Bilgisayar Güvenliği ve Kriptografi

Performance Comparison of LBlock and PRESENT Encryption Algorithms

LBlock ve PRESENT Şifreleme Algoritmalarının Performans Karşılaştırması

Hazırlayan:

Fatih TEKE

Tarih:

02.04.2025

İçindekiler

● Giriş

a. Amaç ve Kapsam

Günlük nesnelere bilgisayar sistemlerinin yaygın entegrasyonu, özellikle Nesnelerin İnterneti (IoT), yerleşik sistemler ve kablosuz sensör ağları alanlarında, kaynakları sınırlı cihazların dağılımında üssel bir artışa yol açmıştır. Bu cihazlar, sınırlı işlem gücü, bellek kapasitesi ve enerji rezervleri ile tanımlanır ve geleneksel güvenlik önlemlerinin uygulanması açısından benzersiz zorluklar ortaya koymaktadır. Gelişmiş Şifreleme Standardı (AES) ve Veri Şifreleme Standardı (DES) gibi standart kriptografik algoritmalar, genellikle bu sınırlı ortamlarda uygulanabilir olmayan işlem kaynakları ve güç tüketimi seviyeleri gerektirir. Bu sınırlama, Kriptografi alanında "Hafif Kriptografi" (Lightweight Cryptography - LWC) [1] olarak bilinen özel bir alanın ortaya çıkmasına yol açmıştır.

Hafif kriptografi, kaynak ayak izini en aza indirirken sağlam güvenlik sunmak üzere özel olarak tasarlanmış hash fonksiyonları, akış şifreleri ve blok şifreleri gibi bir dizi kriptografik ilkeye sahiptir [1]. Bu tür çözümlere olan ihtiyaç, IoT cihazlarından toplanan verilerin siber saldırıların hedefi olabilecek önemli bir kaynak olmasından dolayı, gizliliğin, bütünlüğün ve kimlik doğrulamanın sağlanabilmesi için etkili şifrelemenin gerekliliği ile pekişmektedir [7]. Ayrıca, IoT'nin kritik altyapılar ve sağlık hizmetleri, endüstriyel otomasyon gibi hassas uygulamalarda giderek artan şekilde kullanılması, bu cihazların ve iletişimlerinin güvence altına alınmasının önemini daha da artırmaktadır [1].

Bu kritik ihtiyacı tanıyan uluslararası standardizasyon organları ve araştırma kurumları, sınırlı ortamlara uygun hafif kriptografik algoritmaların geliştirilmesi ve değerlendirilmesi için aktif bir şekilde çalışmaktadır [7]. ABD Ulusal Standartlar ve Teknoloji Enstitüsü (NIST), sınırlı ortamlara uygun hafif kriptografik algoritmaların belirlenmesi için yıllar süren bir program başlatmış ve bu çaba, Şubat 2023'te Ascon algoritmaları ailesinin seçilmesiyle son bulmuştur [8]. Bu gelişme, LWC'nin öneminin küresel anlamda kabul edilmesini göstermektedir [12]. Devam eden standardizasyon çabaları ve IoT cihazlarının basit sensörlerden karmaşık kontrol sistemlerine kadar genişleyen çeşitliliği, farklı senaryoların özel gereksinimlerini karşılamak için bir dizi hafif kriptografik çözüme ihtiyaç duyulduğunu vurgulamaktadır [2]. Bu alandaki temel zorluk, güvenlik, uygulama maliyeti (kaynak kullanımı açısından) ve performans verimliliği arasında hassas bir denge sağlamaktır [13].

Blok şifreler, simetrik anahtar kriptografisinin temel bir kategorisi olup, sabit boyutlu veri blokları üzerinde çalışarak, açık metin bloklarını gizli anahtar altında şifreli bloklara dönüştürür [1]. Bu algoritmalar, dijital bilgilerin gizliliği ve bütünlüğü sağlamak için birçok kriptografik protokolün temelini oluşturur [20]. Blok şifrelerinin ana tasarım ilkeleri, sabit bir blok boyutu ve anahtar boyutu belirlemek, ve güvenlik seviyesini elde etmek için çoklu turlar boyunca yinelemeli olarak uygulanan bir tur fonksiyonu kullanmaktır. Blok şifre tasarımında yaygın olarak kullanılan yapılar arasında Substitution-Permutation Networks (SPN) ve Feistel ağları bulunur, her ikisi de karmaşıklık (anahtar ve şifreli metin arasındaki ilişkiyi karmaşık hale getirmek) ve yayılma (her bir açık metin bitinin birden fazla şifreli metin biti üzerinde etkisini yaymak) sağlamak amacı güder [19]. Yapılarının doğası gereği, blok şifreleri, şifrelenmesi gereken veri miktarının bilindiği durumlarda özellikle etkilidir ve akış şifrelerine kıyasla daha iyi yayılma ve hata iletimi avantajları sunar [1]. Esneklikleri, güvenli iletişim protokollerinden (TLS/SSL gibi) dosya ve disk şifrelemesine kadar birçok güvenlik uygulamasında kullanılmalarına olanak tanır [19].

Hafif blok şifreleri, IoT cihazları ve diğer sınırlı sistemler gibi kaynak kısıtlamaları olan sistemlere özel olarak tasarlanmış bir blok şifresi alt kümesini temsil eder. Geleneksel blok şifrelerinin çoğu, geniş kaynaklara sahip genel amaçlı bilgisayarlarda performans için tasarlanmışken, hafif blok şifreleri, enerji tüketimini, işlem gereksinimlerini, depolama ve bellek kullanımını en aza indirmeye öncelik verir [8]. Bu, geleneksel şifreleme tasarım ilkelerinden sapmayı gerektirir ve genellikle daha küçük blok ve anahtar boyutları, basitleştirilmiş anahtar planlama algoritmaları, azaltılmış şifreleme turları sayısı ve optimize edilmiş ikame kutusu (S-box) tasarımlarının kullanılmasını içerir [24]. Temel amaç, güçlü şifreleme sağlamak olsa da, önemli ölçüde azaltılmış hesaplama yükü ile cihazların sınırlı kapasitelerine uygun hale getirilmiş olmaktır [25]. LWC'nin ortaya çıkışı, 4-bit veya 8-bit mikrodenetleyiciler gibi küçük bellek alanlarına ve katı güç bütçelerine sahip cihazlarda AES gibi kaynak tüketici algoritmaların dağıtılmasının imkansızlığını ele alır. Hafif blok şifrelerinin tasarımı, genellikle belirli uygulamaların taleplerini karşılamak için güvenlik ile verimlilik arasında stratejik bir denge kurmayı içerir [23].

LBlock, 2011 yılında tanıtılan, 64-bit blok boyutu ve 80-bit anahtar ile tasarlanmış hafif bir blok şifresidir [28]. Feistel yapısının bir varyantını kullanır ve 32 turdan oluşur. Şifreleme süreci, 64-bit açık metni iki 32-bit yarıya böler ve ardından 32 tur boyunca yinelemeli olarak bir tur fonksiyonu uygular. Tur fonksiyonu, kendisi bir karmaşıklık fonksiyonu (S) ve bir yayılma fonksiyonu (P) içerir [29]. Karmaşıklık fonksiyonu, güvenlik için kritik olan doğrusal olmayanlık getiren sekiz paralel 4x4 S-box'tan oluşur. Yayılma fonksiyonu (P), sekiz 4-bit kelimenin bir permütasyonudur ve girişteki değişikliklerin şifreleme üzerinden etkin bir şekilde yayılmasını sağlar [29]. Anahtar planlaması, 80-bit anahtardan 32 tur alt anahtarı üretir[31]. Güvenlik değerlendirmeleri, tam 32 turlu LBlock'un, diferansiyel kriptanaliz, doğrusal kriptanaliz, imkansız diferansiyel kriptanaliz ve ilişkili anahtar saldırıları dahil olmak üzere bilinen kriptoanalitik tekniklere karşı yeterli bir güvenlik marjı sağladığını önermektedir [28]. LBlock'un donanım uygulamaları, 0.18 µm teknolojisinde yaklaşık 1320 Kapı Eşdeğeri (GE) gerektirdiği ve 100 KHz'de 200 Kbps'lik bir verim sağladığı bildirilmiştir [28]. 8-bit mikrodenetleyicilerdeki yazılım uygulamaları, her şifreleme işlemi başına yaklaşık 3955 saat döngüsü gerektirir. Ancak, bağımsız istatistiksel analiz, şifrenin çıktısında potansiyel bir rastgelelik eksikliği olduğunu göstermiştir ve kriptoanalitik çabalar, azaltılmış turlu versiyonları başarıyla saldırıya uğratmıştır [31].

PRESENT, 2007 yılında geliştirilen, 64-bit blok boyutuna ve ya 80-bit ya da 128-bit anahtar desteğine sahip başka bir öne çıkan hafif blok şifresidir [25]. Bir Substitution-Permutation Network (SPN) yapısını kullanır ve 31 turdan oluşur. Her turda üç ana işlem yer alır: addRoundKey işlemi (mevcut durumu bir tur anahtarıyla XOR'lamak), bir sBoxLayer ve bir pLayer. sBoxLayer, 64-bit durum üzerinde paralel olarak on altı kez uygulanan tek bir 4-bit'e 4-bit S-box kullanır. pLayer, durumun bitlerini yeniden düzenleyen bir bit permütasyonudur. PRESENT'in tasarımı, yaklaşık 1570 GE gerektiren donanım uygulamaları için son derece optimize edilmiştir [34]. Donanımda verimli olmasına rağmen, bit odaklı permütasyonları, yazılım dostu olmasını zorlaştırmaktadır. PRESENT, hafif kriptografi için uluslararası standartlara dahil edilmiştir ve bu da geniş çapta tanınması ve kabul edilmesini gösterir [37]. Verimliliğine rağmen, PRESENT'in basit tasarımı, daha yavaş karmaşıklık ve yayılma özelliklerine yol açabileceği not edilmiştir [28]. Kriptoanaliz, azaltılmış turlar üzerinde kesilmiş diferansiyel saldırılar ve tam şifre üzerinde biclique saldırıları gibi zayıflıkları ortaya koymuştur [37].

Tablo 1. *LBlock ve PRESENT Özelliklerinin Karşılaştırılması*

Özellik	LBlock	PRESENT
Blok Boyutu	64 bits	64 bits
Anahtar Boyutu	80 bits	80 or 128 bits
Round Sayısı	32	31
Yapı	Variant Feistel	SPN
Donanım GE	~1320	~1570
Yazılım Döngüsü	~3955 (8-bit microcontroller)	Açıkça belirtilmemiş
Anahtar Programı	Açıklanmış	Açıklanmış
S-box	8 4x4	1 4x4 (16 kez uygulandı)
Güvenlik İddiaları	Bilinen saldırılara karşı	Bilinen saldırılara karşı
Bilinen Ataklar	Azaltılmış tur güvenlik açıkları	Kesik diferansiyel, biklique
Standardizasyon	Açıkça belirtilmemiş	ISO/IEC 29167-11, ISO/IEC 29192-2

LBlock ve PRESENT gibi hafif blok şifrelerinin performansını karşılaştırmak, kaynak kısıtlamalı cihazlar bağlamında son derece önemlidir [4]. Bir kriptografik algoritmanın, IoT veya yerleşik sistemler alanındaki belirli bir uygulamaya uygunluğu, uygulama özellikleriyle, yani boyutu (devre alanı veya bellek kullanımı açısından), güç tüketimi ve işlem hızı (verim ve gecikme) gibi faktörlerle büyük ölçüde etkilenir [41]. Bazı hafif şifreler, donanım uygulamalarında mükemmel performans gösterecek şekilde tasarlanmış olup, daha küçük bir silikon alanı ve daha yüksek verim sağlarken, diğerleri daha düşük gecikme ve esneklik sunarak yazılım uygulamaları için daha verimli olabilir [1]. Farklı algoritmalar arasındaki seçim, genellikle bu performans ölçütleri ile sağladıkları güvenlik seviyesi arasında bir dengeyi içerir [4].

IoT uygulamaları için, gecikme, güvenilirlik, ölçeklenebilirlik ve veri iletimi gibi faktörler, belirli bir kullanım senaryosuna bağlı olarak farklı derecelerde önemli olabilir [25]. Örneğin, sağlık izleme ve otonom sürüş sistemleri, gerçek zamanlı tepki için düşük gecikme talep ederken, endüstriyel IoT uygulamaları, maliyetli kesintileri önlemek için güvenilirliği ön planda tutabilir. Kaynak kısıtlamalı IoT kartlarında hafif kriptografik algoritmaların performansını ölçmek için yapılan benchmark (karşılaştırmalı test) çalışmaları, yürütme süresi, bellek kullanımı ve güç tüketimi gibi ölçümleri inceleyerek hangi algoritmanın belirli bir uygulamanın gereksinimlerine en uygun olduğunu belirlemek açısından kritik öneme sahiptir [4]. Raspberry Pi gibi yaygın IoT platformlarında yapılan gerçek dünya değerlendirmeleri, farklı hafif şifreler arasındaki pratik performans trade-off'ları hakkında değerli bilgiler sunabilir [43]. Bu trade-off'ları anlamak, geliştiriciler ve güvenlik mimarları için, kaynak sınırlı cihazları etkili bir şekilde güvence altına almak için en uygun kriptografik araçları seçmek adına önemlidir.

Bu Python çalışması, LBlock ve PRESENT hafif blok şifre algoritmaları arasında deneysel bir performans karşılaştırması yapmayı amaçlamaktadır. Özellikle, bu çalışma, şifreleme ve şifre çözme hızı (verim), bellek kullanımı (RAM ayak izi) ve Python ortamında potansiyel enerji tüketimi açısından performanslarını değerlendirmeyi ve karşılaştırmayı hedeflemektedir. Ayrıca çalışma, bu algoritmaların performansının, IoT uygulamalarında karşılaşılan farklı senaryoları taklit eden değişen veri boyutlarıyla nasıl ölçeklendiğini değerlendirmeyi amaçlamaktadır; küçük sensör verisi paketlerinden daha büyük yazılım güncellemelerine kadar. Bu karşılaştırmayı Python ortamında

gerçekleştirerek, çalışma, bu iki öne çıkan hafif blok şifresinin yazılım performans özellikleri hakkında bilgiler sunmaktadır.

Bu çalışmanın kapsamı, LBlock ve PRESENT algoritmalarının yazılım performansını Python tabanlı uygulamalar kullanarak değerlendirmekle sınırlıdır. Ölçülecek başlıca performans metrikleri şunlardır:

Şifreleme ve Şifre Çözme Hızı: Farklı boyutlardaki verilerin şifrelendiği ve çözüldüğü süre olarak ölçülecek ve bu süre, saniye başına byte cinsinden verim hesaplamak için kullanılacaktır.

Bellek Kullanımı: Şifreleme ve şifre çözme işlemleri sırasında Python sürecinin maksimum RAM tüketimi izlenerek belirlenecektir.

Çalışmada, LBlock ve PRESENT algoritmalarının uygulanmasında saf Python kriptografi algoritmaları geliştirilecektir. Zaman ölçümü için standart Python modülleri (time) ve bellek profil oluşturma için (memory_profiler veya psutil) kullanılacak ve performans verileri toplanacaktır. Test ortamı, belirli donanım (CPU, RAM) ve yazılım (işletim sistemi, Python sürümü) yapılandırmalarıyla standart bir masaüstü veya dizüstü bilgisayardan oluşacaktır ve bu yapılandırmalar ayrıntılı olarak belgelenecektir. Performans, farklı veri boyutları kullanılarak değerlendirilecektir; bu sayede algoritmaların artan veri boyutlarıyla nasıl ölçeklendiği gözlemlenecektir. Çalışma, temel şifreleme performansının doğrudan karşılaştırılması ve basitlik amacıyla öncelikle Elektronik Kod Defteri (ECB) işlem moduna odaklanacaktır.

Bu çalışmanın bazı sınırlamalarını kabul etmek önemlidir. Python ortamında, yorumlanan bir dil olduğu için, performans, genellikle bu algoritmaların dağıtıldığı yerleşik donanım platformlarında elde edilebilecek performansı doğrudan yansıtmayabilir. Ayrıca, doğru enerji tüketimi ölçümleri, standart bir yazılım testi ortamında elde edilmesi zor olabilir. Çalışma, belirli performans metriklerine odaklanacak ve literatürde zaten belirtilenlerin ötesine geçmeden yan kanal saldırılarına karşı direnç veya detaylı güvenlik analizi gibi diğer önemli yönlere girmeyecektir.

