Parolalar Nasıl Saklanmalı 101 & Şifreleme (Encryption)Dünyası | MDISEC Neler Anlattı #1

İLKER YILMAZ
8 min read · Sep 8

> *Hayatımız ve Parolalar*

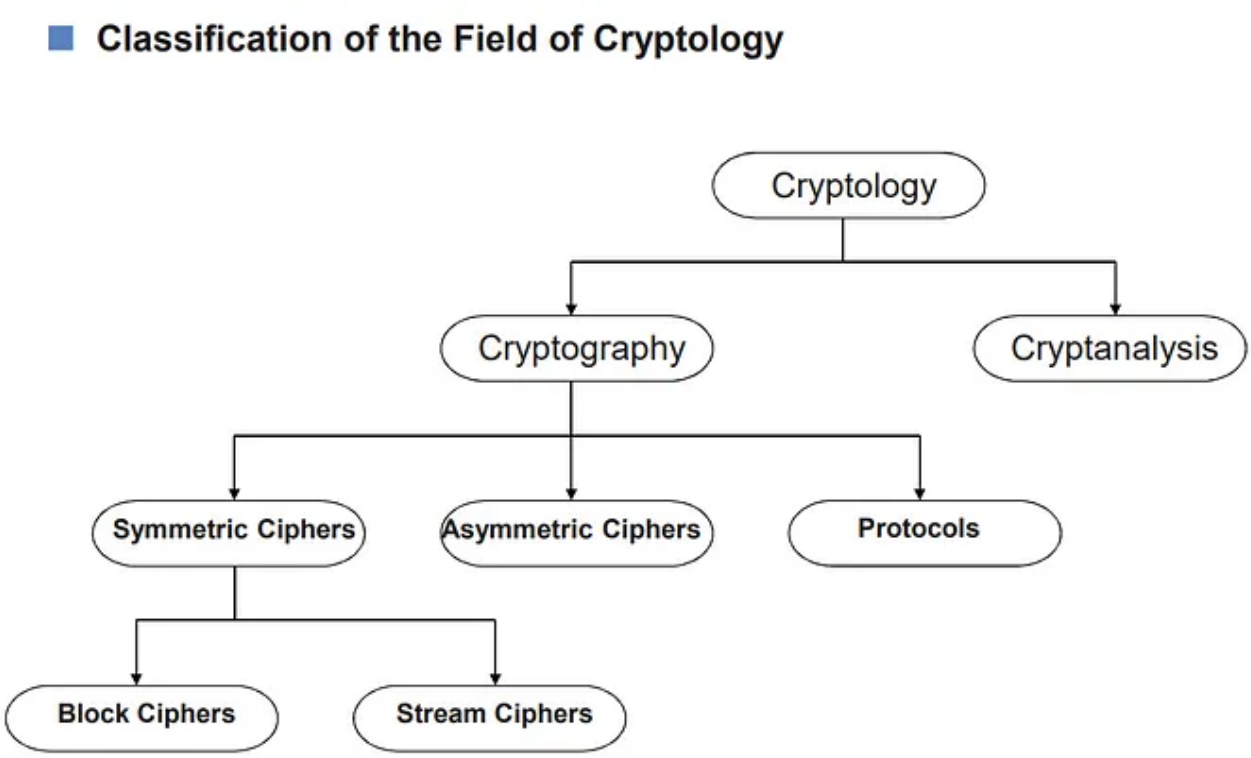
Hepimiz günlük yaşantımızda onlarca sisteme üye olup bu hizmetleri aktif bir şekilde kullanmaktayız. Genel olarak neredeyse tüm bilgilerimizi bu sistemler üzerinde yönetip hayatımızı idame ettirmeye çalışmaktayız. Kullandığımız bu hizmet ve platformların büyük bir çoğunluğunda ise giriş için kullandığımız bir kullanıcı adı ve parola meselesi mevcut. Sisteme kayıt olurken kullandığımız parola ve kullanıcı adımız daha sonra tekrar geldiğimizde sistemin bizi tanıması için kaydedilmekte ve bir veritabanında tutulmaktadır. Peki bu kullanıcı adı ve parolalarımız nasıl saklanmalı?

Kullanıcı Adı ve Parolalar Veritabanlarında Nasıl Saklanmalı ?

Hepinizin de bildiği üzere bu sorunun cevabı tabii ki parolaların gizli bir şekilde tutulmasıdır. Dolayısıyla burada devreye şifreleme giriyor. Gelin hep birlikte genel olarak şifreleme dünyasına bakalım ve daha sonra bizim neyi tercih ettiğimizi konuşalım.

Şifreleme (Encryption) Dünyasına Genel Bakış

Şifreleme dünyasına genel olarak baktığımız zaman aşağıdaki gibi bir sınıflandırmayı rahatlıkla yapabiliriz. Kriptoloji genel olarak kriptografi (cryptography) ve kriptanaliz (cryptanalysis) olmak üzere ikiye ayrılır. Kriptogrofi tarafında amaç şifreleme algoritmaları geliştirip verileri şifrelemek iken kriptanaliz tarafında ise bu şifreleme algoritmalarını matematiksel olarak inceleyip güvenlik problemlerini tespit etmektir.



