EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

BÜYÜK ÖLÇEKLİ RASTSAL VE ASAL SAYI ÜRETİMİ

Enis KARAASLAN

Uluslararası Bilgisayar Anabilim Dalı

Bilim Dalı Kodu : 619.03.03

Sunuş Tarihi : 25 Temmuz 2001

Tez Danışmanı : Doç. Dr. Mehmet Emin DALKILIÇ

Bornova -İZMİR

Sayın **Enis KARAASLAN** tarafından YÜKSEK LİSANS TEZİ olarak sunulan **"Büyük Ölçekli Rastsal ve Asal Sayı Üretimi"** adlı bu çalışma, "Lisansüstü Eğitim ve Öğretim Yönetmeliği"nin 12 inci madde (c) ve (d) bentleri ve Enstitü yönergesinin ilgili hükümleri dikkate alınarak tarafımızdan değerlendirilmiş olup yapılan sözlü savunma sınavında da oy ..birliği..... ile başarılı bulunmuştur. Bu nedenlerle Enis KARAASLAN'ın sunduğu metnin yüksek lisans tezi olarak kabulüne oy ..birliği.. ile karar verilmiştir.

25 Temmuz 2001

Jüri Başkanı :Doç.Dr. Mehmet Emin Dalkılıç İmza .....

Raportör :Yrd.Doç.Dr. Aziz Can Yücetürk İmza .....

Üye :Prof.Dr. Mehmet Cudi Okur İmza .....

Bu tezin kabulü, Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 30/7/01... gün ve .....31/33..... sayılı kararı ile onaylanmıştır.

Dr. Süleyman BORUZANLI
Enstitü Sekreteri

Prof. Dr. Alaeddin TAYSUN
Enstitü Müdürü

ÖZET

BÜYÜK ÖLÇEKLİ RASTSAL VE ASAL SAYI ÜRETİMİ

KARAASLAN, Enis

Yüksek Lisans Tezi, Uluslararası Bilgisayar Ensitüsü

Tez Yöneticisi: Doç. Dr. Mehmet Emin Dalkılıç

Temmuz 2001, 185 sayfa

Bilimin her dalında, özellikle de ağ güvenliği protokolleri, simulasyon ve kriptoloji uygulamalarında yoğun olarak rastsal sayı ve asal sayı üreteçleri kullanılmaktadır. Üretilen sayıların yetersiz veya kusurlu olması, yapılan uygulamanın başarısız olmasına yol açabilmektedir.

Bu çalışmada ilk olarak detaylı bir literatür taraması ile büyük rastsal ve asal sayı üretimi konusunda teorik ve pratik bilgiler derlenmiş ve sınıflandırılarak sunulmuştur. Ardından UNIX işletim sisteminden veri toplanıp rastsal tohum oluşturulmuş ve Blum Blum & Shub üreteci kullanılarak rastsal bit dizileri elde edilmiştir. ENT, FIPS 140-2 ve NIST STS rastsallık test bataryaları kullanılarak test edilen bazı yaygın yazılım paketlerindeki rastsal sayı üreteçlerinde bazı zayıflıklar tespit edilmiştir. GAP ortamında büyük basamaklı rastsal sayılar alınarak bu sayılara asallık testleri uygulanmış; *ufak asallara bölme* ve *bölme yerine toplama* metodlarıyla asal sayıları daha hızlı bulma yöntemleri irdelenmiştir. Ayrıca herhangi bir asallık testinin ilk 246.683 adet Carmichael sayısını elemesi için gereken taban adedi belirlenmiştir. Son olarak, iki asal sayının çarpımından oluşması gereken bir n değerinin eğer sayılardan biri asal değilse çok daha kolay çarpanlarına ayrılabildiği gösterilmiştir.

Anahtar kelimeler: Rastsal Sayı, Asal Sayı, Rastsal Sayı Üreteçleri, Rastsal Bit Üreteçleri, Rastsallık Testleri, Asallık Testleri, GAP, ENT, FIPS 140-2, NIST STS

ABSTRACT

LARGE RANDOM & PRIME NUMBER GENERATION

KARAASLAN, Enis

MSc, International Computing Institute

Supervisor: Associate Professor Dr. M. E. Dalkılıç

July 2001, 185 pages

Random and prime number generators are vastly used in many branches of science, especially in network security protocols, simulation and cryptology. If the generated numbers are insufficient or faulty, this could lead to the failure of the application.

In this study, first a detailed literature search has been carried out, theorical and practical information about long random and prime numbers are collected and classified. Then the random seed is obtained by collecting data from the UNIX operating system and Blum Blum & Shub generator is employed to produce random bit sequences. The random number generators in some commonly used software packages are tested under the ENT, FIPS 140-2 ve NIST STS test programs and some weaknesses are detected. Large random numbers are put into primality tests in the GAP environment; *division by small prime*s and *addition instead of divison* methods are implemented for finding primes more quickly. Also the base number needed to eleminate the first 246.683 Carmichael numbers is given for each primality test. Lastly, it is shown that the n value which must be a product of two prime numbers is factored easily when one of them is not a prime.

Keywords: Random Number, Prime Number, Random Number Generators, Random Bit Generators, Randomness Tests, Primality Tests, GAP, ENT, FIPS 140-2, NIST STS

TEŞEKKÜR

Tez çalışması boyunca bana yardımcı olan sayın hocam Doç Dr. Mehmet Emin Dalkılıç'a, danışmanlığı ve akademik bir çalışma ortamı sağladığı için çok teşekkür ederim.

Bu tez çalışması süresince her zaman bana destek olan ve beni moralman destekleyen başta Gökhan Dalkılıç ve Oguz Kalaycı olmak üzere değerli arkadaşlarıma da teşekkürü bir borç bilirim.

Tez çalışmamda yurtiçi ve yurtdışından birçok kişiyle irtibata geçtim. Buradan GAP grubundan Steve Linton ve Stefan Kohl'e; NIST'den Andrew L. Rukhin'e ve benle diğer yazışan bütün insanlara minnettarlığımı belirtmek istiyorum.

Tabii ki bütün hayatım boyunca her zaman yanımda olan sevgili ailemin destekleri olmasaydı bunu asla başaramazdım.

Bu çalışmamı hayatta en büyük hazinem olan sevgili annem ve babama adıyorum ...

İÇİNDEKİLER

İÇİNDEKİLER (Devam)

İÇİNDEKİLER (Devam)

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

ÇİZELGELER (Devam)

1 GİRİŞ

Kriptografik uygulamalarda "anahtar" olarak kullanılmak üzere çok büyük / çok uzun asal sayılara ihtiyaç duyulmaktadır. Şifrelemede kullanılacak anahtarların rastsal (*random*) yani tahmin edilemez olması gerekmektedir. Rastsal sayılar denilince, "belirli bir dağılıma sahip, birbirinden bağımsız bir dizi sayı" ifade edilmektedir. Bu da kabaca şu demektir: sayılar eşit ihtimalle (*equally probable*) elde edilir ve dizideki diğer sayılara bağlı değildir (Knuth, 1998). Deterministik olan bilgisayar ortamında bunu gerçekleştirirken çeşitli esaslara dikkat edilmelidir.

Yapılan çalışmayı dört ana başlıkta toplayabiliriz :

1. **Rastsal Bit Üretimi :** Bilgisayar ortamında çeşitli formüller kullanarak belli bir örüntüye (*pattern*) sahip olmayan bir dizi sözde rastsal (*pseudo-random*) sayı üretilir. Bu işlemlerde kullanılan başlangıç değeri olan tohum (*seed*), her sayı dizisi üretildikten sonra değiştirilir (Caryl, 2000).

   Rastsal sayı üretmek için çeşitli fiziksel donanım ve özel çipler kullanılabilir fakat bu tez çalışmasında bir bilgisayarda var olan ekipmanla çalışmak esas olarak alınmıştır. Alınacak temel kaynaklardan en önemlisi sistemi kullanan kullanıcılardır. Kullanıcılardan bilgi toplamak zaman alabilir ama saldırganın kullanıcı davranışlarını tahmin etmesi pek mümkün değildir. Donanımsal kaynaklara örnek olarak saat bilgisi, sabit disk'teki hava tribulansı, mikrofon (beyaz gürültü - *white noise*), video, ağ (*network*) bilgileri, bilgisayara özgü bellek istatistikleri ... vb verilebilir (Callas,1996).

2. **Rastsallık Testleri:** Bir sayı dizisinin yeteri kadar rastsal olup olmadığını tespit etmek için bazı istatistiksel testler yapmak gerekir. Bir çok istatistiksel test bulunmaktadır. Bu çalışmada daha çok, yararlılığı ispatlanmış ve bilgisayarda uygulanabilir olan testler ele alınmıştır. İki tür test bulunur :

- *Deneysel - Amprik (Emprical) Testler :* Bilgisayarın, diziden sayı gruplarını işlettiği ve belirli istatistikleri değerlendirdiği testlerdir.
- *Teorik – Kuramsal (Theoretical) Testler :* Rastsal Sayı Üreteçlerini, matematiksel teoremler bazında inceleyen ve bütün periyodu için özellikler çıkarmaya çalışan testlerdir. Belli bir matematiksel formülle üretilen sayı dizileri için geçerlidir (Knuth, 1998).

3. **Üretilen Rastsal Kaynaktan (tohumdan) Asal Sayı Üretimi:** Birden büyük, sadece kendisine ve bire bölünen tam sayılara asal sayı denir. RSA[1]'da olduğu gibi, gizli asal sayılara dayanan anahtar üretme süreçlerinde kullanılmaktadır (RSA, 1998). Pratikte asal sayı üretmek için :
   - n-bit rastsal sayı p üretilir.
   - İlk (*high order*) bit ve son (*low order*) bit'ler 1 olacak şekilde ayarlanır (Son bit'in 1 olması o sayının tek olmasını sağlamakta, ilk bit'in 1 olması da asal sayının istenilen (*required*) uzunlukta olduğunu belirlemektedir).
   - p değerinin ufak asal sayılara (3,5,7,11 ...) bölünüp bölünmediği kontrol edilir.
   - p değeri herhangi bir asallık testine bir rastsal a değeri kullanılarak sokulur. Eğer p değeri bu testi geçerse başka rastsal a değerleri için bu test tekrarlanabilir. Eğer p değeri bu testlerden birisini geçemezse asal bir sayı değildir ve başka bir p değeri yaratılıp aynı işlemlerden geçirilmesi gerekecektir (Schneier, 1996; Seth, 1999).
   - Farklı rastsallık testleri uygulanarak daha güvenilir sonuçlar elde edilebilir (Silverman, 1997).

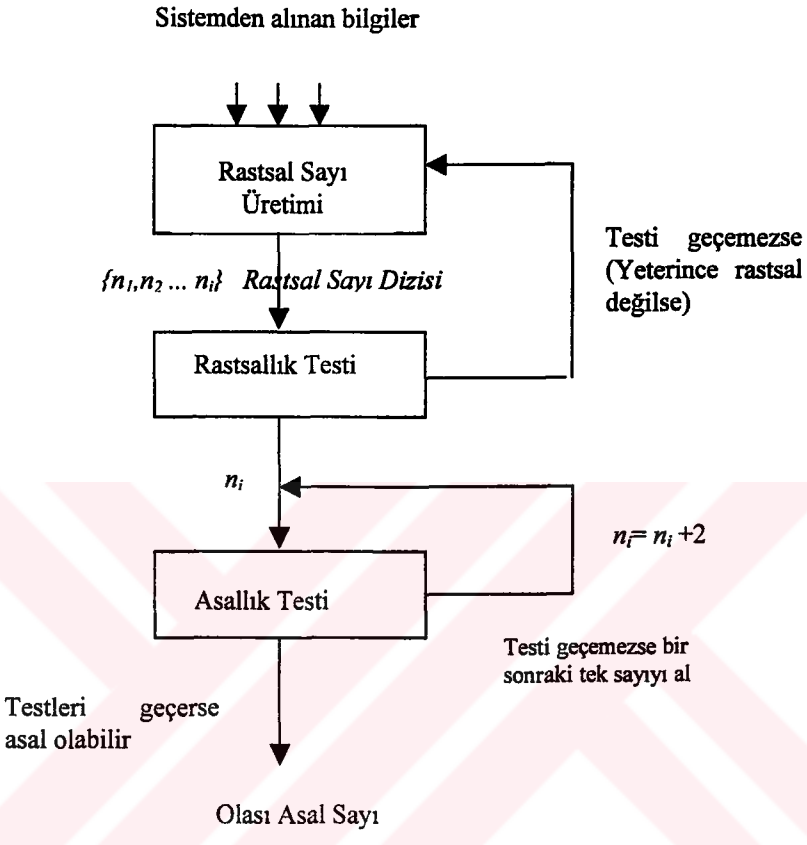
RSA algoritması, "public key cryptography" gereksimlerini başarıyla gerçekleştiren ve günümüzde de sıkça kullanılan bir algoritmadır ve bu algoritmayı geliştiren üç kişinin (Rivest, Shamir, Adleman) adlarının kısaltmalarıyla anılmaktadır (Tanenbaum, 1996).

Bu konuda kullanılabilcek farklı yöntemler tez çalışmasında detaylı olarak belirtilmiştir.

4. **Asallık Testleri:** Asal sayıları bileşik (*composite*) sayılardan ayırt etmek aritmetikte en temel ve önemli konulardan birisi olmuştur. Asallık tanımından yola çıkarsak, bir n tamsayısının asal olması için 2 ile $\sqrt{n}$ arasında hiç bir böleni olmaması gerekir. Bu işlemi ufak sayılar için yapmak mümkün olsa da sayıların büyüklüğü arttıkça bunun hesaplanması mümkün olmamaktadır. Büyük sayıların asal olup olmadıklarını anlamak için daha gelişmiş asallık testleri gerekmektedir (Granville, 1992). Asallık testlerini iki ana başlıkta toplayabiliriz :

   - *Kesin (deterministik) asallık testleri:* Bu tür asallık testleriyle bir sayının asal olup olmadığını kesin olarak belirlemek mümkündür. Bu tür yöntemler genelikle çarpanlara ayırma yöntemine dayanmaktadır.
   - *Olası (probabilistic) asallık testleri:* Bir sayının yüksek olasılıkla asal olduğunu kanıtlama sürecidir. Deterministik yöntemlere göre daha hızlı olduğu ve gerçekte bir sayının asal olup olmadığını ufak hata payları ile kanıtlayabildiği için Olası Asallık Testlerinin kullanılması önerilmektedir. 2^{-100} 'den daha düşük bir hata payı ile bir sayının asal olduğu belirlenebilir (RSA, 1998).

Tez çalışmasında yapılan uygulamanın şeması Şekil 1.1'de gösterilmiştir.

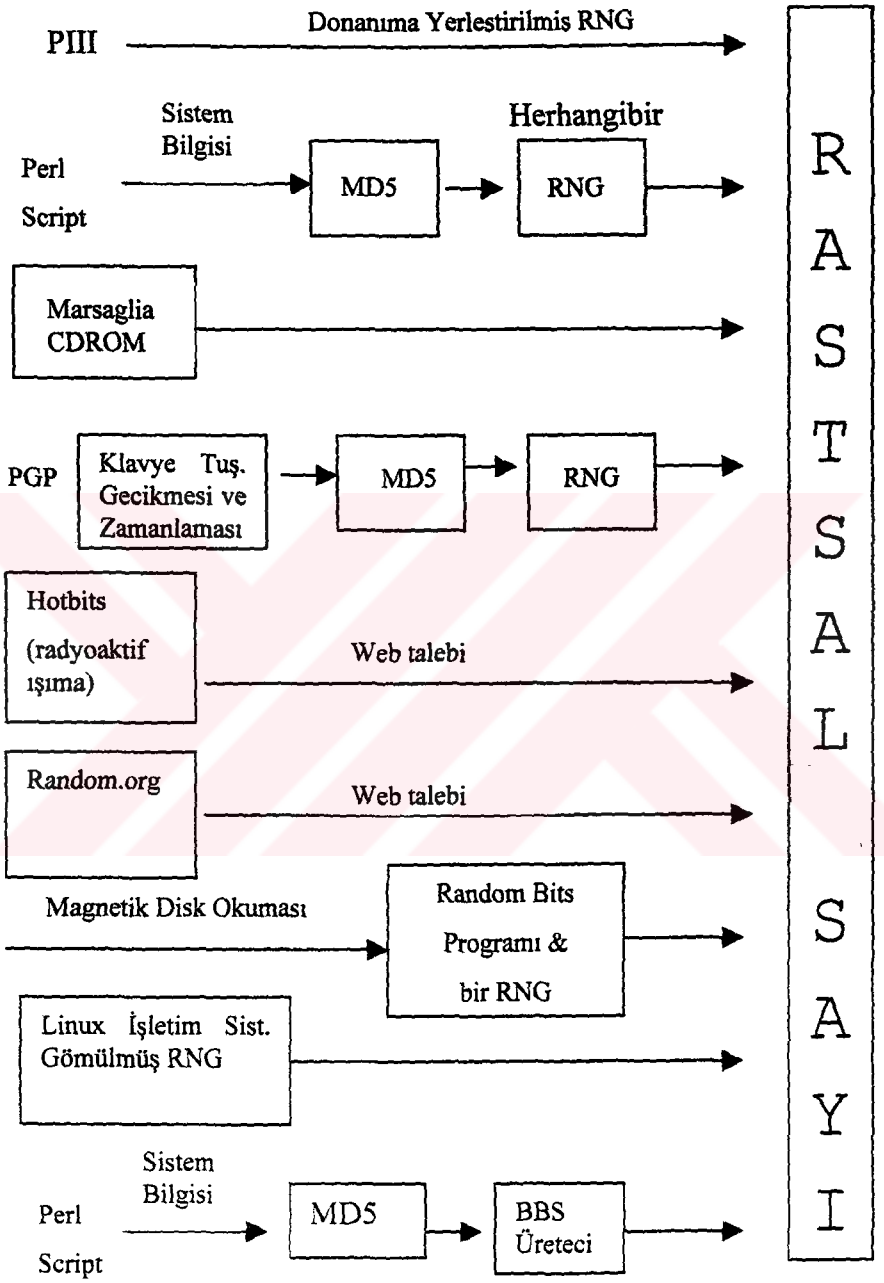


Şekil 1.1 : Uygulamanın Şeması

Tez çalışmasında sistemlerden ve kullanıcılardan rastsal bilgiler toplamak, toplanan bilgilerden rastsal bit dizileri üretmek ve bu üretilen verilerden asal sayı oluşturmak konusunda detaylı teorik bilgiler de sunulmuştur.

Uygulamalar UNIX işletim sisteminde yapılmıştır. Uygulamaların çoğu GAP ortamında kodlanmıştır, ayrıca C ve Perl programlama dilleri de kullanılmıştır. Uygulamalar örnek verilerle ve doğru çalıştığı standartlarla belirlenmiş hazır uygulamalarla kontrol edilmiştir.

Rastsal sayı üretmek için kullanılabilecek bazı rastsal sayı üreteçleri (random number generator: RNG) Şekil 1.2'de gösterilmiştir.



Şekil 1.2 : Bazı Rastsal Sayı Üreteçleri

2 RASTSAL SAYI ÜRETİMİ

Genellikle rastsal sayılar denilince, "belirli bir dağılıma sahip, birbirinden bağımsız bir dizi sayı" ifade edilmektedir. Bu da kabaca şu demektir: sayılar eşit ihtimalle elde edilir ve dizideki diğer sayılara bağlı değildir. Uniform dağılımlarda her sayının gelme ihtimalinin birbirine eşit olduğu unutulmamalıdır (Knuth, 1998).

Bilgisayarla rastsal sayılar üretmek normalde mümkün değildir; üretilen sayılar sözde rastsal (*pseudo-random*) olarak adlandırılır. Çeşitli formüller kullanarak belli bir örüntüye (*pattern*) sahip olmayan bir dizi sözde rastsal sayı üretilir. Bu işlemler bir tohum (*seed*) ile başlatılır ve bu sayı, her sayı dizisi üretildikten sonra değiştirilir (Caryl, 2000). Sözde rastsal sayılar, uygun bir yöntem seçilerek yaratıldığı takdirde bir çok uygulamada iyi sonuçlar vermektedir (Knuth, 1998).

Rastsal sayıları elde etmekte kullanılan tohum; zaman ve tarihe, kullanıcının yazdığı herhangi bir yazıdan üretilen bir sayıya, kullanıcının o anda kullandığı fare göstergecinin (*mouse pointer*) hareket değişikliğine, ...vb. bağlı üretilirse üretilen sayılarda bir örüntü (*pattern*) gözlenmez. Bu da üretecin tekrar başlatıldığında, daha sağlıklı ve tekrar etmeyen sayılar üretmesini sağlar.

2.1 Rastsal Sayıların Uygulama Alanları

Rastsal Sayılarla uğraşan bilim dalları olarak aşağıdakileri saymak mümkündür (Moreau, 1997):

- **Matematik** : Rastsallığın matematiksel temelleri üzerine çalışmalar yapılmaktadır.
- **Bilgisayar Bilimi:** Bilgisayar dilleriyle yazılan programlarla rastsal-sayı üretimi konusunda çalışmalar yapılmaktadır.
- **Fizik ve Finans:** Simulasyon amaçlı rastsal sayılarla uğraşılmaktadır.
- **Şifreleme:** Düz yazıyı karıştırılmış bit'ler haline getirmek için kullanılan yöntemlerde karıştırma fonksiyonları ve anahtar üretiminde rastsal sayılar önemli rol oynamaktadır.

Rastsal sayıların kullanıldığı belli başlı uygulamalar aşağıdaki gibidir[2] :

- **Simulasyon** : Doğal olayların simulasyonunu yaparken, olayları daha gerçekçi olarak ifade etmek ve uygulamak için rastsal sayılar kullanmamız gerekir. Bu tür uygulamalar ve istatistik bilimi sayesinde olayları ve sonuçlarını tahmin etmemiz mümkün olmaktadır.
- **Örnekleme (*Sampling*)** : Bir çok durumda bütün bilgileri toplamak mümkün değildir. Rastsal olarak alınan bir örnekle gerçek ortamı ifade etmemiz, istatistik bilimi sayesinde tahminlerde bulunmamız ve "tipik davranışı" ifade etmemiz mümkün olabilmektedir.
- **Sayısal Analiz (*Numerical Analysis*)** : Çözülmesi güç sayısal problemleri çözmek için, rastsal sayıları kullanan teknikler üretilmiştir.
- **Bilgisayar Programlama (*Computer Programming*)** : Algoritmaların sağlıklı ve etkili çalışıp çalışmadıklarını kontrol etmek için rastsal sayılar kullanılabilmektedir. Ayrıca rastsal çalışan algoritmalar (*randomized algorithms*), deterministik olanlara göre çok daha performanslı ve etkin çalışabilmektedir.
- **Karar Verme (Decision Making)** : Tarafsız bir seçim yapmak bazen kritik önemde olabilir. Verilen kararın, önceki kararlara bağlı olmaması ve tahmin edilemez olması önemlidir.
- **Estetik (*Esthetics*)** : Bilgisayarla yapılan grafik ve müzikte bazı örüntülerin rastsal olarak dağıtılması, üretilen ürünün çok daha gerçekçi olmasını sağlamaktadır.
- **Eğlence (*Recreation*)** : Zar atmaktan gazino oyunlarına, hatta bilgisayar oyunlarına kadar eğlence amaçlı bu oyunlarda rastsallık önemli bir faktör olmaktadır. Hatta bu yüzden, rastsal sayıları içeren algoritmalar *Monte Carlo Metodu* olarak tanımlanmaktadır.

---

[2] Şifreleme hariç diğer uygulamaların açıklamaları için kaynak olarak (Knuth, 1998) alınmıştır

- **Şifreleme (*Computer Security Applications*) :** RSA gibi birçok şifreleme algoritmaları temelde rastsal seçilen bir tohuma dayanmaktadır. Seçilen bu sayının tahmin edilemez olması şifreleme algoritmasının gücünü yani kırılmasının güç olmasını sağlayan önemli bir unsurdur.

2.2 Rastsal Sayıları Üretme Yöntemleri

Bu yöntemleri kabaca üç ana kategoriye sokmak mümkündür :

- Donanımsal Çözümler
- Yazılımsal Çözümler
- Karma (Bileşik) Çözümler

Donanımsal Çözümler :

Gerçek rastsal sayılar üretmek için mutlaka mekanik yöntemlere başvurmak gerekmektedir. Mekanik aksamların kendi kaotik çalışma üsluplarıyla tahmin edilemez sayılar üretmek mümkündür. Kullanılabilecek yöntemlerin çeşitliliği hayal gücüne bağlıdır. Piyango çekilişi gibi uygulamalar bu yöntemle gerçekleştirilmelidir.

Yazılımsal Çözümler :

Geçmiş yıllarda, rastsal sayı üretiminde mekanik yöntemlerin yetersiz kalması yüzünden bilgisayarın matematik işlemleriyle gelişen bilgisayar teknolojisini etkin kullanma yoluna gidildi (Knuth, 1998). Her ne kadar günümüzde mekanik yöntemlerin sorunlarının çoğu giderilmişse de yazılımsal çözümler de etkin çözümler sunmaktadır. Yazılımsal çözümlerin ek donanıma gereksinimi olmaması, taşınabilirlik, esneklik gibi birçok artıları bulunmaktadır.

Bileşik Çözümler :

Donanımsal ve yazılımsal çözümlerin bir araya geldiği bu yöntemler, bize çok güçlü sistemler sunmaktadır. Örneğin, 1995 yılında, Marsaglia[3] Noise-diode devresinin çıkışını deterministik olarak karıştırılmış rap müziği ile birleştirerek 650 megabyte rastsal veri oluşturmuş ve bir CD içinde dağıtımını sağlamıştır. Bu şekilde, milyarlarca iyi test edilmiş rastsal byte içeren veri dosyalarını veya CD 'leri Internetten rahatlıkla temin etmek mümkündür. Bu rastsal sayılar herhangi bir yazılım tarafından rahatlıkla kullanılabilir.

Bu tez çalışmasında rastsal sayılar yazılım yöntemi ile oluşturulmaktadır. Bunun nedenleri aşağıdaki gibidir:

- Yazılım yöntemi ile üzerinde çalışan işletim sistemine daha az bağımlı ve ekstra donanım gerektirmeyen çözümler geliştirilebilmesi,
- Simulasyon amaçlı kullanılacak ekonomik çözümler üretilebilmesi,
- Kullanılan sisteme özgü uyarlamalar yapılabilmesi,
- İstendiğinde üzerinde yapılacak değişikliklerle sistem kolayca ve daha az harcamayla geliştirilebilmesi,
- Akademik bir çalışma olarak sonra yapılacak çalışmalara bir temel oluşturabilmesi,
- Yazılım yönteminde de yazılacak rutinlerle donanımsal kaynaklardan tohum alarak gerçek rastsal sayı üretebilmesidir.

---

[3] George Marsaglia : "Diehard" (Ek 3) adıyla anılan bir dizi rastsallık testinin yazarıdır.

2.3 Donanım Tabanlı Rastsal Sayı Üretimi

Gerçek rastsal sayılar üretmek için mutlaka mekanik yöntemlere başvurmak gerekmektedir. Üretilen sayıların kritik önem taşıdığı şifreleme veya piyango gibi sistemlerde donanımsal çözümlerin kullanılması tercih edilmektedir.

Mekanik çalışan herhangi bir alet çeşitli rastsal (kontrol edilemeyen) etkilere maruz kalacağından kaotik bir ortam yaratacaktır. Bu da gerçek rastsal sayı kaynağı demektir.

Kullanılabilecek yöntemlerin çeşitliliği hayal gücü ile sınırlıdır. Bir ısı kaynaksal (*termal*) gürültü veya radyoaktif ışıma kaynağı, bağımsız çalışan bir osilatör gibi donanımsal kaynaklar veya bilgisayarın içerisinde bulunan bazı mekanik aksamlar ve analog kaynaklar buna örnek verilebilir. Birçok firma, PC veya dizüstü bilgisayarların COM portlarına bağlanarak çalışan ve rastsal sayı üretme işlemlerini yerine getiren ufak aparatlar geliştirmektedir.

Donanımsal yöntemlerin artıları ve güvenlik uygulamalarındaki tercih nedenlerinin neler olduğu (Eastlake ve ark., 1994) 'te ayrıntılı olarak incelenmiştir.

2.3.1 Bilgisayarda Mevcut Donanımlarla

Bilgisayarla birlikte gelen donanımlar kullanılarak gerçek rastsal sayılar yaratılabilir. Bunlara örnek olarak aşağıdakiler verilebilir (Eastlake ve ark, 1994) :

- **Ses / Video Girdisi :** Mikrofondan ses, kameradan video gibi analog girdiler sayısallaştırılarak (*digitization*) yüksek kalitede rastsal bit'ler oluşturulabilir. Özellikle de mikrofon kapalıyken sayısallaştırıcıdan alınan veya lens kapağı kapalı olan bir kameradan alınan girdi gerçekte termal gürültü olacaktır ve bu da bizim için gerçek rastsal sayı kaynağıdır. Mesela ses kartı bulunan tipik bir Unix makinasında "*cat /dev/audio | compress*" komutu random-bit'lerden oluşan bir dosya oluşturacaktır.
- **Mevcut Disk Sürücülerin Kullanımı :** Kaotik hava türbülansları sonucunda disk sürücülerin dönme hızlarında rastsal değişmeler olmaktadır. Bu değişikliklerin gözlenmesi ve

Fourier dönüşümünden geçirilmesi sonucunda disk sürücüler dakikada en az 100 bit gerçek rastsal sayı üretebilmektedirler.

2.3.2 Mevcut Bir Uygulama : Lavarand Projesi

Rastsal Sayı üretmek için geliştirilen ilginç yöntemlerden birisi de "Random Lava" tekniğidir. Silicon Graphics firmasından bir takımın geliştirdiği bu yöntemde Lava Lite[4] 'lar kullanılmaktadır. Lava Lite, kendi başına kaotik bir sistemdir. Bir Lava Lite'in oluşturduğu örüntüler zaman içinde hızla değişmektedir. Lava kabarcıklarını tahmin etmek veya belirli bir şekilde modellemek imkansızdır.

Tasarlanan sistem şu şekilde çalışmaktadır : Bir iş istasyonuna (*workstation*) bağlı bir video kamera ile 6 adet değişik renkte ışık veren Lava Lite gözlemlenmekte ve belirli aralıklarla resimleri çekilmektedir. Yakalanan görüntüler sayısal hale getirilmekte ve mükemmel olmayan bu sayısallaştırma işlemi, resme daha fazla rastsallık katmaktadır.

Daha kullanılabilir hale getirmek için, 900 KB'lık renkli resim, 140 byte'lık tohum sayıya özel bir hash fonksiyonu kullanarak dönüştürülmektedir. Bu 140 byte'lık sayı Blum Blum Shub rastsal-sayı üreteci için tohum olarak kullanılmakta ve böylece tahmin edilemeyen (*unpredictable*) bır tohumla ve iyi bir rastsal sayı üreteciyle bu sistem gerçek rastsal sayı üretebilmektedir (Boling, 1997).

Lavarand projesi ile ilgili daha fazla bilgi için http://lavarand.sgi.com internet adresine başvurulabilir.

---

[4] Lava Lite, 1960'larda popüler olan ve halen de süs eşyası olarak kullanılan, renkli baloncuklar oluşturarak ışık veren lambalara verilen addır.

2.4 Yazılım Tabanlı Rastsal Sayı Üretimi

Doğru yapıldığı takdirde, yazılım yöntemiyle rastsal sayı üretimi iyi sonuçlar verebilir. Ulaşılan sonuç kriptografik sistemler için bile yeterli güvenliği sağlayabilir. Bunun için iki kritik unsur vardır :

1. Üreteç için tohum seçimi,
2. İyi bir PRNG (*Pseudo-random number generator*) – rastsal sayı üreteci kullanmak.

Her PRNG'nin bir tohum ile başlatılması gerekir. Bu tohumun kötü seçilmesi halinde üreteç de sağlıklı sonuçlar vermeyecektir. Tohum seçimi ile ilgili detaylar bölüm 2.5'de belirtilmiştir.

2.4.1 Sözde Rastsal Sayı Üreteçleri

Birçok programlama dili kendi içinde bir PRNG bulundurmaktadır. C programlama dilindeki *rand()* fonksiyonu buna örnek verilebilir. Programlama dilleriyle birlikte gelen bu fonksiyonların birçoğu yeterli performansı gösterememektedirler. Eğer PRNG kısa bir periyoda sahipse, az sayıda üretimden sonra kendisini tekrar etmeye başlayacaktır. Bu da şu tür sonuçlara yol açabilir; simulasyonumuz daha az gerçekçi olur veya kriptolojide kullanıyorsak saldırganın sistemi kırmak için denemesi gereken anahtar sayısı azalır. Ayrıca saldırgan, sayıların dağılımı ve örüntü (*pattern*) hakkında bazı fikirlere sahipse denemesi gereken anahtar sayısı daha da azalacaktır. Programlama dilleriyle gelen PRNG'leri kullanmaktansa, yüksek rastsallık (*randomness*) derecesine sahip olduğu onaylanmış bir PRNG kullanılması tercih edilmelidir.

Rastsal Sayı Üreteçlerini aşağıdaki gibi sınıflara ayırmak mümkündür (NHSE, 1996):

- Doğrusal Benzerlik (*Linear Congruential)* Üreteçleri -LCG
- Diğer Benzerlik Üreteçleri
- Geri Fibonacci (*Lagged Fibonnaci)* Üreteçleri
- Kayan Yazmaç (*Shift Register*) Üreteçleri
- Bileşik (*Combined)* Üreteçler
- Diğer Algoritmalar

- Non-Uniform Dağılımlar (*Distributions*)

Bu üreteçlerin çoğu yüksek performanslı bilgisayarlarda rastsal sayılar üretmek için kullanılmaktadır. Bu üreteçlerin yazılımları NHSE[5]'in bir parçası olduklarından belirli Internet sitelerinden temin edilebilirler (NHSE, 1996).

Bu tez çalışmasında, LCG ve BBS üreteçleri üzerinde yoğunlaşılmıştır.

Doğrusal Benzerlik Üreteçleri (LCG)[6]:

Birçok programlama dili ve ortam bu popüler rastsal sayı üretecini kullanmaktadır. Bu tür üreteçler kırılabilir (*breakable*) olmalarına rağmen basit ve hızlı oldukları için tercih edilmektedirler (Schneier, 1996). Bu üreteçleri kullanmak için 4 tam sayının belirlenmesi gerekir. Çizelge 2.1 'de bu 4 parametre belirtilmiştir.

Çizelge 2.1 : Doğrusal Benzerlik Üreteci Değişken Değerleri

Değişken	Açıklama	Değer Aralığı
m	Mod Değeri	$0 < m$
a	Çarpan	$0 \le a < m$
c	Artış Değeri	$0 \le c < m$
X_0	Başlangıç Değeri	$0 < X_0 < m$

Rastsal sayı dizisini oluşturacak X_n ise Formül 2.1 ile hesaplanmaktadır. Bu üreteçler belirli bir seferden sonra döngüye girecek ve aynı diziyi tekrar üreteceklerdir. Bu tekrarlanan diziye *periyod* denir ve dizinin olabildiğince uzun periyodu olmasına dikkat edilmelidir.

$$x_n = (a * x_{n-1} + c) \bmod m \quad \text{(Formül 2.1)}$$

---

[5] NHSE : National HPCC Software Exchange

[6] Aksi belirtilmediği sürece kaynak olarak (Knuth, 1998) alınmıştır

Elde edilecek dizinin rastsal olabilmesi için m, a, c ve X_0 değerlerinin dikkatlice seçilmesi gerekecektir. Örneğin, mod değeri olan m değeri olabildiğince büyük seçilmelidir çünkü m değişik değer mümkün olduğundan, periyod m elemandan daha fazla olamayacaktır. Bununla beraber üreteç hızının da dikkate alınması gerekeceğinden m değeri için çok büyük değerler de alınmamalıdır. İstenen bir m uzunluğunda periyoda sahip üreteçler geliştirmek için Formül 2.1'deki parametreler aşağıda belirtilen özelliklerde olmalıdır:

1. c ile m sayıları birbirlerine göre asal olmalı,
2. $b = a - 1$ olmak üzere b değeri, m değerini bölen her p değeri için p 'nin katı olmalı,
3. m sayısı 4'ün katı ise, b sayısı da 4'ün katı olmalıdır.

c parametresine 0 değeri verilerek sayı üretimi hızlandırılabilir fakat c değeri 0 'dan farklı alındığında üretecin daha uzun periyodlara sahip olacağı da bir gerçektir. Doğrusal sayı üreteçleri, c için 0 değeri alındığında *Çarpımsal Benzerlik Yöntemi* (*Multiplicative Congruential Method*), 0 dışında c değerleri alındığında *Karışık Benzerlik Yöntemi* (*Mixed Congruential Method*) olarak da adlandırılmaktadır.

Bu üreteçler her m dönüşte tekrar ettiklerinden simulasyon uygulamaları için yeteri kadar iyi değildir (Caryl, 2000). Farklı doğrusal üreteçler birleştirilerek daha uzun periyodlara sahip ve bazı rastsallık testlerinde daha iyi sonuçlar verebilen diziler elde etmek mümkündür. Fakat bu üreteçler kriptografik olarak daha güvenli (*secure*) değildir (Schneier, 1996).

2.4.2 Entropi

Entropiye kısaca belirsizlik ölçüsü diyebiliriz. Bir PRNG, yüksek derecede entropiye sahip olmalıdır. Entropi aynı zamanda rastsallığın ölçütüdür.

Her zaman aynı çıktıyı sağlayan sistemlerde her bit sabit bir değere eşleniktir ve belirsizlik yoktur. Bu tür sistemlerde her bit için "0" entropi bulunmaktadır. Eğer her çıktı için herhangi bir değerin (0/1) gelmesi olasılığı, bir başka değerin (1/0) gelme olasılığına eşitse (gerçek

rastsal) o zaman tam belirsizlik vardır. Bu tür sistemlerde ise her bit için tam "1" entropi bulunmaktadır (Matthews, 1995). Rastsal sayı dizisinde istatistiksel olarak; "0"ın bulunma sıklığı "1"'in bulunma sıklığı kadardır. "00"ın bulunma sıklığı "11" kadar ve tek "0"ın bulunma sıklığının yarısı kadar olmalıdır. Üç sıfırın veya üç birin olma sıklığı da çift sıfırın yarısı kadar olmalıdır. Bu oran benzer şekilde devam etmektedir. Çift çift örnekler aldığımızda ve sonuçları çizime döktüğümüzde grafikte yığılmalar görülmemelidir. Chi-Square gibi çeşitli istatistiksel testlerle bir sayı dizisinin ne kadar rastsal olduğu anlaşılabilir.

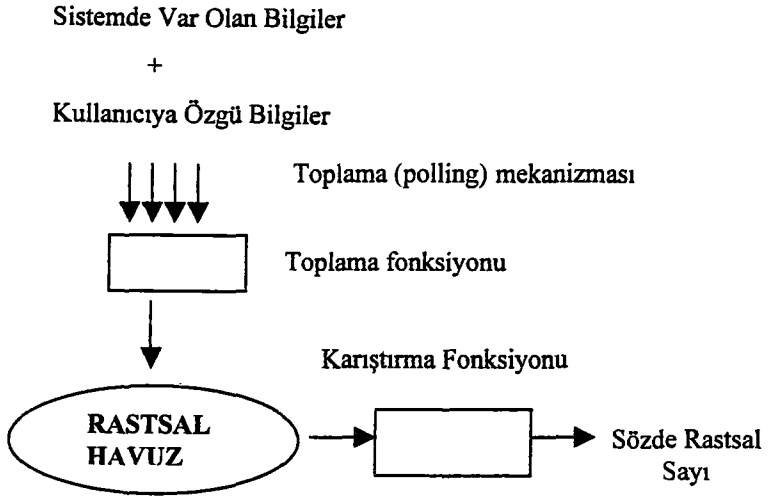
Eğer iyi bir rastsal sayı üreteci kullanmayı istiyorsak yüksek derecede entropiye sahip olması gerekir ki bu da çıktının tahmin edilemez (*unpredictable*) olmasını sağlayacaktır. Gerçek Rastsal Sayı Üreteçleri, genellikle donanımsal çözümler, en yüksek entropiye sahip olacaktır.

2.5 Rastsal Havuza Dayanan Sistemler[7]

"Rastsal havuz"a dayanan sistemlerde rastsal sayı oluşturma sistemi Şekil 2.1'de gösterildiği gibidir. *Toplama fonksiyonu* sistemde var olan bilgileri ve kullananın sağladığı kendine özgü bilgileri toplayarak rastsal havuza yazar. *Karıştırma fonksiyonu* (PRNG de diyebiliriz) ise rastsal havuzdan sözde rastsal (*pseudo random*) sayı çıkartır. Karıştırma fonksiyonu bir "hash" fonksiyonu veya "Triple-DES" gibi kriptografik bir fonksiyon olabilir. Bu fonksiyonun sadece sistemden büyük miktarlarda veri çekildiği zaman kullanılması gerekecektir. Aksi taktirde az kullanımda zaten rastsal olmayan ve tahmin edilebilir veriler iptal edildiği ve tahmin edilemez veriler üretildiği için bu karıştırma fonksiyonuna gerek yoktur ve sistemin geriye kalan kısmı yeterlidir.

