2024-2025 Bahar Yarıyılı
Bilgisayar Mühendisliğinde Özel Konular Dersi-3
3. Ödevi

Konu: Bitcoin için Proof-of-work gerçeklemesi

Açıklama: Proof-of-work (PoW), yaratıcısının önemli miktarda hesaplama çabası harcayarak kriptografik bir kanıt yarattığı bir mekanizmadır. Bu kriptografik kanıt, yaratılışının aksine, çok verimli ve hızlı bir şekilde herhangi biri tarafından doğrulanabilir.

Bitcoin özelinde PoW'un iki işlevi vardır:

1. Geçerli bir blok hash'ine sahip bir blok oluşturmayı o kadar zor hale getirir ki, Bitcoin ağındaki iki düğüm tarafından aynı anda bir blok önerilmesi çok düşük bir olasılıktır.
2. Geçerli bir hash'e sahip bir blok oluşturmak için gereken hesaplama çok fazla olduğundan, bir saldırganın eski bir işlemi değiştirmek amacıyla birçok bloğun hash bağlantılarını değiştirmesi pratikte mümkün değildir.

Bu ödev iki parçadan oluşmaktadır:

1. Birinci bölümde, verilen bir dosya için **PoW hesaplayan bir program** yazmanız istenmektedir. Komut satırından belirteceğiniz zorluk seviyesine göre programınız PoW hesaplayacak ve çıktı olarak bir dizi başlık (header) yazdıracaktır.
2. İkinci bölümde ise birinci bölümde yazdığınız programın oluşturduğu başlıkları doğrulayan bir program yazmanız isenmektedir. Eğer PoW veya hash geçerli değilse, doğrulama başarısız olacaktır.

Hashcash, e-postalarda istenmeyen mesajları (spam) azaltmak için bir mesajı göndermeden önce göndericinin zor bir bulmacayı çözmesini gerektiren bir sistemdir. Gönderici, bu bulmacayı çözdüğüne dair bir kanıt sunar. Eğer kanıt eksik veya geçersizse, mesaj reddedilir. **Bu kanıt, e-posta mesajında "stamp" (damga) adı verilen bir başlık (header) olarak sunulur.**

Hashcash kullanan biri, bu damgayı oluşturmak için birkaç saniye harcayabilir. Bu, az sayıda mesaj gönderen biri için sorun teşkil etmezken, milyonlarca mesaj için damga üretmesi gereken bir spam yapan kişi için neredeyse imkânsız hâle gelir. Bulmaca, e-posta mesajının içeriğini ve alıcının adresini kullandığı için, spam yapan kişinin bir milyon mesaj gönderebilmesi için yıllarca sürecek hesaplamalar yapması gerekir. Alıcı tarafında ise bulmacanın çözümünün verimli bir şekilde doğrulanabilmesi gereklidir. Bu yüzden çözülmesi zor ama doğrulanması kolay bir bulmacaya ihtiyaç vardır.

Bitcoin ve birçok kripto para birimi, Hashcash'in altında yatan fikri kendi sistemlerine uyarlamıştır. **Sisteme katılanlar, Bitcoin blok zincirine yeni bir blok ekleyebilmek için PoW bulmacasını çözdüklerini kanıtlamak zorundadır**.

Bulmaca

Hashcash, mesajın içeriğine bağlı olan, hesaplanması yeterince zor fakat doğrulanması verimli olan bir bulmaca oluşturmak zorundadır. Kriptografik hash fonksiyonları, sabit uzunlukta çıktı veren tek yönlü fonksiyonlardır.

Örneğin, "The grass is greener" metninin SHA-256 hash'i şu şekildedir:
f3ccca8f3852f5e2932d75db5675d59de30d9fa10530dd9855bd4a6cd0661d8e

Bu hash'in hesaplanması yalnızca birkaç milisaniye sürer. Ancak tersini yapmak, yani bu hash'i üreten metni bulmak, kaba kuvvet (brute-force) yöntemiyle denenebilecek sayısız mesaj arasında arama yapmayı gerektirir. Mesela SHA-1 hash fonksiyonunu brute-force yöntemiyle kırmak için 12.000.000 GPU-yıllık hesaplama gücü gereklidir. Daha güçlü bir hash fonksiyonu olan SHA-2'ye karşı aynı saldırı daha da fazla hesaplama gücü gerektirir. Bu durum açıkça göstermektedir ki, bir hash'i tersine çevirmek son derece zor bir problemdir.

