T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
KRİPTOLOJİYE GİRİŞ

ÖDEV
EAP Exchange

Hazırlayan:
ESRA KIZILELMA 1-A B191210040

SAKARYA
Aralık 2023

İçindekiler

SAKARYA
Aralık 2023

Mevcut Şifreleme Algoritmalarının Analizi

Çalışma sonucunda elde edilen verilere göre küçük boyuttaki verilerin şifrelenmesi işleminde DES algoritması AES algoritmasından daha performanslı bir şekilde çalışmaktadır. Şifrelenecek verinin boyutu 1MB ve üzeri olduğu durumlarda AES şifreleme algoritması DES şifreleme algoritmasından daha performanslı bir şekilde çalışmaktadır. 1 MB 'tan küçük verilerin şifrelenmesi işlemi sonucunda elde edilen sonuçlara göre algoritmaların performans sıralamasının büyükten küçüğe doğru Blowfish, Twofish, TEA, IDEA, DES, AES, RC2, 3DES, RSA algoritmaları şeklinde olduğunu gösterir (Tablo 4.2- 4.9). Simetrik şifreleme algoritmaları asimetrik şifreleme algoritmalarına göre çok daha performanslı bir şekilde çalışmaktadır. Çalışmada algoritmalar farklı donanım özelliklerine (AMD, Intel işlemci) ve farklı işletim sistemlerine sahip bilgisayarlarda test edilmiştir.

Kullanılacak algoritmalar seçilirken bilgisayar özellikleri de göz önünde bulundurulmalıdır. Özellikle AMD işlemciye sahip bilgisayarlarda asimetrik şifreleme algoritması olan RSA, diğer simetrik şifreleme algoritmaları olan Blowfish, 96 Twofish, TEA, IDEA, DES, AES, 3DES ve RC2'ye göre çok daha yavaş çalışmaktadır. Şifrelenecek verinin boyutu 1 MB'tan fazla olması durumunda elde edilen sonuçlara göre algoritmaların performans sıralamasının büyükten küçüğe doğru Blowfish, Twofish, TEA, AES, IDEA, DES, RC2, 3DES, RSA algoritmaları şeklinde olduğunu gösterir (Tablo 4.2 - 4.9). Metinler şifrelendikten sonra şifreli hallerinde boyut olarak bir artış olmaktadır. Metinlerin şifreli halleri daha fazla karakter içermektedir. Burada gösterilen performans sonuçlarının algoritmaların güvenilirliğiyle bir ilgisi bulunmamaktadır. [1]

1. AES (Advanced Encryption Standard):

Avantajlar:

- Yüksek güvenlik seviyesi.
- Geniş kabul görmüş bir standart.
- Hızlı ve etkili bir performansa sahiptir.

Zayıflıklar:

- Doğru bir şekilde uygulandığında genellikle güvenlidir, ancak yan kanal saldırıları gibi spesifik senaryolarda zayıf olabilir.

2. RSA (Rivest-Shamir-Adleman):

- Avantajlar:
  - Asimetrik şifreleme için popüler bir seçenek.
  - Dijital imzalama ve anahtar değişimi için kullanılır.

- Zayıflıklar:

- Güvenliği, kullanılan anahtar uzunluğuna bağlıdır. Kısa anahtarlar zamanla kırılabilir hale gelebilir.

3. ECC (Elliptic Curve Cryptography):

- Avantajlar:

  - Asimetrik şifreleme için diğer bir popüler seçenek.

  - Daha küçük anahtar boyutlarıyla yüksek güvenlik sağlar.

- Zayıflıklar:

  - Belirli eğriler veya parametrelerin seçimi hataları güvenliği etkileyebilir.

4. SHA-256 (Secure Hash Algorithm):

- Avantajlar:

  - Hızlı ve güvenilir bir kriptografik hash fonksiyonu.

  - Geniş bir uygulama yelpazesi için kullanılır.

- Zayıflıklar:

  - Teorik olarak, çarpışma saldırıları gibi belirli senaryolarda zayıf olabilir.

5. MD5 (Message Digest Algorithm 5):

- Zayıflıklar:

  - Kırılabilir hale gelmiş ve çarpışma saldırılarına karşı savunmasızdır. Güvenlik uygulamalarında artık önerilmez.

Not: Güvenlik teknolojileri sürekli olarak evrimleşir ve şifreleme algoritmalarının değerlendirmesi zaman içinde değişebilir. Güvenlikle ilgili güncel bilgileri ve önerileri takip etmek önemlidir. Güvenlikle ilgili profesyonel rehberlik almak, özellikle şifreleme ve kriptografi konusunda uzmanlaşmış kişilerle iş birliği yapmak her zaman önerilir.

