T.C.

EGE ÜNİVERSİTESİ

Fen Bilimleri Enstitüsü

Çok Değişkenli Polinom Sistemlerine Dayanan Kuantum Sonrası Güvenilir Şifreleme Sistemleri ve Açık Kaynak Kodlu Uygulamaları

Yüksek Lisans Tezi

Ramazan KOYUTÜRK

Matematik Anabilim Dalı

İzmir

2020

T.C.
EGE ÜNİVERSİTESİ
Fen Bilimleri Enstitüsü

Çok Değişkenli Polinom Sistemlerine Dayanan Kuantum Sonrası Güvenilir Şifreleme Sistemleri ve Açık Kaynak Kodlu Uygulamaları

Ramazan KOYUTÜRK

Danışman: PROF. DR. URFAT NURİYEV
İkinci Danışman: Doç.Dr. SEDAT AKLEYLEK

Matematik Anabilim Dalı
Bilgisayar Bilimleri Yüksek Lisans Programı

İzmir
2020

Ramazan KOYUTÜRK tarafından yüksek lisans tezi olarak sunulan "Çok Değişkenli Polinom Sistemlerine Dayanan Kuantum Sonrası Güvenilir Şifreleme Sistemleri ve Açık Kaynak Kodlu Uygulamaları" başlıklı bu çalışma EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliği ile EÜ Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 27.01.2020 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu~~ ile başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	:Prof. Dr. Urfat NURİYEV	.....
Raportör Üye	: Dr. Öğr. Üyesi Arif GÜRSOY	.....
Üye	: Dr. Öğr. Üyesi Hakan KUTUCU	.....

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Yüksek Lisans Tezi olarak sunduğum **"Çok Değişkenli Polinom Sistemlerine Dayanan Kuantum Sonrası Güvenilir Şifreleme Sistemleri ve Açık Kaynak Kodlu Uygulamaları"** başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

27 / 01 / 2020

İmzası

Ramazan KOYUTÜRK

ÖZET

Çok Değişkenli Polinom Sistemlerine Dayanan Kuantum Sonrası Güvenilir Şifreleme Sistemleri ve Açık Kaynak Kodlu Uygulamaları

KOYUTÜRK, Ramazan

Yüksek Lisans Tezi, Matematik Anabilim Dalı

Tez Danışmanı: Prof. Dr. Urfat NURİYEV

İkinci Tez Danışmanı: Doç. Dr. Sedat AKLEYLEK

Ocak 2020, 71 sayfa

Bu çalışmada kuantum sonrası şifreleme sistem ailelerinden biri olan çok değişkenli polinom sistemlerine dayanan ABC kriptosistemi anlatılmaktadır. Günümüzde kullanılan açık anahtarlı şifreleme sistemlerinin zorluğu çarpanlara ayırma ve ayrık logaritma problemlerine dayanmaktadır. Kuantum bilgisayarlar yeterli büyüklüğe ulaştıkları zaman bu problemleri kullanan şifreleme yöntemleri güvensiz duruma gelecektir. Bu sebeple kuantum bilgisayarlarda çalışan kriptanaliz yöntemlerine karşı dirençli kriptosistemlerin geliştirilmesine ve bunların farklı platformlardaki uygulamalarına ihtiyaç vardır.

Çalışmada öncelikle ABC kriptosisteminde kullanılan matematiksel altyapı anlatılmış ve sonrasında ABC kriptosisteminin teorik yapısı hatırlatılmıştır. Bunlara bağlı olarak CPU üzerinde hem thread'siz ve thread'li hem de GPU üzerinde CUDA kullanılarak bir uygulaması gerçekleştirilmiştir. Her iki işlemci üzerinde de çalıştırılan uygulamanın arasındaki farklar belirtilip karşılaştırılma yapılmıştır.

Anahtar sözcükler: Kuantum sonrası kriptografi, çok değişkenli polinom sistemleri, ABC kriptosistemi, CUDA, CPU, GPU.

ABSTRACT

Quantum Secure Multivariate Polinomial Polynomial System Based Cryptosystems and Their Open Source Implementations

KOYUTÜRK, Ramazan

MSc in Mathematics

Supervisor: Prof. Dr. Urfat NURİYEV

Co-Supervisor: Assoc. Prof. Dr. Sedat AKLEYLEK

January 2020, 71 pages

In this thesis, quantum secure ABC cryptosystem, a member of multivariate polynomial system family, is studied. The computational hardness of the public key cryptographic systems used today is based on integer factorization or discrete logarithm problems. When quantum computers with large number of qubits are built, public key cryptosystems whose hardness depend on integer factorization or discrete logarithm problem will not be secure. Therefore, there is a need for the development of cryptosystems resistant to cryptanalysis methods running on quantum computers and their applications on different platforms.

In this thesis, mathematical background of ABC cryptosystem is detailed. Then, CPU and GPU implementations are provided. In CPU implementation is performed with/without thread. Moreover, GPU implementation is achieved by using CUDA. A detailed comparison for the implementations is given.

Keywords: Post-quantum cryptography, multivariate polynomial, ABC cryptosystem, CUDA, CPU, GPU.

ÖNSÖZ

Kuantum sonrası şifreleme sistemlerinden biri olan çok değişkenli polinom sistemlerine dayanan şifreleme sistemlerinden birisi ABC kriptosistemidir. Genellikle ABC Kriptosisteminin teorik yapısı temel alınarak çalışmalar yapılmaktadır.

Bu tez çalışmasında, CPU üzerinde hem thread'siz ve thread'li hem de GPU üzerinde CUDA kullanılarak ABC kriptosisteminin bir uygulaması gerçekleştirilmiştir. Bu sayede kuantum bilgisayarlar yaygınlaşmasıyla ortaya çıkacak güvenlik zafiyetlerine karşı şimdiden bir çalışma başlatılması önem arz etmektedir.

Tez çalışmamı EEEAG-116E279 numaralı proje kapsamında destekleyen TÜBİTAK'a teşekkür ederim.

İZMİR

27/01/2020

Ramazan KOYUTÜRK

İÇİNDEKİLER DİZİNİ

ŞEKİLLER DİZİNİ

Şekil Sayfa

TABLOLAR DİZİNİ

Tablo Sayfa

1. GİRİŞ

Açık anahtarlı şifreleme sistemlerinin 1970'lerin sonunda geliştirilmesi ile modern şifrelemede köklü bir atılım oldu. O tarihlerden beri açık anahtarlı şifreleme sistemleri, giderek artan bir şekilde iletişim ağlarının, ayrılmaz bir parçası haline gelmiştir. Şifreleme teknikleri, modern toplumda iletişimin güvenliğini garanti altına almak için kullanılan önemli bir araçtır. Son yıllarda internetin yaygınlaşması ile SSL (Secure Sockets Layer)'in de çalışma mantığında bulunan açık anahtarlı kriptosistem kullanımı yaygınlaşmıştır. Bilginin uçtan uça güvenli bir şekilde gönderimi, saklanması, insanların, şirketlerin ve devletlerin bu sistemi kullanmasını ve geliştirmesini sağlamıştır. Bununla birlikte kuantum bilgisayarların ortaya çıkmasıyla klasik açık anahtarlı şifreleme sistemleri güvenirliğini kaybedeceği öngörülmektedir. Bu sebeple kuantum bilgisayarlar tarafından gerçekleştirilecek ataklara karşı açık anahtarlı şifreleme sistemleri geliştirilmesi gerekmektedir. Ayrıca, günümüz işlemci mimarileri çok çekirdekli, kuantum sonrası güvenilir açık anahtarlı sistemlerin bunlara uygun bir şekilde uygulanmalarına ve geliştirilmelerine ihtiyaç vardır (Akleylek ve Koyutürk, 2019).

Günümüzde, internet ve diğer iletişim sistemleri temel olarak dijital imza algoritması (DSA), eliptik eğri DSA veya ilgili algoritmaları kullanan Diffie-Hellman anahtar değişimi, RSA şifrelemesi ve dijital imzalara dayanmaktadır (Rivest and Shamir and Adleman, 1978). Bu şifreleme sistemlerinin güvenliği, tam sayıları çarpanlara ayırma veya ayrık logaritma gibi belirli sayıdaki teorik problemlerin hesaplama zorluğuna bağlıdır; ancak 1994 yılında Peter Shor, kuantum bilgisayarlarda bu problemlerin her birini polinom zamanda çözebileceğini göstermiştir (Shor,1994; Shor,1997). Bu, kuantum bilgisayarların yaygınlaşmasıyla bir gerçeğe dönüşecek ve bu varsayımlara dayanan tüm şifreleme sistemleri güvensiz olacaktır.

Çok değişkenli polinom sistemleri tabanlı kriptosistemlerinin belirli koşullar altında kuantum hesaplama saldırılarına karşı dirençli olduğuna inanılmaktadır. Bunun nedeni, çok değişkenli açık anahtar şifreleme sistemlerinin NP-zor (NP-hard) bir problem olan sonlu bir cisim üzerinde çok değişkenli bir polinom sistemine dayanmasıdır (Buchmann and Butin, 2016; Garey and Johnson, 1979).

Günümüzde açık anahtarlı kriptografide imzalama, anahtar şifreleme, anahtar değişimi gibi işlemler için en çok tercih edilen algoritmalar RSA (Rivest, Shamir and Adleman, 1978), Diffie-Hellman, DSA ve ECC'dir, ancak bu algoritmalar verinin güvenliğini sınırlı bir süre boyunca güvende tutabildiği için yeterince büyük kuantum bilgisayar geldiğinde bu gibi algoritmalar güvensiz hale gelecektir. Bu güvenlik sorununun sebebi, verileri elektriksel işaretleme ile değil de foton olarak adlandırılan ışık tanecikleri tanımlayıp bunları işleyebilen kuantum bilgisayarlardır. Bu bilgisayarlar üzerinde çarpanlara ayırma problemi ve ayrık logaritma problemini polinom zamanda çözen Shor (Shor,1997) algoritması bulunmaktadır. Bu nedenle, kuantum bilgisayar saldırılarından etkilenmeyecek, matematiksel problemlere dayanan klasik şifreleme yöntemlerine alternatifler gerekir. Literatürde kuantum ataklarına karşı dirençli olduğu düşünülen beş ana sınıf bulunmaktadır, bunlar: çok değişkenli polinom sistemleri, kafes, özet, kod ve izojeni tabanlı sistemlerdir. Bu çalışmada çok değişkenli polinom sistemlerini kullanan ABC kriptosistemi incelenecek ve farklı platformlardaki uygulamaları hakkında detaylar verilecektir.

Çok değişkenli polinom sistemleri tabanlı kriptosistemler kullandığı altyapı gereği oldukça hızlı ama anahtar boyutlarından dolayı genelde fazla bellek gereksinimine ihtiyaç duyar. Bu durum akıllı kartlar ve RFID (Landt, 2005) çipleri gibi düşük maliyetli cihazlarda kullanım için onları çekici kılar. Buna benzer birçok pratik çok değişkenli imza şeması (Ding and Schmidt, 2005) mevcut olsa da verimli ve güvenli çok değişkenli şifreleme şemalarının sayısı sınırlıdır.

1.1 Kuantum Bilgisayarlar

Günümüzde elde edilen bilgiyi işlemek ve iletmek için kullanılan bilgisayarların çalışma sistemi klasik mekanikle tamamen açıklanabilir. Fakat klasik mekanik sadece makro ölçekte geçerli bir kavramdır. Daha detaylı ölçeklerde bu kavramın geçerliliğini yitirdiği gözlemlenmiştir. Temel atomların ve moleküllerin davranışları ise kuantum mekaniği ile açıklanmaktadır.

Kuantum mekaniği her boyutta geçerli bir kuramdır (DiVincenzo, 2008). Klasik mekanikte böyle bir durumdan söz edilememektedir. Mikro ölçekten makro ölçekli sistemlere uygulandığında geçerliliğini yitirmez (Ocak, 2018). Ancak

sistem büyüdükçe, klasik mekanik yasalarıyla bir bütün olarak davranışları daha uyumlu hale gelir. Klasik mekanik kuantum mekaniğinin sonlu halidir. Bu sebeple çalışması kuantum mekaniği ilkelerine dayanan makinelerin bir bilgiyi işlemede ve iletmede klasik mekanik ilkelerine dayanan makinelerden daha verimli olacağı söylenmektedir. Bu nedenle uzun yıllardır kuantum bilgisayarların geliştirilmesi için çalışmalar yapılmaktadır (IBM, 2020).

Günümüzde kuantum bilgisayarların hâlâ başlangıç seviyesinde olduğu söylenebilir (DiVincenzo, 2008). Bununla birlikte hem teorik hem de pratik olarak araştırmalar devam etmektedir. IBM tarafından geliştirilen Quantum Experience projesi ile geliştirilmiş 20 kübitlik (kuantum bit) bir kuantum bilgisayarı var ve bu bilgisayar kuantum bilişim deneylerinde kullanılabiliyor (IBM, 2020). IBM'in gelecek planları arasında 50 kübitlik bir kuantum bilgisayar üretimi de bulunmaktadır (IBM, 2020).

Bir bilgisayarın kuantum mekaniği ilkelerine uygun biçimde çalışıp anlamlı sonuçlar vermesi için sağlaması gereken beş koşul vardır (Ocak, 2020).

- Arzu edilen ölçekte üretilebilecek, iyi karakterize edilmiş kübitler içeren bir fiziksel sistem.
- Kübitlerin durumunu bir referans durumuna dönüştürebilmek.
- Geçitlerin işlem zamanından daha uzun kuantum eşevresizlik zamanları.
- "Evrensel" kuantum geçitleri.
- İstenilen kübitler üzerinde istenilen ölçümlerin yapılması.

1.1.1 Kuantum Bilgisayarların Üstünlüğü

Bir kuantum bilgisayarın klasik bir bilgisayara göre ne kadar hızlı olduğunun cevabı çözülmeye çalışılan sorunun ne olduğuna göre değişir. Bazı algoritmalar için kuantum bilgisayarlar, klasik bilgisayarlardan daha hızlı sonuçlar vermez. Örneğin; $f(x)$ bir fonksiyon olmak üzere $f(f(\dots f(x)\dots))$ şeklinde n. fonksiyonun hesaplanma süresi klasik bilgisayarlarda daha hızlıdır. Ancak bazı algoritmalar için kuantum bilgisayarlar klasik olanlardan "biraz" daha hızlıdır. Bazı algoritmalar için

ise kuantum bilgisayarlar, klasik bilgisayarlardan kat ve kat daha hızlıdır. Örneğin; verilen herhangi bir sayının çarpanlarına ayırmak gibi.

Çarpanlara ayırma problemi internetin güvenliği bakımından çok önemlidir. İnternet güvenliğini sağlamak amacıyla kullanılan RSA algoritması, yüzlerce basamaklı sayıları çarpanlarına ayırmaktan geçer (Williams, 1980). Günümüz klasik bilgisayarlarında bu problemin çözümü; yöntemsel olarak en basit şekilde tüm olasılıkları teker teker denemektir. Böyle bir durumda ise şu anki en hızlı bilgisayarlarla bile yüzyıllar sürer (Kocher, 1996). Shor (Shor,1997) algoritması olarak adlandırılan, kuantum bilgisayarları için geliştirilmiş bir algoritmaysa sadece birkaç denemede yüzlerce basamaklı sayıların çarpanlarını bulmaya imkân veriyor.

Günümüzde kimya ve fizik ile ilgili birçok kavram, kuantum sisteminin iyi kavranmasına dayanmaktadır. Fakat yukarıda bahsedilen sistemlerin klasik bilgisayarda verimli bir şekilde oluşturulması neredeyse imkânsızdır. Örneğin, 10 kübit'lik bir kuantum bilgisayarın durumunu klasik bir bilgisayarda saklamak için $2^{10} = 1024$ adet karmaşık sayının hafızada saklanması gerekmektedir. Bu kübit'lerin sayısı 50'ye çıktığında bu sayı $2^{50} = 1125899906842624$'e çıkar. Elde edilen bu sayı şu anki en iyi klasik bir bilgisayarın kapasitesinin çok üzerindedir (Lloyd, 1995).

Günümüzde kuantum bilgisayarlar hiçbir görevi klasik bilgisayarlardan daha hızlı yapamamaktadır. Fakat IBM ve Google gibi büyük firmalar yakın gelecekte bazı algoritmaları klasik bilgisayarlardan daha hızlı sonuçlandıracak bir kuantum bilgisayar geliştireceklerini dile getirmektedirler (Tonbil, 2020).

Günümüzde kuantum bilgisayarı araştırmacıları genellikle çok sayıda kübit içeren bilgisayarlar geliştirmek için çalışmalarını sürdürmektedir. Kuantum bilgisayarlardaki işlem hata oranları genellikle işlem sırasında geçen zamanın decoherence zamanına oranıdır. Bu nedenle hata oranını küçültmek için herhangi bir işlemin decoherence zamanından çok daha kısa bir süre içinde tamamlanabilmesi gerekmektedir. Herhangi bir işlemin decoherence zamanından çok daha kısa bir süre içinde tamamlanmasıyla hata oranı düşürülür. Eğer hata oranı yeteri kadar düşürülür ise kuantum hata düzeltme algoritmalarını da kullanarak

decoherence'tan kaynaklanan hataları düzeltmek ve decoherence zamanından daha uzun süren hesaplamalar yapmak mümkün olur. Ancak bu hataları düzeltme işlemlerinin yapılmasıyla gerekli kübit sayısında da çok fazla derecede artışa sebep olmaktadır (Tonbil, 2020). Örneğin sayıları çarpanlarına ayırmak için öne sürülmüş Shor (Shor,1997) algoritmasını ele alındığında. Eğer çarpanlarına ayrılacak sayı L tane bitle temsil ediliyorsa, bu sayıyı çarpanlarına ayırmak için gerekli kübitlerin sayısının L ile L^2 arasında olacağı düşünülmektedir. Shor algoritması yaklaşık L^2 işlem gerektirir. Tablo 1.1'de farklı bit uzunluklarındaki sayılar için gerekli kübit değerleri verilmiştir.