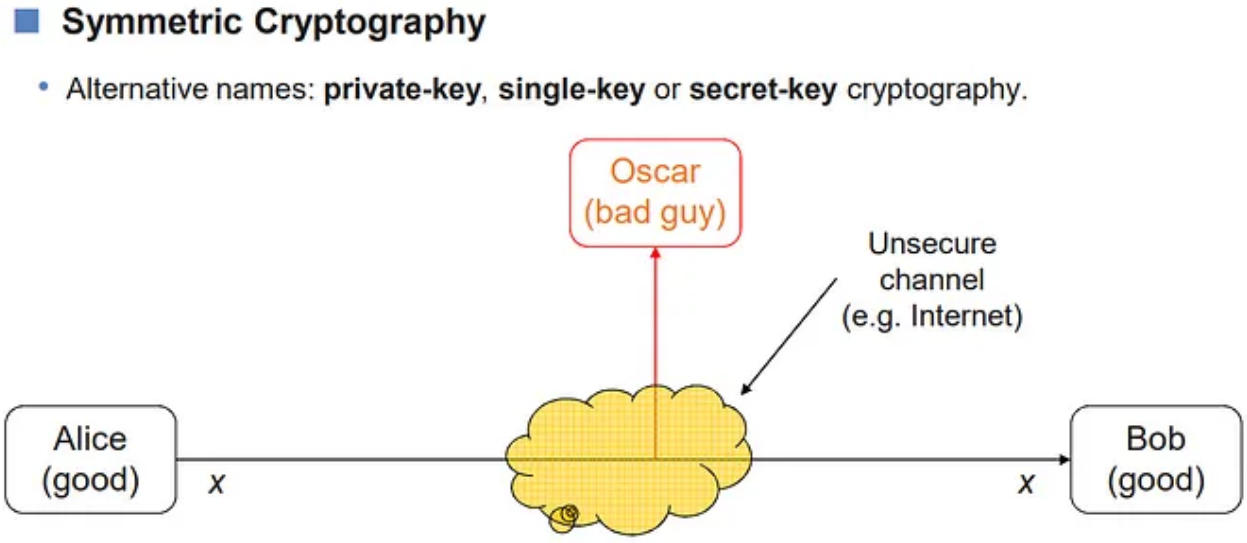
Kriptografi bilimi de kendi içerisinde simetrik (symmetric) ve asimetrik (assymetric) şifreleme olarak ikiye ayrılmaktadır. ikisinde de temel amaç bilgi güvenliğini sağlamaktır ancak ihtiyaçlarımıza göre farklı özelliklere sahiplerdir. Daha sonra simetrik şifreleme algoritmaları da kendi içerisinde ikiye ayrılmaktadır.

Eskiden beri, geçmişten günümüze insanoğlunun ihtiyacı olan şey bilginin güvenliğini sağlamaktır. Yıllar önce henüz bilgisayar dünyası ve internet

altyapısı henüz bu kadar gelişmemişken de hayatımızda korumak ve saklamak istediğimiz çeşitli şeyler mevcuttu. Bugün de bunların yerini artık dijital verilerimiz almış oldu. Günümüzde de bunlara yönelik geliştirilen sistemler ve algoritmalar mevcuttur.

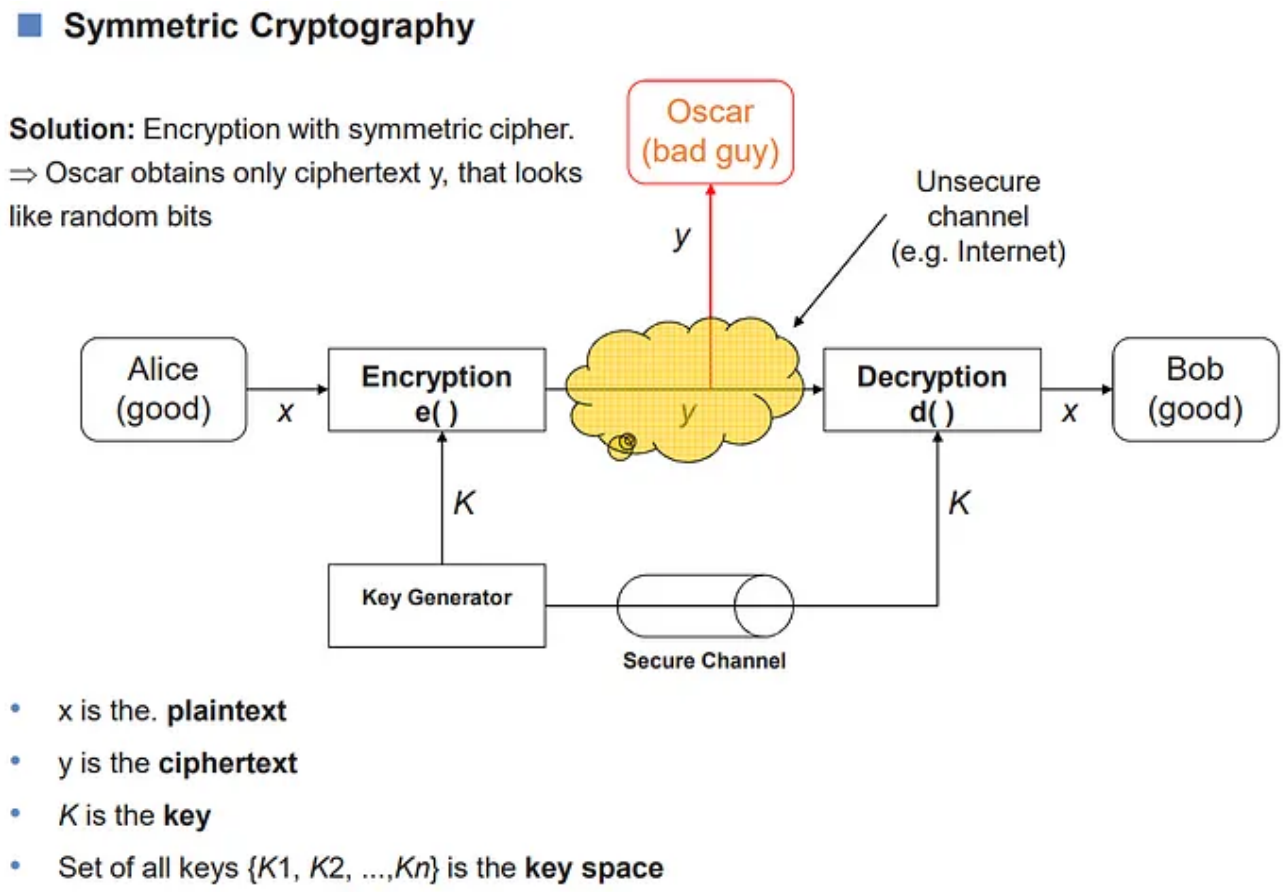
Simetrik (Symmetric) Kriptoloji

Temel problemimiz iki tarafın güvenli bir şekilde haberleşmesi. Kendi aralarında gizli bir şekilde haberleşmek isteyen iki taraf var ve ortada bunun için sağlanan gübenli bir yol bulunmamakta. Dolayısıyla iletişimde kullanılan veriler güvensiz bir ortamda gönderiliyor. Bu mesajı göndermenin bir yolunu aramamız gerekiyor ancak burada farklı tehlikeler de bulunmakta. Çünkü bu ortamda sadece iki taraf yok ve bu iletişimi görebilecek olan güvensiz bir taraf da bulunmakta.