Daha kolay bir bulmaca

Bu bulmacayı daha kolay hale getirebiliriz. Diyelim ki elimizde bir metin (**W**) var ve bunu bir mesaj (**M**) ile birleştiriyoruz. **W||M** ifadesini hash'lediğimizde ortaya çıkan hash değerinin belli bir özelliğe sahip olup olmadığını kontrol ederek bulmacayı daha kolay hâle getirebiliriz.

Yukarıdaki örneğe dönecek olursak, "The grass is greener" metninin SHA-256 hash değeri şudur: **f3ccca8f3852f5e2932d75db5675d59de30d9fa10530dd9855bd4a6cd0661d8e**.

Eğer bu değerin başındaki byte değerlerine ikili tabanda bakacak olursak, göreceğimiz bit dizisi şu şekildedir: **1111 0011 1100…** Şimdi bu mesajı öyle bir şekilde değiştirelim ki SHA-256 hash değerinin **ilk 6 biti 0 olsun**. Bu hash değerini veren bir mesajı tahmin yoluyla bulamayacağımız için yapabileceğimiz tek şey, farklı **W** değerlerini mesajımızın önüne ekleyerek oluşturduğumuz **W||M** ifadesini hash'leyip sonucu kontrol etmektir.

Örneğin, metnin başına 'f' harfini ekleyip hash'ini hesaplarsak, alacağımız sonuç şu olur:

0189108649ff4cd02c8af4e099c8d719ec54eff327df14a89305932926f4bd93

Bu hash değeri, ilk 6 bitin 0 olması koşulunu karşılar çünkü yukardaki hash'in ikili düzendeki karşılığı şu şekildedir:

0000 0001 1000 1001…

Adaptif zorluk

Bulmacanın zorluğunu, bulmamız gereken öncü 0 bitlerinin (leading 0 bits) sayısını değiştirerek ayarlayabiliriz. Bu zorluk ortalama bir değerdir. Şansımız yaver giderse istenilen sonucu üreten bir ön eki (prefix) hızlıca bulabiliriz. Zorluk, baştaki 0 bitlerinin sayısıdır. Bu örnekte, ilk 23 biti 0 olan bir hash sonucunu veren bir prefix bulmak için 1 milyon prefix varyasyonu denemek gerekmektedir. Ancak, ilk 27 biti 0 olan bir hash için bir prefix bulmak yaklaşık 28 milyon deneme gerektirir.

Mesaj boyutundan bağımsız zorluk

Büyük mesajlar için hash hesaplama süresi daha uzundur çünkü daha fazla byte üzerinde işlem yapılması gerekir. Zorluğun mesaj uzunluğuna bağlı olmaması için bulmacayı şu şekilde değiştirebiliriz: Önce mesajın (**M**) hash'ini alırız, ardından bu hash'e bir prefix ekleriz ve ortaya çıkan yeni mesajın hash'ini hesaplayıp sonucu kontrol ederiz. Yani hesaplayacağımız şey: **SHA-256((W||SHA-256((M))** olacaktır.

Bu durumda problemin ortalama zorluğu yalnızca **d** değeriyle, yani hash sonucunun sahip olması gereken baştaki 0-bit sayısı ile tanımlanır. Doğru **W** prefixini bulmak icin harcanacak süre ise birkaç faktöre bağlıdır:

- ***Şans:*** Aradığımız sonucu veren stringleri tamamen şans eseri erken bulabiliriz. Ama ara sıra şanslı olsak da, ortalamada şanslı ve şanssız testler birbirini dengeler.
- ***Bilgisayarınızın hızı ve kodunuzun kalitesi:*** Hash işlemleri, tıpkı Bitcoin madenciliğinde olduğu gibi, GPU'lar veya özel işlemciler kullanılarak önemli seviyede hızlandırılabilir.
- ***En çok da zorluk değeri (d) önemlidir:*** Yani, hedeflenen hash değerinde başta olması gereken 0-bitlerinin sayısı.

Proof-of-Work

Bulunan ön ek, yani **W** string'i, mesajla (**M**) birleştirildiğinde istenilen hash çıktısını üretiyorsa, buna Proof-of-Work (PoW) denir. Bu string, bu değeri bulmak için gerçekten çalıştığımızı gösteren kriptografik kanıttır.

PoW doğrulaması ise son derece verimlidir. Örneğin, "The grass is green" metni için ilk 31 bitin 0 olduğu bir hash elde etmek amacıyla dört milyardan fazla hash üretmek gerekebilir. PoW'u doğrulamak için yalnızca bir hash işlemi yapmak yeterlidir.

