İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİYOMETRİK VERİLERİN POLİNOM İNTERPOLASYONU İLE SAKLANMASI

YÜKSEK LİSANS TEZİ

Aslıhan AKPINAR

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Programı

ARALIK 2019

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

BİYOMETRİK VERİLERİN POLİNOM İNTERPOLASYONU İLE SAKLANMASI

YÜKSEK LİSANS TEZİ

Aslıhan AKPINAR
(509161201)

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Ergün YARANERİ
Eş Danışman: Doç. Dr. Enver ÖZDEMİR

ARALIK 2019

İTÜ, Fen Bilimleri Enstitüsü'nün 509161201 numaralı Yüksek Lisans Öğrencisi Aslıhan AKPINAR, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "BİYOMETRİK VERİLERİN POLİNOM İNTERPOLASYONU İLE SAKLANMASI" başlıklı tezini aşağıdaki imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Ergün YARANERİ** İstanbul Teknik Üniversitesi	.....
Eş Danışman :	**Doç. Dr. Enver ÖZDEMİR** İstanbul Teknik Üniversitesi	.....
Jüri Üyeleri :	**Doç. Dr. Burcu TUNGA** İstanbul Teknik Üniversitesi	.....
	Dr. Öğr. Üye. Elif Segah ÖZTAŞ Karamanoğlu Mehmetbey Üniversitesi	.....
	Dr. Öğr. Üye. Tuğba YILDIRIM İstinye Üniversitesi	.....

Teslim Tarihi : **11 Kasım 2019**
Savunma Tarihi : **13 Aralık 2019**

Kardeşim Alihan'a,

ÖNSÖZ

Öncelikle her türlü bilgi ve tecrübelerini paylaşarak bana yol gösteren danışmanlarım Doç. Dr. Enver ÖZDEMİR ve Doç. Dr. Ergün YARANERİ'ye teşekkür ederim. Bununla birlikte, her zaman beni motive eden ve destekleyen arkadaşlarıma ve ayrıca her zaman beni destekleyen başta babam ve annem olmak üzere, tüm aileme sonsuz teşekkürlerimi sunarım.

Aralık 2019 Aslıhan AKPINAR

İÇİNDEKİLER

Sayfa

KISALTMALAR

AES **:** (Advanced Encryption Standart) Gelişmiş Şifreleme Standartı
Bit **:** (Binary Digit) İkili Sayı
FLOPS **:** (Floating Point Operations per Second) Saniyede Kayan Nokta ile Hesaplama
PFLOPS **:** PetaFLOPS
Hex **:** Heksadesimal

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

BİYOMETRİK VERİLERİN POLİNOM İNTERPOLASYONU İLE SAKLANMASI

ÖZET

Kimlik doğrulama, bilgilere ulaşmak isteyen kişinin yetkisi olduğunu ispatlamasıdır. Genelde kullanılan kimlik doğrulama, kullanıcı adının ve şifrenin doğru girilmesi ile gerçekleştirilir. Bu yöntem dışında insanların biyometrik özellikleri kullanılarak da doğrulama yapılabilmektedir.

Biyometrik verilerin kişiye özgü olmasından dolayı, bu verilerin kimlik doğrulamada kullanılması kişinin kendisini ispatlaması için diğer yöntemlere göre daha güvenilirdir. Şifreler veya kimlik doğrulama için kullanılan araçlar unutulabilir, kötü niyetli kişiler tarafından çalınabilir. Biyometrik veri unutulamaz, başka biri tarafından fiziksel olarak çalınamaz. Bu açıdan bakınca biyometrik veri kullanımı daha avantajlı gibi görünsede şifre veya donanıma bir zarar geldiğinde bu verileri değiştirip kimlik doğrulama sağlanabilirken biyometrik verinin zarar görmesi veya kopyalanması durumunda değiştirilebilecek bir veri olmadığından kimlik doğrulama sağlanamaz.

Biyometrik veriler bilgilere ulaşmak üzere kişinin kendisini ispatlamak için kullanılmasının yanı sıra ulusal kimlik belgesi olarak kullanımı ve suçluların tespit edilmesi gibi alanlarda kullanılmaktadır. Biyometrik verilerin bilimsel olarak kullanılması 19. yüzılın başlarında Alphonse Bertillon, "Bertillonage" adıyla bilinen insan vücudunun boyutlarını fiziksel olarak ölçerek suçluları belirlemek için kullanılan bir tekniği geliştirmesiyle olmuştur. 20. yüzyılın başlarında ise William James Herschel, parmak izlerinin zamanla değişmediği sonucuna vararak parmak izlerini sözleşmelerin reddedilmesini önlemek amacıyla kullanmıştır. Günümüzde ise düzensiz sınır geçişlerini tanımlamak için oluşturulan Avrupa Birliği parmak izi veri tabanı olan ilk çok uluslu biyometrik sistem EURODAC (European Dactyloscopy), Amerika Birleşik Devletleri'ne giren ve çıkan insanların biyometrik verilerinin tutulduğu IDENT (Automated Biometric Identification System), Hindistan'da kullanılan biyometrik veriler ile oluşturulan kimlik numarası olan Aadhaar'ın veri tabanı gibi veri tabanları çok fazla biyometrik veri barındırır. Bunlar gibi veri tabanlarından biyometrik veri çalındığı zaman o biyometrik verinin sahibiymiş gibi birçok işlem yapılabileceğinden ve biyometrik veri inkar edilemeyeceğinden dolayı bu veriler veri tabanlarında yalın halde saklanamaz. Diğer yandan her ölçümde farklı değerler gelebileceğinden şifrelerin saklandığı gibi bir bitin bile değişikliklere yol açtığı özet fonksiyonlar biyometrik verilere direkt olarak uygulanamaz. Bu yüzden biyometrik verilerin saklanması önemli bir meseledir.

Bu çalışmada, iki küme arasında karşılaştırma yaparken orjinal kümeye ait hiçbir veri kaydedilmeyerek biyometrik verilerin saklanması için önerilen algoritma incelenmiştir. Algoritma, orjinal kümenin elemanlarından üretilmiş polinomun özet değeri ve bu polinomu üretmek için $\mathbb{F}_q$'dan rastegele seçilmiş elemanları saklar.

Algoritmada polinom oluşturmak için Newton polinom interpolasyonu ve polinomun özet değerini oluşturmak için SHA-1 algoritması kullanılır. Çalışmanın 1. bölümünde güvenli haberleşmenin özellikleri, kimlik doğrulamada kullanılan yöntemler ve biyometrik verilerin kimlik doğrulamada nasıl kullanıldığı kısaca anlatılmıştır. 2. ve 3. bölümde sırasıyla kriptografik özet fonksiyonları ve polinom interpolasyonu açıklanmıştır. Sonrasında algoritmanın işleyişi anlatılmış, algoritmanın güvenlik analizi yapılmış ve SAGE kütüphanesinden faydalanılarak Python programlama dili ile gerçeklenen algoritma kodu paylaşılmıştır.

STORING BIOMETRIC DATA VIA POLYNOMIAL INTERPOLATION

SUMMARY

Since the internet is widely used, it has become important to exchange data securely. Secure communication ensures that the data of the communication cannot be accessed or changed by unauthorized persons during the communication of the parties. Confidentiality, integrity and authentication are the most important requirements for secure communication. Confidentiality ensures that data remains confidential to unauthorized persons and integrity is used to determine if data has changed since it was created, transmitted, or stored. Confidentiality is provided by symmetric and asymmetric encryption algorithms, while integrity is provided by cryptographic hash functions.

Authentication is a process that the person proves authority in case he or she wants to access data. Authentication is used wherever there is a user account or when communicating between machines. For example, authentication is used when trading on bank sites, logging into an email account, buying airline tickets, logging in to games. The authentication method that is commonly used is performed by entering the user name and password correctly. Apart from this method, biometric data and smartphones,computers,etc. are also used for authentication.

Because biometric data is personal, it is more reliable than other methods to use in authentication for proving one's authority. Passwords or hardware used for authentication can be forgotten or stolen by malicious persons. Biometric data cannot be forgotten and physically stolen by anyone else. From this point of view, the use of biometric data seems to be more advantageous, but when the password or hardware is damaged, the data that is used for authentication can be exchanged, while the biometric data can not be exchanged because it is not unalterable.

As well as biometric data is used to prove the person's identity, it is used in the fields of identification of criminals and used as a national identity document.The scientific use of biometric data was in the early 19th century when Alphonse Bertillon developed a technique used to identify criminals by physically measuring the dimensions of the human body known as "Bertillonage". At the beginning of the 20th century, William James Herschel used fingerprints to prevent rejection of agreements after he discovered that fingerprints did not change over time. Some of the current uses of biometric data are EURODAC(European Dactyloscopy) that is the first multinational biometric system established for the purpose of to identify irregular border crossing jobs, IDENT(Automated Biometric Identification System) that is a system that saves biometric data of people that entering and leaving the United States of America, The Aadhaar that is used as a unique identification document or number that would capture all the details, including demographic and biometric information, of every resident Indian individual. Their databases contain a lot of biometric data.

Biometric data cannot be stored in the databases as plain beacuse when biometric data is stolen from such databases containing biometric data, malicious persons can be act as the owner of that biometric data and biometric data cannot be denied. On the other hand, Its hash value which affected by even change of one bit can not be stored because different values can be obtained in measurements so different hash values are obtained. Therefore, the storage of biometric data is an important issue.

Biometric data is stored via helper data, which must not provide information about the original biometric data. Systems using biometric data are divided into two types according to the way they produce helper data. First type is key-binding schemes which an appropriate algorithm is applied to the biometric data then the result and key are used for generate helper data. To obtain keys, apply an appropriate key retrieval algorithm to the helper data and biometrics at authentication. Second type is key-generation schemes which helper data are only derived from the biometric data and keys are generated from the helper data.

When storing the helper data, irreversibility and unlinkability are described in ISO/IEC FCD 24745 as two properties to be considered. Irreversibility is requires it should not be possible to access the original biometric data from the stored helper data and unlinkability is requires the helper data corresponding to this data should not overlap each other when biometric data is used in various locations. ARM(Attack via record multiplicity),which is the attack resulting from the diversity of records from which the original points of the biometric data were obtained by capturing two or more biometric data used by the user in different locations, is aplicable when unlinkability is not provided. If irreversibility is not provided, the original biometric data is also reached as soon as helper data is accessed.

Most known algorithm in key-binding schemes is the fuzzy vault algorithm presented by Juels and Sudan store the key in biometric data with the help of polynomials. ARM, stolen key inversion and blended substitution attacks to fuzzy vault are presented by Scheirer and Boult.

