
بسم الله الرحمن الرحیم

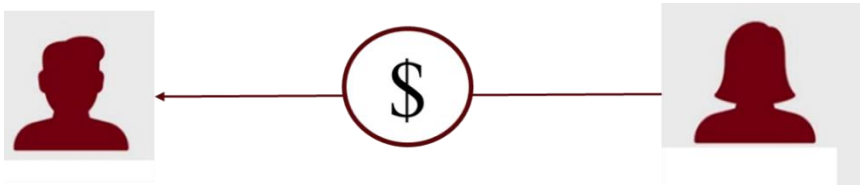
زنجیره‌ی بلوکی بیت‌کوین

برنامه‌نویسی و عملکرد توابع

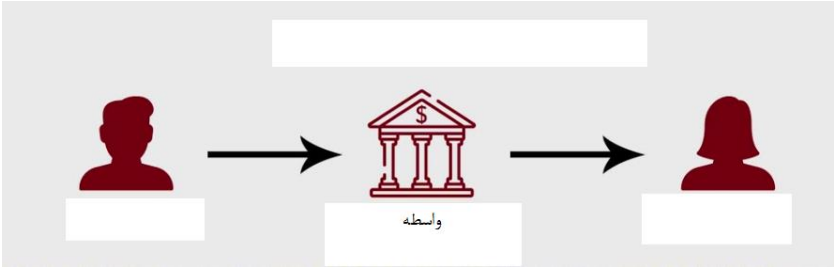
فصل دوم

مفاهیم پایه‌ی زنجیره‌ی بلوکی

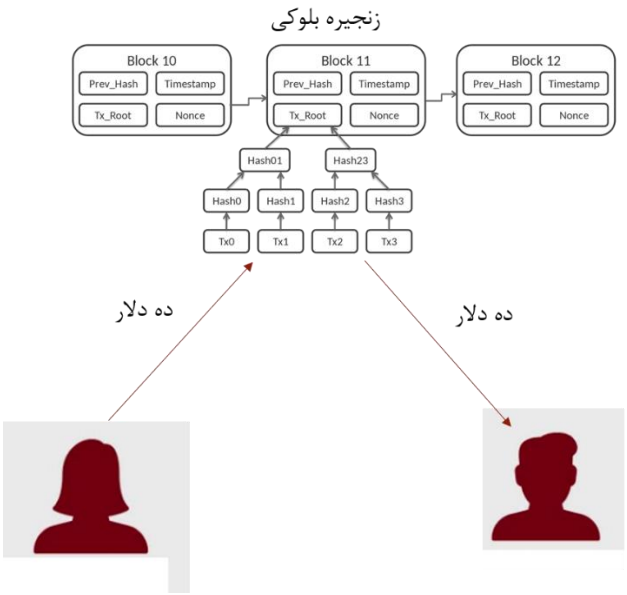
اعتمادسازی در زنجیره بلوکی



شکل ۲-۱: یک سیستم کاملاً بی‌نیاز از اصل اعتمادسازی

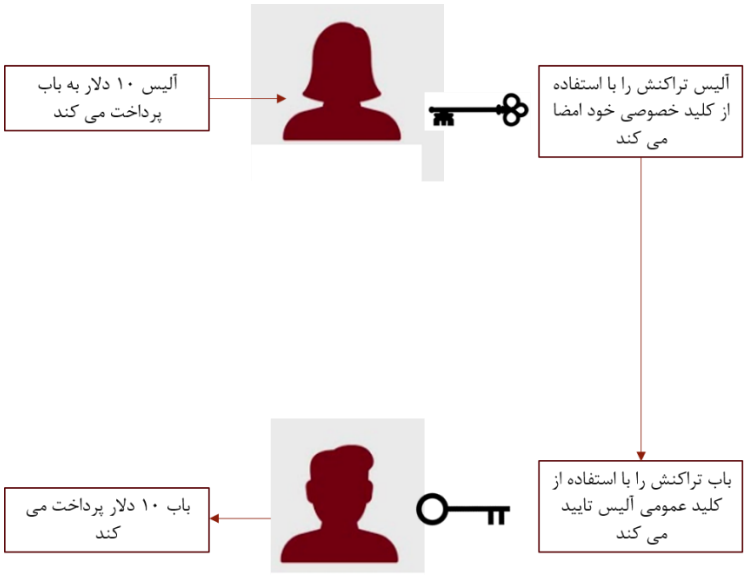


شکل ۲-۲: سیستم تراکنشی بدون در نظر گرفتن مسافت

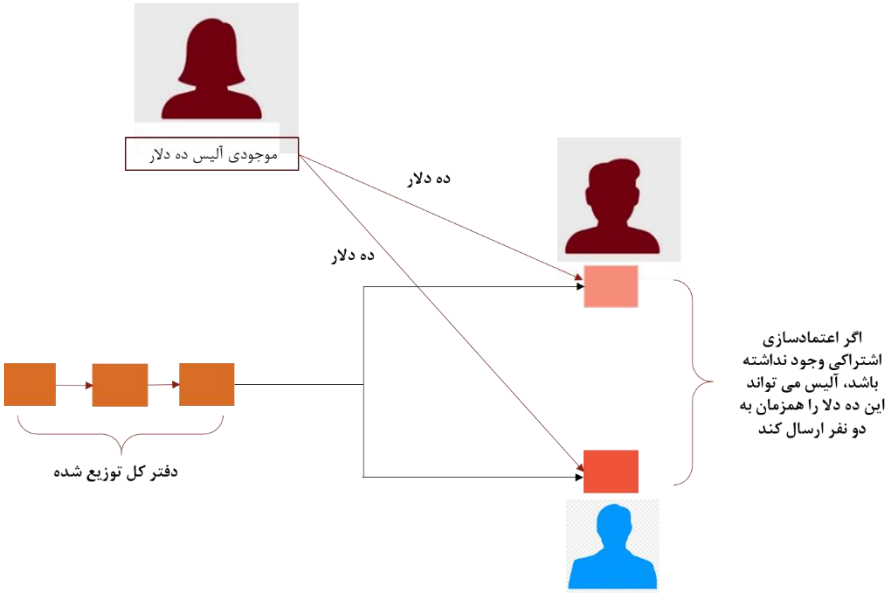


شکل ۲-۳: تراکنش با واسطه‌ی یک زنجیره‌ی بلوکی

رمزنگاری کلید عمومی

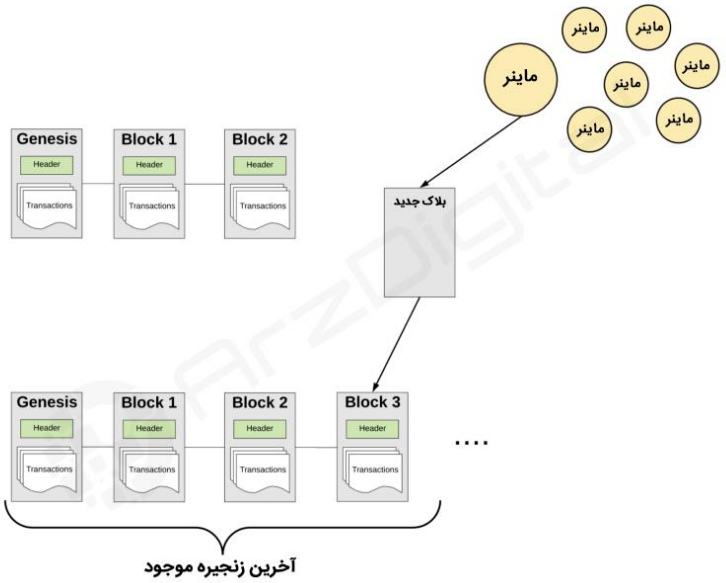


شکل ۲-٤: تائید صحت یک تراکنش با استفاده از رمزنگاری کلید عمومی

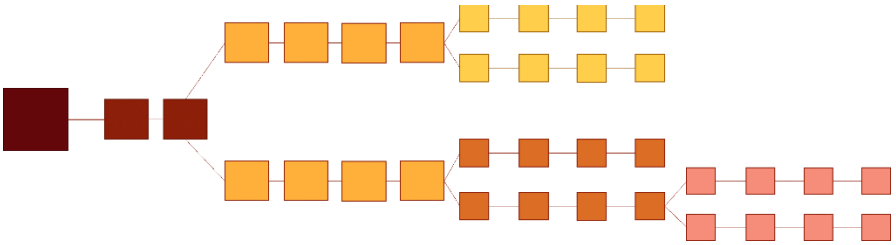


شکل ۲-۵: سیستم اجماع برای جلوگیری از دوباره خرج کردن پول

اجماع

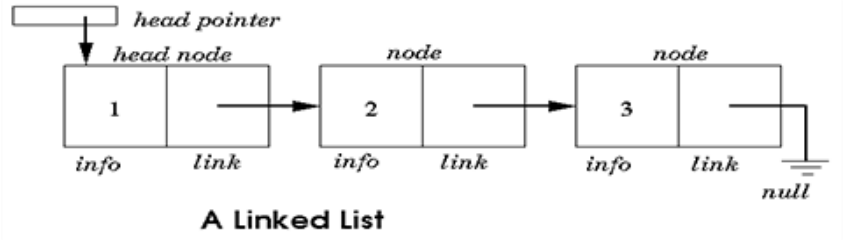


