Fırat Üniv. Müh. Bil. Dergisi
29(2), 119-130, 2017

Science and Eng. J of Fırat Univ.
29(2), 119-130, 2017

Donanım Tabanlı Trivium Akış Şifreleme Algoritmasının FPGA Ortamında Gerçekleştirilmesi

Ali Murat GARİPCAN[1], Ebubekir ERDEM[1], Taner TUNCER[1]

[1]Fırat Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü, ELAZIĞ
aberdem@firat.edu.tr

(Geliş/Received: 01.04.2017; Kabul/Accepted: 06.07.2017)

Özet

Bu çalışmada, kriptografide LFSR(Linear Feedback Shift Register-Doğrusal Geri Beslemeli Kaydırmalı Yazmaç) tabanlı senkron simetrik şifreleme yöntemlerinden biri olan ve rasgele sayı üreteci olarak da kullanılabilenTrivium dizi/akış şifreleme algoritması, VHDL(Very High Speed Hardware Description Language – Çok hızlı donanım tanımlama dili) dili yardımıyla sonlu durum makinası olarak modellenmiştir. Modellemesi yapılan algoritma, FPGA (Field Programmable Gate Array- Sahada Programlanabilir Kapı Dizileri) ortamında gerçeklenmiş ve kriptografik uygulamalarda sayısal verinin şifrelenmesi için anahtar olarak kullanılan rastgele görünümlü bit dizileri üretilmiştir. Algoritmanın doğruluğu, fonksiyonel simülasyon araçları yardımıyla zamansal olarak test edilerek üretilen bit dizilerinin rastgelelik değerleri NIST(National Institute of Standards and Technology – Ulusal Standartlar ve Tek. Enstitüsü) kriterleri doğrultusunda ölçülerek elde edilen sonuçlar yorumlanmıştır.

Anahtar Kelimeler: Trivium, Akış şifreleme, Rasgele sayı üreteçleri, NIST testleri.

Hardware Based Trivium Stream Chiper Algorithm Implementation on FPGA Environment

Abstract

In this study, Trivium array/stream encryption algorithm which is synchronous and one of the symmetric encryption methods based on LFSR(Linear Feedback Shift Register) used in cryptography, is modeled as a finite state machine via the VHDL(very high speed integrated circuit- hardware description language). The modeling algorithm is implemented in the scene FPGA (Field Programmable Gate Array) environment and random-looking bit sequences that are used as keys, are generated for the encryption of numeric data in cryptographic applications. The accuracy of the algorithm is tested temporally with the aid of functional simulation tools and the results obtained by measuring the randomness values of the generated bit sequences in terms of NIST(National Institute of Standards and Technology) criteria, are interpreted.

Keywords : Trivium, Stream chiper, Random number generators, NIST tests.

1. Giriş

Son yıllarda sayısal haberleşme teknolojilerini kullanan kişi sayısının artışına bağlı olarak güvensiz iletim ortamları üzerinden taşınan verinin önem seviyesine bağlı olarak güvenliği önemli bir konu haline gelmiştir. Bu nedenle iletişime konu veriler üzerinde saldırganların olumsuz etkilerini azaltmak ve bu verilerin sadece alıcıları tarafından anlaşılabilirliğini sağlamak için kriptografik çözümlerin kullanılması zorunlu hale gelmiştir. Kriptografi, sayısal iletim ortamı üzerinden aktarılan verilerin yetkisiz erişimlerden korunması amacıyla geri dönüşümü mümkün olacak şekilde değiştirilerek iletilmesini sağlayan algoritmaların tasarımı ve analizi ile ilgilenir. Kriptografide bu maksatla kullanılan algoritmalar, gizli bir anahtar yardımıyla sayısal verilerinin anlaşılmaz hale dönüştürülmesi(şifreleme) ve alıcı tarafta anlaşılabilirliğinin sağlanması ile ilgili tersi süreçleri(de-şifreleme) yerine getirir.

Algoritmaların neredeyse tamamı, şifreleme ve de-şifreleme işlemlerinin yanı sıra bazı kritik noktalarda kesinlikle gizli kalması gereken rasgele olarak üretilmiş değerlerin/sayıların kullanımına ihtiyaç duyar. Bu değerler içerisinde en önemlisi, hiç şüphesiz kripto sistemlerin sürdürülebilir güvenirliği ve gücü konusunda belirleyici öneme sahip anahtar bileşenidir. Öyle ki; anahtar olmaksızın, şifreli metin saldırgan tarafından ele geçirilse dahi de-şifreli halinin elde edilmesi mümkün olmamalıdır. Fakat kriptografi de anahtar olarak kullanılacak tahmin edilmesi zor, tekrar üretilmesi mümkün olmayan rasgele sayı dizilerinin üretilmesi zordur. Ayrıca doğru yöntemler ile üretilmedikleri takdirde kullanıldıkları kriptografik uygulamaların güvenliğini bütünüyle tehlikeye atarlar. Bu nedenle algoritmik esaslara dayalı kripto sistemlerin, karmaşıklığının ve gizliliğin tümüyle üzerinde tutulduğu anahtar dizilerinin üretimi önemli bir probleme karşılık gelir [1].

Kripto sistemler içerisinde kullanılan rasgele sayılar, "Rasgele Sayı Üreteçleri" yardımıyla elde edilirler. Üreteçler; belli bir algoritma, matematiksel bir formül, önceden hesaplanmış tablolar veya sonuçları kestirilemeyen fiziksel süreçleri kullanılarak çeşitli şekillerde rasgele sayıların üretilmesini sağlayan yapılardır. Üreteçler, kendi aralarında sözde, gerçek ve hibrit rasgele sayı üreteçleri olmak üzere üç ayrı tasarım sınıfına ayrılır. Gerçek SRÜ, üretilen sayı dizisinin rasgelelik kaynağı olarak matematiksel olarak ifade edilemeyen fiziksel olayların kestirilemeyen davranışlarını kullanır. Aynı zamanda sözde rasgele sayı üreteci olarak da kullanılan akış şifreleme algoritmaları, başlangıç parametrelerine(tohum değerlere) ihtiyaç duyarlar ve bu parametreleri kullanarak deterministik yollarla rasgele görünümlü sayı dizileri üretirler. Bu üreteçlerin birbirlerine göre üstünlükleri ve eksikleri bulunmakla birlikte kriptografik uygulamaların önem seviyesine göre kullanımları da farklılık göstermektedir.

