Slightly Modified Compliant)	2.4 Ghz IEEE 802.15.4
Bant Genişliği	1 Mbps / 2 Mbps	500 - 667 kbps



Şekil 3.1: Kablosuz Sensör Ağları için basit bir topoloji görseli.

WSN mesajlarının korunması kullanım alanlarına göre çok kritik olabilmektedir. Bu sebeple zayıf donanım özelliklerine sahip bu cihazların hafifsiklet şifreleme yöntemlerini kullanması kaçınılmaz hâle gelmektedir. Kriptografi dünyasında bu ve bunun gibi tasarım kısıtlarını yerine getirebilmek için EU ECRYPT tarafından 3 aşamadan oluşan eSTREAM yarışması düzenlenmiştir. Üçüncü faz finalistleri DRAGON, Py ve Pypy, Salsa20, SOSEMANUK, Phelix, HC-128 ve HC-256, LEX olmuştur [52]. Kazananlar, Profil I (Yazılım) ve Profil II (Donanım) olmak üzere iki kategoride toplanmıştır. Seçilen algoritmaları Tablo 3.2'de görülmektedir.

Tablo 3.2: eSTREAM yarışmasını kazanan algoritmaların listesi

Profil I(Yazılım)	**Profil II(Donanım)**
HC-128	Grain
Rabbit	MICKEY
Salsa20/12	Trivium
SOSEMANUK	

3.2 Dizi Şifrelemenin Temel Kavramları

Bu bölümde Dizi Şifreleme algoritmalarının türleri, performans ölçütleri, LFSR, NFSR ve Tek Kullanımlık Şifre(One Time Pad) gibi önemli temel kavramları anlatılmıştır.

Dizi şifreleme, saklayıcılarının işletim yöntemine göre iki grupta incelenmiştir. Eşzamanlı Dizi Şifreleme algoritmaları önceki şifreli metin veya açık metinden bağımsız olarak her bir biti ya da word'ü birbirinden bağımsız olarak şifreler. Bu sebeple yalnızca rasgele sayı üreteçleri ve ilklendirme vektörleri şifreleme aşamasında kullanılır.

Diğer taraftan, kendi kendine senkron olan veya diğer ismiyle asenkoron dizi şifreleme algoritmaları ise her bir şifreleme operasyonunda önceki şifreli metni girdi olarak alır. Yaptığı bu işlem iletişim sırasında bazı bitler kaybolsa dahi alıcı tarafta yeterli miktarda veri geldiğinde şifre çözme işlemi kendini toparlayarak başarılı şekilde çalışmaya devam etmektedir.

Bunun zıttı olarak senkron dizi şifreleme algoritmalarında şifreli metne bir bit ilave edildiğinde veya eksiltildiğinde alıcı tarafta şifre çözme işlemi sonrası tamamen bozuk sonuçlar elde edilirken, bit sayısı değişmezse yalnızca bozuk bitler etkilenir. Asenkron dizi şifreleme algoritmalarına AES-CFB (Cipher Feedback, Şifre Geri Besleme) modu örnektir.

3.3 Tek Seferlik Şifre (One Time Pad)

Kusursuz şifrelemeyi elde edebilmek için gerçek rasgelelik ihtiva eden, açık metinle aynı uzunlukta veya ondan daha uzun anahtar kullanılmalıdır. Bu tarifte bir anahtar oluşturulup açık metine eklendiğinde (XOR) kusursuz şifreleme gerçekleştirilmiş olur. Bu

işleme İkili Toplamalı Şifreleme, kullanılan anahtara ise One-Time Pad (Tek Seferlik Şifre) adı verilir. Literatürde *Tek Kullanımlık Şerit* şeklinde de geçmektedir.

Tek seferlik denilmesinin nedeni anahtarın birden fazla kez kullanıldığında kusursuz olma özelliğini yitirmesidir. Tek Seferlik Şifre'nin temel dayanağı Amerika menşeli iletişim firması olan AT&T Bell Labs mühendisi Gilbert Vernam tarafından 1917 yılında dünyanın ilk çoklu-alfabetik toplamalı dizi şifreleme yöntemini icat etmesiyle şekil bulmuştur. İcadı delikli kartları kayan anahtar olarak kullanıp telgraf hattı üzerinden gönderilecek verinin şifreleme işlemini gerçekleştiren bir teleyazıcı cihazıdır.

3.4 Donanımsal Nitelikler ve Performans Ölçütleri

Bu bölümde dizi şifreleme algoritmalarında şifreleme hızını etkileyen başlıca faktörler ve belirli bir algoritmanın performansı ilgili bilgi ihtiva eden göstergeler incelenecektir. Bu faktörler *çıktı hızı*, yayılım gecikmesi, operasyonel saat frekansıdır. Bu faktörlere ilave olarak algoritmanın donanım üzerinde gereksinim duyduğu mantıksal kapı sayısını gösteren, GE (Gate Equivalent, Kapı Eşdeğeri) birimi anlatılmıştır.

3.4.1 Donanım Boyutu (Kapı Eşdeğeri)

Sayısal elektronik devreler AND, NOT, OR, XOR, XNOR, NAND ve NOR gibi muhtelif mantıksal kapıların uç uca eklenmesiyle meydana gelmektedir. Şifreleme algoritması tasarımında göz önünde bulundurulması gereken en temel faktörlerden biri devre tasarımı için gerekli olan mantıksal kapı sayısıdır. Bu birim devre üretim teknolojisinden bağımsız bir ölçüttür. Dolayısıyla, 90nm üretim teknolojisi ile üretilmiş bir 750 GE'lik bir şifreleme algoritması, 45nm teknolojiyle üretilmiş 1500 GE'lik bir algoritma ile aynı fiziksel boyuta sahip olacaktır. Tablo 3.3'de bazı bilinen şifreleme algoritmalarının mantıksal kapı sayıları görülmektedir [53].

3.4.2 Çıktı Hızı

Çıktı hızı bir saniyede şifrelenen bit sayısını ifade eden büyüklüktür. Algoritmayı anlatan makale veya yayınlarda genellikle bu büyüklük verilir. Çıktı hızı bit/çevrim ile operasyonel saat frekansının çarpımıyla elde edilir. Örneğin SNOW-3G [38] algoritması

Tablo 3.3: Hafifsiklet dizi şifreleme algoritmalarının karşılaştırılması

Algoritma	**IV Boyutu**	**Anahtar Boyutu**	**Donanım Boyutu**
Trivium	80 Bit	80 Bit	1600 GE
Grain	96 Bit	128 Bit	1300 GE
Espresso	96 Bit	128 Bit	1497 GE
Fruit v2	70 Bit	80 Bit	990 GE
Plantlet (UNI/O)	90 Bit	80 Bit	898 GE
A2U2	-	56 Bit	300 GE

her çevrimde 32-bit işlemekte olup operasyonel frekansı 104 MHz'dir. Bundan dolayı SNOW-3G'nin çıktı hızı 104 MHz x 32-Bit = 3328 Mbps şeklinde hesaplanabilir.

3.4.3 Yayılım Gecikmesi

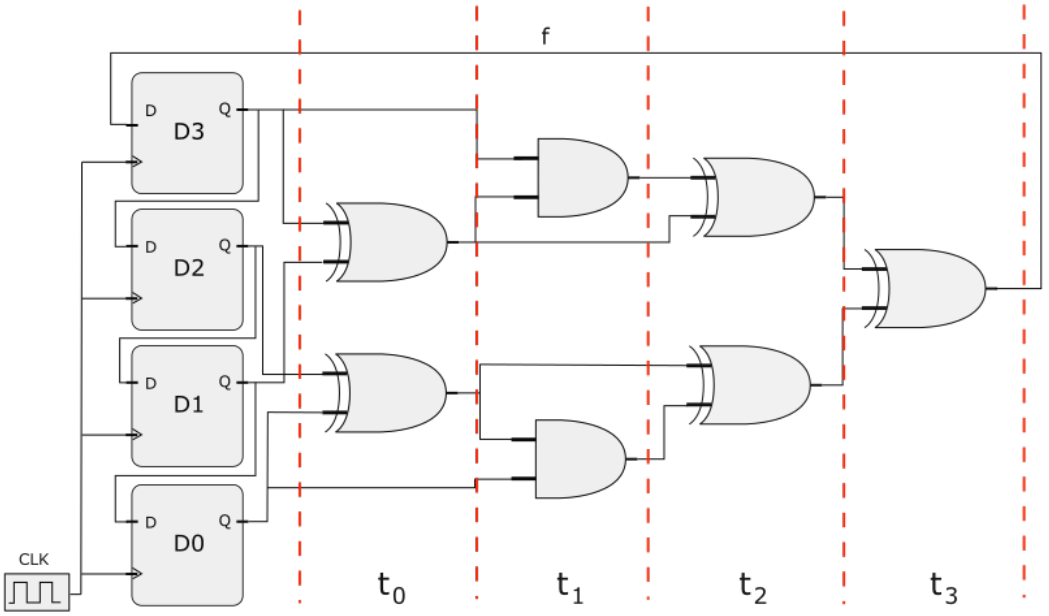
Yayılım gecikmesi elektriksel sinyallerin kaynaktan hedefe ilerlediği süre zarfında geçen zamandır. Devre tasarımında hız limitini belirleyen faktörlerin en önemlisidir. Örnek vermek gerekirse, bir algoritmaya ait iyileştirilmiş ve parallelleştirilmiş bir güncelleme fonksiyonu, kötü tasarlanmış ve çok fazla seri bağlantı içeren bir güncelleme fonksiyonuna sahip aynı algoritmayı çıktı hızı olarak kolaylıkla geride bırakabilir.

Bir sayısal devre yayılım hızı göz önünde bulundurmadan çok yüksek operasyonel saat frekansıyla işletildiğinde bozuk çıktılar oluşturabileceğinden, tam güvenilirlikle çalışması beklenen şifreleme algoritmaları için devre optimizasyon süreci ayrı bir çalışma ve mühendislik gerektirir.

Şekil 3.2'de görüldüğü gibi, 4 bitlik bir LFSR devresinin tasarımında elektrik akımı geçerken mantıksal kapılarda t_0, t_1, t_2 ve t_3 yayılma gecikmeleri sebebiyle geri besleme fonksiyonunun sonucu ancak $t_0 + t_1 + t_2 + t_3 + t_\epsilon$ kadar süre sonra D3 flip-flop'una ulaşabilmektedir.

3.4.4 Operasyonel Saat Frekansı

Elektronik devrede, kare dalga üretecinin 1 saniyede gerçekleştirdiği saat vuruşu sayısıdır. Hz(Hertz) cinsinden ifade edilir. Şifreleme işini yapan elektroniğin ne kadar hızlı çalışacağını tayin eden bu parametre; donanım hızı, güç gereksinimi ve ısı üretimi gibi



Şekil 3.2: Bir LFSR geri besleme fonksiyonu üzerinde yayılım gecikmesi örneği.

kriterler göz önünde bulundurularak belirlenir. Tablo 3.4'da bilinen bazı dizi şifreleme algoritmalarının çalışma frekansları verilmiştir.

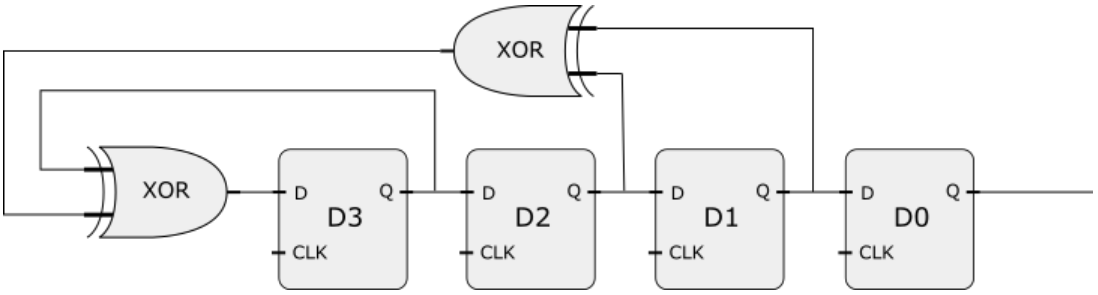
Tablo 3.4: Bilinen bazı algoritmaların operasyonel frekans karşılaştırması [8]

Algoritma	**Bit/çevrim**	**Frekans**	**Çıktı Hızı**
Trivium	1	326 MHz	326 Mbps
Grain	1	177 MHz	177 Mbps
SNOW-3G	32	104 MHz	3328 Mbps
E0	1	187 MHz	187 Mbps

3.5 Lineer Geri Beslemeli Ötelemeli Saklayıcı (LFSR)

Lineer Geri Beslemeli Ötelemeli Saklayıcılar bir dizi D tipi flip-flop'tan meydana gelen ve mevcut durumunu güncellemek için bir lineer fonksiyon (geri besleme fonksiyonu) kullanan saklayıcılardır. Geri besleme fonksiyonu saklayıcının muhtelif bitlerinin XOR'lanmasıyla bir bit oluşturur. Saklayıcı saat vuruşu ile işletildiği anda her bir flip-flop değerini sonrakine aktarır, en baştaki saklayıcıya ise yeni değer olarak geri besleme fonksiyonundan gelen bit yerleşir.

Örneğin, Şekil 3.3'teki 4 bitlik bir LFSR'ı ele alalım. Bu örnekte D3'ün sonraki değeri, diğer bir deyişle geri besleme fonksiyonunun değeri $f = Q_1 \oplus Q_2 \oplus Q_3$'tür. Flip-flop'ların sonraki değerleri $Q_0^{sonraki} = Q_1$, $Q_1^{sonraki} = Q_2$, $Q_2^{sonraki} = Q_3$ ve son olarak $Q_3^{sonraki} = f$ olur.



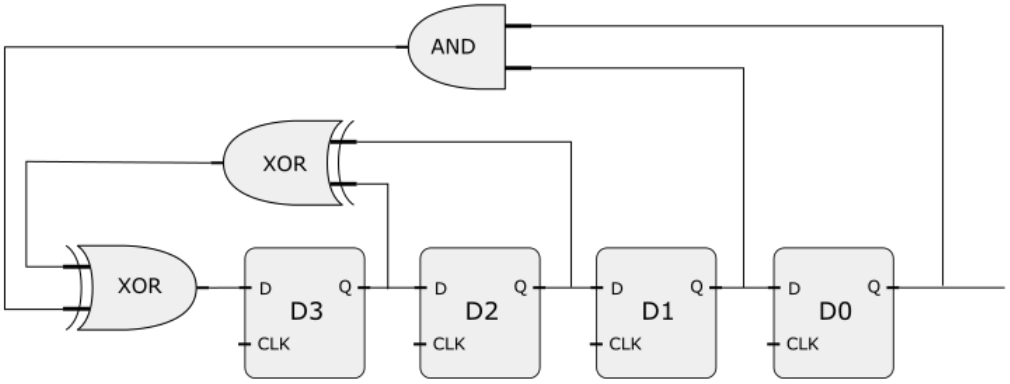
Şekil 3.3: Örnek bir 4 bitlik LFSR devresi

3.6 Lineer Olmayan Geri Beslemeli Ötelemeli Saklayıcı (NLFSR)

Lineer Olmayan Geri Beslemeli Ötelemeli Saklayıcılar RFID ve akıllı kart uygulamaları başta olmak üzere neredeyse tüm güncel dizi şifreleme algoritmalarında kullanılan saklayıcı tipidir. LFSR'lardan farkı geri besleme fonksiyonudur. Geri besleme fonksiyonu en az bir adet lineer olmayan ifade içeriyorsa saklayıcı NLFSR olarak değerlendirilir. Yani, saklayıcıyı meydana getiren bitlerden en az ikisinin çarpımıyla ilgili ifade içermesi gerekmektedir. Bu sayede geri besleme fonksiyonu daha kompleks olur ve LFSR'lara kıyasla daha yüksek bir lineer karmaşıklık ihtiva eder. Şekil 3.4'te, $Q_0.Q_1$ çarpım ifadesi içeren 4 bitlik bir NFSR örneği görülmektedir.

3.7 A5/1 Algoritmasına Hızlı Bakış

A5/1 baz istasyonu anteni ve cep telefonu, SIM kartlı modemler ve SMS otomasyon cihazları gibi mobil cihazlar arasındaki sinyal trafiğini şifrelemekte kullanılan dizi şifreleme algoritmasıdır. 1987 yılında geliştirilen A5/1, Avrupa ülkelerinde kullanılmaya başlandı. GSM protokolü ilk etapta Avrupa'ya uygun tasarlandığından 1989 yılında A5/1'in kasti olarak zayıflatılmış varyasyonu olan A5/2 geliştirilerek Avrupa dışında kullanılmaya başlandı.



Şekil 3.4: Geri besleme fonksiyonu $f = Q_2 \oplus Q_3 \oplus Q_0.Q_1$ olan bir NFSR

Ross Anderson'ın FSE 1994'teki yayınına kadar A5/1 ve A5/2'nin tasarımları gün yüzüne çıkmadı [30]. Sonrasında ise Brienco, Goldberg ve Wagner bu algoritmalarla ilgili tersine mühendislik çalışmaları yayınladı [35]. Ardından Biryukov, Shamir ve Wagner, bir kişisel bilgisayar üzerinde A5/1'in gerçek zamanlı kriptanalizini gerçekleştirdi [36].

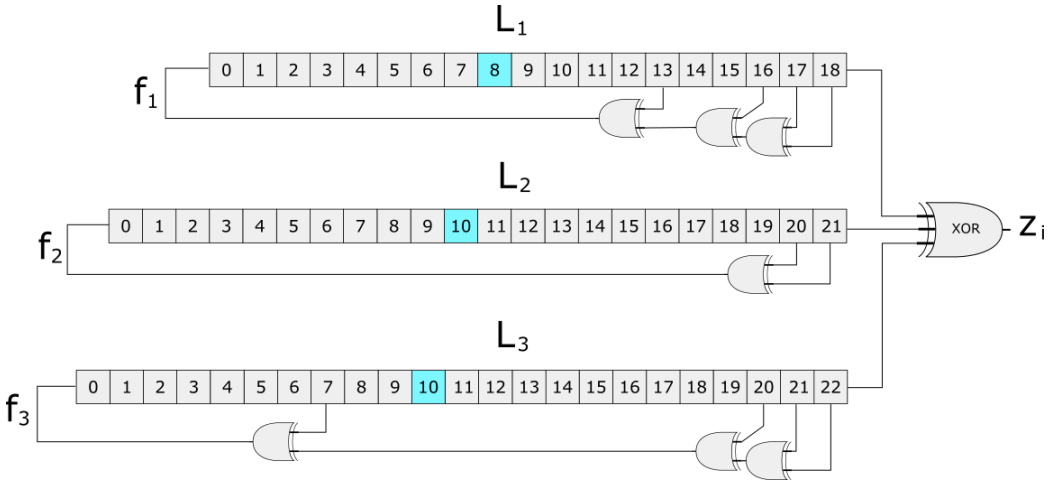
3.7.1 Kayan Anahtar Üretecinin Tasarımı

A5/1'in kayan anahtar üreteci 19-bit, 22-bit ve 23-bit'lik uzunluğa sahip LFSR'lardan oluşmaktadır. İlk LFSR besleme fonksiyonu olarak $f_1 = s_1 \oplus s_{13} \oplus s_{16} \oplus s_{17}$ kullanmaktadır. İkinci ve üçüncü LFSR da sırasıyla $f_2 = s_{20} \oplus s_{21}$ ve $f_3 = s_7 \oplus s_{20} \oplus s_{21} \oplus s_{22}$ şeklindedir. Besleme fonksiyonları bu saklayıcıların en düşük anlamlı bitlerini (LSB) güncellerken, LSB dahil tüm bitleri en yüksek anlamlı bite doğru kaydırır. LSB'nin yerine ise besleme fonksiyonundan çıkan değer uygulanır. Saklayıcı her saat vuruşunda bu yöntemle işletilir.

