Kriptografi

Açık Matematik

Açık Matematik — açık kaynaklı, reklamsız Türkçe matematik notları
acik-matematik.com

Bu sürüm: 20 Eylül 2026

İçindekiler

Giriş

Bu bölümde, bilginin gizlenmesi ve bu gizliliğin kırılması üzerine inşa edilen matematiksel temelleri inceliyoruz. Antik Yunan'daki basit fiziksel algoritmalardan, modern asimetrik şifrelemeye ve asal sayıların gücüne uzanan tam kapsamlı bir yolculuk.
Bu notlar sadece teorik ispatlarla sınırlı kalmayacak; aynı zamanda şifreleme algoritmalarının nasıl çalıştığını deneyimleyebileceğiniz **interaktif hesaplayıcılar** ve algoritmaların programatik çözümlerini içeren **kod implementasyonları** ile desteklenecektir.

1. Kriptografi Terminolojisi ve Temel Kavramlar

1.1 Kriptografiye Giriş

Etimolojik olarak Yunanca *kryptós* (gizli) ve *graphein* (yazmak) kelimelerinin birleşiminden doğan bu alana akademik dünyada üst başlık olarak **kriptoloji** adı verilir.

Tanım 1.1 (Temel Kriptoloji Kavramları).

- **Kriptoloji:** Bilgi güvenliğini, şifreleme sistemlerini ve bu sistemlerin kırılma yöntemlerini matematiksel temelleriyle inceleyen genel bilim dalıdır.
- **Kriptografi:** Bilgiyi belirli algoritmalar kullanarak şifreleyip yetkisiz erişimden gizleme bilimidir. Sistemin savunma mekanizmasını, yani kalkanını inşa eder.
- **Kriptanaliz:** Şifreli metinleri kriptografik anahtar (key) olmadan kırma, algoritmaları analiz etme ve sistemdeki matematiksel açıkları bulma bilimidir. Savunma kalkanını delmeyi amaçlar.

Kriptografi sadece teorik bir bilim değil, aynı zamanda verinin form değiştirdiği pratik bir süreçtir. Bu dönüşüm sürecinde verinin aldığı haller ve uygulanan işlemler şu kavramlarla ifade edilir:

Tanım 1.2 (Şifreleme Süreci ve Temel İşlemler).

- **Düz Metin (Plaintext):** Herhangi bir şifreleme işleminden geçmemiş, herkes tarafından okunabilen orijinal mesaj veya veridir. İletişimin başlangıç noktasını oluşturur.
- **Şifreli Metin (Ciphertext):** Düz metnin bir kriptografik algoritma ve anahtar yardımıyla dönüştürülmüş, yetkisiz kişiler için anlamsız ve okunamaz hale gelmiş versiyonudur.
- **Şifreleme (Encryption) ve Şifre Çözme (Decryption):** Şifreleme, algoritmalar kullanarak düz metni şifreli metne dönüştürme işlemidir. Şifre çözme ise, şifreli metni doğru anahtar kullanarak tekrar orijinal düz metne çevirme (geri döndürme) sürecidir.

1.2 Kriptografinin Tarihsel Evrimi

Tarih boyunca bilginin gücü, onu koruma ihtiyacını doğurmuş; bu ihtiyaç da imparatorlukların kaderini belirlemiştir. Kriptografi sadece bir matematik oyunu değil, savaşların seyrini değiştiren en kritik stratejik araçtır. Savaşlar cephede fiziksel güçle edilse de, arka planda istihbarat ve şifreleme ile kazanılır. Düşmanın haberleşmesini kırmak, orduların stratejik konumlarını önceden bilmek demektir. Kriptografi ve kriptanaliz arasındaki bu sürekli mücadele, insan zekasının sınırlarını zorlayarak matematiğin en yenilikçi dallarından birini inşa etmiştir.
Kriptografinin evrimi, bir tarafın kırılamaz olduğunu iddia ettiği bir şifre icat etmesi ve diğer tarafın o şifreyi kıracak yeni bir matematiksel yöntem geliştirmesi döngüsüyle ilerlemiştir. Geleneksel listeleme yerine, bu evrimsel süreci beş ana tarihsel dönüm noktası üzerinden inceleyebiliriz:

1.2.1 Antik Çağlar ve Fiziksel Şifreler

Kriptografinin bilinen ilk sistematik askeri kullanımı Spartalılara dayanır. **Skytale** adı verilen belirli çaptaki silindirik bir sopaya sarılan derilerle mesajları fiziksel boyutlara bağlı kalarak şifrelemişlerdir (bu kullanımın gerçek bir şifreleme yöntemi mi yoksa mesajın bütünlüğünü doğrulamaya yönelik bir araç mı olduğu tarihçiler arasında hâlâ tartışmalıdır). Roma'nın ünlü komutanı ve diktatörü Jül Sezar ise generalleriyle haberleşirken alfabedeki harfleri belirli bir k kaydırma miktarı kadar öteleyerek tarihin en bilindik yerine koyma (substitution) algoritmalarından biri olan **Sezar Şifresi**'ni yaratmıştır.

1.2.2 Orta Çağ ve Frekans Analizi

Yüzyıllar boyunca Sezar tarzı tek alfabeli (monoalfabetik) şifreler kırılamaz kabul edilmiştir. Dokuzuncu yüzyılda Arap matematikçi El-Kindi, her harfin belirli dillerdeki istatistiksel kullanım sıklıklarını analiz ederek **Frekans Analizi** yöntemini icat etmiştir. Bu matematiksel hamle, dilbilim ve istatistiğin kriptografiye indirdiği ilk büyük darbedir ve kriptanaliz biliminin asıl doğuşu olarak kabul edilir.

1.2.3 Rönesans ve Çoklu Alfabe Kullanımı

Tek alfabeli şifrelerin istatistiksel analizle kırılabildiğinin ortaya konmasının ardından, on altıncı yüzyılda birden fazla alfabe (polialfabetik) kullanan şifreleme yöntemleri geliştirilmiştir. Bugün **Vigenère Şifresi** olarak bilinen ve periyodik bir şifreleme tablosuna (tabula recta) dayanan bu yöntem, aslında ilk kez 1553 yılında İtalyan kriptograf Giovan Battista Bellaso tarafından tanımlanmış, ancak on dokuzuncu yüzyılda hatalı biçimde Fransız diplomat Blaise de Vigenère'e atfedilmiştir. Kaynağı tartışmalı olsa da bu yöntem, karmaşıklığı sayesinde yaklaşık üç yüzyıl boyunca kırılamaz şifre (*Le Chiffre Indéchiffrable*) unvanını korumuştur.

1.2.4 Dünya Savaşları ve Elektromekanik Şifreleme

Yirminci yüzyıla gelindiğinde şifreleme yöntemleri kağıt ve kalemden çıkıp karmaşık rotorlu elektromekanik makinelere evrilmiştir. II. Dünya Savaşı'nda Almanların kullandığı **Enigma**, her tuş vuruşunda şifreleme ayarını değiştirerek milyarlarca farklı kombinasyon üretmiş ve üst düzey bir güvenlik sağlamıştır. Alan Turing ve Bletchley Park'taki ekibin Enigma'nın zafiyetlerini bulmak için geliştirdiği elektromekanik **Bombe** cihazı ve kısa süre sonra Lorenz şifresini kırmak amacıyla inşa edilen, dünyanın ilk programlanabilir elektronik bilgisayarlarından biri kabul edilen **Colossus**, modern bilgisayar biliminin gelişimine öncülük etmiştir.

1.2.5 Modern Çağ ve Bilgi Teorisi

1949 yılında Claude Shannon'ın yayımladığı makalelerle kriptografi, tamamen bilgi teorisi, olasılık ve istatistiğe dayanan bir bilim dalına dönüşmüştür. İşlemcilerin icadıyla birlikte artık metin harfleri değil bitler (**0** ve **1**) şifrelenmekte; modern açık anahtarlı sistemler ise devasa asal sayılar, modüler aritmetik ve eliptik eğriler gibi ileri matematiksel yapılar üzerine kurulmaktadır. İnternet üzerindeki güncel güvenli iletişim protokolleri, bu matematiksel problemlerin hesaplama açısından çözülmesinin (mevcut bilgi ve teknoloji ile) pratikte imkânsıza yakın olduğu varsayımına dayanmaktadır.

1.3 Kriptografi Sahnesinin Değişmez Aktörleri

Kriptografik protokolleri, ağ yapılarını veya algoritmaları incelerken genel ifadeler kullanmak yerine, kriptografi literatüründe evrensel bir standart olarak kabul görmüş üç temel aktör kullanılır. Bu isimlendirme, karmaşık ağ senaryolarını analitik olarak daha anlaşılır hale getirir.

Tanım 1.3 (İletişim Protokolü Aktörleri).

- **Alice (Gönderici):** İletişim sürecinin başlangıç noktasıdır. Gizliliği veya bütünlüğü korunması gereken düz metni (plaintext), güvenli bir şekilde karşı tarafa iletmek isteyen asıl kişiyi temsil eder.
- **Bob (Alıcı):** Alice'in mesajını bekleyen, gelen şifreli metni (ciphertext) kendi algoritmik anahtarı ile çözüp orijinal düz metne ulaşması gereken yetkili hedeftir.
- **Eve (Dinleyici / Eavesdropper):** Alice ve Bob arasındaki iletişim kanalını araya girerek dinleyen **pasif saldırgandır**. Mesajın içeriğini anlık olarak değiştiremez ancak şifreli metni ağ üzerinden kopyalayıp kriptanaliz yöntemleriyle orijinal bilgiye (düz metne) ulaşmaya çalışır.

> **i** Not
>
> İleri seviye protokollerde bu temel üçlüye, kötü niyetli ve mesajı değiştirebilen aktif saldırgan **Mallory** (Malicious), güvenilir sunucu veya hakem **Trent** (Trusted) ve doğrulayıcı **Victor** (Verifier) gibi yeni aktörler de dâhil olmaktadır.

2. Kriptografinin Altın Kuralı: Kerckhoffs Prensibi

19. yüzyılda telgrafın icadı askerî iletişimi radikal biçimde hızlandırdı; ancak aynı zamanda mesajların düşman tarafından yakalanma riskini de devasa ölçüde artırdı. Savaş alanında taşınması zor olan ve ele geçirildiğinde tüm sistemi çökerten "gizli kod kitaplarına" bir alternatif gerekiyordu.
İşte bu ortamda Hollandalı kriptograf **Auguste Kerckhoffs**, 1883'te yayımladığı *La Cryptographie Militaire* adlı makalesinde kriptografi dünyasını kalıcı olarak değiştiren prensibi ortaya koydu.

Tanım 2.1 (Kerckhoffs Prensibi, 1883). Bir şifreleme sisteminin güvenliği, **algoritmanın gizliliğine değil**, yalnızca **anahtarın gizliliğine** dayanmalıdır.
Düşman (Eve), kullanılan sistemin tüm iç işleyişini, donanımını ve yazılım kodlarını bilse bile; anahtara sahip olmadığı sürece şifreli metni çözememelidir.

Bu prensip, günümüzde siber güvenlik dünyasında sıklıkla eleştirilen **"gizlilikle sağlanan güvenlik" (security through obscurity)** yaklaşımının tam zıddıdır. Güvenliği algoritmayı saklayarak sağlamaya çalışmak, algoritma ifşa olduğunda tüm sistemin kalıcı olarak çökmesi anlamına gelir.

Neden yalnızca anahtarı gizliyoruz?

Bir algoritma ifşa olursa —ki savaşta veya siber dünyada er ya da geç ifşa olur— tüm cihazları, yazılımları ve altyapıyı değiştirmek aylar sürer ve devasa bir maliyet yaratır.
Buna karşılık yalnızca **anahtar** ifşa olursa, sistemin geri kalanına hiç dokunmadan **yeni bir anahtar üreterek** güvenliği saniyeler içinde yeniden sağlayabilirsiniz. Kırılganlık (brittleness) ile esneklik (ductility) arasındaki fark tam olarak budur.

2.1 Tarihten Çarpıcı Alıntılar

Kerckhoffs'un bu devrimsel fikri, ilerleyen yıllarda kriptografi ve siber güvenlik devleri tarafından kendi sözleriyle yeniden formüle edilmiştir:

> *"Düşman sistemi biliyor."*
> — **Claude Shannon**, bilgi teorisinin kurucusu. Bu söz literatüre **Shannon'ın maksimi** olarak geçmiştir.

> *"Sisteminizi, rakiplerinizin onu tüm detaylarıyla bildiğini varsayarak tasarlayın."*
> — **Steven M. Bellovin**. NSA'daki standart varsayım, üretilen herhangi bir yeni cihazın bir numaralı seri üretiminin doğrudan Kremlin'e teslim edildiği yönündeydi.

> *"Her sır, potansiyel bir çöküş noktası yaratır. Sırrın açığa çıkması, sistemin feci şekilde çökmesine neden olur. Açıklık ise sisteme esneklik sağlar."*
> — **Bruce Schneier**

2.2 Kerckhoffs'un Altı Temel Kuralı

Kerckhoffs makalesinde yalnızca bu prensibi vermekle kalmamış, aynı zamanda pratik bir askerî şifreleme sisteminin sahip olması gereken **altı kuralı** da listelemiştir.

i Kerckhoffs'un altı orijinal tasarım kuralı

1. Sistem matematiksel olarak olmasa bile **pratik olarak kırılamaz** olmalıdır.
2. Sistem **gizlilik gerektirmemeli** ve düşmanın eline geçmesi bir sorun teşkil etmemelidir. *(Kerckhoffs prensibi olarak anılan kural budur.)*
3. Anahtar, yazılı notlara ihtiyaç duymadan **akılda tutulabilmeli** ve taraflarca kolayca değiştirilebilmelidir.
4. Sistem **telgraf iletişimine** — günümüzde dijital veri aktarımına — uygulanabilir olmalıdır.
5. Cihaz **taşınabilir** olmalı ve kullanımı için birden fazla kişiye ihtiyaç duyulmamalıdır.
6. Sistem **kullanımı kolay** olmalı; kullanıcıları uzun kural listelerine veya zihinsel yüke maruz bırakmamalıdır.

Bugün bilgisayarların işlem gücü sayesinde beşinci ve altıncı kurallar (taşınabilirlik ve insan hafızası) şekil değiştirmiş olsa da; **ikinci kural**, yani Kerckhoffs prensibi, modern kriptografinin tartışılmaz anayasası olmaya devam etmektedir.

! Prensibin yaygın bir yanlış okunuşu

Kerckhoffs prensibi, algoritmanın *mutlaka yayımlanması gerektiğini* söylemez; **gizli kalmasına ihtiyaç duyulmaması gerektiğini** söyler.
Modern pratikte algoritmalar yine de açık biçimde yayımlanır — çünkü bir şifreleme yönteminin güvenliğine duyulabilecek tek makul gerekçe, onun yıllarca dünyanın dört bir yanındaki kriptanalistler tarafından incelenip kırılamamış olmasıdır. AES ve SHA-3 gibi standartların açık yarışmalarla seçilmesinin sebebi tam olarak budur.

3. Kriptografide Matematiksel Temeller ve Notasyon

Modern kriptografi, güvenliğini karmaşık metin karıştırma oyunlarından değil; **Sayılar Teorisi** ve **Soyut Cebir**'in sarsılmaz kurallarından alır. Şifreleme algoritmalarının (özellikle RSA, Diffie-Hellman ve eliptik eğri tabanlı asimetrik sistemlerin) temelinde yatan matematiksel yapılar aşağıda özetlenmiştir.

i EBOB ve EKOK Notasyon Anlaşması

Uluslararası literatüre sadık kalmak adına, notlarımızda Türkçe kısaltmalar (EBOB/EKOK) yerine küresel kriptografi kaynaklarında kullanılan standart matematiksel fonksiyonlar tercih edilecektir:

- $\gcd(a, b)$ **(Greatest Common Divisor):** a ve b sayılarının **en büyük ortak bölenini** ifade eder.
- $\text{lcm}\ (a, b)$ **(Least Common Multiple):** a ve b sayılarının **en küçük ortak katını** ifade eder.

Altın Notasyon: Aralarında Asallık
Notlarımız boyunca karşılaşacağımız en kritik gösterim şudur:

$$\gcd(a, b) = 1 \Leftrightarrow a \ \text{ ve } \ b \ \text{ sayıları aralarında asaldır.}$$

Bu sade gösterim; modüler aritmetikte sadeleştirme yapılabilirliğinin, $\mathbb{Z}_n$ içinde çarpımsal tersin var olmasının ve Euler phi (ϕ) fonksiyonunun hesaplanmasının yegâne matematiksel şartıdır.

3.1 Modüler Aritmetik ve $\mathbb{Z}_n$ Kümeleri

Kriptografide sonsuz sayılarla işlem yapmak bilgisayarlar için pratik (veya güvenli) değildir. Bu nedenle işlemler, sayılar belirli bir n modülüne ulaştığında başa saran dairesel bir sistem üzerinde, yani **modüler aritmetik** kullanılarak yapılır.

Tanım 3.1 (Modüler Denklik (Kongrüans)). a, b ve n tam sayılar ($n > 0$) olmak üzere; eğer n sayısı $(a - b)$ farkını tam bölüyorsa, "a ve b sayıları modülo n'de birbirine denktir" denir ve şu şekilde gösterilir:

$$a \equiv b \pmod{n}$$

3.1.1 Modüler Aritmetiğin Temel Özellikleri

a, b, c, d birer tam sayı ve $n > 0$ bir modül olsun. Cebirsel işlemler, alt alta yazılan denkliklerin taraf tarafa işleme sokulmasıyla çok daha net görülebilir.

- **Toplama ve çıkarma:** Aynı modüle sahip denklikler alt alta toplanabilir veya çıkarılabilir. Modül aynen korunur:

$$\left.\begin{array}{r} a \equiv b \pmod{n} \\ c \equiv d \pmod{n} \end{array}\right\} \Longrightarrow a \pm c \equiv b \pm d \pmod{n}$$

- **Her iki tarafa aynı sayıyı ekleme ve çıkarma:** Klasik cebirde olduğu gibi, modüler aritmetikte de bir denkliğin her iki tarafına aynı sayıyı ekleyebilir veya çıkarabilirsiniz. Modül yine sabit kalır:

$$a \equiv b \pmod{n} \Leftrightarrow a \pm c \equiv b \pm c \pmod{n}$$

- **Sabit bir skalerle çarpma:** Bir denkliğin her iki tarafını aynı k tam sayısıyla çarpabilirsiniz; modül sabit kalır. Aslında bu özellik, $a \equiv b \pmod{n}$ ile $k \equiv k \pmod{n}$ denkliklerinin taraf tarafa çarpımından elde edilir:

$$a \equiv b \pmod{n} \Longrightarrow k \cdot a \equiv k \cdot b \pmod{n}$$

- **Taraf tarafa çarpma:** Aynı modüle sahip iki farklı denkliğin sol tarafları kendi arasında, sağ tarafları kendi arasında çarpılabilir. Modül yine sabit kalır:

$$\left.\begin{array}{c} a \equiv b \pmod{n} \\ c \equiv d \pmod{n} \end{array}\right\} \Longrightarrow a \cdot c \equiv b \cdot d \pmod{n}$$

- **Üs alma:** Taraf tarafa çarpma kuralının doğal bir sonucu olarak ($c = a$ ve $d = b$ alınıp k defa çarpıldığında), bir denkliğin her iki tarafının aynı $k \geq 1$ pozitif tam sayı kuvveti alınabilir:

$$a \equiv b \pmod{n} \Longrightarrow a^k \equiv b^k \pmod{n}$$

- **Denkliği genişletme (modülü de çarpmak):** Tüm denkliği bir $k > 0$ tam sayısıyla çarparak genişletiyorsanız, çözüm kümesinin bozulmaması için **modülü de aynı k sayısıyla çarpmak zorundasınız.** Sadece tarafları çarpıp modülü sabit bırakmak denklemi tamamen değiştirir:

$$a \equiv b \pmod{n} \Leftrightarrow a \cdot k \equiv b \cdot k \pmod{n \cdot k}$$

⚠ Tehlikeli sular: bölme ve sadeleştirme kuralı

Modüler aritmetikte doğrudan "bölme" işlemi yoktur ve klasik cebirdeki gibi her iki tarafı aynı sayıya bölerek sadeleştirme yapmak çok sık yapılan ölümcül bir hatadır.
Çarpım durumundaki bir c tam sayısını her iki taraftan sadeleştirmek istiyorsak, modülün de c ile olan en büyük ortak bölenine bölünmesi zorunludur:

$$a \cdot c \equiv b \cdot c \pmod{n} \Leftrightarrow a \equiv b \pmod{\frac{n}{\gcd(c, n)}}$$

Kriptografide en sık kullanılan özel durum: Eğer sadeleştireceğimiz c sayısı ile modül n **aralarında asal** ise ($\gcd(c, n) = 1$), $\frac{n}{1} = n$ olacağından modül değişmeden doğrudan sadeleştirme yapılabilir:

$$a \cdot c \equiv b \cdot c \pmod{n} \Longrightarrow a \equiv b \pmod{n}$$

! Dikkat: üslerde sadeleştirme yapılamaz

Tabandaki sayılar aralarında asallık kuralına göre sadeleşebilse de, **üsler doğrudan sadeleştirilemez**. Yani:

$$x^a \equiv x^b \pmod{n} \not\Rightarrow a \equiv b \pmod{n}$$

Modüler aritmetikte üsleri birbirine eşitlemek, indirgemek veya sadeleştirmek için modül n'ye göre değil; Euler-Fermat teoremi gereği **$\phi(n)$'e göre** işlem yapmak zorunludur.

3.1.2 $\mathbb{Z}_n$ Kümeleri ve Çarpımsal Ters

Tüm bu modüler işlemlerin yapıldığı kalanlar kümesine $\mathbb{Z}_n$ denir.

Tanım 3.2 ($bb(Z)_n$ Kümesi). n pozitif bir tam sayı olmak üzere, modülo n'de kalanların oluşturduğu tam sayılar kümesi

$$\mathbb{Z}_n = \{0, 1, 2, ..., n-1\}$$

şeklinde gösterilir.

Modüler aritmetikte doğrudan "bölme" işlemi olmadığı için, denklemleri çözmek adına **çarpımsal ters** (multiplicative inverse) kavramı devreye girer. Bir sayıya bölmek yerine, o sayının çarpımsal tersiyle çarparız.

Tanım 3.3 (Çarpımsal Ters (Multiplicative Inverse)). $a \in \mathbb{Z}_n$ olmak üzere, eğer

$$a \cdot x \equiv 1 \pmod{n}$$

denkliğini sağlayan bir x tam sayısı varsa, bu x sayısına a'nın modülo n'deki çarpımsal tersi denir ve $a^{-1} \pmod{n}$ şeklinde gösterilir.

Kriptografik açıdan bu terslerin bulunabilmesi çok kritiktir. $\mathbb{Z}_n$ içinde bir a elemanının çarpımsal tersinin var olması için **gerek ve yeter şart $\gcd(a, n) = 1$ olmasıdır**. Bu ters elemanlar pratikte genişletilmiş Öklid algoritması kullanılarak çok hızlı biçimde hesaplanır.

Neden asal sayılar?

Eğer n sayısı bir p **asal sayısı** olarak seçilirse, $\mathbb{Z}_p$ içinde sıfır hariç *her elemanın* p ile aralarında asal olacağı garanti edilir; yani sıfır hariç her elemanın bir çarpımsal tersi olur. Bu durum $\mathbb{Z}_p$'yi sadece bir halka olmaktan çıkarıp bir **cisim (field)** yapar. Bu yapı, AES gibi modern algoritmaların bel kemiğidir.

3.2 Euler'in Phi (ϕ) Fonksiyonu

Kriptografinin, özellikle de RSA algoritmasının en büyük kahramanlarından biri **Euler'in totient (phi) fonksiyonudur**.

Tanım 3.4 (Euler Phi Fonksiyonu $phi.alt \(n \)$). Bir n pozitif tam sayısı için, $1 \leq a \leq n$ aralığında bulunan ve n ile **aralarında asal** olan tam sayıların adedini veren fonksiyondur.

3.2.1 $\phi(n)$ Fonksiyonunun Özellikleri

Hesaplama yaparken bu fonksiyonun sahip olduğu çarpımsal özellikler hayat kurtarır.

1. **Asal sayı kuralı.** Eğer p bir asal sayı ise, kendisinden küçük tüm pozitif tam sayılarla aralarında asal olacağından:
$$\phi(p) = p - 1$$
2. **Asal kuvvet kuralı.** Eğer p asal ve $k \geq 1$ tam sayı ise:
$$\phi(p^k) = p^k - p^{k-1}$$
3. **Çarpımsallık kuralı.** Eğer m ve n aralarında asal ise ($\gcd(m, n) = 1$):
$$\phi(m \cdot n) = \phi(m) \cdot \phi(n)$$
4. **Genel formül.** Herhangi bir n sayısının asal çarpanlarına ayrılmış hâli $n = p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r}$ ise:
$$\phi(n) = n\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right)\cdots\left(1 - \frac{1}{p_r}\right)$$

3.3 Euler ve Fermat Teoremleri

Şifreleme sırasında veriyi çok büyük üslere çıkardığımızda, modüler aritmetikte bu üsleri küçültmek ve işlemi bilgisayarlar için çözülebilir kılmak adına bu iki teorem kullanılır.

Teorem 3.1 (Fermat'nın Küçük Teoremi). *Eğer p bir asal sayı ve a, p ile bölünemeyen bir tam sayı ise ($\gcd(a, p) = 1$):*

$$a^{p-1} \equiv 1 \pmod{p}$$

Fermat'nın küçük teoremi yalnızca asal modüller için geçerlidir. İsviçreli matematikçi Leonhard Euler bu teoremi **asal olmayan modüller** için genişletmiş ve literatüre RSA algoritmasının anahtar üretim iskeletini kazandırmıştır:

Teorem 3.2 (Euler-Fermat Teoremi). *Eğer a ve n **aralarında asal** iki tam sayı ise ($\gcd(a, n) = 1$):*

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

⚠ Şifre kırmanın zorluğu

RSA sisteminde $n = p \cdot q$ şeklinde devasa iki asal sayının çarpımı herkese açık olarak verilir. Sistemin güvenli olmasının sebebi, n bilinmesine rağmen p **ve** q **çarpanları bilinmeden** $\phi(n)$ **değerinin hesaplanamamasıdır**. Yeterince büyük kuantum bilgisayarlarda çalışacak Shor algoritması gibi hızlı bir asal çarpanlara ayırma yöntemi pratikleşirse, bu teoremlerin dayandığı hesaplama zorluğu ortadan kalkar ve RSA kırılır.

3.4 Wilson Teoremi

Asal sayıları tespit etmek (primality test) veya teorik analizler yapmak için kullanılan güçlü bir başka teorem şudur:

Teorem 3.3 (Wilson Teoremi). *Bir $p \geq 2$ tam sayısının **asal sayı** olması için gerek ve yeter şart:*

$$(p-1)! \equiv -1 \pmod{p}$$

Faktöriyel, sayılar büyüdükçe çok hızlı biçimde büyüdüğü için Wilson teoremi pratikte devasa kriptografik asalların testinde kullanılamayacak kadar yavaştır. Bunun yerine genellikle Miller-Rabin gibi olasılıksal asallık testleri tercih edilir.

4. Skytale Şifresi ve Silindir Mantığı

Terminoloji bölümünde açık metin, şifreli metin ve anahtar kavramlarıyla tanışmıştık. Artık bu sözcükleri ilk kez gerçek bir şifre üzerinde kullanma zamanı. İlk durağımız ne bir formül ne de bir tablo: yaklaşık iki bin beş yüz yıl önce Sparta'da yontulmuş tahta bir sopa.

4.1 Sparta'dan Gelen Emir

MÖ 5. yüzyılın Sparta'sı, hayatın her ayrıntısını savaşa göre düzenlemiş bir asker devletiydi. Devletin en tepesindeki **ephor** adı verilen beş yüksek yönetici, denizaşırı seferlere gönderdikleri komutanlarla haberleşmek zorundaydı — üstelik bu mesajlar, düşman eline geçse bile okunamamalıydı. Çözümleri, tarihin bilinen ilk **askeri şifreleme cihazı** kabul edilen tahta bir silindirdi: **skytale (σκυτάλη)**. Sözcük Yunancada "sopa, asa" anlamına gelir.

Bu cihazı bize en renkli anlatan kaynak, olaylardan yaklaşık beş yüzyıl sonra yazan biyografi ustası Plutarkhos'tur. *Lysandros'un Hayatı*'nda anlattığına göre ephor'lar, sefere çıkan her komutana kendi ellerindekiyle **tıpatıp aynı kalınlıkta** yontulmuş bir sopa verir, mesajlarını da bu sopaya sarılan bir deri şerit üzerine yazarlardı. Ege'de zaferden zafere koşan ve gücü Sparta'yı huzursuz etmeye başlayan ünlü amiral Lysandros'a günün birinde işte böyle bir şerit ulaştı; amiral şeridi sopasına sarınca karşısına çıkan emir kısa ve netti: Sparta'ya dön.

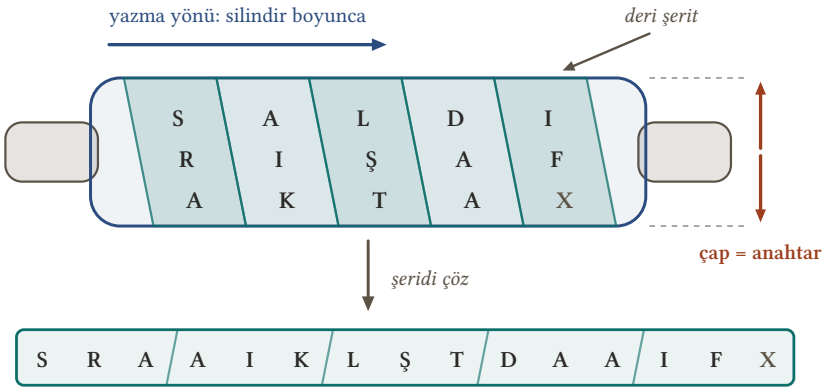
> **i** Tarihçiler arasında küçük bir tartışma
>
> Antik kaynaklar skytale'den sıkça söz etse de, bazı çağdaş tarihçiler cihazın gerçekten bir *şifreleme* aracı mı, yoksa mesajın gerçekten ephor'lardan geldiğini kanıtlayan bir *kimlik doğrulama* aracı mı olduğunu tartışır. Biz bu notlarda geleneksel anlatıyı izleyeceğiz; çünkü mekanizması, birazdan tanımlayacağımız yer değiştirme şifrelerinin en saf ve en öğretici örneğidir.

4.2 Silindir ve Deri Şerit: Fiziksel Mekanizma

Skytale'nin çalışması bir el işi kadar somuttur:

1. Gönderici (Alice), uzun ve dar bir deri ya da parşömen şeridi silindirin etrafına **sarmal biçimde**, hiç boşluk bırakmadan sarar.
2. Mesajını şeridin üzerine, **silindir boyunca** — yani sopanın eksenine paralel satırlar hâlinde — yazar. Her harf, şeridin farklı bir sargısına denk gelir.
3. Şerit sopadan çözüldüğünde, üzerinde yalnızca alt alta gelmiş harflerin karışık bir dizisi kalır. Şeridi taşıyan haberci yakalansa bile — kimi anlatımlara göre şerit bir kemer gibi bele sarılırdı — düşman eline anlamsız bir harf yığını geçer.
4. Alıcı (Bob), şeridi **aynı çaptaki** kendi sopasına sarar; harfler yeniden hizalanır ve mesaj satır satır okunur.

Buradaki kilit gözlem şudur: sistemin gizli bileşeni sopanın kendisi değil, **çapıdır**. Daha kalın ya da daha ince bir sopaya sarılan şerit yine anlamsız kalır; harfler ancak doğru çapta yeniden hizalanır. Modern gözle bakarsak: *yöntem* (şeridi sopaya sar) herkesçe bilinebilir, gizli kalması gereken tek şey *çap* — yani **anahtar**. Bu, Kerckhoffs prensibinin iki bin yıl önceden gelen sezgisel bir örneğidir.



Şekil 4.1: SALDIRI ŞAFAKTA mesajı (dolgu harfi X ile) silindire sarılı deri şeride üç sıra hâlinde, silindir boyunca yazılır. Şerit çözüldüğünde harfler sargı sargı okunur ve ortaya SRAAIKLŞTDAAIFX karışık dizisi çıkar. Mesajı, ancak aynı çapta bir silindire sahip olan kişi geri sararak okuyabilir.

4.3 Silindirden Izgaraya: Matematiksel Model

Sopayı ve deriyi bir kenara bırakıp olan biteni soyutlaştıralım. Silindirin çevresine alt alta m satır yazı sığıyorsa, sopa üzerinde m satırlık bir yazı alanı vardır; mesajın uzunluğu da satır başına düşen harf sayısını, yani n'yi belirler. O hâlde:

- **Sarmak ve yazmak**, mesajı $m \times n$ boyutlu bir ızgaraya **satır satır** yazmak demektir. Buradaki m — çevrenin izin verdiği satır sayısı — anahtarın ta kendisidir; fiziksel dünyadaki karşılığı silindirin çapıdır.
- **Şeridi çözmek**, aynı ızgarayı **sütun sütun** okumak demektir; çünkü şerit üzerinde art arda gelen harfler, sopa üzerinde alt alta duran harflerdir.

Dikkat edilirse bu süreçte hiçbir harf başka bir harfe dönüşmez: `S` yine `S`, `A` yine `A` olarak kalır. Değişen tek şey harflerin **konumlarıdır**. Bu gözlem, klasik kriptografinin iki büyük ailesinden ilkini tanımlar:

Tanım 4.1 (Yer Değiştirme (Transpozisyon) Şifresi). Açık metindeki harflerin kimliklerine dokunmadan yalnızca **konumlarını** belirli bir kurala (ve anahtara) göre değiştiren şifreleme yöntemine **yer değiştirme (transpozisyon) şifresi** denir.
Sonuç olarak şifreli metin, açık metnin harflerinin yeniden sıralanmış hâlidir; yani açık metnin bir **anagramıdır**.

Ailenin ikinci üyesi, harflerin konumunu koruyup kimliğini değiştiren **yerine koyma (substitution)** şifreleridir; onları ileride Sezar şifresiyle tanıyacağız. Skytale bizim için değerlidir, çünkü yer değiştirme fikrini olabilecek en yalın biçimde — düz bir ızgara üzerinde — gösterir.

4.4 Adım Adım: Şifrele ve Çöz

Örnek 4.1. Alice, cephedeki Bob'a "SALDIRI ŞAFAKTA" emrini göndermek istiyor. İkilinin elindeki sopaların çevresine alt alta $m = 3$ satır yazı sığıyor. Şifreli metni bulalım; ardından Bob'un şeridi nasıl çözdüğünü adım adım izleyelim.

> **Çözüm.**
> **Şifreleme (Alice'in yaptığı).**
> Önce boşluğu atalım: `SALDIRIŞAFAKTA` — toplam 14 harf. Anahtar $m = 3$ satır olduğuna göre ızgaranın tamamen dolması gerekir; $3 \times 4 = 12$ hücre yetmez, $3 \times 5 = 15$ hücre ise bir fazla verir. Eksik kalan tek hücreyi anlamsız bir **dolgu (padding)** harfiyle, geleneksel tercihle bir `X` ile tamamlarız: `SALDIRIŞAFAKTAX`.
> Şimdi bu 15 harfi 3×5 ızgaraya **satır satır** yazalım:

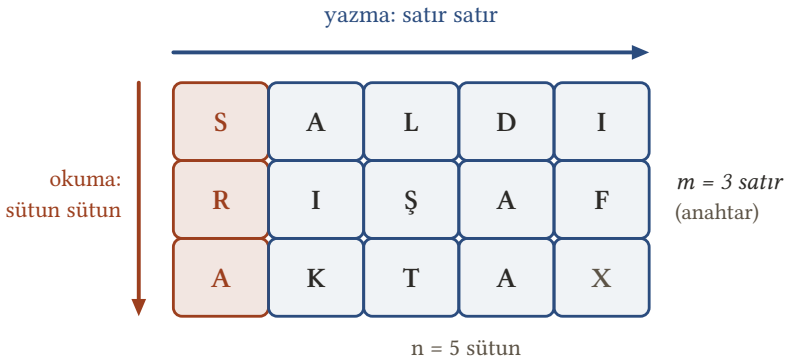
Tablo 4.1: Mesajın 3×5 ızgaraya satır satır yazılışı

	1. sütun	2. sütun	3. sütun	4. sütun	5. sütun
1. satır	S	A	L	D	I
2. satır	R	I	Ş	A	F
3. satır	A	K	T	A	X

Şeridi çözmek, ızgarayı **sütün sütün** (her sütunu yukarıdan aşağıya) okumaktır:

$$\underbrace{\text{SRA}}_{1.\text{ sütun}}\ \underbrace{\text{AIK}}_{2.\text{ sütun}}\ \underbrace{\text{LŞT}}_{3.\text{ sütun}}\ \underbrace{\text{DAA}}_{4.\text{ sütun}}\ \underbrace{\text{IFX}}_{5.\text{ sütun}}$$

Alice'in habercisine teslim ettiği şifreli metin: `SRAAIKLŞTDAAIFX`.



