

**2023-2024 AKADEMİK YILI
BAHAR DÖNEMİ**

BMT402 – Bitirme Tezi

Ders Sorumlusu:
Dr. Öğr. Üyesi Ahmet ALBAYRAK

**HOMOMORFİK ŞİFRELEMENİN
ANLATILDIĞI WİKİ WEB UYGULAMASI
GELİŞTİRİLMESİ**

Hazırlayan:
Ali Berdan KARASOY
Yahya ARI

Öğrenci No:
182120015
182120004

TEŞEKKÜR

Lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Dr. Öğr. Üyesi Ahmet ALBAYRAK’a en içten dileklerimle teşekkür ederim.

Bu çalışma boyunca yardımlarını ve desteklerini esirgemeyen sevgili aileme ve çalışma arkadaşlarıma sonsuz teşekkürlerimi sunarım.

14 Mayıs 2024

Adı Soyadı
Ali Berdan KARASOY
Yahya ARI

İÇİNDEKİLER

	<u>Sayfa No</u>
ŞEKİL LİSTESİ	v
KISALTMALAR	vi
ÖZET	vii
ABSTRACT	viii
1. GİRİŞ	1
1.1. LİTERATÜR ÖZETİ	1
1.2. TEZİN AMACI	2
1.3. LİTERATÜRE KATKI	3
2. MATERYAL VE YÖNTEM	4
2.1. YAZILIM MİMARİSİNİN BELİRLENMESİ.....	4
2.1.1.MVC (Model-View-Controller).....	4
2.1.1.1. Model.....	5
2.1.1.2. View.....	5
2.1.1.3. Controller.....	5
2.2. TÜMLEŞİK GELİŞTİRME ORTAMI (IDE) VE FRAMEWORK.....	6
2.3. KATMANLI MİMARİ.....	7
2.3.1. DataAccessLayer (Veri Erişim Katmanı).....	8
2.3.2. EntityLayer (Varlık Katmanı).....	8
2.3.3. BusinessLayer (İş Katmanı).....	9
2.4. SQL SERVER MANAGEMENT STUDIO.....	9
2.5. KULLANILAN KÜTÜPHANELER VE FRAMEWORK'LER.....	10
2.5.1. Microsoft SEAL (Simple Encrypted Arithmetic Library).....	10
2.5.2. Entity Framework Core tabanlı ASP.NET Core NuGet Paketleri.....	12
2.5.2.1. Microsoft.EntityFrameworkCore.Design.....	12
2.5.2.2. Microsoft.EntityFrameworkCore.SqlServer.....	12
2.5.2.3. Microsoft.EntityFrameworkCore.Tools.....	13
2.5.2.4. Microsoft.VisualStudio.Web.CodeGeneration.Design.....	13
3. HOMOMORFİK ŞİFRELEME.....	13
3.1. HOMOMORFİK ŞİFRELEME NEDİR.....	13

3.2. HOMOMORFİK ŞİFRELEME TARİHÇESİ.....	14
3.2.1. RSA Algoritması.....	15
3.2.2. RSA Algoritmasının Homomorfik Şifrelemeyle İlişkisi.....	17
3.3. HOMOMORFİK ŞİFRELEME YÖNTEMLERİ.....	17
3.3.1. Tam Homomorfik Şifreleme (Fully HE).....	17
3.3.2. Kısmi Homomorfik Şifreleme (Partially HE).....	19
3.3.2.1. RSA Algoritması ve HE.....	19
3.3.2.2. ElGamal Algoritması ve HE.....	19
3.3.2.3. Paillier Algoritması ve HE.....	20
3.3.3. Somut Homomorfik Şifreleme (Concretely HE).....	21
3.4. UYGULAMA ALANLARI.....	21
3.4.1. Bulut Bilişim Güvenliği ve HE.....	21
3.4.2. Ulusal Güvenlik ve HE.....	22
3.4.3. Veri Madenciliği ve HE.....	22
4. HOMOMORFİK ŞİFRELEME WİKİ SAYFASI.....	23
4.1. TASARIM SÜRECİ.....	23
4.2. VERİ TABANI İŞLEMLERİ.....	25
4.2.1. Entity Framework Kurulumu.....	25
4.2.2. DbContext Sınıfı Oluşturma.....	25
4.2.3. Entity Sınıflarının Oluşturulması.....	26
4.2.4. Controller Yapısı.....	28
4.2.5. Migration İşlemleri.....	29
4.2.6. CRUD İşlemleri.....	30
4.2.7. Verilerin Veritabanından Sayfalara Çekilmesi.....	31
4.2.8. Veritabanı.....	32
4.3. HOMOMORFİK ŞİFRELEME UYGULAMASI.....	33
5. SONUÇLAR.....	38
6. KAYNAKÇA.....	39
ÖZGEÇMİŞ	41

ŞEKİL LİSTESİ

	<u>Sayfa No</u>
Şekil-1: Model-View-Controller Çalışma Mantığı.....	6
Şekil-2: Proje'nin Oluşturulması.....	7
Şekil-3: Framework Belirlenmesi.....	7
Şekil-4: Proje'nin Data Access Katmanı.....	8
Şekil-5: Proje'nin Entity Katmanı.....	9
Şekil-6: Proje'nin Business Katmanı.....	9
Şekil-7: SQL Server LocalDB.....	10
Şekil-8: Microsoft SEAL Kütüphanesi.....	11
Şekil-9: Entity Framework Core.....	12
Şekil-10: RSA Algoritmasının Akış Şeması.....	15
Şekil-11: RSA Algoritması Çalışma Mantığı.....	16
Şekil-12: Tam Homomorfik Şifreleme Şeması.....	18
Şekil-13: RSA Algoritmasının Kısmi Homomorfik Şifreleme İşlemi.....	19
Şekil-14: ElGamal Algoritmasının Toplamaya Göre Kısmi Homomorfik Şifrelemesi.....	20
Şekil-15: Paillier Algoritmasının Toplamaya Göre Kısmi Homomorfik Şifrelemesi.....	20
Şekil-16: Web Sayfasının Genel Tasarımı.....	24
Şekil-17: Proje'nin Views Hiyerarşisi.....	24
Şekil-18: Veritabanı Bağlantısı.....	25
Şekil-19: DbSet Tanımlamaları.....	26
Şekil-20: Ana Sayfa Entity Sınıfı.....	27
Şekil-21: Hakkımızda Sayfasının Entity Sınıfı.....	28
Şekil-22: Controller Yapısı.....	28
Şekil-23: CRUD işlemlerini (Create, Read, Update, Delete).....	30
Şekil-24: Verilerin Veritabanından Çekilmesi.....	31
Şekil-25: Veritabanı Görsel 1.....	32
Şekil-26: Veritabanı Görsel 2.....	32
Şekil-27: Homomorfik Şifreleme Uygulaması.....	33
Şekil-28: SEAL Kütüphanesi Değişken Tanımlamaları.....	33
Şekil-29: Şifreleme Metodu.....	33
Şekil-30: Şifreleme İşlemi.....	34
Şekil-31: Base64 Formatında Kodlama.....	35
Şekil-32: Şifreleme Formu.....	36

KISALTMALAR

API	Application Programming Interface (Uygulama Programlama Arayüzü)
CHE	Constrained Homomorphic Encryption (Somut Homomorfik Şifreleme)
CLI	Command-Line Interface (Komut Satırı Arayüzü)
FHE	Fully Homomorphic Encryption (Tam Homomorfik Şifreleme)
HE	Homomorphic Encryption (Homomorfik Şifreleme)
HTTP	Hypertext Transfer Protocol (Hipermetin Transfer Protokolü)
IDE	Integrated Development Environment (Tümleşik Geliştirme Ortamı)
LTS	Long-Term Support (Uzun Süreli Destek)
MVC	Model-View-Controller
ORM	Object-Relational Mapping (Nesne-İlişkisel Eşleme)
PHE	Partially Homomorphic Encryption (Kısmi Homomorfik Şifreleme)
RSA	Rivest-Shamir-Adleman
SEAL	Simple Encrypted Arithmetic Library
SRP	Single Responsibility Principle (Tek Sorumluluk Prensibi)
SSMS	SQL Server Management Studio
UI	User Interface (Kullanıcı Arayüzü)
URL	Uniform Resource Loader (Birleşik Kaynak Konumlayıcı)

ÖZET

HOMOMORFİK ŞİFRELEMENİN ANLATILDIĞI WİKİ WEB UYGULAMASI GELİŞTİRİLMESİ

Ali Berdan KARASOY
Yahya ARI

Düzce Üniversitesi
Teknoloji Fakültesi Bilgisayar Mühendisliği Bitirme Tezi

Danışman: Dr. Öğr. Üyesi Ahmet ALBAYRAK

Mayıs 2024, 40 sayfa

Bu tez çalışması, homomorfik şifrelemenin temel kavramlarını, çalışma mantığını ve programlama dillerindeki uygulamalarını anlamak amacıyla bir wiki web uygulamasının geliştirilmesini hedefler. Bu hedef, Model-View-Controller (MVC) ve katmanlı mimari alt yapısıyla entegre bir şekilde gerçekleştirilir. MVC, geliştirme sürecinde web uygulaması mimarisi için temel olan katmanlı yapıyı kullanarak bir wiki web sayfası aracılığı ile homomorfik şifrelemeyi teorik ve programlama dilleri aracılığı ile pratik anlamda öğretmeyi amaçlamıştır.

Anahtar sözcükler: Homomorfik şifreleme, Temel kavramlar, Wiki web uygulaması, MVC (Model-View-Controller), Katmanlı mimari.

ABSTRACT

DEVELOPING A WIKI WEB APPLICATION EXPLAINING HOMOMORPHIC ENCRYPTION

Ali Berdan KARASOY
Yahya ARI

Düzce University
Faculty of Technology, Computer Engineering
Undergraduate Thesis

Supervisor: Doctoral Faculty Member Ahmet ALBAYRAK

May 2024, 40 pages

This thesis aims to develop a wiki web application to understand the fundamental concepts, workings, and applications of homomorphic encryption in programming languages. This goal is achieved by integrating Model-View-Controller (MVC) and layered architectural infrastructure. MVC, utilizing the foundational layered structure for web application development, aims to teach homomorphic encryption theoretically and practically through a wiki web page.

Keywords: Homomorphic encryption, Fundamental concepts, Wiki web application, MVC (Model-View-Controller), Layered architecture.

1. GİRİŞ

Veri güvenliği günümüz dijital çağında en önemli konulardan biri haline gelmiştir. Özellikle, hassas verilerin saklanması, işlenmesi ve iletilmesi sırasında gizliliğin ve güvenliğin sağlanması büyük önem taşır. Bu noktada, şifreleme metotları kritik bir rol oynamaktadır.

Şifreleme, verilerin dışarıdan erişilmesini engellemek için kullanılan bir yöntemdir. Ancak, geleneksel şifreleme yöntemleri verileri şifreledikten sonra işleme tabi tutulamaz hale getirirler. Bu durum, verilerin işlenmesi veya analiz edilmesi gerektiğinde sorunlar doğurabilir. İşte burada homomorfik şifreleme devreye girer.

1.1. Literatür Özeti:

Homomorfik şifreleme, şifrelenmiş veriler üzerinde matematiksel işlemleri gerçekleştirebilme yeteneği sunar. Bu, şifreli veriler üzerinde hesaplamaların yapılabilmesi anlamına gelir. Örneğin, iki şifreli sayı toplanabilir ve sonuç yine şifreli olarak elde edilebilir. Bu özellik, verilerin gizliliğini korurken aynı zamanda işleme tabi tutulabilmesine olanak tanır.

Homomorfik şifreleme, birçok alanda büyük potansiyele sahiptir. Özellikle, bulut bilişim, sağlık sektörü, finansal hizmetler ve veri analitiği gibi alanlarda gizliliğin korunması ve güvenli veri işleme ihtiyacı giderek artmaktadır. Homomorfik şifreleme, bu alanlarda veri güvenliği ve gizliliğin sağlanmasına katkıda bulunarak güvenilir bir çözüm sunar.

Tez, homomorfik şifreleme ve MVC mimarisinin ayrıntılı bir literatür özetini sunmaktadır. Homomorfik şifrelemenin temelleri, farklı türleri ve kullanımları, ayrıca programlama dillerindeki uygulamaları incelenmiştir. Mevcut literatürdeki araştırmalar ve uygulamaların bir özeti sunulurken, homomorfik şifrelemenin farklı bağlamlarda nasıl kullanılabileceği ve hangi alanlarda daha fazla potansiyel sunduğu vurgulanmıştır.

Aynı şekilde, MVC (Model-View-Controller) mimarisinin avantajları ve uygulamaları da

derinlemesine bir analizle ele alınmıştır. MVC'nin yazılım geliştirme süreçlerindeki rolü, kullanım alanları ve veri güvenliği ile ilişkisi incelenerek, bu mimarinin neden tercih edildiği ve hangi durumlarda daha etkili olduğu açıklanmıştır.

1.2. Tezin Amacı:

Tezin amacı, MVC (Model-View-Controller) katmanlı mimari kullanarak homomorfik şifreleme konusunda bir wiki web sayfası oluşturmak ve ziyaretçilere bu konuyu detaylı bir şekilde anlatmak için bir platform sağlamaktır. Homomorfik şifreleme, verilerin gizliliğini korurken işlemlerin yapılmasına izin veren önemli bir kriptografik tekniktir. Bu tez, homomorfik şifrelemenin temellerini, programlama dillerindeki kullanımlarını ve önemini açıklamayı amaçlamaktadır.