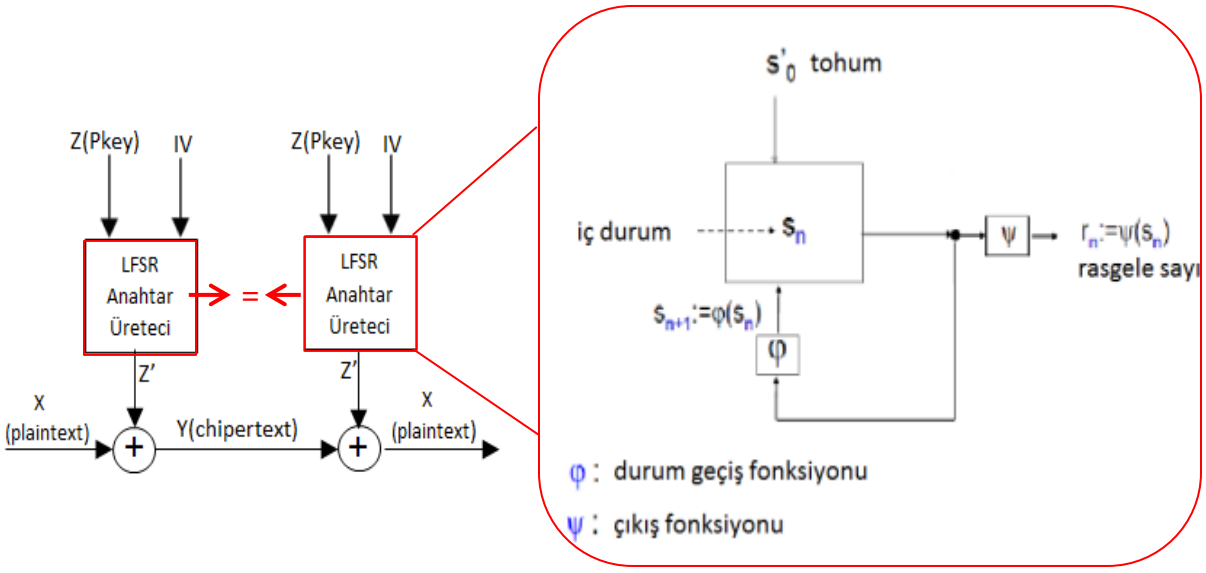
Bu çalışmada, ilk iki bölüm içerisinde kriptografik temel kavramlar, akış şifreleme sistemleri ve bu sistemler içerisinde kullanılan rastgele sayı üreteçlerinin yapısal özellikleri ile ilgili genel bilgiler verilmiştir. Üçüncü bölümde, çalışma içerisinde Trivium akış şifreleme algoritmasının yapısı incelenmiştir. Dördüncü bölümde Trivium akış şifreleme algoritmasının tasarım metodolojisi ve gerçekleme adımları detaylı biçimde açıklanarak kullanılan FPGA mimarisi hakkında bilgiler sunulmuştur. 5. bölümde algoritmanın benzetim araçları ile mantıksal doğrulanması ve FPGA üzerinde gerçeklemesi verilmiştir. 6. bölümde farklı tohum değerlerine karşılık elde edilen anahtar dizilerin rasgelelik ölçümleri sunulmuş olup son bölümde ise elde edilen sonuçlar tartışılmıştır.

2. Akış Şifreleme Sistemleri ve Sözde Rastgele Sayı Üreteçleri

Kriptografide Verman şifreleme olarak da bilinen akış şifreleme sistemleri, blok veya asimetrik şifreleme yöntemlerine göre çok daha yüksek hızlarda şifreleme performansı sunmaktadır. Akış şifreleme için kullanılan algoritmalar, verinin her bir biti yine verinin kendisiyle eşit uzunlukta bir anahtar dizisinin bitleriyle ayrı ayrı XOR fonksiyonundan geçirilmesi esasına dayanmaktadır [2]. Dolayısıyla şifreleme ve de-şifreleme işlemleri için verinin uzunluğuna bağlı olarak değişkenlik gösteren anahtar dizilerinin algoritmalar içerisinde üretilebilmesi şarttır. Şekil 2' de akış şifreleme algoritmalarının genel yapısı verilmiştir. Şifreleme işleminden önce algoritmalar Şekil 1' deki sözde bir rasgele sayı üretecinin genel tasarım mimarisine karşılık gelen işlemleri yerine getirir. Böylece şifreleme ve de-şifreleme işlemi için ihtiyaç duyulan özdeş rasgele anahtar dizilerini kendi içerisinde üretmiş olurlar.

Şekil 1' deki tasarım mimarisi içerisinde $r_1, r_2, \ldots . r_n \in R$ rasgele sayı dizisini, $S_n \in S$ ise saf üretecinin iç durumlarını temsil eder. Tasarım içerisinde sonlu S ve R kümeleri sözde rasgele sayı üretecinin durum uzayı ile çıkış uzayıdır. $\Psi : S \rightarrow R$ çıkış fonksiyonu, $r_n = \Psi(S_n)$ olacak biçimde mevcut S_n durumundan r_n rasgele sayısını üretir. Ardından φ geçiş fonksiyonu kullanılarak S_n durumu $S_{n+1=}\varphi(S_n)$ şeklinde güncellenir. Tasarım içerisinde ilk iç durumu temsil eden ve S_1 değeri tohum değere karşılık gelen S_0 durumunun $S_{1=}\varphi(S_0)$ şeklinde veya daha karmaşık şekilde oluşturulabilir [1,3].

Anahtar dizisi elde edildikten sonra Şekil 2' de açıkça görüldüğü üzere elde edilen anahtar dizisinin her bir bitini şifrelenecek metnin her bir bitiyle bir fonksiyona (genellikle XOR işlemi) sokarak şifreli metni elde eder. Simetrik şifreleme kategorisinde yer alan akış şifreleme sistemlerinde, metnin de-şifreli halinin alıcı tarafta elde edilebilmesi için gönderici tarafta kullanılan tohum değerlerin alıcı tarafta da bilinmesi veya gerekir. Çünkü de-şifreleme için kullanılacak olan özdeş anahtarın, algoritmalar içerisinde ancak şifreleme için kullanılan tohum değerler ile elde edilmesi mümkündür.



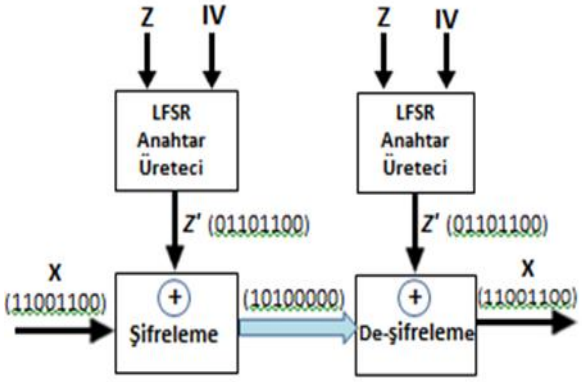
Şekil 1. Sözde bir rasgele sayı üretecinin genel tasarım mimarisi [1]

Donanım tabanlı akış şifreleme sistemlerinde, genelde doğrusal geri beslemeli öteleyici saklayıcılar(LFSR) kullanılır. LFSR' ler iyi istatistiksel özellikler gösteren, geniş periyoda sahip rasgele görünümlü anahtar dizilerini, düşük maliyetle üreten, donanımsal gerçeklemeleri kolay deterministik algoritmalardır. Şifreleme için kullanılan algoritmaların güvenlik ihtiyaçlarına bağlı olarak artan yapısal karmaşıklıkları, işlenme sürelerini arttırarak iletim hızını düşürmenin yanı sıra güç tüketimlerini de arttırmaktadır. Bilgi güvenliğinin yanında işlem maliyeti ve hız kavramlarının da ön plana çıktığı uygulamalarda donanım tabanlı yaklaşımlar çok daha verimli ve etkili sonuçlar ortaya koyarlar [4,5,6,7].