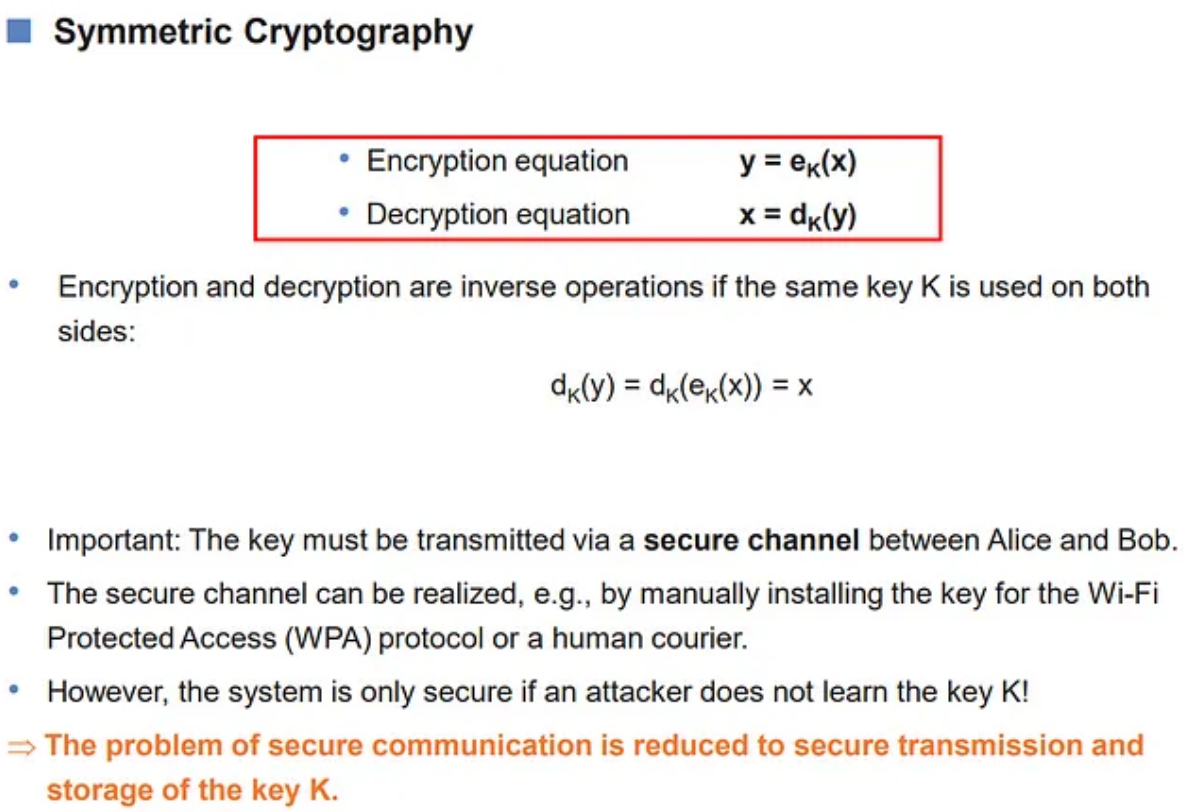


Burada yapmamız gereken şey gönderilmek istenen mesajın şifrelenerek gönderilmesidir. Mesajımızı bir şifreleme fonksiyonundan geçirerek farklı bir mesaja dönüştürmüş oluyoruz. Bu sayede göndermek istediğimiz X verisi kimsenin anlamayacağı bir Y verisine dönüşmüş olacaktır. Alıcı ise burada

şifrelerken kullandığımız anahtarı kullanarak tekrar X verisini elde edebilmektedir. Bu gizliliğin sağlanması için anahtar değerinin de güvenli bir şekilde ortaya iletilmesi gerekiyor. Görebileceğiniz üzere artık yeni problemlerimiz mevcut. Dolayısıyla burada farklı senaryolar ve zafiyetler de ortaya çıkabilmektedir.



Tıpkı evimizin kapısı gibi düşünecek olursak anahtarı sağa çevirdiğimizde kapıyı kilitleyip daha sonra açmak istediğimizde ise sola çevirince kapıyı açmış oluyoruz. Burada da kullandığımız şifreleme algoritmalarında öncelikle verileri bir anahtar değeriyle şifreleyip daha sonra bu anahtar değerimizle verilerin şifresini çözebilir hale geliyoruz.