---

[7] Bu bölümdeki açıklamalar için aksi belirtilmediği sürece kaynak olarak (Gutmann, 1999) alınmıştır



Şekil 2.1 : Rastsal Bit Üretme Sisteminin Çalışma Mantığı

Rastsal Havuz (*Randomness Pool*) ve karıştırma fonksiyonunun tasarlanırken dikkat edilmesi gereken bazı hususlar bulunmaktadır. Üreteç mümkün olduğu kadar fazla kaynaktan veri toplamalıdır. Üretecin değişik platformlarda kullanılacağı dikkate alınırsa donanıma özgü veya işletim sistemine özgü kaynaklar da ayrıca dikkate alınmalıdır. Özellikle güvenlik uygulamalarında kullanılacak üretecin sahip olması beklenen özellikler aşağıda sıralanmıştır:

1. Girdi bilgilerinin analizine karşı dirençli olmalıdır. Yani girdi bilgilerinin bir kısmı bilinse bile üretecin o anki durumu öğrenilememelidir.
2. Üreteç, girdi verilerindeki kasti oynamalara karşı dirençli olmalıdır. Hiçbir kimse kasti olarak üretece seçilmiş girdi besleyip sistemin durumunu etkileyememelidir.
3. Çıktı verilerinin yani üretilen rastsal sayının analizine dirençli olmalıdır. Çıktı verilerinin bir kısmı bilindiğinde önceki ve sonraki çıktılara dair bilgi çıkarılamamalıdır.
4. Üreteç, kendi iç (*internal*) durumunu da korumalıdır ki sistem bilgilerinin analizinden kendi iç durumu öğrenilmemelidir.

5. Üreteç, doğrudan (*explicit*) rastsal havuzu karıştırmalı veya kodun uyumunu (*conformance*) sağlamak için veri çekmelidir.

2.5.1 Rastsallığın Damıtılması (*Distill*)[8]

Donanımsal çözümlerin kullanıl(a)madığı güvenlik uygulamalarında birçok kaynaktan toplanan rastsal girdinin güçlü bir *karıştırma fonksiyonu* ile karıştırılması tavsiye edilmektedir. Bu işleme *rastsallığı damıtmak* denmektedir. Böyle bir fonksiyon toplanan verilerdeki rastsallığı koruyacaktır. Bu yöntem aynı zamanda donanımsal çözümlerde bir sorun olduğunda da tercih edilebilir (Eastlake ve ark, 1994). Belli başlı karıştırma fonksiyonları aşağıda sıralanmıştır :

- **XOR**: Karıştırma fonksiyonlarının en basit ve hızlılarındandır. Dezavantajı, rastsal veriler kendi aralarında bağlantılı (*correlated*) ise bu XOR yapılan dizilerdeki rastsallığı kaldıracaktır.
- **Kriptografik fonksiyonlar**: Mesela bir örneğin (*sample*) ilk 8 byte'ı alınıp DES kullanılarak diğer örneklerle tekrarlanarak kullanıldığında 56 bit'e kadar rastsallık elde edilecektir. Bu yöntemin dezavantajı ise çok yavaş olmasıdır.
- **Checksum (CRC) fonksiyonları:** Bu fonksiyonların doğru olanı seçildiğinde bir çözüm olabilir ama genellikle hata düzeltmek için tasarlandıklarından istenilen performansı veremeyeceklerdir.
- **Hash fonksiyonları:** "Message Digest" veya "Cryptographic Checksum" olarak adlandırılan hash fonksiyonlarının ise birçok yararlı özellikleri vardır. Sayısal imza, bütünlük kontrolü (*integrity check*) ve benzeri işlerde kullanılırlar. Ron Rivest'in belirttiği gibi "Aynı değere hash edilmiş iki mesajı hesaplayarak (*computationaly*) bulmak mümkün değildir. Herhangi bir

[8] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Callas, 1996) alınmıştır.

saldırı 'brute force[9]' saldırıdan daha etkin değildir." Bu da belirlenen rastsallık tanımına uymaktadır.

2.5.2 "Hash" Fonksiyonları[10]

Hash fonksiyonları, değişken büyüklükte mesaj alıp sabit büyüklükte parmak izi veya "*message digest*" üreten algoritmalardır. "Hash", üzerinde çalıştırıldığı örneklerde bulunan entropiyi korumaktadır. Birçok örnekten toplanan entropi, bütün örneklerin entropi toplamına eşittir. "Hash" fonksiyonlarının entropiyi topladığı unutulmamalıdır, yani kendi büyüklüklerinden daha fazla entropi bulunduramazlar. Bu nitelikler hash fonksiyonlarını rastsal sayı üretmek için mükemmel fonksiyonlar haline getirmektedir. Belli başlı hash fonksiyonları Çizelge 2.2'de tanıtılmıştır.

[9] Brute force, akla gelebilecek bütün değerleri deneyerek yapılan bir saldırı şeklidir.

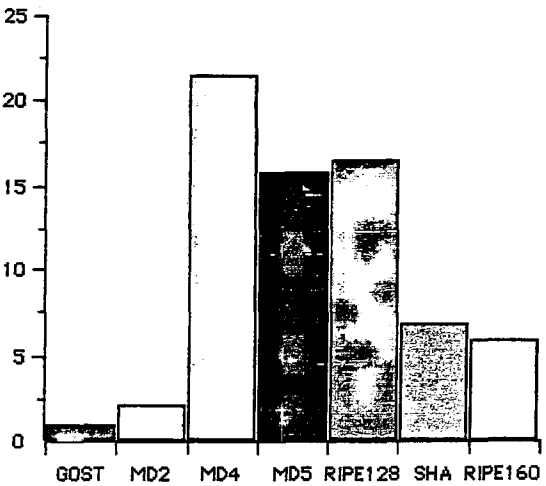
[10] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Callas, 1996) alınmıştır.

Çizelge 2.2: Belli Başlı "Hash" Fonksiyonları

Adı	Niteliği
MD2, MD4, MD5	Ron Rivest tarafından geliştirilmiş olan 128 bitlik fonksiyonlardır. MD2'nin zayıf noktaları yoktur ama diğerlerine nispeten daha yavaştır. Bunun temel nedeni 16 bitlik işlemlerin kullanılmış olmasıdır. 1992 yılında Den Boer, MD4'un son iki round'unda zayıflıklar bulunduğunu ve saldırıya açık olduğunu ispatlayınca Ron Rivest daha güçlü ama nispeten yavaş olan MD5'i geliştirdi (Kaufman ve ark, 1995). MD5 en son versiyondur ve sayısal imzada kullanılmaktadır. Kırıldığı ve zayıf noktalarının olduğu iddia edilmektedir.
SHA (SHS)	NIST[11] 'de MD5 baz alınarak geliştirilmiştir. Amerika hükümetinin önerdiği 168 bitlik standarttır.
GOST	256 bitlik Rus "Hash" fonksiyonu. MD2 'dan daha yavaştır.
RIPE-MD, RIPE-MD-160	Avrupa standardı. RIPE-MD kırıldığı için RIPE-MD-160 geliştirilmiştir.

MD2, MD4 ve MD5'in çalışma şekillerini detaylı olarak incelemek için (Kauffman ve ark, 1995)'e bakılabilir. Burada tanıtılan fonksiyonların hızlarının GOST fonksiyonunun hızı birim alınarak birbirine oranlı hız grafiği Şekil 2.2'deki gibidir. Hız bilgileri Bruce Schneier ve Wai Dai'den alınmıştır (Callas, 1996).

[11] NIST - National Institute of Standards and Technology - Amerikan Ulusal Standartlar ve Teknoloji Enstitüsü



Şekil 2.2 : Hash Fonksiyonlarının Hızlarının Karşılaştırılması

Uygulamalarda MD5 veya SHA'nın kullanılması önerilmektedir. MD5'ın zayıflıkları olduğu bilinmektedir ama bu fonksiyonun rastsallığı damıtmak için kullanılacağı unutulmamalıdır. SHA ise nispeten daha yavaştır ama daha fazla entropi üretilebileceği için tercih edilebilir.

"Hash" fonksiyonları kullanmanın avantajları aşağıda belirtilmiştir:

1. "Hash" edilen verilerin sırası değiştirildiğinde farklı sonuçlar ortaya çıkar. Bu da şu demektir : x ile y'nin hash'i, y ile x'inkine eşit değildir.
2. Entropisi az olan veya entropisi olmayan verileri bile hash'e sokmak sonucu değiştirebilir. Bir çok verinin hash'i alındığında sonuç daha yararlı olacaktır.
3. Değişik boyuttaki gözlemlerden sabit büyüklükte rastsal veri üretilir.

Sonuç olarak, hash edilecek iyi kaynaklar bulunmalı ve bir süreç geliştirilerek düzenli olarak damıtılmalıdır. Buradan üretilecek çıktı bir rastsal sayı üretecine tohum olarak verilmelidir.

2.6 Tohum (*seed*) Üretimi

Her PRNG'nin bir tohum ile başlatılması gerekir. Bu tohumun kötü seçilmesi halinde üreteç de sağlıklı sonuçlar vermeyecektir. Bir PRNG aynı tohum değeri (ilk değer X_0) ile başlatılırsa her zaman aynı çıktıyı verecektir. Birçok uygulama için tohumun tam rastsal olması gerekmez, tahmin edilemez olması yeterlidir. Rastsal sayı iki açıdan değerlendirilebilir:

1. **Miktar (*Quantity*)** : İstenen veri büyüklüğü önemli bir kriterdir. Yeterli miktarda veri sağlanırsa üretilecek tohumun tahmini neredeyse imkansız hale gelecektir.
2. **Kalite** : Kalite entropiye bağlıdır.

2.6.1 Tohum Kaynakları[12]

Tohumu birçok şekilde temin etmek mümkündür. Çizelge 2.3'de kullanılabilecek bazı potansiyel kaynak materyalleri listelenmiştir.

[12] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Matthews, 1995) alınmıştır.

Çizelge 2.3: Tohum için kullanılabilecek kaynakların listesi

Daha az entropi

Tekil (o sisteme özgü) Olaylar	• Konfigürasyon Dosyaları • Sürücü (*Drive*) Konfigürasyonu • Çevre Dizgileri (*Environment Strings*)
Değişken ve Tahmin Edilemez Olaylar	• Bilgisayar Ekranı İçeriği • Tarih ve Zaman • Yüksek çözünürlükte saat örneklemleri • Klavyede son basılan tuş • Log dosyası blokları • Ağ istatistikleri • Süreç istatistikleri • Diğer süreç veya threadler için program sayacı
Harici Rastsal Olaylar	• Zamanla birlikte imleç konumu • Klavyeye basma zamanlaması • Mikrofon Girdisi • Fareye klikleme zamanlaması • Fare imlecinin hareket bilgisi • Bellek istatistikleri • Video Girdisi

Daha çok entropi

Daha çok entropinin olması, oluşturulacak tohumun daha rastsal olmasını sağlayacağından tercih edilen bir seçenektir ama sisteme getireceği yük de dikkate alınmalıdır. Günümüzde güçlü bilgisayar sistemlerinin ekonomik fiyatlarla temin edilebildiği düşünülürse, sisteme gelecek yük de çok önemli bir kısıt olmamaktadır. O zaman izlenmesi gereken tutum olabildiğince fazla "Harici Rastsal Olaylar" grubundan veri toplamaktır (Bkz. Çizelge 2.3). Eğer daha fazla byte gerekiyorsa

toplanan verilere "Tekil Olaylar" ve "Değişken ve Tahmin Edilemez Olaylar" gruplarından toplanan veriler de eklenebilir (Bkz. Çizelge 2.3).

Miktar konusunda dikkat edilmesi gereken husus ise toplanan bütün bitlerin rastsal olup olmadığıdır. Mesela klavyede basılan tuşların analizi yapıldığında her byte'da sadece bir bit'in rastsal olduğu anlaşılmıştır. Bu yüzden gereken her bit için bir byte'lık veri toplanması ve PRNG'ye beslenerek ilk rastsal çıktının sağlanması daha yerinde olacaktır.

Alınacak temel kaynaklardan biri sistemi kullanan kullanıcılardır. Kullanıcılardan bilgi toplamak zaman alabilir ama saldırganın kullanıcı davranışlarını tahmin etmesi pek mümkün değildir. Klavye ve fare hareketleri tahmin edilmesi güç davranışlara birer örnek olarak verilebilir (Callas,1996).

Sistem çalışırken belirli aralıklarla tohum güncellenmelidir. Arka planda çalışan bir uygulamanın kullanıcı davranışını yakalayarak kaynak-sayıyı değiştirmesi tercih edilebilir. Belirli aralıklarla, yeni gelen bilgilerle eski tohumun karıştırılarak yeni tohum oluşturulması daha iyi sonuçlar verecektir.

Bir rastsal-sayı üretecini kullanmadan önce, mümkün olduğu kadar fazla entropi toplanmalı ve bu veri üretecin tohumu olarak kullanılmalıdır. Bir anahtar (*key*) çifti üretilecekse en az anahtar büyüklüğünün beşte veya onda biri kadar entropi bulundurulmalıdır. Mesela 1024 bit uzunluğunda bir anahtar üretilecekse, üreteç tohumunda en az 100 veya 200 bit entropi bulundurulmalıdır (Callas, 1996).

2.6.2 İşletim Sistemlerinden Bilgi Toplamak[13]

Tohum için bilgi toplayacağımız işletim sistemlerini bellek ve süreç (*process*) yönetimlerine göre *az gelişmiş* ve *gelişmiş* sistemler olarak sınıflayabiliriz. Az gelişmiş sistemlere örnek olarak : Windows 3.1, MAC-OS, OS/2; gelişmiş sistemlere örnek olarak Windows 95/98, Windows NT, UNIX ve türevlerini saymak mümkündür.

---

[13] Bu bölümdeki açıklamalar için kaynak olarak (Gutmann, 1999) alınmıştır

Az Gelişmiş Sistemler:

Sınırlı miktarda işe yarar bilgi edinebilen, az gelişmiş sistemlerdir. Bellek korumasının az veya hiç olmadığı, arka planda görev (*task*) veya thread çalıştırmanın zor veya mümkün olmadığı işletim sistemleridir. Bu tür işletim sistemlerinde bilgi edinilirken sistemin durması gerekebilecektir. Bu işletim sistemlerine örnek olarak aşağıdakiler verilebilir (Gutmann, 1999):

- Windows 3.1 : ToolHelp kütüphanesi aracılığıyla bütün sistem ve süreç (process) veri yapılarına ulaşılabilir.
- MAC : Windows 3.1'in sağladığı kadar fazla olmasa da yeterli bilgiye ulaşılabilir.
- OS/2 : Arka planda çalışan süreçlerle bilgi çekmek mümkün olmasına rağmen yeterli bilgiye ulaşılamaz

Gelişmiş Sistemler:

Bu tür işletim sistemleri, sistem ve global veri yapılarını genel erişimden korurlar ama sistem, ağ ve genel kullanım istatistiklerinden oluşan büyük miktarda veri sunarlar. En önemli özelliklerinden birisi ise arka planda bu tür verilerin toplanmasına (*background polling*) izin vermeleridir. Bunlara örnek olarak aşağıdakiler verilebilir :

- Windows 95 : ToolHelp32 fonksiyonları kullanılarak ortalama 5-15 KB sistem bilgisi toplanabilir.
- Windows NT : NetAPI32 fonksiyonlarıyla ve registry'den sistem performans istatistikleri edinme aracılığıyla ortalama 30-40 KB sistem bilgisi toplanabilir.
- UNIX / BeOS: Orta yüklü bir Unix server'da ortalama 20-40 KB sistem bilgisi toplanabilir.

İşletim sistemlerinden toplanan sistem bilgilerinin hepsinin rastsal veya kullanışlı olmadığı da dikkate alınmalıdır. Bu işletim sistemlerinden UNIX, en karmaşık olanıdır. Sistemde var olan yardımcı programlar kullanılarak sistem hakkında istatistik ve bilgilere ulaşılabilir. Bu yardımcı programlardan elde edilen çıktılar alınarak rastsallık havuzuna

eklenir. Unix'te kullanılan komutları değişik parametreler kullanarak daha etkin hale getirmek mümkündür.

Kullanılacak araçlar ve erişilecek kaynaklarda dikkat edilmesi gerekenler aşağıda belirtilmiştir:

1. Çıktı makul derecede (*reasonably*) rastsal olmalı,
2. Çıktı makul zaman aralığında ve amacımıza uygun biçimde üretmeli,
3. Makul miktarda çıktı sağlamalıdır. Mesela, 'pstat' komutunun sağladığı veriler son derecede sınırlı rastsallık içermektedir. Bu tür komutlardan kaçınılmalıdır.

2.6.3 Tohum Temin Eden Fonksiyonlar

Belli başlı tohum temin eden fonksiyonlar olarak aşağıdaki üreteçleri saymak mümkündür :

- Perl Script ile Unix sistem bilgilerinin toplanması
- Linux /dev/random
- EGD (*Entropy Gathering Deamon*)
- Truerand
- Intel'in Donanımsal tabanlı rastsal sayı üreteci
- Cryptlib
- www. random.org (ve benzer internet kaynakları)
- PGP programı
- Sabit disk veya ses kartı kullanılarak

Perl Script ile Unix Sistem Bilgilerinden

Perl dili ile yazılmış bu program (*script*) ile Unix'te var olan sistem komutları çağrılarak sistem ve ağ (*network*) durumu, sanal bellek istatistikleri ve süreç durumları gibi devamlı değişen bilgilerin bulunduğu bir dosya oluşturulmaktadır. Bu bilgiler daha sonra bir girdi olarak MD5[14] 'a verildiğinde oluşacak olan 128 bitlik sayının rastssal olduğu öngörülmektedir (Garfinkel S. ve ark., 1996).

Perl Script, Unix işletim sisteminde 16.000 byte'dan büyük ve değişken (16384 / 22814 / 24576 / 34016) sistem bilgisini alıp 128 bit'lik rastsal sayılar üretmektedir. Üretilen rastsal sayı : 128 bit = 16 byte = 32 hex basamak olmaktadır. Daha detaylı bilgi bölüm 6.1.1'de verilmiştir.

Linux /dev/random

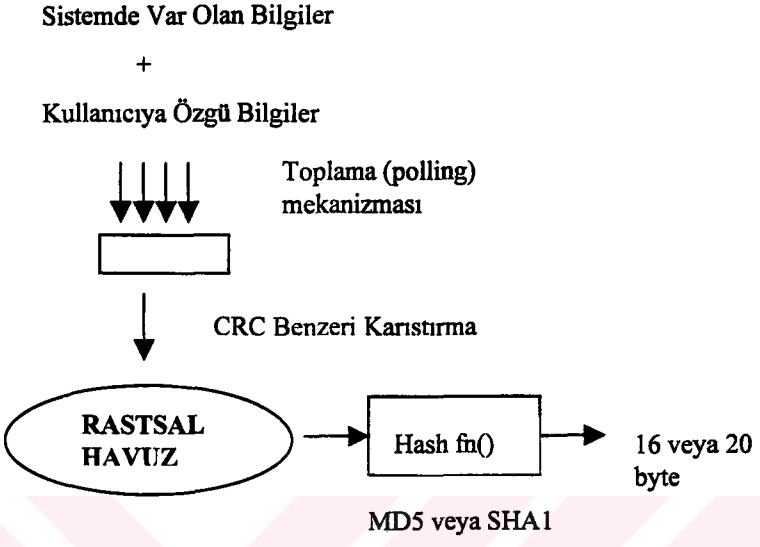
Linux işletim sisteminde bütünleşik olarak bir rastsal sayı üreteci bulunmaktadır. Theodore Y. Ts'o tarafından yazılan bu rastsal bit üretecinin kaynak dosyaları[15] da bulunmaktadır Bu üretecin çalışma biçimi Şekil 2.3'deki gibidir. Sistemde kullanılabilecek fonksiyonlar ve çalışma şekilleri aşağıda belirtilmiştir :

- /dev/random :Entropiyi tahmin eder ve o kadar bit döndürür.
- /dev/urandom :Kullanıcı tarafından talep edilen byte kadar döndüren PRNG fonksiyondur.

---

[14] MD5, değişken büyüklükte veri alıp 128-bit parmak izi veya "message digest" üreten bir algoritmadır.

[15] Kaynak kodu, "/usr/src/linux-2.2.13/drivers/char/random.c" ve "/usr/src/linux-2.2.13/include/linux/random.h" dizinlerinde bulunmaktadır. Bu dizinlerin linux işletim sisteminin sürümüne göre değişiklik gösterebileceği unutulmamalıdır



Şekil 2.3: Linux /dev/random üretecinin çalışma mantığı

EGD- Entropy Gathering Deamon

Linux dışındaki diğer Unix türevleri için, Linux sistemindeki "\dev\random" a benzer işlevde bir deamon[16] hazırlanmıştır. Bu deamon sistemde bir değişiklik olduğu zaman edindiği rastsal bitleri entropi havuzuna eklemektedir.

Truerand ( )

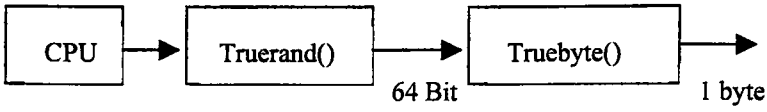
Unix sistemlerinde; işlemci saatiyle, timer interrupt[17] 'ları aralığı arasındaki değişim (*drift*) oranlarından rastsal sayı üretir. Üzerinde hiç bir yük bulunamayan işlemcilerde bile böyle bir değişim bulunmaktadır. Randbyte() interrupt başına en azından .4 bit entropi olduğunu varsayar.

---

[16] Sistem çalışırken arka planda sistemin bir parçası olarak çalışan program.

[17] Donanım veya yazılım tabanlı zamanlayıcı (timer) kesmeleri.

İşletim sistemi tarafından sunulan "scheduler" ve benzeri süreçler de entropi'ye katkıda bulunabilir. Bu üretecin çalışma biçimi Şekil 2.4'deki gibidir.



Şekil 2.4 : Truerand() Üretecinin Çalışma Mantığı

Intel'in Çözümü

Intel 810 anakartlarından itibaren ve daha geliştirilmiş şekilde PIII işlemcilerde rastsal sayı üreteci bulunmaktadır. Bu üreteç 70 Kbps entropi üretmektedir. Bu da 17 saatte yaklaşık 500 MB demektir (INTEL, 2000). Bu üretecin daha detaylı analizi (Jun ve ark, 1999)'da bulunmaktadır. Bu analize göre; bu üreteç PC'ler üzerinde bulunan en güvenilir rastsal sayı kaynağıdır. Intel, RSA'in Crypto-C yazılımının ve Microsoft'un çözümlerinin "*middleware*"[18] olarak kullanılmasını önermektedir. Bu çözümün ücretsiz (*freeware*) olarak kullanılan bir versiyonu henüz bulunmamaktadır (INTEL, 2000).

Cryptlib

Bu yazılım kütüphanesi, bir rastsal sayı üretecinden ve birçok kriptografik fonksiyondan oluşmaktadır. C fonksiyonlarından oluşan bu yazılım kütüphanesi birçok ortamda çalışabilmektedir ve rastsal sayı üretecinin birçok işletim sistemi için desteği bulunmaktadır. Üreteç www.cs.auckland.ac.nz/~pgut001/cryptlib Internet adresinden temin edilebilir. Bu üretecin özellikleri (Gutmann, 1999)'da belirtilmiştir.

[18] Bir tür program arayüzü.

Random.org

www.random.org internet adresinden gerek Java programlarından gelecek yazılım isteği ile, gerekse direkt Internet sitesinden çekme (*download*) yöntemiyle rastsal bit sağlanabilmektedir. Buradan alınan rastsal veri dosyaları bir çok testi başarıyla geçmesine rağmen "Chi Square" testini geçememektedir.

PGP Programı

PGP'nin eski versiyonlarında rastsal havuz kendi içinde bir karıştırma fonksiyonuna tabii tutulurken, son versiyonlarında kullanıcıdan sağlanan bir anahtar ile rastsal havuz karıştırılmaktadır. PGP programını :

"pgp +makerandom=Bit# dosya_adı.dosya_uzantısı"

biçiminde kullanarak istenilen boyutta rastsal veri oluşturulabilir. Örnek :

pgp +makerandom=1024 rastsal.seed

Elde edilen sonucun, PGP'nin eski versiyonlarında bir yazılım hatası (*bug*) yüzünden normalde elde edilenden daha az rastsal olduğu iddiasında bulunulmuştur.

Sabit Disk veya Ses Kartı Kullanılarak

Unix sistemlerinde sabit diskten veya ses kartından rastsal veri elde edilebilmektedir. Bu konuda yapılmış çeşitli çalışmalar bulunmaktadır. Bell laboratuvarlarının sunduğu çözümde Unix işletim sisteminin çekirdek (*kernel*) yapısında bir değişiklik yapmadan sabit diskten rastsal veri oluşturulabileceği belirtilmektedir.

2.7 Rastsal Bit Üretimi

Rastsal Sayıları, bir araya grupladığımız bir dizi rastsal bit olarak tanımlayabiliriz. Matematiksel tanımlara göre, bir karakter dizisi (*string*), onu daha kısa ifade etmemiz mümkün değilse yani sıkıştırılamıyorsa rastsaldır. O zaman *sıkıştırma fonksiyonları* da bir rastsallaştırma fonksiyonudur.

Rastsal bit üretimi için kullanılabilecek bir çok fonksiyon bulunmaktadır. En basit yöntemlerden biri zincirleme hash kullanmaktır. Bir başka yöntem ise "hash" tarafından tohumlanan Şifreleme Sistemleri (*cipher*) veya Şifreleme Sistemi türevi, mesela bir "stream cipher" fonksiyonu kullanmaktır. Seçim yaparken üreteç fonksiyonunun hızı, kalitesi ve kriptografik zayıflıkları olup olmadığı dikkate alınmalıdır.

Kriptografik olarak düşündüğümüzde saldırganın rastsal sayıları tahmin edememesi ve sadece "brute-force[19]" saldırılarla yetinmek zorunda kalması çok önemlidir. Güçlü şifre sistemleri (*strong cyphers*) da bir tür rastsallaştırma fonksiyonudur. Elde edilen şifrelenmiş metin (*cyphertext*) rastsaldır, testleri başarıyla geçer ve gerçekte gürültüye (*noise*) benzer. Burada rastsallaştırma fonksiyonunun mükemmel olması gerektiği unutulmamalıdır.

Var olan üreteçlerden, BBS (Blum Blum&Shub) üreteci bir çok özelliğiyle dikkati çekmektedir ama başlatmak (*initialize*) için 2 çok büyük asal sayının üretilmesini gerektirmesi yüzünden yavaştır. Dolayısıyla asal sayıların bilinmesi esasına dayandığından büyük miktarlarda entropi toplayan bir sistemin son adımı olamayacağı (Callas,1996)'da ileri sürülmüştür. Rastsal bit üreteçleri için iki tasarım felsefesi bulunmaktadır (Kelsey ve ark, 2000):

1. **Entropi Kuyrukları** : PGP, /dev/random, Cryptlib
   - Bütün istekler için yeterli entropi olduğu varsayılır
   - PRNG'in görevi entropiyi damıtmak ve kullanım için kuyruğa sokmaktır.
   - Yeterli entropi görülmediği zaman, işleve son verilir veya sözde rastsal (*pseudo random*) çıktı oluşturulur.

---

[19] Bütün anahtarları tek tek deneyerek yapılan saldırı şekli.

2. **Sözde Rastsal Üreteçler :** ANSI X9.17 Anahtar Üreteci, RSAREF PRNG, Yarrow

- Bir anahtar için yeterli entropi toplanır ve damıtılır.
- Bir kez anahtar elde edildikten sonra bu anahtardan sözde rastsal çıktı üretilir.

Belli başlı rastsal bit üreteçleri aşağıda sıralanmıştır:

- ANSI X9.17 Üreteci
- Blum Blum & Shub (BBS) Üreteci
- Tausworthe Üreteçleri
- Yarrow-160 Kriptografik Sayı Üreteci
- Crypto++
- Cryptlib
- Linux /dev/random

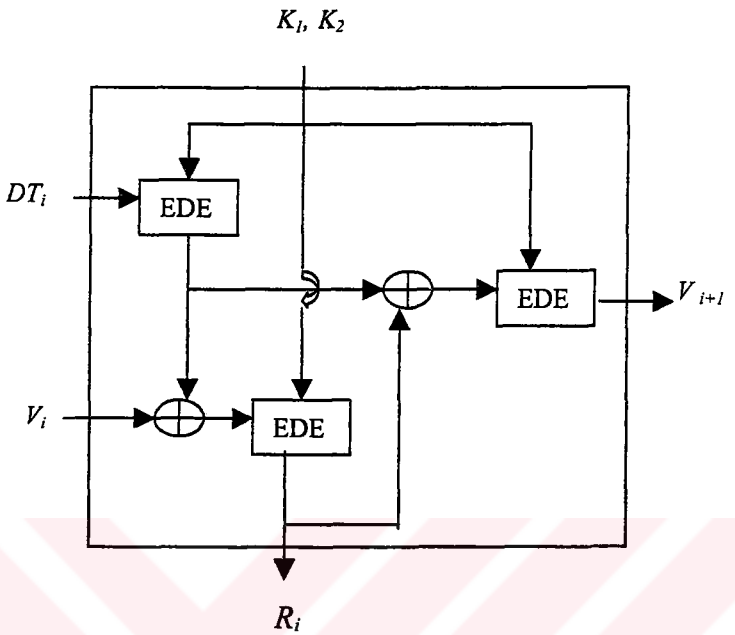
ANSI X9.17 Üreteci[20]

ANSI[21] X9.17 standardı en güçlü kriptografik sözde rastsal sayı üreteçlerinden birisini tanımlamaktadır. PGP de dahil olmak üzere birçok ticari uygulamada kullanılmaktadır. Kullanılan yöntem 112 bit anahtara ve 3 adet triple DES (9 adet DES) 'in kullanılmasına dayanmaktadır. Bu özellikler üretecin kriptografik olarak güçlü olmasını sağlamaktadır. Üretecin çalışma şekli Şekil 2.5'de gösterilmiştir. Şekilde geçen EDE, "Encryption Decryption Encryption" yani triple DES'i; ⊕ işareti ise XOR (Exclusive OR) işlemini belirtmektedir. Üreteçte kullanılan elemanlar Çizelge 2.4'de belirtilmiştir.

---

[20] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Stallings, 1998) alınmıştır

[21] ANSI – American National Standarts Institute



Şekil 2.5 : ANSI X9.17 Sözde Rastsal Sayı Üreteci

Çizelge 2.4 : ANSI X9.17 Üreteci Elemanları

Simge	**Açıklama**	**Boyut**
DT_i	*i* 'ninci aşama için geçerli tarih ve zaman değeri	64 bit
V_i	*i* 'ninci aşama için geçerli tohum değeri	64 bit
K_1, K_2	DES gizli (*secret*) anahtarları	112 bit (56 x 2)
R_i	i 'ninci aşamada üretilen sözde rastsal sayı	64 bit

Üreteç girdi olarak DT_i ve V_i değerlerini almakta ve K_1, K_2 anahtarları kullanılarak 64 bitlik R_i çıktısı elde edilmektedir.

Blum Blum & Shub Üreteci (BBSG)

Gerçekten tahmin edilemeyecek sayılar elde etmek için bu üreteç yaygın olarak kullanılmaktadır. Bu üreteç Formül 2.2'ye dayanmaktadır (Caryl, 2000). Kriptografik güvenli (cryptographically secure) olduğu kanıtlanmıştır (Stallings, 1998).

$$x_y = (x_{y-1})^2 \bmod n \qquad \text{(Formül 2.2)}$$

Bu üreteci kullanırken uyulması gereken kurallar aşağıdaki gibidir (Caryl, 2000):

1. "*x mod 4 = 3*" eşitliğini sağlayan iki adet büyük asal sayı, *p* ve *q* bulunmalıdır.
2. "$n = p * q$" eşitliğini sağlayan *n* değeri hesaplanmalıdır.
3. Öyle bir rastsal *s* sayısı seçilmelidir ki ortak bölenlerin en büyüğü, obeb(*s*, *n*) = 1 olmalıdır.
4. İlk değer "$x_o = s^2 \ mod \ n$" eşitliği ile hesaplanmalı ve sonraki değerler Formül 2.2 kullanılarak bulunmalıdır.

Üretilen sayının "*log log n*" asgari anlamlı (*least significant*) bit'i rastsaldır (Caryl, 2000). Bu üretecin tek dezavantajı, doğrusal benzerlik üreteçlerine göre çok daha yavaş olmasıdır. Üreteç B_i şeklinde ifade edeceğimiz rastsal bit dizileri üretmektedir. Algoritması aşağıda belirtildiği gibidir:

$x_o = s^2 \bmod n$

for i=1 **to** ***bit_adedi***

$x_i = (x_{i-1})^2 \bmod n$

$B_i = x_i \bmod 2$

BBSG'in güvenliği, *n*'in faktörlere ayrılmasının zor olmasına dayanmaktadır. BBSG, *kriptografik olarak güvenli sözde rastsal bit üreteci* olarak adlandırılmaktadır. Bu da "bir sonraki bit" testini geçtiğini

belirtir yani daha önceden *n* değeri bilinen üreteçten çıktı olarak *k* adet bit dizisi verildiği zaman (*k+1*)'ci bit'in ne olduğunu ½'den daha fazla olasılıkla polinom zamanda çözecek bir algoritmanın bulunmaması demektir. Bu da dizinin tahmin edilemez olduğunu belirtir (Stallings, 1998).

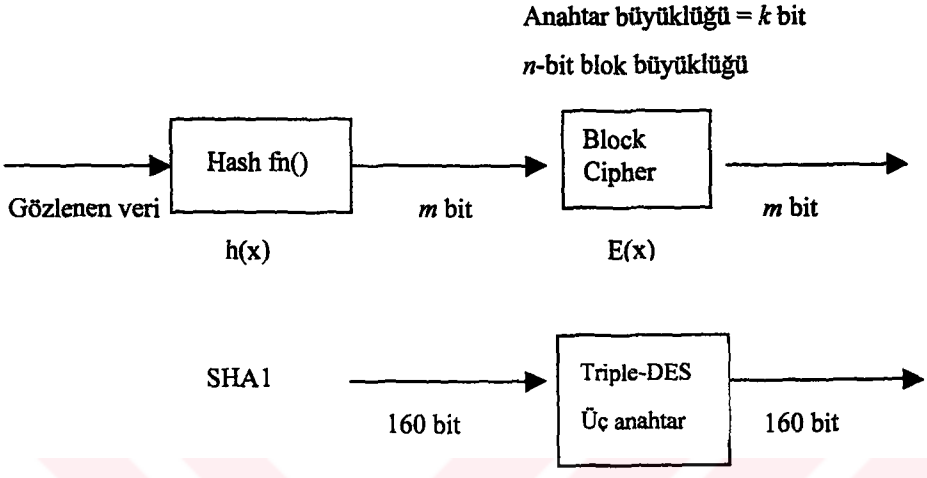
Tausworthe Üreteçleri

Alınan bir tohum verisi üzerinde bit işlemlerine dayanır. Hızlı uygulamaları bilinmektedir. (Tezuka ve ark, 1991) 'de birden fazla Tausworthe Üretecinin birlikte etkin ve hızlı kullanımı incelenmiştir. 64 bit'lik bir tohum üzerindeki bir üreteç örneği Formül 2.3'te verilmiştir (Tezuka ve ark, 1991).

$(x^{57} + x^{55} + x^{54} + x^{53} + ... + x^3 + x^2 + 1)$ (Formül 2.3)

Yarrow-160 Kriptografik Rastsal Sayı Üreteci

Bruce Schneier'in başkanı olduğu Counterpane Systems şirketi tarafından üretilen Yarrow-160 rastsal sayı üreteci, daha iyi PRNG'ler üretilmesini desteklemek için http://www.counterpane.com/yarrow.html internet adresinde program koduyla birlikte ücretsiz olarak dağıtılmaktadır. Schneier'in belirttiği üzere, üretece radyo gürültüsü, ağ (*network*) paketlerinin gelme zamanları ve disk sürücüsünün kaotik özellikleri gibi yeni rastsallık kaynakları eklenmiştir ve birkaç kaynak kapatılsa bile çalışmaktadır. Yarrow-160, hash fonksiyonu olarak SHA1 ve Block Cipher olarak Triple-DES kullanmaktadır ve çalışma biçimi Şekil 2.6'de belirtilmiştir. Bu üreteç, *min(m,k)* bit gücündedir (Kelsey ve ark, 1999).



Şekil 2.6 : Yarrow-160 Üretecinin Çalışma Mantığı

Crypto++ 4.1

Wai Dai tarafından hazırlanan ve kriptografik bir çok kodu içinde bulunduran bir C++ yazılım kütüphanesi paketidir. Bu yazılım kütüphanesi paketi ve dökümantasyonu www.cryptopp.com veya www.eskimo.com/~weidai/cryptlib.html internet adresinden bedava olarak temin edilebilir.

Cryptlib

Bu yazılım kütüphanesi, bir rastsal sayı üretecinden ve birçok kriptografik fonksiyondan oluşmaktadır. Daha detaylı bilgi için bakınız bölüm 2.6.3.

Linux /dev/random

Bu üreteç tohum üretmek için kullanılabileceği gibi rastsal bit üretmek için de kullanılabilir. Daha detaylı bilgi için bakınız bölüm 2.6.3.

3 RASTSALLIK TESTLERİ

Bir sayı dizisinin (*sequence*) periyodunu, pratik uygulamalar için tekrar etmeyecek bir şekilde yeteri kadar uzun yapmak mümkündür. Ama bu dizinin yeteri kadar rastsal olup olmadığını (belirli bir güven aralığında) tespit etmek için bazı istatistiksel testler yapmak gerekir.

Bir dizinin T_1, T_2, ... T_n testlerine göre rastsal olduğu söylenebilir. Ama bir sonraki T_{n+1}'inci testde bu dizinin rastsal olmadığı da gösterilebilir. Tabii ki her uygulanan test, o dizinin rastsallığı hakkında bize daha fazla güven vermektedir (Knuth, 1998). Pratikte, test bataryası denen bağımsız (farklı rastsallıktan sapmaları hedefleyen) bir grup istatistiksel test üreteç çıktısına uygulanır ve eğer üreteç tüm testleri geçerse rastsal kabul edilir. Her dizi dikkatlice test edilmelidir.

İki tür test bulunur (Knuth, 1998):

1. **Deneysel - Amprik (*Emprical*) Testler :** Bilgisayarın, diziden sayı gruplarını işlettiği ve belirli (*certain*) istatistikleri değerlendirdiği testlerdir.

2. **Teorik – Kuramsal (*Theoretical*) Testler :** Diziyi oluşturmak için kullanılan ve "reoccurence" kuralına dayanan sayı-teorik (*number-theoretic*) yöntemlerini kullanarak dizinin karakteristiğini kurduğumuz testlerdir. Başka bir deyişle: Rastsal Sayı Üreteçlerini, matematiksel teoremler bazında inceleyen ve bütün periyodu için özellikler çıkarmaya çalışan testlere denir. Belli bir matematiksel formülle üretilen sayı dizileri için geçerlidir. Doğrusal Benzerlik Üreteçleri (LCG) için geçerli olan bazı teoremler mevcuttur

Spectral Testi, Doğrusal Benzerlik Üreteçleri (LCG)'nin kalitesini ölçmek için kullanılan en önemli testlerden biridir. Bir bakımdan Kuramsal Testlere benzer çünkü dizinin bütün periyodunun özellikleri ile ilgilenir. Sonuçları belirlemek için bir bilgisayar programına ihtiyaç duymasından dolayı da Ampirik Testlere benzer.

3.1 Temel İstatistiksel Test Yöntemleri[22]

Bu bölümde istatistiksel testlerin temelini oluşturan Chi-Square (χ^2) ve Kolmogorov-Smirnov Testine değinilmiştir.

3.1.1 "Chi-Square" Testleri

Chi-Square (χ^2) testi en çok bilinen istatistiksel testlerden biridir. Diğer testlerle de birlikte kullanılmaktadır.

Her gözlemin, k tane kategoriden birine düştüğünü farzedelim. n tane bağımsız gözlem alırız. p_s her gözlemin s kategorisine düşme olasılığı olsun. Y_s de gerçekleştirdiğimiz gözlemde s kategorisine düşen gerçek sayı olsun. İstatistiksel olarak Formül 3.1 sonucunda elde edilen v sayısı dizinin rastsal olup olmadığı hakkında bize bilgi verecektir :

$$v = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s} \qquad \text{(Formül 3.1)}$$

Formülün açılımı ve sadeleştirilmesi sonunda elde edilen, bilgisayar ortamında daha kolay hesaplanabilecek hali Formül 3.2'deki gibidir:

$$v = \frac{1}{n}\sum_{s=1}^{k}\left(\frac{Y_s^2}{p_s}\right) - n \qquad \text{(Formül 3.2)}$$

Elde edilen v sayısı, Çizelge 3.1'deki veriler ışığında incelenerek kabul edilebilir bir değer olup olmadığına karar verilir. Çizelge 3.1, v serbestlik derecesinden (*degrees of freedom*) chi-square dağılımının değerlerini vermektedir. v değeri kategori sayısından bir azdır ($v = k\text{-}1$).