Tablo 1.1 Bir Sayının Çarpanlara Ayrılması İçin Gerekli Kübit Sayısı

Çarpanlarına Ayrılacak Sayı	Gerekli Kübit
L bit	$2L$
10^3 bit	10^7 kübit
2^{11}=2048 bit	2^{23} kübit

Şu an kuantum bilgisayarlarla ilgili olarak bilinen en önemli uygulamaları şunlardır (Jordan, 2019):

- Kriptoanaliz yöntemlerinin daha verimli kullanılması.
- İşlem süresi uzun süren kodların optimizasyonu yapmak.
- Kriptografi, ataklara karşı güvenli iletişim.
- Klasik bilgisayarlar tarafından uzun süren çarpanlara ayırma işlemini hızlı bir şekilde yapması, Shor'un algoritması.
- Grover'in arama algoritması.
- Kuantum mekaniksel sistemlerin verimli olarak simülasyonu.

1.2 Kriptografi

Kriptografi, genel anlamda mesajların gizli tutulması bilimi; başka bir deyişle bilgileri güvence altına alma bilimidir (Akleylek ve Nuriyev, 2005). Kriptografinin sivil alanda da kredi kartı ile internet alışverişi sırasında kart bilgilerinin güvenli şekilde iletilmesi gibi pek çok uygulaması vardır. Açık anahtarlı şifreleme sisteminin gelişmesi ile birlikle (1970) bu konuda önemli gelişmeler yaşanmıştır.

Rivest, Shamir ve Adleman tarafından geliştirilen RSA (Rivest, Shamir ve Adleman, 1977) şifreleme sistemi sivil uygulamalarda kullanılan şifreleme yöntemlerinden biridir. Günümüzde internet ortamında kullanılan Pretty Good Privacy (PGP) paketi (Garfinkel, 1995) RSA şifreleme yöntemine dayanmaktadır.

RSA şifreleme sisteminin gücü çarpanlara ayırma problemine dayanmaktadır. Şu anki çarpanlara ayırma yöntemleri ve bilgisayarların işlem gücü sebebiyle güçlü olduğu savunulmaktadır. Ancak bir sayıyı polinom zamanda çarpanlarına ayıran bir yöntem geliştirildiğinde veya yeterince büyük kuantum bilgisayarlar ortaya çıktığında RSA'nın güvenliği sorgulanmaya başlanacaktır.

Dijital imza algoritması veya eliptik eğri algoritmaları ise ayrık logaritma problemine dayanmaktadır. Bu sebeple bahsi geçen kriptosistemler kuantum çağında RSA ile aynı zayıflığı paylaşmaktadırlar.

Kuantum kriptografi güvenli iletişim için kuantum mekaniğini kullanır. Üçüncü kişilerin şifrelenmiş mesajları okumasını önlemek için matematiksel problemler temelli teknikler kullanan klasik şifrelemenin aksine, kuantum şifreleme, verinin fiziğini temel almaktadır.

Shor'un Algoritması

AT&T Bell araştırma laboratuvarlarında çalışan Peter Shor kuantum bilgisayarların hesaplama gücünü göstermiştir (Shor, 1994,1997). 1994 yılında yayımlamış olduğu bir makale ile kuantum mekaniğini temel alarak çalışan bir bilgisayarın büyük sayıların çarpanlarını çok hızlı bir şekilde bulabileceğini göstermiştir (Shor, 1994).

Grover'ın Algoritması

Kuantum bilgisayarların performansının daha iyi olduğu başka bir problem de sıralanmamış bir listede arama algoritmasıdır (Grover, 1996). Sözlükte bir kelimenin anlamını bulmak harf sırasına göre dizilmiş bir sözlüğe bakıldığını düşünelim. Yapılacak tek şey aranan kelimeyi buluncaya kadar tüm kelimelere teker teker bakmaktır. Ancak AT&T Bell laboratuvarlarından Lov Grover

geliştirdiği algoritma yardımıyla kuantum bilgisayarların bu işi polinom zamanda çözülebileceğini gösterdi (Grover, 1996).

Grover'ın algoritmasının önemi, klasik anlamda "umutsuz" olarak nitelenebilecek bir problem üzerinde ilerleme sağlamasıdır. Aynı şey Shor'un çarpanlara ayırma yöntemi için de geçerlidir.

2. MATEMATİKSEL ALTYAPI

Bu bölümde çok değişkenli polinom sistemlerine dayanan ABC kriptosisteminin matematiksel temeli anlatılmaktadır. ABC sisteminin yapısının anlaşılabilmesi için başta sonlu cisim olmak üzere grup ve halka yapıları bilinmelidir.

2.1 Bazı Temel Tanımlamalar

Bu bölümde şifreleme ve şifre çözmenin bazı temel anlamları ile başlanacaktır.

Açıkmetin (Plaintext): Şifrelenecek olan düz metindir.

Şifremetin (Ciphertext): Şifrelenmiş metin.

Şifreleme (Encryption): Şifrelenmesini istenen bir metni sadece izin verilen kişilerin veya programların okumasına olanak sağlayacak şekilde yeni bir metin elde edilmesidir.

Şifreleme Algoritması (Encryption Algorithm): Bir düz metni şifrelenmiş metne dönüştürmek için gerekli süreçlerin toplamıdır.

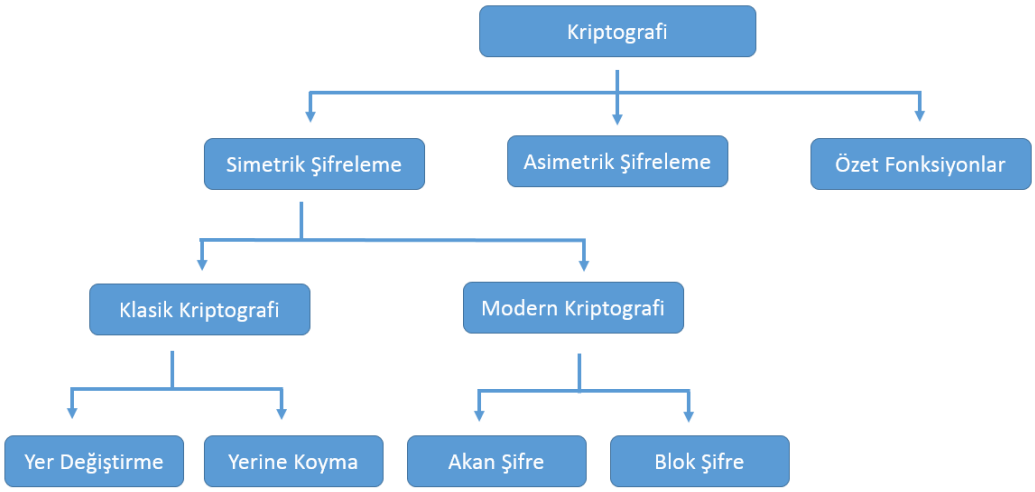
Gizli Anahtar (Secret Key): Şifreleme Algoritması (Encryption Algorithm) çalıştırılırken bazı parametrelerin birkaçını veya tamamını ayarlamak için kullanılır.

Şifre Çözme (Decryption): Şifrelenmiş bir metnin açık (anlaşılır) hale getirilmesidir.

Şifre Çözme Algoritması (Decryption Algorithm): Şifrelenmiş bir metni gizli anahtara sahip kişi veya program tarafından çözümlemesini yapan süreçlerin tamamıdır.

Kriptografi (Cryptography): Şifreleme ve şifre çözmek için gerekli tüm unsurların toplamıdır. Şekil 2.1'de kriptosistemlerin sınıflandırılması gösterilmektedir.

Şekil 2.1 Kriptosistemlerin Sınıflandırılması



Blok Şifre (Block Cipher): Tek bir seferde bir girdi verisi bloğunu işleyerek aynı boyutta şifreli metin oluşturur.

Akan Şifre (Stream Cipher): Bir akış kullanmaktır. Diğer bir deyişle her şifreleme birimindeki işlem, bir önceki şifreleme işlemine bağlı olacak şekilde devam eder.

2.2 Sonlu Cisimler

Bu bölümde başta sonlu cisimler olmak üzere grup, halka ve cisimler konusu üzerinde durulmuştur. Bölümün genelinde (Kak, 2019) ders notlarından yararlanılmıştır.

2.2.1 Sonlu Cisimler

Sonlu cisimlerin ne anlama geldiğini bilinmeden modern kriptografinin veya genel bilgisayar güvenliğinin herhangi bir kısmını anlamak mümkün değildir.

Sonlu cisimler veya başka bir ifadeyle Galois cisimleri kodlama teorisi ve kriptografide oldukça fazla kullanılmaktadır (Lidl and Niederreiter, 1983).

Sonlu cisimler kavramının anlaşılması tezin ana konusu olan ABC kriptosisteminin anlaşılması için önemlidir. Çünkü ABC sistemi temelini oluşturan matematiksel işlemler ve geliştirilen algoritmalar sonlu cisimler üzerinde gerçekleştirilmektedir.

Sonlu cisim kavramını anlamak için bilinmesi gerekenler;

- Grup (Group)
- Değişmeli Grup (Abelian Group)
- Halka (Ring)
- Cisim (Field)

Şekil 2.2'de grup, halka ve cisim yapısı anlatılmaktadır.

Şekil 2.2 Grup, Halka ve Cisim Yapısı (Kayaoğlu,2020).

i - Kapalılık var: $a, b \in A$ ise $a * b \in A$
ii - Birleşme özelliği var: Her $a, b, c \in A$ için $a * (b * c) = (a * b) * c$
iii - Etkisiz (birim) eleman var: Her $a \in A$ için $a * e = e * a = a$ olacak şekilde bir $e \in A$ vardır.
iv - Her elemanın tersi var: Her $a \in A$ için öyle bir $\acute{a} \in A$ vardır ki $a * \acute{a} = \acute{a} * a = e$ olur

} $\langle A, * \rangle$ bir gruptur

v - Değişme özelliği var: Her $a, b \in A$ için $a * b = b * a$

} $\langle A, * \rangle$ bir değişmeli gruptur

vi - Kapalılık var: $a, b \in A$ ise $a \cdot b \in A$
vii - Birleşme özelliği var: Her $a, b, c \in A$ için $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
viii - Sağ ve sol dağılma özelliği var: Her $a, b, c \in A$ için $a \cdot (b * c) = (a \cdot b) * (a \cdot c)$ ve $(a * b) \cdot c = (a \cdot c) * (b \cdot c)$

} $\langle A, *, \cdot \rangle$ bir halkadır

ix - Etkisiz eleman var: Her $a \in A$ için $a \cdot \acute{e} = \acute{e} \cdot a = a$ olacak şekilde $\acute{e} \in A$ vardır.
x - Değişme özelliği var: Her $a, b \in A$ için $a \cdot b = b \cdot a$

} $\langle A, *, \cdot \rangle$ birimli ve değişmeli bir halkadır

xi - Sıfır böleni yok: $a, b \in A$ ve $a \cdot b = e$ ise $a = e$ veya $b = e$

} $\langle A, *, \cdot \rangle$ bir tamlık bölgesidir.

xii - Sıfırdan farklı her elemanın tersi var: $a \neq e$ olmak üzere her $a \in A$ için öyle bir $\acute{a} \in A$ vardır ki $a \cdot \acute{a} = \acute{a} \cdot a = \acute{e}$ olur.

} $\langle A, *, \cdot \rangle$ bir cisimdir

2.2.2 Grup

Bir G kümesi üzerinde $*$ işaretiyle gösterilen bir ikili işlem tanımlı olsun. Eğer bu işlem grup aksiyomları denilen aşağıdaki özellikleri sağlar ise $(G,*)$ yapısına bir grup denir.

- $(G,*)$, * işlemi birleşmeli olmalıdır.

$$\forall a,b,c \in G \text{ için } a*(b*c) = (a*b)*c$$

- G'nin içinde birim eleman (etkisiz eleman = e) vardır.

$$\forall a \in G \text{ için } a*e = e*a = a$$

- $\forall a \in G$ için G'de a'nın ters elemanı $a^{-1} \in G$ vardır.

$$a*a^{-1} = a^{-1}*a = e$$

2.2.3 Değişmeli (Abelian) Grup

Grup olma şartına ek olarak gerçekleşen bu özellik eğer küme elemanlarının üzerindeki işlemler değişmeli ise bu gruba değişmeli (abelian) grup denir.

$\forall a,b \in G$ için $a*b = b*a$ koşulunu sağlar.

Örnek olarak; Z tamsayılar kümesi $+$ (toplama) işlemine göre bir değişmeli gruptur.

2.2.4 Halka

Bir H kümesi üzerinde $+$ ve $*$ işaretleriyle gösterilen iki tane ikili işlem tanımlı olsun. Eğer bu işlemler halka aksiyomları denilen aşağıdaki özellikleri sağlar ise $(H,+,*)$ yapısına bir halka denir.

- H, $+$ işlemine göre değişmeli grup olmalıdır. Değişmeli grup olduğu için birleşme, birim elemana ve ters elemana da sahip olmalıdır.
- $*$ işlemi H'de birleşmelidir

$$\forall a,b,c \in H \text{ için } a*(b*c) = (a*b)*c$$

- $*$ işleminin $+$ işlemi üzerine soldan ve sağdan dağılma özelliği vardır.

$$\forall a,b,c \in H \text{ için } a*(b+c) = (a*b)+(a*c)$$

$$\forall a,b,c \in H \text{ için } (b+c)*a = (b*a)+(c*a)$$

Bu üç özellik ile birlikte

- $*$ işlemine göre H'nin birim elemana sahip ise $(H, +, *)$ halkasına birim elemanlı ya da birimli halka denir.

$$\forall \, a, e \in H \text{ için } a * e = e * a = a$$

- Eğer $*$ işlemi değişmeli ise halka değişmelidir.

$$\forall \, a, b \in H \text{ için } a * b = b * a$$

Birim elemanlı ve değişmeli halkalarda $a * b * 0$ $(e \neq 0)$ ifadesi $a = 0$ veya $b = 0$ durumunu sağlıyorsa bu halkaya integral domain denir. Örneğin aritmetik toplama ve çarpma işlemleri altında tüm tamsayılar ve tüm reel sayılar kümesidir.

2.2.5 Cisim

Bir F kümesi üzerinde $+$ ve $*$ işaretleriyle gösterilen iki tane ikili işlem tanımlı olsun. Eğer bu işlemler cisim aksiyomları denilen aşağıdaki özellikleri sağlar ise $(F, +, *)$ yapısına bir cisim denir.

- $(F, +, *)$ birimli ve değişmeli halkadır.
- F kümesinden $+$ işleminin birim elemanı (0) çıkarıldığında geri kalan küme $*$ işlemine göre değişmeli gruptur.

Genel özellikleri aşağıdaki gibidir:

- F kümesinde $+$ ve $*$ işlemleri tanımlı ve 0, $e \neq 0$ elemanlarına sahiptir.
- $+$ ve 0 ile F değişmeli bir gruptur $(F, +)$.
- $e \neq 0$ ve $*$ ile F değişmeli bir gruptur $(F, *)$.
- $+, *$ işlemleri dağılma özelliğine sahiptir.
- 0: sıfır elemanı, e: çarpmaya göre birim elemanıdır.

Örnek olarak $(Q, +, \cdot)$ rasyonel sayılar kümesi bir cisimdir.

2.2.6 Z_n Üzerinde Modüler Toplama ve Çarpma Arasındaki Asimetri

Z_n kümesinin her elemanı için toplamaya göre tersi vardır ama Z_n kümesinin sıfırdan farklı her elemanının çarpmaya göre tersi yoktur.

Tablo 2.1'de örnek olarak mod 8'de toplamaya ve çarpmaya göre ters elemanları gösterilmektedir.

Tablo 2.1 Z_8 Kümesinin Toplamaya ve Çarpmaya Göre Tersleri.

Z_8	0	1	2	3	4	5	6	7
Toplamaya Göre	0	7	6	5	4	3	2	1
Çarpmaya Göre	-	1	-	3	-	5	-	7

Tablo 2.1'de Z_n kümesinin elemanlarının çarpmaya göre tersleri asal sayıdır. İki pozitif tam sayının 1'in dışında ortak böleni yok ise bu sayılar aralarında asaldır. Başka bir deyişle en büyük ortak böleni (greatest common divisor) 1 olan herhangi iki tam sayı aralarında asaldır.

$Mod\ n$'ye göre toplama ile aritmetik toplama işleminin özellikleri aynnıdır.$(a + b) \equiv (\mathrm{a} + \mathrm{c})\ mod\ n, b \equiv \mathrm{c}$ mod n şeklinde yazılabilir. Ama a ve n aralarında asal olmadıkça, benzer özellik $mod\ n$'ye göre çarpma işleminde uygulanmaz. $(a * b) \equiv (\mathrm{a} * \mathrm{c})\ mod\ n, b \equiv \mathrm{c}$ mod n şeklinde yazılamaz.

Yukarıda belirtilen $mod\ n$'ye göre toplama özelliğinin, Z_n'nin tüm elemanları için, her bir $a \in Z_n$ için toplamaya göre tersi "$-a$" mevcut olduğu gerçeğinden doğru olması gerekir. Sonucu kanıtlamak için denklemin her iki tarafına "$-a$" eklenebilir.

$Mod\ n$'ye göre çarpımı aynı şekilde kanıtlamak için, yukarıdaki ikinci denklemin her iki tarafını "a^{-1}" ile çarpılması gerekmektedir. Ama yukarıda da bahsedildiği üzere Z_n'nin tüm elemanlarının çarpmaya göre tersi yoktur.

Z_n kümesinin elemanlarının çarpmaya göre tersinin olması "a" sayısının "n" sayısı ile aralarında asal olmasına bağlıdır.

2.2.7 Euclid Metodu ile İki Sayının EBOB'unu Bulma

Euclid algoritması matematikte çok sık kullanılan Ebob'u (en büyük ortak bölen, greatest common divisor, gcd) hesaplamak için Euclid'in geliştirdiği bir metottur.

Ebob hesaplaması için kullanılan Euclid algoritmasının sözde kodu aşağıdaki gibidir.

```
gcd (a, b)
    if a == 0
        return b
    endif
    return gcd (b mod a, a)
```

Burada önemli olan nokta $\gcd\ (a, b) = \gcd\ (b, a\ mod\ b)$ şeklinde yazılabilmesidir.