Problemin İşlem Adımları:

Bu ödevde verilen bir dosya için PoW oluşturan **pow-create** adında bir program yazmanız istenmektedir. Ödev iki bölümden oluşmaktadır.

1. Bölüm

pow-create nbits fileName

pow-create programının yapması gerekenler:

1. Dosyanın SHA-256 hash değerini hesapla.
2. Hash değerini okunabilir bir string'e çevir (openssl komutunun ürettiği string ile aynı olmalı).
3. Potansiyel bir PoW string'i seçin. Prefixleri nasıl bulacağınız size kalmış. Bunu yapmanın farklı yöntemleri var, istediğiniz şekilde yapabilirsiniz. Fakat, prefixleri yalnızca yazdırılabilir **7-bit ASCII metinlerden (printable 7-bit ASCII text)** oluşacak şekilde tutun (örneğin, ü, ñ veya é gibi genişletilmiş karakterler kullanmayın). Ayrıca boşluk karakterleri (whitespace characters) da kullanmamalısınız. Testi kolaylaştırmak için, tırnak işaretlerini (" veya ') de string'in bir parçası olarak kullanmayın.
4. (Bunun için harfleri ve sayıları random sıralayarak bir string oluşturabilirsiniz).
5. (3)'teki string'i, (2)'de elde edilen hash string'inin önüne ekle **(W||hash)** ve bu yeni string'in SHA-256 hash'ini hesapla.

6. Eğer elde edilen hash en az **nbits** kadar 0-bit ile başlıyorsa PoW bulunmuştur. Eğer değilse, (3)'e geri dön ve yeni bir string ile dene.

Buradaki amaç, dosyanın (**file**) içeriğinin SHA-256 hash'i ile başlayan bir veriye, bir string (**W**) ekleyip tekrar SHA-256 hash'i almak ve bu hash'in başında **nbits** kadar 0-bit olmasını sağlamaktır. Diyelim ki elimizde walrus.txt adlı bir dosya var ve içeriği şöyle:

> The time has come, the Walrus said,
> To talk of many things:
> Of shoes — and ships — and sealing-wax —
> Of cabbages — and kings —
> And why the sea is boiling hot —
> And whether pigs have wings.

Programınız bu dosyayı okuyacak, içeriğinin SHA-256 hash'ini alacak ve sonrasında rastgele string'ler (**W**) deneyerek şu koşulu sağlayan bir tanesini bulmaya çalışacak:

SHA-256(W||SHA-256((file)) sonucunun başında **nbits** kadar 0-bit olacak

macOS veya Linux sistemlerinde bir dosyanın SHA-256 hash'ini bulmak için **openssl** komutunu kullanabiliriz:

```
$ openssl sha256 < walrus.txt
66efa274991ef4ab1ed1b89c06c2c8270bb73ffdc28a9002a334ec3023039945
```

Bu hash **66ef** ile basliyor. Buradaki **6 karakterinin binary degeri 0110** oldugu icin bu mesajin sadece ilk biti 0'dır.

Örnek:
Eğer zorluk derecesi 20 olan bir hash sonucu istiyorsak (yani en az ilk 20 biti 0 olan), programınızı aşağıdaki şekilde çalıştırınız:

```
$ pow-create 20 walrus.txt
File: walrus.txt
Initial-hash: 66efa274991ef4ab1ed1b8...28a9002a334ec3023039945
Proof-of-work: hl04
Hash: 000002b2311ce58427ab7c1bfd0cb1...3d948c1c603a524dc11fb28
Leading-bits: 22
Iterations: 1496419
Compute-time: 1.75376
```

Yukarıdaki örnekte walrus.txt dosyası için zorluk derecesi 20 olan PoW çözümü **hl04** string'i olmuştur.

Program çıktıları

Programınızın çıktısı e-posta başlıklarında veya HTTP başlıklarında kullanılan standart başlık formatında olmalıdır.
Bu format:
her satırda bir ad-değer çifti içerir. Her satır şu şekilde olmalıdır: başlık adı, iki nokta (:), bir veya daha fazla boşluk, başlık değeri.

Çıktıda yazdırılacak başlıklar şunlardır:
File: İşlem yapılan dosyanın adı.
Initial-hash: Dosyanın SHA-256 hash değeri.
Proof-of-work: İstenilen zorluk derecesinde hash değeri sağlayan **W** string'i.
Hash: PoW değeri ile **Initial-hash** değerinin birleştirilip tekrar SHA-256 ile hash'lenmiş hâli.
Leading-bits: Elde edilen hash'in başında kaç tane 0-bit olduğu. En az istenilen zorluk derecesi kadar olmalıdır.
Iterations: Uygun bir PoW değeri bulunana kadar kaç farklı prefix denemesi yapıldığı.
Compute time: Bu işlemin saniye cinsinden ne kadar sürdüğü.