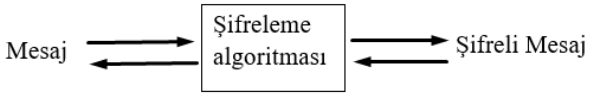
In this thesis, algorithm that is used for comparing biometric data sets have been investigated. Algorithm checks if there is at least $\%t$ match between two biometric data sets. Also algorithm is designed for the scenerio that attacker knows the s elements of biometric data set. The algorithm stores hash of polynomial degree of s which generated from $s+1$ elements of set and values which randomly chosed from $\mathbb{F}_q$, values which randomly chosed from $\mathbb{F}_q$ for generating polynomial and values of polynomial in remaining elements of set. When generating polynomial, algorithm use Newton polynomial interpolation and use SHA-1 algorithm when generate hash values of polynomials. Algorithm is based on two idea. First idea is for (x_i, y_i) where $i = 0, 1, \cdots, n$ are on the unique polynomial which at most degree is n and the second is cryptographic hash algorithms features. When polynomial generated from elements of set and elements of $\mathbb{F}_q$ and applying cryptographic hash function to polynomial, the result is unique. When comparing another biometric data with orjinal biometric data, same procedure, generating polynomial and apply the cryptographic hash function to polynomial, will apply and the result of hash function to polynomial is compared with the stored one. If hash values are same, the algorithm checks if there is at least $\%t$ matching between value of polynomial in remaining elements of set and stored values.

The details of polynomial interpolation, hash function and algorithm have been explained. Then security analysis of the algorithm has been presented. SageMath and Python programming language have been used for written a code of algorithm.

1. GİRİŞ

Bilgisayar sistemleri arasında veri alışverişi yapılmak üzere tasarlanmış elektronik iletişim ağına internet denir. İnternet, oldukça yaygın kullanıldığından beri veri alışverişinin güvenli bir şekilde yapılması önemli bir hale gelmiştir. Güvenli haberleşme, tarafların haberleşmesi esnasında yetkisi olmayan kişiler tarafından haberleşmenin verilerine ulaşılamamasını, değiştirilememesini sağlar. Güvenli haberleşmenin olmazsa olmaz üç şartı aşağıda sunulmuştur [1]:

1) Gizlilik: Veriler, yetkili kişiler dışında kalanlar için gizli tutulur. Gizliliği sağlamak için şifreleme algoritmaları kullanılır. Şifreleme algoritmaları, açık veriyi ve şifreleme anahtarını girdi olarak alır ve çıktı olarak şifreli veriyi verir aynı zamanda şifreli veri ve şifreleme anahtarını girdi olarak aldığında açık mesajı verir. (Şekil 1.1)

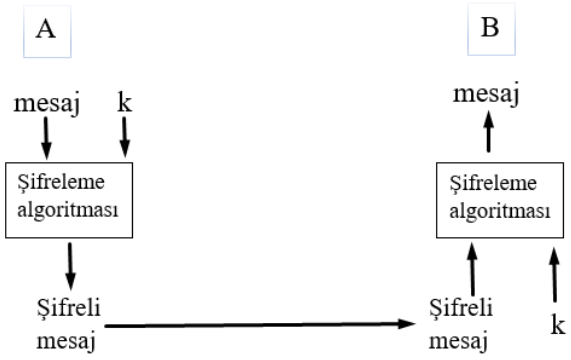


Şekil 1.1 : Şifreleme algoritmalarının genel işleyişi.

Şifreleme algoritmasının mesajdan şifreli mesaj elde edilen fonksiyonuna şifreleme, şifreli mesajdan açık mesaj elde etme fonksiyonuna deşifreleme denir. Kriptografide şifreleme algoritmaları simetrik ve asimetrik olmak üzere ikiye ayrılır.

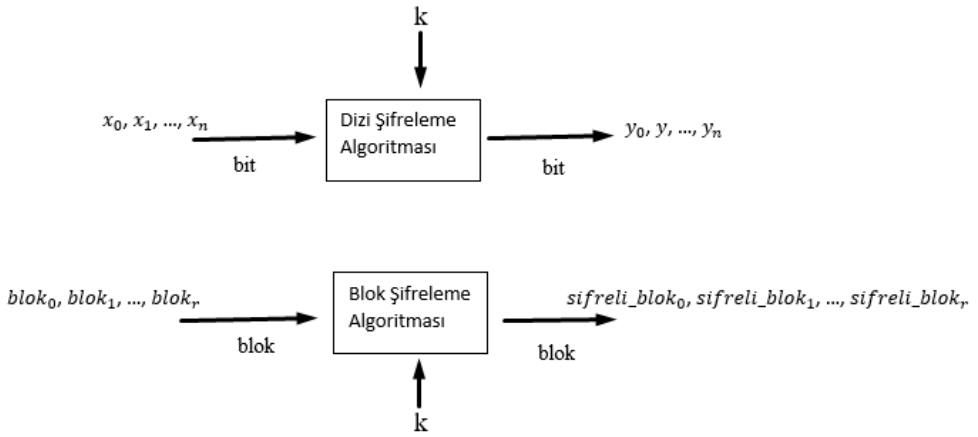
Simetrik Algoritmalar: Şifreleme ve deşifrelemeninin tek anahtarla yapıldığı algoritmalardır. Yani A ile B, mesajları şifrelemek için simetrik algoritma kullanmak üzere anlaşmış olsun. A, B'ye mesaj gönderirken bir k anahtarı ile göndermek istediği mesajı şifreler ve B'ye şifreli mesajı gönderir. B aldığı şifreli mesajı, sadece A'nın kullanmış olduğu k anahtarını kullanarak deşifreleyebilir. Yani A ile B aynı k anahtarına sahip olmalıdır. Simetrik algoritmaların genel işleyişi Şekil 1.2'de gösterilmiştir.

Simetrik algoritmalar, blok şifreleme (block cipher) ve dizi şifreleme (stream cipher) olmak üzere ikiye ayrılır. Blok şifrelemede veri, sabit uzunluklu bloklara bölünür



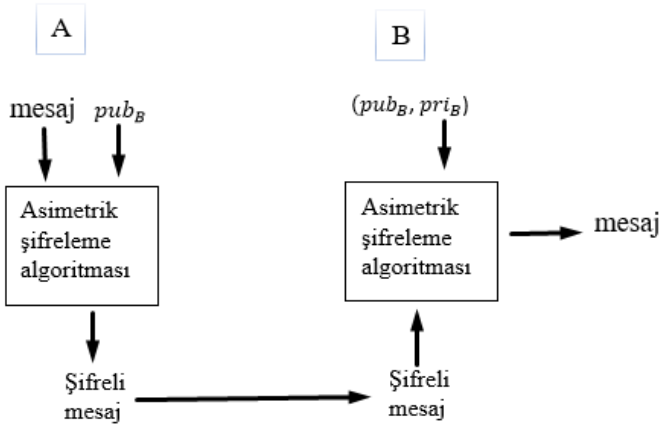
Şekil 1.2 : Simetrik şifreleme algoritmalarının genel işleyişi.

ve her bir blok için şifreleme uygulanır. Dizi şifrelemede ise her bir veri birimi dizi olarak alınır ve bu dizi için tek tek ve algoritmanın yapısına bağlı olarak tekrar tekrar şifreleme yapılır. Dizi şifreleme ve blok şifreleme algoritmalarının genel işleyişi Şekil 1.3'de gösterilmiştir. AES blok şifreleme kullanan simetrik algortimalara, RC4 ise dizi şifreleme kullanan simetrik algoritmalara örnektir. AES algoritmasında blok boyu 128 bit, anahtar boyu olarak 128,192 veya 256 bit kullanılır. RC4'da ise bir anahtar girdisinden üretilen anahtar dizisi ile verinin her bir birimi şifrelenir.



Şekil 1.3 : Dizi şifreleme ve blok şifreleme algoritmalarının işleyişi.

Asimetrik Algoritmalar: Şifreleme ve deşifreleme yapılırken farklı anahtarların kullanıldığı algoritmalardır. Asimetrik algoritmalarda, şifreleme işlemi için açık anahtar ve deşifreleme işlemi için özel anahtar kullanılır. Asimetrik algoritma kullanarak şifreleme yapmak isteyen iki kişi A ve B olsun. A, B'ye şifreli mesaj göndermek istediğinde B'nin açık anahtarı olan pub_B ile mesajı şifreler ve B'ye gönderir. B, ancak kendi özel anahtarı olan pri_B ile şifreli mesajı deşifreleyerek mesaja ulaşabilir. Asimetrik algoritmaların genel işleyişi Şekil 1.4'de verilmiştir.



Şekil 1.4 : Asimetrik şifreleme algoritmalarının genel işleyişi.

Çok büyük sayılarda çarpanlara ayırma problemine dayanan RSA algoritması günümüzde en sık kullanılan asimetrik algoritmalardandır. p ve q asal sayılar olmak üzere $n = p * q$ ve herhangi bir sayının kendisinden küçük, pozitif ve kendisi ile asal olan tam sayıların sayısı olarak tanımlan totient fonksiyonu φ olsun. m ve n aralarında asal sayılar ise $\varphi(mn) = \varphi(m)\varphi(n)$ ve m asal sayı ise $\varphi(m) = (m-1)$ olduğundan $\varphi(n) = (p-1)(q-1)$ olacaktır. $1 < e < \varphi(n)$ olacak şekilde bir e sayısı seçilir ve $de \equiv 1 mod(\varphi(n))$ olacak şekilde d hesaplanır. Burada (e,n) açık anahtar, d ise özel anahtar olarak kullanılır. Örneğin A, B'ye m mesajını gönderirken B'nin açık anahtarı (e_B, n_B) ile $c = m^{e_B} mod(n_B)$ hesaplamasını yaparak şifreli mesajı elde eder ve c şifreli mesajını B'ye gönderir. B, c şifreli mesajını aldığında özel anahtarı d_B ile $m = c^{d_B} mod(n)$ hesaplamasını yaparak A'nın göndermek istediği m mesajına ulaşır.

2) Bütünlük: Verilerin oluşturulmasından, iletilmesinden veya saklanmasından sonra değiştirilmediğinden emin olunur. 2. bölümde anlatılacak olan kriptografik özet değerleri kullanılarak verinin bütünlüğünden emin olunur.

3) Kimlik Doğrulama: Veriye ulaşmak isteyen veya haberleşilen birisinin veya bir şeyin kendini kanıtlamasıdır.