Bu Python tabanlı LBlock ve PRESENT performans karşılaştırmasının, hafif kriptografi alanına değerli deneysel veriler ile katkı sağlaması beklenmektedir. Şifreleme hızı ve bellek kullanımı açısından yazılım performanslarının nicel bir analizini sunarak, bu çalışma genellikle donanım uygulamaları ve teorik güvenlik özelliklerine odaklanan mevcut araştırmalara tamamlayıcı bir katkı yapacaktır [1]. Bulgular, yazılım bağlamında bu iki yaygın şekilde incelenen hafif blok şifresi arasındaki pratik trade-off'lar hakkında içgörüler sunacak ve yazılım uygulamalarının tercih edildiği veya gerekli olduğu kaynak kısıtlamalı uygulamalarda algoritma seçiminde geliştiriciler ve güvenlik uzmanlarına rehberlik edebilir. Farklı veri boyutlarında yapılan değerlendirme, bu algoritmaların yazılımda nasıl ölçeklendiğini daha da aydınlatacaktır. Nihayetinde, bu çalışma, LBlock ve PRESENT'in performans özelliklerini anlamayı derinleştirmeyi amaçlamakta ve hafif kriptografi alanındaki daha geniş bilgi birikimine katkı sağlayarak, bu kritik alandaki gelecekteki araştırma ve geliştirmeleri bilgilendirecektir [44].

b. Arka Plan Bilgisi

Hafif blok şifreleme algoritmaları, kısıtlı kaynaklara sahip cihazlarda bile güvenli ve verimli bir şekilde çalışabilen şifreleme algoritmalarıdır. Bu algoritmalar, düşük hesaplama karmaşıklığı, düşük enerji tüketimi ve küçük bellek gereksinimleri ile karakterize edilir. Hafif blok şifreleme algoritmalarının önemi, özellikle aşağıdaki alanlarda belirgindir:

Nesnelerin İnterneti (IoT): IoT cihazları, genellikle sınırlı işlem gücüne ve enerjiye sahiptir. Bu nedenle, bu cihazlarda güvenli iletişim sağlamak için hafif şifreleme algoritmaları gereklidir.
Akıllı Kartlar: Akıllı kartlar, ödeme sistemleri, kimlik doğrulama ve diğer güvenlik uygulamalarında yaygın olarak kullanılmaktadır. Bu kartlar, genellikle sınırlı kaynaklara sahip olduğundan, hafif şifreleme algoritmaları bu alanda da önemlidir.
Radyo Frekansı Tanımlama (RFID): RFID etiketleri, tedarik zinciri yönetimi, envanter takibi ve diğer uygulamalarda kullanılmaktadır. Bu etiketlerin güvenliği, hafif şifreleme algoritmaları ile sağlanabilir.
Sağlık Cihazları: Giyilebilir sağlık cihazları ve implante edilebilir tıbbi cihazlar gibi alanlarda da hafif şifrelemeye ihtiyaç duyulmaktadır.

Kullanım Alanları:

- IoT cihazları: Sensörler, aktüatörler, akıllı ev cihazları ve endüstriyel IoT cihazları gibi çeşitli IoT cihazlarında güvenli iletişim sağlamak için kullanılır.
- Akıllı kartlar: Ödeme sistemleri, kimlik doğrulama, toplu taşıma kartları ve diğer akıllı kart uygulamalarında kullanılır.
- RFID: Tedarik zinciri yönetimi, envanter takibi, erişim kontrolü ve diğer RFID uygulamalarında kullanılır.
- Sağlık: Giyilebilir sağlık cihazları, implante edilebilir tıbbi cihazlar ve tele-tıp uygulamalarında kullanılır.
- Otomotiv: Araç içi iletişim, araç-araç iletişimi ve araç-altyapı iletişimi gibi otomotiv uygulamalarında kullanılır.
- Savunma sanayi: Askeri iletişim, insansız hava araçları ve diğer savunma uygulamalarında kullanılır.

Hafif blok şifreleme algoritmalarına PRESENT, SIMON ve SPECK ya da LEA örnek olarak verilebilir. Hafif blok şifreleme algoritmalarının genel olarak avantajları:

- Düşük enerji tüketimi
- Düşük hesaplama karmaşıklığı
- Küçük bellek gereksinimleri
- Kısıtlı kaynaklara sahip cihazlarda bile güvenli ve verimli çalışma

gösterilebilir. Bunun yanı sıra dezavantaj olarak:

- Geleneksel şifreleme algoritmalarına göre düşük güvenlik seviyesi
- Bazı algoritmaların patentli olması

söylenebilir.

c. Literatür Taraması

Kaynak kısıtlamalı cihazların hızla yayılması, Nesnelerin İnterneti (IoT), yerleşik sistemler, Radyo Frekansı Tanımlama (RFID) etiketleri ve kablosuz sensör ağları gibi alanları kapsayarak, bu cihazların benzersiz sınırlamalarına uygun sağlam veri güvenliği mekanizmalarına yönelik kritik bir ihtiyaç doğurmuştur. Geleneksel blok şifreleri, örneğin Gelişmiş Şifreleme Standardı (AES), yüksek güvenlik seviyeleri sunsa da, genellikle bu ortamlar için pratik olmamaktadır çünkü hesaplama gücü ve bellek kaynakları üzerinde önemli talepler oluştururlar. Bu durum, minimal hesaplama yükü ile güvenlik sağlamayı amaçlayan hafif blok şifrelerine yönelik önemli bir araştırma dalgasını tetiklemiştir [45]. Bu şifreler, daha küçük blok boyutları, daha basit anahtar planlamaları, azaltılmış tur sayıları ve optimize edilmiş Substitution box (S-box) tasarımları gibi özelliklerle tanımlanır. Hafif

kriptografi geliştirilmesi, genellikle "mimari tasarım üçgeni" olarak adlandırılan, uygulama maliyeti, güvenlik gücü ve performans verimliliği arasında dikkatli bir dengeleme gerektirir [55]. Bağlantılı cihazların genişleyen manzarasını güvence altına alma ihtiyacı, bu nedenle kriptografi araştırmalarındaki itici güç haline gelmiştir ve kaynak kısıtlamaları içinde etkili bir şekilde çalışabilen algoritmalar üzerine odaklanmayı teşvik etmiştir. Bu da, belirli uygulama senaryoları için optimal çözümler elde etmek amacıyla tasarım trade-off'larının sürekli olarak keşfedilmesini gerektirmektedir.

Önerilen çeşitli hafif blok şifreleri arasında, LBlock ve PRESENT, her biri farklı tasarım felsefelerini yansıtan önemli örnekler olarak öne çıkmaktadır [57]. 2011 yılında Wu ve Zhang tarafından tanıtılan LBlock, 64-bit blok boyutu ve 80-bit anahtar boyutuna sahiptir. Tasarımı, özellikle kablosuz sensör ağları gibi kaynak kısıtlamalı donanım ortamlarında verimli bir şekilde uygulanabilir olmayı hedeflemiştir [52]. 2007 yılında Bogdanov ve arkadaşları tarafından yayımlanan PRESENT ise, aynı şekilde 64-bit blok boyutuna sahip olup, 80 ve 128 bit anahtar boyutlarını desteklemektedir. Hafif kriptografi alanında ultra hafif yapısı ile tanınan ve ISO/IEC 29192 standardına dahil edilerek bir referans noktası haline gelmiştir [50]. LBlock ve PRESENT'in incelenmesi, hafif kriptografik çözümlerin evrimindeki farklı aşamaları ve yaklaşımları temsil ettikleri için çok önemlidir; PRESENT, yeni önerilerin genellikle karşılaştırıldığı erken ve etkili bir referans noktası olarak hizmet etmektedir [50]. LBlock'un sonraki gelişimi, hafif kriptografide verimlilik ve güvenlik dikkate alındığında sürekli bir iyileştirme çabasını yansıtmaktadır [52].

Kaynak kısıtlamalı ortamlarda pratik performansın önemi göz önünde bulundurulduğunda, belirli platformlar üzerindeki ampirik değerlendirmeler, bu algoritmaların gerçek dünyadaki davranışlarını anlamak için gereklidir [49]. Teorik analizler ve donanım odaklı değerlendirmeler, çoğu IoT cihazında bulunan genel amaçlı işlemcilerdeki yazılım uygulamalarının inceliklerini tam olarak yansıtmayabilir [59]. Yazılım performansı, işlemci talimat seti mimarisi, derleyici optimizasyonlarının verimliliği ve bellek erişim desenleri gibi faktörlerden etkilenebilir; bu faktörler, teorik modeller veya donanım simülasyonlarında doğru bir şekilde temsil edilmeyebilir. Bu nedenle, Python tabanlı bir performans karşılaştırma çalışması, LBlock ve PRESENT'in yazılım düzeyindeki özellikleri hakkında değerli bilgiler sunabilir ve farklı IoT senaryolarındaki pratik dağıtımlar için ilgili platforma özgü davranışları ortaya koyabilir.

Birçok akademik çalışma, LBlock ve PRESENT'in performansını çeşitli platformlarda doğrudan karşılaştırmıştır. Dikkate değer bir çalışma, araştırmacılar tarafından yapılan ve bu iki blok şifresinin Raspberry Pi 3 üzerinde performansını değerlendiren bir çalışmadır, bu platform IoT uygulamaları için popülerdir. Değerlendirme, şifreleme ve şifre çözme işlemleri için enerji tüketimi, verim, çalışma süresi, RAM ve ROM tüketimi gibi anahtar metriklere odaklanmıştır ve farklı yük boyutları üzerinden gerçekleştirilmiştir [57].

Bu çalışmanın bulguları, PRESENT'in şifreleme ve şifre çözme işlemleri için LBlock'tan önemli ölçüde daha fazla enerji tükettiğini, özellikle yük boyutu 32 byte'dan 2048 byte'a yükseldikçe daha belirgin hale geldiğini göstermiştir. Bu, enerji verimliliğinin kritik olduğu batarya ile çalışan IoT cihazları için, LBlock'un PRESENT'e kıyasla daha uygun bir seçim olabileceğini düşündürmektedir [57]. Aksine, LBlock, Raspberry Pi 3 üzerinde tüm test edilen yük boyutları için hem şifreleme hem de şifre çözme işlevlerinde daha yüksek ortalama verim hızları göstermiştir. Hızlı veri işlemeye ihtiyaç duyulan uygulamalarda, bu daha yüksek verim, LBlock'un bu özel platformdaki performans avantajını göstermektedir. Çalışma süresi açısından, PRESENT, özellikle 1024 ve 2048 byte'lık daha büyük yüklerde şifreleme ve şifre çözme modlarında düzensiz bir artış sergilerken, LBlock'un çalışma süresi, yük boyutuyla daha öngörülebilir şekilde artmıştır. PRESENT'in daha büyük veri kümeleri ile

tutarsız performansı, Raspberry Pi 3'te potansiyel yazılım uygulama verimsizliklerini veya darboğazlarını gösterebilir [57].

Bellek tüketimi açısından, her iki algoritma da RAM kullanımında benzer eğilimler göstererek, yük boyutu arttıkça tüketimin arttığını ortaya koymuştur. Ancak, LBlock, PRESENT'e kıyasla sürekli olarak biraz daha yüksek RAM tüketimi göstermiştir. Her iki algoritma da kaynak kısıtlamalı cihazlar için uygun olan nispeten küçük RAM ayak izleri korusa da, LBlock'un biraz daha yüksek gereksinimi, son derece sınırlı belleğe sahip cihazlar için bir dikkate alındırma olabilir. ROM tüketimi için yapılan ölçümler de yük boyutları arttıkça her iki şifre için de artış eğilimi göstermiştir. Veriler, LBlock'un, PRESENT'e kıyasla daha fazla ROM gerektirdiğini göstermiştir. Bu, PRESENT'in daha küçük bir kod boyutuna sahip olabileceğini ve bu durumun çok sınırlı depolama kapasitesine sahip cihazlar için avantajlı olabileceğini düşündürmektedir [57].

Daha net bir genel bakış sunmak için, aşağıdaki tablo, yukarıda bahsedilen çalışmaya dayalı olarak Raspberry Pi 3'te LBlock ve PRESENT'in performans karşılaştırmasını özetlemektedir:

Tablo 2. *LBlock ve PRESENT Performans Karşılaştırması*

Metric	Yük Boyutu	LBlock (Şifreleme)	PRESENT (Şifreleme)	LBlock (Deşifreleme)	PRESENT (Deşifreleme)
Enerji Tüketimi	32-2048 bayt	Daha Düşük	Daha Yüksek	Daha Düşük	Daha Yüksek
Verim	32-2048 bayt	Daha Yüksek	Daha Düşük	Daha Yüksek	Daha Düşük
Yürütme Süresi	32-2048 bayt	Daha Doğrusal	Düzensiz Artış	Daha Doğrusal	Düzensiz Artış
RAM Tüketimi	32-2048 bayt	Biraz Daha Yüksek	Biraz Daha Düşük	Biraz Daha Yüksek	Biraz Daha Düşük
ROM Tüketimi	32-2048 bayt	Daha Yüksek	Daha Düşük	Daha Yüksek	Daha Düşük

Performans özelliklerinin, algoritmaların değerlendirildiği belirli hedef platforma ve ortama bağlı olarak önemli ölçüde değişebileceğini belirtmek önemlidir [62]. Örneğin, AES, PRESENT ve LBlock da dahil olmak üzere on hafif algoritmanın, bellek kullanımı, enerji tüketimi, verimlilik ve çalışma süresi gibi metrikler üzerinden bulut altyapısında değerlendirilmesi, kaynakları sınırlı yerleşik cihazlar üzerinde yapılan bir değerlendirmeye kıyasla farklı sonuçlar verebilir [62]. Bulut ortamları, IoT cihazlarına göre çok daha fazla hesaplama gücü ve belleğe sahip olduğundan, bu durum performans darboğazlarını ve kriptografik algoritmalar için optimal seçimleri değiştirebilir.

Ayrıca, PRESENT ve LBlock dahil olmak üzere 13 hafif şifrenin, farklı mikrodenetleyici mimarileri (8-bit ATmega, 16-bit MSP430 ve 32-bit ARM işlemcileri) üzerinde karşılaştırıldığı bir makale, çalışma süresi, RAM ayak izi ve ikili kod boyutu gibi metriklere odaklanmıştır [49]. Bu metrikler açısından LBlock ve PRESENT'in göreli performansı, işlemci mimarilerindeki ve algoritmaların her bir işlem üzerindeki verimliliğindeki farklılıklardan dolayı bu platformlar arasında değişebilir. LBlock ve PRESENT arasındaki performans sıralaması, ayrıca hız öncelikli, bellek kullanımını minimize etme veya kod boyutunu küçültme gibi uygulamanın belirli optimizasyon hedeflerinden de etkilenebilir [49]. Bu bulgular, belirli bir uygulama ve donanım için en uygun hafif şifrenin

seçilmesine rehberlik edecek platforma özel performans değerlendirmelerinin önemini vurgulamaktadır.

Hafif blok şifrelerinin yerleşik sistemlerde ve IoT cihazlarında dağıtımını düşünürken, donanım uygulama maliyetleri, genellikle Kapı Eşdeğerleri (GE) cinsinden ölçülen, kritik bir faktördür. LBlock'un 0.18 µm teknolojisindeki donanım uygulamasının yaklaşık olarak 1320 GE gerektirdiği ve 100 KHz saat frekansında 200 Kbps'lik bir verim elde ettiği tahmin edilmiştir [52]. Bu alan maliyeti, çeşitli bileşenler arasında dağılmıştır; bunlar arasında 64-bit veri kaydı (yaklaşık 384 GE), anahtar ekleme mantığı (yaklaşık 87 GE), sekiz paralel 4x4 S-kutusu içeren S-kutusu katmanı (yaklaşık 174.8 GE), permütasyon katmanı (0 GE, çünkü bağlantı ile uygulanır), 32-bit XOR işlemi (yaklaşık 87 GE), 80-bit anahtar kaydı (yaklaşık 480 GE), anahtar planı S-kutuları (yaklaşık 43.7 GE), 5-bit sabit XOR (yaklaşık 13.5 GE) ve kontrol mantığı (yaklaşık 50 GE) yer alır [52]. Dikkat çekici olarak, kaydedicileri yeniden kullanan daha alan dostu bir LBlock uygulaması, yaklaşık 866.3 GE'lik daha küçük bir ayak izi elde edebilir, ancak bu, ek RAM gereksinimi ve daha düşük verim ile yapılan bir takas anlamına gelir. Bu, LBlock'un donanım uygulamasında bir dereceye kadar esneklik sunduğunu, tasarımcıların alan veya performans için optimize edilmiş sürümler arasında seçim yapmalarına olanak tanıdığını göstermektedir.

Buna karşılık, PRESENT-80 blok şifresinin donanım uygulama maliyeti, 64-bit veri yolu kullanarak bir saat döngüsünde bir tur işleyen, alan optimize edilmiş bir şifreleme yalnızca uygulaması için yaklaşık 1570 GE'dir [51]. PRESENT-80'deki farklı modüllerin alan gereksinimleri şunlardır: veri durumu (384.39 GE), anahtar durumu (480.49 GE), S-kutusu katmanı (448.45 GE), anahtar planında kullanılan S-kutusu (28.03 GE), permütasyon katmanı (0 GE), tur sayacı durumu (28.36 GE), anahtar planı sayacı XOR (13.35 GE), sayaç için kombinatoryal mantık (12.35 GE), anahtar XOR işlemi (170.84 GE) ve diğer çeşitli mantık (3.67 GE). Bu uygulama, 80-bit anahtar ile 64-bit bir bloğu şifrelemek için 32 saat döngüsü gerektirir ve simüle edilmiş güç tüketimi 5µW'dir [51]. POWER-optimize edilmiş bir PRESENT-80 sürümü, 3.3µW'lik daha düşük güç tüketimi sağlarken, alan maliyetinde hafif bir artışla 1623 GE'ye ulaşmaktadır. Ayrıca, 128-bit anahtar kullanan PRESENT-128'in tahmin edilen alanı yaklaşık 1886 GE'dir. PRESENT'in tasarımı, donanım verimliliğini ön planda tutarak, nispeten küçük bir kapı sayısı sağlar, bu da onu çok kaynak kısıtlı donanım ortamlarında dağıtım için özellikle uygun hale getirir. Donanım maliyetlerini karşılaştırırken, her iki şifre de hafif olarak kabul edilse de, belirli GE sayısı, uygulama tercihlerine ve anahtar boyutuna göre değişebilir. Doğrudan bir karşılaştırma yapmak, bu faktörlerin dikkate alınmasını gerektirir, böylece belirli bir donanım platformu için en uygun şifre belirlenebilir.