Blowfish, Bruce Schneier tarafından 1993 yılında tasarlanan simetrik bir blok şifreleme algoritmasıdır. İşte Blowfish'in avantajları ve zayıflıkları:

Avantajları:

Hızlı İşlem: Blowfish, genellikle hızlı bir şifreleme algoritması olarak kabul edilir. Özellikle donanım kaynakları sınırlı olan ortamlarda tercih edilebilir.

Anahtar Uzunluğu Esnekliği: Blowfish, 32 bit ila 448 bit arasında değişen anahtar uzunluklarıyla çalışabilir. Bu, kullanıcılara güvenlik seviyelerini ihtiyaçlarına göre özelleştirme esnekliği sağlar.

Patent Sorunu Olmaması: Blowfish, patent sorunları olmadan geniş bir şekilde kullanılabilir. Algoritmanın genelde açık kaynaklı uygulamalarda ve yazılımlarda kullanılması bu özelliği destekler.

Zayıflıkları:

Diğer Alternatiflerin Popülerliği: Blowfish, Twofish ve AES gibi diğer şifreleme algoritmalarının popülerleşmesiyle birlikte gölgede kalmıştır. Bu nedenle, genellikle yeni projelerde veya güvenlik gereksinimleri yüksek olan uygulamalarda tercih edilmemektedir.

Bloklama Bağımlılığı: Blowfish, 64 bitlik bloklarla çalışır ve bu, bazı uygulamalarda dezavantaj olarak görülebilir. Özellikle, küçük paketlerle çalışan iletişim protokollerinde bu durum performans sorunlarına yol açabilir.

Bilinen Anahtar Bağımlılıkları: Bazı kriptoanalistler, Blowfish'in belirli anahtar bağımlılıkları nedeniyle güvenlik açısından daha zayıf olabileceğine dair endişelerini dile getirmişlerdir. Ancak, bu tür endişeler her zaman pratik saldırılara dönüşmemiştir.

Sonuç olarak, Blowfish hala bazı uygulamalarda kullanılabilen güvenilir bir şifreleme algoritmasıdır, ancak daha modern alternatiflerle karşılaştırıldığında popülerliği azalmıştır. Güvenlik gereksinimleri ve performans önceliklerine bağlı olarak, bir şifreleme algoritması seçerken dikkate alınabilir.

Şifreleme Algoritması Yazarken Kullanılan Dilin Etkisi

Şifreleme algoritması yazmak ciddi bir güvenlik sorumluluğu gerektirir ve bu alanda derin bilgi ve uzmanlık gerektirir. Ayrıca, güvenliği sağlamak ve algoritmanın doğru çalıştığından emin olmak için matematiksel analiz ve güvenlik testleri yapılması önemlidir. Bu tür bir projeye başlamadan önce kriptografi konusunda yeterli bilgiye sahip olduğunuzdan emin olmalısınız.

Ancak, eğer yine de şifreleme algoritması yazmaya kararlıysanız, genellikle güvenli ve performanslı kod yazmada rahat olan bir dil seçmelisiniz. İşte bu tür bir projede kullanabileceğiniz bazı diller:

1.C/C++:

- Performans önemliyse ve düşük seviye kontrol gerekiyorsa, C veya C++ tercih edilebilir. Bu diller, doğrudan bellek kontrolü ve yüksek performans sağlar.

2. Python:

- Prototip oluşturmak veya daha yüksek seviyede bir dilde çalışmak istiyorsanız Python kullanabilirsiniz. Ancak, performans konusunda C/C++ kadar etkili olmayabilir.

3. Java:

- Platform bağımsız olması ve nesne yönelimli programlama sağlaması açısından Java tercih edilebilir. Ancak, performans bakımından C/C++ kadar olmayabilir.

4. Rust:

- Yeni bir dil olan Rust, bellek güvenliği konusunda önemli avantajlar sağlar ve C/C++'a benzer performansa sahiptir. Güvenlik odaklı projelerde kullanılabilir.

5. Go:

- Go, hızlı geliştirme ve performans açısından dengeli bir dil olarak bilinir. Temiz ve basit bir syntaxa sahiptir.

Unutmayın ki, kriptografi hataları ciddi güvenlik açıklarına yol açabilir ve bu nedenle kriptografik algoritmaların tasarımı ve uygulanması konusunda özenle çalışılmalıdır. Ayrıca, güvenlik uzmanları ve kriptografi uzmanları tarafından değerlendirilmelidir. Önceki cevapta belirtildiği gibi, bu tür bir projede profesyonel rehberlik almak önemlidir.

SAKARYA
Aralık 2023

Blok Şifreleme Algoritması Yapısı

Bloğa dayalı şifreleme algoritmaları, veriyi belirli boyutlardaki bloklara bölen ve ardından her bloğu bağımsız olarak şifreleyen algoritmaları ifade eder. Bu tür bir algoritma tasarlamak karmaşık bir süreçtir ve güvenlik açısından dikkatlice düşünülmesi gereken birçok faktör bulunmaktadır. İşte temel adımlar ve prensipler:

1. Blokların Bölünmesi:

- Veri, belirli bir boyuttaki bloklara bölünmelidir. Örneğin, 128 bit boyutundaki bloklar sıkça kullanılır.