Banka sitelerinde işlem yaparken, eposta hesabına giriş yaparken, uçak bileti alırken, oyunlara giriş yaparken kısaca kullanıcı hesabının olduğu her yerde veya makineler arası iletişimde kimlik doğrulama kullanılır. Çeşitli kimlik doğrulama faktörleri vardır. Eski bir güvenlik atasözü kimlik doğrulama faktörlerinin "bildiğiniz, sahip olduğunuz veya olduğunuz bir şey" [2] olduğunu söyler. Bu üç faktör, bilgi faktörü (knowledge factor), sahiplik faktörü (possession factor) ve özgünlük faktörüdür (inherence factor). Bilgi faktörü, kullanıcı adı, şifre veya güvenlik sorusunun cevabı

gibi bilgilerdir. Sahiplik faktörü, doğrulama kodlarının, bir kullanımlık şifrelerin geldiği telefonlar veya şifre olarak kullanılan donanım gibi aygıtlara sahip olmaktır. Özgünlük faktörü ise parmak izleri, yüz tanıma gibi biyometrik verilere dayanan faktördür. Bu faktörlerden sadece birinin kullanıldığı kimlik doğrulama türüne tek faktörlü kimlik doğrulama (single-factor authentication) ve bu faktörlerden farklı ikisinin kullanılmasına iki faktörlü kimlik doğrulama (two-factor authentication) denir. Ayrıca bu faktörlere ek olarak konum veya zaman faktörlerinin kullanılabileceği de düşünülürse faktörlerin çoklu kullanıldığı duruma da çok faktörlü kimlik doğrulama (multifactor authentication) denir. Mobil kimlik doğrulama, tek kullanımlık şifreler, biyometik veriler kimlik doğrulama yöntemleri olarak kullanılabilir. Örneğin, karekodu okutarak veya parmak iziyle kimlik doğrulanabilinir. Biyometrik veriler kimlik doğrulamada genelde iki şekilde kullanılır [3]:

1) Anahtarın biyometrik veriler yardımıyla saklanması.

2) Biyometrik verilerden anahtar üretilmesi.

Biyometrik verilerin bir başkası tarafından oluşturulmasının zor olması sebebiyle kimlik doğrulamada biyometrik verilerin kullanımı şifre veya donanımsal anahtarlar kullanılmasından daha güvenilirdir. Şifreler unutulabilir, hatırlanması zor olan bir şifre konulduğunda bir yere not alındıysa ulaşılabilinir, genelde internet siteleri şifrelerin özet değerini saklar ancak özet değerlere yapılan kaba kuvvet saldırısı (brute-force attack) veya sözlük saldırısı (dictionary attack) yardımıyla şifrenin kendisi ele geçirilebilinir. Donanımsal anahtar kullanıldıysa donanım çalınabilir, kopyalanabilir veya unutulabilinir. Ayrıca biyometrik veri kullanımı kişinin fiziksel olarak o anda orada olduğunu gösterir. Örneğin, parmak izleriyle derslere giren öğrenciler orada bulunduklarını ispatlamış olurlar. Bu avantajların yanı sıra yanık, yara gibi fiziksel değişiklikler, sıcaklık ve nem gibi etkenlerden dolayı biyometrik veri her ölçüldüğünde tam olarak aynı olmaz. Ayrıca biyometrik veri bir defa ele geçirildiğinde şifreler gibi değiştirilemez ve verinin ele geçirilmesi büyük tehditler doğurabilir. Bu yüzden biyometrik verilerin nasıl saklanacağı önemlidir. Biyometrik verilerin yalın halde veri tabanında saklanması, saldırganın veri tabanına ulaştığı andan itibaren biyometrik verilerle çok rahat işlem yapabilmesine olanak sağlar. Ayrıca biyometrik verilerin özet değerleri verilerin ölçümlerinden kaynaklı olarak her defasında farklı geleceğinden bu

veriler için direkt özet değeri saklanamaz. Bu nedenle biyometrik verilerin saklanması önemli bir konudur.

Biyometrik verilerden üretilen veya bu verilerle saklanan anahtarları elde etmek için yardımcı veri adı verilen bilgiler veri tabanında saklanır. Yardımcı veriler saklanırken dikkat edilmesi gereken iki özellik ISO/IEC FCD 24745'de aşağıdaki şekilde açıklanmıştır [3]:

Tek yönlülük(Irreversibility): Saklanan yardımcı verilerden orjinal biyometrik verilere ulaşılamaması gerekir.

İlişkilendirilememe(Unlinkability): Biyometrik veri çeşitli yerlerde kullanıldığında, bu verilere karşılık gelen yardımcı veriler birbirleriyle çakışmamalıdır.

Anahtarın biyometrik verilerle saklandığı durumlarda, anahtarın elde edilmesi; kimlik doğrulama esnasında uygun bir algoritmanın biyometrik verilere uygulanarak veri tabanında bulunan yardımcı veriler ile gerçekleştirilir. Soutar ve arkadaşlarının sundukları [4–6] Mytec1 ve Mytec2, Juels ve Wattenberg [7]'in sunduğu bulanık bağlantı (fuzzy commitment), Juels ve Sudan [8]'nın sunduğu bulanık kasa (fuzzy vault) yapıları anahtarı biyometrik veriler ile saklarlar. Bulanık kasa yapısı k gizli anahtarı bir p polinomunun katsayıları ile ifade eder. U evrensel kümesinden elemanlara sahip A kümesinin elemanları ve bu elemanların polinomdaki değerleri bulunur. Bunların yanı sıra rasgele seçilmiş C kümesinden elemanlar karıştırılarak saklanır. Eğer B kümesinin elemanlarından bu polinomun katsayıları elde edilirse kasa açılmış olur. Scheirer [9] tarafından gösterilen kullanıcının farklı yerlerde kullandığı iki veya daha fazla biyometrik verileri ele geçirmesiyle biyometrik verinin orjinal noktalarının elde edildiği kayıtların çeşitliliğinden kaynaklanan saldırı (ARM - Attack via record multiplicity), zararlı yazılımlar kullanarak sistemin içinden anahtarın elde edildiği çalınan anahtar inversiyon saldırısı (Stolen key inversion attack) ve saldırganın kendi biyometrik verisini kullanıcının verisi gibi kullanmasına veriye eklenen rasgele değerler yerine kendi biyometrik verisiyle değiştirmesiyle olanak sağlayan harmanlaşmış yer değiştirme saldırısı (Blended substitution attack) bulanık kasaya yapılan saldırılar arasındadır.

Biyometrik verilerden direkt olarak anahtar üretme fikrini ise 1994'de Bodo öne sürmüş ve patentini almıştır [10]. Biyometrik verilerden anahtar üretildiği durumlarda

anahtarlar direkt olarak biyometrik verilerden üretilir ve bu durumda yardımcı veri yalnızca doğru biyometrik veriden elde edilir. Yardımcı veriler ile anahtar üretme şemaları bulanık çıkarıcı (fuzzy extractor) veya güvenli taslak (secure sketch) olarak tanımlanır. Bulanık çıkarıcı biyometrik veriden oluşturduğu rassal dizileri yardımcı veriler ile kıyaslar. Güvenli taslak ise yardımcı veriler yardımıyla orjinal biyometrik veriye ulaşır. Davida ve arkadaşları [11, 12] biyometrik verinin kendisi veya özet değerinin anahtar olarak kullanıldığı kişisel şablon şeması (private template scheme), yardımcı verilerin biyometrik verilerden sabit anahtar oluşturacak şekilde düzenlendiği niceleme şemaları (quantization schemes) biyometrik veriden anahtar üretildiği yapılara örnek oluştururlar.

Bu çalışmada, biyometrik verileri direkt olarak kimlik doğrulamada kullanılabilecek iki kümeyi karşılaştıran algoritma [13] incelenmiş, gerçeklenmiş ve güvenlik analizi yapılmıştır.

2. KRİPTOGRAFİK ÖZET FONKSİYONLARI

2.1 Giriş

Özet fonksiyonlar, herhangi bir boyuttaki girdiye karşılık sabit boyutta bir çıktı veren fonksiyonlardır. Bu çıktılara mesaj özeti veya özet değeri denir. Kriptografik özet fonksiyonunda aşağıdaki özelliklerin olması beklenir [1].

1) Tek yönlülük: Mesaj özetinden girdinin elde edilemememesidir.

2) İkinci öngörüntü direnci(Second pre-image resistance): Bir mesajın mesaj özetine karşılık gelen başka mesajın olmamasıdır. Yani bir m_1 mesajı için, $m_1 \neq m_2$ olmak üzere $h(m_1) = h(m_2)$ olacak şekilde m_2 mesajının bulunması pratikte mümkün olmamalıdır.

3) Çakışma direnci(Collision resistance): Mesaj özeti aynı olan herhangi iki mesaj bulunamamalıdır. Yani herhangi m_1,m_2 mesajları için $m_1 \neq m_2$ olmak üzere $h(m_1) \neq h(m_2)$ olmalıdır.

4) Değişken mesaj boyutu: Fonksiyon her uzunluktan mesaj için uygulanabilir olmalı.

5) Sabit çıktı boyutu: Girdi ne olursa olsun fonksiyonun çıktısı sabit uzunlukta olmalıdır.

6) Verimlilik: Hesaplanması hızlı ve verimli olmalıdır.

Bu özellikleri taşıyan bir özet fonksiyonuna verilen girdide 1 bit bile değişiklik yapıldığında fonksiyonun çıktısını değiştirir. Bu yüzden mesaj özetlerini parmakizlerine benzetebiliriz. Yani her mesaja ait tek bir mesaj özeti bulunur. Mesaj özetleri internet protokolleri, mesaj bütünlüğünü doğrulama, dijital imza, veri saklama gibi birçok yerde kullanılır. Örneğin CRC-8 (Cyclic Redundancy Check - Döngüsel Fazlalık Kontrolü) fonksiyonu herhangi bir boyuttaki girdiye karşılık 8 bit çıktı veren fonksiyondur. Herhangi bir boyuttaki girdiye karşılık sabit bir çıktı verdiğinden CRC-8 fonksiyonu bir özet fonksiyonudur. Ancak çıktı olarak gelecek değer 0 ile 2^8 arasında bir değer alabileceğinden farklı iki mesajın görüntüsünün çakışma ihtimali çok yüksektir. Bu yüzden CRC fonksiyonu özet fonksiyonudur ancak kriptografik

özet fonksiyonu değildir. Kriptografik özet fonksiyonlarından olan SHA-1 algoritması aşağıda detaylandırılmıştır.

2.2 SHA-1(Secure Hash Algorithm) Algoritması

2001 yılında RFC-3174 [14] ile yayınlanan SHA-1 algoritması, boyutu 2^{64} bitten daha küçük olan bir mesaj veya dosya için 160-bit boyutunda mesaj özeti verir. SHA-1 algoritmasında girdi olarak verilecek verinin boyutu bit olarak hesaplanır. Bit, 0 veya 1 değeri alabilen en küçük bilgi depolayan birimdir. Bitler ile işlem yapan AND, OR, XOR ve NOT fonksiyonları Çizelge 2.1'de verilmiştir.

Çizelge 2.1 : AND, OR, XOR ve NOT fonksiyonları.

x	y	x AND y	x OR y	x XOR y	NOT x
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

Bitler ile işlem yapan sola kaydırma (left shift) ve sağa kaydırma (right shift) operatorleri bulunur. Sola kaydırma "$<<$" ile, sağa kaydırma ise "$>>$" ile ifade edilir. İfadenin solunda kalan değer kaydırılacak değeri, sağındaki ise kaç tane kaydırılması gerekiğini gösterir. Örneğin, 37 sayısının 8-bit gösterimi 00100101'dir. $37 << 3$ işlemi yanı 37 sayısı 3 bit sola kaydırma işlemi yapılırsa,

00101000

elde edilir. Yani en soldaki bitler silinerek sağa 0 eklenir.