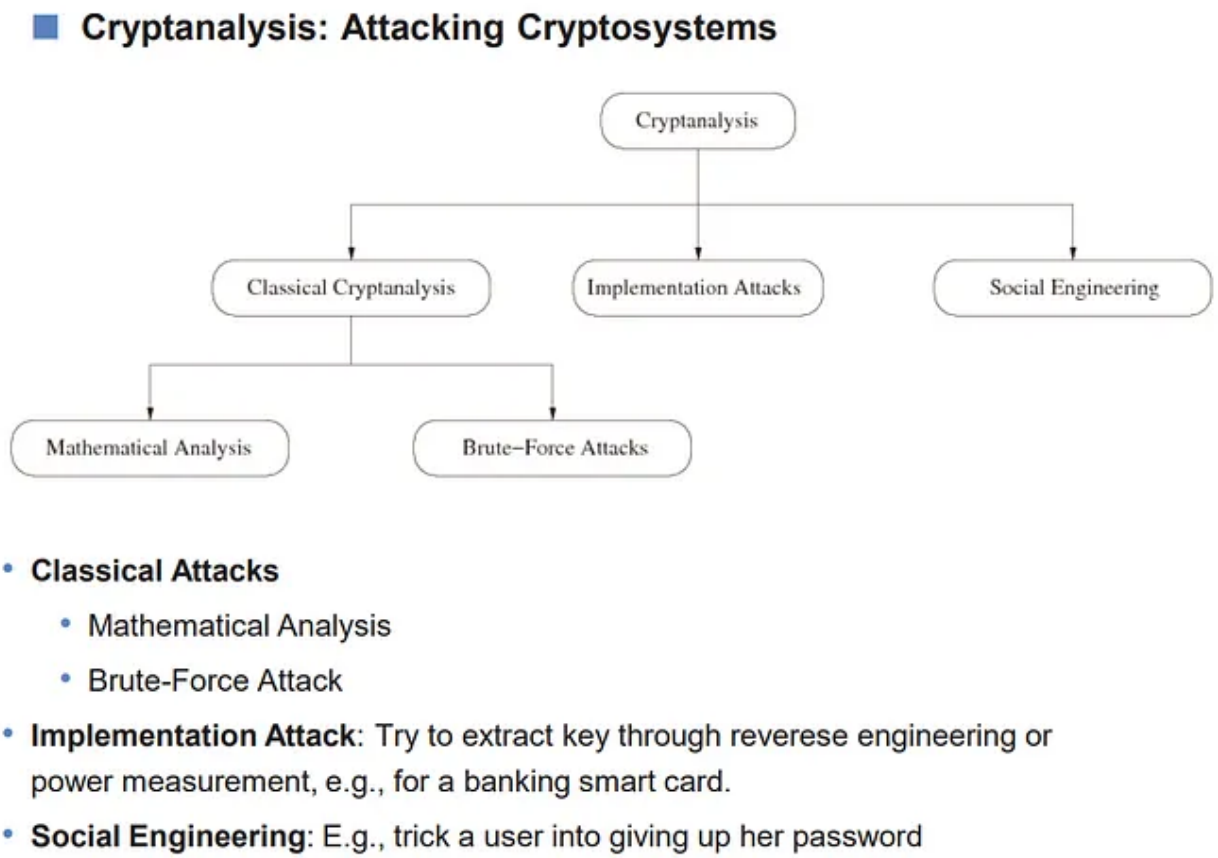


Dolayısıyla anahtarımız da mutlaka güvenli bir kanal üzerinden gönderilmeli. Onun haricinde anahtara sahip olmayan biri eve girememekle birlikte brute-force ya da farklı yöntemler deneyerek kapıyı açmaya çalışacaktır. Bizim de sistemi buna karşı güçlendirmemiz gerekmektedir. Dolayısıyla buradan sonra artık problemimiz anahtar değerinin güvenli bir kanal üzerinden gönderilmesi problemine dönüşmekte ve üzerine çalışmalar yapılması gerekmektedir. Bu da simetrik şifrelemenin ötesinde farklı konulara dikkat ederek yapılması gereken bir şeydir. Tüm bu sistem özet olarak simetrik, gizli anahtarlı, tek anahtarlı şifreleme sistemi olarak kabul edilmektedir.

Kriptanaliz

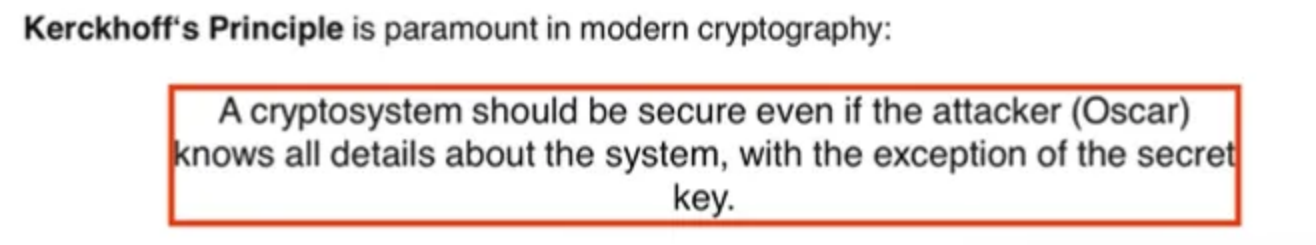
Bu aşamalardan sonra ihtiyacımız olan şey kriptanalizdir. Kullanılan herhangi bir şifreleme algoritmasının matematiksel olarak bir ispatı yoksa bu durumda yapılması gereken tek şey onu kırmaktır. Güvenliği hakkında

sürekli bir soru işareti kalmış olur. Kriptanaliz de burada devreye girmektedir. Bir şifreleme algoritmasını kullanacaksanız onun matematiksel ispatının olup olmadığıyla ilgilenmelisiniz.



Kerckhoff Prensibi

Tüm bunlara baktıktan sonra devreye giren bir prensip de Kerckhoff Prensibidir. Kerckhoff Prensibine göre bir şifreleme sistemi sadece ve şu durumda güvenli olabilir; saldırganın her şeye sahip olduğunu ve sadece algoritmanın gizli anahtarını bilmediğini düşünelim. Bu durumda şifrelemeyi çözemiyorsa bu algoritmanın güvenli olduğunu düşünebiliriz.



Dolayısıyla bir şifreleme algoritmasının güvenliği onun gizliliğiyle sağlanamayacaktır. Detayları gizleyerek bu algoritmanın güvenliğini sağlayamayız.