Mikrodenetleyicilerde yazılım uygulama performansı, IoT cihazları için bir diğer kritik konudur. LBlock'un 8-bit mikrodenetleyicilerdeki yazılım uygulamasının, tek bir 64-bit düz metin bloğunu şifrelemek için yaklaşık 3955 saat döngüsü gerektirdiği bildirilmiştir [52]. Bu performans, hafif blok şifreleri arasında rekabetçi olarak kabul edilmektedir. LBlock'un tasarımı, her turda 4-bit kelime permütasyonu ve her turun sağ yarısında 8-bit dönüşüm içerdiğinden, yazılım platformlarına, özellikle 8-bit mikrodenetleyicilere, özellikle uygun hale gelir [52]. 8-bit odaklı bir yazılım uygulamasında, sekiz 4-bit S-kutusu ve tur fonksiyonu içindeki 4-bit kelime permütasyonu, dört 8-bit arama tablosu kullanılarak verimli bir şekilde birleştirilebilir ve uygulanabilir, bu da daha hızlı işleme katkıda bulunur. Bu, LBlock'un tasarımının yazılım verimliliğini dikkate alarak yapıldığını ve mikrodenetleyici tabanlı IoT cihazları için uygulanabilir bir seçenek olduğunu gösteriyor.

Buna karşılık, sağlanan araştırma kesitleri, PRESENT'in mikrodenetleyicilerdeki yazılım uygulama performansına dair ayrıntılı bilgiler sunmamaktadır [51]. PRESENT'i açıklayan makale, öncelikle

donanım verimliliğine odaklanmakla birlikte, şifrenin yazılımda uygulanabileceğini de belirtmektedir [51]. PRESENT'in tasarımının sadeliği, gerçekten de yazılım uygulamalarını basitleştirebilir. Bununla birlikte, LBlock ve PRESENT'in kapsamlı bir karşılaştırması, diğer mevcut literatüre dayalı olarak PRESENT'in çeşitli mikrodenetleyici mimarilerindeki yazılım performansına dair daha fazla araştırma gerektirmektedir.

Hafif blok şifrelerinin güvenliği son derece önemlidir ve hem LBlock hem de PRESENT, kapsamlı kriptanalitik incelemelere tabi tutulmuştur. LBlock'un güvenlik değerlendirmesi, başlangıçta diferansiyel kriptanaliz, lineer kriptanaliz, imkansız diferansiyel kriptanaliz ve ilgili anahtar saldırıları gibi bilinen saldırılara karşı yeterli bir güvenlik marjı gösterdiği belirtilmiştir [52]. Ancak, sonraki araştırmalar bazı potansiyel zayıflıkları ortaya çıkarmıştır. Örneğin, LBlock'un anahtar planlamasında yapılan diferansiyel hata analizi (DFA), anahtar arama alanını daraltabileceğini göstermiştir [58]. Bu, LBlock'un başlangıçta güçlü olarak kabul edilmesine rağmen, anahtar planlamasının bazı saldırı türlerine karşı savunmasız olabileceğini düşündürmektedir. Ayrıca, küp saldırıları, LBlock'un azaltılmış tur versiyonlarına (7, 8 ve 9 tur) ve hatta tam 32-tur versiyonuna, tek bit sızıntı yan kanal küp saldırı modeli altında başarılı bir şekilde uygulanmıştır [63]. Bu, LBlock'un, özellikle saldırganın yan kanal bilgilerine erişebildiği senaryolarda küp saldırılarına karşı savunmasız olabileceğini gösterir. LBlock tarafından üretilen şifre metninin NIST İstatistiksel Test Paketi ile yapılan istatistiksel analizi, %1 anlamlılık seviyesinde ideal rastgelelikğe sahip olmadığını ortaya koymuştur [59]. LBlock'un rastgeleliğini artırmak için önerilen değişikliklere rağmen, bu ilk bulgu, çıktısının mükemmel bir şekilde rastgele bir diziden ayırt edilemez olmadığını ve bunun bazı uygulamalar için olumsuz etkileri olabileceğini göstermektedir. Kriptanalitik çalışmalar, LBlock üzerindeki imkansız diferansiyel saldırıları da geliştirmiş, bu saldırıları 21 ve 22 tura kadar uzatmıştır [63]. Bu sürekli araştırma, LBlock'un güvenlik sınırlarını anlamamızı geliştiriyor ve saldırılar, şifrenin toplam turlarının önemli bir kısmına ulaşmıştır.

Öte yandan, PRESENT, diferansiyel ve lineer kriptanaliz gibi ana kriptanalitik tekniklere karşı iyi bir direnç göstermiştir [51]. Analizler, diferansiyel ve lineer karakteristiklerdeki aktif S-kutularının sayısı için alt sınırlar belirlemiş ve tam 31 turda pratik saldırıların, veri açısından uygulanabilir olmayan miktarlara ihtiyaç duyacağını kanıtlamıştır [51]. Bu, bu geleneksel kriptanalitik yöntemlere karşı güçlü bir güvenlik marjı olduğunu göstermektedir. Ayrıca, kelime benzeri yapılarla karşıt olan yapısal saldırılar, PRESENT'in çoğunlukla bit düzeyindeki tasarımı nedeniyle daha az uygun kabul edilmektedir. PRESENT'in basit cebirsel yapısı da analiz edilmiştir ve sonuçlar, cebirsel saldırıların önemli bir tehdit oluşturma olasılığının düşük olduğunu göstermektedir [51]. PRESENT'in anahtar planlaması, ilgili anahtar ve kayma saldırılarına karşı dirençli olacak şekilde tasarlanmış bir tur-bağımlı sayaç ve doğrusal olmayan bir işlem içerir. Anahtar planlamasının, tüm anahtar kayıt bitlerinin kullanıcı tarafından sağlanan anahtara tur 21'de doğrusal olmayan bağımlılığı gibi özelliklerinin, bu tür saldırılara karşı yeterli koruma sağladığı düşünülmektedir [51]. Ancak bazı analizler, PRESENT'in anahtar planlamasında, tur anahtarları arasındaki belirgin bağımlılığı işaret ederek, bunun yavaş ve tahmin edilebilir bit geçişlerine yol açabileceğini belirtmiştir [56]. Ayrıca, geliştirilmiş diferansiyel hata analizi, PRESENT'e karşı belgelenmiş olup, bu da hata enjekte saldırılarına karşı duyarlılığını göstermektedir [61]. Biclique kriptanalizi, PRESENT için kapsamlı anahtar aramasından biraz daha iyi sonuçlar göstermiştir. Bu, tam bir kırılma olmasa da, şifrenin güvenlik marjında küçük bir azalma olduğu anlamına gelmektedir.

Hafif blok şifrelerinin kaynak sınırlı ortamlarda uygulanması, enerji tüketimi, bellek ayak izi ve hız/verimlilik gibi birkaç faktörün dikkatlice değerlendirilmesini gerektirir [45]. Enerji verimliliği, özellikle batarya ile çalışan IoT cihazları için kritik öneme sahiptir ve araştırmalar, hafif kriptografik

algoritmaların enerji kullanımını analiz etmeye ve minimize etmeye odaklanmıştır. Şönt direnç analizi gibi teknikler kullanan çalışmalar, çeşitli hafif blok şifrelerinin enerji tüketimini değerlendirmiş ve bazı yeni öneriler, enerji tüketimi açısından mevcut algoritmalara kıyasla üstün performans sergilemiştir [45]. CMOS kapılarının enerji tüketimi modeli, döngü başına tüketilen enerji ile blok şifrelerinin açılmamış mimarilerindeki tur sayısı arasında ikili bir ilişki olduğunu göstermekte ve bu da tur sayısının enerji verimliliği üzerindeki etkisini vurgulamaktadır [66].

Bellek ayak izi, hem RAM hem de ROM gereksinimlerini kapsayan bir başka önemli kısıtlamadır. Farklı hafif algoritmaların RAM ve ROM kullanımı üzerine yapılan karşılaştırmalı analizler, algoritmaya ve platforma bağlı olarak önemli değişiklikler göstermiştir [47]. Özellikle, bazı çalışmalarda PRESENT ve CRAFT, en verimli RAM kullanan algoritmalar arasında yer almıştır [47]. Raspberry Pi 3 üzerinde yapılan doğrudan karşılaştırmada, LBlock, PRESENT'e kıyasla biraz daha yüksek RAM tüketimi ve daha yüksek ROM tüketimi göstermiştir [57]. Bu algoritmalar arasındaki seçim, hedef cihazın belirli bellek sınırlamalarına bağlı olabilir.

Hız ve verimlilik, gerçek zamanlı veri işleme gerektiren IoT uygulamaları için çok önemlidir. Raspberry Pi 3 üzerinde yapılan değerlendirme, LBlock'un PRESENT'ten daha yüksek verimlilik sağladığını göstermiştir [57]. Hafif şifrelerin boru hatlı (pipelined) uygulamaları, bazı durumlarda enerji tüketimini azaltarak daha yüksek verimlilik elde etmektedir [65]. Ancak, genellikle verimlilik ile diğer parametreler, örneğin donanım alanı ve enerji tüketimi arasında denge kurma gerekliliği bulunmaktadır [48]. Gerekli hız ve verimlilik, belirli uygulamaya bağlıdır ve bir şifrenin seçimi, bu ihtiyaçların diğer kaynak sınırlamalarıyla dengelenmesini içerebilir.

Bu temel değerlendirmelerin ötesinde, diğer uygulama yönleri de hafif blok şifrelerinin seçimini etkileyebilir. Bunlar arasında donanım ve yazılım uygulamaları arasındaki denge [25], şifrenin mimari tasarımının (Substitution-Permutation Network (SPN) ile Feistel ağı) uygulama verimliliği üzerindeki etkisi [14], yazılım performansını optimize etmek için kullanılan lookup tabloları [8], yazılım hızını artırmak için bit-sliced uygulamalarına uygunluk [14] ve donanım maliyetlerini azaltmak için birleşik şifreleme ve çözme devrelerinin kullanımı [6] yer alır. Bu çeşitli uygulama tekniklerini ve mimari seçimleri anlamak, hafif blok şifrelerinin farklı kaynak sınırlı ortamlarda etkili bir şekilde uygulanabilmesi için çok önemlidir.

Hafif kriptografi alanı dinamik bir alandır ve kaynak sınırlı cihazların gelişen güvenlik ihtiyaçlarını karşılamaya yönelik devam eden araştırmalar bulunmaktadır. Öne çıkan eğilimlerden biri, mevcut algoritmaların güvenliğini iyileştirme çabalarıdır. Örneğin, araştırmalar, PRESENT gibi algoritmalardaki zayıflıkları ele almaya devam etmektedir, bu da anahtar planı ve yayılma özelliklerinin güçlendirilmesine yönelik çalışmaları içermektedir [55]. DNA-PRESENT gibi PRESENT varyantları, hem güvenliği hem de performansı artırma potansiyeliyle araştırılmaktadır. Dinamik anahtar-bağımlı S-box'ların tanıtılması, şifre metninin rastgeleliğini iyileştirmeyi amaçlayan bir başka yaklaşımdır [56]. Bu çabalar, ortaya çıkan tehditlere karşı mevcut hafif algoritmaların güçlendirilmesi ve iyileştirilmesi amacını taşımaktadır.

Bir diğer önemli araştırma yönü, artırılmış güvenlik ve verimlilik sunabilecek yeni kriptografik tekniklerin keşfidir. Bu, şifreleme algoritmalarının temel özelliklerini iyileştirmek amacıyla DNA kriptografisi ve kaotik sistemler gibi geleneksel olmayan yöntemleri incelemeyi içermektedir [55]. Araştırmacılar ayrıca, yenilikçi yapısal tasarımlara sahip yeni hafif blok şifreleri geliştirmeye aktif olarak devam etmektedir [60]. Ayrıca, kuantum bilgisayarlarının artan beklentisiyle birlikte, IoT

cihazları için uzun vadeli güvenliği sağlamak amacıyla kuantum sonrası kriptografik çözümler geliştirmeye yönelik artan bir odak bulunmaktadır.

Mevcut araştırmalar ayrıca, hafif şifrelerin belirli güvenlik özelliklerinin iyileştirilmesine odaklanmaktadır. Bu, şifre metni üzerinde küçük değişikliklerin önemli değişikliklere yol açmasını sağlamak için kritik öneme sahip olan çığ etkisini iyileştirmeye yönelik bir odaklanmayı içermektedir. Yan kanal saldırılarına karşı direnç, kriptografik işlemler sırasında sızan bilgileri kullanan saldırılara karşı bir diğer araştırma alanıdır ve anahtarlamalı permütasyon gibi tekniklerin geliştirilmesi üzerinde çalışılmaktadır [55].

Hafif kriptografik algoritmaların titiz ve tutarlı bir şekilde değerlendirilmesi ihtiyacı, kapsamlı değerlendirme çerçevelerinin geliştirilmesine yönelik bir eğilime yol açmıştır. Bu çerçeveler, algoritmaları güvenlik, uygulama maliyeti (kapı eşdeğerleri açısından) ve performans metrikleri (örneğin, gecikme, verimlilik ve enerji verimliliği) gibi birden fazla parametre üzerinden değerlendirmeyi amaçlamaktadır [55]. Ayrıca, farklı hafif şifre adayı algoritmalarının adil karşılaştırmalarını kolaylaştırmak için karşılaştırma çerçeveleri geliştirilmektedir [49].

Son olarak, sürekli bir araştırma eğilimi, hafif kriptografinin kaynak sınırlı cihazların özel kısıtlamalarına optimize edilmesidir [55]. Bu, düşük donanım uygulama maliyetine sahip algoritmaların önceliklendirilmesini ve kriptografik çözümlerin, sağlık ve endüstriyel ortamlar gibi çeşitli IoT uygulamalarının benzersiz gereksinimlerine göre özelleştirilmesini içermektedir. Bu sürekli odaklanma, hafif kriptografinin kaynak sınırlı cihazların çeşitlenmiş ve evrilen manzarasında geçerliliğini ve etkinliğini korumasını sağlamaktadır.

Akademik literatür, hem LBlock hem de PRESENT'in farklı performans ve güvenlik özelliklerine sahip önemli hafif blok şifreleri olduğunu ortaya koymaktadır. Raspberry Pi 3 üzerinde yapılan karşılaştırmalı çalışmalar, LBlock'un genellikle daha düşük enerji tüketimi ve daha yüksek verimlilik sergilediğini, ancak biraz daha yüksek RAM ve ROM kullanımına sahip olduğunu göstermektedir [57]. Ancak, performans, belirli platform ve uygulama yapısına bağlı olarak değişkenlik gösterebilir. Donanım uygulama maliyetleri, kapı eşdeğerleri cinsinden ölçülmüş olup, optimizasyon hedeflerine ve anahtar boyutuna bağlı olarak küçük değişiklikler göstermektedir [51]. Güvenlik analizleri, her iki algoritmanın da titiz bir şekilde incelendiğini, LBlock'un DFA ve küp saldırılarına karşı bazı zayıflıklar sergilediğini, oysa PRESENT'in geleneksel kriptanalize karşı güçlü bir direnç gösterdiğini, ancak anahtar planında bazı zayıflıklar ve hata analizi saldırılarına karşı duyarlılık sergilediğini ortaya koymaktadır [58]. Uygulama dikkate alındığında, hedef uygulamanın gereksinimlerine göre enerji tüketimi, bellek ayak izi ve hız dengesinin sağlanmasının önemi vurgulanmaktadır [45]. Mevcut araştırma eğilimleri, mevcut algoritmaların güvenliğini artırmaya, yenilikçi teknikler keşfetmeye, belirli güvenlik özelliklerini iyileştirmeye, kapsamlı değerlendirme çerçeveleri geliştirmeye ve kaynak sınırlı cihazlar için optimizasyon yapmaya odaklanmaktadır [55].