Bu çalışma, kullanıcıların veri güvenliğini artırmak ve gizliliklerini korumak için homomorfik şifreleme konusunda bilinçlenmelerine ve bu teknolojinin pratik kullanımlarını öğrenmelerine katkıda bulunmayı amaçlamaktadır. Tezin odak noktası, kullanıcıların homomorfik şifreleme hakkında daha derin bir anlayış geliştirmelerine yardımcı olacak bilgileri sunmak ve bu teknolojinin gerçek dünya uygulamalarında nasıl kullanılabileceğini göstermektir.

Bu doğrultuda, tezin ilerleyen bölümlerinde homomorfik şifrelemenin temel prensipleri ve farklı türleri detaylı bir şekilde incelenecek ve programlama dillerindeki kullanımlarına örnekler verilecektir. Ayrıca, MVC mimarisinin homomorfik şifreleme ile nasıl entegre edilebileceği ve bu entegrasyonun bir web uygulaması üzerinde nasıl deneyim ve uygulanabileceğini pratik örneklerle gösterilecektir.

Böylelikle, tez çalışması homomorfik şifreleme konusunda hem teorik bir anlayış sağlayacak hem de bu teknolojinin gerçek dünya uygulamalarında nasıl kullanılabileceğini göstererek kullanıcıların bilgi düzeyini artıracaktır. Bu sayede, kullanıcılar homomorfik şifrelemenin potansiyelini daha iyi anlayacak ve bu teknolojiyi günlük yaşamlarında kullanma konusunda daha kendilerine güvenli olacaklardır.

1.3. Literatüre Katkı:

İlk olarak, homomorfik şifrelemenin temel prensipleri ve kullanım alanları hakkında kapsamlı bir bilgilendirme hazırlanmıştır. Homomorfik şifrelemenin ne olduğu, nasıl çalıştığı ve neden önemli olduğu gibi konular ele alındı.

Homomorfik şifreleme uygulamalarının farklı programlama dillerinde nasıl geliştirilebileceği incelenmiştir. Python, Java, C# ve JavaScript gibi yaygın olarak kullanılan dillerde homomorfik şifreleme örnekleri sunulmuş ve her bir dil için kullanılan kütüphaneler ve araçlar açıklanmıştır. Ardından, homomorfik şifreleme uygulamalarının kullanıcı dostu arayüzlerle nasıl desteklenebileceği ele alınmıştır. Kullanıcıların homomorfik şifreleme işlemlerini kolayca gerçekleştirebilecekleri arayüz tasarımları yapılmış ve kullanıcı deneyimi konularına odaklanılacaktır.

Son olarak, kullanıcıların homomorfik şifreleme işlemlerini gerçekleştirebilecekleri interaktif formlar ve pratik örnekler sunulmuştur. Bu örnekler, kullanıcıların homomorfik şifreleme konusunda daha iyi anlamalarını ve uygulamalarını deneyimlemelerini sağlayacaktır.

Bu çalışma, homomorfik şifreleme konusunda literatüre yeni bir bakış açısı getirerek, farklı programlama dillerinde ve platformlarda homomorfik şifreleme uygulamalarının nasıl geliştirilebileceğini ve kullanıcı dostu arayüzlerle nasıl desteklenebileceğini açıklamayı amaçlamaktadır. Bu sayede, homomorfik şifreleme teknolojisinin daha geniş bir kitleye yayılmasına ve kullanılmasına katkıda bulunulacaktır.

2. MATERYAL VE YÖNTEM

Bu bölümde, tezin yöntem ve materyal bölümüne giriş yapılacaktır. Araştırmanın yapısal bir özeti sunularak, kullanılan metodoloji ve kaynaklar hakkında genel bir bakış sağlanacaktır. Homomorfik şifreleme konusundaki çalışmamız, bu bölümde ele alınan materyel ve yöntemlerin ayrıntılı bir açıklamasını sunacaktır. Kullanılan kütüphaneler, yazılım mimarisi ve veri kaynakları hakkında bilgi verilecektir. Ayrıca, araştırmanın etik kurallara uygunluğunu sağlamak için alınan önlemler de açıklanacaktır. Bu şekilde, okuyucular araştırmanın yapısal temelini anlayacak ve sonuçların geçerliliğini değerlendirebileceklerdir.

2.1. Yazılım Mimarisinin Belirlenmesi:

2.1.1. MVC (Model-View-Controller)

Tez çalışması kapsamında kullanılan yazılım mimarisi MVC (Model-View-Controller) olarak belirlenmiştir. ASP.NET Core MVC çerçevesi, ASP.NET Core ile kullanılmak üzere iyileştirilmiş hafif, açık kaynak, yüksek oranda test edilebilir bir sunu çerçevesidir [1]. Bu mimari, yazılımın farklı katmanlarını (veri işleme, kullanıcı arayüzü, iş mantığı) ayrıştırarak modüler bir yapı oluşturmayı hedefler. Bu, kodunuzun daha düzenli, bakımı daha kolay ve yeniden kullanılabilir olmasını sağlar. Her bileşen kendi amacına odaklanır ve birbirinden bağımsız olarak geliştirilebilir. Bu da kodun daha modüler olmasını ve gerektiğinde değişiklik yapmanın daha kolay olmasını sağlar.

MVC, HTML, CSS ve JavaScript ile uyumlu bir yapı sağlar. Bu da ön uç geliştiricilerin daha rahat çalışmasını ve tasarımı kolayca entegre etmelerini sağlar. MVC, hem sunucu tarafı hem de istemci tarafı geliştirme için uygun bir yapı sunar. C# gibi güçlü sunucu tarafı dilleriyle sunucu mantığını yönetirken, HTML, CSS ve JavaScript ile istemci tarafı deneyimini şekillendirir. MVC'nin geniş bir topluluğu vardır ve bu da geliştiricilerin ihtiyaç duydukları her türlü kaynağa ve yardıma erişmelerini sağlar. Ayrıca, çeşitli geliştirme araçları ve kütüphaneler mevcuttur, bu da geliştirme sürecini hızlandırır.

2.1.1.1. Model:

Model, uygulamanın veri ve iş mantığını temsil eder. Bu kısım, verilerin depolandığı ve işlendiği yerdir. Desenin merkezi bileşeni. Uygulamanın kullanıcı arayüzünden bağımsız dinamik veri yapısıdır [2]. Veritabanı işlemleri, dosya işlemleri, API çağrıları gibi işlemler bu katmanda gerçekleştirilir. Model, genellikle uygulama içindeki diğer bileşenlerle doğrudan etkileşim kurmaz. Bunun yerine, Controller ile iletişim kurarak veri değişikliklerini ve güncellemelerini sağlar.

Model, genellikle yeniden kullanılabilir ve modüler olacak şekilde tasarlanır. Bu, farklı parçaların (örneğin, bir kullanıcı nesnesi veya bir ürün nesnesi gibi) aynı Model sınıfı içinde tanımlanabileceği anlamına gelir. Bu yaklaşım, kod tekrarını azaltır ve bakımı kolaylaştırır. Genellikle diğer katmanlara bağımlı olmadan test edilebilir. Bu, Model'deki iş mantığının doğru çalışıp çalışmadığını doğrulamak için kolayca birim testleri yazabileceğiniz anlamına gelir.

2.1.1.2. View:

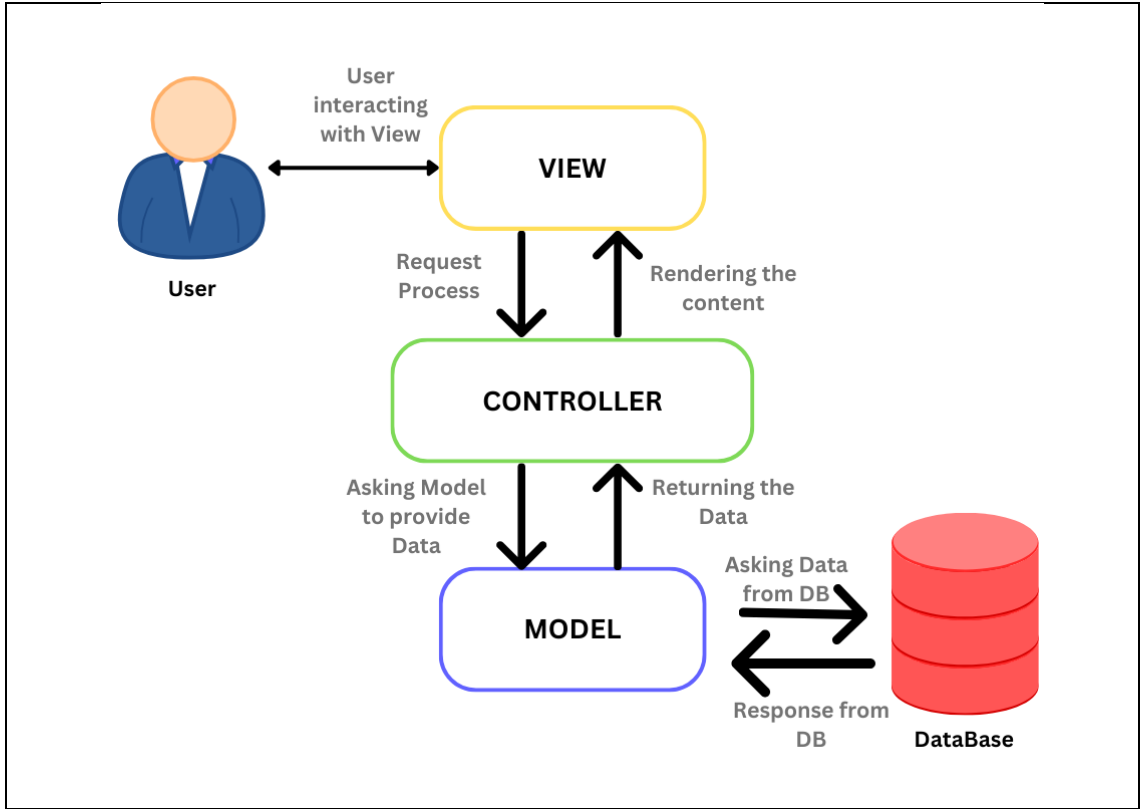
View, kullanıcı arayüzünün (UI) temsil edildiği katmandır. ASP.NET Core MVC'de görünüm, işaretlemede RazorC# programlama dilini kullanan .cshtml dosyalarıdır [3]. HTML, CSS, JavaScript veya benzeri teknolojiler kullanılarak oluşturulur. Bu katman, kullanıcıya görsel ve işlevsel olarak uygulamanın sunduğu her şeyi gösterir. Görünüm, kullanıcı girişlerini alır ve Controller'a yönlendirir. Model ile etkileşime giremez, sadece Controller'dan gelen verileri görüntüler. Controller üzerinden Model'den veri alır ve bu verileri kullanarak kullanıcı arayüzünü oluşturur. Bu, View ve Model arasında sıkı bir bağlantı olmamasını ve böylece uygulamanın daha modüler olmasını sağlar.

2.1.1.3. Controller:

Controller (Denetleyici), MVC (Model-View-Controller) deseninin merkezinde yer alan katmandır ve uygulamanın iş mantığını yönetir. Controller, kullanıcı tarafından yapılan istekleri işler [4]. Örneğin, bir web uygulamasında bir URL'ye yapılan istekler, Controller tarafından karşılanır ve bu isteklere göre işlemler yapılır. Controller, Model ve View arasında bir aracıdır. Kullanıcıdan gelen istekleri alır, gerektiğinde Model'den veri alır veya günceller, sonra bu verileri kullanarak uygun Görünümü seçer ve Görünüm'e verileri ileterek kullanıcıya sunar. Kullanıcı tarafından yapılan etkileşimler (tıklamalar, form gönderimleri vb.), Controller tarafından algılanır ve gerektiğinde uygun işlemleri yapmak

için işlenir. Örneğin, bir butona tıklanması durumunda ilgili işlevin çağrılması gibi. Controller, işlem sonuçlarını kullanıcıya göstermek için uygun View'i seçer ve Model'den aldığı veya oluşturduğu verileri bu View'e ileterek kullanıcı arayüzünün güncellenmesini sağlar.

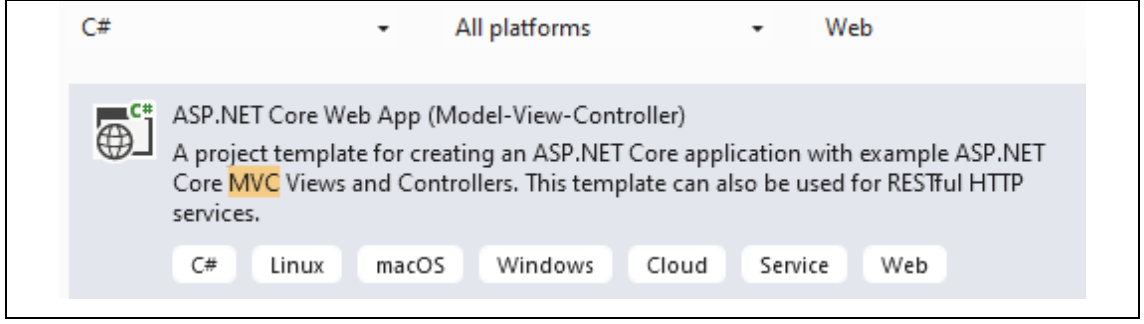
Modüler ve Tek Sorumluluk İlkesi (SRP) Kuralına Uyar: İyi bir Controller, tek bir işlevi yerine getirmeli ve bağımsız olmalıdır. Bu, kodunun okunabilirliğini ve yeniden kullanılabilirliğini artırır ve bakımını kolaylaştırır.