2. Anahtar Yönetimi:

- Blok şifreleme algoritmaları genellikle bir anahtar ile çalışır. Güvenlik, bu anahtarın nasıl yönetildiği ve kullanıldığına bağlıdır. Anahtarlar güçlü ve rastgele olmalıdır.

3. Alt Anahtar Üretimi:

- Genellikle anahtar, alt anahtarlar olarak adlandırılan daha küçük parçalara bölünür. Bu alt anahtarlar, şifreleme ve şifre çözme işlemleri sırasında kullanılır.

4. Permutasyon ve Substitution (Yerine Koyma):

- Bloklar üzerinde permutasyon (yer değiştirme) ve substitution (yerine koyma) operasyonları gerçekleştirilir. Bu adımlar, şifreleme işleminin temelini oluşturur.

5. Iterative (Yinelemeli) İşlemler:

- Blok şifreleme algoritmaları genellikle belirli sayıda yinelemeli işlem içerir. Bu, güvenliği artırmak ve şifreleme işlemini daha karmaşık hale getirmek amacıyla yapılır.

6. Diffusion ve Confusion:

- Diffusion, giriş verisinin tüm bloklara eşit olarak dağılmasını ifade eder. Confusion ise şifreleme işleminin anahtarın karmaşıklığını artırarak gizemini korumasını sağlar.

7. Performans Ölçümleri:

- Tasarladığınız algoritmanın performansını değerlendirmek için çeşitli metrikler kullanabilirsiniz. Bu metrikler arasında şifreleme ve şifre çözme süreleri, anahtar uzunluğu, saldırılara karşı direnç ve kaynak kullanımı bulunabilir.

8. Matematiksel Analiz:

- Algoritmanızın matematiksel dayanakları ve güvenlik özellikleri üzerine bir analiz yapmalısınız. Bu, potansiyel saldırılara karşı ne kadar dayanıklı olduğunu anlamanıza yardımcı olacaktır.

9. Standartlara Uygunluk:

- Tasarladığınız algoritmanın, endüstri standartlarına uygun olup olmadığını değerlendirmelisiniz. FIPS (Federal Information Processing Standards) gibi standartlar, şifreleme algoritmalarının güvenliği ve etkinliği konusunda belirli gereksinimleri içerir.

Bu adımları takip ederek, mevcut literatürden esinlenerek güçlü ve güvenli bir blok şifreleme algoritması tasarlamaya çalışabilirsiniz. Ancak, gerçek bir şifreleme algoritması tasarlama süreci oldukça karmaşıktır ve bu alanda uzmanlık gerektirir. Bu nedenle, profesyonel bir kriptografi uzmanının gözetimi altında çalışmanız önerilir.

Güçlü Blok Şifreleme Algoritmalarının Güvenlik Açısından Değerlendirilmesi

Günümüzde blok şifreleme algoritmaları, şifrelemenin gerektiği birçok alanda kullanılmaktadırlar. Dolayısıyla bu algoritmaların gücüde güvenlik açısından çok önemlidir. Blok şifreleme algoritmalarının gücü, anahtar uzunluğuna, yapılan saldırılara karşı dayanıklılığına bağlıdır. Bunun yanında saldırıların başarılı sayılabilmesi için geniş anahtar arama saldırısı da bir kıstas olarak kullanılmaktadır. Yani anahtar arama saldırısından daha az maliyete mal olan saldırlar başarılı sayılmaktadır. Dolayısıyla algoritmanın tasarımında kullanılan anahtar yönetimi, yani bir ana anahtardan döngülere giriş olan anahtarlar elde etme yöntemi, S kutuları ve döngü sayısı algoritmanın yapılan saldırılara dayanıklılığını da etkilemektedir. Buna ek olarak algoritmaya yapılan saldırı da kullanılacak açık metin/şifreli metinlerin sayısı da, birçok gelişmiş saldırı yöntemleri bu verileri gerektirmekte, algoritmanın gücünü ortaya koymaktadır. Her ne kadar yukarıda bahsedilen teknikler geniş anahtar arama saldırısından daha etkili olsa da daha az açık metnin ve şifreli metnin kullanıldığı saldırı yöntemleri geliştirmek gereklidir. AES algoritmasında olduğu gibi yeni saldırı tipleri demek yeni algoritmalarda bu saldırı tiplerinin dikkate alındığı daha güçlü şifreleme algoritmaları demektir.[2]

Şifreleme algoritmalarının tasarımında ve güvenli uygulamalarında temel prensipleri belirlemek önemlidir. İşte bu prensipler ve şifreleme algoritmalarına olan etkileri:

1. Simetrik ve Asimetrik Şifreleme:

- Etki: Simetrik şifreleme genellikle daha hızlıdır ve düşük kaynak tüketimine sahiptir. Asimetrik şifreleme ise genellikle anahtar yönetimini kolaylaştırır ve güvenli anahtar değişimi sağlar.