Peki tüm bunları neden anlattık ? Sözün özü şu ki amacımız şuan sadece kullanıcının parolasını güvenli bir şekilde depolamak. Dolayısıyla anlayacağınız üzere kullanıcının parolasını bizim açık bir şekilde görmemize gerek yok. Demem o ki bu parolayı yukarıda anlattığımız şifreleme yöntemleri yerine geri döndürülemez bir formata çevirirsek bizim için daha iyi olacaktır. Çünkü şifrelemede kullandığımız anahtar değerinin güvenliğini sağlamakla uğraşmak istemiyoruz. Burada da yardımımıza hashing kavramı yetişiyor. Şimdi genel olarak hashing kavramından ve bizim bunu nasıl kullanacağımızdan bahsedelim...

Hashing Nedir ?

Hashing aslında bir özetleme fonksiyonudur. Aldığı bilgiyi daha küçük bir formata dönüştürerek sunmasıdır. Büyük bir mesajı küçük bir mesaja sığdırmaktır. Hashing algoritmaları şifrelemede de kullanılabilmektedir.

Dolayısıyla parolalarımızı bu hashing algoritmalarından geçirerek açık bir şekilde tutmak yerine hash'lenmiş bir formatta tutabilmekteyiz.

Hashing algoritmaları tek yönlü çalıştıkları için ortaya çıkan sonuçtan geriye dönüş mümkün değildir. Ancak aldığımız kullanıcı adı ve parolamızı tekrar aynı algoritmadan geçirdiğimizde ortaya çıkan sonuç veritabanındaki sonuç ile aynı olacağı için karşılaştırmayı doğru bir şekilde yapabilmekteyiz. Eğer doğru parola girildiyse veritabanındaki hash'lenmiş sonuç ile uyuşacaktır ve sisteme giriş yapılacaktır.

Bu tek yönlü oluşu biraz detaylandırabiliriz. Kriptoloji'de en önemli noktalardan biri de modüler aritmetiktir. Burada modüler aritmetiğin neden bu kadar güçlü olduğunu değerlendirelim.

Modüler Aritmetik

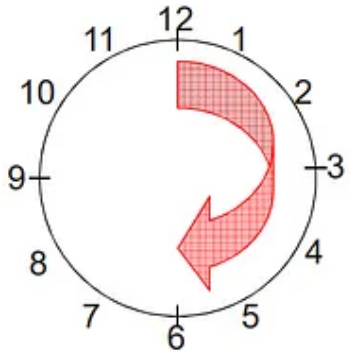
Aşağıdaki görsel üzerinden de görebileceğimiz üzere herhangi bir değerin mod'unu aldığımızda ortaya çıkan sonuç bizim için bir değer oluşturmaktadır. Ortaya çıkan bu değer üzerinden geriye dönmek istediğimizde ise büyük bir olasılık denizi ile karşılaşmaktayız. Dolayısıyla hangi değerin bu sonucu verdiğini bulmak oldukça zorlaşmakta.

Short Introduction to Modular Arithmetic

Generally speaking, most cryptosytems are based on **sets of numbers** that are

1. **discrete** (sets with integers are particularly useful)
2. **finite** (i.e., if we only compute with a finiely many numbers)

Seems too abstract? --- Let's look at a finite set with discrete numbers we are quite familiar with: a clock.



Interestingly, even though the numbers are incremented every hour we never leave the set of integers:

1, 2, 3, … 11, 12, 1, 2, 3, … 11, 12, 1, 2, 3, …:

Örneğin 76 sayısının mod10'a göre sonucuna baktığımıza 6 sonucuna ulaşmaktayız. Ancak 6 sonucunun nereden geldiğini bulmak istediğimizde ise karşımıza 6, 16, 26, 36, … , 10000000006 gibi çok büyük bir olasılık denizi çıkmaktadır. Yani kalan tekil (unique) olmamış oluyor. Bu kalanı farklı sayılar ile elde edebilmekteyiz. Hem eksi hem de artı yönde sonsuz bir anahtar uzayı oluşmaktadır. Dolayısıyla anahtar kümemiz çok geniş olduğu için deneme-yanılma, kaba kuvvet (brute-force) yöntemiyle kolay bir şekilde sonuca ulaşamamaktayız. Hash almak çok kolay olmasına rağmen geriye dönmek pratik olarak mümkün değildir...

Kısa bir özet yapacak olursak sistemimize giriş yapmak isteyen kullanıcı, kullanıcı adı ve parolası ile kayıt olup giriş yaptı. Kullanıcının parolasını

hash algoritmamızdan geçirerek kimsenin anlamayacağı bir formata çevirip veritabanındaki tablolarımızda tutar hale geldik. Artık kullanıcı yeniden geldiğinde parolasını tekrar girecek ve biz hash algoritmamızdan geçirdikten sonra gerekli karşılaştırmayı yapıp aksiyon alacağız. Doğru parola girildiğinde kullanıcıyı içeri almış olacağız. Buraya kadar her şey güzel olmakla birlikte pek yeterli olmamaktadır. Gelin hep birlikte bunun sebeplerini inceleyelim.

Bizleri Karşılayan Yeni Problemler ve Çözümleri

Kullanıcının parolasını aldıktan sonra sadece hash algoritmalarından geçirip bu şekilde veritabanına kaydetmek maalesef yeterli olmayacaktır. Günümüzde **computational power** diye bir ifade vardır. Yani hesaplama gücü. Artık bilgisayarların oldukça kapsamlı ve güçlü hesaplama güçleri mevcut. Dolayısıyla aldığımız önlemlerin de bu oranda güçlendirilmesi gerekmektedir.