Şekil-1: Model-View-Controller Çalışma Mantiğı. [5]

2.2. Tümüleşik Geliştirme Ortamı (IDE) Ve Framework

Homomorfik şifreleme wiki web sayfası geliştirme sürecinde kullanılan geliştirme ortamı (IDE) olan Visual Studio, projenin hayata geçirilmesinde önemli bir rol oynadı. Bu projeyi MVC (Model-View-Controller) mimarisiyle şekillendirildi ve altyapı olarak .NET 8'in uzun süreli destek sürümünü tercih edildi.



Şekil-2: Proje'nin Oluşturulması.

Projenin gelecekteki gereksinimlerine uygun olması göz önünde bulundurularak bu, projenin uzun vadeli büyüme ve gelişme planlarına uygun bir temel oluşturması için projede kullanılan hedef framework .Net8.0 (Long Term Support) olarak belirlenmiştir.

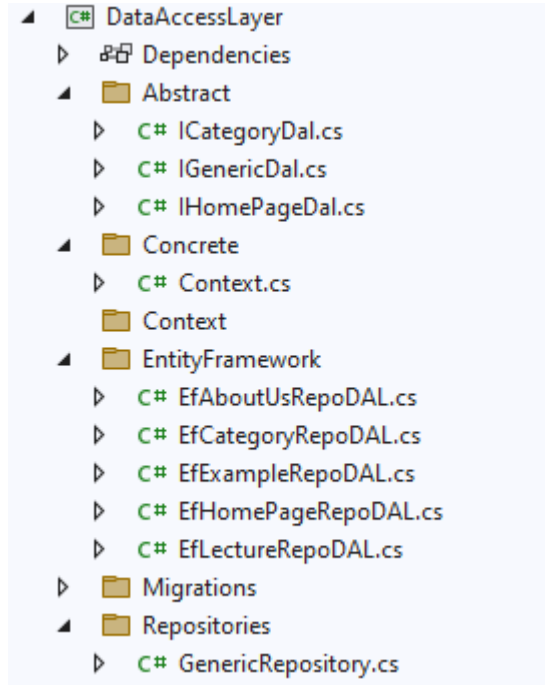
Şekil-3: Framework Belirlenmesi.

2.3. Katmanlı Mimari:

MVC (Model-View-Controller) mimarisi, birçok yazılım projesinde yaygın olarak kullanılan bir yapıdır. Bununla birlikte, MVC'nin daha karmaşık projelerde daha iyi yönetilebilmesi ve ölçeklenebilmesi için katmanlı bir mimari benimsenmesi yaygındır. İşte katmanlı mimari kullanmanın ve DataAccessLayer, EntityLayer ve BusinessLayer'ın bir MVC projesinde nasıl kullanıldığının açıklaması.

2.3.1. DataAccessLayer (Veri Eriřim Katmanı):

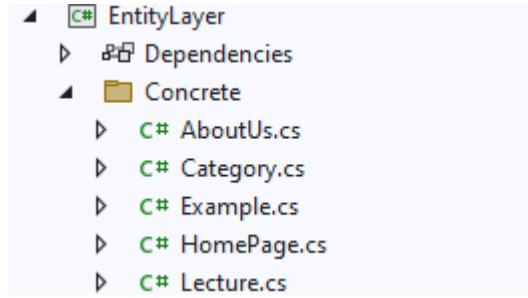
Bu katman, veritabanına eriřimi yönetir. Veritabanı ile iletiřim kurar, sorguları yürütür ve veri kaynağına eriřimi sağlar. Bu katmanda genellikle ORM (Object-Relational Mapping) araçları kullanılır. Bu, veritabanı tablolarını ve iliřkilerini nesnelere dönüřtürür ve veritabanı iřlemlerini yönetir [6]. DataAccessLayer, veriye eriřim kodunu soyutlar ve projenin geri kalan kısımlarından ayrı tutar.



řekil-4: Proje'nin Data Access Katmanı.

2.3.2. EntityLayer (Varlık Katmanı):

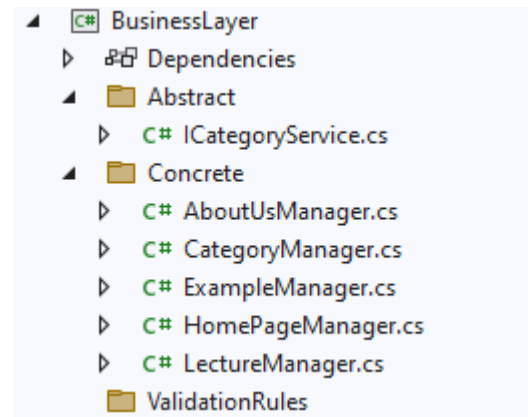
Bu katman, veritabanındaki tablolara karřılık gelen nesneleri içerir. Veritabanındaki her bir tablo, bu katmanda bir varlık olarak temsil edilir. Varlık sınıfları genellikle veritabanı tablolarının alanlarını ve iliřkilerini yansıtır [7]. Bu sınıflar, veri iřleme iřlevselliğini içermez; yalnızca veriyi temsil ederler.



Şekil-5: Proje'nin Entity Katmanı.

2.3.3. BusinessLayer (İş Katmanı):

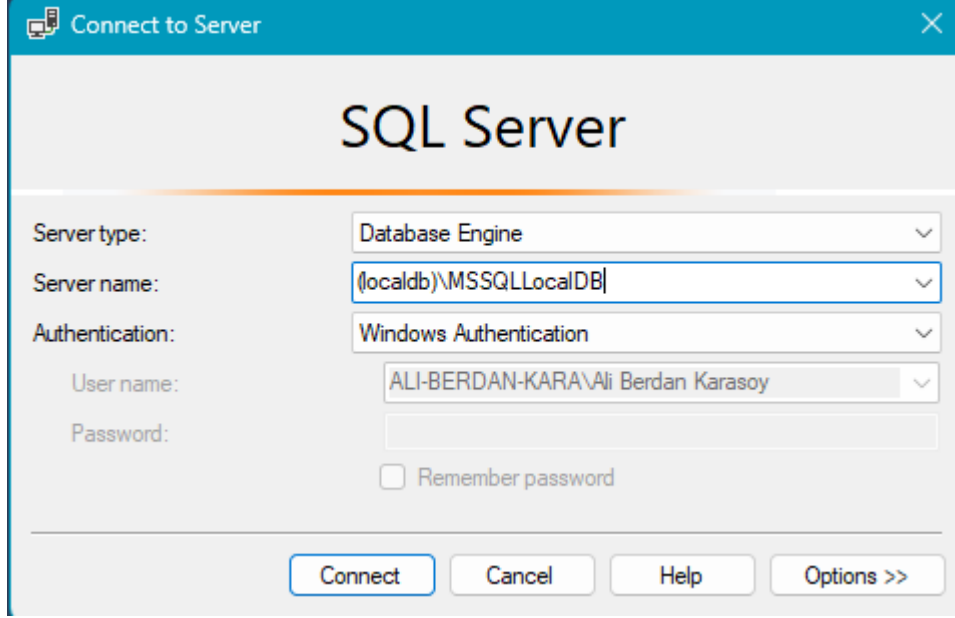
Bu katman, uygulamanın iş mantığını içerir. İş kurallarını uygular, doğrulama işlemlerini gerçekleştirir ve iş süreçlerini yönetir. BusinessLayer, genellikle varlık ve veri erişim katmanları arasında bir aracıdır. Veri erişiminden gelen veriyi işler, iş kurallarını uygular ve sonuçları sunucu tarafında işler. Bu katman, uygulamanın dış dünyayla olan etkileşimini yönetir ve MVC'nin kontrolörlerine veri sağlar.



Şekil-6: Proje'nin Business Katmanı.

2.4. SQL Server Management Studio:

MVC web projelerinde SQL Server Management Studio'nun (SSMS) kullanılması, genellikle MVC projelerinde tercih edilen bir yaklaşımdır çünkü bu, veritabanı işlemlerini (oluşturma, sorgulama, güncelleme, silme vb.) yönetmek için SQL Server'ı kullanmayı içerir. MVC projeleri genellikle Microsoft teknolojileriyle geliştirilir ve SQL Server, bu ekosistemin önemli bir parçasıdır. SSMS, SQL Server veritabanlarıyla sorunsuz bir şekilde entegre olur ve MVC projenizin veritabanı katmanıyla etkileşimde bulunmanıza olanak tanır.



Şekil-7: SQL Server LocalDB

"(localdb)\MSSQLLocalDB" SQL Server LocalDB özelliğini ifade eder. SQL Server LocalDB, geliştirme ve test amacıyla hafif bir veritabanı motoru sağlayan bir özelliktir. Bu, bir sunucu hizmeti olarak çalışmaz, ancak yerel bir dosya tabanlı veritabanı hizmeti olarak çalışır. "(localdb)" öneki, yerelde çalışan bir SQL Server veritabanı örneğini belirtir. Bu, yereldeki bir veritabanı dosyasına erişim sağlar.

"MSSQLLocalDB" öneki, varsayılan olarak yüklenen bir SQL Server LocalDB örneğini ifade eder. Bu, bilgisayarınızda yüklü olan SQL Server LocalDB örneğini belirtir. Bu şekilde kullanıldığında, "(localdb)\MSSQLLocalDB", yereldeki varsayılan SQL Server LocalDB örneğine bağlanmayı ifade eder. Bu şekilde bir adlandırma, geliştirme veya test senaryolarında yerel veritabanı kullanılması gerektiğinde sıkça karşımıza çıkar.

2.5. Kullanılan Kütüphaneler Ve Framework'ler:

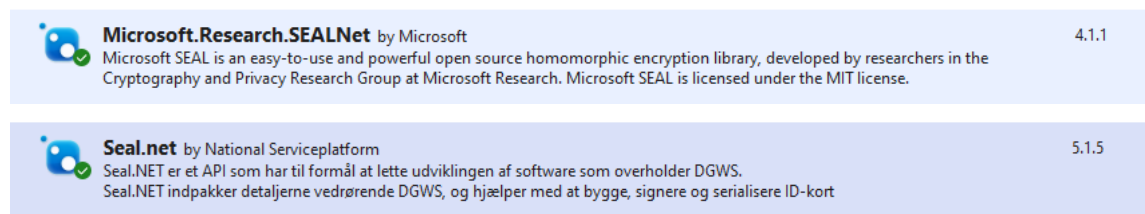
2.5.1. Microsoft SEAL (Simple Encrypted Arithmetic Library):

Microsoft SEAL (Simple Encrypted Arithmetic Library), gizlilik koruması gerektiren hassas veriler üzerinde işlem yaparken şifreleme ve hesaplama yapmayı sağlayan bir kütüphanedir. Açık kaynaklıdır (MIT Lisansı altında) ve harici bağımlılıklar olmadan standart C++ ile yazılmıştır ve bu nedenle çapraz platformda derlenebilir. C# ile yazılmış resmi bir .NET sarmalayıcı mevcuttur ve .NET uygulamalarının SEAL ile etkileşimini

kolaylaştırır [8]. Özellikle homomorfik şifreleme konseptine dayanır. Homomorfik şifreleme, şifreli veriler üzerinde doğrudan işlem yapılmasına izin veren bir şifreleme türüdür; bu da, verilerin şifreli olarak kalmasını sağlar, ancak işleme ve analiz yapılabilir. Microsoft SEAL, C++ dilinde geliştirilmiştir ve .NET Core, .NET Framework ve Python gibi birçok farklı platformda kullanılabilir. Veri analizi, makine öğrenmesi, bulut hesaplama ve diğer alanlarda kullanılan gizlilik odaklı uygulamalarda sıkça tercih edilir. Bu kütüphane, çeşitli şifreleme şemaları ve algoritmaları sunar, özellikle de tamamen homomorfik şifreleme (Fully Homomorphic Encryption - FHE) ve kısmi homomorfik şifreleme (Partially Homomorphic Encryption - PHE) gibi. Bu, farklı güvenlik ve işlem gereksinimlerine uygun olarak esnek bir kullanım sağlar.

SEAL kütüphanesi, Microsoft'un araştırma ve geliştirme birimi tarafından sürekli olarak güncellenmekte ve geliştirilmektedir. Ayrıca, belirli bir işlem için gerekli olan şifreleme seviyesini ve performansı dikkate alarak çeşitli yapılandırma seçenekleri sunar. Bu, farklı kullanım senaryolarına ve gereksinimlere uygun esneklik sağlar.

Visual Studio derleme yapılandırma menüsünden Microsoft SEAL'in Hata Ayıklama modunda (optimizasyon olmadan) veya Yayınlama modunda derlenip derlenmemesi kolayca değiştirilebilir. Lütfen dikkat edin, Yayınlama modu dışında, performans Hata Ayıklama modunda Yayınlama modundan kat ve kat daha kötü olacağından, Hata Ayıklama modunun sadece SEAL'in kendisini hata ayıklama amacıyla kullanılması gerektirir [9].