2. Akış Şifreleme ve Blok Şifreleme:

- Etki: Akış şifreleme, sürekli bir veri akışını şifrelemek için kullanılır ve genellikle hızlıdır. Blok şifreleme, belirli boyutlardaki blokları ayrı ayrı şifreler ve genellikle daha karmaşık bir anahtar yönetimi gerektirir.

3. Anahtar Yönetimi:

- Etki: Güvenli anahtar yönetimi, algoritmanın güvenliğini doğrudan etkiler. Anahtarların oluşturulması, değiştirilmesi ve güvenli bir şekilde saklanması, şifreleme sisteminin dayanıklılığını artırır.

4. Blokların Bölünmesi ve Döngü Sayısı:

- Etki: Blok boyutu ve döngü sayısı, algoritmanın güvenliği ve performansı üzerinde etkilidir. Daha büyük bloklar ve daha fazla döngü genellikle daha güvenli ancak daha yavaş işlemeye neden olabilir.

5. S Kutuları ve P Kutuları:

- Etki: S kutuları ve P kutuları, şifreleme algoritmalarının dayanıklılığını artırarak saldırılara karşı direncini güçlendirir. Bu kutuların tasarımı, algoritmanın güvenliğini büyük ölçüde belirler.

6. Anahtar ve Blok Boyutları:

- Etki: Daha uzun anahtarlar ve büyük bloklar, genellikle daha güvenli ancak daha yavaş şifreleme sağlar. Ancak, performans ve güvenlik arasında bir denge bulunması önemlidir.

Bu prensipler, bir şifreleme algoritmasının tasarımında ve güvenli uygulamasında dikkate alınmalıdır. Her şifreleme algoritması farklı ihtiyaçları karşılamak üzere tasarlanır, bu nedenle kullanım senaryonuza ve güvenlik gereksinimlerinize uygun bir algoritma seçimi önemlidir.

Şifreleme algoritmalarını tasarlarken ve değerlendirirken, uzmanlardan ve kriptografi topluluğundan gelen güncel bilgileri de takip etmek önemlidir.

Performans Analizi

Python projelerinizde performans analizi yapmak için Visual Studio Code'da kullanabileceğiniz bazı eklentiler şunlardır:

Pylint: Pylint, Python kodunuzdaki hataları, stil ihlallerini ve performans sorunlarını kontrol etmek için kullanılır. Bu eklenti, kod kalitesini artırmak ve olası sorunları belirlemek için oldukça kullanışlıdır.

Pyright: Pyright, Python için bir statik analiz aracıdır. Kodunuzu taramak ve hızlı bir şekilde hataları ve performans sorunlarını belirlemek için kullanılabilir. Aynı zamanda güçlü bir otomatik tamamlama özelliği sunar.

Bandit: Bandit, Python projelerinizde güvenlik açıklarını kontrol etmek için kullanılır. Projenizin güvenliği ile ilgili potansiyel sorunları belirlemek amacıyla tasarlanmıştır.

Py-spy: Py-spy, Python uygulamalarınızın çalışma anındaki performansını izlemek için kullanılır. Profil oluşturarak, uygulamanızın hangi bölümlerinin daha fazla zaman aldığını belirleyebilirsiniz.

Memory Profiler: Memory Profiler, Python uygulamalarınızın bellek kullanımını izlemek ve analiz etmek için kullanılır. Uygulamanızın bellek tüketimini belirleyerek olası sızıntıları tespit etmeye yardımcı olabilir.

SnakeViz: SnakeViz, Python projelerinizde oluşturulan cProfile veya profile çıktılarını görselleştirmek için kullanılır. Bu, uygulamanızın performansını daha ayrıntılı bir şekilde incelemenize yardımcı olabilir.

timeit: Python'un standart kütüphanesinde bulunan timeit modülü, belirli kod bloklarının çalışma süresini ölçmek için kullanılır. Bu modülü doğrudan kullanarak, kodunuzun belirli bölümlerinin ne kadar süre aldığını ölçebilirsiniz.

Bellek kullanımı

Belllek kullanamı şifrelenen verinin veya anahtrarın boyutundan bağımsız belirli bir aralıkta

Algoritmanın performansını değerlendirmek için çeşitli testler ve performans metrikleri kullanılabilir. İşte genel olarak dikkate almanız gereken bazı testler ve metrikler:

1. Zaman (Hız) Performansı:

- Algoritmanın işlem süresini ölçmek için çeşitli giriş veri setleri kullanın.

- Başlangıç, orta ve son noktalarda farklı boyutlarda veri setleri üzerinde performansı test edin.