19-Bit'lik LFSR-1 L_1, 22-Bit'lik LFSR-2 L_2 ve 23-Bit'lik LFSR-3 de L_3 olsun. A5/1'de saklayıcılar devrede bulunan saat kontrol ünitesi tarafından birbirlerinden bağımsız olarak işletilir. Bu kontrol ünitesi L_1'in 9. biti olan s_8'e, L_2'nin 11. biti olan s_{10}'a ve L_3'ün 11. biti olan s_{10} bitlerine bakarak çoğunluk 0 ise kontrol biti 0 olanları, çoğunluk 1 ise bu biti 1 olanları işletir. L_1, L_2 ve L_3'ün saat kontrol bitleri, her bir ihtimal göz önünde bulundurularak Tablo 3.5 oluşturulmuştur.

Tablo 3.5: A5/1'in saat kontrol ünitesine ait çalışma tablosu

$L_1^8, L_2^{10}, L_3^{10}$	**Çoğunluk**	**İşletilecek Saklayıcı**
000	0	L_1, L_2, L_3
001	0	L_1, L_2
010	0	L_1, L_3
011	1	L_2, L_3
100	0	L_2, L_3
101	1	L_1, L_3
110	1	L_1, L_2
111	1	L_1, L_2, L_3



Şekil 3.5: A5/1 algoritmasına ait mantıksal devre görseli. f_1, f_2 ve f_3 geri besleme fonksiyonları, L_1, L_2 ve L_3 de LFSR'lardır. Mavi renkte gösterilen bitler saat kontrol birimine bağlıdır.

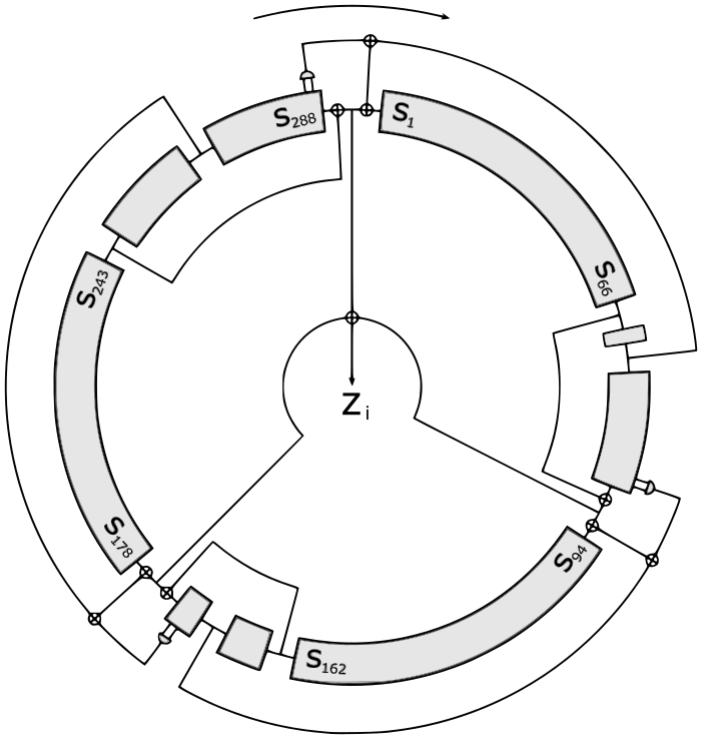
3.7.2 İlklendirme Fazı

Şekil 3.5'te görüldüğü gibi, üç LFSR'ın da MSB'leri XOR'lanarak kayan anahtarı üretmek üzere z_i çıktısını oluşturmaktadır. İlklendirme fazının ilk adımı olarak saklayıcıların tüm bitleri sıfırlanır. Daha sonra gizli anahtara K'ya ait bitler saklayıcılara, çoğunluk kontrolü olmaksızın doldurulur. Bu işlemi takiben, çerçeve numarası F'ye ait bitler üç saklayıcıya yüklenir ve sistem 100 defa olması gerektiği gibi işletilir. Son adım olarak kayan anahtara yansıyan 100 bit kullanılmadan atılır. Bu ilklendirme prosedürü sonunda 228 bit'lik kayan anahtar oluşturulup 114 bit'i giden paketleri şifrelemek için, diğer 114-bit'i de gelen paketleri çözmek için kullanılır.

3.8 Trivium Algoritmasına Hızlı Bakış

Blok şifreleme tasarım prensiplerinden ilham alınarak Christophe De Canniére ve Bart Preneel tarafından geliştirilen hafifsiklet dizi şifreleme Trivium, hız ve donanım alan gereksinimi arasında dengeli bir konum elde etme gayesiyle tasarlanmıştır [5]. eSTREAM yarışmasının Profil II (Donanım) portföyü için seçilen Trivium'un patenti alınmamış; fakat ISO/IEC 29192-3 uluslararası standardı haline getirilmiştir [54].

Toplamda 288-bit'lik NLFSR iç durum saklayıcı kapasitesine sahip olan Trivium, 2^{64} bit çıktıyı 80-bit gizli anahtar ve 80-bit ilklendirme vektörü kullanarak oluşturur. Trivium'un tasarım yaklaşımı giriş ve çıkış bitleri arasındaki lineer korelasyonun azaltılması üzerine kurulmuştur.



Şekil 3.6: Trivium'un mantıksal tasarımına ait görsel [5]

3.9 Espresso Algoritmasına Hızlı Bakış

2015 yılında Elena Dubrova ve Martin Hell tarafından tasarlanmış olan Espresso, ultra-hafifsiklet dizi şifreleme algoritmalarının güncel örneklerinden biridir [31]. Espresso,

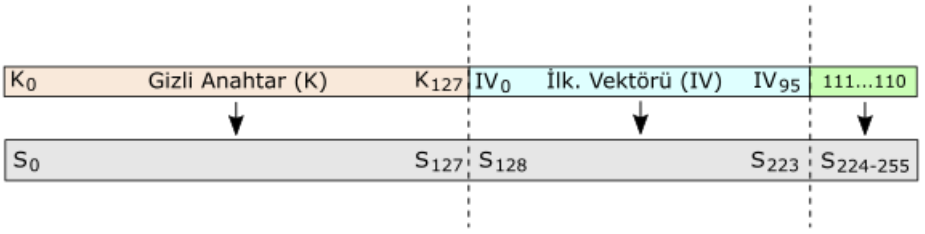
232ns gibi çok düşük iletim gecikmesi, 2.22 Gbit/sn çıktı hızı ve 1497 GE donanım alan gereksinimi ile 5G iletişimini hedeflemektedir. 5G iletişim teknolojisini 4G'den ayıran fark, çok düşük veri iletim gecikmeleri gerektiren uygulamaların hayata geçmesini mümkün kılmasıdır. Bu uygulamalara en iyi örneklerden biri hastaneye fiziksel erişimi bulunmayan kilometrelerce uzaktaki bir operatör doktorun, insan güdümlü robotik cerrahi kullanarak uzaktan ameliyat gerçekleştirmesini sağlayan, Çin'de ilk testleri gerçekleştirilen teknomedikal üründür [55][56].

Espresso 256-bit'lik NLFSR kullanarak iç durumunu günceller. 128-bit gizli anahtar K ve 96-bit IV ilklendirme fazında NLFSR'ı Figür 3.7'de görüldüğü gibi doldurmak için kullanır. K ve IV içeriği olduğu gibi NFSR'a doldurulduktan sonra sistem bu işlem için özel olarak tasarlanmış iki güncelleme fonksiyonu kullanarak 256 kez işletilir. Bu özel fonksiyonlar z_i çıktı biti ve mevcut saklayıcı bitlerini kullanarak x_{255} ve x_{217}'yi günceller. Bu fonksiyonlar aşağıdaki gibi tanımlanmıştır:

$$g_{255} = x_0 \oplus x_{41}x_{70} \oplus z(x)$$

$$g_{217} = x_{218} \oplus x_3x_{32} \oplus z(x)$$

Son adım olarak sistem kayan anahtar oluşturma fazına geçmeden önce, NLFSR üç kez daha işletilir.



Şekil 3.7: Espresso'nun ilklendirme fazının ilk aşaması

Bölüm 4

Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteçleri

Dizi şifreleme algoritmalarının TMDTO(time-memory-data-tradeoff) saldırılarına karşı dirençli olabilmesi için iç durum büyüklüğünün gizli anahtardan en az iki kat büyük olması gerektiği ile ilgili temel kural vardır [57]. Bu kurala göre yeterli güvenlik seviyesinde modeller üretebilmek adına WSN düğümleri ve RFID etiketleri gibi kısıtlı kaynaklara sahip cihazlar için güç tüketimi, güvenlik ve donanım boyutu arasında zor bir denge kurulması gerekmektedir.

Kısıtlı kaynaklara sahip donanımlarda daha etkin bir çözüm önerisi olarak 2015'te Armknecht ve Mikhalev FSE(Fast Software Encryption) etkinliğinde dizi şifreleme algoritmalarına yeni bir tür kazandırdı [1]. Anahtarla Güncelleme Fonksiyonu olan Kayan Anahtar Üreteçleri için önerdikleri ilk somut tasarım, Sprout ismini verdikleri dizi şifreleme algoritmasıdır. Armknecht ve Mikhalev, diğer yaklaşımlardan farklı olarak sabit gizli anahtarın yalnızca ilklendirme aşamasında değil, iç durumun güncellemesinde de kullanılarak donanım alan gereksinimini azaltabileceklerini savunmuştur. Yayından kısa süre sonra güvenlik açıklıkları kullanılarak Lallemand/Virginie [25], Zhang/Gong [26] ve Kara&Esgin [27] çeşitli saldırı metotları geliştirilmiştir.

Özünde iyi bir fikir olmasına karşın Sprout'un asıl kusuru ana fikrinde değil, bu fikrin uygulanmasında yatıyordu. Nitekim, 2017 yılında yayınladıkları makalede [3] Plantlet algoritmasında bu kusur giderilerek Guess-and-Determine(Tahmin et ve Belirle) saldırısına karşı koymayı başarmışlardır.

Bu bölümde *Anahtarla Güncelleme Fonksiyonu olan Kayan Anahtar Üreteçleri (AGF-KAÜ)* ve onun alt kümesi olarak tez çalışmamızdaki geliştirilmiş saldırı algoritmasının asıl hedefi olan *ABGBF-KAÜ(KSG With Boolean KFF)[4]* tanımlanmıştır. ABGBF-KAÜ'un ilk örneği olan Sprout yapısal olarak incelenmiş; ilklendirme fazı, geri besleme ve çıktı fonksiyonları açıklanmıştır.

4.1 Tanımlar

Tanım 1. **Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteci(AGF-KAÜ) [1]):** *Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteci aşağıdaki üç kümeyi içerir:*

- *Anahtar uzayı* $\mathcal{K} = GF(2)^k$,
- *İlklendirme vektörü uzayı* $\mathcal{IV} = GF(2)^v$,
- *Değişken iç durum uzayı* $\mathcal{S} = GF(2)^\sigma$

ve aşağıdaki üç fonksiyondan oluşur:

- *İlklendirme fonksiyonu* $Init : \mathcal{IV} \times \mathcal{K} \longrightarrow \mathcal{K}$,
- *Güncelleme fonksiyonu* $Upd : \mathcal{K} \times \mathcal{S} \longrightarrow \mathcal{S}$ *ifadesini geçerleyen* $Upd_k : \mathcal{S} \longrightarrow \mathcal{S}$, $Upd_k(st) := Upd(k, st)$ *fonksiyonları her* $k \in \mathcal{K}$ *için birebir ve örten,*
- *Çıktı fonksiyonu* $Out : \mathcal{S} = GF(2)$ *(boole çıktı veren fonksiyon)*

Aşağıda bu tanıma uyan ve uymayan birer örnek verilmiştir.

Örnek: 10-bit uzunluğunda LFSR türünde L iç durumuna ve 20-bit uzunluğunda sabit K anahtarı ROM üzerinde bulunan bir kayan anahtar üretecinin güncelleme fonksiyonu f aşağıdaki gibi tanımlanmıştır. Bu üretecin *KSG with KUF* olma durumunu tartışınız.

$$f(L, K) = l_1 \oplus l_3 \oplus l_5 \oplus l_7 \oplus l_9 \oplus k_3k_7 \oplus k_{13}k_{18}$$
$$l_9' = f(L, K)$$

Çözüm: Güncelleme fonksiyonu verilen üreteç L iç durumunun l_9 bitinin yeni değerini oluşturmakta ve K anahtarının bitlerinden faydalanmaktadır. Bu yüzden bu üreteç bir *Anahtarlı Güncelleme Fonksiyonu olan Kayan Anahtar Üreteci*'dir.

Örnek: 48 bit uzunlukta NLFSR türünde N iç durumuna sahip ve 56 bit uzunlukta K anahtarı EEPROM üzerinde bulunan bir kayan anahtar üretecinin bir *Anahtarla Güncellenen Kayan Anahtar Üreteci* olduğu bilinmektedir. Bu üretecin güncelleme fonksiyonu f aşağıdakilerden hangisi olamaz?

$A)$ $f = n_0 \oplus n_1 n_3 \oplus n_5 k_7 \oplus n_2 k_{40} \oplus n_4 \oplus n_{24} k_{55}$

$B)$ $f = n_0 \oplus n_3 n_7 \oplus n_1 k_4 \oplus n_6 k_{10} \oplus n_4 \oplus n_{26} n_{27}$

$C)$ $f = n_0 \oplus n_2 n_4 \oplus n_0 k_4 \oplus n_3 n_9 \oplus n_5 \oplus n_{28} n_{29}$

$D)$ $f = n_0 \oplus n_2 n_4 \oplus n_0 n_4 \oplus n_3 n_9 \oplus n_5 \oplus n_{30} n_{31}$

$E)$ $f = n_0 \oplus n_2 n_4 \oplus k_0 n_4 \oplus n_3 n_9 \oplus n_5 \oplus n_{30} n_{31}$

Çözüm: A seçeneğinde $n_5 k_7$, $n_2 k_{40}$ ve $n_{24} k_{55}$ ifadeleri; B seçeneğinde $n_1 k_4$ ve $n_6 k_{10}$ ifadeleri, C seçeneğinde $n_0 k_4$ ifadesi ve son olarak E seçeneğinde $k_0 n_4$ ifadesi bulunduğundan dolayı bu fonksiyonların anahtarla güncellendiği anlaşılmaktadır. Toplam ifadeleri arasında k biti içermeyen tek fonksiyon D seçeneğinde belirtilendir. Bu yüzden cevap D'dir.

Tanım 2. **Anahtarlı Boole Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteci (ABGBF-KAÜ) [4]:** *Kara ve Esgin'in makalesinde [4] verilen tanıma göre bir AGF-KAÜ aşağıdaki şartları sağladığında* ***Anahtarlı Boole Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteci (Keystream Generator with Boolean Keyed Feedback Function)*** *olur:*

- *Kayan anahtarın her bir 1 -bit çıktısı için güncelleme fonksiyonu $\mathcal{F}$ ve çıktı fonksiyonu $\mathcal{G}$ bir kez çalıştırılmalı.*
- *$\mathcal{F}(\mathcal{K}, \mathcal{S})$ güncelleme fonksiyonu $\mathcal{S}$ iç durumu üzerinde 1-bit'lik değişiklik yapmalı. Eğer 1 bitten fazla sayıda değişiklik yapıyor ise değişikliğin yalnızca 1 bitlik bir kısmı anahtara bağlı olmalıdır.*

Örnek: Aşağıda güncelleme fonksiyonu verilen kayan anahtar üreteçlerinin hangileri ABGBF-KAÜ tanımına uymaktadır?

$I)$ $f \mid f = f_1 \longrightarrow s_{10},\ f_2 \longrightarrow s_{11}\ ve\ f_1 = s_4 \oplus s_5 \oplus s_6k_1\ ,\ f_2 = s_1 \oplus s_2 \oplus s_3k_2$

$II)$ $f \mid f = f_1 \oplus f_2 \longrightarrow s_{10}\ ve\ f_1 = s_4 \oplus s_5 \oplus s_6k_1\ ,\ f_2 = s_1 \oplus s_2 \oplus s_3k_2$

$III)$ $f \mid f = f_1 \longrightarrow s_{10},\ f_2 \longrightarrow s_{11}\ ve\ f_1 = s_4 \oplus s_5 \oplus s_6k_1\ ,\ f_2 = s_1 \oplus s_2 \oplus s_3$

Çözüm: I'de güncelleme fonksiyonu f_1 ve f_2 olmak üzere iki alt fonksiyondan oluşmaktadır. f_1 iç durumun s_{10} bitini, f_2 ise s_{11} bitini güncellemektedir. f_1 fonksiyonu s_6k_1 ifadesi olduğu için anahtara bağımlı, f_2 fonksiyonu ise s_3k_2 ifadesinden dolayı anahtara bağımlıdır. İç durumda anahtara bağlı olarak iki bit değiştiği için I. madde ABGBF-KAÜ'ye ait değildir.

II'de f fonksiyonu I'de olduğu gibi iki alt fonksiyondan meydana gelmekte; fakat nihayetinde iç durumdan yalnızca 1 bit, yani s_{10} değişmektedir. Bu sebeple fonksiyonlar anahtarın hangi bitlerini kullanırsa kullansın yalnızca güncellemeden s_{10} etkilendiği için II ABGBF-KAÜ'ye ait bir fonksiyondur.

III'te ise iç durumun s_{10} ve s_{11} bitleri değişmektedir; fakat s_{11}'i değiştiren f_2 fonksiyonunda anahtar bitleri kullanılmamıştır. Bu sebeple III de ABGBF-KAÜ'ye ait bir fonksiyon olarak değerlendirilebilir.

4.2 Sprout Algoritması

Bu kısımda Armknecht ve Mikhalev'in FSE 2015 etkinliğinde yayınladığı makalesinde [1] AGF-KAÜ(KSG with KUF) olarak tanımladıkları türe ait verdikleri ilk somut örnek olarak Sprout ayrıntılı şekilde incelenecektir. Öncelikle Sprout'un ilham alındığı tasarım, mevcut dizi şifreleme algoritmalarındaki aksayan ve donanım tasarımında zorluğa neden olan noktalar açıklanmıştır. Daha sonra bu zorluklara karşı önerilen çözüm modeli görsellerle desteklenerek anlatılmıştır.

4.2.1 Çıkış Noktası

Güç gereksinimi ve donanım alanının kritik olduğu RFID etiketleri ve WSN cihazlarında hafıza bileşenlerinin etkin kullanılması çok önemlidir. Zira NLFSR ve LFSR gibi elemanlar, geri besleme fonksiyon devresi ile birlikte en çok güç tüketen ve yer kaplayan bileşenler arasındadır [58].