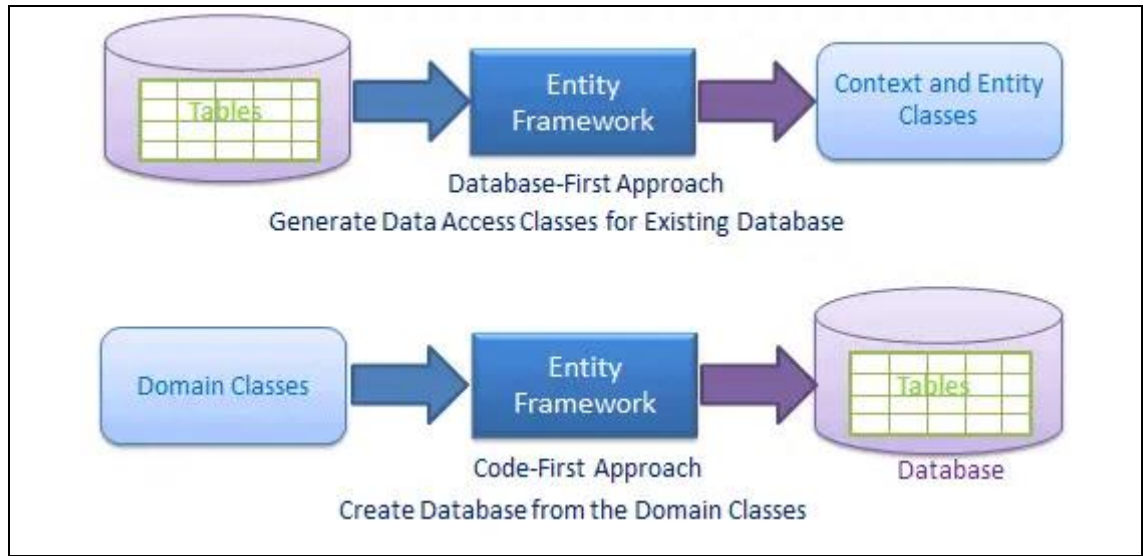
Şekil-8: Microsoft SEAL Kütüphanesi.

Visual Studio'da Tools menüsü altında NuGet Package Manager aracılığıyla Microsoft SEAL kütüphanelerini projeye eklendi. Console ile SEAL kütüphanesini kurmak için aşağıdaki komut çalıştırılabilir. Paket başarıyla yüklendikten sonra, projenizi yeniden derleyerek kütüphaneyi kullanmaya başlanabilir.

2.5.2. Entity Framework Core tabanlı ASP.NET Core Web Sayfası Geliştirirken Kullanılan NuGet Paketleri.

Bu paketler, genellikle ASP.NET Core projesinin csproj dosyasında veya Package Manager Console aracılığıyla projeye eklenir. Bu paketleri projenize ekleyerek, Entity Framework Core tabanlı bir ASP.NET Core projesi geliştirmeye başlayabilir ve veritabanı işlemleri ile ilgili kolaylık sağlayabilirsiniz.

Bu paketlerin işlevleri şu şekildedir:



Şekil-9: Entity Framework Core [10].

2.5.2.1. Microsoft.EntityFrameworkCore.Design:

Entity Framework Core ile kod tabanlı veritabanı işlemleri yapmak için geliştirme araçları sağlar. Özellikle, DbContext sınıflarını kodlamak ve migration'ları yönetmek için kullanılır. Bu paket, migration'ları oluşturmak, uygulamak, geri almak ve yönetmek için gerekli araçları içerir [11]. Projede 8.0.4 versiyonu kullanılmıştır.

2.5.2.2. Microsoft.EntityFrameworkCore.SqlServer:

Entity Framework Core tabanlı bir ASP.NET Core projesinde SQL Server veritabanı ile etkileşim kurmak için gerekli olan SQL Server sağlayıcısını içerir. Bu paket, SQL Server'a erişim sağlayan Entity Framework Core tarafından kullanılan bileşenleri içerir. Projede 8.0.4 versiyonu kullanılmıştır.

2.5.2.3. Microsoft.EntityFrameworkCore.Tools:

Entity Framework Core migration işlemlerini yönetmek için gerekli olan araçları sağlar. Bu paket, Package Manager Console veya .NET CLI üzerinden migration'ları oluşturmak, uygulamak ve geri almak için gerekli komutları içerir. Projede 8.0.4 versiyonu kullanılmıştır.

2.5.2.4. Microsoft.VisualStudio.Web.CodeGeneration.Design:

ASP.NET Core projelerinde kod üretimi (scaffolding) işlemlerini yapmak için gerekli araçları içerir. Özellikle, Controller ve Views gibi MVC bileşenlerini otomatik olarak oluşturmak için kullanılır. Bu paket, Entity Framework Core ile etkileşim kurarak veritabanı işlemlerini gerçekleştirmek ve bu işlemlere dayalı olarak Controller ve Views kodlarını otomatik olarak oluşturmak için kullanılır. Projede 8.0.2 versiyonu kullanılmıştır.

3. HOMOMORFİK ŞİFRELEME

3.1. Homomorfik Şifreleme Nedir:

Homomorfik şifreleme, şifrelenmiş veriler üzerinde hesaplamalar yapmaya izin veren bir şifreleme yöntemidir. Bu teknoloji, şifresi çözülmeden önce bile işlemler yapılmasını sağlar ve sonuç olarak elde edilen şifreli veri, şifre çözüldüğünde orijinal veri üzerinde yapılan işlemlerin sonucuna eşittir [12]. Bu özellik, dış kaynaklı depolama, hesaplama ve gizliliği koruma gibi çeşitli alanlarda kullanılabilir.

Homomorfik şifreleme, özellikle ticari bulut ortamlarında şifrelenmiş verilerin işlenmesine ve dışarıdan kaynaklanmasına olanak tanır. Yüksek düzeydeki endüstrilerde, veri paylaşımını kısıtlayan gizlilik engellerini ortadan kaldırarak yeni hizmetlerin sunulmasını sağlar.

Bu teknoloji, şifreli metinler üzerinde hesaplama yapılmasına izin verir ve sonuç olarak elde edilen şifreli veri, şifresi çözüldüğünde düz metin üzerinde yapılan işlemlerin

sonucuna eşleşir. Bu, bilgi işlem gücü gerektiren kullanıcıların bulut sunucularında veriye erişimini kısıtlayarak veri sızıntısını önler.

Homomorfik şifreleme, gizli anahtara erişim olmadan şifreli veriler üzerinde hesaplama yapmak için ek bir değerlendirme özelliğine sahiptir. Bu tür hesaplamaların sonucu her zaman şifreli olarak kalır. Homomorfik şifreleme, simetrik veya açık anahtar şifrelemesinin bir uzantısı olarak görülebilir. Matematikteki homomorfizm kavramını yansıtarak, şifreleme ve şifre çözme işlevleri, düz metin ve şifreli metin uzayları arasında bir tür yapısal benzerlik oluşturur. Homomorfik, cebirdeki homomorfizmi ifade eder: şifreleme ve şifre çözme fonksiyonları, düz metin ve şifreli metin uzayları arasındaki homomorfizmler olarak düşünülebilir.

3.2. Homomorfik Şifreleme Tarihçesi:

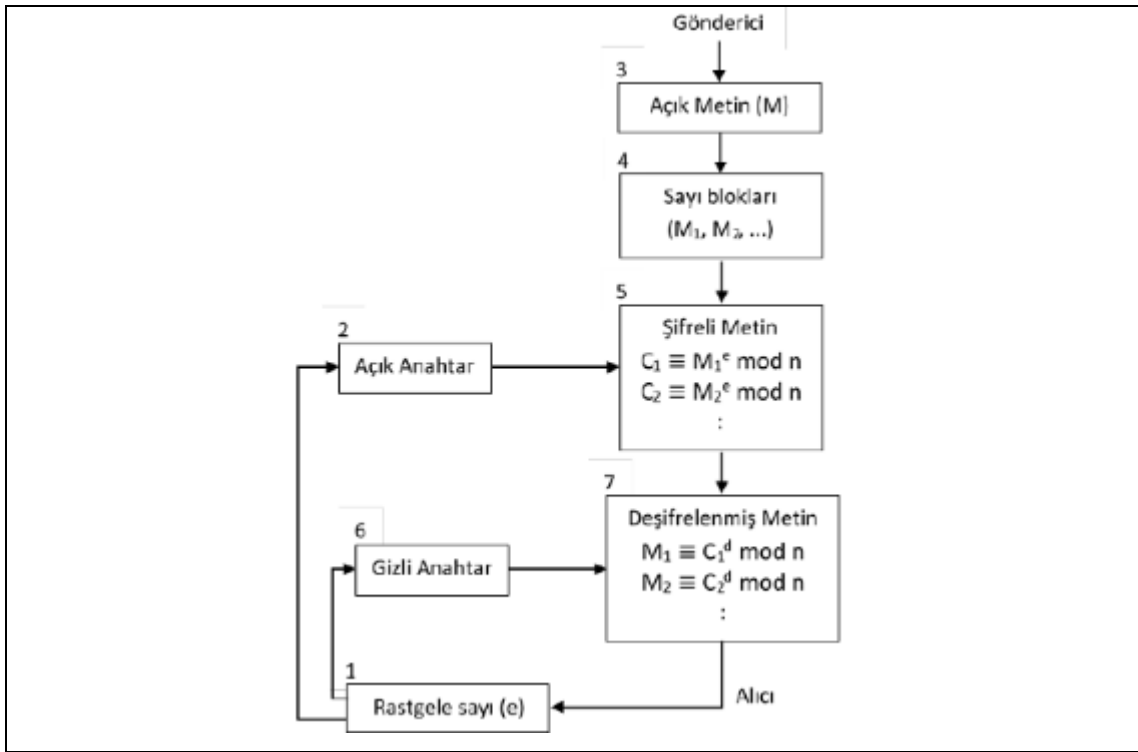
Homomorfik şifrelemenin kökenleri, 1978'de Ron Rivest, Leonard Adleman ve Michael Dertouzos tarafından icat edilen RSA algoritmasına dayanır. RSA, açık anahtarlı şifreleme yöntemlerinin önde gelen bir örneğidir ve bir mesajın şifrenmesi ve şifresinin çözülmesi için iki ayrı anahtar kullanır. Ancak, RSA'nın doğrudan homomorfik özelliklere sahip olmadığı görülmektedir [13].

Homomorfik şifreleme konsepti, 1978'de Rivest, Adleman ve Dertouzos'un çalışmalarından hemen sonra gelişmeye başladı. 2009 yılında, Craig Gentry, homomorfik şifreleme konusunda devrim niteliğinde bir adım attı. Gentry, şifrelenmiş veriler üzerinde işlem yapmayı sağlayan ilk tam homomorfik şifreleme protokolünü geliştirdi. Bu protokol, şifreleme alanında büyük bir ilerleme olarak kabul edildi çünkü daha önce, şifre çözme işlemi yapılmadan şifreli veriler üzerinde işlem yapmak mümkün değildi.

Gentry'nin çalışmaları, homomorfik şifreleme alanında büyük bir araştırma ve geliştirme çabasına yol açtı. Ardından, çeşitli araştırmacılar ve kuruluşlar, homomorfik şifreleme protokollerini daha da geliştirdi ve uygulamalarını genişletti. Bu teknoloji, gizliliği koruyarak bulut bilişim, veri analitiği, sağlık hizmetleri ve finans gibi birçok alanda kullanılmaya başlandı.

3.2.1. RSA Algoritması:

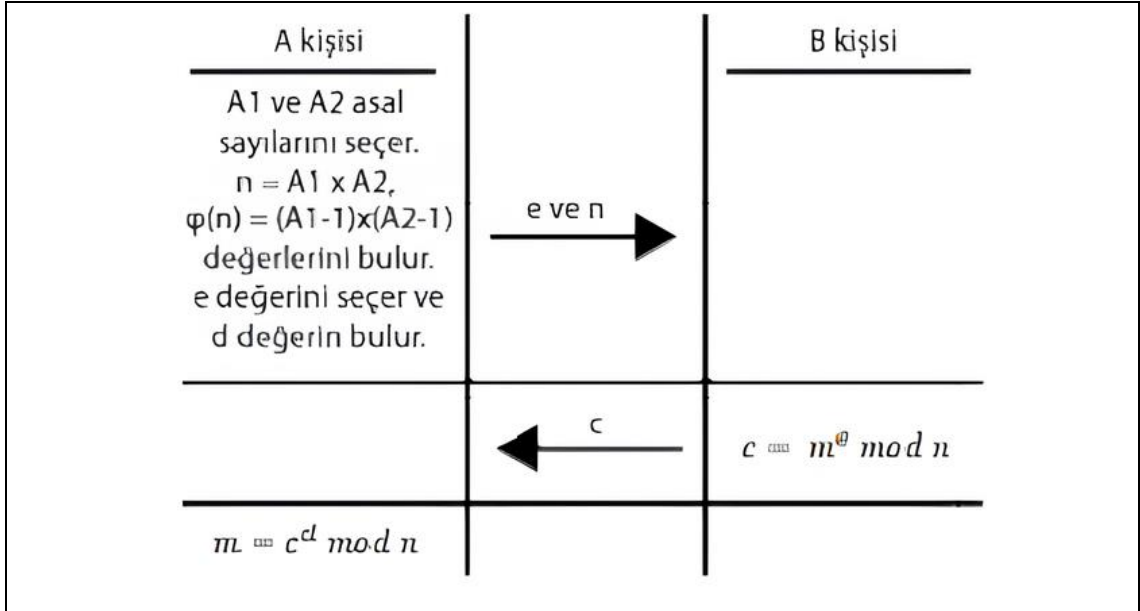
RSA (Rivest-Shamir-Adleman), güvenli veri iletimi için yaygın olarak kullanılan en eski açık anahtarlı şifreleme sistemlerinden biridir. Algoritmanın adı, 1977'de kamuya açıklanan Ron Rivest, Adi Shamir ve Leonard Adleman'ın soyadlarının baş harflerinden gelir. İlgili sistem, aslında İngiliz sinyal istihbarat teşkilatı olan Hükümet İletişim Merkezi'nde (GCHQ) 1973'te gizlice geliştirilmişti ve gizliliği ancak 1997'de kaldırılmıştı, bu gelişme İngiliz matematikçi Clifford Cocks tarafından gerçekleştirilmişti [14].



Şekil-10: RSA Algoritmasının Akış Şeması.