Euclid'in GCD Algoritmasının Çalışma Adımları

Varsayalım ki; $b_1 > b_2$, $b_2 > b_3$, $b_3 > b_4$, … şeklinde devam etsin.

$$\gcd(b_1, b_2) = \gcd(b_2, b_1\ mod\ b_2) = \gcd(b_2, b_3)$$
$$= \gcd(b_3, b_2\ mod\ b_3) = \gcd(b_3, b_4)$$
$$= \gcd(b_4, b_3\ mod\ b_4) = \gcd(b_4, b_5)$$
$$\dots$$
$$\dots$$
$$b_{m-1}\ mod\ b_m\ \equiv\ 0$$

oluncaya kadar devam eder. Sonuç $\gcd(b_1, b_2) = b_m$ olur.

Yukarıdaki yinelemeli algoritmada $b_1 > b_2$'yi kabul etmemize rağmen, hangisi büyük olduğuna bakmaksızın herhangi iki negatif olmayan b_1 ve b_2 tam sayıları için çalışacağı unutulmamalıdır. Birinci tam sayı ikinciden küçük ise ilk adımda ikinci sayı ile yer değiştirilir.

2.2.8 Sonlu Cisimler

Daha önceki bölümlerde Z_n'nin kalanlar kümesinin değişmeli bir halka olduğundan bahsedilmiştir. Bunun sebebi ise Z_n'nin sadece değişmeli bir halka olması ve sonlu bir cisim olmamasıdır. Çünkü Z_n'deki her elemanın çarpmaya göre tersi olmayabilir. Yukarıda da gösterildiği gibi, Z_n'nin elemanı olan "a", eğer ki $a\ mod\ n$'ye göre aralarında asal değil ise a'nın çarpmaya göre tersi yoktur.

Bir sayının asal sayı olması için $mod\ n$ seçilirse, sayının asal olabilmesi için bir ve kendisinden başka böleni olmamalıdır. "n" asal sayısı için sıfırdan farklı her $a \in Z_n$ "n" sayısı ile aralarında asal olacaktır. Her sıfırdan farklı $a \in Z_n$ için n asal sayısına göre çarpmaya göre bir tersi vardır.

"p"nin asal bir sayı olduğu varsayılır ise, Z_p sonlu bir cisimdir. Z_p asal sonlu cisim olarak da ifade edilebilir. Bir cismi ayrıca $GF(p)$ şeklinde de ifade edilebilir. Burada GF Galois Cismi olarak adlandırılır. "p" asal sayısı için Z_p'nin sıfırdan farklı her bir elemanın çarpmaya göre bir tersi vardır. Çelişki yöntemiyle kanıtlanırsa; sıfırdan farklı $a \in Z_p$'nin b ve c şeklinde farklı iki tane çarpmaya göre tersi olsun. Başka bir deyişle;

$a * b = 1\ mod\ p$ ve $a * c = 1\ mod\ p$

$a * (b - c) \equiv 0$ mod p $\equiv$ p mod p şeklinde yazılabilir.

Ancak bu imkansızdır çünkü bir asal sayı bu şekilde çarpanlarına ayrılamaz.

2.2.9 Z_n Kümesinin Elemanlarının Çarpmaya Göre Tersi

Genel olarak $a \in Z_n$'nin çarpmaya göre tersini bulmak için $a * b \equiv 1$ mod n denkliğini sağlayan bir $b \in Z_n$ bulunmalıdır. Şimdiye kadar bahsedilenler göz önüne alındığında $ebob(a, n) = 1$ koşulunu sağlayan tüm $a \in Z_n$'lerin çarpmaya göre tersi vardır. "n" sayısı "p" asal sayısına eşit olduğu sürece Z_n'nin tüm elemanları için geçerlidir.

Modüler aritmetikte verilen bir "a" tam sayısının çarpmaya göre tersini bulmak için Bezout'un biriminden yararlanılır. Genel olarak a ve n gibi herhangi bir pozitif tam sayı çifti olduğunda aşağıdaki denklem her zaman x ve y (pozitif, negatif veya sıfır olabilir) tam sayıları için gösterilebilir.

$$ebob(a, n) = x * a + y * n \qquad \text{(Denklem 2.1 Bezout birimi)}$$

Denklem 2.1 Bezout Birimi olarak adlandırılıyor.

Örneğin $a = 16$ ve $n = 6$ olsun, $ebob(16,6) = 2$'dir. $2 = (-1) * 16 + 3 * 6 = 2 * 16 + (-5) * 6$ şeklinde de yazabiliriz. Bu durum verilen a ve n sayılarının Bezout Biriminde herhangi bir eşitsizliğe sahip olmadığını gösterir.

2.2.9.1 Kalanlar ile GCD Yinelemesini Tekrar Yazmak (Genişletilmiş Euclid Algoritması)

Burada tek odaklanacağımız nokta bölüm 2.2.7'deki yinelemeli yapıdaki kalanlar olacaktır.

0 kalanına ulaşmadan önce, sondan bir önceki kalanın 1 olduğundan emin olunmalıdır. Başka bir deyişle eğer sayılar aralarında asal ise bu kalan muhtemelen iki sayının ebob'udur.

$\gcd(b_1, b_2)$:

$$b_3 = b_1 - q_1.b_2$$

$$\begin{aligned} b_4 &= b_2 - q_2.b_3 \\ &= b_2 - q_2.(b_1 - q_1.b_2) \\ &= b_2 - q_2.b_1 + q_1.q_2.b_2 \\ &= -q_2.b_1 + (1 + q_1.q_2).b_2 \end{aligned}$$

$$\begin{aligned} b_5 &= b_3 - q_3.b_4 \\ &= (b_1 - q_1.b_2) - q_3.(-q_2.b_1 + (1 + q_1.q_2).b_2) \\ &= b_1 + q_2q_3.b_1 - q_1.b_2 - q_3.(1 + q_1.q_2).b_2 \\ &= (1 + q_2.q_3).b_1 - (q_1 - q_1.q_2 - q_3).b_2 \end{aligned}$$

...

$$b_m = (\dots).b_1 + (\dots).b_2$$

b_m 1'e eşit olduğunda algoritma durmalıdır. Çünkü burada b_1 ve b_2 aralarında asal olmaktadır. Aksi durumda b_m 0'a eşit olduğunda b_1'in $mod\ b_2$'de çarpmaya göre tersi yoktur.

Eğer b_m 1'e eşit olduğu için algoritma durduysa, o zaman b_m için b_1'in çarpmada genişletilmiş hâli, $mod\ b_2$'de b_1'in çarpmaya göre tersidir. Yukarıda bahsedilen adımlar uygulandığında, genişletilmiş Euclid algoritması elde edilir.

Genişletilmiş Euclid Algoritması Örnekleri

- mod 17'de 32'nin çarpmaya göre tersi

$$\gcd(32,17)$$
$$= \gcd(17,15) \mid \text{kalan } 15 = 1 * 32 - 1 * 17$$
$$= \gcd(15,2) \mid \text{kalan } 2 = 1 * 17 - 1 * 15$$
$$\mid = 1 * 17 - 1 * (1 * 32 - 1 * 17)$$
$$\mid = (-1) * 32 + 2 * 17$$
$$= \gcd(2,1) \mid \text{kalan } 1 = 1 * 15 - 7 * 2$$
$$\mid = 1 * (1 * 32 - 1 * 17) - 7 * ((-1) * 32 + 2 * 17)$$
$$\mid = 8 * 32 - 15 * 17$$

Buradan mod 17'de 32'nin çarpmaya göre tersini 8 buluruz.

2.2.10 Polinom Aritmetiği

Sonlu cisimleri polinom kümeleri üzerinde tanımlamak, dijital hesaplama için özellikle uygun olan sınırlı sayıda sayı oluşturulmasını sağlayacaktır. Bu sayılar sonlu bir cisim oluşturduğundan, hatasız bir şekilde tüm aritmetik işlemleri gerçekleştirebilmektedir.

2.2.10.1 Polinom Halkaları

R birimli bir halka, x bir belirsiz, $n \in$ N ve $0 \leq i \leq n$ için $a_i \in$ Rolmak üzere $f(x) = a_0x^0 + a_1x^1 + \cdots + a_nx^n$ ifadesine R katsayılı x'e göre bir polinom denir. $0 \leq i \leq n$ için $a_i \in$ R elemanlarına $f(x)$ polinomunun katsayıları ve a_ix^i'ye de $f(x)$ polinomunun terimleri adı verilir. R üzerinde x belirsizine göre bütün polinomların kümesi $R[x]$ ile gösterilir.

R birimli bir halka, x bir belirsiz, $n, m \in$ N ve

$$f(x) = a_0x^0 + a_1x^1 + \cdots + a_nx^n \in \text{R[x]}$$

$$g(x) = b_0x^0 + b_1x^1 + \cdots + b_mx^m \in \text{R[x]}$$

olmak üzere eğer her $0 \leq i \in$ Z için $a_i = b_i$ ise $f(x)$ ve $g(x)$ polinomları eşittir denir ve $f(x) = g(x)$ ile gösterilir ($i > n$ için $a_i = 0$ ve $i > m$ için $b_i = 0$ şeklinde tanımlanır).

R birimli bir halka, x bir belirsiz ve $n, m \in$ N için

$$f(x) = a_0x^0 + a_1x^1 + \cdots + a_nx^n \in \text{R[x]}$$

$$g(x) = b_0x^0 + b_1x^1 + \cdots + b_mx^m \in \text{R[x]}$$

olsun. $k = maks\{m, n\}$ ve $c_i = \sum_{j=0}^{i} a_j b_{i-j}$ olmak üzere

$f(x) + g(x) = \sum_{i=0}^{k} (a_j + b_i)x^i$, $f(x) \,.\, g(x) = \sum_{i=0}^{m+n} c_i\, x^i$

Şeklinde tanımlı işlemlerle $(R[x], +, .)$ birimli bir halkadır. Bu halkaya R üzerindeki polinomlar halkası denir. Eğer R değişmeli ise $R[x]$ halkası da değişmelidir.

Bir polinom Denklem 2.6'daki gibi ifade edilir. Burada "n"ler negatif olmayan tam sayılardır. "S" kümesi şeklinde ifade edilen ve $a_0, a_1, \ldots, a_n$'den oluşan kümeye katsayılar kümesi denir.

$$a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0 \qquad \text{(Denklem 2.6 Polinom Denklem)}$$

$a_n \neq 0$ olduğu durumda "n." dereceden bir polinomdur. Sıfırıncı dereceden bir polinoma ise sabit polinom denir. Polinomlar üzerinde de toplama, çıkarma, çarpma ve bölme gibi işlemler gerçekleştirilir.

2.2.10.2 Polinomlar Üzerinde Aritmetik İşlemler

- İki polinomu toplama

$$f(x) = a_2x^2 + a_1x + a_0$$
$$g(x) = b_1x + b_0$$
$$f(x) + g(x) = a_2x^2 + (a_1 + b_1)x + (a_0 + b_0)$$

- İki polinomu çıkartmak

$$f(x) = a_2x^2 + a_1x + a_0$$

$$g(x) = b_3x^3 + b_0$$

$$f(x) - g(x) = -b_3x^3 + a_2x^2 + a_1x + (a_0 - b_0)$$

- İki polinomu çarpmak

$$f(x) = a_2x^2 + a_1x + a_0$$

$$g(x) = b_3x^3 + b_0$$

$$f(x) * g(x) = a_2b_1x^3 + (a_2b_0 + a_1b_1)x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$$

Bir Polinomun Diğerine Bölümü

$8x^2 + 3x + 2$ ikinci dereceden polinomunu $2x + 1$ birinci dereceden polinomuna bölmek istendiğinde; aşağıdaki adımları gerçekleştirilir:

- Hem bölüneni hem de bölenin kuvvetlerini büyükten küçüğe doğru azalan şekilde yazılmalıdır.
- Bölünenin ilk terimini bölenin ilk terimine bölünmeli ve sonucu bölümün ilk terimi olarak yazılmalıdır. Bu örnekte bölümün ilk terimi $8x^2$ ve bölenin ilk terimi $2x$'dir. Buradan bölümün ilk terimi $4x$'dir.
- Böleni, elde edilen bölüm terimiyle çarpılır ve sonuç bölünenin altına yazılır böylece x'in aynı kuvvetleri eşleşecektir. Bölümden, elde edilen ifadeyi çıkarıyoruz. Örnekte, $4x$ kere $2x + 1$, $8x^2 + 4x$'e eşittir. Bölünenden bu çıkarıldığında kalan $-x + 2$'dir.

İşlemin sonucu olarak $8x^2 + 3x + 2$'in $2x + 1$'e bölümünden $4x - 0{,}5$ bölüm ve $2{,}5$ de kalandır. Başka bir deyişle $(8x^2 + 3x + 2)/(2x + 1) = 4x - 0{,}5 + 2{,}5/(2x + 1)$ şeklinde gösterilebilir.

Sonlu cisimlerde bölme işlemi yoktur bunun yerine ters alma ve çarpma işlemi vardır.

Katsayıları Sonlu Cisimde Olan Polinomlar Üzerinde Aritmetik İşlemler

Katsayılar Z_7 $(GF(7))$'de olan sonlu cismine ait olan tüm polinomlar sınıfını düşünüldüğünde;

- Z_7 kümesinde iki polinomun toplamı

$$f(x) = 5x^2 + 4x + 6$$

$$g(x) = 2x + 1$$

$$f(x) + g(x) = 5x^2 + 6x$$

- Z_7 kümesinde iki polinomun farkı

$$f(x) = 5x^2 + 4x + 6$$

$$g(x) = 2x + 1$$

$$f(x) - g(x) = 5x^2 + 2x + 5$$

Z_7'de 2'nin toplamaya göre tersi 5 ve 1'in 6'dır. Buradan $4x - 2x$ de aynı şekilde $4x + 5x$'dir ve "$6 - 1$" ile "$6 + 6$" $mod\ 7$'de aynıdır.

- Z_7 kümesinde iki polinomun çarpımı

$$f(x) = 5x^2 + 4x + 6$$

$$g(x) = 2x + 1$$

$$f(x) * g(x) = 3x^3 + 6x^2 + 2x + 6$$

- Z_7 kümesinde iki polinomun bölümü

$$f(x) = 5x^2 + 4x + 6$$

$$g(x) = 2x + 1$$

$$f(x)/g(x) = 6x + 6$$

2.2.11 Sonlu Cisimler Üzerinde Polinom Bölmesi

$f(x)$, $g(x) \in F[x]$ olmak üzere $g(x) = f(x)h(x)$ olacak biçimde $h(x) \in$ F[x] varsa $f(x)$, $g(x)$'i böler veya $g(x)$, $f(x)$ ile bölünür denir ve $f(x) | g(x)$ ile gösterilir. Bu durumda $g(x)$ polinomuna $f(x)$'in bir katı, $f(x)$ polinomuna da $g(x)$'in bir böleni adı verilir.

Polinomlar İçin Bölüm Algoritması

$f(x)$, $g(x) \in F[x]$ ve $f(x) \neq 0$ olsun.

Bu durumda $g(x) = f(x)q(x) + r(x) = 0$ veya $derr(x) < derf(x)$ olacak biçimde tek türlü $q(x)$, $r(x) \in F[x]$ vardır.

Bir polinomun katsayıları cisimlerden oluşmuşsa bir cisim üzerinde tanımlandığı söylenebilir. Sonlu cisimler üzerinde polinom bölmesi yapmak diğer aritmetik işlemlerden biraz daha uzundur. Bunun sebebi işlemleri yaparken hem toplamaya hem de çarpmaya göre ters işlemlerinin yapılmasıdır.

Örnek olarak $5x^2 + 4x + 6$'yı $2x + 1$'e bölünsün. Öncelikle $5x^2$'yi $2x$'e bölerek başlanır. Bu $GF(7)$'de 5'i 2'ye bölmektir. 5 sayısını 2'ye bölmek ile 5'i 2'nin çarpmaya göre tersi ile çarpmak aynıdır. 2'nin çarpmaya göre tersi 4'tür. Çünkü $2 * 4\ mod\ 7$'de 1'dir. Buradan;

$5/2 = 5 * 2^{-1} = 5 * 4 = 20\ mod\ 7 = 6$'dır.

Burada bölümün ilk terimi $6x$'dir. $6x$ ve $2x + 1$'in çarpımı $5x^2 + 6x$ olduğundan, bölünen $5x^2 + 4x + 6$'dan $5x^2 + 6x$'i çıkarılır. Sonuç $(4 - 6)x + 6$, ki buda (6'nın toplamaya göre tersi 1 olduğu için) $(4 + 1)x + 6$ ile aynıdır.

Bir sonraki aşamada yeni bölünenimiz $5x + 6$'dır. Yeni bölüm terimini bulmak için, bölenin ilk terimiyle $5x$'i bölmek gerekmektedir (yani $2x$ ile). Bunun sonucunda bölümün bir sonraki terimi tekrar 6 oluyor.

Sonuç olarak katsayıları $GF(7)$ kümesinden oluşan $5x^2 + 4x + 6$'nın $2x + 1$'e bölümünden $6x + 6$ bölüm ve sıfır kalanı elde edilir. Buradan;

$5x^2 + 4x + 6 = (2x + 1) * (6x + 6)$ şeklinde yazılabilir.

2.2.12 $GF(2)$ Üzerinde Tanımlı Polinomlar

$GF(2)$ ile Z_2 izomorftur. $Z_2\ mod\ 2$ ile ilişkilidir.$GF(2)$ küçük bir sonlu cisimdir. $GF(2)$, $\{0, 1\}$ kümesinin elemanlarından oluşmaktadır. Bu kümenin toplama, çıkarma ve çarpma kuralları aşağıdaki gibidir.

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$

$0 - 0 = 0$

$1 - 0 = 1$

$0 - 1 = 0 + 1 = 1$

$1 - 1 = 1 + 1 = 0$

$0 * 0 = 0$

$0 * 1 = 0$

$1 * 0 = 0$

$1 * 1 = 1$

Buradan çıkarılacak sonuç $GF(2)$ üzerindeki toplama işlemi lojik XOR işlemine ve çarpma işlemi de lojik AND işlemine denk olduğudur.