Mikhalev ve Armknecht Sprout'u anlattığı makalesinde bu konuya önem vermiş ve ilklendirme fazına ilave olarak iç durumun değiştirildiği güncelleme fonksiyonlarında devrede sabit bir hafızada bulunan gizli anahtarın kullanıldığı yeni modeli önermiştir [1]. TMDTO(time-memory-data-tradeoff) saldırılarına karşı savunulan, iç durumun anahtardan iki kat uzun olması gerektiği ile ilgili altın kural bu sayede geçersiz olacaktı. Grain 128a'dan yola çıkılarak tasarlanan Sprout'un kendisi kritik güvenlik açıklıkları sebebiyle bu alanda sık kullanılan bir algoritma olmadı; ama Fruit-80, Plantlet gibi yeni somut algoritmalara ilham kaynağı oldu [29][3].

4.2.2 Tasarım

Tıpkı Grain 128a'da olduğu gibi Sprout'un iç durumu LFSR ve cebirsel saldırıların önüne geçmek için NLFSR türü saklayıcıların bileşiminden meydana gelmektedir. 40-bit'lik LFSR işleyiş sırasında yine 40-bit'ten oluşan NLFSR'ı beslemektedir. z_i çıktı fonksiyonu ise her iki saklayıcının bitlerini kullanmaktadır. Sprout iç durum boyutuyla aynı şekilde 80-bit'lik bir gizli anahtar barındırır. Yapıdaki 9-bit'lik bir sayaç ise 80-bit'lik anahtar ile birlikte işleyişte NLFSR'ı güncellemekte kullanılır. Algoritmanın anahtar bağımlı olmasını sağlayan şey **Round Key Function(Anahtar Çevrim Fonksiyonu)**'dur. Şekil 4.1'de görüldüğü gibi LFSR, NLFSR ve sayaç değerleri bu fonksiyonda kullanılmaktadır. Fonksiyon aşağıda verilmiştir.

$$k_t^* = k_t,\ 0 \leq t \leq 79;$$
$$k_t^* = (k_{t\ mod\ 80}) \cdot (l_4 + l_{21} + l_{37} + n_9 + n_{20} + n_{29}),\ t \geq 80;$$

Bu fonksiyonda görüldüğü üzere ilk 80 saat vuruşunda tüm anahtar bitleri kullanılmakta, daha sonraki işletimlerde ise sayacın 80'e bölümünden kalanı $l_4 + l_{21} + l_{37} + n_9 + n_{20} + n_{29}$ toplamıyla AND işlemine tabi tutulmaktadır. LFSR saklayıcısını güncelleyen fonksiyon

ise aşağıda verilmiştir:

$$P(x) = x^{40} + x^{35} + x^{25} + x^{20} + x^{15} + x^6 + 1$$

Sprout'un çıktı fonksiyonunun bir parçası olan $h(x)$ ise LFSR ve NLFSR saklayıcı bitlerini kullanan lineer olmayan bir fonksiyondur. Tanımda $x_0, ..., x_8$ değişkenleri sırasıyla $n_{t+4}, l_{t+6}, l_{t+8},\ l_{t+10},\ l_{t+32},\ l_{t+17},\ l_{t+19},\ l_{t+23},\ t_{t+38}$ bitlerini temsil etmekte olup, $h(x)$ aşağıdaki gibi tanımlanmıştır:

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$$

Çıktı bitini oluşturan asıl fonksiyon olan z_i ise $h(x)$, l_{t+30} ve NLFSR saklayıcısının sayaçla değişen 7 adet bitinin toplamınla XOR işlemine tabi tutulur. NLFSR'daki bu bitler aşağıdaki toplam sembolüyle ifade edilmiştir.

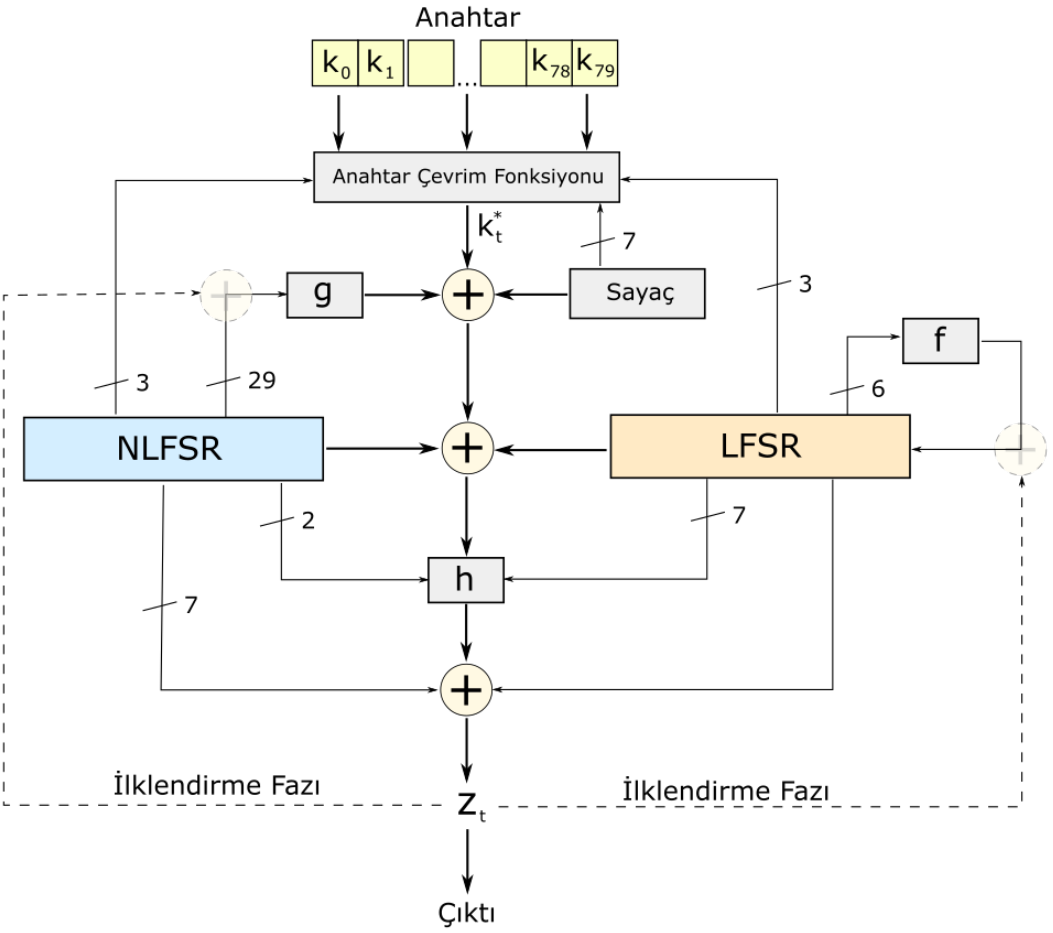
$$\sum_{j \in B} n_{t+j}\ ve,$$
$$B = \{1, 6, 15, 17, 23, 28, 34\}$$

Çıktı fonksiyonunun en sade hali aşağıdaki gibidir:

$$z_t = h(x) + l_{t+30} + \sum_{j \in B} n_{t+j}$$

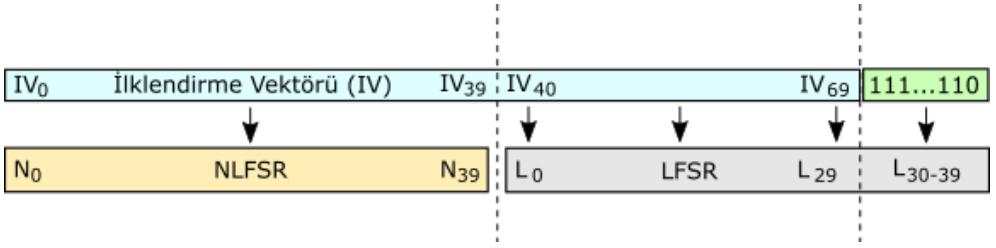
4.2.3 İlklendirme Fazı

İlklendirme işleminin ilk adımı olarak IV'ye(İlklendirme Vektörüne) ait bitler NLFSR'a yüklenir. IV'nin geri kalan 30 biti ise LFSR'a, LFSR'ın kalan son 10 bitine ise l_{30}'den l_{38}'e kadar '1', son bit olan l_{39}'a '0' yüklenir. Son olarak sistem 320 saat vuruşu kadar çıktı vermeksizin işletilir. Çıktı vermek yerine bu değer LFSR ve NLFSR'a geri besleme değeri olarak kullanılır. l_{t+40} değeri ilklendirme çıktısı olan z_t ve kendi geri besleme fonksiyonundan gelen değer ile XOR'lanarak $l_{t+40} = z_t \oplus f(L)$ olur. NLFSR'dan n_{t+40} biti ise ilklendirme aşamasına özel olarak çıktı değeri z_t, anahtar çevrim fonksiyonu k_t^*, LFSR'dan bir bit olan l_t, sayaç biti c_t^4 ve NLFSR'ın kendi geri besleme fonksiyonu



Şekil 4.1: Sprout'un tasarımı

olan $g(N_t)$'nin XOR'lanması ile işletilir. Yani, $n_{t+40} = z_t \oplus k_t^* \oplus l_t \oplus c_t^4 \oplus g(N_t)$ olur. İlklendirmenin özeti Şekil 4.2'de verilmiştir.



Şekil 4.2: Sporut'un ilklendirme fazı

4.2.4 Gerçekleştirilen Saldırılar

FSE-2015'te duyurulduktan sonra Sprout ile ilgili birçok kriptanaliz makalesi yayınlandı [27][26][25]. Önce Lallemand ve Naya-Plasencia [25] Sprout üzerinde çalışan ve tam tarama işleminden (exhaustive search) 2^{10} kat daha hızlı bir şekilde anahtarı tespit eden bir algoritma geliştirdi. Daha sonra Kara ve Esgin [27] ISR(İç Durum Geri Kazanım) ve KR(Anahtar Geri Kazanım) algoritmalarıyla bu işlemi 2^{10} kat daha hızlandırdı. Son olarak Zhang ve Gong anahtar tespit işlemini daha düşük bir veri karmaşıklığı ile yeniden düzenleyerek Kara ve Esgin saldırısını da 2^{10} kat hızlandırmış oldu [26].

Sprout'a yapılan saldırılar anahtar bitinin lineer olmayan şekilde güncelleme fonksiyonuna karışmış olmasından kaynaklanmaktadır. Zira, *Anahtar Çevrim Fonsiyonu* üzerinden gelen k_t^* biti AND işlemiyle güncelleme fonksiyonunda yer almakta. Bu da çarpımın diğer yüzünün, yani $(l_4 + l_{21} + l_{37} + n_9 + n_{20} + n_{29})$ ifadesinin 1 olduğu anlarda bu bitin tahmin edilebilmesini mümkün kılmaktadır. Saldırı yeterince süre çalıştırıldığında iç durum bitleri ve anahtar bitleri elde edilebilmektedir.

Bölüm 5

ABGBF-KAÜ Ailesine Yönelik Genel Kapsamlı Saldırı

Bu bölümde Tanım 2'de tarifi yapılan ABGBF-KAÜ(KSG with Boolean KFF) için Kara ve Esgin'in makalesinde [4] modellenen saldırı algoritması genel hatlarıyla anlatılacaktır. Şimdiye kadar yapılan kriptanaliz çalışmaları Sprout'u doğrudan hedef alan çalışmalardı [25–27]. Kara ve Esgin'in 2018 yılında yayınlanan makalesinde [4] akademide ABGBF-KAÜ tanımı ilk kez yapılmış ve bu kayan anahtar üreteç ailesine genel çapta bir saldırı modellenmiştir. Tez çalışmamızda bu algoritma zaman ve veri karmaşıklığı yönlerinden geliştirilmiştir. Geliştirilmiş algoritma Bölüm 6'da anlatılmıştır.

5.1 Saldırının Açıklaması

KE saldırısının önemli bir kısmını İç Durum Geri Kazanma için yapılan (Internal State Recovery) algoritması teşkil etmektedir. İDGK algoritması olası tüm iç durumları, kayan anahtar çıktısına bakarak test edip, uygunsuz olanları eleme üzerine kuruludur. İDGK algoritması yalnızca zayıflık teşkil eden iç durumlar üzerinde çalışır. Zira zayıflık teşkil eden iç durumlar anahtar hakkında ipucu verme ihtimali olan iç durumlardır. İç durumun tahmin edilebilirliği ise Pr_g (Guess Capacity, Probability of Guess) başka bir deyişle Tahmin Kapasitesi olarak geçmektedir.

5.1.1 Tahmin Kapasitesi (Pr_g)

Saldırının en önemli öncülü dizi şifreleme algoritmasının güncelleme fonksiyonunun 0.5 üzeri bir ortalama tahmin kapasitesi ihtiva ediyor olmasıdır. Yani $Pr_g > 0.5$ olmalıdır. Aksi durumda tahmin etme mekanizması çalışmayacaktır. Kara ve Esgin'in [4] tanımına göre ileri yönde işletilen S iç durumuna ait Tahmin Kapasitesi formülü aşağıdaki gibidir:

$$Pr_g(S)_f = \frac{1}{2} + |\frac{\#\{K : f_{\mathcal{F}}(K,S) = 0\}}{2^k} - \frac{1}{2}|$$

Herhangi S iç durumundan bağımsız olarak bir algoritmanın güncelleme fonksiyonuna ait Ortalama Tahmin Kapasitesi [4] formülü ise aşağıdaki gibidir:

$$Pr_g = \frac{1}{2} + 2^{-s} \sum_{S} |\frac{\#\{K : f_{\mathcal{F}}(K,S) = 0\}}{2^k} - \frac{1}{2}|$$

Formüldeki, $\#\{K : f_{\mathcal{F}}(K,S) = 0\}$ ifadesi, f ileri yönde işletim için güncelleme fonksiyonunun olası tüm anahtar ihtimalleri için kaç adet '0' sonucu vereceğinin sayısı olup, k ise anahtar boyunun bit cinsinden değeridir. Mutlak değer içerisinde olmasının nedeni çıktıların '0' ya da '1' yönelimli oluşmasının aynı sonucu vermesi için kullanılmıştır. Örneğin güncelleme fonksiyonu daima '0' üretiyor olsun. $\frac{\#\{K:f_{\mathcal{F}}(K,S)=0\}}{2^k}$ ifadesi 1 sonucunu vereceği için $Pr_g = 1$ olur. Daima '1' üreten bir güncelleme fonksiyonunda ise $\frac{\#\{K:f_{\mathcal{F}}(K,S)=0\}}{2^k}$ ifadesi 0 sonucunu vereceği için yine $Pr_g = 1$ olur.

Ortalama Tahmin Kapasitesi formülü ise basitçe, tüm iç durumlar için hesaplanan tahmin kapasitelerinin aritmetik ortalamasıdır. s sembolü iç durum büyüklüğünün bit cinsinden değeridir.

5.1.2 Çıktı Kapasitesi (θ)

Bir saklayıcının geri besleme değerini bilmeden en fazla kaç kez işletilerek çıktı alınabileceğini gösteren parametredir. Tamamen geri besleme fonksiyonunun saklayıcıya olan bağlantı noktalarıyla ilgilidir. Örneğin LSB'ye doğru kayan 20-bit'lik bir LFSR'ın geri besleme fonksiyonu $f(s) = s_0 \oplus s_3 \oplus s_{15}$ ise Çıktı Kapasitesi $\theta = 19 - 15 + 1 = 5$'tir. İlave edilen 1 iç durumun hiç ilerletilmeden o anki halinden de çıktı alınabildiği için eklenmiştir.

5.1.3 Karavana İhtimali (ϵ)

İDGK algoritmasının kayan anahtarı veren iç durumlardan doğru olanı gözden kaçırma ihtimalidir. Bu sayı algoritmanın çalışma süresiyle ters orantılıdır. Algoritmanın %99.9 başarımla çalışması isteniyorsa bu sayıya 0.001 verilmelidir.

5.1.4 Sonlandırma Değeri (α_{ter})

İDGK algoritmasının, sistemi kaç saat vuruşu kadar ilerleteceğini gösteren değerdir. Algoritmanın saat sayacı bu değere ulaşınca sonlanır. α_{ter} değeri aşağıdaki formülle hesaplanır:

$$\alpha_{ter} = \frac{1}{(Pr_g - 1)^2}\left(\sqrt{-2ln\epsilon} + \sqrt{2ln2 \cdot (s - \theta - d - 1)}\right)^2$$

5.1.5 Eşik Değeri (α_{thr})

İDGK algoritmasında iç durumlar taranırken her bir iç durum için uyuşmazlık sayacı (Mismatch Counter) tutulur. Bu sayaç eşik değerini geçtiğinde elenmiş olur. Bu eşik değeri olan α_{thr} aşağıdaki formülle hesaplanır:

$$\alpha_{thr} = \lfloor \alpha_{ter}(1 - Pr_g + \alpha_{ter}) \rfloor$$

5.1.6 İç Durum Zaafiyet Göstergesi (d)

İDGK çalıştırılmadan önce, istenilirse, her bir iç duruma ait *anahtar bilgisi olmadan hesaplanabilecek çıktı biti sayısı* kontrol edilerek, bir tabloda tutulabilir. Kara ve Esgin'in makalesinde [4] Offline Phase(Çevrimdışı Faz) olarak geçmektedir. Zorunlu bir işlem adımı değildir. Çevrimdışı Faz çalıştırılmazsa tüm işlemlerde bu değer '0' alınır.

5.2 İç Durum Geri Kazanım Algoritması

İDGK algoritması girdi olarak birden fazla iç durum ve kayan anahtar çıktısını alır ve sonuç olarak o kayan anahtarı üretmekte kullanılan iç durumu tespit eder. Bunu yaparken

önceki kısımda anlatılan parametreleri kullanır. Algoritma çağrılmadan önce ϵ kullanıcı girdisi olarak alınır. Buna bağlı olarak Pr_g, ardından da Pr_g değerine bağlı olan α_{ter} ve α_{thr} değerleri hesaplanır.

5.2.1 İDGK Sözde Kodu

Kara ve Esgin'in makalesinde [4] İç Durum Geri Kazanım Saldırısı (Internal State Recovery Attack) olarak tanımladığı bu algoritma, saldırı mekanizmasının daha iyi anlaşılması için ayrıntılı şekilde izah edilmiştir.

Girdiler: Boş olmayan bir iç durum aday listesi **S**, kayan anahtardan bir parça $\{ z_{t+1+\theta_f}, ..., z_{t+\theta_f+\alpha_{ter}} \}$, sonlandırma değeri olan α_{ter}, ortalama tahmin kapasitesi Pr_g ve karavana ihtimali ϵ

Kodun Açıklaması

KE saldırısında bahsi geçmemiş olmasına karşın α_{ter}'in hesaplanması, θ değerinin kullanıcı tarafından girilmesi ve kayan anahtarın yeterli uzunlukta çıkartılıp önceden hazırlanmış olması gerekmektedir. Ayrıca bu noktada boole dönüşü olan *IsPossibleToDetermine* alt yordamının içinde çözülen denklem geri besleme fonksiyonun gerçeklenme şekline bağlı olarak değişmektedir.