Açık anahtarlı bir şifreleme sisteminde, şifreleme anahtarı geneldir ve gizli (özel) tutulan şifre çözme anahtarından farklıdır. Bir RSA kullanıcısı, iki büyük asal sayıya ve bir yardımcı değere dayalı bir ortak anahtar oluşturur ve bu anahtarı yayınlar. Asal sayılar gizli tutulur. Mesajlar, genel anahtar aracılığıyla herkes tarafından şifrelenebilir, ancak şifresi yalnızca özel anahtarı bilen biri tarafından çözülebilir. RSA'nın güvenliği, iki büyük asal sayının çarpımını çarpanlara ayırmanın pratik zorluğuna, yani "çarpanlara ayırma sorununa" dayanır. RSA şifrelemesinin kırılması RSA sorunu olarak bilinir. Faktoring problemi ne kadar zor olursa olsun, yeterince büyük bir anahtar kullanıldığında

sistemi yenmek için yayınlanmış bir yöntem yoktur. RSA nispeten yavaş bir algoritmadır, bu nedenle kullanıcı verilerini doğrudan şifrelemek için yaygın olarak kullanılmaz. Daha sıklıkla, RSA, simetrik anahtar şifrelemesi için paylaşılan anahtarları iletmek için kullanılır ve bunlar daha sonra toplu şifreleme-şifre çözme için kullanılır.



Şekil-11: RSA Algoritması Çalışma Mantığı.

RSA algoritmasının çalışma mantığına daha derinlemesine bir bakalım:

RSA algoritması, matematiksel işlemlere dayanır ve asal sayılarla ilgilidir. Anahtar oluşturma aşamasında, iki büyük asal sayı seçilir, p ve q. Bu asal sayılar, $N = pq$ olarak adlandırılan bir bileşik sayının parçaları oluşturur. Sonra $\varphi(N) = (p-1)(q-1)$ hesaplanır[15].

Sonra, N'den küçük ve N ile aralarında ortak böleni olmayan bir e sayısı seçilir. Bu e, genellikle 65537 gibi küçük bir asal olmayan sayıdır, çünkü bu, hızlı modüler üs alma işlemleri sağlar ve güvenlik için uygun kabul edilir.

Daha sonra, bir dizi matematiksel işlem kullanılarak özel bir anahtar olan d hesaplanır. Bu, e ve $\varphi(N)$ arasındaki modüler tersidir, yani $(ed \equiv 1 \pmod{\varphi(N)})$.

Kullanıcı, şifrelemek istediği mesajı alır ve bu mesajı genel anahtar olan (N, e) ile şifreler.

Bu işlem, mesajı $m^e \bmod N$ olarak şifreler.

Şifrelenmiş mesajı alıcı alır ve özel anahtar olan d kullanarak mesajı çözer. Bu işlem, şifreli mesajı $c^d \bmod N$ olarak çözer, burada c şifreli mesajı temsil eder. RSA'nın güvenliği, büyük asal sayıları çarpanlarına ayırmak için şu ana kadar bilinen etkili bir yöntem bulunmamasına dayanır. Bu nedenle, RSA, güvenli veri iletimi ve dijital imza gibi alanlarda yaygın olarak kullanılmaktadır.

3.2.2. RSA Algoritmasının Homomorfik Şifrelemeyle İlişkisi:

RSA'nın homomorfik şifreleme ile doğrudan bir bağlantısı bulunmamaktadır. RSA algoritması, temel olarak açık anahtarlı şifreleme için kullanılan bir algoritma olup homomorfik özelliklere sahip değildir. Homomorfik şifreleme ise, şifrelenmiş veriler üzerinde işlem yapmayı mümkün kılan bir kriptografik tekniktir. Bu teknik, verileri şifreledikten sonra dahi matematiksel işlemlerin yapılabilmesine olanak tanır.

Ancak, homomorfik şifreleme alanındaki araştırmalar bazı durumlarda RSA veya diğer açık anahtarlı şifreleme algoritmaları ile ilişkilendirilmiştir. Örneğin, homomorfik şifreleme protokolleri oluşturulurken ve analiz edilirken karmaşık matematiksel teknikler kullanılabilir ve bu teknikler bazen RSA algoritmasının matematiksel özelliklerine dayanabilir. Ancak, bu, homomorfik şifrelemenin doğrudan RSA ile ilişkili olduğu anlamına gelmez. Homomorfik şifreleme ile RSA arasında dolaylı bir bağlantı olabilir ancak homomorfik şifreleme genellikle RSA veya diğer şifreleme algoritmalarının özelliklerinden ziyade tamamen farklı bir kavram olarak ele alınır. RSA algoritması çarpma işlemine dayandığı için, homomorfik şifreleme ile RSA arasındaki ilişki genellikle bu işleme dayanır. Ancak tam homomorfik şifreleme gibi karmaşık şifreleme şemaları genellikle RSA gibi algoritmalarla verimlilik açısından zor veya pratik olmayabilir. Bu nedenle, homomorfik şifreleme genellikle daha karmaşık şifreleme şemaları üzerinde uygulanır.

3.3. Homomorfik Şifreleme Yöntemleri:

3.3.1. Tam Homomorfik Şifreleme (Fully Homomorphic Encryption - FHE):

Tam homomorfik şifreleme (FHE), veriyi şifreli bir şekilde bulanıklaştırma yöntemidir. Bu şifreleme yöntemi, şifreli veri üzerinde matematiksel işlemleri gerçekleştirmenizi

sağlar ve sonuçlarını şifreli bir biçimde elde etmenizi sağlar. Bu, özellikle gizliliğin korunması gereken hassas verilerin işlenmesinde büyük bir potansiyele sahiptir. FHE'nin temel amacı, bulanıklaştırılmış veri üzerinde işlem yapabilme yeteneğiyle geleneksel şifreleme yöntemlerinin sınırlamalarını aşmaktır. Tam homomorfik şifreleme, veri üzerinde toplama, çıkarma, çarpma ve diğer matematiksel işlemleri gerçekleştirebileceğiniz bir sistem sağlar. Bu işlemler şifreli veri üzerinde gerçekleşir ve sonuçlar şifreli olarak elde edilir. Bunun anahtarı, şifreleme sürecinin, şifreli metin üzerinde işlem yapılmasına izin veren bir yapıya sahip olmasıdır. Tam-Homomorfik Şifreleme yapısı temelde halka homomorfizmi olarak düşünülebilir (Yi, Paulet & Bertino, 2014)[16].

X açık metin uzayı, Y şifreli metin uzayı, Encrypt() şifreleme algoritması, Decrypt() şifre çözme algoritması olarak belirlensin. Açık metinler uzayı X'nin bir halka olduğunu (X, \oplus, \otimes) ve şifreli metinler uzayı Y'nin bir halka olduğunu (Y, \oplus, \otimes) varsayalım. Buna göre şifreleme işlemini X halkasından Y halkasına tanımlı fonksiyon olarak düşünebiliriz (Yi vd., 2014)[16].

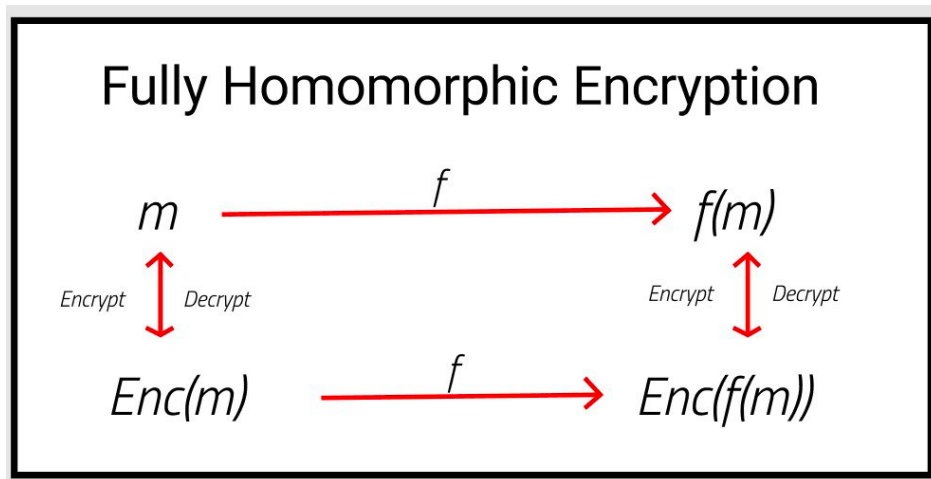
Encrypt() : $X \rightarrow C$

$m_1, m_2 \in X$ olsun;

$$\text{Encrypt}(m_1) \oplus \text{Encrypt}(m_2) = \text{Encrypt}(m_1 \oplus m_2) = \text{Decrypt}(\text{Encrypt}(m_1 \oplus m_2)) \\ = m_1 + m_2$$

$$\text{Encrypt}(m_1) \otimes \text{Encrypt}(m_2) = \text{Encrypt}(m_1 \otimes m_2) = \text{Decrypt}(\text{Encrypt}(m_1 \otimes m_2)) \\ = m_1 \times m_2$$

Bu şekilde Tam-Homomorfik Şifreleme yapısı gösterilebilir.



Şekil-12: Tam Homomorfik Şifreleme Şeması.

3.3.2. Kısmi Homomorfik Şifreleme (Partially Homomorphic Encryption - PHE):

Kısmi homomorfik şifreleme (PHE), veriyi belirli matematiksel işlemlere tabi tutabilen ancak tüm işlemleri desteklemeyen bir şifreleme tekniğidir. PHE, veriyi şifreli bir biçimde işlem yapılabilir hale getirerek gizliliği korurken, belirli hesaplamaların yapılmasına izin verir. Kısmi homomorfik şifreleme, belirli bir matematiksel işlem türü üzerinde homomorfik özellik sunar. Örneğin, bir PHE şeması sadece toplama işlemlerini destekleyebilir veya sadece çarpma işlemlerini destekleyebilir [17]. Şifreli veri üzerinde yalnızca bu belirli işlemleri gerçekleştirmek mümkündür. RSA, ElGamal ve Paillier Kısmi-Homomorfik Şifreleme özelliği göstermektedir.

3.3.2.1. RSA Algoritması ve HE:

RSA algoritması bilinen ilk Homomorfik Şifreleme yöntemlerinden biridir.

Çarpmaya göre Kısmi Homomorfik Şifreleme özelliği gösterir (Parmar vd, 2014)[16]. m_1 1. açık metin, m_2 2. açık metin, n mod değeri, y açık anahtar olsun. RSA algoritmasının çarpmaya göre Kısmi Homomorfik Şifreleme özelliğini aşağıdaki gibi gösterilebilir;

$$m_1^y \bmod n \times m_2^y \bmod n = m_1^y \times m_2^y \bmod n = (m_1 \times m_2)^y \bmod n$$

Şekil-13: RSA Algoritmasının Kısmi Homomorfik Şifreleme İşlemi.

Şekil-13 'de görüldüğü gibi iki açık metnin şifrelenmiş olarak çarpımı açık metin olarak çarpımının şifrelenmiş haline eşit olur.

3.3.2.2. ElGamal Algoritması ve HE:

ElGamal şifreleme algoritması, açık anahtarlı bir şifreleme yöntemidir ve homomorfik şifreleme için de temel oluşturabilir. İlk olarak Diffie-Hellman protokolünün genelleştirilmiş bir versiyonu olarak geliştirilmiştir ve özellikle dijital imza ve gizli anahtarlı şifreleme alanlarında kullanılır. ElGamal algoritması toplamaya göre Kısmi Homomorfik Şifreleme özelliği göstermektedir.

m_1 1. açık metin, m_2 2. açık metin, y mod değeri, K ortak anahtar olsun. ElGamal algoritmasının toplamaya göre Kısmi Homomorfik Şifreleme özelliğini aşağıdaki gibi gösterilebilir;

$$(K \times m_1) \bmod y + (K \times m_2) \bmod y = (K \times m_1) + (K \times m_2) \bmod y = K(m_1 + m_2) \bmod y$$

Şekil-14: ElGamal Algoritmasının Toplamaya Göre Kısmi Homomorfik Şifrelemesi.

Şekil-14 'de görüldüğü gibi iki açık metnin şifrelenmiş olarak toplamı açık metin olarak toplamının şifrelenmiş haline eşittir.

3.3.2.3. Paillier Algoritması ve HE:

Paillier şifreleme algoritması, homomorfik şifreleme alanında önemli bir rol oynayan bir kriptografik protokoldür. Bu algoritma, ElGamal gibi açık anahtarlı bir şifreleme algoritmasıdır. Paillier şifreleme algoritması, Pascal Paillier tarafından 1999 yılında tanıtılmıştır. İki ana matematiksel problem olan RSA ve Diffie-Hellman problemine dayanır.

Paillier algoritması toplamaya göre Homomorfik Şifreleme özelliği göstermektedir (Parmar vd, 2014)[16].

Şifrelenmiş metinlerin toplamı, açık metinlerin toplamına eşittir. m_1 1. açık metin, m_2 2. açık metin, n mod değeri, g açık anahtar, r seçilen rastgele değer olsun. Paillier algoritmasının toplamaya göre Kısmi Homomorfik Şifreleme özelliğini aşağıdaki gibi gösterilebilir;

$$(g^{m_1} \times r^n) \bmod n^2 \times (g^{m_2} \times r^n) \bmod n^2 = (g^{m_1} \times r^n) \times (g^{m_2} \times r^n) \bmod n^2 = r^{2n} g^{m_1+m_2} \bmod n^2$$

Şekil-15: Paillier Algoritmasının Toplamaya Göre Kısmi Homomorfik Şifrelemesi.