$GF(2)$'de Tanımlı Polinomlar Üzerinde Aritmetik İşlemler

- $GF(2)$ üzerinde iki polinomun toplamı

$$f(x) = x^2 + x + 1$$
$$g(x) = x + 1$$
$$f(x) + g(x) = x^2$$

- $GF(2)$ üzerinde iki polinomun farkı

$$f(x) = x^2 + x + 1$$
$$g(x) = x + 1$$
$$f(x) - g(x) = x^2$$

- $GF(2)$ üzerinde iki polinomun çarpımı

$$f(x) = x^2 + x + 1$$
$$g(x) = x + 1$$
$$f(x) * g(x) = x^3 + 1$$

- $GF(2)$ üzerinde iki polinomun bölümü

$$f(x) = x^2 + x + 1$$
$$g(x) = x + 1$$

$$f(x)/g(x) = x + 1/(x + 1)$$

Bölen $x + 1$ ile bölüm x'in çarpımı $x^2 + x$'dir. Daha sonra üzerine kalan 1 eklendiğinde elde edilen sonuç $x^2 + x + 1$'dir.

2.2.13 İndirgenemez (Asal) Polinomlar

$g(x)$ polinomu $f(x)$'i kalansız bölüyor ise $g(x)$, $f(x)$'in bir çarpanıdır. Bir F cismi üzerinde tanımlı $f(x)$ polinomu herhangi iki polinomun çarpımı sonucunda elde edilemiyorsa $f(x)$'e indirgenemez denir. Bir indirgenemez polinom ayrıca asal polinom olarak da adlandırılır.

2.2.14 $GF(2^n)$ Formundaki Sonlu Cisimler

$GF(2)$ sonlu cismi bir grup operatörü olarak $mod\ 2$ toplaması, halka operatörü olarak ise $mod\ 2$ çarpmasından ve $\{0, 1\}$ kümesinden oluşmaktadır. $GF(2)$'de polinom örnekleri olarak $\{x + 1,\ x^2 + x + 1,\ x^2 + 1,\ x^3 + 1,\ x,\ 1,\ x^4, \ldots\}$ verilebilir.

Tüm bu polinomların katsayılarında 0 ve 1 kullanılmıştır. Açıkçası negatif katsayıya sahip polinomlarda gösterilebilir ancak daha öncede bahsedildiği gibi $GF(2)$'de "-1" ile "$+1$" aynıdır. Bu polinomlar $GF(2)$ sonlu cisminin kurallarına uygun bir şekilde yapıldığı sürece toplama ve çıkarma işlemlerinde kullanılabilir. Bir halkadan oluşan bu şekilde polinomlara, polinom halkası denir.

2.2.15 Modüler Polinom Aritmetiği

İndirgenemez bir polinom kendisinden daha küçük dereceli polinomlara dönüştürülemez. $GF(2)$ üzerinde tanımlı tüm polinomlar kümesinden, $GF(2)$ üzerinde tanımlı 3. dereceden tanımlı sadece iki tane indirgenemez polinom vardır. Bunlar;

$x^3 + x + 1$ ve $x^3 + x^2 + 1$'dir

$GF(2)$ üzerinde tanımlı tüm polinomlar kümesi için, polinom aritmetiğini mod indirgenemez $x^3 + x + 1$ polinomuyla yapılacaktır. Başka bir deyişle bir

polinom aritmetiği yapıldığında sonucu indirgenemez polinom ile mod'u alınacaktır. Örneğin, bir polinom çarpması yapıldığında elde edilen polinomun derecesi indirgenemez polinoma eşit veya daha büyük ise elde edilen polinomu indirgenemez polinom ile mod işlemine tabi tutulduğunda kalan polinom sonuç olmaktadır.

Örnek olarak;

$$(x^2 + x + 1) * (x^2 + 1)\ mod\ (x^3 + x + 1)$$
$$= (x^4 + x^3 + x^2) + (x^2 + x + 1)\ mod\ (x^3 + x + 1)$$
$$= (x^4 + x^3 + x + 1)\ mod\ (x^3 + x + 1)$$
$$= -x^2 - x$$
$$= \ x^2 + x$$

$GF(2)$'de $1 + 1 = 0$'dır bu nedenle x^2'li terim işlem sonucunda silinmiştir.

3. ABC KRİPTOSİSTEMİ

Şifreleme teknikleri, modern toplumda iletişimin güvenliğini garanti altına almak için önemli bir araçtır. Bu alandaki en iyi bilinen algoritmalar RSA, DSA ve ECC'dir (Rivest and Shamir and Adleman, 1978). Ancak, yeterince büyük kuantum bilgisayarlar geldiğinde bunlar gibi algoritmalar güvensiz hale gelecektir. Bunun nedeni, tam sayılı çarpanlara ayırma ve kuantum bilgisayarlarda polinom zamanlarında ayrık logaritmalar gibi sayılar teorisi problemlerini çözen Shor (Shor,1997) algoritmasıdır. Bu nedenle, kuantum bilgisayar saldırılarından etkilenmeyen matematiksel problemlere dayanan klasik açık anahtarlı şifreleme sistemlerine alternatifler gerekir.

Kafes, kod ve hash tabanlı şifreleme sistemlerinin yanı sıra, çok değişkenli şifreleme de bunun için ana adaylardan biridir (Bernstein, Buchmann and Dahmen, 2009). Çok değişkenli şemalar çok hızlıdır ve yalnızca hesaplama kaynakları gerektirir; bu da akıllı kartlar ve RFID çipleri gibi düşük maliyetli cihazlarda kullanım için onları çekici kılar. Bununla birlikte, birçok pratik çok değişkenli imza şeması (Ding and Schmidt, 2005) mevcut olsa da, verimli ve güvenli çok değişkenli şifreleme şemalarının sayısı sınırlıdır.

Açık anahtar olarak bir kübik şema elde edilip, rastgele ikinci dereceden polinomlara sahip kare matrisler kullanılıyor. Şemaya yapılan bir cebirsel saldırının, en azından aynı büyüklükteki rastgele ikinci dereceden bir sistemi çözmek kadar zor olduğu varsayılıyor (Ding, Petzoldt and Wang, 2009).

ABC, kuantum sonrası bilgisayarlar için öngörülen bir şifreleme yöntemidir. Son zamanlarda Tao (Tao, Diene, Tang and Ding, 2013) "basit matris şeması" veya "ABC" olarak adlandırılan matris çarpımına dayalı yeni bir basit ve verimli çok değişkenli açık anahtar şifreleme düzeni sunmuştur. Daha sonrasında, Ding, Petzoldt ve Wang (Tao, Xiang, Petzoldt and Ding, 2015) kübik polinomları gelişmiş bir ABC türevini önerdiler ve en azından rastgele ikinci dereceden bir denklem çözme kadar zor olan cebirsel saldırıları kullanarak bunu kırdıklarını gösterdiler. Çok yakın bir zamanda Tao, Xiang, Petzoldt ve Ding (Ding, Petzoldt and Wang, 2014) ABC şemasını kare matris yerine kare olmayan matris kullanarak

genelleştirdiler. Petzoldt, Ding ve Wang'ın ABC yapısında şifre çözme hatalarını ortadan kaldırmak için, matrislerin tensör çarpımını kullanan yeni bir ABC sürümünü önermiştir (Petzoldt, Ding and Wang, 2016). Bununla birlikte Hashimoto (Hashimoto, 2016), bu türevin güvenliğinin ABC kökenli şemadan daha zayıf olduğunu göstermiştir. Daha sonra, Peng, Tang, Chen, Wu ve Zhang (Peng, Tang, Chen, Wu and Zhang, 2016) verimliliği artırmak için modern x64 CPU'nun özelliklerinden yararlanarak ABC'nin uygulanmasını optimize etmiştir.

3.1 Temel ABC Şifreleme Şeması ve Çok Değişkenli Şifreleme

Verimlilik nedenleriyle, çok değişkenli polinom sistemleri genellikle K gibi q elemanlı bir sonlu cisim üzerinde birçok değişkene sahip ikinci dereceden polinom sistemidir.

Denklem 3.1. Çok değişkenli ikinci dereceden polinomlar.

$$p^{(1)}(x_1, \dots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}^{(1)} x_i x_j + \sum_{i=1}^{n} p_i^{(1)} x_i + p_0^{(1)}$$

$$p^{(2)}(x_1, \dots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}^{(2)} x_i x_j + \sum_{i=1}^{n} p_i^{(2)} x_i + p_0^{(2)}$$

...

$$p^{(m)}(x_1, \dots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij}^{(m)} x_i x_j + \sum_{i=1}^{n} p_i^{(m)} x_i + p_0^{(m)}$$

Çok değişkenli şifreleme sisteminin güvenliği, çok değişkenli ikinci dereceden polinom (MQ) problemine dayanır. Denklem 3.1'de gösterildiği gibi $p^{(1)}(x), p^{(2)}(x), \dots, p^{(m)}(x)$ olarak verilen m değişkenli ikinci dereceden polinomlar $p^{(1)}(\bar{x}) = 0, \dots, p^{(m)}(\bar{x}) = 0$ olacak şekilde bir $\bar{x} = (\bar{x}_1, \dots, \bar{x}_n)$ vektörü bulunur.

MQ problemi ($m \cong n$ için) $GF(2)$ cismi üzerindeki ikinci dereceden polinomlar için bile NP-zor problem olduğu kanıtlanmıştır (Garey and Johnson,1979).

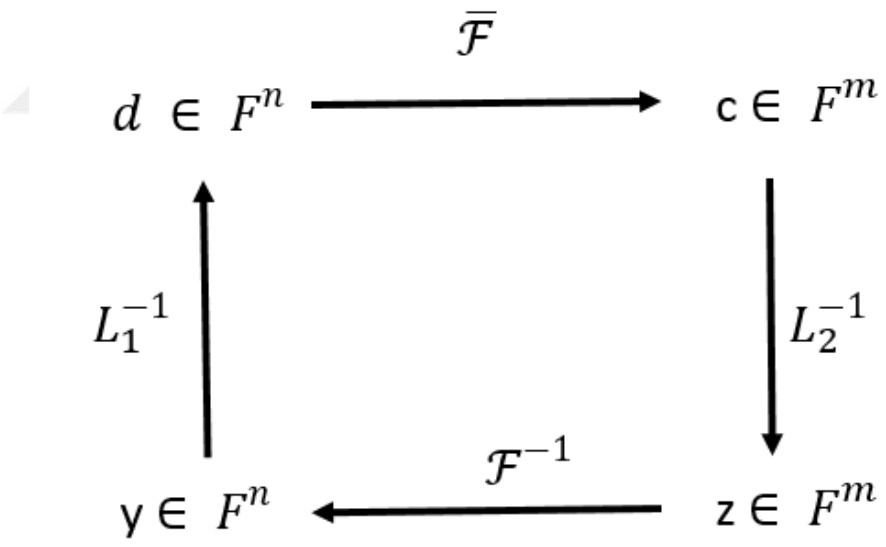
MQ probleminin temelli bir açık anahtarlı kriptosistem oluşturmak için, kolayca tersi alınabilir bir ikinci dereceden $\mathcal{F} : F^n \rightarrow F^m$ polinomlar ile başlanır. Açık anahtarlıda F'nin yapısını gizlemek için, iki tane tersi olan afin (veya lineer) $L_1 : F^n \rightarrow F^n$ ve $L_2 : F^m \rightarrow F^m$ yapısı oluşturulur.

Açık anahtarı oluşturacak yapı, $\bar{\mathcal{F}} = L_2 \; o \; F \; o \; L_1$.

Gizli anahtar $L_1, \mathcal{F}$ ve L_2'den oluşur ve bu sayede açık anahtarın tersinin alınmasına olanak sağlar.

Çok değişkenli polinomlarda, şifreleme ve şifre çözme işlemlerinin çalışma süreci Şekil 3.1'de verilmiştir.

Şekil 3.1 ABC Kriptosisteminin Şeması



Şifreleme: $d \in F^n$ şifrelemek için, tek bir basit hesaplama $c = \bar{\mathcal{F}}(d)$ yapılması yeterlidir. d mesajının şifrelenmiş hali $c \in F^m$'dir.

Şifre Çözme: $c \in F^m$ şifrelenmiş mesajı çözmek için, özyinelemeli olarak $z = L_2^{-1}(c), y = \mathcal{F}^{-1}(z)$ ve $d = L_1^{-1}(y)$ işlemleri yapılmalıdır. $d \in F^n$, şifrelenmiş olan c metnine karşılık gelen düz metindir.

3.2 ABC Kriptosisteminin Yapısı

Anahtar Üretimi: F, q elemana sahip sonlu bir cisim olarak ele alınır. $s \in S$ elemanı için, $n = s^2$ ve $m = 2n$ koşullarını oluşturur ve aşağıdaki gibi üç matris tanımlanır.

$$A = \begin{pmatrix} x_1 & \cdots & x_s \\ \vdots & \ddots & \vdots \\ x_{(s-1)(s+1)} & \cdots & x_n \end{pmatrix}, B = \begin{pmatrix} b_1 & \cdots & b_s \\ \vdots & \ddots & \vdots \\ b_{(s-1)(s+1)} & \cdots & b_n \end{pmatrix}, C = \begin{pmatrix} c_1 & \cdots & c_s \\ \vdots & \ddots & \vdots \\ c_{(s-1)(s+1)} & \cdots & c_n \end{pmatrix}$$

Burada $(x_1, \dots, x_n)$'ler $F[x_1, \dots, x_n]$ çok değişkenli polinomun lineer monomialidir. Ayrıca buradaki $(b_1, \dots, b_n)$ ve $(c_1, \dots, c_n)$'ler, $(x_1, \dots, x_n)$'in lineer birleşiminden rastgele seçilir.

$E_1 = AB$ ve $E_2 = AC$ çarpımlarından E_1 ve E_2 matrisleri elde edilir. Şemadaki $\bar{\mathcal{F}}$, E_1 ve E_2'nin m bileşenlerinden oluşur.

Şemada açık anahtar, rastgele seçilmiş iki tersinir $L_2 : F^m \to F^m$ ve $L_1 : F^n \to F^n$ lineer denkleme sahip $\bar{\mathcal{F}} = L_2 \, o \, F \, o \, L_1 : F^n \to F^m$ denkleminden meydana gelir. Gizli anahtar ise B ve C matrislerinden ve L_1 ve L_2 lineer denklemlerinden meydana gelir.

Şifreleme: Bir $d \in F^n$ mesajını şifrelemek için tek bir hesaplama $c = \bar{\mathcal{F}}(d) \in F^m$ yapılması yeterlidir.

Şifre Çözme: Şifrelenmiş $c \in F^m$ mesajını çözmek için aşağıdaki üç adımı takip edilir.

i. $z = L_2^{-1}(c)$'yi hesaplanır. $z \in F^n$ vektörünün elemanlarından aşağıdaki gibi $\overline{E_1}$ ve $\overline{E_2}$ matrisleri oluşturulur.

$$E_1 = \begin{pmatrix} z_1 & \cdots & z_s \\ \vdots & \ddots & \vdots \\ z_{(s-1)(s+1)} & \cdots & z_n \end{pmatrix}, E_2 = \begin{pmatrix} z_{n+1} & \cdots & z_{n+s} \\ \vdots & \ddots & \vdots \\ z_{n+(s-1)(s+1)} & \cdots & z_m \end{pmatrix}$$

ii. İkinci adımda ise $y = (y_1, \dots, y_n)$ şeklinde $\mathcal{F}(y) = z$ koşulunu sağlayan bir vektör bulunur. Bunun yapılabilmesi için dört farklı koşul vardır.

   a. Eğer $\overline{E_1}$'in tersi varsa, $B\overline{E_1^{-1}}\overline{E_2} - C = 0$ denklemi düşünülür.
   b. Eğer $\overline{E_1}$'in tersi yoksa ama $\overline{E_2}$'nin varsa, $C\overline{E_2^{-1}}\overline{E_1} - B = 0$ denklemi düşünülür.
   c. Eğer ne $\overline{E_1}$ ne de $\overline{E_2}$'nin tersi yoksa ama $\bar{A} = A(y)$'nın tersi varsa, $\overline{A^{-1}}\overline{E_1} - B = 0$ ve $\overline{A^{-1}}\overline{E_2} - C = 0$ denklemleri düşünülür.
   d. Eğer $\overline{E_1}$, $\overline{E_2}$ ve $\bar{A}$'nın hiçbirinin tersi yok ise şifre çözme başarısızdır.

iii. Son olarak, çözümlenmiş mesajı bulmak için $d = L_1^{-1}(y_1, \dots, y_n)$ hesaplaması yapılır.

İkinci adımdaki şifre çözmede başarısızlık oranı $1/q$'dur.

Şifre çözme işleminin ikinci basamağındaki lineer sistemlerin $y^{(1)}, \dots, y^{(l)}$ gibi birden fazla çözümü olabilir. Böyle bir durumda bir dizi olası şifresi çözülmüş mesaj elde etmek için (her bir dizi için) üçüncü adımın gerçekleştirilmesi gerekir. Daha sonrasında bu düz metinler teker teker şifrelenerek hangisinin verilen şifreli mesaja karşılık geldiği test edilir.

3.3 ABC Kriptosistemi için Gerekli Algoritmalar

ABC kriptosisteminin uygulamasını yazmak için gerekli temel algoritmalar bulunmaktadır. Bunlar; GF(2^8) sonlu cismi üzerinde "toplama", "çıkarma", "çarpma" ve "bölme" işlemleri, GF(2^8)'deki bir elemanın "tersini alma", verilen herhangi bir kare matrisin "tersini" ve "transpozunu" alma, herhangi iki kare "matrisin çarpımını" bulma ve bir matrisin "Echelon matrisini" hesaplamadır. Bunlara ek olarak "anahtar üretimi", "şifreleme" ve "şifre çözme" için algoritmalar da vardır. Algoritmalardaki değerlerin bazıları ve fonksiyon geri dönüş veri tipi "unsigned char" olarak tanımlanmıştır. Okunabilirliği arttırmak amacıyla "unsigned char" veri tipini "WORD" ismine ataması yapılır.