شکل ۲-٦: سیستم اجماع با رویکرد استخراج



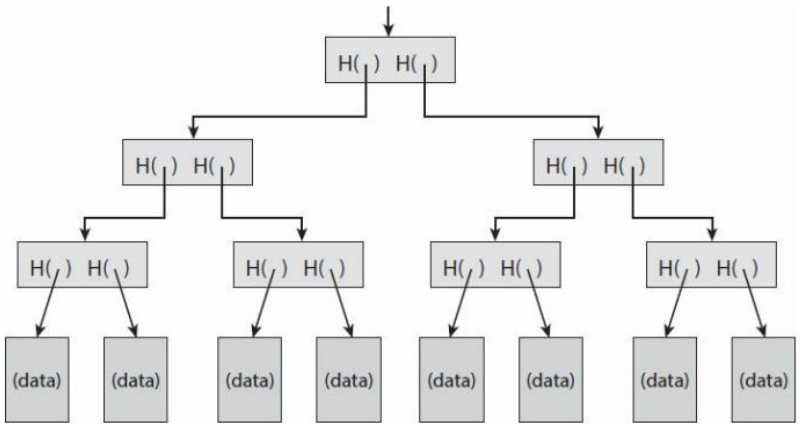
شکل ۲-۷: چند شاخه شدن زنجیره‌ی بلوکی در فرآیند استخراج و رقابت میان استخراج‌کنندگان

هش و ساختار داده‌ها

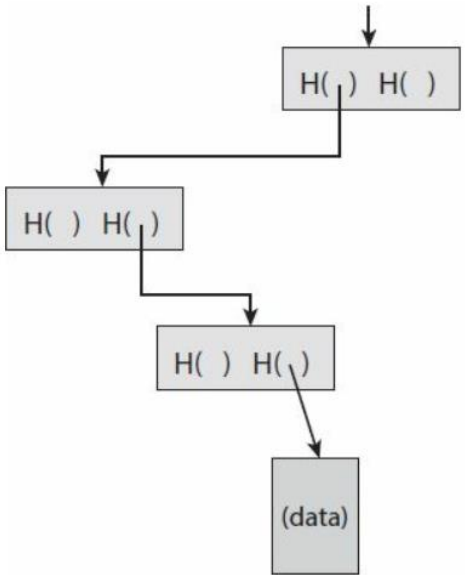


شکل ۲-۸: لیست پیوندی یک‌طرفه خطی

درخت مرکل

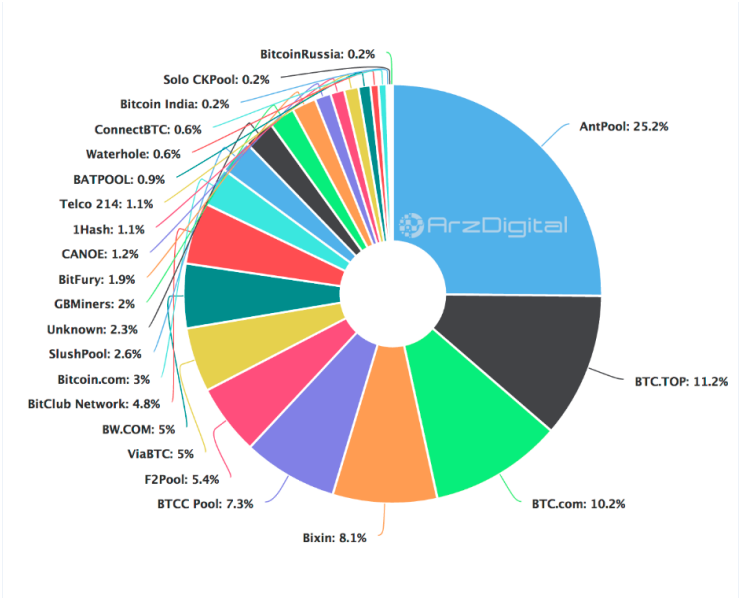


شکل ۲-۹: درخت مرکل

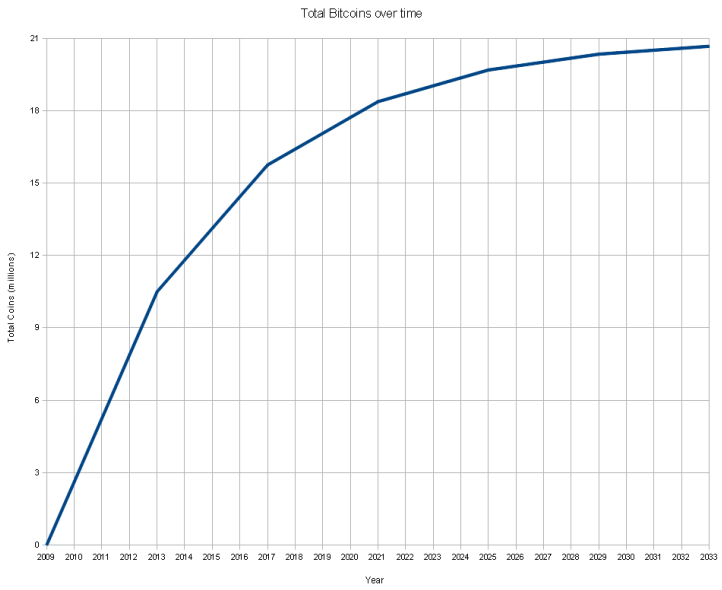


شکل ۲-۱۰: اثبات عضویت یک گره در یک زنجیره‌ی بلوکی

استخر استخراج بیت‌کوین

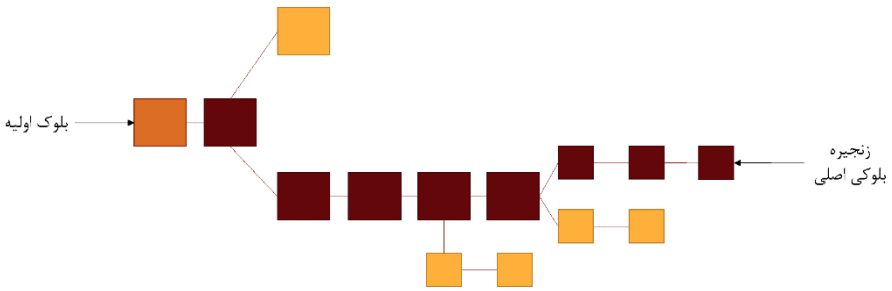


شکل ۲-۱۱: استخراج معتبر و حجم استخراج در این استخرها



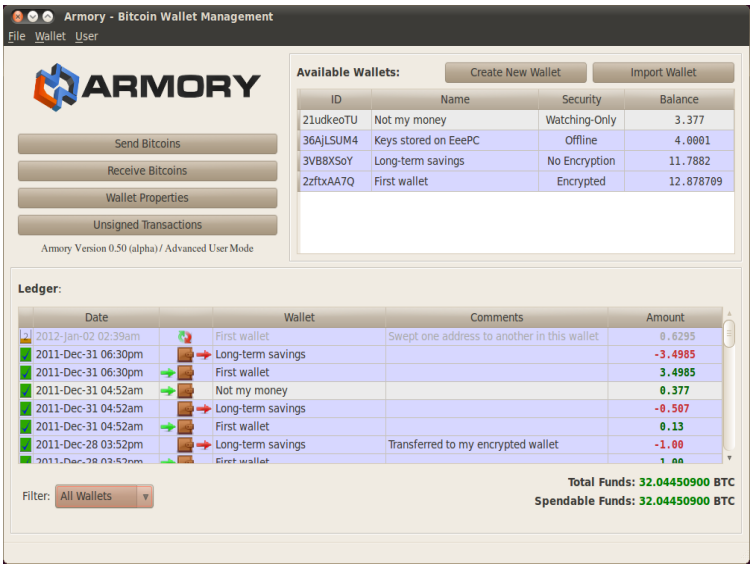
شکل ۲–۱۲: میزان تولید بیت‌کوین تا سال ۲۰۰۳

انشعاب



شکل ۲–۱۳: زنجیره‌ی استاندارد طولانی‌ترین زنجیره‌ای است که بیشترین استخراج‌کننده‌ی فعال را در خود دارد.