---

[22] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Knuth, 1998) alınmıştır.

Çizelge 3.1: Chi Square Dağılımı Değerleri

	P=1%	P=5%	P=25%	P=50%	P=75%	P=95%	P=99%
v=1	0.00016	0.00393	0.1015	0.4549	1.323	3.841	6.635
v=2	0.02010	0.1026	0.5754	1.386	2.773	5.991	9.210
				. . .			
v=50	14.95	18.49	24.48	29.34	34.80	43.77	50.89
v>30	$v+\sqrt{2v}x_p+\frac{2}{3}x_p^2-\frac{2}{3}+O(1/\sqrt{v})$						
xp=	-2.33	-1.64	-6.74	0.0	0.674	1.64	2.33

Çizelge 3.1'de v satırında ve p sütunu altındaki tablo değeri x olsun. v değerinin x'den az veya eşit olma olasılığı p'dir. Tabii ki bunun için yeterli sayıda örnek alınmalı yani n sayısı yeteri kadar büyük olmalıdır. n sayısı, np_s değerini en azından 5 veya daha büyük bir sayı olmasını sağlayacak şekilde seçilmelidir. Tabii ki sayı ne kadar büyük seçilirse test o kadar sağlıklı olacaktır. Sonuç olarak n sayısı çok büyük seçilmelidir.

Eğer v değeri tablo değerlerine göre %1 den küçük veya %99 dan büyükse, bu sayı dizisinin *yeteri kadar rastsal olmadığı* kararına varılır ve dizi kabul edilmez. Eğer v değeri tablo değerlerine göre %1 - %5 veya %95 - %99 aralığındaysa bu sayı dizisi *şüpheli* (suspect) , %5 - %10 veya %90 - %95 aralığındaysa bu sayı dizisi *neredeyse (almost) şüphelidir*.

Chi-Square testi en azından 3 kez değişik veri kümeleriyle yapılır ve 3 testin en azından 2 sindeki sonuç şüpheliyse (*suspect*), sayı dizisinin yeteri kadar rastsal olmadığı sonucuna varılır.

3.1.2 "Kolmogorov-Smirnov" (KS) Testi

Chi-Square testinde, gözlemler sonlu sayıda kategoriler içerisine düşebiliyordu. Sonsuz sayıda değerler içinden elde edilen rastsal değerleri incelerken "Kolmogrov-Smirnov" testini uygulayabiliriz.

n tane birbirinden bağımsız gözlem X_1, X_2 ... X_n , "*F(x)*" sürekli (*continious*) fonksiyonu tarafından belirlenmiş olsun. İstatistiksel değerler, Formül 3.3 ve Formül 3.4 ile hesaplandığında K_n^+ ve K_n^- istatistiklerini bulunur.

$$K_n^+ = \sqrt{n} \max_{-\infty<x<+\infty} \left(F_n(x) - F(x)\right) \qquad \text{(Formül 3.3)}$$

$$K_n^- = \sqrt{n} \max_{-\infty<x<+\infty} \left(F(x) - F_n(x)\right) \qquad \text{(Formül 3.4)}$$

Bu istatistik değerler Çizelge 3.2 'ye göre dağılmış olmalıdırlar.

Çizelge 3.2: KS Testinde Dağılımı Değerleri

	p=1%	*p*=5%	*p*=25%	*p*=50%	*p*=75%	*p*=95%	*p*=99%
n=1	0.01000	0.05000	0.2500	0.5000	0.7500	0.9500	0.9900
n=2	0.01400	0.06749	0.2929	0.5176	0.7071	1.0980	1.2728
			. . .				
n=30	0.04354	0.1351	0.3509	0.5605	0.8036	1.1916	1.4801
n>30	$y_p^2 = \frac{1}{2} \ln(1/(1-p))$ için, $y_p - 1/6\, n^{-1/2} + O(1/n)$						
y_p=	0.07089	0.1601	0.3793	0.5887	0.8326	1.2239	1.5174

Bilgisayarda hesaplanmaları kolaylaştırmak için aşağıdaki yol önerilmiştir :

1. X_1, X_2 ... X_n birbirinden bağımsız gözlemleri belirlenir,
2. Bu gözlemler küçükten büyüğe sıralanacak şekilde düzenlenir: $X_1 \le X_2 \le \ldots \le X_n$
3. Formül 3.5 uygulanarak istatistiksel değerler hesaplanır.

$$K_n^+ = \sqrt{n} \max_{-\infty<x<+\infty} \left(\frac{j}{n} - F(X_j) \right)$$
$$K_n^- = \sqrt{n} \max_{-\infty<x<+\infty} \left(F(X_j) - \frac{j-1}{n} \right) \qquad \text{(Formül 3.5)}$$

"Kolmogrov-Smirnov" (KS) testinde alınan örnek sayısı n'in çok büyük olmasına gerek yoktur. Tablo kayıtları gerçek değerleri vermektedir.

χ^2 testi, KS testi ile birlikte kullanıldığında daha iyi sonuçlar verebilmektedir. Mesela 10 bağımsız χ^2 testi sonucunda V_1, V_2 ... V_{10} değerleri oluşacaktır. Bunların şüpheli olanlarını saymak ve karşılaştırmak yerine, bu elde edilen 10 değerin ampirik dağılımları belirlenip ve Çizelge 3.1'den elde edilebilecek gerçek dağılımla karşılaştırıldığında daha açık bir sonuca ulaşılır.

3.2 İstatistiksel (*Ampric*) Testler[23]

Ampirik testler sayı dizilerinin rastsallığını incelemek için kullanılan geleneksel (*traditional*) yöntemlerdir. Her test <*Un*> U_0 U_1 U_2 ... doğal sayılar dizisine uygulanacaktır. Bu testler, Rastsal Sayı üreteçlerini kara kutu olarak ele alıp algoritmalarını analiz etmezler (PLAB, 2000). Bazı testler ise esas olarak tamsayı dizileri için tasarlanmıştır. Bir çok istatistiksel test bulunmaktadır. Burada daha çok, yararlılığı ispatlanmış ve bilgisayarda uygulanabilir olanları ele alınacaktır.

Knuth, (Knuth, 1998) kitabında 11 adet testi incelemiştir. Bunlar:

1. Frekans (*Equidistribution - Frequence*) Testi
2. Seri (*Serial*) Testi
3. Boşluk (*Gap*) Testi
4. Poker Testi
5. Kupon Toplayıcısı (*Coupon Collector*) Testi
6. Permutasyon (*Permutation*) Testi
7. Koşu (*Run*) Testi
8. T'nin Maksimumu (*Maximum-of-t*) Testi

---

[23] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Knuth, 1998) alınmıştır

9. Çarpışma (*Collision*) Testi
10. Yaşgünü Araları (*Birthday Spacings*) Testi
11. Seri Bağlılaşım (*Serial Correlation*) Testi

Bazı testlerin ise öncelikle tamsayı dizileri için tasarlandığı ve 32-bitlik tam sayılar için kullanılmaları gerektiği unutulmamalıdır. Testler incelendiğinde elde edilen başarı açısından zayıf ve güçlü testler ortaya çıkmaktadır. Frekans Testi ve Seri Bağlılaşım Testi, her Rastsal Sayı Üreteci'nin bu testleri başarıyla geçmesi yüzünden en güçsüz testler olarak nitelendirilmektedirler. Çalışma Testi ve T'nin Maksimumu Testi ise güçlü testlerden sayılmaktadırlar.

3.3 Mevcut Uygulama Paketleri

Bilgisayarlar hızlandıkça Rastsal Sayı Üreteçleri yetersiz kalmaya başlamış ve buna bağlı olarak da eskiden var olan testlerde de bazı geliştirilmeler gerekmiştir. George Marsaglia'ın Diehard paketi gibi bu konuda çeşitli çalışmalar ortaya çıkmıştır (Knuth, 1998). Rastsal sayıları test etmek için kullanılan belli başlı uygulama paketleri aşağıdaki gibidir:

- ENT
- NIST STS
- Diehard
- FIPS 140-1 ve 140-2 Standardı

Bu testlerin kullanımı hakkında detaylı bilgi bölüm 6.2'de verilmiştir.

3.3.1 ENT Paketi[24]

Herhangi bir program veya veri dosyasını, byte veya bit bazında rastsallık testine tabii tutan ve sonuçlarını bir rapor şeklinde sunan bir

[24] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Walker, 1998) alınmıştır

programdır. Bu program; rastsal sayı üreteçlerini, sıkıştırma algoritmalarını, dosyanın yoğunluğunun (*density*) önemli olduğu herhangi bir uygulamayı değerlendirmek için kullanılabilir. Pakette bulunan testler aşağıdaki gibidir :

1. Entropi
2. Ki-Kare (*Chi-Square*)
3. Aritmetik Ortalama (*Arithmetic Mean*)
4. Pi Sayısının Monte Carlo Değeri (*Monte Carlo Value for Pi*)
5. Serial Correlation Coefficient

ENT deney sonuçları ideal değerden sapma olarak verilmekte ve üretecin deneyden geçme/kalma kararı kullanıcıya bırakılmaktadır. ENT paketi kullanıma hazır olarak (www.fourmilab.ch/random) Internet adresinde ücretsiz olarak mevcuttur. Bu deneyler hakkında daha detaylı bilgi bölüm 6.2.2'de verilmiştir.

3.3.2 NIST STS Paketi

NIST-STS paketi (tam adı NIST STS-1.3) ABD Ulusal Standartlar ve Teknoloji Enstitüsü İstatistiksel Mühendislik ve Bilgisayar Güvenliği Birimlerinin ortak çalışmasının bir sonucu olup 16 ayrı test içerir. Bunlar Frekans (*monobit*), Blok İçi Frekans, Koşu, Uzun Koşu, İkili Matrix Rankı, Ayrık Fourier Dönüşümü (*Spectral*), Non-overlapping Template Matching, Overlapping Template Matching, Maurer'in "Evrensel İstatistiksel", Lempel-Ziv Sıkıştırma, Doğrusal karmaşıklık, Seri, Yaklaşık Entropi, Cumulative Sums (*Cusum*), Random Excursions, Random Excursions Variant testleridir. Bu paketin dökümanında (Rukhin ve ark., 2000) paketin kullanımı, rastsallık testleri ve deney sonuçlarının yorumlanması konusunda geniş bilgi bulunmakta ve bu haliyle başlı başına bir referans niteliği taşımaktadır. İçinde Linear Congruential (LCG), Quadratic Congruential (QCG), Cubic Congruential (CCG), Secure Hash Algorithm (SHA1G), Blum-Blum-Shub (BBSG) gibi 9 standart üreteç içeren NIST-STS-1.3 paketi kullanıma hazır olarak (http://csrc.nist.gov/rng/rng2.html) Internet adresinde ücretsiz olarak

mevcuttur. Bu testler hakkında daha detaylı bilgi bölüm 6.2.3'de verilmiştir.

3.3.3 DieHard Paketi

DieHard, George Marsaglia'nın (Runs standart testi dışında) kendisinin geliştirdiği 15 adet testden oluşmaktadır. Her ne kadar Diehard testleri, 32-bitlik tam sayıları test etmek için oluşturulmuşsa da günümüzde Donanımsal ve Yazılımsal Rastsal Bit Üreteçlerinin testi için de yaygın olarak kullanılmaktadır. Bu testler, rastsal tamsayılardan oluşan büyük binary dosyalar (10-11 MB, en azından 80 milyon bitten oluşan bir dosya) üzerinde çalışmaktadır. Ufak dosyalar üzerinde de çalışabildiği belirtilmekle beraber önerilmemektedir. Programın nasıl çalıştırıldığı EK-3'de gösterilmiştir. DieHard testlerinin açıklamaları ve testlerden beklenen ideal sonuçlar için (Baldwin, 1998)'e başvurulabilir.

DieHard paketiyle birlikte KISS32 ve Universal Random Number Generator gibi çeşitli üreteçler gelmektedir. Bunun yanı sıra Marsaglia'nın (http://stat.fsu.edu/pub/diehard) Internet adresinde, çeşitli rastsal sayı üreteçlerinden ürettiği 16 adet 10 Megabyte'lık dosyadan oluşan 4,8 milyar rastsal bit bulunmakta ve bu verilerin üretilme yöntemleri hakkında detaylı bilgi sunmaktadır.

3.3.4 FIPS 140-1 ve 140-2 Standardı

Şifreleme amaçlı kullanılan rastsal sayı üreteçlerinin FIPS[25] 140-2 standardına göre belirli testlere tabii tutulması önerilmektedir. Güvenlik seviyesi 3 ve 4 için önerilen bu testlerin yerine benzer veya daha kuvvetli testler kullanılabilir. Bu standarda göre; rastsal sayı üretecinden alınan 20.000 bit'den oluşan bit dizisi şu testlere tabii tutulmalıdır (Brown, 1994):

1. Monobit Testi
2. Poker Testi

---

[25] FIPS - Federal Information Processing Standard - NIST tarafından yayınlanmaktadır ve federal bilgi kaynakları için teknik standartlar üretmekten ve rehberlik etmekten sorumludur (INTEL, 1999).

3. Koşu (*Runs*) Testi

4. Uzun Koşu (*Long Run*) Testi

Bu testlerden herhangi birinden başarısız olunması durumunda rastsal sayı üretecinin hatalı olduğu hükmüne varılacaktır (INTEL, 1999). INTEL, bütün donanımsal rastsal sayı üreteçlerinin FIPS 140-1 rastsallık testlerini geçmesi gerektiğini belirtmiştir. Bu testler herhangi bir hatayı tespit etmek için yeterlidir (Jun ve ark, 1999).

1999 yılının sonlarında FIPS 140-2 standardı çıkmış ve testlerin geçerli olduğu aralıklarda değiştirilmeler (daraltılmalar) gerçekleştirilmiştir. Uygulamalarda aralık değerleri FIPS 140-2'ye göre alınmıştır. Bu testler hakkında daha detaylı bilgi bölüm 6.2.1'de verilmiştir. FIPS standardına göre test yapan uygulama koduna EK-7'den ulaşılabilir.

3.4 Uygulanan Bazı Rastsallık Testleri

Bölüm 6'da rastsal sayı üreteçlerini test etmek için uyguladığımız testlerin bazıları aşağıda belirtilmiştir:

- Chi Square (χ^2) Testi
- Entropi Testi
- Artimetik Ortalama
- Pi Sayısının Monte Carlo Değeri
- Seri Korelasyon Katsayıları Testi
- Frekans (*Monobit*) Testi
- Koşu Testi
- Uzun Koşu Testi
- Spectral Testi
- Seri (*Serial*) Testi
- Poker Testi

Chi Square (χ^2) Testi

Temel düşüncesi üretilen sayıların iyice dağılıp dağılmadığını kontrol etmektir. Detaylı bilgi için bakınız bölüm 3.1.

Entropi Testi[26]

Bir veri kümesinin bilgi yoğunluğuna (veya belirsizliğine) entropi denir. Genellikle karakter başına bit sayısı olarak ifade edilir.

Rastsal dizilerde entropi maksimum limit değerine ulaşır ve bu diziler sıkıştırılamaz. Burada test edilen dizinin entropisinin maksimum limite yakınlığı dizinin rastsallığına işaret eder. Veri kümesi ASCII karakterlerden (256 adet karakter) oluşuyorsa (karakter başına 8 bit); entropi en fazla "$log_2 256 = 8$"e yakın olacaktır. Veri kümesi hex sayılardan (16 adet karakter) oluşuyorsa (karakter başına 4 bit); entropi en fazla "$log_2 16 = 4$"e yakın olacaktır. Entropi bu değerlere yakınsa o dosyanın sıkıştırılması, dosya büyüklüğünü değiştirmeyecektir. Yakın değilse, mesela 4.9 entropiye sahip bir ASCII dosya ise, optimal bir sıkıştırma ile boyutu %38 azalabilir.

Aritmetik Ortalama Testi

Bütün byte'ların (-b parametresi verildiyse bit'lerin) toplamını alıp dosya uzunluğuna bölünmesinden çıkan sonucu bildirilir. Eğer dosya, karakter dizilerinden oluşuyorsa rastsal olması için 127,5 (-b parametresi verildiyse 0,5) civarı olması gerekmektedir (Walker, 1998).

Pi Sayısının Monte Carlo Değeri Testi

Birbirini takip eden 6 byte'lık diziler alınarak bir karenin içinde 24 bitlik X ve Y koordinatları olarak kullanılır. Eğer rastsal üretilen noktanın uzaklığı, karenin içerisindeki çemberin çapından daha küçükse bir isabet (*hit*) olarak kabul edilir. İsabetlerin yüzdesi Pi'nin değerini ölçmek için kullanılır. Büyük veri dizileri (en az birkaç bin byte) için bu değer Pi'nin gerçek değerine yaklaşacaktır (Walker, 1998).

---

[26] Aksi belirtilmediği sürece kaynak olarak (Walker, 1998) alınmıştır.

Seri Korelasyon Katsayıları Testi

Dosyadaki herhangi bir byte'ın bir önceki byte'a ne derece bağlı olduğunun bir ölçüsünü verir. Rastsal diziler için bu değer 0 'a yakın olmalıdır (Walker, 1998).

Frekans (Monobit) Testi

Rastsal bir dizide 0'ların ve 1'lerin sayısının yaklaşık olarak aynı oranda (0,5) olması beklenir. Bu test, dizinin tamamında 0 ve 1 lerin oranının 0,5'e yakınlığını değerlendirir. FIPS 140 testinde bunun uygulanması aşağıdaki gibidir (FIPS 140-2, 1999):

1. 20.000 bit dizisindeki "1" lerin adedi sayılır ve X olarak nitelendirilir.
2. Eğer $9.725 < X < 10.275$ ise bu test başarıyla geçilmiş demektir.

Koşu Testi

Koşu (*run*), birbirini ardışık takip eden 1 veya 0'lar dizisidir. Bunların (hem 0 hem de 1'ler için) adedi sayılır. Bu testin başarıyla geçilmesi için birbirini takip eden k (FIPS140'da : $0<k<7$) uzunluğundaki koşuların rastsal bir dizi için beklenen değerlerle uyumlu olmasına bakılır. Koşu testinin özellikleri aşağıda belirtilmiştir (FIPS 140-2, 1999):

1. Koşu, birbirini ardışık takip eden 1 veya 0'ların maksimum dizisidir. Bunların (hem 0 hem de 1'ler için) adedi ölçülür.
2. Testin başarıyla geçilmesi için birbirini takip eden (1-6 uzunluğundaki) dizilerin Çizelge 3.3'deki değerlere uyması gerekmektedir. 6'dan daha uzun diziler de 6 olarak kabul edilecektir. Hem "0" lar hem de "1" 'ler ölçüldüğünden 12 ölçüm de bu aralıklarda olmalıdır.

Çizelge 3.3 : Koşma Testi Aralık Değerleri

Koşu'nun Uzunluğu	Beklenen Aralık
1	2,343-2,657
2	1,135-1,365
3	542-708
4	251-373
5	111-201
6+	111-201

Uzun Koşu Testi

Uzun Koşu (*Long Run*) testinin özellikleri aşağıda belirtilmiştir (FIPS 140-2, 1999):

1. 34 uzunluğundaki birbirini ardışık takip eden 1 veya 0 değerlerine uzun koşu denir
2. 20,000 lik bit dizisinde hiç uzun koşu olmamalıdır.

Spectral Testi

Doğrusal benzerlik üreteçleri, birçok rastsallık testini geçmesine rağmen üretilen sayıların 2,3,...,*n* boyutlu grafiklerde belirli bir örüntü izlediği görülebilir. 2 boyutlu noktalamada (*plot*) noktaların paralel çizgilerde yer aldığı, 3 boyutlu noktalamada noktaların paralel yüzeylerde yer aldığı görülecektir. Mesela *n* rastsal sayı üretip bunları bir noktanın koordinatları olarak belirleyip, *n* boyutlu bir hiperkübe (*hypercube*) yerleştirdiğimizde bütün noktalar, *n*-1 boyutlu hiper yüzeylerde yer alacaktır. Spectral testi bu paralel yüzeyler arasındaki uzaklığı ölçmekte ve üretecin ne kadar rastsal olduğu hakkında bize bilgi vermektedir. İyi bir rastsal sayı üretecinde her boyut için yüzeyler birbirine yakın aralıkta bulunmalıdır (Deley, 1991).

Spectral testi ayrık FFT (*Fast Fourier Transform*) de tepe değerlerine odaklanır. Bu testin amacı test edilen dizide rastsallıktan

sapmayı işaret edecek peryodik özellikleri (birbirine yakın tekrar eden örüntüleri) yakalamaktır (Rukhin ve ark., 2000). Bu testin LCG gibi *lattice* yapısına sahip rastsal sayı testinde kullanılabileceği ve üreteç için uygun parametrelerin seçiminde yararlı olduğu belirtilmiştir (PLAB, 2000).

Seri (Serial) Testi

Bu testin odağında dizinin tamamında çakışan m-bit örüntülerin (*overlapping m-bit paterns*) sıklıkları vardır. Bu m-bit örüntülerin rastsal bir dizide olması gereken değerlerle uyumlu olmasına bakılır (Rukhin ve ark., 2000).

Poker Testi

Poker testinin özellikleri aşağıda belirtilmiştir (FIPS 140-2, 1999):

1. 20.000 bitlik dizi, birbirini takip eden 5.000 adet 4 bit parçalara bölünür. Olabilecek 16 değişik 4 bitlik kombinasyonun kaçar kez geçtiği ölçülür. *f(i)*, i'nin 16 değişik değeri için ($0 \leq i \leq 15$) 4 bitlik değerlerin kaçar kez geçtiğini gösteren bir fonksiyon olarak tanımlansın. Formül 3.6 ile X değeri ölçülür :

$$X = (16/5000) * \left(\sum_{i=0}^{15} [f(i)]^2\right) - 5000 \qquad \text{(Formül 3.6)}$$

2. $1{,}03 < X < 46{,}17$ ise test başarıyla geçilmiş demektir.

3.5 Test Sonuçlarının Değerlendirilmesi

Verilen bir *s* dizisinin uygulanan istatistiksel bir testi geçip geçmediğini belirleme konusunda üç farklı yaklaşım kullanılabilir (Soto, 1999):

- ***Eşik (Threshold) Değeri:*** Verilen *s* dizisi için hesaplanan istatistiksel değer *f(s)* bir eşik değeri *E* ile karşılaştır. Eğer *f(s)<E* ise dizi testden kalır; yani rastsal olmadığına karar verilir. Örneğin, ENT paketindeki entropi testi için eşik değeri *E= 0,99*8 = 7,92* (8 = byte için maksimum entropi değeri) olarak belirlenebilir.
- ***Sabit Aralıklar:*** Bu kez eşik değeri yerine test sonucunun içine düşmesi gereken bir aralık belirlenir ve eğer *f(s)* bu aralığın dışına düşerse dizi testi geçemez. FIPS140 test bataryası bu yaklaşımı kullanır.
- ***Olasılık (probability) Değerleri:*** Bu yaklaşım test istatistiği *f(s)* ve buna karşılık gelen olasılık değerini (*o-değeri*) hesaplamayı gerektirir. *o-değeri* rastsal bir dizide *f(s)* değerine eşit yada büyük bir değer görülme olasılığıdır. Dolayısı ile küçük değerler (tipik olarak, *o-değeri*<0,05 veya *o-değeri*<0,01) *s* dizisinin rastsallıktan uzak olduğuna kanıt teşkil eder. Bu durumda sabit bir anlamlılık (*significance*) düzeyi α için, *o-değeri* < α ise *s* dizisi testde başarısız olur. α değeri çok büyük seçilirse rastsal olan diziler "rastsal değil" şeklinde sınıflandırılabilir. Buna *Type I hata* denir. Tersi durumda (α değeri çok küçük seçilirse) rastsal olmayan diziler rastsal olarak kabul edilebilir. Buna *Type II hata* denir. α değeri genellikle [0.001, 0.01] aralığında seçilir.

ENT paketi test sonuçlarının değerlendirmesi için kriterler sağlamaz. Bu durum ENT paketinin kullanımını güçleştiren önemli bir dezavantajıdır. FIPS140 testleri için ise sabit aralıklar şeklinde verilen kesin kriterler vardır. Örneğin, 20.000 uzunluğundaki dizi için 1'lerin sayısı [9725,10275] aralığında ise dizi testi geçer; değilse kalır. Bu durum FIPS140 'ın kullanımını kolaylaştırmakla fakat alternatif değerlendirme esnekliği tanımamaktadır.

NIST-STS ise test sonuçlarının değerlendirilmesinde iki ayrı kriter kullanmaktadır (Rukhin ve ark, 2000):

(1) testi geçen dizilerin oranı :

Verilen $\alpha = 0{,}01$ için kabul edilebilir oran aralığı $\hat{p} = 1 - \alpha$ ve m örnek sayısı için

$$\hat{p} \pm \sqrt{\frac{\hat{p}(1-\hat{p})}{m}} = 0{,}99 \pm \sqrt{\frac{0{,}99(0{,}01)}{1000}} = 0{,}99 \pm 0{,}0094392$$

olarak hesaplanır. Yani üretecin testi geçmesi için oran 0,9805607'nin üzerinde olmalıdır. Örneğin eğer 100 dizi test edilmiş (*m=1000*), $\alpha = 0.01$ ve 993 dizinin *o-değeri* $\geq 0{,}01$ ise testi geçen dizi oranı 993/1000 = 0,993 ve bu oran 0,9805607'den büyük olduğundan üreteç testi geçer. Eğer 1000 dizide 974 dizi testi geçerse oran 974/1000=0,974 < 0,9805607 olur ve üreteç testden kalır.

(2) o-değerlerinin eşit dağılımı (uniformity):

o-değerleri [0,1] aralığında yer alır. Bu aralık 10 eşit alt aralığa ayrılır ve her alt aralıktaki o-değerleri sayılır. Ardından Chi Square χ^2 testi kullanılarak *o-değerlerinin* bir *o-değeri* (o-değeri$_T$) aşağıdaki formüller kullanılarak bulunmaktadır. F_i alt-aralık i deki o-değeri sayısı ve *m* örnek sayısı, igamc = "incomplete gamma fonksiyonu" olmak üzere,

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - m/10)^2}{m/10} \quad \text{ve} \quad \textit{o-değeri}_T = \texttt{igamc}(9/2, \chi^2/2)$$

elde edilir. Örneğin eğer *n*=100 dizi test edilmiş ve bu 10 alt aralık için *o-değer* dağılımı

(16 14 8 9 12 7 11 7 11) ise *o-değeri*$_T$*=0,304126* ve bu değer 0,0001'den büyük olduğundan üreteç testi geçer. Eğer için o-değer dağılımı (3 3 5 8 8 12 17 24 8 12) ise *o-değeri*$_T$ *=0,000013* ve bu değer 0,0001'den küçük olduğundan üreteç testden kalır.

4 ASAL SAYILAR

Birden büyük, sadece kendisine ve bire bölünen tam sayılara *asal* sayı denir. Asal olmayan sayılara ise *bileşik* (*composite*) sayı denir Asal sayılar, RSA'da olduğu gibi gizli asal sayılara dayanan anahtar üretme süreçlerinde kullanılmaktadır (RSA, 1998).

Asal sayılar detaylı olarak ilk kez antik Yunan matematikçileri tarafından incelendi.[27] M.Ö. 500-300 yılları arasında Pythagoras'nın okulunda asal sayıların temelleri keşfedildi. Euclid, her tam sayının asalların çarpımı olarak tek bir şekilde yazılabileceğini ispatladı. M.Ö. 200 yılında Eratosthenes, asal sayıları hesaplayan "Sieve of Eratosthenes" algoritmasını geliştirdi. Karanlık çağ da denilen uzun bir süre asal sayılar konusunda hiçbir çalışma yapılmadı. 17'inci yüzyılda Fermat'ın sağladığı gelişmelerle asal sayılar yeniden keşfedildi. Fermat'ın teoremine göre herhangi bir n sayısının asal olması için $obeb(a,n)=1$ eşitliğini sağlayan her a sayısı ile $a^{(n-1)} = 1 \pmod n$ eşitliğini sağlaması gerekmektedir. Bu teoremin temelleri 2000 yıl önce geliştirilmiş eski bir Çin hipotezine dayanmaktadır. Bu hipoteze göre n'in asal olması için n'in $2^n - 2$'yi bölmesi gerekmektedir.

Mersenne, asal olan n değerleri için 2^n-1'in de asal olduğunu iddia etti. Bu eşitliği sağlayan sayılara Mersenne Sayıları denmektedir. Her ne kadar bu formül bütün asal n değerleri için geçerli olmasa da bu formül bilinen en büyük asalların bulunmasını sağlamaktadır. Daha sonraki yıllarda Euler, Fermat'ın teoremini geliştirerek $n \geq 1$ için $[1,n]$ aralığında n'e göreceli asal (*relatively prime*) sayıların adedini veren Euler phi (*totient*) fonksiyonu $Q(n)$'i oluşturdu (Menezes ve ark, 1997).

Legendre ve Gauss, büyük n değerleri için n'e yakın asalların yoğunluğunun $1/\log n$ olduğu sonucunu buldular. Bu sonuç Asal Sayı Teoremi olarak bilinmektedir. Bu teorem 1896 yılında Hadamard ve Valle Poussin tarafından ispatlanmıştır.

Asal sayılar konusunda çözüm bekleyen birçok problem bulunmaktadır. Bu tür problemler çözülürken Sayılar Kuramında ve

---

[27] Bu bölümdeki tarihçe için (O'Connor ve ark, 1996) referans olarak alınmıştır.

matematik başta olmak üzere birçok bilim dalında da ilerlemelere yol açıldığı unutulmamalıdır.

4.1 Tanımlamalar

Bu bölümde asal sayılarla ilgili belli başlı tanımlamalar yer almıştır.

Mersenne Asalları:

$n > 1$ için $2^n - 1$ formatında bulunan asal sayılardır. Şu ana kadar bulunan en büyük asal sayı $2^{2976221}$-1, 895.932 basamaklı bir Mersenne asal sayısıdır (Emerson, 1997).

Internet üzerinde en büyük Mersenne sayısını bulmak için yarışmalar düzenlenmekte ve bu amaç için kullanılacak yazılım bedava olarak dağıtılmaktadır. Detaylı bilgiye www.mersenne.org internet adresinden ulaşılabilir.

Güçlü (*Strong*) Asal:

Özellikle RSA'ya olan güvenlik saldırıları, daha güvenli asal sayılar elde etme ihtiyacını doğurmuş ve "*güçlü asal*" kavramı ortaya atılmıştır (Penzhorn, 1992). Bu Pollard'ın *P-1* ve Williams'ın *P+1* çarpanlara ayırma algoritmalarına karşı bir güvenlik önlemi olarak ortaya çıkmıştır (Silverman, 1997).

Aşağıdaki şartları sağlayan doğal sayılara güçlü asal denir (Penzhorn, 1992):

- x asal olmalı,
- $x > 10^{100}$ (en az 100 basamak) olmalı,
- $(x-1)$' in büyük bir asal faktörü y olmalı ,
- $(x+1)$'in büyük bir asal faktörü olmalı,
- $(y-1)$'in de büyük bir asal faktörü olmalı.

Bu tür kısıtlamalarla üretilen bu asal sayılara dayanan RSA PKS anahtarların pratikte diğer anahtarlardan daha güvenli olmadığı RSA'in

(Silverman, 1997) makalesinde nedenleri ile belirtilmektedir. Makalede "güçlü asal" üretmenin, rastsal asal üretmekten daha fazla zaman almayan yöntemleri de olduğu belirtilmektedir.

Güvenli (*Safe*) Asal:

Tek sayı olan p için, $p = 2x + 1$ eşitliğini sağlayan bir x asal değeri varsa p'ye güvenli asal denir. Bu tür asalların kriptografide kullanımının yararlı olduğuna ve RSA'daki $N=p*q$ eşitliğindeki p ve q eğer güvenli asalsa çarpanlara ayırmanın daha zor olduğu bir yaygın kanaattir (Scheepers, 1999). Ama bu tür sayılar, diğerlerine göre daha az bulunduğu için önceden listeleri çıkarılabileceğinden bazı zayıflıklar getirebilir.

İkiz (*Twin*) Asallar:

Aralarında sadece bir sayı olan ardışık asallara ikiz asallar denir.

Örnek: 3-5; 5-7; 11-13; 17-19; 29-31; ... vb

Sonsuz sayıda asal olduğuna göre, sonsuz sayıda ikiz asal sayı bulunmaktadır. Bu tür sayıların ve asal sayıların sıklıkları değişkendir, yani belirli bir formülleri yoktur (Scheepers, 1999).

4.2 Carmichael Sayıları

Fermat'ın teoremine (1640) göre : *n*'nin asal olması için bütün a değerleri için *n*'nin a^n -*a*'yı bölmesi gerekmektedir. Yine de bütün *a* tabanlarına göre bu testi yanıltan sayılar bulunmaktadır. 1899 yılında Korselt bu tür sayıların kriterlerini belirlemiştir. *n*'in bütün *a* değerleri için a^n -*a* 'yı bölmesi için aşağıdaki özelliklere sahip olmalıdır:

1. *n*, kare bağımsız[28] (*square free*) olmalıdır.
2. *n*'yi bölen bütün *p* asal değerleri için (*n-1*) de (*p-1*) değerlerine bölünmelidir.

Bu tür *n* sayılarının örneklerini 1910 yılında Carmichael vermiş ve bu kritere uyan sayılar onun adıyla anılmaya başlanmıştır. "$x^{(n-1)} = 1 \ mod \ n$" eşitliğini her *x* sayısı ile sağlayan bileşik sayılara Carmichael sayıları denmektedir.

Bu tür sayılardan en ufağı 561'dir ve 561=3 x 11 x 17 şeklinde asal çarpanlarına ayrılmaktadır. İlk Carmichael sayıları üç asal faktörden oluşmaktadır. Dört asal faktör için 41.041'e, beş asal faktör için 825.265 sayısına ulaşmak gerekmektedir. 10^{16} 'ya (17 basamağa) kadar 246.683 adet Carmichael sayısı bulunmaktadır ve bu sayıların en fazla 10 adet asal faktöre sahip olduğu gözlenmiştir (Pinch, 1998). Bu asal sayıları bulunduran dosyalar ftp.dpmms.cam.ac.uk adresindeki /pub/Carmichael dizininden temin edilebilir.

Yeteri kadar büyük *x* sayısı için, *x* sayısına kadar $x^{2/7}$ den daha fazla Carmichael sayısı olduğu 1992 yılında Alford, Granville ve Pomerance tarafından ispatlanmıştır (Granville, 1992).

---

[28] Bir sayının asal faktörleri tekrar etmiyorsa o sayının *kare bağımsız* (*square free*) olduğu söylenir. Örneğin 30 sayısı 2x3x5 olduğu için kare bağımsızdır.

4.3 Asal Sayı Adedi ve Yoğunluğu

Büyük bir *n* sayısı için, *n*'den daha küçük olan asal sayıların adedi Formül 4.1 ile hesaplanmaktadır.

$$\prod_{n}^{\infty} n = \frac{n}{\ln n} \qquad \text{(Formül 4.1)}$$

Uzunluğu verilen bir sayının asal olma olasılığını da hesaplamak mümkündür. *r* sayısı $1 \leq r \leq 10^{100}$ aralığından rastgele çekilmiş bir sayı olsun. $n = 10^{100}$ için, hesaplama yöntemleri aşağıda belirtilmiştir (Penzhorn, 1992):

- *r* 'nin asal olma olasılığı :

Pr (r asal) = Π(n) / n = 1 / ($\ln^{29}$ n)

Pr (r) = 1 / 230

- Tek olan *r* 'lerin asal olma olasılığı :

Pr (r asal | r tek) = Π(n) / (n/2) = 2 / (ln n)

Pr (r) = 1 / 115

- Küçük asal faktörlere bölünmemesi şartını da dikkate aldığımızda bu oran :

Pr (r asal | obeb(r, 2 ve 3) =1) = Π(n) / (n * ½ * (1 – 1/3))

Pr (r) = 3 Π(n) / n = 1 / 77

olmaktadır.

Bir *x* sayısı civarındaki asal sayı yoğunluğu (*density of primes*) Formül 4.2 ile hesaplanmaktadır. Bu formül, *x* sayısı civarındaki asal sayıların oranını vermektedir (Segre, 2000). Yoğunluk, kabaca *x*'in logaritmasıyla orantılı, basamak adediyle ters orantılı olacaktır.

$$\frac{1}{\ln(x)} - \frac{1}{\ln(x^2)} \qquad \text{(Formül 4.2)}$$

---

[29] ln fonksiyonu dökümanda *e* tabanında logaritma ($\log_e$) fonksiyonu olarak kullanılmaktadır.

4.4 Sayılarla Asallar

Asal sayıların yoğunluğu konusunda bir fikir vermesi açısından 10^{10}'a kadar olan asal sayılarda asal sayı dağılımı Çizelge 4.1'de verilmiştir.

Çizelge 4.1 : 10^{10}'a Kadar Olan Sayılarda Asal Dağılımı

10^{10} 'a kadar	**4.4.1.1.1.1 Toplam Sayı Adedi**
Sayılar	10.000.000.000
Asal Sayılar	450.000.000
2 Tabanına Göre Fermat Asallık testini geçen bileşik (*composite*) sayılar	15.000.000
Carmichael Sayılar (herhangi bir tabana göre asallık testini geçen bileşik sayılar)	1.547

Asal sayılar hakkında daha iyi bir fikir vermesi açısından aşağıdaki bilgiler verilebilir :

- $n < 2{,}5 * 10^{10}$ 'da 2^n-2'yi bölen bir n bulma ihtimali 1/50.000 'den azdır.
- 100 basamaklı asal sayıların adedi $3{,}904219 * 10^{97}$ 'dir. Tüm 100 basamaklı sayıların listesinin bilgisayarda kaplayacakları alan $3{,}904219 * 10^{97} * 50$ byte olmaktadır. 1 Megabyte'ın 10^6 byte, 1 Gigabyte'ın 10^9 byte, 1 Terabyte'ın 10^{12} byte olduğu dikkate alınırsa bu sayıların hepsinin hiçbir şekilde bir yerde saklanamayacağı daha açıkça görülmektedir.

5 ASALLIK TESTLERİ

Asal sayıları bileşik sayılardan ayırt etmek aritmetiğin en temel ve önemli konularından biri olmuştur. Bu konu günümüzde hala geçerliliğini korumaktadır.

Asallık tanımından yola çıkarsak bir n tamsayısının asal olması için 2 ile $\sqrt{n}$ arasında hiç bir böleni olmaması gerekir. Bu yöntemi ufak sayılar için uygulamak mümkün olsa da sayıların büyüklüğü arttıkça bunun hesaplanması mümkün olmamaktadır. 100 basamaklı bir tamsayının bu yöntemle asal olup olmadığını ölçmek pratik değildir. Bunu bir örnekle açıklarsak : 101 basamaklı ilk sayı olan (1.10^{100})'un asal olup olmadığını bu yöntemle hesaplarken 2 ile $\sqrt{10^{100}}$ arasındaki her sayıya bölmek gerekecektir. Her bölme işlemi 1 milisecond (ms) alındığında bu işlemler 10^{50}-1 ms sürecektir. Bu da işlemlerin yaklaşık olarak $1.9 * 10^{41}$ yıl sürmesi demektir.

Büyük sayıların asal olup olmadıklarını anlamak için daha gelişmiş asallık testleri gerekmektedir (Granville, 1992). Asallık testlerini iki ana gruba ayırmak mümkündür :

- Kesin (*deterministic*) Asallık Testleri
- Olası (*probabilistic*) Asallık Testleri

5.1 Kesin (*deterministic*) Asallık Testleri

Bu tür asallık testleriyle kesin olarak bir sayının asal olup olmadığını belirlemek mümkündür. Bu tür yöntemler genelikle çarpanlara ayırmaya dayanmaktadır. Şu an kullanılmakta olan en geçerli iki yöntem aşağıda belirtilmiştir :

1. Cyclotomic Ring Testi (Bosma-Cohen-Lenstra)
2. Elliptic Curve Testi (Atkin-Goldwasser-Killian)

Bu yöntemler o kadar karmaşıktır ki, uygulamada bir hata yapma olasılığı, olası asallık testinde hata yapma olasılığından daha fazladır (Silverman, 1997). Çarpanlara ayırmaya dayalı yeni yöntemler geliştirilmekle birlikte bu konuda bir standarttan söz etmek henüz

mümkün değildir. Ayrıca *Miller&Rabin* olası asallık testinin deterministik versiyonu gibi değişik yöntemler de bulunmaktadır.