8 bit depolayan alan bayt, 2 bayt depolayan alan ise word ile ifade edilir. SHA-1 algoritmasında, word 32-bit yani 4 bayt depolayan alan olarak ile ifade edilir ve boş mesajın boyutu 0 olarak alınır. Eğer girdinin boyutu 8'in katı ise mesaj hex olarak ifade edilir. Hex, bir verinin 16'lık tabanda ifade edildiği bir sayı sistemidir. Bu sistemde veri 4 bit ile ifade edilir. Yani veri,

$$0 = 0000,\ 1 = 0001,\ 2 = 0010,\ 3 = 0011,\ 4 = 0100,\ 5 = 0101,\ 6 = 0110,\ 7 = 0111,$$
$$8 = 1000,\ 9 = 1001,\ a = 1010,\ b = 1011,\ c = 1100,\ d = 1101,\ e = 1110,\ f = 1111$$

değerlerini alabilir. Örneğin a onluk sayı sisteminde,

$$a = 1010 = 0*(2^0)+1*(2^1)+0*(2^2)+1*(2^3) = 10$$

değerine karşılık gelir.

SHA-1 algoritması mesaj özetini hesaplarken 512-bit bloklar kullanır. Bu yüzden mesaj boyutu 512-bit'in katı değil ise ekleme(padding) yapılır. Mesajın bit olarak ifadesinin boyu hesaplanır ve mesaj boyutunun iki word gösterimi elde edildikten sonra mesajın bit gösteriminin hemen arkasına '1' eklenir daha sonra ekleme yapılmış mesajda en sonda bulunan iki word, mesaj boyutunu ifade edeceği göz önüne alınarak mesaj boyutu $512*n$ olacak şekilde '0' eklenir. Örneğin,

m = 01100001 01100010 01100011 01100100 01100101 olsun.

mesaj boyu $l = 40$ ve l nin hex olarak 2 word gösterimi 00000000 00000028 olduğundan m mesajına aşağıdaki gibi ekleme yapılır.

m = 6162636465 m'nin hex gösterimi olmak üzere

61626364 65800000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000028

şeklinde ekleme yapılır.

Ekleme sonucu oluşan mesaj $16*n$ tane word içerir. n, mesajda 512-bitlik kaç blok olduğunu ifade eder. Örneğin, yukarıdaki örnekte $16*(n=1) = 16$ word bulunur.
SHA-1 algoritmasında $0 \leq t \leq 79$ olmak üzere f(t) fonksiyonları kullanılır. B, C, D 32-bitlik wordler olmak üzere,

$$f(t;B,C,D) = (B\ AND\ C)\ OR\ ((NOT\ B)\ AND\ D),\ (0 \leq t \leq 19) \tag{2.1}$$

$$f(t;B,C,D) = B\ XOR\ C\ XOR\ D,\ (20 \leq t \leq 39) \tag{2.2}$$

$$f(t;B,C,D) = (B\ AND\ C)\ OR\ (B\ AND\ D)\ OR\ (C\ AND\ D),\ (40 \leq t \leq 59) \tag{2.3}$$

$$f(t;B,C,D) = B\ XOR\ C\ XOR\ D,\ (60 \leq t \leq 79) \tag{2.4}$$

$$S(X,n) = (X << n)\ OR\ (X >> 32-n) \tag{2.5}$$

kullanılan fonksiyonlardır. Ayrıca $0 \leq t \leq 79$ olmak üzere kullanılan sabitler hex gösterimi ile aşağıdaki gibidir.

$$K(t) = 5A827999, \ (0 \leq t \leq 19) \tag{2.6}$$

$$K(t) = 6ED9EBA1, \ (20 \leq t \leq 39) \tag{2.7}$$

$$K(t) = 8F1BBCDC, \ (40 \leq t \leq 59) \tag{2.8}$$

$$K(t) = CA62C1D6, \ (60 \leq t \leq 79) \tag{2.9}$$

Mesaj özeti heasaplanırken paddingten oluşan mesaj, her biri beş tane 32-bit word'den oluşan iki yapı ve seksen tane 32-bit word olan çıktı kullanılır.
İlk yapının beş word'ü A, B, C, D ile, ikinci yapının beş word'ü $H0$, $H1$, $H2$, $H3$, $H4$ ile ifade edilsin.Ayrıca seksen word'den oluşan çıktı $w(0)$, $w(1)$, ..., $w(79)$ ile ifade edilsin ve TEMP bir word olan bir yapı olsun.

Mesaj için işlemlere başlamadan önce

$$H0 = 67452301 \tag{2.10}$$

$$H1 = EFCDAB89 \tag{2.11}$$

$$H2 = 98BADCFE \tag{2.12}$$

$$H3 = 10325476 \tag{2.13}$$

$$H4 = C3D2E1F0 \tag{2.14}$$

olarak alınır.

$1 \leq i \leq n$ olmak üzere $M(i)$'lere (Eklemeden oluşan 16-word bloklar) uygulanacak işlemler aşağıdaki şekildedir:

1) $M(i)$, $w(0)$ en soldaki word olmak üzere 16 word'e bölünür. $w(0)$, $w(1)$, ..., $w(15)$.

2) $16 \leq t \leq 79$ için

$$w(t) = S(w(t-3) \ XOR \ w(t-8) \ XOR \ w(t-14) \ XOR \ w(t-16), 1) \tag{2.15}$$

3)

$$A = H0, \ B = H1, \ C = H2, \ D = H3, \ E = H4 \tag{2.16}$$

4) $0 \leq t \leq 79$ için

$$TEMP = S(A,5) \ + \ f(t;B,C,D) \ + \ E \ + \ W(t) \ + \ K(t) \tag{2.17}$$

$$E = D;\ D = C;\ C = S(B,30);\ B = A;\ A = TEMP \tag{2.18}$$

5)

$$H0 = H0 + A,\ H1 = H1 + B,\ H2 = H2 + C,\ H3 = H3 + D,\ H4 = H4 + E \tag{2.19}$$

En son bloktan ($M(n)$) çıkan 160-bit $H0H1H2H3H4$ mesaj özeti olur.

Kısaca, SHA-1 algoritması mesajın boyu 512-bit'in katı değilse 512-bit'in katı olacak hale getirir. $H0 - H4$ başlangıç değerlerini atadıktan sonra mesajın her 512-bitlik bloğu için 1-4 adımlarını uygular. En son bloktan çıkan 160-bit $H0H1H2H3H4$ verisi mesaj özeti olur.

3. POLİNOM İNTERPOLASYONU

3.1 Giriş

Bir f fonksiyonunda $i = 0, 1, \cdots, n$ olmak üzere x_i noktalarına karşılık gelen değerler $f(x_i)$'ler olsun. f fonksiyonunun tanımı bilinmeden $(x_i, f(x_i))$ noktalarını kullanarak herhangi x noktası için $f(x)$ değerini bulma işlemine interpolasyon denir.

3.2 Doğrusal İnterpolasyon

x_0 ve x_1 noktaları için bir $f(x)$ fonksiyonunun değerleri $f(x_0)$ ve $f(x_1)$ bilinsin ancak $f(x)$ fonksiyonunun tanımı bilinmesin. Bu durumda $f(x)$, birinci dereceden polinom yardımı ile aşağıdaki şekilde hesaplanabilir. $p_1(x)$ birinci dereceden bir polinom olmak üzere

$$p_1(x) = a_0 + a_1 x \tag{3.1}$$

olsun. Polinomun x_0 ve x_1 noktalarında alacağı değer ile $f(x)$ fonksiyonunun bu noktalarda alacağı değer aynı olacağından,

$$f(x_0) = p_1(x_0) = a_0 + a_1 x_0 \tag{3.2}$$

$$f(x_1) = p_1(x_1) = a_0 + a_1 x_1 \tag{3.3}$$

olur. Buradan

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \tag{3.4}$$

$$a_0 = f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} x_0 \tag{3.5}$$

eşitlikleri elde edilir. Bu durumda p_1 polinomu,

$$p_1(x) = (f(x_0) - \frac{f(x_1) - f(x_0)}{x_1 - x_0} x_0) + (\frac{f(x_1) - f(x_0)}{x_1 - x_0}) x \tag{3.6}$$

olur. p_1 polinomunun Newton formunda gösterimi,

$$p_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0) \tag{3.7}$$

ve Lagrange formunda gösterimi,

$$p_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1) \tag{3.8}$$

şeklindedir.

3.3 Lagrange İnterpolasyonu

$i = 0, 1, \cdots, n$ iken x_i noktalarına karşılık gelen f fonksiyonunun görüntüleri $f(x_i)$'ler olsun. $L_i(x)$'ler n. dereceden polinomlar olmak üzere

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i) \tag{3.9}$$

olsun. Bu durumda $p_n(x_j) = f(x_j)$ olmalıdır. Dolayısıyla,

$$p_n(x_j) = \sum_{i=0}^{n} L_i(x_j) f(x_i) \tag{3.10}$$

$$f(x_j) = \sum_{i=0}^{n} L_i(x_j) f(x_i) \tag{3.11}$$

$$f(x_j) = L_0(x_j) f(x_0) + L_1(x_j) f(x_1) + \cdots + L_j(x_j) f(x_j) + \cdots + L_n(x_j) f(x_n) \tag{3.12}$$

olur. Bu durumda eşitliğin sağ tarafının sol tarafına eşit olması için $L_j(x_j) = 1$ ve $i \neq j$ iken $L_i(x_j) = 0$ olması gerekir. O zaman

$$L_i(x_j) = 1 \quad \text{if } i = j \tag{3.13}$$

$$L_i(x_j) = 0 \quad \text{if } i \neq j \tag{3.14}$$

olur. $L_i(x)$ polinomu x_i noktası için hesaplandığında $L_i(x_i) \neq 0$ ve $L_i(x_i) = 1$ olması gerektiğinden

$$L_i(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_n)} \tag{3.15}$$

elde edilir. Yani

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{3.16}$$

olur. Dolayısıyla

$$p_n(x) = \sum_{i=0}^{n} (\prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}) f(x_i) \tag{3.17}$$

polinom elde edilir. Yani,

$$p_n(x) = \sum_{i=0}^{n} (L_i(x)) f(x_i) \tag{3.18}$$

olur.

Teorem 3.4.1 $i = 0, 1, \cdots, n$ olmak üzere x_i ayrık noktaları ve y_i noktaları K cisminde olsun. (x_i, y_i) noktaları derecesi n den küçük olan bir tek $p(x)$ polinomunun grafiğinden geçer. [15]

İspat: Teoremin ispatı iki kısımdan oluşur. Önce n. dereceden $p(x)$ polinomunun var olduğu sonra tek olduğu ispatlanır.