کیف پول



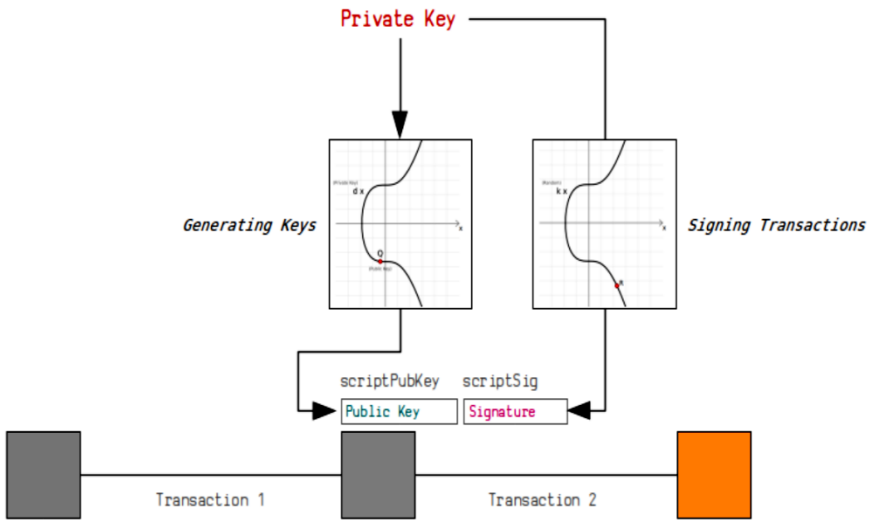
شکل ۲-۱۴: نمایی از کیف پول Armory

شکل ۲-۱۵: مراحل ایجاد کیف پول کاغذی

شکل ۲-۱۶: ایجاد کیف پول کاغذی

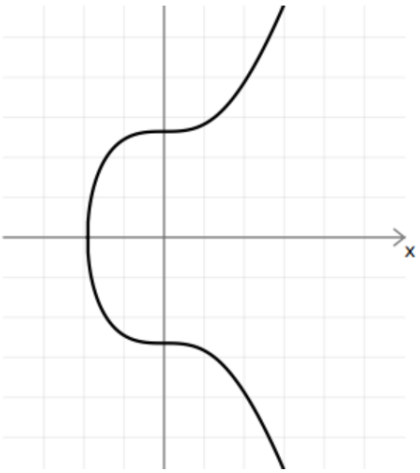
فصل سوم

الگوریتم امضای دیجیتالِ منحنی بیضی‌گون



شکل ۳-۱: الگوریتم ECDSA

منحنی بیضی‌گون



شکل ۳-۲: منحنی بیضی‌گون

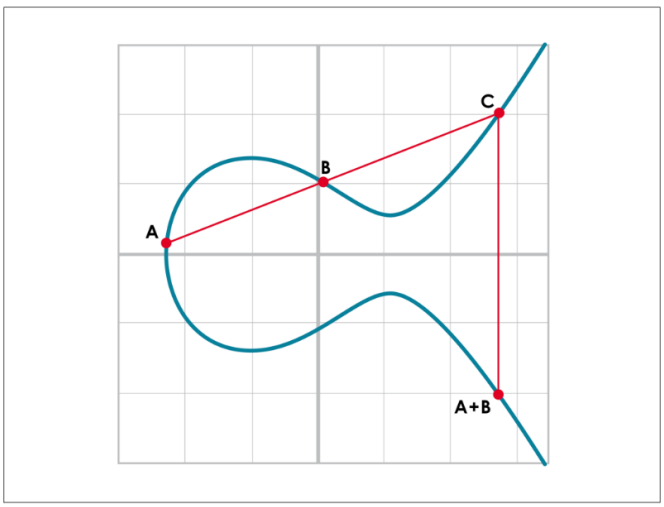
```
class Point:

    def __init__(self, x, y, a, b):
        self.a = a
        self.b = b
        self.x = x
        self.y = y
        if self.y**2 != self.x**3 + a * x + b:
            raise ValueError('({}, {}) is not on the
curve'.format(x, y))

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y \
            and self.a == other.a and self.b == other.b
```

1

2

ایجاد شیء Point و خطای متناظر با آن

```
>>> from ecc import Point
>>> p1 = Point(-1, -1, 5, 7)
>>> p2 = Point(-1, -2, 5, 7)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "ecc.py", line 143, in __init__
    raise ValueError('({}, {}) is not on the
curve'.format(self.x, self.y))
ValueError: (-1, -2) is not on the curve
```



شکل ۳-۳: point addition

کد Point addition در پایتون

```
>>> from ecc import Point
>>> p1 = Point(-1, -1, 5, 7)
>>> p2 = Point(-1, 1, 5, 7)
>>> inf = Point(None, None, 5, 7)
>>> print(p1 + inf)
Point(-1,-1)_5_7
>>> print(inf + p2)
Point(-1,1)_5_7
>>> print(p1 + p2)
Point(infinity)
```

```
class Point:

    def __init__(self, x, y, a, b):
        self.a = a
        self.b = b
        self.x = x
        self.y = y
        if self.x is None and self.y is None:
            return
        if self.y**2 != self.x**3 + a * x + b:
            raise ValueError('({}, {}) is not on the
curve'.format(x, y))

    def __add__(self, other):
        if self.a != other.a or self.b != other.b:
            raise TypeError('Points {}, {} are not on the
same curve'.format
            (self, other))

        if self.x is None:
            return other
        if other.x is None:
            return self
```

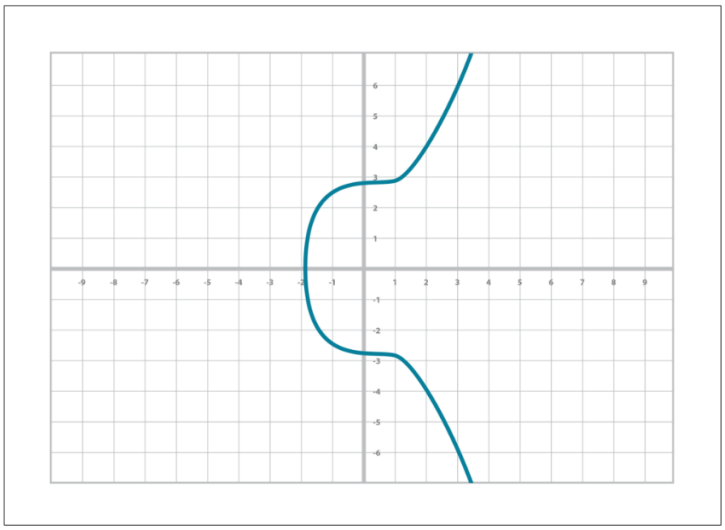
1

2

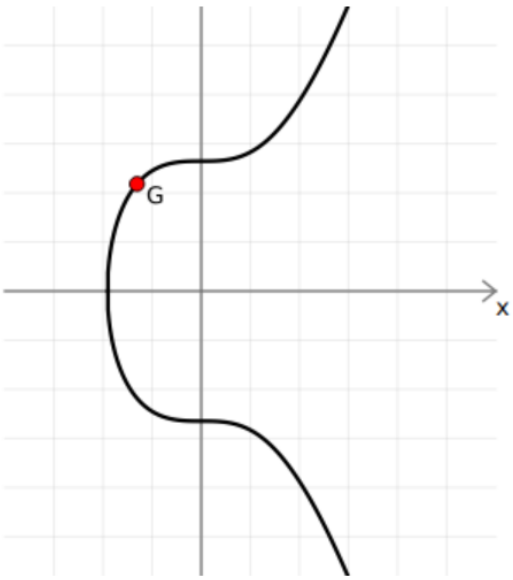
3

4

پارامترها



شکل ۳-٤: منحنی بیضی‌گون secp256k1

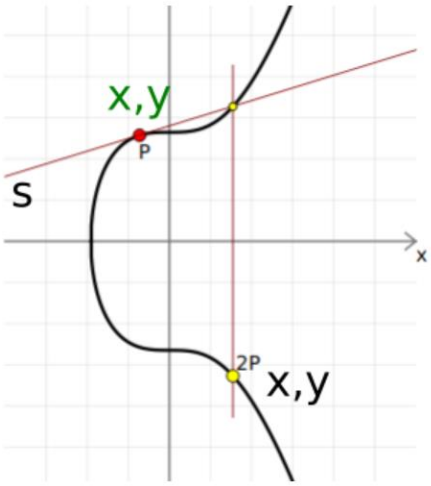


شکل ۳-۵: هر منحنی بیضی‌گون شامل یک نقطه‌ی تولید کننده‌ی G است.