LBlock ve PRESENT'in Python tabanlı bir performans karşılaştırma çalışması, bu mevcut bilgi birikimine yazılım performanslarına odaklanarak değerli bilgiler sağlayabilir. Böyle bir çalışma, Python bağlamında şifreleme ve şifre çözme hızı gibi performansın belirli yönlerini ve kaynak kullanımını (örneğin, CPU zamanı, bellek kullanımı) inceleyebilir. Python'un uygulama dili olarak seçilmesi önemlidir çünkü genellikle çeşitli IoT senaryolarında, özellikle daha yüksek seviyeli platformlarda prototipleme ve dağıtım için kullanılır. Bu hafif şifrelerin Python'da performansını değerlendirmek, donanım odaklı analizler ile pratik yazılım dağıtımları arasındaki boşluğu kapatmaya

yardımcı olabilir. Çalışma, teorik analizlerden veya donanım benchmark'larından belirgin olmayan, Python ortamına özgü performans özelliklerini ortaya çıkarabilir.

Python tabanlı bir performans karşılaştırma çalışmasının potansiyel bulguları, farklı uygulamalar için en uygun hafif şifrenin seçilmesi açısından pratik sonuçlar doğurabilir, özellikle Python tabanlı IoT platformları veya hızlı prototipleme gerektiren sistemler için. Sonuçlar, uygulayıcılara, Python ortamında LBlock ve PRESENT arasındaki yazılım performansı ve kaynak kullanımı açısından yapılacak tercihlerdeki dengeyi gösterebilir. Ayrıca, çalışma, her iki algoritma için yazılım düzeyinde optimizasyonların faydalı olabileceği alanları belirleyebilir. Elde edilen sonuçlara dayanarak, gelecekteki araştırmalar, bu ve diğer hafif şifrelerin farklı yüksek seviyeli programlama dillerindeki performanslarını keşfedebilir veya Python kütüphanelerinin ve uygulama tekniklerinin verimlilik üzerindeki etkisini inceleyebilir. Sonuç olarak, bu tür pratik değerlendirmeler, hafif kriptografi hakkında daha kapsamlı bir anlayışa katkı sağlar ve kaynak sınırlı cihazların sürekli genişleyen ortamında güvenli ve verimli dağıtımlar için bilinçli kararlar alınmasına yardımcı olur.

● Algoritmaların İncelenmesi

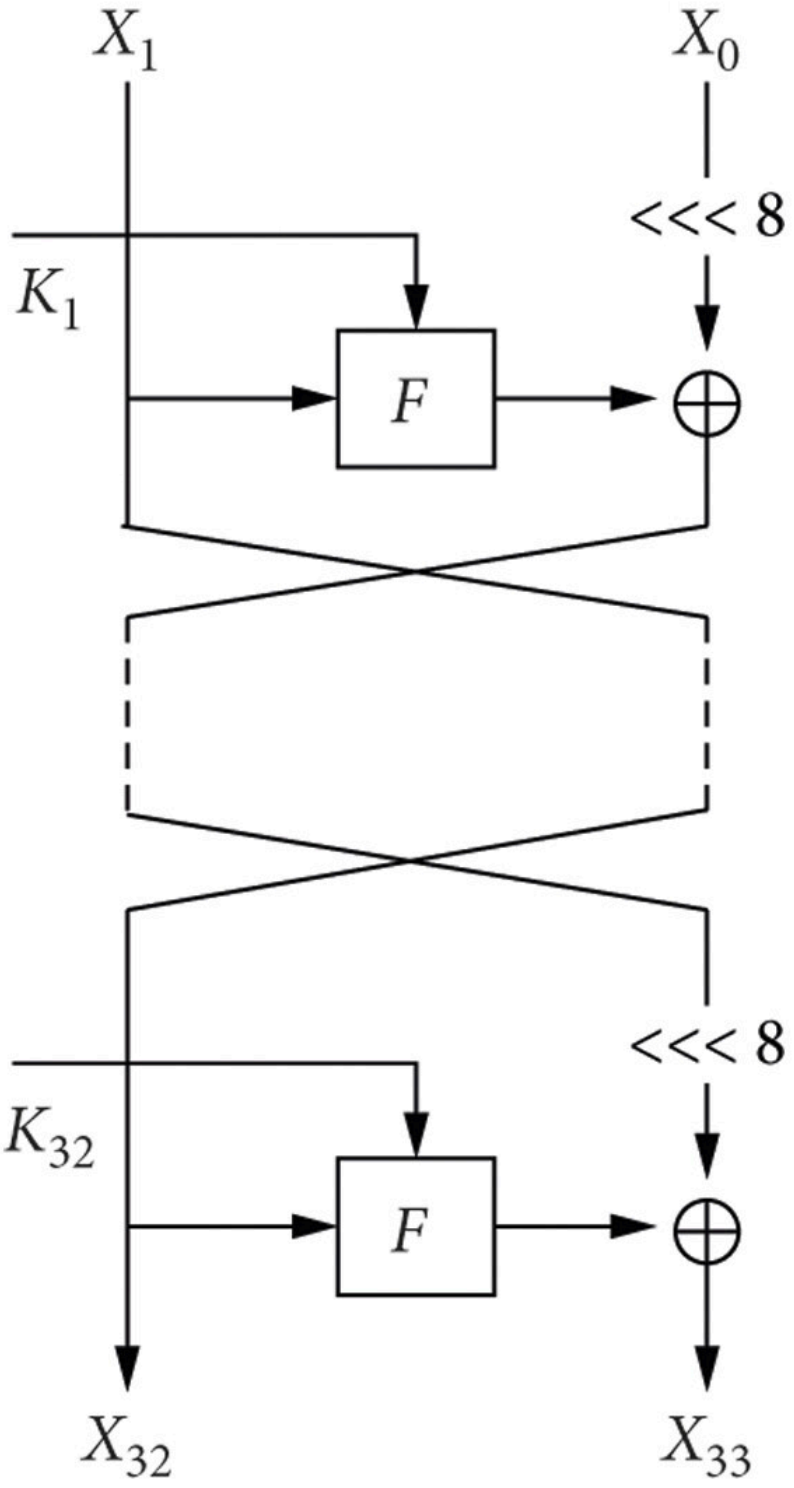
Nesnelerin İnterneti (IoT) alanındaki gelişmeler başta olmak üzere, kaynak kısıtlı cihazların sayısındaki artış, güvenlik ve verimlilik arasında bir denge sunan kriptografik algoritmaların geliştirilmesini zorunlu kılmaktadır. Geleneksel şifreleme algoritmaları, örneğin AES, yüksek güvenlik seviyeleri sunmalarına rağmen, sınırlı güç, işlem yetenekleri ve hafızaya sahip cihazlar için uygun olmayabilirler. Hafif kriptografi, bu zorlukların üstesinden gelmek için tasarlanmış algoritmalar sunarak, kaynakların az olduğu ortamlarda güvenli iletişimi ve veri korumasını mümkün kılmaktadır. Kablosuz sensör ağlarından RFID etiketlerine kadar geniş bir uygulama yelpazesinde bu tür algoritmalara ihtiyaç duyulmaktadır.

LBlock ve PRESENT, kaynak kısıtlı cihazlar için uygunlukları nedeniyle kriptoloji araştırmaları topluluğunda önemli ilgi gören hafif blok şifreleme algoritmalarının önde gelen örneklerindendir. Tasarımları, düşük uygulama maliyetlerini (donanım kapıları veya yazılım yürütme döngüleri açısından) önceliklendirirken, yeterli güvenlik seviyeleri sağlamayı amaçlamaktadır. IoT platformlarındaki uygulamalar bağlamında, farklı çalışmalar LBlock ve PRESENT'in performans ve güvenlik özelliklerini karşılaştırmıştır.

a. LBlock Algoritması

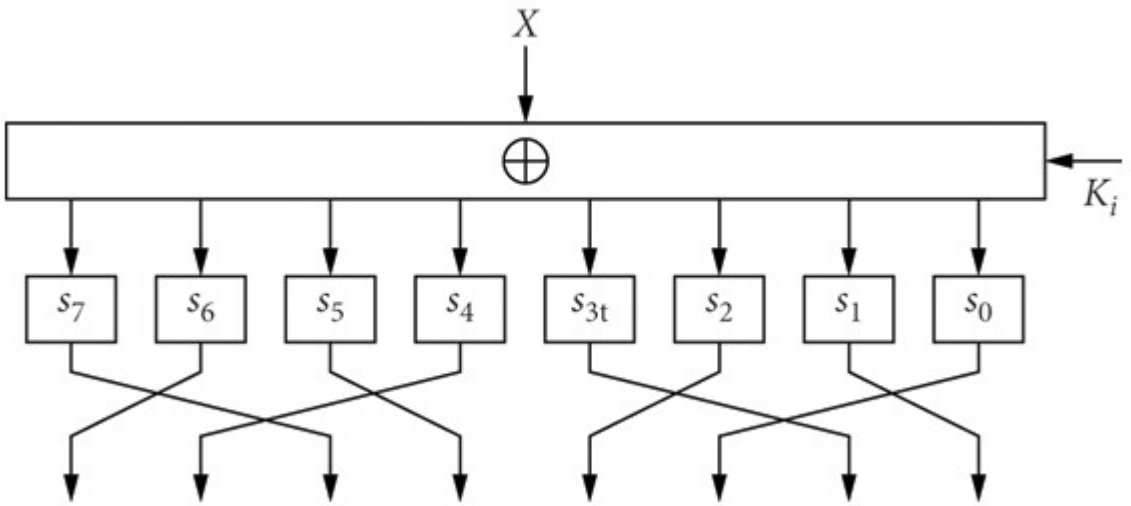
Çalışma Prensibi

LBlock, 64 bitlik düz metin blokları üzerinde çalışır ve 80 bitlik bir gizli anahtar kullanır. Blok şifrelemelerinde yaygın bir tasarım ilkesi olan Feistel benzeri bir yapı kullanır. Bir Feistel ağında, veri bloğu iki yarıya bölünür ve tur fonksiyonu bir yarıya uygulanır, sonuç daha sonra diğer yarı ile birleştirilir. Bu işlem belirli sayıda tur boyunca tekrarlanır. LBlock, 32 turdan oluşur. Algoritmanın yinelemeli yapısı, düz metnin yüksek düzeyde güvenlik elde etmek için birden çok dönüşümden geçmesini sağlar. LBlock'un Feistel yapısı, bazı senaryolarda uygulamayı basitleştirerek, aynı tur fonksiyonunun tersine çevrilmiş bir anahtar takvimi ile hem şifreleme hem de şifre çözme için kullanılmasına olanak tanır. Bu yapıdaki doğal simetri, hem şifreleme hem de şifre çözme işlevlerinin gerekli olduğu donanım ve yazılım uygulamaları için avantajlı bir özelliktir.



Şekil 1. *LBlock Şifreleme Tur Yapısı*

(https://onlinelibrary.wiley.com/doi/10.1155/2020/8837671)



Şekil 2. *LBlock F Fonksiyonu*

(https://onlinelibrary.wiley.com/doi/10.1155/2020/8837671)

Evreleri ve Matematiksel İfadeler

LBlock'taki şifreleme süreci aşağıdaki evrelere ayrılabilir:

- **Başlangıç:** 64 bitlik düz metin bloğu *M*, *X1* (sol yarı) ve *X0* (sağ yarı) olarak adlandırılan iki adet 32 bitlik yarıya ayrılır. Matematiksel olarak, *M = X1 | | X0*.
- **Tur Fonksiyonu:** LBlock, 2'den 33'e kadar her *i* turu için aşağıdaki işlemi gerçekleştirerek 32 tur boyunca yineleme yapar :
    - *Xi = F(Xi-1, Ki-1) ⊕ (Xi-2 <<< 8)*
- Burada *F*, tur fonksiyonunu, *Ki-1*, *i-1* turu için 32 bitlik tur alt anahtarını ve *<<< 8*, 8 bitlik sola döngüsel kaydırmayı ifade eder. *F(X, K)* tur fonksiyonu kendi içinde birkaç adım içerir:
    - 32 bitlik girdi *X*, tur alt anahtarı *K* ile XOR işlemine tabi tutulur.
    - Elde edilen sonuç daha sonra bir S-kutusu katmanından *S* geçirilir. Bu katman, sekiz paralel 4x4 S-kutusundan oluşur. S-kutusu katmanına 32 bitlik girdinin

      Z = Z7 | | Z6 | | Z5 | | Z4 | | Z3 | | Z2 | | Z1 | | Z0

      olduğu varsayılır, burada her *Zj* 4 bitlik bir kelimedir. *S(Z)* çıktısı daha sonra

      s7(Z7) | | s6(Z6) | | s5(Z5) | | s4(Z4) | | s3(Z3) | | s2(Z2) | | s1(Z1) | | s0(Z0)

      olur, burada *sj*, *j*-inci 4 bitlik S-kutusunu temsil eder. Bu S-kutularına örnekler [32] verilmiştir.
    - S-kutusu katmanının çıktısı daha sonra bir permütasyon katmanına *P* tabi tutulur. Bu permütasyon, sekiz adet 4 bitlik kelime üzerinde çalışır :
        - Permütasyona girdinin

$Z = Z7 \,||\, Z6 \,||\, Z5 \,||\, Z4 \,||\, Z3 \,||\, Z2 \,||\, Z1 \,||\, Z0$

olduğu varsayılırsa, çıktı U,

$U = U7 \,||\, U6 \,||\, U5 \,||\, U4 \,||\, U3 \,||\, U2 \,||\, U1 \,||\, U0$

olarak verilir, burada

$U7 = Z6,\ U6 = Z4,\ U5 = Z7,\ U4 = Z5,\ U3 = Z2,\ U2 = Z0,\ U1 = Z3,\ U0 = Z1$.

- **Sonuç:** 32 turdan sonra, elde edilen son iki adet 32 bitlik yarı (*X32* ve *X33*) birleştirilerek 64 bitlik şifreli metin *C* elde edilir.

Tur fonksiyonundaki doğrusal olmayan S-kutusu katmanı ve doğrusal permütasyon katmanının birleşimi, hem karıştırma (anahtar ve şifreli metin arasındaki ilişkiyi karmaşıklaştırma) hem de yayılma (her bir düz metin bitinin etkisini şifreli metin boyunca yayma) elde etmek için kritik öneme sahiptir. Claude Shannon tarafından tanımlanan bu iki özellik, modern blok şifrelerinin güvenliği için temeldir. *Xi-2*'ye XOR işleminden önce uygulanan 8 bitlik sola döngüsel kaydırma, farklı turlar arasında daha fazla karıştırma ve bağımlılık sağlayarak algoritmanın belirli kriptanalitik saldırılara karşı direncini artırır. Turlar arası bu geri besleme mekanizması, bireysel turların etkisini izole etmeyi ve analiz etmeyi zorlaştırarak genel güvenliğe katkıda bulunur.

Tasarım Gerekçeleri

LBlock'un tasarımı, kaynak kısıtlı cihazlar için hafif bir uygulama önceliği taşırken, yeterli bir güvenlik seviyesini de korur. Feistel yapısının seçimi, nispeten basit bir tur fonksiyonuna ve verimli donanım ve yazılım uygulamaları olasılığına olanak tanır. 4x4 S-kutuları, kısıtlı ortamlar için uygun küçük bir ayak iziyle doğrusal olmama özelliği sağlamak üzere seçilmiştir. Paralel olarak sekiz farklı S-kutusunun kullanılması, substitüsyon katmanının güvenlik özelliklerini geliştirmeyi amaçlar. Kelime tabanlı permütasyon katmanı *P*, birkaç tur içinde iyi bir yayılma sağlamak üzere tasarlanmıştır, bu da düz metindeki değişikliklerin hızla şifreli metnin büyük bir bölümünü etkilemesini sağlar. Belirli permütasyonun, hem şifreleme hem de şifre çözme yönlerinde 8 turda en iyi yayılmayı sağlayacak şekilde dikkatlice seçildiği düşünülmektedir. Güvenlik değerlendirmeleri, LBlock'un diferansiyel kriptanaliz, doğrusal kriptanaliz, imkansız diferansiyel kriptanaliz ve ilişkili anahtar saldırıları gibi bilinen saldırılara karşı makul bir güvenlik marjı sağladığını göstermiştir. Örneğin, 15 turluk bir LBlock'ta en az 32 aktif S-kutusunun bulunduğu gösterilmiştir, bu da diferansiyel saldırılara karşı direncine katkıda bulunur. LBlock'un tasarımı, kaynak kullanımını en aza indirme ve güvenliği en üst düzeye çıkarma arasındaki bir ödünleşmeyi temsil eder. Belirli parametreler (tur sayısı, S-kutusu tasarımı, permütasyon) muhtemelen bu ödünleşmelerin kapsamlı bir analizine dayanarak seçilmiştir. Kriptografik tasarım, amaçlanan uygulamalar için istenen dengeyi elde etmek üzere çeşitli faktörlerin dikkatli bir şekilde değerlendirilmesini içeren yinelemeli bir süreçtir.

Anahtar Takvimi ve Matematiksel İfadeler

80 bitlik ana anahtar *K*, 32 tur için 32 bitlik tur alt anahtarları *K1, K2,..., K32* üretmek için kullanılır. Anahtar takvimi aşağıdaki gibi ilerler:

1. İlk tur anahtarı *K1*, ana anahtar *K*'nın en soldaki 32 bitidir.