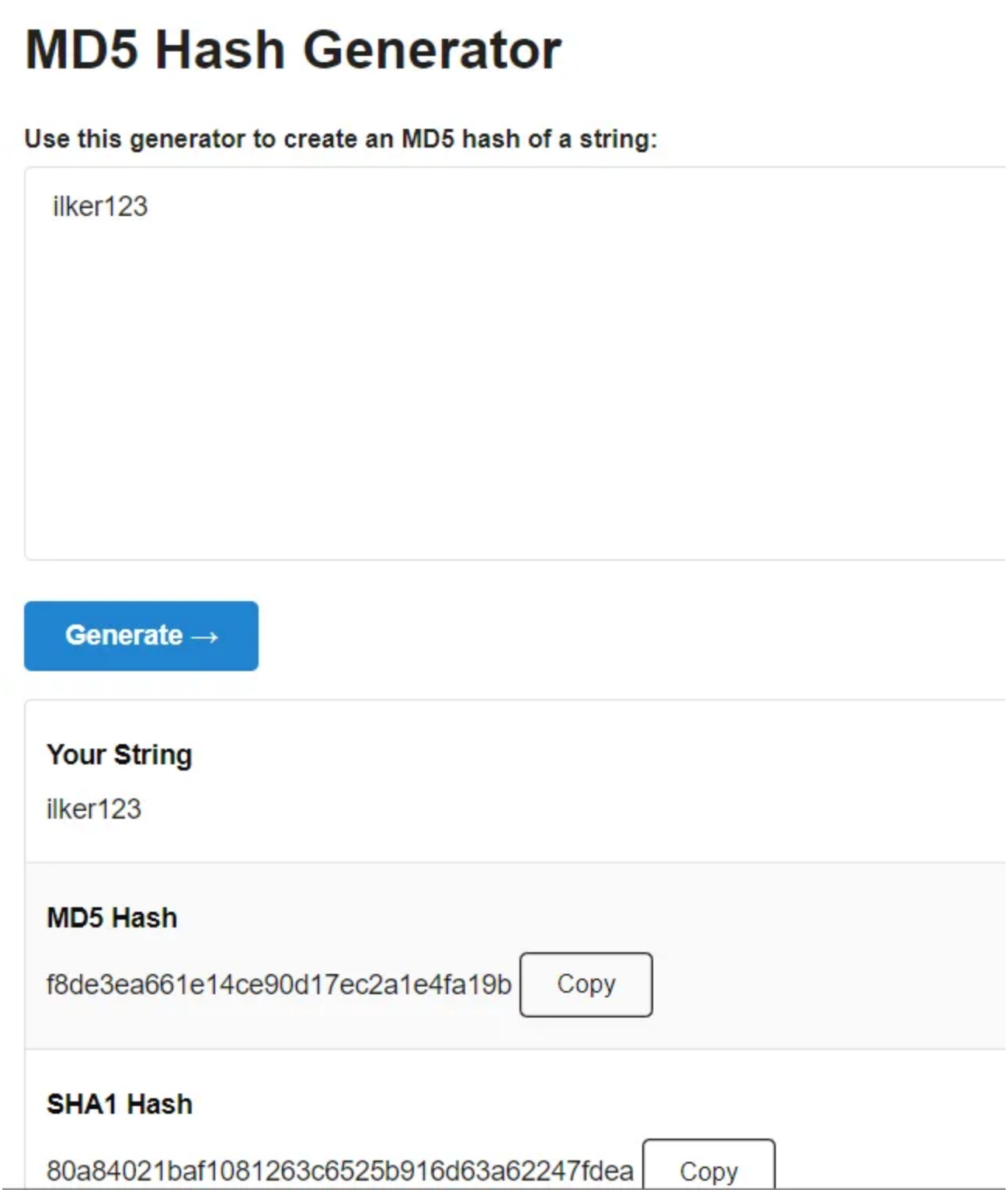
Şimdi bir web sitesinin kullanıcılar tablosunu elde etmiş biri için konuşalım. Kullanıcıların hash'lerinin elimizde var olduğunu düşünelim. Aynı zamanda bu web sitesi için bir parola politikasının olduğunu da biliyoruz. Örneğin büyük harfle başlayıp, özel karakter ve sayı da içermesi gibi kuralları olduğunu düşünelim. Bu bilgiler elimizdeyken akıllıca yöntemler kullanarak brute-force yapabiliriz. Oldukça yaygın olarak kullanılan ve bilinen bir şifre kırma aracı olan **Hashcat** sayesinde bahsettiğimiz senaryoları çok kolay bir şekilde tasarlayıp brute-force yapabiliriz. Güçlü ekran kartı ve işlemciler sayesinde de çok fazla sayıda denemeler yapıp sonuca ulaşabiliriz.

Ayrıca sitemize üye olan kullanıcılar aynı şifreyi kullanıyor olabilir. Siz de kendi adınıza düşünecek olursanız eğer çok karmaşık bir parola oluşturmadıysanız başka bir kullanıcı ile aynı parolayı kullanma ihtimaliniz

yüksek olabilir. Daha önce herhangi bir şekilde hack'lenmiş olan ve parolaları sızdırılmış olan milyonlarca parola internet üzerinde mevcut. Tüm bunlar rainbow table dediğimiz yapılarda tutularak hizmete hazır bir şekilde sunulmaktadır. Burada da rainbow table'lar ile çok kolay bir şekilde gerekli karşılaştırmalar yapılarak daha önce sızdırılmış olan parolalar bilindiği için hash'lenmiş parolaların karşılığı kolayca bulunabilmektedir.

Aşağıda bu çalışmanın bir örneği mevcuttur

Burada kolay bir parola seçerek hash karşılığını bulabilmekteyiz.



Herkeste olabilecek bir parolanın hash'lenmiş karşılığı

Burada da hash'lenmiş olan parolamızın çok kolay bir şekilde bulunabildiğini görmekteyiz.