Özellikle Mersenne asallarını bulmak için tasarlanan ve kullanılan *Lucas-Lehmer* testi de deterministik bir asallık testidir. Mersenne sayılarını bulmakta çok iyi sonuçlar vermektedir (Emerson, 1997). $s \geq 3$ olmak üzere "$n = 2^s - 1$" eşitliğini sağlayan n değerinin Mersenne asalı olması için aşağıdaki iki koşulu sağlaması gerekmektedir:

- s asal olmalı,
- u_o=4 ve k=[1, s-2] için "$u_{k+1} = (u_k^2 - 2) \bmod n$" eşitliğinde u değerleri 0 'a eşit olmalıdır (Menezes ve ark, 1997).

5.2 Olası (*probabilistic*) Asallık Testleri

Asallık testi, bir sayının asal olduğunu kanıtlama sürecidir. Olası asallık testleri ise bir sayının yüksek olasılıkla asal olduğunu kanıtlama sürecidir.

Olası asallık testlerinde, asal sayı üretmek için ilk önce n-bitlik bir rastsal sayı üretilir ve asallık testine tabi tutularak istenildiği kadar küçültülebilen bir hata payı ile sayının asal olup olmadığı belirlenir; asal olanlar seçilir ve kullanılır. Deterministik yöntemlere göre daha hızlı olduğu ve gerçekte bir sayının asal olup olmadığını ufak hata payları ile kanıtlayabildiği için Olası Asallık Testlerinin kullanılması önerilmektedir. 2^{-100} den daha düşük bir hata payı ile bir sayının asal olduğu belirlenebilir (RSA, 1998). Asal sayı testlerini gerçekleştirirken dikkat edeceğimiz hususlar şunlar olabilir :

1. Çift sayıları test etmeye gerek yoktur.
2. Asallığı test edilen sayıların; 3, 5, 7, 11 ... gibi küçük asal sayılara bölünüp bölünmediğine bakarak birçok sayı elenebilir. Örneğin 23 sayısına kadar olan asal sayılar alınırsa, asal sayı olmayanların en azından 2/3'ü ayıklanabilir ve 3 kat daha hızlı sonuca gidilebilir (Segre, 2000).

Dikkat etmemiz gereken temel kriter, hata oranının 2^{-100} 'den daha büyük olmaması gerektiğidir (Silverman, 1997). Literatürde adı geçen Olası Asallık testleri aşağıda verilmiştir:

- Fermat Testi
- Slovay & Strassen Testi
- Lehmann Testi
- Miller&Rabin Testi
- Frobenius Testi
- Bileşik Testler

5.2.1 Tanık (*witness*) Kavramı[30]

Bir sayının asal olmadığını (bileşikliğini) kanıtlamak için çarpanlara ayırmaya çalışmak anlamsızdır çünkü çarpanların yoğunluğu genelde çok azdır. Bunun yerine Olası Asallık Testleri kullanılarak daha etkin bir çalışma yapılabilir. Olası Asallık Testleri, tanık (*witness*) kavramına dayanmaktadır ve tanıklık fonksiyonları olarak da adlandırılırlar. Tanık, *n* sayısının bileşikliğini (*composite*) göstermek için kullanılan 1 ile *n* arasında bulunan herhangi bir sayıdır. Tanıklık fonksiyonlarına göre, tanık sayıların yoğunluğu (*d* değeri) değişmektedir ve bu değerler Çizelge 5.1'de gösterilmiştir. Daha büyük *d* değerleri, itimat eşiğine[31] (*confidence threshold*) daha hızlı yaklaşma (*convergence*) demektir.

Çizelge 5.1 : Tanıklık Fonksiyonlarında yoğunluk (*d*) değerleri

Tanıklık Fonksiyonları (Olası Asallık Testleri)	***Tanık Sayıların Yoğunluğu (d değeri)***
Lehmann	0,5
Miller&Rabin	0,75
Solovay – Strassen	0,5

[30] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Segre, 2000) alınmıştır

[31] İtimat Eşik Değeri, ulaşılması hedeflenen hata payı yani asal diye belirlenen sayının bileşik olma olasılığıdır. Asal olarak belirlenen bir sayının, hata payı (bileşik olma olasılığı) 2^{-100} olarak belirlenirse, 2^{-100} itimat eşik değeri olur.

Olası asallık testine sokulan bir sayının *i* iterasyon sonunda asal ilan edilmesine rağmen bileşik olma olasılığı; tanık sayıların yoğunluğu *d* olarak kabul edilirse, Formül 5.1 ile hesaplanmaktadır. Bir sayının asal olma olasılığı ise Formül 5.2 ile hesaplanmaktadır.

$$P(\text{bileşik sayı}) = (1-d)^i \qquad \text{(Formül 5.1)}$$

$$P(\text{asal sayı}) = 1 - P(\text{bileşik sayı}) = 1 - (1-d)^i \qquad \text{(Formül 5.2)}$$

Çizelge 5.1'den tanıklık fonksiyonlarını kıyaslamak mümkündür. Örneğin, Miller&Rabin testinde yoğunluk daha fazla olduğu için daha az adımda seçilen eşiğe ulaşılabilir. Lehmann yönteminin bir rastsal sayı için uygulanması sonucunda 1 veya -1 çıkarsa sayı testi geçmektedir. Yine de o sayının bileşik olma olasılığı 0,5 den daha az olmakla beraber hala mümkündür.

Olası asallık testlerinin iterasyonları arttırılarak belirtilen hata paylarının daha da düşmesi sağlanabilir. Aslında bu oranlar oldukça abartılıdır. Çoğu rastsal sayı için, rastsal seçilen bir *a* değerinin tanık olma olasılığı %99,99 'dur (Schneier, 1996).

5.2.2 Yalancı-tanıklık (*non-witness*) Kavramı

Bir bileşik *n* sayısı için, W(n) kümesi *n*'in bileşik olduğunu kanıtlayabilecek sayılardan yani *tanık* sayılardan oluşsun. Tümleyen (*complementary*) küme L(n), $L(n) = Z_n^{32} - W(n)$ elemanlarından oluşacaktır ve bu kümenin elemanları *yalancı-tanık* olarak adlandırılır. Eğer testlerde parametre olarak yalancı-tanıklar kullanılırsa yanlış sonuçlara ulaşılması mümkündür. Çünkü testler bileşik sayının asal olduğunu bildireceklerdir. Bu tür yanlışlıklarla karşılaşmamak için bu tür testleri (yeteri kadar büyük bir t sayısı için) *t* kere tekrarlamamız hata olasılığını daha da düşürecektir (Menezes ve ark, 1997). *Miller&Rabin* testi icin yalancı-tanıkların sayısının çok az olduğu Higgins'ın yaptığı araştırmalarda ortaya çıkmıştır (Higgins,2000).

---

[32] Z_n, *n* sayısına kadar olan tam sayısı kümesini, yani {-n..., -2, -1, 0,1,2, ... n}kümesini temsil etmektedir (Menezes ve ark, 1997).

5.2.3 Fermat Testi

Fermat'ın teoremine göre herhangi bir *n* sayısının asal olması için [1, n-1] aralığından taban olarak alınan herhangi bir *a* sayısı ile Formül 5.3'deki eşitliği sağlaması gerekmektedir. Bu test, olası asallık testlerinin temelini oluşturmaktadır (Menezes ve ark, 1997). *n* sayısına, *a* tabanına göre *Fermat sözde asalı* denir (Penzhorn, 1992).

$$a^{(n-1)} = 1 \pmod{n} \qquad \text{(Formül 5.3)}$$

Fermat testi, Carmichael sayılarını tespit etmekte başarısız kalmaktadır. Diğer Olası Asallık Testleri, Fermat'ın teoremini temel olarak alsalar da aynı zamanda diğer birçok durumu da kontrol ettiklerinden daha gerçeğe yakın sonuçlar döndürmektedirler (Granville, 1992).

5.2.4 Lehmann Testi

n sayısının asal sayı olup olmadığını test etmek için rastgele olarak seçilen (*randomly chosen*) *a* sayılarıyla, *b* değeri Formül 5.4 ile bulunur (Segre, 2000). Eğer bütün *b* değerleri 1 veya –1 ise, fakat sadece 1 veya sadece -1 değilse *n* asal olarak kabul edilebilir (Menezes ve ark, 1997).

$$b = a^{((n-1)/2)} \pmod{n} \qquad \text{(Formül 5.4)}$$

Formül 5.4'ün Formül 5.3'ten nasıl elde edildiği aşağıda belirtilmiştir (Segre, 2000):

$$a^{(n-1)} = 1 \pmod{n}$$
$$b^2 = a^{(n-1)} \pmod{n}$$
$$b = a^{((n-1)/2)} \pmod{n}$$

Aslında bu fonksiyonun temelleri Legendre'nin geliştirdiği $\left(\frac{a}{n}\right)$ Legendre sembolüne dayanmaktadır. Legendre Sembolü'nün aldığı değerler ve açıklamaları Çizelge 5.2'de belirtilmiştir (Menezes ve ark, 1997).

Çizelge 5.2 : Legendre Sembolü Değerleri ve Açıklamaları

$\left(\frac{a}{n}\right)$	+1	eğer *a*, mod *n*'ye göre "quadratic residue"[33] ise
	-1	eğer *a*, mod *n*'ye göre "non-quadratic residue"[34] ise
	0	Eğer *a*, *n*'i bölerse

Euler'in teoremine göre, obeb(a,n) =1 ve *n* bir asal sayıysa Formül 5.5'deki eşitlik sağlanmaktadır. Bu teorem verilen *n* sayısının asal olup olmadığını test etmek için kullanılabilir. Sonuç olarak *a* tabanına göre Euler sözde asalı Formül 5.6'da olduğu gibi de ifade edilebilir. Yeteri kadar *a* sayısı denediği zaman *n*'in asal olup olmadığı anlaşılabilmektedir.

$$\left(\frac{a}{n}\right) = a^{(n-1)/2} \pmod n \qquad \text{(Formül 5.5)}$$

$$a^{(n-1)/2} = \left(\frac{a}{n}\right) \pmod n \qquad \text{(Formül 5.6)}$$

PGP'nin eski versiyonları $a^{(n-1)}$ değerini sadece bir *a* değeri için ölçüyorlar ve eğer sonuç 1 ise *n*'in asal olduğunu varsayıyorlardı. Ama Carmichael Sayıları bütün *a* değerleri için $a^{(n-1)} = 1 \ mod \ n$ eşitliğini sağlamaktadır ve asal değildir. Bu sayıların ender bulundukları belirtilmektedir.

$a^{(n-1)/2} = 1 \ mod \ n$ eşitliğini seçeceğimiz *a* değerleri için sağlayan ve asal olmayan sayılar da olabilir. Eğer *n* asal değilse ve *t* adet rastsal seçilmiş *a* sayısı kullanıldıysa, sonuç olarak *b* değerinin 1 veya –1 den

---

[33] Eğer *n* asal; *a* sayısı 0 'dan büyük ve *n*'den küçük ise ve $x^2 = a \ (mod \ n)$ değerini sağlayan herhangi bir *x* değeri varsa, *a* sayısı *mod n*'ye göre bir "quadratic residue" dür. Bütün a değerleri bu özelliği sağlamamaktadır. Mesela mod 7'ye göre quadratic residue'ler 1,2 ve 4 olmaktadır. (Schneier, 1996)

[34] "Quadratic nonresidue" ler ise bu özelliği sağlamayan sayılardır. Mesela mod 7'ye göre 3,5 ve 6 "quadratic nonresidue" olmaktadır. (Schneier, 1996)

farklı sayı gelme olasılığı en azından $2^{(-t)}$ olmaktadır. t sayısı büyük alınırsa bu olasılık istenildiği kadar azaltılabilir ve böylece bir sayının asal olup olmadığından daha emin olabiliriz.

5.2.5 Slovay & Strassen Testi

Public-key kriptografide kullanılmış ilk testdir. Kendisinden daha hızlı ve en az onun kadar doğru olan *Miller&Rabin*'in kullanılması önerilmektedir (Menezes ve ark, 1997).

Slovay&Strassen algoritmasında n sayısının asallığını bulmak için Jacobi sembolü kullanılmaktadır. Jacobi sembolü, n asalsa Legendre sembolüne eşit olmaktadır (Detaylı bilgi için bakınız bölüm 5.2.4.). Algoritmanın aşamaları aşağıdaki gibidir (Schneier, 1996):

1. n 'den ufak rastsal bir sayı olan a seçilir
2. Eğer gcd (a,n) $\neq$ 1 ise o zaman n asal değildir ve testi geçemez
3. $j = a^{(n-1)/2} \bmod n$ hesaplanır
4. Jacobi sembolü olan J (a, n) hesaplanır
5. Eğer $j \neq$ J (a, n) ise n kesin olarak asal degildir ve testi geçemez
6. Eğer j = J (a, n) ise n'nin asal olmama olasılığı %50'den fazla değildir.

5.2.6 Miller&Rabin (M&R) Testi

Miller&Rabin testi, literatürde "güçlü asallık testi" olarak geçmektedir. *M&R* testinde, tek sayı olan n'in asal olup olmadığını test etmek için ilk önce Formül 5.7'u sağlayan s ve r değerleri hesaplanır (Menezes ve ark, 1997).

$$n - 1 = 2^s r \qquad \text{(Formül 5.7)}$$

[1, n-1] aralığından taban olarak kullanılacak bir a değeri seçilir. Formül 5.8 veya Formül 5.9'deki eşitlik sağlanıyorsa n sayısının a tabanına göre güçlü asal olduğu kabul edilir (Menezes ve ark, 1997).

$$a^r = 1 \pmod n \qquad \text{(Formül 5.8)}$$

$a^{2^j r} = 1 \pmod{n} \quad (0 \le j \le s-1)$ (Formül 5.9)

n sayısının asallığını test ederken; *n* aday asal sayısı 2'den büyük ve tek herhangi bir tamsayı, *a* taban (*base*) değeri ise 2 ... n-1 dizisinden seçilmiş rastsal bir sayı olsun. C(n,a) bileşik (*composite*) "boolean" fonksiyonunun özellikleri aşağıda belirtilmiştir (SCM, 2000):

- Eğer *n* asal ise, C(n,a) fonksiyonu sonucu "2 ... n-1" aralığındaki her *a* için yanlış (false) olmalıdır.
- Eğer *n* bileşik ise C(n,a) fonksiyonu sonucu "2 ... p-1" aralığındaki *a*'ların en fazla ¼ 'ü için yanlış olmalıdır. Eğer taban *a* için test başarısız olursa; *n*, *a tabanına güçlü sözde asal* olarak tanımlanır.

Bu test, diğer rastsal tabanlar için tekrarlanabilir. *n* sayısının asallık testini *1/4 n +1* adet taban için uygulayacak zamana sahip olunsaydı *n*'nin asal olup olmadığı kesin olarak ortaya çıkardı. Ama büyük *n* değerleri için bu çok zaman alacağı için sadece belirli bir kısmında bu uygulanmaktadır ve olasılık (*probability*) devreye girmektedir. Ne kadar fazla tabanla bu işlem yapılırsa bu sayının asal olmama olasılığı daha da azalacak, *t* adet taban için testi geçen sayının asal olmama olasılığı maksimum $(1/4)^t$ olacaktır. Mesela test 30 kez tekrarlandığında, *testi geçen sayının* asal olmama olasılığı en fazla $8{,}3 \times 10^{-25}$ olacaktır. Bu da: 0,000000000000000000000000083 olmaktadır (SCM, 2000). Daha gerçekçi hesaplamalar da yapılmıştır. *x* bit asal adaylarında (x>100), bir tabanla uygulanan bir testin hatalı sonuç döndürme olasılığı *k* katsayısı için $1 / 4x\,2^{(k/2)^{(1/2)}}$ 'den daha düşük olmaktadır. 256 bitlik bir *n* sayısı için, 6 test sonucunda hatalı cevap alma olasılığı 2^{-51}'den daha düşük olmaktadır (Schneier, 1996).

Miller&Rabin testinde, bileşik sayıların asal sanılma olasılığı daha azalmaktadır. *a* sayılarının en az 1/3 'ünün tanık olması garantidir (Schneier, 1996). Bu test için yalancı-tanıkların çok az olduğu Higgins'ın yaptığı araştırmada (Higgins, 2000) ortaya çıkmıştır. Bununla birlikte, testi geçen bileşik sayıların varlığı da bilinmektedir. Alford, Granville ve Pomerance tarafından bu bileşik sayıların varlığı kanıtlanmıştır. Bleichenbacher, 100'den küçük ve eşit tabanlar kullanıldığında *Miller&Rabin* testini geçen 55 basamaklı bir bileşik sayı bulmuştur. *Miller&Rabin* testinin 2, 3, 5, 7, 11, 13 ve 23 tabanlarına göre uygulandığında 10^{16}'dan küçük sayılar için doğru bir asallık testi olduğu

kanıtlanmıştır. Çeşitli tabanlar için kullanılabilecek aralıklar Jaeschke tarafından belirlenmiştir (Maurer, 1994). Bu ve benzeri ispatların referansları ayrıntılı olarak Maurier'in makalesinde (Maurer, 1994) bulunmaktadır.

Uygulama

n rastsal sayısının *M&R* asallık testi için ilk önce *n-1*'in ikiye bölünme sayısı olan *s* değeri ile Formül 5.7'deki eşitliği sağlayan *r* değeri hesaplanır. Geriye kalan aşamaların adım adım uygulanması aşağıda verilmiştir (Schneier, 1996):

1. *n* den küçük olacak bir rastsal *a* sayısı bulunur.
2. $j = 0$ olarak ayarlanıp $z = a^r \bmod n$ hesaplanır.
3. Eğer $(z = 1)$ veya $(z = n - 1)$ ise *n* asallık testini geçer ve asal olabilir.
4. Eğer $(j > 0)$ ve $(z = 1)$ ise *n* asal değildir.
5. $j = j + 1$ olarak ayarlanır. Eğer $(j < s)$ ve $(z \neq n-1)$ ise $(z = z^2 \bmod n)$ olarak ayarlanır ve 4. Adıma geri dönülür. Eğer $(z = n - 1)$ ise *n* asallık testini geçer ve asal olabilir.
6. Eğer $(j = s)$ ve $(z \neq n - 1)$ ise o zaman *n* sayısı asal değildir.

M&R Testinde Asal Taban Almanın Avantajları[35]

Güçlü asallık testi olarak adlandırılan *M&R* testini yanıltan sayılar *güçlü yalancı tanık* olarak adlandırılmaktadır. Bazı bileşik sayılar, çok az sayıda güçlü yalancı tanığa sahiptir. Mesela bileşik 105 (3 x 5 x 7) sayısının yalancı tanıkları 1 ve 104'tür. Buradan varılan genelleme şudur: Bir *n* sayısı 2 veya daha fazla ilk tek asal sayının çarpımından oluşuyorsa bu sayının yalancı-tanıkları sadece 1 ve *n*-1 olmaktadır. Bu yüzden *Miller&Rabin* testinde güçlü yalancı tanıklar yüzünden yanlış sonuçlara varılmaması için taban olarak ilk asalların (2, 3, 5, 7 gibi) alınması

---

[35] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Menezes ve ark, 1997) alınmıştır.

önerilmektedir. Birçok bileşik sayı için güçlü yalancı tanıkların adedi, $Q(n)$[36] fonksiyonu için maksimum $Q(n) / 4$ olmaktadır.

İlk t adet asal tabana göre *Miller&Rabin* testinde asal olduğu sonucu çıkan en ufak pozitif bileşik sayı (Ψt) değerleri Çizelge 5.3'de gösterilmiştir. $n < \Psi t$ sayısı için belirtilen t sayısında ilk asal tabanı kullanmak yeterli olacaktır. Elde edilen sonuç her zaman doğru olacaktır.

Çizelge 5.3: *Miller&Rabin* testindeki en ufak Ψt değerleri

t	Ψt
1	2.047
2	1.373.653
3	25.326.001
4	3.215.031.751
5	2.152.302.898.747
6	3.474.749.660.383
7	341.550.071.728.321
8	341.550.071.728.321

5.2.7 Frobenius Testi[37]

Frobenius Sözde Asalı, Sonlu Alan (*finite field*) kuramına dayanmaktadır. Ayrıca bu kurama dayanan *Frobenius* testinin diğer testlerin genelleştirilmesi ve güçlendirilmesi ile oluşturulduğu belirtilmektedir.

---

[36] Euler phi (totient) fonksiyonu $Q(n)$, $n \geq 1$ için [1,n] aralığında n 'e göreceli asal olan sayıların adedini vermektedir.

[37] Bu bölümde kaynak olarak (Grantham, 1998a) alınmıştır

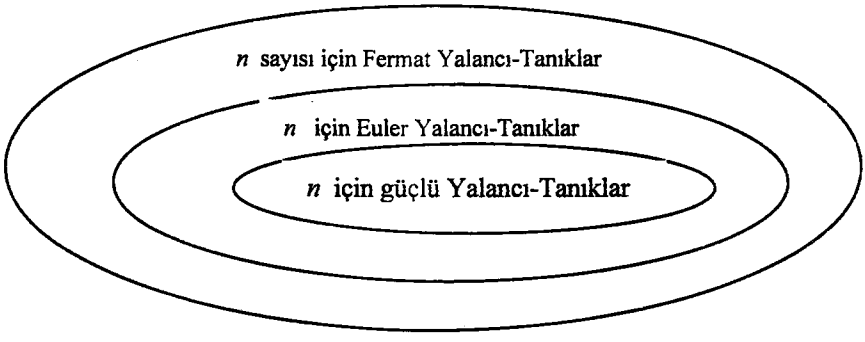
Frobenius testi ile bir bileşik sayıyı asal olarak belirleme hata oranının 1/7710 olduğu ölçülmüştür. *Frobenius* testinin *M&R* testinden üç kat daha yavaş çalıştığı ama 3 round'lu *M&R* testinin hata oranının en fazla $(1/4)^3=1/64$ olduğu belirtilmiştir. Bu da *Frobenius* testinin aynı zaman aralığında *M&R* testinden daha az hata oranı ile çalıştığını göstermektedir. *Frobenius* testi ile ilgili detaylı açıklama (Grantham, 1998a) ve (Grantham, 1998b) 'de bulunmaktadır.

5.2.8 Testlerin Karşılaştırması[38]

Lehmann testi, *Slovay&Strassen*'in bir varyasyonudur. *Lehmann* dışındaki diğer üç temel teste baktığımızda, *n* bileşik sayısının asal olduğu sonucunu verecek yalancı tanık kümelerindeki kesişmeler aşağıda belirtilmiştir:

- *a* sayısı, *n* sayısı için *Euler Yalancı Tanığı* ise aynı zamanda *Fermat Yalancı Tanığı*'dır.
- *a* sayısı, *n* sayısı için *Güçlü (Miller&Rabin) Yalancı Tanığı* ise aynı zamanda *Euler Yalancı Tanığı*'dır.

Yalancı Tanıklık Kümeleri, Şekil 5.1'de daha açık bir şekilde gözükmektedir.



Şekil 5.1 : Yalancı Tanıklık Kümeleri

[38] Bu bölümde aksi belirtilmediği sürece kaynak olarak (Menezes ve ark, 1997) alınmıştır.

Bu üç temel testi birbiriyle karşılaştırdığımızda *Miller&Rabin* testinin daha iyi olduğu ortaya çıkmaktadır. Bunun nedenleri aşağıda belirtilmiştir:

- *Fermat* testi, Carmichael sayılarını bulmakta zayıf kalmaktadır.
- *Solovay&Strassen* testi, çalışma zamanı olarak daha fazladır.
- *Solovay&Strassen* testi, Jacobi sembol hesaplamaları yüzünden uygulanması daha zordur.
- *Slovay&Strassen* testi $(1/2)^t$ hata payıyla çalışırken *Miller&Rabin* testi $(1/4)^t$ hata payıyla daha gerçeğe yakın sonuçlar sunmaktadır.
- *Miller&Rabin* testi en kötü şartlarda bile (*worst case*) en fazla diğerleri kadar çalışma zamanına ihtiyaç duymaktadır.

Yukarıda belirtilen nedenlerden dolayı tez uygulamasında bu üç asallık testinden sadece *Miller&Rabin* seçilmiş, bunlara ek olarak *Lucas* ve *Lehmann* testlerinin de uygulamaları yapılmıştır. Olası asallık testlerinin hata olasılıkları Çizelge 5.4'te verilmiştir. *Frobenius* testinin uygulaması yapılmamıştır.

Çizelge 5.4: Olası Asallık Testlerinin Hata Olasılıkları

Test	**Bir taban için Hata olasılığı**	***n* taban için Hata Olasılığı**
Miller&Rabin	1/4	$(1/4)^n$
Lehmann	1/7710	$(1/7710)^n$
Slovay/Strassen	1/2	$(1/2)^n$
Frobenius	1/2	$(1/2)^n$

5.2.9 Ufak Asallara Bölme

Kesin veya Olası Asallık testleri uygulanmadan önce bu sayıların Ufak Asallara Bölme testinden geçirilmesi testlerde hızlanma sağlayabilmektedir. Bölünecek asal sayı adedinin dikkatlice seçilmesi gerekmektedir. Tek sayıların 3,5 ve 7 ile bölüp bölünmediğinin testinin

bu sayıların %54'ünü, 100'den küçük asallara bölmenin %76'sını, 256'dan küçük sayılara bölmenin ise %80'ini elediği belirlenmiştir. Herhangi bir asalın katı olmayan n'den küçük tek sayı asallarının oranı $1,12 / ln\ n$ olmaktadır (Schneier, 1996).

Ronald L. Rivest, 33 SUN Sparcstation iş istasyonundan oluşan bir ağ ortamında yaklaşık 718 milyon adet 256 bitlik sayıyı ilk önce "ufak bölen" testine, bu testi geçenleri de 2 tabanına göre *Fermat* testine tabii tutmuştur. Ufak Bölen testini 43.741.404 sayı geçmiş, bunların 4.058.000 adedi *Fermat* testini geçebilmiştir. Her iki testi de geçen sayılar 8 iterasyonlu *Miller&Rabin* testine sokulmuştur. Bu testler göstermiştir ki ilk iki testi geçen sayılar, son testi de geçmektedir yani asal olması muhtemeldir. Bu da istatistiksel olarak, ufak böleni olmayan sayıların sözde (*pseudo*) asal olma ihtimalinin çok az olduğunu göstermektedir (Rivest, 1990).

5.3 Bileşik Testler[39]

Miller&Rabin testini uyguladıktan sonra *Lucas* testi veya t adet *Frobenius* testi uygulanarak asallık testinin daha iyi sonuçlar vermesi sağlanabilir. *Miller&Rabin* testinden sonra t adet *Frobenius* testi kullanıldığında, 512 bitlik bir asal sayının hatalı olma (bileşik olma) olasılığı $1,5 * 10^{-17} * (1/1770)^t$ den az olmaktadır.

Miller&Rabin testinin, *Lucas* testi ile birlikte kullanılması önerilmektedir. Şu ana kadar bu iki testi birlikte geçen bir bileşik (*composite*) sayı bulunmamıştır. Bu iki test birlikte kullanıldığında oluşabilecek hata olasılığını matematiksel olarak hesaplayan bir formül bulunmamaktadır.

---

[39] Bu bölümde aksi belirtilmediği sürece kaynak olarak (Silverman, 1997) alınmıştır

5.4 Pratikte Asal Sayı Üretme Esasları

Asal sayı üretmek için aşağıda belirtilen aşamalar adım adım uygulanmalıdır:

1. *n*-bit rastsal sayı *p* üretilir.
2. İlk (high order) bit ve son (*low order*) bit'ler 1 olacak şekilde ayarlanır (Son bit'in 1 olması o sayının tek olmasını sağlamakta, ilk bit'in 1 olması da asal sayının istenilen (*required*) uzunlukta olduğunu belirlemektedir).
3. *p*'nin ufak asal sayılara (3,5,7,11 ...) bölünmediği kontrol edilmelidir. Birçok uygulamada 256'dan küçük bütün asallarla kontrol ederken, en etkin olanı 2000'den küçük asallara bölünmediğini kontrol etmektir.
4. *Miller&Rabin* testi bir rastsal *a* değeri için uygulanır. Eğer *p* bu testi geçerse başka rastsal *a* değerleri için bu test tekrarlanabilir. Ufak *a* değerleri seçilmelidir ki hesaplamalar daha hızlı olsun. Testde kullanılacak a değerlerinin sayısı 5 (Seth, 1999) veya 10 (Segre, 2000) olarak seçilebilir. Eğer *p* değeri bu testlerden birisini geçemezse asal bir sayı değildir ve başka bir *p* değeri yaratılıp aynı işlemlerden geçirilmesi gerekecektir (Schneier, 1996; Seth, 1999).
5. *Miller&Rabin* testini geçen sayılar, *Lucas* veya *Frobenius* testine sokularak daha güvenilir sonuçlar elde edilebilir (Silverman, 1997).

Her seferde rastsal *p* değeri oluşturmak yerine, ilk seferden sonra başlangıç rastsal sayısını arttırarak asal bir sayı bulana kadar da işleme devam edilebilir (Schneier, 1996 ; Seth, 1999).

Asallık testinin yeterince kuvvetli olması için, testde sayının basamak adedi kadar taban (*a* değeri) alınması tavsiye edilmektedir (Pinch, 1993). Dikkat etmemiz gereken temel kriter, hata oranının 2^{-100} 'den daha büyük olmaması gerektiğidir (Silverman, 1997). Digital Signature Standard (DSS)'de; üretilen sayıların asallığını test ederken, *Miller&Rabin* algoritmasının en az 50 kez kullanılmasıyla kabul edilecek bir hata olasılığına, yani 2^{-100} 'e ulaşılabileceği belirtilmektedir (Burrows, 1994).

Miller&Rabin ile birlikte *Lucas* veya *Frobenius* testlerinin birlikte kullanılması önerilmektedir. Bu iki testi birden geçen bir sayı henüz bilinmemektedir (Silverman, 1997).

6 UYGULAMALAR VE SONUÇLAR: RASTSAL SAYILAR

Bu bölümde rastsal sayı üretmek için kullanılan üreteçler ve rastsallık testleri ele alınacaktır. Test sonuçları verilecek ve varılan sonuçlar açıklanacaktır.

6.1 Rastsal Sayı Üreteçleri

Rastsal sayı üreteci olarak kullanılan üreteçler aşağıdaki gibidir:

- Sistem Hakkında bilgi Toplayan Perl Script
- Blum Blum & Shub Üreteci

6.1.1 Sistem hakkında bilgileri toplayan Perl Script

Bu scriptle Unix'te var olan sistem fonksiyonları çağrılarak sistem ve ağ (*network*) durumu, sanal bellek istatistikleri ve süreç durumları gibi devamlı değişen bilgilerin bulunduğu bir dosya oluşturulmaktadır. Bu bilgiler daha sonra bir girdi olarak MD5 hash fonksiyonuna verildiğinde oluşacak olan 128 bitlik sayının rastsal olduğu öngörülmektedir (Garfinkel ve ark, 1996). Kullanılan perl script'in kaynak kodu EK-4 'de verilmiştir.

Kullanılan Fonksiyonların Açıklanması[40]

Bu scripteki ana fonksiyonlar bazı parametreleriyle aşağıda açıklanmıştır:

- **Vmstat :** Süreçler, sanal bellek, disk, trap ve merkezi işlem birimi(cpu) aktiviteleri hakkında rapor verir. *"vmstat" komutu* bazı sürücüler için desteklenmektedir ve çeşitli parametreleri vardır. Örneğin, *"-s"* parametresi makinanın boot olmasından bu yana olan event'lerin sayısını, *"-i"* parametresi her cihaz için kesme (*interrupt*) sayısını ve *"-c"* parametresi "Cache

[40] Kaynak olarak (SUN, 2000) alınmıştır.

flushing" istatistiklerini raporlamaktadır. "*vmstat 5*" komutuyla her beş saniyede bir sistemin yapılanları rapor etmesi sağlanabilir. Bu komut çalıştırıldığında elde edilen ekran çıktısı aşağıda verilmiştir.

```
sun% vmstat 5
 procs     memory            page            disk        faults      cpu
 r b w   swap  free  re  mf  pi  po  fr  de  sr s0 s1 s2 s3   in   sy  cs us sy
id
 0 0 0 11456 4120   1  41  19   1   3   0   2  0  4  0  0  48  112  130   4 14  82
 0 0 1 10132 4280   0   4  44   0   0   0   0  0 23  0  0 211  230  144   3 35  62
 0 0 1 10132 4616   0   0  20   0   0   0   0  0 19  0  0 150  172  146   3 33  64
 0 0 1 10132 5292   0   0   9   0   0   0   0  0 21  0  0 165  105  130   1 21  78
 1 1 1 10132 5496   0   0   5   0   0   0   0  0 23  0  0 183   92  134   1 20  79
 1 0 1 10132 5564   0   0  25   0   0   0   0  0 18  0  0 131  231  116   4 34  62
 1 0 1 10124 5412   0   0  37   0   0   0   0  0 22  0  0 166  179  118   1 33  67
 1 0 1 10124 5236   0   0  24   0   0   0   0  0 14  0  0 109  243  113   4 56  39
```

- **Netstat:** Ağ durumunu (*network status*) gösterir. Ağla ilişkili çeşitli veri yapılarının içeriklerini değişik biçimlerde sunar. Değişik parametreleri vardır. Örneğin, "*netstat –a*" komutu server süreçlerininkiler hariç bütün soketlerin ve bütün "routing table" içeriklerini gösterir.
- **Nfsstat:** NFS ve RPC ile ilgili istatistiksel bilgileri gösterir. Örneğin, "*nfsstat –cnrs*" komutu ile bütün bilgilerin gösterilmesi sağlanabilir.
- **Ps:** Süreç durumlarını rapor eder. Hiç bir parametre girilmediğinde sadece o terminale ait süreçleri belirtecektir. Parametreler doğru girildiğinde sistemdeki bütün süreçler hakkında bilgi almak mümkündür.
- **Diğer:** Kulllanılan diğer komutlara örnek olarak *ls*, *getppid* ve *time* verilebilir. *ls* komutu dosyaları listeler. *getppid* komutu,

ana süreç (*Parent Process*) Id'sini döndürür. *time* komutu, zaman bilgisini döndürür.

Perl Üretecinin Unix İşletim Sisteminde Kullanılması

Üreteç Unix ortamında çalıştırıldığında ilk olarak üretilecek rastsal sayının kaydedileceği çıktı dosyasının adını istemektedir. Daha sonra kaç adet 128 bit (16 byte) rastsal veri istendiği kullanıcıdan talep edilir. Üreteç her birim çalışmasında (döngüde) 128 bit (16 byte) ürettiğinden ardışık çalışmalar arasında en az 4.6 saniye çalışmasına ara verip sistem bilgilerinin değişmesini bekleyecektir. Ardışık üretilen 16 byte'lık verilerin aynı satıra aralarına boşluk konulmadan yazılması için program *"-n"* parametresiyle çalıştırılmalıdır. Programın 1280 bit üretmek için çalıştırılması Şekil 6.1'de gösterilmiştir. Oluşan dosya bir editör aracılığıyla incelendiğinde onaltılık (*hex*) basamaklardan oluştuğu görülebilir.

```
**PHARM**/usr/users/staff/enisk/md5>UretecSon -n
*************************************************
* Sistem Bilgilerinden Rastsal Tohum-Veri Uretimi *
*    Bitirme Projesi -1999-2000 - Enis Karaaslan  *
*************************************************
Verilerin yazilacagi cikti dosyasinin adini giriniz:
cikti0531
Kac adet 16 Byte'lik Rastsal Bit üretilecegini giriniz :10
1280 bit uretilecek. Lutfen Bekleyiniz
..........
***** islem sonu *******

/usr/users/academic/enis/md5/md5-c>pico cikti0531
5dddce57cc97690c905843233d68ea26c53fae1afdd738e85f1014ccb4e
3f27e53101138c6ebfb0ee833db42aecc7c61caf5b1e05edc952136476d
8d0f3081e83cc7220b248c6ff5fd50a393bf6212c344f848c440f48a439
6c5fa6eee2bc8677a135f2ddd3b82e3bcfd2ba78b5ea1a184d217758eb0
6ad386d8f13114d951e95eb5bd3b5fe8f9133fbfe8d126aa39fdcf9154e
a12adfface0b211b554ed5c4d
```

Şekil 6.1 : Sistemden Bilgi Toplayan Üretecin Ekran Çıktısı

6.1.2 Blum, Blum, Shub Üreteci (BBSG)

Gerçekten tahmin edilemeyecek sayılar elde etmek için bu üreteç yaygın olarak kullanılmaktadır. Bu üreteç hakkında detaylı bilgi bölüm 2.7'de verilmiştir.

İki farklı üreteç üzerinde çalışma yapılmıştır. Bunlar; ilk 1028 asalı taban alıp çalışan *Blum Üreteci* ve rastsal verilerden istenilen basamak adedinde asal taban oluşturup üzerinde çalışan *BlumPro Üreteci*'dir. Bahsedilen bu iki üreteç GAP ortamında kodlanmıştır ve kaynak kod EK-5'de verilmiştir.

Blum Üreteci

p ve q asallarını bulurken ilk 1028 asal kullanılmıştır. Bu ilk 1028 asaldan 518 adedi (x mod 4 = 3) eşitliğini sağlamaktadır. Programın başında bu 518 adet sayının hesaplanmasını yapmak yerine bu sayıların bir dizi olarak koda eklenmesi hesaplamayı hızlandıracaktır. GAP ortamının sağladığı rastsal sayı üreteçlerinden "*Random(list)*" komutu kullanılarak bu 518 asaldan eşit dağılımlı bir seçim yapılmaktadır. Hızın önemli olduğu uygulamalarda yine GAP ortamının sağladığı "*Pseudorandom(list)*" komutu kullanılarak eşit dağılıma yakın seçimler yapılabilmektedir.

Uygulamada "*Random(list)*" komutu kullanılarak bu 518 asaldan p ve q değerleri seçilmiştir. p ve q değerlerinin eşit olabilme ihtimali de olduğundan oluşabilecek farklı p ve q değişkenlerinin adedi "518 * 518 = 268,324" olmaktadır.

Uygulamada kullanılacak s rastsal değeri de 10,000,000 adet sayıdan oluşan bir listeden seçilmektedir ve bu sayının n değeriyle aralarında asal olması şartı bulunmaktadır. Programa istenilen bit adedi girilmekte ve o sayıda üretilen rastsal bit'ler ekrana yazdırılmaktadır.

Teorik olarak polinom-zamanlı saldırılara karşı üretecin güçlü olması için n'nin 1,000,000'dan büyük bir sayı olması gerekmektedir (Ritter, 2000).

BlumPro Üreteci

BlumPro fonksiyonu ile verilen rastsal hex dosyasından istenen basamak adedinde p ve q asal değerleri üretilmekte ve bu işlemlerin sonucunda n değeri döndürülmektedir. Büyük n değerleriyle kriptografik olarak daha güvenilir diziler oluşturulabilmektedir.

Fonksiyonlar

GAP ortamında yazılan kodun ana fonksiyonları aşağıda belirtildiği gibidir:

- **Blum (BitAdedi, AsalFlag):** *AsalFlag* değeri olarak "0" verildiğinde ilk 1028 asaldan üretilen *n* değeriyle, "1" verildiğinde GAP ortamının rastsal üretecinin sağladığı değerlerden üretilen *n* değeriyle çalışır. "0" değeri kullanıldığında daha hızlı sonuç vermektedir.
- **InitBlumPro (DosyaAd, BaseAdedi, AsalBasamak, AsalFlag, Istek, CiktiDosya):** *AsalFlag* değeri olarak 1 değeri verildiğinde DosyaAd ile belirtilen dosyadaki rastsal onaltılık(*hex*) basamaklardan, istenilen basamak (*AsalBasamak*) adedinde *p*, *q* ve *s* değerlerini üretir. *AsalFlag* değeri olarak 0 değeri verildiğinde ilk 1028 asaldan *p*,*q* ve *s* değerleri üretilmektedir. Bu değerler kullanılarak *n* değeri oluşturulmakta ve *BlumPro()* fonksiyonu tarafından kullanılmaktadır.
- **BlumPro (BAdet, BitAdedi):** *InitBlumPro()* fonksiyonu ile belirlenen değerler kullanılarak bir döngü içerisinde *BAdet* adet *BitAdedi* sayısında rastsal bit dizileri üretilir.
- **OkuHex (DosyaAd, BasamakAdedi):** Verilen dosya adından *BasamakAdedi* sayısı kadar hex basamak okur.
- **Hextoint (Sayi, BasamakAdedi):** *Sayi* değişkeninden Okunan hex digitlerden *BasamakAdedi* nde bir asal sayı oluşturur.
- **Sayisal (BitDizisi):** Üretilen rastsal bitleri tam sayıya dönüştürür.

Örnek çıktılar EK-6'da verilmiştir.

6.2 Rastsallık Test Bataryalarının Kullanımı

Literatürde yüzlerce rastsallık testi mevcut olup halen yenileri de geliştirilmektedir. Pek çok alternatif arasından uygun rastsallık test bataryalarını seçmede kriter olarak güvenilirlik, standartlık ve kolay uygulanabilirlik esas olarak alındı.

Tez uygulamalarında kullanılan test bataryaları aşağıdaki gibidir:

- FIPS 140
- ENT
- NIST STS

Bu bölümde bu uygulama paketlerinin nasıl kullanıldığı örneklerle anlatılmıştır.