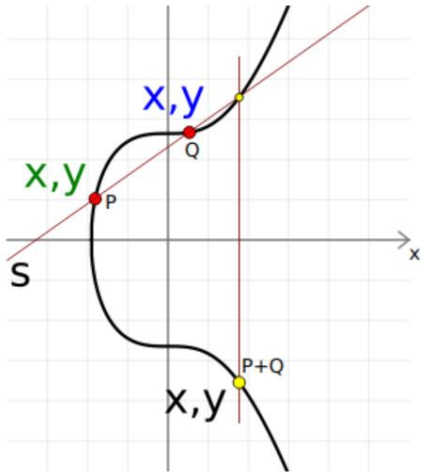
وارون ضربی (هم‌نهشتی)

```
def inverse(a, m = $p)
  m_orig = m # store original modulus
  a = a % m if a < 0 # make sure a is positive
  prevy, y = 0, 1
  while a > 1
    q = m / a
    y, prevy = prevy - q * y, y
    a, m = m % a, a
  end
  return y % m_orig
end
```

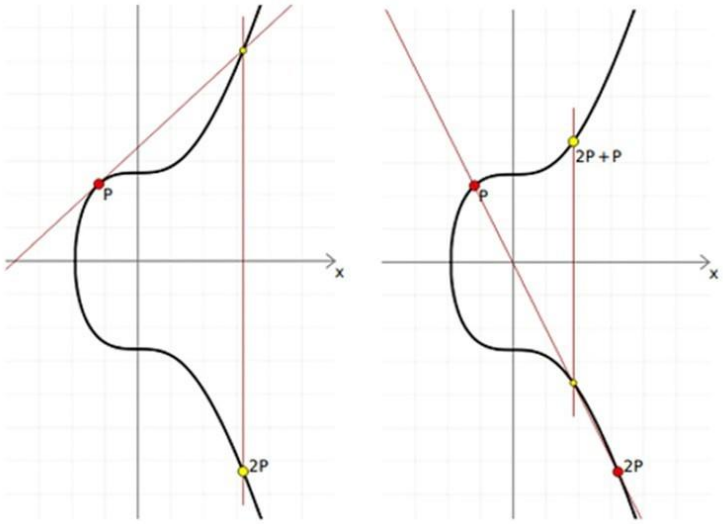
دو برابر کردن



اضافه کردن

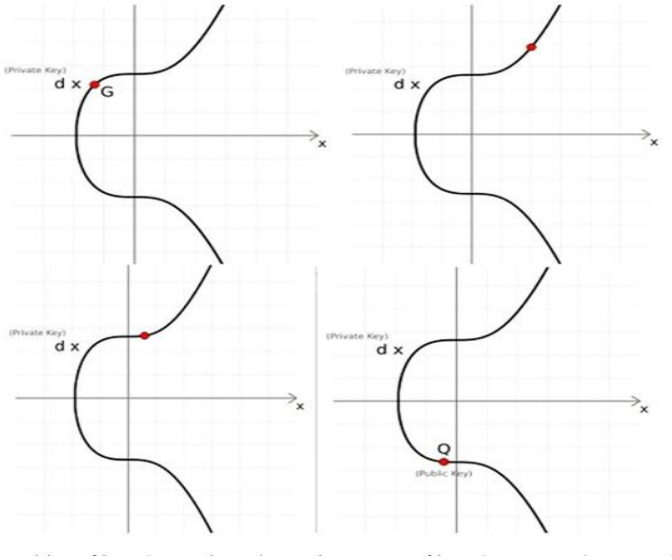


الگوریتم دو برابر کردن و اضافه کردن



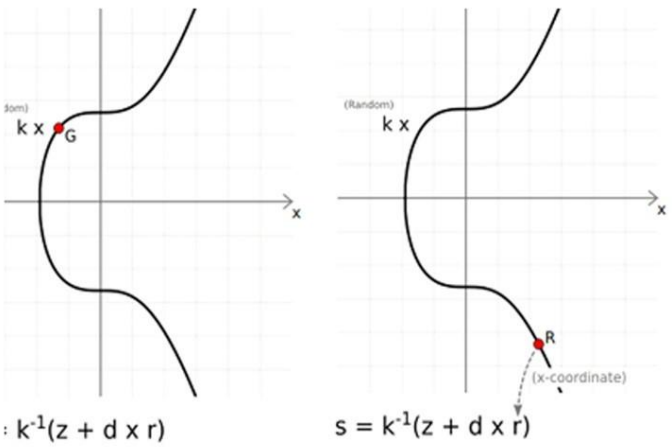
شکل ۳-۸: حاصل‌ضرب 3*p دقیقاً برابر است با یک عملیات double() به همراه یک عملیات add(): 3p = 2p + p

تولید جفت کلیدهای عمومی و خصوصی

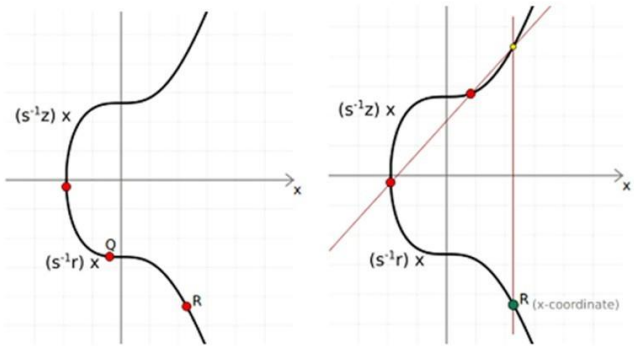


شکل ۳-۹: d کلید خصوصی است (یک عدد صحیح)، G نقطه‌ی تولیدکننده است (یک نقطه)، Q کلید عمومی است (یک نقطه)

امضا



شکل ۳-۱۰: بخش‌های امضا



شکل ۳-۱۱: بررسی معتبر بودن امضا

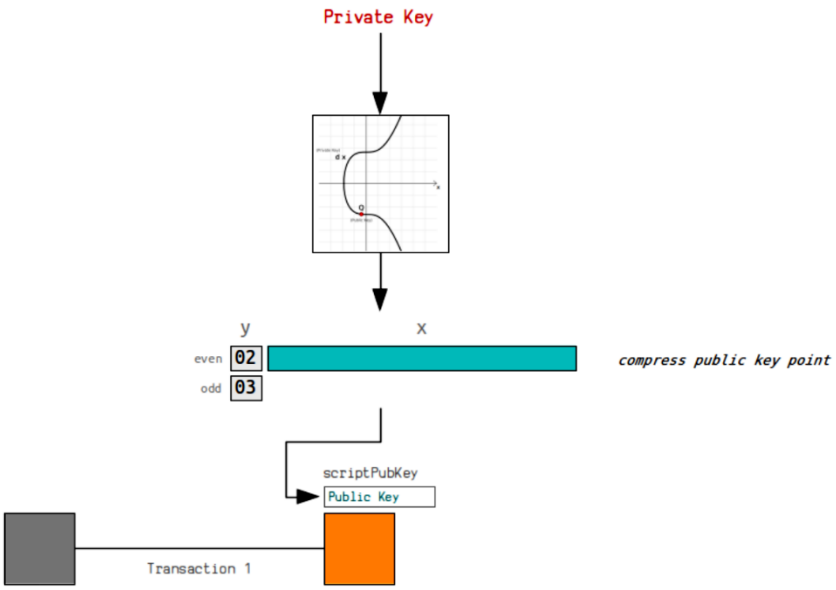
کد امضا و تائید آن

```
# Python 2.7.6 - Super simple Elliptic Curve Presentation.
No imported libraries, wrappers, nothing. # For educational
purposes only
# Below are the public specs for Bitcoin's curve - the
secp256k1
Pcurve = 2**256 - 2**32 - 2**9 - 2**8 - 2**7 - 2**6 - 2**4 -
1 # The proven prime
N=0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8C
D0364141 # Number of points in the field
Acurve = 0; Bcurve = 7 # This defines the curve. y^2 = x^3 +
Acurve * x + Bcurve
Gx =
55066263022277343669578718895168534326250603453777594175500
187360389116729240
Gy =
32670510020758816978083085130507043184471273380659243275938
904335757337482424
GPoint = (Gx,Gy) # This is our generator point. Tillions of
dif ones possible
#Individual Transaction/Personal Information
privKey =
75263518707598184987916378021939673586055614731957507592904
438851787542395619 #replace with any private key
RandNum =
28695618543805844332113829720373285210420739438570883203839
695181764147912 34 #replace with a truly random number
HashOfThingToSign =
86032112319101611046176971828093669637772856272773459297323
797145286374828050 # the hash of your message/transaction

def modinv(a,n=Pcurve): #Extended Euclidean
Algorithm/'division' in elliptic curves
    lm, hm = 1,0
```