Akış şifreleme sistemleri kendi içerisinde iyi istatistiksel özelliklere sahip anahtar dizileri üretebilirler. Fakat algoritmik yapılarından dolayı standart bir tanıma karşılık geldiklerinden ötürü, üretilen anahtar dizilerinin gerçekte rasgele oldukları söylenemez. Bu durum üreteçlerin çekirdek iç durumlarının veya tohum değerlerinin bilinmesi durumunda üretilecek anahtar bitlerinin tamamının tahmin edilmesini mümkün kılar. Bu nedenle hassas kriptografik uygulamalarda tohum değerlerinin olabildiğince rastgele ve tahmini zor olmalıdır. Tohum değerlerin ve üreteçlerin çekirdek durumlarının mutlak gizliliğinin sağlanması sistemin güvenliği açısından mutlak öneme sahiptir. Ayrıca algoritmaların içyapısı itibariyle, Şekil 1' de gösterilen durum ve geçiş fonksiyonlarının olabildiğince karmaşık bağıntılar içermesi çıkışta elde edilen rasgele görünümlü sayı dizilerinin tahmin edilmesini zorlaştırır [8].

Genel yapısı Şekil 2' de verilen akış şifrelemede gerçek gizli anahtar (Z), LFSR yapısının çıkışındaki kayar anahtardan farklıdır. K bitlik gizli anahtar Z, LFSR tabanlı üreteci kontrol ederek kayar anahtar(asıl anahtar) dizisi Z_1' , Z_2' ,...., Z_n' nin üretilmesini sağlar. Burada

Z' anahtar dizisini üreten K bitlik gerçek gizli anahtar, Z' ye tohum (seed) denir. Gerçek gizli anahtar (Z) hiçbir zaman şifreleme işleminde doğrudan kullanılmaz. Gizli anahtarın uzunluğu kayar anahtarın uzunluğu yanında oldukça küçüktür ($K \ll N$). Şifreli metin bitleri, Denklem 1' deki gibi ikili açık-metin bitlerinin kayar anahtar dizisiyle terim terim modülo-2 toplanması sonucunda elde edilirler [9]. Tipik bir akış şifreleme sisteminin, şifreleme ve de-şifreleme işlemleri için kullandığı yöntem Denklem 1 ve Denklem 2' de verilmiştir. X_n , şifrelenecek metnin n. biti, Y_n, anahtar dizisinden geçirilerek elde edilmiş şifreli metnin n. biti, Z'_n ise LFSR çıkışındaki kayar anahtar(şifreleme ve de-şifreleme anahtarı) dizisinin n. bitidir.



Şekil 2. Akış şifreleme sistemlerinin genel yapısı

$$Y_n = X_n \oplus Z'_n \quad n = 1,2 \ldots N \ (Şifreleme) \qquad (1)$$

$$X_n = Y_n \oplus Z'_n \quad n = 1,2 \ldots N \ (De\ şifreleme) \qquad (2)$$

Akış şifreleme sistemlerinde şifreleme ve şifre çözme işlemleri, iç yapısı zamana bağlı durum değiştiren eşdeğer donanımsal yapılar üzerinde gerçekleşmektedir. De-şifreli metnin elde edilmesi süreci, şifreleme işleminin tersine girdi olarak şifrelenmiş metin bitlerini LFSR yapısının çıkışındaki kayar anahtar dizisiyle terim terim modülo-2 toplanması sonucunda elde edilir. Şifreli metnin açık halinin elde edilmesi süreci Denklem 2' deki gibidir.

3. Trivium Akış/Dizi Şifreleme Algoritması

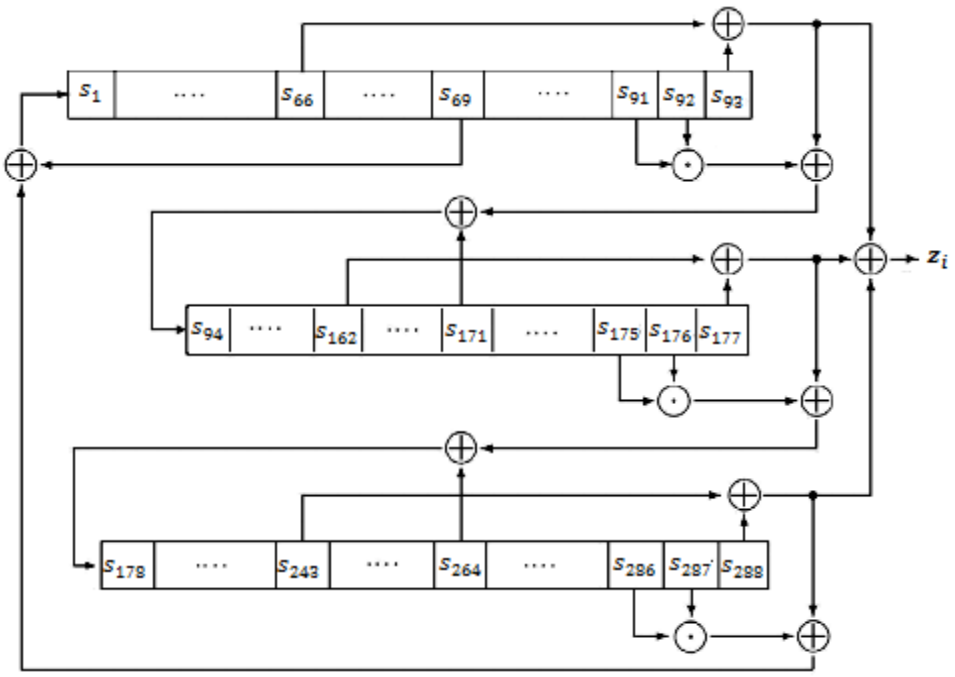
Mevcut akış şifreleme algoritmalarının daha az gerçekçi kripto analitik saldırılarla karşı karşıya kaldıkları ve bu anlamda blok şifreleme algoritmalarına göre daha az ilgi odağı oldukları görülmüştür. Bu nedenle kolay kontrol edilebilir güvenlik ölçütleriyle yeni akış şifreleme algoritmalarının tasarımı amacıyla 2004-2008 yılları arasında eSTREAM(the ECRYPT Stream Cipher Project) projesi başlatılmıştır. Proje yazılım, donanım ve yazılım-donanım olmak üzere üç ayrı kategoride düzenlenmiştir. Christophe De Canniere ve Bart Preneel tarafından 2006 yılında sunulan Trivium, iki yıllık değerlendirmeler sonucunda 2008 yılında donanım kategorisinden Grain [10] ve MICKEY [11] ile birlikte kabul edilen üç algoritmadan biri olmuştur [12,13].

Grain ve MICKEY algoritmaları, kendi içerisinde bir adet LFSR, bir adet NFSR(Nonlinear Feedback Shift Register-Doğrusal Olmayan Geri Beslemeli Saklayıcı) olmak üzere toplam iki adet, shift register yapısını kullanır. MICKEY algoritmasında shift register yapıları düzensiz saat girişleriyle tetiklenir. Her üç algoritma donanımsal gerçekleme için ihtiyaç duyulan lojik kapı sayısının azlığına bağlı olarak donanımsal karmaşıklığı ve enerji tüketimleri azdır. Bu nedenle küçük donanımsal yapılar üzerinde gerçeklemeye yatkındırlar. Şifreleme performanslarının, algoritmalar içerisinde kullanılan tohum değerlerin uzunluğuna bağlı olarak değişebileceği unutulmamalıdır. MICKEY algoritması, Trivium ve Grain' e göre gerçekleme için daha az donanımsal kaynağa ihtiyaç duymaktadır [7, 14,15].