Şekil 4.2: Sarmak, mesajı 3×5'lik ızgaraya satır satır yazmaktır; şifrelemek ise aynı ızgarayı sütün sütün okumaktır. Kırmızı sütün, şifreli metnin ilk üç harfini verir; X yalnızca ızgarayı tamamlayan dolgu harfidir.

Şifre çözme (Bob'un yaptığı).
Bob'un elinde aynı çapta bir sopa var; yani anahtarı biliyor: $m = 3$. Gelen şerit 15 harf olduğuna göre ızgaranın sütun sayısını hemen hesaplar:

$$n = \frac{15}{3} = 5.$$

Her sütunda $m = 3$ harf bulunacağından, Bob şifreli metni üçerli gruplara ayırır — her grup bir sütundur:
`SRA` — `AIK` — `LŞT` — `DAA` — `IFX`
Bu grupları soldan sağa **sütün sütün** ızgaraya yerleştirir: birinci sütuna yukarıdan aşağıya `S`, `R`, `A`; ikinci sütuna `A`, `I`, `K`; ve böyle devam eder. Ortaya çıkan ızgara, yukarıdaki tablonun tıpatıp aynısıdır. Artık **satır satır** okumak yeterli:
`SALDI` + `RIŞAF` + `AKTAX` = `SALDIRIŞAFAKTAX`
Sondaki dolgu harfi `X` atılınca emir ortaya çıkar: **SALDIRI ŞAFAKTA**. ⊠

4.5 Anahtar Uzayı ve Kırılma

Şifreli şeridi ele geçiren dinleyici Eve'in işi ne kadar zor? Bir şifrenin gücünü ölçmenin ilk kaba yolu, denenebilecek anahtarların sayısına — **anahtar uzayına (keyspace)** — bakmaktır. Skytale'de anahtar, satır sayısı m'dir ve bu sayının alabileceği değerler acınacak kadar azdır: ele geçen şerit 15 harflikse

ızgaranın tam dolması için m'nin 15'i bölmesi gerekir; adaylar yalnızca 1, 3, 5 ve 15'tir. Üstelik $m = 1$ (tek satırlık upuzun bir ızgara) ile $m = 15$ (her satıra tek harf) metni hiç değiştirmez. Eve'in gerçekten deneyeceği iki anahtar kalır:

Tablo 4.2: On beş harflik şifreli metin için olası anahtarların tek tek denenmesi

Denenen anahtar m	Izgaradan okunan metin	Sonuç
1 veya 15	`SRAAIKLŞTDAAIFX`	metin değişmedi
3	`SALDIRIŞAFAKTAX`	**anlamlı Türkçe!**
5	`SKARLAAŞIATFIDX`	anlamsız

Bütün anahtarları sırayla deneyip anlamlı çıktıyı seçme yöntemine **kaba kuvvet (brute force)** saldırısı denir. Modern şifrelerde bu saldırı astronomik anahtar sayıları yüzünden umutsuzdur; skytale'de ise kalem, kâğıt ve birkaç dakika yeter. Hatta Eve'in kâğıda bile ihtiyacı yoktur: yanında kalınlıkları farklı birkaç sopa taşıyıp ele geçirdiği şeridi sırayla her birine sarması yeterlidir.

Skytale'nin ikinci ve daha derin zayıflığı istatistikseldir. Yer değiştirme hiçbir harfi değiştirmediği için şifreli metin, açık metnin anagramıdır: örneğimizde `A` her iki metinde de dört, `I` iki kez geçer — **harf sıklıkları tıpatıp aynıdır**. Sıradan bir Türkçe (ya da antik örnekte Yunanca) metnin harf dağılımını gören Eve, daha ilk bakışta elindekinin bir yer değiştirme şifresi olduğunu anlar; şifre, metnin dilinden gelen istatistiksel izleri **hiç gizleyememiştir**. Konumları dağıtmanın gücünü ve tek başına neden yetersiz kaldığını, ileride Shannon'ın yayılma prensibiyle çok daha derinlemesine inceleyeceğiz.

Yine de ızgara fikrini küçümsemeyelim: mesajı bir tabloya *bir yönde yazıp başka yönde okumak*, tükenmez bir şifre üretme kalıbıdır. Satırları zikzak çizerek, sütunları farklı sıralarla ya da sarmal rotalarla okuyarak bambaşka şifreler elde edilebilir. Bir sonraki bölümde tam da bunu yapacağız.

5. Zigzag (Rail Fence) ve Rota (Route) Şifrelemesi

Skytale bölümünde şifreleme tarifimiz iki adımdan oluşuyordu: harfleri tabloya **satır satır yaz**, sonra **sütun sütun oku**. Bu bölümde aynı oyunun daha yaratıcı iki çeşidini inceleyeceğiz. Değişen tek şey, harflerin üzerinde yürüdüğü **yol**: önce harfleri hayali bir çit üzerinde aşağı yukarı dalgalandıran zigzag şifrelemesi, ardından tabloyu dilediğimiz güzergâhta dolaşan rota şifrelemesi.
Baştan vurgulayalım: iki yöntem de bir önceki bölümde tanıştığımız **yer değiştirme (transposition)** ailesindendir. Hiçbir harf başka bir harfe dönüşmez; yalnızca adresi değişir. Dolayısıyla üreteceğimiz her şifreli metin, açık metnin özenle karıştırılmış bir **anagramıdır** — bu basit gözlem, bölümün sonunda güvenlik analizimizin de anahtarı olacak.

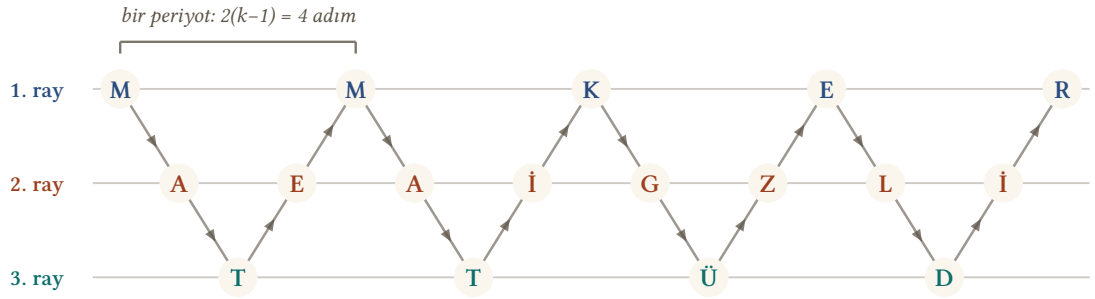
5.1 Zigzag (Rail Fence) Şifrelemesi

Yöntemin İngilizce adı *rail fence*, yani "çit rayı"dır; çünkü harflerin dizilimi, Amerikan çiftliklerindeki zikzak desenli ahşap çitlerin çapraz raylarını andırır. Türkçede kısaca **zigzag şifrelemesi** diyeceğiz.

Tanım 5.1 (Zigzag (Rail Fence) Şifrelemesi). $k \geq 2$ bir tam sayı olsun. Zigzag şifrelemesi üç adımda çalışır:

1. Üst üste k hayali **ray (rail)** düşünülür.
2. Açık metnin harfleri en üst raydan başlayarak çapraz biçimde **aşağı**, en alt raya varınca **yukarı** yazılır; bu iniş–çıkış dalgası metin bitene kadar sürer. Desen kendini her $2(k-1)$ adımda bir tekrarlar.
3. **Şifreli metin (ciphertext)**, rayların yukarıdan aşağıya sırayla, her rayın da kendi içinde soldan sağa okunmasıyla elde edilir.

Anahtar, yalnızca ray sayısı k'dir.



Şekil 5.1: Üç ray üzerinde zigzag: MATEMATİKGÜZELDİR mesajının harfleri, ince oklarla gösterilen aşağı–yukarı dalgalanan yol boyunca sırayla yerleştirilir; desen her $2(k-1) = 4$ adımda bir tekrarlanır. Şifreli metin raylar tek tek okunarak oluşur: önce 1. ray (MMKER), sonra 2. (AEAİGZLİ), en son 3. (TTÜD).

Buradaki $2(k-1)$ periyodunu görmenin en kolay yolu dalgayı izlemek: $k = 3$ ray için yol, ray numaraları cinsinden $1, 2, 3, 2$ dizisini yazar ve başa döner — toplam $2 \cdot (3-1) = 4$ adım. En üst ve en alt raylar dalganın "tepe" ve "dip" noktaları olduğu için desende birer kez, aradaki raylar ise hem inişte hem çıkışta uğranıldığı için ikişer kez görünür. Bu küçük muhasebe, birazdan şifre çözmenin tüm yükünü taşıyacak.

Örnek 5.1. **MATEMATİK GÜZELDİR** mesajını $k = 3$ anahtarıyla zigzag yöntemiyle şifreleyelim; ardından yalnızca şifreli metni ve k'yi bilen Bob'un mesajı nasıl geri kazandığını adım adım izleyelim.

Çözüm.
Şifreleme. Klasik şifrelerde alışıldığı üzere boşlukları atıyoruz (boşluklar kelime sınırlarını ele vererek düşmana bedava ipucu verir): metnimiz `MATEMATİKGÜZELDİR`, toplam 17 harf. Harfleri üç ray üzerinde dalgalandıralım:

```
1. ray:  M · · · M · · · K · · · E · · · R
2. ray:  · A · E · A · İ · G · Z · L · İ ·
3. ray:  · · T · · · T · · · Ü · · · D · ·
```

Şimdi rayları sırayla okuyoruz: birinci ray `MMKER`, ikinci ray `AEAİGZLİ`, üçüncü ray `TTÜD`. Bunları uç uca ekleyince şifreli metin çıkar: `MMKERAEAİGZLİTTÜD`.
Şifre çözme: ray uzunluklarının muhasebesi. Asıl öğretici kısım burası. Bob'un elinde yalnızca `MMKERAEAİGZLİTTÜD` (17 harf) ve $k = 3$ anahtarı var; tabloyu görmüyor. Şifreli metni raylara bölebilmesi için önce **her raya kaç harf düştüğünü** hesaplaması gerekir.
Desenin periyodu $2(k-1) = 2 \cdot 2 = 4$ adımdır ve her tam dalga $(1, 2, 3, 2)$ birinci raya 1, ikinci raya 2, üçüncü raya 1 harf bırakır. Metin uzunluğunu periyoda bölelim:

$$17 = 4 \cdot 4 + 1$$

Demek ki 4 tam dalga ve artakalan 1 adım var. Tam dalgalardan gelen paylar: birinci raya $4 \cdot 1 = 4$, ikinci raya $4 \cdot 2 = 8$, üçüncü raya $4 \cdot 1 = 4$ harf. Artakalan tek adım ise desenin ilk adımıdır, yani birinci raya düşer. Sonuç:

Tablo 5.1: Zigzag deşifrelemesinde ray uzunluklarının hesabı ($n = 17$, $k = 3$)

Ray	Tam dalgalardan	Artıktan	Toplam
Birinci	4	1	5
İkinci	8	0	8
Üçüncü	4	0	4

Sağlama: $5 + 8 + 4 = 17$ ✓. Artık Bob şifreli metni bu uzunluklarda dilimleyebilir: ilk 5 harf `MMKER` birinci ray, sonraki 8 harf `AEAİGZLİ` ikinci ray, son 4 harf `TTÜD` üçüncü raydır.
Son adım, zigzag yolunu boş tabloda yeniden yürümek: $1, 2, 3, 2, 1, 2, 3, 2, \ldots$ desenini izleyip her adımda sırası gelen rayın **soldaki ilk kullanılmamış harfini** almak yeterli:

```
ray sırası:  1 2 3 2 1 2 3 2 1 2 3 2 1 2 3 2 1
alınan harf: M A T E M A T İ K G Ü Z E L D İ R
```

Harfler yan yana `MATEMATİKGÜZELDİR` metnini, yani açık mesajımızı verir. Şifreleme de çözme de tutarlı: dalga aritmetiği çalışıyor.
⊠

Deşifrelemenin püf noktası

Zigzag çözerken en sık yapılan hata, şifreli metni raylara **eşit parçalara bölerek** dağıtmaya çalışmaktır. Oysa örnekte gördüğümüz gibi raylar eşit dolmaz: uç raylar dalganın tepe ve dip noktaları olduğundan seyrek, orta raylar iki kat sık dolar. Doğru bölme her zaman $n = q \cdot 2(k-1) + r$ muhasebesinden geçer.

5.2 Rota (Route) Şifrelemesi

Zigzagda yol sabittir: aşağı, yukarı, aşağı, yukarı... **Rota (route) şifrelemesi** ise yolu tamamen serbest bırakır. Alice ile Bob, harflerin tabloda hangi güzergâhla okunacağını önceden kararlaştırır; Eve bu güzergâhı bilmediği sürece elindeki harf yığınını dizemez.

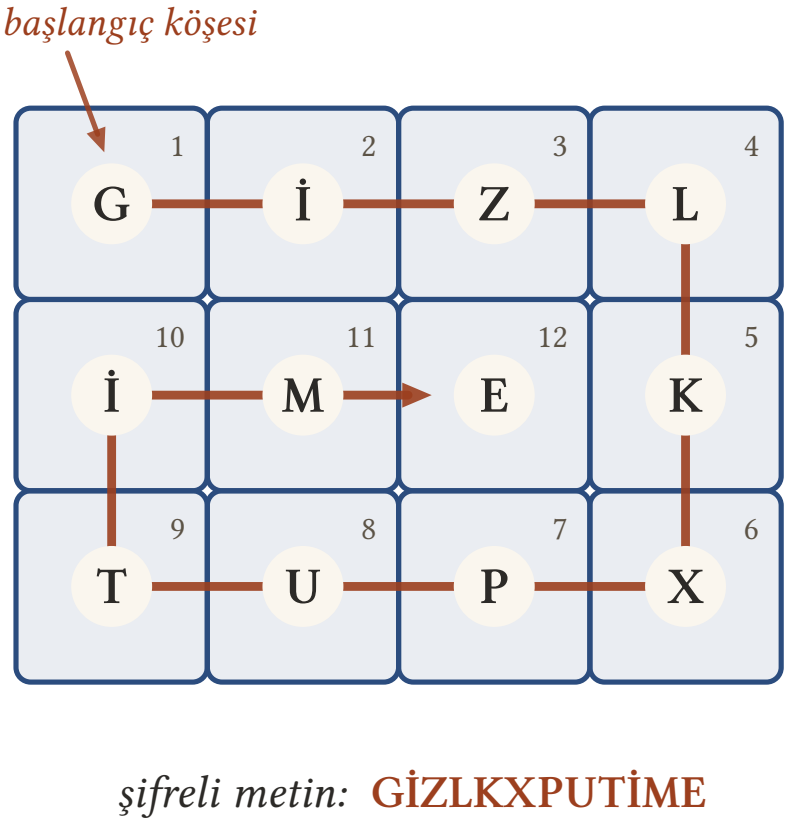
Tanım 5.2 (Rota (Route) Şifrelemesi). Alice ile Bob önceden $m \times n$ boyutlu bir tablo ve bir **okuma rotası (route)** üzerinde anlaşır:

1. Açık metin tabloya **satır satır** yazılır; boş kalan hücreler anlamsız **dolgu (padding)** harfleriyle tamamlanır.
2. Şifreli metin, tablonun anlaşılan rota boyunca dolaşılıp harflerin bu sırayla okunmasıyla elde edilir.

Anahtar üç bilginin birleşimidir: tablonun boyutu, rotanın türü ve rotanın başlangıç köşesi ile yönü.

Rota için hayal gücü tek sınırdır; klasik repertuvarın gözdeleri şunlardır:

- **Spiral:** bir köşeden başlayıp saat yönünde (ya da tersinde) içe doğru kıvrılarak tabloyu salyangoz gibi dolaşmak,
- **Boustrophedon:** köşeden köşeye, satırları bir soldan sağa bir sağdan sola süpürmek — Yunanca adı "öküzün tarlayı sürüşü"nden gelir; öküz saban çekerken satır sonunda geri dönüp ters yönde ilerler,
- **Yılankavi sütunlar:** sütunları bir aşağı bir yukarı, iniş çıkışlarla kat etmek.



Şekil 5.2: Rota şifresi: GİZLİMEKTUPX metni 3×4 tabloya satır satır yazılır, sonra sol üst köşeden başlayıp saat yönünde içe kıvrılan spiral rota boyunca okunur; köşelerdeki küçük sayılar okuma sırasını gösterir. Sonuç: GİZLKXPUTİME.

Örnek 5.2. GİZLİ MEKTUP mesajını 3×4 bir tabloya yazıp sol üst köşeden başlayan, saat yönünde içe kıvrılan spiral rotayla şifreleyelim.

Çözüm.
Boşluğu atınca metnimiz GİZLİMEKTUP, yani 11 harf; tablomuzda ise $3 \cdot 4 = 12$ hücre var. Boş kalacak tek hücreye dolgu olarak X ekliyoruz ve harfleri satır satır yerleştiriyoruz:

```
G İ Z L
İ M E K
T U P X
```

Şimdi anlaşılan rotayı yürüyelim: sol üst köşeden üst satır boyunca sağa (G İ Z L), sağ kenardan aşağı (K X), alt satır boyunca sola (P U T), sol kenardan yukarı (İ) ve son olarak içeride kalan iki hücre soldan sağa (M E). Harfleri bu sırayla dizince şifreli metin çıkar: GİZLKXPUTİME.
Şifre çözme. Bob tam tersini yapar: 3×4 boyutunda boş bir tablo çizer, şifreli metnin harflerini **aynı spiral güzergâh boyunca** hücrelere tek tek yerleştirir. Spiral, harfleri tam da Alice'in okuduğu hücrelere geri bırakacağından tablo yukarıdaki hâline döner; Bob tabloyu satır satır okuyup GİZLİMEKTUPX metnine ulaşır. Sondaki X'in dolgu olduğu bellidir, atılır: **GİZLİ MEKTUP**.
⊠

⚠ Spiralin sızıntısı

Örnekteki şifreli metnin ilk dört harfine bakın: GİZL — açık metnin ilk satırı, olduğu gibi! Sol üst köşeden saat yönünde başlayan spiralin ilk hamlesi daima üst satırı kopyalar. Başlangıç köşesini veya yönü değiştirmek bu sızıntıyı başka bir kenara taşır ama yok etmez: rota şifrelerinde açık metnin **bitişik harf blokları**, şifreli metinde de bitişik bloklar hâlinde gezinir. Dikkatli bir Eve için bu, bulmacanın yarısının hazır çözülmüş olması demektir.

5.3 Anahtar Uzayı ve Güvenlik

Bu iki şifreyi Kerckhoffs prensibinin terazisine koyalım: düşmanın yöntemi bildiğini, yalnızca anahtarı bilmediğini varsayacağız.
Zigzagda anahtar tek bir sayıdır. Üstelik seçenekler de kısıtlıdır: n harflik bir mesajda işe yarayan ray sayıları $2 \le k \le n-1$ aralığında kalır ($k \ge n$ seçilirse her harf kendi rayına düşer ve "şifreli" metin açık metnin kopyası olur). Bizim 17 harflik örneğimizde bu, topu topu 15 olası anahtar demek. Eve tüm adayları kâğıt kalemle bir çay molasında, bilgisayarla göz açıp kapayıncaya kadar dener; anlamlı Türkçe üreten k kendini hemen ele verir. Bu, tam anlamıyla bir **kaba kuvvet saldırısı (brute force attack)** senaryosudur.
Rota şifresinin anahtarı daha zengindir: tablo boyutu, rota türü, başlangıç köşesi ve yön birlikte seçilir. Yalnızca spiral ailesini sayalım: 4 köşe $\times$ 2 dönüş yönü $\times$ 2 kıvrılma yönü (içe veya dışa) $= 16$ farklı rota. Buna boustrophedon çeşitlerini, yılankavi sütunları, köşegen güzergâhları ve olası tablo boyutlarını da eklesek, en cömert sayımla birkaç yüz anahtara ulaşırız. Kulağa zigzagdan iyi geliyor; ama modern ölçekte birkaç yüz sayısı da sıfırdan farksızdır.

Tablo 5.2: Transpozisyon şifrelerinin anahtar uzayları — hepsi kaba kuvvetle taranabilecek kadar küçüktür

Şifre	Anahtarın içeriği	Kabaca anahtar sayısı
Skytale	çubuğun kalınlığı (satır sayısı)	bir avuç
Zigzag	ray sayısı k	$n-2$ dolayında
Rota	tablo boyutu + rota + köşe + yön	birkaç yüz

Asıl zafiyet ise anahtar sayısından da derindedir ve **bütün transpozisyon ailesini** vurur: harflerin yerini değiştirmek, sayılarını değiştirmez. Örneğimizdeki MMKERAEAİGZLİTTÜD metninde — tıpkı açık metinde olduğu gibi — ikişer tane M, A, T, E ve İ vardır. Eve bir harf sayımı yaptığında dağılımın Türkçenin doğal harf dağılımıyla çakıştığını görür ve iki şeyi birden öğrenir: karşısındaki şifre bir yerine koyma değil, bir

transpozisyondur ve elindeki metin açık metnin bir **anagramıdır**. Usta çözücüler için gerisi sabırlı bir bulmacadır: sık harf öbeklerini yan yana getirerek anagram çözer gibi metni dizerler. Tarih bunun çarpıcı bir örneğini saklar: Amerikan İç Savaşı'nda Kuzey ordusunun telgraf servisi, harfler yerine tam kelimeleri tabloya dizen kelime tabanlı rota şifreleri kullanmış; Güney'in kriptanalistleri bu mesajlar karşısında öyle çaresiz kalmıştır ki ele geçirdikleri şifreli metinleri kimi zaman gazetelerde yayımlayıp okurlardan çözüm yardımı istemişlerdir.

Peki harflerin yerini değiştirmek boşa kürek çekmek mi? Tam tersine. Transpozisyonun yaptığı iş — açık metindeki komşulukları kırıp harfleri metnin geneline dağıtmak — ileride Shannon prensiplerini incelerken göreceğimiz **yayılma (diffusion)** fikrinin atasıdır. Tek başına zayıf olan bu işlem, yerine koymayla kaynaştırılıp defalarca tekrarlandığında bambaşka bir güce kavuşur: ileride inceleyeceğimiz DES algoritmasının kalbinde, bitlerin yerini değiştiren P-permütasyonu olarak transpozisyon modern kriptografiye geri dönecek.

5.4 Bir Sonraki Durak: Mesajın Varlığını Gizlemek

Zigzag da rota da mesajın *içeriğini* gizler; ama Eve'in eline geçen `MMKERAEAİGZLİTTÜD` gibi bir metin, anlamsız harf yığınıyla adeta "ben bir şifreyim!" diye bağırır ve saldırganı davet eder. Ya mesaj, kimsenin şüphesini çekmeyen masum bir metnin içine saklanabilseydi? Bir sonraki bölümde Francis Bacon'ın zarif düzenini inceleyeceğiz: içeriği değil, **mesajın varlığını** gizleyen bir şifre.

6. Bacon Şifreleme

İngiliz filozof ve devlet adamı **Francis Bacon** (1561–1626), bilimsel yöntemin kurucularından biri olduğu kadar gizli yazışma sanatının da tutkulu bir zanaatkârıydı. Gençliğinde Paris'teki diplomatik görevi sırasında geliştirdiğini söylediği yöntemini ilk kez 1605 tarihli *Of the Proficience and Advancement of Learning* adlı eserinde kısaca anmış; sistemin tam tarifini ise 1623'te yayımladığı Latince genişletme *De Augmentis Scientiarum*'da vermiştir.
Bacon'ın hayali, kendi sözleriyle *omnia per omnia* — "her şeyi her şeyle yazmak" — idi: gizli mesaj, şifreli olduğu belli olan tuhaf bir harf yığınına değil, tamamen **masum görünen sıradan bir metnin içine** saklanacaktı. Mektubu ele geçiren Eve, ortada gizlenecek bir şey olduğundan şüphelenmeyecekti bile.

> **i** Dürüst bir konumlandırma: bu aslında bir şifre mi?
>
> Bu bölüm, kitabın harflerin *yerlerini* değiştiren yöntemlere ayırdığımız kısmında yer alsa da, Bacon'ın yöntemi teknik olarak bir **yer değiştirme (transposition)** şifresi değildir; hatta tam anlamıyla bir şifre bile sayılmaz. İki katmandan oluşur:
>
> 1. Herkesçe bilinebilecek sabit bir **kodlama (encoding)** — her harfe değişmez bir kod sözcüğü atanır; ortada gizli bir anahtar yoktur.
> 2. Asıl marifet olan **steganografi (steganography)** — Yunanca *steganós* (örtülü) + *graphein* (yazmak): mesajın içeriğini değil, bizzat **varlığını** gizleme sanatı.
>
> Terminoloji bölümünde gördüğümüz kriptografi tanımını hatırlayın: kriptografi mesajı okunamaz kılar, ama şifreli metnin ortada olduğu herkesçe görülür. Steganografi ise tam tersine, ortada bir mesaj olduğu gerçeğini saklar. Bacon'ın yöntemi bu ikinci aileye aittir; tarihsel akışı bozmamak için onu klasik şifrelerin arasında, ait olduğu çağda anlatıyoruz.

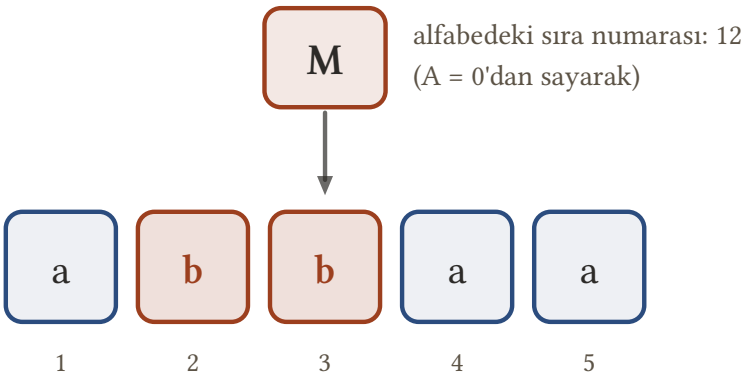
6.1 Beş Yuvalı Kod: Çift Harfli Alfabe

Bacon'ın sisteminin ilk katmanı, bugün **çift harfli şifre (biliteral cipher)** olarak anılan kodlamadır. Fikir şaşırtıcı derecede sadedir: alfabenin *her* harfini, yalnızca iki sembolden — a ve b — oluşan **beş yuvalık** bir kod sözcüğüyle temsil etmek.
Peki neden tam olarak beş yuva? Her yuvaya iki değerden biri yazılabildiğine göre, beş yuvanın üretebileceği farklı kalıp sayısı

$$2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^5 = 32$$

olur; bu, 26 harflik alfabenin tamamına yeter ($32 \geq 26$) ve üstelik 6 kalıp da artar. Dört yuva ise yetmezdi: $2^4 = 16 < 26$. Yani beş, iki sembolle bütün alfabeyi kodlayabilen **en küçük** yuva sayısıdır.

Tanım 6.1 (Bacon Kodu (Çift Harfli Kod)). Bacon kodu, alfabenin her harfine $\{a, b\}$ sembolleri üzerinden 5 uzunluğunda bir **kod sözcüğü (codeword)** atayan sabit bir kodlamadır. Modern 26 harfli sürümde kural şudur: harfin alfabedeki sıra numarası alınır (A için 0, B için 1, ..., Z için 25) ve bu sayı, yalnızca iki rakamla sayan bir düzende — a küçük, b büyük rakam olacak biçimde — beş haneli olarak yazılır.



Şekil 6.1: Bacon kodunda her harf, yalnızca a ve b sembollerinden oluşan beş yuvalık bir kalıba açılır: alfabede 12 numaralı harf olan M'nin kodu `abbaa`'dır. İki değerli beş yuva toplam 32 farklı kalıp üretir; bu da 26 harfin tamamına yeter.

Bu kurala göre A = *aaaaa*, B = *aaaab* ile başlar; her adımda "iki rakamlı sayaç" bir ilerler ve Z = *bbaab* ile biter. Tablonun tamamı şöyledir:

Tablo 6.1: Modern 26 harfli Bacon kod tablosu

Harf	Kod	Harf	Kod	Harf	Kod
A	`aaaaa`	J	`abaab`	S	`baaba`
B	`aaaab`	K	`ababa`	T	`baabb`
C	`aaaba`	L	`ababb`	U	`babaa`
D	`aaabb`	M	`abbaa`	V	`babab`
E	`aabaa`	N	`abbab`	W	`babba`
F	`aabab`	O	`abbba`	X	`babbb`
G	`aabba`	P	`abbbb`	Y	`bbaaa`
H	`aabbb`	Q	`baaaa`	Z	`bbaab`
I	`abaaa`	R	`baaab`		

i Bacon'ın orijinal tablosu 24 harflikti

Bacon'ın kendi çağının Latin kökenli alfabesinde **I ile J** tek harf sayılıyordu; **U ile V** de öyle. Bu yüzden orijinal tablo 24 harflikti ve I/J ile U/V ortak birer kod paylaşıyordu. Sonuç olarak tarihî metinlerde göreceğiniz tablonun kodları, I/J birleşmesinin ardından K'den itibaren bizimkinden bir miktar kayar: örneğin orijinal tabloda T = *baaba* iken modern tabloda T = *baabb*'dir. Bu bölümde Tablo 6.1'daki modern 26 harfli sürümü kullanacağız.

Kodlamanın kendisi mekanik bir sözlük işleminden ibarettir: mesajın her harfi tablodan karşılığıyla değiştirilir.

Örnek 6.1. `MAT` kelimesini Bacon koduyla kodlayınız.

Çözüm.
Tablo 6.1′dan her harfin kod sözcüğünü sırayla okuyalım:

1. **M** — alfabede 12 numaralı harf → `abbaa`
2. **A** — 0 numaralı harf → `aaaaa`
3. **T** — 19 numaralı harf → `baabb`

Sonuç: `MAT` kelimesinin Bacon kodu, beşerli üç grup hâlinde

```
abbaa aaaaa baabb
```

dizisidir. Toplam $3 \times 5 = 15$ sembol kullandık.
⊠

6.2 Asıl Sihir: Mesajı Görünmez Kılmak

Buraya kadar yaptığımız şeyin güvenlikle pek ilgisi yok: `abbaa aaaaa baabb` dizisini bir kâğıda yazıp gönderirseniz, bu tuhaf dizi şifreli bir mesaj taşıdığınızı dünyaya ilan eder. Bacon'ın dehası ikinci katmandadır: **bu diziyi hiç göndermezsiniz.**
Bunun yerine, gizlenecek her harf için en az beş harf içeren, tamamen masum herhangi bir **taşıyıcı metin (cover text)** seçilir. Taşıyıcı metnin her harfi, birbirinden **belli belirsiz farklı iki biçimden** biriyle dizilir: Bacon'ın önerisinde bunlar birbirine çok benzeyen iki matbaa yazı biçimiydi. Sistemi bilen alıcı, harflerin *ne söylediğine* değil *nasıl basıldığına* bakar: birinci biçimdeki her harfi a, ikinci biçimdeki her harfi b olarak okur, elde ettiği diziyi beşerli gruplar ve tablodan çözer.
Taşıyıcı metnin içeriği tamamen önemsizdir — alışveriş listesi de olabilir, hava durumu sohbeti de. Mesaj harflerde değil, **harflerin biçimlerinde** yaşar.
Aşağıdaki örnekte iki biçimi gözle kolay seçilsin diye abartılı biçimde ayırıyoruz: normal harf a, **kalın (bold)** harf b anlamına gelsin. Gerçek bir uygulamada fark bundan çok daha ince olurdu.

Örnek 6.2. `MAT` mesajını, `Kediler süt sever` taşıyıcı cümlesinin içine Bacon yöntemiyle gizleyiniz.

Çözüm.
Adım 1 — Kapasite kontrolü. Gizlenecek mesaj 3 harf olduğundan $3 \times 5 = 15$ yuvaya ihtiyacımız var. Taşıyıcı cümlede boşluklar sayılmaz: `Kediler süt sever` tam 15 harf içerir — kapasite tam yetiyor.
Adım 2 — Kod dizisini harflerin üzerine sermek. Örnek 6.1′da bulduğumuz `abbaa aaaaa baabb` dizisini, taşıyıcının 15 harfiyle soldan sağa eşleştirelim (kelime sınırlarının gruplarla örtüşmesi gerekmez):

```
K  e  d  i  l  e  r     s  ü  t     s  e  v  e  r
a  b  b  a  a  a  a     a  a  a     b  a  a  b  b
```

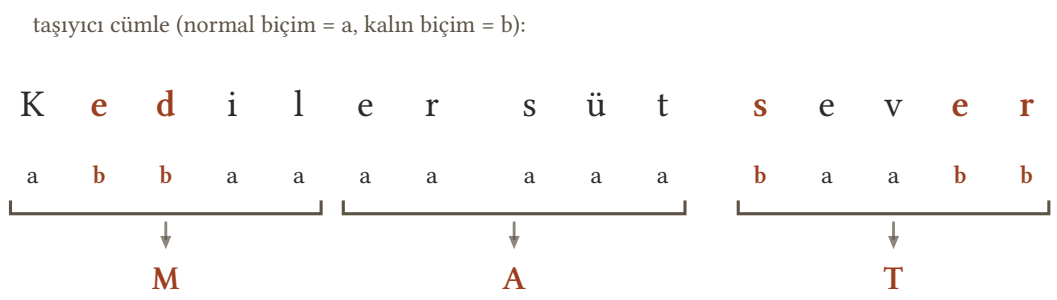
Adım 3 — Biçimleri uygulamak. b'ye denk gelen harfleri kalın, a'ya denk gelenleri normal dizersek masum cümlemiz ortaya çıkar:

K**ed**iler süt **sev****er**

Eve'in gördüğü tek şey kedilerle ilgili sıradan bir cümledir.
Adım 4 — Alıcının çözümü. Sistemi bilen Bob ters yönde ilerler: harflerin biçimlerinden a/b dizisini çıkarır, beşerli gruplar ve Tablo 6.1′dan çözer:

$$\underbrace{abbaa}_{\mathbf{M}}\ \underbrace{aaaaa}_{\mathbf{A}}\ \underbrace{baabb}_{\mathbf{T}}$$

Gizli mesaj geri geldi: `MAT`.



Şekil 6.2: Gizli kanalın görünür hâli: taşıyıcı cümlenin her harfi iki biçimden birinde dizilir (normal = a, kalın = b). Harf biçimlerinden okunan a/b dizisi beşerli gruplandığında `abbaa aaaaa baabb`, yani MAT mesajı ortaya çıkar.

⊠

6.3 Güvenlik ve Tarihsel Önemi

Güvenlik gözlüğümüzü takalım. Bacon'ın sisteminde dikkat çekici bir eksik var: **anahtar yok.** Kod tablosu sabittir ve iki biçimin hangisinin a hangisinin b olduğu sistemin kendisinin parçasıdır. Yöntemi bilen *herkes*, ele geçirdiği metindeki biçim farkını görebildiği anda mesajı çözer. Kerckhoffs prensibi diliyle söylersek: sistemin bütün güvenliği, algoritmanın — daha doğrusu ortada bir mesaj olduğunun — gizli kalmasına dayanır. Gizleme bir kez fark edildiğinde geriye hiçbir koruma kalmaz. Bu yüzden modern kriptografi steganografiyi şifrelemenin *yerine* değil, olsa olsa *yanına* koyar.

Buna karşılık yöntemin düşünsel mirası, güvenlik değerinin kat kat üzerindedir. Bacon'ın yaptığı şeye bir kez daha bakın: alfabenin her harfini, **yalnızca iki değer alabilen beş sembole** indirgedi. Bu, harfleri iki durumlu sinyallere kodlama fikrinin — yani ikili kodlamanın — zamanından 250 yıldan fazla önce ortaya konmuş hâlidir. On dokuzuncu yüzyılın sonunda telgraf için geliştirilen ve her harfi beş adet iki durumlu sinyalle temsil eden **Baudot kodu** ile bilgisayar çağının **ASCII**'si, Bacon'ın bu fikrinin doğrudan torunlarıdır. Aynı fikri 0 ve 1'lerle, modern kriptografiye geçişte yeniden karşılayacağız.

Son bir uyarı notu: Bacon'ın şifresinin ünü zamanla bilim dışı sulara da taşmış, on dokuzuncu ve yirminci yüzyılda "Baconcı" araştırmacılar Shakespeare'in eserlerini aslında Bacon'ın yazdığını kanıtlamak umuduyla First Folio baskılarının harf biçimlerinde bu şifreyi aramışlardır — yeterince istekli bir gözün istatistiksel titizlik olmadan her yerde "gizli mesaj" bulabileceğini gösteren ibretlik bir hikâye.

7. Kriptosistemin Formal Tanımı

Matematiksel şifreleme algoritmalarına (Sezar, Affine, Hill vb.) geçmeden önce, bir şifreleme sisteminin evrensel anatomisini tanımlamamız gerekir. Tüm modern ve klasik kriptosistemler matematiksel olarak **beş bileşenli bir yapı** ile ifade edilir.