PoW Doğrulaması

Programınızın çıktısını, **openssl** komutuyla kolayca test edebilirsiniz. **pow-create** komutunuzu çalıştırın:

$ ./pow-create 20 walrus.txt
Initial-hash: 66efa274991ef4ab1ed1b8...28a9002a334ec3023039945
Proof-of-work: hl04
Hash: 000002b2311ce58427ab7c1bfd0cb1...3d948c1c603a524dc11fb28
Leading-bits: 22
Compute-time: 1.75376

Yukarıdaki çıktı biraz kısaltılmıştır. Burada tek ilgilendiğimiz şey, PoW değeri olan **hl04**'tür.

openssl komutunu **sha256** argümanıyla çalıştırarak kendi hash'imizi kontrol edebiliriz. Bu değer, **Initial-hash** başlığıyla eşleşmelidir:

$ openssl sha256 <walrus.txt
66efa274991ef4ab1ed1b89c06c2c8270bb73ffdc28a9002a334ec3023039945

Daha sonra, PoW stringi ile orijinal hash'in hex çıktısının birleştirilmiş halinin SHA-256 hash'ini buluruz:

$ echo -n 'hl0466efa27499...9002a334ec3023039945' | openssl sha256
000002b2311ce58427ab7c1bfd0cb1679906b24343d948c1c603a524dc11fb28

Gördüğünüz gibi, bulduğumuz hash değerinin ilk 22 biti 0 değerine sahip. Bu da istediğimiz zorluk derecesi olan 20'yi karşıladığı için **hl04** stringi geçerli bir PoW'dur.

İpuçları

- Bu çok kısa bir ödev olacak zira SHA-256 hash fonksiyonunu kendiniz yazmayacaksınız. Python'un **hashlib** modulunu kullanabilirsiniz.
- Hash fonksiyonunuzun, **openssl** ile aynı sonuçları verdiğini test edin. Bu işlem prefixleri test etmek için gereklidir. Alıcının sizin kodunuzu kendi koduyla doğrulayabilmesi gerekir.
- Kodunuzu küçük zorluk seviyeleriyle test edin. Eğer 20 bit'lik bir zorluk seviyesine kadar PoW stringleri üretebiliyorsanız, kodunuz muhtemelen daha yüksek zorluk seviyeleri için de çalışacaktır.

2. Bölüm

Ödevinizin ikinci bölümünde **pow-check** isminde bir doğrulama programı yazmanız istenmektedir:

pow-check powheader fileName

Komut, **pow-create** komutu tarafından üretilmiş başlıkları içeren bir dosya ile orijinal mesajın bulunduğu bir dosyayı alır ve başlıkları bu dosyaya karşı doğrular.

Gerçekleştirdiği testler şunlardır:

- Başlıktaki **Initial-hash** değerini kontrol eder. Bu, mesajın SHA-256 hash değeri olmalıdır.
- Başlıktaki PoW stringi ile bu ilk hash değerini birleştirip yeni bir hash hesaplar. Bu hesaplanan değer, Hash başlığındaki değerle eşleşmelidir.
- Son olarak, **Leading-bits** başlığındaki sayı, **Hash** başlığındaki hash'in başındaki 0-bit sayısıyla aynı olmalıdır.

pow-check komutunun çıktısı "başarılı" ya da "başarısız" olacaktır. Eğer testler geçerse, sadece "başarılı" mesajı yazdırılmalıdır. Herhangi bir test başarısız olursa, hangi test(ler)in başarısız olduğunu belirtilmelidir.

Teslim İşlemleri: Ödev raporunuzu ve **Python** dilinde yazdığınız programınızı **15 Haziran 2025 Pazar 23.45'e kadar** sisteme yükleyiniz. Ödev raporu yaptığınız işlemlerin kısa anlatımı ve örnek ekran çıktılarından oluşmalıdır. Dosyalar ismi ÖğrenciNumarasıAdSoyad olan klasör içinde olmalıdır. Ödev raporunuzu (pdf olarak) ve yazdığınız program kodunuzu birlikte zip'leyerek sisteme yükleyiniz.

Ödev Kaynağı: https://people.cs.rutgers.edu/~pxk/419/hw/a-13.html