6.2.1 FIPS 140 Paketi

Şifreleme amaçlı kullanılan rastsal sayı üreteçlerinin FIPS 140-2 standardına göre Monobit, Poker, Koşu (*Runs*), Uzun Koşu (*Long Run*) testlerine tabi tutulması önerilmektedir. Detaylı bilgi bölüm 3.3.4'de verilmiştir. C programlama dilinde kodlanılan FIPS140'ın kaynak kodu Ek-7'de verilmiştir.

FIPS deneyi, DOS komut satırında dosya ismi verilerek çalıştırıldığında gelen menü 4 testden bir tanesini veya programdan çıkış isteğini seçmemizi isteyecektir.

```
FIPS 140-1 TESTLERI
 [1]Monobit Testi
 [2]Poker Testi
 [3]Kosu Testi
 [4]Uzun Kosu Testi
 [5]Programdan cikis
Seciminizi Girin lutfen : 1
```

Monobit testini seçmek için "1" seçeneği girildiğinde 0 ve 1 lerin bulunma olasılıkları ve testi geçip geçmediği ekranda gösterilecektir.

```
Deger    Karakter SIKLIGI   Kesir
  0              9981      0.499050
  1             10019      0.500950
Toplam:         20000      1.000000
Testi geçti
```

Koşu testi için de "3" seçeneği girildiğinde beklenen koşu değerleri aralığı ve gözlenen değerler belirtilecektir.

```
0 için :
 Beklenen 1 uzunlugunda kosu 2267-2733 gozlenen : 2508
 Beklenen 2 uzunlugunda kosu 1079-1421 gozlenen : 1265
 Beklenen 3 uzunlugunda kosu  502- 748 gozlenen : 646
 Beklenen 4 uzunlugunda kosu  223- 402 gozlenen : 305
 Beklenen 5 uzunlugunda kosu   90- 223 gozlenen : 137
 Beklenen 6 uzunlugunda kosu   90- 223 gozlenen : 157
1 e basip devam edin : 1

1 için :
 Beklenen 1 uzunlugunda kosu 2267-2733 gozlenen : 2559
 Beklenen 2 uzunlugunda kosu 1079-1421 gozlenen : 1182
 Beklenen 3 uzunlugunda kosu  502- 748 gozlenen : 648
 Beklenen 4 uzunlugunda kosu  223- 402 gozlenen : 306
 Beklenen 5 uzunlugunda kosu   90- 223 gozlenen : 172
 Beklenen 6 uzunlugunda kosu   90- 223 gozlenen : 150
```

Aynı şekilde diğer testler de uygulanabilir. Programdan çıkmak için "5" seçeneğinin girilmesi yeterli olmaktadır.

6.2.2 ENT Paketi[41]

Ent paketi, herhangi bir program veya veri dosyasını, byte veya bit bazında rastsallık testine tabii tutan ve sonuçlarını bir rapor şeklinde sunan bir programdır. Ent programı DOS veya UNIX ortamlarında çalışabilmektedir. Eğer bir dosya belirtilmemişse komut satırından girilenleri girdi olarak alacaktır. Çıktı her zaman ekranda verilmektedir. Programı çalıştırırken kullanılabilecek parametreler Çizelge 6.1'de gösterilmiştir.

Çizelge 6.1 : Ent Programı Parametreleri ve Açıklamaları

Parametre	Açıklama
-b	Dosya bit bazında değerlendirilir.
-c	Her byte'ın (-b parametresi kullanıldıysa bit'in) kaç kez geçtiğini bir tablo olarak sunar.
-f	Deneyden önce büyük harfleri küçük harflere dönüştürür.
-t	Bilgileri herhangi bir tablo veya programlama dilinde kolaylıkla kullanabilmek için aralarında virgül olacak şekilde yazdırır (*Terse Mode*).
-u	Parametreleri gösterir.

Programın değişik çalıştırma şekilleri örnek olarak aşağıda verilmiştir:

Ent dosya_ismi	(Byte bazında test yapar)
Ent –c dosya_ismi	(Karakterlerin bulunma sıklıklarını da yazar)
Ent –b dosya_ismi	(Bit bazında test yapar)
Ent -bc dosya_ismi	(Bit'lerin (0-1) bulunma sıklıklarını da yazar)

[41] Aksi belirtilmediği sürece bu bölümde kaynak olarak (Walker, 1998) alınmıştır.

Ent deneyine aşağıda belirtilen dosyalar sokulduğunda sonuçlar Çizclge 6.2'deki gibi olmaktadır:

- 1meg.1 :Random.org'dan 1Mb rastsal veri
- 1meg.2 :Random.org'dan 1Mb rastsal veri
- winnt256.bmp :Bitmap Dosyası
- norand.txt :Rastsal olmayan bir yazı
- ent.c :Ent programının kaynak kodu.

Çizelge 6.2 : Ent Deneyi Sonuç Tablosu

	Byte başına Entropi	Yapılabilecek sıkıştırma yüzdesi	Chi Square	Aritmetik Ortalama	Pi için Monte Carlo değeri (hata yüzdesi)	Seri Korelasyon Katsayıları
Beklenen	8	0	<%1, >%99 yetersiz	127,5	0	0,00
Winnt256.bmp	5,878008	26	0,01	161,1342	28.	0,488372
1meg.1	7,999814	0	25,00	127,5226	0.	0,001057
1meg.2	7,999866	0	99,50	127,4532	0.	-0,000018
Norand.txt	3,705558	53	0,01	87,8475	27.	0,874797
Ent.c	4,913949	38	0,01	70,6394	27.	0,492822

Çizelge 6.2'nin "beklenen" satırında ideal değerlerin nasıl olduğu gösterilmiştir. Rastsal olmayan dosyaların bu değerlerden nasıl farklı sonuçlar verdiği belirgin bir biçimde gözükmektedir. Random.org'dan çekilen "1meg.1" ve "1meg.2" dosyaları ideal değerlere çok yakın sonuçlar vermelerine karşın "1meg.2" dosyası Chi Square testini geçememiştir. Bu sorun bu dosyayı dağıtan www.random.org sunucusunun ilgililerine iletilmişse de henüz çözülememiştir.

6.2.3 NIST STS Paketi

NIST paketini kullanırken ilk olarak deneyin uygulanacağı örneklemin (*sample*) bit uzunluğunun parametre olarak belirtilmesi gerekmektedir. Kaç adet örneklem alınacağı ise sonraki aşamalarda belirtilmektedir. Mesela, aşağıdaki ekran çıktısında gösterilen *assess 10000* komutu kullanılarak 10.000 bitlik örnekler alınabilir.

```
**PHARM**/usr/users/staff/enisk/sts-1.3>assess 10000
_____________________________________________________________________

                        G E N E R A T O R   O P T I O N S
_____________________________________________________________________

[00] Input File                     [01] G Using SHA-1
[02] Linear Congruential            [03] Blum-Blum-Shub
[04] Micali-Schnorr                 [05] Modular Exponentiation
[06] Quadratic Congruential I       [07] Quadratic Congruential II
[08] Cubic Congruential             [09] XOR
[10] ANSI X9.17 (3-DES)             [11] G Using DES
_____________________________________________________________________

OPTION ---->
```

Bu aşamada belirli bir veri dosyasından veya hazır üreteçlerden veri alınabilir. *0* seçeneği girildiğinde deneyin uygulacağı veri dosyasının adı istenilecektir.

```
OPTION ----> 0
User Prescribed Input File: data/BBS.dat
```

Veri dosyasının adı girildiğinde karşımıza uygulanabilecek 16 istatistiksel testin listesi gelecektir. *1* seçeneği girildiğinde bütün testler uygulanacak, *0* seçeneği girildiğinde testler arasından seçim yapmak mümkün olabilecektir.

```
                    S T A T I S T I C A L   T E S T S

[01] Frequency                              [02] Block Frequency
[03] Cumulative Sums                        [04] Runs
[05] Longest Run of Ones                    [06] Rank
[07] Discrete Fourier Transform             [08] Nonperiodic Template
Matchings
[09] Overlapping Template Matchings         [10] Universal Statistical
[11] Approximate Entropy                    [12] Random Excursions
[13] Random Excursions Variant              [14] Serial
[15] Lempel-Ziv Complexity                  [16] Linear Complexity

                              INSTRUCTIONS
        Enter 0 if you DO NOT want to apply all of the
        statistical tests to each sequence and 1 if you DO.
```

0 seçeneği girildiğinde uygulanması istenilen testler için *1* diğerleri için *0* girilmesinin istenileceği bir ekrana ulaşılacaktır.

```
                    S T A T I S T I C A L   T E S T S

[01] Frequency                              [02] Block Frequency
[03] Cumulative Sums                        [04] Runs
[05] Longest Run of Ones                    [06] Rank
[07] Discrete Fourier Transform             [08] Nonperiodic Template
Matchings
[09] Overlapping Template Matchings         [10] Universal Statistical
[11] Approximate Entropy                    [12] Random Excursions
[13] Random Excursions Variant              [14] Serial
[15] Lempel-Ziv Complexity                  [16] Linear Complexity

                              INSTRUCTIONS
        Enter a 0 or 1 to indicate whether or not the numbered
        statistical test should be applied to each sequence.
        For example, 1111111111111111 applies every test to
        each sequence.

                              1234567891111111
                                      0123456
                              1111000000000100
```

Her teste özgü olan parametreler istenilecek ve kaç adet örneklem alınacağı kullanıcıdan istenilecektir.

```
How many bitstreams should be generated? 10000
```

Veri dosyasının türü yani bitlerden oluşan ASCII dosya mı yoksa onaltılık (*hex*) basamaklardan oluşan bir ikili (*binary*) dosya mı olduğunun belirtilmesi istenecektir.

```
[0] BITS IN ASCII FORMAT
[1] HEX DIGITS IN BINARY FORMAT

Select input mode:  1
```

Belirtilen seçeneklerle test işlemi başlayacaktır.

```
Statistical Testing In Progress.....
```

Deney bittiğinde deneyin bittiğine dair bir ekran çıktısı gelecektir.

```
Statistical Testing Complete!!!!!!!!!!!!
```

İstatistiksel analiz sonunda *FinalAnalysisReport* dosyası yaratılacaktır. Bu dosya bir editörle açılarak incelendiğinde görülecek deney sonuçları Şekil 6.2'de gösterilmiştir. Örneğimizde 4 adet test uygulandığı için o dört testin sonuçları bulunmaktadır. Eğer veri dizisi bu testlerden herhangi birinden geçememiş olsaydı o değerin yanında * işareti ile belirtilecekti. 10 eşit aralıktaki o-değerleri (p-value) sayılmış C1-C10 sütunlarında verilmiştir. 11. sütunda bulunan *"Proportion"* değeri bu 10 o-değerinin Chi-square testi kullanılarak o-değerinin hesaplanmasıyla bulunmaktadır.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING
SEQUENCES
------------------------------------------------------------------------------
------------------------------------------------------------------------------
  C1   C2   C3 C4   C5   C6 C7   C8   C9   C10   P-VALUE   PROPORTION  S.TEST
------------------------------------------------------------------------------
 984   962 1029 922 1094 955   953 1028   998 1075 0.000915 0.9895 Freq
1009   937 1055 996 1011 960   994 1004 1033 1001 0.354222 0.9917 Blck-Freq
 963  1020   985 913 1079 943 1000 1044 1003 1050 0.005170 0.9902 Cusum
 997   949 1010 936 1049 931 1018 1027 1039 1044 0.030016 0.9915 Cusum
 999   976 1023 962 998 1010 1010   987 1004 1031 0.917870 0.9910 Runs

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 0.987015 for a sample
size = 10000 binary sequences.
```

Şekil 6.2: NIST STS Analiz Raporu Örneği

6.3 Rastsal Sayı Üreteçlerinin Deney Sonuçları

FIPS 140-2, ENTve FIPS STS1.3 test bataryalarını kullanarak *MidSquare* (Kötülüğü tescilli bir üreteç (Knuth, 1998; L'Ecuyer, 1996)), *Myrand* (Sıradan bir LCG üretecin bilinçli olarak bozulmuş versiyonu), *Microsoft Excel 97*, *Borland C 5.0*, *Delphi* 5.0 ve KISS32 (Marsaglia, 1997) rastgele sayı üreteçleri aşağıdaki düzenekler kullanarak test edildi. Excel, Delphi ve Borland C rand üreteçleri farklı tohum (*seed*) değerleri ile test edildi. Üreteçlerin yanlarında verilen parametreler o üreteçle birlikte kullanılan tohum değerlerini vermektedir. Kullanılan parametrelerin karşılık geldiği değerler Çizelge 6.3 'de gösterilmiştir.

Çizelge 6.3: Parametrelerin karşılık geldiği tohum değerleri

Parametre	Tohum değeri
1	0
2	1997
3	time()

Çizelgelerde * ile işaretli sonuçlar başarı kriterini sağlayamamaktadır.

6.3.1 FIPS140-2 Deney Sonuçları

Her üreteç için 20,000 bitlik bir dizi oluşturulmuş ve FIPS140-2 standardında bahsedilen 4 testi bu dizi üzerinde uygulanıp örnek sonuçları Çizelge 6.4a ve 6.4b'de gösterilmiştir.

Çizelge 6.4a: FIPS 140-2 Deney Sonuçları

	Ideal	KISS32	Delphi(1)	Delphi(2)	MyRand	MidSquare
Monobit	[9725 , 10275]	9944	9972	10019	*9256	*6898
Poker	[1,03 , 46,17]	14.604	9.491	17.638	*196.24	*29415,05
Koşu_1	[2343 , 2657]	2478	2556	2508	*2290	*636
_2	[1135, 1365]	1200	1229	1265	1154	*3104
_3	[542, 708]	612	655	646	588	*1244
_4	[251, 373]	322	320	305	295	*621
_5	[111, 201]	167	141	137	156	*2
_6+	[111, 201]	173	152	157	*312	*4
UzunKoşu	0	0	0	0	0	0

Çizelge 6.4b: FIPS 140-2 Deney Sonuçları

	Ideal	Crand(1)	Crand(2)	Crand(3)	Excel(1)	Excel(2)
Monobit	[9725 , 10275]	10140	10046	9969	9980	9998
Poker	[1,03 , 46,17]	21,8944	14,2080	12,4160	12,1024	12,2880
Koşu_1	[2343, 2657]	2507	2408	2449	2527	2551
_2	[1135, 1365]	1240	1259	1256	1252	1212
_3	[542, 708]	643	629	629	651	655
_4	[251, 373]	293	301	317	316	294
_5	[111, 201]	141	172	154	158	175
_6+	[111, 201]	152	154	164	144	146
UzunKoşu	0	0	0	0	0	0

6.3.2 ENT Deney Sonuçları

FIPS140-2 için kullanılan diziler kullanıldı ve ENT paketindeki 5 testi bu diziler üzerinde uygulanıp örnek sonuçları Çizelge 6.5a ve 6.5b'de gösterildi. Çizelgede '?' ile işaretli olanlar seçilecek başarı kriterine göre başarısız olma olasılığı yüksek olan sonuçlardır.

Çizelge 6.5a : ENT Deney Sonuçları

	İdeal	KISS32	Delphi(1)	Delphi(2)	MyRand	MidSqr.
Entropy	8	7,999984	7,932521	7,929718	?7,833586	*2,115004
ChiSquare	<%1,>%99	259,04	232,65	233,47	677,47	154065,71
	Yetersiz	(%75,00)	(%75,00)	(%75,00)	*(%0,01)	*(%0,01)
ArthMean	127,5	127,5327	126,5828	128,4540	?121,5960	*147,7304
Pi	3,14159	3,14160	3,15384	3,06730	3,29807	2,009615
		(%0)	(%0,39)	?(%2,36)	?(%4,98)	*(%36,3)
SerialCoef.	0,00	–0,000150	–0,006970	?0,028210	*0,119787	0,001014

Çizelge 6.5b: ENT Deney Sonuçları

	Ideal	Crand(1)	Crand(2)	Crand(3)	Excel(1)	Excel(2)
Entropy	8	? 7,908555	?7,915205	7,929130	7,929320	7,929298
ChiSquare	<%1,>%99	312,93	284,87	241,86	239,40	232,65
	Yetersiz	?%1	%10	%50	%75,00	%75
ArthMean	127,5	? 129,7916	128,1044	128,1268	126,7756	128,638
Pi	3,14159	3,11538	3,14423	3,11538	3,24038	3,14423
		(%0,83)	(%0,08)	(%0,83)	?(%3,14)	(%0,08)
SerialCoef.	0,00	0,005302	0,017510	0,003091	0,003381	-0,019381

Not: KISS32 üreteci için DieHard paketi ile hazır gelen 11 Mbyte'lık hazır dosya kullanılmış, FIPS140 bunun ilk 20,000 bitini ENT ise tamamını girdi olarak kullanmıştır.

6.3.3 NIST STS.1.3 Deney Sonuçları

NIST-STS standardında verilen testlerden Frekans (*monobit*), Blok İçi Frekans, Ayrık Fourier Dönüşümü (*Spectral*), Seri, Yaklaşık Entropi, Cumulative Sums (*Cusum*) uygulandı.

Her üreteç için *dizi uzunluğu* = *n=1.000* bitlik *örnek adedi=m=1.000*[42] dizi oluşturuldu, anlamlılık düzeyi α= 0,01 için yukardaki deneylerin sonuçları Çizelge 6.6a ve 6.6b'de gösterilmiştir.

Çizelge 6.6a : NIST STS 1.3 Deney Sonuçları

		KISS32	Delphi(1)	Delphi(2)	MyRand	MidSquare
Frekans	o-değeri	0,000006*	0,000066*	0,000232	0,000000*	0,000000*
	Oran	0,9890	0,9660*	0,9680*	0,2250*	0,000000*
Blok Frek.	o-değeri	0,069430	0,373625	0,417219	0,000000*	0,000000*
	Oran	0,9940	0,9710*	0,9680*	0,2125*	0,000000*
Cusum	o-değeri1	0,001378	0,007975	0,000047*	0,000000*	0,000000*
	Oran1	0,9910	0,9660*	0,9720*	0,2250*	0,000000*
	o-değeri2	0,007862	0,014652	0,000011*	0,000000*	0,000000*
	Oran2	0,9890	0,9680*	0,9720*	0,2250*	0,000000*
Spectral	o-değeri	0,000000*	0,000000*	0,000000*	0,000000*	0,000000*
	Oran	1,0000	0,9760*	0,9760*	0,2625*	1.0000
Yaklaşık Entropi	o-değeri	0,189625	0,029796	0,064015	0,000000*	0,000000*
	Oran	0,9750*	0,9660*	0,9660*	0,2250*	0,000000*
Seri	o-değeri1	0,484646	0,029598	0,353733	0,000000*	0,0100*
	Oran1	0,9890	0,9580*	0,9720*	0,2250*	0,000000*
	o-değeri2	0,000000*	0,000000*	0,000000*	0,000000*	0,000000*
	Oran2	0,9930	0,9620*	0,9720*	0,2250*	0,000000*

[42] Excel için 250, MyRand ve MidSquare için 100

Çizelge 6.6b : NIST STS 1.3 Deney Sonuçları

		Crand(1)	Crand(2)	Crand(3)	Excel(1)	Excel(2)
Frekans	o-değeri	0,008207	0,000000 *	0,001544	0,083526	0,150340
	Oran	0,9640*	0,9690*	0,9630*	0,9920	0,9880
Blok Frek.	o-değeri	0,428095	0,139655	0,146982	0,179584	0,006107
	Oran	0,9690*	0,9740*	0,9650*	0,9800	0,9880
Cusum	o-değeri1	0,000002 *	0,021262	0,043087	0,016950	0,073872
	Oran1	0,9610*	0,9690*	0,9640*	0,9920	0,9880
	o-değeri2	0,000010 *	0,005557	0,000004*	0,083526	0,191687
	Oran2	0,9660*	0,9700*	0,9660*	0,9920	0,9880
Spectral	o-değeri	0,000000 *	0,000000 *	0,000000*	0,000000 *	0,000000 *
	Oran	0,9750*	0,9760*	0,9750*	0,9960	1,0000
Yaklaşık Entropi	o-değeri	0,011464	0,003909	0,000769	0,486588	0,171867
	Oran	0,9640*	0,9690*	0,9560*	0,9920	0,9920
Seri	o-değeri1	0,017667	0,552383	0,000347	0,948298	0,463512
	Oran1	0,9640*	0,9720*	0,9660*	0,9920	0,9880
	o-değeri2	0,000000 *	0,000000	0,000000*	0,000000 *	0,000000 *
	Oran2	0,9690*	0,9690*	0,9640*	0,9880	0,9880

6.4 Sonuçların İrdelenmesi

Elde edilen sonuçlar irdelendiğinde aşağıdaki sonuçlar ortaya çıkmaktadır:

1. Başlangıç (tohum) değerine göre üretecin test performansı değişebilmektedir. Bazı üreteçler için (örneğin Çizelge 6.5 Delphi kolonları) bu değişim gözardı edilemeyecek boyutlara ulaşmaktadır. Bu tür değişimler, uygulamanın sonucunu olumsuz etkileyebilir.
2. FIPS 140-2 ve ENT test bataryalarının her ikisinin de kesin kusurlu üreteçleri (*Mid Square, MyRand*) rahatlıkla tespit ettiği, bununla beraber ENT deneylerinin FIPS 140-2 standardının algılayamadığı daha ufak sapma ve kusurları da belirlediği (Çizelge 6.5'de '?' ile işaretlenmiş sonuçlar) görülmüştür. NIST STS ise diğer iki deney paketinin belirleyemediği kusurları da (KISS32 üretecinde olduğu gibi) tespit etmiştir.
3. FIPS140-2 ve ENT paketlerinde aynı deneyi birden fazla dizi ile tekrar etmek için programı tekrar tekrar çalıştırmak ve sonuçları bir araya getirmek gerekmektedir. Oysa NIST STS1.3'de büyük bir veri dosyası oluşturup bunun üzerinde bir defada n uzunluğunda m farklı dizi test edilebilmekte ve NIST STS1.3 paketi bu m deneyin sonucunu toplu olarak analiz edilmiş şekilde sunmaktadır. Ayrıca aynı veri dosyası için farklı n ve m değerleri için deneyler gerçekleştirmek mümkündür.
4. NIST STS1.3 ve FIPS 140-2 standardında deney sonuçlarının yorumları kesin kriterlere bağlanmış, ENT paketinde ise sonuçların *Ideal*'den sapma (yani hata) yüzdeleri verilip sonuç kullanıcının yorumuna bırakılmıştır.

7 UYGULAMALAR VE SONUÇLAR: ASAL SAYILAR

Asal sayı üreteçlerinin kodlanması ve denenmeleri Unix işletim sistemi altında GAP yazılım ortamında gerçekleştirilmiştir. UNIX ortamında sistemden bilgi toplanıp rastsal dosyalar oluşturulmakta ve bu rastsal verilerden asal sayılar üretilmektedir. Tez çalışmasında GAP yazılımında var olan hazır fonksiyonların kullanımına ilaveten GAP dilinde yazılan ufak programlarla da asal sayı üretme ve test işlemleri yapılmıştır. Ayrıca 17 basamağa kadar olan Carmichael sayıları da deneylere tabii tutulmuş ve herhangi bir asallık testinin ilk 246.683 adet Carmichael sayısını elemesi için gereken taban adedi belirlenmiştir.

Web üzerinde asallık ile ilgili "en büyük asal sayıyı bulma" gibi yarışmalar bulunmakta, genellikle de bu tür büyük bir sayı bulma aslında biraz şansa dayanmaktadır. Ne yazık ki bunun teorisi ve nedenleri üzerinde bu kadar yoğun durulmamaktadır. Mesela 10^{12}'ye kadar n'inci asal sayı http://www.math.Princeton.EDU/~arbooker/nthprime.html Internet adresinden bulunabilir. Bu siteden 1.000.000.000.000 (1 katrilyon)'uncu asal sayının 29.996.224.275.833 olduğu öğrenilebilir. Bu siteden 900.000.000.000 'cu asal sayı istendiğinde aşağıdaki cevapla karşılaşılacaktır:

```
The 900,000,000,000th prime is 26,898,370,231,697
```

Bu sayının asallığını test etmek için ECCP (*Eliptic Curve Primality Proving*) paketi http://www.fsp.com/cgi-bin/xrunecppc Internet adresinden CGI script aracılığıyla kullanılabilir. Bu işlemin sonucu ve paketin kullandığı yöntemler EK-2'de verilmiştir. ECCP algoritmasının en hızlı ve en gerçeğe yakın asal sayı testi olduğu iddia edilmektedir. Bu algoritma hakkında daha detaylı bilgi ve progam kodu http://pauillac.inria.fr/algo/morain/ Internet adresinden temin edilebilir.

7.1 GAP Ortamında Program Geliştirme

GAP kısaltması, Gruplar Algoritmalar ve Programlama'yı (Groups, Algorithms and Programming) simgeler. GAP yazılımının özelliklerini aşağıdaki gibi sıralamak mümkündür:

- GAP, Discrete Abstract Algebra'da hesaplama yapmak için geliştirilmiş bir yazılım paketidir.
- Yüksek basamaklı aritmetik işlemleri ve birçok matematiksel fonksiyonu kullanıcıya sağlamaktadır.
- Bedava olarak Internet'ten temin edilebilir.
- Birçok işletim sisteminde kullanılabilmektedir.
- Açık ve eklentilerle genişletilebilecek (*extensible*) bir sistemdir. Kullanıcı GAP diliyle yazdığı kendi programlarını, sistemdeki herhangi bir fonksiyonu kullandığı gibi kullanabilmektedir. İstenirse yazılan programlar, paylaşım paketi (*share package*) şeklinde diğer kullanıcılarla paylaşılabilmektedir.

GAP hakkında daha detaylı bilgi EK-8'de verilmiştir. Tez çalışmasında asal sayı testleri, GAP paketiyle UNIX ve PC ortamlarında uygulanmıştır.

Mevcut kütüphane dosyası : "integer.gi" ve içerdiği fonksiyonlar

Gap'in "\lib" dizininde bulunan "integer.gi" dosyasında doğal sayılarla ilgili çeşitli fonksiyonlar bulunmaktadır. Bu fonksiyonlar Çizelge 7.1'de açıklamalarıyla birlikte tanıtılmıştır (GAPRef, 1999).

Çizelge 7.1[43] : GAP "integer.gi" Kütüphanesi Asal Sayı Fonksiyonları

Fonksiyon Adı	Fonksiyon Tanımı
IsPrimeInt(*n*)	*n*'in asallığını test eder. 10^13'e kadar olan sayılarda kesin sonuçlar vermektedir. bu sayıdan büyük değerler için uyarı mesajı verir.
IsProbablyPrimeInt(*n*)	*n*'in asallığını test eder. Büyük değerler için uyarı mesajı vermez.
IsPrimePowerInt(*n*)	*n* bir "prime power" ise "true", değilse "false" döndürür.
NextPrimeInt(*n*)	*n*'den büyük olan ilk asal sayıyı döndürür.
PrevPrimeInt(*n*)	*n*'den küçük olan en büyük asal sayıyı döndürür.
FactorsInt(*n*)	*n*'in asal faktörlerinin listesini döndürür.
PrintFactorsInt(*n*)	Asal faktör listesini daha okunabilir bir şekilde listeler.
PrimePowersInt(*n*)	*n*'in asal faktörlere ayrılmasının listesini döndürür.
DivisorsInt(*n*)	*n*'in bölenlerininin listesini döndürür.

Ayrıca bazı asal sayıların küçükten büyüğe sıralanmış (*sorted*) olarak listeleri tutulmaktadır. Bunlar aşağıda belirtilmiştir:

- 1000'den küçük olan ilk 168 asal sayı, *Primes* komutu aracılığıyla çağrılabilmektedir. *n* sayısı için *Primes[n]* listesiyle *n*'inci asal sayı bulunabilir. Aşağıdaki örnekte 10'uncu asal sayının 29 olduğu görülmektedir.

```
gap> Primes[10];
29
```

- Ayrıca Richard Brent'in tablolarından alınan 607 adet değişen büyüklüklerde (15-58 basamak arasında) asal sayıların listesi de sıralanmış olarak tutulmaktadır. *n* sayısı için *Primes2[n]*

[43] Buradaki fonksiyonlar, "IsPrimeInt" olası (*probable*) asallık testine dayanmaktadır.

listesiyle *n*'inci asal sayı bulunabilir. Aşağıdaki örnekte listedeki 200'üncü asal sayının 15888756269 olduğu görülmektedir.

```
gap> Primes2[200];
15888756269
```

Verilen bir *n* sayısına kadar olan asalları gösteren program

Verilen n sayısına kadar olan asalları gösteren GAP kodu aşağıdaki gibidir:

```
# UBE - Bitirme Tezi Calismalari - 04.2000 - Enis Karaaslan
#  Verilen n sayısına kadar olan asalları gösteren program
goster_asal := function(sayi)
        local p, n, primes, numbers;
        primes := [ ];
        numbers := [2..sayi];
        for p in numbers do
            Add(primes,p);
            for n in numbers do
                if n mod p =0 then Unbind (numbers [n-1]);
                fi;
            od;
        od;
        Print ("2 den ", n, " e kadar olan asal sayilar sunlardir", "\n");
        return primes;
end;
```

Program çalıştırıldıktan sonra "goster_asal(1000);" veya benzeri komutlarla bulunan asallar görüntülenebilmektedir. Ekran çıktısı aşağıdaki gibi olmaktadır.

```
gap> goster_asal(1000);
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173,
179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,
241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311,
313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379,
383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521,
523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601,
607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673,
677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757,
761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839,
853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919,
929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997 ]
```

Program çalıştıktan sonra elde edilen asal sayı bilgisi *primes* değişkeninde tutulmaktadır. Bu değişken çeşitli parametrelerle çağrılıp istenilen asallar elde edilebilir. Mesela aşağıda ilk 20 asalın listelenmesi sağlanmıştır.

```
gap> primes { [1..20] };
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
53, 59, 61, 67, 71 ]
```

30'dan küçük asallar ise *primes* değişkeninin aşağıdaki şekilde çağrılmasıyla elde edilir.

```
gap> Filtered (primes, x-> x < 30);
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

Dosyadan Hex Veri Okuyup Doğal Sayıya Çevirmek

Asallık testlerinin uygulanacağı sayıları elde etmek için rastsal sayı üreteçlerinden onaltı tabanlı (*hexadecimal*) bilgilerin okunması ve bunların doğal sayıya dönüştürülmesi gerekebilmektedir. Bunlar için *okuhex()* ve *hextoint()* fonksiyonları kullanılmaktadır. Fonksiyonların açıklamaları aşağıda belirtildiği gibidir:

- ***okuhex (DosyaAd, BasamakAdedi)*** **:** Bu kod, hexadecimal sayılardan oluşan dosyayı açmakta, bu dosyadan satır satır okuma yapmakta ve ve bunları doğal sayıya dönüştürmektedir. Sayıya *numbers[1] ... numbers[n]* değişkenleriyle ulaşılabilir. Bu programın kaynak kodu EK-9'da bulunmaktadır. Parametreler ve açıklamaları aşağıdaki gibidir :
  - *DosyaAd :* hexadecimal sayıların bulunduğu dosya
  - *BasamakAdedi :* kaç basamak hexadecimal basamağın okunacağı
- ***hextoint (HexDeger, BasamakAdedi)*** **:** Verilen hexedecimal değeri ondalık tamsayıya çevirir. Parametreler ve açıklamaları aşağıdaki gibidir:
  - *HexDeger :* hexadecimal verilerin bulunduğu değişken
  - *BasamakAdedi :* kaç basamak doğal sayıya dönüştürüleceği

7.2 Asal Sayı Testleri

Çeşitli rastsal sayı üreteçlerinden onaltı tabanında verilerin okunması ve bunların doğal sayıya dönüştürülmesi işlemlerinden sonra elde edilen sayılar çeşitli asallık testlerine tabii tutulmaktadır. Uygulanan testler ve parametreleri aşağıda belirtilmiştir:

- *Asal_mı Testi :* asal_mi(Sayi, Yontem, TabanAdedi)
- *IsPrime Testi :* IsProbablyPrimeInt(Sayi)
- *Lehmann Testi :* Lehmann(Sayi, Yontem, TabanAdedi)
- *Miller&Rabin Testi :* MilRab(Sayi, Yontem, TabanAdedi)
- *Lucas Testi :* Lucas(Sayi)

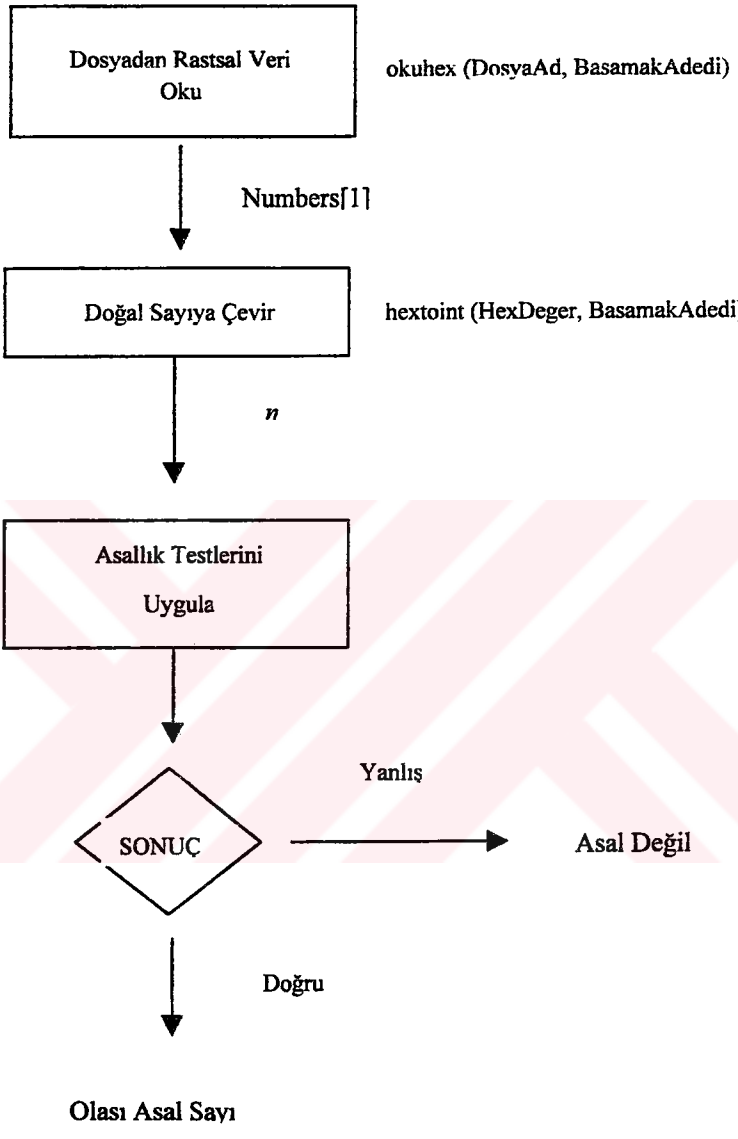
Belirtilen adette asal sayı bulmak için ise *asal_bul* fonksiyonu kullanılmaktadır:

Asal_bul(Sayi, Yontem, TabanAdedi)

Asal_mi, Lehmann, Lucas, Miller&Rabin testlerinin ve Asal_bul fonksiyonunun kaynak kodu EK-9'da bulunmaktadır. IsPrimeInt() testinin kaynak kodu EK-10'da verilmiştir. Testlerin parametreleri ve açıklamaları aşağıdaki gibidir:

- *Sayi* : Asallık Testinin uygulanacagı sayıdır.
- *TabanAdedi* : Asallık testinde taban olarak alınacak sayı adedidir.
- *Yontem* : Taban (*base*) olarak alınacak sayının elde edilme yöntemini belirlemektedir. "0" değeri verildiğinde *Taban Adedi* kadar rastsal sayı taban olarak alınacaktır. "0" dışındaki değerler icin *Taban Adedi* kadar ilk asal sayı alınacaktır.

Asal sayı test uygulaması Şekil 7.1'de gösterilmiştir.



Şekil 7.1 : Asal Sayı Deney Uygulaması Şeması

Miller&Rabin ve Lehmann testleri uygulanırken belirli bir hata payı hedeflenerek testde taban olarak kullanılacak sayıların adedi hesaplanmalıdır. Mesela Miller&Rabin testi sonucunda her bir taban

değeri için testin bileşik bir sayıyı asal olarak tanıma olasılığı ¼ = 2^{-2} olmaktadır. 2^{-100} hata payını hedef olarak alırsak, bu değere ulaşmak için 50 farklı tabanda işlem yapmamız yeterli olacaktır. Bu testler verilen parametreye göre, taban değerlerini rastsal üretmekte veya ilk asal sayıları almaktadır. Kaç adet bu sayılardan alınacağı parametre olarak verilebilmektedir. Bu değerler asal sayılardan seçilirse daha başarılı sonuçlara ulaşılabilecektir (Menezes ve ark, 1997).

Asal_mi fonksiyonunda alınan sayının diğer testlere tabi tutulmadan önce 8191'e kadar olan bütün asallarla bölünüp bölünmediğinin de test edildiği dikkate alınırsa asal olma olasılığının daha da arttığı görülecektir.

Deney sonuçlarında da görüleceği gibi, başlangıç sayısı rastsal olarak alındığında deneyin süresi değişkenlik göstermektedir. 250 basamaklı bir (olası) asal sayı bulmak en azından bir dakika zaman almaktadır.

7.3 Asal Sayı Üreteçleri

Pratikte Asal Sayı Üretme Esasları Bölüm 5.5'de belirtilmişti. GAP ortamında geliştirilen uygulamada bu esaslar temel olarak alınmıştır. GAP ortamındaki uygulamamızda kullandığımız temel unsurlar aşağıda belirtildiği gibidir:

- 8191'e kadar olan ilk 1028 asal sayı alınmıştır ve her test edilen sayının bu sayılara bölünüp bölünmediği kontrol edilmektedir.
- Her seferinde rastsal *p* değeri oluşturmak yerine, ilk seferden sonra o rastsal sayıyı 2 arttırarak asal bir sayı bulana kadar da işleme devam etme yöntemi kullanılmaktadır.
- Ayrıca her seferinde yeni sayıları listedeki asal sayılara bölmek yerine, bir önceki bölmeden kalan sayıları 2 arttırıp o asal sayıya göre modunu almaktadır. Böylece yüksek basamaklı sayıların, listedeki asallara bölünmesinden dolayı oluşacak zaman kaybı en aza indirilmektedir.
- Lehmann ve Miller&Rabin testleri verilen parametreye göre, taban değerlerini rastsal üretmekte veya ilk asal sayıları

almaktadır. Bu sayılardan kaç adet alınacağı parametre olarak verilmektedir.

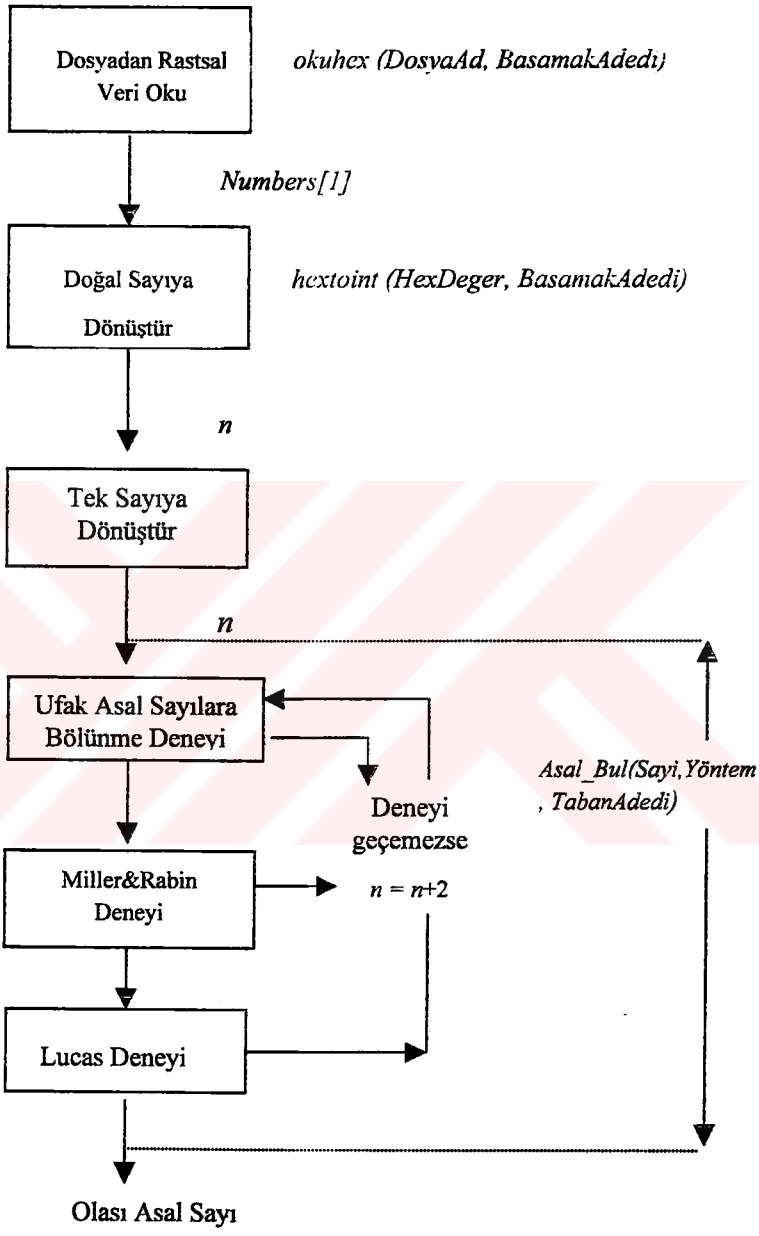
- Miller&Rabin testini geçen sayılar Lucas testine tabii tutulmaktadır.