- En iyi durum, ortalama durum ve en kötü durum senaryolarını değerlendirin.

- İşlem sürelerini karşılaştırmak için farklı algoritmalarla kıyaslama yapın.

2. Bellek Tüketimi:

- Algoritmanın bellek kullanımını ölçün.

- Veri seti boyutlarına göre bellek tüketimini inceleyin.

- Bellek sızıntılarını kontrol etmek için uzun süreli çalışmalarda bellek kullanımını izleyin.

3. Doğruluk ve Başarı Oranları:

- Algoritmanın doğruluğunu ve başarı oranını belirlemek için doğruluk testleri yapın.

- Hata oranlarını, yanlış pozitif ve yanlış negatif oranları da dahil olmak üzere çeşitli metriklerle değerlendirin.

- Sınıflandırma problemleri için doğruluk, hassasiyet, özgüllük, F1 puanı gibi metrikleri kullanın.

4. Kararlılık ve Güvenilirlik:

- Algoritmanın farklı girişlere karşı kararlılığını test edin.

- Giriş verilerindeki küçük değişikliklere karşı algoritmanın nasıl tepki verdiğini değerlendirin.

5. Skalabilite:

- Algoritmanın veri seti boyutları arttıkça nasıl bir performans sergilediğini test edin.

- Veri seti boyutlarına göre işlem sürelerini ve bellek kullanımını ölçün.

6. Paralel İşleme:

- Algoritmanın paralel işleme yeteneklerini test edin, özellikle çok çekirdekli sistemlerde.

- Paralel işleme kullanılarak hız artışını değerlendirin.

7. Güvenlik Testleri:

- Algoritmanın güvenlik açısından zayıflıklarını belirlemek için güvenlik testleri yapın.
- Veri setlerine kötü niyetli girişler ekleyerek algoritmanın nasıl tepki verdiğini kontrol edin.

8. Kaynak Kullanımı:

- CPU, GPU, disk I/O gibi kaynakları izleyerek algoritmanın bu kaynakları nasıl kullandığını değerlendirin.

Bu testler ve metrikler, tasarlanan algoritmanın performansını kapsamlı bir şekilde değerlendirmenize yardımcı olabilir. Testlerinizi gerçek dünya senaryolarına yakın olacak şekilde düzenlemeye çalışın ve çeşitli senaryolarda algoritmanın nasıl davrandığını anlamak için farklı test durumlarını göz önünde bulundurun.

Eğer şifreleme süresi şifrelenecek verinin büyüklüğüne bağlı olarak değişiyorsa, bu durumu optimize etmek için aşağıdaki stratejileri düşünebilirsiniz:

Blokları Paralel İşleme: Büyük veri setleri üzerinde çalışırken, veriyi küçük bloklara bölüp bu blokları paralel olarak işleyebilirsiniz. Paralel işleme, işlem gücünden daha etkin bir şekilde yararlanmanıza olanak tanır.

Veriyi Ön İşleme: Veriyi şifreleme öncesinde önceden işleyerek (örneğin, sıkıştırma veya özetleme) veri boyutunu azaltabilirsiniz. Bu, şifreleme süresini azaltabilir ve daha hızlı bir işlem sağlayabilir.

Optimize Edilmiş Kütüphaneler: Kullandığınız şifreleme kütüphanesi veya algoritmasının optimize edilmiş bir sürümünü kullanmak performans artışına yol açabilir. Python'da, örneğin, C diline derlenmiş ve performans açısından optimize edilmiş kütüphaneleri kullanmayı düşünebilirsiniz.

Algoritma Değişiklikleri: Farklı şifreleme algoritmalarını değerlendirebilir ve performanslarına göre seçim yapabilirsiniz. Farklı algoritmalar, farklı veri türleri ve boyutları için daha iyi performans gösterebilir.

Donanım Hızlandırma: Donanım tabanlı şifreleme hızlandırma teknolojilerini kullanmayı düşünebilirsiniz. Örneğin, Intel'in AES-NI veya AMD'nin XOP (eXtended Operations) gibi donanım hızlandırma özellikleri, şifreleme işlemlerini hızlandırabilir.

Özel Optimizasyonlar: Blowfish algoritması için özel optimizasyonlar yapabilirsiniz. Bu, özellikle belirli bir uygulama veya senaryo için performansı artırabilir. Ancak, bu tür optimizasyonlar genellikle algoritmanın karmaşıklığını artırabilir.

Şifreleme süresini iyileştirmek için bu stratejileri uygularken, güvenlik gereksinimlerinizi ve algoritmanın güvenliğini dikkate almalısınız. Performansı artırmak için yapılan değişikliklerin güvenlik üzerinde olumsuz bir etkisi olmamalıdır.