3.3.1 $GF(2^8)$ Cisminin Elemanları

$GF(2^8)$'de sonlu cisminde uygulama oluşturulduğu için eleman sayısı 256 tanedir ve bunlar "Logtable" ve "Alogtable" isminde iki farklı dizide tutulur. Ama farklı değerlerdeki cisimlerde de işlem yapabilmesi için "FIELD" isminde bir integer tanımlanır ve istenildiği şekilde değiştirilebilir.

```
WORD Logtable[FIELD] = { 0, 0, 25, 1, 50, 2, 26, 198, 75, 199, 27, 104, 51, 238, 223, 3,100,
4, 224, 14, 52, 141, 129, 239, 76, 113, 8, 200, 248, 105, 28, 193, 125, 194, 29, 181, 249, 185,
39, 106, 77, 228, 166, 114, 154, 201, 9, 120, 101, 47, 138, 5, 33, 15, 225, 36, 18, 240, 130, 69,
53, 147, 218, 142, 150, 143, 219, 189, 54, 208, 206, 148, 19, 92, 210, 241, 64, 70, 131, 56, 102,
221, 253, 48, 191, 6, 139, 98, 179, 37, 226, 152, 34, 136, 145, 16, 126, 110, 72, 195, 163, 182,
30, 66, 58, 107, 40, 84, 250, 133, 61, 186, 43, 121, 10, 21, 155, 159, 94, 202, 78, 212, 172, 229,
243, 115, 167, 87, 175, 88, 168, 80, 244, 234, 214, 116, 79, 174, 233, 213, 231, 230, 173, 232,
44, 215, 117, 122, 235, 22, 11, 245, 89, 203, 95, 176, 156, 169, 81, 160, 127, 12, 246, 111, 23,
196, 73, 236, 216, 67, 31, 45, 164, 118, 123, 183, 204, 187, 62, 90, 251, 96, 177, 134, 59, 82,
161, 108, 170, 85,41, 157, 151, 178, 135, 144, 97, 190, 220, 252, 188, 149, 207, 205, 55, 63, 91,
209, 83, 57, 132, 60, 65, 162, 109, 71, 20, 42, 158, 93, 86, 242, 211, 171, 68, 17, 146, 217, 35,
32, 46, 137, 180, 124, 184, 38, 119, 153, 227, 165, 103, 74, 237, 222, 197, 49, 254, 24, 13, 99,
140, 128, 192, 247, 112, 7};
```

```
WORD Alogtable[FIELD] = {1, 3, 5, 15, 17, 51, 85, 255, 26, 46, 114, 150, 161, 248, 19, 53, 95,
225, 56, 72, 216, 115, 149, 164, 247, 2, 6, 10, 30, 34, 102, 170, 229, 52, 92, 228, 55, 89, 235,
38, 106, 190, 217, 112, 144, 171, 230, 49, 83, 245, 4, 12, 20, 60, 68, 204, 79, 209, 104, 184,
211, 110, 178, 205, 76, 212, 103, 169, 224, 59, 77, 215, 98, 166, 241, 8, 24, 40, 120, 136, 131,
158, 185, 208, 107, 189, 220, 127, 129, 152, 179, 206, 73, 219, 118, 154, 181, 196, 87, 249, 16,
48, 80, 240, 11, 29, 39, 105, 187, 214, 97, 163, 254, 25, 43, 125, 135, 146, 173, 236, 47, 113,
147, 174, 233, 32, 96, 160, 251, 22, 58, 78, 210, 109, 183, 194, 93, 231, 50, 86, 250, 21, 63, 65,
195, 94, 226, 61, 71, 201, 64, 192, 91, 237, 44, 116, 156, 191, 218, 117, 159, 186, 213, 100,
172, 239, 42, 126, 130, 157, 188, 223, 122, 142, 137, 128, 155, 182, 193, 88, 232, 35, 101, 175,
234, 37, 111, 177, 200, 67, 197, 84, 252, 31, 33, 99, 165, 244, 7, 9, 27, 45, 119, 153, 176, 203,
70, 202, 69, 207, 74, 222, 121, 139, 134, 145, 168, 227, 62, 66, 198, 81, 243, 14, 18, 54, 90,
238, 41, 123, 141, 140, 143, 138, 133, 148, 167, 242, 13, 23, 57, 75, 221, 124, 132, 151, 162,
253, 28, 36, 108, 180, 199, 82, 246, 1};
```

3.3.2 Toplama ve Çıkarma İşlemi

Toplama işlemi yaparken $GF(2^8)$'de tanımlı elemanlar kullanılır. Çıkarma işlemi ise toplamanın tersi olduğundan algoritma olarak aynıdır. Veri tipi olarak yukarıda oluşturulan "WORD" tipini kullanılır.

```
Toplama(a, b)
   return a+b
```

```
Çıkarma(a, b)
    return a+b
```

3.3.3 Çarpma İşlemi

Çarpma işleminde GF(2^8)'de tanımlı herhangi iki eleman çarpılır.

```
Çarpma(a, b)
     if(a && b)
          return Alogtable[(Logtable[a] + Logtable[b]) % (FIELD - 1)]
     else
          return 0
```

3.3.4 Bölme İşlemi

Bölme işleminde aritmetik bir bölme işlemi yapar gibi paydanın sıfır olup olmadığı durumu kontrol edilir ve daha sonrasında $GF(2^8)$'de bir elemanın tersini alma işlemi gerçekleştirilir. Sonlu cisimler üzerinde bölme işlemi yoktur.

```
Bölme İşlemi(a, b)
    j = 0
    if  b == 0  // paydanın sıfır durumu kontrol ediliyor
        Algoritmayı durdur
    if a == 0  // payın sıfır olma durumu
         return 0
    j = Logtable[a] - Logtable[b]  // j değeri hesaplanıyor
    if j < 0
         j = j + (FIELD - 1)
    return Alogtable[j]  // j değerinin GF(2^8)'deki değeri döndürülüyor
```

3.3.5 $GF(2^8)$'in Elemanlarının Tersini Bulma

Burada yapılan işlem, $GF(2^8)$ cisminin verilen herhangi bir elemanının tersini bulmadır.

```
Tersini Alma(a)
    if a = 0
```

```
        return 0  // 0'ın tersi 0 olduğu için 0 döndürülüyor
    else
        return Alogtable[((FIELD - 1) - Logtable[a])]
```

3.3.6 Bir Matrisin Tersini Bulma

ABC Kriptosisteminin algoritmasında verilen herhangi bir matrisin tersi bulunur.

```
Matrin Tersini Alma(a[], n)
    int *is, *js, i, j, k, l, u, v, d, p        // gerekli değişkenleri tanımlanıyor
    is = (int *)malloc(n * sizeof(int))  // integer boyutunda yer alınıyor
    js = (int *)malloc(n * sizeof(int))  // integer boyutunda yer alınıyor
    for k = 0 to k ≤ (n - 1)
        d = 0
        for i = k to i ≤ (n - 1)
            for j = k to j ≤ (n - 1)
                l = i * n + j
                p = a[l]
                if (p > d)
                    d = p
                    is[k] = i
                    js[k] = j
                end if
            end for
         end for
        if d == 0                       // d değeri 0 olursa matrisin tersi yoktur
            free(is)                    // is için alınan yer serbest bırakılıyor
            free(js)                    // is için alınan yer serbest bırakılıyor
            return 0
        end if
        if is[k] != k
            for j = 0 to j ≤ (n - 1)
                u = k * n + j
                v = is[k] * n + j
                p = a[u]
                  a[u] = a[v]
                  a[v] = p
            end for
        end if
        l= k * n + k
        a[l] = inv(a[l])                    // GF(2^8)'de elemanının tersi alınıyor
        for j = 0 to j ≤ (n - 1)
            if j != k
                u= k * n + j
                  a[u] = mul(a[u], a[l])  // çarpma işlemi fonksiyonu çağırılıyor
            end if
        end for
        for i = 0 to i ≤ (n - 1)
            if i != k
                for j = 0 to j ≤ (n - 1)
                    if j != k
                        u= i * n + j
```

```
                        a[u] = a[u] ^ (mul(a[i * n + k], a[k * n + j]))
                    end if
                end for
            end if
        end for
        for i = 0 to i ≤ (n - 1)
            if i != k
                u = i * n + k
                a[u] = mul(a[u], a[l])
            end if
        end for
    end for
    for k = (n - 1) to k ≥ 0
        if js[k] != k
            for j = 0 to j ≤ (n - 1)
                u= k * n + j
                v = js[k] * n + j
                  p = a[u]
                  a[u] = a[v]
                  a[v] = p
             end for
        end if
        if is[k] != k
            for i = 0 to i ≤ (n - 1)
                u = i * n + k
                  v = i * n + is[k]
                  p = a[u]
                  a[u] = a[v]
                  a[v] = p
              end for
          end if
    end for
    free(is)                    // is için alınan yer serbest bırakılıyor
    free(js)                    // js için alınan yer serbest bırakılıyor
    return (1)
```

3.3.7 İki Matrisin Çarpımını Bulma

A matrisi $m \times n$, B $n \times k$ boyutunda matrisler olmak üzere $A \times B$ matrisinin çarpımını bulma algoritmasında; A, B ve C matrislerini tek boyutlu dizi olarak tanımlanıyor.

```
Matris Çarpımı(a[], b[], m, n, k, c[])
   int i, j, l, u
   for i = 0 to i < (m - 1)
      for j = 0 to j < (k - 1)
          u = i * k + j
          c[u] = 0
          for l = 0 to l <= (n - 1)
              c[u] = c[u] ^ (mul(a[i * n + l], b[l * k + j]));
          end for
       end for
```

```
    end for
```

3.3.8 Bir Matrisin Transpozu

Bu algoritmada A ve B matrislerini tek boyutlu birer dizi olarak tanımlanıyor.

```
Matris Transpozu(a[], b[], n)
    int i, j
    for i = 0 to i <= (n - 1)
         for j = 0 to j <= (n - 1)
               b[i * n + j] = a[j * n + i]
          end for
    end for
```

3.3.9 Sabit Terimlerin Çarpımı

$c = (a_1 * x_1 + \cdots + a_n * x_n) * (b_1 * x_1 + \cdots + b_n * x_n)$ şeklinde sabit terimlerin çarpımını bulan algoritma.

```
Sabit Bulma(a[], b[], c[], n, r)
    t = 0
    temp = 0
    for i = 0 to i < n
        for j = i to j < n
            if i == j
                for k = 0 to k < r
                    t = add(t, mul(a[k * n + i],b[k * n + i]))
                end for
                c[temp++] = t
                  t = 0
            end if
            if i != j
                for k = 0 to k < r
                    t = add(t, add(mul(a[k * n + i], b[k * n + j]), mul(a[k * n + j], b[k * n + i])));
                end for
                c[temp++] = t
                  t = 0
            end if
        end for
    end for
    return 1
```

3.3.10 Basamak (Echelon) Matris'in Hesaplanması

Basamak matris'i hesaplama algoritmasında argüman olarak, basamak matrisini öğrenmek istediğimiz matrisin tek boyutlu dizisini, satır ve sütun sayısı verilmektedir.

```
Echelon Matris(M[], rows, cols)
    lead = 0
    rix = 0
    iix = 0
    for rix = 0 to rix < rows
        if lead ≥ cols
            return 0
        end if
        iix = rix
        while  M[iix*ncols+lead] == 0 do
            iix = iix + 1
            if iix = rows
                iix = rix
                  lead = lead + 1
                if cols == lead
                    return 0
                end if
            end if
        end while
        for j = 0 to j < cols
            temp = M[rix * ncols + j]
              M[rix * ncols + j] = M[iix * ncols + j]
              M[iix * ncols + j] = temp
        end for
        de = M[rix * ncols + lead]
        if de != 0
            for j = 0 to j < cols
                M[rix*ncols + j] = divs(M[rix * ncols + j], de)
            end for
        end if
        for j = 0 to j < rows
            if j != rix
                sb = M[j * ncols + lead]
                for k = 0 to k < cols
                    M[j * ncols + k] = sub(M[j * ncols + k], mul(sb, M[rix * ncols + k]))
                end for
            end if
        end for
        lead = lead + 1
        return 1
```

3.3.11 Anahtar Üretimi

Anahtar üretimi uzun olduğu için bloklar halinde algoritması ifade ediliyor. Tanımlamada oluşturulan WORD değişkenini kullanılıyor.

Tanımlama Bloğu

```
Anahtar Üretimi(*sk, *pk)
    WORD  S[VARIABLE*VARIABLE]
    WORD INVS[VARIABLE*VARIABLE], ST[VARIABLE*VARIABLE]
    WORD  M[VARIABLE*VARIABLE], U[VARIABLE*VARIABLE]
    WORD V[VARIABLE*VARIABLE]
    WORD  T[EQUATION*EQUATION], INVT[EQUATION*EQUATION]
    WORD  B[BROW*BCOL*VARIABLE], C[CROW*CCOL*VARIABLE]
    WORD A[AROW*ACOL*VARIABLE];
    WORD  TEMPA[ACOL*VARIABLE], TEMPB[BROW*VARIABLE]
    WORD TEMPC[CROW*VARIABLE], TEMPF[CENTRAL_MAP_SIZE
    WORD *F;
    WORD *FB;
    int i, j, s, t, ij = 0, eof, count, flag = 0;
    WORD W =  0;
    F = (WORD *) malloc (((AROW * BCOL + AROW * CCOL) * CENTRAL_MAP_SIZE) *
sizeof (WORD));
    FB = (WORD *) malloc (((AROW * BCOL + AROW * CCOL) * CENTRAL_MAP_SIZE)
* (sizeof (WORD)));
```

Anahtar üretiminin yapıldığı blok;

L_1 matrisi oluşturuluyor, tekil olmadığı kontrol edilip S ile lineer dönüşümü başlatılıyor.

```
    eof = 0
    while eof == 0 do
        for i=0 to i < VARIABLE
            for j = 0 to j < VARIABLE
                INVS[i * VARIABLE + j] = S[i * VARIABLE + j] = rand() % FIELD
            end for
        end for
        eof = matrixinv(INVS, VARIABLE)
    end while
```

Aynı şekilde L_2 matrisi oluşturuluyor ve aynı işlemler tekrarlanıyor. Ayrıca T^{-1} hesaplanıyor.

```
eof = 0
while eof == 0 do
    for i = 0 to i < EQUATION
        for j = 0 to j < EQUATION
            INVT[i * EQUATION + j] = T[i * EQUATION + j] = rand() % FIELD
        end for
    end for
    eof =  matrixinv(INVT, EQUATION)
end while
```

Gizli anahtarda T^{-1} saklanıyor.

```
for i = 0 to i < EQUATION
    for j = 0 to j < EQUATION
        sk[ij++] = INVT[i * EQUATION + j]
    end for
end for
```

A, B ve C matrisleri üretiliyor.

```
for i = 0 to i < AROW * ACOL * VARIABLE
    A[i] = rand() % FIELD
end for

for i = 0 to i < BROW * BCOL * VARIABLE
    sk[ij++] = B[i] = rand() % FIELD
end for

for i = 0 to i < CROW * CCOL * VARIABLE
    sk[ij++] = C[i] = rand() % FIELD
end for
```

$A \times B$ matrisi hesaplanıyor.

```
count = 0
flag = 0
for i = 0 to i < AROW
    for j = 0 to j < ACOL * VARIABLE
        TEMPA[j] = A[i * ACOL * VARIABLE + j]
    end for
    for j = 0 to j < BCOL
        for s = 0 to s < BROW
            for t = 0 to t < VARIABLE
                TEMPB[s * VARIABLE + t] = B[j * VARIABLE + s * BCOL * VARIABLE +
t]
            end for
```

```
            end for
            tensorproduct(TEMPA, TEMPB, TEMPF, VARIABLE, BROW)
            for s = 0 to s < VARIABLE
                for t = s to t < VARIABLE
                    sk[ij++] = F[flag++] = TEMPF[count++]
                end for
            end for
            count = 0
        end for
```

$A \times C$ matrisi hesaplanıyor.

```
    for i = 0 to i < AROW
        for s = 0 to s < ACOL
            for t = 0 to t < VARIABLE
                TEMPA[s * VARIABLE + t] = A[i * ACOL * VARIABLE + s *
                    VARIABLE + t]
            end for
        end for
        for j = 0 to j < CCOL
            for s = 0 to s < CROW
                for t = 0 to t < VARIABLE
                    TEMPC[s * VARIABLE + t] = C[j * VARIABLE + s * CCOL * VARIABLE +
t]
                end for
            end for
            tensorproduct(TEMPA, TEMPC, TEMPF, VARIABLE, CROW)
            for s = 0 to s < VARIABLE
                for t = s to t < VARIABLE
                    sk[ij++] = F[flag++] = TEMPF[count++]
                end for
            end for
            count = 0
        end for
```

$i = [0, i]$ olmak üzere $i < AROW * (BCOL + CCOL)$'ler için $FoS = Transpoze(S) * F_i * S$ hesaplanıyor.

```
        Matris Transpozu Alma(S, VARIABLE, ST)
        count = 0
        flag = 0
        int num = 0
        for i = 0 to i < AROW * (BCOL + CCOL)
            for s = 0 to s < VARIABLE * VARIABLE
                M[s] = (WORD) 0
            end for
            for s = 0 to s < VARIABLE
                for t = s to t < VARIABLE
                    M[s * VARIABLE + t] = F[count++]
                end for
```

```
        end for
        matris çarpımı (ST, M, VARIABLE, VARIABLE, VARIABLE, U)
          matris çarpımı(U, S, VARIABLE, VARIABLE, VARIABLE, V)
        for s = 0 to s < VARIABLE
            for t = s to t < VARIABLE
                if t == s
                    FB[flag++] = V[s * VARIABLE + t]
                end if
                if t != s
                    FB[flag++] = add((WORD)V[s * VARIABLE + t],
                            (WORD)V[t * VARIABLE + s])
                end if
            end for
        end for
    end for
```

$ToFoS$ hesaplanıyor.

```
    Matris çarpımı (T, FB, EQUATION, EQUATION, CENTRAL_MAP_SIZE, pk)
    for i = 0 to i < VARIABLE  // S^-1 saklanıyor
        for j = 0 to j < VARIABLE
            sk[ij++] = INVS[i * VARIABLE + j]
        end for
    end for
    free(F)
    free(FB)
    return 1
```

3.3.12 Şifreleme Algoritması

Şifreleme algoritmasında argüman olarak açık anahtarın, şifrelenecek metnin, şifrelenmiş metni saklayacak değişkenin başlangıç adresleri ve şifrelenecek metnin uzunluğu verilmektedir.

```
Şifreleme (*pk, *plaintext, *ciphertext, plaintextlen)
    int i, j, k, ij = 0, eof = 0
    WORD temp, x[VARIABLE];
    if plaintextlen != VARIABLE
        return (0)
    for k = 0 to k < VARIABLE
        x[k] = (WORD)plaintext[k]
    end for
    for k = 0 to k < EQUATION
        temp = 0
        for i = 0 to i < VARIABLE
            for j = i to j < VARIABLE
                temp =temp ^ mul((WORD)pk[ij++], mul(x[i], x[j]))
```

```
            end for
        end for
        ciphertext[k] = temp
    end for
    return 1
```