Belirtilen adette asal sayı bulmak için *Asal_bul(Sayi, Yontem, TabanAdedi)* fonksiyonu kullanılmaktadır. Fonksiyonun parametreleri ve açıklamaları aşağıdaki gibidir:

- *Sayi* : Asallık deneyinin uygulanmaya başlanacağı başlangıç sayısını belirler.
- *Yontem* : Taban (*base*) olarak alınacak sayının elde edilme yöntemini belirler. "0" değeri verildiğinde *TabanAdedi* kadar rastsal sayı taban olarak alınacak. "0" dışındaki değerler icin *TabanAdedi* kadar ilk asal sayı alınacak.
- *TabanAdedi*: Asallık deneyinin kullanacağı taban olarak alınacak sayı adedidir.

Asal_bul[44] fonksiyonunun çalışma şeması Şekil 7.2'de gösterilmiştir.

[44] Asal_bul fonksiyonu dökümanda, Lucas testinin Miller&Rabin testinden daha önce veya sonra kullanılmasına göre Asal_bul veya Asal_bulMod adları ile kullanılmış ve geçtiği yerlerde aradaki fark açıklanmıştır.



Şekil 7.2: *Asal_bul* Fonksiyonun Çalışma Şeması

Uygulamalar

250 basamaklı bir sayı alıp, 50 adet taban kullanarak o sayıya en yakın asal sayıyı bulmak için *Asal_bul()* fonksiyonunu uyguladığımızda ekran çıktısı aşağıdaki gibi olmaktadır:

```
gap>sayi;

98436862349814798107047124604891204712894612471206498249846
23984632492046239846234896230498236402946234896234982346238
94632894623424602394620394623098462348916249860248923642380
94623049826342304623049263402346230462342498127309183589750
93501798908437

gap>asal_bul(sayi,0,50);

Bu islem : 80056 milisecond cpu time surmustur.

98436862349814798107047124604891204712894612471206498249846
23984632492046239846234896230498236402946234896234982346238
94632894623424602394620394623098462348916249860248923642380
94623049826342304623049263402346230462342498127309183589750
93501798908671

Olasi asal sayidir.
```

Bu işlemde asal sayıyı bulmak için sırayla 98..437, 98..439, ..., 98..671 sayıları denenmiştir. Bu da demektir ki; (671 – 437) / 2 + 1 = 118 sayı denenmiştir. Kayıtlar (loglar) incelendiğinde sırayla denenen sayıların hiçbirinin Miller&Rabin testinin 2 tabanı testini geçemediği görülmüştür. Tabii ki bu sayı aralığı için böyle bir sonuç rastlantısal olmaktadır. 2 tabanını geçip de diğer tabanlarda takılan sayılar da görmek mümkündür.

Bulunan asal sayı, GAP programındaki *IsPrime()* fonksiyonuna verildiğinde olası asal sonucu döndürülmektedir.

```
gap> IsPrimeInt(deger);

# I  beyond the guaranteed bound of the probabilistic
primality test

true
```

7.4 Farklı Bit Uzunluklarında 100'er Asal Sayı Bulma Deneyi

Bu deneyde rastgele seçilen bir başlangıç değerinden itibaren çeşitli bit büyüklüklerinde ilk 100 asal sayının bulunması ve ortalama zaman ölçümlerinin hesaplanması gerçekleştirilmiştir. Deneylerde kullanılan bit değerlerine karşılık gelen basamak adetleri Çizelge 7.2'de belirtilmiştir. Örneğin; 32 bit bir sayı 10 (ondalık) basamaklı bir değere karşılık gelmektedir. Alacağımız basamak değerinin bu sayıyı da kapsaması için $10^y > 2^{32}$ olmalıdır. Bu yüzden *Kullanılan Değer*, basamak değerinden bir fazla olarak 11 alınmıştır.

Çizelge 7.2: Bit Adedine Karşılık Gelen Basamak Adedi

Bit Adedi	Basamak Adedi	Kullanılan Değer
32	10	11
64	20	21
128	39	40
256	78	79
512	155	156
1024	309	310
2048	617	618

Başlangıç değerleri olarak iki örneklem (Örneklem-1, Örneklem-2) alınmıştır. Alınan değerler EK-11'de verilmiştir. Deneyler sonucunda bulunan değerler ve açıklamaları aşağıdaki gibidir:

- **Denenen Sayı Adedi:** 100 Asal bulmak için deneye tabii tutulan sayı adedi.
- **Ortalama Zaman:** 1 Asal bulmak için gereken süre (milisaniye işlemci zamanı – milisecond *cpu time*).
- **EnAz (min) Zaman:** Bir sayının testinin aldığı minimum zaman (milisaniye işlemci zamanı).
- **EnÇok (max) Zaman:** Bir sayının testinin aldığı maksimum zaman (milisaniye işlemci zamanı).

Yukarıda da belirtildiği gibi *EnAz Zaman* ve *EnÇok Zaman*, herhangi bir sayının asallık testinin aldığı minimum ve maksimum değerlerdir. *EnAz Zaman* ve *EnÇok Zaman, Asal_bul* ve *NextPrimeInt* fonksiyonlarında direkt asal buldukları için diğer testlerden farklı olarak bir asal bulmak için harcanan minimum ve maksimum zamanı göstermektedir. Bu yüzden bu testlerde *EnAz Zaman* 'ın 0 dan büyük olduğu gözlemlenebilmektedir. *Ortalama Zaman* ise bir asal sayının bulunması için geçen ortalama süredir. Mesela örneklem-1'de 618 basamaklı 100 asal bulmak için 88.743 sayı taranmıştır. *Ortalama Zaman*, burada yaklaşık 887 ardışık tek sayının asallık testine tabii tutulma süresine karşılık gelmektedir.

Deneyler 360 Mhz Ultra Sparc II işlemcili, 128 Mb RAM, 256 Kb ön belleğe sahip SUN Ultra 5 iş istasyonlarında, Solaris işletim sisteminde GAP programı kullanılarak uygulanmıştır. Zaman ölçümleri kullanılan işlemci zamanını vermektedir ve *milisecond cpu time* birimiyle gösterilmektedir. Örneklem-1 için gözlenen deney sonuçları Çizelge 7.3a, b, c, d, e, f ve g'de verilmiştir. Bu tablolarda *Ortalama Zaman*'ın *EnÇok Zaman*'dan büyük olması durumu ortaya çıkabilmektedir. Bunun nedeni ise *Ortalama Zaman*'ın bir asalı bulmak için gereken zaman, *EnÇok Zaman* 'ın ise bir sayıyı test etmek için gereken zaman olmasıdır.

Sonuçları karşılaştırmak için farklı bir örneklem (Örneklem-2) alınmış, bu aralıkta asal bulmak için kullanılan *NextPrimeInt* ve *Asal_bul* fonksiyonları uygulanarak gözlenen deney sonuçları Çizelge 7.3h ve j'de verilmiştir.

Miller&Rabin ve Lucas testleri (Çizelge 7.3.f, Çizelge 7.3.g) 2048 bit (618 basamak) için işlemleri makul bir zamanda bitiremediklerinden bu test 10 asal sayı bulmak için uygulanmış ve sadece ortalama zaman bulmak için kullanılmıştır.

Çizelge 7.3a: IsPrimeInt Asallık Deneyi Sonuçları (Örneklem-1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	88.743
Ortalama Zaman	10,20	28,70	121,80	598,80	6.118,30	64.472,30	1.077.799,20
EnAz Zaman	0	0	0	0	0	0	0
EnÇok Zaman	50	60	60	120	720	3.370	22.650

Çizelge 7.3.b: NextPrimeInt Asallık Deneyi Sonuçları (Örneklem-1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	88.743
Ortalama Zaman	10,4	29,2	116	593,3	5910,4	62.874	1.071.297,6
EnAz Zaman	0	10	20	80	460	3.040	22.410,0
EnÇok Zaman	100	160	420	2800	27690	410.860	5.049.410,0

Çizelge 7.3c: Asal_mı Asallık Deneyi Sonuçları[45] (Örneklem-1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	88.743
Ortalama Zaman	9,90	27,80	111,50	678,30	8.158,70	94.609,90	1.660.472,10
EnAz Zaman	0	0	0	0	0	0	0
EnÇok Zaman	60	60	60	340	840	3.400	29.540

[45] Asal_mi deneyinde ufak asallara bölme deneylerine ek olarak sırasıyla Lucas ve Miller&Rabin Deneyleri uygulanmıştır.

Çizelge 7.3.d: Asal_miMod[46] Asallık Deneyi Sonuçları (Örneklem 1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	88.743
Ortalama Zaman	10	24,4	84,7	419,1	4347	47.863,1	832.190,2
EnAz Zaman	0	0	0	0	0	0	0
EnÇok Zaman	50	50	60	370	740	3.580	22.430

Çizelge 7.3.e: Asal_bul Asallık Deneyi Sonuçları (Örneklem-1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	88.743
Ortalama Zaman	44,20	85,30	176,20	642,60	4.714,10	48.129,40	831.614,90
EnAz Zaman	0	10	30	90	440	3.010	22.340
EnÇok Zaman	190	530	810	290	22.980	300.970	3.928.550

Çizelge 7.3.f: Miller & Rabin Asallık Deneyi Sonuçları (Örneklem-1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	8.643
Ortalama Zaman	12,50	48,20	302,90	2.280,10	30.225,00	360.745,40	6.339.635
EnAz Zaman	0	0	0	10	120	900	7.260
EnÇok Zaman	60	60	50	80	430	2.570	14.430

[46] Asal_miMod deneyinde ufak asallara bölme deneylerine ek olarak sırasıyla Miller&Rabin ve Lucas Deneyleri uygulanmıştır.

Çizelge 7.3.g: Lucas Asallık Deneyi Sonuçları (Örneklem-1)

Basamak Adedi	11	21	40	79	156	310	618
Denenen Sayı Adedi	1.255	2.319	5.150	8.965	20.325	36.045	8.643
Ortalama Zaman	18,90	85,50	576,90	4665,5	62.085,30	743.833,2	12.900.634
EnAz Zaman	0	0	0	30	180	1.930,0	14530
EnÇok Zaman	40	40	300	340	810	5.740,0	26900

Çizelge 7.3.h: NextPrimeInt Asallık Deneyi Sonuçları (Örneklem-2)

Basamak Adedi	11	21	40	79	156	310	618
Ortalama Zaman	10,1	30,1	129.3	625,4	5.323	51.457	1.063.246
EnAz Zaman	0	10	20	90	450	4.050	97.570
EnÇok Zaman	80	160	580	2380	23.720	238.420	2.929.200

Çizelge 7.3j: Asal_bulMod[47] Asallık Deneyi Sonuçları (Örneklem-2)

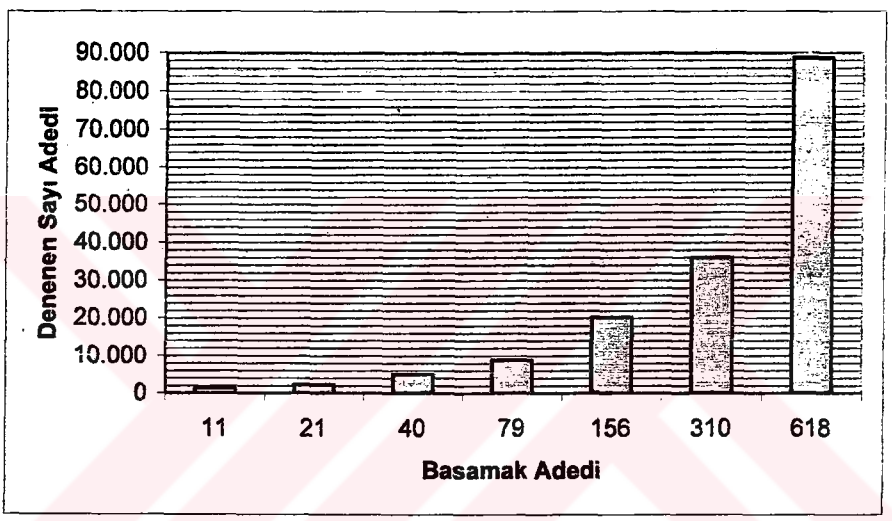
Basamak Adedi	11	21	40	79	156	310	618
Ortalama Zaman	37	82	216,7	653,6	4.242	39.908	679.825
EnAz Zaman	0	10	20	80	440	2.980	22.460
EnÇok Zaman	160	270	1100	2210	18.610	183.330	2.846.570

[47] Asal_bulMod deneyinde ufak asallara bölme deneylerine ek olarak sırasıyla Miller&Rabin ve Lucas Deneyleri uygulanmıştır.

Deneylerin Yorumlanması:

Bazı deneylerde *EnAz Zaman*'ın 0 çıkmasının nedeni; test edilen sayının Ufak Asallara Bölme testini geçememesi yani küçük asal çarpana (Örneğin 3,5,7 gibi) sahip olmasından dolayı olmaktadır.

Örneklem-1 ile yapılan deneylerin sonuçları incelendiğinde, basamak adedi (bit adedi) arttıkça denenmesi gereken sayıların değişimi Şekil 7.3'de verilmiştir.



Şekil 7.3: Basamak Sayısına Göre Denenen Sayı Adedinin Değişimi

Bulunma oranı; verilen basamak adedine kadar, sadece tek sayılar dikkate alınarak ortalama kaç sayıda (denemede) bir asal sayı bulunabildiğini göstermektedir. Örneklem-1 ve örneklem-2 kullanılarak, n sayısından küçük sayılar için bulunma oranları Formül 7.1 ile hesaplanmış (Stallings, 1998) ve örneklerde gözlenen değerlerle Çizelge 7.4'de karşılaştırılmıştır. Örneğin 11 basamaklı sayılar için bulunma oranı $(\ln 10^{11}) / 2 = 11 (\ln 10) / 2 = 12{,}66$ olmaktadır.

$$\text{Bulunma Oranı} = \frac{\ln n}{2} = \frac{\log_e n}{2} \qquad \text{(Formül 7.1)}$$

Çizelge 7.4: Asal Sayıların Bulunma Oranları

Basamak Adedi	11	21	40	79	156	310	618
Beklenen	12,66	24,18	46,05	90,95	179,60	356,90	711,50
Örneklem1	12,55	23,19	51,5	89,65	203,25	360,45	887,43
Örneklem 2	10,24	25,6	48,83	105,33	180,015	393,72	719,02
En Çok Sapma	-%18,41	%10,39	%11,83	%17,49	%13,17	%9,23	%24,73

Testlerin *ortalama çalışma süreleri*, bir asal bulmak için harcanması gereken ortalama zamanı vermektedir. Değişik testler için kaydedilen bu değerler Çizelge 7.5a'da verilmiştir.

Çizelge 7.5a: Asallık Testlerinin Ort. Çalışma Sürelerinin (msec) Karşılaştırılması

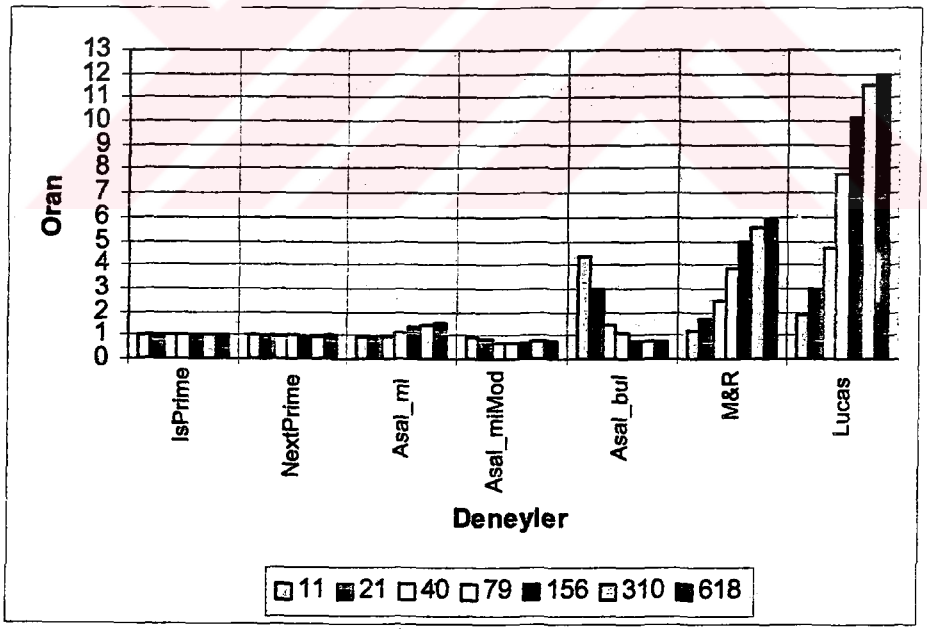
Basamak Adedi	11	21	40	79	156	310	618
IsPrimeInt	10,20	28,70	121,80	598,80	6.118	64.472	1.077.799
NextPrimeInt	10,40	29,20	116,00	593,30	5.910	62.874	1.071.298
Asal_mi	9,90	27,80	111,50	678,30	8.159	94.610	1.660.472
Asal_miMod	10,00	24,40	84,70	419,10	4.347	47.863	823.190
Asal_bul	44,20	85,30	176,20	642,60	4.714	48.129	831.615
Miller&Rabin	12,50	48,20	302,90	2.280,10	30.225	360.745	6.404.468
Lucas	18,90	85,50	576,90	4665,5	62.085	743.833	12.900.634

IsPrime testinin çalışma süresi 1 birim alındığında diğer testlerin hızları orantısal olarak Çizelge 7.5b'de verilmiş ve bu değerler grafiksel olarak Şekil 7.4'da gösterilmiştir. Üç ana asallık testinin (*Asal_bul, Asal_miMod ve IsPrimeInt*) orantılı değerleri grafiksel olarak Şekil 7.5'de gösterilmiştir. Bu grafiklerden de açıkça gözüktüğü gibi, hazırladığımız

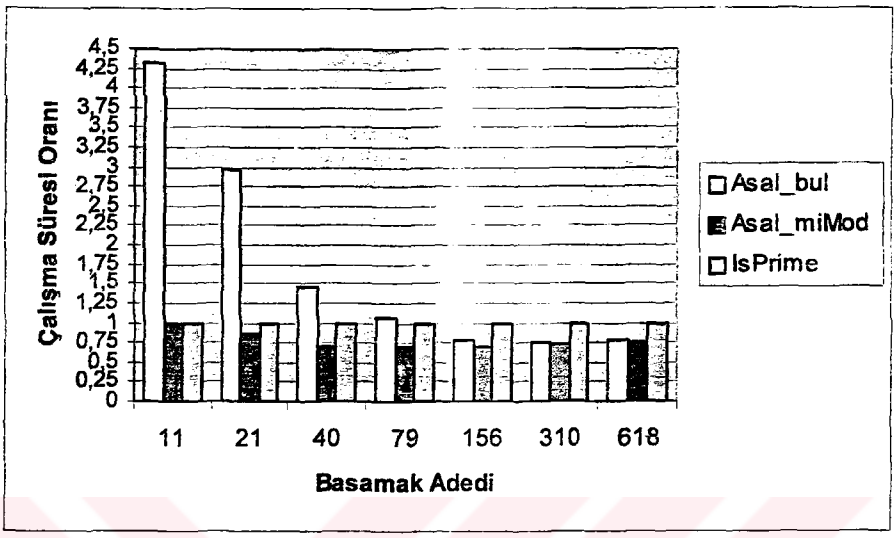
fonksiyonlar özellikle yüksek basamaklı asallar üzerinde daha hızlı çalışmakta ve *IsPrime* testinden daha iyi sonuçlar almaktadırlar.

Çizelge 7.5b : Asallık Testlerinin Çalışma Sürelerinin *IsPrime* 'a Göre Oranı

Basamak Adedi	IsPrime	NextPrime	Asal_mi	Asal_miMod	Asal_bul	MilRab	Lucas
11	1,00	1,02	0,97	0,98	4,33	1,23	1,85
21	1,00	1,02	0,97	0,85	2,97	1,68	2,98
40	1,00	0,95	0,92	0,70	1,45	2,49	4,74
79	1,00	0,99	1,13	0,70	1,07	3,81	7,79
156	1,00	0,97	1,33	0,71	0,77	4,94	10,15
310	1,00	0,98	1,47	0,74	0,75	5,60	11,54
618	1,00	0,99	1,54	0,76	0,77	5,94	11,97



Şekil 7.4: Asallık Testlerinin Çalışma Sürelerinin Oranlanması



Şekil 7.5: Üç Ana Asallık Testinin Çalışma Sürelerinin Oranlanması

7.5 Beş Milyonluk Bir Aralıkta Asal Sayıların Taranması

Deney için rastsal üretilen 121 basamaklı,

"2348968374995988875397324048393805242975249630696577940513434981394636450896272084167233718097757682084289441377126251479"

sayısından itibaren birbirini takip eden 1 milyonluk 5 ayrı sayı dizisi kullanılmıştır. Deneyler yoğun ve uzun işlemci kullanımı gerektirdiğinden üzerlerinde çok az iş yükü olan Sun Ultra 5 iş istasyonlarında uygulanmıştır.

Miller&Rabin testi, 50 taban için kullanarak her aralıkta bulunabilecek asal sayı adedi belirlenmiş ve bu değer referans noktası olarak alınmıştır. Diğer testler de 6 adet ve 1 adet tabanla çalıştırılarak bulunabilecek değerler tespit edilmiştir. Çizelge 7.6'de çeşitli testlerin belirtilen aralıklarda buldukları asal sayılar verilmiştir. Testler 1 adet taban kullanıldıklarında bile verilen aralıkta aynı sonucu döndürmektedirler. Çizelge 7.7'de kullanılan testlerin ortalama çalışma süreleri verilmiş ve bu veriler grafiksel olarak Şekil 7.6'da gösterilmiştir. Sonuçlardan da görüldüğü gibi, her ne kadar *IsPrime* fonksiyonu daha iyi sonuçlar verse de hazırladığımız fonksiyonlar da iyi sonuçlar vermiştir.

Hazırladığımız fonksiyonlarda *Miller&Rabin* testi *Lucas* testinden önceye alınarak daha iyi sonuçlar alınabileceği de unutulmamalıdır.

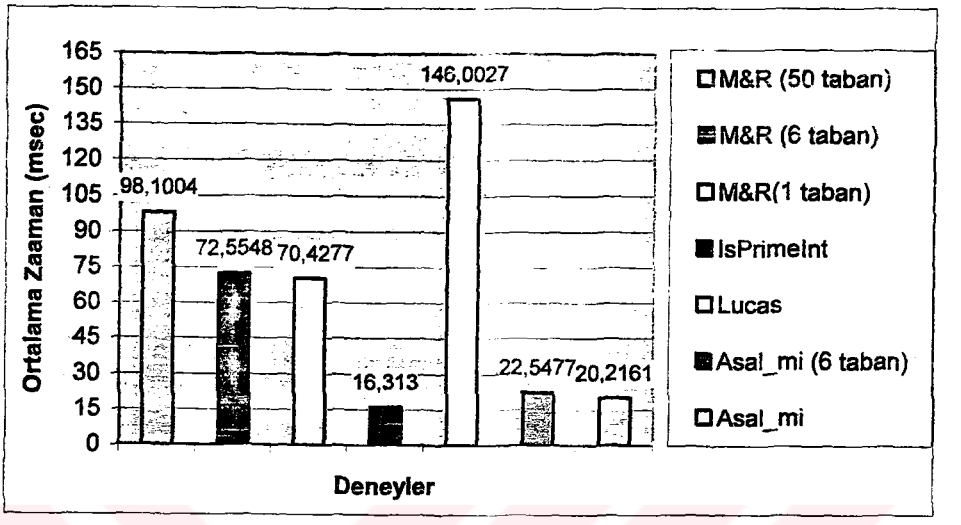
Sonuç olarak, asallık testlerini yanıltan sayıların adedi çok azdır ve birçok test bir taban için bile aynı sonuçları döndürebilmektedir.

Çizelge 7.6: Beş Milyonluk Aralıkta Bulunan Asallar

	Aralık1	**Aralık2**	**Aralık3**	**Aralık4**	**Aralık5**
Miller&Rabin (50 taban)	3683	3662	3610	3588	3544
Miller&Rabin (6 taban)	3683	3662	3610	3588	3544
Miller&Rabin (1 taban)	3683	3662	3610	3588	3544
IsPrimeInt	3683	3662	3610	3588	3544
Lucas	3683	3662	3610	3588	3544
Asal_mi (6 taban)	3683	3662	3610	3588	3544
Asal_mi (1 taban)	3683	3662	3610	3588	3544

Çizelge 7.7: Uygulanan Asallık Testlerinin Ortalama Çalışma Süreleri (msec)

Kullanılan Fonksiyon	**Ortalama Zaman**
Miller & Rabin (50 adet taban için)	98,1004
Miller & Rabin (6 adet taban için)	72,5548
Miller & Rabin (1 adet taban için)	70,4277
IsPrimeInt	16,3130
Lucas	146,0027
Asal_mi (6 adet taban için)	22,5477
Asal_mi *(1 adet taban için)* (ufak sayılara bölme + Lucas + Miller& Rabin)	20,2161



Şekil 7.6: Asal Sayı Testlerinin Ortalama Çalışma Süreleri

7.6 Bir Milyonluk İki Ayrı Aralıkta Asallık Testleri

Örneklem1 ve Örneklem2'den alınan rastsal sayılardan başlayan 1 milyon sayılık iki ayrı aralıktaki sayılar sırayla 4 ayrı teste tutulmuştur:

- Test1 (T1): Ufak Asallara Bölme
- Test2 (T2): Sözde Asallık Testi (2 tabanında Fermat testi)
- Test3 (T3): Miller & Rabin Asallık Testi
- Test4 (T4): Lucas Asallık Testi

Testler 21, 40, 79 ve 156 basamak sayılar için tekrarlanmış ve her testi geçen sayı adetleri not edilmiştir. Testlere asal olmayan sayıların yakalanması 1 ile, bu testlere takılmayan sayılar 0 ile belirtilmiştir. Deney sonuçları Çizelge 7.8'de verilmiştir. Çizelgeden de görüldüğü gibi 21 basamak için bütün testler sayıların yaklaşık %87,5 'inin asal olmadığını yakalamakta, %8' ini ise ufak asallara bölme dışındaki bütün testler yakalayabilmektedir.

Elde edilen bulgular, (Rivest, 1990)'de de belirtildiği gibi ufak böleni olmayan sayıların sözde (*pseudo*) asal olma ihtimalinin çok az olduğunu desteklemektedir. Daha detaylı bilgi için bakınız bölüm 5.2.9.

Çizelge 7.8: Bir Milyonluk Aralıkta Testlerin Karşılaştırılması

				21 basamak		40 Basamak		79 basamak		156 basamak	
T1	T2	T3	T4	Örn-1	Örn-2	Örn-1	Örn-2	Örn-1	Örn-2	Örn-1	Örn-2
1	1	1	1	437.851	437.590	437.740	437.813	437.726	437.886	437.821	437.615
1	1	1	0								
1	1	0	1								
1	1	0	0								
1	0	1	1								
1	0	1	0								
1	0	0	1								
1	0	0	0								
0	1	1	1	40.394	41.605	51.340	51.337	56.730	56.682	59.368	59.611
0	1	1	0								
0	1	0	1								
0	1	0	0								
0	0	1	1								
0	0	1	0								
0	0	0	1								
0	0	0	0	21.755	20.805	10.920	10.850	5.544	5.432	2.811	2.774

7.7 Carmichael Sayılarının Test Edilmesi

Bilindiği gibi Carmichael sayıları asal olmadıkları halde asallık testlerini yanıltan sayılardır. Yapılan çalışmada 10^{16} 'ya (17 basamağa) kadar olan 246,683 adet Carmichael sayısının test edilmesi yerine getirildi. Çeşitli testlerdeki çalışma hızları ölçüldü ve oranlandı. Ayrıca herhangi bir asallık testinin ilk 246,683 adet Carmichael sayısını elemesi (asal olmadığını testpit etmesi) için gereken taban adedi belirlendi.

Testler, Sun Ultra 5 iş istasyonu üzerinde çalışan Solaris Unix işletim sistemi altında gerçekleştirildi. Zaman ölçümleri, GAP ortamında var olan *Runtime()* fonksiyonu ile alındı. Bu fonksiyon kullanılan işlemci zamanını vermektedir (*CPU time*) ve ölçümler *milisecond cpu time* (msec) birimiyle gösterilmektedir. *Runtime()* fonksiyonu çalıştırmadan çalıştırmaya en çok %1 değişiklik göstermektedir. Bu değişme oranı dikkate alınmazsa gerçeğe yakın sonuçlar elde edildiği varsayılabilir. Uygulanan testler :

- *Asal_mi* Testi: Ufak (ilk 1028) asal sayılara bölme + *Lucas* + *Miller&Rabin* Testleri (1 adet taban için)
- *Miller&Rabin* Testi (Verilen taban adedinde)
- *Lehmann* Testi (Verilen taban adedinde)
- *Lucas* Testi
- *IsPrime* Testi

Uygulanan testlerin parametreleri ve açıklamaları aşağıdaki gibidir:

- *DosyaAd* : Carmichael sayılarının alınacağı rastsal verileri bulunduran dosya.
- *Yontem* : Taban olarak alınacak sayının elde edilme yöntemini belirler. "0" değeri verildiğinde *BaseAdedi* kadar rastsal sayı taban olarak alınır. "0" dışındaki değerler icin *BaseAdedi* kadar ilk asal sayı alınacaktır.
- *TabanAdedi* : Asallık testinin kullanacağı taban olarak alınacak sayı adedi

Ölçüm için kulllanılan program, bütün testler için uygulanmaktadır. Sadece testin çağrıldığı satır her fonksiyon için değişiktir. Test çağrılmadan önce ve sona erdikten sonra zaman ölçümleri alınıp değerlendirilmekte ve ortalama değerler hesaplanmaktadır. Programın kodu EK-12'de verilmiştir. 17 basamağa kadar olan Carmichael sayıları "Carmichael-16.txt" dosyasında tutulmaktadır. Üç rastsal taban için deneyin uygulanması aşağıdaki gibidir :

```
gap> test("Carmichael-16.txt",0,3);
```

Değişik sayıda taban alındığında bulunan sözde asalların (asallık testini yanıltan sayıların) sayısı Çizelge 7.9'de verilmiştir. Bulunan asal adedi, rastsal tabanlar alındığı için değişik çalıştırmalarda değişik (ama yakın) sonuçlar vermektedir. Değişmeyen şey ise en az 6 adet tabanla 0 değerine yani hatasız sonuçlara ulaşılmasıdır.

Çizelge 7.9 : Carmichael Deneyinde Tabana Göre Bulunan Asallar

	1 Taban	2 Taban	3 Taban	4 Taban	5 Taban	6 Taban
Asal_mi	0	-	-	-	-	-
Lucas	0	-	-	-	-	-
Miller&Rabin	1818	143	27	4	2	0
Lehmann	272	88	29	10	1	0
IsPrime	0	-	-	-	-	-

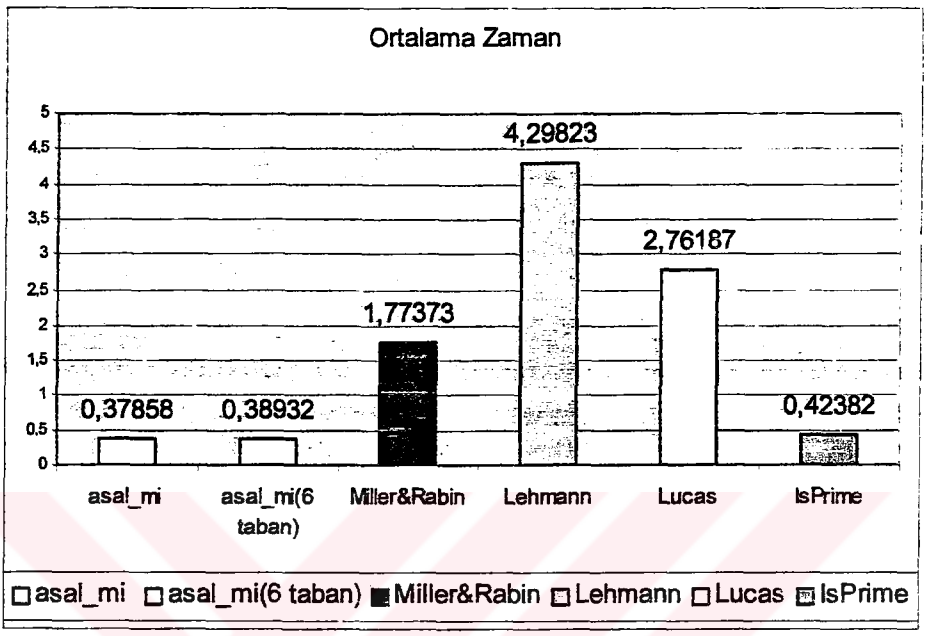
Carmichael sayılarının deneyinde alınan tabana göre çalışma zamanları Çizelge 7.10'da verilmiştir. Hatasız sonuçlar için gereken ortalama çalışma süreleri ise Çizelge 7.11'de verilmiştir. Çalışma zamanı ölçüm değerleri okuma kolaylığı açısından 5. derece hassasiyete kadar alınmıştır. Şekil 7.7'de ise Asallık Testlerinin ortalama çalışma sürelerinin grafiği verilmiştir.

Çizelge 7.10: Carmichael Deneyinde Tabana Göre Çalışma Zamanları (msec)

	1 Taban	2 Taban	3 Taban	4 Taban	5 Taban	6 Taban
Asal_mi	0,3785815	-	-	-	-	-
Miller&Rabin	1,51364	1,46653	1,47127	1,74920	1,75439	1,77372
Lehmann	1,48867	2,31479	2,93830	2,93830	3,91316	4,29823
Lucas	2,76187					
IsPrime	0,42382	-	-	-	-	-

Çizelge 7.11: Carmichael Deneyinde Ortalama Çalışma Süreleri (msec)

Kullanılan Fonksiyon	Ortalama Zaman
Asal_mi (1 adet rastsal taban için) (ufak sayılara bölme + Lucas + Miller&Rabin)	0,37858
Asal_mi (6 adet rastsal taban için)	0,38932
Miller&Rabin (6 adet rastsal taban için)	1,77373
Lehmann (6 adet rastsal taban için)	4,29823
Lucas	2,76187
IsPrime	0,42382



Şekil 7.7: Carmichael Deneyinde Ortalama Çalışma Süreleri

Sonuçlar incelendiğinde, *Asal_mi* ve *IsPrime* testlerinin diğer testlere göre daha hızlı olduğu gözükmektedir. Bu da ufak asallara bölmenin, asal sayıları bulma işlemini hızlandırdığını ve diğer testlere gerek kalmadan sonucun belli olduğunu göstermektedir. Hatırlanacağı gibi asal_mi fonksiyonunda ilk önce ufak asallara bölünme testi yapılmakta, eğer bu test geçilirse sırasıyla *Lucas* ve daha sonra da *Miller&Rabin* testine geçilmektedir.

7.8 İki Asal Sayının Çarpımından Oluşan Sayıların Faktörlere Ayrılması

Güvenlik protokollerinin çoğu, büyük (512, 1024 bit) asal sayıların çarpımından oluşan modülus n değerlerini anahtar (*key*) olarak kullanır. Bu tür protokollerin güçlü olması, modulüsün çarpanlara ayrılmasının zorluğuna dayanır. Bu bölümde iki asal sayının (p, q) çarpımından oluşan n sayısının çarpanlara ayrılmasının, biri asal diğeri asal olmayan iki sayının çarpımından oluşan y sayısının çarpanlara ayrılmasından daha zor olduğu deneylerle gösterilmeye çalışılmıştır.

Bunun için GAP ortamında kodlanan BBS üreteci kullanılmıştır. BBS üreteci; 95 basamaklık p, 97 basamaklık q asallarıyla ve tohum (seed) değeri olarak 95 basamaklık s değeri ile başlatılmıştır (initialized). Bu büyüklükler (Noll ve Ark., 2001)'de belirtilen minimum değerlerdir. Bu değerlerle üreteç başlatıldıktan sonra uygulamada x-bit uzunluğunda 1000 adet a ve 1000 adet b sayısı BBS üreteci tarafından üretilmiş ve bu değerlere en yakın asal sayılar *Asal_bul* fonksiyonu ile bulunmuştur.

Çarpanlara ayırma GAP ortamında FactInt 1.1[48] paketi ile yapılmaktadır. Daha iyi çarpanlara ayırma yöntemlerinin bulunabileceği aşikardır ama burada göstermek istediğimiz şey, asal çarpanlara sahip sayıların daha zor faktörlere ayrıldığıdır.

İki asalın çapımından oluşan n sayısının çarpanlara ayrılma süreleri Çizelge 7.12'de, biri asal ve biri asal olmayan iki sayının çarpımından oluşan y sayısının çarpanlara ayrılma süreleri Çizelge 7.13'de verilmiştir. Elimizdeki kodla 128 bit ve daha büyük sayıların çarpımından oluşan sayıların faktörlere ayrılması makul bir zaman aralığında gerçekleşemediği için en son 96 bit iki sayının çarpımından oluşan sayının çarpanlara ayrılması gerçekleştirilmiştir. 96 bit iki asal sayının çarpımından oluşan bir sayının çarpanlara ayrılması ortalama 112 dakika aldığından, deney bu bit büyüklüğü için 10 örnek üzerinde uygulanmıştır.

---

[48] FactInt 1.1 (Routines for Integer Factorization), GAP ortamı için Stefan Kohl tarafından hazırlanan çarpanlara ayırma yazılım paketidir. http://www-gap.dcs.st-andrews.ac.uk/~gap/Share/factint.html internet adresinden detaylı bilgi ve paket temin edilebilir.

Tutarlılık açısından biri asal biri asal olmayan 96 bitlik sayıların çarpımının faktörlere ayrılmasında da 10 örnek kullanılmıştır.

Çizelge 7.12: İki Asalın Çapımından Oluşan *n* Sayısının Çarpanlara Ayrılma Süreleri (msec)

Çarpanların bit uzunluğu	**16 bit**	**32 bit**	**64-bit**	**96 bit**
En Az	0	80	5.350	2.555.480
En Çok	40	9.750	123.380	11.475.550
Ortalama	10,12	1.614,93	67.619,31	6.745.135

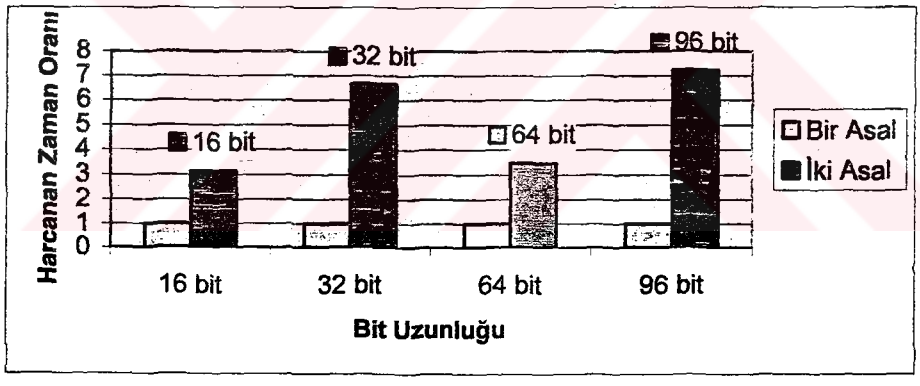
Çizelge 7.13: Biri Asal ve Biri Asal Olmayan iki sayının çarpımından Oluşan *n* Sayısının Çarpanlara Ayrılma Süreleri (msec)

Çarpanların bit uzunluğu	**16 bit**	**32 bit**	**64 bit**	**96 bit**
En Az	0	10	60	4.920
En Çok	30	8870	85.770	4.156.480
Ortalama	3,23	242,33	19.479,54	934.243

Eğer biri asal diğeri asal olmayan iki sayının çarpımından oluşan *y* sayılarının ortalama çarpanlara ayrılma süresi 1 birim olarak alınırsa, iki asal sayının çarpımından oluşan sayıların ortalama çarpanlara ayrılma süresi Çizelge 7.14'de belirtilmiş ve bu veriler grafiksel olarak Şekil 7.8'de gösterilmiştir. Buradan da görüldüğü gibi asal sayıların çarpımından oluşan sayıların çarpanlara ayrılması daha zordur.