"Kriptografik algoritma" terimi, genellikle güvenli ve gizli iletişim için kullanılan şifreleme, özetleme (hashing), dijital imza ve benzeri işlemleri gerçekleştiren matematiksel formülleri ifade eder. Kriptografik algoritmaların değerlendirilmesi ve seçilmesi ciddi bir konudur ve bir dizi kriteri içerir:

1. Matematiksel Güvenlik: Algoritmanın dayandığı matematiksel problemlerin zorluğu ve bu zorluğun kırılması için gereken hesaplama gücü göz önünde bulundurulmalıdır.

2. Anahtar Uzunluğu: Algoritmanın güvenliği genellikle kullanılan anahtar uzunluğuna bağlıdır. Daha uzun anahtarlar, genellikle daha güvenli şifreleme sağlar.

3. Dayanıklılık: Algoritmanın zaman içinde dayanıklılığı ve uzun vadeli güvenliği değerlendirilmelidir. Matematiksel keşifler veya teknolojik gelişmelerle algoritmanın kırılması mümkün olabilir mi?

4. Genel Kabul: Algoritmanın endüstri standardı olarak ne kadar kabul gördüğü önemlidir. Yaygın kullanılan ve güvenilen algoritmalar, genellikle daha fazla denetim ve değerlendirmeye tabi tutulmuşlardır.

5. Performans: Algoritmanın şifreleme ve çözme süreleri, bellek kullanımı ve genel sistem performansı göz önünde bulundurulmalıdır. Veri boyutları büyüdükçe veya daha fazla güvenlik gereksinimi olduğunda nasıl performans gösteriyor?

6. Widespread Kullanım ve Test: Algoritmanın geniş bir kullanıcı kitlesi tarafından kullanılıyor olması ve çeşitli uygulamalarda test edilmiş olması önemlidir. Bu, algoritmanın pratikteki performansını gösterir.

7. Düzeltme ve Güncelleme Kolaylığı: Algoritmanın güvenlik açıkları keşfedildiğinde veya ihtiyaç duyulduğunda güncellenip düzeltilebilir olması önemlidir.

8. Saldırıya Dayanıklılık: Algoritmanın çeşitli saldırılara karşı dayanıklılığı değerlendirilmelidir, örneğin, diferansiyal kriptoanaliz veya yan kanal saldırılarına karşı direnci.

9. Standartlar ve Sertifikasyon: Algoritmanın bir veya daha fazla kriptografik standart veya sertifikasyon kuruluşu tarafından onaylanmış olması güvenilirliğini artırabilir.

Bu kriterler, bir kriptografik algoritmanın güvenliği ve güvenilirliği hakkında bir genel bakış sağlar. Ancak, kullanım senaryonuza ve güvenlik gereksinimlerinize bağlı olarak daha spesifik kriterler de olabilir.

Performans Testleri Sonucu

```
Only 8 byte input !!
Enter data to encrypt: 1
Enter key: 1
Valid Input !!!
Encrypted data is:  12601061029754453122
Bellek Kullanımı (14234, 14480)
Encryption Time: 7.220029592514038 seconds
```

```
Only 8 byte input !!
Enter data to encrypt: 1
Enter key: a
Valid Input !!!
Encrypted data is:  16436292583247932543
Hex value : 0xe419710570e16c7f
Data after decryption is :  1
Bellek Kullanımı (14238, 14484)
Encryption Time: 3.3770008087158203 seconds
```

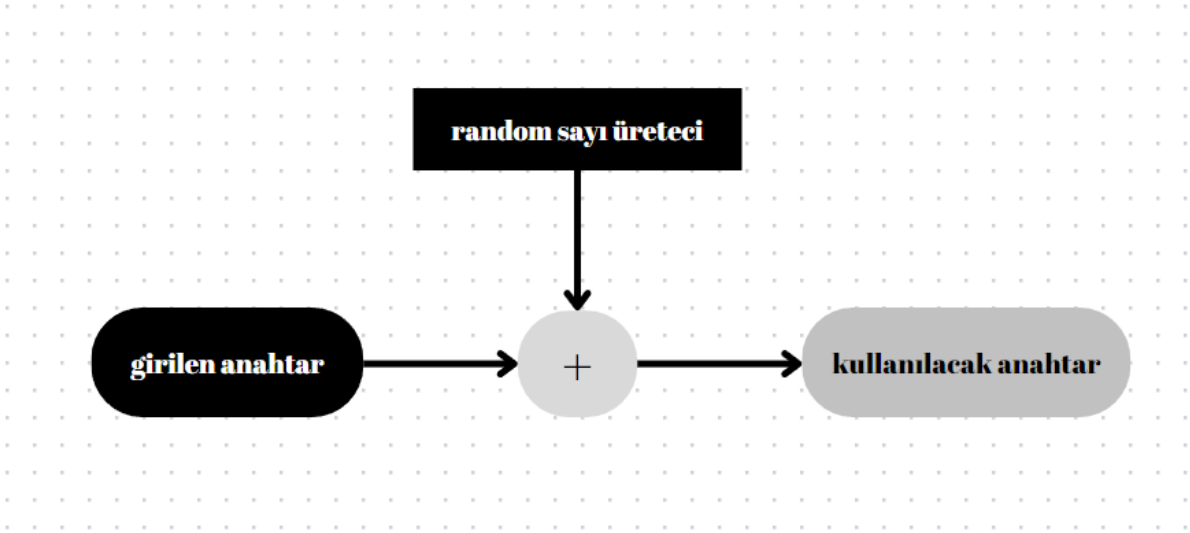
```
Enter data to encrypt: 1
Enter key: abcdefgh
Valid Input !!!
Encrypted data is:  18042252700849472797
Hex value : 0xfa62f53d4df7051d
Data after decryption is :  1
Bellek Kullanımı (14295, 14541)
Encryption Time: 7.993834972381592 seconds
```

```
Enter data to encrypt: 1
Valid Input !!!
Encrypted data is:  11884561811416897087
Bellek Kullanımı (14238, 14484)
Encryption Time: 3.694977283477783 seconds
```