1) $y_i = f(x_i)$ olsun. Bu durumda Lagrange polinom interpolasyonuyla elde edilen n. dereceden $p(x)$ polinomu

$$p_n(x) = \sum_{i=0}^{n} (L_i(x)) f(x_i) \tag{3.19}$$

şeklinde olmalıdır. Bu durumda $y_i = f(x_i)$ olduğu için $f(x_i)$'ler var olduğundan $p_n(x)$ polinomunun varlığı $L_i(x)$'in varlığına bağlıdır. $L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$ olduğundan ve $L_i(x)$'i tanımsız yapan değer olmadığından $L_i(x)$ vardır ve tanımı gereği en fazla n. dereceye çıkar. Ayrıca $L_i(x_j) = \prod_{j=0, j\neq i}^{n} \frac{x_j - x_j}{x_i - x_j}$ olacağından

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \text{ olmak üzere } L_i(x_j) = \delta_{ij} \tag{3.20}$$

olarak tanımlanabilir. Bu yüzden

$$p_n(x_j) = \sum_{i=0}^{n} (\delta_{ij}) f(x_i) \tag{3.21}$$

eşitliği elde edilir ve $i \neq j$ olan tüm durumlar için $\delta_{ij} = 0$ ve $i = j$ iken $\delta_{ij} = 1$ olduğu göz önüne alınırsa

$$p_n(x_j) = f(x_j) \tag{3.22}$$

sağlanır. Sonuç olarak $y_i = f(x_i)$ olarak alınırsa grafiği (x_i, y_i) noktalarından geçen ve en fazla n. dereceden olabilecek bir $p_n(x)$ polinomunun varlığı ispatlanmış olur.

2) (x_i, y_i) noktalarından grafiği geçen iki polinom $p_n(x)$ ve $p'_n(x)$ olsun. Ayrıca

$$d(x) = p_n(x) - p'_n(x) \tag{3.23}$$

olsun. Bu durumda $p_n(x)$ ve $p'_n(x)$ n. dereceden polinomlar olduğundan $d(x)$, n. dereceden bir polinom olur. Ayrıca $i = 0, 1, \cdots, n$ için

$$d(x_j) = p_n(x_j) - p'_n(x_j) = f(x_j) - f(x_j) = 0 \tag{3.24}$$

olur. $d(x)$ polinomu n. dereceden bir polinom olduğundan en az $n+1$ köke sahip olması gerekir ancak $i = 0, 1, \cdots, n$ için $d(x_j) = 0$ bulunduğundan bu yalnızca $d(x) = 0$ olması durumunda sağlanır. Bu durumda $p_n(x) = p'_n(x)$ elde edilir. Sonuç olarak (x_i, y_i) noktalarından grafiği geçen ve en fazla n. dereceden olan $p_n(x)$ polinomu tek olur.

3.4 Newton Bölünmüş Farklar İnterpolasyonu

$x_0, x_1, \cdots, x_{k-1}$ noktalarını kullanarak elde ettiğimiz $k-1$ dereceden polinom $P_{k-1}(x)$ olsun. $(x_k, f(x_k))$ noktası $x_0, x_1, \cdots, x_{k-1}$ noktalarına eklenirse elde edilecek olan k dereceli polinom $P_{k-1}(x)$ polinomu yardımıyla aşağıdaki gibi elde edilir.

$h(x)$, k dereceli bir polinom olmak üzere $P_k(x) = P_{k-1}(x) + h(x)$ olsun. P_{k-1}, $i = 0, 1, \cdots, k-1$ için elde edilmiş olan polinom olduğundan $P_{k-1}(x_i) = f(x_i) = P_k(x_i)$ olmalıdır. Dolayısıyla $i = 0, 1, \cdots, k-1$ için $h(x_i) = 0$ olur. Yani

$$h(x) = A(x-x_0)(x-x_1)\cdots(x-x_k) \tag{3.25}$$

şeklinde olmalıdır. O zaman genel olarak $P_k(x)$, $x_0, x_1, \cdots, x_{k-1}$ noktalarının x_k noktasındaki bölünmüş farkı $A = f[x_0, x_1, \cdots, x_{k-1}; x_k]$ olmak üzere

$$P_k(x) = P_{k-1}(x) + A(x-x_0)(x-x_1)\cdots(x-x_k) \tag{3.26}$$

dir.

$$P_0(x) = f(x_0) \tag{3.27}$$

$$P_1(x) = P_0(x) + f[x_0; x_1](x-x_0) \tag{3.28}$$

$$P_2(x) = P_1(x) + f[x_0, x_1; x_2](x-x_0)(x-x_1) \tag{3.29}$$

.

.

.

$$P_{k-1}(x) = P_{k-2}(x) + f[x_0, x_1, \cdots, x_{k-2}; x_{k-1}](x-x_0)(x-x_1)\cdots(x-x_{k-2}) \tag{3.30}$$

denklemleri taraf tarafa toplanırsa aşağıdaki genel denklem elde edilir.

$$P_{k-1}(x) = f[x_0] + f[x_0; x_1](x-x_0) + \cdots + f[x_0, x_1, \cdots, x_{k-2}; x_{k-1}](x-x_0)(x-x_1)\cdots(x-x_{k-2}) \tag{3.31}$$

Buradaki $f[x_0]$,$f[x_0; x_1]$,$\cdots$,$f[x_0, x_1, \cdots, x_{k-2}; x_{k-1}]$ bölünmüş farklar Newton ve Lagrange interpolasyonlarındaki x_k'nın katsayısı olan

$$f[x_0, x_1, \cdots, x_{k-1}; x_k] = \sum_{i=0}^{k} \frac{f(x_i)}{\prod\limits_{j=0, j\neq i}^{k} x_i - x_j} \tag{3.32}$$

dan elde edilir. Dolayısıyla,

$$f[x_0] = f(x_0) \tag{3.33}$$

$$f[x_0; x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_1; x_0] \tag{3.34}$$

$$f[x_0,x_1;x_2] = \frac{f[x_1;x_2]-f[x_0;x_1]}{x_2-x_0} \tag{3.35}$$

.

.

.

$$f[x_0,x_1,\cdots,x_{k-1};x_k] = \frac{f[x_1,x_2,\cdots,x_{k-1};x_k]-f[x_0,x_2,\cdots,x_{k-2};x_{k-1}]}{x_k-x_0} \tag{3.36}$$

olur. Gerekli bölünmüş farkları hesaplayarak k dereceli $P_k(x)$ polinomuna

$$P_k(x) = f[x_0]+f[x_0;x_1](x-x_0)+\cdots+f[x_0,x_1,\cdots,x_{k-1};x_k](x-x_0)(x-x_1)\cdots(x-x_{k-1}) \tag{3.37}$$

denklemini kullanarak ulaşılabilinir.

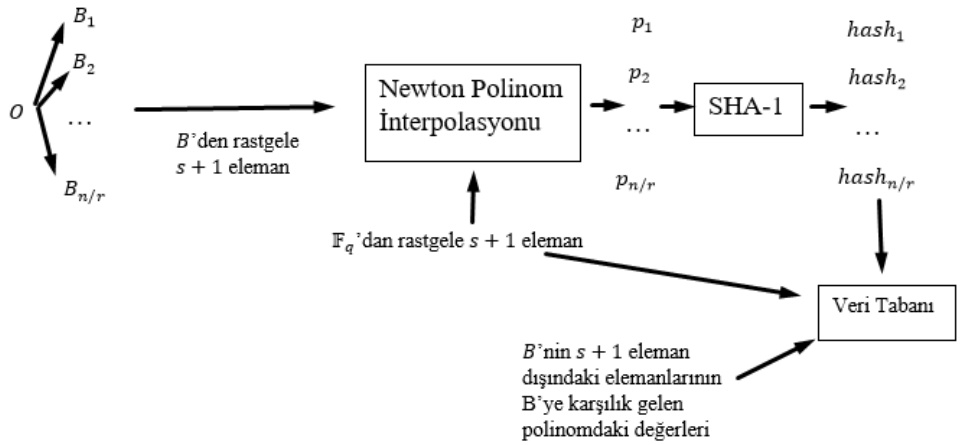
4. İKİ KÜMEYİ KARŞILAŞTIRAN ALGORİTMA

Algoritma $i = 0, 1, \cdots, n$ olmak üzere x_i ayrık noktaları ve bunlara karşılık gelen y_i noktalarının tek bir $p(x)$ polinomunun grafiğinden geçmesine ve özet fonksiyonlarının tek yönlü olmasına dayanır. Burada polinomu üretmek için Newton bölünmüş farklar interpolasyonu ve kriptografik özet algoritması olarak SHA-1 kullanılacaktır.

4.1 Algoritmanın İşleyişi

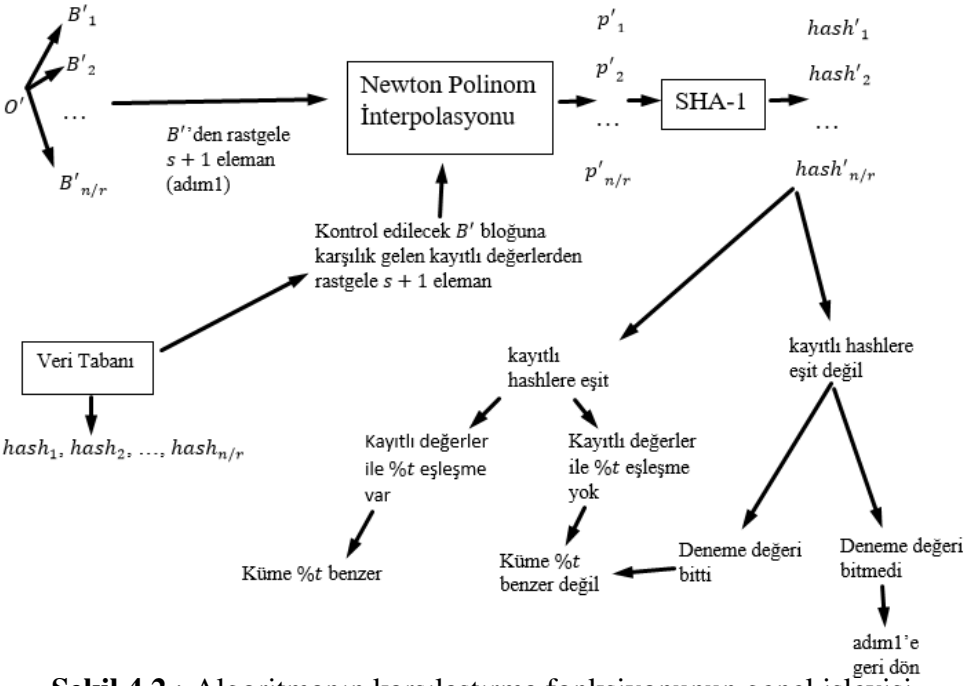
Orjinal kümenin elemanlarına ait hiç bir veri kaydetmeden orjinal küme ile başka küme arasında karşılaştırma yapan algoritma, kimlik doğrulama yapılırken biyometrik verilerin güvenli bir şekilde kullanılması için önerilmiştir. Algoritmada, polinom interpolasyonu ve kriptografik özet fonksiyonları yardımıyla biyometrik veri kaydedilmeden başka bir biyometrik veri ile karşılaştırılmasına imkan sağlayacak veriler kaydedilir. Saldırganın orjinal kümeden $\%s$ eleman bildiği durumda orjinal küme ile başka bir küme arasında $\%t$ benzerlik olup olmadığı kontrol edilir. q, p asal sayısının kuvveti olarak ifade edilebilen bir sayı olmak üzere q elemanlı sonlu cisim $\mathbb{F}_q$ olsun. Orjinal küme bloklara bölünerek her bloğa karşılık, blok elemanlarının bir kısmı ve $\mathbb{F}_q$'dan rastgele seçilen değerler ile polinom üretilir. Polinomun özet değeri, onu üretmek için kullanılan $\mathbb{F}_q$'dan seçilen rastgele değerler ve bloğun polinom üretmek için kullanılmayan elemanlarının polinomdaki değerleri kaydedilir. Bu durumda orjinal kümeyle başka bir kümeyi karşılaştırmak için veri tabanında orjinal kümenin bloklarına ait polinom özet değerleri, polinomların üretilmesi için blok elemanlarıyla beraber kullanılan değerler ve blokların kullanılmayan elemanlarının polinom değerleri bulunur. Algoritmanın kayıt fonksiyonu genel olarak Şekil 4.1'de verilmiştir.