Çizelge 7.14: Çarpanlara Ayırma Sürelerinin Karşılaştırılması

Çarpanların bit uzunluğu	16 bit	32 bit	64 bit	96 bit
Bir Asal	1	1	1	1
İki Asal	3,133127	6,664176949	3,471299117	7,219893539



Şekil 7.8 : Çarpanlara Ayırma Sürelerinin Karşılaştırılması

8 SONUÇ ve YORUMLAR

Büyük ölçekli rastsal ve asal sayılar; bilgisayar güvenliği, kriptografik protokoller, simulasyon, örnekleme (*sampling*), sayısal analiz ve karar verme gibi birçok alanda kullanılmaktadır. Bu sayıların verimli ve güvenilir bir yöntemle üretilmesi bu alanlarda faydalı olacaktır. Tez çalışmasında yüksek basamaklı rastsal asal sayıların oluşturulma süreci bilgisayar ortamında gerçekleştirilmiştir.

Rastsal ve asal sayılar üzerlerinde birçok araştırma yapılmasına rağmen hala gizemlerini ve sırlarını korumaktadırlar. Özellikle rastsallık konusu ile bilgisayar ortamlarının deterministik tabiatının uyuşmadığı iddia edilmektedir. Sanıldığının aksine bilgisayar ortamı bu konuda bize çeşitli esneklikler ve kolaylıklar sağlamaktadır. Deterministik bir ortam olan bilgisayarda bunu yapabilmenin etkin yöntemleri araştırılmış ve tartışılmıştır.

Rastsal Sayılar konusunda elde edilen veriler doğrultusunda çıkarılan sonuçlar şunlardır:

- Rastsal Sayı Üreteçleri, implementasyonlarında sorun olabileceği ihtimali dikkate alınarak *mutlaka* rastsallık testlerine tabii tutulmalıdırlar.
- Yaptığımız deneylerde Micosoft Excel, Delphi, Borland C gibi yaygın yazılım paketlerinde bulunan rastsal sayı üreteclerinin zayıflıkları olduğunu belirledik.
- Sistemden bilgi toplayıp tohum üretmenin yararlı bir yöntem olduğu anlaşılmış ve uygulamalarla bunun yöntemleri belirtilmiştir. Bu tohum aynı zamanda rastsal sayı olarak da kullanılabilir.
- Aşağıdaki üreteçler simulasyon vb. amaçlar için uygundur:
  - SHA1G
  - BBSG (Blum Blum & Shub Generator)
  - Tausworthe
- İncelediğimiz istatistiksel test bataryaları arasında NIST STS paketini, arkasında güçlü desteği olması, içerdiği testler ve

üreteçler bakımından kapsamlı olması, kullanımının ve sonuçları yorumlamanın kolay olması ve çok ayrıntılı dökümante edilmiş olması dolayısı ile (hem üreteç hem de test bataryası) tavsiye ediyoruz.

- Rastsal Sayı Üreteci ile üretilmiş bir rastsal havuzun belirli testleri geçmesi onun iyi bir üreteç olduğunu kanıtlamadığı gibi, bazı rastsallık testlerinde başarısız olan bir Rastsal Sayı Üreteci de tümden başarısız sayılmamalıdır. Aslında her uygulama için, o uygulamaya has testler bulunması gerekmektedir.
- Hiçbir üretecin ne kadar amprik test geçerse geçsin %100 kusursuz olamayacağını (ama simulasyon için yeterli olabileceğini) hatırlamakta yarar var.

Asal Sayılar konusunda elde edilen veriler doğrultusunda çıkarılan sonuçlar şunlardır:

- Deney sonuçlarından da görüldüğü gibi Miller&Rabin, Lucas ve IsPrime testleri bir taban için bile büyük ihtimalle doğru sonuçlar döndürmektedirler. Asal_mi ve Asal_bul fonksiyonları ise ufak asallara bölme, Miller&Rabin ve Lucas testlerini birlikte kullandıklarından daha az hata olasılığıyla çalışmaktadırlar.
- 156 basamak (512 bit) sayılardan itibaren Asal_bul fonksiyonu daha hızlı çalışmaktadır. Bunun nedeni; her seferinde yeni sayıları listedeki asal sayılara bölmek yerine, bir önceki bölünmeden kalan sayıları 2 arttırıp o asal sayıya göre modunu almasıdır. Böylece yüksek basamaklı sayıların, listedeki asallara bölünmesinden dolayı oluşacak zaman kaybı minimuma indirilmektedir. Yüksek basamaklı asal sayıların bulunmasında asal_bul fonksiyonunda uygulanan yöntem kullanılmalıdır.
- 17 basamağa kadar olan Carmichael sayılarını elemek için Miller&Rabin ve Lehmann testlerinin en az 6 taban için çalıştırılması gerekmektedir.

Bu çalışmanın temel katkıları şunlardır :

- Rastsal ve Asal sayılar konusunda ilk Türkçe referanslardan birisi olması ve bu konuda güncel uygulamaların çoğunun açıklanması,
- Teorik olarak değinilen konuların bilgisayar ortamında uygulanması ve elde edilen deney sonuçları ışığında analizi,
- Güncel rastsal sayı üreteçlerinin ve rastsal sayı testlerinin incelenmesi ve irdelenmesi,
- *Ufak asallara bölme* ve *bölme yerine toplama* metoduyla asal sayıları daha hızlı bulma yöntemlerinin uygulanması ve bu yöntemlere dikkat çekilmesi.
- Güvenlik protokollerinde kullanılmak üzere, büyük (512, 1024 bit) asal sayıların (p, q) çarpımından oluşan modülus n değerlerinin; p ya da q 'nun asal olmaması (sözde asal) durumunda çok daha kolay çarpanlara ayrıldığının 16, 32, 64 ve 96 bit çarpanlar üzerinde gösterilmesi,
- İstatistik Sempozyumu 2000'de (DIE, 2000) verilen bildiride rastsal sayı testlerinin tanıtılması ve programla dilleriyle birlikte gelen rastsal sayı üreteçlerinin zayıflıklarının belirtilmesidir.

Bu çalışmanın ardından gelecek çalışmalar şunlardır:

- Gap ortamında yazılan *asal_mi* ve *asal_bul* fonksiyonlarının ve diğer asallık testlerinin gerekli formal değişikliklerden sonra GAP kullanıcılarına bir paylaşım paketi (*share package*) olarak sunulması ve gerekli dökümantasyonun yapılması,
- Elliptic Curve metodlarının kullanıldığı deterministik asallık testlerinin incelenmesi ve uygulanması,
- Cryptlib gibi (rastsal sayı üreteci ve kriptografik fonksiyonlardan oluşan) yazılım kütüphanelerinin ve Intel'in donanım tabanlı rastsal sayı üretecinin incelenmesi ve kullanılması,

- Tez süresince toplanan dökümanların, programların bir CD'de toplanması ve bu konudaki çalışmaların bir Internet sitesi aracılığıyla ilgilenenlerle paylaşılmasıdır.

KAYNAKLAR DİZİNİ

Baldwin, R.W., 1998, Preliminary Analysis of the BSAFE 3.x Pseudorandom Number Generators, RSA Laboratories' Bulletin, Number 8-September 3

Burrows, J.H., 1994, Digital Signature Standard (DSS), Federal InformationProcessing Standards Publication 186,

http://www.itl.nist.gov/fipspubs/fip186.htm

Boling, D., 1997, Random Lava, July 1997 issue of Microsoft Interactive Developer

Brown, R.H. , 1994, Security Requirements for Cryptography Modules

www.cerberussystems.com/INFOSEC/stds/fip140-1.htm

Callas , J., 1996, Using and Creating Cryptographic-Quality Random Numbers

Caldwell C.K., 1997, Finding Primes & Proving Primality,

http://www.utm.edu/research/primes/prove1.html

Caryl, M., 2000, Abstract Data Types

http://www.catachan.demon.co.uk/Programming/adt.html

Deley, D.W., 1991, Computer Generated Random Numbers, http://www.world.std.com/~franl/crypto/random-numbers.html

DIE, 2000, Devlet İstatistik Enstitüsü, İstatistik Araştırma Sempozyumu 2000, Bildiriler Kitabı

Eastlake D., Crocker S. and Schiller J., 1994, RFC 1750 - Randomness Recommendations for Security, Network Working Group, http://blitzen.canberra.edu.au/RFC/rfc/rfc1750.html

Emerson P., 1997, Prime Number Generation and Primality Testing, MSc Thesis for the degree of Bachelor of Arts with a major in Computer Science at Middlebury College.

KAYNAKLAR DİZİNİ (Devam)

FIPS140-2, 1999, FIPS 140-2, "Security Requirements for Cryptographic Modules." Federal Information Processing Standards Publication 140-2, U.S. Dept. of Commerce/NIST

GAPRef, 1999, GAP Release 4.1 - Reference Manual, The GAP Group, School of Mathematical and Computational Sciences University of St. Andrews, 26.07.99

GAPProg, 1999, GAP Release 4.1 – Programming in GAP 4, The GAP Group, School of Mathematical and Computational Sciences University of St. Andrews

GAPExt, 1999, GAP Release 4.1 – Extending GAP, The GAP Group, School of Mathematical and Computational Sciences University of St. Andrews

Garfinkel S., Spafford G., 1996, Practical Unix & Internet Security, Second Edition, O'Reilly & Associates , Inc., ISBN:1-56592-148-8

Granville A., 1992, Primality Testing & Carmichael Numbers, Notices Amer. Math. Soc. 39 (696s-700s)

Grantham, J., 1998, Frobenius Pseudoprimes, Institute for Defense Analyses, Center for Computing Sciences

Grantham, J., 1998, A Probable Prime Test with High Confidence, Journal of Number Theory 72, 32-47

Gutmann, P., 1999, Software Generation of Practically Strong Random Numbers

INTEL, 1999, The Intel Random Number Generator, Intel Platform Security Division

INTEL, 2000, Frequent Asked Questions about Intel Random Number Generator

http://developer.intel.com/design/security/rng/rngfaq.htm

KAYNAKLAR DİZİNİ (Devam)

Higgins, B.C., 2000, The Rabin-Miller Probabilistic Primality Test, Some Results on the Number of Non-Witnesses to Compositeness

Kaufman, C., Perlman, R. and Speciner, M., 1995, Network Security, Private Communication in a Public World

Kelsey, J., Schneier, B. and Ferguson N., 1999, Yarrow-160 Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator

Kelsey, J., Schneier, B. and Ferguson N., 2000, The Yarrow-160 Presentation Slides

Knuth, D.E., 1998, The Art of Computer Programming - Volume 2 Seminumerical Algorithms (Third Edition), Addison Wesley, ISBN:0-201-89684-2

L'Ecuyer, P., 1996, Random Number Generation, Chapter 4 of the *Handbook on Simulation*, Ed.: Jerry Banks, Wiley

Marsaglia, G., 1997 , Instructions for using DIEHARD: a battery of tests of randomness, http://stat.fsu.edu/~geo/diehard.html

Matthews, T., 1995, Suggestions For Random Number Generation In Software, An RSA Data Security Engineering Report

Maurer, U.M., 1994, Fast Generation of Prime Numbers& Secure Public-Key Cryptographic Parameters, to appear in Journal of Cryptography

Menezes, A. and Oorschot P., 1997, Handbook of Applied Cryptography, CRC Press

Moreau, T., 1997, Pseudo-Random Generators, a High-Level Survey-in-Progress, www.connotech.com/RNG.HTML

Moreau, T., 1996, **T.**, A Practical "Perfect" Pseudo Number Generator - Blum Blum Shub Generator,

http://www.connotech.com/BBS.HTM

KAYNAKLAR DİZİNİ (Devam)

NHSE, 1996, NHSE Random Number Software Catalog, Paul Coddington, Notheast Parallel Architectures Center at Syracuse University, http://nhse.npac.syr.edu:8015/rhse-rw/catalog/random/

NOLL L.C. and Cooper S., 2001, Seeding pseudo-random number generators with chaotic data, An unpublished draft, received by private communication from Landon Curt Noll (http://www.isthe.com/chongo/)

O'Connor, J.J. and Robertson E.F., 1996, History of Prime Numbers

http://www-history.mcs.st-andrews.ac.uk/history/HistTopics/Prime_numbers.html

Penzhorn W.T., 1992, Fast Algorithms For The Generation of Large Primes For The RSA Cryptosystem

Pinch, R.G.E., 1994, Some Primality Testing Algortihms, Proc 4th Rhine workshop on Computer Algebra, Karlsruhe

www.chalcedon.demon.co.uk/rcam.html

Pinch, R.G.E., 1998, The Carmichael Numbers Up to 10^{16}

www.chalcedon.demon.co.uk/rcam.html

PLAB, 2000, Tests for Random Numbers, PLAB project,

http://random.mat.sbg.ac.at/tests/

Ritter, 2000, Randomness Tests; Blum, Blum & Shub

http://www.io.com/%7eritter/NEWS2/TESTSBBS.HTM

Rivest, R., 1990, Finding Four Million Large Random Primes, Advances in Cryptology, CRYPTO'90, LNCS 537, pp. 625-626

KAYNAKLAR DİZİNİ (Devam)

Rivest, R., 1992, RFC 1321 - The MD5 Message Digest Algortihm, Network Working Group

RSA, 1998, RSA FAQ v4, Frequently Asked Questions About Today's Cryptography – What's Primality Testing?, http://www.rsasecurity.com/rsalabs/faq/2-5-1.html

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J. and Vo, S., Eylül 2000, Computer Security, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22, (http://csrc.nist.gov/rng/rng2.html).

Jun, B. And Kocher P., 1999, The Intel Random Number Generator, Cryptography Research, Inc., White Paper prepeared for Intel Corporation

Schneier B., 1996, Applied Cryptography (Second Edition), John Wiley & Sons Inc.

Scheepers, M., 2000, "MATH 100" Ders Notları, BSU Mathematics & Computer Science, http://diamond.idbsu.edu/~marion/teaching/

SCM, 2000, Scheme Library, Theory of the Miller&Rabin Test

http://www.sm.u-bordeaux2.fr/~corsini/Pedagogie/SCM/slib/The_Miller-Rabin_Test.html

Seth, A., 1999, The Data Encryption Page Newsletter - Vol. 1 No. 1 – 2,

http://visitweb.com/crypto/

Segre, A., 2000, Computer and Network Security, Iowa Üniversitesi "Data Security" Ders Notları (last modified 97),

www.geocities.com/SiliconValley/Network/2811/primes/lect8.htm

KAYNAKLAR DİZİNİ (Devam)

Silverman, R.D., 1997, Fast Generation of Random, Strong RSA Primes, RSA Laboratories' Crypto Bytes Magazine - Volume 3, Number 1

Soto, J., 1999 Juan Soto, "Statistical Testing of Random Number Generators", Proceedings of the 22nd National Information Systems Conference.

Stallings, W., 1998, Cryptography & Network Security Principles & Practice - Second Edition, Prentice Hall

Stinson, D.R., 1995, Cryptography Theory and Practice, CRC Press

SUN, 2000, Sun-OS Manual Pages

Tanenbaum, A.S., 1996, Computer Networks, Prentice Hall

Tezuka, S., L'ecuyer, P., 1991, Efficent and Portable Combined Tausworthe Random Number Generators, ACM Transactions on Modeling and Simulation, Vol.1, No.2, Pages 99-112

Walker, J., 1998, Ent Manual, www.fourmilab.ch/random/

EKLER

Ek 1 Kullanılan Kelimler ve Kısaltmalar

İngilizce	Türkçe
Base	Taban
Candidate	Aday
Complementary	Tümleyen
Composite	Bileşik
Congruential	Benzerlik
Candidate	Aday
Correlation	Bağlılaşım, korelasyon
Composite	Bileşik
CPU Time	İşlemci Zamanı
Density	Yoğunluk
Digital	Sayısal
Digitize	Sayısallaştırmak
Embedded	Gömülmüş (Yerleştirilmiş)
Emprical	Deneysel, Ampirik
Entropy	Entropi
Environment	Ortam
Event	Olay
Extensible	Genişletilebilir
Extent	Derece
External	Harici
Freedom	Serbestlik
Gap	Boşluk
Generation	Üretim
Hit	İsabet
Input	Girdi
Integer Numbers	Tam Sayılar

İngilizce	Türkçe
Interpreter	Çevirmen / Yorumlayıcı
Key	Anahtar
Network	Ağ
Noise	Gürültü
Number-theoretic	Sayı-teorik
Number Theory	Sayılar Kuramı
Observation	Gözlem
Pattern	Örüntü
Period	Periyodik
Pointer	Göstergeç
Position	Konum
Primality	Asallık
Procedure	Yöntem
Parent Process	Ana Süreç
Process	Süreç
Pseudo	Sözde
Pseudo Random Sequence	Sözde Rastgele Dizi
Random	Rastsal
Real Numbers	Gerçel Sayılar
Run	Koşu
Randomness	Rastsallık
Sample	Örneklem
Seed	Tohum
Sequence	Dizi
Square free	Kare bağımsız
Statistics	İstatistik
String	Dizgi
Suspect	Şüpheli

İngilizce	Türkçe
Test	Test
Thermal	Termal
Uniform	Birbiçimli / Eşyapılı
Workstation	İş istasyonu

Kısaltmalar:

ANSI : American National Standarts Institute

BBSG : Blum Blum & Shub Generator

CCG : Cubic Congruential Generator

FIPS : Federal Information Processing Standard

NIST-STS:National Institute of Standards and Technology Statistical Test Suite

KS : Kolmogorov-Smirnov

LCG : Linear Congruential Generator

MD : Message Digest

M&R : Miller & Rabin

PGP : Pretty Good Privacy

PRNG : Pseudo Random Number Generator – Sözde Rastsal Sayı Üreteci

QCG : Quadratic Congruential

RNG : Random Number Generator

RSA : Rivest, Shamir, Adleman

SHA : Secure Hash Algorithm

Ek 2 Web'den Çalıştırılan ECCP[49]'nin Ekran Çıktısı

% **ECPP (Version V3.4.1) by Fran\c{c}ois MORAIN** (morain@inria.inria.fr)

Working on 26898370231697
% Performing a quick compositeness test
% This number might be prime
% Entering ECPP
% **Starting phase 1: building** the sequence of primes
% Pmax=5000
% N_0=26898370231697
% next D is 0
% Cofactor after sieve is a probable prime
% D[[0]]=-1
% Factor= 157^1
% Factor= 2^4
% End of depth 0 at 0.000000 s
% Pmax=5000
% N_1=10707949933
% next D is 0
% Factorization completed using sieve only
% D[[1]]=-1
% Factor= 1907^1
% Factor= 953^1
% Factor= 491^1
% Factor= 3^1
% Factor= 2^2
% Cofactor is 1
% End of depth 1 at 0.000000 s
% Pmax=5000
% N_2=1907
% next D is 0
% Factorization completed using sieve only
% D[[2]]=-1
% Factor= 953^1
% Factor= 2^1
% Cofactor is 1
% End of depth 2 at 0.000000 s
% Pmax=5000
% N_3=953
% next D is 0
% Factorization completed using sieve only
% D[[3]]=-1
% Factor= 17^1
% Factor= 7^1
% Factor= 2^3
% Cofactor is 1
% Time for building is 0.000000 s
% Starting phase 2: proving
% Starting proving job for step 0
% N_0 is prime
% Time for proof[0] is 0.000000 s
% Starting proving job for step 1
% Using complete factorization theorem
% N_1 is prime
% Time for proof[1] is 0.000000 s
% Starting proving job for step 2
% N_2 is prime
% Time for proof[2] is 0.000000 s
% Starting proving job for step 3
% Using complete factorization theorem
% N_3 is prime
% Time for proof[3] is 0.000000 s
% Time for building is 0.000000 s
% Time for proving is 0.000000 s
% Total time is 0.000000 s
This number is prime

[49] Asallık deneyine www.fsp.com/cgi-bin/xrunecppc internet adresinden ulaşılabilir.

Ek 3 Die Hard Programının Kullanımı

Bu paketteki testler, 32-bitlik tam sayı üreten rastsal sayı üreteçlerini test etmek için geliştirilmiştir. Paketle birlikte gelen ek programlardan önemlileri aşağıdaki çizelgede gösterilmiştir (Marsaglia, 1997):

Program İsmi	Açıklaması
Asc2bin.exe	Hex verilerden oluşan (32-bit tamsayı 8 hex basamaktan oluşmalıdır) bir Ascii dosyayı, DieHard'in kullanabileceği bir binary dosyaya dönüştürür.
Makewhat.exe	Deneyin kullanabileceği büyük rastsal dosyalar üretmektedir. Bunun için pakette bulunan birçok rastsal sayı üreteci kullanılabilmektedir. Bu programın nasıl çalıştırıldığı EK-3'de gösterilmiştir.
meld.exe	Diğer 2 binary dosyayı birleştirerek yeni bir binary dosya oluşturur
Getmor.exe	32-bit rastsal tam sayıları okur ve standart normal rastsal sayı değişkenlerinden oluşan bir binary dosya oluşturur.

DieHard paketiyle birlikte gelen ek programlar ve açıklamaları

Diehard programını çalıştırdığımızda test edilecek dosyanın adını ve eğer deney sonuçlarından dosyaya çıktı alınması isteniyorsa, çıktı dosyasının adı istenmektedir. Daha sonra 15 testden seçim yapmamız beklenmektedir. Bu da bir dizi "1" veya "0" ile yapılmaktadır. "1" belirtilen testin seçildiğini, "0" seçilmediğini belirtir. Mesela "111111111111111" bütün testleri seçmektedir.

Diehard ↵

```
Enter filename (<=15 characters):
```

1meg.2 ↵

```
Enter name of output file (<=15 characters):
```

1meg.out ↵

```
Which tests do you want performed?
```

```
 For all tests, enter 15 1`s        111111111111111
 For, say, tests 1,3,7 and 14, enter      101000100000010
     HERE ARE YOUR CHOICES:
      1  Birthday Spacings
      2  Overlapping Permutations
      3  Ranks of 31x31 and 32x32 matrices
      4  Ranks of 6x8 Matrices
      5  Monkey Tests on 20-bit Words
      6  Monkey Tests OPSO,OQSO,DNA
      7  Count the 1`s in a Stream of Bytes
      8  Count the 1`s in Specific Bytes
      9  Parking Lot Test
     10  Minimum Distance Test
     11  Random Spheres Test
     12  The Sqeeze Test
     13  Overlapping Sums Test
     14  Runs Test
     15  The Craps Test
 Enter your choices, 1`s yes, 0`s no. using 15 columns:
123456789012345

111111111111111↵
```

Makewhat Programını Çalıştırma

```
C:\test\diehard>makewhat
This program makes a file of random integers
for tests by DIEHARD.  Select one from this list:
 1.    A multiply-with-carry (MWC) generator x(n)=a*x(n-
       1)+carry mod 2^32
 2.  A MWC generator on pairs of 16 bits
 3.  The "Mother of all random number generators"
 4.  The KISS  generator
 5.  The simple but very good generator COMBO
 6.  The lagged Fibonacci-MWC combination ULTRA
 7.      A   combination   MWC/subtract-with-borrow   (SWB)
         generator, period ~ 10^364
 8.  An extended congruential generator
 9.  The Super-Duper generator
10.  A subtract-with-borrow generator
11.  Any specified congruential generator
12.  The 31-bit generator ran2 from Numerical Recipes
13.  Any specified shift-register generator, 31 or 32 bits
14.  The system generator in Microsoft Fortran
15.  Any lagged-Fibonacci generator,  x(n)=x(n-r) op x(n-s)
16.  An inverse congruential generator

  Enter your choice, 1 to 16:       4
Enter four seed integers, not zero 4 5 6 9
      Please wait.............            FINISHED
  2,867,200 32-bit random integers (11,468,800 bytes)
have been written to the file KISS.32
```

Ek 4 Sistemden Bilgi Toplayan Perl Kodu[50]

```
#!/usr/local/bin/perl
# randbits - Gene Spafford spaf@cs.purdue.edu
# generate random seed string based on state of system
#
# Uses state of various kernel structures as random "seed"
# Mashes them together ad uses MD5 to spread around
#
# Usage : randbits [-n] [-h] [-H] [keylen]
#             where
#                   -n means to emit no trailing line feed
#                   -h means to give output in hex (default)
#                   -H mens hex output, but use uppercase letters
#                   keylen is the number of  bytes to the random key (default is 8)
#
# Edited & Changed by Enis Karaaslan - Master Thesis Work 1999-2001

$ENV{'PATH'} = "/bin:/usr/bin:/usr/etc:/usr/ucb:/etc:" . $ENV{'PATH'};

# we start with the observation that most machines have either a BSD
# core command set, or a system V-ish command set. We'll build from these

print "*******************************************\n";
print "* Sistem Bilgilerinden Rastsal Tohum-Veri Üretimi     *\n";
print "*     Bitirme Projesi 1999-2000 - Enis Karaaslan      *\n";
print "*******************************************\n";
print "Verilerin yazilacagi cikti dosyasinin adini giriniz:\n";
$FILE=<STDIN>;
#remove the new line
chop $FILE;

#open file -ek
open(DOSYA,">$FILE");

print "Kac Adet 16 Byte'lik Rastsal Bit üretilecegini giriniz:";
$answer=<STDIN>;
print ($answer*16*8);
printf " bit üretilecek. Lutfen Bekleyiniz\n";

#loop basi -ek
$keylen=16;
```

---

[50] Kaynak olarak (Garfinkel ve ark, 1996) alınmıştır.

```
for($i=1; $i < $answer+1 ; $i++ )
{

$BSD = "ps -agxlww ; netstat -s ; vmstat -s ;";
$SYSV ="ps -eflj ; netstat -s ; vmstat -nr ;";

if (-e "/sdmach") {
  $_="NeXT";
} elsif (-x "/usr/bin/uname" || -x "/bin/uname"){
  $_='/etc/version';
} else {
  die "How do I tell what OS this is?";
}

$_="SunOS 5.7";

/^AIX 1/&&(          $noise= $BSD . 'pstat -afipSsT') ||
/^CLIX 3/&&(         $noise= "ps -efl; nfsstat -nr") ||
/^DYNIX/&&(          $noise= $BSD . 'pstat -ai') ||
/^FreeBSD 2/&&(      $noise= $BSD . 'vmstat -i') ||
/^HP-UX 7/&&(        $noise= $SYSV) ||
/^HP-UX A.09/&&(     $noise= $SYSV . 'vmstat -s') ||
/^IRIX(64)? [56]/&&( $noise= $SYSV) ||
/^Linux 1/&&(        $noise= "ps -agxlww ; netstat -i; vmstat") ||
/^NeXT/&&(           $noise= 'ps agxlww ; netstat -s; vm_stat')||
/^OSF1/&&(           $noise= $SYSV . 'vmstat -i') ||
/^SunOS 4/&&(        $noise= $BSD . 'vmstat -afipSsT;vmstat -i') ||
/^SunOS 5/&&(        $noise= $SYSV . 'vmstat -i;vmstat -s') ||
/^SunOS 5.7/&&(      $noise= $SYSV . 'vmstat -i;vmstat -s')||
/^ULTRIX 4/&&(       $noise= $BSD . 'vmstat -s')||

die "No 'noise' commands defined for this OS.  Edit and retry !";
#### End of things you may need to modify

require 'getopts.pl';
require 'open2.pl';

($prog= $0) =~ s|.*/||;
#$usage="usage: $prog [-n] [-h | -H] [keylength] \n";

&Getopts('nhH') || die $usage;

defined($keylen=shift) || ($keylen=8);
die $usage if($keylen =~ /\D/);
die $usage if($opt_H && $opt_h);

die "Maximum keylength is 16 bytes (32 hex digits)\n" if ($keylen>16);
```

```
# Run the noise command and include whatever other state we
# can conviniently (portably find)

@junk=times();
$buf=`$noise` . $$ . getppid() . time . join(",%ENV) . "@junk" . 'ls
-lai';

#buf variable'inin degeri print ediliyor - test amacli
open(DENEME,">cikti");
print DENEME $buf;

&open2('m_out', 'm_in', "md5") || die "Cannot run md5 command: $!";
print m_in $buf;
close m_in;
$buf=<m_out>;

($buf =~ y/a-f/A-F/) if $opt_H;
#new line'i sil
chop $buf;
print DOSYA $buf;
print DOSYA "\n" unless $opt_n;

#wait for at least 4.6 seconds -ek
select(undef, undef, undef, 4.75 );

print ".";

}
#end of for loop -

close(DOSYA);
print "\n*****  islem sonu ******* \n";
```

Ek 5 BBS Üreteci Kodu

```
# Uluslararasi Bilgisayar Enstitisü - Bitirme Tezi Calismasi
# Enis Karaaslan 23.05.2001
# Blum Blum & Shub Rassal Sayi Üreteci

BlumNumbers := []; numbers :=[]; BlumUretSayi := 0;
p := 0; q := 0; s:= 0; n :=0; xson :=0; xilk :=0; BAdet := 0;
output := "cikti"; # cikti dosya
# R=320 icin p 95, q 97, s 95 basamak
basamak := 95; BasamakS := 95;

# ************************************************************************
OkuHex := function (DosyaAd)
local Fs, l, t, j, basamak2, basamak3, t2, t3;
j := 1;

if IsExistingFile(DosyaAd)=false then Print ("Dosya yok. islem sonu\n");
else Print ("Dosya var. islem Yapilabilir\n");

Fs := InputTextFile(DosyaAd);
numbers := [];
while not IsEndOfStream(Fs) do
        l := ReadLine(Fs);
        if l = fail then break; fi;
        if (basamak > Length(l)-1)
        then  # daha bundan sonra da okuma yapilacaksa
          l := l{[1..Length(l)-1]}; # strip newline
          basamak := basamak - Length(l);
else  # son alinacak sayilarsa
```

```
            # p sayisi icin okumak
            t := l{[1..basamak]};
            Add(numbers,t); Print("geldi\n");
            # q sayisi icin okumak
            basamak2 := 2 * (basamak) +3;
            t2 := l{[basamak+1..basamak2]};
            Add(numbers,t2); Print("geldi\n");
            # s sayisi icin okumak
            basamak3 := (basamak2 +1) + basamak;
            t3 := l{[basamak2+1..basamak3]};
            Add(numbers,t3); Print("geldi\n");
            break;
fi;
Add(numbers, l);
od;  # end of while - IsEndOfStream(Fs)

Print ("Dosya islemi sonu \n");
Print (numbers, "\n");
fi; # end of file operation
end; # end of function
# ******************************************************
# Bitleri decimal formata donusturmek ...
Sayisal := function(BitDizisi)
      local n, i;
      n := 0; BlumUretSayi :=0;
          for i in BitDizisi do
                    n := 2 * n + i;
          od;
        Print ("Bitten sayisal degere: ", n, "\n");
      BlumUretSayi := n;
end; # end of Sayisal
```

```
# ***************************************************
InitBlumPro := function (DosyaAd, BaseAdedi, AsalFlag, Istek, CiktiDosya)
local i, aday, flagx, AsalP, AsalQ, sayi, BasamakS, flag, dsayac, gecici;
flag := 0; AsalP :=0 ; AsalQ :=0 ;
sayi := [2 .. 10000001];

# AsalFlag 1 ise random veriden p,q degerlerinin olusturulacagini belirtir.
output := OutputTextFile (CiktiDosya, false);
if (AsalFlag = 0) then
        while i <= 1028 do
                if XPrimes[i] mod 4 = 3 then
                AddSet(BlumNumbers, XPrimes[i]);
                fi;
                i := i + 1;
        od;
        p := Random (BlumNumbers);
        q := Random (BlumNumbers);
        n := p * q;
        s := Random (sayi);
else
        OkuHex(DosyaAd); # p,q, s degerleri hesaplaniyor
        flagx := 0;
        aday := hextoint(numbers[1],basamak);
        while flagx = 0 do
                AsalP := asal_bul( aday , Istek, BaseAdedi);
                if AsalP mod 4 = 3 then flagx :=1;
                else aday := AsalP + 2; # 1 sonraki sayiyi dene
                fi;
        od;

        flagx := 0;
        aday := hextoint(numbers[2],basamak+2);
        while flagx = 0 do
                AsalQ := asal_bul( aday , Istek, BaseAdedi);

                if AsalQ mod 4 = 3 then flagx :=1;
                else aday := AsalQ + 2; # 1 sonraki sayiyi dene
                fi;
        od;
```

```
        p := AsalP;
        q := AsalQ;
        n := p * q;
        s := hextoint (numbers[3], basamak);
        s := s mod n; # 1 ile n arasinda bir sayi olacak */
fi; # end of if-then-else
flag := Gcd(s,n);
while flag <> 1 do
        s := s + Random(sayi);
        flag := Gcd(s,n);
        # Print ("Donguye girdi \n");
od;

Print ("p:=",p," q:=",q," n:=",n," s:=", s,"\n");
xilk  := (s * s) mod n;

end; # end of initblumpro
# ***************************************************************
BlumPro := function(BAdet, bitadedi)

local xsayac, dsayac, B, bits, x, AraDeger;
x := 1; xsayac := BAdet;
bits :=[];

while x <= xsayac do
        dsayac := 1;
        while dsayac <= bitadedi do
                xson := (xilk * xilk) mod n;
                B := xson mod 2;
                Add(bits,B);
                xilk := xson;
                dsayac := dsayac + 1;
        od; # end of while dsayac ...
        x := x + 1;
od; # end of x < xsayac

Sayisal(bits);
bits := [];

end; # end of blum_pro
# ***************************************************************
Blum := function(BitAdedi, AsalFlag)

local i,p,q,n,s, sayi, flag, dsayac, xilk, xson, B, bits;

i:=1; flag := 0;

sayi := [2 .. 10000001];
```

```
if (AsalFlag = 0) then
        while i <= 1028 do
                if XPrimes[i] mod 4 = 3 then
                AddSet(BlumNumbers, XPrimes[i]);
                fi;
                i := i + 1;
        od;
        p := Random (BlumNumbers);
        q := Random (BlumNumbers);
else
        p := AsalP;
        q := AsalQ;
fi; # end of if-then-else

n := p * q;
Print ("p :=",p," q:=",q," n:=",n,"\n");
s := Random(sayi);
flag := Gcd(s,n);

while flag <> 1 do
        s := Random(sayi);
        flag := Gcd(s,n);
        Print ("Donguye girdi \n");
od;

Print ("Random s :=",s,"\n");
dsayac := 1;
xilk  := (s * s) mod n;
bits :=[];
```

```
while dsayac <= BitAdedi do
        xson := (xilk * xilk) mod n;
        B := xson mod 2;
        Print(B);
        Add(bits,B);
        xilk := xson;
        dsayac := dsayac + 1;
od;

Print ("\n"); Sayisal(bits);
Print ("Decimal karsiligi: ", BlumUretSayi," olmaktadir\n");
end; # end of Blum
```

Ek 6 BBS Üretecinden Örnek Çıktılar

Blum(BitAdedi, AsalFlag) Fonksiyonu Çıktıları

Örnek1: 100 bit'lik rastsal sayı üretmek

```
gap> Blum(100, 0);
p :=7499 q:=7547 n:=56594953
Random s :=6117591
00011001101100001011110000111101100011110001101000010000011
00000110011010111101100101111110100000100
Decimal      karsiligi:      1272125620839606679555058557 48
olmaktadir
```

Örnek2: 1024 bit'lik rastsal sayı üretmek

```
gap> Blum(1024, 0);
p :=7723 q:=5867 n:=45310841
Random s :=6217525
10011001000001001100100101000100110010011010000111101100011
11011100111111111100100110011111111001111110010111100011101 10
11010110100110111010010111011001110010111110010111001110111
01001011110010110111101010101011111001001001111110100111111
10110000001001111110000111111010001110000110011011110110010
10110000111111110011101111000110011001111010010100001111010
10111011001111101001100011111010001000000011111111011101000
01010001110011000110010001010010001010011101011010000011100
00010101100101000011000011101011001111111100100011100100000
11110100110000101101001111001010001001100011011101101110011
11001010001101001100101100011100100011100101000100001010011
10000000111001100101101101111010110011011111001110111010110
00101001011011001111101001111101100000001001010000011001010
11011000010111100001111111011001011011011011101101001100000
10000001111011101011010001011111011110110101101001100011000
```

```
00110010011001001010010101000011101100001111010010101111111
01010011101011101111111010101101110100110110100100001111111
01100111111010111011011
Decimal karsiligi:
10745338262285513132606334338591632226629268415411758698532
71619430946923372723562957346201409663005073119600690954798
20996831684899752016625397696454327034206232737809756756025
26134541386656386034902624782217867187264064477989992882879
57277555419492437896609645729733060811259436237134031998131
17860075339227
olmaktadir
```

BlumPro Fonksiyonu Örnek Çıktılar

"rassal1606" dosyasından alınan değerlerle 2048 bit rastsal sayı üretmek

```
gap> Reread ("asalson2001mod.txt");
gap> Reread ("Blumpro2001_FairCoin");
gap> InitBlumPro ("rassal1606", 8, 1, 1, deneme.txt);
Dosya var. islem Yapilabilir
Dosya islemi sonu
["e8df737d8a3670ba348c2ecb701aee6a249c3d0aba1960f44bb0ece4c
d3614a9f2597305a6ea47bc39cea2e64bc6a2f",
"8fbd6702aeb9832a78954c62a5eab4fa8f6370a2f7ed5ece986cfe8a37
c6a7e835eeff66bcbe36592fb3625b8061acbc25",
"4c5130594c16e763dd1bf25200cf2f2b49b0a9968c46151f47bf0751c9
16e52f1b9892118bd6f9fbbeec41ccaf5726bf" ]
p:=224015080928419959583027272071247600839161837644250673343
03023583025585604568740924994691583458 3
q:=566363858114242317724377197364590570033638896716260681236
99645969212273695876452777717926110421 47
n:=126874045510394152204124020951629625533551417406639929320
85135707476731500881304075414386436887267105477121835254097
21236552156382430538164884456042992637676017683326358822355
518570327893169701
```

```
s:=11746283517807225440964898712688734950029535942082755998
777757655571654407293232507112675548037


gap> BlumPro(256, 8);

bitten sayisal degere:

439079295622978143196796627801757921978683880224845090131 53
904433493252230265726445837030747071439499261372209737883 89
388517781941883246326243465055432223683420431992512177850 34
334274285716232930358911988540865290694170503271307977284 06
127121248445480090837487298593312653351385049891000031564 67
884989582849252430637721023554155317776389766007023105035 66
645119331501521035277885794453270031819238474058597396156 13
903799088852168586607507162208426016880905810234803586402 48
699982761296745438225695663294285919093956769202305023092 05
180856754766819006020567545655570839210129643266796689981 10
165290127530915375019029 2
```