Tanım 7.1 (Kriptosistem (Şifreleme Sistemi)). Bir kriptosistem, aşağıdaki koşulları sağlayan bir $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ beşlisidir:

- $\mathcal{P}$ **(Plaintext):** Açık metinlerin (şifrelenmemiş orijinal mesajların) oluşturduğu sonlu küme.
- $\mathcal{C}$ **(Ciphertext):** Şifreli (kapalı) metinlerin oluşturduğu sonlu küme.
- $\mathcal{K}$ **(Key Space):** Anahtar uzayı; kullanılabilecek olası tüm anahtarların oluşturduğu sonlu küme.
- $\mathcal{E}$ **(Encryption):** Şifreleme (kapama) fonksiyonları kümesi.
- $\mathcal{D}$ **(Decryption):** Deşifreleme (açma) fonksiyonları kümesi.

7.1 Fonksiyonların Çalışma Prensibi

Her $k \in \mathcal{K}$ anahtarı için, bir şifreleme kuralı $e_k \in \mathcal{E}$ ve buna karşılık gelen bir deşifreleme kuralı $d_k \in \mathcal{D}$ tanımlanır:

$$e_k : \mathcal{P} \to \mathcal{C}$$

$$d_k : \mathcal{C} \to \mathcal{P}$$

Sistemin tutarlı olabilmesi için, seçilen her bir $k \in \mathcal{K}$ anahtarı ve her bir $x \in \mathcal{P}$ açık metin parçası için, şifrelenmiş metnin tekrar geri açılabileceğini garanti eden şu koşul sağlanmalıdır:

$$(d_k \circ e_k)(x) = d_k\left(\, e_k(x) \,\right) = x, \quad \forall x \in \mathcal{P}$$

> **i** Birebirlik (injective) şartı
>
> Şifreleme fonksiyonu e_k, matematiksel olarak kesinlikle **birebir** olmalıdır.
> Eğer fonksiyon birebir olmazsa ve farklı iki açık metin ($x_1 \neq x_2$) aynı şifreli metne dönüşürse:
>
> $$e_k(x_1) = e_k(x_2)$$
>
> şifreyi çözen kişi, elindeki şifreli metni deşifre ettiğinde orijinal metnin x_1 mi yoksa x_2 mi olduğunu bilemez. Benzersiz bir çözüm elde edilebilmesi için birebirlik şarttır.

7.2 Kümelerin Somutlaştırılması: Alfabe ve Mod 26

Matematiksel şifreleme fonksiyonlarının harfler üzerinde işlem yapabilmesi için harfleri sayılara dönüştürmemiz gerekir. Bu notlardaki tüm klasik algoritmalarda uluslararası standart olan **26 harfli İngiliz alfabesi** kullanılır. Türkçeye özgü karakterler (ç, ğ, ı, ö, ş, ü) alfabe boyutunu değiştirip modüler aritmetiği ve standart ASCII tablolarını bozduğu için denklemlere dâhil edilmez.
Bu bağlamda, açık metin ($\mathcal{P}$) ve şifreli metin ($\mathcal{C}$) kümelerimiz $\mathbb{Z}_{26}$ olarak tanımlanır:

$$\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26} = \{0, 1, 2, \ldots, 25\}$$

7.2.1 Harf–Sayı Dönüşüm Tablosu

İşlemlerde sıfırdan başlama (0-index) kuralı geçerlidir: **A** harfi 0, **Z** harfi ise 25 değerini alır.

Tablo 7.1: Klasik şifreleme algoritmalarında kullanılan harf–sayı karşılıkları

Harf	Değer	Harf	Değer
A	0	**N**	13
B	1	**O**	14
C	2	**P**	15
D	3	**Q**	16
E	4	**R**	17
F	5	**S**	18
G	6	**T**	19
H	7	**U**	20
I	8	**V**	21
J	9	**W**	22
K	10	**X**	23
L	11	**Y**	24
M	12	**Z**	25

i Hatırlatma: modüler aritmetik ve tablo kullanımı

Klasik şifreleme algoritmalarının tamamında (Sezar, Affine, Vigenère, Hill) matematiksel işlemler Tablo 7.1 referans alınarak ve sonuçlar her zaman **mod 26**'ya göre hesaplanarak yürütülecektir. Bir işlem sonucu negatif çıkarsa veya 25'i geçerse, sonucun mod 26'daki denki bulunarak tabloya geri dönülür.

8. Sezar Şifrelemesi (Caesar Cipher)

Klasik kriptografinin en bilinen ve tarihsel olarak en eski algoritmalarından biri olan Sezar şifrelemesi, adını onu generalleriyle gizli mesajlaşmak için kullanan Romalı komutan Julius Sezar'dan alır. Orijinal kullanımında Sezar, alfabedeki her harfi daima **üç harf ileri** kaydırarak mesajlarını gizlemiştir.
Günümüzde "Sezar şifresi" terimi, sadece üç birimlik kaydırmayı değil; harflerin alfabe üzerinde belirli ve sabit bir sayı kadar ileri veya geri kaydırıldığı **genelleştirilmiş kaydırma şifrelerini (shift ciphers)** ifade etmek için kullanılır.

8.1 Sistemin Formal Tanımı

Tanım 8.1 (Sezar Şifrelemesinin Matematiksel Modeli). Bir önceki bölümde tanımladığımız beş bileşenli kriptosistem anatomisini $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, İngiliz alfabesi kullanarak mod 26 üzerinde Sezar algoritmasına uyarlarsak şu yapıyı elde ederiz:

- **Açık metin ($\mathcal{P}$):** $\mathbb{Z}_{26} = \{0, 1, 2, ..., 25\}$
- **Şifreli metin ($\mathcal{C}$):** $\mathbb{Z}_{26} = \{0, 1, 2, ..., 25\}$
- **Anahtar uzayı ($\mathcal{K}$):** $\mathbb{Z}_{26}$ — kullanılabilecek kaydırma miktarları.
- **Şifreleme fonksiyonu ($\mathcal{E}$):** Her $k \in \mathcal{K}$ ve $x \in \mathcal{P}$ için şifreleme, açık metin karakterine anahtar değerinin eklenmesiyle yapılır:

$$e_k(x) \equiv x + k \pmod{26}$$

- **Deşifreleme fonksiyonu ($\mathcal{D}$):** Her $y \in \mathcal{C}$ için açma işlemi, şifreli karakterden anahtar değerinin çıkarılmasıyla yapılır:

$$d_k(y) \equiv y - k \pmod{26}$$

8.2 Anahtar Uzayının Boyutu ve Güvenlik Zafiyeti

Sezar şifrelemesinde anahtar uzayının büyüklüğü **doğrudan seçilen alfabenin boyutuna bağlıdır**. Şifreleme uzayımız 26 harfli İngiliz alfabesiyle sınırlandırıldığı için, harfleri kaydırabileceğimiz farklı miktar sayısı da 26'dır.

> **i** Toplam anahtar uzayı
>
> İngiliz alfabesi ($m = 26$) için Sezar şifrelemesinin anahtar uzayı:
>
> $$| \mathcal{K} | = 26$$
>
> Ancak $k = 0$ durumu metni hiç değiştirmeyeceği için, etkili şifreleme yapan anahtar sayısı **25**'tir.

Bu neden kritik bir sorundur? Kerckhoffs prensibini hatırlayalım: düşmanın şifreleme algoritmasını (her harfe $x + k \pmod{26}$ uygulandığını) bildiği kabul edilir; sistemin güvenliği yalnızca k anahtarının gizliliğine dayanmalıdır. Fakat olası yalnızca 25 anahtar bulunduğundan, düşman şifreli bir metni ele geçirdiğinde hiçbir zekice matematiğe başvurmadan **kaba kuvvet saldırısı (brute force attack)** ile tüm ihtimalleri deneyebilir. Bir insan bile kâğıt kalemle birkaç dakikada 25 kaydırmanın tamamını test edip anlamlı metni bulabilir.
Modern kriptografide güvenli kabul edilen sistemlerin anahtar uzayı (örneğin AES-256'da 2^{256}) astronomik seviyededir. Sezar'ın alfabe boyutuna bağımlı 25'lik anahtar uzayı ise modern standartlarda "yok" hükmündedir.

8.3 Çözümlü Uygulamalar

Aşağıdaki örneklerde, bir önceki bölümdeki harf–sayı dönüşüm tablosunu referans alacağız. Çözümlere bakmadan önce tabloyu kullanarak denklemleri kendiniz kurmayı deneyin.

Örnek 8.1. Açık metni `MATH` olan bir mesajı, $k = 7$ anahtarını kullanarak Sezar şifrelemesi ile şifreleyiniz.

Çözüm.
Algoritmamız: $e_7(x) \equiv x + 7 \pmod{26}$.
Her bir harfi sırayla tablodan sayısal karşılığına çevirip denklemde yerine koyalım:

1. $\mathbf{M} \to 12 \Longrightarrow 12 + 7 = 19 \Longrightarrow \mathbf{T}$
2. $\mathbf{A} \to 0 \Longrightarrow 0 + 7 = 7 \Longrightarrow \mathbf{H}$
3. $\mathbf{T} \to 19 \Longrightarrow 19 + 7 = 26 \equiv 0 \pmod{26} \Longrightarrow \mathbf{A}$ *(modüler döngü burada devreye girdi)*
4. $\mathbf{H} \to 7 \Longrightarrow 7 + 7 = 14 \Longrightarrow \mathbf{O}$

Sonuç: `MATH` kelimesi, $k = 7$ anahtarıyla `THAO` olarak şifrelenir.
$\boxtimes$

Örnek 8.2. $k = 5$ anahtarı ile şifrelenmiş olan `GTD` kapalı metnini deşifre ediniz.

Çözüm.
Algoritmamız: $d_5(y) \equiv y - 5 \pmod{26}$.

1. $\mathbf{G} \to 6 \Longrightarrow 6 - 5 = 1 \Longrightarrow \mathbf{B}$
2. $\mathbf{T} \to 19 \Longrightarrow 19 - 5 = 14 \Longrightarrow \mathbf{O}$
3. $\mathbf{D} \to 3 \Longrightarrow 3 - 5 = -2$. Eksi bir sonuç bulduğumuzda, pozitif denki bulana kadar modülü ekleriz:

$$-2 \equiv -2 + 26 \equiv 24 \pmod{26} \Longrightarrow \mathbf{Y}$$

Sonuç: Şifreli `GTD` metninin açık hâli `BOY` kelimesidir.
$\boxtimes$

Örnek 8.3. Düşman haberleşme hattından `WKLV` şifreli metnini ele geçirdiniz. Bunun aslında İngilizce `THIS` kelimesi olduğu bilindiğine göre, kullanılan Sezar anahtarını (k) cebirsel olarak bulunuz.

Çözüm.
Şifreleme kuralının $e_k(x) \equiv y \pmod{26}$ olduğunu biliyoruz. İlk harf olan **T** (açık metin) ile **W** (şifreli metin) arasındaki denklemi kurarak doğrudan çözüme ulaşabiliriz.
Tabloya göre:

- $\mathbf{T} \to 19$ (açık metin, x)
- $\mathbf{W} \to 22$ (şifreli metin, y)

Denklemi kuralım:

$$19 + k \equiv 22 \pmod{26}$$

$$k \equiv 22 - 19 \pmod{26}$$

$$k \equiv 3 \pmod{26}$$

Doğrulama. Bulduğumuz $k = 3$ anahtarının diğer harfler için de çalışıp çalışmadığını test edelim:

- $\mathbf{H}\ (7) + 3 = 10 \Longrightarrow \mathbf{K}$ ✓
- $\mathbf{I}\ (8) + 3 = 11 \Longrightarrow \mathbf{L}$ ✓
- $\mathbf{S}\ (18) + 3 = 21 \Longrightarrow \mathbf{V}$ ✓

Sonuç: Düşmanın kullandığı anahtar $k = 3$'tür — yani orijinal Julius Sezar anahtarı.
$\boxtimes$

8.4 İnteraktif Sezar Hesaplayıcı

Yukarıda öğrendiğimiz şifreleme ve deşifreleme fonksiyonlarının ($x \pm k \pmod{26}$) gerçek zamanlı olarak nasıl çalıştığını görmek için aşağıdaki aracı kullanabilirsiniz. Farklı mesajlar yazıp anahtar (k) değerini kaydırarak metnin şifreli uzayda nasıl evrildiğini test edin.
(Bu etkileşimli bileşen yalnızca web sürümünde çalışır.)

9. Affine (Afin) Şifreleme

Sezar şifrelemesi, harfleri yalnızca sabit bir miktar kaydırarak (toplama işlemiyle) gizler. Affine (afin) şifreleme ise bu mantığı bir adım ileri taşıyarak, modüler aritmetik üzerinde hem **çarpma** hem de **toplama** işlemini aynı anda kullanan doğrusal bir yerine koyma algoritmasıdır.
Kısacası, klasik cebirdeki $y = ax + b$ doğru denkleminin alfabe (mod 26) üzerine inşa edilmiş hâlidir.

9.1 Sistemin Formal Tanımı

Afin şifrelemesinde anahtarımız artık tek boyutlu bir skaler değil, $k = (a, b)$ şeklinde tanımlanan sıralı bir sayı ikilisidir.

Tanım 9.1 (Afin Şifrelemesinin Matematiksel Modeli).

- **Açık metin ($\mathcal{P}$):** $\mathbb{Z}_{26} = \{0, 1, 2, ..., 25\}$
- **Şifreli metin ($\mathcal{C}$):** $\mathbb{Z}_{26} = \{0, 1, 2, ..., 25\}$
- **Anahtar uzayı ($\mathcal{K}$):** $k = (a, b)$ formunda olmak üzere; $a, b \in \mathbb{Z}_{26}$ ve $\gcd(a, 26) = 1$ koşulunu sağlayan tüm sıralı ikililerin kümesi.
- **Şifreleme fonksiyonu ($\mathcal{E}$):** Her $k = (a, b) \in \mathcal{K}$ ve $x \in \mathcal{P}$ için:

$$e_k(x) \equiv a \cdot x + b \pmod{26}$$

- **Deşifreleme fonksiyonu ($\mathcal{D}$):** Her $k = (a, b) \in \mathcal{K}$ ve $y \in \mathcal{C}$ için, fonksiyonun tersi alınarak bulunur. Burada a^{-1}, a'nın mod 26'ya göre çarpımsal tersidir:

$$d_k(y) \equiv a^{-1} \cdot (y - b) \pmod{26}$$

9.2 Anahtar Seçme Koşulları ve Birebirlik Şartı

Afin şifrelemesinde b (kaydırma) değeri için $\mathbb{Z}_{26}$ içindeki tüm sayılar seçilebilir. Ancak a (çarpan) değeri rastgele seçilemez.
Şifreleme fonksiyonunun çözülebilmesi için **mutlaka birebir olması** gerekir. Eğer a sayısı 26 ile aralarında asal değilse, yani $\gcd(a, 26) \neq 1$ ise, fonksiyon birebirliğini kaybeder.

! Örnek bir felaket senaryosu

Kuralı ihlal edip $\gcd(2, 26) = 2 \neq 1$ olmasına rağmen $k = (2, 3)$ seçtiğimizi varsayalım. Denklemimiz $e_k(x) \equiv 2x + 3 \pmod{26}$ olur:

- **A** harfi (0) şifrelendiğinde: $2 \cdot 0 + 3 \equiv 3 \implies$ **D**
- **N** harfi (13) şifrelendiğinde: $2 \cdot 13 + 3 = 29 \equiv 3 \pmod{26} \implies$ **D**

Hem A hem de N harfi aynı şifreli harfe dönüştü. Mesajı alan kişi **D** harfini deşifre etmek istediğinde orijinal harfin A mı yoksa N mi olduğunu asla bilemez; sistem çöker.

i Toplam anahtar uzayı ve Euler'in phi fonksiyonu

Afin şifrelemesinde a değerleri için modül m ile aralarında asal olan sayıların adedini bulmamız gerekir; bu değer **Euler'in phi fonksiyonu (ϕ)** ile hesaplanır. b değeri için ise modül kadar seçenek vardır. Dolayısıyla afin kriptosisteminin toplam anahtar uzayı:

$$|\ \mathcal{K}\ | = \phi(m) \cdot m$$

İngiliz alfabesi ($m = 26$) için 26 ile aralarında asal olan sayıların adedi:

$$\phi(26) = \phi(2) \cdot \phi(13) = (2-1) \cdot (13-1) = 12$$

Geçerli olan bu 12 çarpan şunlardır:

$$a \in \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$$

a için 12, b için 26 seçenek olduğundan toplam anahtar uzayı:

$$|\ \mathcal{K}\ | = 12 \cdot 26 = 312$$

Sezar'ın 25'lik anahtar uzayına göre daha geniş görünse de, $|\ \mathcal{K}\ | = 312$ modern bilgisayarlar için kaba kuvvet saldırılarıyla saniyeler içinde taranabilir.

9.3 Çözümlü Uygulamalar

Aşağıdaki örneklerde işlemleri harf–sayı dönüşüm tablosunu kullanarak yapınız.

Örnek 9.1. Açık metni `MATH` olan bir mesajı, $k = (5, 8)$ anahtarını kullanarak afin şifrelemesi ile şifreleyiniz.

Çözüm.
Şifreleme kuralımız: $e_k(x) \equiv 5x + 8 \pmod{26}$.
Harfleri tablodan sayıya çevirip denklemde yerine koyalım:

1. **M** $\to 12 \Longrightarrow 5 \cdot 12 + 8 = 68 \equiv 16 \pmod{26} \Longrightarrow$ **Q**
2. **A** $\to 0 \Longrightarrow 5 \cdot 0 + 8 = 8 \Longrightarrow$ **I**
3. **T** $\to 19 \Longrightarrow 5 \cdot 19 + 8 = 103 \equiv 25 \pmod{26} \Longrightarrow$ **Z**
4. **H** $\to 7 \Longrightarrow 5 \cdot 7 + 8 = 43 \equiv 17 \pmod{26} \Longrightarrow$ **R**

Sonuç: `MATH` kelimesi `QIZR` olarak şifrelenir.
$\boxtimes$

Örnek 9.2. $k = (7, 2)$ anahtarı ile şifrelenmiş olan `QCF` kapalı metnini deşifre ediniz.

Çözüm.
Deşifreleme formülümüz: $d_k(y) \equiv a^{-1} \cdot (y - b) \pmod{26}$.
Öncelikle $a = 7$ sayısının mod 26'daki çarpımsal tersini bulmalıyız; yani $7x \equiv 1 \pmod{26}$ denkliğini sağlayan x'i arıyoruz. $7 \cdot 15 = 105 = 4 \cdot 26 + 1$ olduğundan:

$$7^{-1} \equiv 15 \pmod{26}$$

Yeni deşifreleme denklemimiz $d_k(y) \equiv 15 \cdot (y - 2) \pmod{26}$ olur:

1. **Q** $\to 16 \Longrightarrow 15 \cdot (16 - 2) = 15 \cdot 14 = 210 \equiv 2 \pmod{26} \Longrightarrow$ **C**
2. **C** $\to 2 \Longrightarrow 15 \cdot (2 - 2) = 0 \Longrightarrow$ **A**
3. **F** $\to 5 \Longrightarrow 15 \cdot (5 - 2) = 15 \cdot 3 = 45 \equiv 19 \pmod{26} \Longrightarrow$ **T**

Sonuç: Şifreli `QCF` metninin açık hâli `CAT` kelimesidir.
$\boxtimes$

Örnek 9.3. Düşmandan ele geçirilen AGIY şifreli metninin orijinalinde OKAY kelimesi olduğu bilinmektedir. Kullanılan (a, b) afin anahtarını bulunuz.

Çözüm.
$e_k(x) \equiv a \cdot x + b \pmod{26}$ olduğunu biliyoruz. İlk iki harf üzerinden iki bilinmeyenli bir denklem sistemi kurabiliriz.
O $(14) \to$ **A** (0):

$$14a + b \equiv 0 \pmod{26} \tag{1}$$

K $(10) \to$ **G** (6):

$$10a + b \equiv 6 \pmod{26} \tag{2}$$

b'yi yok etmek için (1) numaralı denklemden (2) numaralıyı taraf tarafa çıkaralım:

$$(14a + b) - (10a + b) \equiv 0 - 6 \pmod{26}$$

$$4a \equiv -6 \equiv 20 \pmod{26}$$

Kritik aşama. Mod 26'da $4a \equiv 20$ denkliğini çözerken doğrudan 4'e bölemeyiz. $\gcd(4, 26) = 2$ ve $2 \mid 20$ olduğundan bu denkliğin mod 26'da tam olarak iki kökü vardır. Her iki tarafı ve modülü 2'ye bölersek $2a \equiv 10 \pmod{13}$, buradan $a \equiv 5 \pmod{13}$ elde edilir. Dolayısıyla:

1. $a \equiv 5 \pmod{26}$
2. $a \equiv 5 + 13 = 18 \pmod{26}$

Ancak afin kuralları gereği $\gcd(a, 26) = 1$ olmak zorundadır. $\gcd(18, 26) = 2 \neq 1$ olduğundan $a = 18$ değeri **geçersizdir**; demek ki $a = 5$ olmalıdır.
Bulduğumuz $a = 5$ değerini (1) numaralı denklemde yerine koyalım:

$$14 \cdot 5 + b \equiv 0 \pmod{26}$$

$$70 + b \equiv 0 \pmod{26}$$

$$18 + b \equiv 0 \pmod{26}$$

$$b \equiv -18 \equiv 8 \pmod{26}$$

Sonuç: Düşmanın kullandığı anahtar $k = (5, 8)$ ikilisidir — yani Örnek 9.1 içinde kullandığımız anahtarın aynısı.
Doğrulama: OKAY metnini $k = (5, 8)$ ile şifreleyelim. O $(14) \to 78 \equiv 0 \Longrightarrow$ A, K $(10) \to 58 \equiv 6 \Longrightarrow$ G, A $(0) \to 8 \Longrightarrow$ I, Y $(24) \to 128 \equiv 24 \Longrightarrow$ Y. Gerçekten AGIY elde edilir.
$\boxtimes$

9.4 İnteraktif Afin Hesaplayıcı

$e_k(x) \equiv a \cdot x + b \pmod{26}$ denkleminin pratikte nasıl çalıştığını aşağıdaki araçla test edebilirsiniz. Dikkat ederseniz, fonksiyonun birebir olma şartını korumak için **çarpan (a)** menüsünde yalnızca 26 ile aralarında asal olan o 12 geçerli anahtar yer almaktadır. Yanlarında modüler tersleri (a^{-1}) de hesaplanmıştır.
(Bu etkileşimli bileşen yalnızca web sürümünde çalışır.)

10. Alfabe Permütasyonu (Yerine Koyma) Şifrelemesi

Soyut cebirsel perspektiften bakıldığında, klasik şifreleme yöntemlerinin büyük bir kısmı alfabe kümesinin kendi üzerindeki birebir ve örten dönüşümleriyle, yani permütasyonlarıyla ifade edilir. Kriptografi literatüründe **monoalfabetik yerine koyma şifresi** olarak bilinen bu yapı, matematiksel olarak tamamen bir **simetrik grup (S_n)** işlemidir.
Bu sistemde harflerin metin içindeki konumları sabit kalır; ancak her karakterin alfabedeki sayısal değeri bir π permütasyon fonksiyonuna sokularak karakterin kimliği değiştirilir.

Tanım 10.1 (Simetrik Grup (S_n)). Boş olmayan bir A kümesinin kendi üzerine tanımlı tüm **birebir ve örten** fonksiyonlarının (permütasyonlarının) bileşke işlemi ($\circ$) altında oluşturduğu gruba **simetrik grup** denir ve Sym (A) ile gösterilir.
Eğer A kümesi n elemanlı sonlu bir küme ise (örneğin $A = \{0, 1, 2, ..., n-1\}$), bu grup S_n sembolüyle ifade edilir. Grubun mertebesi, elemanların tüm olası dizilimlerinin sayısına eşittir:

$$\mid S_n \mid = n!$$

Simetrik gruplar, sonlu bir kümenin elemanlarının tüm olası yeniden dizilimlerini tek bir cebirsel çatı altında toplar. Kriptografik açıdan bu grup, bir alfabedeki harflerin yerini değiştirmek için kullanılabilecek tüm **geçerli ve kapalı** eşleme kurallarının kümesidir. Küme büyüdükçe olası dizilimlerin sayısı faktöriyel hızında artarak devasa bir kombinatoryal uzay meydana getirir.
Şimdi küçük bir örnekle simetrik grubun soyut yapısını somutlaştırıp, iki satırlı matris gösterimiyle permütasyon fonksiyonlarını inceleyelim.

Örnek 10.1 (Simetrik Grup S_6 ve İki Satırlı Matris Gösterimi). Kümemiz $A = \{0, 1, 2, 3, 4, 5\}$ olsun. Bu küme üzerindeki tüm permütasyonlar S_6 grubunu oluşturur ve grubun eleman sayısı $\mid S_6 \mid = 6! = 720$'dir. Bu gruptan bir π elemanını ayrık döngü formunda şöyle tanımlayalım:

$$\pi = (0\,3\,5)(1\,4)(2)$$

Permütasyonun elemanları nasıl eşlediğini sistematik görmek için **iki satırlı matris gösterimi** kullanılabilir. Üst satıra kümenin orijinal elemanları sıralı yazılır, alt satıra bu elemanların π altındaki görüntüleri yerleştirilir:

$$\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 2 & 5 & 1 & 0 \end{pmatrix}$$

Hem matris gösterimi hem de döngü tanımı üzerinden fonksiyonun değerleri şöyle elde edilir:

- $\pi(0) = 3$ — matriste altındaki değer 3; döngüde 0'dan hemen sonra 3 gelir.
- $\pi(1) = 4$ — matriste altındaki değer 4; döngüde 1'den hemen sonra 4 gelir.
- $\pi(2) = 2$ — tek elemanlı bir döngü olduğu için fonksiyon altında sabittir.
- $\pi(3) = 5$ — döngüde 3'ten hemen sonra 5 gelir.
- $\pi(4) = 1$ — $(1\,4)$ döngüsünün sonundan başa dönülür.
- $\pi(5) = 0$ — $(0\,3\,5)$ döngüsünün sonundan başa dönülür.

Simetrik grupların bu soyut yapısı, klasik kriptolojide **yerine koyma (substitution)** şifrelerinin temel direğidir. Alfabedeki her harfi sayısal bir indeksle eşleştirdiğimizde, bir metni şifrelemek aslında alfabenin elemanlarını rastgele seçilmiş bir π permütasyonuyla karıştırmaktan ibarettir.
Şifrelemenin benzersiz ve tamamen geri döndürülebilir olabilmesi için seçilen fonksiyonun simetrik grubun bir elemanı olması şarttır. Fonksiyon birebir ve örten olmazsa birden fazla harf aynı şifreli karaktere eşleneceği için geriye dönük benzersiz bir deşifre anahtarı (π^{-1}) üretilemez.

10.1 Yerine Koyma Şifrelemesinin Formal Tanımı

Tanım 10.2 (Alfabe Permütasyonu Kriptosistemi). İngiliz alfabesi üzerinde ($\mathbb{Z}_{26}$) çalıştığımızı varsayarsak, beş bileşenli kriptosistem anatomisi şöyle tanımlanır:

- **Açık metin ($\mathcal{P}$):** Elemanları $\mathbb{Z}_{26} = \{0, 1, 2, \ldots, 25\}$ kümesinden seçilen karakter dizileri.
- **Şifreli metin ($\mathcal{C}$):** Elemanları $\mathbb{Z}_{26}$ kümesinden seçilen karakter dizileri.
- **Anahtar uzayı ($\mathcal{K}$):** $\mathbb{Z}_{26}$ kümesi üzerindeki tüm olası permütasyonların oluşturduğu **simetrik grup** S_{26}.
- **Şifreleme fonksiyonu ($\mathcal{E}$):** Açık metindeki τ. karakterin sayısal değeri x_τ, seçilen π permütasyonuna girdi olarak verilir:

$$e_\pi(x_\tau) = \pi(x_\tau)$$

- **Deşifreleme fonksiyonu ($\mathcal{D}$):** Şifreli değeri çözmek için permütasyonun tersi kullanılır:

$$d_\pi(y_\tau) = \pi^{-1}(y_\tau)$$

10.2 Anahtar Uzayı ve Güvenlik Analizi

Permütasyon tabanlı bu şifrenin anahtar uzayı, S_{26} simetrik grubunun mertebesine eşittir:

$$\mid S_{26} \mid = 26! \approx 4,03 \times 10^{26}$$

Bu değer, modern AES-128 şifrelemesindeki anahtar sayısına ($2^{128} \approx 3,4 \times 10^{38}$) kıyasla küçük kalsa da, klasik dönemin kaba kuvvet saldırıları için aşılamaz bir büyüklüktü.
Ancak sistem, harflerin kimliğini **statik** biçimde değiştirdiği için doğal dilin istatistiksel yapısını gizleyemez: bir metindeki `E` harfinin frekansı neyse, şifreli karşılığının frekansı da aynı kalır. Bu yüzden devasa anahtar uzayına rağmen **frekans analiziyle** çok kısa sürede kırılır. Anahtar uzayının büyüklüğünün tek başına güvenlik anlamına gelmediğinin en çarpıcı örneğidir.

10.3 Döngü Gösterimi ve Sabit Elemanlar Kuralı

Soyut cebirde permütasyonlar genellikle hantal matris gösterimleri yerine **ayrık döngülerin çarpımı** şeklinde yazılır. Bu notasyonda işlem yaparken unutulmaması gereken en hayati kural şudur:

> **i** Sabit eleman kuralı
>
> Eğer $\mathbb{Z}_{26}$'nın bir elemanı, tanımlanan permütasyon döngüleri içinde hiç yer almıyorsa, o eleman permütasyon altında sabit kalır ve **kendisine eşlenir**: $\pi(x) = x$.

Aşağıdaki örneklerde alfabenin bir kısmını kapsayan bir permütasyon anahtarı tanımlanmış, bazı sayılar döngülere bilerek dâhil edilmeyerek bu kuralın nasıl çalıştığı gösterilmiştir.

10.4 Çözümlü Uygulamalar

Örnek 10.2. $\mathbb{Z}_{26}$ alfabesi üzerinde tanımlı $\pi \in S_{26}$ permütasyon anahtarı ayrık döngüler formunda aşağıda verilmiştir. Bu anahtarı kullanarak `MATHS` açık metnini şifreleyiniz.

$$\pi = (0\ 12\ 2\ 19)(1\ 5\ 8\ 20\ 24)(3\ 14\ 17)(4\ 7\ 11\ 22)$$

Çözüm.
Öncelikle `MATHS` açık metnindeki harflerin sayısal karşılıklarını bulalım:
M $\to 12$, **A** $\to 0$, **T** $\to 19$, **H** $\to 7$, **S** $\to 18$.
Şimdi şifreleme fonksiyonunu ($y_\tau = \pi(x_\tau)$) her değer için uygulayıp döngüdeki haritayı takip edelim:

1. $(0\ 12\ 2\ 19)$ döngüsünde 12'den sonra 2 gelir:

$$\pi(12) = 2 \Longrightarrow \mathbf{C}$$

2. Aynı döngüde 0'dan sonra 12 gelir:

$$\pi(0) = 12 \Longrightarrow \mathbf{M}$$

3. Döngünün son elemanı her zaman ilk elemana döner:

$$\pi(19) = 0 \Longrightarrow \mathbf{A}$$

4. $(4\ 7\ 11\ 22)$ döngüsünde 7'den sonra 11 gelir:

$$\pi(7) = 11 \Longrightarrow \mathbf{L}$$

5. 18 sayısı tanımlanan hiçbir döngüde yer **almamaktadır**; sabit eleman kuralı gereği kendisine eşlenir:

$$\pi(18) = 18 \Longrightarrow \mathbf{S}$$

Sonuç: `MATHS` açık metni, π fonksiyonu altında `CMALS` olarak şifrelenir.
$\boxtimes$

Örnek 10.3. Aynı $\pi = (0\ 12\ 2\ 19)(1\ 5\ 8\ 20\ 24)(3\ 14\ 17)(4\ 7\ 11\ 22)$ permütasyon anahtarı kullanılarak şifrelenmiş olan `CMALS` kapalı metnini deşifre ediniz.

Çözüm.
Deşifreleme için permütasyonun tersini almalıyız. Döngü notasyonunda bir permütasyonun tersi, her döngünün elemanlarını **sondan başa doğru** yazmakla elde edilir:

$$\pi^{-1} = (19\ 2\ 12\ 0)(24\ 20\ 8\ 5\ 1)(17\ 14\ 3)(22\ 11\ 7\ 4)$$

Not: Döngü içinde yer almayan elemanlar ters permütasyonda da sabit kalır.

Şifreli metnimiz `CMALS` harflerinin sayısal değerleri: $\mathbf{C} \to 2$, $\mathbf{M} \to 12$, $\mathbf{A} \to 0$, $\mathbf{L} \to 11$, $\mathbf{S} \to 18$.
Ters haritayı takip edelim:

1. $(19\ 2\ 12\ 0)$ döngüsünde 2'den sonra 12 gelir:

$$\pi^{-1}(2) = 12 \Longrightarrow \mathbf{M}$$

2. Aynı döngüde 12'den sonra 0 gelir:

$$\pi^{-1}(12) = 0 \Longrightarrow \mathbf{A}$$

3. Döngünün sonundan başına döneriz:

$$\pi^{-1}(0) = 19 \Longrightarrow \mathbf{T}$$

4. $(22\ 11\ 7\ 4)$ döngüsünde 11'den sonra 7 gelir:

$$\pi^{-1}(11) = 7 \Longrightarrow \mathbf{H}$$

5. 18 sayısı ters döngülerde de yoktur, kendine eşlenir:

$$\pi^{-1}(18) = 18 \Longrightarrow \mathbf{S}$$

Sonuç: Deşifreleme işlemi kusursuz çalışmış ve orijinal `MATHS` kelimesine ulaşılmıştır.
$\boxtimes$

11. Hill Şifrelemesi

Klasik şifreleme sistemlerinin istatistiksel frekans analizine yenik düşmesini engellemek için tasarlanan Hill şifrelemesi, kriptografiye **lineer cebiri (matrisleri)** sokan ilk büyük devrimdir.
Bu sistemde harfler tek başına değil, belirlenen bir n uzunluğundaki **sütun vektörleri** hâlinde gruplanarak bir anahtar matrisiyle çarpılır. Böylece bir harfin şifreli karşılığı, yanındaki harflere de bağlı hâle gelir; tek harflik frekans analizi işe yaramaz olur.

11.1 Sistemin Formal Tanımı

Hill sisteminde işlemler n boyutlu vektörler ve $n \times n$ boyutlu kare matrisler üzerinden yürütülür. Notlarımızda kolaylık olması adına $n = 2$, yani 2×2 matrisler kullanacağız.

Tanım 11.1 (Hill Şifrelemesinin Matematiksel Modeli).

- **Açık metin ($\mathcal{P}$):** Elemanları $\mathbb{Z}_{26}$'dan alınan n boyutlu sütun vektörlerinin kümesi.
- **Şifreli metin ($\mathcal{C}$):** Elemanları $\mathbb{Z}_{26}$'dan alınan n boyutlu sütun vektörlerinin kümesi.
- **Anahtar uzayı ($\mathcal{K}$):** Elemanları $\mathbb{Z}_{26}$'da olan ve mod 26'ya göre tersi alınabilen tüm $n \times n$ boyutlu kare matrisler.
- **Şifreleme fonksiyonu ($\mathcal{E}$):** Her $K \in \mathcal{K}$ anahtar matrisi ve $\mathbf{x} \in \mathcal{P}$ açık metin vektörü için:

$$e_K(\mathbf{x}) \equiv K \cdot \mathbf{x} \pmod{26}$$

- **Deşifreleme fonksiyonu ($\mathcal{D}$):** Her $\mathbf{y} \in \mathcal{C}$ şifreli metin vektörü için:

$$d_K(\mathbf{y}) \equiv K^{-1} \cdot \mathbf{y} \pmod{26}$$

11.2 Anahtar Seçme Koşulu: Determinant ve Ters Matris

Afin şifrelemesinde anahtarın birebir olabilmesi için a değerinin 26 ile aralarında asal olmasını istemiştik. Hill şifrelemesinde ise anahtar matrisin **tersinin var olması** şarttır.
Bir 2×2 matrisin tersi şu formülle bulunur:

$$K^{-1} \equiv \big(\det K\big)^{-1} \cdot \operatorname{adj}\,(K) \pmod{26}$$

Bu formülün mod 26'da çalışabilmesi için $\det(K)$ değerinin mod 26'da çarpımsal bir tersinin olması gerekir.

i Geçerli bir Hill anahtarı için altın kural

Bir K matrisinin Hill şifrelemesinde anahtar olarak kullanılabilmesi için determinantının 26 ile aralarında asal olması **zorunludur**:

$$\gcd\big(\det(K), 26\big) = 1$$

$26 = 2 \cdot 13$ olduğundan bu, pratikte şu demektir: determinantın mod 26'daki değeri ne **2 ile** ne de **13 ile** bölünebilmelidir. Determinant bu kurala uymuyorsa ters matris hesaplanamaz ve şifrelenen metin bir daha asla geri açılamaz.

i Dolgu (padding) kuralı

Şifrelenecek mesajın harf sayısı, vektör boyutu n'in tam katı değilse, mesajın sonuna anlamsız harfler (genellikle `X` veya `Z`) eklenerek son blok tamamlanır.