Karşılaştırılacak küme de bloklara bölünerek, orjinal bloklara karşılık polinomları üretmek için kaydedilen $\mathbb{F}_q$'nun elemanları ile polinomlar oluşturulup, polinomların özet değerleri elde edilir. Her bloğa karşılık gelen polinomun özet değeri kayıtlı özet değer ile eşleşirse karşılaştırılacak bloktan polinom üretmek için kullanılmayan



Şekil 4.1 : Algoritmanın kayıt fonksiyonunun genel işleyişi.

elemanların polinom değerlerinin kayıtlı değerler ile kümenin genelinde *%t* eşleşip eşleşilmediğine bakılır. Eğer eşleşirse kümeler *%t* oranında benzer olur. Eğer eşleşme olmaz ise belirlenen deneme sayısı kadar bloklardan polinom üreterek karşılaştırma işlemi tekrarlanır. Deneme sayısı sonlanıncaya kadar eşleşme olmamışsa kümeler *%t* benzer değildir. Biyometrik veriler ile kimlik doğrulama yapıldığı düşünülürse iki biyometrik veri kümesi arasında *%t* benzerlik bulunduğunda kimlik doğrulama sağlanmış olur. Eğer bu kümeler *%t* benzer değilse kimlik doğrulama gerçekleşmez. Algoritmanın karşılaştırma fonksiyonu genel olarak Şekil 4.2'de verilmiştir.



Şekil 4.2 : Algoritmanın karşılaştırma fonksiyonunun genel işleyişi.

Yani, orjinal küme $O = \{x_1, \cdots, x_n\}$ ve $O' = \{x'_1, x'_2, \cdots, x'_n\}$ olsun. Ayrıca saldırgan orjinal kümeden $\%s$ kadar elemanı bilsin. Öncelikle orjinal küme her blokta r elaman olacak şekilde bloklara ayrılır. Kümenin genelinde $\%t$ benzerlik olup olmadığına bakılacağından her blok için bakılacak benzerlik oranı $T = \frac{t}{100} \times r$ ve kümenin genelinde $\%s$ elemanın bilindiği varsayıldığından her bloğa karşılık bilindiği varsayılan eleman sayısı $S = \frac{s}{100} \times r$ olmak üzere:

1) O kümesi her blokta r eleman olacak şekilde $B_1, B_2, \cdots, B_{n/r}$ bloklarına ayrılır. Her blok için aşağıdaki işlemler tekrarlanır.

2) $B = \{x_1, x_2, \cdots, x_r\}$ olmak üzere $x_1, x_2, \cdots, x_{S+1}$ elemanları için $\mathbb{F}_q$'dan $y_1, y_2, \cdots, y_{S+1}$ değerleri rastgele seçilir.

3) $(x_1, y_1), (x_2, y_2), \cdots, (x_{S+1}, y_{S+1})$ kümesi için newton bölünmüş farklar yöntemi ile interpolasyon yapılarak $p(x)$ polinomu elde edilir.

4) B bloğunun geriye kalan elemanları $x_{S+2}, x_{S+3}, \cdots, x_r$ elemanları için $p(x_{S+2}), p(x_{S+3}, \cdots, p(x_r))$ değerleri hesaplanır. Bu değerler $y_{S+2} = p(x_{S+2}), y_{S+3} = p(x_{S+3}), \cdots, y_r = p(x_r)$ olsun.

5) Elde edilen $p(x)$ polinomunun SHA-1 özet değeri hesaplanır.

6) Polinomun özet değeri ile birlikte $y = \{y_1, y_2, \cdots, y_r\}$ değerleri depolanır.

Daha sonra O kümesi ile karşılaştırılacak olan O' kümesi için algoritmanın karşılaştırma işlemi aşağıdaki gibi gerçekleşir.

1) O' kümesi her blokta r eleman olacak şekilde $B'_1, B'_2, \cdots, B'_{n/r}$ bloklarına ayrılır. Her blok için aşağıdaki işlemler tekrarlanır.

2) $B' = \{x'_1, x'_2, \cdots, x'_r\}$ olmak üzere rastgele seçilen $x'_1, x'_2, \cdots, x'_{S+1}$ elemanları için bloğa karşılık gelen kaydedilmiş y değerleri eşleştirilir.

3) $(x'_1, y_1), (x'_2, y_2), \cdots, (x'_{S+1}, y_{S+1})$ kümesi için newton bölünmüş farklar yöntemi ile interpolasyon yapılarak $p'(x)$ polinomu elde edilir.

4) Polinomun SHA-1 özet değeri traile ulaşılana kadar $x'_1, x'_2, \cdots, x'_{S+1}$ rastgele seçilerek hesaplanır.

-Eğer özet değeri kayıtlı özet değeri ile eşleşirse $x'_1, x'_2, \cdots, x'_{S+1}$ elemanları dışında kalan $x'_{s+2}, x'_{s+3}, \cdots, x'_r$ için $y'_{s+2} = p(x'_{s+2}), y'_{s+3} = p(x'_{s+3}), \cdots, y'_r = p(x'_r)$ hesaplanır, kayıtlı olan $y_{s+2}, y_{s+3}, \cdots, y_r$ değerleri ile karşılaştırılır ve bu değerlerle T kadar eşleşip

eşleşmediğini kontrol eder. Her blok için T benzerlik elde edilirse O' ile O arasında $\%t$ benzerlik sağlanmış olur.

4.2 Algoritmanın Güvenlik Analizi

Saldırganın, bilgileri kayıtlı olan kişi gibi davranması için bir polinom üretmesi gerekir ve bu polinomun özet değeri ile kaydedilmiş değerin eşleşmesi ve elinde olan noktalardan y değerlerinden en az $\%t$ benzerlik elde etmesi gerekir. Doğru noktaları bulmak için kaba kuvvet saldırısı yapabilir. Kaba kuvvet(brute-force) saldırısı, elde edilmek istenen değeri bulmak için tüm olası değerlerin denenmesi ile yapılır. Örneğin, 2-bit için yapılacak kaba kuvvet saldırısında taranacak olan değerler

$$00,\ 01,\ 10,\ 11$$

olduğundan 2-bit için olası kombinasyon sayısı 4'tür. Buradan hareketle çeşitli anahtar boyutları için olası kombinasyon sayıları Çizelge 4.1'de verilmiştir.

Çizelge 4.1 : Çeşitli anahtar boyutları için olası kombinasyon sayıları.

Anahtar Boyutu	Olası Bit Kombinasyonları
1-bit	2
2-bit	4
4-bit	16
8-bit	256
16-bit	65536
32-bit	4.2×10^9
56-bit	7.2×10^{16}
64-bit	1.8×10^{19}
128-bit	3.4×10^{38}
192-bit	6.2×10^{57}
256-bit	1.1×10^{77}

Her blokta oluşturulması gereken polinomun derecesi S'dir. Bu durumda saldırganın bu polinomu elde etmesi için $S+1$ noktaya ihtiyacı vardır. Bloklardaki bir noktanın bir bayta karşılık geldiği düşünülürse ve bir baytın 8 bit olduğu göz önüne alınırsa saldırganın bir blok için $2^{(S+1)*8}$ deneme yapması gerekir. Tüm bloklar için

$$2^{(S+1)*8*(n/r)} \tag{4.1}$$

deneme yapması gerekir. Saldırganın veri tabanında yönetim yetkisi yoksa, her blokta polinomun özet değeri eşleştikten sonra $T = \frac{t \times r}{100}$ benzerliği yakalaması gerekir. Bu durumda bir blok için 2^{T*8} yani tüm bloklar için

$$2^{(T*8)*(n/r)} \tag{4.2}$$

deneme daha yapması gerekir. Saldırgan, polinom özeti ve *%t* benzerlik oranını yakalamak için toplamda

$$2^{(S+T+1)*8*(n/r)} \tag{4.3}$$

deneme yapması gerekir.

Denemelerin yapılması için gereken zaman FLOPS'lar yardımıyla hesaplanabilir. FLOPS, mikroişlemcilerin hız performansını göstermek için kullanılan bir ölçüdür. Haziran 2019'da açıklanan verilere [16] göre en hızlı super bilgisayarın tepe hızı 148.6 PFLOPS'dur. PFLOPS, saniyede bir katrilyon kayan nokta işlemini ifade eder. Bu durumda

$$148.6\text{PFLOPS} = 148.6 \times 10^{15}\text{FLOPS} \tag{4.4}$$

olur. Kombinasyon kontrolü için gerekli FLOPS sayısını 1000 varsayalım, saniyedeki kombinasyon kontrol sayısı

$$\frac{148.6 \times 10^{15}}{1000} = 148.6 \times 10^{12} \tag{4.5}$$

olarak hesaplanır. Bir yıldaki saniye sayısı

$$365 \times 24 \times 60 \times 60 = 31536000 \tag{4.6}$$

dır. Bu durumda 128-bit için yapılan kaba kuvvet saldırısı için gerekli olan zaman yaklaşık olarak,

$$\frac{3.4 \times 10^{38}}{(148.6 \times 10^{12}) \times 31536000} = 7.2 \times 10^{16} \text{ yıl} \tag{4.7}$$

olur. 128-bit'in güvenlik seviyesi olduğu kabul edilirse saldırganın veri tabanında yönetim yetkisi olmadığı durumda

$$2^{(S+T+1)*8*(\frac{n}{r})} > 2^{128} \tag{4.8}$$

$$(S+T+1)*8*(\frac{n}{r}) > 128 \tag{4.9}$$

$$(S+T+1)*(\frac{n}{r}) > 16 \tag{4.10}$$

sağlanmalıdır.