Bit tabanlı senkron akış şifreleme algoritması olan Trivium algoritması güvenlik, hız ve esneklikten fedakarlık etmeden bir dizi şifreleme sisteminin ne kadar basitleştirilebileceğini araştırmak için tasarlanmıştır. En basit tasarım yapısına sahip olmasının yanı sıra Trivium, Grain ve MICKEY ile performans olarak kıyaslandığında çıkış bit oranı yüksektir ve şifreleme işlemi için en az süreye ihtiyaç duyan bir algoritmadır. Bu nedenle hız kavramının öne çıktığı kriptografik

uygulamalar için düşük enerji tüketimiyle de ön plana çıkmaktadır [7,15, 16].

Şekil 3' de görüldüğü üzere Trivium içyapısı itibariyle 288 bitlik tek parça gibi davranan her biri 93, 84 ve 111 bitlik üç ayrı doğrusal olmayan LFSR(register) yapısını kullanır. Algoritma fonksiyonel olarak başlangıç değerlerinin(private key ve initial vector) yüklenerek iç durum oluşturulması ve anahtar dizinin elde edilmesi gibi iki ayrı aşamadan oluşur. 288 bitlik LFSR' nin 15 adet biti mantıksal düzeyde işlemlere tabi tutularak üç ayrı noktadan sisteme non-lineer geri besleme olarak verilmektedir. Algoritma 80 bitlik gizli anahtar ve 80 bitlik başlangıç vektörleriyle çıkışta $\mathbf{2^{64}}$ bitlik sahip anahtar dizisi üretebilir.



Şekil 3. Trivium akış şifreleme algoritmasının donanımsal içyapısı

Şekil 3'te görüldüğü üzere her registerin çıkışı aslında diğer registerin girişine bağlı olacak şekilde ayarlanmıştır. Bu nedenle algoritma, tek parça olarak davranan dairesel bir register yapısı şeklinde düşünülebilir [17]. Trivium' un, anahtar dizisinin elemanlarını çıkış olarak üretmeye hazır hale gelmeden önce yükleme ve kaydırma gibi ön işlemler ile iç durum güncellemesi yapması gerekir. Yükleme işlemi için, 80 bitlik gizli anahtar(private key) ile 80 bitlik başlangıç vektörü(initial vector) LFSR üzerine yüklenerek 288, 287, 286. bitler lojik '1' yapılır. Kaydırma işleminde ise oluşturulan 288 bitlik iç durum dört tam tur boyunca(4*288=1152 adım) döndürülür. Seçilen 15 özel bitin mantıksal işlemden geçirilerek geri beslenmesi yoluyla iç durum güncellemesi her adımda yapılarak hazır duruma getirilir. Bu aşamada anahtar dizisine ait lojik seviyede herhangi bir çıkış gözlenmez. Algoritmanın başlangıç parametreleri Tablo 1' de verilmiştir.

Tablo 1. Trivium parametreleri

Gizli Anahtar	80 bit
Başlangıç Vektörü	80 bit
İç durum uzunluğu	288 bit

Algoritmaya ait Tablo 2 ve Tablo 3' te verilen sözde kod yapısı içerisinde kullanılan tanımlamalar aşağıdaki gibidir:

- $(s_m, \ldots\ldots, s_n)$ = 288 bitlik LFSR yapısı
- S_i = 288 bitlik LFSR üzerindeki *i.* bit
- $z_t = t$. sürede üretilen anahtar biti
- $\oplus$ operatörü, XOR işlemi
- . işareti AND işlemi
- t_1, t_2, t_3 = aktif geri besleme bitleri

Trivium' un sırasıyla iç durum oluşturma ve anahtar bitlerini üretme işlemlerine karşılık sözde kod yapısı sırasıyla Tablo 2 ve Tablo 3' deki gibidir.

Tablo 2. İç durum işlemleri sözde kod yapısı

$(s_1, s_2, \dots \dots, s_{93}) \leftarrow (K_1, \dots, K_{80}, 0, \dots 0)$
$(s_{94}, s_{95}, \dots \dots, s_{177}) \leftarrow (IV_1, \dots, IV_{80}, 0, \dots 0)$
$(s_{178}, s_{179}, \dots \dots, s_{288}) \leftarrow (0, \dots .0, 1, 1, 1)$
for i=1 to 4*288 do
$t_1 \leftarrow s_{66} \oplus s_{93}$
$t_2 \leftarrow s_{162} \oplus s_{177}$
$t_3 \leftarrow s_{243} \oplus s_{288}$
$t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
$t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}$
$t_3 \leftarrow t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69}$
$(s_1, s_2, \dots \dots, s_{93}) \leftarrow (t_3, s_1, \dots, s_{92})$
$(s_{94}, s_{95}, \dots \dots, s_{177}) \leftarrow (t_1, s_{94}, \dots, s_{176})$
$(s_{178}, s_{179}, \dots, s_{288}) \leftarrow (t_2, s_{178}, \dots, s_{287})$
end for

Tablo 3. Anahtar üretme işlemleri sözde kod yapısı

for i=1 to n do
$t_1 \leftarrow s_{66} \oplus s_{93}$
$t_2 \leftarrow s_{162} \oplus s_{177}$
$t_3 \leftarrow s_{243} \oplus s_{288}$
$z_i \leftarrow t_1 \oplus t_2 \oplus t_3$
$t_1 \leftarrow t_1 \oplus s_{91} \cdot s_{92} \oplus s_{171}$
$t_2 \leftarrow t_2 \oplus s_{175} \cdot s_{176} \oplus s_{264}$
$t_3 \leftarrow t_3 \oplus s_{286} \cdot s_{287} \oplus s_{69}$
$(s_1, s_2, \dots, s_{93}) \leftarrow (t_3, s_1, \dots, s_{92})$
$s_{94}, s_{95}, \dots, s_{177}) \leftarrow (t_1, s_{94}, \dots, s_{176})$
$(s_{178}, s_{179}, \dots, s_{288}) \leftarrow (t_2, s_{178}, \dots, s_{287})$
end for

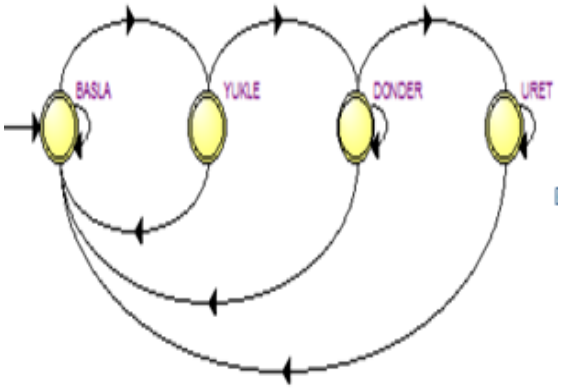
Sözde kodları Tablo 3' de verilen ikinci aşamada ise, başlangıç durumuna getirilmiş 288 bitlik LFSR yapısı şifrelenecek metnin bit cinsinden toplam uzunluğu (n bit) kadar kaydırılır. Her kaydırma işleminde LFSR üzerinden seçilmiş 15 özel bit Şekil 1' de belirtilen çıkış ve geçiş fonksiyonlarına karşılık gelen mantıksal işlemlerden geçirilir. Lojik düzeyde her bir anahtar bit çıkışı için LFSR' nin iç durum güncellemesi ilk aşamada olduğu gibi tekrarlı bir biçimde istenen sayıda anahtar bit üretilinceye kadar yapılmaya devam eder [18].