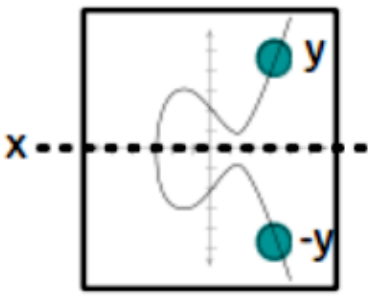
```
    low, high = a%n,n
    while low > 1:
        ratio = high/low
        nm, new = hm-lm*ratio, high-low*ratio
        lm, low, hm, high = nm, new, lm, low
    return lm % n

def ECadd(xp,yp,xq,yq): # Not true addition, invented for
EC. It adds Point-P with Point-Q.
    m = ((yq-yp) * modinv(xq-xp,Pcurve)) % Pcurve
    xr = (m*m-xp-xq) % Pcurve
    yr = (m*(xp-xr)-yp) % Pcurve
    return (xr,yr)

def ECdouble(xp,yp): # EC point doubling,  invented for EC.
It doubles Point-P.
    LamNumer = 3*xp*xp+Acurve
    LamDenom = 2*yp
    Lam = (LamNumer * modinv(LamDenom,Pcurve)) % Pcurve
    xr = (Lam*Lam-2*xp) % Pcurve
    yr = (Lam*(xp-xr)-yp) % Pcurve
    return (xr,yr)

def EccMultiply(xs,ys,Scalar): # Double & add. EC
Multiplication, Not true multiplication
    if Scalar == 0 or Scalar >= N: raise Exception("Invalid
Scalar/Private Key")
    ScalarBin = str(bin(Scalar))[2:]
    Qx,Qy=xs,ys
    for i in range (1, len(ScalarBin)): # This is invented
EC multiplication.
        Qx,Qy=ECdouble(Qx,Qy); # print "DUB", Qx; print
        if ScalarBin[i] == "1":
            Qx,Qy=ECadd(Qx,Qy,xs,ys); # print "ADD", Qx;
print
    return (Qx,Qy)

print; print "******* Public Key Generation *********"
xPublicKey, yPublicKey = EccMultiply(Gx,Gy,privKey)
print "the private key (in base 10 format):"; print privKey;
print
print "the uncompressed public key (starts with '04' & is
not the public address):"; print "04",xPublicKey,yPublicKey

print; print "******* Signature Generation *********"
xRandSignPoint, yRandSignPoint = EccMultiply(Gx,Gy,RandNum)
r = xRandSignPoint % N; print "r =", r
s = ((HashOfThingToSign + r*privKey)*(modinv(RandNum,N))) %
N; print "s =", s

print; print "******* Signature Verification *********>>"
w = modinv(s,N)
xu1, yu1 = EccMultiply(Gx,Gy,(HashOfThingToSign * w)%N)
xu2, yu2 = EccMultiply(xPublicKey,yPublicKey,(r*w)%N)
x,y = ECadd(xu1,yu1,xu2,yu2)
print r==x; print
```

الگوریتم ECDSA در بیت‌کوین

ایجاد یک کلید عمومی

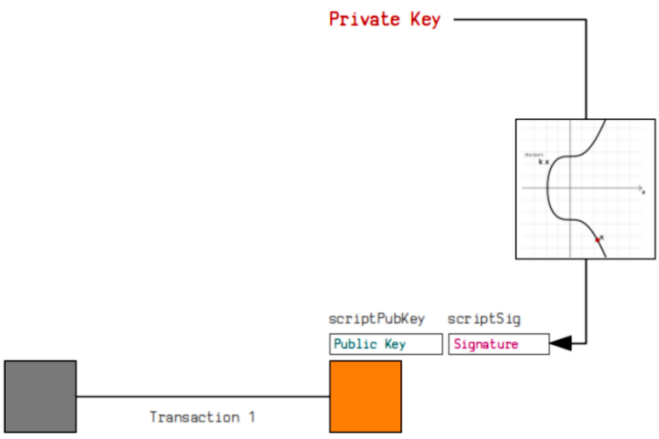


شکل ۳-۱۲: یک کلید عمومی را می‌توان در بالای اسکریپت قفل شده‌یک خروجی قرار داد.

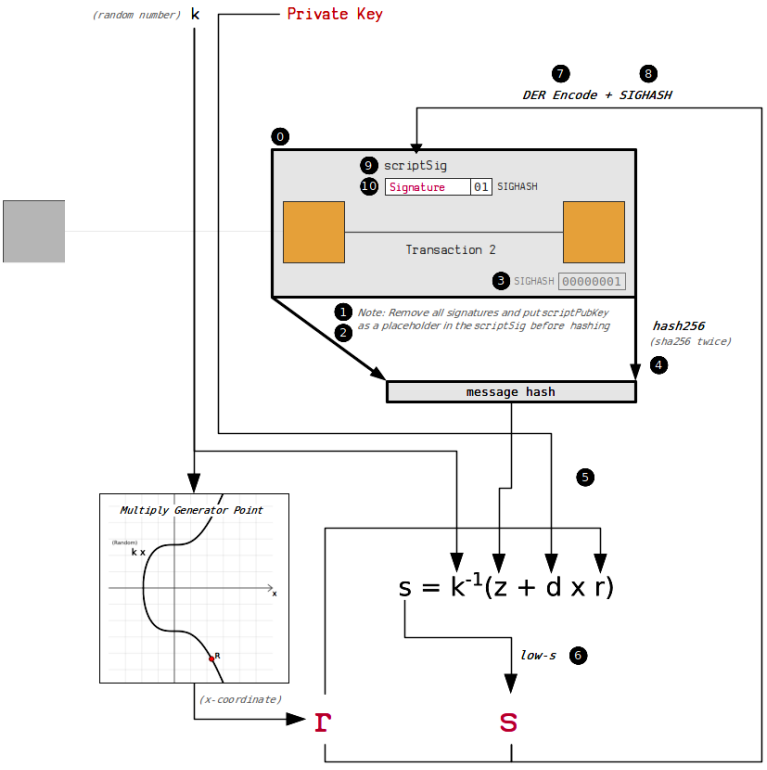


شکل ۳-۱۳: منحنی بیضی‌گون متقارن است.

امضا کردن یک تراکنش

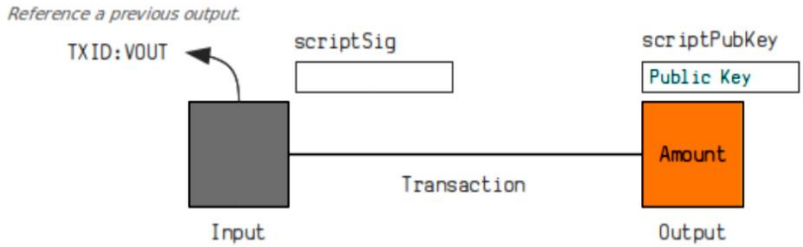


شکل ۳-۱٤: امضای یک تراکنش



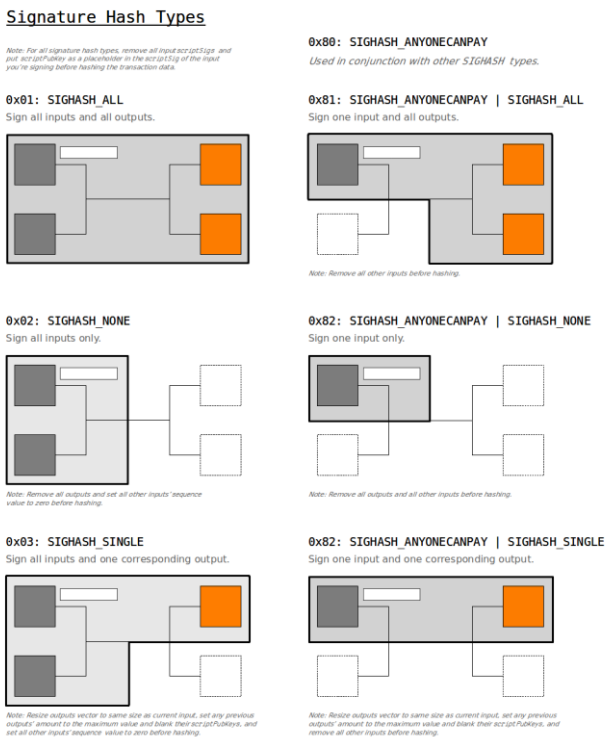
شکل ۳-۱۵ فرایند امضای یک تراکنش در بیت‌کوین