Yukarıdaki hesaplamalar saldırganın orjinal kümeden hiç veri bilmemesi durumunda hesaplandı ancak incelenen algoritma saldırganın orjinal kümeden s kadar eleman bildiği duruma göre tasarlanmıştır. Saldırganın orjinal kümeden s kadar bildiği ve yönetici yetkilerine sahip olduğu senaryoda saldırgan, kayıtlı y değerlerine ve özet değerlerine ulaşabilir. Algoritmada saldırganın her bloktan S eleman bildiği varsayılarak $S+1$ noktadan S dereceli polinom elde edilir. Yönetici yetkisi olan saldırganın orjinal kümeyi tespit edebilmesi için doğru polinomu üretmesi gerekir. Doğru polinomu ürettikten sonra elinde y değerleri bulunduğundan orjinal kümenin elemanlarına ulaşabilir. Her bloğa karşılık doğru polinomu elde edebilmesi için elindeki S noktaya ek bir noktaya daha ihtiyaç duyar. Bu durumda yönetici yetkisi olan saldırganın yapması gereken deneme sayısı

$$2^8 * (\frac{n}{r}) \tag{4.11}$$

olur. Polinomu elde ettikten sonra elindeki y değerlerini ve polinomu kullanarak orjinal kümeyi elde edebilir.

4.3 Algoritmanın Gerçeklenmesi

Anlatılan algoritma SageMath [17] notebook ve Python [18] programlama dili kullanılarak gerçeklenmiştir. Algoritma kodu aşağıda paylaşılmıştır.

```
import random
import hashlib
import json
from decimal import Decimal
import datetime
```

```
def getCoeffs(x, y, q):
    n = len(y)
    coef_pyramid = createMatrix(n, n)
    coef_pyramid[0] = y
    for i in range(0, n):
        for j in range(n - i - 1):
            asd = inverse_mod(x[j + i + 1] - x[j], q)
            coef_pyramid[i + 1][j] = Mod((coef_pyramid[i][j + 1] - coef_pyramid[i][j]) * asd, q)
    coef_pyramid_new = []
    for j in range(0, n):
        coef_pyramid_new.append(coef_pyramid[j][0])
    return coef_pyramid_new


def createMatrix(rowCount, colCount):
    mat = []
    for i in range(rowCount):
        rowList = []
        for j in range(colCount):
            rowList.append(0)
        mat.append(rowList)

    return mat


def newtonInt(x, y, q):
    coefVector = getCoeffs(x, y, q)
    # modq da polinom cismi olusturma
    K = Zmod(q)
    P = PolynomialRing(K, implementation='NTL', names=('t',))
    (t,) = P._first_ngens(1)

    n = len(coefVector)

    finalPol = 0
    for i in range(n):
        p = 1
        for j in range(i):
            p_temp = t - x[j]
            p = p * p_temp
        p *= coefVector[i]
        finalPol = finalPol + p
    return finalPol


#  Kumeyi lenS elemanli bloklara ayirir.
def splitBlock(compS, lenS):
    newCompS = []
    up = lenS
    down = 0
    plus = lenS
    u = len(compS) / lenS
    for j in range(1, u + 1):
        newCompS.append(compS[down:up])
        down = up
        up = up + plus
    return newCompS
```

```
# Bloklardan S+1 eleman olacak sekilde sirali listler olusturur.
def splitBlock2s(blockS, lens):
    newCompS = []
    n = len(blockS)
    for i in range(0, n - lens + 1):
        newCompS.append(blockS[i:i + lens])
    return newCompS


def getRandom(n):
    L = GF(q)
    y = []
    for i in range(0, n):
        y.append(L.random_element())
    return y


def write2file(filepath, hashpoly, newy):
    newyy = json.loads(str(newy), parse_float=Decimal)
    dict_object = []
    for i in range(0, len(hashpoly)):
        dict_object.append(dict(hashvalue=hashpoly[i], yvalue=newyy[i]))
    with open(filepath, "w") as f:
        for line in dict_object:
            json.dump(line, f)
            f.write("\n")


def readfromfile(filepath):
    data = []
    ready = []
    readhash = []
    with open(filepath, "r") as f:
        for line in f:
            data.append(json.loads(line))
    for i in range(0, len(data)):
        ready.append(data[i]["yvalue"])
        readhash.append(data[i]["hashvalue"])
    return readhash, ready


# saveHash fonksiyonu orjinal S kumesini bloklara bolup, her blok icin
# rasgele s+1 eleman secer, polinom hesaplatir
# ve y degerlerini kaydeder.
def saveHash(filePath, S, r, bigS):
    hashPoly = []
    newy = []
    splittedS = splitBlock(S, r)
    n = len(splittedS)
    a = bigS + 1
    for i in range(0, n):
        savey = []
        kk = 0
        for ii in range(0, r):
            savey.append(0)
        newS = []
        hashF = hashlib.sha1()
        rand_items = random.sample(splittedS[i], int(a))
```

```
        y = getRandom(a)

        for rand_item in rand_items:
            position = splittedS[i].index(rand_item)
            savey[position] = y[kk]
            kk += 1

        poly = newtonInt(rand_items, y)
        hashF.update(str(poly))
        hashPoly.append(hashF.hexdigest())

        for el in splittedS[i]:
            if el not in rand_items or splittedS[i].count(el) > 1:
                newS.append(el)
        for el in newS:
            position = splittedS[i].index(el)
            savey[position] = poly(el)
        newy.append(savey)

    write2file(filePath, hashPoly, newy)


# bu algoritma bloktan rasgele s+1 eleman secip polinom uretir
# ve hash degeri hesaplatr.
def algorithmRandom(comS, savedhash, savedy, bigS, bigT, r):
    splitcoS = splitBlock(comS, r)  # comS yi r elemanli bloklara boler.
    n = len(splitcoS)
    print "kac blok var: " + str(n)
    a = bigS + 1  # S+1 sayisini belirler
    passedblock = 0

    for i in range(0, n):
        trail = r - bigT
        # splitcoS icerisinde r elamanli bloklar bulunur.
        # Bu blokalrdan rasgele s+1 eleman polinom retilir
        # ve polinomun ozet degerinin eslesip eslesmedigine bakilir
        # bu dongu en fazla trail kere tekrarlanir.
        while trail != 0:
            sim = a
            coy = []
            newcoS = []
            randcos_items = random.sample(splitcoS[i], int(a))
            hashF = hashlib.sha1()

            for el in splitcoS[i]:
                if el not in randcos_items or splitcoS[i].count(el) > 1:
                    newcoS.append(el)

            for rand_item in randcos_items:
                position = splitcoS[i].index(rand_item)
                coy.append(savedy[i][position])

            coPoly = newtonInt(randcos_items, coy)
            hashF.update(str(coPoly))

            if hashF.hexdigest() == savedhash[i]:
                for el in newcoS:
                    if coPoly(el) in savedy[i]:
                        sim += 1
```

```
            else:
                trail -= 1

            if sim >= bigT:
                print "THIS BLOCK PASS"
                print trail
                trail = 0
                passedblock += 1


    print "eslesen blok sayisi: " + str(int(passedblock))
    print "eslesmeyen blok sayisi: " + str(int(n - passedblock))


def main():
    S = list(map(int, raw_input("Orjinal kume: ").split(',')))
    compS = list(map(int, raw_input("Karsilastirilacak kume: ").split(',')))

    q = int(raw_input("Sonlu cisim icin q sayisi: "))

    s = int(raw_input("Saldirganin elindeki eleman sayisi: "))
    t = int(raw_input("Istenen benzerlik orani: "))
    r = int(raw_input("Blok boyu: "))

    filePath = str(raw_input("Verilerin kaydedilecegi dosya yolu: "))

    chooseS = (s * r) / 100
    chooseT = (t * r) / 100

    saveHash(filePath, S, r, chooseS)
    start = datetime.datetime.now()
    readhash, ready = readfromfile(filePath)
    algorithmRandom(compS, readhash, ready, chooseS, chooseT, r)
    stop = datetime.datetime.now()

    print "Time elapsed: "
    print stop - start

if __name__ == "__main__":
    main()
```

Ek olarak algoritmanın karşılaştırma kısmında bloklardan sıralı bir şekilde $s+1$ elemanlı listeler oluşturarak listelerden rastgele bir liste seçildiği ve polinomun bu listenin elemanları ile oluşturulduğu hali de gerçeklenmiştir. Yani örneğin karşılaştırılacak küme $S' = \{0,1,2,3,4,5,6,7,8,9\}$, $r = 5$, $s = 2$ olsun. Bu durumda oluşacak bloklar