Daha iyi anlaşılabilmesi için bu işlemi bir örnekle açıklayalım. 8-Bit'lik tek bir LFSR'dan meydana gelen, $f_t(s) = s_3 \oplus s_5 k_3 \oplus s_7 k_4$ geri besleme fonksiyonu ile güncellenen, $z_t(s) = s_3 \oplus s_5 \oplus s_7$ çıktı fonksiyonu ile kayan anahtar üreten bir dizi şifreleme algoritması olsun. Bu noktada geri besleme değerinin belirlenebilmesi için geri besleme fonksiyonda lineer olmayan şekilde yer alan anahtar bitlerinin hiçbirinin sıfırlanmaması gerekir. Dolayısıyla s_5 ve s_7 bitlerinin 1 olduğu yerlerde *IsPossibleToDetermine* alt yordamı 'true' değeriyle sonuçlandığı için algoritma, çıktının θ saat vuruşu kadar ilerisini kontrol ederek geri besleme değerini belirleyebilir. Belirlemekte kullandığı fonksiyon da çıktı fonsiyonunun θ kadar kaydırılmış hali olur. Çıktı fonksiyonunu $\theta = 1$ kadar kaydırdığımızda s_7 biti LFSR 1 kez kaydırıldığı için $f_t(s)$ haline gelecektir. $f_t(s) = z_{t+1}(s) \oplus s_4 \oplus s_6 \oplus$ denklemi çözüldüğü takdirde geri besleme değeri $f_t(s)$ belirlenmiş olur.

Algoritma 2.1 İç Durum Geri Kazanım [4]

procedure ISR($\mathbf{S}, z, \epsilon, Pr_g$) ▷ Saldırının ana fonksiyon imzası
  $\epsilon_{ter} \leftarrow \sqrt{\frac{-ln\epsilon}{2\alpha_{ter}}}$ ▷ ϵ_{ter} hesapla
  $\alpha_{thr} \leftarrow \lfloor \alpha_{ter}(1 - Pr_g + \alpha_{ter}) \rfloor$ ▷ α_{thr} hesapla
  $CUR \leftarrow Liste < s > ()$ ▷ İşlem gören iç durumları tutacak boş liste oluştur
  $NEW \leftarrow Liste < s > ()$ ▷ Devredecek iç durumları tutacak boş liste oluştur
  $CUR \leftarrow \mathbf{S}$ ▷ Aday listesini CUR'e doldur
  $Her\ \#MM(s) \in \mathbf{S} \leftarrow 0$ ▷ Her bir s iç durumu için uyuşmazlık sayacını sıfırla
  for $i \leftarrow t, (t + \alpha_{ter} - 1)$ **do** ▷ i değişkeni t'den $t + \alpha_{ter} - 1$'e kadar döngü
    for each $s : CUR$ **do** ▷ CUR'deki her bir s iç durumu için döngü
      if $Pr_g(s) = 0.5$ **then**
        $fb_{sugg} \leftarrow null$
        $\#MM(s) \leftarrow \#MM(s) + 0.5$ ▷ Uyuşmazlık sayacını 0.5 artır
      else
        $fb(s) \leftarrow fb_{sugg}$
      end if
      if IsPossibleToDetermine($fb_{i+1}, z_{i+1+\theta_f}$) **then**
        Determine fonksiyonunu çalıştır (Algoritma 2.2)
      else
        Check & Guess fonksiyonunu çalıştır (Algoritma 2.3)
      end if
    end for
    if IsEmpty(NEW) **then**
      Döngüden çık
    end if
    $CUR \leftarrow NEW$ ▷ NEW'in içeriği CUR'e kopyalanır.
    $NEW \leftarrow Liste < s > ()$ ▷ Liste boşaltılır.
  end for
  return CUR listesindeki adayların *kökleri(root'ları)*
end procedure

Algoritma 2.2 İç Durum Geri Kazanım - Determine [4]

procedure DETERMINE($\mathbf{S}, z, \epsilon, Pr_g$) ▷ Alt yordamın ana fonksiyon imzası
  Geri besleme değerini fb_{det} karşılık gelen z çıktı biti üzerinden hesapla
  s iç durumunu fb_{det} kullanarak güncelle
  if $f_{sugg} \neq null$ **then** ▷ Önerilen değer mevcutsa...
    if $fb_{sugg} \neq fb_{det}$ **then** ▷ Önerilen ve belirlenen değer uyuşmuyorsa...
      $\#MM(s) \leftarrow \#MM(s) + 0.5$ ▷ Uyuşmazlık sayacını 0.5 artır
    end if
  end if
  if $\#MM(s) \leq \alpha_{thr}$ **then** ▷ Uyuşmazlık değeri eşik değerinden küçük veya eşitse
    Güncellenmiş s iç durumunu, $\#MM(s)$ değerini ve *kök* değerini NEW'e ekle
  end if
end procedure

5.3 Determine Algoritması

Açıklama: Determine (Belirleme) alt yordamı Tahmin Kapasitesi $Pr_g > 0.5$ olan bir iç durum aynı zamanda belirleme işlemini sağlıyorsa çalışır. Belirlenen geri besleme değeriyle f_{det}, mevcut s iç durumu işletildiğinde oluşan $f_i(s)$ geri besleme değeri karşılaştırılır. Farklı olduğu durumlarda s iç durumu için tutulan $\#MM(s)$ uyuşmazlık değeri 1 artırılarak güncellenir. Uyuşmazlık değeri eşik değeri aşıyorsa s iç durumu elenir.

5.4 Check & Guess Algoritması

Algoritma 2.3 İç Durum Geri Kazanım - Check & Guess [4]

procedure CHECKANDGUESS(s, α_{thr}, NEW) ▷ Alt yordamın ana fonksiyon imzası
  if $z_i(s) \neq keystream_i$ **then** ▷ s ile üretilen çıktı biti kayan anahtardaki bitle uyuşmuyorsa
    s iç durumunu ele (NEW'e eklenmezse elenmiş olur)
  else
    s iç durumundan s^0 ve s^1 olmak üzere iki kopya oluştur
    s^0 iç durumunu '0' geri beslemesiyle güncelle
    s^1 iç durumunu '1' geri beslemesiyle güncelle
    if $fb_{sugg} \neq null$ **then** ▷ Önerilen değer mevcutsa...
      if $fb_{sugg} == 0$ **then** ▷ Önerilen değer 0 ise...
        $\#MM(s^1) \leftarrow \#MM(s^1) + 1$ ▷ s^1'in uyuşmazlık değerini 1 artır
      else
        $\#MM(s^0) \leftarrow \#MM(s^0) + 1$ ▷ s^0'ın uyuşmazlık değerini 1 artır
      end if
    end if
    if $\#MM(s^0) \leq \alpha_{thr}$ **then** ▷ s^0'ın uyuşmazlık değeri eşik değerin altındaysa
      s^0 iç durumunu, $\#MM(s^0)$ değerini ve *kök* değerini NEW'e ekle
    end if
    if $\#MM(s^1) \leq \alpha_{thr}$ **then** ▷ s^1'ın uyuşmazlık değeri eşik değerin altındaysa
      s^1 iç durumunu, $\#MM(s^1)$ değerini ve *kök* değerini NEW'e ekle
    end if
  end if
end procedure

Açıklama: Eğer bir s iç durumunun geri besleme değeri belirlenemiyorsa iki farklı ihtimal ile işletilir. Check&Guess işlemi bu işlemi yaparken öncelikle s'nin oluşturduğu çıktı ile i saat vuruşu anındaki kayan anahtarın çıktı bitinin aynı olup olmadığını kontrol eder. Aynı çıktı oluşturuluyorsa, s iç durumunun s^0 ve s^1 olmak üzere iki kopyasını oluşturarak sırasıyla 0 ve 1 geri besleme değerleriyle işletir. Önerilen değer f_{sugg} mevcut

ise geri besleme değeri örtüşmeyen kopyanın uyuşmazlık sayacını artırır. Son olarak uyuşmazlık değerlerinin eşik değeri geçip geçmediğine bakılarak eleme işlemi yapılır.

5.5 Anahtar Geri Kazanım Fazı

Anahtar gezi kazanımı için, İDGK sonucunda doğru olduğu tespit edilen bir iç durumun t anındaki hali ve o andaki belirlenmiş geri besleme değeri kullanılarak anahtar bitleri elde edilir. Bu işlem için Algoritma 2.1 (İGDK) algoritmasının değiştirilmiş bir sürümünün kullanılabileceği Kara ve Esgin'in makalesinde [4] Key Recovery Phase bölümünde ifade edilmiştir. Bu tez çalışmasındaki iyileştirme ve düzeltmelere Anahtar Geri Kazanım Fazı dahil değildir.

AGK fazının her bir t saat vuruşu anında yaptığı işi basit bir örnekle açıklayalım. 8-bit'lik anahtara ve LFSR boyuna sahip, $f_t(s) = s_3 \oplus s_5.k_0 \oplus s_7.k_2$ geri besleme fonksiyonu ve $z_t(s) = s_2 \oplus s_4 \oplus s_6$ çıktı fonksiyonuna sahip olsun. Anahtar bitleriyle lineer olmayan durumdaki s_5 ve s_7 bitleri t anında 1 olduğu $s_{0-7} = \{0, 1, 0, 1, 0, 1, 0, 1\}$ iç durumu için görüldüğü üzere $f_t(s) = z_{t+1} \oplus s_3 \oplus s_5$ şeklinde belirlenebilmektedir. Üreteç çıktısında $z_{t+1}(s) = 1$ için;

$$f_t(s) = 1 \oplus 1 \oplus 1$$
$$f_t(s) = 1$$

olur. Şimdiyse elde edilen $f_t(s)$ değeri geri besleme fonksiyonunda yerine konulduğunda;

$$f_t(s) = s_3 \oplus s_5.k_0 \oplus s_7.k_2$$
$$1 = 1 \oplus 1.k_0 \oplus 1.k_2$$
$$k_0 \oplus k_2 = 1 \oplus 1$$
$$k_0 \oplus k_2 = 0$$

olur. Algoritma ilerletildikçe anahtar parça parça tamamlanacak ve halen bilinmeyen anahtar bitleri kaldıysa bunlar tam tarama işlemi yapılarak elde edilecektir. Sistem t'den $t+1$'e geçtiğinde anahtardan gelen bitler de kayacağı için, $k_7^{t+1} = k_0^t$ ve $k_0^{t+1} = k_1^t$ ve $k_1^{t+1} = k_2^t$ olur. $t+1$ anında elde edeceğimiz anahtar bitlerini içeren denklem de,

eğer geri besleme değeri belirlenmişse, $k_1^t \oplus k_2^t = k_0^{t+1} \oplus k_1^{t+1}$ olur. Bu yöntemle 11 saat vuruşunda elde edilen bitleri alt alta yazarsak;

$$
\begin{aligned}
k_0^t \oplus k_2^t &= 0 \\
k_1^t \oplus k_3^t &= 1 \\
k_2^t \oplus k_4^t &= ? \\
k_3^t \oplus k_5^t &= ? \\
k_4^t \oplus k_6^t &= ? \\
k_5^t \oplus k_7^t &= 0 \\
k_6^t \oplus k_0^t &= ? \\
k_7^t \oplus k_1^t &= ? \\
k_0^t \oplus k_2^t &= ? \\
k_1^t \oplus k_3^t &= 1 \\
k_2^t \oplus k_4^t &= \mathbf{1} \\
k_3^t \oplus k_5^t &= ?
\end{aligned}
$$

Görüldüğü gibi geri besleme belirlenebildiği durumlarda anahtarlar elde edilebildi. Üçüncü satırda $k_2^t \oplus k_4^t$ elde edilemezken, ilerleyen süreçle geri besleme değeri belirlendiği için bu ifadenin $'1'$ olduğu görülmektedir. İşletime devam edilip tüm bitlerin ilişkileri tespit edildiğinde geriye tüm anahtar ihtimalleri arasından eldeki denklemlere uyanlar bulunur. Bu şekilde gizli anahtar elde edilmiş olur.

Bölüm 6

Geliştirilmiş Saldırı Algoritması

Önceki bölümde Kara ve Esgin'in makalesinde [4] ABGBF-KAÜ ailesi için tanımlamış olduğu saldırı metodunun nasıl çalıştığı ayrıntılı olarak incelenmişti. Bu bölümde öncelikle bu saldırı metodu performans bakımından incelenmiş ve darboğaz oluşturan noktalar vurgulanmıştır. Daha sonra iyileştirme ve düzeltme adına üç nokta belirlenmiş ve bunların performans üzerindeki etkisi ortaya konulmuştur.

6.1 Mevcut Algoritmadaki Darboğaz Noktaları

Kara ve Esgin'in makalesinde tarifi yapılan [4] İDGK algoritması *Çevrimdışı Faz* çalıştırılmadığında tüm iç durum ihtimallerinin taranmasını gerektirmektedir. Algoritma işletilirken s-bit'lik bir iç durum boyutu için 2^s sayıda iç durum taranmakta, geri besleme belirlenemediği anlarda tetiklenen Check & Guess Algoritması (2.2) sebebiyle algoritmanın çalıştırıldığı ortamda Ek-A'da görüldüğü gibi yüzde 40 artan bir bellek gereksinimi ortaya çıkmaktadır (Bkz.: Ek-A).

Bu artan bellek gereksinimi *Kayan Pencere* yöntemi ile her defasında sınırlı sayıda iç durum ile algoritma çalıştırılarak ve son tahlilde üretilen çıktılar birleştirilerek önüne geçilebilir. Bu yöntem kullanıldığında bellek gereksinimi kontrol altına alınabilmektedir. Testlerimizde kayan pencere büyüklüğü 1000 olarak alınmıştır. Yani 2^{20} adet iç durum 1000'erli gruplar halinde algoritmada ayrı ayrı işlenmiş ve sonuçlar derlenmiştir.

6.2 Hata Düzeltmesi

Kara ve Esgin'in makelesindeki Determine (Algoritma 2.2) algoritmasında elenmek üzere olan (yani uyuşmazlık sayacı α_{thr} değerini geçen) iç durum belirlenmiş geri besleme değeri fb_{det} ile işletilmektedir. Sonucu etkilemeyen bu fazladan işlem nedeniyle Determine fonksiyonu içerisinde elenerek yok edilen her bir iç durum için İDGK algoritmasının 1 saat vuruşu fazladan çalıştırılması anlamına gelir. Aşağıda tarifi yapılan düzeltme sayesinde bu fazladan işletimin önüne geçilmiştir. Düzeltilmiş Determine algoritması aşağıdaki gibidir:

6.2.1 Sözde Kodlar

İyileştirilmiş Determine Algoritması (İyileştirme No:2)

procedure DETERMINE($\mathbf{S}, z, \epsilon, Pr_g$) ▷ Alt yordamın ana fonksiyon imzası
  if $f_{sugg} \neq null$ **then** ▷ Önerilen değer mevcutsa...
    if $fb_{sugg} \neq fb_{det}$ **then** ▷ Önerilen ve belirlenen değer uyuşmuyorsa...
      $\#MM(s) \leftarrow \#MM(s) + 1$ ▷ Uyuşmazlık sayacını 1 artır
    end if
  end if
  if $\#MM(s) \leq \alpha_{thr}$ **then** ▷ Uyuşmazlık değeri eşik değerinden küçük veya eşitse
    Geri besleme değerini fb_{det} karşılık gelen z çıktı biti üzerinden hesapla
    s iç durumunu fb_{det} kullanarak güncelle
    Güncellenmiş s iç durumunu, $\#MM(s)$ değerini ve *kök* değerini NEW'e ekle
  end if
end procedure

Düzeltme işlemi öncelikli olarak uygulanmış, 1 No'lu ve 3 No'lu iyileştirmeler düzeltilmiş algoritma üzerine kurgulanmıştır. Bölüm 7'da düzeltmenin performans üzerindeki etkisi gösterilmiştir.

6.3 İyileştirme No:1

1 No'lu iyileştirmemizde Tahmin Kapasitesi için yeni bir tanım yapılmış ve İDGK algoritmasında önerilen geri besleme değeri dikkate alınarak her bir aday iç durum bağımsız olarak işletilmiştir. Böylece tahmin yürütülemeyen, yani Tahmin Kapasitesi $Pr_g(s) = 0.5$ olan iç durumlar için algoritma çekimser davranıp hiçbir işlem yapmadan iç durumu işletmeye devam etmeyi öngörmektedir.

Hatırlatmak gerekirse, İDGK algoritmasının orijinal hâlinde Tahmin Kapasitesi $Pr_g(s) = 0.5$ olduğu durumlarda iç durumun uyuşmazlık sayacı 0.5 puan artırılıyordu. Bu yaklaşımın iki adet sorunu vardır. İlki, üzerinde çalışılan iç durum doğru olan ise, algoritma uyuşmazlık sayacını haksız yere 0.5 puan artırmış olur. Doğru olan iç durumun elenmesine sebep olabilir. İkincisi ise geri besleme tahmininde avantaj sağlamayan $Pr_g(s) = 0.5$ anları için algoritmanın boş yere çalışarak, kriptanaliz gerçekleştiren sistemin kaynaklarını boşa tüketmesidir.

6.3.1 İyileştirilmiş Algoritma

Ortalama Tahmin Kapasitesi formülünün oluşturulmasında da yukarıda tarifi yapılan çekimser davranma yöntemi kullanılmıştır. Ortalama alınırken $Pr_g(s) = 0.5$ olan iç durumlar göz ardı edileceğinden, 2^s adet iç durum yerine, $Pr_g(s) \neq 0.5$ olduğu a adet iç durum hesaba katılacaktır. Bu iç durumları barındıran bir A kümesi için formül aşağıdaki duruma gelir.

$$Pr_g^{eski} = \frac{1}{2} + 2^{-s} \sum_{S} |\frac{\#\{K : f_{\mathcal{F}}(K,S) = 0\}}{2^k} - \frac{1}{2}|$$

$$Pr_g^{yeni} = \frac{1}{2} + \frac{1}{a} \sum_{S \in A} |\frac{\#\{K : f_{\mathcal{F}}(K,S) = 0\}}{2^k} - \frac{1}{2}|$$

Yeni algoritmada her bir iç durum bağımsız olarak testlere girdiğinden algoritmanın her halukarda durması için t_{max} parametresini ilave ettik. Bu parametre kullanıcı inisiyatifinde bir parametre olup α_{ter}'in iki katından büyük bir sayı olması önerilir. Saldırının hazırlık sürecinde önceden üretilmesi gereken kayan anahtar çıktısı da en az t_{max} uzunluğunda olacak şekilde değiştirilmelidir.

Girdi olarak algoritmaya verilen **S** dizisindeki her bir iç durum orijinal algoritmada α_{ter} kadar işletilirken, iyileştirilmiş algoritmada ise her bir iç durumun α_{ter} sayısı kadar teste girmesi sağlanır. Yani $Pr_g \neq 0.5$ olduğu durumlar sayılır, bu süreçte Orijinal Algoritmada olduğu gibi uyuşmazlık sayacı α_{thr} değerini geçenler elenir. α_{ter} adet teste girip elenmeden kalanlar ise *STORED* adını verdiğimiz ayrı bir listeye alınır. Son olarak *STORED* listesine alınan iç durumlara ait eşsiz kök değerler tespit edilerek gösterilir.