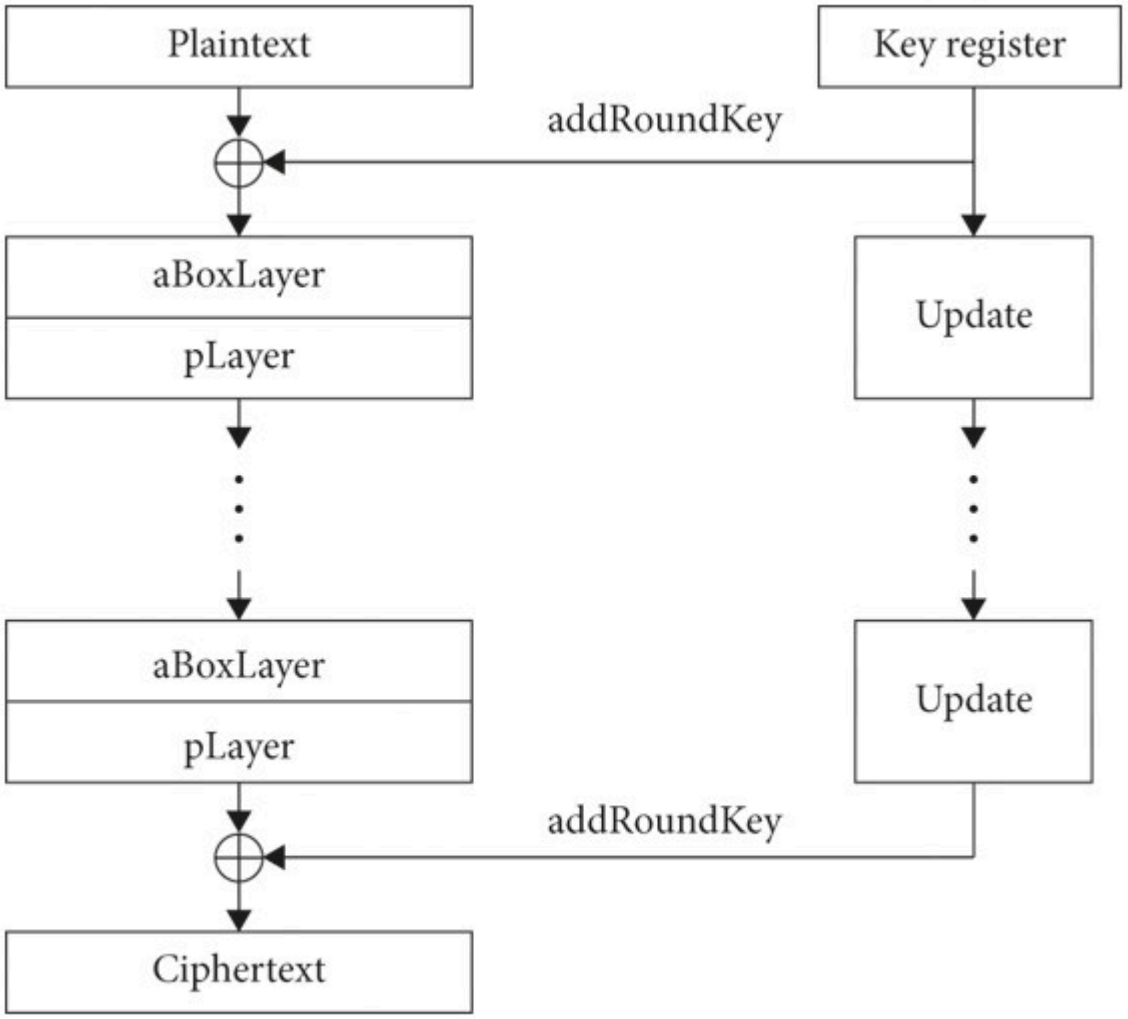
2. 1'den 31'e kadar her *i* için:
    - 80 bitlik anahtar *K*, 29 bit sola döngüsel olarak kaydırılır.
    - *K*'nın en soldaki 4 biti *s9* S-kutusu ile ve sonraki 4 biti *s8* S-kutusu ile işlenir. Bunlar, anahtar takviminde kullanılan belirli 4 bitlikten 4 bitliğe S-kutularıdır.
    - *K*'nın belirli 5 biti (en soldaki bitin 79 olduğu varsayılarak 62, 61, 60, 59, 58. bitler), tur numarasının *i*'nin 5 bitlik ikilik gösterimi olan [*i*]2'nin belirli 5 biti ile XOR işlemine tabi tutulur.
    - Güncellenen 80 bitlik *K*'nın en soldaki 32 biti, bir sonraki tur anahtarı *Ki+1* olarak alınır.

Anahtar takvimi, her turun ana anahtardan türetilen farklı bir alt anahtar kullanmasını sağlayarak, birden çok turda aynı anahtarın kullanılmasını istismar edebilecek basit saldırıları önler. Anahtar takvimine S-kutusu işlemlerinin ve tur sayacının dahil edilmesi, doğrusal olmama özelliği ekler ve gelecekteki tur anahtarlarının önceki anahtarlardan doğrudan türetilmesini engeller. Güçlü bir anahtar takvimi, saldırganların tur anahtarları dizisini kolayca tahmin etmesini önlediği için bir blok şifresinin genel güvenliği için hayati öneme sahiptir.

b. PRESENT Algoritması

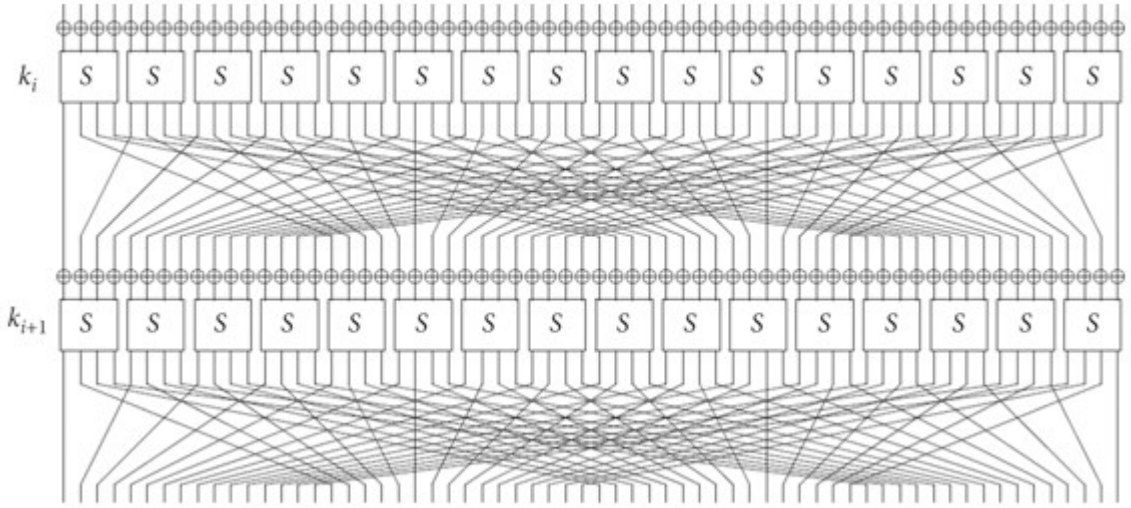
Çalışma Prensibi

PRESENT, 64 bitlik düz metin bloklarını işleyen ve 80 bitlik veya 128 bitlik gizli anahtarları destekleyen ultra hafif bir blok şifreleme algoritmasıdır. 80 bitlik anahtar sürümü, genellikle sıkı kaynak kısıtlamalarına sahip uygulamalar için tercih edilir. LBlock'un aksine, PRESENT bir Substitüsyon-Permütasyon (SP) ağı mimarisi kullanır. Bir SP ağında, tüm veri bloğu her turda bir dizi substitüsyon ve permütasyon işleminden geçer. PRESENT, 31 turdan oluşur ve ardından son bir anahtar ekleme (beyazlatma) işlemi gelir. SP ağı yapısı, veri bloğunun paralel olarak işlenmesine olanak tanır, bu da donanım uygulamalarında hız açısından avantajlı olabilir. Tüm bloğa aynı anda işlemler uygulayarak, SP ağları, her turda bloğun yalnızca yarısının işlendiği Feistel ağlarına kıyasla potansiyel olarak daha yüksek verim elde edebilir. PRESENT, özellikle donanım açısından son derece kısıtlı ortamlar (RFID etiketleri, kablosuz sensör ağları vb.) için tasarlanmış, çok **hafif (ultra-lightweight)** bir blok şifreleme algoritmasıdır. 2007 yılında Bogdanov ve arkadaşları tarafından geliştirilmiştir.



Şekil 3. *PRESENT Şifreleme Algoritması*

(https://onlinelibrary.wiley.com/doi/10.1155/2020/8837671)



Şekil 4. *PRESENT Şifreleme Tur Yapısı*

(https://onlinelibrary.wiley.com/doi/10.1155/2020/8837671)

Evreleri ve Matematiksel İfadeler

PRESENT'taki şifreleme süreci, 31 tur boyunca tekrarlanan aşağıdaki adımları içerir:

- **addRoundKey:** 64 bitlik durum $b = b63...b0$, 1'den 31'e kadar her i için 64 bitlik bir tur anahtarı

  $Ki = \kappa i63...\kappa i0$

  ile XOR işlemine tabi tutulur. Matematiksel olarak,

  $0 \leq j \leq 63$ için $bj \leftarrow bj \oplus \kappa ij$.

- **sBoxLayer:** 64 bitlik durum,

  $wi = b4$i+3 | | b4$i+2$ | | $b4$i+1 | | b4i

  olacak şekilde 16 adet 4 bitlik kelimeye $w15...w0$ ayrılır. Her 4 bitlik kelime daha sonra aynı 4 bitlikten 4 bitliğe S-kutusu S aracılığıyla paralel olarak substitüe edilir. PRESENT'ta kullanılan S-kutusu (onaltılık gösterimde) aşağıdaki gibidir:

  x |0 |1 | 2 | 3 |4 | 5 |6 |7 |8 |9 |A |B |C |D | E | F

  --|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---

  S[x]| C | 5 | 6 | B | 9 | 0 | A | D | 3 | E | F | 8 | 4 | 7 | 1 | 2

  Güncellenmiş durumun nibble'ları daha sonra $S[w15]...S[w0]$ olur.

- **pLayer:** S-kutusu katmanının 64 bitlik çıktısı, sabit bir bit permütasyonuna P tabi tutulur. i-inci bit, $P(i)$ konumuna taşınır. Permütasyon tablosu 'te verilmiştir.

31 turdan sonra, 32. tur anahtarı $K32$ ile son bir **addRoundKey** işlemi gerçekleştirilir. Her turda aynı S-kutusunun 16 nibble boyunca tutarlı bir şekilde kullanılması, PRESENT'ın donanım uygulamasını basitleştirerek ultra hafif yapısına katkıda bulunur. Aynı bileşeni donanımda birden çok kez çoğaltmak, genellikle birkaç farklı bileşen uygulamaktan daha verimli olabilir. pLayer'daki bit düzeyinde permütasyon, tüm 64 bitlik durum boyunca kapsamlı bir karıştırma sağlayarak güçlü bir yayılma sağlar. Bitleri bireysel düzeyde yeniden düzenleyerek, permütasyon katmanı, her bir girdi bitinin etkisini çok sayıda farklı çıktı bitine çok sayıda tur boyunca etkili bir şekilde yayar.

Tasarım Gerekçeleri

PRESENT'ın temel tasarım amacı, RFID etiketleri ve sensör düğümleri gibi en kaynak kısıtlı cihazlar için uygun, son derece küçük bir donanım ayak izi elde etmektir. SP ağı yapısının basitliği ve tek, küçük bir S-kutusunun kullanımı, donanım verimliliğine katkıda bulunur. Bit permütasyon

katmanı da donanımda basit tel geçişleri olarak verimli bir şekilde uygulanabilir. PRESENT, amaçlanan uygulamaları için yeterli güvenlik sağlamayı amaçlar. 80 bitlik anahtar sürümü, tipik olarak etiket tabanlı dağıtımlarda bulunan düşük güvenlikli uygulamalar için yeterli kabul edilir. Tasarım, donanım verimliliğiyle bilinen DES ve AES finalisti Serpent gibi şifrelerden ilham almıştır. Ultra hafif uygulamaya odaklanılması nedeniyle, PRESENT'taki S-kutusu, potansiyel olarak daha güçlü güvenlik özelliklerinden ziyade öncelikle küçük donanım alanı için seçilmiştir. Bu, diğer hafif şifrelere kıyasla bir ödünleşme olabilir. PRESENT'ın tasarımı, birincil hedef uygulamalarının gömülü sistemler ve çok sınırlı kaynaklara sahip cihazlar olması nedeniyle, yazılım performansından ziyade donanım verimliliğini güçlü bir şekilde vurgular. Donanım maliyetinin ve güç tüketiminin en önemli olduğu ortamlarda, bu faktörler için optimizasyon, yazılım yürütme hızına göre öncelik taşır.

Anahtar Takvimi ve Matematiksel İfadeler

80 bitlik anahtar sürümü için, PRESENT'ın anahtar takvimi, 80 bitlik kullanıcı anahtarı $K = k79k78...k0$'dan 32 adet 64 bitlik tur anahtarı $K1, K2,..., K32$ üretir :

1. 1'den 32'ye kadar her i için:
    - 80 bitlik anahtar K, 61 bit sola döngüsel olarak kaydırılır.
    - K'nın en soldaki 4 biti (79-76. bitler) PRESENT S-kutusu aracılığıyla geçirilir.
    - K'nın 19, 18, 17, 16 ve 15. konumlarındaki (soldan, 79'dan başlayarak) 5 biti, sağdaki en düşük anlamlı bit olacak şekilde 5 bitlik tur sayacı i ile XOR işlemine tabi tutulur.
    - K'nın mevcut 64 biti (79-16. bitler), tur anahtarı Ki olarak alınır.

128 bitlik anahtar sürümü, daha büyük anahtar boyutu için uygun ayarlamalar içeren benzer adımlara sahiptir ve anahtar takviminin her turunda iki S-kutusu uygulaması içerir. LBlock'a benzer şekilde, PRESENT anahtar takvimi de döngüsel kaydırmalar ve S-kutusu aramaları gibi verimli işlemler içerir. Tur sayacı ile XOR işlemi, farklı tur anahtarlarının üretilmesini sağlayarak, anahtar tekrarını istismar edebilecek saldırılara karşı güvenliği artırır. Daha büyük kaydırma değeri (61 bit), anahtar bitlerinin daha hızlı karışmasına katkıda bulunur. Hafif bir şifre için iyi tasarlanmış bir anahtar takvimi hem güvenli hem de hesaplama açısından ucuz olmalıdır.

c. LBlock ve PRESENT Algoritmalarının Karşılaştırılması

Mimari ve Operasyonel Farklılıklar

LBlock, 32 turlu bir Feistel ağ yapısı kullanırken, 64 bitlik veri bloğu iki adet 32 bitlik yarıya bölünür ve tur fonksiyonu bir yarı üzerinde çalışır. Öte yandan PRESENT, 31 turlu bir SP ağ yapısı kullanır ve tur fonksiyonu tüm 64 bitlik veri bloğuna uygulanır. LBlock'taki S-kutusu katmanı paralel olarak kullanılan sekiz farklı 4x4 S-kutusundan oluşurken, PRESENT aynı 4x4 S-kutusunu on altı kez paralel olarak kullanır. LBlock'taki permütasyon katmanı 4 bitlik kelime düzeyinde çalışırken, PRESENT bit düzeyinde bir permütasyon kullanır. LBlock'un tur fonksiyonu, S-kutusu katmanından önce tur alt anahtarıyla bir XOR işlemi içerirken, PRESENT her turun başında tur anahtarını XOR ile birleştirir. Feistel ve SP ağ yapısı arasındaki seçim genellikle farklı optimizasyon hedeflerini yansıtır. Feistel ağları, belirli tur fonksiyonları için alan verimliliği açısından avantajlar sunabilirken, SP ağları paralel işlem nedeniyle donanımda daha iyi yayılma ve daha yüksek verim sağlayabilir. Bu mimari

farklılıklar, şifrelerin performans özelliklerini ve güvenlik özelliklerini etkiler, bu da belirli uygulama gereksinimlerine bağlı olarak birini diğerinden daha uygun hale getirir.

Güvenlik ve Performans Analizi

Hem LBlock hem de PRESENT hafif olacak şekilde tasarlanmıştır, ancak farklı performans özellikleri sergilerler. PRESENT genellikle daha ultra hafif olarak kabul edilir ve sıklıkla daha küçük donanım ayak izleri elde eder. LBlock, hafif olmasına rağmen, bazı yazılım uygulamalarında biraz daha iyi performans veya farklı bir güvenlik profili sunarak farklı bir denge sağlayabilir. Güvenlik analizleri, her iki algoritmanın da çeşitli kriptanalitik saldırılara karşı yeterli güvenlik marjı sağladığını göstermektedir. Ancak bazı çalışmalar , PRESENT'ın basit tasarımının diğer şifrelere kıyasla daha yavaş yayılmaya yol açabileceğini belirtmiştir. Ek olarak, PRESENT'ın S-kutusu öncelikle donanım verimliliği için seçilmiştir, bu da belirli gelişmiş saldırılara karşı direnci açısından etkileri olabilir. IoT platformlarındaki karşılaştırmalar , her iki algoritmanın da bu alandaki pratik alaka düzeyini gösteriyor, ancak belirli performans ve güvenlik ödünleşmelerinin IoT cihazlarının kısıtlamalarına göre dikkatlice değerlendirilmesi gerekiyor. PRESENT'ın "ultra hafif" doğası, LBlock'a kıyasla yayılma hızı ve S-kutusu gücü açısından potansiyel ödünleşmelerle birlikte gelir; LBlock, kaynak kullanımı ve güvenlik sağlamlığı arasında biraz farklı bir dengeye öncelik verebilir. Aşırı kaynak kısıtlı ortamlarda, donanım maliyetini en aza indirmek genellikle öncelik taşır, bu da belirli güvenlik ödünleşmeleri olan daha basit bileşenlere yol açabilir ve bunların belirli uygulama için dikkatlice değerlendirilmesi gerekir.