$$S'_1 = \{0,1,2,3,4\} \text{ ve } S'_2 = \{5,6,7,8,9\} \tag{4.12}$$

olur. S'_1 bloğu için polinom oluştururken $s+1$ elaman seçimi için önce

$$\{0,1,2\}, \{1,2,3\}, \{2,3,4\} \tag{4.13}$$

şeklinde listeler oluşturulur. Daha sonra bu listeler arasından rastgele bir liste seçilir. Seçilen listenin elemanlarından *s* dereceli polinom oluşturulur. Bu polinomun özet değeri alınır ve kayıtlı özet değeri ile karşılaştırır. Eğer özet değerleri eşleşirse polinom üretmek için kaydedilen *y* değerlerinin dışında kalan *y* değerleri arasında *%t* eşleşmenin olup olmadığında bakılır. Eğer eşleşme sağlanmazsa polinom oluşturup karşılaştırma işlemi deneme sayısı kadar devam eder. Bloklardan sıralı liste seçerek polinom üreten algoritma kodu aşağıda paylaşılmıştır.

```
import random
import hashlib
import json
from decimal import Decimal
import datetime


def getCoeffs(x, y, q):
    n = len(y)
    coef_pyramid = createMatrix(n, n)
    coef_pyramid[0] = y
    for i in range(0, n):
        for j in range(n - i - 1):
            asd = inverse_mod(x[j + i + 1] - x[j], q)
            coef_pyramid[i + 1][j] = Mod((coef_pyramid[i][j + 1] - coef_pyramid[i][j]) * asd, q)
    coef_pyramid_new = []
    for j in range(0, n):
        coef_pyramid_new.append(coef_pyramid[j][0])
    return coef_pyramid_new


def createMatrix(rowCount, colCount):
    mat = []
    for i in range(rowCount):
        rowList = []
        for j in range(colCount):
            rowList.append(0)
        mat.append(rowList)

    return mat


def newtonInt(x, y, q):
    coefVector = getCoeffs(x, y, q)
    # modq da polinom cismi olusturma
    K = Zmod(q)
    P = PolynomialRing(K, implementation='NTL', names=('t',));
    (t,) = P._first_ngens(1)

    n = len(coefVector)

    finalPol = 0
    for i in range(n):
        p = 1
        for j in range(i):
            p_temp = t - x[j]
```

```
            p = p * p_temp
        p *= coefVector[i]
        finalPol = finalPol + p
    return finalPol


# Kumeyi lenS elemanli bloklara ayirir.
def splitBlock(compS, lenS):
    newCompS = []
    up = lenS
    down = 0
    plus = lenS
    u = int(len(compS) / lenS)
    for j in range(1, u + 1):
        newCompS.append(compS[down:up])
        down = up
        up = up + plus
    return newCompS


# Bloklardan S+1 eleman olacak sekilde listler olusturur.
def splitBlock2s(blockS, lens):
    newCompS = []
    n = len(blockS)
    for i in range(0, n - lens + 1):
        newCompS.append(blockS[i:i + lens])
    return newCompS


def getRandom(n):
    L = GF(q)
    y = []
    for i in range(0, n):
        y.append(L.random_element())
    return y


def write2file(filepath, hashpoly, newy):
    newyy = json.loads(str(newy), parse_float=Decimal)
    dict_object = []
    for i in range(0, len(hashpoly)):
        dict_object.append(dict(hashvalue=hashpoly[i], yvalue=newyy[i]))
    with open(filepath, "w") as f:
        for line in dict_object:
            json.dump(line, f)
            f.write("\n")


def readfromfile(filepath):
    data = []
    ready = []
    readhash = []
    with open(filepath, "r") as f:
        for line in f:
            data.append(json.loads(line))
    for i in range(0, len(data)):
        ready.append(data[i]["yvalue"])
        readhash.append(data[i]["hashvalue"])
    return readhash, ready
```

```
# saveHash fonksiyonu orjinal S kumesini bloklara bolup, her blok icin
# rasgele s+1 boyunda list secer, polinom hesaplatir
# ve y degerlerini kaydeder.
def saveHash(filePath, S, r, bigS):
    hashPoly = []
    newy = []
    splittedS = splitBlock(S, r)
    n = len(splittedS)
    a = bigS + 1
    for i in range(0, n):
        savey = []
        kk = 0
        for ii in range(0, r):
            savey.append(0)
        newS = []
        hashF = hashlib.sha1()
        rand_items = random.sample(splittedS[i], int(a))

        y = getRandom(a)

        for rand_item in rand_items:
            position = splittedS[i].index(rand_item)
            savey[position] = y[kk]
            kk += 1

        print "rand_items: \n"
        print rand_items
        poly = newtonInt(rand_items, y)
        hashF.update(str(poly))
        hashPoly.append(hashF.hexdigest())

        for el in splittedS[i]:
            if el not in rand_items or splittedS[i].count(el) > 1:
                newS.append(el)
        for el in newS:
            position = splittedS[i].index(el)
            savey[position] = poly(el)
        newy.append(savey)

    write2file(filePath, hashPoly, newy)


# bu algoritma bloktan rasgele s+1 list secip polinom uretir
# ve hash degeri hesaplatr.
def algorithmList(comS, savedhash, savedy, bigS, bigT, r):
    splitcoS = splitBlock(comS, r)  # comS yi r elemanli bloklara boler.
    n = len(splitcoS)
    print "kac blok var: " + str(n)
    a = bigS + 1  # S+1 sayisini belirler
    passedblock = 0.

    for i in range(0, n):

        splitblocked = splitBlock2s(splitcoS[i], a)
        trail = r - bigT

        # splitblocked icerisinde S+1 elamanli listler bulunur.
```

```
        # Bu listlerden random olarak secilip polinom hesaplanir
        # ve polinomun ozet degerinin eslesip eslesmedigine bakilir
        # bu dongu en fazla trail kere tekrarlanir.
        while trail != 0:
            sim = a
            coy = []
            newcoS = []
            randcos_items = random.sample(splitblocked, 1)
            for randcos_item in randcos_items:
                hashF = hashlib.sha1()

                for el in splitcoS[i]:
                    if el not in randcos_item or splitcoS[i].count(el) > 1:
                        newcoS.append(el)

                for randcositem in randcos_item:
                    position = splitcoS[i].index(randcositem)
                    coy.append(savedy[i][position])

                coPoly = newtonInt(randcos_item, coy)
                hashF.update(str(coPoly))

                if hashF.hexdigest() == savedhash[i]:
                    for el in newcoS:
                        if coPoly(el) in savedy[i]:
                            sim += 1

                else:
                    trail -= 1

            if sim >= bigT:
                print "THIS_BLOCK_PASS"
                trail = 0
                passedblock += 1


    print "eslesen_blok_sayisi:_" + str(int(passedblock))
    print "eslesmeyen_blok_sayisi:_" + str(int(n - passedblock))


def main():
    S = list(map(int, raw_input("Orjinal_kume:_").split(',')))
    compS = list(map(int, raw_input("Karsilastirilacak_kume:_").split(',')))

    q = int(raw_input("Sonlu_cisim_icin_q_sayisi:_"))

    s = int(raw_input("Saldirganin_elindeki_eleman_sayisi:_"))
    t = int(raw_input("Istenen_benzerlik_orani:_"))
    r = int(raw_input("Blok_boyu:_"))

    filePath = str(raw_input("Verilerin_kaydedilecegi_dosya_yolu:_"))

    chooseS = (s * r) / 100
    chooseT = (t * r) / 100

    saveHash(filePath, S, r, chooseS)
    start = datetime.datetime.now()
    readhash, ready = readfromfile(filePath)
    algorithmList(compS, readhash, ready, chooseS, chooseT, q)
```

```
    stop = datetime.datetime.now()

    print "Time elapsed: "
    print stop - start

if __name__ == "__main__":
    main()
```

5. SONUÇ VE ÖNERİLER

Kullanılan algoritma, orjinal kümeye ait veriler yerine bu verilerden üretilen polinomun özet değerini saklayarak ISO/IEC FCD 24745'de geçen tek yönlülük özelliğini sağlar. Kayıtlı olan değerlerden biyometrik veriye ulaşmak isteyen saldırganın elinde orjinal kümeye ait hiçbir veri yoksa

$$2^{(S+T+1)*8*(n/r)} \tag{5.1}$$

deneme yapması gerekir. Biyometrik verilerin ölçümleri farklı gelebileceğinden ve polinomları üretmek için kullanılan y değerlerinin büyük bir sayı olan q için $\mathbb{F}_q$'dan rasgele seçildiğinden farklı veri tabanlarındaki bu verilerden üretilen polinomların özet değerlerinin çakışması zor olabilir ancak saldırgan polinomların özet değerlerinin ve y değerlerinin kayıtlı olduğu veri tabanlarından eşleşen özet değerleri yakaladığında y değerlerinde çakışma yoksa kaba kuvvet için gerekli olan kombinasyon sayısını

$$2^{T*8*(n/r)} \tag{5.2}$$

değerine düşürür. y değerlerinde çakışma varsa kaba kuvvet için gerekli kombinasyon sayısı daha da düşebilir. Bu yüzden polinomun özet değeri oluşturulurken her veri tabanı farklı bir ek parametre (salt) kullanırsa biyometrik verinin kullanıldığı veri tabanları için eşleşecek özet değerleri polinomları üretmek için kullanılan y değerleri aynı geldiği durumda bile farklı olacaktır. Bu durumda ISO/IEC FCD 24745'de geçen ilişkilendirememe özelliği de sağlanmış olur. Algoritmanın işleyişinde orjinal kümeden rastgele $s+1$ eleman seçilerek polinom üretildiğinden algoritma, saldırganın orjinal küme elemanlarından s tanesini bildiği durumda bile güvenlik sağlar. Çünkü s.dereceden bir polinom üretilebilmesi için en az $s+1$ noktaya ihtiyaç vardır. Ayrıca algoritma 200 elemanlı bir küme için mikrosaniyeler (saniyenin bir milyonda biri) içinde karşılaştırma yapar. Orjinal algoritmaya ek olarak yazılan algoritma kodunda karşılaştırma işlemi daha hassastır çünkü listelerden rastgele bir liste seçildiğinden

bloklarda eşleşmeyen eleman olduğu zaman o elemanın bulunduğu liste birden fazla olacağından rastgele liste seçildiğinde farklı elemanların bulunduğu listelerin gelme olasılığı daha fazladır. Ayrıca algoritmaya ek olarak yazılan algoritma kodunda da mikrosaniyeler içinde karşılaştırma yapılır.

KAYNAKLAR

[1] **Paar, C. ve Pelzl, J.** (2009). *Understanding cryptography: a textbook for students and practitioners*, Springer Science & Business Media.

[2] **Url-2**, `https://searchsecurity.techtarget.com/definition/authentication`.

[3] **Rathgeb, C. ve Uhl, A.** (2011). A survey on biometric cryptosystems and cancelable biometrics, *EURASIP Journal on Information Security*, *2011*(1), 3.

[4] **Soutar, C.**, **Roberge, D.**, **Stoianov, A.**, **Gilroy, R. ve Kumar, B.V.** (1998). Biometric Encryption: enrollment and verification procedures, *Optical Pattern Recognition IX*, cilt3386, International Society for Optics and Photonics, s.24–35.

[5] **Soutar, C.**, **Roberge, D.**, **Stoianov, A.**, **Gilroy, R. ve Kumar, B.V.** (1998). Biometric encryption using image processing, *Optical Security and Counterfeit Deterrence Techniques II*, cilt3314, International Society for Optics and Photonics, s.178–188.

[6] **Soutar, C.**, **Roberge, D.**, **Stojanov, S.**, **Gilroy, R. ve Kumar, B.V.**, (1999), Biometric Encryption, in ICSA Guide to Cryptography, R. K. Nichols, Ed.

[7] **Juels, A. ve Wattenberg, M.** (1999). A fuzzy commitment scheme, *Proceedings of the 6th ACM conference on Computer and communications security*, ACM, s.28–36.

[8] **Juels, A. ve Sudan, M.** (2006). A fuzzy vault scheme, *Designs, Codes and Cryptography*, *38*(2), 237–257.

[9] **Scheirer, W.J. ve Boult, T.E.** (2007). Cracking fuzzy vaults and biometric encryption, *2007 Biometrics Symposium*, IEEE, s.1–6.

[10] **Bodo, A.** (1994). Method for producing a digital signature with aid of a biometric feature, *German patent DE*, *42*(43), 908.

[11] **Davida, G.I.**, **Frankel, Y. ve Matt, B.J.** (1998). On enabling secure applications through off-line biometric identification, *Proceedings. 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186)*, IEEE, s.148–157.

[12] **Davida, G.I.**, **Frankel, Y.**, **Matt, B. ve Peralta, R.** (1999). On the relation of error correction and cryptography to an online biometric based identification scheme, *Workshop on coding and cryptography*, Citeseer.

[13] **Özdemir, E.**, **Özkirişçi, N.A. ve Saçma, O.** Comparing two data sets, *yazım aşamasında*.

[14] **Eastlake, D. ve Jones, P.**, (2001), US secure hash algorithm 1 (SHA1).

[15] **Cheney, W. ve Kincaid, D.** (2008). Numerical mathematics and computing. Thompson Learning, *Inc., Belmont*.

[16] **Url-1**, `https://www.top500.org/lists/2019/06/`.

[17] **Url-3**, `http://www.sagemath.org`.

[18] **Url-4**, `https://www.python.org/`.

ÖZGEÇMİŞ

Ad Soyad: Aslıhan AKPINAR

Doğum Tarihi ve Yeri: 17.01.1994/ANKARA

ÖĞRENİM DURUMU:

- **Lisans:** 2016, Gazi Üniversitesi, Fen Fakültesi, Matematik Bölümü