Hash'lenmiş parolanın karşılığı

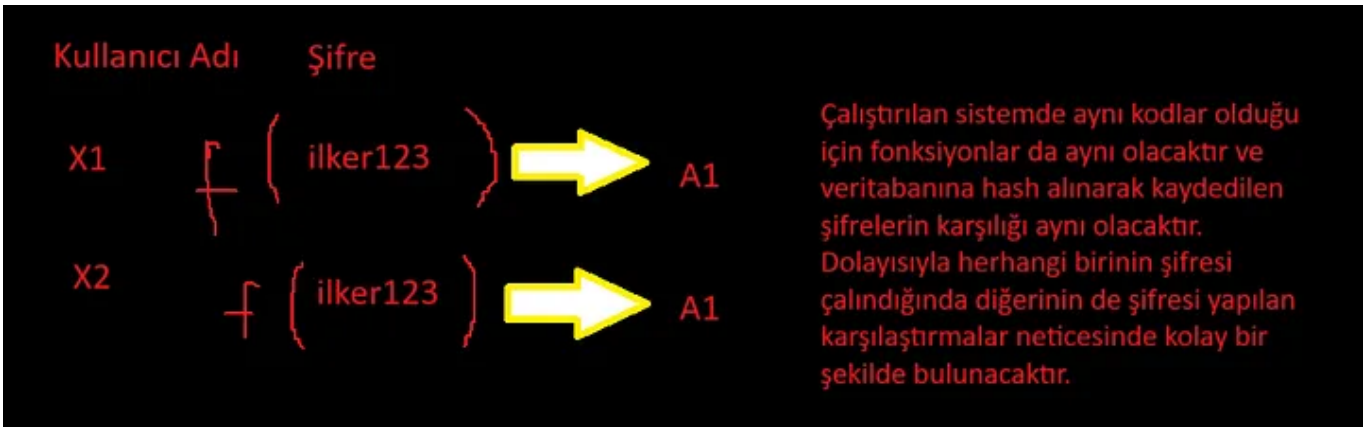
Sanırım buradaki sorunu artık saptayabilmişizdir. Bu sorun kullanıcının daha karmaşık şifreler girmesiyle ya da şifre yöneticilerini kullanmasıyla da uzun vadede çözülmeyecektir. Dolayısıyla sistemi tasarlayanların bir çözüm bulması gerekir.

Salting (Tuzlama) Nedir ?

Öncelikle bir kullanıcının parolasına sahip olduğumuzu düşünelim. Kullanıcı sisteme giriş yaparken parolasına ekleyeceğimiz rasgele değer ile birlikte hash'leyip veritabanında tutarsak artık farklı kullanıcılar aynı parolayı kullansa bile eklediğimiz değerler rasgele ve farklı olacağı için hash sonuçları da birbirinden farklı olacaktır. Dolayısıyla aynı şifreyi kullanan kişilerin hash sonuçları artık aynı olmayacaktır.

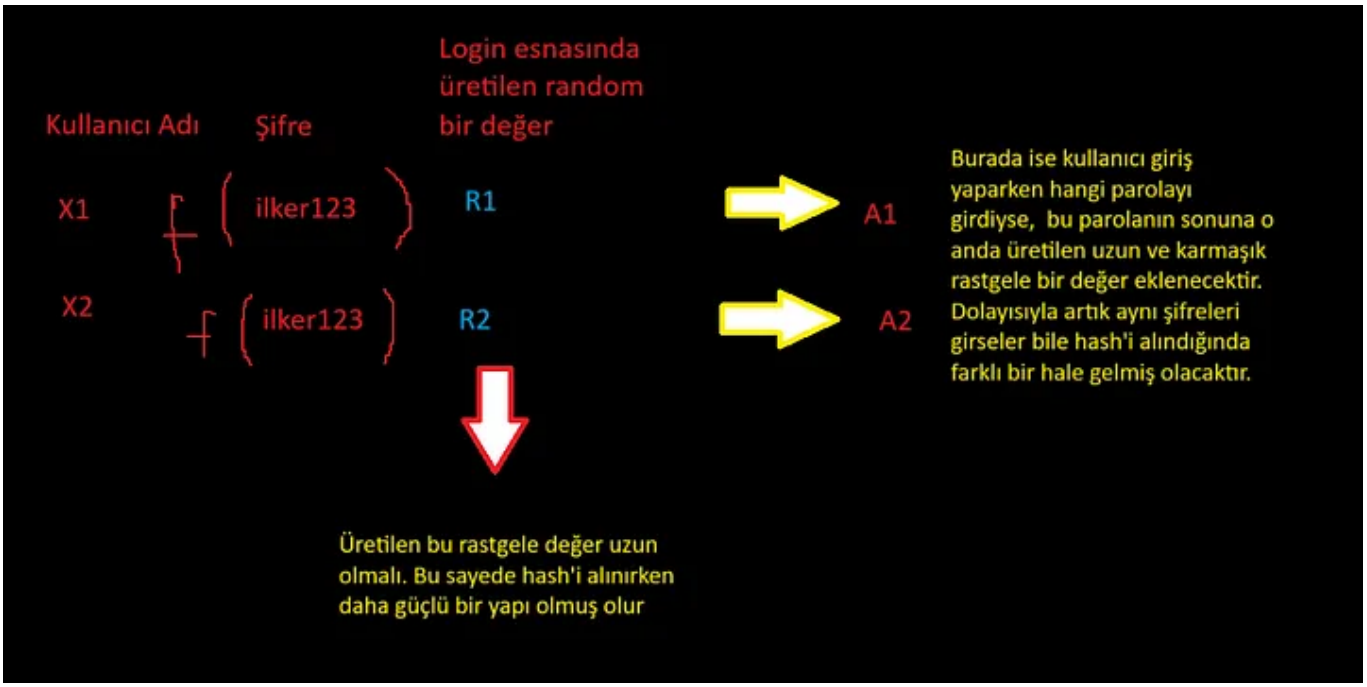
Şimdi bu konuyu biraz detaylandıralım.

Buradaki görselden de görebileceğimiz üzere iki kullanıcı eğer aynı parolayı kullanıyorsa hash algoritmalarından geçirildiğinde tekrar aynı sonucu elde etmiş olacağız. Dolayısıyla bu parolaların bulunabilirliği de artacaktır. Çünkü daha önce başka bir sitede bu parola sızdırılmış ve hash karşılığı biliniyor olabilir.