6.3.2 Sözde Kodlar

Geliştirilmiş İç Durum Geri Kazanım (İyileştirme No:1)

procedure ISR($\mathbf{S}, z, \epsilon, t_{max}, Pr_g$) ▷ Saldırının ana fonksiyon imzası
  $\epsilon_{ter} \leftarrow \sqrt{\frac{-ln\epsilon}{2\alpha_{ter}}}$ ▷ ϵ_{ter} hesapla
  $\alpha_{thr} \leftarrow \lfloor \alpha_{ter}(1 - Pr_g + \alpha_{ter}) \rfloor$ ▷ α_{thr} hesapla
  $CUR \leftarrow Liste < s > ()$ ▷ İşletilmekte olan iç durumların listesi
  $NEW \leftarrow Liste < s > ()$ ▷ Sonraki iterasyona devredecek iç durumların listesi
  $STORED \leftarrow Liste < s > ()$ ▷ Testlerden geçen iç durumların listesi
  $CUR \leftarrow \mathbf{S}$ ▷ Aday listesini CUR'e doldur
  $Her\ \#MM(s) \in \mathbf{S} \leftarrow 0$ ▷ Her bir s iç durumu için uyuşmazlık sayacını sıfırla
  for $i \leftarrow t\ ,\ t_{max}$ **do** ▷ i değişkeni t'den t_{max}'e kadar döngü
    for each $s : CUR$ **do** ▷ CUR'deki her bir s iç durumu için döngü
      if $\#SV(s) \geq \alpha_{ter}$ **then** ▷ Önerilen değer sayısı α_{ter}'i geçtiyse
        $STORED \leftarrow s$ ▷ s iç durumunu STORED listesine al
        continue ▷ Döngüye sonraki elemandan devam et
      end if
      if $Pr_g(s) = 0.5$ **then**
        $fb_{sugg} \leftarrow null$ ▷ Önerilen değeri 'yok' olarak işaretle
      else
        $fb_{sugg} \leftarrow fb(s)$ ▷ Geri besleme değerini önerilen değere ata
        $\#SV(s) \leftarrow \#SV(s) + 1$ ▷ iç durumun önerilen değer sayacını artır
      end if
      if IsPossibleToDetermine($fb_{i+1}, z_{i+1+\theta_f}$) **then**
        Determine fonksiyonunu çalıştır (Algoritma 2.2)
      else
        Check & Guess fonksiyonunu çalıştır (Algoritma 2.3)
      end if
    end for
    if IsEmpty(NEW) **then** ▷ NEW listesi boş ise
      break ▷ Döngüden çık
    end if
    $CUR \leftarrow NEW$ ▷ NEW'in içeriği CUR'e kopyalanır.
    $NEW \leftarrow Liste < s > ()$ ▷ Liste boşaltılır.
  end for
  return Reduce($STORED$) ▷ STORED listesindeki adayların kökleri
end procedure

6.3.3 İyileştirmenin Performansa Etkisi

1 No'lu iyileştirmenin amacı Tahmin Kapasitesi 0.5'ten büyük olan iç durumları işlemek olduğu için performans Ortalama Tahmin Kapasitesinin 0.5'e yakınsamasıyla doğru orantılı olarak yükselmektedir. Bu nedenle, Ortalama Tahmin Kapasitesi eski formüle göre 1 olan bir tasarım için performans aynı seviyededir. Ayrıntılı performans ölçümleri

Bölüm 7'da verilmiştir. Hızlanma oranı ($C_{hizlanma}$) bu iyileştirme için aşağıdaki formülle hesaplanabilir:

$$r = a/2^s$$
$$C_{hizlanma} = \frac{\alpha_{thr}^{eski}}{\alpha_{thr}^{yeni}} \times r$$

Formüldeki r hızlanma çarpanı, α_{thr}^{eski} eski Tahmin Kapasitesi değerine bağlı olarak hesaplanan eşik değeri, α_{thr}^{yeni} ise yeni Tahmin Kapasitesi formülüne bağlı olarak hesaplanan eşik değeridir. Sonuç olan $C_{hizlanma}$ değeri ise hızlanmayı kat cinsinden ifade eder. Örneğin bir fonksiyon için $r = a/2^s = 786432/1048576 = 0.75$ olsun. Bu fonksiyon için $\alpha_{thr}^{eski} = 32.26$ ve $\alpha_{thr}^{yeni} = 16.09$ eşik değerleri hesaplanmış olsun. Bu algoritmanın İyileştirme No:1 ile elde edeceği hızlanma $C_{hizlanma} = \frac{32.26}{16.09} \times 0.75 \approx 1.5$ kat olarak bulunur. Tablo 7.3 üzerinde örnek geri besleme fonksiyonları için bu değer hesaplanarak sağdaki iki sütunda gerçek ve öngörülen değerler karşılaştırılmıştır.

6.4 İyileştirme No:3

İyileştirme No:3, İDGK algoritmasının işletilme sürecini aşamalara bölüp sonuca daha hızlı gitmeyi amaçlar. $\alpha_{ter} = 1000$ olduğu bir durumu örnek olarak ele alırsak, yeni yöntemle iyileştirme kapsamında bu değer 10'a parçalanıp $\alpha_{ter} = 100$ ve bu değere uygun olarak hesaplanmış olan α_{thr} kullanılarak algoritmanın ilk aşaması işletir. Tüm iç durumları test etmesi sebebiyle ilk aşama en uzun süren aşama olacaktır.

İkinci aşamada $\alpha_{ter} = 200$ olurken α_{thr} bu değere göre yeniden hesaplanır. Takip eden tüm aşamalar için süreç benzer şekilde devam eder. Her aşamada daha az iç durum işletileceğinden algoritmanın tamamlanma süresi buna bağlı olarak git gide daha da kısalır. Aşamalarda α_{ter}, α_{thr}, kalan iç durum sayıları ve tamamlanma süreleriyle ilgili ayrıntılı rapor Bölüm 7'da verilmiştir.

6.4.1 Sözde Kodlar

İyileştirme No:3'te, önceki iyileştirme ve düzeltmeden farklı olrak algoritma sabitleri ve değişkenleri görülmektedir. Bunlar arasında $C_{max\_sieve\_count}$, $C_{alpha\_terminate\_divider}$,

İç Durum Geri Kazanım (İyileştirme No:3)

procedure ISRNo3($\mathbf{S}, z, \epsilon, Pr_g$) ▷ Saldırının ana fonksiyon imzası
  Algoritma Sabitlerini Ayarla
  $C_{max\_sieve\_count} \leftarrow 10$
  $C_{alpha\_terminate\_divider} \leftarrow 10$
  $C_{max\_roots\_to\_stop} \leftarrow 1$
  $C_{alpha\_threshold\_multiplier} \leftarrow 0.75$
  Algoritma Parametrelerini Hesapla
  $\alpha_{ter} \leftarrow \frac{\alpha_{ter}}{C_{alpha\_terminate\_divider}}$
  $\epsilon_{ter} \leftarrow \sqrt{\frac{-ln\epsilon}{2\alpha_{ter}}}$
  $\alpha_{thr} \leftarrow \lfloor \alpha_{ter}(1 - Pr_g + \alpha_{ter}) \rfloor$
  $V_{clock\_offset} \leftarrow 0$
  $hitStates \leftarrow \mathbf{S}$ ▷ Adayları hitStates listesine al
  for $sieveIndex \leftarrow 0,\ C_{max\_sieves}$ **do**
    $V_{global\_alpha\_terminate} \leftarrow \alpha_{ter} \times (sieveIndex + 1)$
    $\epsilon_{ter} \leftarrow \sqrt{\frac{-ln\epsilon}{2.V_{global\_alpha\_terminate}}}$
    $\alpha_{thr} \leftarrow \lfloor V_{global\_alpha\_terminate}(1 - Pr_g + V_{global\_alpha\_terminate}) \rfloor$
    $\alpha_{thr} \leftarrow \alpha_{thr} \times C_{alpha\_threshold\_multiplier}$
    $bufferedStates \leftarrow$**SIEVE**($hitStates$, z, ϵ, Pr_g)
    if $numOfRoots(bufferedStates) > C_{max\_roots\_to\_stop}$ **then**
      $hitStates \leftarrow bufferedStates$ ▷ Sonraki iterasyona devret
      $V_{clock\_offset} \leftarrow V_{clock\_offset} + \alpha_{ter}$ ▷ Offset değerini artır
      continue
    else
      break ▷ Döngüden çık
    end if
    bufferedStates içeriğini ve istatistikleri göster
  end for
end procedure

$C_{max\_roots\_to\_stop}$, $C_{alpha\_threshold\_multiplier}$ ve V_{clock_offset} bulunmaktadır.

Elek yöntemi çalışırken aday iç durumların sayısı bir yandan uyuşmazlık sayaçları α_{thr} değerini geçenlerden dolayı azalırken, öte yandan CheckandGuess algoritması sebebiyle çoğalmaktadır. Yanlış bir yapılandırma algoritmanın hiç bitmemesine sebep olduğu için sigorta amaçlı $C_{max\_sieve\_count}$ sabiti kullanılmıştır. Doğru yapılandırılan bir algoritma için bu sayı sonsuz dahi olsa sorun teşkil etmez ve algoritma yeterli sayıda iterasyondan sonra $C_{max\_roots\_to\_stop}$ sabitinden daha az kök sayısı ihtiva eden bir aday grubu yakaladığında sonlanacaktır.

$C_{alpha\_terminate\_divider}$ ise süreci parçalara ayırmakta kullanılan en temel algoritma sabitidir. Her bir aşamada saat vuruşunun iç durumlar için kaldığı yerden devam etmesi

İç Durum Geri Kazanım (İyileştirme No:3)(Devam)

29: **procedure** SIEVE($\mathbf{S}, z, \epsilon, Pr_g, \alpha_{ter}, V_{clock\_offset}$)
30: $CUR \leftarrow Liste < s > ()$
31: $NEW \leftarrow Liste < s > ()$
32: $CUR \leftarrow \mathbf{S}$
33: **for** $i \leftarrow V_{clock\_offset}, (V_{clock\_offset} + \alpha_{ter})$ **do**
34: **for each** $s : CUR$ **do**
35: **if** $Pr_g(s) = 0.5$ **then**
36: $fb_{sugg} \leftarrow null$
37: $\#MM(s) \leftarrow \#MM(s) + 0.5$
38: **else**
39: $fb(s) \leftarrow fb_{sugg}$
40: **end if**
41: **if** IsPossibleToDetermine($fb_{i+1}, z_{i+1+\theta_f}$) **then**
42: Determine fonksiyonunu çalıştır (Algoritma 2.2)
43: **else**
44: Check & Guess fonksiyonunu çalıştır (Algoritma 2.3)
45: **end if**
46: **end for**
47: **if** IsEmpty(NEW) **then**
48: Döngüden çık
49: **end if**
50: $CUR \leftarrow NEW$
51: $NEW \leftarrow Liste < s > ()$
52: **end for**
53: **return** CUR listesindeki adayların *kökleri(root'ları)*
54: **end procedure**

gerektiği için V_{clock_offset} değişkeni her aşama başlangıcında güncellenecek şekilde ayarlanmıştır.

Her iterasyonda çağrılan **Sieve** isimli alt yordam Orijinal Algoritma'dan bir miktar farklılaştırılmış, V_{clock_offset} değerine göreceli olarak işletilecek şekilde yapılandırılmıştır. Bu değişikliğin yanı sıra iç durumlara ait uyuşmazlık değerleri sıfırlanmamış; aksine, iç durumların ileriki aşamalarda elenmelerini sağlamak için kaldıkları yerden saymaya devam etmeleri sağlanmıştır.

Ayrıntılarına yer verilmeyen numOfRoots alt yordamı ise iç durum listesini parametre olarak alıp farklı kök sayısını döndürmektedir. Algoritmanın sonlanma şartı ise bu alt yordamdan gelen kök sayısının $C_{max\_roots\_to\_stop}$ sabitinden küçük olmasıyla gerçekleşir.

6.5 Geliştirilmiş Algoritmanın Nihai Tasarımı

Bu bölümde bahsedilen iyileştirmelerin bir araya getirilmesiyle oluşturulan İç Durum Geri Kazanım algoritmasıdır. İyileştirme ve düzeltmeler birbirinden bağımsız noktaları etkilediği için hızlanma üç değişikliğin ayrı ayrı oluşturduğu performans artırımından fazladır. Performans ayrıntıları Bölüm 7'de görülebilir.

6.5.1 Sözde Kodlar

İç Durum Geri Kazanım (Nihai Tasarım 1/3)

procedure ISRFINAL($\mathbf{S}, z, \epsilon, Pr_g$)
  $C_{max\_sieve\_count} \leftarrow 30$
  $C_{alpha\_terminate\_divider} \leftarrow 10$
  $C_{max\_roots\_to\_stop} \leftarrow 1$
  $C_{alpha\_threshold\_multiplier} \leftarrow 0.75$
  $\alpha_{ter} \leftarrow \frac{\alpha_{ter}}{C_{alpha\_terminate\_divider}}$
  $\epsilon_{ter} \leftarrow \sqrt{\frac{-ln\epsilon}{2\alpha_{ter}}}$
  $\alpha_{thr} \leftarrow \lfloor \alpha_{ter}(1 - Pr_g + \alpha_{ter}) \rfloor$
  $V_{clock\_offset} \leftarrow 0$
  $hitStates \leftarrow \mathbf{S}$
  for $sieveIndex \leftarrow 0,\ C_{max\_sieves}$ **do**
    $V_{global\_alpha\_terminate} \leftarrow \alpha_{ter} \times (sieveIndex + 1)$
    $\epsilon_{ter} \leftarrow \sqrt{\frac{-ln\epsilon}{2.V_{global\_alpha\_terminate}}}$
    $\alpha_{thr} \leftarrow \lfloor V_{global\_alpha\_terminate}(1 - Pr_g + V_{global\_alpha\_terminate}) \rfloor$
    $\alpha_{thr} \leftarrow \alpha_{thr} \times C_{alpha\_threshold\_multiplier}$
    $bufferedStates \leftarrow$**sieve**($hitStates$,z,$\epsilon$,$Pr_g$,$V_{global\_alpha\_terminate}$,$V_{clock\_offset}$)
    if $numOfRoots(bufferedStates) > C_{max\_roots\_to\_stop}$ **then**
      $hitStatesCumulative \leftarrow bufferedStates$
      $V_{clock\_offset} \leftarrow V_{clock\_offset} + \alpha_{ter}$
      continue
    else
      break
    end if
    bufferedStates içeriğini ve istatistikleri göster
  end for
end procedure

Geliştirilmiş İç Durum Geri Kazanım (Nihai Tasarım 2/3)

procedure SIEVE($\mathbf{S}, z, \epsilon, t_{max}, Pr_g, V_{global\_alpha\_terminate}, V_{clock\_offset}$)
  $CUR \leftarrow Liste < s > ()$
  $NEW \leftarrow Liste < s > ()$
  $STORED \leftarrow Liste < s > ()$
  $CUR \leftarrow \mathbf{S}$
  for $i \leftarrow t + V_{clock\_offset}$, $t_{max} + V_{clock\_offset}$ **do**
    for each $s : CUR$ **do**
      if $\#SV(s) \geq V_{global\_alpha\_terminate}$ **then**
        $STORED \leftarrow s$
        continue
      end if
      if $Pr_g(s) = 0.5$ **then**
        $fb_{sugg} \leftarrow null$
      else
        $fb_{sugg} \leftarrow fb(s)$
        $\#SV(s) \leftarrow \#SV(s) + 1$
      end if
      if IsPossibleToDetermine($fb_{i+1}, z_{i+1+\theta_f}$) **then**
        Determine fonksiyonunu çalıştır (Algoritma 2.2)
      else
        Check & Guess fonksiyonunu çalıştır (Algoritma 2.3)
      end if
    end for
    if IsEmpty(NEW) **then**
      break
    end if
    $CUR \leftarrow NEW$
    $NEW \leftarrow Liste < s > ()$
  end for
  return Reduce($STORED$)
end procedure

Geliştirilmiş İç Durum Geri Kazanım (Nihai Tasarım 3/3)

procedure DETERMINE($\mathbf{S}, z, \epsilon, Pr_g$)
  if $f_{sugg} \neq null$ **then**
    if $fb_{sugg} \neq fb_{det}$ **then**
      $\#MM(s) \leftarrow \#MM(s) + 1$
    end if
  end if
  if $\#MM(s) \leq \alpha_{thr}$ **then**
    Geri besleme değerini fb_{det} karşılık gelen z çıktı biti üzerinden hesapla
    s iç durumunu fb_{det} kullanarak güncelle
    Güncellenmiş s iç durumunu, $\#MM(s)$ değerini ve *kök* değerini NEW'e ekle
  end if
end procedure

Bölüm 7

Geliştirilmiş Algoritmanın Performans Analizi

7.1 Ön Bilgiler

Önceki bölümde tanımı yapılan ve ayrıntıları açıklanan düzeltme ve iki iyileştirmeye ait performans analizleri bu bölümde anlatılmıştır.

Her bir geliştirmenin Orijinal Algoritmayla ayrı ayrı kıyaslanmasının yanı sıra iki iyileştirmenin bütünleşik hâli olan Nihai Tasarım ile de kıyaslama yapılmıştır. Süre bazlı karşılaştırmalar test yapılan sisteme göre farklılık göstereceğinden kıyaslamalar donanımdan bağımsız bir ölçü birimi olması için Saat Vuruşu/İç Durum (Clocks/State) olarak da gerçekleştirilmiştir.

7.1.1 Benzetimin Bilgisayar Ortamında Gerçeklenmesi

LFSR, NFSR ve diğer kayan anahtar üreteci bileşenleri bilgisayar ortamında C++ programlama dili kullanılarak gerçeklenmiştir. Farklı saklayıcı ve anahtar boylarıyla, farklı geri besleme ve çıktı fonksiyonlarıyla test edilmesini kolaylaştırmak için Nesne Yönelimli Programlama tekniklerinden faydalanılmıştır. Herhangi bir kriptografik kütüphane veya yardımcı kod kullanılmamıştır.

Simülasyon programı Qt 5.12.1 interaktif geliştirme ortamı kullanılarak kodlanmış ve MinGW 7.3.0 C++ derleyicisiyle 64-Bit mimariye uyumlu çalıştırılabilir dosya olarak

derlenmiştir. Herhangi bir paralel işlem, çok iş parçacıklı mimari vb. hızlandırmalar kullanılmamıştır. Ölçümler yapılmadan önce test bilgisayarının ağ bağlantısı koparılmış ve tüm yan uygulamalar kapatılmıştır. Sonuçların sağlığı açısından her bir test aynı şartlarda gerçekleştirilmiştir. Test bilgisayarında kurulu olan işletim sistemi Microsoft Windows® 10 Pro (64-Bit Türkçe) olup testlerden önce tüm güncellemeleri yapılmıştır.

7.1.2 Test Sistemi

Performans ölçümlerinin yapıldığı bilgisayara ait özellikler aşağıda verilmiştir.

Tablo 7.1: Test bilgisayarına ait donanım özellikleri

Bileşen	**Marka**	**Model**	**Açıklama**
Anakart	ASUS	M5A97 LE R2.0	AMD 970 Çipsetli Anakart
İşlemci	AMD	FX-8350	8 Fiziksel Çekirdek @ 4.2GHz
Bellek	Kingston	9905403-442.A00LF	Dual Channel DDR3-1333
Ekran Kartı	MSI	GeForce GTX 1080	8 GB GDDR5X
Güç Kaynağı	Cooler Master	Silent Pro Gold	550 Watt
Sabit Disk	Western Digital	WD1003FZEX	1 TB, 7200 rpm
İşletim Sistemi	Microsoft	Windows 10 Pro 64-Bit	Sürüm 1803 Yapı 17134.765

7.1.3 Test Senaryosu ve Test Fonksiyonları

Herhangi bir İç Durum Geri Kazanımı algoritmasının test edilmesi için önceden bilinen; ama algoritmanın işleyişine dahil edilmeyen bir İlk İç Durum belirlenir. Saldırının yapılacağı şifreleyiciden bu ilk iç durum kullanılarak elde edilen kayan anahtar çıktısı İDGK girdi olarak verilir. Algoritma sonlandığında önceden bilinen bu iç duruma ulaşılıp ulaşılmadığı kontrol edilir. İç durum adaylarının oluşturulması ve kayan anahtar çıktısı alınması işlemleri ilklendirme safhası olarak adlandırılmıştır. Toplam sürelerin ölçümünde bu safha ihmal edilmiştir.