Tipik Kullanım Senaryoları

PRESENT, minimum donanım ayak izine yaptığı vurgu ile özellikle RFID etiketleri, sensör ağları ve donanım maliyeti ve güç tüketiminin kritik sınırlamalar olduğu diğer uygulamalar gibi son derece kaynak kısıtlı cihazlar için çok uygundur. LBlock, hafif olmasına rağmen, biraz daha fazla kaynak mevcut olduğunda veya farklı bir performans ve güvenlik dengesi istendiğinde tercih edilebilir. RFID etiketlerine kıyasla biraz daha fazla işlem gücü veya belleğe sahip kablosuz sensör düğümleri gibi uygulamalar için uygun olabilir. LBlock ve PRESENT arasındaki seçim nihayetinde, kaynak kısıtlamalarının düzeyi, istenen güvenlik seviyesi ve performans hedefleri dahil olmak üzere uygulamanın özel gereksinimlerine bağlıdır. "Ultra hafif" ve "hafif" arasındaki ayrım, bir kaynak optimizasyonu spektrumunu gösterir; PRESENT bu spektrumun alt ucunu hedeflerken, LBlock biraz daha yüksek bir konumda yer alır ve potansiyel olarak farklı bir ödünleşme kümesi sunar. Bu spektrumu anlamak, benzersiz kaynak sınırlamaları ve güvenlik ihtiyaçları olan belirli bir uygulama için en uygun algoritmayı seçmek için çok önemlidir.

Tablo 3. *LBlock ve PRESENT'ın Temel Özellikleri*

Özellik	LBlock	PRESENT
Blok Boyutu	64 bit	64 bit
Anahtar Boyutu(ları)	80 bit	80 bit, 128 bit
Tur Sayısı	32	31
Yapı	Varyant Feistel Ağı	SP-Ağı
S-Kutusu Boyutu	4x4 (8 farklı paralel)	4x4 (16 aynı paralel)
Permütasyon	4-bit kelime seviyesinde	Bit seviyesinde

Sonuç

LBlock ve PRESENT, kaynak kısıtlı ortamlarda güvenlik sağlamak üzere tasarlanmış iki önemli hafif blok şifreleme algoritmasıdır. Feistel yapısı ve güvenlik ile verimlilik arasında bir dengeye odaklanmasıyla LBlock, çeşitli uygulamalar için sağlam bir çözüm sunar. SP ağı ve ultra hafif uygulamaya yaptığı vurgu ile PRESENT, özellikle en kaynak kısıtlı cihazlar için uygundur. Bu algoritmalar arasındaki seçim, uygulamanın özel kısıtlamalarına ve gereksinimlerine bağlıdır. Mevcut donanım kaynakları, istenen güvenlik düzeyi ve performans hedefleri gibi faktörlerin dikkatlice değerlendirilmesi gerekir. Hafif kriptografi alanındaki devam eden araştırmalar, kaynak kısıtlı cihazların gelişen güvenlik ihtiyaçlarını karşılamak için yeni tasarım ilkelerini ve optimizasyonları keşfetmeye devam etmektedir. LBlock ve PRESENT gibi algoritmaların daha fazla analizi ve karşılaştırılması, alanı ilerletmek ve giderek daha fazla birbirine bağlı hale gelen dünyada güvenli iletişimi ve veri işlemeyi sağlamak için çok önemlidir.

● Python Uygulaması

Geliştirilen bir uygulama aracılığıyla LBlock ve PRESENT hafif blok şifreleme algoritmalarının performansını karşılaştırmak amaçlamıştır. Uygulama, her iki algoritma için şifreleme ve deşifreleme işlemlerini gerçekleştirmekte ve bu işlemlerin süresini ve bellek kullanımındaki değişimini ölçmektedir. Karşılaştırma, belirtilen bir veri seti üzerinde yapılan testlerin sonuçlarına dayanmaktadır.

a. Uygulama Detayları

Geliştirilen uygulama, birkaç Python modülünün birleşiminden oluşmaktadır:

- **main.py:** Kullanıcı arayüzünü yöneten ve ana iş akışını kontrol eden ana betik.
    - Kullanıcıya metin tabanlı bir menü sunar (Dosya Şifrele, Dosya Deşifrele, Çıkış).
    - Kullanıcıdan hangi algoritmanın (LBlock veya PRESENT) kullanılacağını seçmesini ister.
    - İşlem yapılacak giriş ve sonuçların kaydedileceği çıkış dosyalarını seçmek için tkinter kütüphanesi aracılığıyla grafiksel dosya seçme diyalogları kullanır. (Bu, uygulamanın grafiksel bir ortamda çalıştırılmasını gerektirir).
    - Gerekli anahtarın (her iki algoritma için 80-bit) kullanıcı tarafından hexadecimal formatta girilmesini sağlar ve doğrular. (Dosya içinde Keys.txt adında bir dosya anahtar seçimi için hazırlanmıştır.)
    - Seçilen algoritma ve işleme göre ilgili fonksiyonları (encrypt_lblock, decrypt_lblock, encrypt_present, decrypt_present) çağırır.
    - Performans ölçümünü (utils.measure_performance) tetikler.
    - Sonucu çıkış dosyasına yazar ve ölçülen performansı tablo formatında konsola basar (utils.display_performance_table).
- **lblock.py:** LBlock algoritmasının 80-bit anahtar ve 64-bit blok boyutlu versiyonunu uygular.
    - Algoritmanın sabitlerini (S-Box'lar, P-Permütasyonu), anahtar planını (key_schedule_lblock), round fonksiyonunu (function_F_lblock) ve blok

şifreleme/deşifreleme (encrypt_block_lblock, decrypt_block_lblock) mantığını içerir (Makaleden yararlanılmıştır).
    - Byte dizileri üzerinde çalışan ana şifreleme/deşifreleme fonksiyonlarını (encrypt_lblock, decrypt_lblock) sunar.
- **present.py:** PRESENT algoritmasının 80-bit anahtar ve 64-bit blok boyutlu versiyonunu uygular.
    - Algoritmanın sabitlerini (S-Box, P-Layer), anahtar planını (key_schedule_80_present), round adımlarını (AddRoundKey, S-Box, P-Layer) ve blok şifreleme/deşifreleme (encrypt_block_present, decrypt_block_present) mantığını içerir (Makaleden yararlanılmıştır).
    - Byte dizileri üzerinde çalışan ana şifreleme/deşifreleme fonksiyonlarını (encrypt_present, decrypt_present) sunar.
- **utils.py:** Yardımcı fonksiyonları ve performans ölçüm altyapısını sağlar.
    - Byte/integer/hex dönüşümleri, dosya okuma/yazma, byte formatlama gibi genel araçlar içerir.
    - measure_performance fonksiyonu ile çekirdek kriptografik işlemin süresini (time.time) ve bellek kullanımındaki farkı (psutil ile RSS ölçümü) hesaplar.
    - display_performance_table ile ölçüm sonuçlarını formatlı bir şekilde sunar.
- **Key_exchange.py**
    - **generate_rsa_key_pair() (Alice):** Alice için bir RSA özel anahtar ve buna karşılık gelen bir genel anahtar çifti oluşturur. key_size=2048 bit, günümüzde RSA için kabul edilebilir bir güvenlik seviyesidir.
    - **serialize_public_key_to_pem() (Alice):** Alice'in RSA genel anahtarını PEM (Privacy Enhanced Mail) formatına dönüştürür. Bu format, anahtarların metin tabanlı olarak saklanması ve iletilmesi için standart bir yoldur.
    - **deserialize_pem_to_public_key() (Bob):** Bob, Alice'ten aldığı PEM formatındaki genel anahtar metnini tekrar kullanılabilir bir RSA genel anahtar nesnesine dönüştürür.
    - **generate_symmetric_key() (Bob):** Bob, LBlock veya PRESENT gibi simetrik şifreleme algoritmalarında kullanılacak olan asıl anahtarı üretir. Bu örnekte, 80-bit (10 byte) uzunluğunda rastgele bir anahtar üretilir.
    - **encrypt_symmetric_key_with_rsa() (Bob):** Bob, ürettiği 80-bitlik simetrik anahtarı, Alice'ten aldığı RSA genel anahtarı ile şifreler. **OAEP (Optimal Asymmetric Encryption Padding):** RSA ile küçük verileri (örneğin bir simetrik anahtar) şifrelerken kullanılan güvenli bir dolgulama şemasıdır. Güvenliği artırır.
    - **decrypt_symmetric_key_with_rsa() (Alice):** Alice, Bob'dan gelen şifrelenmiş simetrik anahtarı, kendi RSA özel anahtarını kullanarak çözer. Yine OAEP dolgulaması kullanılır.
    - **simulate_key_exchange():** Yukarıdaki fonksiyonları kullanarak tüm anahtar değişim sürecini adım adım simüle eder. Alice RSA anahtarlarını üretir, genel anahtarını Bob'a "gönderir". Bob simetrik anahtarı üretir, Alice'in genel anahtarı ile şifreler ve Alice'e "gönderir". Alice şifreli mesajı çözer. Son olarak, Bob'un orijinalde ürettiği simetrik anahtar ile Alice'in çözdüğü anahtarın aynı olup olmadığını kontrol eder.

Önemli Uygulama Seçimleri ve Kısıtlamalar:

- **Çalışma Modu (Mode of Operation):** Her iki algoritma da **ECB (Electronic Codebook)** modunda uygulanmıştır. ECB modunda, her veri bloğu bağımsız olarak aynı anahtarla

şifrelenir/deşifre edilir. Bu mod, basitliği nedeniyle bazen eğitim amaçlı kullanılsa da, **güvensizdir**. Aynı düz metin blokları aynı şifreli metin bloklarını üreteceğinden, veri içindeki desenleri korur ve sızdırır.

- **Padding (Dolgu):** Şifreleme sırasında, eğer girdi verisi blok boyutunun (8 byte) tam katı değilse, sonuna eksik byte sayısı kadar **boşluk karakteri (ASCII 32, 0x20)** eklenmektedir. Bu, standart bir padding şeması (örn. PKCS#7) **değildir**. Deşifreleme fonksiyonları bu eklenen padding'i **otomatik olarak kaldırmaz**. Bu durum, orijinal verinin sonunda boşluk karakterleri varsa veri kaybına veya bozulmasına yol açabilir ve orijinal veri uzunluğunun kesin olarak kurtarılmasını engeller.
- **Bağımlılıklar:** Uygulama, bellek ölçümü için harici psutil kütüphanesine ve dosya seçimi için standart tkinter kütüphanesine bağımlıdır. psutil kurulu değilse veya grafiksel ortam (tkinter için) mevcut değilse uygulama hata verecektir.

b. Test Ortamı ve Veri Seti

Performans testlerinin sonuçları, testlerin yapıldığı ortamdan önemli ölçüde etkilenmektedir. Bu rapor için spesifik test ortamı detayları sağlanarak, ideal bir raporda aşağıdaki bilgilerin yer alması sağlanmıştır:

- **Donanım:**
  - CPU: 12th Gen Intel(R) Core(TM) i7-12700H, 2700 Mhz, 14 Core(s), 20 Logical Processor(s)
  - RAM: 32 GB
- **Yazılım:**
  - İşletim Sistemi: Microsoft Windows 11 Home Single Language
  - Python Sürümü: Python 3.12.10
  - Kütüphane Sürümleri: psutil : 7.0.0

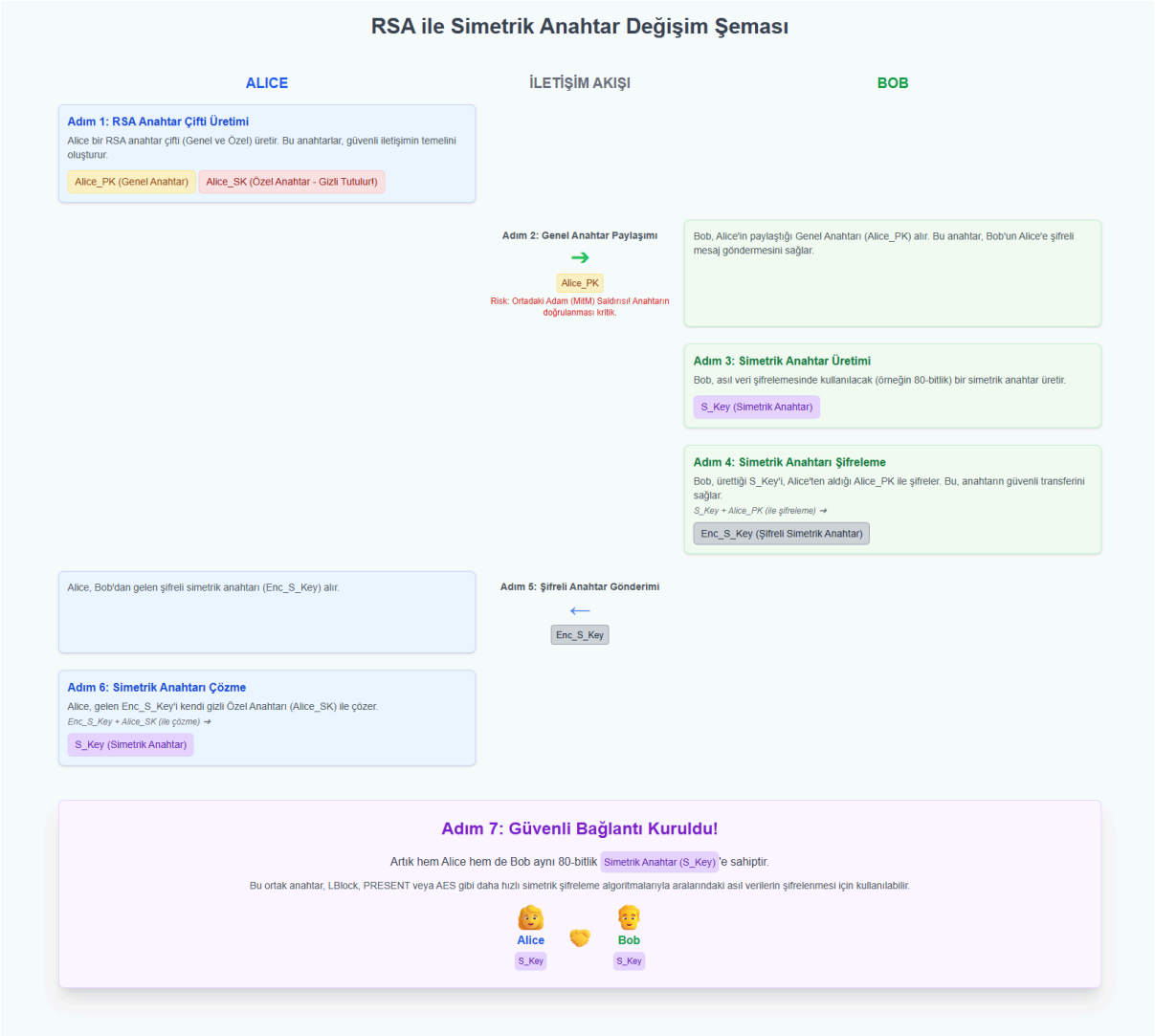
Veri Seti

Testler, çalışma klasöründe bulunan dört adet metin dosyası üzerinde gerçekleştirilmiştir:

- metin1.txt
- metin2.txt
- metin3.txt
- metin4.txt

c. RSA ile Güvenli Simetrik Anahtar Değişimi: Alice ve Bob Örneği

Alice ve Bob'un, LBlock veya PRESENT gibi bir simetrik şifreleme algoritmasında kullanmak üzere ortak bir gizli anahtarı (örneğin, 80-bitlik bir anahtar) RSA kullanarak nasıl güvenli bir şekilde belirlediklerini inceleyelim:



Şekil 5. *Anahtar Değişim Yapısı*

Adım 1: Alice'in Hazırlığı - RSA Anahtar Çiftinin Üretilmesi

- Alice, anahtar değişim sürecini başlatmak için bir RSA anahtar çifti üretir. Bu çift, bir adet **Özel Anahtar (Private Key)** ve bir adet **Genel Anahtar (Public Key)** içerir.
- **Detay:**
    - **Özel Anahtar:** Sadece Alice tarafından bilinir ve çok gizli tutulur. Bu anahtar, Alice'in genel anahtarıyla şifrelenmiş mesajları çözebilmesini ve dijital imzalar oluşturabilmesini sağlar.
    - **Genel Anahtar:** Alice'in özel anahtarıyla matematiksel olarak bağlantılıdır ancak özel anahtar genel anahtardan türetilemez. Alice bu anahtarı, kendisiyle güvenli iletişim kurmak isteyen herkesle (bu örnekte Bob ile) paylaşabilir. Genel anahtar, mesajları Alice için şifrelemek ve Alice'in dijital imzalarını doğrulamak için kullanılır.
- **Örnekteki Yansıma:** generate_rsa_key_pair() fonksiyonu bu işlemi gerçekleştirir. Genellikle 2048-bit veya daha yüksek bir RSA anahtar boyutu güvenlik için tercih edilir.