4. Trivim Akış Şifreleme Algoritmasının FPGA Üzerinde Gerçeklemesi

Uygulama için, içerdikleri mantıksal bloklar ile bu bloklar arasındaki bağlantıları, istenen fonksiyona göre üretimden sonra yüksek seviyeli donanım tanımlama dilleri ile tekrar tekrar programlanabilme özelliğine sahip FPGA donanımı kullanılmıştır. Sahada programlanabilme özelliğine sahip FPGA donanımlarının en özelliklerinden birisi de paralel işlem yapabilme yeteneğidir. Aynı anda birden fazla işlem yapabilen FPGA donanımları; diğer entegrelere göre çok daha yüksek hızlarda çalışarak zaman ve maliyet anlamında kullanıcılara önemli kolaylıklar sağlamanın yanı sıra karmaşık devre elemanlarının işlevselliğini arttırmaktadır [19].

Trivium akış şifreleme algoritmasının gerçekleme işlemi için Altera Cyclone IV EP4CE114FC6 serisi FPGA donanım kartı kullanmıştır. Tasarım aşamasında davranışsal modelleme işlemleri için yüksek hızlı donanım tanımlama dillerinden biri olan VHDL dili kullanılmıştır. Oluşturulan davranışsal modelin sentezleme, optimizasyon ve simülasyon işlemleri için ise Altera'ya ait Quartus II ile yine aynı yazılım tarafından desteklenen fonksiyonel ve zamansal simülasyon aracı Modelsim-Altera yazılım geliştirme araçları kullanılmıştır.

LFSR tabanlı akış şifreleme sistemleri; genelde sınırlı sayıda durum ve bu durumlar arasındaki geçişler ile durumların kendisine karşılık gelen kombinasyonel eylemlerden oluşan bir davranışsal makina modelidir. Dolayısıyla ilk olarak Trivium akış şifreleme algoritması Moore senkron ardışıl devre modeli olarak tasarlanmış ve tasarımdaki durumların neler olacağı, sayısı ve birbirleriyle ilişkileri ile kontrol geçişleri belirlenmiştir. VHDL dili içerisinde ardışıl devrelerin modellenmesi için FSM(Finite State Machine-sonlu durum makinası) kodlama tekniği kullanılmıştır. Kodlanan algoritmanın kendisine karşılık gelen ardışıl devreye ait dört durumlu FSM modeli Şekil 4' teki gibidir.



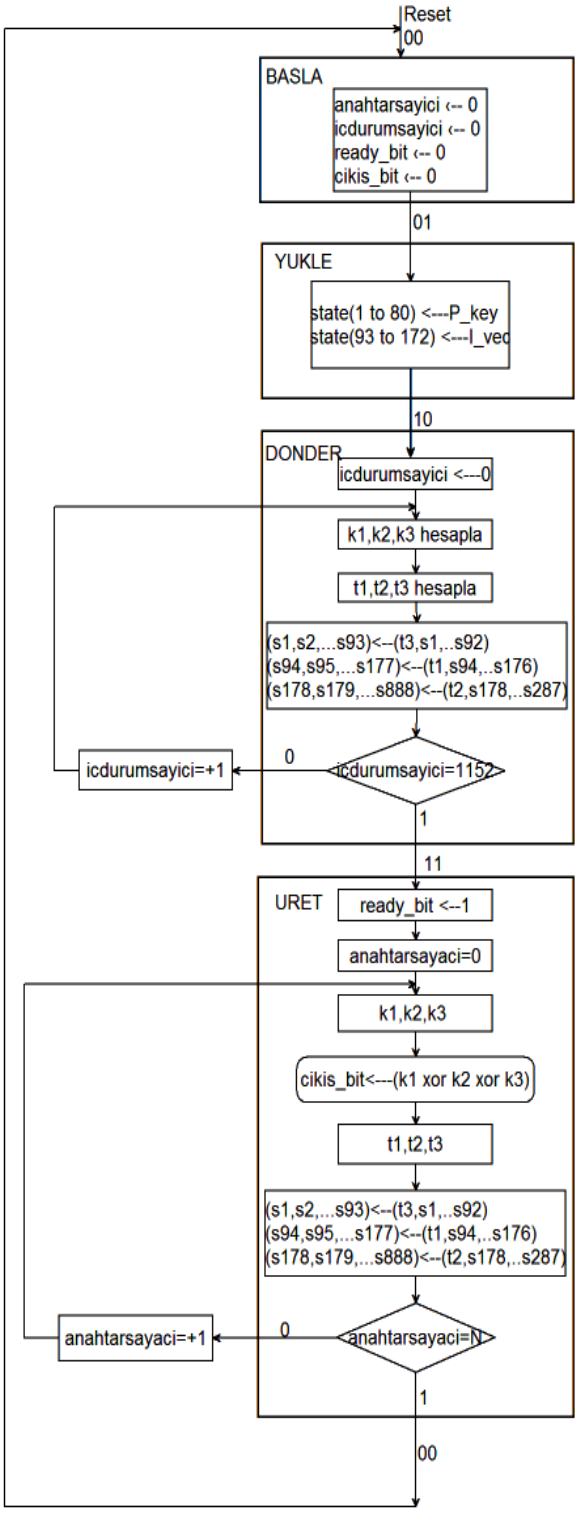
Şekil 4. Dört durumlu FSM Modeli

FSM modelinde her bir durum ve bu durumlar içerisinde tanımlı kombinasyonel işlemler ile ilgili davranışsal tanımlamalar yapılmıştır. Süreç kısaca aşağıdaki gibi özetlenebilir.

İlk olarak giriş çıkış portları ile ilgili genel tanımlamalar yapılmıştır. Ardından 288 bitlik LFSR yapısına ait vektörel tanımlama işlemi yapılarak bütün değerlerine lojik '0' değeri atanır. Daha sonra algoritmanın çalışması için gerekli olan hexadicamal olarak rastgele atanmış 80 bitlik tohum değer (gizli anahtar) ile tüm elamanları lojik '0' olacak şekilde başlangıç vektörü (IV) tanımlamaları yapılmış ve LFSR yapısı üzerinde belirlenen aralıklara yerleştirilmiştir.

Durumlar arası geçiş işlemleri için iki bitlik lojik kontrol değişkeni tanımlaması yapılmıştır. FSM, durumlar arasında geçişler tanımlanan iki bitlik lojik kontrol değişkenin "00" değeri için "BASLA", "01" değeri için "YUKLE", "10" değeri için DONDÜR ve "11" değeri için "URET" pozisyonunda olup bu durumlar için sırasıyla aşağıdaki işlemleri yapacaktır.

Sonlu durum makinasının algoritmik yapısını gösteren ASM(Algorithmic State Machine – Algoritmik Durum Makinası) diyagramı ile her durum içerisinde yapılan kombinasyonel işlemler Şekil 5' teki gibidir.



Şekil 5. FSM' ye ait ASM diagramı

4.1.“BASLA” durumu

FSM, başlangıç konumunda boş pozisyondadır. Bu konumda, iç durum güncelleme ve anahtar bitlerini üretmek için tanımlanan sayaç sinyalleri ile anahtar çıkışı lojik ‘0’ seviyesindedir. Ayrıca çıkış bitlerinin sadece iç durum güncellemesinden sonra çıkışa aktarılması için kullanılan kontrol sinyali lojik ‘0’ seviyesindedir. FSM, durum değiştirmek için durumlar arasında geçişleri sağlayan kontrol sinyalini bekler. Kontrol sinyali “01” olduğunda durum değiştirerek yükleme konumuna geçer.