ایجاد یک تراکنش:



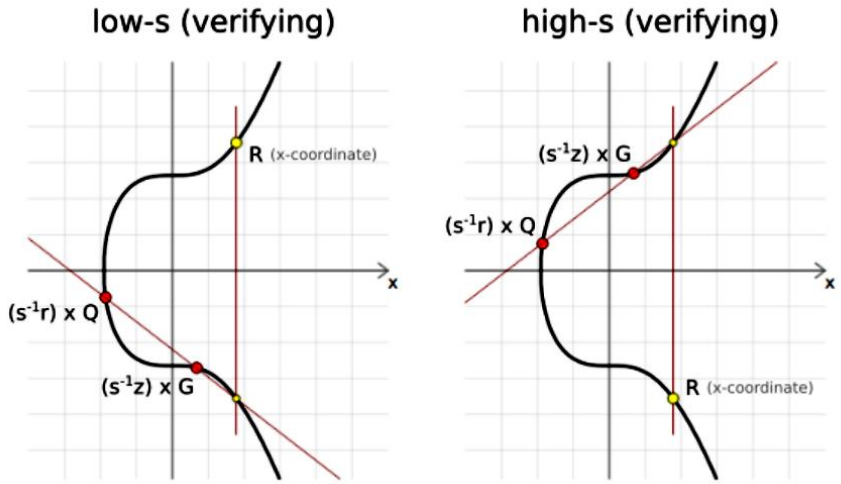
شکل ۳-۱٦: فرآیند ایجاد تراکنش

الصاق نوع هش امضا به داده‌های تراکنش:



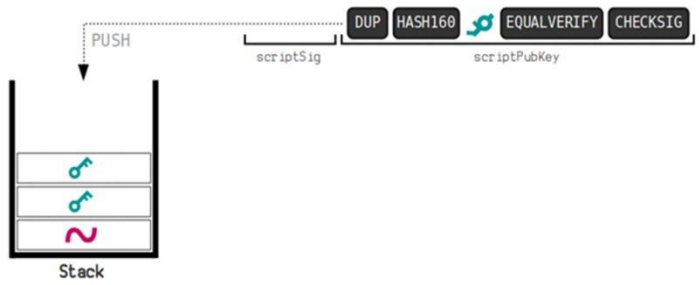
شکل ۳-۱۷: الصاق نوع هش به انتهای داده‌های تراکنش

استفاده از مقدار پایین s:



شکل ۳–۱۸: قرینه مقدار s، هنگام اعتبارسنجی نقاط مخالف منحنی را محاسبه می‌کند، اما مختصات x سومین نقطه، یکسان خواهد بود.

ایجاد تراکنش باز کردن قفل:



شکل ۳–۱۹: باز شدن قفل ورودی

کد پایتون تراکنش امضا شده مبتنی بر الگوریتم ECDSA

```
#!/usr/bin/env python3
import collections
import hashlib
import random

EllipticCurve = collections.namedtuple('EllipticCurve', 'name
p a b g n h')
curve = EllipticCurve(
    'secp256k1',
    # Field characteristic.
    p =
0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffff
fc2f,
    # Curve coefficients.
    a=0,
    b=7,
    # Base point.
    g =
(0x79be667ef9dcbbac55a06295ce870b07029bfcdb2dce28d959f2815b16f
81798,
0x483ada7726a3c4655da4fbfc0e1108a8fd17b448a68554199c47d08ffb10
d4b8),
    # Subgroup order.
    n =
0xfffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd036
4141,
    # Subgroup cofactor.
    h=1,
)
# Modular arithmetic
############################################################

def inverse_mod(k, p):
    """Returns the inverse of k modulo p.
    This function returns the only integer x such that (x * k)
% p == 1.
    k must be non-zero and p must be a prime.
    """
    if k == 0:
        raise ZeroDivisionError('division by zero')
    if k < 0:
        # k ** -1 = p - (-k) ** -1 (mod p)
        return p - inverse_mod (-k, p)
    # Extended Euclidean algorithm.
    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = p, k
    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t
    gcd, x, y = old_r, old_s, old_t
    assert gcd == 1
```

```
    assert (k * x) % p == 1
    return x % p

# Functions that work on curve points
########################################
def is_on_curve(point):
    """Returns True if the given point lies on the elliptic
curve."""
    if point is None:
        # None represents the point at infinity.
        return True
    x, y = point
    return (y * y - x * x * x - curve.a * x - curve.b) %
curve.p == 0

def point_neg(point):
    """Returns -point."""
    assert is_on_curve(point)
    if point is None:
        # -0 = 0
        return None
    x, y = point
    result = (x, -y % curve.p)
    assert is_on_curve(result)
    return result

def point_add(point1, point2):
    """Returns the result of point1 + point2 according to the
group law."""
    assert is_on_curve(point1)
    assert is_on_curve(point2)
    if point1 is None:
        # 0 + point2 = point2
        return point2
    if point2 is None:
        # point1 + 0 = point1
        return point1
    x1, y1 = point1
    x2, y2 = point2
    if x1 == x2 and y1 != y2:
        # point1 + (-point1) = 0
        return None
    if x1 == x2:
        # This is the case point1 == point2.
        m = (3 * x1 * x1 + curve.a) * inverse_mod(2 * y1,
curve.p)
    else:
        # This is the case point1 != point2.
        m = (y1 - y2) * inverse_mod(x1 - x2, curve.p)
    x3 = m * m - x1 - x2
    y3 = y1 + m * (x3 - x1)
    result = (x3 % curve.p,
              -y3 % curve.p)
    assert is_on_curve(result)
    return result
```

```
def scalar_mult(k, point):
    """Returns k * point computed using the double and
point_add algorithm."""
    assert is_on_curve(point)
    if k % curve.n == 0 or point is None:
        return None
    if k < 0:
        # k * point = -k * (-point)
        return scalar_mult(-k, point_neg(point))
    result = None
    addend = point
    while k:
        if k & 1:
            # Add.
            result = point_add(result, addend)
        # Double.
        addend = point_add(addend, addend)
        k >>= 1
    assert is_on_curve(result)
    return result

# Keypair generation and ECDSA
################################################
def make_keypair():
    """Generates a random private-public key pair."""
    private_key = random.randrange(1, curve.n)
    public_key = scalar_mult(private_key, curve.g)
    return private_key, public_key

def hash_message(message):
    """Returns the truncated SHA521 hash of the message."""
    message_hash = hashlib.sha512(message).digest()
    e = int.from_bytes(message_hash, 'big')
    # FIPS 180 says that when a hash needs to be truncated,
the rightmost bits
    # should be discarded.
    z = e >> (e.bit_length() - curve.n.bit_length())
    assert z.bit_length() <= curve.n.bit_length()
    return z

def sign_message(private_key, message):
    z = hash_message(message)
    r = 0
    s = 0
    while not r or not s:
        k = random.randrange(1, curve.n)
        x, y = scalar_mult(k, curve.g)
        r = x % curve.n
        s = ((z + r * private_key) * inverse_mod(k, curve.n))
% curve.n
    return (r, s)

def verify_signature(public_key, message, signature):
    z = hash_message(message)
    r, s = signature
    w = inverse_mod(s, curve.n)
```

```
    u1 = (z * w) % curve.n
    u2 = (r * w) % curve.n
    x, y = point_add(scalar_mult(u1, curve.g),
                     scalar_mult(u2, public_key))
    if (r % curve.n) == (x % curve.n):
        return 'signature matches'
    else:
        return 'invalid signature'
print('Curve:', curve.name)
private, public = make_keypair()
print("Private key:", hex(private))
print("Public key: (0x{:x}, 0x{:x})".format(*public))
msg = b'Hello!'
signature = sign_message(private, msg)
print()
print('Message:', msg)
print('Signature: (0x{:x}, 0x{:x})'.format(*signature))
print('Verification:', verify_signature(public, msg,
signature))
msg = b'Hi there!'
print()
print('Message:', msg)
print('Verification:', verify_signature(public, msg,
signature))
private, public = make_keypair()
msg = b'Hello!'
print()
print('Message:', msg)
print("Public key: (0x{:x}, 0x{:x})".format(*public))
print('Verification:', verify_signature(public, msg,
signature))
```

فصل چهارم

پیاپی‌سازی

قالب غیر فشرده‌ی SEC (استاندارد رمزنگاری کارآمد غیر فشرده)

```
047211a824f55b505228e4c3d5194c1fcfaa15a456abdf37f9b9d97a4040afc073dee6c8906498
4f03385237d92167c13e236446b417ab79a0fcae412ae3316b77

        - 04 - Marker
        - x coordinate - 32 bytes
        - y coordinate - 32 bytes
```