Adım 2: Alice'in Genel Anahtarını Bob ile Paylaşması

- Alice, ürettiği genel RSA anahtarını Bob'a iletir.

- **Detay:** Bu paylaşım adımının güvenliği çok önemlidir. Eğer bir saldırgan (örneğin, Eve) bu aşamada araya girip Bob'a kendi sahte genel anahtarını Alice'inmiş gibi iletirse, tüm sistemin güvenliği tehlikeye girer (Ortadaki Adam Saldırısı - Man-in-the-Middle Attack). Gerçek dünya uygulamalarında bu risk, Dijital Sertifikalar ve Sertifika Otoriteleri (CA) gibi mekanizmalarla azaltılır. Geliştirilen simülasyonda, Alice'in genel anahtarını (genellikle PEM formatında metin olarak) doğrudan Bob'a verdiğini varsayıyoruz.
- **Örnekteki Yansıma:** serialize_public_key_to_pem() ile Alice genel anahtarını metin formatına dönüştürür. Bob ise deserialize_pem_to_public_key() ile bu metni tekrar kullanılabilir bir anahtar nesnesine çevirir.

Adım 3: Bob'un Simetrik Anahtarı Üretmesi

- Bob, Alice ile yapacağı asıl veri şifrelemesinde kullanmak üzere bir simetrik anahtar üretir. Bu anahtar, LBlock/PRESENT algoritmasının gerektirdiği boyutta (örneğin, 80-bit) ve tamamen rastgele olmalıdır.
- **Detay:** Simetrik anahtarın rastgeleliği ve tahmin edilemezliği, güvenliğin temelini oluşturur.
- **Örnekteki Yansıma:** generate_symmetric_key() fonksiyonu, os.urandom() gibi kriptografik olarak güvenli bir rastgele sayı üreteci kullanarak bu 80-bitlik (10 byte) anahtarı oluşturur.

Adım 4: Bob'un Simetrik Anahtarı Alice'in Genel Anahtarıyla Şifrelemesi

- Bob, az önce ürettiği simetrik anahtarı, Alice'ten aldığı **genel RSA anahtarı** ile şifreler.
- **Detay:** Bu şifreleme işlemi sonucunda elde edilen şifreli metin, yalnızca Alice'in **özel RSA anahtarı** ile çözülebilir. Bu, simetrik anahtarın Bob'dan Alice'e iletilirken gizliliğini sağlar. RSA ile şifreleme yapılırken, güvenlik açıklarını önlemek ve performansı artırmak için OAEP (Optimal Asymmetric Encryption Padding) gibi dolgulama şemaları kullanılır.
- **Örnekteki Yansıma:** encrypt_symmetric_key_with_rsa() fonksiyonu bu işlemi gerçekleştirir.

Adım 5: Bob'un Şifreli Simetrik Anahtarı Alice'e Göndermesi

- Bob, RSA ile şifrelediği simetrik anahtarı Alice'e gönderir.
- **Detay:** Bu şifreli mesaj artık açık bir kanal üzerinden bile (örneğin, internet) gönderilebilir, çünkü içeriği sadece Alice'in özel anahtarı ile çözülebilir. Bir saldırgan bu mesajı ele geçirse bile, Alice'in özel anahtarına sahip olmadığı için simetrik anahtarı elde edemez.
- **Örnekteki Yansıma:** Simülasyonda bu adım, şifreli verinin bir değişkenden diğerine aktarılmasıyla temsil edilir.

Adım 6: Alice'in Şifreli Simetrik Anahtarı Kendi Özel Anahtarıyla Çözmesi

- Alice, Bob'dan gelen şifreli mesajı alır ve kendi **özel RSA anahtarını** kullanarak mesajı çözer.
- **Detay:** Başarılı bir deşifreleme sonucunda Alice, Bob'un orijinalde ürettiği simetrik anahtarı elde eder.
- **Örnekteki Yansıma:** decrypt_symmetric_key_with_rsa() fonksiyonu bu işlemi yapar.

Adım 7: Güvenli Bağlantı Kuruldu!

- Artık hem Alice hem de Bob, aynı 80-bitlik simetrik anahtara sahiptir.
- **Detay:** Bu ortak simetrik anahtar, artık LBlock, PRESENT veya AES gibi daha hızlı olan simetrik şifreleme algoritmalarıyla aralarındaki asıl verilerin şifrelenmesi ve deşifrelenmesi için kullanılabilir. RSA, yalnızca bu ilk simetrik anahtar değişiminin güvenliğini sağlamak

için kullanılmıştır. Asıl veri transferi genellikle simetrik şifreleme ile yapılır çünkü asimetrik şifreleme (RSA) hesaplama açısından daha yoğundur ve büyük verileri şifrelemek için uygun değildir.

RSA tabanlı anahtar değişimi, bir tarafın (Alice) bir genel/özel anahtar çifti oluşturmasını, genel anahtarını diğer tarafla (Bob) paylaşmasını içerir. Bob daha sonra asıl kullanılacak simetrik anahtarı üretir, bunu Alice'in genel anahtarıyla şifreler ve Alice'e gönderir. Alice, kendi özel anahtarını kullanarak bu şifreyi çözer ve simetrik anahtarı elde eder. Bu yöntem, "anahtar dağıtım problemini" çözerek iki tarafın güvenli olmayan bir kanal üzerinden bile ortak bir gizli anahtar üzerinde anlaşmasını sağlar. Bu, modern kriptografinin temel taşlarından biridir ve SSL/TLS gibi protokollerde yaygın olarak kullanılır.

a. Performans Metrikleri

Uygulama (utils.measure_performance aracılığıyla) aşağıdaki iki ana metriği ölçmektedir:

Süre (sn) (Duration):

Ölçüm Yöntemi: time.time() fonksiyonu kullanılarak, çekirdek şifreleme/deşifreleme fonksiyonunun (encrypt_xxx/decrypt_xxx) çağrılmasından hemen önce ve hemen sonra alınan zaman damgalarının farkıdır.

Anlamı: İlgili kriptografik operasyonun tamamlanması için geçen **duvar saati süresini** (wall-clock time) temsil eder. Bu süre, CPU hızına, Python yorumlayıcısının verimliliğine ve algoritmanın kendi işlem karmaşıklığına bağlıdır.

Bellek Farkı (Memory Difference):

Ölçüm Yöntemi: psutil.Process(os.getpid()).memory_info().rss kullanılarak, işlemin çekirdek kriptografik fonksiyondan önceki ve sonraki **RSS (Resident Set Size)** değerleri arasındaki farktır. RSS, bir işlemin ana bellekte (RAM) kapladığı fiziksel bellek miktarını ifade eder.

Anlamı: Bu metrik, kriptografik operasyon sırasında işlemin bellek kullanımındaki **tahmini değişimi** gösterir.

Önemli Notlar: utils.py kodunda da belirtildiği gibi, bu değer bir **tahmindir** ve yanıltıcı olabilir. Python'un Çöp Toplama (Garbage Collection - GC) mekanizması, işletim sisteminin bellek yönetimi ve diğer çalışan işlemler bu değeri etkileyebilir. Negatif veya sıfır çıkması mümkündür. Algoritmanın gerçek teorik bellek ihtiyacını değil, pratik uygulamadaki yaklaşık değişimi yansıtır.

● Performans Karşılaştırması ve Analiz

a. Performans Sonuçları

Dört farklı metin dosyası (metin1.txt, metin2.txt, metin3.txt, metin4.txt) üzerinde LBlock ve PRESENT algoritmalarıyla yapılan şifreleme ve deşifreleme işlemleri sonucunda elde edilen süre (saniye cinsinden) ve bellek farkı (KB cinsinden) değerleri aşağıdaki tabloda verilmiştir.

Tablo 4. *Performans Sonuçları*

Algoritma	Metin No	İşlem	Veri Boyutu (Giriş/Şifreli)	Süre (sn)	Bellek Farkı
LBlock	1	Şifreleme	462 B	16.452	8.00 KB
		Deşifreleme	464 B	9.843	0 B
	2	Şifreleme	632 B	19.793	16.00 KB
		Deşifreleme	632 B	39.135	0 B
	3	Şifreleme	712 B	22.161	28.00 KB
		Deşifreleme	712 B	23.275	16.00 KB
	4	Şifreleme	1.20 KB	23.919	132.00 KB
		Deşifreleme	1.20 KB	19.061	128.00 KB
PRESENT	1	Şifreleme	462 B	17.385	0 B
		Deşifreleme	464 B	14.777	0 B
	2	Şifreleme	632 B	30.082	8.00 KB
		Deşifreleme	632 B	28.274	4.00 KB
	3	Şifreleme	712 B	28.664	32.00 KB
		Deşifreleme	712 B	23.407	24.00 KB
	4	Şifreleme	1.20 KB	33.472	104.00 KB
		Deşifreleme	1.20 KB	40.740	124.00 KB

b. Sonuçların Analizi

Süre (Duration) Analizi:

- **Genel Eğilim:** Veri boyutu arttıkça, beklendiği gibi hem LBlock hem de PRESENT için şifreleme ve deşifreleme süreleri genel olarak artış göstermiştir.
- **LBlock vs. PRESENT (Şifreleme):**
  - metin1 (462B): LBlock (0.016452s) < PRESENT (0.017385s) -> LBlock daha hızlı.
  - metin2 (632B): LBlock (0.019793s) < PRESENT (0.030082s) -> LBlock daha hızlı.
  - metin3 (712B): LBlock (0.022161s) < PRESENT (0.028664s) -> LBlock daha hızlı.
  - metin4 (1.20KB): LBlock (0.023919s) < PRESENT (0.033472s) -> LBlock daha hızlı.

Bu implementasyonda, LBlock algoritması şifreleme işlemlerinde tüm veri setleri için PRESENT algoritmasından daha hızlı performans göstermiştir.

- **LBlock vs. PRESENT (Deşifreleme):**
  - metin1 (464B): LBlock (0.009843s) < PRESENT (0.014777s) -> LBlock daha hızlı.
  - metin2 (632B): PRESENT (0.028274s) < LBlock (0.039135s) -> PRESENT daha hızlı. (LBlock deşifrelemesi bu testte beklenenden yavaş kalmıştır).
  - metin3 (712B): LBlock (0.023275s) ≈ PRESENT (0.023407s) -> Süreler neredeyse aynı.
  - metin4 (1.20KB): LBlock (0.019061s) < PRESENT (0.040740s) -> LBlock önemli ölçüde daha hızlı.

Deşifreleme sürelerinde LBlock, metin2 haricinde genellikle PRESENT'ten daha hızlı veya benzer performans sergilemiştir. metin2'deki LBlock deşifreleme süresindeki artış, anlık bir sistem yoğunluğu veya Python'un çöp toplama (GC) mekanizmasının o anki etkisi gibi bir aykırı durum olabilir.

- **Şifreleme vs. Deşifreleme Süreleri (Algoritma İçi):**
    - **LBlock:** Deşifreleme süreleri bazen şifrelemeden daha kısa (metin1, metin4), bazen daha uzun (metin2), bazen de çok yakın (metin3) olmuştur. LBlock'un Feistel yapısında şifreleme ve deşifreleme benzer operasyonları (farklı anahtar sıralamasıyla) içerir, bu nedenle sürelerin yakın olması beklenmiştir.
    - **PRESENT:** Deşifreleme süreleri genellikle şifreleme sürelerine oldukça yakın veya biraz daha kısadır (metin1, metin2, metin3). Ancak metin4'te deşifreleme, şifrelemeden biraz daha uzun sürmüştür. PRESENT'in SPN yapısında da deşifreleme, şifreleme adımlarının tersi ve benzer karmaşıklıkta olduğu için sürelerin yakın olması beklenmiştir.

Bellek Farkı (Memory Difference) Analizi:

- **Genel Eğilim:** "Bellek Farkı" metriği, süreye kıyasla daha fazla değişkenlik göstermektedir ve yorumlanması daha zordur. Bu metrik, işlemin anlık RSS (Resident Set Size) değişimini gösterir ve Python'un dinamik bellek yönetimi, GC aktivitesi ve işletim sistemi tarafından etkilenmektedir.
- **LBlock vs. PRESENT:**
    - metin1 Şifreleme: LBlock (8KB) > PRESENT (0B)
    - metin2 Şifreleme: LBlock (16KB) > PRESENT (8KB)
    - metin3 Şifreleme: PRESENT (32KB) > LBlock (28KB)
    - metin4 Şifreleme: LBlock (132KB) > PRESENT (104KB)

Deşifreleme işlemleri için de benzer bir değişkenlik gözlemlenmektedir. Tek bir algoritmanın sürekli olarak daha az veya daha çok bellek farkı yarattığını söylemek zordur.

Değerlerin Yorumu:

- "0 B" değerlerinin sık görülmesi, ölçüm anında net bir RSS artışı olmadığını veya tahsis edilen belleğin işlem bitmeden serbest bırakıldığını gösterebilir.
- En büyük dosya olan metin4 için her iki algoritmanın da şifreleme ve deşifreleme sırasında daha belirgin bellek farkları (100KB üzeri) göstermesi, daha fazla veri bloğunun işlenmesiyle ve potansiyel olarak ciphertext_bytes gibi geçici yapıların büyümesiyle ilişkilendirilebilir.
- Genel olarak, gözlemlenen bellek farkları (birkaç KB ile ~130KB arasında) bu boyuttaki veriler için makul kabul edilebilir ve her iki algoritmanın da "hafif" (lightweight) doğasıyla uyumludur; yani büyük bellek ayak izleri bırakmamaktadırlar.

c. Güvenlik Analizi

Bu performans analizi, algoritmaların sadece hız ve kaynak kullanımı yönlerini ele almaktadır. Uygulamanın güvenlik yönü, kullanılan şifreleme modu ve dolgu (padding) yönteminden dolayı **kritik zafiyetler** içermektedir:

1. **ECB (Electronic Codebook) Modu:**
    - Her veri bloğunun bağımsız olarak aynı anahtarla şifrelenmesi, düz metindeki desenlerin şifreli metne yansımasına neden olmaktadır. Bu, özellikle yapısal veriler (metin, resim vb.) için güvenlik açığı oluşturur. Örneğin, bir metin dosyasındaki sık

tekrarlanan kelimeler veya boşluklar, şifreli metinde de tekrarlayan bloklar olarak görülebilir.
    - Bu mod, genellikle sadece kısa ve tamamen rastgele verilerin (örn. tek bir anahtar) şifrelenmesi gibi çok özel durumlar dışında kullanılmamalıdır.
2. **Standart Olmayan Padding (Boşluk Karakteri ile Dolgu):**
    - Düz metnin sonuna eksik byte sayısı kadar boşluk (0x20) eklenmesi standart bir yöntem değildir. (Aynı veri üzerinde farklı algoritma performanslarını ölçmek için kullanılmıştır)
    - Eğer orijinal düz metin zaten boşluk karakterleriyle bitiyorsa, deşifreleme sonrası bu padding'in orijinal veriden ayırt edilmesi imkansızdır, bu da veri bütünlüğünü bozar.
    - Deşifreleme işlemi, eklenen bu padding'i otomatik olarak kaldırmamaktadır. Bu, kullanıcının ek bir işlem yapmasını gerektirmektedir ve hata olasılığını artırabilir.
    - Güvenli uygulamalar için, PKCS#7 gibi, padding'in varlığını ve uzunluğunu net bir şekilde belirten standart padding şemaları kullanılabilir.