4.2. “YUKLE” durumu

FSM, yükleme durumunda 80 bitlik gizli anahtar ile başlangıç vektörünü sırasıyla 288 bitlik LFSR yapısının ilk 0-80 ve 93-172. bitleri arasına yerleştirir. Daha sonra LFSR yapısının en ağırlıklı üç bit (288,287,286. bitler) lojik ‘1’ seviyesine getirilerek, sonlu durum makinası, anahtar üretmeden önceki iç durum güncellemesi için hazır hale gelerek bir sonraki duruma geçer.

4.3. “DONDÜR” durumu

FSM, anahtar dizisine ait çıkış bitlerini üretmeye başlamadan önce 288 bitlik vektör tabanlı LFSR yapısını dört tam tur boyunca(4*288=1152 defa) kaydırır. Her kaydırma işleminde LFSR yapısı üzerinde seçilmiş 15 özel bit mantıksal işlemlerden geçirilerek geri besleme noktalarından sisteme girdi olarak verilir. 1152 defa tekrar eden bu durum için, iç durum alt sınırı temsil eden sayaç tanımlanmış ve ayrı bir process bloğu içerisinde her saat darbesinde alt sınır değeri arttırılarak aynı durum içerisinde üst sınır değeriyle(1152) kontrolü yapılmıştır. FSM sayaç değeri üst sınır değerine ulaşana kadar aynı konumda kalarak kaydırma ve döndürme işlemlerini devam ettirir. Sayaç değeri üst sınır değere ulaştığı zaman saat darbesiyle bir sonraki duruma yani üretme pozisyonuna geçer ve anahtar bit dizisine ait lojik değişimleri çıkışa verir.

VHDL dilinin paralel çalışma özelliğinden dolayı aynı mimari içerisinde tanımlanan kod/process blokları yazılış sırasına bakılmaksızın eş zamanlı çalışırlar. Bu nedenle oluşan karmaşıklığın giderilmesi ve FSM’ nin anahtar dizisine ait lojik değişimleri sadece anahtar üretme konumundayken çıkışa vermesi için bir bitlik lojik kontrol değişkeni kullanılmıştır. Bu anlamda lojik değişken, multiplexer gibi davranan bir kod bloğunun seçici ucu olarak düşünülmüştür. Lojik kontrol değişkeni FSM “URET” durumunda iken lojik ‘1’ seviyesindedir ve anahtar bitlerinin değişkenliği çıkışta gözlenebilir. Diğer durumlarda lojik ‘0’ olup, anahtar çıkış bitleri daima lojik ‘0’ seviyesinde kalır.

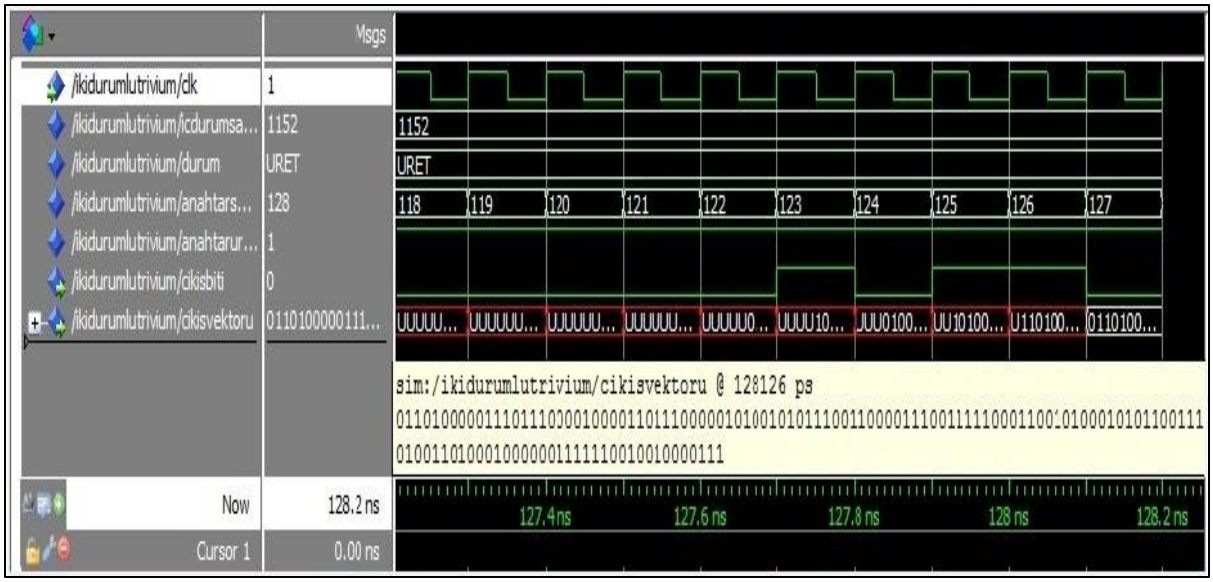
4.4. “URET” durumu

İç durum güncellemesini tamamlayan FSM, bu konumda bir önceki durumdan farlı olarak anahtar dizisine ait çıkış bitlerini üretir. Her çıkış bitinden sonra LFSR’ nin içyapısı, geri besleme yoluyla kaydırılarak güncellenir ve bir sonraki anahtar bitini üretmeye hazır konuma gelir. Bu durum, çıkış bitlerinin sayıca şifrelenecek metnin bit türünden sayısal uzunluğuna eşit oluncaya kadar tekrarlı bir şekilde devam eder. FSM’ nin anahtar dizisinin tüm bitlerini üretene kadar aynı konumda kalabilmesi için sayaç tanımlanmıştır. Her anahtar biti için sayaç değeri arttırılarak üst sınır değere ulaşıp ulaşmadığı kontrol edilmiştir. Anahtar dizisine ait tüm bitler üretildikten sonra sayaç değeri sıfırlanır ve FSM “BASLA” durumuna geri döner.

5. Algoritmanın Doğrulanması

FSM olarak modellenen Trivium akış şifreleme algoritması iki aşamada doğrulanmıştır. İlk aşamada FPGA bordu üzerine yüklenmeden önce mantıksal hataların önüne geçmek ve modelin doğru davranış sergilediğini görebilmek için Modelsim-Altera 10.1d benzetim aracı kullanılmıştır. Algoritma ilk olarak benzetim ortamında 32, 64, 128 bitlik üç ayrı anahtar dizisi için sırayla çalıştırılmıştır. Kullanılan lojik seviyedeki giriş, çıkış ve sayaç ve kontrol sinyalleri ile durumların zamana bağlı değişimleri izlenerek mantıksal doğrulaması yapılmıştır. Benzetim sonuçları şekil 6’ daki gibidir.

Gerçekleme aşamasında, mantıksal doğrulaması yapılan algoritmaya ait kodlar lojik düzeyde sentezlenerek, yerleştirme ve yollandırma işlemlerinin ardından FPGA donanımı üzerine yüklenerek, anahtar dizilerine ait lojik çıkışlar gözlemlenmiştir.