11.3 Çözümlü Uygulamalar

Aşağıdaki örneklerde işlemleri harf–sayı dönüşüm tablosunu kullanarak yapınız. Vektör boyutumuz $n = 2$'dir.

Örnek 11.1. Açık metni `HELP` olan mesajı, aşağıdaki K anahtar matrisiyle şifreleyiniz.

$$K = \begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix}$$

Çözüm.
Öncelikle anahtarın geçerli olup olmadığını determinantla kontrol edelim:

$$\det(K) = (3 \cdot 5) - (3 \cdot 2) = 15 - 6 = 9$$

$\gcd(9, 26) = 1$ olduğu için bu geçerli bir anahtardır.
Metnimiz `HELP`, ikişerli bloklara ayrılır: **HE** ve **LP**.
1. blok: HE $(7, 4)$

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 7 \\ 4 \end{pmatrix} = \begin{pmatrix} 3(7) + 3(4) \\ 2(7) + 5(4) \end{pmatrix} = \begin{pmatrix} 21 + 12 \\ 14 + 20 \end{pmatrix} = \begin{pmatrix} 33 \\ 34 \end{pmatrix}$$

Mod 26'ya göre denklerini alalım:

$$\begin{pmatrix} 33 \\ 34 \end{pmatrix} \equiv \begin{pmatrix} 7 \\ 8 \end{pmatrix} \pmod{26}$$

Tabloya göre $7 = \mathbf{H}$, $8 = \mathbf{I}$. İlk şifreli blok: **HI**.
2. blok: LP $(11, 15)$

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 11 \\ 15 \end{pmatrix} = \begin{pmatrix} 33 + 45 \\ 22 + 75 \end{pmatrix} = \begin{pmatrix} 78 \\ 97 \end{pmatrix}$$

Mod 26'ya göre: $78 = 26 \cdot 3 + 0 \equiv 0$ ve $97 = 26 \cdot 3 + 19 \equiv 19$, yani

$$\begin{pmatrix} 78 \\ 97 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 19 \end{pmatrix} \pmod{26}$$

Tabloya göre $0 = \mathbf{A}$, $19 = \mathbf{T}$. İkinci şifreli blok: **AT**.
Sonuç: `HELP` kelimesi Hill şifrelemesiyle `HIAT` olarak şifrelenir.
Dikkat edin: açık metindeki `E` ve `L` tamamen farklı harflere dönüşürken, aralarındaki istatistiksel bağ matris çarpımının içinde erimiştir.
⊠

Örnek 11.2. Şifreli `HIAT` metnini, aynı K anahtar matrisini kullanarak deşifre ediniz.

$$K = \begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix}$$

Çözüm.
Deşifreleme için $d_K(\mathbf{y}) \equiv K^{-1} \cdot \mathbf{y} \pmod{26}$ formülünü kullanacağız. Önce K^{-1} ters matrisini bulmalıyız.
1. adım — determinantın tersi. $\det(K) = 9$ bulmuştuk. 9'un mod 26'daki çarpımsal tersini arıyoruz:

$$9x \equiv 1 \pmod{26} \Longrightarrow x = 3 \quad (\text{çünkü} \quad 9 \cdot 3 = 27 \equiv 1)$$

2. adım — ek (adjoint) matris. 2×2 boyutlu bir $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ matrisinin ek matrisi $\left(\begin{smallmatrix} d & -b \\ -c & a \end{smallmatrix}\right)$'dır:

$$\text{adj }(K) = \begin{pmatrix} 5 & -3 \\ -2 & 3 \end{pmatrix}$$

3. adım — ters matrisi oluşturmak.

$$K^{-1} \equiv 3 \cdot \begin{pmatrix} 5 & -3 \\ -2 & 3 \end{pmatrix} = \begin{pmatrix} 15 & -9 \\ -6 & 9 \end{pmatrix} \pmod{26}$$

Negatif değerleri mod 26'da pozitife çevirelim ($-9 \equiv 17$, $-6 \equiv 20$):

$$K^{-1} \equiv \begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \pmod{26}$$

Sağlama için $K \cdot K^{-1}$ çarpımına bakabiliriz:

$$\begin{pmatrix} 3 & 3 \\ 2 & 5 \end{pmatrix} \begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} = \begin{pmatrix} 105 & 78 \\ 130 & 79 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \pmod{26}$$

4. adım — şifreli blokları ters matrisle çarpmak.
1. blok: HI $(7, 8)$

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \end{pmatrix} = \begin{pmatrix} 105 + 136 \\ 140 + 72 \end{pmatrix} = \begin{pmatrix} 241 \\ 212 \end{pmatrix} \equiv \begin{pmatrix} 7 \\ 4 \end{pmatrix} \pmod{26}$$

$7 \Longrightarrow$ **H**, $4 \Longrightarrow$ **E**.
2. blok: AT $(0, 19)$

$$\begin{pmatrix} 15 & 17 \\ 20 & 9 \end{pmatrix} \begin{pmatrix} 0 \\ 19 \end{pmatrix} = \begin{pmatrix} 0 + 323 \\ 0 + 171 \end{pmatrix} = \begin{pmatrix} 323 \\ 171 \end{pmatrix} \equiv \begin{pmatrix} 11 \\ 15 \end{pmatrix} \pmod{26}$$

$11 \Longrightarrow$ **L**, $15 \Longrightarrow$ **P**.
Sonuç: Matris çarpımları ve mod 26 kuralları bizi tekrar orijinal `HELP` mesajına ulaştırdı.
$\boxtimes$

12. Vigenère Şifrelemesi

Monoalfabetik şifrelerin (Sezar, afin) en zayıf noktası, dildeki harf frekanslarını —örneğin İngilizcede E harfinin diğerlerinden çok daha sık geçmesini— aynen korumasıdır. 16. yüzyılda Giovan Battista Bellaso tarafından tasarlanan ve sonradan yanlışlıkla Blaise de Vigenère'in adıyla anılan bu sistem, **polialfabetik (çoklu alfabe)** yapısıyla bu istatistiksel zafiyeti çözer.
Temelde sistem, tek bir sabit kaydırma yerine bir **anahtar kelime** kullanarak mesajdaki her harf için farklı ve döngüsel bir Sezar kaydırması uygular.

Tanım 12.1 (Vigenère Şifreleme Sistemi). Vigenère şifrelemesinde işlemler, n uzunluğundaki açık metin vektörleri ile m uzunluğundaki anahtar vektörleri (kelimeleri) arasında gerçekleşir:

- **Açık metin ($\mathcal{P}$):** Elemanları $\mathbb{Z}_{26}$'dan alınan n boyutlu vektörlerin kümesi:
$$\mathbf{x} = (x_0, x_1, ..., x_{n-1}) \in (\mathbb{Z}_{26})^n$$
- **Şifreli metin ($\mathcal{C}$):** Elemanları $\mathbb{Z}_{26}$'dan alınan n boyutlu vektörlerin kümesi:
$$\mathbf{y} = (y_0, y_1, ..., y_{n-1}) \in (\mathbb{Z}_{26})^n$$
- **Anahtar uzayı ($\mathcal{K}$):** Elemanları $\mathbb{Z}_{26}$'dan alınan m uzunluğundaki vektörlerden (kelimelerden) oluşan küme:
$$K = (k_0, k_1, ..., k_{m-1}) \in (\mathbb{Z}_{26})^m$$
- **Şifreleme fonksiyonu ($\mathcal{E}$):** $\mathbf{x}$ vektörünün i. elemanı, anahtar vektörünün döngüsel olarak sıraya denk gelen elemanıyla toplanır:
$$e_K(x_i) \equiv x_i + k_{i \bmod m} \pmod{26}$$
- **Deşifreleme fonksiyonu ($\mathcal{D}$):** $\mathbf{y}$ şifreli vektörünün i. elemanından anahtar vektöründeki ilgili eleman çıkarılır:
$$d_K(y_i) \equiv y_i - k_{i \bmod m} \pmod{26}$$

> **i** Anahtar uzayı ve Kasiski incelemesi
>
> Vigenère şifrelemesinin anahtar uzayı, kullanılan anahtar kelimenin uzunluğuna (m) doğrudan bağlıdır. Anahtardaki her bir harf için 26 farklı seçenek olduğundan:
> $$| \mathcal{K} | = 26^m$$
> Örneğin yalnızca beş harfli bir anahtar kelime ($m = 5$) kullanılıyorsa:
> $$| \mathcal{K} | = 26^5 = 11\,881\,376$$
> Bu devasa anahtar uzayı, kendi döneminin kaba kuvvet saldırılarını imkânsız kılmıştır. Sistem yaklaşık üç yüzyıl kırılamamış; ancak 1863'te Friedrich Kasiski'nin şifreli metinde tekrar eden hece aralıklarını ölçerek **anahtar uzunluğunu (m) tespit etmesiyle** zafiyeti ortaya çıkmıştır. Anahtar uzunluğu bilindiğinde metin m adet bağımsız Sezar şifresine ayrışır ve her biri klasik frekans analiziyle tek tek kırılır.

12.1 Çözümlü Uygulamalar

Aşağıdaki örneklerde harf–sayı dönüşüm tablosunu kullanınız. Hataya düşmemek için açık metnin altına anahtar kelimeyi **döngüsel olarak** harf harf yazmak en pratik yoldur.

Örnek 12.1. Açık metni `MATH` olan bir mesajı, $K = \texttt{KEY}$ anahtarı kullanarak Vigenère şifrelemesi ile şifreleyiniz.

Çözüm.
Açık metnimiz dört harfli, anahtar kelimemiz ise üç harflidir ($m = 3$). Öncelikle anahtarı, açık metnin uzunluğuna ulaşana kadar döngüsel olarak tekrarlarız:

Konum i	0	1	2	3
Açık metin	M	A	T	H
Anahtar	K	E	Y	K

Şimdi şifreleme formülüne göre her harfi alt alta toplayalım:

1. **M** (12) + **K** $(10) \Longrightarrow 12 + 10 = 22 \Longrightarrow$ **W**
2. **A** (0) + **E** $(4) \Longrightarrow 0 + 4 = 4 \Longrightarrow$ **E**
3. **T** (19) + **Y** $(24) \Longrightarrow 19 + 24 = 43 \equiv 17 \pmod{26} \Longrightarrow$ **R**
4. **H** (7) + **K** $(10) \Longrightarrow 7 + 10 = 17 \Longrightarrow$ **R**

Sonuç: `MATH` kelimesi Vigenère ile `WERR` olarak şifrelenir.

> **Dikkat edin:** Açık metinde hiç tekrar eden harf olmamasına rağmen şifreli metinde yan yana iki `R` oluştu. Frekans analizini çökerten polialfabetik özellik tam olarak budur — aynı şifreli harf farklı açık harflerden gelebilir.

⊠

Örnek 12.2. $K = \texttt{KEY}$ anahtarı ile şifrelenmiş olan `WERR` kapalı metnini deşifre ediniz.

Çözüm.
Deşifreleme kuralımız: $d_K(y_i) \equiv y_i - k_{i \bmod m} \pmod{26}$.
Yine anahtar kelimeyi mesaj uzunluğuna kadar tekrarlıyoruz:

Konum i	0	1	2	3
Şifreli metin	W	E	R	R
Anahtar	K	E	Y	K

Alt alta çıkarma işlemlerini yapalım:

1. **W** (22) − **K** $(10) \Longrightarrow 22 - 10 = 12 \Longrightarrow$ **M**
2. **E** (4) − **E** $(4) \Longrightarrow 4 - 4 = 0 \Longrightarrow$ **A**
3. **R** (17) − **Y** $(24) \Longrightarrow 17 - 24 = -7 \equiv -7 + 26 = 19 \pmod{26} \Longrightarrow$ **T**
4. **R** (17) − **K** $(10) \Longrightarrow 17 - 10 = 7 \Longrightarrow$ **H**

Sonuç: Çıkarma işlemi ve modüler aritmetik bizi orijinal `MATH` mesajına kusursuzca geri götürdü.
⊠

13. İkili Sistem, Boolean Cebri ve XOR Mantığı

Klasik kriptografide şifreleme işlemleri harfler ve alfabenin boyutu (mod 26) üzerinden yapılıyordu. Ancak 20. yüzyılın ortalarında bilgisayarların icadıyla kriptografi kâğıt kalemden çıkıp işlemcilere taşındı. Modern kriptografide şifrelenen şey `A` harfi değil; o harfi veya herhangi bir veriyi (resim, ses, video) temsil eden **0 ve 1'lerden oluşan bit dizileridir**.

13.1 İkili Sistem (Base-2) ve Boolean Cebri

Bilgisayarların temel yapı taşı olan transistörler yalnızca iki durumu ayırt edebilir: elektrik var (1) veya elektrik yok (0). Bu yüzden modern şifreleme sistemleri $\mathbb{Z}_{26}$ yerine $\mathbb{Z}_2 = \{0, 1\}$ cismi üzerinde çalışır.
19. yüzyılda İngiliz matematikçi George Boole tarafından geliştirilen **Boolean cebri**, yalnızca *doğru (1)* ve *yanlış (0)* değerleriyle yapılan mantıksal işlemleri tanımlar. Kriptografide en sık kullandığımız üç temel mantık kapısı şunlardır:

- **AND (VE):** Yalnızca her iki girdi de 1 ise sonuç 1'dir; çarpma işlemine benzer.
- **OR (VEYA):** Girdilerden en az biri 1 ise sonuç 1'dir.
- **NOT (DEĞİL):** Girdiyi tersine çevirir (1 ise 0, 0 ise 1 yapar).

Ancak modern kriptografinin asıl kahramanı bu üçü değil, özel bir mantık kapısı olan **XOR**'dur.

13.2 XOR İşlemi (Exclusive OR)

XOR (dışlayıcı VEYA), "yalnızca biri doğruysa doğrudur, ikisi aynıysa yanlıştır" mantığıyla çalışır ve kriptografide $\oplus$ sembolüyle gösterilir.
Matematiksel olarak XOR işlemi, aslında **mod 2'de toplama işleminden** başka bir şey değildir:

$$A \oplus B \equiv A + B \pmod 2$$

Farklı bitler 1, aynı bitler 0 sonucunu verir:

Tablo 13.1: XOR doğruluk tablosu

A (açık metin biti)	B (anahtar biti)	$A \oplus B$ (şifreli bit)
0	0	0
0	1	1
1	0	1
1	1	0

13.3 Neden Kriptografinin Kalbinde XOR Var?

"Neden AES veya OTP gibi sistemler veriyi şifrelemek için toplama, çıkarma veya AND kullanmıyor da XOR kullanıyor?" sorusunun cevabı, XOR'un şu dört cebirsel özelliğidir:

1. **Birim eleman özelliği (identity):** Bir biti 0 ile XOR'lamak onu değiştirmez.

$$A \oplus 0 = A$$

2. **Kendi kendini yok etme (nilpotent özellik):** Bir biti kendisiyle XOR'larsanız sonuç daima 0 olur — sihir buradadır.

$$A \oplus A = 0$$

3. **Değişme özelliği (commutative):** Sıranın önemi yoktur.

$$A \oplus B = B \oplus A$$

4. **Birleşme özelliği (associative):** İşlem önceliğinin önemi yoktur.

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

Mükemmel geri dönüşebilirlik (şifre çözme ispatı)

XOR'un nilpotent özelliği, kriptografide **aynı fonksiyonun hem şifreleme hem de deşifreleme yapabilmesini** sağlar.
Açık metin (P) ve anahtar (K) bitlerini XOR'layarak şifreli metni (C) elde ettiğimizi varsayalım:

$$C = P \oplus K$$

Şimdi bu şifreli metni tekrar aynı anahtarla XOR'layalım. Birleşme ve yok etme özelliklerini kullanarak orijinal metne nasıl döndüğümüzü izleyin:

$$C \oplus K = (P \oplus K) \oplus K = P \oplus (K \oplus K)$$

$K \oplus K = 0$ olduğundan denklem şu hâle gelir:

$$C \oplus K = P \oplus 0 = P$$

Yani şifrelerken de çözerken de **aynı matematiksel işlemi** yaparız. Bu, donanım (çip) tasarımında devasa bir maliyet ve hız tasarrufu sağlar.

13.4 Çözümlü Uygulamalar

Modern sistemlerde XOR işlemi tek bir bit yerine, uzun bit dizileri (baytlar veya bloklar) üzerinde **karşılıklı (bitwise)** olarak uygulanır.

Örnek 13.1. 8 bitlik (1 bayt) $P = 10110100$ açık metnini, $K = 01101011$ anahtarını kullanarak XOR ile şifreleyiniz. Ardından bulduğunuz sonucu tekrar aynı anahtarla XOR'layarak deşifre ediniz.

Çözüm.
Bitleri alt alta yazarak sütun sütun XOR'layalım: aynıysa 0, farklıysa 1.
1. aşama — şifreleme ($C = P \oplus K$):

```
P :  1 0 1 1 0 1 0 0
K :  0 1 1 0 1 0 1 1
     ---------------
C :  1 1 0 1 1 1 1 1
```

Şifreli metin: `11011111`
2. aşama — deşifreleme ($P = C \oplus K$):

```
C :  1 1 0 1 1 1 1 1
K :  0 1 1 0 1 0 1 1
     ---------------
P :  1 0 1 1 0 1 0 0
```

Orijinal metin: `10110100`
Görüldüğü gibi, aynı anahtarla ikinci kez XOR'landığında başlangıç noktasına kusursuz biçimde geri dönülmüştür.
⊠

14. Claude Shannon Prensipleri: Karışıklık ve Yayılma

Klasik şifrelerin (Sezar, afin, Vigenère) tamamı sonunda kriptanalistlere boyun eğmiştir; çünkü metnin içindeki dilden kaynaklı istatistiksel izleri —örneğin İngilizcede `E` harfinin sıklığını— tam olarak yok edememişlerdir.
1949'da bilgi kuramının kurucusu **Claude Shannon**, bir şifreleme sisteminin istatistiksel analizleri tamamen işe yaramaz hâle getirebilmesi için iki temel prensip tanımlamıştır: **karışıklık (confusion)** ve **yayılma (diffusion)**. Modern sistemlerin kalbini oluşturan bu iki kavramı anlamak, dijital kriptografiyi anlamak demektir.

14.1 Karışıklık (Confusion)

Karışıklık prensibinin tek bir amacı vardır: **şifreli metin ile anahtar arasındaki ilişkiyi olabildiğince karmaşık ve çözülemez hâle getirmek.**
Düşman şifreli metin üzerinde istatistiksel bir analiz yaptığında, kullanılan anahtar hakkında en ufak bir ipucu bile elde edememelidir. Anahtarın tek bir biti bile şifreli metnin karakterini tamamen değiştirmelidir.
Nasıl sağlanır? Karışıklık genellikle **yerine koyma (substitution)** işlemleriyle sağlanır. Klasik dönemdeki Sezar ve afin şifreleri çok basit karışıklık örnekleridir. Modern şifrelerde (örneğin AES'te) bu işlem karmaşık, doğrusal olmayan **S-kutuları (substitution boxes)** ile gerçekleştirilir.

14.2 Yayılma (Diffusion)

Yayılma prensibinin amacı ise şudur: **açık metnin istatistiksel yapısını, şifreli metnin tamamına yayarak yok etmek.**
Açık metindeki tek bir harf veya bit değiştiğinde, şifreli metindeki *birçok* harf veya bit değişmelidir. Böylece açık metindeki dilden kaynaklı kalıplar (tekrar eden heceler, sık kullanılan harfler) şifreli metnin içinde eriyip kaybolur.
Nasıl sağlanır? Yayılma genellikle **yer değiştirme (permütasyon / transpozisyon)** işlemleriyle sağlanır. Klasik dönemde Hill şifrelemesi, matris çarpımı sayesinde güçlü bir yayılma örneğidir. Modern şifrelerde bu işlem **P-kutuları (permutation boxes)** veya karmaşık bit kaydırmalarıyla yapılır.

> Kriptografinin kutsal kâsesi: çığ etkisi (avalanche effect)
>
> Yayılma prensibinin ne kadar başarılı çalıştığı **çığ etkisi** ile ölçülür.
> Açık metinde veya anahtarda **yalnızca 1 bitlik** ufak bir değişiklik yaptığınızda şifreli metnin bitlerinin yaklaşık **%50'si** rastgele biçimde değişiyorsa, o sistemde güçlü bir çığ etkisi var demektir.
> Dağın tepesinden yuvarlanan tek bir kar topunun (1 bit) aşağı indiğinde devasa bir çığa (metnin yarısının değişmesine) dönüşmesi gibi düşünülebilir. Modern AES ve SHA-256 algoritmaları güçlü bir çığ etkisine sahiptir.

14.3 Çarpım Şifreleri ve SPN Yapısı

Shannon, yalnızca karışıklık ya da yalnızca yayılma kullanmanın tek başına yeterli güvenliği sağlamadığını fark etti. Çözüm olarak bu iki işlemi art arda, defalarca tekrar eden bir yapı önerdi. Bu birleşik yapılara **çarpım şifreleri (product ciphers)** denir.
Modern şifreleme standartlarının temelini oluşturan **SPN (Substitution-Permutation Network)** mimarisi bu fikirden doğmuştur. Veri bloğu sisteme girdiğinde:

1. **S-kutusu (karışıklık):** Veri bitleri kendi içinde başka bitlerle değiştirilir.
2. **P-kutusu (yayılma):** Değişen bu bitlerin sıralamaları karıştırılarak tüm bloğa dağıtılır.

3. Bu döngü (raund) aynı veri üzerinde 10, 12 veya 16 kez tekrar edilerek mesaj çözülemez bir yapıya dönüştürülür.

14.4 Klasik Şifrelerin Shannon Analizi

Bugüne kadar öğrendiğimiz klasik şifrelerin Shannon prensipleri açısından durumunu bir tabloyla özetleyelim:

Tablo 14.1: Klasik şifrelerin Shannon prensipleri açısından karşılaştırması

Algoritma	Karışıklık	Yayılma	Genel güvenlik analizi
Sezar şifresi	Var (çok zayıf)	Yok	İstatistiksel frekansları korur, saniyeler içinde kırılır.
Afin şifreleme	Var (orta)	Yok	Tek harf tek harfe dönüştüğü için frekans analiziyle kırılır.
Vigenère şifresi	Var (çoklu)	Yok	Çığ etkisi yoktur; Kasiski metoduyla anahtar uzunluğu bulunup kırılır.
Hill şifrelemesi	Yok	**Var (güçlü)**	Matris çarpımıyla harfler birbirine yayıldığı için dönemine göre devrimseldir.
Modern AES	**Mükemmel**	**Mükemmel**	S ve P kutularını defalarca tekrarlayarak yüksek güvenlik sağlar.

15. Vernam Şifresi ve One-Time Pad (Kırılamazlık İspatı)

Kriptografi tarihindeki tüm algoritmalar (Sezar, Vigenère, Enigma, hatta günümüzdeki RSA ve AES) yeterli zaman ve işlem gücü verildiğinde teorik olarak kırılabilir. Kırılamayacağı matematiksel olarak ispatlanmış **tek** şifreleme sistemi ise **One-Time Pad (tek kullanımlık şerit)** algoritmasıdır.
Temeli, 1917'de Gilbert Vernam'ın icat ettiği ikili toplama işlemine (**XOR**) dayanır.

Tanım 15.1 (One-Time Pad (Vernam Şifresi) Kriptosistemi). Modern One-Time Pad, alfabe harfleri yerine doğrudan bit dizileri üzerinde işlem yapar. Mesaj uzunluğunun n bit olduğunu varsayarsak, beş bileşenli kriptosistem anatomisi şöyle tanımlanır:

- **Açık metin ($\mathcal{P}$):** Uzunluğu n olan tüm olası bit dizilerinin kümesi, $\{0,1\}^n$.
- **Şifreli metin ($\mathcal{C}$):** Uzunluğu n olan tüm olası bit dizilerinin kümesi, $\{0,1\}^n$.
- **Anahtar uzayı ($\mathcal{K}$):** Uzunluğu n olan tüm olası bit dizilerinin kümesi, $\{0,1\}^n$.
- **Şifreleme fonksiyonu ($\mathcal{E}$):** Açık metin vektörü ile anahtar vektörünün karşılıklı XOR'lanmasıdır:

$$e_K(x) \equiv x \oplus K \pmod 2$$

- **Deşifreleme fonksiyonu ($\mathcal{D}$):** Şifreli metnin aynı anahtarla tekrar XOR'lanmasıdır — XOR'un nilpotent, yani kendisinin tersi olma özelliği gereği:

$$d_K(y) \equiv y \oplus K \pmod 2$$

Teorem 15.1 (Shannon'un Kusursuz Gizlilik (Perfect Secrecy) İlkesi). *Claude Shannon, 1949 yılında bir sistemin kusursuz gizliliğe sahip olabilmesi için şu şartı sağlaması gerektiğini matematiksel olarak ispatlamıştır:*

$$P(x \mid y) = P(x)$$

*Yani şifreli metni (y) ele geçiren bir düşmanın, orijinal mesajın (x) ne olduğuna dair yapacağı olasılık hesabı, şifreli metni hiç görmeden yapacağı tahminle **birebir aynı olmalıdır**. Şifreli metin, orijinal metin hakkında **sıfır** bilgi verir.*

> ***İspat (One-Time Pad için).***
> *One-Time Pad bu şartı sağlar; çünkü her x açık metni ve her y şifreli metni için $x \oplus K = y$ eşitliğini sağlayan **benzersiz ve tek bir K anahtarı** mutlaka vardır (nitekim $K = x \oplus y$).*
> *Anahtar tamamen rastgele ve düzgün dağılımlı seçildiğinden, şifreli metnin altından "SALDIR" kelimesinin çıkma ihtimali ile "BEKLE" kelimesinin çıkma ihtimali matematiksel olarak tamamen eşittir. Düşman sonsuz işlem gücüne sahip olsa bile doğru mesajı ayırt edemez; çünkü ortada analiz edilecek istatistiksel bir iz yoktur.*
> ■

15.1 One-Time Pad'in Üç Altın Kuralı

Bu sistemin kusursuz olabilmesi için aşağıdaki üç kuralın **istisnasız** uygulanması gerekir; biri bile ihlal edilirse sistem çöker.

1. **Tam rastgelelik (true randomness):** Anahtar, hiçbir algoritmik kurala bağlı olmayan, fiziksel olaylardan (radyoaktif bozunma, atmosferik gürültü vb.) elde edilmiş gerçek rastgele bitlerden oluşmalıdır. Bilgisayarların ürettiği sözde rastgele (pseudo-random) sayılar kesinlikle kullanılamaz.
2. **Uzunluk şartı:** Anahtarın uzunluğu, şifrelenecek açık metnin uzunluğuna **eşit veya ondan büyük** olmalıdır: $\mid K \mid \geq \mid x \mid$.

3. **Tek kullanımlık (never reuse):** Bir anahtar şeridi yalnızca bir mesaj için kullanılmalı, ardından fiziksel veya dijital olarak **imha edilmelidir**.

Örnek 15.1 (İki Kez Kullanılan Şerit (Two-Time Pad) Zafiyeti). Üçüncü kural ihlal edilip aynı K anahtarı ile iki farklı mesaj (x_1 ve x_2) şifrelenirse, düşman bu durumu matematiksel olarak nasıl istismar eder?

Çözüm.
Aynı K anahtarı kullanıldığında üretilen şifreli metinler şunlardır:

$$y_1 = x_1 \oplus K$$
$$y_2 = x_2 \oplus K$$

Düşman hattı dinleyip bu iki şifreli metni ele geçirir ve bunları birbiriyle XOR'larsa, XOR'un değişme ve birleşme özellikleri ile $K \oplus K = 0$ kuralı gereği anahtarlar birbirini yok eder:

$$\begin{aligned} y_1 \oplus y_2 &= (x_1 \oplus K) \oplus (x_2 \oplus K) \\ &= x_1 \oplus x_2 \oplus (K \oplus K) \\ &= x_1 \oplus x_2 \oplus 0 \\ &= x_1 \oplus x_2 \end{aligned}$$

Sonuç: Düşmanın elinde artık anahtardan tamamen arındırılmış, doğrudan iki orijinal mesajın birbirine XOR'lanmış hâli vardır. Doğal dil analizleriyle bu iki metin birbirinden ayrıştırılıp okunabilir. Tarihteki ünlü **Venona Projesi**'nde Sovyet şifrelerinin kırılma sebebi tam olarak anahtar şeritlerinin yeniden kullanılmasıdır.
■

15.2 Madem Kusursuz, Neden Her Yerde Kullanmıyoruz?

Akla gelen ilk soru şudur: *"Eğer One-Time Pad kırılamıyorsa, neden bankalar, mesajlaşma uygulamaları veya ordular sürekli AES ve RSA gibi teorik olarak kırılabilir algoritmalar kullanıyor?"*
Cevap, kriptografinin en büyük açmazı olan **anahtar dağıtım probleminde (key distribution problem)** gizlidir.
One-Time Pad'in ikinci kuralı, anahtarın en az mesaj kadar uzun olmasını gerektirir. Arkadaşınıza 10 GB'lık bir video dosyasını OTP ile şifreleyerek göndermek isterseniz, ona öncesinde **tamamen rastgele üretilmiş 10 GB'lık bir anahtar dosyasını** mutlak güvenli bir yolla (örneğin bir flash belleği kuryeyle) ulaştırmanız gerekir.

> ! Paradoks
>
> Eğer 10 GB'lık bir veriyi düşmanın eline geçmeden karşı tarafa ulaştırabilecek kadar güvenli bir kanalınız zaten varsa, neden o kanaldan anahtarı göndermek yerine *doğrudan şifrelemek istediğiniz videoyu* göndermiyorsunuz?

İşte bu pratik imkânsızlık yüzünden One-Time Pad gündelik hayatta kullanılamaz; yalnızca Soğuk Savaş dönemindeki Moskova–Washington "kırmızı telefon" hattı gibi çok yüksek güvenlikli ve düşük veri hacimli askerî/diplomatik haberleşmelerde kullanılabilmiştir.
Bu problem, kriptografinin bir sonraki büyük devrimini —**asimetrik (açık anahtarlı) kriptografiyi**— doğuran temel motivasyondur.

16. Akan Şifreler ve Anahtar Akışı Üreteçleri (LCG, LFSR)

One-Time Pad'in kusursuz gizlilik sunduğunu, ancak mesajla aynı boyutta rastgele bir anahtara ihtiyaç duyduğu için anahtar dağıtım problemi nedeniyle pratik olmadığını gördük.
Akan şifreler (stream ciphers), OTP'nin XOR tabanlı şifreleme mantığını taklit eden; ancak çok daha kısa ve yönetilebilir bir **kök anahtar (seed key)** kullanarak bunu uygulanabilir hâle getiren modern sistemlerdir.
Temel felsefesi şudur: gönderici ve alıcı kısa bir kök anahtarı paylaşır. Her iki taraftaki şifreleme birimi, bu kısa anahtarı kullanarak matematiksel bir algoritmayla **sözde rastgele (pseudo-random)** ve çok uzun bir **anahtar akışı (keystream)** üretir. Ardından bu akış, tıpkı One-Time Pad'de olduğu gibi açık metinle XOR'lanır.

16.1 Sistemin Formal Tanımı

Akan şifrelemede işlemler bit dizileri üzerinde eşzamanlı olarak gerçekleşir.

Tanım 16.1 (Akan Şifreleme Sistemi).

- **Açık metin ($\mathcal{P}$):** İkili formdaki açık metin bitlerinin dizisi, $x = x_1, x_2, \ldots$
- **Şifreli metin ($\mathcal{C}$):** İkili formdaki şifreli metin bitlerinin dizisi, $y = y_1, y_2, \ldots$
- **Anahtar uzayı ($\mathcal{K}$):** Belirli ve sabit uzunluktaki kök anahtarların kümesi.
- **Anahtar akışı üreteci:** $K \in \mathcal{K}$ kök anahtarını alarak $z_1, z_2, \ldots$ şeklinde açık metin uzunluğunda sözde rastgele bir dizi üreten **deterministik** fonksiyondur.
- **Şifreleme fonksiyonu ($\mathcal{E}$):** Açık metnin her τ. biti, anahtar akışının ilgili τ. bitiyle XOR'lanır:

$$e_{z_\tau}(x_\tau) \equiv x_\tau \oplus z_\tau \pmod 2$$

- **Deşifreleme fonksiyonu ($\mathcal{D}$):** Şifreli metnin her τ. biti, aynı şekilde üretilmiş anahtar akışıyla tekrar XOR'lanır:

$$d_{z_\tau}(y_\tau) \equiv y_\tau \oplus z_\tau \pmod 2$$

16.2 Anahtar Akışı Nasıl Üretilir?

Akan şifrelerin kalbi anahtar akışı üretecidir. Üretilen dizi dışarıdan bakan bir düşman için tamamen rastgele görünmelidir; oysa aslında matematiksel bir formüle dayanan deterministik bir yapıdır. Bu diziler tarihsel olarak sayılar teorisi ve lineer cebir tabanlı fonksiyonlarla üretilmiştir.

16.2.1 1. Doğrusal Eşlenik Üreteci (Linear Congruential Generator — LCG)

Kriptografik olarak güçlü olmasa da, sözde rastgele sayı üretiminin temel mantığını anlamak için en iyi örnektir; doğrudan modüler aritmetiğe dayanır.
Bir S_0 başlangıç değeri (kök anahtar) belirlenir ve her τ adımında yeni durum şu doğrusal bağıntıyla hesaplanır:

$$S_\tau \equiv (a \cdot S_{\tau-1} + c) \pmod m$$

Burada a çarpan, c artış miktarı, m ise modüldür. Elde edilen S_τ sayısının kendisi doğrudan kullanılabileceği gibi, ikili sisteme indirgemek için mod 2'si alınarak z_τ anahtar biti elde edilebilir:

$$z_\tau \equiv S_\tau \pmod 2$$

Bu yöntem çok basittir; ancak periyodu m değerine ve seçilen katsayıların aralarında asallık durumlarına sıkı sıkıya bağlıdır.

16.2.2 2. Doğrusal Geri Beslemeli Kaydırmalı Yazmaç (LFSR)

LFSR, LCG'nin $\mathbb{Z}_2$ cismine uyarlanmış, donanım üzerinde çok daha hızlı çalışan lineer cebirsel versiyonudur. Sisteme L uzunluğunda bir bit dizisi (başlangıç durumu) verilir.
Her τ zaman adımında mevcut bitler bir sağa kaydırılır. En sağdan düşen bit, **anahtar akışı bitimiz** z_τ olur. Boşalan en sol haneye ise, içerideki belirli bitlerin XOR'lanmasıyla elde edilen yeni bir bit yazılır.
$\tau > L$ adımları için üretilen bitin genel doğrusal tekrar bağıntısı şöyledir:

$$z_\tau \equiv c_1 z_{\tau-1} + c_2 z_{\tau-2} + \ldots + c_L z_{\tau-L} \pmod 2$$

Buradaki $c_i \in \{0, 1\}$ katsayıları, hangi indekslerdeki bitlerin geri beslemeye dâhil edileceğini belirler. Uygun bir **primitif polinom** seçildiğinde L boyutlu bir LFSR, kendini tekrar etmeden önce tam $2^L - 1$ adet rastgele görünümlü bit üretebilir.
Ancak sistem tamamen lineer olduğu için **Berlekamp-Massey algoritmasıyla** kolayca çözülür: yalnızca $2L$ bitlik bir anahtar akışı parçası, tüm üreteci geri çıkarmaya yeter. Bu yüzden modern sistemlerde birden fazla LFSR, doğrusal olmayan fonksiyonlarla harmanlanarak kullanılır.

16.3 Anahtar Periyodu ve Döngü Tehlikesi

Akan şifreler, ürettikleri anahtar akışının kalitesi ölçüsünde güvenlidir. Ancak hiçbir sözde rastgele üreteç sonsuza kadar birbirinden farklı değerler üretemez; dizi belirli bir adımdan sonra mutlaka başa döner ve **kendini tekrar etmeye** başlar.
Anahtar dizisinin kendini tekrar edene kadar ürettiği benzersiz değer adedine **periyot (period)** denir.

⚠ Periyot kısalığı zafiyeti

Şifrelenecek mesajın uzunluğu üretecin periyodundan uzunsa, aynı anahtar akışı mesajın ilerleyen kısımlarında **ikinci kez** kullanılmış olur. Bu da One-Time Pad'de gördüğümüz ölümcül **two-time pad** zafiyetini tetikler ve şifre kolayca kırılır.

- **LCG üreteçleri** için ulaşılabilecek maksimum periyot, modül değeri kadardır (m).
- **LFSR üreteçleri** için ulaşılabilecek maksimum periyot $2^L - 1$'dir (L: yazmaçtaki bit sayısı).

Modern akan şifreler (örneğin ChaCha20), mesaj ne kadar uzun olursa olsun pratikte asla döngüye girmeyecek kadar büyük periyotlara sahip olacak şekilde tasarlanır.