Şekil-15'de görüldüğü gibi iki açık metnin şifrelenmiş olarak toplamı açık metin olarak toplamının şifrelenmiş haline eşittir.

3.3.3. Somut Homomorfik Şifreleme (Concretely Homomorphic Encryption):

Somut Homomorfik Şifreleme (Concretely Homomorphic Encryption), genellikle özel bir işlemi gerçekleştirmek için tasarlanmış homomorfik şifreleme şemalarını ifade eder. Bu şifreleme türü, belirli bir işlemi gerçekleştirmek için optimize edilmiş bir homomorfik şifreleme şeması içerir. Örneğin, somut homomorfik şifreleme, belirli bir işlem için mükemmel uyum sağlayacak şekilde tasarlanabilir. Bu, çarpma işlemi veya toplama işlemi gibi belirli bir matematiksel operasyonu gerçekleştirmek için optimize edilmiş bir homomorfik şifreleme şeması geliştirmek anlamına gelir.

Somut homomorfik şifreleme, belirli bir işlem için yüksek verimlilik ve düşük hesaplama maliyeti sağlayabilir. Özellikle belirli bir uygulama veya kullanım senaryosu için optimize edilmiş homomorfik şifreleme şemaları geliştirmek, veri gizliliğini korumak ve gizlilik korumalı hesaplamaları daha etkili hale getirmek için önemli bir adımdır. Ancak, somut homomorfik şifreleme genellikle genel amaçlı homomorfik şifreleme tekniklerine göre daha sınırlıdır. Belirli bir işlem için optimize edilmiş olmaları nedeniyle, genel homomorfik şifreleme şemaları kadar geniş bir işlem yelpazesini desteklemeyebilirler. Bununla birlikte, belirli bir işlem için optimize edilmiş olmaları, performansı artırabilir ve pratik uygulamalarda daha etkili olmalarını sağlayabilir.

3.4. Uygulama Alanları:

3.4.1. Bulut Bilişim Güvenliği ve HE:

Bulut bilişim, birçok kuruluşun veri depolama ve işleme ihtiyaçlarını karşılamak için kullandığı bir hizmet modelidir. Ancak, bulut bilişimdeki ana endişelerden biri, bulut sağlayıcısının sunucularında barındırılan verilerin güvenliği ve gizliliğidir. Homomorfik şifreleme, bu endişeleri hafifletmek için bir çözüm sunar. Verilerin fiziksel sunucularda depolanırken dışarıdan erişilemeyecek şekilde korunmasını sağlar. Dolayısıyla, verilere yetkisiz erişim riski azalır. Bulut sağlayıcıları, homomorfik olarak şifrelenmiş veriler üzerinde işlem yapabilir. Bu, müşterilerin verilerini güvenli bir şekilde sunucuya yükleyip analiz yapmalarını veya işlem yaptırmalarını sağlar. Örneğin, bir müşteri, bulut sağlayıcısına homomorfik olarak şifrelenmiş bir veri kümesi gönderebilir ve bu veriler üzerinde istenilen hesaplamaları gerçekleştirebilir, sonuçları alabilir [18].

3.4.2. Ulusal Güvenlik ve HE:

Devlete ait binalar, şebekeler, jeneratörler ve benzeri yapılar arasında entegre bir şekilde çalışan zeki bir ağ oluşturulduğunda, bu sistem belirli veriler üretecektir. Bu veriler, çeşitli devlet organlarıyla paylaşılacak ve gerekli analizlerin ve değerlendirmelerin yapılması gerekecektir. Bu işlemlerin güvenli bir şekilde gerçekleştirilmesi için homomorfik şifreleme yapısı kullanılarak gerekli hesaplamalar ve incelemeler yapılabilir. Bu tür bilgilerle, kötü niyetli yazılımların, anormalliklerin ve izinsiz girişlerin tespit edilerek güvenlik sağlanabilir.

Böylesine kapsamlı bir yapı, çok geniş bir alandan çok miktarda bilgi içerdiği için, bu bilgilerin korunması ve kötü niyetli kişilerin erişimine engel olunması son derece önemlidir. Örneğin, şebeke ve güç dağıtım bilgilerinin kötü niyetli kişiler tarafından ele geçirilmesi durumunda, bu alanlara saldırılar gerçekleştirilebilir.

3.4.3. Veri Madenciliği ve HE:

Veri Madenciliği, büyük veri kümeleri üzerinde desenler, ilişkiler ve trendler bulmayı amaçlayan bir disiplindir. Ancak, bu tür analizler sıklıkla kişisel gizlilik endişelerine neden olabilir çünkü hassas verilerin işlenmesini gerektirebilir. Homomorfik şifreleme, bu endişeleri gidermek için bir çözüm sunar [21].

Veri Şifreleme: İlk adım, hassas verilerin homomorfik olarak şifrlenmesidir. Bu şifreleme süreci, verilerin matematiksel işlemlerle şifrlenmesini sağlar, ancak şifreleme sonucu, orijinal verilerden türetilen herhangi bir bilgiyi ifşa etmez.

Şifrelenmiş Veri Analizi: Homomorfik olarak şifrelenmiş veriler üzerinde analiz yapmak, veri madenciliğini mümkün kılar. Örneğin, bir model oluşturmak için kullanılan makine öğrenimi algoritmaları, şifrelenmiş veriler üzerinde çalışabilir.

Sonuçların Şifrlenmesi: Analiz sonuçları, homomorfik olarak şifrelenmiş olarak elde edilir. Bu sonuçlar, orijinal verilerin gizliliğini korurken, analiz sonuçlarının faydalı bilgilere dönüştürülmesini sağlar.

Sahip Olunan Bilginin Korunması: Veri madenciliği sırasında elde edilen bilgiler, genellikle değerlidir ve ticari sırlar veya kişisel bilgiler içerebilir. Homomorfik şifreleme,

bu bilgilerin gizliliğini koruyarak, yetkisiz erişimden korur.

Güvenli Veri Paylaşımı: Şifrelenmiş verilerin analizi, farklı kuruluşlar veya işbirliği yapan taraflar arasında güvenli bir şekilde paylaşılabilir. Herhangi bir aşamada, verilerin şifrelenmiş olması, veri güvenliğini sağlar. Homomorfik şifreleme, veri madenciliği alanında büyük bir potansiyele sahiptir çünkü gizlilik endişelerini hafifletirken, verilerin analiz edilmesini mümkün kılar. Bu, hassas verilerle çalışan kuruluşlar için önemli bir avantaj sağlar ve veri madenciliğinin daha geniş bir şekilde kullanılmasını teşvik eder.

4. HOMOMORFİK ŞİFRELEME WİKİ SAYFASI

Homomorfik şifreleme, tarihçesi, kullanım alanları, algoritmaları, programlama dillerindeki kullanımı ve günlük hayattaki uygulamalarını anlatan bir wiki web sayfası hazırlandı.

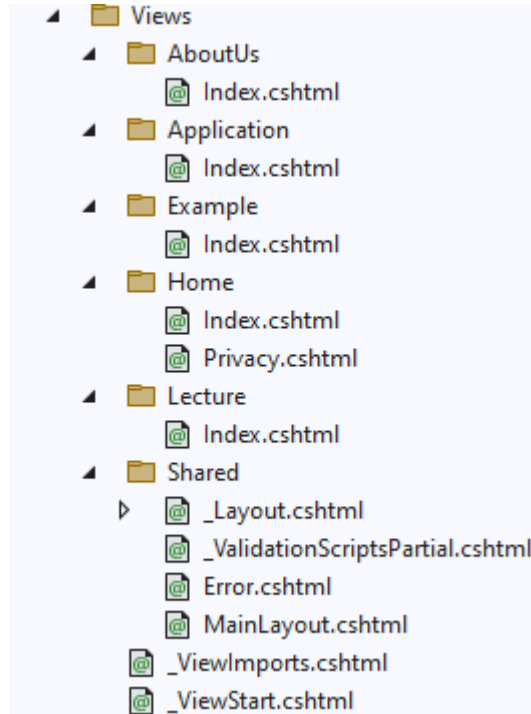
4.1. Tasarım Süreci:

Web sayfasının oluşturulması sürecinde, MVC (Model-View-Controller) katmanlı mimari kullanıldı. Bu mimari, uygulamanın farklı katmanlara ayrılmasını ve işlerin daha organize bir şekilde yapılmasını sağlar. “Views” çatısı altında Razor View Engine ile “.cshtml” sayfaları oluşturuldu. HTML, CSS ve JavaScript kullanarak sayfaların tasarımı yapıldı ve estetik açıdan hoş ve kullanıcı dostu görünüm sağlandı. Sayfaların tasarımında minimalizm ön plandaydı. Minimalist bir tasarım, kullanıcıların sayfa içinde kaybolmadan istedikleri bilgilere ulaşmasını sağlar. İçerik bölümü, sayfaların sol ve orta kısımlarına yerleştirildi ve sağ kısım menü ve temel konuların başlıkları ile dolduruldu. Bu düzenleme, kullanıcıların bilgilere kolayca erişmesini sağlar.

Genel tasarım şablonu (MainLayout.cshtml) bu özellikler doğrultusunda hazırlandıktan sonra, ana şablon diğer tüm sayfalara uygulandı. Bu, her sayfanın ayrı bir tasarıma ihtiyaç duymadan, tek bir şablona dayanarak oluşturulmasını sağlar. Ayrıca, “Views”, “Shared” klasörü altında bulunan bu dosya ile kod tekrarından kaçınıldı ve uygulamanın bakımı kolaylaştırıldı.



Şekil-16: Web Sayfasının Genel Tasarımı.



Şekil-17: Proje'nin Views Hiyerarşisi.

4.2. Veritabanı İşlemleri:

4.2.1. Entity Framework Kurulumu:

İlk adım olarak, projeye Entity Framework'ü yüklendi. Bunun için NuGet paket yöneticisi veya komut satırı kullanılabilir. Komut satırı için “Install-Package EntityFramework” çalıştırılmalıdır.

4.2.2. DbContext Sınıfı Oluşturma:

Veritabanı işlemlerini gerçekleştirmek için bir DbContext sınıfı oluşturulur. Bu sınıf, Entity Framework tarafından sağlanan DbContext sınıfından türetilir. Bu sınıf, veritabanı bağlantısı ve tablo bağlantıları gibi işlemleri yönetir.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseSqlServer
        ("server=(localdb)\\MSSQLLocalDB;database=CoreTezDb; integrated security=true;");
}
```

Şekil-18:Veritabanı Bağlantısı.

Şekil-18’deki kod parçası, Entity Framework ile çalışırken DbContext sınıfını yapılandırmak için kullanılır. OnConfiguring metodu, DbContext sınıfının bir üyesidir ve DbContext sınıfı oluşturulduğunda otomatik olarak çağrılır.

Bu metod, DbContextOptionsBuilder nesnesini alır ve bu nesne aracılığıyla DbContext sınıfının yapılandırılması sağlanır. Yapılandırma genellikle veritabanı bağlantısının belirlenmesi ile ilgilidir.

Örneğin, bu kod parçasında, optionsBuilder üzerinden UseSqlServer metodu çağrılarak SQL Server veritabanına bağlanmak için gerekli olan bağlantı dizesi belirtilmiştir. Bağlantı dizesi aşağıdaki gibidir:

```
"server=(localdb)\\MSSQLLocalDB;database=CoreTezDb; integrated security=true;"
```

Bu dize, server parametresi ile yerel bir SQL Server veritabanına (localdb) bağlanıldığını, database parametresi ile CoreTezDb adında bir veritabanını hedeflendiğini ve integrated

security=true parametresi ile Windows kimlik doğrulamasının (Windows Authentication) kullanıldığını belirtir. Bu şekilde yapılandırma yapılarak, DbContext sınıfı, belirtilen veritabanına bağlanacak ve bu veritabanı üzerinde işlemler yapabilecektir.

```
0 references
public DbSet<Category> Categories { get; set; }
0 references
public DbSet<HomePage> Paragraphs { get; set; }
0 references
public DbSet<AboutUs> AboutUs { get; set; }
0 references
public DbSet<Lecture> Lectures { get; set; }
```

Şekil-19: DbSet Tanımlamaları.

Bu, C# programlama dilinde Entity Framework Core kullanarak bir veritabanı modelini tanımlayan bir sınıf içinde bulunan DbSet'lerdir. DbSet, bir veritabanı tablosunu temsil eder. Yani bu DbSet'ler, belirli bir veritabanı tablosundaki verileri sorgulamak, eklemek, güncellemek ve silmek için kullanılır.

Örneğin, "Lectures" DbSet'i, "Lecture" sınıfının nesnelerini içeren bir veritabanı tablosunu temsil eder. "Paragraphs" DbSet'i, "HomePage" sınıfının nesnelerini içeren bir tabloyu temsil eder ve diğer DbSet'ler de benzer şekilde ilgili sınıfların nesnelerini içeren tabloları temsil eder.

Bu kod parçası, bir uygulamanın veritabanında kategoriler, paragraflar, hakkımızda bilgisi ve dersler gibi öğeleri depolamak için kullanılan veritabanı bağlamını tanımlar.

4.2.3. Entity Sınıflarının Oluşturulması:

Veritabanı tablolarını temsil eden entity (varlık) sınıfları oluşturulur. Bu sınıflar genellikle veritabanı tablosundaki sütunları ve ilişkileri temsil eder. Entity sınıfları, bir ORM (Object-Relational Mapping) aracı kullanılarak ilişkisel bir veritabanı ile ilişkilendirilmiş bir uygulamada, veritabanı tablolarını temsil eden sınıflardır. Bu sınıflar, veritabanındaki her bir kaydı (satırı) bir nesneye dönüştürür ve bu nesneler üzerinde işlem yapmayı sağlar.