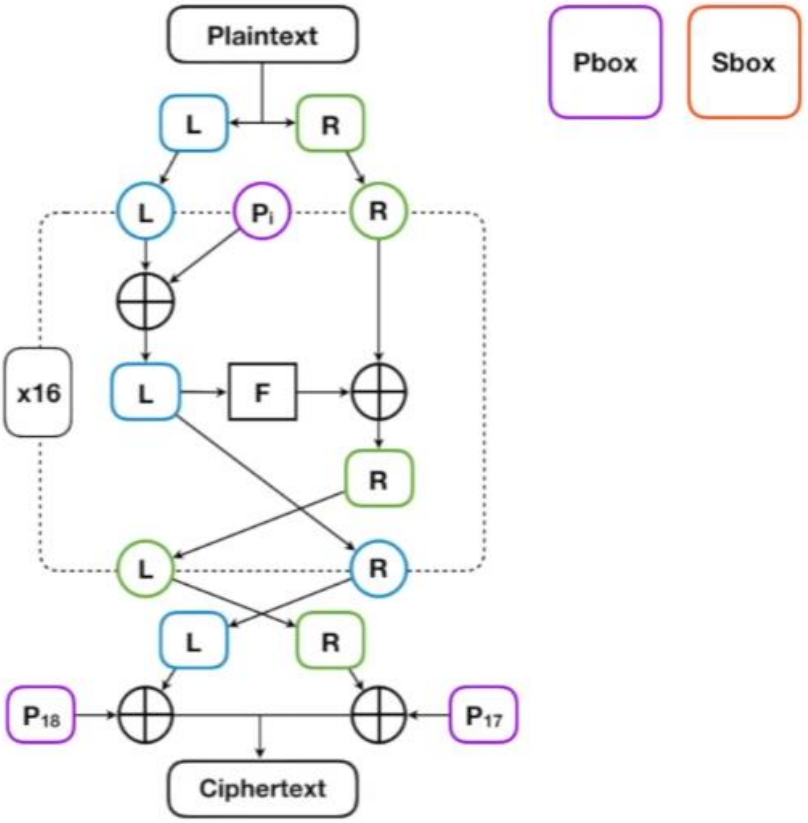
```
Only 8 byte input !!
Enter data to encrypt: asdfgh
Enter key: 123
Valid Input !!!
Encrypted data is:  10652867724802491482
Hex value : 0x93d696c12013845a
Data after decryption is :  asdfgh
Bellek Kullanımı (14436, 14682)
Encryption Time: 7.0855629444122314 seconds
PS C:\Users\kizil\Desktop>  c:; cd 'c:\Users\kizi
1\pythonFiles\lib\python\debugpy\adapter/../..\de
Only 8 byte input !!
Enter data to encrypt: çömnbgsa
Enter key: çömnha
Invalid Input!!
Bellek Kullanımı (13444, 13690)
Encryption Time: 6.211394309997559 seconds
PS C:\Users\kizil\Desktop>  c:; cd 'c:\Users\kizi
1\pythonFiles\lib\python\debugpy\adapter/../..\de
Only 8 byte input !!
Enter data to encrypt: 12345678
Enter key: 12345678
Valid Input !!!
Encrypted data is:  656929909830384163
Hex value : 0x91de245bdc4fa23
Data after decryption is :  12345678
Bellek Kullanımı (14441, 14687)
Encryption Time: 5.9983484745025635 seconds
PS C:\Users\kizil\Desktop>  c:; cd 'c:\Users\kizi
1\pythonFiles\lib\python\debugpy\adapter/../..\de
Only 8 byte input !!
Enter data to encrypt: 123456sa
Enter key: asdfghj
Valid Input !!!
Encrypted data is:  16309434239480473898
Hex value : 0xe256c0079bd3a92a
Data after decryption is :  123456sa
Bellek Kullanımı (14452, 14698)
Encryption Time: 7.1419830322265625 seconds
PS C:\Users\kizil\Desktop>
```

```
1\pythonFiles\lib\python\debugpy\adapter/../..\
Only 8 byte input !!
Enter data to encrypt: 987lşi+%$
Enter key: 456+j8jj
Invalid Input!!
Bellek Kullanımı (13425, 13671)
Encryption Time: 11.877941370010376 seconds
PS C:\Users\kizil\Desktop>  c:; cd 'c:\Users\ki
1\pythonFiles\lib\python\debugpy\adapter/../..\
Only 8 byte input !!
Enter data to encrypt: +9
Enter key:
Valid Input !!!
Encrypted data is:  4246245715305495467
Hex value : 0x3aedb1fb01cefbab
Data after decryption is :  +9
Bellek Kullanımı (14368, 14614)
Encryption Time: 4.464276313781738 seconds
PS C:\Users\kizil\Desktop>  c:; cd 'c:\Users\ki
1\pythonFiles\lib\python\debugpy\adapter/../..\
Only 8 byte input !!
Enter data to encrypt: a
Enter key: a
Valid Input !!!
Encrypted data is:  7133826974413314930
Hex value : 0x630071d62b015b72
Data after decryption is :  a
Bellek Kullanımı (14230, 14476)
Encryption Time: 3.8159289360046387 seconds
PS C:\Users\kizil\Desktop>  c:; cd 'c:\Users\ki
1\pythonFiles\lib\python\debugpy\adapter/../..\
Only 8 byte input !!
Enter data to encrypt: 1
Enter key: 1
Valid Input !!!
Encrypted data is:  14559342943135821178
Hex value : 0xca0d2d44a0b36d7a
Data after decryption is :  1
Bellek Kullanımı (14238, 14484)
Encryption Time: 1.8990635871887207 seconds
PS C:\Users\kizil\Desktop>
```