16.4 Akan Şifrelerin Öne Çıkan Özellikleri

1. **Hız ve donanım verimliliği.** Akan şifreler veriyi bit bit veya bayt bayt işledikleri için verinin tamamının belleğe yüklenmesini beklemezler; donanım üzerinde çok hızlı çalışırlar. Bu özellik onları canlı video yayınları, Bluetooth ve kablosuz iletişim için vazgeçilmez kılar.
2. **Hata yayılımı yoktur (no error propagation).** İletişim hattındaki gürültü nedeniyle şifreli metindeki tek bir bit bozulursa, alıcı tarafta deşifre edilen açık metinde yalnızca ona karşılık gelen tek bit hatalı çıkar; hata sonraki bitlere yayılmaz. Blok şifrelerde ise tek bir bit hatası çoğu kipte bloğun tamamını bozar.
3. **Senkronizasyon zorunluluğu.** Buna karşılık, taraflar anahtar akışında aynı konumda olmak zorundadır. İletimde bir bit tamamen kaybolursa akış kayar ve sonraki tüm veri anlamsızlaşır.

16.5 Çözümlü Uygulama

Aşağıdaki örnekte açık metnimiz doğrudan 8 bitlik (1 baytlık) ikili bloklar hâlinde verilmiştir. Anahtar akışını üretmek için — bayt sınırını korumak adına — $m = 256$ olan bir LCG üreteci kullanılmıştır.

Örnek 16.1. $x = [10101010, 11110000]$ ikili açık metnini, LCG ($S_0 = 10,\ a = 5,\ c = 7,\ m = 256$) üreteci ve XOR işlemiyle şifreleyiniz.

Çözüm.

Şifreleme fonksiyonumuz $e(x_\tau) \equiv x_\tau \oplus S_\tau$, anahtar akışı üretimimiz ise $S_\tau \equiv (aS_{\tau-1} + c) \pmod{256}$ şeklindedir.
Açık metin bloklarımız: $x_1 = 10101010_2$ ve $x_2 = 11110000_2$.
1. adım — ilk bloğun şifrelenmesi. Önce birinci anahtar akışı değerini onluk tabanda hesaplayalım:

$$S_1 \equiv (5 \cdot 10 + 7) \pmod{256} = 57_{10}$$

57 sayısını ikili sisteme çevirelim: $S_1 = 00111001_2$. Şimdi bitwise XOR uygulayalım:

```
x₁ :  1 0 1 0 1 0 1 0
S₁ :  0 0 1 1 1 0 0 1
      ─────────────────
y₁ :  1 0 0 1 0 0 1 1
```

İlk şifreli blok: $y_1 = 10010011_2$.
2. adım — ikinci bloğun şifrelenmesi. Sistemin kalbi burasıdır: yeni anahtar, bir önceki duruma ($S_1 = 57$) bağlı olarak güncellenir.

$$S_2 \equiv (5 \cdot 57 + 7) \pmod{256} \equiv 292 \pmod{256} = 36_{10}$$

36 sayısını ikili sisteme çevirelim: $S_2 = 00100100_2$.

```
x₂ :  1 1 1 1 0 0 0 0
S₂ :  0 0 1 0 0 1 0 0
      ─────────────────
y₂ :  1 1 0 1 0 1 0 0
```

İkinci şifreli blok: $y_2 = 11010100_2$.
Sonuç: Açık metin blokları bu LCG tabanlı akan şifreleme sistemiyle $[10010011, 11010100]$ şifreli dizisine dönüştürülmüştür.
Dikkat edilmesi gereken nokta: her blok **farklı** bir anahtar değeriyle şifrelenmiştir. Aynı S değeri iki blokta tekrarlansaydı, two-time pad zafiyeti doğardı.
$\boxtimes$

17. Sonlu Cisimler (Galois Cisimleri)

DES ve AES'e dalmadan önce ufak bir matematik molası vermemiz gerekiyor. Modern blok şifrelerin — özellikle AES'in — her adımı, baytlar üzerinde yapılan cebirsel işlemlerle tanımlanır. Bu kısa köprü bölümünde o işlemlerin yaşadığı yapıyı tanıyacağız: **sonlu cisimler (finite fields)**.

17.1 Her Adım Geri Alınabilmeli

Bir şifreleme algoritmasındaki her adım **geri alınabilir (invertible)** olmak zorundadır; aksi hâlde mesajı meşru alıcı bile açamaz. Toplamayı çıkarmayla geri alırız. Çarpmayı geri almak içinse "bölme" gerekir — modüler dünyada bunun adı, sıfırdan farklı her elemanın bir **çarpımsal tersinin** var olmasıdır.
Bu şartın sancısını klasik bölümlerde bizzat çektik:

- Afin şifrelemesinde çarpan olarak $\mathbb{Z}_{26}$'nın yalnızca 26 ile aralarında asal 12 elemanını kullanabildik; geri kalanların tersi yoktu.
- Hill şifrelemesinde anahtar matrisin determinantı $\gcd(\det K, 26) = 1$ koşulunu sağlamazsa mesaj bir daha asla açılamıyordu.

Sorunun kökü 26'nın asal olmamasıdır: $2 \cdot 13 \equiv 0 \pmod{26}$ olduğundan, sıfırdan farklı iki elemanın çarpımı sıfır çıkabilir ve böyle "sıfır bölenlerin" tersi olamaz. Oysa matematiksel temeller bölümünde görmüştük: modül bir p asalı seçilirse $\mathbb{Z}_p$'de sıfır hariç *her* elemanın tersi vardır. Cebirde bu "her şeyin tersi var" cennetinin özel bir adı vardır.

Tanım 17.1 (Cisim (Field)). Üzerinde toplama ve çarpma adında iki işlem tanımlanmış bir F kümesi şu koşulları sağlıyorsa **cisim (field)** adını alır:

- Her iki işlem de **birleşmeli** ve **değişmelidir**; çarpma toplama üzerine **dağılır**: $a(b+c) = ab + ac$.
- Toplamanın birim elemanı 0, çarpmanın birim elemanı 1 kümenin içindedir.
- Her a elemanının bir toplamsal tersi $-a$ vardır; yani çıkarma daima yapılabilir.
- **Taç özellik:** 0 dışındaki *her* a elemanının bir çarpımsal tersi a^{-1} vardır; yani sıfırla bölme hariç bölme daima yapılabilir.

Kısacası cisim; toplama, çıkarma, çarpma ve bölmenin hiç takılmadan çalıştığı bir sayı sistemidir. $\mathbb{Q}$ ve $\mathbb{R}$ bildiğimiz sonsuz örneklerdir; ama bilgisayarlar sonsuz kümelerle çalışamaz. Kriptografinin aradığı şey, **sonlu** cisimlerdir.

17.2 Galois Cisimleri: $\mathrm{GF}(p)$ ve $\mathrm{GF}(p^n)$

En kolay sonlu cisim örneklerini zaten tanıyoruz: p asal olmak üzere her $\mathbb{Z}_p$ bir cisimdir ve sonlu cisim gösteriminde $\mathrm{GF}(p)$ olarak yazılır. Peki başka boyutlarda sonlu cisim var mıdır?

> **i** Galois'nın mirası: hangi boyutlarda sonlu cisim var?
>
> Cebirin bize ispatsız aktaracağımız temel armağanı şudur: q elemanlı bir sonlu cisim **ancak ve ancak** $q = p^n$ (p asal, $n \geq 1$) biçimindeyse vardır ve aynı eleman sayısına sahip tüm sonlu cisimler özünde (izomorfizme kadar) birbirinin aynısıdır.
> Bu biricik yapı, grup teorisinin temellerini atıp henüz 20 yaşındayken bir düelloda hayatını kaybeden Fransız matematikçi Évariste Galois'nın (1811–1832) onuruna **Galois cismi (Galois field)** $\mathrm{GF}(p^n)$ olarak adlandırılır.
> Örneğin $4 = 2^2$, $9 = 3^2$ ve $256 = 2^8$ elemanlı birer sonlu cisim vardır; ama $6 = 2 \cdot 3$ elemanlı bir cisim yoktur — 26 elemanlı da yoktur!

En küçük Galois cismi ise modern kriptografinin ta kendisidir: $\mathrm{GF}(2) = \{0, 1\}$. Bu cismin toplama tablosu tam olarak **XOR**, çarpım tablosu tam olarak **AND** kapısıdır. İkili sistem bölümünde yazdığımız $A \oplus B \equiv A + B \pmod 2$ eşitliği, aslında "XOR, $\mathrm{GF}(2)$ cisminin toplamasıdır" cümlesinin ta kendisiydi.

17.3 Hedef: Bir Bayt = Bir Sayı

Bilgisayarlar baytlarla konuşur: bellekteki her hücre 8 bitlik, yani $2^8 = 256$ farklı değer alabilen bir pakettir. Şifreleme adımlarını doğrudan baytlar üzerinde tanımlayabilmek için 256 elemanlı bir cisim istiyoruz — böylece "bir bayt = bir sayı" olur ve dört işlemin tamamı her bayt için çalışır. $256 = 2^8$ bir asal kuvveti olduğundan Galois'nın teoremi böyle bir cismin var olduğunu garanti eder: $\mathrm{GF}(2^8)$.

⚠ Tuzak: mod 256 aritmetiği cisim değildir

İlk akla gelen aday olan $\mathbb{Z}_{256}$, yani "mod 256 aritmetiği" bir cisim **değildir**. 256 asal olmadığı için yine sıfır bölenler ortaya çıkar:

$$2 \cdot 128 \equiv 0 \pmod{256}$$

Bu yüzden hiçbir çift sayının — elemanların tam yarısının! — çarpımsal tersi yoktur. $\mathbb{Z}_{26}$'da başımıza gelenin aynısı. $\mathrm{GF}(2^8)$'i inşa etmenin doğru yolu tam sayılarla değil, **polinomlarla** hesap yapmaktır.

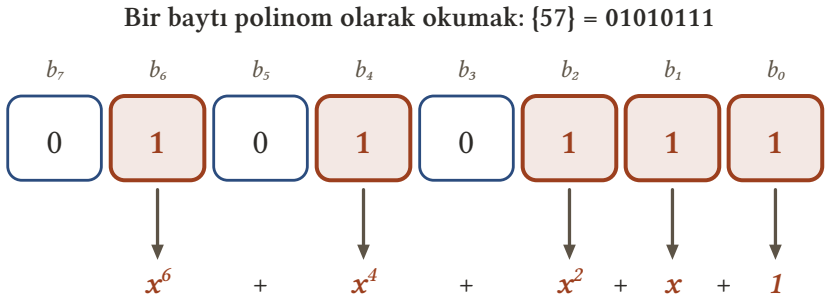
17.4 Baytlar Polinom Olarak

Fikir şaşırtıcı derecede basittir: $b_7 b_6 \ldots b_1 b_0$ bitlerinden oluşan bir baytı, katsayıları $\mathrm{GF}(2)$'de olan ve derecesi en çok 7 olan bir polinom olarak *okuruz*:

$$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \leftrightarrow b_7 x^7 + b_6 x^6 + \ldots + b_1 x + b_0$$

AES literatürü baytları onaltılık (hex) sistemde yazıp süslü paranteze alır; biz de bu geleneğe uyacağız. Örneğin $\{57\}$ baytı ikilik sistemde 01010111'dir ve polinom karşılığı şudur:

$$\{57\} = 01010111 \leftrightarrow x^6 + x^4 + x^2 + x + 1$$



Şekil 17.1: Bir baytın sekiz biti, katsayıları GF(2)'de olan bir polinomun katsayı listesidir: {57} = 01010111 baytı, yalnızca 1 olan bitlerin kuvvetleri alınarak $x^6 + x^4 + x^2 + x + 1$ polinomuna dönüşür.

Polinomun x'ine bir sayı koymayacağız; o yalnızca katsayıları raflara dizen bir **yer tutucudur**. Bütün bilgi, $\mathrm{GF}(2)$'de yaşayan katsayılardadır.

17.5 Toplama: Eski Dostumuz XOR

İki polinomu toplarken aynı dereceli katsayılar $\mathrm{GF}(2)$'de, yani mod 2'de toplanır. Bunun bayt karşılığı tek kelimedir: **XOR**. Örneğin $\{57\} \oplus \{83\}$:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2$$

x ve 1 terimleri iki kez göründükleri için yok oldular ($1 + 1 = 0$). Bayt dilinde aynı işlem:

$$01010111 \oplus 10000011 = 11010100 \Longrightarrow \{57\} \oplus \{83\} = \{D4\}$$

Dikkat ederseniz her eleman kendi toplamsal tersidir: XOR bölümünden bildiğimiz $A \oplus A = 0$ özelliği, $\mathrm{GF}(2^8)$'de "çıkarma ile toplama aynı işlemdir" anlamına gelir. Donanım için bundan güzeli olamaz.

17.6 Çarpma: $m(x)$ Polinomuna Göre Kalan

Asıl marifet çarpmadadır. İki polinomu çarptığımızda derece 14'e kadar çıkabilir — sonuç artık bir bayta sığmaz. $\mathbb{Z}_p$'de büyüyen sayıları mod p ile küçültüyorduk; burada da çarpımı sabit bir polinoma bölüp **kalanı** alırız. Asal sayı p'nin rolünü ise özel bir polinom türü üstlenir.

Tanım 17.2 (İndirgenemez Polinom (Irreducible Polynomial)). Katsayıları $\mathrm{GF}(2)$'de olan ve daha küçük dereceli iki polinomun çarpımı biçiminde yazılamayan polinomlara **indirgenemez polinom (irreducible polynomial)** denir. İndirgenemez polinomlar, polinom dünyasında asal sayıların oynadığı rolü oynar: $\mathbb{Z}_p$'nin cisim olması nasıl p'nin asallığına bağlıysa, "mod $m(x)$" aritmetiğinin cisim olması da $m(x)$'in indirgenemezliğine bağlıdır.

AES standardı bu iş için şu sabit, 8. dereceden indirgenemez polinomu seçmiştir:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

Bit karşılığı 1 0001 1011, yani hex gösterimiyle $\{11B\}$'dir. Çarpma işleminin tamamı şu akışla yürür: polinomları çarp, çıkan uzun polinomu $m(x)$'e böl, kalanı bayta geri çevir.

Şekil 17.2: GF(2⁸)'de çarpmanın akışı: iki bayt polinom olarak çarpılır, derecesi 14'e kadar çıkabilen ara sonuç indirgenemez m(x) polinomuna bölünüp kalan alınır ve sonuç yeniden tek bir bayta sığar: {57} · {83} = {C1}.

Örnek 17.1. FIPS-197 standardının klasik örneğini hesaplayalım: $\mathrm{GF}(2^8)$'de $\{57\} \cdot \{83\}$ çarpımını bulunuz.

> **Çözüm.**
> **1. adım — polinom çarpımı.** $\{57\} = x^6 + x^4 + x^2 + x + 1$ ve $\{83\} = x^7 + x + 1$ polinomlarını dağılma özelliğiyle çarpalım. Soldaki polinomun her terimini $x^7 + x + 1$ ile çarpıyoruz:

$$x^6 \cdot (x^7 + x + 1) = x^{13} + x^7 + x^6$$
$$x^4 \cdot (x^7 + x + 1) = x^{11} + x^5 + x^4$$
$$x^2 \cdot (x^7 + x + 1) = x^9 + x^3 + x^2$$
$$x \cdot (x^7 + x + 1) = x^8 + x^2 + x$$
$$1 \cdot (x^7 + x + 1) = x^7 + x + 1$$

Katsayılar $\mathrm{GF}(2)$'de toplandığı için **çift sayıda görünen terimler yok olur**: x^7, x^2 ve x ikişer kez göründüklerinden silinir. Geriye kalan:

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

2. adım — $m(x)$'e göre kalan. Derece $13 > 7$ olduğu için indirgeme şart. Polinom bölmesinde çıkarma yerine XOR kullanılır: en yüksek dereceli terimi silmek için $m(x)$'i uygun kuvvetle çarpıp ekleriz.
x^{13}'ü silmek için $x^5 \cdot m(x) = x^{13} + x^9 + x^8 + x^6 + x^5$ ekleyelim:

$$x^{11} + x^4 + x^3 + 1$$

x^{11}'i silmek için $x^3 \cdot m(x) = x^{11} + x^7 + x^6 + x^4 + x^3$ ekleyelim:

$$x^7 + x^6 + 1$$

Derece artık 7'ye indi; bu bizim kalanımızdır.
3. adım — bayta dönüş. $x^7 + x^6 + 1 = 11000001 = \{C1\}$. Sonuç:

$$\{57\} \cdot \{83\} = \{C1\}$$

$\boxtimes$

17.7 xtime: $\{02\}$ ile Çarpmanın Kısayolu

Görünüşte hantal olan bu çarpma, donanımda neden ışık hızındadır? Sır, x ile — yani $\{02\}$ baytıyla — çarpmanın sadeliğindedir. x ile çarpmak her terimin derecesini bir artırır; bayt dilinde bu, **baytı 1 bit sola kaydırmaktır**. İki durum vardır:

- En soldaki bit $b_7 = 0$ ise kaydırma yeterlidir; sonuç zaten bayta sığar.
- $b_7 = 1$ ise kaydırma sonucunda bir x^8 terimi doğar. $m(x)$'ten dolayı $x^8 \equiv x^4 + x^3 + x + 1 \pmod{m(x)}$ olduğundan, taşan bit atılır ve sonuç $\{1B\}$ ile XOR'lanır.

AES literatürü bu işleme **xtime** adını verir. İki hızlı örnek:

$$\{57\} \cdot \{02\}: \quad 01010111 \overset{\text{kaydır}}{\rightarrow} 10101110 = \{AE\} \quad \text{(taşma yok)}$$

$$\{AE\} \cdot \{02\}: \quad 10101110 \overset{\text{kaydır}}{\rightarrow} \mathbf{1}\,01011100 \overset{\mathbf{1}\text{ at, }\oplus\{1B\}}{\rightarrow} 01000111 = \{47\}$$

Herhangi bir baytla çarpma, xtime zincirinin uygun halkalarının XOR'lanmasıyla elde edilir. Bunu bir örnekle görelim.

Örnek 17.2. $\mathrm{GF}(2^8)$'de $\{57\} \cdot \{13\}$ çarpımını xtime zinciriyle hesaplayınız.

Çözüm.
Önce çarpanı ikinin kuvvetlerine ayıralım: $\{13\} = 00010011$, yani $\{13\} = \{10\} \oplus \{02\} \oplus \{01\}$. Cisim aksiyomlarındaki dağılma özelliği sayesinde:

$$\{57\} \cdot \{13\} = \{57\} \cdot \{10\} \oplus \{57\} \cdot \{02\} \oplus \{57\} \cdot \{01\}$$

Şimdi xtime'ı zincirleme uygulayarak $\{57\}$'nin ikinin kuvvetleriyle çarpımlarını üretelim:

$$\{57\} \cdot \{01\} = \{57\}$$
$$\{57\} \cdot \{02\} = \{AE\}$$
$$\{57\} \cdot \{04\} = \{AE\} \cdot \{02\} = \{47\}$$
$$\{57\} \cdot \{08\} = \{47\} \cdot \{02\} = \{8E\}$$
$$\{57\} \cdot \{10\} = \{8E\} \cdot \{02\} = \{07\}$$

İhtiyacımız olan üç halkayı XOR'layalım:

$$\{07\} \oplus \{AE\} \oplus \{57\}: \quad 00000111 \oplus 10101110 \oplus 01010111 = 11111110$$

Sonuç:

$$\{57\} \cdot \{13\} = \{FE\}$$

AES'in MixColumns adımı donanımda tam olarak bu yöntemle hesaplanır: kocaman bir cisim çarpması, birkaç bit kaydırma ve XOR'a dönüşür.
⊠

17.8 Her Baytın Tersi Var

Gelelim bu bölümü başlatan soruya: bölme çalışıyor mu? Evet. $m(x)$ indirgenemez olduğu için mod $m(x)$ polinom aritmetiği gerçekten bir cisimdir: **sıfır dışındaki 255 baytın her birinin çarpımsal tersi vardır** (ispatsız kabul ediyoruz). Üstelik tam sayılar için öğrendiğimiz genişletilmiş Öklid algoritması, neredeyse hiç değişmeden polinomlar için de çalışır ve bu tersleri hızla bulur. Örneğin $\{53\} \cdot \{CA\} = \{01\}$ olduğundan $\{53\}^{-1} = \{CA\}$'dır.
Bu tersler bir sonraki hedefimizin kalbinde atar: AES bölümünde göreceğimiz **S-kutusu**, her baytı önce $\mathrm{GF}(2^8)$'deki çarpımsal tersiyle değiştirir — algoritmanın doğrusal olmayan gücü, yani Shannon'un karışıklık prensibi buradan gelir. **MixColumns** adımı ise sütunları $\{01\}, \{02\}, \{03\}$ sabitleriyle çarparak yayılmayı sağlar.

17.9 Özet: Cisim Teorisi Çipe Sığıyor

Tablo 17.1: $\mathrm{GF}(2^8)$ aritmetiğinin donanımdaki karşılıkları

$\mathrm{GF}(2^8)$ işlemi	Bilgisayar nasıl yapar?	AES'te nerede?
Toplama = çıkarma	Tek bir XOR komutu	AddRoundKey (tur anahtarı ekleme)
$\{02\}$ ile çarpma	1 bit sola kaydır; taşarsa $\{1B\}$ ile XOR (xtime)	MixColumns
$\{03\}$ ile çarpma	xtime sonucu $\oplus$ baytın kendisi	MixColumns
Ters alma b^{-1}	256 girişlik hazır tablo	SubBytes (S-kutusu)

Soyut cebirin en zarif yapılarından biri, çipin üzerinde birkaç XOR kapısına ve bir kaydırıcıya dönüşüyor — kriptografinin güzelliği tam da bu köprüde saklı. Artık DES'in bit permütasyonlarına ve ardından bu aritmetiğin üzerinde yükselen AES'e hazırız.

18. DES (Data Encryption Standard) ve Feistel Ağı

DES, modern kriptografinin dönüm noktasıdır. 1970'lerde IBM tarafından "Lucifer" projesi adıyla geliştirilen ve 1977'de Amerikan Ulusal Standartlar Bürosu (NBS, günümüzdeki NIST) tarafından federal standart olarak kabul edilen, simetrik anahtarlı bir **blok şifreleme** algoritmasıdır.
Günümüzde 56 bitlik kısa anahtar boyutu nedeniyle kaba kuvvet saldırılarına karşı yetersiz kalıp yerini AES'e bırakmış olsa da; modern şifreleme mimarilerinin (Feistel ağları, S-kutusu ve P-kutusu mantığı) temelini anlamak için DES'i öğrenmek zorunludur.

18.1 Harflerden Bitlere Geçiş

Şu ana kadar gördüğümüz klasik şifrelemelerde (Sezar, afin, Vigenère, permütasyon) hep İngiliz alfabesi ($\mathbb{Z}_{26}$) üzerinden karakterlerin kimliğini veya yerini değiştirdik.
DES ile birlikte bu "insan odaklı" yapı tamamen terk edilir. Algoritma harflerin ne olduğunu bilmez; doğrudan **işlemcinin dilinde**, yani 0 ve 1'ler üzerinde çalışır. Açık metniniz —ister bir metin dosyası, ister fotoğraf, ister video olsun— önce makine diline çevrilir ve tamamen bit seviyesindeki işlemlerle (XOR, bit kaydırma, bit permütasyonları) şifrelenir.

18.2 Blok ve Anahtar Yapısı

DES algoritması veriyi bir bütün olarak değil, **64 bitlik bloklar** hâlinde işler. Mesaj ne kadar uzun olursa olsun sistem bunu 64 bitlik parçalara böler ve her bloğu sırayla şifreler.
Algoritmanın anahtar yapısı ise kriptografi tarihinin en ilginç tasarımlarından biridir:

- **Girdi anahtarı:** DES'e dışarıdan verilen orijinal anahtar **64 bittir**.
- **Efektif anahtar boyutu:** Orijinal anahtarın her 8. biti (8, 16, 24, ..., 64) bir **eşlik biti (parity bit)** olarak hata kontrolü amacıyla kullanılır ve şifreleme işlemine dâhil edilmez. Bu yüzden DES'in gerçek kriptografik gücü **56 bit** ile sınırlıdır.

18.3 DES Anahtar Üretim Algoritması (Key Schedule)

DES, her 64 bitlik veri bloğunu şifrelemek için tam **16 döngü (round)** kullanır. Şifrelemenin güvenli olabilmesi için bu 16 döngünün her birine, orijinal anahtardan türetilmiş **farklı ve benzersiz 48 bitlik alt anahtarlar** ($K_1, K_2, ..., K_{16}$) verilmesi gerekir.
64 bitlik tek bir kök anahtardan 16 farklı alt anahtar üreten bu mekanizmaya **anahtar üretimi (key schedule)** denir. İşlem üç temel aşamada gerçekleşir.

18.3.1 Adım 1 — PC-1: Eşlik Bitlerini Eleme ve İkiye Bölme

Sisteme giren orijinal kök anahtar 64 bitten oluşur:

$$K = k_1, k_2, k_3, ..., k_{63}, k_{64}$$

Algoritma bu anahtarın her 8. bitini eşlik biti olarak görür ve şifrelemeye katmaz. Anahtar dizisi **PC-1 (Permuted Choice 1)** matrisinden geçirilir; bu matrisin içinde eşlik bitlerinin indeksleri bulunmadığından 8 adet eşlik biti otomatik olarak elenir.

Tablo 18.1: PC-1 tablosu — 56 bit seçimi

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Tablonun uygulanma mantığı. Bu işlemi, daha önce işlediğimiz **ayrık permütasyon fonksiyonu** mantığıyla düşünmeliyiz. Orijinal 64 bitlik anahtarın bitlerini k, oluşacak yeni 56 bitlik dizinin bitlerini k' olarak tanımlayalım. PC-1 tablosu aslında bizim π permütasyon anahtarımızdır ve bitlerin konum indeksleri üzerinde çalışır:

$$k'_\tau = k_{\pi(\tau)}, \quad \tau \in \{1, 2, \ldots, 56\}$$

Tablonun ilk elemanlarına göre:

$$\pi(1) = 57 \Longrightarrow k'_1 = k_{57}, \qquad \pi(2) = 49 \Longrightarrow k'_2 = k_{49}, \qquad \ldots$$

Elde edilen bu 56 bit, tablonun üst yarısına karşılık gelen C_0 ve alt yarısına karşılık gelen D_0 olmak üzere ikiye bölünür:

$$K_{56} = \underbrace{k_{57}, k_{49}, \ldots, k_{36}}_{C_0 \text{(sol yarı } -28 \text{ bit)}} \underbrace{k_{63}, k_{55}, \ldots, k_4}_{D_0 \text{(sağ yarı } -28 \text{ bit)}}$$

18.3.2 Adım 2 — Dairesel Sola Kaydırma (Circular Left Shift)

Algoritma 16 döngü boyunca C ve D yarılarını kendi içlerinde sola kaydırarak sürekli günceller. "Dairesel" olmasının anlamı şudur: **sınırın dışına çıkan bit silinmez, bloğun en sağına geri döner.**

i Görsel örnek: 1 bit dairesel kaydırma

8 bitlik bir X bloğumuz olsun: $X = \mathbf{1}0110011$.
Bir bit sola kaydırıldığında en soldaki **1** kopar ve en sağa kuyruk olur:

$$X' = 01100111$$

Kaydırma miktarı döngü numarasına göre sabittir:

- **1., 2., 9. ve 16. döngülerde:** yalnızca **1 bit** dairesel kaydırılır.
- **Diğer tüm döngülerde: 2 bit** dairesel kaydırılır.

Örneğin ilk iki döngü için yeni yarılar şöyle oluşur:

$$C_1 = \text{LeftShift}\ (C_0, 1), \qquad D_1 = \text{LeftShift}\ (D_0, 1)$$
$$C_2 = \text{LeftShift}\ (C_1, 1), \qquad D_2 = \text{LeftShift}\ (D_1, 1)$$

18.3.3 Adım 3 — PC-2: Sıkıştırma Permütasyonu

Döngüye ait kaydırılmış C_i ve D_i yarıları yan yana getirilerek birleştirilir ($28 + 28 = 56$ bit). Ancak şifreleme çekirdeği (Feistel fonksiyonu) her döngüde **48 bitlik** bir anahtara ihtiyaç duyar.

Bu noktada **PC-2 (Permuted Choice 2)** tablosu devreye girer. Tablo bir filtre gibi davranır: 56 bitlik diziyi karıştırır ve içinden önceden belirlenmiş 8 biti atarak (sıkıştırarak) o döngünün 48 bitlik K_i alt anahtarını üretir.

Tablo 18.2: PC-2 tablosu — 48 bit seçimi

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Matematiksel dönüşümün özeti şöyledir:

$$\underbrace{C_i}_{28\text{ -bit}} \;\|\; \underbrace{D_i}_{28\text{ -bit}} \xrightarrow{\text{birleştirilir}} CD_i\ (56\text{ -bit})$$

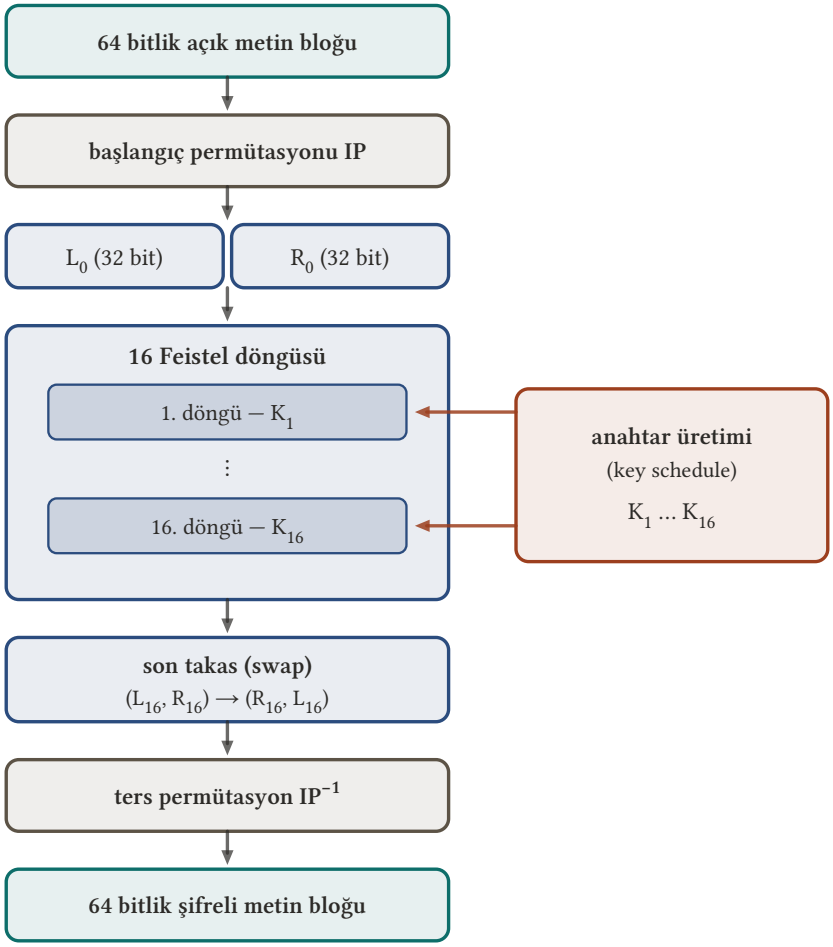
$$CD_i\ (56\text{ -bit}) \xrightarrow{\text{PC-2 filtresi}} K_i\ (48\text{ -bit alt anahtar})$$

Bu üç adım 16 döngünün tamamı için zincirleme olarak tekrarlanır ve şifreleme işleminde kullanılacak 16 alt anahtardan oluşan set hazır hâle getirilmiş olur. Artık elimizde $K_1, K_2, \ldots, K_{16}$ var; sıra bu anahtarlarla verinin kendisini şifrelemeye geldi.

18.4 Şifrelemenin Genel Akışı

64 bitlik bir açık metin bloğu, DES'in içinden şu duraklardan geçerek çıkar:

1. **Başlangıç permütasyonu (IP — Initial Permutation):** Bloğun 64 biti, sabit bir tabloya göre yeniden sıralanır.
2. **16 Feistel döngüsü:** Blok iki yarıya bölünür ve her döngüde anahtar üretiminden gelen $K_1, K_2, \ldots, K_{16}$ alt anahtarlarından biri kullanılarak dönüştürülür.
3. **Son takas (swap):** 16. döngünün ardından 32 bitlik iki yarı birbiriyle yer değiştirir.
4. **Ters permütasyon (IP^{-1}):** Başlangıçtaki sıralama işleminin tam tersi uygulanır ve 64 bitlik şifreli metin bloğu elde edilir.



Şekil 18.1: DES'in kuşbakışı akışı: 64 bitlik blok başlangıç permütasyonu IP'den geçip iki yarıya bölünür, anahtar üretiminden gelen K_1, …, K_{16} alt anahtarlarıyla 16 Feistel döngüsünde işlenir; son takas ve IP^{-1} ile şifreli metin elde edilir.

Akışı sembollerle özetlersek:

$$\text{açık metin} \xrightarrow{IP} (L_0, R_0) \xrightarrow{\text{16 döngü}} (L_{16}, R_{16}) \xrightarrow{\text{takas}} (R_{16}, L_{16}) \xrightarrow{IP^{-1}} \text{şifreli metin}$$

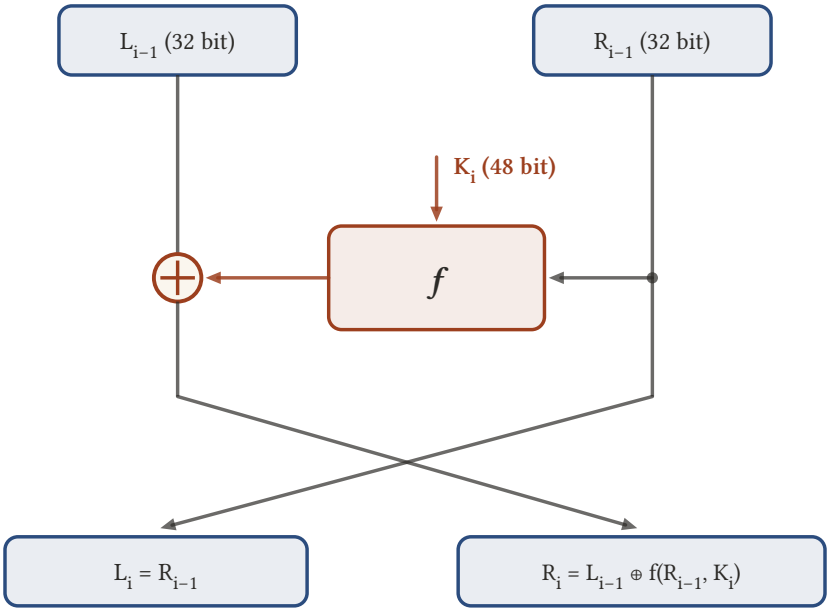
> **i** IP güvenliğe hiçbir şey katmaz
>
> Başlangıç permütasyonu IP ve tersi IP^{-1}, bitleri yalnızca sabit ve herkesçe bilinen bir düzende yeniden sıralar; anahtara hiç dokunmaz. Bu yüzden **kriptografik güce sıfır katkısı vardır.** Varlık nedeni tamamen tarihseldir: 1970'lerin donanımlarında verinin yongaya yükleniş sırasını kolaylaştırmak için eklenmiştir. Matematiksel bir ders vermediği için bu tabloyu basmıyoruz; asıl olay birazdan geleceğimiz 16 döngünün içindedir.

18.5 Feistel Yapısı

IP'den çıkan 64 bitlik blok, tıpkı anahtarın C_0 ve D_0 olarak bölünmesi gibi, iki eşit yarıya ayrılır: soldaki 32 bit L_0 (left), sağdaki 32 bit R_0 (right). Bundan sonraki 16 döngünün her biri, IBM mühendisi Horst Feistel'in adını taşıyan tek ve çok zarif bir kurala göre işler:

$$L_i = R_{i-1}, \qquad R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

Kelimelerle söylersek: **sağ yarı hiç değişmeden çaprazlama sola taşınır**; sol yarı ise, sağ yarının f adlı bir fonksiyondan geçirilmiş hâliyle XOR'lanarak yeni sağ yarıyı oluşturur. f'nin içini birazdan açacağız; şimdilik onu "sağ yarı ile alt anahtarı öğüten bir karıştırıcı kutu" olarak düşünmemiz yeterli.



Şekil 18.2: Tek bir Feistel döngüsü: sağ yarı R_{i-1} hiç değişmeden çaprazlama sola taşınır; sol yarı L_{i-1} ise sağ yarının f fonksiyonundan geçmiş hâliyle XOR'lanarak yeni sağ yarıyı oluşturur. Anahtar devreye yalnızca f'nin içinde girer.

Bu yapının neden dâhice olduğunu görmek için kendimize şu soruyu soralım: mesajı alan taraf bu işlemi nasıl geri saracak? f karmaşık bir karıştırıcıysa, geri dönmek için f'nin tersini hesaplamak gerekmez mi? İşte Feistel yapısının bütün sırrı, cevabın **hayır** olmasıdır.