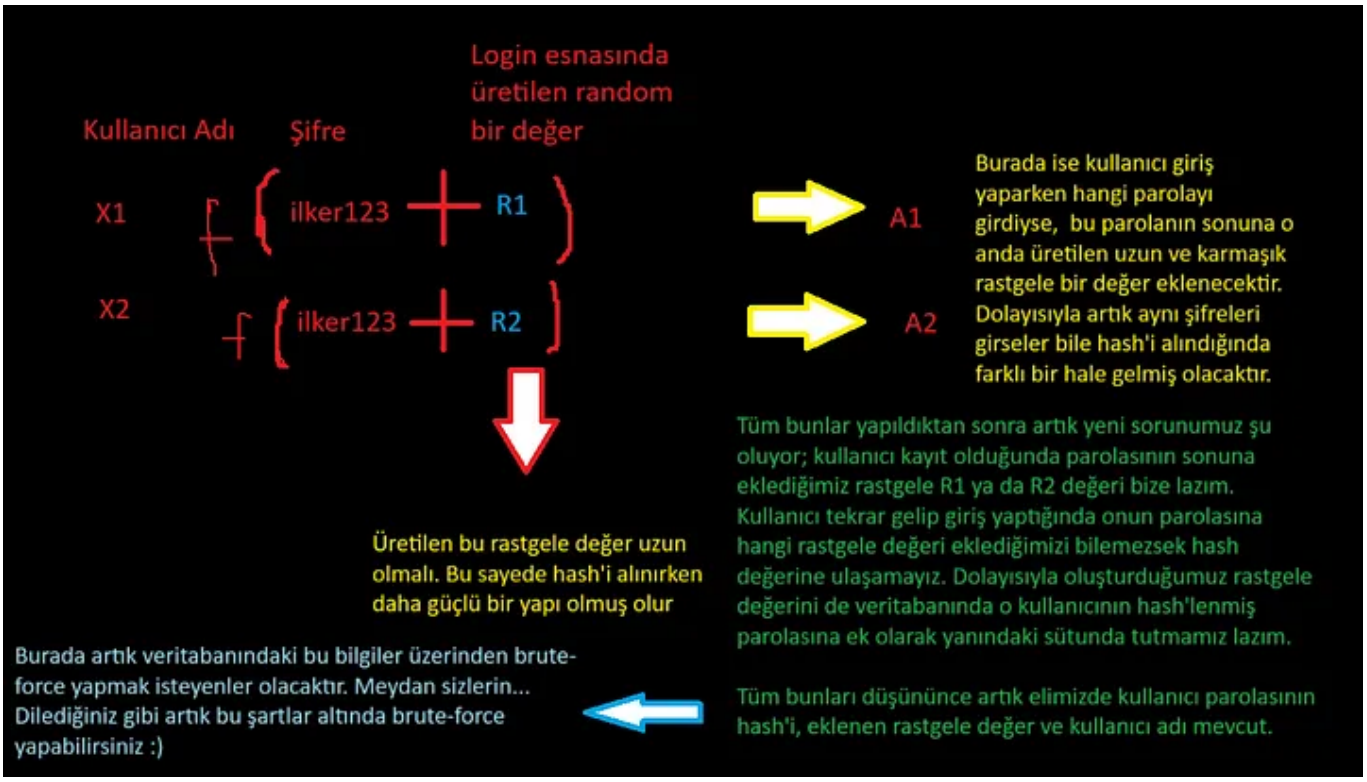
Biz burada kullanıcının giriş yapacağı esnada R1 gibi rasgele ve uzun bir değer üretip parolaya eklemeliyiz. Bu rasgele değer ne kadar uzun ve karmaşık olursa brute-force yapmak da o kadar zorlaşacaktır.

Burada iki parola aynı olsa da artık eklenecek olan değerler; R1 ve R2 olacağı için hash sonuçları birbirinden farklı olacaktır.



Ancak burada da yeni sorunumuz kullanıcı tekrar geldiğinde R1 değerinin bulunması olacaktır. Çünkü R1 değerini kullanıcı giriş yaparken rasgele üretmiştik. Dolayısıyla R1 değerine de daha sonra ihtiyacımız olduğu için

veritabanında tutmalıyız. Burada da bir güvenlik sorunu olacağını düşünebilirsiniz ama artık işler daha da zor bir hale gelmiş oldu. Bizim oluşturduğumuz şartlar altında dileyen kişi brute-force yapabilir. Bu mimaride artık kullanıcı parolaları aynı olsa bile hash sonuçları aynı olmayacağı için kullanıcıların aynı parolaya sahip olduğu bilgisine direkt olarak erişilmemekte. Tek tek denenmesi gerekiyor. Parola + R1 birleştikten sonra hash alınınca ortaya çıkan sonuç ile karşılaştırmalar yapılmalı. R1 değeri de uzun ve karmaşık oldukça hash alma süresi artacaktır. Bu sayede brute-force yapmak da zorlaşacaktır.



Buraya kadar olan kısımda parolaların veritabanlarında nasıl tutulması gerektiğine değindik, hem şifreleme algoritmalarına hem de hashing kavramına açıklık getirmiş olduk. Eklenebilecek daha fazla içerik olmasına rağmen yazının daha fazla uzamaması adına bu kadarının yeterli olduğunu

düşünüyorum. Yorum, öneri ve görüşlerinizi de alt kısımda belirtebilirsiniz. Okuduğunuz için teşekkür eder ve kolaylıklar dilerim 💯💯...

KAYNAKÇA

MDISEC — Parolalar Nasıl Saklanmalı ?

Kriptoloji — Fatih Özkaynak

Hashing — Sadi Evren Şeker

Çeşitli Makaleler ve Bloglar:

The difference between Encryption, Hashing and Salting

Hash Functions

Rainbow Table Attack