Anahtar üretimi şeması



Algoritma Tasarımı grafiği diyagramı



https://github.com/EsraKizilelma/kriptoloji

[1] Şifreleme Algoritmalarının Performans Analizi, Eylül 2010, “https://acikerisim.sakarya.edu.tr/bitstream/handle/20.500.12619/80357/T04682.pdf?sequence=1” Bilg.Müh. Ümit GÜNDEN, Nejat YUMUŞAK

[2] Modern Şifreleme Algoritmaları, “https://dergipark.org.tr/tr/download/article-file/319383”, Fatih ŞAHİN

[3] ERKOÇ, K., Kriptoloji ve Bilgi Güvenliği, Y.Lisans, Sakarya Üniversitesi, Bilgisayar ve Bilişim Mühendisliği, sf. 7, 16, 28-30, 44-46, 48, 2004.

[4] CW Kriptoloji – Kriptografi Seminerleri Sonuç Bildirgesi, sf. 8-9, 2009.

[5] KOBLITZ, N., A Course in Number Teory and Cryptography, SpringerVerlag, New York , 1994 [6] HELLMAN, M.E., An Overview of Puplic Key Cryptography, IEEE Communications Society Magazine, 1978.

[7] YERLÐKAYA, T., Şifreleme Algoritmalarının Analizi, Doktora, Trakya Üniversitesi, Bilgisayar Mühendisliği Bölümü, Sf. 8, 22-27, 2006.

[8] ATAY, S., Eliptik Eğri Tabanlı Kriptografik Protokol ve Akıllı Kart Üzerinde Bir Uygulama, Ağ ve Bilgi Güvenliği Sempozyumu, 2005.

[9] ERHAN M. , RSA Algoritmasını Kullanan Şifreleme / Deşifreleme Yazılımının Tasarımı, Y.Lisans, Ðstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, 1993.

[10] YILDIRIM, M., DES ve DES Benzeri Şifreleme Sistemlerinin Diferansiyel Kripto Analizi, Y.Lisans, Ðstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, sf.8-13, 1995.

[11] ÜLGER C., Holotransformasyon Metodu ile Veri Şifreleme, Doktora, Marmara Üniversitesi, Fen Bilimleri Enstitüsü, 1999.

[12] ANDÐÇ, E., Bilgisayar Haberleşmesinde Şifreleme(Kripto) Yazılımıyla Güvenliğin Sağlanması, Y.Lisans, Marmara Üniversitesi, Fen Bilimleri Enstitüsü, sf. 9, 2002.

[13] ÇALIŞKAN, E.M., Şifreleme Algoritmalarının Performans-Kripto 98 Analizleri ve Eğitimde Kullanılması, Y.Lisans, Marmara Üniversitesi, Fen Bilimleri Enstitüsü, sf. 66-72, 2004.