Testlerimizde, yine kendi tasarımımız olan 20-Bit'lik LFSR saklayıcıya sahip, 256-bit anahtar uzunluğu olan bir dizi şifreleyici benzetimi kullandık. Bu benzetim uygulaması

Kara ve Esgin'in İDGK algoritması da dahil olmak üzere aşağıdaki kiplerde çalışabilmektedir:

- Orijinal (KE) + Düzeltme
- Yalnız İyileştirme No:1
- Yalnız İyileştirme No:3
- İyileştirme No:1 ve 3 (Nihai Tasarım)

İyileştirme No:1'de Tahmin Kapasitesi formülü farklı olduğu için, eski formüle kıyasla daha büyük sonuçlar çıkartabilmekteydi. Eski ve yeni formüller arasındaki sonuç farkının 0'dan 0,438'e kadar değiştiği 19 adet geri besleme fonksiyonu oluşturuldu. Bu fonksiyonların her biri için yukarıdaki çalışma kiplerinin tamamı hem 0.01 hata oranı hem de 0.001 hata oranı için koşturuldu. Sonuçlar fonksiyona özel ayrı sayfada ve tüm fonksiyonların derlendiği genel bir tabloda derlendi. Bunun üzerinden fonksiyon bazlı hız artırımı, Tahmin Kapasitesi farkı bazlı hız artırımı ve iyileştirmelerin çeşitli kıyaslamalarını içeren grafikler oluşturuldu. Sonuç tabloları ilerleyen bölümde ayrıntılı şekilde açıklanacaktır.

Performans testlerinde kullanılan geri besleme fonksiyonları aşağıdaki gibidir:

$$f_1 = s_0 \oplus s_2 \oplus (s_{17}.s_{18}.k_5)$$

$$f_3 = s_0 \oplus s_2 \oplus (s_5.s_{10}) \oplus (s_{17}.k_5)$$

$$f_5 = s_0 \oplus s_2 \oplus (s_5.s_{10}) \oplus (s_{17}.s_{18}.k_5) \oplus (s_{19}.k_{24})$$

$$f_6 = s_{10} \oplus s_{12} \oplus (s_5.k_0) \oplus (s_3.k_{10}.k_{11})$$

$$f_7 = s_{11} \oplus s_{12} \oplus (s_5.k_0) \oplus (s_3.s_{10}.k_{10}.k_{11})$$

$$f_8 = s_{11} \oplus s_{12} \oplus (s_5.k_0.k_3.k_{10}) \oplus (s_4.k_1)$$

$$f_{10} = s_0 \oplus s_2 \oplus (s_{15}.s_{16}.s_{17}.s_{18}.k_5.k_{10}.k_{15}.k_{20}.k_{25}.k_{30})$$

$$f_{11} = s_0 \oplus s_2 \oplus (s_{17}.s_{18}.k_5.k_{10}.k_{15}.k_{20})$$

$$f_{12} = s_0 \oplus s_2 \oplus (s_{17}.k_5.k_{10}.k_{15}.k_{20})$$

$$f_{13} = s_5 \oplus s_{10} \oplus (s_{11}.k_5) \oplus (s_{15}.k_{10}) \oplus (s_{17}.k_{15})$$

$$f_{14} = s_5 \oplus s_{10} \oplus (s_{11}.k_5) \oplus (s_{15}.k_{10}) \oplus (s_{17}.k_{15}.k_{16})$$

$$f_{15} = s_5 \oplus s_{10} \oplus (s_{11}.k_5) \oplus (s_{15}.k_{10}) \oplus (s_{17}.s_{19}.k_{15}.k_{16})$$

$$f_{16} = s_5 \oplus s_{10} \oplus (s_{11}.k_5) \oplus (s_{17}.s_{19}.k_{15}.k_{16})$$

$$f_{17} = s_5 \oplus s_{10} \oplus (s_{11}.k_5.k_{10}) \oplus (s_{17}.s_{19}.k_{15}.k_{16})$$

$$f_{18} = s_5 \oplus s_{10} \oplus (s_{11}.k_5.k_{10}.k_{11}) \oplus (s_{17}.s_{19}.k_{15}.k_{16}.k_{17})$$

$$f_{20} = s_5 \oplus s_{10} \oplus (s_{11}.s_{12}.k_5) \oplus (s_{17}.s_{19}.k_{15}.k_{16})$$

$$f_{21} = s_5 \oplus (s_{10}.k_7) \oplus (s_{11}.s_{12}.k_5.k_6) \oplus (s_{17}.s_{19}.k_{15}.k_{16})$$

$$f_{22} = s_5 \oplus (s_{10}.s_{11}.k_{107}) \oplus (s_{12}.s_{13}.k_{105}.k_{106}) \oplus (s_{17}.s_{19}.k_{115}.k_{116})$$

$$f_{sprout} = k_t.(s_3 \oplus s_9 \oplus s_{19})$$

Çıktı fonksiyonu f_1'den f_{22}'ye kadarki geri besleme fonksiyonları için aşağıdaki gibidir:

$$z_t = s_0^t \oplus s_1^t \oplus s_3^t \oplus s_8^t \oplus s_{11}^t \oplus s_{13}^t.s_{19}^t$$

Sprout örneğinin gerçek çıktı fonksiyonuna yakın olması için aşağıdaki çıktı fonksiyonu kullanılmıştır:

$$z_t = s_0^t \oplus (s_5^t.s_{19}^t) \oplus s_1^t \oplus s_3^t \oplus s_9^t \oplus s_{11}^t \oplus s_{13}^t \oplus s_{17}^t$$

7.1.4 Performans Metrikleri

Bu kısımda performans kıyaslamalarında kullanılan sütunların okunmasını kolaylaştıracak açıklamalar verilmiştir. Her bir kriter tanımlanarak bir örnekle açıklanmıştır.

Ortalama Hız (Saat Vuruşu/İç Durum): İDGK algoritmasının 2^{20}'lik iç durum uzayında her bir iç durum için ortalama kaç saat vuruşu işletildiğini gösterir. Bu metrik ne kadar düşükse İDGK algoritması o kadar kısa sürer. Örneğin 400 c/s hızındaki işletilmiş olan bir İDGK oturumu, 4000 c/s hızında işletilmiş olan İDGK oturumundan 10 kat daha hızlıdır. Bu metrik İDGK algoritması aynı kalsa dahi geri besleme fonksiyonuna bağlı olarak değişir.

Kat ve Yüzde Bazlı Hızlanma: Performans ölçümlerinde kat ve yüzde bazlı hızlanma bilgileri verilmiş olup, bu hızlanma Orijinal (KE) algoritmaya kıyasla olan hızlanmayı gösterir. Örneğin 1,04 kat hızlanma, İyileştirilmiş algoritmada Orijinal algoritmaya kıyasla yüzde 4 hızlanma gözlemlendiği anlamına gelir.

Toplam Saat Vuruşu: İDGK algoritması işletilirken simülatördeki LFSR'ın kaç defa kaydırma işlemine tabi tutulduğunu gösteren metriktir. Bu değer toplam iç durum sayısı olan 2^{20} 'ye bölündüğünde Ortalama Hız bulunur.

Yineleme Sayısı: İDGK algoritmasının saldırı süresi boyunca kaç defa tekrarlandığını gösterir. İyileştirme No:3'te çok safhalı bir işlem söz konusu olduğu için yineleme sayısı 1'den fazla olmakta ve hata oranı yineleme sayısına bağlı olarak hesaplanmaktadır. Örneğin 3 kez yinelenen bir İDGK algoritmasında tek iterasyona ait hata oranı başta $\epsilon = 0.001$ olarak ayarlandıysa başarım oranı algoritma bitiminde $0.999^3 \approx 0,997$ ve gerçek hata oranı $\epsilon_{efektif} \approx 0.003$ olacaktır.

Toplam Süre: İDGK algoritmasının, ilklendirme sürecinin sonundan itibaren, algoritma bitiminde tespit edilen iç durumların sergilenmesine kadar geçen süreyi gösteren metriktir. Ölçümler test sistemine bağlıdır. Dolayısıyla, simülatör daha güçlü bir bilgisayar sisteminde test edildiğinde daha iyi sonuçlar çıkacaktır. Toplam süre tüm yinelemeler dahil toplamı gösterir.

7.2 Test Sonuçları

Test sisteminde 19 adet geri besleme fonksiyonu $\epsilon = 0.001$ ve $\epsilon = 0.01$ hata oranları için ayrı ayrı tablo oluşturulmuştur (Bkz: Tablo 7.2). Tablolar kalın çizgilerle üç kısma ayrılmıştır. Soldaki kısım geri besleme fonksiyonunun karakteristiğini gösteren bilgilendirme bölümüdür. Orta kısımda Orijinal (KE) algoritmanın hız değeri, iyileştirmelerin ayrı ayrı performansları ve birleştirildiğinde ortaya çıkan Nihai Algoritma hızı *Saat Vuruşu/İç Durum(Clocks/State)* cinsinden verilmiştir. Sağ kısımda ise iyileştirmelerin Orijinal algoritmaya göre kıyaslandığında kaç kat hızlandığını gösteren sayılar verilmiştir.

TABLO 7.2: 19 adet geri besleme fonksiyonuna ait hata oranı $\epsilon = 0.01$ ve $\epsilon = 0.001$ için test ölçüm sonuçları

Geri Besleme Fonksiyonu	Pr_g			α_{ter}		α_{thr}		Ortalama Hız (Saat Vuruşu/İç Durum) Hata: 0.01 (Düşük daha iyi)						Hızlanma (Yüksek daha iyi)			
	Eski	Yeni	Fark	Eski	Yeni	Eski	Yeni	Orijinal (0.01)	Orijinal (Adil Hata 1)	Orijinal (Adil Hata 2)	No:1	No:3	Nihai (1+3)	No:1 (Kat)	No:3 (Kat)	Nihai (Kat)	Nihai (%)
F13	0,5625	1,0000	0,438	4126	64	1902,60	12,14	4079,46	4079,46	3564,82	186,49	314,80	57,04	21,88	12,96	62,50	6149,73
F14	0,5938	0,8750	0,281	1834	114	810,05	30,45	1443,94	1443,94	1440,67	220,37	142,45	43,42	6,55	10,14	33,18	3217,76
F15	0,6094	0,9375	0,328	1347	84	581,86	19,16	1181,31	1082,82	1060,44	150,91	104,79	31,23	7,83	10,33	33,95	3295,19
F6	0,6875	0,8750	0,188	458	114	175,60	30,45	357,32	330,56	330,56	122,90	37,95	21,77	2,91	8,71	15,19	1418,70
F5	0,6875	1,0000	0,313	458	64	175,60	12,14	343,37	317,06	304,42	68,17	37,82	18,25	5,04	8,38	16,68	1567,73
F21	0,6914	0,8828	0,191	440	110	167,61	28,81	340,04	311,14	311,14	105,15	36,08	18,65	3,23	8,62	16,68	1568,08
F16	0,7188	0,9375	0,219	336	84	122,31	19,16	234,14	218,61	218,61	69,87	28,09	15,01	3,35	7,78	14,56	1356,11
F7	0,7188	0,9375	0,219	336	84	122,31	19,16	237,50	221,73	212,17	68,24	28,04	15,39	3,48	7,91	13,79	1278,58
F8	0,7188	0,9375	0,219	336	84	122,31	19,16	243,66	224,44	212,52	68,97	27,82	15,25	3,53	8,07	13,94	1294,03
SPROUT	0,7500	1,0000	0,250	257	64	88,58	12,14	122,51	119,88	117,77	51,39	22,25	15,36	2,38	5,39	7,67	666,78
F3	0,7500	1,0000	0,250	257	64	88,58	12,14	176,61	162,17	158,40	48,56	22,08	15,26	3,64	7,34	10,38	938,19
F22	0,7871	0,8828	0,096	195	110	62,70	28,81	125,19	110,65	110,65	74,89	17,31	13,27	1,67	6,39	8,34	734,07
F20	0,8281	0,9375	0,109	149	84	44,13	19,16	86,75	76,65	74,08	48,66	13,29	10,68	1,78	5,77	6,94	593,76
F1	0,8750	1,0000	0,125	114	64	30,45	12,14	61,60	53,68	51,69	34,21	11,34	10,70	1,80	4,73	4,83	383,00
F17	0,9375	0,9375	0,000	84	84	19,16	19,16	39,90	32,02	32,02	39,68	7,99	7,99	1,01	4,01	4,01	300,55
F18	0,9688	0,9688	0,000	73	73	15,25	15,25	32,04	25,95	23,99	32,11	7,99	8,00	1,00	3,25	3,00	200,02
F12	0,9688	0,9688	0,000	73	73	15,25	15,25	31,87	23,91	23,91	31,84	10,00	10,01	1,00	2,39	2,39	139,01
F11	0,9844	0,9844	0,000	68	68	13,58	13,58	27,88	20,03	20,03	27,75	8,00	8,00	1,00	2,50	2,50	150,20
F10	0,9990	0,9990	0,000	64	64	12,20	12,20	25,96	18,00	18,00	25,96	8,00	8,00	1,00	2,25	2,25	124,95

Geri Besleme Fonksiyonu	Pr_g			α_{ter}		α_{thr}		Ortalama Hız (Saat Vuruşu/İç Durum) Hata: 0.001 (Düşük daha iyi)						Hızlanma (Yüksek daha iyi)			
	Eski	Yeni	Fark	Eski	Yeni	Eski	Yeni	Orijinal (0.001)	Orijinal (Adil Hata 1)	Orijinal (Adil Hata 2)	No:1	No:3	Nihai (1+3)	No:1 (Kat)	No:3 (Kat)	Nihai (Kat)	Nihai (%)
F13	0,5625	1,0000	0,438	4857	75	2254,46	16,09	5354,85	5354,85	3994,84	334,55	387,50	59,99	16,01	13,82	66,59	6558,86
F14	0,5938	0,8750	0,281	2159	134	963,45	38,26	1690,97	1690,97	1719,26	272,94	171,20	50,59	6,20	9,88	33,99	3298,60
F15	0,6094	0,9375	0,328	1586	99	693,54	24,68	1515,37	1376,95	1262,52	192,69	127,79	37,64	7,86	10,78	33,55	3254,59
F6	0,6875	0,8750	0,188	539	134	211,58	38,26	424,25	405,17	405,17	157,53	45,95	25,50	2,69	8,82	15,89	1488,73
F5	0,6875	1,0000	0,313	539	75	211,58	16,09	410,16	393,38	393,38	89,13	45,75	20,14	4,60	8,60	19,53	1853,20
F21	0,6914	0,8828	0,191	517	129	201,80	36,23	442,45	380,07	380,07	136,90	45,00	24,65	3,23	8,45	15,42	1441,88
F7	0,7188	0,9375	0,219	396	99	148,36	24,68	288,24	276,62	255,51	88,15	35,01	18,58	3,27	7,90	13,76	1275,56
F8	0,7188	0,9375	0,219	396	99	148,36	24,68	261,03	276,89	263,03	88,95	32,62	18,45	2,93	8,49	14,25	1325,40
F16	0,7188	0,9375	0,219	396	99	148,36	24,68	279,96	268,63	262,60	88,75	35,05	18,02	3,15	7,67	14,57	1357,24
SPROUT	0,7500	1,0000	0,250	303	76	108,10	16,39	131,12	128,81	127,11	65,73	28,24	15,40	1,99	4,56	8,26	725,53
F3	0,7500	1,0000	0,250	303	75	108,10	16,09	217,64	205,09	201,99	64,84	28,03	15,31	3,36	7,32	13,20	1219,70
F22	0,7871	0,8828	0,096	230	129	77,15	36,23	178,14	141,36	141,36	96,87	21,31	18,21	1,84	6,63	7,76	676,12
F20	0,8281	0,9375	0,109	176	99	54,91	24,68	105,02	96,26	97,12	48,68	17,22	13,23	2,16	5,59	7,34	633,86
F1	0,8750	1,0000	0,125	134	75	38,26	16,09	82,60	70,64	70,64	45,22	14,32	10,72	1,83	4,93	6,59	558,78
F17	0,9375	0,9375	0,000	99	99	24,68	24,68	49,75	43,86	43,86	49,02	9,99	9,99	1,02	4,39	4,39	338,95
F12	0,9688	0,9688	0,000	86	86	19,92	19,92	44,31	35,70	33,74	39,71	10,00	10,01	1,12	3,57	3,37	237,18
F18	0,9688	0,9688	0,000	86	86	19,92	19,92	40,06	36,19	34,12	40,11	9,99	9,99	1,00	3,62	3,41	241,41
F11	0,9844	0,9844	0,000	80	80	17,87	17,87	41,47	29,85	29,85	35,89	10,01	10,00	1,16	2,98	2,98	198,33
F10	0,9990	0,9990	0,000	76	76	16,28	16,28	39,49	27,88	27,88	33,82	8,00	8,00	1,17	3,48	3,48	248,35

TABLO 7.3: 19 adet geri besleme fonksiyonuna ait İyileştirme No:1 için hata oranı $\epsilon = 0.01$ ve $\epsilon = 0.001$ iken beklenen hızlanma oranı sonuçları