Teorem 18.1 (Feistel Yapısının Tersinirliği). *f fonksiyonu ne olursa olsun — tersinir olmasa, hatta bilgi kaybettirse bile — Feistel döngüsü her zaman tersinirdir. (L_i, R_i) çiftinden bir önceki (L_{i-1}, R_{i-1}) çifti şu formüllerle geri kazanılır:*

$$R_{i-1} = L_i, \qquad L_{i-1} = R_i \oplus f(L_i, K_i)$$

İspat.
Birinci eşitlik döngü kuralının doğrudan kendisidir: $L_i = R_{i-1}$ tanımlandığına göre $R_{i-1} = L_i$ olur; sağ yarı zaten hiç şifrelenmeden karşıya taşınmıştır.
İkinci eşitlik için R_i'nin tanımını yerine yazalım ve az önce bulduğumuz $R_{i-1} = L_i$ eşitliğini kullanalım:

$$R_i \oplus f(L_i, K_i) = \Big(L_{i-1} \oplus f(R_{i-1}, K_i) \Big) \oplus f(R_{i-1}, K_i)$$

XOR bölümünde gördüğümüz birleşme özelliğiyle parantezi kaydıralım ve kendi kendini yok etme özelliğini ($A \oplus A = 0$) uygulayalım:

$$= L_{i-1} \oplus \underbrace{\Big(f(R_{i-1}, K_i) \oplus f(R_{i-1}, K_i) \Big)}_{=0} = L_{i-1} \oplus 0 = L_{i-1}$$

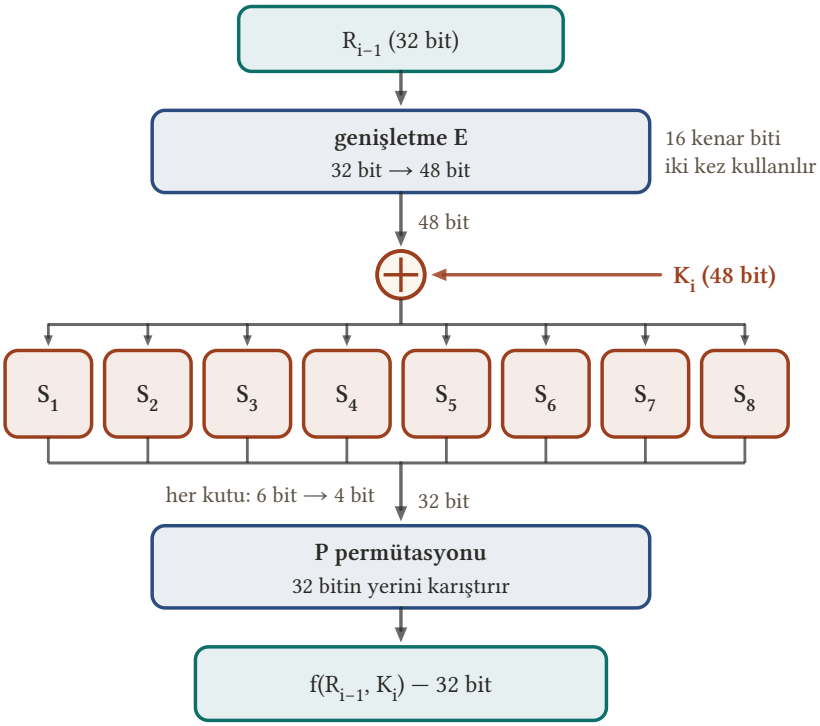
*Dikkat edilirse bu adımların hiçbirinde f'nin tersini almadık; f'yi yalnızca **ileri yönde**, elimizde zaten bulunan $L_i = R_{i-1}$ girdisiyle bir kez daha hesapladık.* ■

Şifre çözme = aynı devre, anahtarlar ters sırada

Teorem 18.1'in pratik sonucu muazzamdır: DES'te şifre çözmek için **ayrı bir algoritma yoktur.** Şifreli metni aynı 16 döngülük devreye sokar, alt anahtarları yalnızca ters sırada $(K_{16}, K_{15}, \ldots, K_1)$ verirsiniz; devre her döngüyü kendiliğinden geri sarar.
Bu, 1970'lerin pahalı donanım çağında bir deha hamlesiydi: tek bir yonga hem şifreleme hem çözme yapabiliyordu. Daha da önemlisi, tasarımcılara tam bir özgürlük verdi — madem yapının tersinirliği f'ye hiç bağlı değil, o hâlde f **istediğimiz kadar vahşi ve doğrusal olmayan** bir karıştırıcı olabilir. Şimdi bu özgürlüğün nasıl kullanıldığına bakalım.

18.6 Kalp: f Fonksiyonu

DES'in bütün kriptografik gücü, her döngüde çalışan $f(R_{i-1}, K_i)$ fonksiyonunda toplanmıştır. f, 32 bitlik sağ yarıyı ve 48 bitlik alt anahtarı alıp 32 bitlik bir çıktı üretir; bunu dört aşamada yapar.



Şekil 18.3: f fonksiyonunun dört aşaması: E genişletmesi 32 biti 48 bite çıkarır, sonuç alt anahtar K_i ile XOR'lanır, 48 bit sekiz S-kutusundan geçerek 32 bite iner ve P permütasyonu bu bitleri bloğun geneline dağıtır.

18.6.1 Aşama 1 — Genişletme E: 32 Bitten 48 Bite

İlk sorun boyut uyuşmazlığıdır: elimizdeki yarı 32 bit, alt anahtar K_i ise 48 bittir. **Genişletme tablosu E (expansion)**, tıpkı PC-1 ve PC-2 gibi konum indeksleriyle çalışan bir seçim tablosudur; ancak bu kez bit atmak yerine bit **çoğaltır**:

Tablo 18.3: E genişletme tablosu — 32 bitten 48 bite

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Tabloyu dikkatle sayarsanız 48 hücrede yalnızca 32 farklı indeks görürsünüz: **16 kenar biti iki kez kullanılır** ($48 = 32 + 16$). Bu bilinçli bir tekrardır — bir bit iki komşu gruba birden sızdığı için, tek bir girdi bitindeki değişiklik ilerleyen aşamalarda birden fazla S-kutusunu etkiler. Çığ etkisini besleyen **yayılma (diffusion)** çarklarından ilki burada döner.

18.6.2 Aşama 2 — Alt Anahtarla Karıştırma

Genişletilmiş 48 bit, o döngünün alt anahtarıyla XOR'lanır:

$$E(R_{i-1}) \oplus K_i$$

Gösterişsiz görünen bu tek satır aslında kritik bir gerçeği saklar: **anahtar, DES'in veri yoluna yalnızca burada girer.** Permütasyonlar, genişletmeler, S-kutuları — hepsi sabit ve herkesçe bilinir; gizli olan tek şey bu XOR'a karışan 48 bittir.

18.6.3 Aşama 3 — S-Kutuları: Doğrusal Olmayan Kalp

XOR'dan çıkan 48 bit, 6'şar bitlik **8 gruba** bölünür ve her grup kendi **S-kutusuna (substitution box)** girer: $S_1, S_2, \ldots, S_8$. Her kutu 6 bitlik girdisini 4 bitlik bir çıktıya dönüştürür; böylece $8 \cdot 6 = 48$ bit içeri girer, $8 \cdot 4 = 32$ bit dışarı çıkar.
Bir S-kutusu, satır ve sütunlardan oluşan sabit bir arama tablosudur. 6 bitlik $b_1 b_2 b_3 b_4 b_5 b_6$ girdisinin adresi şöyle çözülür:

- **Dış iki bit** $b_1 b_6$ birleştirilir ve satır numarasını verir: 0–3 arası.
- **İç dört bit** $b_2 b_3 b_4 b_5$ sütun numarasını verir: 0–15 arası.

Kesişimdeki sayı (0–15 arası) kutunun çıktısıdır ve 4 bit olarak yazılır. İşte S_1 kutusunun resmî tablosu:

Tablo 18.4: S1 kutusu — satırlar dış bitlerle, sütunlar iç bitlerle seçilir

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Örnek 18.1. S_1 kutusuna 6 bitlik 011011 dizisi giriyor. Kutunun 4 bitlik çıktısını bulunuz.

> **Çözüm.**
> Girdiyi bit bit adlandıralım: $b_1 b_2 b_3 b_4 b_5 b_6 = 0\,1\,1\,0\,1\,1$.

Satır: dış bitler $b_1 b_6 = 01$, yani ikilik sistemde $01_2 = 1$. Satır 1.
Sütun: iç bitler $b_2 b_3 b_4 b_5 = 1101$, yani $1101_2 = 8 + 4 + 0 + 1 = 13$. Sütun 13.
Tablo 18.4′de satır 1 ile sütun 13'ün kesişiminde 5 yazar. Çıktı, 5 sayısının 4 bitlik gösterimidir:

$$011011 \xrightarrow{S_1} 5 = 0101$$

Kutuya 6 bit girdi, 4 bit çıktı; dizinin boyu kutunun içinde eridi. ⊠

$S_2, S_3, \ldots, S_8$ kutuları da tamamen aynı mantıkla çalışan, standartta sabitlenmiş farklı tablolardır; sekizini birden basarak sayfayı işgal etmeyeceğiz.
Asıl vurgulanması gereken şudur: permütasyonlar, genişletme ve XOR'un tamamı **doğrusal** işlemlerdir — girdiyle çıktı arasında denklem sistemleriyle çözülebilecek düzgün bir cebirsel ilişki korurlar. **S-kutuları ise DES'in doğrusal olmayan tek bileşenidir.** Shannon'un diliyle söylersek, **karışıklık (confusion)** prensibinin bu şifredeki tek taşıyıcısı S-kutularıdır: anahtar ile şifreli metin arasındaki ilişkiyi denklemle izlenemez hâle getiren adım budur. Tarihsel bir not olarak, 1970′lerde NSA'in bu kutuların sabitlerini kimseye gerekçe göstermeden değiştirmesi yıllarca "arka kapı mı gizlendi?" tartışması yaratmış; ancak diferansiyel kriptanaliz 1990′larda kamuya açık olarak keşfedilince, değişikliklerin kutuları tam da bu saldırıya karşı **güçlendirdiği** anlaşılmıştır.

18.6.4 Aşama 4 — P Permütasyonu

Sekiz kutudan çıkan $8 \cdot 4 = 32$ bit son olarak **P permütasyonundan** geçer; bu tablo hiçbir biti atmadan 32 bitin yerlerini değiştirir:

Tablo 18.5: P permütasyon tablosu — 32 bitlik çıktının yeniden sıralanması

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

P'nin görevi, her S-kutusunun 4 bitlik çıktısını bloğun dört bir yanına savurmaktır: aynı kutudan çıkan bitler, bir sonraki döngüde E genişletmesi tarafından **farklı** S-kutularına dağıtılır. Bu, Shannon'un **yayılma (diffusion)** prensibinin ta kendisidir — tek bir bitlik değişim, birkaç döngü içinde bloğun tamamına bulaşır ve çığ etkisi doğar.

18.7 Şifre Çözme

Şifre çözme bölümü, DES anlatımlarının en kısa bölümüdür ve bunu Teorem 18.1′e borçluyuz. Alıcı taraf, şifreli metni **aynı algoritmadan** geçirir; tek fark alt anahtarların sırasıdır: ilk döngüye K_{16}, ikincisine K_{15}, sonuncusuna K_1 verilir. Her döngü, teoremdeki $R_{i-1} = L_i$ ve $L_{i-1} = R_i \oplus f(L_i, K_i)$ formülleri gereği bir öncekini geri sarar; f'nin, S-kutularının veya P'nin tersini hesaplamak hiçbir zaman gerekmez. Akışın uçlarındaki yardımcı adımlar da kendiliğinden temizlenir: IP ile IP^{-1} zaten birbirinin tersidir, son takas ise iki yarıyı değiştirmekten ibaret olduğundan kendi kendisinin tersidir. Yeni bir matematik yok; şifreleme devresini kurduğumuz anda çözme devresini de bedavaya almış olduk.

18.8 DES'in Sonu: Kaba Kuvvetin Zaferi

DES'in mimarisi bugün bile saygı duyulan bir tasarımdır; onu emekliye ayıran şey yapısındaki bir çatlak değil, anahtarının kısalığıdır. Efektif anahtar 56 bit olduğuna göre anahtar uzayı

$$2^{56} \approx 7,2 \cdot 10^{16}$$

anahtardan oluşur. 1977'de bu sayı aşılamaz bir duvar gibi görünüyordu; ancak Moore yasası duvarı her 18 ayda bir inceltti ve 1990'ların sonunda üç darbe art arda geldi:

- **1997 — DESCHALL:** RSA firmasının ödüllü meydan okuması, internet üzerinden anahtar uzayını paylaşan binlerce gönüllü bilgisayarın ortak çalışmasıyla yaklaşık 96 günde sonuçlandı; bir DES anahtarı ilk kez halka açık biçimde kaba kuvvetle bulundu.
- **1998 — Deep Crack:** Electronic Frontier Foundation (EFF), yaklaşık 250 000 dolarlık bütçeyle **Deep Crack** adında özel amaçlı bir makine inşa etti. Bu makine bir DES anahtarını yaklaşık **56 saatte** buldu — artık sıradan zengin bir kurumun bile DES kırabileceği kanıtlanmıştı.
- **1999 — 22 saat:** Deep Crack ile distributed.net ağındaki on binlerce bilgisayar güçlerini birleştirdi ve süre **22 saat 15 dakikaya** düştü.

Akademik cephede diferansiyel kriptanaliz (2^{47} seçilmiş açık metin) ve lineer kriptanaliz (2^{43} bilinen açık metin) gibi kaba kuvvetten "daha akıllı" saldırılar da bulunmuştur; ancak bu kadar veriyi toplamak pratikte imkânsıza yakındır. DES'i tarihe gömen, zarif bir matematiksel gedik değil, ham hesaplama gücü olmuştur.

18.9 Çifte DES Neden Çare Değil? Ortada Buluşma Saldırısı

Anahtar kısaysa akla ilk gelen yama bellidir: DES'i iki farklı anahtarla **art arda iki kez** uygulamak. Bu yapıya **2DES** denir:

$$C = E_{K_2}\big(E_{K_1}(P)\big)$$

Toplam $56 + 56 = 112$ bitlik anahtar malzemesi kullanıldığına göre güvenliğin de 2^{112}'lik bir aramaya denk gelmesini bekleriz. Ne yazık ki bu beklenti yanlıştır. Denklemin iki tarafına D_{K_2} uygularsak şifrelemenin tam ortasındaki değeri iki yönden de yakalayabileceğimizi görürüz:

$$D_{K_2}(C) = E_{K_1}(P)$$

Eşitliğin sağı orta değere **açık metinden ileri giderek**, solu ise **şifreli metinden geri gelerek** ulaşır. Elinde tek bir (P, C) çifti olan bir saldırgan bunu şöyle sömürür: önce 2^{56} olası K_1 anahtarının her biri için $E_{K_1}(P)$ değerini hesaplayıp sonuçları bir tabloya koyar. Ardından 2^{56} olası K_2 anahtarının her biri için $D_{K_2}(C)$ değerini hesaplar ve her sonucu tabloda arar. İki liste **ortada buluştuğunda** — yani bir $D_{K_2}(C)$ değeri tabloda bulunduğunda — eldeki (K_1, K_2) çifti güçlü bir adaydır; birkaç yedek (P, C) çiftiyle doğrulanarak kesinleştirilir.
Bu saldırıya **ortada buluşma (meet-in-the-middle)** denir ve maliyeti iki tam aramanın toplamıdır: $2^{56} + 2^{56} = 2^{57}$ DES işlemi (artı 2^{56} girdilik tablo için devasa ama ilkesel olarak mümkün bir bellek). Yani 2DES, kâğıt üstünde 112 bitlik görünse de, kırması **tek DES'in yalnızca iki katı iş gerektiren** bir hedeftir.

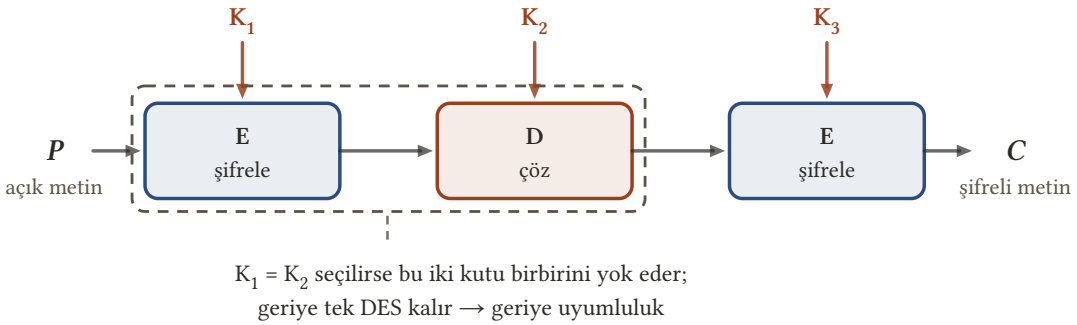
Örnek 18.2. Ortada buluşma saldırısı, 2DES'e yönelik saf kaba kuvvet aramasını kaç kat ucuzlatır?

> **Çözüm.**
> Saf kaba kuvvet, 112 bitlik anahtar malzemesinin tamamını tarar: 2^{112} deneme. Ortada buluşma ise 2^{57} işlemle sonuca ulaşır. Kazanç oranı:
>
> $$\frac{2^{112}}{2^{57}} = 2^{55} \approx 3,6 \cdot 10^{16}$$
>
> Saldırı, işlem sayısını yaklaşık 36 katrilyon kat azaltır; "iki kat şifrele, iki kat güvenli ol" sezgisi paramparça olur. ⊠

18.10 Üçlü DES (3DES)

Madem iki katman yetmiyor, üç katmana çıkalım. **3DES (Triple DES)**, DES'i üç anahtarla art arda uygular; ancak dizilim şaşırtıcıdır — şifrele, **çöz**, şifrele:

$$C = E_{K_3}\Big(D_{K_2}\big(E_{K_1}(P)\big)\Big)$$



Şekil 18.4: 3DES'in E–D–E zinciri: açık metin önce K_1 ile şifrelenir, K_2 ile çözülür, K_3 ile yeniden şifrelenir. $K_1 = K_2$ seçildiğinde ilk iki kutu birbirini yok eder ve sistem tek DES'e dönüşür — eski donanımla geriye uyumluluğun sırrı.

Ortadaki adımın şifre çözme olması güvenlikle ilgili değildir; D de E kadar iyi bir karıştırıcıdır. Sebep tamamen mühendislik zekâsıdır:

> **E-D-E dizilimi ve geriye uyumluluk**
>
> Üç anahtar da aynı seçilirse ($K_1 = K_2 = K_3 = K$) ortadaki D_K, ilk E_K'yi tam olarak geri alır ve zincirden geriye yalnızca son E_K kalır:
>
> $$E_K\Big(D_K\big(E_K(P)\big)\Big) = E_K(P)$$
>
> Yani bir 3DES yongası, anahtarları eşitleyerek **sıradan bir DES cihazı gibi** çalıştırılabilir. Sahadaki milyonlarca eski DES donanımıyla haberleşebilmek — 1990'ların bankacılık dünyasında paha biçilmez bir özellik — E-D-E diziliminin tek ve yeterli gerekçesidir. E-E-E dizilimi bu zarif geri çekilme yolunu kapatırdı.

Anahtarların nasıl seçildiğine göre üç standart kullanım biçimi vardır:

Tablo 18.6: 3DES anahtarlama seçenekleri

Seçenek	Anahtar ilişkisi	Nominal boyut	Efektif güvenlik
3 bağımsız anahtar	$K_1 \neq K_2 \neq K_3$	168 bit	Ortada buluşma nedeniyle ≈ 112 bit
2 anahtar	$K_1 = K_3 \neq K_2$	112 bit	Pratik saldırılar daha da aşağı çeker
Tek anahtar	$K_1 = K_2 = K_3$	56 bit	Tek DES'e eşdeğer — yalnızca geriye uyumluluk

Tabloya dikkat: üç bağımsız anahtar bile 168 bitlik nominal gücü vermez; bir önceki bölümdeki ortada buluşma fikri 3DES'e de uyarlanabildiğinden efektif güvenlik 2^{112} mertebesinde kalır.

3DES görevini onlarca yıl onurluca yaptı, ama iki yapısal yükten hiç kurtulamadı: her blok için DES'in üç kez çalışması gerektiğinden **üç kat yavaştır** ve blok boyu hâlâ **64 bittir** — büyük veri hacimlerinde doğum günü paradoksu yüzünden blok çakışmaları istatistiksel bilgi sızdırmaya başlar (2016′da gösterilen Sweet32 zafiyetinin özü budur). Nitekim NIST, 3DES'i 2019′da resmen "kullanımdan kaldırılıyor (deprecated)" statüsüne aldı ve 31 Aralık 2023 itibarıyla yeni uygulamalardaki kullanımını tamamen sonlandırdı. Kısa anahtar sorununa yama üstüne yama yapmak yerine sıfırdan, modern ilkelerle tasarlanmış bir şifreye geçmek gerekiyordu. O şifre, bir sonraki bölümün konusu olan AES'tir.

19. AES (Advanced Encryption Standard) ve SPN Yapısı

DES bölümünde gördüğümüz tablo açıktı: 56 bitlik efektif anahtar, 1990'ların sonunda kaba kuvvet saldırılarına günler hatta saatler içinde yenik düşüyordu. Geçici çare olan 3DES ise güvenliği kurtarsa da üç kat şifreleme yükü nedeniyle yavaştı. Dijital çağın yeni bir standarda ihtiyacı vardı.

19.1 DES'ten AES'e: Açık Bir Yarışma

1997'de NIST (Amerikan Ulusal Standartlar ve Teknoloji Enstitüsü) tarihinin en önemli kararlarından birini verdi: yeni standardı kapalı kapılar ardında değil, **herkese açık bir yarışmayla** seçecekti. Dünyanın dört bir yanından gelen **15 aday algoritma** 1998'de duyuruldu; üç yıl boyunca uluslararası kriptografi topluluğu bu adayları kırmak için yarıştı. Ekim 2000'de kazanan ilan edildi: Belçikalı kriptograflar **Joan Daemen** ve **Vincent Rijmen**'in tasarladığı **Rijndael** algoritması. Algoritma, Kasım 2001'de **FIPS-197** belgesiyle resmî standart hâline geldi ve o günden beri **AES (Advanced Encryption Standard)** adıyla anılıyor.

> **i** Açıklık bir zayıflık değil, güçtür
>
> Bu yarışma, Kerckhoffs prensibinin eylemdeki hâlidir: algoritmanın her satırı yıllarca dünyanın en iyi kriptanalistlerinin saldırısına açık tutulmuş, güvenlik yalnızca **anahtarın gizliliğine** emanet edilmiştir. DES'in S-kutularının gizemli tasarım kriterlerinin yarattığı "arka kapı mı var?" kuşkusunun tam tersi bir yaklaşımdır bu.

Mimari açıdan AES, DES'ten köklü biçimde ayrılır. DES bir **Feistel ağıydı**: her döngüde bloğun yalnızca yarısı işlenir, bu sayede çekirdek fonksiyon tersinir olmasa bile şifre çözülebilirdi. AES ise Shannon bölümünde tanıttığımız **SPN (Substitution-Permutation Network — yerine koyma–permütasyon ağı)** mimarisini kullanır: her döngüde **bloğun tamamı** dönüştürülür. Bunun bedeli, şifre çözmenin çalışabilmesi için her adımın **kendi başına tersinir** olmak zorunda olmasıdır — birazdan her adımın bu koşulu nasıl sağladığını tek tek göreceğiz.

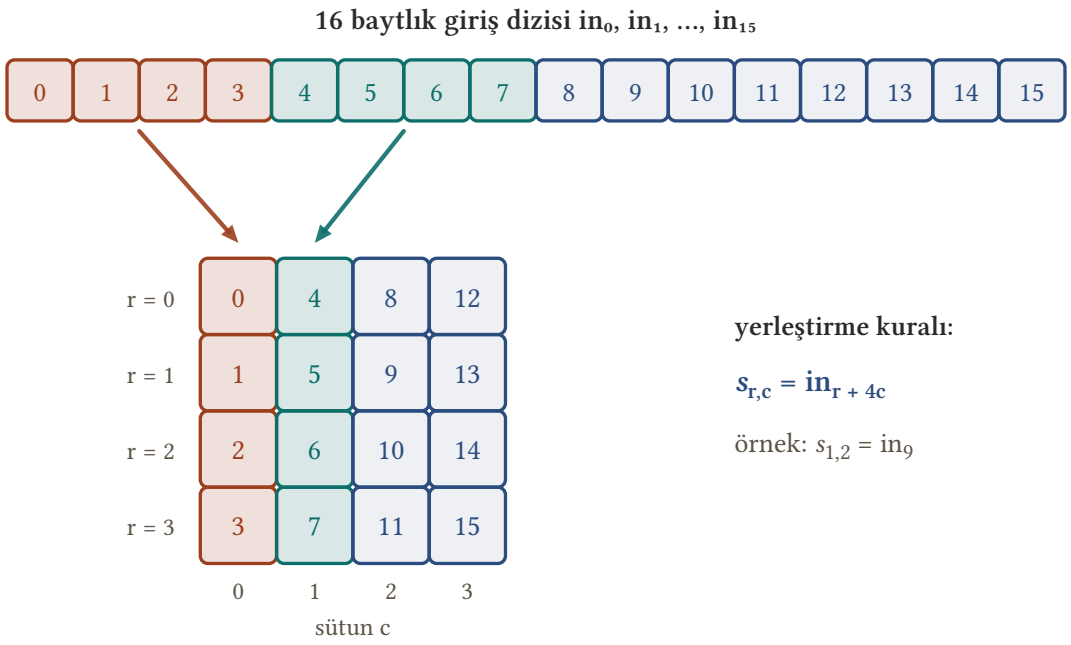
19.2 Durum Matrisi (State)

AES, veriyi her zaman **128 bitlik bloklar** hâlinde işler. Algoritmanın zarif taraflarından biri, bu 128 biti uzun bir bit dizisi olarak değil, kompakt bir tablo olarak ele almasıdır.

Tanım 19.1 (Durum Matrisi (State)). 128 bitlik blok 16 bayta bölünür ve baytlar **sütun sütun** 4×4'lük bir matrise yerleştirilir. Bu matrise **durum (state)** denir. Giriş dizisinin baytlarını $\text{in}_0, \text{in}_1, \ldots, \text{in}_{15}$ ile gösterirsek yerleştirme kuralı şudur:

$$s_{r,c} = \text{in}_{r+4c}, \qquad 0 \le r, c \le 3$$

Yani ilk dört bayt birinci sütunu, sonraki dört bayt ikinci sütunu doldurur. Örneğin $s_{1,2} = \text{in}_9$ olur.



Şekil 19.1: 128 bitlik blok, 16 bayt olarak **sütun sütun** 4×4 durum matrisine yerleştirilir: ilk dört bayt birinci sütunu, sonraki dört bayt ikinci sütunu doldurur. Hücredeki sayı baytın giriş dizisindeki sırasıdır; kural $s_{r,c} = \text{in}_{r+4c}$.

Bundan sonraki her AES işlemi bu matris üzerinde çalışır: kimi adım hücreleri tek tek, kimi satırları, kimi de sütunları dönüştürür. Kritik nokta şudur: matrisin her hücresi sıradan bir bayt değil, bir önceki bölümde inşa ettiğimiz $\text{GF}(2^8)$ cisminin bir elemanıdır. Toplama XOR'dur, çarpma $m(x)$ modülünde polinom çarpımıdır — AES'in tüm aritmetiği bu cisimde döner.

19.3 Üç Sürüm, Tek Blok Boyutu

AES'in üç resmî sürümü vardır ve aralarındaki fark yalnızca **anahtar uzunluğudur**; blok her sürümde 128 bittir. Anahtar, 32 bitlik **kelimeler (word)** hâlinde sayılır ve kelime sayısı N_k ile gösterilir. Anahtar uzadıkça döngü sayısı N_r de artar:

Tablo 19.1: AES sürümleri — blok her zaman 128 bittir, yalnızca anahtar uzunluğu değişir

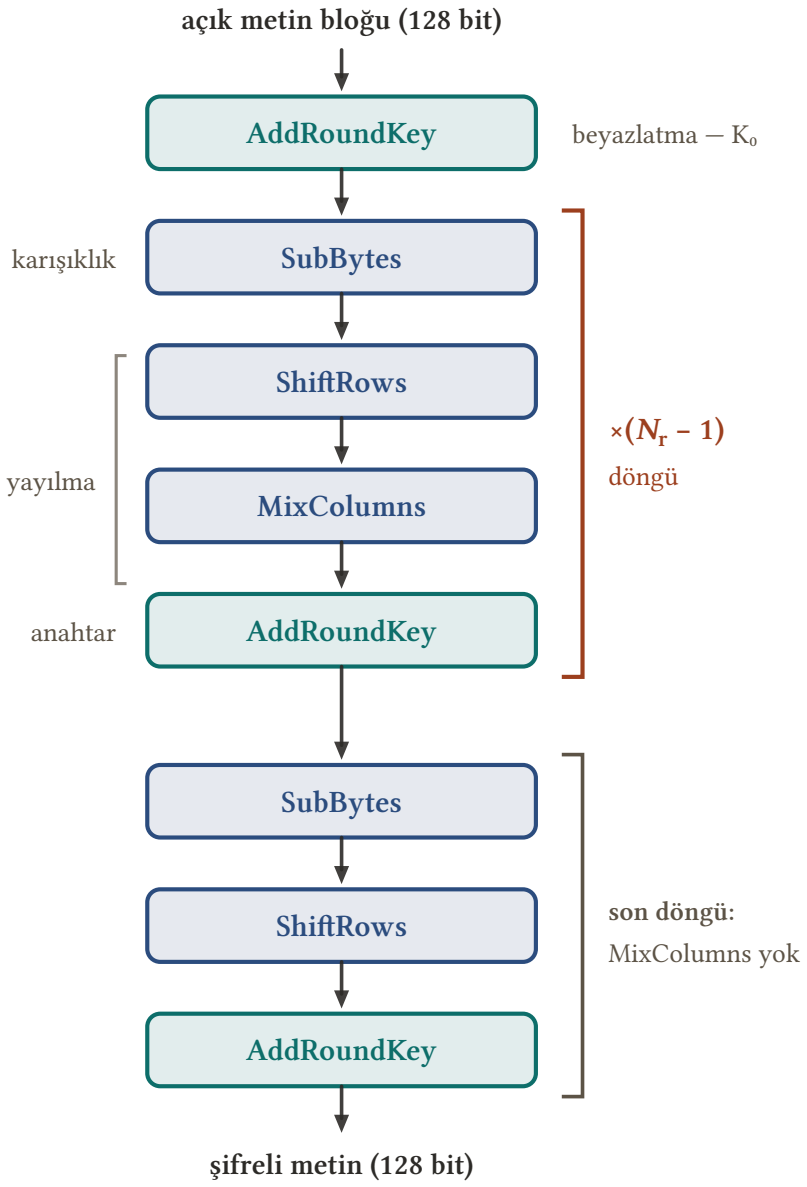
Sürüm	Anahtar uzunluğu	Anahtar kelime sayısı (N_k)	Döngü sayısı (N_r)	Tur anahtarı sayısı ($N_r + 1$)
AES-128	128 bit	4	10	11
AES-192	192 bit	6	12	13
AES-256	256 bit	8	14	15

Daha uzun anahtar neden daha çok döngü gerektirir? Uzun anahtar daha büyük bir saldırı yüzeyi (özellikle anahtar genişletme üzerinden gelen ilişkili-anahtar teknikleri) sunar; ek döngüler bu sürümlere daha geniş bir **güvenlik payı** bırakır. Tur anahtarı sayısının $N_r + 1$ olmasının nedenini de birazdan göreceğiz: döngüler başlamadan önce fazladan bir anahtar karıştırma adımı vardır.

19.4 Bir Döngünün Anatomisi

Şifreleme, durum matrisinin üzerinden defalarca geçen dört temel işlemin bestesidir. Akış şöyledir:

1. **Başlangıç:** Daha ilk döngü başlamadan durum, ilk tur anahtarıyla XOR'lanır (**AddRoundKey**). Bu adıma **beyazlatma (whitening)** denir; amaç, saldırganın anahtardan bağımsız tek bir işlem bile gözlemleyememesidir.
2. **Tam döngüler:** $N_r - 1$ kez şu dörtlü sırayla uygulanır: **SubBytes → ShiftRows → MixColumns → AddRoundKey**.
3. **Son döngü:** Aynı sıra, ama **MixColumns atlanarak**: SubBytes → ShiftRows → AddRoundKey.



Şekil 19.2: AES şifrelemesinin akışı: önce beyazlatma amacıyla bir **AddRoundKey**, ardından $N_r - 1$ kez tekrarlanan tam döngü ve MixColumns adımı atılmış son döngü. Mavi kutular veri dönüşümlerini, yeşil kutular anahtarın karıştırıldığı adımı gösterir.

Son döngüde MixColumns'un atlanmasının nedeni basittir: MixColumns doğrusal ve anahtardan bağımsız olduğundan, sondaki bir MixColumns saldırganın şifreli metne InvMixColumns uygulayıp son tur anahtarını eşdeğer bir anahtarla değiştirmesiyle bedavaya soyulabilir — güvenliğe hiçbir katkısı yoktur. Atılması ise şifreleme ile şifre çözme yapısını birbirine simetrik hâle getirir.
Bu dört işlem, Shannon'un iki prensibinin doğrudan mühendislik karşılığıdır:

- **SubBytes → karışıklık (confusion)**: doğrusal olmayan bayt değişimi.

- **ShiftRows + MixColumns → yayılma (diffusion)**: baytları önce satırlar, sonra sütunlar boyunca tüm bloğa dağıtır.
- **AddRoundKey → anahtarın karıştırılması**: anahtar malzemesinin veriye işlendiği tek adım.

Şimdi her adımı yakından inceleyelim.

19.5 SubBytes — S-Kutusu

SubBytes, durumun 16 baytının her birini sabit bir tabloya — **S-kutusuna (S-box)** — göre değiştirir. DES'in sekiz küçük S-kutusunun aksine AES'in tek ve 256 girdili bir S-kutusu vardır; asıl fark ise tasarım felsefesindedir: AES'in S-kutusu keyfî bir tablo **değildir**, temiz bir matematiksel formülden üretilir.

Tanım 19.2 (AES S-Kutusu). Bir x baytının S-kutusu değeri iki adımda hesaplanır:

1. **Cisim tersi:** x'in $\mathrm{GF}(2^8)$ içindeki çarpımsal tersi x^{-1} alınır (terslerin varlığını biliyoruz); tersi olmayan tek eleman için $\{00\} \mapsto \{00\}$ kabul edilir.
2. **Affine dönüşüm:** Çıkan baytın bitlerine sabit bir doğrusal karıştırma uygulanır ve üzerine $\{63\}$ sabiti XOR'lanır. Bit düzeyinde kural şudur ($c = \{63\} = 01100011$ olmak üzere):

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

Tablonun 256 girdisini buraya dökmenin bir anlamı yok; önemli olan üretim reçetesidir. Bir girdiyi baştan sona kendimiz hesaplayalım.

Örnek 19.1. $S(\{53\})$ değerini, yani $\{53\}$ baytının S-kutusu karşılığını hesaplayınız.

Çözüm.
Adım 1 — cisim tersi. Sonlu cisimler bölümündeki yöntemlerle (deneme ya da genişletilmiş Öklid algoritması) $\{53\}$'ün tersi bulunur:

$$\{53\}^{-1} = \{CA\}$$

Sağlaması: $\{53\} \cdot \{CA\} = \{01\}$.
Adım 2 — affine dönüşüm. $\{CA\} = 11001010$ baytının bitleri (sağdan sola) $b_0 = 0$, $b_1 = 1$, $b_2 = 0$, $b_3 = 1$, $b_4 = 0$, $b_5 = 0$, $b_6 = 1$, $b_7 = 1$'dir. Formülü ilk iki bite uygulayalım ($c_0 = 1$, $c_1 = 1$):

$$b'_0 = b_0 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 \oplus c_0 = 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$
$$b'_1 = b_1 \oplus b_5 \oplus b_6 \oplus b_7 \oplus b_0 \oplus c_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$$

Aynı hesap sekiz bitin tamamı için tekrarlandığında sonuç 11101101 çıkar:

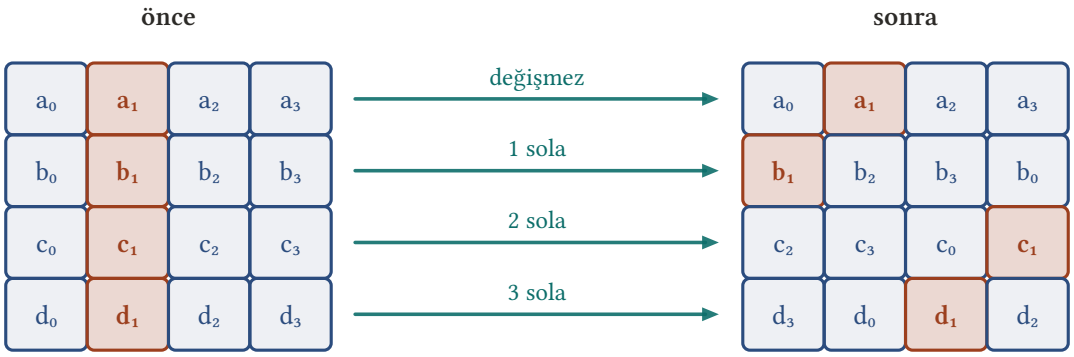
$$S(\{53\}) = \{ED\}$$

Bu değer FIPS-197'deki resmî S-kutusu tablosuyla birebir örtüşür.
$\boxtimes$

Peki neden bu kadar zahmetli bir reçete? Çünkü $x \mapsto x^{-1}$ dönüşümü $\mathrm{GF}(2^8)$ üzerinde bilinen **en doğrusal olmayan** fonksiyonlardan biridir; bu da diferansiyel ve doğrusal kriptanaliz gibi DES döneminde geliştirilen güçlü saldırı ailelerine karşı en yüksek direnci sağlar. Affine katman ise cebirsel yapıyı gizler ve $S(x) = x$ gibi sabit noktaları engeller. Üstelik hem ters alma hem affine dönüşüm tersinir olduğundan bileşke de tersinirdir: şifre çözmede kullanılacak **InvSubBytes** tablosu her zaman vardır.