```

54 references
public class HomePage
{
    [Key]
    72 references
    public int ParagraphID { get; set; }
    36 references
    public string? ParagraphTitle { get; set; }
    72 references
    public string? ParagraphContent { get; set; }
    36 references
    public bool ParagraphStatus { get; set; }

    36 references
    public int CategoryID { get; set; }
    0 references
    public Category Category { get; set; }
}

```

Şekil-20: Ana Sayfa Entity Sınıfı.

Şekil-20'deki C# kodu, EntityLayer.Concrete adlı bir isim alanında bulunan HomePage adlı bir sınıfı tanımlar. Bu sınıfın, bir veritabanı tablosunu temsil eden bir Entity sınıfıdır.

ParagraphID: Bu özellik, paragrafın benzersiz bir kimliğini (ID) temsil eder. [Key] niteliğiyle işaretlenmiştir, bu da bu özelliğin birincil anahtar (primary key) alanı olduğunu belirtir.

ParagraphTitle: Paragrafın başlığını temsil eden bir dize (string) özelliğidir. Başlık isteğe bağlı (? işaretiyle belirtilmiştir), yani null olabilir.

ParagraphContent: Paragrafın içeriğini temsil eden bir dize (string) özelliğidir. İçerik de isteğe bağlı (? işaretiyle belirtilmiştir), yani null olabilir.

ParagraphStatus: Paragrafın durumunu (aktif veya pasif) belirten bir boolean özelliğidir. Boolean özelliği ile içeriğin sayfada gösterilip gösterilmeyeceği ayarlanabilir.

CategoryID: Bu özellik, paragrafın hangi kategoriye ait olduğunu belirten bir tam sayıdır. Bu alan, Category sınıfındaki bir kategoriye yönelik bir dış anahtar (foreign key) ilişkisini temsil eder.

Category: Bu özellik, paragrafın ait olduğu kategoriye belirten bir Category nesnesidir. Bu ilişki, paragrafın bir kategoriye ait olduğunu gösterir ve Entity Framework tarafından ilişkisel veritabanı sorguları oluşturulurken kullanılır.

Bu sınıf, bir veritabanı tablosunu temsil ederken, ilişkili bir kategoriye ve bu kategoriye ait paragraflara erişmek için kullanılır.

```
16 references
public class AboutUs
{
    12 references
    public int AboutUsId { get; set; }
    3 references
    public string? AboutUsContent { get; set; }
    3 references
    public string? AboutUsDanisman { get; set; }
    4 references
    public string? AboutUsUniversite { get; set; }
    4 references
    public string? AboutUsFakulteBolum { get; set; }
    4 references
    public string? AboutUsIsim { get; set; }
    4 references
    public string? AboutUsNumara { get; set; }
}
```

Şekil-21: Hakkımızda Sayfasının Entity Sınıfı.

4.2.4. Controller Yapısı:

```
public class LectureController : Controller
{
    LectureManager lectureManager = new LectureManager(new EfLectureRepoDAL());

    0 references
    public IActionResult Index()
    {
        var values = lectureManager.GetAllCategories();
        return View(values);
    }
}
```

Şekil-22: Controller Yapısı.

Şekil-22'deki C# kodu, bir ASP.NET Core MVC uygulamasında LectureController isimli controller sınıfını temsil ediyor. LectureController sınıfı, Controller sınıfından türetilmiş bir sınıftır. Bu, ASP.NET Core uygulamalarında kullanılan bir controller sınıfının

temelini oluşturur. `LectureManager` adında bir `LectureManager` nesnesi oluşturulur ve `EfLectureRepoDAL` sınıfından türetilen bir nesne kullanılarak başlatılır. Bu, `LectureManager` sınıfının veri erişim katmanı üzerindeki operasyonları yönetmesine olanak tanır.

`Index` metodu, uygulamanın ana sayfasını temsil eder `lectureManager.GetAllCategories()` çağrısıyla tüm kategorileri alır ve bu kategorileri bir görünüme (view) ile döndürür. Bu kod parçası, bir Homomorfik Şifreleme Wiki uygulamasında, konu anlatımı sayfası ilgili işlemleri yönetmek için kullanılan bir controller'ı temsil ediyor.

4.2.5. Migration İşlemleri:

Migration, Entity Framework gibi ORM (Object-Relational Mapping) araçlarında kullanılan bir terimdir. Bir migration, veritabanı şemasındaki değişiklikleri tanımlayan ve uygulayan bir işlemdir.

Genellikle bir projede veritabanı şemasındaki değişiklikler, yeni bir sınıf eklendiğinde, bir sınıfın özellikleri değiştirildiğinde veya bir ilişki tanımlandığında gerçekleşir. Bu değişiklikler, veritabanı şemasında tabloların oluşturulması, değiştirilmesi, silinmesi veya ilişkilendirilmesi şeklinde olabilir.

Migration'lar, veritabanı şemasındaki bu değişiklikleri kod olarak tanımlamak için kullanılır. Bu tanımlamalar, projenin kaynak kodunda bulunur ve Entity Framework gibi ORM araçları tarafından desteklenen komutlarla oluşturulur ve uygulanır.

Migration'ları etkinleştirmek için package manager console'a: "Enable-Migrations" komutu girilmelidir.

Migration'ı oluşturmak için: "Add-Migration InitialCreate" komutu kullanılır.

Migration'ları veritabanına uygulamak için: "Update-Database" komutu kullanılır ve migration uygulanır, veritabanı oluşturulur. Proje çalıştırılmadan önce "Update-Database" komutu ile migration kayıtları güncellenmelidir.

4.2.6. CRUD İşlemleri:

```
5 references
public class GenericRepository<T> : IGenericDal<T>
{
    where T : class

    5 references
    public void Delete(T t)
    {
        using var c = new Context();
        c.Remove(t);
        c.SaveChanges();
    }

    1 reference
    public T GetById(int id)
    {
        using var c = new Context();
        return c.Set<T>().Find(id);
    }

    5 references
    public List<T> GetListAll()
    {
        using var c = new Context();
        return c.Set<T>().ToList();
    }

    5 references
    public void Inset(T t)
    {
        using var c = new Context();
        c.Add(t);
        c.SaveChanges();
    }

    5 references
    public void Update(T t)
    {
        using var c = new Context();
        c.Update(t);
        c.SaveChanges();
    }
}
```

Şekil-23: CRUD işlemlerini (Create, Read, Update, Delete).

Constructor (Yapıcı Metot): Sınıfın kendisi genel bir sınıf (Generic) olduğu için, bu sınıfın herhangi bir türü için kullanılabilir. T parametresi, bu türü belirtir.

Delete (Sil): Belirli bir varlık (T) silmek için kullanılır. Önce bir Context oluşturulur, bu Context üzerinden varlık silinir ve değişiklikler kaydedilir.

GetById (Id'ye Göre Getir): Belirli bir id'ye sahip varlığı (T) getirir. Yine bir Context oluşturulur ve Find metoduyla varlık bulunur.

GetListAll (Tüm Listeyi Getir): Tüm varlıkları (T) getirir. Set<T>() metoduyla tüm varlıkların kümesi alınır ve ToList() metoduyla bu küme bir liste haline getirilir.

Insert (Ekle): Yeni bir varlık (T) eklemek için kullanılır. Yine bir Context oluşturulur, varlık eklenir ve değişiklikler kaydedilir.

Update (Güncelle): Var olan bir varlığı (T) güncellemek için kullanılır. Bir Context oluşturulur, varlık güncellenir ve değişiklikler kaydedilir.

Bu yapı, veritabanı işlemlerini merkezi bir yerde yönetmek için oldukça kullanışlıdır ve kod tekrarını önler.

4.2.7. Verilerin Veritabanından Sayfalara Çekilmesi:

```
<h2>Açıklama</h2>
<p>
    @foreach (var item in Model)
    {
        if (@item.ParagraphID == 105)
        {
            @item.ParagraphContent
        }
    }
</p>
```

Şekil-24: Verilerin Veritabanından Çekilmesi.

Şekil-23'deki kod, bir modelden (bir koleksiyon) gelen verileri döngü içinde işler ve belirli bir koşula göre bu verileri görüntüler.

Model, görünüme iletilen bir modeldir (bir koleksiyon). Her bir öge (item), Model içindeki öğeleri temsil eder. Döngü içinde her bir öge için ParagraphID özelliği kontrol edilir. Eğer ögenin ParagraphID özelliği 105'e eşitse, o ögenin ParagraphContent özelliği görüntülenir. Yani, bu kod, Model içindeki her bir ögenin ParagraphID özelliğini kontrol eder ve eğer bu özellik 105'e eşitse, o ögenin ParagraphContent özelliğini görüntüler. Bu HTML içinde bir paragraf (<p>) etiketi içinde yapılır.

4.2.8. Veritabanı.

	ParagraphID	ParagraphTitle	ParagraphContent	ParagraphStatus	CategoryID
	100	Temelde üç tür h...	Toplama homomor...	True	100
	101	1. Toplama Hom...	Bu tür homomorfiz...	True	101
	102	2. Çıkarma Hom...	PHE, şifreli metin ü...	True	102
	103	3. Çarpma Homo...	Somewhat homom...	True	103
	104	Homomorfik Wiki	Homomorfik şifrel...	True	104
	105	Homomorfik Wi...	Homomorfik şifrel...	True	105
	106	Homomorfik Wi...	Özellikle sağlık sekt...	True	106
	107	Homomorfik Wi...	Homomorfik şifrel...	True	107
	108	Homomorfik Wi...	Kısmen homomorf...	True	108
	109	Homomorfik Wi...	Bir dereceye kadar ...	True	109
	110	Homomorfik Wi...	Tamamen homom...	True	110
	111	Homomorfik Wi...	Homomorfik şifrel...	True	111
	112	Homomorfik Wi...	"Homomorfik şifrel...	True	112
	113	Tarihçe	Homomorfik şifrel...	True	113
	114	Tarihçe	Gentry'nin çalışma...	True	114
	115	FHE, Tam Homo...	Tamamen homom...	True	115

	Column Name	Data Type	Allow Nulls
▶	ParagraphID	int	<input type="checkbox"/>
	ParagraphTitle	nvarchar(MAX)	<input checked="" type="checkbox"/>
	ParagraphContent	nvarchar(MAX)	<input checked="" type="checkbox"/>
	ParagraphStatus	bit	<input type="checkbox"/>
	CategoryID	int	<input type="checkbox"/>
			<input type="checkbox"/>

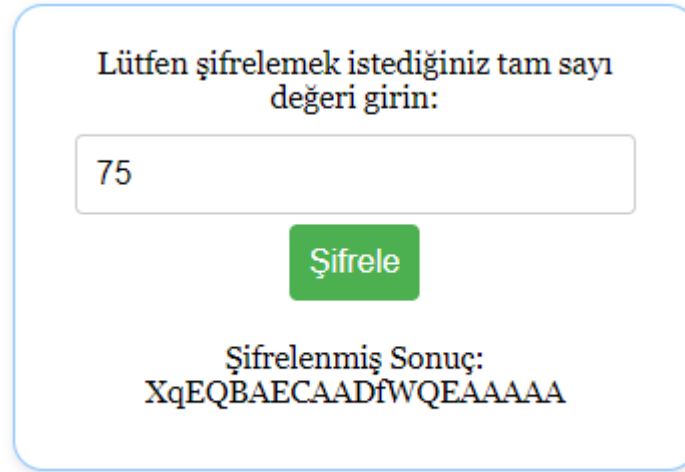
Şekil-25: Veritabanı Görsel 1.

AboutUsId	AboutUsContent	AboutUsDanisman	AboutUsFakult...	AboutUsIsim	AboutUsNumara	AboutUsUniversite
1	Homomorfik Şif...	Dr. Öğr. Üyesi Ah...	Teknoloji Fakülte...	Ali Berdan K...	182120015	Düzce Üniversitesi
2	Homomorfik Şif...	Dr. Öğr. Üyesi Ah...	Teknoloji Fakülte...	Yahya Arı	182120004	Düzce Üniversitesi

	Column Name	Data Type	Allow Nulls
▶	AboutUsId	int	<input type="checkbox"/>
	AboutUsContent	nvarchar(MAX)	<input checked="" type="checkbox"/>
	AboutUsDanisman	nvarchar(MAX)	<input checked="" type="checkbox"/>
	AboutUsFakulteBolum	nvarchar(MAX)	<input checked="" type="checkbox"/>
	AboutUsIsim	nvarchar(MAX)	<input checked="" type="checkbox"/>
	AboutUsNumara	nvarchar(MAX)	<input checked="" type="checkbox"/>
	AboutUsUniversite	nvarchar(MAX)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Şekil-26: Veritabanı Görsel 2.

4.3. Homomorfik Şifreleme Uygulaması:



Lütfen şifrelemek istediğiniz tam sayı değeri girin:

Şifrele

Şifrelenmiş Sonuç:
XqEQBAECAADfWQEAAAAA

Şekil-27: Homomorfik Şifreleme Uygulaması.

```
private SEALContext context;  
private KeyGenerator keygen;  
private Encryptor encryptor;  
private Decryptor decryptor;  
0 references
```

Şekil-28: SEAL Kütüphanesi Değişken Tanımlamaları.

SEALContext, KeyGenerator, Encryptor ve Decryptor türünden özel alanlar sınıfın içinde tanımlanmıştır. Bu alanlar, Microsoft Research SEAL kütüphanesinden gelen nesneleri yönetmek için kullanılır.