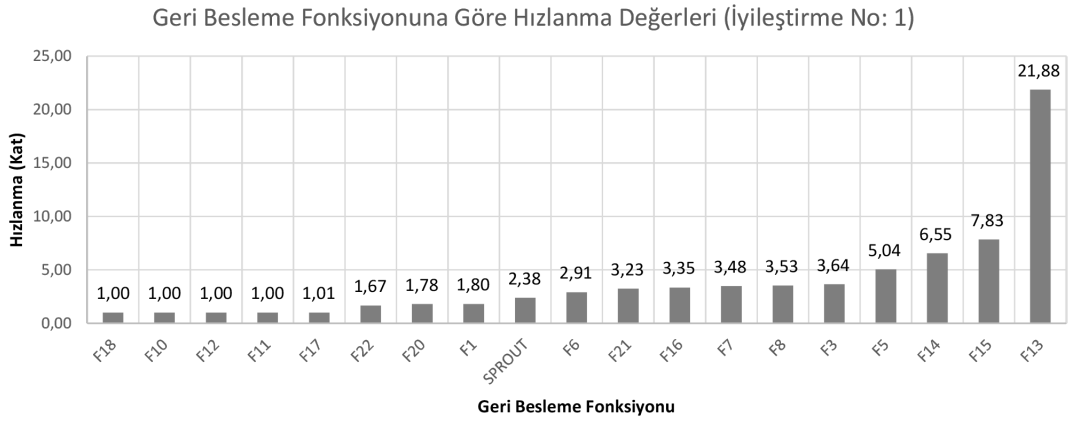
Geri Besleme Fonksiyonu	Pr_g		α_{ter}		Beklenen Hızlanma (No:1)			α_{thr}		Ortalama Hız (Saat Vuruşu/İç Durum) Hata: 0.01 (Düşük daha iyi)		Hızlanma (Yüksek daha iyi)		
	Eski	Yeni	Eski	Yeni	a	r	$C_{\text{hızlanma}}$	Eski	Yeni	Orijinal (0.01)	No:1	No:1 (Kat)	$C_{\text{hızlanma}}$	Sapma(%)
F13	0,5625	1,0000	4126	64	131072	0,125	19,59	1902,60	12,14	4079,46	186,49	21,88	19,59	10,44
F14	0,5938	0,8750	1834	114	262144	0,25	6,65	810,05	30,45	1443,94	220,37	6,55	6,65	1,50
F15	0,6094	0,9375	1347	84	262144	0,25	7,59	581,86	19,16	1181,31	150,91	7,83	7,59	3,00
F5	0,6875	1,0000	458	64	393216	0,375	5,42	175,60	12,14	343,37	68,17	5,04	5,42	7,69
F6	0,6875	0,8750	458	114	524288	0,5	2,88	175,60	30,45	357,32	122,90	2,91	2,88	0,83
F21	0,6914	0,8828	440	110	524288	0,5	2,91	167,61	28,81	340,04	105,15	3,23	2,91	10,03
F16	0,7188	0,9375	336	84	524288	0,5	3,19	122,31	19,16	234,14	69,87	3,35	3,19	4,74
F7	0,7188	0,9375	336	84	524288	0,5	3,19	122,31	19,16	237,50	68,24	3,48	3,19	8,27
F8	0,7188	0,9375	336	84	524288	0,5	3,19	122,31	19,16	243,66	68,97	3,53	3,19	9,64
SPROUT	0,7500	1,0000	257	64	524288	0,5	3,65	88,58	12,14	122,51	51,39	2,38	3,65	53,03
F3	0,7500	1,0000	257	64	524288	0,5	3,65	88,58	12,14	176,61	48,56	3,64	3,65	0,31
F22	0,7871	0,8828	195	110	786432	0,75	1,63	62,70	28,81	125,19	74,89	1,67	1,63	2,34
F20	0,8281	0,9375	149	84	786432	0,75	1,73	44,13	19,16	86,75	48,66	1,78	1,73	3,08
F1	0,8750	1,0000	114	64	786432	0,75	1,88	30,45	12,14	61,60	34,21	1,80	1,88	4,48
F17	0,9375	0,9375	84	84	1048576	1	1,00	19,16	19,16	39,90	39,68	1,01	1,00	0,54
F12	0,9688	0,9688	73	73	1048576	1	1,00	15,25	15,25	31,87	31,84	1,00	1,00	0,08
F18	0,9688	0,9688	73	73	1048576	1	1,00	15,25	15,25	32,04	32,11	1,00	1,00	0,22
F11	0,9844	0,9844	68	68	1048576	1	1,00	13,58	13,58	27,88	27,75	1,00	1,00	0,48
F10	0,9990	0,9990	64	64	1048576	1	1,00	12,20	12,20	25,96	25,96	1,00	1,00	0,00

Geri Besleme Fonksiyonu	Pr_g		α_{ter}		Beklenen Hızlanma (No:1)			α_{thr}		Ortalama Hız (Saat Vuruşu/İç Durum) Hata: 0.001 (Düşük daha iyi)		Hızlanma (Yüksek daha iyi)		
	Eski	Yeni	Eski	Yeni	a	r	$C_{\text{hızlanma}}$	Eski	Yeni	Orijinal (0.001)	No:1	No:1 (Kat)	$C_{\text{hızlanma}}$	Sapma(%)
F1	0,8750	1,0000	134	75	786432	0,75	1,78	38,26	16,09	82,60	45,22	1,83	1,78	2,38
F10	0,9990	0,9990	76	76	1048576	1	1,00	16,28	16,28	39,49	33,82	1,17	1,00	14,36
F11	0,9844	0,9844	80	80	1048576	1	1,00	17,87	17,87	41,47	35,89	1,16	1,00	13,45
F12	0,9688	0,9688	86	86	1048576	1	1,00	19,92	19,92	44,31	39,71	1,12	1,00	10,39
F13	0,5625	1,0000	4857	75	131072	0,125	17,51	2254,46	16,09	5354,85	334,55	16,01	17,51	9,39
F14	0,5938	0,8750	2159	134	262144	0,25	6,29	963,45	38,26	1690,97	272,94	6,20	6,29	1,61
F15	0,6094	0,9375	1586	99	262144	0,25	7,03	693,54	24,68	1515,37	192,69	7,86	7,03	10,66
F16	0,7188	0,9375	396	99	524288	0,5	3,01	148,36	24,68	279,96	88,75	3,15	3,01	4,71
F17	0,9375	0,9375	99	99	1048576	1	1,00	24,68	24,68	49,75	49,02	1,02	1,00	1,48
F18	0,9688	0,9688	86	86	1048576	1	1,00	19,92	19,92	40,06	40,11	1,00	1,00	0,13
F20	0,8281	0,9375	176	99	786432	0,75	1,67	54,91	24,68	105,02	48,68	2,16	1,67	22,66
F21	0,6914	0,8828	517	129	524288	0,5	2,79	201,80	36,23	442,45	136,90	3,23	2,79	13,82
F22	0,7871	0,8828	230	129	786432	0,75	1,60	77,15	36,23	178,14	96,87	1,84	1,60	13,14
SPROUT	0,7500	1,0000	303	76	524288	0,5	3,36	108,10	16,09	131,12	65,73	1,99	3,36	68,39
F3	0,7500	1,0000	303	76	524288	0,5	3,36	108,10	16,09	217,64	64,84	3,36	3,36	0,05
F5	0,6875	1,0000	539	75	393216	0,375	4,93	211,58	16,09	410,16	89,13	4,60	4,93	7,13
F6	0,6875	0,8750	539	134	524288	0,5	2,76	211,58	38,26	424,25	157,53	2,69	2,76	2,66
F7	0,7188	0,9375	396	99	524288	0,5	3,01	148,36	24,68	288,24	88,15	3,27	3,01	8,08
F8	0,7188	0,9375	396	99	524288	0,5	3,01	148,36	24,68	261,03	88,95	2,93	3,01	2,43

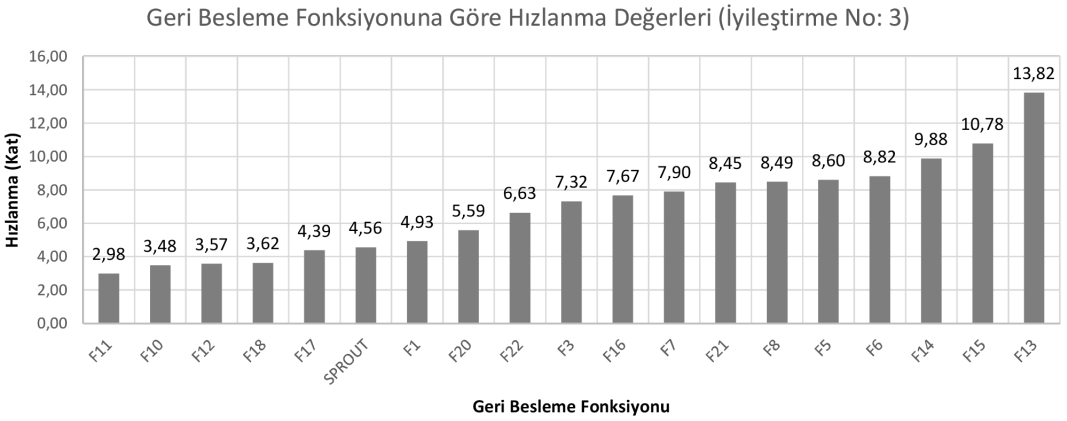
7.2.1 Grafiklerin Yorumlanması

Tablodaki verilere dayanarak Şekil 7.1, 7.2, 7.3, 7.4, 7.7, ve 7.8, çubuk grafikleri hazırlanmıştır. Şekil 7.1, 7.2 ve 7.3, 7.4 için grafiklerde yatay eksen 18 geri besleme fonksiyonunu, dikey eksen ise bu fonksiyonda İyileştirme No:1 ve İyileştirme No:3'ün orijinal algoritmayı kaç kat hızlandırdığını ifade eden sayıyı temsil etmektedir. Yeni ve eski tahmin kapasite değeri farkı en büyük olan f_{13} diğer fonksiyonlara göre en büyük hızlanma oranı yakalamıştır. Tahmin Kapasitesi farkı 0 olan f_{10}, f_{11}, f_{12}, f_{17} ve f_{18} fonksiyonları ise İyileştirme No:1 için hızlanma göstermezken, İyileştirme No:3 için α_{ter} değerleriyle doğru orantılı şekilde hız artışı göstermiştir.

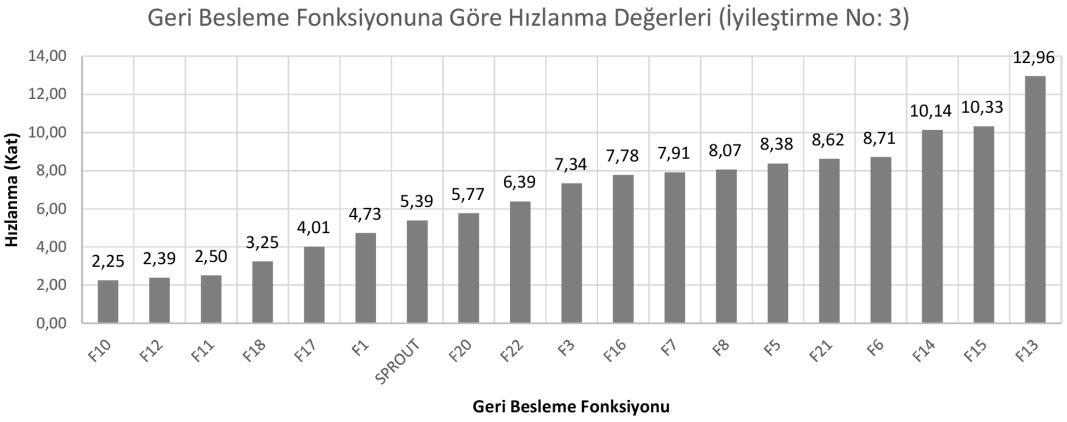
Şekil 7.1: Hata oranı $\epsilon = 0.001$ için İyileştirme No:1 hızlanma oranları



Şekil 7.2: Hata oranı $\epsilon = 0.01$ için İyileştirme No:1 hızlanma oranları



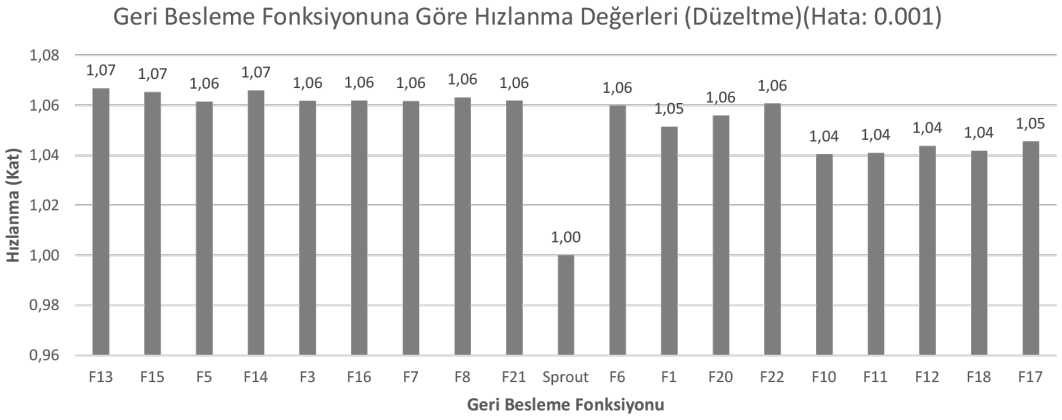
Şekil 7.3: Hata oranı $\epsilon = 0.001$ için İyileştirme No:3 hızlanma oranları



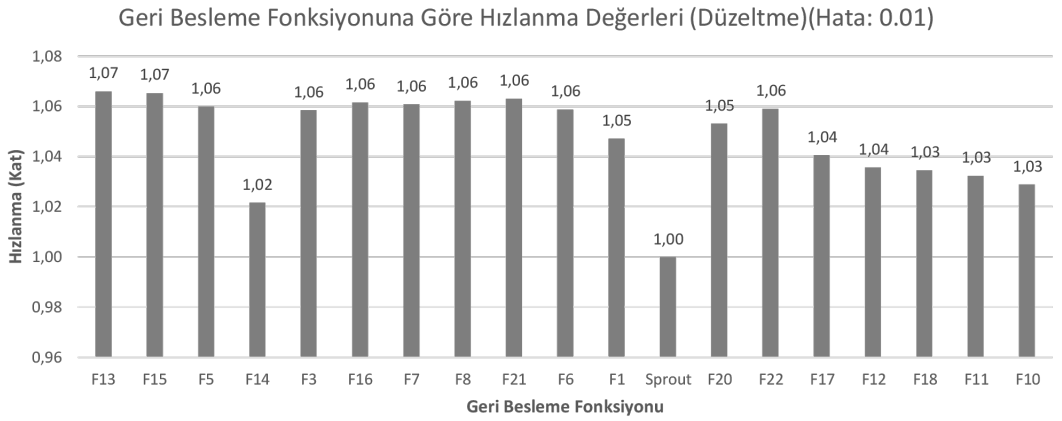
Şekil 7.4: Hata oranı $\epsilon = 0.01$ için İyileştirme No:3 hızlanma oranları

Tablo 7.4: Saldırı İyileştirme No:3 açık vaziyette f_{11} geri besleme fonksiyonu için çalışırken her bir elek iterasyonu için kalan aday iç durum sayısı (Hata:$\epsilon = 0.001$)

Elek İndeksi	**Kalan Aday Sayısı**	**Kök Sayısı**	**Süre(ms)**
0	685732	393533	29150
1	46893	33361	9224
2	1093	848	475
3	27	20	15
4	4	2	≈ 0
5	2	1	≈ 0

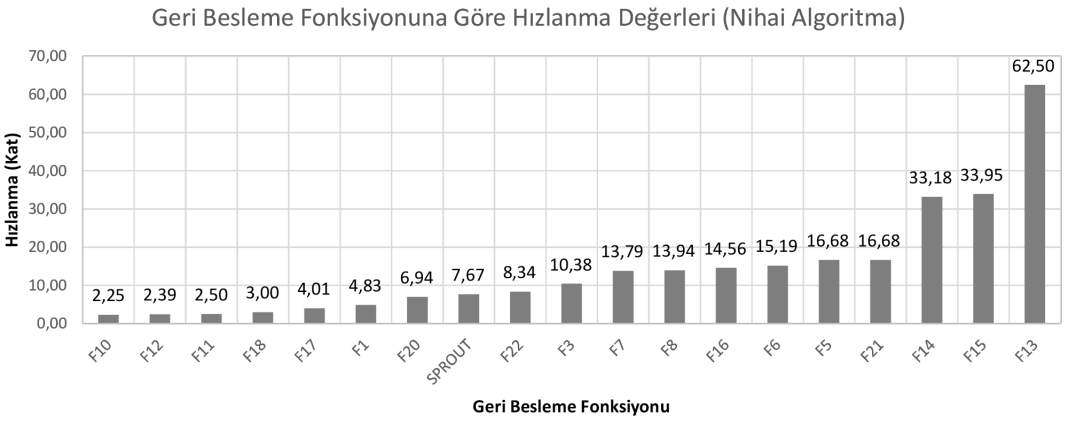


Şekil 7.5: Hata oranı $\epsilon = 0.001$ için düzeltilmiş algoritma hızlanma oranları



Şekil 7.6: Hata oranı $\epsilon = 0.01$ için düzeltilmiş algoritma hızlanma oranları

Şekil 7.7: Hata oranı $\epsilon = 0.001$ için Nihai Algoritma hızlanma oranları



Şekil 7.8: Hata oranı $\epsilon = 0.01$ için Nihai Algoritma hızlanma oranları

Bölüm 8

Sonuç

8.1 Yeni Algoritmanın Tasarımı

ABGBF-KAÜ ailesi için genel kapsamlı KE saldırısının [4] iyileştirilmesi için yapılan araştırmalarımızda öncelikle İyileştirme No:1 adını verdiğimiz; orijinal algoritmadaki, potansiyel doğru iç durumların elenmesine sebep olabilecek noktaya müdahale edilerek yeni bir Ortalama Tahmin Kapasitesi(Pr_g) tanımı yapılmış ve saldırı algoritması bu tanıma uygun şekilde güncellenmiştir. Yeni algoritmada Tahmin Kapasitesi 0.5 olan durumlar göz ardı edilip, yalnızca 0.5'ten büyük olan iç duruma özgü tahmin kapasiteleri işleme alındığı için doğru iç durumun uyuşmazlık değerinin haksız yere yükseltilmesinin önüne geçilmiştir. Ayrıca, yeni Pr_g formülüne göre yalnızca 0.5'ten büyük iç durumlar işleme alındığından Sonlanma Değeri (α_{ter}) küçüldüğü için algoritmanın çalışma süresi iyileştirilmiştir (Bkz.: Şekil 7.1 ve 7.2).

İkinci bir iyileştirme olan İyileştirme No:3'te ise sonlandırma değerini eşit parçalara bölerek algoritmanın aşama aşama çalışmasını sağladık. Bu iyileştirmede sonlandırma değerine paralel olarak eşik değeri de küçüldüğü için yanlış iç durumların daha kısa sürede elendiğini gördük (Bkz.:Tablo 7.4, Şekil 7.3 ve 7.4).

Bu iki iyileştirmenın yanı sıra 1 adet düzeltme noktası keşfedilmiş ve elenen iç durumların fazladan 1 saat vuruşu işletilmesinin önüne geçilmiştir. Tasarımın son aşaması olarak, 1 düzeltme ve 2 iyileştirme birleştirilerek Nihai Algoritma haline getirilmiştir.

8.2 Bulgular

Orijinal ve Nihai Algoritma 19 adet örnek geri besleme fonksiyonu için adil bir ortamda test edilmiştir. %99.9 başarım oranı için orijinal algoritmaya oranla en düşük 2,98 kat, en yüksek 66.59 kat hızlanma kaydedilmiştir. Öte yandan başarım oranı her iki algoritma için %99'a düşürülüp testler tekrarlandığında en düşük 2,25, en yüksek 62,50 kat hızlanma gözlemlenmiştir (Bkz.: Şekil 7.7 ve 7.8). Ortalama hızlanma ise %99.9 başarımda yürütülen testlerde 15,17 kat, %99 başarımla yürütülen testlerde ise 14,36 kat olarak kaydedilmiştir.

8.3 Bilinen Kısıtlar

KE saldırısı ve araştırmalarımız sonucunda tasarladığımız yeni algoritma çalışmak için doğası gereği Tahmin Kapasitesinin(Pr_g) 0.5'ten büyük olmasını gerektirmektedir. Aksi durumda geri besleme değeri tahmin edilemediği için zayıf iç durum tespiti mümkün olmamaktadır.

8.4 İleriye Yönelik Araştırma Konuları

İç Durum Geri Kazanım algoritması, sonlanma (α_{ter}) ve eşik (α_{thr}) değerlerinin optimal kullanımı üzerine daha yoğun bir çalışma yapılarak daha da hızlandırılabilir. Bölüm 7'de kaydedilen hızlanma oranları, α_{ter} ve α_{thr}'a ait varsayılan formüllerle hesaplandığında gerçekleşen hızlanma oranlarıdır. Bu noktada formüllerin, geri besleme fonksiyonu ve çıktı fonksiyonunun yapısından faydalanacak şekilde geliştirilmesi bir alternatif olarak düşünülebilir.