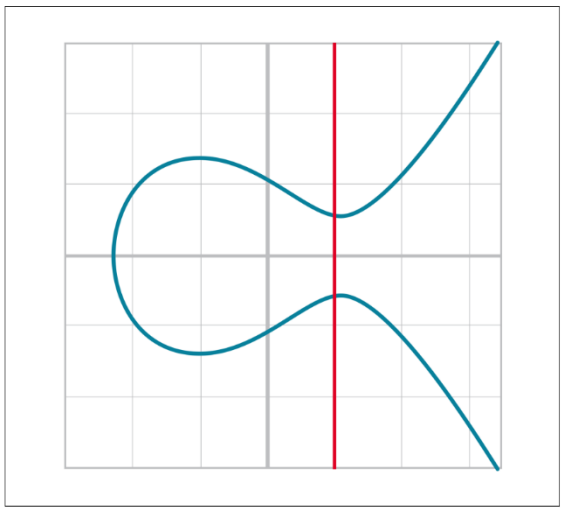
شکل ٤-١: قالب غیر فشرده‌ی SEC

پیاده‌سازی الگوریتم *پیاپی‌ساز* SEC غیر فشرده

```
class S256Point(Point):
...
    def sec(self):
        '''returns the binary version of the SEC format'''
 return b'\x04' + self.x.num.to_bytes(32, 'big') \
           + self.y.num.to_bytes(32, 'big')
```

1

قالب فشرده‌ی SEC (استاندارد رمزنگاری کارآمد فشرده)



شکل ٤-٢: دو مقدار ممکن برای y جایی است که این خط عمودی منحنی را قطع می‌کند

شکل ٤-٣: قالب فشرده‌ی کلید عمومی به روش SEC

کد پایتون *قالب فشرده‌ی کلید عمومی* به روش SEC

```
class S256Point(Point):
...
    def sec(self, compressed=True):
        '''returns the binary version of the SEC format'''
        if compressed:
            if self.y.num % 2 == 0:
                return b'\x02' + self.x.num.to_bytes(32,
'big')
            else:
                return b'\x03' + self.x.num.to_bytes(32,
'big')
        else:
            return b'\x04' + self.x.num.to_bytes(32, 'big')
+ \
                self.y.num.to_bytes(32, 'big')
```

کد پایتون کلاس S256Field

```
class S256Field(FieldElement):
...
    def sqrt(self):
        return self**((P + 1) // 4)
```

متد برای تجزیه[1] و مشخص کردن y

```
class S256Point:
...
    @classmethod
    def parse(self, sec_bin):
        '''returns a Point object from a SEC binary (not
hex)'''
        if sec_bin[0] == 4:
            x = int.from_bytes(sec_bin[1:33], 'big')
            y = int.from_bytes(sec_bin[33:65], 'big')
            return S256Point(x=x, y=y)
        is_even = sec_bin[0] == 2
        x = S256Field(int.from_bytes(sec_bin[1:], 'big'))
        # right side of the equation y^2 = x^3 + 7
        alpha = x**3 + S256Field(B)
        # solve for left side
        beta = alpha.sqrt()
        if beta.num % 2 == 0:
            even_beta = beta
            odd_beta = S256Field(P - beta.num)
        else:
            even_beta = S256Field(P - beta.num)
            odd_beta = beta
        if is_even:
            return S256Point(x, even_beta)
        else:
            return S256Point(x, odd_beta)
```

1

2

3

4

امضاهای DER (قوانین کدنگاری متمایز)

```
3045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf213
20b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d7f67801
c31967743a9c8e10615bed

        - 30 - Marker
        - 45 - Length of sig
        - 02 - Marker for r value
        - 21 - r value lenth
        - 00ed...8f - r value
        - 02 - Marker for s value
        - 20 - s value length
        - 7a98...ed - s value
```

شکل ٤-٤: قالب DER

[1] Parse

کد پایتون کلاس پیاپی‌ساز امضا

```
class Signature:
...
    def der(self):
        rbin = self.r.to_bytes(32, byteorder='big')
        # remove all null bytes at the beginning
        rbin = rbin.lstrip(b'\x00')
        # if rbin has a high bit, add a \x00
        if rbin[0] & 0x80:
            rbin = b'\x00' + rbin
        result = bytes([2, len(rbin)]) + rbin
        sbin = self.s.to_bytes(32, byteorder='big')
        # remove all null bytes at the beginning
        sbin = sbin.lstrip(b'\x00')
        # if sbin has a high bit, add a \x00
        if sbin[0] & 0x80:
            sbin = b'\x00' + sbin
        result += bytes([2, len(sbin)]) + sbin
        return bytes([0x30, len(result)]) + result
```

1

انتقال کلید عمومی

کد پایتون کدنگاری اعداد در مبنای ۵۸

```
BASE58_ALPHABET =
'123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'
...
def encode_base58(s):
    count = 0
    for c in s:
        if c == 0:
            count += 1
        else:
            break
    num = int.from_bytes(s, 'big')
    prefix = '1' * count
    result = ''
    while num > 0:
        num, mod = divmod(num, 58)
        result = BASE58_ALPHABET[mod] + result
    return prefix + result
```

1

2

3

قالب آدرس

```
def encode_base58_checksum(b):
    return encode_base58(b + hash256(b)[:4])
```

الگوریتم hash160 در پایتون (helper.py)

```
def hash160(s):
    '''sha256 followed by ripemd160'''
    return hashlib.new('ripemd160',
hashlib.sha256(s).digest()).digest()
```

1

به‌روزرسانی کلاس S256Point با hash160

```
class S256Point:
...
    def hash160(self, compressed=True):
        return hash160(self.sec(compressed))

    def address(self, compressed=True, testnet=False):
        '''Returns the address string'''
        h160 = self.hash160(compressed)
        if testnet:
            prefix = b'\x6f'
        else:
            prefix = b'\x00'
        return encode_base58_checksum(prefix + h160)
```

قالب واردکردن کیف پول[1]

کد کلاس کلید خصوصی به کمک پایتون

```
class PrivateKey
...
    def wif(self, compressed=True, testnet=False):
        secret_bytes = self.secret.to_bytes(32, 'big')
        if testnet:
            prefix = b'\xef'
        else:
            prefix = b'\x80'
        if compressed:
            suffix = b'\x01'
        else:
            suffix = b''
        return encode_base58_checksum(prefix + secret_bytes
+ suffix)
```

[1] Wallet Import Format: WIF

فصل پنجم

تراکنش‌ها

اجزای تراکنش

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۱: نمایش هگزادسیمال تراکنش معمولی که هر بخش با رنگ متفاوتی نشان داده شده است

کلاس تراکنش (Tx)

```
class Tx:

    def __init__(self, version, tx_ins, tx_outs, locktime,
testnet=False):
        self.version = version
        self.tx_ins = tx_ins
        self.tx_outs = tx_outs
        self.locktime = locktime
        self.testnet = testnet

    def __repr__(self):
        tx_ins = ''
        for tx_in in self.tx_ins:
            tx_ins += tx_in.__repr__() + '\n'
        tx_outs = ''
        for tx_out in self.tx_outs:
            tx_outs += tx_out.__repr__() + '\n'
        return 'tx: {}\nversion:
{}\ntx_ins:\n{}tx_outs:\n{}locktime: {}'.format(
            self.id(),
            self.version,
            tx_ins,
            tx_outs,
            self.locktime,
        )

    def id(self):
        '''Human-readable hexadecimal of the transaction
hash'''
        return self.hash().hex()

    def hash(self):
        '''Binary hash of the legacy serialization'''
        return hash256(self.serialize())[::-1]
```

1

2

3

4

کد تجزیه تراکنش

```
class Tx:
    ...
    @classmethod
    def parse(cls, serialization):
        version = serialization[0:4]
        ...
```

1

2

کد اصلی تجزیه‌ی تراکنش

```
class Tx:
    ...
    @classmethod
    def parse(cls, stream):
        serialized_version = stream.read(4)
 ...
```

1

نسخه

```
01000000 01813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۲: نسخه‌ی تراکنش

ورودی‌ها

```
01000000 01813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278a feffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۳: ورودی تراکنش

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ٥-٤: تعداد ورودی‌های یک تراکنش