19.6 ShiftRows

İkinci adım göz açıp kapayıncaya kadar biter: durumun her satırı, **satır numarası kadar** sola dairesel kaydırılır. Sıfırıncı satır yerinde kalır; birinci satır 1, ikinci satır 2, üçüncü satır 3 bayt döner.



Şekil 19.3: ShiftRows her satırı kendi numarası kadar sola döndürür: 0. satır sabit kalır, 1. satır bir, 2. satır iki, 3. satır üç bayt kayar. Solda aynı sütunda duran dört bayt (koyu hücreler) sağda dört farklı sütuna dağılmıştır.

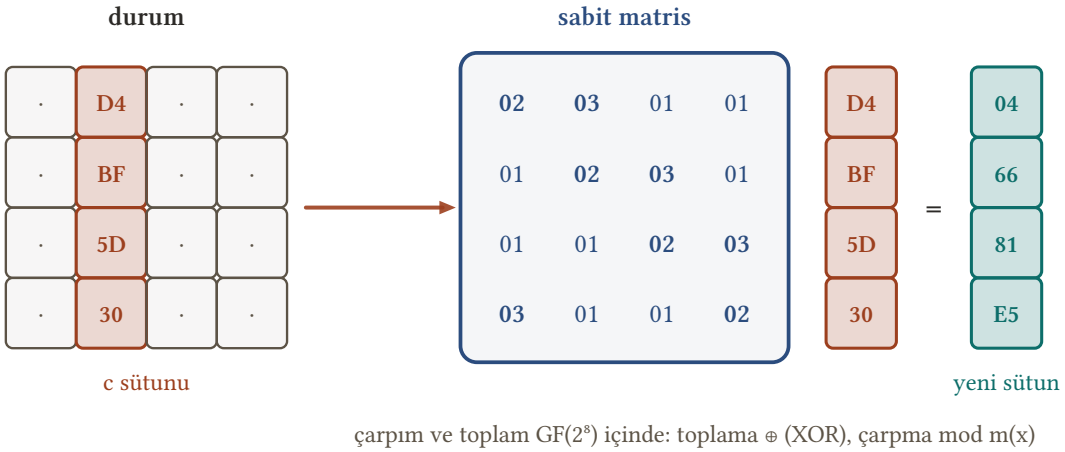
Bu masum görünen adımın işlevi kritiktir: kaydırmadan önce aynı sütunda duran dört bayt, kaydırmadan sonra **dört farklı sütuna** dağılır. Böylece hemen ardından gelen MixColumns, tek bir sütunun etkisini bloğun tamamına taşıyabilir.

19.7 MixColumns

Yayılmanın ağır topu MixColumns'tur. Bu adım durumun her sütununu 4 baytlık bir vektör olarak alır ve onu $\mathrm{GF}(2^8)$ üzerinde sabit bir matrisle çarpar:

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} \{02\} & \{03\} & \{01\} & \{01\} \\ \{01\} & \{02\} & \{03\} & \{01\} \\ \{01\} & \{01\} & \{02\} & \{03\} \\ \{03\} & \{01\} & \{01\} & \{02\} \end{pmatrix} \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix}$$

Buradaki toplama XOR, çarpma ise sonlu cisimler bölümünde öğrendiğimiz $m(x)$ modülünde çarpmadır. Matris girdilerinin $\{01\}$, $\{02\}$ ve $\{03\}$ gibi küçük değerler seçilmesi tesadüf değildir: $\{02\}$ ile çarpmak tek bir `xtime` adımı, $\{03\}$ ile çarpmak `xtime` artı bir XOR'dur — donanımda ve yazılımda son derece ucuzdur. Sonuçta yeni sütunun **her baytı, eski sütunun dört baytının birden** fonksiyonudur.



Şekil 19.4: MixColumns durumun her sütununu ayrı ayrı ele alır: sütun, sabit döngüsel matrisle GF(2^8) üzerinde çarpılır ve yerine yeni sütun yazılır. Örnekteki (D4, BF, 5D, 30) sütunu (04, 66, 81, E5) sütununa dönüşür.

Örnek 19.2. FIPS-197'nin örnek şifrelemesinde, birinci döngüde SubBytes ve ShiftRows adımlarından çıkıp MixColumns'a giren durumun ilk sütunu $(\{D4\}, \{BF\}, \{5D\}, \{30\})$'dur. MixColumns'un bu sütunu $(\{04\}, \{66\}, \{81\}, \{E5\})$ sütununa dönüştürdüğünü, ilk baytı elle hesaplayarak doğrulayınız.

Çözüm.
Matrisin ilk satırına göre yeni sütunun ilk baytı şudur:

$$s'_0 = \{02\} \cdot \{D4\} \oplus \{03\} \cdot \{BF\} \oplus \{5D\} \oplus \{30\}$$

Çarpımları sonlu cisimler bölümündeki `xtime` hilesiyle hesaplayalım.
$\{02\} \cdot \{D4\}$: $\{D4\} = 11010100$'ü bir bit sola kaydırırız: $10101000 = \{A8\}$. Taşan bit 1 olduğundan $\{1B\}$ ile XOR'larız:

$$\{A8\} \oplus \{1B\} = \{B3\}$$

$\{03\} \cdot \{BF\}$: $\{03\} = \{02\} \oplus \{01\}$ olduğundan bu çarpım xtime $(\{BF\}) \oplus \{BF\}$'dir. $\{BF\} = 10111111$ kaydırılınca $01111110 = \{7E\}$; taşma yine var, $\{7E\} \oplus \{1B\} = \{65\}$. Son olarak:

$$\{65\} \oplus \{BF\} = \{DA\}$$

XOR zinciri: Dört terimi birleştirelim:

$$s'_0 = \{B3\} \oplus \{DA\} \oplus \{5D\} \oplus \{30\} = \{69\} \oplus \{5D\} \oplus \{30\} = \{34\} \oplus \{30\} = \{04\} \checkmark$$

Kalan üç bayt da matrisin diğer satırlarıyla aynı yöntemle hesaplanır ve sırasıyla $\{66\}$, $\{81\}$, $\{E5\}$ bulunur.
$\boxtimes$

İki döngüde tam yayılma

ShiftRows ile MixColumns'un ortak eseri çarpıcıdır: girişteki tek bir baytın etkisi, bir döngüde kendi sütununun tamamına, **iki döngüde ise 16 baytın hepsine** ulaşır. Shannon bölümünde tanımladığımız çığ etkisinin AES'te bu kadar hızlı oturmasının nedeni budur.

19.8 AddRoundKey

Dördüncü adım en basitidir: durumun 128 biti, o döngünün 128 bitlik **tur anahtarıyla (round key)** bit bit XOR'lanır. Sadeliğine aldanmayın — bu, algoritmanın **anahtara bağlı olan tek adımıdır**; SubBytes, ShiftRows ve MixColumns sabit ve herkesçe bilinen dönüşümlerdir. AddRoundKey olmasaydı AES, anahtarsız ve herkesin tersine çevirebileceği süslü bir karıştırmadan ibaret olurdu.
Bu adımın tersi kendisidir: XOR bölümünden bildiğimiz $A \oplus A = 0$ özelliği sayesinde aynı tur anahtarıyla bir kez daha XOR'lamak durumu eski hâline döndürür.

19.9 Anahtar Genişletme (Key Expansion)

Toplam $N_r + 1$ tur anahtarı, yani $4(N_r + 1)$ kelime gerekir — AES-128 için 44 kelime. Tek bir gizli anahtardan bu malzemeyi üreten mekanizmaya **anahtar genişletme (key expansion)** denir; DES'teki key schedule'ın AES karşılığıdır.
Gizli anahtar önce 32 bitlik kelimeler hâlinde okunur: $w_0, w_1, \ldots, w_{N_k-1}$. Sonraki her kelime, $i \geq N_k$ için şu kuralla üretilir:

$$w_i = w_{i-N_k} \oplus \begin{cases} \mathrm{SubWord}\big(\mathrm{RotWord}\ (w_{i-1})\ \big) \oplus \mathrm{Rcon}_{i/N_k}, & i \equiv 0 \pmod{N_k} \\ w_{i-1}, & \text{aksi hâlde} \end{cases}$$

Buradaki üç yardımcı işlem tek satırda özetlenebilir:

- **RotWord:** kelimenin 4 baytını bir bayt sola döndürür — $(a_0, a_1, a_2, a_3) \mapsto (a_1, a_2, a_3, a_0)$.
- **SubWord:** kelimenin 4 baytının her birine S-kutusunu uygular.
- **Rcon:** ilk baytı x'in $\mathrm{GF}(2^8)$ içindeki kuvvetleri ($\{01\}, \{02\}, \{04\}, \ldots, \{36\}$), kalan üç baytı sıfır olan **döngü sabitidir**.

(AES-256'da bir ek incelik vardır: $i \equiv 4 \pmod 8$ olduğunda da kelimeye yalnızca SubWord uygulanır.)
Amaç iki katmanlıdır: RotWord ile SubWord her tur anahtarının bir öncekinden **bambaşka görünmesini** sağlar; Rcon sabitleri ise tüm döngülerin aynı işleme tabi tutulmasından doğabilecek **simetrileri kırar** — aynı kelimenin farklı döngülerde aynı sonucu üretmesi engellenir.

19.10 Şifre Çözme

Feistel yapısında şifre çözme neredeyse bedavaydı: DES'te aynı devre, alt anahtarları ters sırada vermekle şifre çözüyordu. SPN'de bu lüks yoktur; bunun yerine her adım **tek tek tersine çevrilir**. Neyse ki tüm adımlar bu iş için tasarlanmıştır:

- **InvSubBytes:** S-kutusunun ters tablosu,
- **InvShiftRows:** satırları aynı miktarlarda bu kez **sağa** döndürme,
- **InvMixColumns:** sabit matrisin $\mathrm{GF}(2^8)$ üzerindeki ters matrisiyle çarpma,
- **AddRoundKey:** zaten kendi tersi.

Şifre çözme, bu ters işlemleri şifrelemenin **ters sırasında** ve tur anahtarlarını sondan başa vererek uygular. Son döngüden MixColumns'un atılmış olması burada meyvesini verir: şifreleme ile şifre çözme akışları birbirinin aynadaki görüntüsü olur.

19.11 Güvenlik Durumu

Yirmi yılı aşkın acımasız bir incelemenin ardından tablo şudur: **tam döngülü AES'e karşı bilinen pratik bir saldırı yoktur.** Akademik cephedeki en iyi sonuç, 2011'de yayımlanan **biclique saldırısıdır**: AES-128'in anahtarını yaklaşık $2^{126{,}1}$ işlemde bulur — kaba kuvvetten yalnızca dört kat kadar hızlı, yani pratikte hiçbir anlamı olmayan marjinal bir iyileştirme. AES-256'ya karşı bilinen **ilişkili-anahtar (related-key)** saldırıları da yalnızca saldırganın birbirine özel biçimde bağlı anahtarlarla şifreleme yaptırabildiği yapay senaryolarda çalışır; doğru kullanılan protokollerde karşılığı yoktur.
Hızın da güvenliğin bir bileşeni olduğunu unutmayalım: modern işlemcilerdeki **AES-NI** komut seti, her döngüyü donanımda tek komutla çalıştırır. Bu sayede AES hem telefonunuzdaki disk şifrelemesinde hem de saniyede milyonlarca bağlantı kuran sunucularda hissedilmeyecek kadar hızlıdır — güvenli olduğu için değil, güvenli **ve** ucuz olduğu için her yerdedir.

Kuantum ufkuna gelince: Grover algoritması kaba kuvvet aramasını karesel hızlandırır, bu da etkin anahtar uzunluğunu kabaca yarıya indirmek demektir. AES-128'in etkin gücü kuantum saldırgana karşı 64 bit düzeyine iner ve rahatsız edici bir sınıra yaklaşır; **AES-256 ise 128 bitlik kuantum-sonrası güvenlik payıyla** rahat koltuğunda oturmaya devam eder. Simetrik kriptografinin kuantum çağında da ayakta kalacak olması, bir sonraki bölümlerde göreceğimiz açık anahtarlı sistemler için söylenemeyecektir.

20. Asimetrik Şifreleme ve Açık Anahtar Kriptografisine Giriş

Klasik kriptografi sistemlerinde (Sezar, afin, Hill vb.) şifreleme ve deşifreleme işlemleri için aynı anahtar ya da birbirine kolayca dönüştürülebilen simetrik anahtarlar kullanılır. Bu durum modern ağ yapılarında ciddi bir **anahtar dağıtım problemi (key distribution problem)** doğurur: güvenli iletişim kurmak isteyen iki tarafın, öncelikle güvenli olmayan bir kanal üzerinden gizli anahtarı birbirine ulaştırması gerekir.
Asimetrik şifreleme —yaygın adıyla **açık anahtar kriptografisi**— bu temel tıkanıklığı **çift anahtar** konseptiyle çözerek kriptografide tam anlamıyla bir paradigma değişimi yaratmıştır. Bu sistemde her kullanıcının matematiksel olarak birbirine bağlı olan, ancak birinden diğerinin hesaplanması bilgisayarsal olarak imkânsız olan iki farklı anahtarı bulunur: **açık anahtar (public key)** ve **gizli anahtar (private key)**.

> **i** Tarihsel not
>
> Asimetrik şifreleme fikri teorik olarak ilk kez 1976 yılında **Whitfield Diffie** ve **Martin Hellman** tarafından ortaya atılmış; pratik ve olgun ilk matematiksel model ise 1977 yılında **Ron Rivest, Adi Shamir ve Leonard Adleman** tarafından kendi soyadlarının baş harflerini taşıyan **RSA algoritması** ile kurulmuştur.

20.1 Kriptosistemin Formal Tanımı

Asimetrik yaklaşım, klasik kriptosistem tanımındaki tekil anahtar uzayını ikiye ayırır ve fonksiyonel bağımlılıkları yeniden şekillendirir.

Tanım 20.1 (Asimetrik Kriptosistem). Bir asimetrik kriptosistem, aşağıdaki koşulları sağlayan bir $(\mathcal{P}, \mathcal{C}, \mathcal{K}_{pub}, \mathcal{K}_{priv}, \mathcal{E}, \mathcal{D})$ altılısıdır:

- $\mathcal{P}$ **(Plaintext):** Açık metinlerin oluşturduğu sonlu küme.
- $\mathcal{C}$ **(Ciphertext):** Şifreli metinlerin oluşturduğu sonlu küme.
- $\mathcal{K}_{pub}$ **(Public Key Space):** Şifreleme amacıyla kullanılan ve herkese açık olarak ilan edilen açık anahtarların (pk) sonlu kümesi.
- $\mathcal{K}_{priv}$ **(Private Key Space):** Yalnızca şifreyi çözecek alıcı tarafından gizli tutulan gizli anahtarların (sk) sonlu kümesi.
- $\mathcal{E}$ **(Encryption):** Şifreleme fonksiyonları kümesi.
- $\mathcal{D}$ **(Decryption):** Deşifreleme fonksiyonları kümesi.

Sistemden seçilen her (pk, sk) anahtar çifti için, $\mathcal{E}$ kümesinden bir e şifreleme fonksiyonu ve $\mathcal{D}$ kümesinden bir d deşifreleme fonksiyonu tanımlanır:

$$e_{pk} : \mathcal{P} \to \mathcal{C}$$

$$d_{sk} : \mathcal{C} \to \mathcal{P}$$

Sistemin matematiksel olarak tutarlı ve işlevsel olabilmesi için her $x \in \mathcal{P}$ açık metni üzerinde şu **tersinelenebilirlik şartı** sağlanmalıdır:

$$d_{sk}\left(e_{pk}(x) \right) = x, \quad \forall x \in \mathcal{P}$$

20.2 Asimetrik Sistemlerin Dayandığı Fikir

Açık anahtar kriptografisinin tamamı, **tek yönlü kapılı fonksiyonlar (trapdoor one-way functions)** fikrine dayanır: bir yöne hesaplanması kolay, ters yöne hesaplanması pratikte imkânsız olan; ancak gizli bir bilgiye ("kapı") sahip olan kişi için tersinin de kolay hesaplanabildiği fonksiyonlar.
Bu derste inceleyeceğimiz üç sistem, birbirinden farklı iki zor probleme yaslanır:

- **Diffie-Hellman** ve **ElGamal** — ayrık logaritma probleminin zorluğuna,
- **RSA** — büyük tam sayıları asal çarpanlarına ayırmanın zorluğuna.

Bu problemlerin "zor" olduğu bugüne kadar ispatlanmış değildir; yalnızca uzun yıllardır kimse yeterince hızlı bir algoritma bulamamıştır. Modern iletişim güvenliğinin tamamı bu varsayımın üzerinde durmaktadır.

21. Diffie-Hellman Anahtar Değişimi

Asimetrik kriptografinin doğuşunu simgeleyen **Diffie-Hellman anahtar değişimi**, 1976 yılında Whitfield Diffie ve Martin Hellman tarafından yayımlanmıştır. Bu protokol, iki tarafın (geleneksel olarak Alice ve Bob) güvenli olmayan ve herkes tarafından dinlenebilen bir kanal üzerinden, önceden hiçbir gizli bilgi paylaşmadan **ortak bir gizli anahtar (shared secret)** oluşturmasını sağlar.
Diffie-Hellman bir şifreleme algoritması değil, bir **anahtar değişim protokolüdür**. Burada üretilen ortak gizli anahtar, sonraki iletişimde verileri çok daha hızlı şifreleyebilen simetrik algoritmalarda (örneğin AES) kullanılır. Protokolün güvenliği, **ayrık logaritma probleminin (Discrete Logarithm Problem — DLP)** bilgisayarsal zorluğuna dayanır.

> **i** Temel mantık: renk karışımı analojisi
>
> Diffie-Hellman'ın işleyişi genellikle bir renk karışımı benzetmesiyle açıklanır:
>
> 1. Alice ve Bob herkesin görebileceği **ortak bir başlangıç rengi** (örneğin sarı) seçer.
> 2. Her iki taraf da kimseye göstermediği **gizli birer renk** seçer (Alice mavi, Bob kırmızı).
> 3. Taraflar ortak renk ile kendi gizli renklerini karıştırıp elde ettikleri karışımı birbirlerine gönderir (Alice yeşil, Bob turuncu gönderir). Kanalı dinleyen bir saldırgan, bu karışımlardan orijinal gizli renkleri ayrıştıramaz.
> 4. Alice, Bob'dan gelen turuncuya kendi gizli rengi maviyi ekler; Bob ise Alice'ten gelen yeşile kendi gizli rengi kırmızıyı ekler.
> 5. Sonuçta her iki taraf da **tamamen aynı nihai rengi** elde eder.
>
> Modüler üs alma işlemi, bu "karıştırması kolay, ayrıştırması zor" özelliği matematiksel olarak birebir taklit eder.

21.1 Protokolün Adımları

Diffie-Hellman'ın formal kurgusu modüler aritmetik ve grup teorisi üzerine kuruludur.
1. Kamusal parametrelerin seçimi. Taraflar, herkesin erişebileceği şu iki parametre üzerinde anlaşır:

- Büyük bir asal sayı p,
- $\mathbb{Z}_p^*$ çarpımsal grubunun bir üreteci g (yani g'nin mod p'deki kuvvetleri gruptaki tüm elemanları üretmelidir).

2. Gizli ve açık değerlerin üretilmesi. Taraflar birbirinden bağımsız olarak birer gizli sayı seçer ve buna karşılık gelen açık değeri hesaplayıp karşı tarafa gönderir:

- **Alice**, $1 < a < p-1$ aralığında rastgele bir **gizli** tam sayı a seçer ve açık değerini hesaplayıp Bob'a gönderir:

$$A \equiv g^a \pmod{p}$$

- **Bob**, $1 < b < p-1$ aralığında rastgele bir **gizli** tam sayı b seçer ve açık değerini hesaplayıp Alice'e gönderir:

$$B \equiv g^b \pmod{p}$$

3. Ortak gizli anahtarın hesaplanması. Her iki taraf, karşıdan aldığı açık değeri kendi gizli sayısıyla üstel olarak işler:

- Alice, Bob'dan aldığı B değerini kendi gizli anahtarıyla işler:

$$K_A \equiv B^a \pmod{p}$$

- Bob, Alice'ten aldığı A değerini kendi gizli anahtarıyla işler:

$$K_B \equiv A^b \pmod{p}$$

21.2 Matematiksel Yapı ve Doğruluk İspatı

Tanım 21.1 (Diffie-Hellman Ortak Gizli Anahtarı). Alice ve Bob'un protokol sonunda bağımsız olarak hesapladıkları K_A ve K_B değerleri matematiksel olarak birbirine eşittir. Bu ortak değere **Diffie-Hellman ortak gizli anahtarı (shared secret)** denir:

$$K = K_A = K_B$$

Teorem 21.1 (Protokolün Matematiksel Tutarlılığı). *Güvenli olmayan kanal üzerinden iletilen $A \equiv g^a \pmod{p}$ ve $B \equiv g^b \pmod{p}$ açık değerleri kullanılarak hesaplanan $B^a \pmod{p}$ ve $A^b \pmod{p}$ değerleri her zaman aynı $g^{ab} \pmod{p}$ sonucunu verir.*

İspat.
Alice'in hesaplama adımını ele alalım; Bob'dan gelen B değerinin a. kuvvetini mod p altında hesaplamaktadır:

$$K_A \equiv B^a \pmod{p}$$

Bob'un açık değer üretiminden $B \equiv g^b \pmod{p}$ olduğunu biliyoruz. Yerine koyarsak:

$$K_A \equiv (g^b)^a \pmod{p}$$

Üs kurallarına göre $(g^b)^a = g^{ba} = g^{ab}$ olduğundan:

$$K_A \equiv g^{ab} \pmod{p}$$

Şimdi Bob'un adımını inceleyelim; Alice'ten gelen A değerinin b. kuvvetini hesaplamaktadır:

$$K_B \equiv A^b \pmod{p}$$

Alice'in açık değer tanımından $A \equiv g^a \pmod{p}$ olduğunu biliyoruz; yerine koyduğumuzda:

$$K_B \equiv (g^a)^b \equiv g^{ab} \pmod{p}$$

Görüldüğü üzere her iki işlem de aynı nihai değere ulaşır:

$$K_A \equiv K_B \equiv g^{ab} \pmod{p}$$

Böylece tarafların gizli anahtarlarını (a ve b) kanala hiç vermeden, yalnızca açık bileşenler üzerinden aynı gizli K değerinde buluşabilecekleri kanıtlanmış olur.
$\boxtimes$

⚠ Kritik güvenlik uyarısı: ortadaki adam saldırısı

Diffie-Hellman protokolü, pasif dinleyicilere karşı ayrık logaritma probleminin zorluğu sayesinde güvenlidir. Ancak asıl zayıflığı, **kimlik doğrulama (authentication)** mekanizmasının bulunmamasıdır.
Alice ve Bob karşı tarafın kimliğini doğrulamadığı için, araya giren aktif bir saldırgan (Eve) Alice'e kendini Bob, Bob'a kendini Alice olarak tanıtabilir. Alice ile ayrı, Bob ile ayrı birer ortak anahtar kurarak tüm iletişimi okuyabilir ve değiştirebilir.
Bu tehlikeyi önlemek için modern sistemlerde Diffie-Hellman adımları **dijital imzalar** veya sertifikalarla birlikte kullanılır.

21.3 Çözümlü Uygulama

Örnek 21.1. Bir Diffie-Hellman anahtar değişiminde ortak parametreler $p = 13$ ve üreteç $g = 2$ olarak belirlenmiştir. Alice gizli değerini $a = 4$, Bob ise $b = 3$ olarak seçmiştir. Tarafların üreteceği kamusal açık değerleri ve süreç sonunda elde edecekleri ortak gizli anahtarı adım adım hesaplayınız.

Çözüm.
1. Kamusal açık değerlerin üretilmesi.
Alice'in açık değeri:

$$A \equiv g^a \pmod{p} \Longrightarrow A \equiv 2^4 = 16 \pmod{13}$$

$$16 - 13 = 3 \Longrightarrow A = 3$$

Alice, $A = 3$ değerini kanal üzerinden Bob'a gönderir.
Bob'un açık değeri:

$$B \equiv g^b \pmod{p} \Longrightarrow B \equiv 2^3 = 8 \pmod{13} \Longrightarrow B = 8$$

Bob, $B = 8$ değerini kanal üzerinden Alice'e gönderir.
2. Ortak gizli anahtarın hesaplanması.
Alice'in hesabı: Bob'dan gelen $B = 8$ değerini kendi gizli sayısı $a = 4$ ile işler:

$$K_A \equiv 8^4 \pmod{13}$$

Hesabı kolaylaştırmak için $8^2 = 64 \equiv 12 \equiv -1 \pmod{13}$ denkliğini kullanalım:

$$8^4 = (8^2)^2 \equiv (-1)^2 = 1 \Longrightarrow K_A = 1$$

Bob'un hesabı: Alice'ten gelen $A = 3$ değerini kendi gizli sayısı $b = 3$ ile işler:

$$K_B \equiv 3^3 = 27 \pmod{13}$$

$$27 - 26 = 1 \Longrightarrow K_B = 1$$

Sonuç: Alice ve Bob gizli sayılarını hiç paylaşmadan $K = 1$ ortak gizli anahtarında başarıyla buluşmuşlardır.
$\boxtimes$

Neden ortak anahtar 1 çıktı?

Yukarıdaki örnekte $K = g^{ab} = 2^{12} \pmod{13}$ hesaplanmaktadır. 2 sayısı mod 13'te bir **primitif kök** olduğundan mertebesi $\phi(13) = 12$'dir; dolayısıyla $2^{12} \equiv 1 \pmod{13}$ olur.
Yani $K = 1$ sonucu bir hata değil, seçilen küçük sayıların ($ab = 12$ tam olarak mertebeye eşit) yarattığı bir tesadüftür. Gerçek uygulamalarda p en az 2048 bitlik bir asal seçildiği için böyle bir çakışma pratikte imkânsızdır.

22. RSA (Rivest-Shamir-Adleman) Algoritması

Asimetrik şifrelemenin teorik altyapısını inceledikten sonra, bu paradigmanın dünyadaki ilk ve en yaygın uygulaması olan **RSA** algoritmasını ele alacağız.
RSA'nın güvenliği, sayılar teorisinin en temel zor problemlerinden biri olan **büyük sayıların asal çarpanlarına ayrılması (integer factorization problem)** ilkesine dayanır: bilgisayarlar için iki büyük asal sayıyı çarpmak (ileri yön) saliseler alırken, çarpım sonucundan hareketle orijinal asal çarpanları bulmak (geri yön) günümüz işlemcileriyle astronomik süreler alabilir.

> **i** Gerekli matematiksel ön bilgiler
>
> RSA adımlarını tam olarak kavrayabilmek için şu iki kavramın hatırlanması hayati önem taşır:
>
> 1. **Euler totient fonksiyonu $\phi(n)$:** Bir n pozitif tam sayısı için, n ile aralarında asal olan pozitif tam sayıların adedidir. p ve q iki farklı asal sayı ise, fonksiyonun çarpımsallık özelliği gereği $\phi(p \cdot q) = (p-1)(q-1)$ olur.
> 2. **Modüler çarpımsal ters:** $a \cdot d \equiv 1 \pmod{\phi(n)}$ denkliğini sağlayan d değeridir. Bu değer **genişletilmiş Öklid algoritması** kullanılarak polinom zamanda hızlıca hesaplanır.

22.1 Anahtar Üretim Süreci (Key Generation)

Şifreli mesajı alacak taraf, kendi anahtar çiftini oluşturmak için şu adımları uygular:

1. Birbirinden bağımsız, rastgele ve çok büyük iki asal sayı p ve q seçilir.
2. Bu iki asalın çarpımı olan **kamusal modül** hesaplanır:
$$n = p \cdot q$$
3. Bu modüle ait Euler totient değeri hesaplanır:
$$\phi(n) = (p-1)(q-1)$$
4. $1 < a < \phi(n)$ aralığında, $\phi(n)$ ile aralarında asal olan bir **açık üs (encryption exponent)** seçilir:
$$\gcd\big(\ a, \phi(n)\ \big) = 1$$
5. Seçilen a değerinin mod $\phi(n)$ altındaki çarpımsal tersi olan **gizli üs (decryption exponent)** hesaplanır:
$$a \cdot d \equiv 1 \pmod{\phi(n)}$$

Sonuçta:

- **Açık anahtar (pk):** (a, n) çiftidir; herkesin erişebileceği şekilde ilan edilir.
- **Gizli anahtar (sk):** (d, n) çiftidir; alıcı tarafından kesinlikle gizli tutulur.

p, q ve $\phi(n)$ değerleri de süreç sonunda güvenli biçimde imha edilmelidir — çünkü bunlardan herhangi biri gizli anahtarı doğrudan ele verir.

Alternatif: Carmichael totient fonksiyonu

Teorik anlatımlarda kolay anlaşılması açısından genellikle Euler totient fonksiyonu tercih edilir. Ancak gerçek uygulamalarda ve modern **PKCS#1** standartlarında anahtar üretimi sıklıkla **Carmichael totient fonksiyonu** ile yapılır:

$$\lambda(n) = \text{lcm } (p-1, q-1)$$

$\lambda(n)$ değeri $\phi(n)$'e eşit veya ondan küçük bir çalışma alanı sunduğu için, modüler tersi alındığında bilgisayarsal olarak daha küçük ve dolayısıyla daha hızlı işlenen bir gizli anahtar üretilmesini sağlar. Algoritmanın şifreleme, deşifreleme ve genel matematiksel mantığı bu değişiklikten etkilenmez.

22.2 Fonksiyonların Tanımı ve Doğruluk İspatı

Tanım 22.1 (RSA Şifreleme ve Deşifreleme Fonksiyonları). Gönderici, iletmek istediği açık metni $0 \leq m < n$ şartını sağlayan bir $m \in \mathbb{Z}_n$ sayısına dönüştürür.
Şifreleme fonksiyonu, alıcının açık anahtarı $pk = (a, n)$ ile şifreli metni üretir:

$$c \equiv e_{pk}(m) \equiv m^a \pmod{n}$$

Alıcı, güvenli olmayan kanaldan gelen c metnini **deşifreleme fonksiyonu** ve gizli anahtarı $sk = (d, n)$ ile çözer:

$$m \equiv d_{sk}(c) \equiv c^d \pmod{n}$$

Teorem 22.1 (RSA Algoritmasının Doğruluğu). *Her $m \in \mathbb{Z}_n$ açık metni için, şifreleme ve ardından deşifreleme işlemleri uygulandığında orijinal mesaj hatasız biçimde elde edilir:*

$$(m^a)^d \equiv m^{ad} \equiv m \pmod{n}$$

İspat.
Anahtar üretim adımından $a \cdot d \equiv 1 \pmod{\phi(n)}$ şartının geçerli olduğunu biliyoruz. Modüler denklik tanımı gereği bu, bir $k \in \mathbb{Z}$ tam sayısı için şu anlama gelir:

$$ad = k \cdot \phi(n) + 1$$

Bu durumda deşifreleme çıktısını üs kurallarıyla açalım:

$$m^{ad} = m^{k\phi(n)+1} = m \cdot \left(m^{\phi(n)} \right)^k$$

*Bu ifadenin mod n altındaki denkliğini ispatlamak için **Çin kalan teoremi** gereği, ifadenin hem mod p hem de mod q altında m'ye denk olduğunu göstermek yeterlidir. İspatı mod p için yapalım; mod q için tamamen simetriktir.*
***Durum 1 — $\gcd(m, p) = 1$ ise.** Fermat'nın küçük teoremi uyarınca $m^{p-1} \equiv 1 \pmod{p}$ olur. $\phi(n) = (p-1)(q-1)$ eşitliğini yerine koyarsak:*

$$m^{ad} \equiv m \cdot \left(m^{(p-1)(q-1)} \right)^k \equiv m \cdot \left(\left(m^{p-1} \right)^{q-1} \right)^k \equiv m \cdot \left(1^{q-1} \right)^k \equiv m \pmod{p}$$

***Durum 2 — $\gcd(m, p) \neq 1$ ise.** p asal olduğundan m sayısı p'nin tam katı olmak zorundadır, yani $m \equiv 0 \pmod{p}$. Sıfırın her pozitif kuvveti sıfır olacağından:*

$$m^{ad} \equiv 0^{ad} \equiv 0 \equiv m \pmod{p}$$

Her iki durumda da $m^{ad} \equiv m \pmod{p}$ elde edilir. Aynı adımlar mod q için uygulandığında $m^{ad} \equiv m \pmod{q}$ bulunur.
p ve q farklı asallar olduğundan aralarında asaldır; $(m^{ad} - m)$ farkı hem p'ye hem q'ya bölünüyorsa çarpımlarına da bölünmek zorundadır:

$$m^{ad} \equiv m \pmod{p \cdot q} \implies m^{ad} \equiv m \pmod{n}$$

Böylece deşifreleme fonksiyonunun şifreli metni her koşulda orijinal açık metne dönüştürdüğü kanıtlanmış olur.
⊠

22.3 Çözümlü Uygulama

Örnek 22.1. Bir RSA kurgusunda başlangıç asalları $p = 7$ ve $q = 11$ olarak seçilmiştir. Açık üs değeri $a = 7$ olduğuna göre anahtar çiftlerini hesaplayınız; ardından $m = 9$ açık metnini şifreleyip deşifre ediniz.

Çözüm.
1. Anahtar üretimi.