8.5 Son Yorumlar

Anahtarlı Boolean Geri Besleme Fonksiyonu olan Kayan Anahtar Üreteçleri, donanım alanı dar ve güç gereksinimi düşük donanımlar için ideal bir şifreleme algoritması türü olduğundan yeni tasarımlarda karşılaşmamız muhtemeldir. Çalışmamızda KE saldırısının

farklı bakış açılarıyla nasıl hızlandırılabildiğinin farkına varılmasını sağladık. Bu bakış açıları saldırı metotlarını geliştirmek isteyenler için yeni bir ışık olabilir.

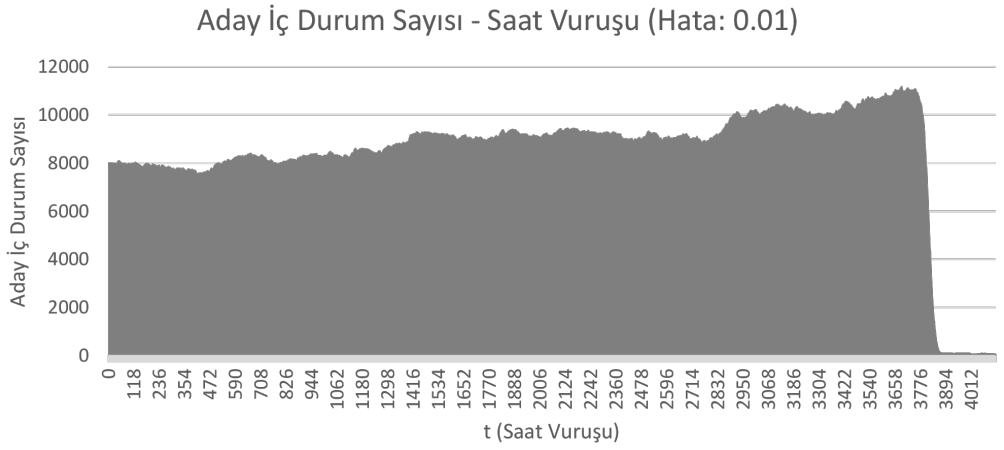
Ek A

KE Algoritması Bellek Kullanımı Raporu

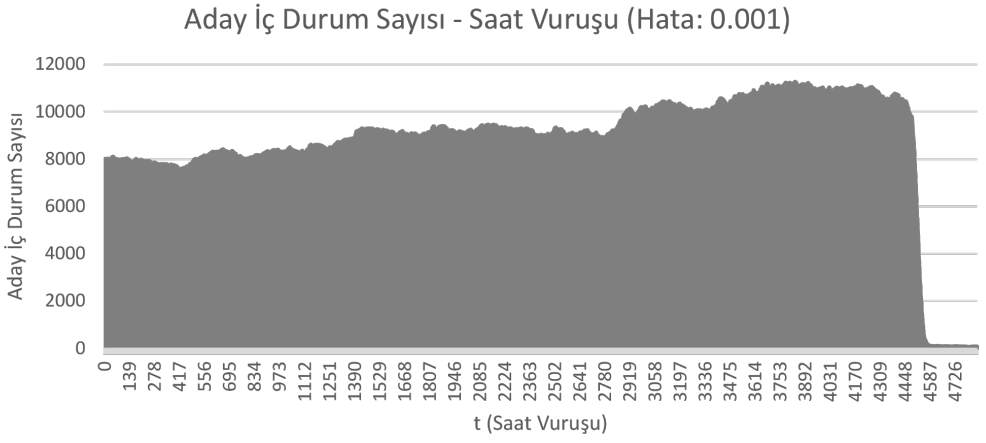
KE algoritmasının aday iç durumları işlerken bellekte ne ölçüde büyüdüğünü göstermek için algoritmada bu sayıyı dosyaya yazan küçük bir ilave yapılarak dosyaya günlük tutması sağlandı. Başlangıç, bitiş ve doğru iç durumu sırasıyla 590235, 598235 ve 594235 olduğu test durumunda aşağıdaki hata oranı 0.001 için yüzde 40.675, 0.01 için yüzde 39.675 artış gösterdiği gözlemlenmiştir.

Tablo A.1: KE algoritmasının f_{13} geri besleme fonksiyonunu işletirken bellek kullanımına ait en üst nokta değerleri

$\epsilon = 0.001$		$\epsilon = 0.01$	
Başlangıç	**Zirve**	**Başlangıç**	**Zirve**
iç durum: 8000 t = 0	iç durum: 11254 t = 3844	iç durum: 8000 t = 0	iç durum: 11174 t = 3689
40.675% artış		39.675% artış	



Şekil A.1: %99 başarım oranı için KE algoritmasının f_{13} geri besleme fonksiyonunu işletirkenki bellek kullanımı



Şekil A.2: %99.9 başarım oranı için KE algoritmasının f_{13} geri besleme fonksiyonunu işletirkenki bellek kullanımı

Ek B

Benzetim Uygulaması Kaynak Kodları

Bu bölümde orijinal KE saldırısı[4] ve yeni tasarladığımız saldırı algoritmasını içeren, C++ programlama diliyle yazılmış benzetim(simülasyon) yazılımına ait kodlar açıklanmıştır.

B.1 Geliştirme Süreci

Öncelikle KE saldırısı modeli koda dökülmüş ve çalıştırılmıştır. Çalışmamız iki ayrı projede yürütüldü. İlk proje olan **original_isr**'da, KE algoritması orijinal haliyle çalıştırılabildiği gibi, Bölüm 6'da anlattığımız düzeltme ve İyileştirme No:3 özelliklerinin opsiyonel olarak aktif edilebildiği sürümdür. İyileştirme No:1 daha köklü bir değişiklik gerektirdiğinden projenin ikinci bir kopyasını alarak **improved_isr** ismiyle kaydettik. İkinci proje, İyileştirme No:1'de anlatılan yönteminin kalıcı olarak uygulandığı ve orijinal halinden tamamen koptuğu, İyileştirme No:3'ün opsiyonel olarak aktif edilebildiği sürümdür. Nihai Algoritma olarak adlandırdığımız sürüm **improved_isr** projesinin düzeltme ve İyileştirme No:3'ün aktif edildiği halidir.

C++ programlama dilinin makro özelliğini kullanarak yaptığımız opsiyonel düzeltmeler doğası gereği çalışma zamanında değil, derlenme zamanında belirlendiği için kapatıldığında orijinal algoritmanın performansını etkilememektedir.

B.2 Proje Yapısı

Projeyi oluşturan kaynak kod dosyalarının içeriği basitçe aşağıdaki tabloda açıklanmıştır. Her iki proje de aynı yapıya sahiptir.

Tablo B.1: Projeyi oluşturan dosyalar

Dosya Adı	**Açıklama**
main.cpp	Uygulamanın başlangıç noktası. Main fonksiyonun bulunduğu dosya
internalstaterecovery.cpp	Saldırı algoritmasının bulunduğu dosya
constants.h	Algoritma sabitlerinin ve geri besleme fonksiyonlarının bulunduğu dosya. Ayrıca iyileştirme ve düzeltmelerin açılıp kapatılabildiği kısımları da içermektedir.
calculationutility.cpp	Algoritma kodlarının sade görünmesi için matematiksel hesapların yapıldığı yardımcı sınıfı içeren dosyadır
baseregister.cpp	Temel saklayıcı mantığını içeren üst sınıftır
keyregister.cpp	BaseRegister sınıfından kalıtım alan, anahtarın yüklenerek döngüsel kaydırma yapıldığı sınıfı içeren dosyadır
linearfeedbackshiftregister.cpp	BaseRegister sınıfından kalıtım alan, son bitini güncelleme yeteneğine sahip olan LFSR sınıfını içeren dosya
statewithmismatchcounter.cpp	Uyuşmazlık sayacı, kök ve iç durum saklama yeteneği olan sınıfı içeren dosyadır
keystreamgenerator.cpp	İlk iç durum ve ilk anahtar değeri kullanarak istenilen uzunlukta kayan anahtar çıktısı veren sınıftır

B.3 Proje 1

KE saldırısı, düzeltme ve İyileştirme No:3'ün opsiyonel olarak aktif edilebildiği ilk proje aşağıdaki GitHub adresinde bulunan v1.0 dağıtımı indirilerek temin edilebilir.

https://github.com/ilkeryasinyildiz/original_isr/releases

B.4 Proje 2

İyileştirme No:1'in KE algoritmasına kalıcı olarak uygulandığı, düzeltme ve İyileştirme No:3'ün opsiyonel olarak aktif edilebildiği ikinci proje aşağıdaki GitHub adresinde bulunan v1.0 dağıtımı indirilerek temin edilebilir.

https://github.com/ilkeryasinyildiz/improved_isr/releases

Kaynaklar

[1] F. Armknecht and V. Mikhalev. On lightweight stream ciphers with shorter internal states. In *International Workshop on Fast Software Encryption*, pages 451–470. Springer, 2015.

[2] V.A. Ghafari, H. Hu, and C. Xie. Fruit: ultra-lightweight stream cipher with shorter internal state. *IACR. http://eprint. iacr. org/2016/355*, 2016.

[3] V. Mikhalev, F. Armknecht, and C. Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52–79, Feb. 2017. doi: 10.13154/tosc.v2016.i2.52-79. URL `https://tosc.iacr.org/index.php/ToSC/article/view/565`.

[4] O. Kara and M. F. Esgin. On analysis of lightweight stream ciphers with keyed update. *IEEE Transactions on Computers*, PP:1–1, 06 2018. doi: 10.1109/TC.2018.2851239.

[5] C. De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In S. K. Katsikas, J. López, M. Backes, S. Gritzalis, and B. Preneel, editors, *Information Security*, pages 171–186, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-38343-7.

[6] Silicon Labs. Em351/em357. `https://www.silabs.com/documents/public/data-sheets/EM35x.pdf`, 2013. Erişim: 2019-04-07.

[7] NXP Semiconductors. Jns-5148. `https://www.nxp.com/docs/en/brochure/JN5148_PB_1v4.pdf`, 2010. Erişim: 2019-04-07.

[8] P. Kitsos, N. Sklavos, G. Provelengios, and A. Skodras. Fpga-based performance analysis of stream ciphers zuc, snow3g, grain v1, mickey v2, trivium and e0. *Microprocessors and Microsystems*, 37:235–245, 03 2013. doi: 10.1016/j.micpro.2012.09.007.

[9] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[10] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in Cryptology - ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000. ISBN 3-540-41404-5.

[11] S. Babbage. Improved exhaustive search attacks on stream ciphers. Security and Detection 1995, European Convention IET, 1995.

[12] J. D. Golić. Cryptanalysis of alleged A5 stream cipher. In *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 239–255. Springer, 1997. ISBN 3-540-62975-0.

[13] S. Babbage and M. Dodd. The stream cipher mickey 2.0. *ECRYPT Stream Cipher*, 2006.

[14] M. Hell, T. Johansson, and W. Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[15] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann anmd J.B. M. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466. Springer, 2007.

[16] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, and C. vd. Prince–a low-latency block cipher for pervasive computing applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 208–225. Springer, 2012.

[17] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni. Midori: a block cipher for low energy. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 411–436. Springer, 2015.

[18] J. Guo, T. Peyrin, A. Poschmann, and M. Robshaw. The led block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 326–341. Springer, 2011.

[19] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: an ultra-lightweight blockcipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 342–357. Springer, 2011.

[20] R. Beaulieu, S.Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers. The simon and speck lightweight block ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[21] G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong. The simeck family of lightweight block ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 307–329. Springer, 2015.

[22] C. De Canniere, O. Dunkelman, and M. Knežević. Katan and ktantan—a family of small and efficient hardware-oriented block ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 272–288. Springer, 2009.

[23] W. Wu and L. Zhang. LBlock: A lightweight block cipher. In *Applied Cryptography and Network Security*, volume 6715 of *LNCS*, pages 327–344. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21553-7.

[24] F. Karakoç, H. Demirci, and A. E. Harmancı. Itubee: a software oriented lightweight block cipher. In *International Workshop on Lightweight Cryptography for Security and Privacy*, pages 16–27. Springer, 2013.

[25] V. Lallemand and M. Naya-Plasencia. Cryptanalysis of full sprout. In *Annual Cryptology Conference*, pages 663–682. Springer, 2015.

[26] B. Zhang and X. Gong. Another tradeoff attack on sprout-like stream ciphers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 561–585. Springer, 2015.

[27] M. F. Esgin and O. Kara. Practical cryptanalysis of full sprout with tmd tradeoff attacks. In *International Conference on Selected Areas in Cryptography*, pages 67–85. Springer, 2015.

[28] S. Dey and S. Sarkar. Cryptanalysis of full round fruit. *IACR Cryptology ePrint Archive*, 2017:87, 2017.

[29] V.A. Ghafari and H. Hu. Fruit-80: A secure ultra-lightweight stream cipher for constrained environments. *Entropy*, 20(3):180, 2018.

[30] R. Anderson. On fibonacci keystream generators. In *International Workshop on Fast Software Encryption*, pages 346–352. Springer, 1994.

[31] E. Dubrova and M. Hell. Espresso: A stream cipher for 5g wireless communication systems. *Cryptography and Communications*, 9, 12 2015. doi: 10.1007/s12095-015-0173-2.

[32] ISO/IEC 8859-1:1998. Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1. Standard, International Organization for Standardization, Geneva, CH, April 1998.

[33] ISO/IEC 10646:2017. Information technology — Universal Coded Character Set (UCS). Standard, International Organization for Standardization, Geneva, CH, December 2017.

[34] P. Chown. Advanced encryption standard (aes) ciphersuites for transport layer security (tls). RFC 3268, RFC Editor, June 2002. URL `https://www.rfc-editor.org/rfc/rfc3268.txt`.

[35] M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of a5/1. `http://www.scard.org/gsm/a51.html`, 1999. May 1999.

[36] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of a5/1 on a pc. In G. Goos, J. Hartmanis, H. V. Leeuwen, and B. Schneier, editors, *Fast Software Encryption*, pages 1–18, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44706-1.

[37] 3GPP. *Specification of the 3GPP confidentiality and integrity algorithms; Document 2: Kasumi specification.* Number TS35.202 in TS. 3GPP, December 2008. URL `http://www.3gpp.org/ftp/Specs/html-info/35202.htm`. Rel-8 v8.0.0.

[38] 3GPP. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2; Document 2: SNOW 3G specification.* Number TS35.216 in TS. 3GPP,

December 2008. URL http://www.3gpp.org/ftp/Specs/html-info/35216.htm. Rel-8 v8.0.0.

[39] R. Housley. Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP). RFC 3686 (Proposed Standard), January 2004. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc3686.txt.

[40] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015. ISSN 2070-1721. URL https://www.rfc-editor.org/rfc/rfc7465.txt.

[41] M. Morii and Y. Todo. Cryptanalysis for rc4 and breaking wep/wpa-tkip. *IEICE Transactions*, 94-D:2087–2094, 11 2011. doi: 10.1587/transinf.E94.D.2087.

[42] M. David, D. C. Ranasinghe, and T. Larsen. A2u2: A stream cipher for printed electronics rfid tags. In *2011 IEEE International Conference on RFID*, pages 176–183, April 2011. doi: 10.1109/RFID.2011.5764619.

[43] Y. Luo, Q. Chai, G. Gong, and X. Lai. A lightweight stream cipher wg-7 for rfid encryption and authentication. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–6. IEEE, 2010.

[44] J. Ouellette. NewScientist incredible physics behind the deadly 1919 boston molasses flood. https://www.newscientist.com/article/2114116-incredible-physics-behind-the-deadly-1919-boston-molasses-flood/, November 2016. Erişim: 2019-04-07.

[45] A.J. Garcia-Sanchez, F. Garcia-Sanchez, F. Losilla, P. Kulakowski, J. Garcia-Haro, A. Rodríguez, J.V. López-Bao, and F. Palomares. Wireless sensor network deployment for monitoring wildlife passages. *Sensors*, 10(8):7236–7262, 2010. ISSN 1424-8220. doi: 10.3390/s100807236. URL http://www.mdpi.com/1424-8220/10/8/7236.

[46] J. P. Dominguez-Morales, A. Rios-Navarro, M. Dominguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, D. Cascado-Caballero, A. Jimenez-Fernandez, and A. Linares-Barranco. Wireless sensor network for wildlife tracking and behavior classification of animals in doñana. *IEEE Communications Letters*, 20(12):2534–2537, Dec 2016. ISSN 1089-7798. doi: 10.1109/LCOMM.2016.2612652.

[47] D. Anthony, W.P. Bennett, M.C. Vuran, M. B. Dwyer, S. Elbaum, A. Lacy, M. Engels, and W. Wehtje. Sensing through the continent: towards monitoring migratory birds using cellular sensor networks. In *Proceedings of the 11th international conference on Information Processing in Sensor Networks*, pages 329–340. ACM, 2012.

[48] W.P. Bennett Jr. Simulation, development and deployment of mobile wireless sensor networks for migratory bird tracking. 2012.

[49] T. Bari, I.F.A. Kuipers, and I.O.W. Visser. Birdtracking: A wireless sensor network to observe bird life. 2010.

[50] R. Kays, B. Kranstauber, P. Jansen, C. Carbone, M. Rowcliffe, T. Fountain, and S. Tilak. Camera traps as sensor networks for monitoring animal communities. In *2009 IEEE 34th Conference on Local Computer Networks*, volume 1, pages 811 – 818, 11 2009. doi: 10.1109/LCN.2009.5355046.

[51] M. Hillman. An overview of zigbee networks. `https://www.mwrinfosecurity.com/assets/Whitepapers/mwri-zigbee-overview-finalv2.pdf`, 2006. Erişim: 2019-04-07.

[52] ECRYPT. estream: the ecrypt stream ciphger project. `http://www.ecrypt.eu.org/stream`, 2008. Erişim: 2019-04-07.

[53] M. Philip and Vaithiyanathan. A survey on lightweight ciphers for iot devices. In *2017 International Conference on Technological Advancements in Power and Energy (TAP Energy)*, pages 1–4, 12 2017. doi: 10.1109/TAPENERGY.2017.8397271.

[54] ISO/IEC 29192-3:2012. Information technology – Security techniques – Lightweight cryptography – Part 3: Stream ciphers. Standard, International Organization for Standardization, Geneva, CH, October 2012.

[55] Q. Zhang, J. Liu, and G. Zhao. Towards 5g enabled tactile robotic telesurgery. *arXiv preprint arXiv:1803.03586*, 2018.

[56] Independent A. Cuthbertson. Surgeon performs world's first remote operation using '5g surgery' on animal in china. `https://www.independent.co.uk/life-style/gadgets-and-tech/news/5g-surgery-china-robotic-operation-a8732861.html`, 2019. Erişim: 2019-01-17.

[57] V.A. Ghafari, H. Hu, and M. Alizadeh. Necessary conditions for designing secure stream ciphers with the minimal internal states. *IACR Cryptology ePrint Archive*, 2017:765, 2017.

[58] S. Banik, V. Mikhalev, F. Armknecht, T. Isobe, W. Meier, A. Bogdanov, Y. Watanabe, and F. Regazzoni. Towards low energy stream ciphers. *IACR Trans. Symmetric Cryptol.*, 2018:1–19, 2018.