Şekil 6. Trivium akış şifreleme algoritmasının benzetim sonuçları

Sonlu durum makinası, üç ayrı anahtar dizisi için 0.2 ns(nanosaniye) ile 115.4 ns arasında yükleme kaydırma işlemlerini gerçekleştirerek iç durum güncelleme işlemlerini bitirerek ve anahtar üretme pozisyonuna geçmiştir. Anahtar dizilerinin 115,4 ns den sonra 32 bit için 118,6 ns, 64 bit için 120,6 ns ve 128 bit için 128.2 ns sürede üretildiği görülmüştür. Bir ".txt" uzantılı metin dosyasına rassallık ölçümü için yazılan bir milyon adet bitin ise yine 115.4 ns den başlayarak 100115.499 ns süre içerisinde üretildiği gözlemlenmiştir. Ayrıca algoritma benzetim ortamında adım adım çalıştırılarak herhangi bir durumda "reset" girişi lojik '1' seviyesine çekildiğinde sonlu durum makinasının başlangıç konumuna geri döndüğü görülmüştür.

Doğrulamanın ikinci aşamasında rasgele sayılardan oluşan anahtar dizilerinin gerçek bir rastgele diziden beklenen yeterlilikleri karşılayıp karşılamadığına bakmak için teste tabi tutulmuştur. Bu konuda pek çok test(FIBS 140, DieHard, NIST vs.) bulunmasına rağmen bu çalışma kapsamında hipotez tabanlı NIST 800-22 istatistiksel test süiti kullanılmıştır. Diğer testlere oranla daha güçlü bir yapı içeren NIST 800-22 yazılım süiti rasgelelik ölçümü için kendi içerisinde 15 ayrı kritere ait testten oluşmaktadır. Teste tabi tutulan bit dizisinin başarılı olabilmesi için tüm testleri başarıyla geçmesi gerekmektedir.

Doğrulamanın ikinci aşamasında ise çıkış olarak elde edilen, şifreleme ve de-şifreleme için kullanılacak anahtar dizisinin, NIST kriterlerine tabi tutularak rasgeleliğin istatistiksel ölçüde sağlanıp sağlanmadığı kontrol edilmeye çalışılmıştır. Bu aşamada sınırlı sayıda bitlerden oluşan anahtar dizilerinin aksine yüksek yoğunlukta veri içeren(1 milyon adet bit) anahtar çıkışları elde edilmeye çalışılmıştır.

Söz konusu çıkış bit değerinin yüksek olması nedeniyle anahtar dizisine ait çıkış bitlerinin FPGA üzerinde gözlemlenmesi mümkün değildir. Bu nedenle kod yapısı içerisinde tampon bellek ve dosya yapısı kullanılmıştır. VHDL dili hem sentez hem de benzetim için kullanılabilir. Tümüyle benzetime uygun olduğu halde, bazı kodlar sentezlenebilir değildir. Dosya yapıları da FPGA yapıları üzerinde donanımsal karşılıklarının oluşturulamaması nedeniyle sentezlenemezler.

Sentezlenemeyen kod yapısı sadece benzetim ortamında çalıştırılarak çıkış bitleri, bir metin dosyasına kaydedilerek elde edilmiştir. Elde edilen metin dosyası NIST 800-22 test suitine dahil edilerek rasgelelik değerleri ölçülmüş ve Tablo 4 ile Tablo 5' teki sonuçlar elde edilmiştir.

Tablo 4. İlk anahtar dizisine ait test sonuçları

Test	P- Değer	Sonuç
Frekans Testi	0.576	Başarılı
Blok Frekans Testi	0.709	Başarılı
Akış Testi	0.084	Başarılı
Blok İçindeki En Uzun Birler Testi	0.946	Başarılı
İkili Matris Rankı Testi	0.327	Başarılı
Ayrık Fourier Dönüşümü Testi	0.720	Başarılı
Örtüşmeyen Şablon Eşleştirme Testi	0.532	Başarılı
Örtüşen Şablon Eşleştirme Testi	0.466	Başarılı
Maurer'in Evrensel İstatistik Testi	0.129	Başarılı
Doğrusal Kar. Testi	0.907	Başarılı
Seri Testi	0.193 0.552	Başarılı
Yaklaşık Entropi Testi	0.387	Başarılı
Kümülatif Toplam Testi	0.616	Başarılı

Tablo 5. İkinci anahtar dizisine ait test sonuçları

Test	P- Değer	Sonuç
Frekans Testi	0.602	Başarılı
Blok Frekans Testi	0.870	Başarılı
Akış Testi	0.694	Başarılı
Blok İçindeki En Uzun Birler Testi	0.659	Başarılı
İkili Matris Rankı Testi	0.368	Başarılı
Ayrık Fourier Dönüşümü Testi	0.143	Başarılı
Örtüşmeyen Şablon Eşleştirme Testi	0.032	Başarılı
Örtüşen Şablon Eşleştirme Testi	0.764	Başarılı
Maurer'in Evrensel İstatistik Testi	0.624	Başarılı
Doğrusal Kar. Testi	0.427	Başarılı
Seri Testi	0.281 0.542	Başarılı
Yaklaşık Entropi Testi	0.883	Başarılı
Kümülatif Toplam Testi	0.301	Başarılı

6. Sonuçlar

Bu çalışma kapsamında, şimdiye kadar başarıya ulaşmış analitik bir saldırının gözlemlenmediği LFSR tabanlı Trivium akış şifreleme algoritması, Altera – Quartus II ortamında VHDL donanım tanımlama dili yardımıyla modellenmiştir. Diğer LFSR tabanlı şifreleme algoritmalarında olduğu gibi başlangıç değerlerine ihtiyaç duyan Trivium akış şifreleme algoritmasının seçilen parametreler doğrultusunda elde edilen anahtar dizileri istatistiksel testlere tabi tutularak rastgelelikleri ölçülmüş ve değişimleri incelenmiştir. Çalışma kapsamında elde edilen sonuçlar algoritmanın kendisi ile sonuçlarının analizi olmak üzere iki ayrı aşamada değerlendirilmiştir.

Diğer donanım tabanlı yaklaşımlar gibi kendi içerisinde sözde rastgele sayı üreteçlerini kullanan Trivium akış şifreleme algoritması, bilgisayar kaynaklarının(enerji tüketimleri, hafıza boyutları ve işlemci performansları) sınırlı olduğu durumlarda daha az karmaşıklık içeren yapısı ve paralel işleme tekniklerinin kullanıldığı donanımsal yapılar üzerinde gerçeklemeye yatkın bir algoritmadır. Algoritma bu yönüyle şifreleme işlemleri için hızlı ve efektif çözümler üretebilmektedir. Anahtar dizilerini, yazılımsal çözümlere göre düşük maliyetle üretebilen Trivium algoritmasında, anahtar dizilerine ait periyodun mümkün olduğunca uzun tutulması ve her peryot için farklı bir tohum/gizli anahtar değerinin kullanılması gerekmektedir. Ayrıca rasgeleliğe kaynak oluşturan ve algoritmanın üzerine kurulu olduğu tohum anahtar değerin, olabildiğince rastgele dağılıma sahip olması ve LFSR yapısının güncellemeyle oluşan iç durumlarının mutlaka gizli kalması algoritmanın güvenirliği konusunda belirleyici öneme sahiptir. Yeterince komplex olması istenen çıkış ve geçiş fonksiyonlarının karmaşıklık seviyesinin arttırılması için Trivium içerisine sonlu bir küme içerisinden ek girdiler(genelde kaotik davranış sergileyen) dahil edilebilir. Ek girdiler, herhangi bir sözde rasgele sayı üretecinin kriptografik açıdan daha güvenilir hale dönüştürür. Üreteci hibrit bir yapıya dönüştüren ek girdiler kullanılarak çıkışta elde edilen sayıların istatistiksel kaliteleri arttırılabilir. Ayrıca

çekirdek iç durumların veya tohum değerin bilinmesi durumunda üreteç çıkışlarının yüksek doğrulukla tahmin edilmesinin önüne de geçilebilir.