3.3.13 Şifre Çözme Algoritması

Şifre çözme algoritmasında argüman olarak gizli anahtarın, şifrelenecek metnin, şifrelenmiş metnin başlangıç adresleri ve şifrelenmiş metnin uzunluğu verilmektedir.

```
Şifre Çözme(*sk, *decrypttext, *ciphertext, ciphertextlen)
    int i, j, k, ij = 0, eof = 0;
    WORD  x[AROW * ACOL + VARIABLE], y[EQUATION], y1[EQUATION]
    WORD y2[EQUATION], X[VARIABLE]
    WORD coeff[ACOL * (BCOL + CCOL) * (ACOL * AROW + VARIABLE)]
    WORD TP, xn
    if ciphertextlen != EQUATION
        return (0)
    for k = 0 to k < EQUATION
        y[k] = (WORD)ciphertext[k]
    end for
    for k = 0 to k < AROW * ACOL + VARIABLE
        x[k] = (WORD)0
    end for
    for i = 0 to i < ACOL * (BCOL + CCOL) * (ACOL * AROW + VARIABLE)
        coeff[i] = (WORD)0
    end for
    WORD Sqrttable[FIELD]
    for i = 0 to i < FIELD
        Sqrttable[i] = mul(i, i)
    end for


    k = 0
    for i = 0 to i < EQUATION  // T^-1 hesaplanıyor
        y1[i] = 0
        for j = 0 to j < EQUATION
            y1[i] = add(y1[i], mul( (WORD)sk[ij++], y[j]))
        end for
    end for

    for k = 0 to k < ACOL  // F^-1 hesaplanıyor
        for i = 0 to i < BCOL
            for j = 0 to j < AROW
                coeff[(k * BCOL + i) * (ACOL * AROW + VARIABLE) + k * AROW + j] = y1[j
* BCOL + i]
            end for
        end for
        for i = 0 to i < CCOL
            for j = 0 to j < AROW
```

```
                coeff[ACOL * BCOL * (ACOL * AROW + VARIABLE) + (k * CCOL + i) *
(ACOL * AROW + VARIABLE) + k * AROW + j] =
                                        y1[AROW * BCOL + j * CCOL + i]
            end for
        end for
    end for
    for i = 0 to i < ACOL * BCOL
        for j = AROW * ACOL to j < (ACOL * AROW + VARIABLE)
            coeff[i * (AROW * ACOL + VARIABLE) + j] = (WORD)sk[ij++]
        end for
    end for
    for i = 0 to i < ACOL * CCOL
        for j = AROW * ACOL to j < (ACOL * AROW + VARIABLE)
            coeff[ACOL * BCOL * (AROW * ACOL + VARIABLE) + i * (AROW * ACOL +
VARIABLE) + j] = (WORD)sk[ij++]
        end for
    end for

    echelonform(coeff, ACOL * (BCOL + CCOL), AROW * ACOL + VARIABLE)
    WORD WM[VARIABLE];
    for i = 0 to i < VARIABLE - 1
        WM[i] = coeff[(i + AROW * ACOL) * (ACOL * AROW + VARIABLE) + (ACOL *
AROW + VARIABLE - 1)]
    end for
    WM[VARIABLE - 1] = 1
    for k = 0 to k < EQUATION
        TP = 0
        for i = 0 to i < VARIABLE
            for j = i to j < VARIABLE
                TP = add(TP, mul(sk[ij++], mul(WM[i], WM[j])))
            end for
        end for
        y2[k] = TP
    end for
    for i = 0 to i < EQUATION
        if y2[i] != 0
            break
        end if
    end for
    TP = mul(inv(y2[i]), y1[i])
    for i = 0 to i < FIELD
        if Sqrttable[i] == TP
            xn = (WORD)i
        end if
    end for
    X[VARIABLE-1] = xn
    for i = 0 to i < VARIABLE - 1
        X[i] = mul(WM[i], X[VARIABLE - 1])
    end for
    for i = 0 to i < VARIABLE  // S^-1 hesaplanıyor
        decrypttext[i] = 0
        for j = 0 to j < VARIABLE
            decrypttext[i] = add(decrypttext[i], mul( (WORD)sk[ij++], X[j]))
        end for
    end for
    return 1
```

3.3.14 Asimptotik Karmaşıklık

ABC Kriptosisteminin çalışması için gerekli algoritmaların asimptotik karmaşıklığı ($Big(O)$) tablosu aşağıdadır. Tablodaki toplama, çıkarma, $GF(2^8)$'in elemanlarının çarpımı, bölme ve $GF(2^8)$'in elemanın tersi fonksiyonlarının değerleri tablo kullanılarak gerçekleştirilmiştir

Tablo 3.1 ABC Kriptosisteminde Kullanılan Fonksiyonların Asimptotik Karmaşıklık Değerleri

Algoritma İsmi	**Big O Değeri**
Toplama	O(1)
Çıkarma	O(1)
GF(2^8)'in Elemanlarının Çarpma	O(1)
Bölme	O(1)
GF(2^8)'de Elemanın Tersi	O(1)
Bir Matrisin Tersini Bulma	O(n^3)
Matris Çarpımı	O(n^3)
Bir Matrisin Transpozu	O(n^2)
Herhangi İki Denklemin Sabitleri Çarpımı	O(n^3)
Bir Matrisin Basamak Formunu Bulma	O(n^3)
Anahtar Üretimi	O(n^4)
Şifreleme	O(n^3)
Şifre Çözme	O(n^3)

4. ANAHTAR BOYUTLARININ DÜŞÜRÜLMESİ ve PERFORMANS İYİLEŞTİRİLMESİ

Bu bölümde, ABC kriptosisteminin farklı platformlar için uygulama detayları verilmektedir. ABC sisteminin çalışmasında üç ana aşaması bulunmaktadır. Bunlar "anahtar üretimi", "şifreleme" ve "şifre çözme"dir. Programın gerek standart gerekse paralel uygulamasında parametre kümeleri Tablo 4.1'de verilmiştir. Uygulama hem "Windows 8.1 x64" hem de "Kali Linux 64-Bit 2018.2" işletim sistemlerinde çalıştırılmıştır. Tablo 4.1'deki değerlerin boyutu 4-byte (32-bit) şeklindedir.

Tablo 4.1. Değerler Boyutu

S	8
N	64
M	128
CENTRAL_MAP_SIZE (Ortadaki Dönüşüm)	2080
PUBLIC_KEY_SIZE (Açık Anahtar Boyutu)	266240
SECRET_KEY_SIZE (Gizli Anahtar Boyutu)	294912

4.1 Performans İyileştirilmesi

Performans iyileştirilmesinin en çok gerekli olduğu kısım anahtar üretim kısmıdır. Buradaki asimptotik karmaşıklık, iç içe dört kez "for" döngüsü kullanıldığından $O(n^4)$'tür.

Standart uygulamada gerek ön tanımlı (define) gerekse "Main()" fonksiyonu içinde tanımlamalar yapıldıktan sonra sırasıyla "anahtar üretimi", "şifreleme" ve "şifre çözme" için yazılan fonksiyonları çağırarak uygulama gerçekleşmektedir. Bölüm II'de adlandırılan gerekli algoritmaların bu üç kısımda (anahtar üretimi, şifreleme, şifre çözme) kaç kez çağrıldığı Tablo 4.2'de gösterilmiştir. Çağrılma sayıları, her bir fonksiyon için genel (generic) olarak tanımlanan bir değişkenin fonksiyon içinde arttırılması ile elde edilmiştir.

Tablo 4.2 Fonksiyonların Çağırılma Sayısı

	Anahtar Üretimi	Şifreleme	Şifre Çözme	Toplam
Toplama	4452352	0	286720	4739072
Çıkarma	0	0	2064512	2064512
Çarpma	107740992	532480	2617792	110891264
Bölme	0	0	16256	16256
Tersini Alma	193	0	1	194
Matris Tersi	2	0	0	2
Matris Çarpımı	257	0	0	257
Matris Trans.	1	0	0	1
Denk. Kats. He.	128	0	0	128
Echelon	0	0	1	1

Matris çarpımının performansını arttırmak amacıyla kod optimize edilebilir ya da thread mantığı ile paralel hesaplama seçenekleri kullanılabilir. 1970'li yıllarda tasarlanan standart C fonksiyonları birden fazla thread ile kullanıma uygundur. Standart C kütüphanelerinde tek thread (single thread) için ve çoklu thread (multi thread) için olmak üzere iki versiyon'u vardır. Ancak kütüphane UNIX ve LINUX ortamında POSIX kütüphanesi olarak geçmektedir. "pthread" kelimesi ise bu kütüphanenin baş harfi olan P harfi ile thread kelimesinin birleştirilmesinden oluşmuştur.

Bu çalışmada ABC algoritmasında öncelikle anahtar üretimi fonksiyonu içinde gerekli olan matrislerin üretilmesi ve değerlerinin yazılması işlemlerinde kullanılacaktır.

4.1.1 Pthread Kullanımı

Bir thread'in tanımlaması yapıldıktan ve başlatıldıktan sonra, hazır durumuna geçer. Zamanlama algoritmasına (scheduling algorithm) (Cho, Ravindran and Jensen, 2006) bağlı olarak, çalışır duruma geçer. Algoritmanın çalışma sırasında thread işlemini bitirir veya iptal edilir ise bitmiş durumuna geçer. Algoritmanın uygulanmasına göre çalışma sırasında beklemeye alınabilir. Örneğin farklı bir işlemi yapan başka bir thread beklenebilir veya bir sistem kaynağına erişmek

isteyebilir. Bu bekleme sebebi geçtikten sonra tekrardan hazır duruma geçer ve hazır sırasında (ready queue) beklemeye başlar.

C dilinde thread'i kullanabilmek için "pthread.h" kütüphanesi eklenmesi gerekmektedir. Ayrıca kodu Linux işletim sisteminde derlemesi yapılırken "-lpthread" parametresinin verilmesi gerekmektedir. Bunun yapılmasının sebebi ise derleme sırasında phtread kütüphanesinin bağlanması (link) gereğidir.

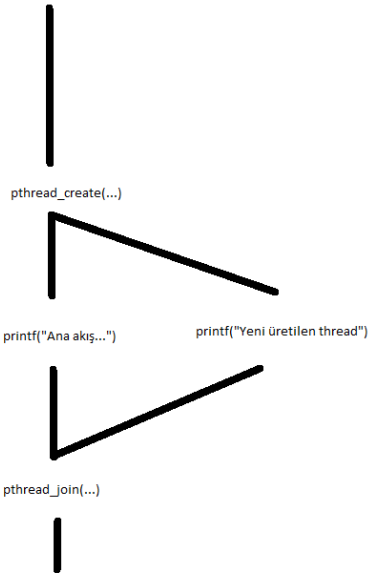
Örnek 1:

```
#include <pthread.h>
#include <stdio.h>

void *thread_function(void* arg){
    printf("Yeni oluşturulan thread…\ n");
}

void main(){
   pthread_t thread_id; void *thread_sonuc;
   pthread_create( &thread_id, NULL, thread_function, NULL );
   printf("Ana algoritma…\n");
   pthread_join( thread_id, &thread_sonuc);
}
```

Örnek 1'deki algoritmanın çalışmasında öncelikle main fonksiyonu içerisinde bir thread bilgisi tutabilecek ve pthread_t yapısında (struct) bir değişken tanımlaması yapılmıştır. Bu tanımlanan değişken daha sonraki örneklerde de kullanılacaktır. Örnek 1'deki "pthread_create" fonksiyonunun ilk argümanı çok önemlidir. Bu argüman atıf ile çağırma (call by reference) işlemi olarak düşünülmelidir ve burada yapılan işlem thread üretimi sırasında argüman verilerek, oluşturulan yeni thread'in bilgisini almaktır.

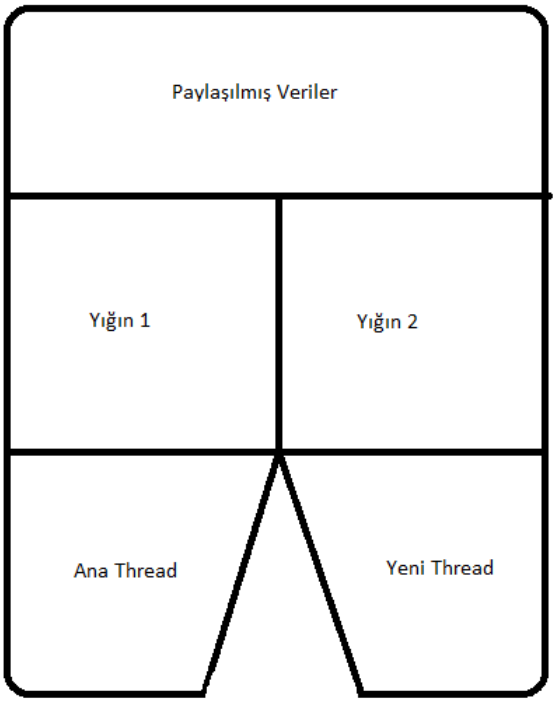
Daha sonrasında ekrana "Ana algoritma" yazısı gösterilmiştir. Burada dikkat edilmesi gereken husus, aslında yeni bir thread oluşturulurken en az iki thread olmasıdır. Bunlardan birisi yeni üretilmiş olan thread'dir. Diğeri ise bu thread'i üreten thread olan ana (main) thread'dir. Bu durum Şekil 4.1'de ifade edilebilir:

Şekil 4.1 Tread Akış Şeması



Şekil 4.1 görüldüğü üzere, algoritmanın akışı sırasında çağrılmış olan bir pthread_create fonksiyonu ile hafızada iki farklı thread üretilmiştir. Bu üretilen thread'ler aynı anda çalışmaya devam etmiştir. Ana (main) thread'deki algoritmanın çalışması tamamlandıktan sonra, pthread_join fonksiyonu ile yeni üretilen thread'in bitmesi beklenmiştir.

Şekil 4.2 Hafızada Anlık İki Lifin Durumu



Hafızada anlık olarak iki lifin bulunduğu durum, Şekil 4.2'de gösterilmiştir. Bu şekilden anlaşılacağı üzere hem ana thread hem de oluşturulan yeni thread için iki ayrı yığın (stack) alanı bulunmaktadır. Bu alanlarda liflerin kendisine özel bilgileri tutulmaktadır. Örneğin anlık olarak algoritmada çalıştırılan fonksiyonun hangi satırının çalıştırıldığı, daha sonrasında bu fonksiyon işlemini bitirdikten sonra hangi fonksiyona geri çağırılacağı gibi bilgiler tutulur. Bu duruma karşılık, thread'in üretilmesi sırasında tanımlı olan bütün değişkenler, iki fonksiyon arasında da paylaşılmıştır. Bu paylaşılan değerlere de paylaşılmış değişkenler (shared variable) adı verilir.

Örnek 2:

```
#include <pthread.h>
#include <stdio.h>
#include <string.h>

void *thread_function(void* arg){
    printf("Yeni thread oluşturuluyor \n");
```

```
    return (void*) strdup("Bir dizgi geliyor");
}

void main(){
    pthread_tthread_id;
    void *thread_sonuc=0; pthread_create( & thread_id, NULL,
thread_function, NULL );
    printf("Ana algoritma");
    pthread_join( thread_id,&thread_sonuc);
    if ( thread_result!= 0 )
    printf("Ana algoritmadan gelen %sn", thread_sonuc);
}
```

Örnek 2'de, pthread_join fonksiyonu, Örnek 1'de de olduğu gibi yeni üretilen thread'i beklemek için kullanılmıştır. Bu kullanımda ikinci argümanda üretilen yeni thread'den gelen veriyi almak için atıf ile çağırma (call by reference) şeklinde verilmiştir.

"thread_sonuc" isimli fonksiyon, Örnek 1'den farklı olarak bir değeri geri (return) döndürmektedir. Buna ek olarak fonksiyonun döndürdüğü değer, pthread_join fonksiyonunun ikinci argümanına atanmıştır. Bu atama işleminde kullanılan değişkenin tipi ise "void *"dır. C dilinde değişkenin tipi belirsiz ise bu değişken tipi kullanılır. Başka bir deyişle "void *" tipin belirsizliği anlamındadır. Gelen değer, sıfırdan farklı olması durumunda ekrana ana thread'ten yazdırılır.

Örneklerde bahsedilen fonksiyonların ayrıntıları aşağıda verilmiştir:

```
int pthread_create( pthread_t *tid, cons pthread_attr_t *attr,
void* funk, void * arg);
```

"pthread_create" fonksiyonu yeni bir lif thread oluşturmak için kullanılır. 4 argüman alır. Bunlar sırasıyla, oluşturulacak olan yeni lifin thread'in bilgisini tutan tid, thread oluşumu sırasında verilecek olan özellikler (attr), thread'in çalıştıracağı fonksiyon (buradaki parametre bir fonksiyon göstericisi (function pointer) olarak verilmektedir), fonksiyon göstericisi olarak alınan parametreye verilecek olan parametreler. Diğer bir deyişle oluşturulan yeni thread, bir fonksiyonu çalıştırırken bu fonksiyona aktarılacak olan parametreler şeklindedir.