- Kamusal modül: $n = 7 \cdot 11 = 77$
- Totient değeri: $\phi(n) = (7-1)(11-1) = 6 \cdot 10 = 60$
- Seçilen $a = 7$ değerinin $\phi(n) = 60$ ile aralarında asal olduğu doğrulanır: $\gcd(7, 60) = 1$.
- Gizli üs hesabı, $7d \equiv 1 \pmod{60}$:

$$7 \cdot 43 = 301 = (60 \cdot 5) + 1 \equiv 1 \pmod{60} \Longrightarrow d = 43$$

Buradan anahtarlar: **açık anahtar** $pk = (7, 77)$, **gizli anahtar** $sk = (43, 77)$.
2. Şifreleme süreci. Gönderici $m = 9$ mesajını şifreler:

$$c \equiv m^a \pmod{n} \Longrightarrow c \equiv 9^7 \pmod{77}$$

Hesabı kolaylaştırmak için üssü parçalayalım:

$$\begin{aligned} 9^1 &\equiv 9 \pmod{77} \\ 9^2 &= 81 \equiv 4 \pmod{77} \\ 9^4 &\equiv 4^2 = 16 \pmod{77} \end{aligned}$$

$9^7 = 9^4 \cdot 9^2 \cdot 9^1$ olduğundan:

$$c \equiv 16 \cdot 4 \cdot 9 = 576 \pmod{77}$$

$77 \cdot 7 = 539$ olduğundan $576 - 539 = 37$, yani $c = 37$.
3. Deşifreleme süreci. Alıcı gelen $c = 37$ şifreli metnini gizli anahtarıyla çözer:

$$m \equiv c^d \pmod{n} \Longrightarrow m \equiv 37^{43} \pmod{77}$$

Ardışık kare alma (square-and-multiply) yöntemiyle üsleri indirgeyelim:

$$\begin{aligned} 37^1 &\equiv 37 \pmod{77} \\ 37^2 &= 1369 = (77 \cdot 17) + 60 \equiv 60 \equiv -17 \pmod{77} \\ 37^4 &\equiv (-17)^2 = 289 = (77 \cdot 3) + 58 \equiv 58 \equiv -19 \pmod{77} \\ 37^8 &\equiv (-19)^2 = 361 = (77 \cdot 4) + 53 \equiv 53 \pmod{77} \\ 37^{16} &\equiv 53^2 = 2809 = (77 \cdot 36) + 37 \equiv 37 \pmod{77} \\ 37^{32} &\equiv 37^2 \equiv 60 \pmod{77} \end{aligned}$$

43 sayısının ikilik açılımı $43 = 32 + 8 + 2 + 1$ olduğundan:

$$37^{43} = 37^{32} \cdot 37^8 \cdot 37^2 \cdot 37^1 \equiv 60 \cdot 53 \cdot 60 \cdot 37 \pmod{77}$$

İşlemi kolaylaştırmak için $60 \equiv -17$ denkliğini kullanalım ve terimleri ikişer gruplayalım:

$$(-17) \cdot (-17) = 289 \equiv 58 \equiv -19 \pmod{77}$$

$$53 \cdot 37 = 1961 = (77 \cdot 25) + 36 \equiv 36 \pmod{77}$$

Bu iki sonucu çarpalım:

$$(-19) \cdot 36 = -684$$

-684 sayısının pozitif denkini bulalım ($77 \cdot 9 = 693$):

$$-684 + 693 = 9$$

Sonuç: Deşifreleme tamamlanmış ve orijinal açık metin olan $m = 9$ değerine başarıyla geri dönülmüştür.
$\boxtimes$

23. ElGamal Kriptosistemi ve Ayrık Logaritma Problemi

RSA'nın çarpanlara ayırma zorluğuna dayanan yapısını inceledikten sonra, asimetrik kriptografinin diğer büyük sütunu olan **ElGamal kriptosistemini** ele alacağız. 1985 yılında Taher ElGamal tarafından geliştirilen bu algoritma, gücünü **ayrık logaritma probleminden (Discrete Logarithm Problem — DLP)** alır.
ElGamal, Diffie-Hellman anahtar değişim protokolünün şifreleme ve deşifreleme yapabilecek şekilde genişletilmiş bir modifikasyonudur. Günümüzde yaygın olarak kullanılan eliptik eğri kriptografisinin (ECC) de temel mantıksal şemasını oluşturur.

i Ön bilgi: ayrık logaritma problemi (DLP)

Bir p asal sayısı ve $\mathbb{Z}_p^*$ çarpımsal grubunun bir g **üreteci** verildiğinde,

$$y \equiv g^a \pmod{p}$$

eşitliğini sağlayan a üssünü bulma işlemine **ayrık logaritma problemi** denir.
p yeterince büyük seçilirse (modern standartlarda en az 2048 bit), a ve g değerlerinden y'yi hesaplamak polinom zamanda çok kolayken; y ve g değerlerinden hareketle gizli a'yı bulmak pratikte imkânsızdır.

23.1 Anahtar Üretim Süreci

Mesaj alıcısı, kendi açık ve gizli anahtar çiftini kurgulamak için şu adımları izler:

1. Çok büyük bir p asal sayısı seçilir.
2. $\mathbb{Z}_p^*$ grubuna ait bir g **üreteci** belirlenir; g'nin kuvvetleri mod p'de gruptaki tüm elemanları üretmelidir.
3. $1 < a < p-1$ aralığında rastgele bir **gizli anahtar** a seçilir.
4. Bu gizli anahtara karşılık gelen kamusal değer hesaplanır:

$$y \equiv g^a \pmod{p}$$

Sonuçta:

- **Açık anahtar:** (p, g, y) üçlüsüdür; herkesin erişimine açılır.
- **Gizli anahtar:** Yalnızca alıcıda saklanan a tam sayısıdır.

23.2 Fonksiyonların Tanımı ve Doğruluk İspatı

Tanım 23.1 (ElGamal Şifreleme ve Deşifreleme Fonksiyonları). Gönderici, iletmek istediği açık metni $m \in \mathbb{Z}_p^*$ olacak şekilde sayısal bir mesaja dönüştürür. Burada g, $\mathbb{Z}_p^*$ grubunun kamusal üreteci; $y \equiv g^a \pmod{p}$ ise alıcının açık anahtarıdır.
Şifreleme fonksiyonu, sürece rastgelelik katmak amacıyla her mesaj için $1 \leq k \leq p-2$ aralığında geçici (ephemeral) bir k tam sayısı seçer. Şifreli metin bir sayı çiftinden oluşur:

$$c_1 \equiv g^k \pmod{p}$$
$$c_2 \equiv m \cdot y^k \pmod{p}$$

Alıcı, kendisine ulaşan (c_1, c_2) çiftini gizli anahtarını kullanan **deşifreleme fonksiyonu** ile çözer:

$$d_a(c_1, c_2) \equiv c_2 \cdot \left(c_1^a\right)^{-1} \pmod{p}$$

⚠ Sık karşılaşılan bir karışıklık: k üzerindeki koşul

Bazı kaynaklarda ElGamal şifrelemesi için $\gcd(k, p-1) = 1$ koşulu verilir. Bu koşul aslında **ElGamal imza şemasına** aittir; orada k'nin mod $p-1$'de tersinin alınması gerektiği için zorunludur. **Şifrelemede** ise böyle bir kısıt yoktur: $1 \leq k \leq p-2$ aralığındaki her k değeri çalışır; nitekim aşağıdaki çözümlü örnekte $k = 4$ ve $p - 1 = 10$ olmasına ($\gcd(4, 10) = 2 \neq 1$) rağmen algoritma kusursuz sonuç vermektedir.
Asıl kritik kural şudur: **aynı k değeri iki farklı mesaj için asla yeniden kullanılmamalıdır.** Aksi hâlde $c_2/c_2' = m/m'$ elde edilir ve mesajlardan biri bilindiğinde diğeri de çözülür — bu, one-time pad'deki iki kez kullanılan şerit zafiyetinin ElGamal'deki karşılığıdır.

Teorem 23.1 (ElGamal Algoritmasının Doğruluğu). *Her $m \in \mathbb{Z}_p^*$ açık metni ve rastgele seçilen her k geçici anahtarı için, deşifreleme fonksiyonu orijinal mesajı hatasız biçimde geri döndürür.*

İspat.
Deşifreleme fonksiyonunun tanımından yola çıkarak c_1 ve c_2 bileşenlerini yerlerine koyalım. Şifreleme adımından $c_1 \equiv g^k$ ve $c_2 \equiv m \cdot y^k$ olduğunu; açık anahtar tanımından ise $y \equiv g^a \pmod p$ eşitliğini biliyoruz:

$$\begin{aligned} c_2 \cdot \left(c_1^a \right)^{-1} &\equiv \left(m \cdot y^k \right) \cdot \left((g^k)^a \right)^{-1} \pmod p \\ &\equiv \left(m \cdot (g^a)^k \right) \cdot \left(g^{ka} \right)^{-1} \pmod p \\ &\equiv m \cdot g^{ak} \cdot g^{-ak} \pmod p \\ &\equiv m \cdot g^0 \pmod p \\ &\equiv m \cdot 1 \equiv m \pmod p \end{aligned}$$

*Burada g^{ak} ifadesi, şifreleme sırasında oluşturulan **ortak gizli bilgidir**. Deşifreleme esnasında alıcı bu değeri c_1^a işlemiyle yeniden üretir ve çarpımsal tersini alarak sistemden sadeleştirir. Böylece deşifrelemenin doğruluğu kanıtlanmış olur.*
$\boxtimes$

ℹ Olasılıksal şifreleme (probabilistic encryption)

RSA'da aynı m mesajı aynı açık anahtarla şifrelendiğinde her zaman aynı c şifreli metnini üretir; bu **deterministik** yapı, saldırganın olası mesajları tek tek şifreleyip karşılaştırmasına olanak tanır.
ElGamal'de ise şifrelemeye dâhil olan geçici k parametresi sayesinde, aynı mesaj aynı anahtarla defalarca şifrelense bile **her seferinde tamamen farklı bir (c_1, c_2) çifti** üretilir. Bu özellik ElGamal'i semantik güvenlik açısından çok daha güçlü kılar.
Bunun bedeli ise **veri genişlemesidir (message expansion)**: şifreli metin, açık metnin iki katı uzunluğundadır.

23.3 Çözümlü Uygulama

Örnek 23.1. Bir ElGamal kurgusunda kamusal parametreler $p = 11$ ve üreteç $g = 2$ olarak seçilmiştir. Alıcının gizli anahtarı $a = 3$ olduğuna göre açık anahtarı hesaplayınız. Ardından göndericinin $k = 4$ geçici anahtarını kullanarak şifrelediği $m = 5$ açık metnine ait (c_1, c_2) çiftini bulunuz ve bu şifreli metni deşifre ediniz.

Çözüm.
1. Anahtar üretim aşaması.
Verilenler: $p = 11$, $g = 2$, $a = 3$. Kamusal değer:

$$y \equiv g^a \pmod p \Longrightarrow y \equiv 2^3 = 8 \pmod{11}$$

- **Açık anahtar:** $(p, g, y) = (11, 2, 8)$

- **Gizli anahtar:** $a = 3$

2. Şifreleme aşaması. Gönderici $m = 5$ mesajını ve $k = 4$ değerini kullanır.
c_1 bileşeni:

$$c_1 \equiv g^k \pmod{p} \Longrightarrow c_1 \equiv 2^4 = 16 \equiv 5 \pmod{11}$$

c_2 bileşeni:

$$c_2 \equiv m \cdot y^k \pmod{p} \Longrightarrow c_2 \equiv 5 \cdot 8^4 \pmod{11}$$

8'in kuvvetlerini mod 11 altında indirgeyelim:

$$\begin{aligned} 8^1 &\equiv 8 \pmod{11} \\ 8^2 &= 64 \equiv 9 \equiv -2 \pmod{11} \\ 8^4 &\equiv (-2)^2 = 4 \pmod{11} \end{aligned}$$

O hâlde:

$$c_2 \equiv 5 \cdot 4 = 20 \equiv 9 \pmod{11}$$

Şifreli metin çifti: $(c_1, c_2) = (5, 9)$.
3. Deşifreleme aşaması. Alıcı $(5, 9)$ çiftini alır ve gizli anahtarı $a = 3$ ile çözer:

$$m \equiv c_2 \cdot \left(c_1^a \right)^{-1} \pmod{p}$$

Adım A — ortak gizli bilgiyi hesapla:

$$c_1^a \equiv 5^3 = 125 \pmod{11}$$

$11 \cdot 11 = 121$ olduğundan $125 - 121 = 4$, yani $c_1^a \equiv 4 \pmod{11}$.
Adım B — çarpımsal tersi bul: $4 \cdot x \equiv 1 \pmod{11}$ şartını sağlayan x değerini arıyoruz. $4 \cdot 3 = 12 \equiv 1 \pmod{11}$ olduğundan $4^{-1} \equiv 3 \pmod{11}$.
Adım C — mesajı çöz:

$$m \equiv 9 \cdot 3 = 27 \pmod{11}$$

$11 \cdot 2 = 22$ olduğundan $27 - 22 = 5$, yani $m = 5$.
Sonuç: ElGamal kriptosistemi başarıyla çalışmış ve orijinal açık metin olan $m = 5$ değerine kayıpsız biçimde geri dönülmüştür.
$\boxtimes$

24. Kriptografik Özet (Hash) Fonksiyonlarına Giriş

Asimetrik ve simetrik şifreleme algoritmaları verinin **gizliliğini (confidentiality)** sağlarken; dijital dünyada mesajların yolda değiştirilmediğinden (**bütünlük — integrity**) ve gönderenin gerçekten iddia ettiği kişi olduğundan (**kimlik doğrulama — authentication**) da emin olmamız gerekir. İşte bu noktada modern kriptografinin "İsviçre çakısı" sayılan **kriptografik özet (hash) fonksiyonları** devreye girer.

> **i** Kısa tarihçe
>
> Hash fonksiyonu kavramı ilk olarak 1950'lerde **Hans Peter Luhn** tarafından veritabanlarında metinleri hızlıca aramak ve sınıflandırmak amacıyla ortaya atıldı. 1976'da Diffie ve Hellman'ın asimetrik şifrelemeyi icadıyla birlikte dijital imzalarda kullanılmak üzere kriptografik bir boyut kazandı. Modern kriptografik hash fonksiyonlarının (MD5, SHA ailesi) matematiksel bel kemiğini ise, 1979 ve 1989 yıllarında birbirinden bağımsız çalışan **Ralph Merkle** ve **Ivan Damgård**'ın kurduğu mimari oluşturmuştur.

24.1 Nerede ve Nasıl Kullanılır?

Özet fonksiyonları şifreleme **yapmaz** (geri döndürülemezler); bunun yerine verinin benzersiz bir "dijital parmak izini" çıkarırlar:

- **Parola saklama.** Şifreleriniz veritabanlarında düz metin olarak değil, hash değerleri olarak saklanır. Veritabanı çalınsa bile hash'ten orijinal şifreye dönülemez.
- **Dosya bütünlüğü (checksum).** İndirdiğiniz bir dosyanın eksik veya değiştirilmiş olup olmadığını anlamak için dosyanın hash değeri, yayıncının ilan ettiği değerle karşılaştırılır.
- **Dijital imzalar.** Devasa bir PDF dosyasını RSA ile imzalamak çok uzun sürer; bunun yerine dosyanın kısa hash'i alınır ve yalnızca bu özet imzalanır.
- **Blokzincir.** Bitcoin ve Ethereum gibi ağlarda blokları birbirine kriptografik olarak bağlamak ve madencilik (proof of work) yapmak için yoğun biçimde SHA-256 kullanılır.

24.2 Matematiksel Tanım ve Güvenlik Kriterleri

Tanım 24.1 (Kriptografik Özet (Hash) Fonksiyonu). Bir kriptografik hash fonksiyonu H; rastgele ve sonsuz uzunlukta olabilen herhangi bir $m \in \{0,1\}^*$ mesajını girdi olarak alıp, sabit ve önceden belirlenmiş n uzunluğunda (örneğin 256 bit) bir $h \in \{0,1\}^n$ özet değerine dönüştüren **deterministik** bir fonksiyondur:

$$H : \{0,1\}^* \to \{0,1\}^n, \qquad h = H(m)$$

Bu fonksiyonun kriptografik olarak güvenli kabul edilebilmesi için şu üç **direnç (resistance)** şartını sağlaması zorunludur:

1. **Ön-görüntü direnci (pre-image resistance / tek yönlülük):** Yalnızca h özet değeri biliniyorken, $H(m) = h$ şartını sağlayan orijinal m mesajını bulmak bilgisayarsal olarak imkânsız olmalıdır.
2. **İkinci ön-görüntü direnci (second pre-image resistance):** Belirli bir m_1 mesajı elimizdeyken, aynı özeti üreten farklı bir $m_2 \neq m_1$ mesajı bulmak bilgisayarsal olarak imkânsız olmalıdır.
3. **Çakışma direnci (collision resistance):** Özet değerleri aynı olan *herhangi* iki farklı mesaj ($m_1 \neq m_2$ ve $H(m_1) = H(m_2)$) bulmak bilgisayarsal olarak imkânsız olmalıdır.

i Çakışmalar zaten vardır

Güvercin yuvası (pigeonhole) ilkesi gereği, sonsuz sayıda girdiyi sonlu sayıda çıktıya eşleyen bir fonksiyonda çakışmalar **matematiksel olarak kaçınılmazdır**. Kriptografik iddia, çakışmaların *var olmadığı* değil; **bulunmalarının** pratikte imkânsız olduğudur.

24.3 Çözümlü Uygulama: Basit Bir Hash Fonksiyonu

Ön-görüntü ve çakışma kavramlarını anlayabilmek için modüler aritmetiğe dayanan çok basit bir hash fonksiyonu tanımlayalım. Gerçek dünyada ters görüntü bulmak imkânsız olsa da, bu basit örnekte matematiğin nasıl işlediğini görebiliriz.

Örnek 24.1. $H : \mathbb{Z} \to \mathbb{Z}_{11}$ şeklinde tanımlanan ve kuralı $H(x) \equiv (4x + 5) \pmod{11}$ olan temel bir hash fonksiyonu verilmiştir.
a) Bu fonksiyon için özeti $H(x) = 2$ olan bir girdi (ters görüntü) bulunuz.
b) Bulduğunuz bu girdiyle çakışan, farklı bir girdi daha bularak bir çakışma çifti oluşturunuz.

Çözüm.
a) Ön-görüntü bulma.
$H(x) \equiv 2 \pmod{11}$ şartını sağlayan bir x değeri arıyoruz:

$$4x + 5 \equiv 2 \pmod{11}$$

Her iki taraftan 5 çıkaralım:

$$4x \equiv 2 - 5 \equiv -3 \pmod{11}$$

Negatif sayıyı mod 11'de pozitif dengiyle değiştirelim ($-3 + 11 = 8$):

$$4x \equiv 8 \pmod{11}$$

$\gcd(4, 11) = 1$ olduğundan her iki tarafı 4'e sadeleştirebiliriz:

$$x \equiv 2 \pmod{11}$$

Böylece $H(x) = 2$ sonucunu veren girdilerden birinin $x = 2$ olduğunu bulduk. Sağlaması: $4(2) + 5 = 13 \equiv 2 \pmod{11}$ ✓
b) Çakışma çifti oluşturma.
Modüler aritmetiğin doğası gereği, girdiye modülün tam katlarını eklediğimizde sonuç değişmez. $x = 2$ için özet 2 ise, $x = 2 + 11 = 13$ girdisi için de özet aynı çıkmalıdır. Sağlayalım:

$$H(13) = 4(13) + 5 = 57 = (11 \cdot 5) + 2 \equiv 2 \pmod{11}$$

Sonuç: $H(2) = 2$ ve $H(13) = 2$ bulunmuştur. $2 \neq 13$ olmasına rağmen aynı özeti ürettikleri için $\{2, 13\}$ kümesi bu fonksiyon için bir **çakışma çiftidir**.
⊠

24.4 Merkle-Damgård Mimarisinin Anatomisi

Matematikçiler şu soruyla karşılaştılar: *"Sonsuz uzunluktaki bir veriyi, sabit uzunlukta güvenli bir çıktıya dönüştüren bir fonksiyonu tek seferde nasıl yazarız?"*
Merkle ve Damgård bu sorunu bir **zincirleme** mantığıyla çözdü. Günümüzde MD5, SHA-1 ve SHA-2 ailelerinin kullandığı bu mimari süreci küçük bloklara böler:

1. **Doldurma (padding).** Gelen mesaj, önce blok boyutunun tam katı olacak şekilde sonuna ek veri getirilerek tamamlanır. Ayrıca güvenlik gereği eklenen dolgunun sonuna mesajın orijinal uzunluğu da yazılır — buna *Merkle-Damgård güçlendirmesi* denir.

2. **Sıkıştırma fonksiyonu.** Mimari, koca bir veriyi tek seferde yutmaya çalışmaz; yalnızca belirli uzunlukta bir mesaj bloğu ve bir önceki adımın sonucunu alıp sabit uzunlukta çıktı üreten küçük bir f fonksiyonu kullanır.
3. **Zincirleme (chaining).** Sisteme sabit bir **başlangıç vektörü (IV)** verilir. Mesaj bloklara ayrılır ve her blok sırayla sıkıştırma fonksiyonundan geçirilir.

24.4.1 Zincirleme Hesaplama Gerçekte Nasıl Çalışır?

Bu sistemi bir **bayrak yarışı** gibi düşünebilirsiniz. Elimizde uzun bir mesaj var ve bunu m_1, m_2, m_3 olmak üzere üç bloğa böldük:

- **Başlangıç.** Yarışa başlamadan önce elimizde sistemin belirlediği sabit bir değer, başlangıç vektörü h_0 vardır.
- **1. aşama.** Fonksiyon, ilk mesaj parçası m_1 ile h_0'ı alır, karıştırır ve h_1'i üretir: $f(h_0, m_1) \to h_1$
- **2. aşama.** Artık yeni bayrak h_1'dir. Fonksiyon sıradaki bloğu bir önceki sonuçla karıştırır: $f(h_1, m_2) \to h_2$
- **3. aşama.** Son blok da önceki sonuçla karıştırılır: $f(h_2, m_3) \to h_3$

Sonuç: Zincirin sonundaki h_3 değeri, tüm mesajın nihai özet değeridir. Böylece her blok, kendinden sonraki tüm hesaplamayı kelebek etkisi gibi değiştirir.

> **i** Merkle-Damgård güvenliği
>
> Sistemin kalbinde çalışan blok bazlı sıkıştırma fonksiyonu f **çakışmaya dayanıklı** ise, bu zincirleme mimariyle kurulan devasa ana özet fonksiyonu da matematiksel olarak çakışmaya dayanıklı kabul edilir. Merkle-Damgård teoreminin özü budur: güvenlik, küçük bileşenden bütüne taşınır.

24.5 Çözümlü Uygulamalar: Merkle-Damgård Yapısı

Örnek 24.2. $\tilde{H} : \{0,1\}^3 \to \{0,1\}^2$ şeklinde tanımlanan bir sıkıştırma fonksiyonu verilmiştir. Çıktı kuralı şudur: **ilk bit**, girdideki bitlerin toplamının mod 2 değerini (pariteyi); **ikinci bit** ise girdinin ortasındaki biti temsil eder.

x (girdi)	$\tilde{H}(x)$ (çıktı)
000	00
001	10
010	11
011	01
100	10
101	00
110	01
111	11

a) $\tilde{H}$ sıkıştırma fonksiyonu için bir çakışma çifti bulunuz.
b) Başlangıç vektörü $h_1 = 00$ olan ve $\tilde{H}$ kullanılarak Merkle-Damgård yapısıyla oluşturulan genel H fonksiyonu için $H(1101)$ değerini hesaplayınız.
c) $H(1101)$ ile çakışan, farklı uzunlukta bir girdi bulunuz.

> **Çözüm.**
> **a) Sıkıştırma fonksiyonu için çakışma.**
> Çakışma, $\tilde{H}(x) = \tilde{H}(x')$ ve $x \neq x'$ durumunda gerçekleşir. Tablo incelendiğinde birden fazla çakışma görülür:
>
> - $\tilde{H}(001) = 10$ ve $\tilde{H}(100) = 10 \implies \{001, 100\}$ bir çakışmadır.

- $\tilde{H}(010) = 11$ ve $\tilde{H}(111) = 11 \Longrightarrow \{010, 111\}$ bir çakışmadır.

b) Merkle-Damgård hesaplaması.
Sıkıştırma fonksiyonumuz 3 bit girdi alıp 2 bit çıktı verir. Her iterasyonda bir önceki adımın 2 bitlik çıktısı ile mesajın sıradaki 1 biti birleştirilir (concatenation, $\|$).
Girdi mesajımız $x = x_1x_2x_3x_4 = 1101$ için adımlar:

1. **Başlangıç:** $h_1 = 00$
2. $x_1 = 1$ işlenir: $h_2 = \tilde{H}(00 \parallel 1) = \tilde{H}(001) = 10$
3. $x_2 = 1$ işlenir: $h_3 = \tilde{H}(10 \parallel 1) = \tilde{H}(101) = 00$
4. $x_3 = 0$ işlenir: $h_4 = \tilde{H}(00 \parallel 0) = \tilde{H}(000) = 00$
5. $x_4 = 1$ işlenir: $h_5 = \tilde{H}(00 \parallel 1) = \tilde{H}(001) = 10$

Nihai sonuç: $H(1101) = 10$.
c) Ana fonksiyon için çakışma bulma.
Hesaplama sürecindeki ara değerlere bakıldığında, 2. adımda elde edilen $h_2 = 10$ dikkat çekicidir. Bu ara değer, aslında yalnızca $x = 1$ (tek bitlik) girdisinin özetidir:

$$H(1) = \tilde{H}(00 \parallel 1) = \tilde{H}(001) = 10$$

Böylece $H(1) = 10$ ve $H(1101) = 10$ olduğu saptanmıştır. $1 \neq 1101$ olmasına ve farklı uzunlukta olmalarına rağmen aynı sonucu verdiklerinden, $\{1, 1101\}$ çifti genel H fonksiyonu için bir çakışma teşkil eder.
Bu örnek, dolgu adımına mesaj uzunluğunun neden yazıldığını da gösterir: Merkle-Damgård güçlendirmesi olmadan, farklı uzunluktaki mesajlar bu şekilde kolayca çakıştırılabilir.
⊠

Örnek 24.3. Merkle-Damgård mimarisini kullanan kendi tasarımımız olan H özet fonksiyonu ile $m =$ `MATH` mesajının özet değerini, yani $H(\texttt{MATH})$ sonucunu bulunuz.
Sistem kuralları:

- H fonksiyonu mesajı 1 harflik bloklara böler; dolayısıyla 4 harfli girdi için $m_1 = M, m_2 = A, m_3 = T$, $m_4 = H$ olmak üzere 4 iterasyon gereklidir.
- Harflerin sayısal değerleri 0–25 tablosuna göredir ($A = 0$, $M = 12$, $T = 19$, $H = 7$).
- Başlangıç vektörü $h_0 = 5$ olarak sabitlenmiştir.
- Sıkıştırma fonksiyonu f, bir önceki durumu yeni mesaj bloğuyla şu formülle birleştirir:

$$f(h_{i-1}, m_i) = (h_{i-1} \cdot 3 + m_i) \pmod{26}$$

- Genel özet fonksiyonumuz zincirleme yapı gereği şöyle tanımlanır:

$$H(m) = f\ (\ f(f(f(h_0, m_1), m_2), m_3), m_4\ ) = h_4$$

Çözüm.
f sıkıştırma adımını sırayla dört kez uygulayacağız. Başlangıç durumumuz $h_0 = 5$'tir.
1. iterasyon ($m_1 = M \Longrightarrow 12$):

$$h_1 \equiv (5 \cdot 3 + 12) = 27 \equiv 1 \pmod{26}$$

2. iterasyon ($m_2 = A \Longrightarrow 0$): Bir önceki zincir çıktısı $h_1 = 1$ kullanılır.

$$h_2 \equiv (1 \cdot 3 + 0) = 3 \pmod{26}$$

3. iterasyon ($m_3 = T \Longrightarrow 19$):

$$h_3 \equiv (3 \cdot 3 + 19) = 28 \equiv 2 \pmod{26}$$

4. iterasyon ($m_4 = H \Longrightarrow 7$):

$$h_4 \equiv (2 \cdot 3 + 7) = 13 \pmod{26}$$

Sonuç: Tüm bloklar zincirleme kuralla işlendiği için son iterasyon değeri bütün mesajın özetine eşittir:

$$H(\texttt{MATH}) = h_4 = 13$$

Tabloya göre 13 sayısının karşılığı **N** harfidir. Böylece `MATH` kelimesinin bu sistemdeki nihai özet değeri `N` olarak hesaplanmıştır.
⊠

25. Dijital İmzalar ve RSA İmza Şeması

Asimetrik şifrelemeyle verinin **gizliliğini**, kriptografik özet fonksiyonlarıyla da verinin **bütünlüğünü** korumayı öğrendik. Ancak dijital dünyada çözülmesi gereken büyük bir problem daha vardır: **kimlik doğrulama (authentication)** ve **inkâr edememezlik (non-repudiation)**.
İnternet üzerinden gelen bir e-postanın veya banka talimatının *gerçekten* iddia edilen kişi tarafından gönderildiğini ve gönderenin sonradan "bunu ben yapmadım" diyemeyeceğini nasıl garanti edebiliriz? İşte bu noktada, asimetrik şifreleme mantığının tam ters yönünde çalışan **dijital imzalar** devreye girer.

> **i** Paradigma değişimi: şifreleme ve imzalama
>
> Asimetrik sistemlerde anahtar kullanım yönü amaca göre tamamen değişir:
>
> - **Gizlilik (şifreleme) amacıyla:** Mesaj, *alıcının* **açık anahtarıyla** şifrelenir; yalnızca *alıcının* **gizli anahtarıyla** çözülebilir. Yani herkes mesaj gönderebilir, yalnızca alıcı okuyabilir.
> - **Kimlik doğrulama (imzalama) amacıyla:** Mesaj, *göndericinin* **gizli anahtarıyla** imzalanır; herkes *göndericinin* **açık anahtarıyla** bu imzayı doğrulayabilir. Yani yalnızca sahibi imza atabilir, herkes doğruluğunu teyit edebilir.

25.1 Dijital İmzanın Formal Tanımı

Tanım 25.1 (Dijital İmza Şeması). Bir dijital imza şeması; mesaj uzayı $\mathcal{M}$, imza uzayı $\mathcal{S}$ ve anahtar uzayı $\mathcal{K}$ olmak üzere şu üç algoritmadan oluşur:

1. **Anahtar üretimi (KeyGen):** Göndericiye ait bir (pk, sk) anahtar çifti üretir. Burada pk herkesin bildiği açık anahtar, sk ise yalnızca göndericinin bildiği gizli anahtardır.
2. **İmza oluşturma (Sign):** Göndericinin gizli anahtarı ve $m \in \mathcal{M}$ mesajı girdi olarak alınır, $s \in \mathcal{S}$ dijital imzası üretilir:
$$s = \text{Sign}_{sk}(m)$$
3. **İmza doğrulama (Verify):** Göndericinin açık anahtarı, m mesajı ve s imzası girdi olarak alınır. İmza gerçekten bu mesaj için ve bu gizli anahtarla üretilmişse **geçerli**, aksi hâlde **geçersiz** çıktısı üretilir:
$$\text{Verify}_{pk}(m, s) \in \{\text{Geçerli}, \text{Geçersiz}\}$$

25.2 Hash-and-Sign Paradigması

Teorik olarak koca bir dosyayı doğrudan asimetrik anahtarla imzalayabilirsiniz. Ancak büyük sayılarla üs alma işlemleri son derece yavaştır; 2 GB'lık bir video dosyasını doğrudan RSA ile imzalamak pratikte kullanılamaz sürelere yayılır.
Ayrıca doğrudan mesajı imzalamak, **varoluşsal sahtecilik (existential forgery)** adı verilen saldırılara kapı aralar: ham RSA imzası çarpımsal olarak homomorfiktir, yani $s_1 \cdot s_2$ değeri $m_1 \cdot m_2$ mesajının geçerli imzası olur. Bu nedenle modern kriptografide **asla doğrudan mesaj imzalanmaz**.

Altın kural: önce özetle, sonra imzala

Dijital imza atılmadan önce mesajın, çakışmaya dayanıklı bir H hash fonksiyonuyla (örneğin SHA-256) özeti çıkarılır. Ardından gizli anahtarla yalnızca bu **küçük özet değeri** imzalanır:

$$s = \text{Sign}_{sk}\left(H(m) \right)$$

Alıcı da mesajın özetini kendisi hesaplar ve imzadan çıkan değerle karşılaştırır. Dosya ne kadar büyük olursa olsun, imzalama daima sabit boyutlu (örneğin 256 bit) bir sayı üzerinden çok hızlı biçimde gerçekleşir.

25.3 RSA Dijital İmza Algoritması

En yaygın bilinen ve kök sertifikalarda hâlâ kullanılan imza yöntemi RSA tabanlıdır. Matematiksel altyapısı, daha önce öğrendiğimiz RSA şifrelemesiyle aynı asalları ve totient kurallarını kullanır.

Tanım 25.2 (RSA İmza Şeması). **1. Anahtar üretimi.** İki büyük asal p ve q seçilir; $n = p \cdot q$ ve $\phi(n) = (p-1)(q-1)$ hesaplanır. $\gcd(a, \phi(n)) = 1$ olacak şekilde açık üs a seçilir ve $a \cdot d \equiv 1 \pmod{\phi(n)}$ denkliğini sağlayan gizli üs d bulunur.

- Açık anahtar (doğrulama için): (a, n)
- Gizli anahtar (imzalama için): (d, n)

2. İmza oluşturma (gönderici). Gönderici mesajın özetini $h = H(m)$ olarak çıkarır ve kendi **gizli anahtarıyla** imzalar:

$$s \equiv h^d \pmod{n}$$

Oluşan (m, s) çifti alıcıya gönderilir.
3. İmza doğrulama (alıcı). Alıcı gelen imzayı göndericinin **açık anahtarıyla** çözerek özet değerine ulaşmaya çalışır:

$$h' \equiv s^a \pmod{n}$$

Alıcı aynı zamanda gelen mesajın özetini kendisi de hesaplar. Eğer $h' = H(m)$ ise imza geçerlidir: belge değiştirilmemiştir ve gizli anahtarın sahibi tarafından imzalanmıştır.

Teorem 25.1 (RSA İmzasının Doğruluğu). *Doğrulama adımındaki $s^a \pmod{n}$ işleminin sonucu her zaman orijinal mesajın özeti olan $H(m)$ değerine denktir.*

> ***İspat.***
> *İmza oluşturma denkleminden $s \equiv H(m)^d \pmod{n}$ olduğunu biliyoruz. Doğrulama fonksiyonunda s yerine bu eşdeğerini koyalım:*
>
> $$s^a \equiv \left(H(m)^d \right)^a \equiv H(m)^{d \cdot a} \pmod{n}$$
>
> *Anahtar üretim adımından $a \cdot d \equiv 1 \pmod{\phi(n)}$ olduğunu ve çarpmanın değişme özelliği gereği $d \cdot a = a \cdot d$ olduğunu biliyoruz.*
> *RSA şifrelemesinin doğruluk ispatında Fermat'nın küçük teoremi ve Çin kalan teoremi yardımıyla gösterdiğimiz kural burada da birebir geçerlidir:*
>
> $$H(m)^{a \cdot d} \equiv H(m) \pmod{n}$$
>
> *Böylece $s^a \equiv H(m) \pmod{n}$ eşitliği sağlanır ve imzanın doğru biçimde eşleştiği kanıtlanmış olur.*
> $\boxtimes$

25.4 Çözümlü Uygulama: Sayısal RSA İmzası

Örnek 25.1. Bir kullanıcının RSA anahtar çifti için başlangıç asalları $p = 5$ ve $q = 11$, açık üs değeri ise $a = 3$ olarak belirlenmiştir. Bu kullanıcı, özet değeri $H(m) = 14$ olarak hesaplanan bir PDF dosyasını imzalayacaktır.
Gerekli anahtarları oluşturunuz, dosyanın dijital imzasını hesaplayınız ve alıcı tarafında bu imzanın nasıl doğrulandığını gösteriniz.

Çözüm.
1. Anahtar üretimi.

- Modül: $n = 5 \cdot 11 = 55$
- Euler totient: $\phi(n) = (5-1)(11-1) = 4 \cdot 10 = 40$
- Açık üs $a = 3$ verilmiş; $\gcd(3, 40) = 1$ şartı sağlanıyor.
- Gizli üs, $3d \equiv 1 \pmod{40}$:

$$3 \cdot 27 = 81 = (40 \cdot 2) + 1 \equiv 1 \pmod{40} \Longrightarrow d = 27$$

Buradan: **açık anahtar** $(3, 55)$, **gizli anahtar** $(27, 55)$.
2. İmza oluşturma (gönderici adımı).
Gönderici özet değerini yalnızca kendisinin bildiği $d = 27$ ile imzalar:

$$s \equiv 14^{27} \pmod{55}$$

Ardışık kare alma yöntemiyle hesabı küçültelim:

$$\begin{aligned} 14^1 &\equiv 14 \pmod{55} \\ 14^2 &= 196 = (55 \cdot 3) + 31 \equiv 31 \pmod{55} \\ 14^4 &\equiv 31^2 = 961 = (55 \cdot 17) + 26 \equiv 26 \pmod{55} \\ 14^8 &\equiv 26^2 = 676 = (55 \cdot 12) + 16 \equiv 16 \pmod{55} \\ 14^{16} &\equiv 16^2 = 256 = (55 \cdot 4) + 36 \equiv 36 \pmod{55} \end{aligned}$$

Üssü parçalayalım: $27 = 16 + 8 + 2 + 1$, dolayısıyla

$$14^{27} = 14^{16} \cdot 14^8 \cdot 14^2 \cdot 14^1 \equiv 36 \cdot 16 \cdot 31 \cdot 14 \pmod{55}$$

Parçalı çarpalım:

$$36 \cdot 16 = 576 = (55 \cdot 10) + 26 \equiv 26 \pmod{55}$$

$$31 \cdot 14 = 434 = (55 \cdot 7) + 49 \equiv 49 \equiv -6 \pmod{55}$$

Son çarpım:

$$26 \cdot (-6) = -156$$

-156'nın mod 55 altındaki pozitif denkini bulalım $(55 \cdot 3 = 165)$:

$$-156 + 165 = 9 \Longrightarrow s = 9$$

Oluşturulan dijital imza: $s = 9$.
3. İmza doğrulama (alıcı adımı).
Alıcı dosyayı ve $s = 9$ imzasını alır; göndericinin açık anahtarı $(a, n) = (3, 55)$ ile imzayı açar:

$$h' \equiv s^a \pmod{n} \Longrightarrow h' \equiv 9^3 = 729 \pmod{55}$$

$55 \cdot 13 = 715$ olduğundan:

$$729 - 715 = 14 \Longrightarrow h' = 14$$

Sonuç: Alıcının imzadan çıkardığı değer ($h' = 14$), dosyanın orijinal özet değeriyle ($H(m) = 14$) birebir eşleşmiştir. İmza geçerlidir: belge yolda değiştirilmemiştir ve yalnızca $d = 27$ gizli anahtarına sahip kişi tarafından oluşturulabilir.
$\boxtimes$