```
1 reference  
private Ciphertext HomomorphicEncryption(int value, Encryptor encryptor)  
{  
    // İstenen veriyi şifrele  
    Plaintext plain = new Plaintext(value.ToString());  
    Ciphertext encrypted = new Ciphertext();  
    encryptor.Encrypt(plain, encrypted);  
  
    // Şifreli veriyi geri döndür  
    return encrypted;  
}
```

Şekil-29: Şifreleme Metodu.

Şekil-29'daki HomomorphicEncryption fonksiyonu, bir tam sayı değerini (int değeri) şifrelemek için kullanılır. İşlevsel olarak adımları şu şekildedir: Gelen tam sayı değerini bir Plaintext nesnesine dönüştürür. Plaintext, SEAL kütüphanesinde şifrelenmemiş metin veriyi temsil eder. Bir Ciphertext nesnesi oluşturur. Ciphertext, SEAL kütüphanesinde şifrelenmiş veriyi temsil eder. encryptor.Encrypt metodu kullanılarak, oluşturulan Plaintext nesnesini şifreler ve sonucu Ciphertext nesnesine kaydeder. Şifrelenmiş veriyi içeren Ciphertext nesnesini geri döndürür. Bu fonksiyon, belirtilen bir tam sayı değerini şifrelemek için encryptor nesnesini kullanır. Şifrelenmiş sonuç, bir Ciphertext nesnesi olarak döndürülür ve daha sonra işlem yapılabilir veya depolanabilir.

```
[HttpPost]
0 references
public IActionResult Encrypt(int myTextbox)
{
    // Parametrelerin ayarlanması
    EncryptionParameters parms = new EncryptionParameters(SchemeType.BFV);
    parms.PolyModulusDegree = 4096;
    parms.CoeffModulus = CoeffModulus.BFVDefault(parms.PolyModulusDegree);
    parms.PlainModulus = new Modulus(256);

    // SEALContext oluştur
    SEALContext context = new SEALContext(parms, true, SecLevelType.None);

    // Anahtar oluşturma
    KeyGenerator keygen = new KeyGenerator(context);
    PublicKey publicKey;
    keygen.CreatePublicKey(out publicKey); // Genel anahtar oluşturuluyor
    SecretKey secretKey = keygen.SecretKey;

    // Şifreleme ve deşifreleme nesnelerini oluşturma
    Encryptor encryptor = new Encryptor(context, publicKey);
    Decryptor decryptor = new Decryptor(context, secretKey);

    // Kullanıcının girdiği değeri homomorfik olarak şifrele
    Ciphertext encryptedResult = HomomorphicEncryption(myTextbox, encryptor);

    // Şifrelenmiş sonucu ViewBag üzerinden Index view'ına aktar
    ViewBag.EncryptedResult = encryptedResult;

    return View("Index");
}
```

Şekil-30: Şifreleme İşlemi.

Şekil-30'daki kod bir HTTP POST isteğini işleyen bir metottur. İsteğin işlenmesi sırasında aşağıdaki adımlar gerçekleştirilir:

myTextbox isimli bir integer parametre alır. Bu muhtemelen bir web formunda bir metin kutusuna girilen değeri temsil eder.

Şifreleme için gerekli olan parametrelerin ayarlandığı EncryptionParameters nesnesi oluşturulur. Bu parametreler, şifreleme şemasını (SchemeType), polinom modulus derecesini (PolyModulusDegree), katsayı moduluslerini (CoeffModulus) ve düz modulusü (PlainModulus) içerir.

Bir SEALContext nesnesi oluşturulur. Bu, SEAL kütüphanesindeki işlemler için bir bağlam sağlar. Önceden belirlenmiş parametrelerle birlikte kullanılır.

Bir anahtar üretici (KeyGenerator) oluşturulur ve genel (public) anahtar üretilir. Bu genel anahtar, veriyi şifrelemek için kullanılacaktır.

Encryptor ve Decryptor nesneleri oluşturulur. Bunlar, sırasıyla veriyi şifrelemek ve şifreli veriyi deşifrelemek için kullanılır.

HomomorphicEncryption fonksiyonu aracılığıyla kullanıcının girdiği değer homomorfik olarak şifrelenir.

Sonuç, ViewBag üzerinden bir görünüme (View) aktarılır ve "Index" görünümüne yönlendirilir.

Bu metod, kullanıcının girdiği değeri homomorfik olarak şifreler ve şifrelenmiş sonucu bir ViewBag ile birlikte bir görünüme gönderir.

```
private string ShortenEncodedCiphertext(Ciphertext ciphertext, int maxLength = 20)
{
    using (System.IO.MemoryStream stream = new System.IO.MemoryStream())
    {
        ciphertext.Save(stream);
        string base64String = Convert.ToBase64String(stream.ToArray());
        return base64String.Length <=
            maxLength ? base64String : base64String.Substring(0, maxLength);
    }
}
```

Şekil-31: Base64 Formatında Kodlama.

Bu ShortenEncodedCiphertext fonksiyonu, bir Ciphertext nesnesini Base64 formatında kodlayan ve ardından belirli bir maksimum uzunluğa kadar kısaltan bir metodu temsil ediyor. İşlevsellik adım adım şu şekildedir:

Bir MemoryStream oluşturulur. Bu, bellekte bir akış oluşturmak için kullanılır. Ciphertext nesnesi, Save metodu kullanılarak bu bellek akışına kaydedilir. Bu, Ciphertext nesnesinin içeriğini bellek akışına yazmayı sağlar. Bellek akışındaki veri, ToArray metodu kullanılarak bir bayt dizisine dönüştürülür. Bu bayt dizisi, Convert.ToBase64String metodu kullanılarak Base64 formatına dönüştürülür ve bir dizeye atanır. Oluşan Base64 dizesinin uzunluğu, belirtilen maksimum uzunluğu aşarsa, dize belirtilen maksimum uzunlukta kısaltılır. Aksi halde, dize doğrudan geri döndürülür. Bu fonksiyon, bir Ciphertext nesnesini Base64 formatında kodlayarak ve ardından belirtilen maksimum uzunluğa kadar kısaltarak, şifrelenmiş veriyi belirli bir uzunlukta temsil eder. Bu, şifrelenmiş veriyi kısaltılmış bir formatta saklamak veya iletmek gibi senaryolarda kullanışlı olabilir. Örneğin, kullanıcıya kısaltılmış bir şekilde sunulabilir ve daha fazla ayrıntıya ihtiyaç duyulduğunda tamamen geri yüklenebilir.

```
<form action="@Url.Action("Encrypt", "Application")" method="post">
  <label for="myTextbox">Lütfen şifrelemek istediğiniz tam sayı değeri girin:
</label>
  <input type="number" id="myTextbox" name="myTextbox" step="1" required>
  <button type="submit">Şifrele</button>
  <br />
  @if (ViewBag.EncryptedResult != null)
  {
    <label>Şifrelenmiş Sonuç:
      @ShortenEncodedCiphertext((Ciphertext)ViewBag.EncryptedResult)
    </label>
  }
</form>
```

Şekil-32: Şifreleme Formu.

Şekil-32'deki HTML formu, bir tam sayı değerini kullanıcıdan alıp bu değeri bir HTTP POST isteğiyle bir ASP.NET MVC uygulamasında belirtilen bir kontrolcü ve eyleme iletecek şekilde tasarlanmıştır. Formun işlevselliği şu şekildedir:

action özelliği, formun verilerinin gönderileceği URL'yi belirtir. Bu URL, "Encrypt" eylemi için "Application" kontrolcüsünde bulunur. method özelliği, verilerin HTTP isteği

ile gönderileceđi metodun belirtilmesi iin kullanılır. Bu durumda, post metodu kullanılmaktadır. input etiketi, kullanıcının tam sayı deęerini girmesi iin bir metin kutusu saęlar. type="number" zellięi, sadece sayısal deęerlerin kabul edildięini belirtir. step="1" zellięi, adım deęerinin 1 olduęunu belirtir, yani yalnızca tam sayılar kabul edilir. required zellięi, bu alanın boş bırakılamayacaęını belirtir. if bloęu, ViewBag'deki EncryptedResult deęerinin varlıęını kontrol eder. Eęer bu deęer varsa, ShortenEncodedCiphertext fonksiyonu kullanılarak řifrelenmiř sonucun kısaltılmıř bir versiyonu ekrana yazdırılır. Bu form, kullanıcının bir tam sayı deęerini girip "řifrele" butonuna tıkladıktan sonra bu deęerin řifrelenmiř sonucunu ekrana gstermek iin tasarlanmıřtır.

5. SONUÇLAR

Bu çalışma, MVC katmanlı mimari altyapısı kullanılarak bir wiki web sayfası tasarlayarak homomorfik şifrelemenin temellerini, tarihini ve şifreleme metotlarını açıkladı. Wiki sayfası, homomorfik şifreleme konusunda derinlemesine bir içerik sunarak, bu önemli kriptografik yöntemin evrimini ve kullanım alanlarını ele aldı. Sayfada, homomorfik şifreleme konseptinden bahsederken, farklı şifreleme metotlarına ve bu metotların pratik uygulamalarına odaklanıldı. Özellikle, C# programlama dili için geliştirilmiş olan SEAL kütüphanesi kullanılarak homomorfik şifreleme işlemleri detaylı bir şekilde incelendi. Bu kütüphane, homomorfik hesaplama işlemlerinin kolayca gerçekleştirilmesini sağlayarak, kullanıcıların pratik uygulamalara odaklanmasına olanak tanıdı. Ayrıca, sayfa üzerinde homomorfik şifreleme işlemlerinin nasıl gerçekleştirileceğine dair Java, Python programlama dilleri ile örnekler sunuldu. Bu örnekler, kullanıcıların teorik bilgileri pratiğe dönüştürmesine yardımcı olarak, homomorfik şifrelemenin günlük yaşamdaki potansiyel uygulamalarını gösterdi. Son olarak, şifreleme işlemlerinin gerçekleştirilebilmesi için kullanıcı dostu form uygulamaları da tasarlandı. Bu uygulamalar, kullanıcıların kolayca verilerini şifreleyip çözebilmelerini sağlayarak, homomorfik şifreleme teknolojisinin geniş kitlelere ulaşmasına olanak sağladı.

Bu çalışmanın sonuçları, homomorfik şifrelemenin karmaşıklığını anlamak ve pratik uygulamalarını gerçekleştirmek isteyen araştırmacılar ve geliştiriciler için değerli bir kaynak oluşturuyor. Gelecekte, bu çalışmanın temel aldığı prensiplerin daha da geliştirilmesi ve yaygınlaştırılmasıyla, homomorfik şifreleme teknolojisinin daha geniş bir kullanıcı kitlesi tarafından benimsenmesine katkı sağlanabilir.

6. KAYNAKÇA

- [1] <https://learn.microsoft.com/tr-tr/aspnet/core/mvc/overview?view=aspnetcore-8.0>
- [2] <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [3] <https://learn.microsoft.com/tr-tr/aspnet/core/mvc/views/overview?view=aspnetcore-8.0>
- [4] <https://learn.microsoft.com/tr-tr/aspnet/core/mvc/controllers/actions?view=aspnetcore-8.0>
- [5] <https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2>
- [6] <https://medium.com/kodluyoruz/asp-net-core-ile-%C3%A7ok-katmanli%C4%B1-yap%C4%B1-ve-veri-taban%C4%B1-entegrasyonu-fe3444c5271a>
- [7] <https://appwizrd.com/blog/n-tier-architecture-cok-katmanli-mimari/66>
- [8] https://en.wikipedia.org/wiki/Microsoft_SEAL
- [9] Microsoft SEAL is an easy-to-use and powerful homomorphic encryption library.: microsoft/SEAL, Microsoft, 2019-11-20, archived from the original on 2019-05-31, retrieved 2019-11-20
<https://web.archive.org/web/20190531131953/https://github.com/microsoft/SEAL>
- [10] <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>
- [11] <https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Design>
- [12] https://en.wikipedia.org/wiki/Homomorphic_encryption
- [13] [https://en.wikipedia.org/wiki/RSA_\(cryptosystem\)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))
- [14] <https://dergipark.org.tr/en/download/article-file/818246>
- [15] https://www.researchgate.net/publication/336925416_Sifreleme_Yontemleri_ve_RSA_Algoritmasi_Uzerine_Bir_Inceleme
- [16] <https://content.e-bookshelf.de/media/reading/L-3927336-101be8f546.pdf>
- [17] <https://nyuad.nyu.edu/content/dam/nyuad/news/news/2019/june/CoPHEE-report.pdf>
- [18] <https://inet-tr.org.tr/inetconf19/bildiri/61.pdf>
- [19] Acar, A., Aksu, H., Uluagac, A. S., Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation
- [20] Ogburn, M., Turner, C., Dahal, P. (2013). Homomorphic encryption. Procedia Computer Science

- [21] arXiv:2006.10091v1 [cs.DC] <https://doi.org/10.48550/arXiv.2006.10091> - <https://arxiv.org/pdf/2006.10091>
- [22] <https://github.com/microsoft/SEAL>
- [23] <https://www.microsoft.com/en-us/research/project/microsoft-seal/>

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Ali Berdan KARASOY
Doğum Tarihi ve Yeri : 1999, İstanbul
Yabancı Dili : İngilizce
E-posta : berdankarasoy@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Mühendisliği	Düzce Üniversitesi	Devam Ediyor
Lise	Bilişim Teknolojileri	İsmet Aktar Teknik Meslek Lisesi	2017

KİŞİSEL BİLGİLER

Adı Soyadı : Yahya ARI
Doğum Tarihi ve Yeri : 1999, İstanbul
Yabancı Dili : İngilizce
E-posta : ariiyahya3@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Mühendisliği	Düzce Üniversitesi	Devam Ediyor
Lise	Kontrol Otomasyon	Tuzla Teknik Meslek Lisesi	2017