● Sonuç ve Değerlendirme

a. Çalışmanın Sonuçları

Uygulama, LBlock ve PRESENT algoritmalarının 80-bit anahtar ve 64-bit blok boyutlu versiyonlarını başarıyla implemente etmiş ve kullanıcıların metin dosyalarını bu algoritmalarla şifreleyip deşifrelemesine olanak tanımaktadır. Performans ölçümleri, süre ve tahmini bellek farkı metriklerini kapsamaktadır. Bu implementasyonda LBlock, şifreleme işlemlerinde test edilen tüm veri boyutları için PRESENT'e göre genellikle daha hızlı bulunmuştur. Deşifreleme işlemlerinde de LBlock, bir aykırı durum dışında, genellikle daha hızlı veya PRESENT ile benzer performans göstermiştir. "Bellek Farkı" metriği daha değişken sonuçlar vermiş olup, net bir kazanan belirlemek zordur. Her iki algoritma da küçük dosyalar için düşük bellek farkları göstermiştir, bu da hafif yapılarıyla tutarlık sağlamaktadır. Uygulamanın kullandığı ECB modu ve standart olmayan boşluk karakteriyle padding yöntemi, ciddi güvenlik zafiyetlerine yol açmaktadır. Bu nedenle, uygulama güvenli iletişim veya veri saklama için uygun değildir ve yalnızca test amaçlıdır.

b. Gelecek Çalışmalar

ECB yerine, CBC (Cipher Block Chaining), CTR (Counter Mode) veya GCM (Galois/Counter Mode) gibi standart ve güvenli şifreleme modları implemente edilebilir. GCM gibi modlar aynı zamanda kimlik doğrulama da sağlamaktadır. Boşluk karakteriyle padding yerine PKCS#7 gibi standart ve geri döndürülebilir padding şemaları kullanılabilir. PRESENT ve LBlock'un (varsa) farklı anahtar ve blok boyutlarını destekleyen versiyonları implemente edilerek karşılaştırmaya yapılabilir. Çok daha büyük ve çeşitli türde (örn. imaj, ses dosyaları) veri setleriyle testler yapılarak algoritmaların ölçeklenebilirliği incelenebilir. Aynı algoritmaların C/C++ gibi daha düşük seviyeli ve derlenmiş dillerle implemente edilip Python sonuçlarıyla karşılaştırılması, dilin performans üzerindeki etkisini de gösterecektir. Eğer mümkünse, hafif kriptografi için optimize edilmiş donanımlarda (örn. IoT cihazları, mikrodenetleyiciler) bu algoritmaların performansları incelenebilir. Özellikle mobil ve IoT cihazları için kritik olan enerji tüketimlerinin ölçülmesi ve karşılaştırılması oldukça faydalı olacaktır. İyileştirilmiş implementasyonlar üzerinde diferansiyel/lineer kriptanaliz gibi standart saldırı vektörlerine karşı dayanıklılıklarının teorik ve pratik olarak incelenmesi (bu genellikle algoritma tasarımcılarının alanına girse de, implementasyon hataları da zafiyet yaratabilir), algoritmaların performans değerleerindeki küçük ya da büyük değişiklikleri açıklamada faydalı olacaktır..

c. Genel Değerlendirme

Bu proje, LBlock ve PRESENT gibi iki önemli hafif blok şifreleme algoritmasının temel çalışma prensiplerini ve bir programlama dilinde nasıl hayata geçirilebileceğini anlamak açısından python programlama dili ile implementasyonunu sunmaktadır. Performans ölçüm altyapısı, farklı algoritmaların veya aynı algoritmanın farklı veri büyüklüklerinde göreceli hız ve kaynak kullanımı hakkında fikir edinilmesini sağlamıştır.

Elde edilen süre sonuçları, bu spesifik Python ortamında LBlock'un genellikle PRESENT'ten biraz daha hızlı olduğunu göstermiştir. Ancak, Python'un yorumlanan bir dil olması ve ölçümlerin milisaniye düzeyinde olması nedeniyle, bu farkların mutlak performans hakkında kesin yargılara varmak için yeterli olmayabileceği, daha çok bu implementasyona özgü olduğu unutulmamalıdır.

Çalışmanın en önemli kısıtı, güvenlik açısından kritik olan çalışma modu ve padding seçimleridir. Bu durum, uygulamanın pratik kullanımını engellemekle birlikte, bu konuların neden önemli olduğunu vurgulamak açısından da öğreticidir.

Sonuç olarak, bu çalışma, kriptografi alanındaki algoritmaların implementasyonu ve temel performans analizi konularında pratik bir deneyim sunmakta, ancak gerçek dünya uygulamaları için çok daha titiz güvenlik önlemlerinin ve standartlara uyumun gerekliliğini açıkça ortaya koymaktadır.

Kaynak kod geliştirilmesinde örnekler alınan kaynaklar:

Proje kodları link : https://github.com/fatihtk/lblock-vs-present

- **https://github.com/DXCyber409/mycryptor**
- **https://github.com/aartuuroo20/QuantumComputing**
- **https://github.com/100472136/Practica_Criptografia**

● Kaynakça

[1] Lightweight Block Ciphers for IoT based applications: A Review - Research India Publications, accessed April 2, 2025, https://www.ripublication.com/ijaer18/ijaerv13n5_26.pdf
[2] apjcriweb.org, accessed April 2, 2025, http://apjcriweb.org/content/vol10no1/5.pdf
[3] Exam SY0-701 topic 1 question 252 discussion - ExamTopics, accessed April 2, 2025, https://www.examtopics.com/discussions/comptia/view/145387-exam-sy0-701-topic-1-question-252-discussion/
[4] Efficiency and Security Evaluation of Lightweight Cryptographic Algorithms for Resource-Constrained IoT Devices - MDPI, accessed April 2, 2025, https://www.mdpi.com/1424-8220/24/12/4008
[5] Lightweight cryptography methods - Taylor & Francis Online, accessed April 2, 2025, https://www.tandfonline.com/doi/full/10.1080/23742917.2017.1384917
[6] LBC-IoT: Lightweight Block Cipher for IoT Constraint Devices - Tech Science Press, accessed April 2, 2025, https://www.techscience.com/cmc/v67n3/41626/html
[7] Lightweight Cryptography Applicable to Various IoT Devices : NEC Technical Journal, accessed April 2, 2025, https://www.nec.com/en/global/techrep/journal/g17/n01/170114.html
[8] Lightweight Cryptographic Algorithms: A Position Paper - SciTePress, accessed April 2, 2025, https://www.scitepress.org/Papers/2024/127929/127929.pdf
[9] A Review of Lightweight Security and Privacy for Resource-Constrained IoT Devices, accessed April 2, 2025, https://www.techscience.com/cmc/v78n1/55430/html
[10] A Survey of Efficient Lightweight Cryptography for Power-Constrained Microcontrollers, accessed April 2, 2025, https://www.mdpi.com/2227-7080/13/1/3
[11] Strong Cryptosystems Needed for Resource-Constrained IoT - XSOC CORP, accessed April 2, 2025, https://www.xsoccorp.com/post/strong-cryptosystems-needed-for-resource-constrained-iot
[12] NIST Selects 'Lightweight Cryptography' Algorithms to Protect Small Devices, accessed April 2, 2025, https://www.nist.gov/news-events/news/2023/02/nist-selects-lightweight-cryptography-algorithms-protect-small-devices
[13] Lightweight Cryptography | Secure-IC, accessed April 2, 2025, https://www.secure-ic.com/blog/lightweight-cryptography/
[14] Lightweight Cryptography - NIST Computer Security Resource Center | CSRC, accessed April 2, 2025, https://csrc.nist.rip/projects/lightweight-cryptography
[15] Lightweight Cryptography | NIST, accessed April 2, 2025, https://www.nist.gov/programs-projects/lightweight-cryptography
[16] Report on Lightweight Cryptography | NIST, accessed April 2, 2025, https://www.nist.gov/publications/report-lightweight-cryptography
[17] Lightweight Cryptography | CSRC - NIST Computer Security Resource Center, accessed April 2, 2025, https://csrc.nist.gov/projects/lightweight-cryptography
[18] The importance of light-weight encryption cipher in restricted IoT systems to make intelligent technology safer for devices - ADS, accessed April 2, 2025, https://ui.adsabs.harvard.edu/abs/2021ApNan.tmp..308K/abstract
[19] SEO Insights into the Fundamentals of Block Cipher Principles - tanmay s dikshit, accessed April 2, 2025, https://www.tanmay.pro/blog/building-blocks-of-security-unraveling-the-principles-of-block-cipher-encryption
[20] Block cipher - Wikipedia, accessed April 2, 2025, https://en.wikipedia.org/wiki/Block_cipher

[21] Block Cipher Principles. Technological advances in the field of… | by Benedict Ell Nino | Medium, accessed April 2, 2025, https://medium.com/@benedict.ellnino/block-cipher-principles-79941c97640f
[22] Understanding block ciphers in cryptography - Infosec, accessed April 2, 2025, https://www.infosecinstitute.com/resources/cryptography/block-ciphers/
[23] SUMER: A New Family of Lightweight and Traditional Block Ciphers with Multi-Modes, accessed April 2, 2025, https://ijcsm.researchcommons.org/cgi/viewcontent.cgi?article=1187&context=ijcsm
[24] Survey and Benchmark of Lightweight Block Ciphers for Wireless Sensor Networks? - Cryptology ePrint Archive, accessed April 2, 2025, https://eprint.iacr.org/2013/295.pdf
[25] Balancing Security and Efficiency: A Power Consumption Analysis of a Lightweight Block Cipher - MDPI, accessed April 2, 2025, https://www.mdpi.com/2079-9292/13/21/4325
[26] A Review of Lightweight Cryptographic Schemes and Fundamental Cryptographic Characteristics of Boolean Functions - Scientific Research Publishing, accessed April 2, 2025, https://www.scirp.org/journal/paperinformation?paperid=114685
[27] Design Space of Lightweight Cryptography - NIST Computer Security Resource Center, accessed April 2, 2025, https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session5-mouha-paper.pdf
[28] (PDF) LBlock: A Lightweight Block Cipher - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/226759286_LBlock_A_Lightweight_Block_Cipher
[29] LBlock: A Lightweight Block Cipher⋆ - Cryptology ePrint Archive, accessed April 2, 2025, https://eprint.iacr.org/2011/345.pdf
[30] (PDF) LBlock: A Lightweight Block Cipher. - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/221651896_LBlock_A_Lightweight_Block_Cipher
[31] The Comparative Study of Randomness Analysis between Modified Version of LBlock Block Cipher and its Original Design - CyberSecurity Malaysia, accessed April 2, 2025, https://www.cybersecurity.my/data/content_files/53/1647.pdf
[32] Lightweight Block Cipher Algorithms: Review Paper - ER Publications, accessed April 2, 2025, https://www.erpublications.com/uploaded_files/download/download_20_05_2016_08_04_01.pdf
[33] Encryption Procedure for LBlock been passed by each cube. The results... - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/figure/Encryption-Procedure-for-LBlock-been-passed-by-each-cube-The-results-have-been-confirmed_fig1_291007966
[34] PRESENT: An Ultra-Lightweight Block Cipher - IACR, accessed April 2, 2025, https://www.iacr.org/archive/ches2007/47270450/47270450.pdf
[35] Impossible differential cryptanalysis of LBlock with concrete investigation of key scheduling algorithm - CiteSeerX, accessed April 2, 2025, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=e3aa0543837e8fde1bc0c0d2f248ce1a234aea7d
[36] A Survey on Lightweight Block Ciphers - International Journal of Computer Applications, accessed April 2, 2025, https://research.ijcaonline.org/volume96/number17/pxc3896923.pdf
[37] PRESENT - Wikipedia, accessed April 2, 2025, https://en.wikipedia.org/wiki/PRESENT
[38] C implementation of the lightweight block cipher PRESENT first published by Andrey Bogdanov. - GitHub, accessed April 2, 2025, https://github.com/Pepton21/present-cipher
[39] (PDF) PRESENT: an ultra-lightweight block cipher - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/221291660_PRESENT_an_ultra-lightweight_block_cipher

[40] PRESENT - Cryptography, accessed April 2, 2025, https://doc.sagemath.org/html/en/reference/cryptography/sage/crypto/block_cipher/present.html
[41] A Survey on Lightweight Cryptography Approach For IoT Devices Security - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/369781102_A_Survey_on_Lightweight_Cryptography_Approach_For_IoT_Devices_Security
[42] LightWeight Cryptographic Algorithms | by Akarsh Singh - Medium, accessed April 2, 2025, https://medium.com/@akarshsingh916/lightweight-cryptographic-algorithms-af4cea712531
[43] Comparing PRESENT and LBlock block ciphers over IoT Platform - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/366848922_Comparing_PRESENT_and_LBlock_block_ciphers_over_IoT_Platform
[44] How to Answer: What Are Your Strengths and Weaknesses? - Betterteam, accessed April 2, 2025, https://www.betterteam.com/strengths-and-weaknesses
[45] Balancing Security and Efficiency: A Power Consumption Analysis of a Lightweight Block Cipher - MDPI, accessed April 2, 2025, https://www.mdpi.com/2079-9292/13/21/4325
[46] Recent Advances and Trends in Lightweight Cryptography for IoT Security - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/345144109_Recent_Advances_and_Trends_in_Lightweight_Cryptography_for_IoT_Security
[47] A Survey of Efficient Lightweight Cryptography for Power-Constrained Microcontrollers, accessed April 2, 2025, https://www.mdpi.com/2227-7080/13/1/3
[48] Lightweight cryptography methods - Taylor & Francis Online, accessed April 2, 2025, https://www.tandfonline.com/doi/full/10.1080/23742917.2017.1384917
[49] Triathlon of Lightweight Block Ciphers for the Internet of Things - NIST Computer Security Resource Center, accessed April 2, 2025, https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session7-grobschadl-paper.pdf
[50] I-PRESENTTM: An Involutive Lightweight Block Cipher - Scientific Research Publishing, accessed April 2, 2025, https://www.scirp.org/journal/paperinformation?paperid=48057
[51] (PDF) PRESENT: an ultra-lightweight block cipher - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/221291660_PRESENT_an_ultra-lightweight_block_cipher
[52] LBlock: A Lightweight Block Cipher⋆ - Cryptology ePrint Archive, accessed April 2, 2025, https://eprint.iacr.org/2011/345.pdf
[53] Simon and Speck: Block Ciphers for Internet of Things - NIST Computer Security Resource Center, accessed April 2, 2025, https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session1-shors-paper.pdf
[54] Evaluating the Performance of Lightweight Ciphers in Constrained Environments—The Case of Saturnin - MDPI, accessed April 2, 2025, https://www.mdpi.com/2624-6120/3/1/7
[55] DNA-PRESENT: An Improved Security and Low-Latency ..., accessed April 2, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC11679402/
[56] DNA-PRESENT: An Improved Security and Low-Latency, Lightweight Cryptographic Solution for IoT - MDPI, accessed April 2, 2025, https://www.mdpi.com/1424-8220/24/24/7900
[57] (PDF) Comparing PRESENT and LBlock block ciphers over IoT ..., accessed April 2, 2025, https://www.researchgate.net/publication/366848922_Comparing_PRESENT_and_LBlock_block_ciphers_over_IoT_Platform

[58] Comparative Study of Block Ciphers Implementation for Resource-Constrained Devices (Review) | Request PDF - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/379884106_Comparative_Study_of_Block_Ciphers_Implementation_for_Resource-Constrained_Devices_Review
[59] The Comparative Study of Randomness Analysis between Modified Version of LBlock Block Cipher and its Original Design - CyberSecurity Malaysia, accessed April 2, 2025, https://www.cybersecurity.my/data/content_files/53/1647.pdf
[60] (PDF) LBlock: A Lightweight Block Cipher - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/226759286_LBlock_A_Lightweight_Block_Cipher
[61] Randomness Analysis on Lightweight Block Cipher, PRESENT - Science Publications, accessed April 2, 2025, https://thescipub.com/pdf/jcssp.2020.1639.1647.pdf
[62] (PDF) Performance Evaluation of Lightweight Encryption Algorithms for IoT-Based Applications - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/349097545_Performance_Evaluation_of_Lightweight_Encryption_Algorithms_for_IoT-Based_Applications
[63] On the Security of LBlock against the Cube Attack and Side Channel Cube Attack, accessed April 2, 2025, https://www.researchgate.net/publication/291007966_On_the_Security_of_LBlock_against_the_Cube_Attack_and_Side_Channel_Cube_Attack
[64] Statistical Analysis on LBlock Block Cipher - CyberSecurity Malaysia, accessed April 2, 2025, https://www.cybersecurity.my/en/knowledge_banks/journal_conference/main/detail/2406/index.html
[65] FPGA Modeling and Optimization of a SIMON Lightweight Block Cipher - PMC, accessed April 2, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC6412312/
[66] Exploring Energy Efficiency of Lightweight Block Ciphers | Request PDF - ResearchGate, accessed April 2, 2025, https://www.researchgate.net/publication/309018378_Exploring_Energy_Efficiency_of_Lightweight_Block_Ciphers
[67] Balancing Security and Efficiency: A Power Consumption Analysis of a Lightweight Block Cipher - Preprints.org, accessed April 2, 2025, https://www.preprints.org/manuscript/202410.0769/v1
[68] On the Efficiency of Software Implementations of Lightweight Block Ciphers from the Perspective of Programming Languages - Cryptology ePrint Archive, accessed April 2, 2025, https://eprint.iacr.org/2019/1218.pdf
[69] Comparison of hardware and software implementations of selected lightweight block ciphers, accessed April 2, 2025, https://www.semanticscholar.org/paper/Comparison-of-hardware-and-software-implementations-Diehl-Farahmand/6c6b91497c00a273def0f7f72832eb124b60d0df
[70] GFRX: A New Lightweight Block Cipher for Resource-Constrained IoT Nodes - MDPI, accessed April 2, 2025, https://www.mdpi.com/2079-9292/12/2/405
[71] Lightweight Block Cipher Algorithms: Review Paper - ER Publications, accessed April 2, 2025, https://www.erpublications.com/uploaded_files/download/download_20_05_2016_08_04_01.pdf