Ek 7 FIPS 140 Kaynak Kodu

```
# Uluslararasi Bilgisayar Enstitisü - Bitirme Tezi Calismasi
# Enis Karaaslan -12.09.2000
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include "c:\test\ent\iso8859.c"
#include "c:\test\ent\iso8859.h"
#include "c:\test\ent\randtest.h"
#define FALSE 0
#define TRUE  1
#define V (void)

void monobit(void);
void poker(void);
void kosu (void);
void displayBits(unsigned);
void initialize (void);
int dosya_ac();

/* Scan input file and count character occurrences */
  int b, oc, i, sayac = 0;
  unsigned char ocb, ob ;
  int binary = TRUE;
  long ccount[2];           /* Bins to count occurrences of values */
  long totalc = 0;          /* Total character count */
  FILE *fp ;
  char *DosyaAdi;
```

```
/* ********************************************** */
 int main(int argc, char *argv[]) {
 int secim,tus;
  if (argc !=2)  {printf ("Usage : test dosya_adi \n"); exit(1); }
 else DosyaAdi = argv[1];

 while (1) {
        printf ("\nFIPS 140-1 TESTLERI \n
        [1]Monobit Testi\n
        [2]Poker Testi\n
        [3]Kosu Testi\n
        [4]Uzun Kosu Testi\n
        [5]Programdan cikis\n\n");
         printf   ( "Seciminizi Girin lutfen : \n");
        scanf ("%d", &secim);

      switch (secim) {
        case 1:
                initialize(); monobit(); break;
        case 2:
                poker(); break;
        case 3:
                kosu(); break;
        case 4:
                printf ("Kosu testinde eger varsa uzun kosu degerleri print
                edilmektedir \n");
                break;
        case 5:
                exit(0); break;
        default :
```

```
                printf ("Yanlis Secim. Lutfen tekrar giriniz \n"); break;
        } /* end of switch case */
    } /* end of while */
  } /* end of main function */
/* ********************************************** */
  void monobit(){
   unsigned displayMask1 = 0x80; int c;
   memset (ccount, 0, sizeof ccount);          /* Bellekte temizlenir */
   for (i=0; i < 2; i++) ccount[i] = 0;
   sayac = 0;

   if (dosya_ac()) exit(0);                    /* Dosya acilmazsa geri don */
   for (sayac=0; sayac < 2500 ; sayac++)
   {
        ob = ' '; oc = ' '; ocb = ' ';
        oc = fgetc(fp);
        ocb = (unsigned char) oc;
        displayBits(ocb);
        totalc += binary ? 8 : 1;
        ob = ocb;

          for (c=1; c <= 8; c++) {
                if (ob & displayMask1) ccount[1]++;
                else ccount[0]++;
                ob <<= 1;
          }
} /* end of outer for loop */
 fclose(fp);
 printf("Value Char Occurrences Fraction\n");
        for (i = 0; i < (binary ? 2 : 256); i++) {
```

```
            if (ccount[i] > 0) {
                printf("%3d  %c  %10ld  %f\n", i,
                    (!isISOprint(i) || isISOspace(i)) ? ' ' : i,
                    ccount[i], ((double) ccount[i] / totalc));
            } /* end of if
          } /* end of for i loop */

 printf("\nTotal:   %10ld  %f\n\n", totalc, 1.0);
    if ( (ccount[1] > 9654) &  (ccount[1] < 10,346) )
          printf ("Testi basariyla gecti\n");
    else  printf ("Testi gecemedi");
  } /* end of monobit function */
/* ********************************************** */
void displayBits(unsigned value)
{       unsigned displayMask = 0x80;
        unsigned c;
        printf ("%7u = ", value);

        for (c=1; c <= 8; c++) {
                putchar (value & displayMask ? '1' : '0' );
                value <<= 1;
                if (c%8 == 0) putchar(' ');
        } /* end of for loop */
        putchar ('\n');
} /*end of fn */
/* ********************************************** */
void poker(){
 int pokercount[16]; /* 16 kombinasyonun gecme SIKLIGI */
 unsigned MASK1 = 0x0F;
 unsigned MASK2 = 0xF0;
```

```
float toplam = 0, sonuc = 0;
ob = ' '; oc = ' '; ocb = ' ';
memset (pokercount, 0, sizeof pokercount);          /* Bellekte temizlenir */

for (i=0; i < 16; i++) pokercount[i] = 0;
if (dosya_ac()) exit(0);                            /* dosya acilmazsa geri don */

for (sayac=0; sayac < 2500 ; sayac++)
{     ob = ' '; oc = ' '; ocb = ' ';
      oc = fgetc(fp);
      ocb = (unsigned char) oc;
      totalc += 8 ;
      ob = ocb;
      ob = ob & MASK1;  /* 0000 abcd */

   ocb = ocb & MASK2; /* xvyz 0000      */
   ocb = ocb >> 4;         /* sola 4 bit shift */
                           /* 0000 xvyz      */
   pokercount[ob]++;
   pokercount[ocb]++;
} /* end of for loop */
fclose(fp);
toplam = 0; sonuc = 0;

for (i=0; i < 16; i++)
{ printf (" %d gelme adedi : %d \n", i,pokercount[i]);
  toplam = toplam + ( pokercount[i] * pokercount[i] );}
 printf ("Toplam : %.4f \n", toplam );
 sonuc = ((16 * toplam) / 5000) - 5000;
 printf ("Sonuc : %.4f ", sonuc );              /* ondalikli */
```

```
   if ( (sonuc > 1.03) &  (sonuc < 57.4) ) printf ("Testi basariyla gecti\n");
    else   printf ("Testi gecemedi");
} /* end of poker */
/* ********************************************** */
void kosu(){
int OCountArray[7], ZCountArray[7], ZeroCount=0, OneCount=0;
int dummy = 0;
sayac =0;
totalc =0;

      for (b=0;b<7;b++) {OCountArray[b]=0; ZCountArray[b]=0;} /* initialize */
      b = 0;
      if (dosya_ac()) exit(0);       /* dosya acilmazsa geri don */

        for (sayac=0; sayac < 2500 ; sayac++)
        {       ob = ' '; oc = ' '; ocb = ' ';
                oc = fgetc(fp);
                ocb = (unsigned char) oc;
                totalc += binary ? 8 : 1;
                ob = ocb;
                for (b = 0; b < 8; b++) {
                if (ob & 1)  {                          /* 1'dir */
                        OneCount ++;                    /* 1 sayisini arttir */
                         if (ZeroCount == 0 ) dummy++;  /* Birsey yapma */
                else
                {if((ZeroCount >0)&(ZeroCount < 6) ) ZCountArray [ZeroCount] ++;
                else if ((ZeroCount >= 6) & (ZeroCount < 34)) ZCountArray [6] ++;}

                if (ZeroCount >= 34 ) {
```

```
                printf ("Long Run of  length %d is present ", ZeroCount);
                ZCountArray [6] ++;}
                ZeroCount = 0;}
        else {                                  /* 0'dir */
                ZeroCount ++;                   /* 0 sayisini arttir */
                if (OneCount == 0 ) dummy++;    /* Birsey yapma */
                else
        {if((OneCount >0)&(OneCount < 6) )
                        OCountArray [OneCount] ++;
        else if ((OneCount >= 6 ) & (OneCount < 34) )
                        OCountArray [6] ++;
        }
        if (OneCount >= 34 ) {
                printf ("Long Run of  length %d is present ", ZeroCount);
                OCountArray [6] ++;}
        OneCount = 0;
        } /* end of if-then-else */
        ob >>= 1;  /* shift left one bit */
        } /* end of for statement */
 } /* end of for loop */
 fclose(fp);

for (i=1; i <= 5; i++)
{
printf (" %d adet 0 gelme adedi : %d \n", i,ZCountArray[i]);
printf (" %d adet 1 gelme adedi : %d \n", i,OCountArray[i]);}
 printf (" 6 veya daha fazla 0 gelme adedi : %d \n",ZCountArray[i]);
 printf (" 6 veya daha fazla 1 gelme adedi : %d \n",OCountArray[i]);
 printf ("0 için : \n");
printf("Beklenen 1 uzunlugunda calisma 2267-2733 gozlenen:%d\n",ZCountArray[1] );
```

```
printf("Beklenen 2 uzunlugunda calisma 1079-1421 gozlenen:%d\n",ZCountArray[2] );
printf ("Beklenen 3 uzunlugunda calisma  502- 748 gozlenen:%d \n",ZCountArray[3] );
printf ("Beklenen 4 uzunlugunda calisma  223- 402 gozlenen:%d \n",ZCountArray[4] );
printf ("Beklenen 5 uzunlugunda calisma   90- 223 gozlenen:%d \n",ZCountArray[5] );
printf ("Beklenen 6 uzunlugunda calisma   90- 223 gozlenen:%d \n",ZCountArray[6] );
printf  ( "1 e basip devam edin : \n");
scanf ("%d", &dummy);
printf ("1 için : \n");
printf ("Beklenen 1 uzunlugunda calisma 2267-2733 gozlenen:%d\n",OCountArray[1]);
printf ("Beklenen 2 uzunlugunda calisma 1079-1421 gozlenen:%d\n",OCountArray[2]);
printf ("Beklenen 3 uzunlugunda calisma  502- 748 gozlenen:%d\n", OCountArray[3]);
printf ("Beklenen 4 uzunlugunda calisma  223- 402 gozlenen:%d\n", OCountArray[4]);
printf ("Beklenen 5 uzunlugunda calisma   90- 223 gozlenen:%d\n", OCountArray[5]);
printf ("Beklenen 6 uzunlugunda calisma   90- 223 gozlenen:%d\n", OCountArray[6]);
} /* end of kosu */
/* ********************************************** */
void initialize (void) {
        int j;
        for (j = 0; j < 2; j++) ccount[j] = 0;
        totalc = 0;
  }
/* ********************************************** */
int dosya_ac() {
        if ((fp = fopen(DosyaAdi, "rb")) == NULL)
        { fprintf(stderr, "Cannot open output file.\n");
          return 1;
        }else return 0;
}
/* ********************************************** */
```

Ek 8 GAP Yazılımı Hakkında Detaylı Bilgi

GAP kısaltması, Gruplar Algoritmalar ve Programlama'yı (*Groups, Algorithms and Programming*) simgeler. Bu amacı belirtmek için seçilmiş bir addır. GAP, Discrete Abstaract Algebra'da hesaplama yapmak için geliştirilmiş bir yazılım paketidir. GAP, bedava, açık ve eklemelerle genişletilebilirdir. Sistemin genişletilebilir olmasının anlamı şudur; kullanıcı GAP dili ile yazdığı kendi ek programlarını, sistemdeki herhangi bir programı kullandığı gibi çalıştırabilmektedir. Ayrıca isteyen kullanıcılar kendi yazdıkları programları paylaşım paketleri (*share package*) şeklinde diğer kullanıcılarla paylaşabilmektedirler. Ayrıca var olan yöntemlerin değiştirilerek kullanılması da mümkündür. GAP paylaşım paketlerinin büyük bir kısmı GAP dilinde yazılmış olmasına karşın, C dilinde ve hatta diğer bazı dillerde yazılıp sisteme entegre edilmiş bazı paketler de mevcuttur. Hangi dilde yazılmış olursa olsun bu paketlerdeki fonksiyonlar direkt GAP içerisinden çağrılabilirler ve işlem sonuçları direkt GAP ortamına dönecektir. Sistem 4 ana parçadan oluşmaktadır (GAPRef, 1999):

1. **Kernel**

   C dilinde yazılmıştır ve kullanıcıya sağladıkları aşağıda sıralanmıştır:

   - Otomatik dinamik bellek yönetimi sağlanmaktadır. Kullanıcının bu gibi detayla ilgilenmesine gerek yoktur.
   - Temel aritmetik ve cebirsel fonksiyonlar bulunmaktadır.
   - GAP dili için yorumlayıcı (*interpreter*) bulunmaktadır.
   - Altta çalışan işletim sisteminden bağımsız olarak; dosya işlemleri ve dış programların çalıştırılması gibi temel işlemlere izin veren bazı sistem fonksiyonları bulunmaktadır.
   - Algoritmaları test etmek, debug etmek ve işlem zamanlarını belirlemek için gerekli fonksiyonlar (işlevsellik) sağlanmaktadır.

2. **GAP Fonksiyon Kütüphanesi**: Cebirsel işlemleri ve diğer bazı algoritmaları gerçekleştiren fonksiyonlar mevcuttur. Bu

fonksiyonlar kullanıcının incelemesine ve değiştirmesine açıktır.

3. **Grup Teorik Veri Kütüphanesi:** Gruplar hakkında teorik bilgi bulunmaktadır.

4. **Dökümantasyon** : Tex veya HTML formatında yazılmış dökümantasyon mevcuttur.

GAP ortamında programlama hakkında detaylı bilgi için (GAPProg, 1999)'dan, GAP paketine ek paket yazarken dikkat edilmesi gerekenler için (GAPExt, 1999)'dan yararlanılabilir.

Tarihçe

1985 yılında Aachen'de Prof. Joachim Neubuser tarafından Lehrstuhl Üniversitesi Matematik bölümünde GAP yazılımı geliştirilmeye başlandı. 1997 yılında Prof. Neubuser'in emekliliğine kadar GAP yazılımının gelişimi aynı üniversitede devam etti. Bu yıla kadar yazılım, uluslararası platformda kullanılmaya ve geliştirilmeye başlanmıştı. Yazılım bundan sonraki yıllarda St. Andrews Üniversitesi Matematik ve Bilgisayar Bilimleri Bölümü'nde geliştirilmeye devam etti. Aachen'de halen geliştirilmekte olan sistemin, tekrar tasarlanması ve neredeyse tekrar yazılması çalışmaları bitirildi. Daha kullanılabilir bir ortam kazanan yazılım paketi 4.1 versiyonu ile genel kullanım için dağıtılmaya başlandı. Çalışmalar halen iki üniversite tarafından ortaklaşa devam ettirilmektedir. GAP Yazılım Paketi, finansal olarak birçok kuruluş ve Avrupa Komisyonu (ESPRIT programı) tarafından desteklenmektedir.

Uygulama

Projeyi gerçekleştirirken GAP yazılımının ilk önce 4.1 ve sonra 4.2 versiyonu kullanılmıştır. Bu sistemin ilk önce PC üzerinde denemeleri yapılmış, asıl olarak Sun Solaris işletim sistemi üzerinde kullanılmıştır.

Editörde yazılan programın GAP ortamına aktarılması

Herhangi bir text editörde yazdığımız programı (programın ana dizininde (*root*) bulunuyorsa),

Read ("dosya_adı.dosya_eki");

komutuyla GAP ortamına aktarabilir ve içindeki fonksiyonları kullanabiliriz. Dosyada yapılacak herhangi bir düzenlemeden sonra değişikliklerin geçerli olması için,

Reread ("dosya_adı.dosya_eki");

komutu ile dosya sisteme tekrar okutulmalıdır.

Eğer dosya farklı bir dizindeyse tam konumu belirtilmelidir:

Reread ("/usr/users/students/enisk/gapkod/TestBBS");

Ek 9 Asallık Test Kodları

```
# Uluslararasi Bilgisayar Enstitüsü - Bitirme Tezi Calismasi
# Enis Karaaslan 06.2001

Sayac_UfakBol :=0;
Sayac_MilRab :=0;
Sayac_Lucas :=0;
sayac := 1; # kacinci denemede basarili oldugunu tutar
asbolsayac :=1; # asallara bolunen sayilarin sayaci
hexdigits :="0123456789abcdefABCDEF";
hexvals := [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,10,11,12,13,14,15];
numbers :=[];

############################################################
# 8191 e kadar olan asal sayilar
# RSA EURO'dan alinmistir
############################################################
XPrimes :=
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47,
53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313,
317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389,
397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461,
463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547,
557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617,
619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691,
701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773,
787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859,
863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947,
953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021,
1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 1087, 1091,
1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163,
1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231,
1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301,
1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399,
1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459,
1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531,
1543, 1549, 1553, 1559, 1567, 1571,.1579, 1583, 1597, 1601,
1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657, 1663, 1667,
1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747,
1753, 1759, 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831,
1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907,
```

1913, 1931, 1933, 1949,.1951, 1973, 1979, 1987, 1993, 1997,
1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2069,
2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137,
2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213, 2221, 2237,
2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297,
2309, 2311,.2333, 2339, 2341, 2347, 2351, 2357, 2371, 2377,
2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441,
2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543,
2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633,
2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693,
2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749, 2753,
2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837,
2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917,
2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999, 3001, 3011,
3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089,
3109, 3119, 3121, 3137, 3163, 3167, 3169, 3181, 3187, 3191,
3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271,
3299, 3301, 3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347,
3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433, 3449,
3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527,
3529, 3533, 3539, 3541, 3547, 3557, 3559, 3571, 3581, 3583,
3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671,
3673, 3677, 3691, 3697, 3701, 3709, 3719, 3727, 3733, 3739,
3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833,
3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917,
3919, 3923, 3929, 3931, 3943, 3947, 3967, 3989, 4001, 4003,
4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079,
4091, 4093, 4099, 4111, 4127, 4129, 4133, 4139, 4153, 4157,
4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243,
4253, 4259,.4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337,
4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409, 4421, 4423,
4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513,
4517, 4519, 4523, 4547,.4549, 4561, 4567, 4583, 4591, 4597,
4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673,
4679, 4691, 4703, 4721, 4723, 4729, 4733, 4751, 4759, 4783,
4787, 4789, 4793, 4799, 4801, 4813,.4817, 4831, 4861, 4871,
4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943, 4951,
4957, 4967,.4969, 4973, 4987, 4993, 4999, 5003, 5009, 5011,
5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101,
5107, 5113, 5119, 5147, 5153, 5167, 5171, 5179, 5189, 5197,
5209, 5227, 5231, 5233,.5237, 5261, 5273, 5279, 5281, 5297,
5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399,
5407, 5413, 5417, 5419, 5431, 5437, 5441, 5443, 5449, 5471,
5477, 5479, 5483, 5501, 5503, 5507,.5519, 5521, 5527, 5531,
5557, 5563, 5569, 5573, 5581, 5591, 5623, 5639, 5641, 5647,
5651, 5653,.5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711,
5717, 5737, 5741, 5743, 5749, 5779, 5783, 5791, 5801, 5807,
5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857, 5861, 5867,

```
5869, 5879, 5881, 5897,.5903, 5923, 5927, 5939, 5953, 5981,
5987, 6007, 6011, 6029, 6037, 6043, 6047, 6053, 6067, 6073,
6079, 6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151,
6163, 6173, 6197, 6199, 6203, 6211,.6217, 6221, 6229, 6247,
6257, 6263, 6269, 6271, 6277, 6287, 6299, 6301, 6311, 6317,
6323, 6329, 6337, 6343, 6353, 6359, 6361, 6367, 6373, 6379,
6389, 6397, 6421, 6427, 6449, 6451, 6469, 6473, 6481, 6491,
6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581,
6599, 6607, 6619, 6637,.6653, 6659, 6661, 6673, 6679, 6689,
6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779,
6781, 6791, 6793, 6803, 6823, 6827, 6829, 6833, 6841, 6857,
6863, 6869, 6871, 6883, 6899, 6907,.6911, 6917, 6947, 6949,
6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013,
7019, 7027,.7039, 7043, 7057, 7069, 7079, 7103, 7109, 7121,
7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213,
7219, 7229, 7237, 7243, 7247, 7253, 7283, 7297, 7307, 7309,
7321, 7331, 7333, 7349,.7351, 7369, 7393, 7411, 7417, 7433,
7451, 7457, 7459, 7477, 7481, 7487, 7489, 7499, 7507, 7517,
7523, 7529, 7537, 7541, 7547, 7549, 7559, 7561, 7573, 7577,
7583, 7589, 7591, 7603, 7607, 7621,.7639, 7643, 7649, 7669,
7673, 7681, 7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741,
7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841, 7853,
7867, 7873, 7877, 7879, 7883, 7901, 7907, 7919, 7927, 7933,
7937, 7949, 7951, 7963, 7993, 8009, 8011, 8017, 8039, 8053,
8059, 8069, 8081, 8087,.8089, 8093, 8101, 8111, 8117, 8123,
8147, 8161, 8167, 8171, 8179, 8191] ;

############################################################
#                    OKUHEX
# Dosyadan hex okuma islemleri
############################################################
okuhex:= function (DosyaAd, BasamakAdedi)

local s, l;

if IsExistingFile(DosyaAd)=false
        then Print ("Dosya yok. islem sonu\n");
else Print ("Dosya var. islem Yapilabilir\n");

        s := InputTextFile(DosyaAd);
        numbers := [];

        while not IsEndOfStream(s) do
                l := ReadLine(s);
                if l = fail then break;fi;
                if (BasamakAdedi > Length(l)-1)
                        then  # daha bundan sonra da okuma yapilacaksa
```

```
                    l := l{[1..Length(l)-1]}; # strip newline
                    BasamakAdedi := BasamakAdedi - Length(l);
                else  # son alinacak sayilarsa
                    l :=l{[1..BasamakAdedi]};
                    Add (numbers, l); Print ("geldi \n");
                    break;
                fi;
                Add(numbers, l);
        od;

        Print ("Dosya islemi sonu \n");
        Print (numbers[1], "\n");
fi; # end of file operation
end;
############################################################
#                        HEX to INT
# hex sayilardan olusan string i alip integer sayi donduren fonksiyon
############################################################
hextoint:= function(HexDeger, BasamakAdedi)

local c, n, string, sayi;
n := 0;
sayi:=0;

# String den alinan hex sayisal bilgi tamsayiya donusturuluyor
for c in HexDeger do
        n:= 16*n + hexvals[Position(hexdigits,c)];
od;
sayi := n;

# Burada basamak sayisini istenen BasamakAdedi sayisina donusturuyoruz
# Bunu string e donusturerek ve Length komutunu kullanarak yapiyoruz
string := String(sayi);
if (BasamakAdedi < Length(string) ) then string := string{[1..BasamakAdedi]};
fi;

n := Int(string); # Geriye kalan string tamsayiya donusturuluyor
return n;
end; # program sonu

############################################################
#                    LUCAS PSEUDOPRİME TEST
# GAP 4R1 versiyonundaki integer.gi library dosyasindan alinmistir.
############################################################
TraceModQF := function ( p, k, n )
  local  trc;
  if k = 1  then
```

```
    trc := [ p, 2 ];
  elif k mod 2 = 0  then
    trc := TraceModQF( p, k/2, n );
    trc := [ (trc[1]^2 - 2) mod n, (trc[1]*trc[2] - p) mod n ];
  else
    trc := TraceModQF( p, (k+1)/2, n );
    trc := [ (trc[1]*trc[2] - p) mod n, (trc[2]^2 - 2) mod n ];
  fi;
  return trc;
end;
############################################################
#                    LUCAS ANA FONKSİYON
############################################################

Lucas := function(Sayi)
local p;

# find a quadratic nonresidue $d = p^2/4-1$ mod $n$
p := 2;  while Jacobi( p^2-4, Sayi ) <> -1  do p := p+1;  od;

# for a prime $n$ the trace of $(p/2+\sqrt{d})^n$ must be $p$
# and the trace of $(p/2+\sqrt{d})^{n+1}$ must be 2
if TraceModQF( p, Sayi+1, Sayi ) = [ 2, p ]  then
    return true;
fi;

# $n$ is not a prime
return false;
end;

# ############################################################
#                    LEHMANN TESTI
# Sayi             : Asallik Testinin uygulanacagi sayi
# Yontem           :Taban olarak alinacak sayinin elde edilme yontemini belirler
#         0                          : rastsal x sayi alinacak
#         Diger degerler icin        :ilk x asal sayi alinacak
# TabanAdedi     : Taban ("base") olarak alinacak sayi adedi
############################################################
Lehmann := function(Sayi, Yontem, TabanAdedi)

local SayacAna, i, x, g, countpos, countneg;

# initialize
g:= [2 .. 100000] ;
SayacAna := 1;  # initialize
countpos := 0; countneg :=0;
# negatifse pozitif sayiya cevir
```

```
if Sayi<0 then Sayi:=-Sayi; fi;

# cift bir sayi ise asal degildir
if (IsEvenInt(Sayi)=true) then  return false;  fi;

while SayacAna <= TabanAdedi do
        if Yontem = 0 then
                x := Random ( g );
        else
                # ilk asallar "base" olarak alinir
                x := XPrimes[SayacAna];
        fi;

        x := x mod Sayi;
        i := (Sayi - 1) /2;
        x := PowerModInt(x, i, Sayi);                   # x^i mod sayi  yi hesaplar

        if (x = 1) then
                countpos := countpos + 1;
        else
                if (x = Sayi - 1) or (x = -1)
                then countneg := countneg + 1;
                else return false;              # Asal degil
                fi; #innermost if
        fi; #outer if

        SayacAna := SayacAna + 1;
od; # End of while

if (countpos<TabanAdedi)  then return true;
else return false;                              # Asal degil
fi;

end; # fonksiyon sonu

##########################################################
#                   MILLER & RABIN TESTI
# Sayi              : Asallik Testinin uygulanacagi sayi
# Yontem            :Taban olarak alinacak sayinin elde edilme yontemini belirler
#       0                               : rastsal x sayi alinacak
#       Diger degerler icin             :ilk x asal sayi alinacak
# TabanAdedi        : Taban ("base") olarak alinacak sayi adedi
##########################################################
MilRab := function (Sayi, Yontem, TabanAdedi)
local b, j, m, flag, bolen, a, z, SayacAna;
# initialize
bolen := [2 .. 100000];
```

```
SayacAna :=1;

# negatifse pozitif sayiya cevir
if Sayi<0 then Sayi:=-Sayi; fi;

# cift bir sayi ise asal degildir
#if (IsEvenInt(Sayi)=true) then
#return false; ; # yani yanlis sonuc
#fi;

# Sayinin ikiye bolunme sayisi olan b hesaplanır
b := 0; m := Sayi -1;
while m mod 2 = 0 do
        b := b + 1;
        m := m / 2;
od;

while SayacAna <= TabanAdedi do
        j :=0 ; flag := 0;
        if Yontem = 0 then
                a := Random ( bolen );
                a := a mod Sayi;                        # sayidan ufak olmali
        else
                # ilk asallar "base" olarak alinabilir
                a := XPrimes[SayacAna];
        fi;
        z := PowerModInt (a, m, Sayi);                  # a^m mod p
        if (z = 1) or (z = Sayi - 1) then flag := 1;fi;

        while flag = 0  do
         if (j > 0) and (z = 1) then return false; fi;
        j := j + 1;
        if (j < b) and  (z <> (Sayi -1)) then z := PowerModInt (z, 2, Sayi);
        elif z = (Sayi - 1) then break;                 # Asal olabilir
        fi;

        if (j = b ) and ( z <> Sayi -1) then return false; fi;      # Asal degil
        od ; # while flag dongusu sonu
SayacAna := SayacAna + 1;
od; # while sayacana - iteration dongusu sonu

return true;                                            # Asal olabilir

end; # Fonksiyon Sonu

############################################################
#                   ASAL_BUL FONKSİYONU
```

```
# Verilen sayıya en yakın asalı bulur
#############################################################
asal_bul := function(Sayi, Yontem, TabanAdedi)

local p, index, Kalan, length, sflag;
Kalan := [];
# diger bolmeler yerine "Sayi+2" icin bu array'den "kalan+2" degeri kullanilacak
sayac := 1;
asbolsayac := 1;
length :=  Length (XPrimes);   # Xprimes listesindeki bilinen asallarin sayisi

# negatifse pozitif sayiya cevir
if Sayi<0 then Sayi:=-Sayi; fi;

# cift bir Sayi ise asal degildir
if (IsEvenInt(Sayi)=true) then
        #Bir sonraki tek sayiyi al
        Sayi := Sayi + 1;
        sayac := sayac +1;
fi;

# Kayitli asalsa, asal oldugunu belirtilir
if Sayi in XPrimes then return Sayi; fi;
if Sayi in Primes2 then return Sayi; fi;

# Onceki sartlari gecmis ve 1000'den kucukse o zaman asal degildir
while Sayi<=1000  do
        Sayi := Sayi +2 ;                       # Bir sonraki sayiyi al
        sayac := sayac + 1;
        if Sayi in XPrimes then return Sayi; fi;
od;

# Ufak Sayılara Bölme Testi
index :=1;

for p in XPrimes do
        Kalan[index] := Sayi mod p;
        if Kalan[index] = 0 then            # Bu tam bolundugunu gosterir
                #Bölündü. Bir sonraki sayi ile asallik testlerine devam edilir.
                # İlk sayı, butun asallara bolunerek kalan[] array'i olusturulur
                while index < length do
                        index := index +1 ;
                        Kalan[index] := Sayi mod p;
                od;
                Sayi := Sayi +2;                #Bir sonraki sayi alinir
                index := 1;
                sayac := sayac + 1;             #Sayac da arttirilir
```

```
                asbolsayac := asbolsayac + 1; #Asallara bolunen sayi sayaci
                break;                        # Loop'dan cikilir
        fi;

        index := index + 1 ;                  # Asal sayiya bolunmediyse artilir
        od;

  sflag := 0;

 while sflag = 0 do

        # index degiskenine bakarak loop'tan break ile cikip cikmadigini anlasilir
        if index > 1 then             #Bolme testini gecmis
        else                          #Yeni Sayi
        # 2. Kayitli asallara bölünüp bölünmediginin testi yapiliyor
        while index <= length do
                # Bolmeden sadece basit bir toplamayla halledebiliyoruz
                Kalan[index] := (Kalan[index] + 2) mod XPrimes[index];
                if Kalan[index] = 0 then   # Bu tam bolundugunu gosterir
                        while index < length do
                        index :=index +1 ;
                        # kalanlari da ikiser arttir
                        Kalan[index] := (Kalan [index] + 2) mod XPrimes[index];
                        od;
                        Sayi := Sayi +2;                    # Bir sonraki sayi alinir
                        sayac := sayac + 1;                 #sayac'i da arttirilir
                        asbolsayac := asbolsayac + 1; # asallara bolunen sayi sayac
                        index := 1;
                else  index := index + 1;
                fi;
        od; # end of while

        fi;

        # bolunme testlerini gecen sayi icin Miller&Rabin ve Lucas testlerini uygula
        sflag :=1;
        if (MilRab (Sayi, Yontem, TabanAdedi) = false ) then
                Sayi:= Sayi + 2; sayac := sayac + 1; index :=1;
                sflag := 0;
        else    if (Lucas (Sayi) = false ) then
                        Sayi:= Sayi + 2; sayac := sayac +1;
                        sflag := 0; index := 1;
                else    # Testi basariyla gecti
                fi;
        fi;
 od; # end of while sflag = 0
#Print(Sayi, " Olası asal sayidir ", sayac, ". denemede bulundu\n");
```

```
#Print(asbolsayac, " tane sayi, ufak asal sayilara bolundu\n");
return Sayi;
end;
#############################################################
#          ASAL_MI  FONKSIYONU
# Verilen sayinin asal olup olmadigini kontrol eder.
#############################################################
asal_mi := function(Sayi, Yontem, TabanAdedi)

        local p, Kalan, length;
        length :=  Length (XPrimes);   #Xprimes listesindeki bilinen asallarin sayisi

        # negatifse pozitif sayiya cevir
        if Sayi<0 then Sayi:=-Sayi; fi;
        # cift bir sayi ise asal degildir
        if (IsEvenInt(Sayi)=true) then return false;  fi;
        # Kayitli asalsa, asal oldugunu belirt
        if Sayi in XPrimes then return(true); fi;
        if Sayi in Primes2 then return(true); fi;

        # onceki sartlari gecmis ve 1000'den kucukse o zaman asal degildir
        if Sayi<=1000   then return false; fi;

        # Bilinen ufak asal sayılara bölerek asallık testi yap
        for p in XPrimes do
                Kalan := Sayi mod p;
                if Kalan = 0 then           # Bu tam bolundugunu gosterir
                        Sayac_UfakBol := Sayac_UfakBol + 1;
                        return false;
                fi;
        od;

        #bolunme testlerini gecen sayi icin Lehmann ve Miller testlerini uygula
        if (MilRab (Sayi, Yontem, TabanAdedi) = false ) then
                Sayac_MilRab := Sayac_MilRab + 1;
                return false;
        else if (Lucas (Sayi) = false ) then
                        Sayac_Lucas := Sayac_Lucas + 1;
                        return false;
                else    return true;
                fi;
        fi;
end;
# Program Kodu Sonu
```

Ek 10 IsPrimeInt() Kodu

```
##############################################################################
#F  IsPrimeInt( <n> ) . . . . . . . . . . . . . . . . . . . test for a prime
##      IsPrimeInt' does trial divisions by the primes less  than 1000 to detect composites
## with a factor less than 1000 and  primes  less  than  1000000.
##      `IsPrimeInt' then checks that $n$ is a strong pseudoprime to the  base 2.
## This uses Fermats theorem which says $2^{n-1}=1$ mod $n$ for a prime $n$.
## If $2^{n-1}\<>1$ mod $n$, $n$ is composite, `IsPrimeInt' returns `false'.
## There are composite numbers for which $2^{n-1}=1$,  but they are  seldom.
##      Then `IsPrimeInt' checks that $n$ is a Lucas pseudoprime for $p$, choosen so
## that the discriminant $d=p^2/4-1$ is an  quadratic nonresidue mod $n$.
## I.e., `IsPrimeInt' takes the root $a = p/2+\sqrt{d}$ of $x^2 - px + 1$ in
## the  ring $Z_n[\sqrt{d}] and computes the  traces of $a^n$ and $a^{n+1}$.
## If $n$ is a prime, this  ring is the field of  order $n^2$ and raising to
## the $n$th power is conjugation, so $trace(a^n)=p$ and $trace(a^{n+1})=2$.
## However, these identities hold only for extremly few composite numbers.
##      Note that  this  test  for $trace(a^n) = p$  and  $trace(a^{n+1}) = 2$ is
## usually formulated using the Lucas sequences  $U_k = (a^k-b^k)/(a-b)$ and
## $V_k = (a^k+b^k)=trace(a^k)$, where one tests $U_{n+1} = 0, V_{n+1} = 2$.
## However, the trace test is equivalent and requires fewer multiplications.
## Thanks to Daniel R. Grayson (dan@symcom.math.uiuc.edu)  for  telling  me.
##      `IsPrimeInt' can be shown to return the correct answer for $n < 10^{13}$,
## by testing against R.G.E. Pinch's list of all pseudoprimes to base 2 less
## than $10^{13}$ ('ftp://dpmms.cam.ac.uk/pub/rgep/PSP/psp13.gz').
##      Better descriptions of the algorithm and related topics can be found  in:
## G. Miller, cf. Algorithms and Complexity ed. Traub, AcademPr, 1976, 35-36
## C. Pomerance et.al., Pseudoprimes to 25*10^9, MathComp 35 1980, 1003-1026
## D. Knuth, Seminumerical Algorithms  (TACP II),  AddiWesl,  1973,  378-380
## G. Gonnet, Heuristic Primality Testing, Maple Newsletter 4,  1989,  36-38
```

```
## R. Baillie, S. Wagstaff, Lucas Pseudoprimes, MathComp 35 1980, 1391-1417
## R. Pinch, Some Primality Testing Algorithms, Notic. AMS 9 1993, 1203-1210

TraceModQF := function ( p, k, n )
  local trc;
  if k = 1 then
    trc := [ p, 2 ];
  elif k mod 2 = 0 then
    trc := TraceModQF( p, k/2, n );
    trc := [ (trc[1]^2 - 2) mod n, (trc[1]*trc[2] - p) mod n ];
  else
    trc := TraceModQF( p, (k+1)/2, n );
    trc := [ (trc[1]*trc[2] - p) mod n, (trc[2]^2 - 2) mod n ];
  fi;
  return trc;
end;
MakeReadOnlyGlobal( "TraceModQF" );

InstallGlobalFunction( IsProbablyPrimeInt, function( n )
  local p, e, o, x, i;

  # make $n$ positive and handle trivial cases
  if n < 0      then n := -n;     fi;
  if n in Primes   then return true;  fi;
  if n in Primes2  then return true;  fi;
  if n <= 1000     then return false; fi;

  # do trial divisions by the primes less than 1000
  # faster than anything fancier because $n$ mod <small int> is very fast
  for p in Primes do
```

```
    if n mod p = 0  then return false;  fi;
    if n < (p+1)^2  then AddSet( Primes2, n );  return true;   fi;
od;

# do trial division by the other known primes
for p  in Primes2  do
    if n mod p = 0  then return false;  fi;
od;

# find $e$ and $o$ odd such that $n-1 = 2^e * o$
e := 0;  o := n-1;   while o mod 2 = 0  do e := e+1;  o := o/2;  od;

# look at the seq $2^o, 2^{2 o}, 2^{4 o}, .., 2^{2^e o}=2^{n-1}$
x := PowerModInt( 2, o, n );
i := 0;
while i < e  and x <> 1  and x <> n-1  do
    x := x * x mod n;
    i := i + 1;
od;


# if it is not of the form $.., -1, 1, 1, ..$ then $n$ is composite
if not (x = n-1 or (i = 0 and x = 1))  then
    return false;
fi;

# there are no strong pseudo-primes to base 2 smaller than 2047
if n < 2047  then
    AddSet( Primes2, n );
    return true;
```

```
  fi;

  # make sure that $n$ is not a perfect power (especially not a square)
  if SmallestRootInt(n) < n  then
    return false;
  fi;

  # find a quadratic nonresidue $d = p^2/4-1$ mod $n$
  p := 2;  while Jacobi( p^2-4, n ) <> -1  do p := p+1;  od;

  # for a prime $n$ the trace of $(p/2+\sqrt{d})^n$ must be $p$
  # and the trace of $(p/2+\sqrt{d})^{n+1}$ must be 2
  if TraceModQF( p, n+1, n ) = [ 2, p ]  then
    return true;
  fi;

  # $n$ is not a prime
  return false;
end);

InstallGlobalFunction(IsPrimeInt,function ( n )
local p;
 p:=IsProbablyPrimeInt(n);
 if p then
  if n>10^13 then
   Info(InfoWarning,1,
       "beyond the guaranteed bound of the probabilistic primality test");
  fi;
  if 1000 < n then
   AddSet( Primes2, n );
```

Ek 11 100 Asal Sayı Bulma Testinde Kullanılan Sayılar

Örneklem1:

11 Basamak (> 32 Bit)

16002900809

21 Basamak (> 64 Bit)

175953755186215647081

40 Basamak (> 128 Bit)

1329468985642297249959579938394202368665

79 Basamak (> 256 Bit)

1214388187060545058916783247482827552379791480743695665159592941311452430875695

156 Basamak (> 512 Bit)

633280621127153645333258726392839709025266932897531339618298168313680919159322852156458965283026287519687577836653555297186804166434598168368167798087118571

310 Basamak (> 1024 Bit)

172216011773558752989262249174653191530943033118069075120456338537360790089004859944156400387372686100551076461067428105910665798184351809961945331576572767380959962972247927066114876274765710811373834016692051912092054911026587172161662145187948288724377747974813854852552219799788200188309271656536795298768 5

618 Basamak (> 2048 Bit)

104283021521421050503374829549741760738615826671145157658785342334606111383350486221681522144129526709720466080661912441769519913018539172942140176960247600395899689204339821888985531766855487188893945513434449725807475411909290701430219166176948868232009641029281036543094935719765010930293542967067814284558326168989513082959185077861510111994318893744557066950064467329886656580521052573949836805663880216912740797892150924477725361388138289455930393932127208407492164614120967158616295134584614661997889799177311430438461480462727216804829468544610041713866313917085600553270096090369906079555939523702715630600369

Örneklem2

11 Basamak (> 32 Bit)

57215777917

21 Basamak (> 64 Bit)

629094131111692619851

40 Basamak (> 128 Bit)

4753300862930974922171168022313421077385

79 Basamak (> 256 Bit)

4341848121187508495876914115162899484858497243879854298948080627891666300680839

156 Basamak (> 512 Bit)

226419221161964811088254349240007561878160802347433251206719643146819143180884879670460267443705361782618345229791454779315323843380560416114086867182672928

310 Basamak (> 1024 Bit)

615730435395080534487495481453374131329464546353460692099231398455338034334983548154688272554827626402650868822990720342590808524490038035977524812812820670742691461835201298218053842862259491913644953269944652472985330868453640227657151831032816292255775576648713868342561433842025244988802283260462211791648 9

618 Basamak (> 2048 Bit)

455349935306741325299032860165166778926473889796475582506502864919735559738173985514311317740307387935027556871749379774838243119126106960282827592166958184525727970462007226036847968027231002969796504806600739160863302253608054143329048628078610685398469441499802577220597414881490785988371053716492449962705275731272282377252174263176536756362017556517069434526655180376714219577831338261789718020506063738952918632567926404206262247187189865866091824385152980404906701858922672119791055379364943906121925782691008837319721489567474791864726496288890791807102547795504339195765724148144857638047199548551772667134999

Ek 12 Carmichael Sayılarını Test Eden Kod

Bu kod kullanılan bütün asallık testleri için uygulanmaktadır. Sadece asallık testinin çağrıldığı satır her test için değişiktir. Test çağrılmadan önce ve sona erdikten sonra zaman ölçümleri alınıp değerlendirilmekte ve ortalama değerler hesaplanmaktadır.

```
# Enis Karaaslan - Tez Calismasi
# Carmichael Sayilari test eden fonksiyon -  14.06.2000

test := function(DosyaAd, Yontem, TabanAdedi)
local l, s, car, sl, sayi, count, tmin, tmax, ttop, SayiAdedi, olcflag,t, t2, t1, d;
count := 0; olcflag := 0; SayiAdedi := 0; ttop := 0;

if IsExistingFile(DosyaAd) = false
then Print("Dosya Yok. islem sonu \n");
else Print ("Dosya var. islem yapilabilir \n");

s := InputTextFile(DosyaAd);
while not IsEndOfStream(s) do
        SayiAdedi := SayiAdedi + 1;
        l:=ReadLine(s);
        if l=fail then      Print ("Dosya sonu. En son okunan deger:", sayi,"\n");
                            Print (count, "adet asal sayi bulundugu sonucu dondu\n");
                           break;
                           fi;
        sl := SplitString(l, " \n"," \n");
      sayi := Int(sl[1]);
      t1 := Runtime();
      # Asallik testinin cinsinin belirtildigi nokta
```

```
        if (asal_mi(sayi, Yontem, TabanAdedi) = true) then count := count + 1; fi;
        t2 := Runtime();
        t := t2 -t1;

        if (olcflag = 0) then
                tmin := t;
                tmax := t;
                olcflag := 1;
          else
                if t > tmax then tmax := t;
                elif t < tmin then tmin := t;
                fi; # innermost if
          fi;
        ttop := ttop + t;
od;

Print ("Bu islem minimum : ", tmin , " milisecond cpu time surmustur.\n");
Print ("Bu islem maximum : ", tmax , " milisecond cpu time surmustur.\n");
d := ttop / SayiAdedi;
Print ("Bu islem ortalama : ", d , " milisecond cpu time surmustur.\n");

fi; # end of file operation
end; # end of test function
```

ÖZGEÇMİŞ

Soyadı :KARAASLAN

Adı :Enis

Doğum Tarihi :17.4.1976

Doğum Yeri :İzmir

Uyruk :TC

Adres :Ege Üniversitesi, Uluslararası Bilgisayar Enstitüsü, Network Yönetim Grubu, Bornova, İZMİR

E-mail :enis@bornova.ege.edu.tr

Ev Telefon :0232 2471831

İş Telefon :0232 3423232 / 227

İş Fax :0232 3887230

Yabancı Dil :İngilizce, Almanca

Eğitim :

1998- Yüksek Lisans, Ege Üniversitesi, UBE Bilgisayar Ağ Teknolojileri Bilim Dalı

1994-1998 Lisans, Ege Üniversitesi, Bilgisayar Mühendisliği Bölümü

1991-1994 Lise, İzmir Atatürk Lisesi