Fonksiyon sonuç olarak bir int değer döndürür. Eğer int değer 0 ise başarılı, farklı durumlarda ise hata değeri anlamını taşımaktadır.

```
int pthread_equal(pthread_t *tid1, pthread_t *tid2);
```

Argüman olarak aldığı iki thread'i karşılaştırır. Eğer ki sonuçta eşitseler 0 değerini aksi bir eşitlik durumunda ise 0 dışında bir değer döndürür. Buradaki karşılaştırma işlemi iki thread'in içeriğine bakılarak yapılmaktadır ve dahası derin karşılaşıtrma (deep compare) adı verilen bir karşılaştırmadır. Sığ karşılaştırma (shallow compare) için ise "==" işlemi (operator) kullanılabilir.

```
int pthread_exit(pthread_t *tid);
```

Argüman olarak aldığı thread bilgisini sonlandırır. Verilen argümandaki thread'in çalışması sonlandırılarak hafızadan kaldırılır.

```
int pthread_join(pthread_t tid, void ** ptr);
```

Argüman olarak verilen thread bitene kadar bekler ve bu thread tarafından döndürülen değeri ikinci parametresi olan ptr ile alır.

```
int pthread_detach (pthread_t tid);
```

Argüman olarak aldığı thread'i hafızadan kaldırır. Bu fonksiyon ile thread sadece hafızadaki kopyası silinir.

```
int pthread_cancel(pthread_t tid);
```

Argüman olarak aldığı thread'i iptal eder (cancel).

```
pthread_t pthread_self();
```

Bu fonksiyon çağrıldığında, o andaki çalışan thread'in bilgisi döndürülür.

```
int sched_yield();
```

Bu fonksiyon, zamanlayıcıya (scheduler) içinden çağrıldığı thread'in çalışma durumundan alınarak bekleme durumuna geçirilmesini söyler. Bu sayede o anda beklemekte olan başka bir thread çalışabilecektir.

4.1.2 Performans İyileştirilmesinin Algoritmaya Uygulanması

ABC algoritmasının öncelikle anahtar üretimi fonksiyonu içinde gerekli olan matrislerin üretimi ve değerlerinin yazılması işlemlerinde kullanılan ve https://gitlab.com/ramcho/abc adresinden erişilebilen uygulama kodlarında tanımlama bloğunda kullanılan;

```
WORD  S[VARIABLE*VARIABLE];
WORD INVS[VARIABLE*VARIABLE];
WORD INVT[EQUATION*EQUATION];
WORD T[EQUATION*EQUATION];
```

matrisleri genel tanımlama için "abc.h" isimli header dosyasının içine eklenmiştir. Böylelikle bağımsız olarak hem threadler'de kullanılırken hem de fonksiyonlarda işlemlere girdiğinde işlenmiş değerler kaybolmamaktadır.

Buradaki "S, INVS, T ve INVT" matrisleri ABC kriptosisteminin gerçekleşmesi için gerekli olan L_1 ve L_2 matrislerini oluşturulmasında (tekil olmadığı kontrol edilip) S ile lineer dönüşümü başlatılmasında ve T^{-1} hesaplamasında kullanılmaktadır. Daha sonrasında anahtar üretimi fonksiyonunun içinde aşağıdaki kod eklemesi yapılır.

```
pthread_t thread_id;
```

```
void *thread_result;
pthread_create( &thread_id, NULL, thread_routine, NULL );
pthread_join( thread_id, &thread_result);
```

Burada öncelikle “thread_id” isimli bir thread tanımlanır. Benzer şekilde tipi “void” olan “thread_result” isimli bir işaretçi tanımlanır. Pthread kütüphanesinde ön tanımlı olan “pthread_create” fonksiyonu ile oluşturulan thread algoritmaya tanıtılır ve hangi fonksiyonu çağırması gerektiği parametre olarak verilmektedir. En son satırda ise thread’i çağırma işlemi yapılmaktadır.

```
void *thread_routine(void* arg){
    int eof = 0, i, j;
    while (eof == 0) {
        for (i = 0; i < VARIABLE; i++) {
            for (j = 0; j < VARIABLE; j++) {
                INVS[i * VARIABLE + j] = S[i * VARIABLE + j]
=rand() % FIELD;
            }}
        eof = matrixinv(INVS, VARIABLE);
    }
    eof=0;
    while (eof == 0) {
        for (i = 0; i < EQUATION; i++) {
            for (j = 0; j < EQUATION; j++) {
                INVT[i * EQUATION + j] = T[i * EQUATION + j] =
rand() % FIELD;
        }}
    eof = matrixinv(INVT, EQUATION);
}}
```

“*thread_routine” isimli fonksiyon “abc.c” isimli dosyada main dışında herhangi bir yere eklenir. Bu kod bloğu çalıştığında “S, INVS, T ve INVT” isimli matrislere rastgele değerler üretilip ataması yapılmıştır.

4.2 CUDA ile Performans Geliştirilmesi

ABC algoritmasına bakıldığında performans artışı sağlamak için $GF(2^8)$ üzerinde yapılan toplama, çıkarma ve çarpma işlemleri fonksiyonlarını GPU üzerinde gerçekleştirilebilir. Daha önceki bölümde anlatılan hem standart hem de pthread kullanılarak çalıştırılan ABC algoritması CPU üzerinde çalışmaktadır.

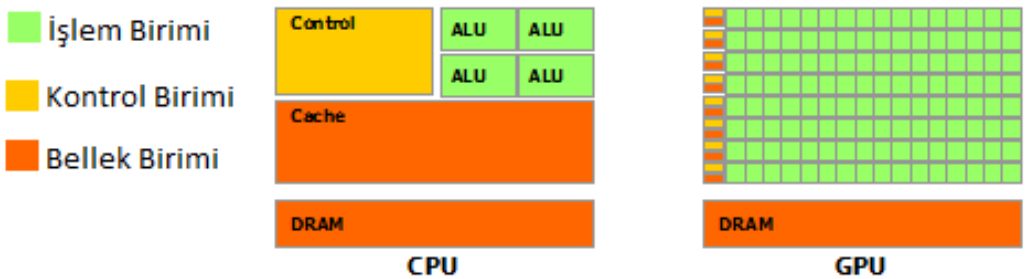
Grafik İşlem Birimi (GPU)'nin temel görevi bilgisayarda oluşturulan görüntülerin ekrana verilmesini sağlamaktır. Bu sebeple ilk GPU'lar sadece bu görevi yerine getirmekteydi. Zaman içerisinde Merkezi İşlem Birimi (CPU)'nin karşılaşılan büyük hesaplama problemlerinde yetersiz kaması üzerine GPU'nun donanımsal paralelliğinden yararlanma fikri ortaya çıkmıştır. GPU'ların programlanabilir bir arayüze sahip olmasını ve yüksek seviyeli dillerle programlanabilmesini sağlamak için GPGPU modeli oluşturulmuştur.

4.2.1 NVIDIA CUDA Kullanımı

CUDA (Compute Unified Device Architecture), NVIDIA firmasının 2006 yılında GPU'nun donanımsal hesaplama gücünden faydalanmak amacıyla sunduğu paralel hesaplama mimarisidir. Linux, Windows ve Mac Osx platformları üzerinde çalışabilmektedir. FORTRAN, C/C++ ve Python gibi dilleri destekleyen bir API'dir. Rakiplerine göre avantajları paylaşımlı bellek kullanımı, GPU'dan daha hızlı veri okuma ve bit düzeyinde işlem yapılabilmesine olanak sağlaması olarak sayılabilir.

GPU'nun CPU'dan farkı, SIMD (Single Instruction Multiple Data-Tek Komut Çoklu Veri) mimarisine sahip olmasıdır. Şekil 4.3'de CPU (soldaki) ve GPU (sağdaki)'ların donanımsal olarak karşılaştırılması gösterilmiştir. CPU hesaplamalarını seri bir şekilde gerçekleştirirken, GPU hesaplamalarını yapısal olarak paralel olması sebebiyle paralel bir şekilde gerçekleştirmektedir.

Şekil 4.3 CPU, GPU Karşılaştırılması



Standart bir C ile yazılmış “Hello World” ve CUDA ile yazılmış “Hello World” Şekil 4.4’de gösterilmiştir. CUDA değer tanımlamalarında CPU üzerindeki değişkenlere “host” GPU üzerindeki değişkenlere ise “device” denilmektedir.

Şekil 4.4 Standart C ve CUDA

Standart C

```
void c_hello(){
        printf("Hello World!\n");
}

int main() {
        c_hello();
        return 0;
}
```

CUDA

```
__global__ void cuda_hello(){
        printf("Hello World from GPU!\n");
}

int main() {
        cuda_hello<<<1,1>>>();
        return 0;
}
```

Örnek 1:

```
#include <stdio.h>
#define N 1000
__global__ void add(int *a, int *b) {
    int i = blockIdx.x;
    if (i<N) {
        b[i] = 2*a[i];
    }
}
int main() {
    int ha[N], hb[N];
    int *da, *db;
    cudaMalloc((void **)&da, N*sizeof(int));
    cudaMalloc((void **)&db, N*sizeof(int));
    for (int i = 0; i<N; ++i) {
        ha[i] = i;
    }
    cudaMemcpy(da, ha, N*sizeof(int), cudaMemcpyHostToDevice);
    add<<<N, 1>>>(da, db);
    cudaMemcpy(hb, db, N*sizeof(int), cudaMemcpyDeviceToHost);
    for (int i = 0; i<N; ++i) {
        printf("%d\n", hb[i]);
    }
```

```
    cudaFree(da);
    cudaFree(db);
    return 0;
}
```

Örnek 1'de CPU üzerinde tanımlanan bir diziyi GPU üzerinde 2 ile çarpıp tekrardan CPU üzerinde ekrana yazdırılıyor.

```
cudaMalloc((void **)&(degisken adı), sizeof(değişken türü));
```

CPU üzerinde işaretçi olarak tanımlanmış bir değişken GPU üzerinde tanımlanabilmesi için "cudaMalloc" fonksiyonu kullanılır.

"cudaMemcpy" fonksiyonu CPU üzerindeki değişkeni GPU'ya veya GPU üzerindeki değişkeni CPU'ya kopyalamak için kullanılır.

```
cudaMemcpy(GPU değişkeni, CPU değişkeni, sizeof(değişken türü),
cudaMemcpyHostToDevice);
```

Burada kullanılan "cudaMemcpyHostToDevice" argümanı ile CPU üzerindeki değişken GPU üzerindeki değişkene kopyalanmıştır.

```
cudaMemcpy(CPU değişkeni, GPU değişkeni, sizeof(değişken türü),
cudaMemcpyDeviceToHost);
```

Burada kullanılan "cudaMemcpyDeviceToHost" argümanı ile GPU üzerindeki değişken CPU üzerindeki değişkene kopyalanmıştır.

4.2.2 CUDA Kullanılarak ABC Algoritmasının Uygulanması

ABC algoritmasında gerekli olan algoritmalar içinde en çok kullananlardan birisi $GF(2^8)$'de toplama işlemidir. Algoritmanın geneline bakıldığında "anahtar

üretimi" fonksiyonu içerisinde bir kez ve "şifre çözme" fonksiyonun da üç kez toplama işlemi fonksiyonu çağırılmaktadır.

```
__global__ void addcuda(WORD *a, WORD *b, WORD *c) {

	*c = *a ^ *b;

}
```

C++ programlama dilindeki fonksiyon tanımına benzer şekilde ancak CUDA'ya özgü bir şekilde "main" fonksiyonun dışında bir fonksiyon tanımlanmıştır. Buradaki "a", "b" ve "c" işaretçi "WORD" tipindeki değişkenler "a" ve "b" toplanması istenen değişkenler. "c" ise işlemin sonucunu saklayıp GPU üzerinden CPU üzerine taşıyacak değişkendir.

Anahtar üretimi fonksiyonu içerisinde "addcuda" fonksiyonun çağırılması aşağıdaki gibidir. Burada öncelikle "WORD" veri tipinde "*tempA", "*tempB", "*tempC" ve "tempFB" tanımlanıyor. Sonrasında "cudaMalloc" ön tanımlı CUDA fonksiyonu ile GPU üzerinde boyut tanımlaması yapılıyor. "cudaMemcpy" fonksiyonu ile CPU üzerindeki değerler GPU'ya kopyalanıyor. "addcuda<<<1, 1>>>()" ile GPU üzerindeki fonksiyon çağırılıyor.

Fonksiyon işlemini tamamladıktan sonra GPU üzerindeki değeri tekrardan CPU'ya kopyalamak için "cudaMemcpy" fonksiyonu kullanıyor.

```
WORD *tempA, *tempB, *tempC, tempFB;
cudaMalloc((void **)&tempA, sizeof(WORD));
cudaMalloc((void **)&tempB, sizeof(WORD));
cudaMalloc((void **)&tempC, sizeof(WORD));

for (s = 0; s < VARIABLE; s++) {
	for (t = s; t < VARIABLE; t++) {
		if (t == s) {
			FB[flag++] = V[s * VARIABLE + t];
		}
		if (t != s) {

			cudaMemcpy(tempA, &V[s * VARIABLE + t], sizeof(WORD),
cudaMemcpyHostToDevice);
			cudaMemcpy(tempB, &V[t * VARIABLE + s], sizeof(WORD),
cudaMemcpyHostToDevice);
			addcuda<<<1, 1>>>(tempA, tempB, tempC);
```

```
                    cudaMemcpy(&tempFB, tempC, sizeof(WORD),
cudaMemcpyDeviceToHost);


                    FB[flag++] = tempFB;

                        }
                    }
                }

cudaFree(tempA);
cudaFree(tempB);
cudaFree(tempC);
```

5. UYGULAMA SONUÇLARI ve KARŞILAŞTIRILMASI

Bu bölümde çok değişkenli polinomlar temelli ABC kriptosistemi algoritmasının hem thread'li ve thread'siz hem de standart ve CUDA kullanılarak geliştirilen uygulamasının sonuçlarına göre karşılaştırılması yapılmıştır.

5.1 Thread'li ve Tread'siz Uygulamanın Karşılaştırılması

ABC kriptosistemi için hazırlanan uygulama iki farklı bilgisayarda üç farklı ide ortamında hem thread'siz hem de pthread kütüphanesi ile denenmiştir. Tablo 5.1'de kullanılan bilgisayar özellikleri, derleme ortamı ve işletim sistemi bilgileri özetlenmiştir.

Tablo 5.1 Programın Çalışma Ortamı

İndeks	Bilgisayar Özellikleri	Ram	Derleme Ortamı	İşletim Sistemi
1	Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz	12 Gb	Visual Studio, 2013	Windows 8.1
2	Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz	12 Gb	Dev C++, 5.6.1	Windows 8.1
3	Intel(R) Core(TM) i3-4000M CPU @ 2.40GHz	8 Gb	GCC, 8.3.0	Kali Linux

Tablo 5.2'de ABC kriptosistemi için hazırlanan uygulamanın hem thread'li hem de thread'siz olarak üç farklı ortamdaki toplam çalışma süreleri mikro saniye (μs) cinsinden verilmiştir.

Tablo 5.2 Programın Tamamının Çalışma Süresi

	Thread'siz	Thread'li
İndeks 1	4336000 μs	3637000 μs
İndeks 2	1171000 μs	1125000 μs
İndeks 3	1460734 μs	1440778 μs

Tablo 5.3'de ABC kriptosistemi için hazırlanan uygulamanın "şifreleme" ve "şifre çözme" fonksiyonlarının çalışma süreleri üç farklı ortam için hem thread'li hem de thread'siz olarak mikro saniye cinsinden verilmiştir.

Tablo 5.3 Şifreleme ve Şifre Çözme Fonksiyonlarının Çalışma Süreleri

	Thread'siz		Thread'li	
	Şifreleme	Şifre Çözme	Şifreleme	Şifre Çözme
İndeks 1	16000 µs	132000 µs	12000 µs	128000 µs
İndeks 2	8000 µs	28000 µs	4000 µs	24000 µs
İndeks 3	5458 µs	30872 µs	5428 µs	30620 µs

Uygulamanın çalıştırılması sonucunda elde edilen süreler ve "Tablo 4.2"de ayrıntılı olarak gösterilen ABC kriptosistemi için gerekli olan fonksiyonların "anahtar üretimi", "şifreleme" ve "şifre çözme"de kaç defa çağrıldığını ve kullanıldığı sayılarak değerlendirilmiştir. Thread'li veya thread'siz olarak çalıştırılan kodda, fonksiyonların çağrılma işlemlerinde çok az bir fark vardır, ancak süre bazında gelişme sağlanmıştır. Süre ölçümü ise Tablo 5.2'de her iki kodda da "anahtar üretimi"nden önce başlayıp "şifre çözme"nin tamamlanmasıyla bitirilmiştir. Tablo 5.3'de ise şifreleme ve şifre çözme olarak thread'li ve thread'siz olarak ölçülmüştür. Her üç ortamda da $GF(2^8)$'de bir elemanın tersini alma fonksiyonu thread'siz yapıda 194 kez çağırılırken thread'li yapıda 193 kez çağrılmıştır.

İndeks numarası 1 olan derleyici ve bilgisayarda thread'siz kodun çalışması sonucu 4336000 mikro saniyelik bir sonuç elde edilmiştir. Thread'li yapıda ise 3637000 mikro saniyelik bir sonuç elde edilmiştir. Burada thread'li yapı sayesinde %16'lık bir zaman kazanımı sağlanmıştır.

İndeks numarası 2 olan derleyici ve bilgisayarda thread'siz kodun çalışması sonucu 1171000 mikro saniyelik bir sonuç elde edilmiştir. Thread'li yapıda ise 1125000 mikro saniyelik bir sonuç elde edilmiştir. Burada thread'li yapı sayesinde %4'lük bir zaman kazanımı sağlanmıştır.

İndeks numarası 3 olan derleyici ve bilgisayarda thread'siz kodun çalışması sonucu 1460734 mikro saniyelik bir sonuç elde edilmiştir. Thread'li yapıda ise 1440778 mikro saniyelik bir sonuç elde edilmiştir. Burada thread'li yapı sayesinde %1'lik bir zaman kazanımı sağlanmıştır.

Uygulamaların çalışma ortamları fark etmeksizin, paralel uygulamalar, performans olarak standart olanlarına göre daha iyidir. ABC kriptosisteminin şifre çözme aşamasında kullanılan L_1, L_2 lineer dönüşümlerinin tanımlandığı aşama bu farkı oluşturur. ABC kriptosisteminin uygulamasında bu aşama "anahtar üretimi" fonksiyonu içinde gerçekleşmektedir. Paralel uygulamada L_1, L_2 ve L_1^{-1}, L_2^{-1} dönüşümlerinin matris tanımlaması ve rastgele oluşturulan elemanlarının atama işlemleri thread yöntemi ile yapılmaktadır. Bu sayede 3 farklı ortam için ortalama %7'lik bir performans gelişimi sağlanmıştır.