İkinci aşamada algoritmanın çalıştırılmasıyla elde edilen yüksek yoğunluktaki rastgele görünümlü anahtar dizilerinin, rasgelelikleri üzerindeki sapmalar ölçülerek kriptografik uygulamalar için uygunlukları incelenmiştir. Bu amaçla NIST 800-22 yazılım süitinde elde edilen anahtar dizilerinin her bir kriter için α ve P-değer parametreleri dikkate alınmıştır. Kriptolojik uygulamalar için önem seviyesine karşılık gelen tipik α parametresi [0.001–0.01], başarım parametresi P-değeri ise [0-1] aralığında değişim göstermektedir. P-değer parametresinin 0 değeri için anahtar dizisinin rasgeliğinden söz edilemezken, 1 değeri için ise anahtar dizisinin mükemmel rasgeleliğe sahip olduğu kabul edilir. Anahtar dizilerinin kriptolojik uygunluğu için P-değer parametresinin daima α parametresinden büyük olması gerek şarttır, fakat yeter şart değildir.

Zira rasgele sayılar için rasgelelik kavramının, sayı dizisinin boyutunun artışına bağlı olarak matematiksel olarak kanıtlanması oldukça zordur. Bu nedenle rasgele sayılar için kriptografik yeterliliğin odak noktası hipotez tabanlı istatistiksel analize dayalı yaklaşımlar olup bu amaçla geliştirilmiş pek çok test yazılımı bulunmaktadır. Fakat testlerde kaçınılmaz hata payının bulunabileceği ve bu nedenle sonuçların test gruplarına göre değişebileceği unutulmamalıdır. Herhangi bir test grubu için kriterlerin tümünün başarımı sayı dizisinin kriptografik uygunluğu için bazen yeterli olmayabilir. Bu doğrultuda üreteçler veya rasgele sayı dizileri hakkında yorum yapmak için test sonuçları dikkatli incelenmeli ve mümkünse geçerliliği yüksek olan testler kullanılmalıdır.

İki farklı tohum değere karşılık elde edilen anahtar dizileri için ölçüm sonuçları Tablo 4 ve Tablo 5' te verilmiştir. Değişen tohum değerleriyle birlikte elde edilen anahtar dizilerinin her bir kriter için ölçülen rasgelelik değerlerinin değiştiği görülmüştür. Bu haliyle her bir kriter için elde edilen test sonuçlarına ait P-değer başarım parametresinin α parametresinden büyük olduğu ve kriptolojik uygulamalar için gerekli rasgelelik kalitesinin sağlandığı görülmektedir. Her iki anahtar değer için test sonuçlarına bakıldığında başarım parametrelerinin kimi kriterler için düşük olduğu bu durumun gerek algoritmanın deterministik yapısından gerekse tohum değerin rasgelelik anlamında olasılıksal dağılımından kaynaklanmaktadır. Ayrıca deneme yanılma yoluyla girdi parametresi olarak kullanılan bazı tohum anahtar dizileri için çıkışta rasgeleliğin sağlanamadığı görülmüştür. Tohum anahtar değerlerin mutlak gizliliğinin yanı sıra rastgele dağılıma sahip olmaları durumu sonuçların ve algoritmanın başarım oranını arttırır.

7. Kaynaklar

1. Özkaynak, F. (2015). Kriptolojik Rasgele Sayı Üreteçleri. *Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi*, **8**-2.

2. Garcıa-Bosque, M., Perez, A., Sanchez-Azqueta, C., Celma, S. (2016). MEMS-Based Seed Generator Applied to a Chaotic Stream Cipher. *Proc. SPIE Microtechnologies*, Barcelona.

3. Koç, Ç. (2009) Cryptographic Engineering, Springer-Verlag.

4. Çakır, B. (2012). Akış Şifreleme Algoritmalarının Karşılaştırmalı Olarak İncelenmesi. Yüksek Lisans Tezi, Hacettepe Üniversitesi Fen Bilimleri Enstitüsü.

5. Sakallı, M. T., Buluş, E., Şahin, E., Büyüksaraçoğlu, F. (2007). Akış Şifrelerinde Tasarım Teknikleri ve Güç İncelemesi. *Akademik Bilişim Sempozyumu*, Kütahya.

6. Gürünlü, B. (2017). Trivium Chiper(Trivium Dizi Şifreleme Algoritması. Erişim: http://bilgisayarkavramlari.sadievrenseker.com/2009/06/10/trivium-cipher-trivium-dizi-sifreleme-algoritmasi/.

7. Kitsos, P., Sklavos, N., Provelengios, G., Skodras, A. N. (2012). FPGA-based Performance Analysis of Stream Chipers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0. *Microprocessors and Microsystems*.

8. Avaroğlu, E., Tuncer, T., Ozer, A.B., Türk, M. (2014). A New Method for Hybrid Pseudo Random Number Generator. J Micro- Electron Electron Compon Mater **44**: 303-311.

9. Kaplan, T. (2006). Trivium Dizi Şifreleme Algoritmasının FPGA Üzerinde Gerçeklenmesi. Bitirme Tezi, Elektrik-Elektronik Mühendisliği İstanbul Teknik Üniversitesi.

10. Agren, M., Hell, M., Johansson, T., Meier, W. (2017). A New Version of Grain-128 with Authentication. *In ECRYPT Symmetric Key Encryption Workshop 2011*, Available at http://skew2011.mat.dtu.dk/.

11. Babbage, S., Dodd, M. (2017).The Stream Cipher MICKEY 2.0, ECRYPT Stream Cipher,Available at http://www.ecrypt.eu.org/stream/p3ciphers/ mickey/mickey_p3.pdf

12. Bulens, P., Kalach, K., Standaert, F., Quisquater, J. (2007). FPGA Implementations of eSTREAM Phase-2 Focus Candidates with Hardware Profile. *Technical Report* 2007/024, *ECRYPT eSTREAM*.

13. Schaumont, P. R. (2010). Trivium Crypto-Coprocessor. *A Practical Introduction to Hardware/Software Codesign*. pp 337-368.

14. Mukherjee, P. (2013). *An Overview of eSTREAM Ciphers*, Centre of Excellence in Cryptology Indian Statistical Institute, Kolkata, India.

15. Yun, J., Park, K., Shin, Y., Kim, H. (2017). An efficient stream cipher for resistive RAM, *IEICE Electronics Express*, Vol.**14**, No.7, 1–6 (DOI: 10.1587/elex.14.20170179).

16. Cannièr, C. D., Preneel, B. (2008). The Stream Cipher Trivium, eSTREAM, the ECRYPT Stream Project Available at: http://www.ecrypt.eu.org/stream.[trivium_p3], 2017.

17. Paar, C., Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*, Springer.

18. Cannièr, C. D., Preneel, B. (2006). Trivium – A Stream Cipher Construction Inspired by Block Cipher Design Principles. ECRYPT Stream Cipher, Available at :http://www.ecrypt.eu.org/stream/papersdir/2006/021. pdf, 2017.

19. Sarıtaş, E., Karataş, S. (2015). *Her Yönüyle FPGA ve VHDL*. 1. Basım, Palme Yayıncılık Ankara.