Performans gelişimiyle veriler daha güvenli, daha hızlı şifrelenip iletilmekte ve çözülmektedir. Bu sayede veriler daha verimli şekilde kullanılıp zaman kaybı azaltılmaktadır. Zaman kaybının azaltılması, yüksek verimlilik ve güvenlik uygulamanın amacıdır.

Farklı ortamlarda farklı süreler elde edilmesini nedeni öncelikle işletim sistemi farklılığıdır. İndeks numarası 3 olan bilgisayarda Unix tabanlı "Kali Linux" işletim sistemi bulunmaktadır. İndeks numarası 1 ve 2 olan bilgisayarlarda ise "Windows" işletim sistemi bulunmaktadır. İşletim sistemleri farklılığına ek olarak CPU mimarisi farklılığı (Tablo 5.1'de ayrıntılı olarak verilmiştir) süre değişkenliğini etkilemiştir.

Aynı bilgisayar üzerinde çalışan İndeks 1 ve İndeks 2 numaralı ortamlarda ise süre farkının oluşmasının temel sebebi; paralel programlamadır. Paralel uygulama çalışmaya başladığında main içerisinde "anahtar üretimi" fonksiyonu çağrıldığında aynı anda bir thread oluşmaktadır. Bu thread programın ana akışından farklı bir paralel yol izlemekte ve yapması gereken işlemleri tamamlayıp tekrar ana akışa dönmektedir. Bu sayede "anahtar üretimi" fonksiyonu içerisinde fazladan tanımlama ve özyinelemeli fonksiyon çağırımı yapılmamış olur. Bir diğer sebep ise farklı ide'ler kullanılmasıdır. Visual Studio, Dev-C++'a göre daha kompleks bir ide olduğundan çalışma süresinde farklılığa sebep olmuştur. "Anahtar Üretimi" fonksiyonunda kullanılan matrislerin thread yapısı ile değer atanması sonucu Tablo 5.2'de ayrıntılı olarak verilen süreler elde edilmiştir.

5.2 Standart ve CUDA Kullanılarak Geliştirilen Uygulamanın Karşılaştırılması

Bu bölümde anlatılan uygulama üç farklı bilgisayarda üç farklı grafik işlemcisinde aynı ide (Visual Studio 2017) ortamında hem CPU üzerinde hem de GPU üzerinde denenmiştir. Tablo 5.4'te kullanılan bilgisayar özellikleri, derleme ortamı ve işletim sistemi bilgileri özetlenmiştir.

Tablo 5.4 Programın Çalışma Ortamı

İndeks	Bilgisayar Özellikleri	Ram	Grafik İşlemcisi	İşletim Sistemi
1	Intel(R) Core(TM) i7-4500U CPU @ 1.80 GHz	12 Gb	GeForce GT 740M 2 Gb	Windows 8.1
2	Intel(R) Core(TM) i7-4210U CPU @ 1.70 GHz	8 Gb	GeForce GT 840M 2 Gb	Windows 10
3	Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz	16 Gb	GeForce GTX 1050Ti 4 Gb	Windows 10

Tablo 5.5'de ABC kriptosistemi için hazırlanan uygulamanın hem standart hem de CUDA kullanılarak geliştirilen üç farklı ortamdaki toplam çalışma süreleri mikro saniye (µs) cinsinden verilmiştir.

Tablo 5.5 Programın Tamamının Çalışma Süreleri

	Standart	CUDA
İndeks 1	4336000 µs	6089299 µs
İndeks 2	5123000 µs	6589299 µs
İndeks 3	2214000 µs	4856900 µs

Tablo 5.6'da ABC kriptosistemi için hazırlanan uygulamanın üç farklı ortam için CPU üzerinde "memory" kullanımı gösterilmektedir.

Tablo 5.6 Uygulamanın Hafıza Tüketimi

	Standart	CUDA
İndeks 1	90 MB	76 MB
İndeks 2	100 MB	87 MB
İndeks 3	127 MB	118 MB

Tablo 5.7'de ABC kriptosistemi için hazırlanan uygulamanın "şifreleme" ve "şifre çözme" fonksiyonlarının çalışma süreleri üç farklı ortam için hem standart uygulama hem de CUDA kullanılarak geliştirilip mikro saniye cinsinden verilmiştir.

Tablo 5.7 Şifreleme ve Şifre Çözme Fonksiyonlarının Çalışma Süreleri

	Standart		CUDA	
	Şifreleme	Şifre Çözme	Şifreleme	Şifre Çözme
İndeks 1	16000 µs	132000 µs	8000 µs	3291599 µs
İndeks 2	19000 µs	145000 µs	11000 µs	3701789 µs
İndeks 3	10000 µs	78000 µs	3000 µs	2479900 µs

Uygulamaların çalıştırılması sonucunda elde edilen süreler ve "Tablo 5.8"de ve "Tablo 5.9"da ayrıntılı olarak gösterilen ABC kriptosistemi için gerekli olan fonksiyonların "anahtar üretimi", "şifreleme" ve "şifre çözme"de kaç defa çağrıldığını ve kullanıldığı sayılarak değerlendirilmiştir. Standart uygulama veya CUDA kullanılarak yazılmış kodda fonksiyonların çağrılma işleminde de fark edileceği üzere $GF(2^8)$ üzerinde toplama işleminin tamamı GPU üzerinde yapılmaktadır. Buna ek olarak toplam sürede bir artış gözlemse de "şifreleme" işleminin yapılmasında süre konusunda bir iyileşme sağlanmıştır. Süre bazında gelişime ek olarak hafıza kullanımında da bir gelişme sağlanmıştır. Süre ölçümü ise Tablo 5.5'de her iki kodda da "anahtar üretimi"nden önce başlayıp "şifre çözme"nin tamamlanmasıyla sonlandırılmıştır. Tablo 5.7'de ise şifreleme ve şifre çözme olarak standart uygulama ve CUDA kullanılarak geliştirilen uygulamanın süreleri ölçülmüştür.

Tablo 5.8 Standart Uygulama Fonksiyonların Çağırılma Sayısı

	Anahtar Üretimi	Şifreleme	Şifre Çözme	Toplam
Toplama	4452352	0	286720	4739072
Çıkarma	0	0	2064512	2064512
Çarpma	107740992	532480	2617792	110891264
Bölme	0	0	16256	16256
Tersini Alma	192	0	1	193
Matris Tersi	2	0	0	2
Matris Çarpımı	257	0	0	257
Matris Trans.	1	0	0	1
Denk. Kats. He.	128	0	0	128
Echelon	0	0	1	1

Tablo 5.9 CUDA Kullanılan Uygulamanın Fonksiyonları Çağırma Sayısı

	Anahtar Üretimi	Şifreleme	Şifre Çözme	Toplam
Toplama	0	0	0	0
Çıkarma	0	0	2064512	2064512
Çarpma	107740992	532480	2617792	110891264
Bölme	0	0	16256	16256
Tersini Alma	192	0	1	193
Matris Tersi	2	0	0	2
Matris Çarpımı	257	0	0	257
Matris Trans.	1	0	0	1
Denk. Kats. He.	128	0	0	128
Echelon	0	0	1	1

İndeks numarası 1 olan derleyici ve bilgisayarda standart kodun çalışması sonucu 4336000 mikro saniyelik bir sonuç elde edilmiştir. CUDA kullanılarak geliştirilen yapıda ise 6089299 mikro saniyelik bir sonuç elde edilmiştir. Burada standart koda göre %40'lık bir zaman artışı gözükmektedir. Memory tüketiminde ise standart uygulama 90 MB iken CUDA kullanılmış yapı 76 MB'lık bir sonuç elde etmiştir. Burada standart koda göre %15'lik bir memory tasarrufu sağlanmıştır.

Şifreleme fonksiyonuna göre 16000 mikro saniyeden 8000 mikro saniyeye gerileyerek %50'lik bir zaman kazanımı sağlanmıştır.

İndeks numarası 2 olan derleyici ve bilgisayarda standart kodun çalışması sonucu 5123000 mikro saniyelik bir sonuç elde edilmiştir. CUDA kullanılarak geliştirilen yapıda ise 6589299 mikro saniyelik bir sonuç elde edilmiştir. Burada standart koda göre %28'lik bir zaman artışı gözükmektedir. Memory tüketiminde ise standart uygulama 100 MB iken CUDA kullanılarak geliştirilen yapı 87 MB'lık bir sonuç elde edilmiştir. Burada standart koda göre %13'lük bir memory tasarrufu sağlanmıştır. Şifreleme fonksiyonuna göre 19000 mikro saniyeden 11000 mikro saniyeye gerileyerek %42'lik bir zaman kazanımı sağlanmıştır.

İndeks numarası 3 olan derleyici ve bilgisayarda standart kodun çalışması sonucu 2214000 mikro saniyelik bir sonuç elde edilmiştir. CUDA kullanılarak geliştirilen yapıda ise 4856900 mikro saniyelik bir sonuç elde edilmiştir. Burada standart koda göre %19'luk bir zaman artışı gözükmektedir. Memory tüketiminde ise standart uygulama 127 MB iken CUDA kullanılarak geliştirilen yapı 118 MB'lık bir sonuç elde edilmiştir. Burada standart koda göre %7'lik bir memory tasarrufu sağlanmıştır. Şifreleme fonksiyonuna göre 10000 mikro saniyeden 3000 mikro saniyeye gerileyerek %70'lik bir zaman kazanımı sağlanmıştır.

Uygulamaların çalışma ortamları fark etmeksizin, CUDA kullanılarak geliştirilen uygulamalar, performans olarak süre bazında standart uygulamaya daha yavaş olmakla birlikte hafıza kullanımında daha iyidir. Bu farkın sebebi ise $GF(2^8)$ üzerindeki toplama işleminin GPU üzerinde yapılmasıdır. GPU üzerinde gerçekleşen toplama işleminin üç tanesi şifre çözme fonksiyonu bir tanesi de anahtar üretimi fonksiyonu içerisinde gerçekleşmektedir.

Farklı ortamlarda farklı süreler elde edilmesini nedeni CPU ve GPU'dur. Her üç bilgisayarda da "visual studio 2017" kullanılmıştır. CPU ve GPU mimarilerinin farklılığı da (Tablo 5.4'te ayrıntılı olarak verilmiştir) süre ve memory değişkenliğini etkilemiştir.

6. SONUÇ ve GELECEK ÇALIŞMALAR

Günümüzde veriyi işlemek ve iletmek için kullanılan bilgisayarlar klasik mekanik ile tam olarak açıklanabilmektedir. Fakat madde daha küçük boyutlarda düşünüldüğünde klasik mekanikte geçerliliğini yitirmektedir. Klasik mekaniğin aksine kuantum mekaniği atomlar ve moleküllerin davranışları ile açıklanır. İşte tam bu noktadan sonra klasik şifreleme yöntemleri geçerliliğini yitirmeye başlıyor.

Çok değişkenli açık anahtar şifreleme sistemlerinden biri olan ABC kriptosisteminin, kuantum hesaplama saldırılarına karşı direnç gösterebileceğine inanılmaktadır. Bu çalışmada elde edilen sonuçlar neticesinde bilgisayarların çalışma ortamı fark etmeksizin thread'li bir yapı ile geliştirilmiş uygulama, süre bazında daha iyi sonuçlar vermektedir. Bunun sebebi uygulamada kullanılan thread sayesinde uygulamayı zaman konusunda yoran fonksiyon ve matris işlemlerinin paralel olarak hesaplanmasıdır. Süre bazında geliştirilmesi için thread yapısını sadece ön tanımlı değişkenler için değil, uygulamanın geneline entegre edilmesi daha iyi sonuçlar elde edilmesine olanak sağlamıştır.

Thread'li yapıya ek olarak ABC kriptosisteminin grafik işlemcisi NVIDIA tarafından geliştirilen CUDA ile birlikte GPU üzerinde de fonksiyon bazında çalıştırılması ile birlikte hem hafıza kullanımında hem de şifreleme işleminde performans gelişimi sağlanmıştır. Belli fonksiyonlar özelinde değil uygulamanın tamamının grafik işlemcisi üzerinde çalıştırılması standart işlemciye nazaran daha iyi sonuçlar elde edilmesine olanak sağlayacaktır.

Daha sonraki çalışmalar için, temel ABC kriptosistemi algoritmasının performans artışının sağlanması (hem CPU hem de GPU için), asimptotik karmaşıklığın azaltılması, farklı diller için yazılımının yapılması ve son olarak da herhangi bir projede kullanılabilir hale getirilerek kullanılması amaçlanmaktadır.

KAYNAKLAR DİZİNİ

Akleylek, S., Nuriyev, U., 2005, Steganografi ve Steganografinin Yeni Bir Uygulaması, Signal Processing and Communications Applications Conference, Proceedings of the IEEE 13th

Akleylek, S., Koyutürk, R., 2019, Kuantum Sonrası Güvenilir ABC Şifreleme Sisteminin Farklı Platformlardaki Uygulamaları, 12. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı

Benvenuto, C., J., 2012, Galois Field in Cryptography, https://sites.math.washington.edu/~morrow/336_12/papers/juan.pdf (Erişim tarihi: 01 Ocak 2020)

Buchmann, J.A. and Butin, D., 2016, Post-Quantum Cryptography: State of the Art. The New Codebreakers, LNCS 9100, 88-108 p.

Cho, H., Ravindran, B. and Jensen, E. D., 2006, An Optimal Real-Time Scheduling Algorithm for Multiprocessors.

Ding, J. and Schmidt, D., 2005, Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J. Keromytis, A.D. Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, Springer, Heidelberg, 164–175 p.

Ding, J., Petzoldt, A. and Wang. L.C., 2014, The cubic simple matrix encryption scheme, PQCrypto 2014, LNCS 8772, 76-87 p.

DiVincenzo, D. P., 2008, The Physical Implementation of Quantum Computation, Watson Research Center.

Garey, M.R. and Johnson, D.S., 1979, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York.

Garfinkel, S., 1995, PGP : Pretty good privacy, Beijing : O'Reilly.

KAYNAKLAR DİZİNİ (devam)

Grover, L.K., 1996, A fast quantum mechanical algorithm for database search, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, 212 p.

Hashimoto, **Y.**, 2016, A note on tensor simple matrix encryption scheme, IACR.

IBM, "What is quantum computing", https://www.ibm.com/quantum-computing/learn/what-is-quantum-computing/ (Erişim tarihi: 01 Ocak 2020)

IBM, Experience, https://www.ibm.com/quantum-computing/technology/experience/ (Erişim tarihi: 01 Ocak 2020)

Kak, A., Computer and Network Security, https://engineering.purdue.edu/kak/compsec/NewLectures/ (Erişim tarihi: 29 Ocak 2019)

Kayaoğlu., G., İ., T., "Soyut Matematık II", http://www.gilona.com/files/grup,%20halka,%20cisim.pdf (Erişim tarihi: 01 Ocak 2020)

Kocher, P.C., 1996, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, CRYPTO 1996, 104-113 p.

Landt, J., 2005, The history of RFID, IEEE Potentials, Volume: 24, Issue: 4, 8–11 p.

Lidl, R. and Niederreiter H., 1983, Finite Fields. Boston.

Lloyd, S., 1995, Quantum-Mechanical Computers, , Scientific American, Vol. 273, No. 4, 140-145 p.

Ocak, M.E., 2018, Kuantum Bilgisayarlar, Tübitak Bilim ve Teknik Dergisi Haziran.

KAYNAKLAR DİZİNİ (devam)

Peng, Z., Tang, S., Chen, J., Wu, C. and Zhang, X., 2016, Fast Implementation of Simple Matrix Encryption Scheme on Modern x64 CPU, ISPEC 2016, 151-166 p.

Petzoldt, A., Ding. J. and Wang, L.C., 2016, Eliminating decryption failures from the simple matrix encryption scheme, IACR

Rivest, R.L., Shamir, A., Adleman,L., 1977, On Digital Signatures and Public-Key Cryptosystems.

Rivest, R.L., Shamir, A. and Adleman, L., 1978, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Commun. ACM 21(2), 120–126 p.

Shor, P., 1994, Algorithms for quantum computation: discrete logarithms and factoring, In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 124–134 p.

Shor, **P.,** 1997, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, SIAM J. Computing, vol. 26, no. 5, 1484–1509 p.

Tao, C., Diene, A., Tang, S. and Ding, **J.**, 2013, Simple matrix scheme for encryption, PQCrypto, LNCS 7932, 231-242 p.

Tao, C., Xiang, H., Petzoldt, A. and Ding, J., 2015, Simple Matrix - a multivariate public key cryptosystem (MPKC) for encryption, Finite Fields and Their Applications 35, 352-368 p.

Tonbil, E., "Kuantum Bilgisayarlar", https://www.tonbil.com/kuantum-bilgisayarlar/ (Erişim tarihi: 01 Ocak 2020)

Williams, H., 1980, A modification of the RSA public key encryption procedure, IEEE Transactions on Information Theory, Vol. IT-26, No. 6.

KAYNAKLAR DİZİNİ (devam)

Jordan, S., Quantum Algorithm Zoo, http://quantumalgorithmzoo.org (Erişim tarihi: 05 Aralık 2019)

TEŞEKKÜR

Bu çalışmamın her safhasında ve her anında, manevi desteklerini her an hissettiğim ve hiç eksik etmeyen aileme, bilgi ve tecrübesiyle bana yön veren, beni destekleyen ve bu çalışmamda benden tüm yardımlarını esirgemeyen sayın hocalarım Prof. Dr. Urfat NURİYEV'e ve Doç. Dr. Sedat AKLEYLEK'e ve her zaman yanımda olan arkadaşlarıma çok teşekkür ederim.

ÖZGEÇMİŞ

Adı ve Soyadı : Ramazan KOYUTÜRK

Doğum Yeri : İZMİR

Doğum Tarihi : 07.03.1992

Eğitim Durumu

Lise : Buca Lisesi (2010)

Lisans : Ege Üniversitesi (Temmuz 2016)

Çalıştığı Kurum ve Yıl

Nesan Teknoloji A.Ş. (05.2019 - …)

Atanova İnternet Reklam Bilişim Ve Pazarlama Ltd. Şti (08.2015 – 12.2017)

Yayınlar

- S. Akleylek ve R. Koyutürk (2019). Kuantum Sonrası Güvenilir ABC Şifreleme Sisteminin Farklı Platformlardaki Uygulamaları, 12. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı, 16-17 Ekim 2019. İstanbul / TÜRKİYE.