کد پایتون روش *varints*

```
def read_varint(s):
    '''read_varint reads a variable integer from a stream'''
    i = s.read(1)[0]
    if i == 0xfd:
        # 0xfd means the next two bytes are the number
        return little_endian_to_int(s.read(2))
    elif i == 0xfe:
        # 0xfe means the next four bytes are the number
        return little_endian_to_int(s.read(4))
    elif i == 0xff:
        # 0xff means the next eight bytes are the number
        return little_endian_to_int(s.read(8))
    else:
        # anything else is just the integer
        return i

def encode_varint(i):
    '''encodes an integer as a varint'''
    if i < 0xfd:
        return bytes([i])
    elif i < 0x10000:
        return b'\xfd' + int_to_little_endian(i, 2)
    elif i < 0x100000000:
        return b'\xfe' + int_to_little_endian(i, 4)
    elif i < 0x10000000000000000:
        return b'\xff' + int_to_little_endian(i, 8)
    else:
        raise ValueError('integer too large: {}'.format(i))
```

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۵: بخش‌های مختلف یک ورودی تراکنش

کلاس TxIn:

```
class TxIn:
    def __init__(self, prev_tx, prev_index, script_sig=None,
sequence=0xffffffff):
        self.prev_tx = prev_tx
        self.prev_index = prev_index
        if script_sig is None:
            self.script_sig = Script()
        else:
            self.script_sig = script_sig
        self.sequence = sequence

    def __repr__(self):
        return '{}:{}'.format(
            self.prev_tx.hex(),
            self.prev_index,
        )
```

تجزیه‌ی اسکریپت

```
>>> from io import BytesIO
>>> from script import Script
>>> script_hex =
('6b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031cc
f\
cf21320b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d7f
67801c31967743a9c8\
e10615bed01210349fc4e631e3624a545de3f89f5d8684c7b8138bd94bdd
531d2e213bf016b278\
a')
>>> stream = BytesIO(bytes.fromhex(script_hex))
>>> script_sig = Script.parse(stream)
>>> print(script_sig)
3045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf213
20b0277457c98f0220\
7a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e1061
5bed01 0349fc4e631\
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278a
```

1

خروجی‌ها

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۶: تعداد خروجی‌های یک تراکنش معمولی

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۷: خروجی یک تراکنش در سیستم عددی هگزادسیمال

کد پایتون کلاس TxOut:

```
class TxOut:

    def __init__(self, amount, script_pubkey):
        self.amount = amount
        self.script_pubkey = script_pubkey

    def __repr__(self):
        return '{}:{}'.format(self.amount,
self.script_pubkey)
```

قفل زمانی

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c39800000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۵-۸: قفل زمانی

کد کردن تراکنش‌ها

```
class TxOut:
...
    def serialize(self):
        '''Returns the byte serialization of the transaction
output'''
        result = int_to_little_endian(self.amount, 8)
        result += self.script_pubkey.serialize()
        return result
```

1

کد کلاس TxIn:

```
class TxIn:
...
    def serialize(self):
        '''Returns the byte serialization of the transaction
input'''
        result = self.prev_tx[::-1]
        result += int_to_little_endian(self.prev_index, 4)
        result += self.script_sig.serialize()
        result += int_to_little_endian(self.sequence, 4)
        return result
```

کلاس Tx:

```
class Tx:
...
    def serialize(self):
        '''Returns the byte serialization of the
transaction'''
```

```
        result = int_to_little_endian(self.version, 4)
        result += encode_varint(len(self.tx_ins))
        for tx_in in self.tx_ins:
            result += tx_in.serialize()
        result += encode_varint(len(self.tx_outs))
        for tx_out in self.tx_outs:
            result += tx_out.serialize()
        result += int_to_little_endian(self.locktime, 4)
        return result
```

کارمزد تراکنش

:TXfetcher

```
class TxFetcher:
    cache = {}
    @classmethod
    def get_url(cls, testnet=False):
        if testnet:
            return 'http://testnet.programmingbitcoin.com'
        else:
            return 'http://mainnet.programmingbitcoin.com'

    @classmethod
    def fetch(cls, tx_id, testnet=False, fresh=False):
        if fresh or (tx_id not in cls.cache):
            url =
'{}/tx/{}.hex'.format(cls.get_url(testnet), tx_id)
            response = requests.get(url)
            try:
                raw = bytes.fromhex(response.text.strip())
            except ValueError:
                raise ValueError('unexpected response:
{}'.format(response.text))
            if raw[4] == 0:
                raw = raw[:4] + raw[6:]
                tx = Tx.parse(BytesIO(raw), testnet=testnet)
                tx.locktime = little_endian_to_int(raw[-4:])
            else:
                tx = Tx.parse(BytesIO(raw), testnet=testnet)
            if tx.id() != tx_id:
                raise ValueError('not the same id: {} vs
{}'.format(tx.id(), tx_id))
            cls.cache[tx_id] = tx
        cls.cache[tx_id].testnet = testnet
        return cls.cache[tx_id]
```

1

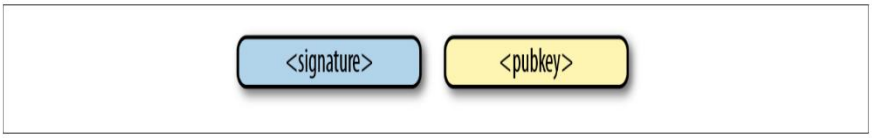
کلاس TXIn:

```
class TxIn:
...
    def fetch_tx(self, testnet=False):
        return TxFetcher.fetch(self.prev_tx.hex(),
testnet=testnet)

    def value(self, testnet=False):
        '''Get the output value by looking up the tx hash.
        Returns the amount in satoshi.
        '''
        tx = self.fetch_tx(testnet=testnet)
        return tx.tx_outs[self.prev_index].amount
    def script_pubkey(self, testnet=False):
        '''Get the ScriptPubKey by looking up the tx hash.
        Returns a Script object.
        '''
        tx = self.fetch_tx(testnet=testnet)
        return tx.tx_outs[self.prev_index].script_pubkey
```

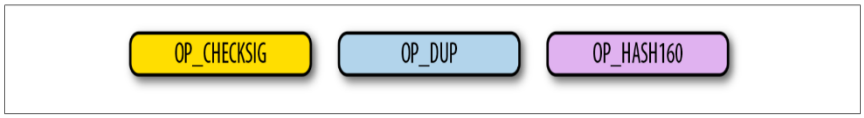
فصل ششم

Script

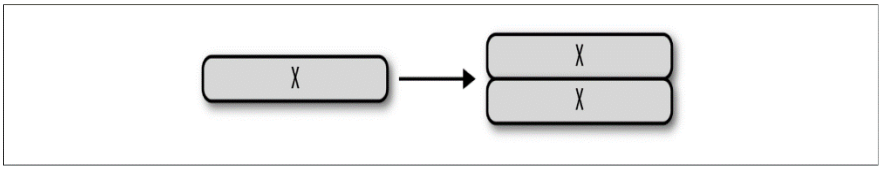
اسکریپت چگونه کار می‌کند؟



شکل ٦-١: عناصر

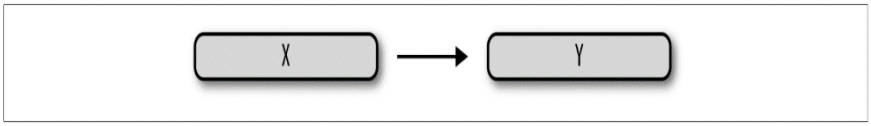


شکل ٦-٢: عملکردها

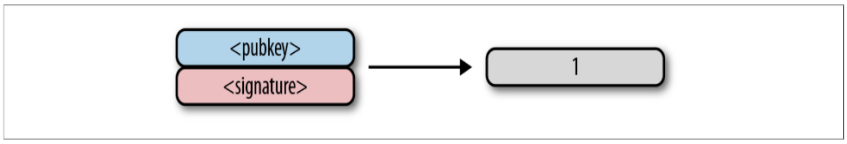


شکل ٦-٣: OP_DUP عنصر بالایی را کپی می‌کند.

مثال‌هایی از عملیات‌ها



شکل ٦-٤: OP_HASH160 یک sha256 به همراه ripemd160 (با نام مستعار hash160) روی عنصر بالایی پشته انجام می‌دهد.



شکل ٦-٥: OP_CHECKSIG بررسی می‌کند آیا امضای pubkey معتبر است یا خیر.

کدگذاری Opcodeها

```
def op_dup(stack):                          1
    if len(stack) < 1:
        return False
    stack.append(stack[-1])                 2
    return True
...
OP_CODE_FUNCTIONS = {
...
    118: op_dup,                            3
...
}
```

```
def op_hash256(stack):
    if len(stack) < 1:
        return False
    element = stack.pop()
    stack.append(hash256(element))
    return True
...
OP_CODE_FUNCTIONS = {
...
    170: op_hash256,
...
}
```

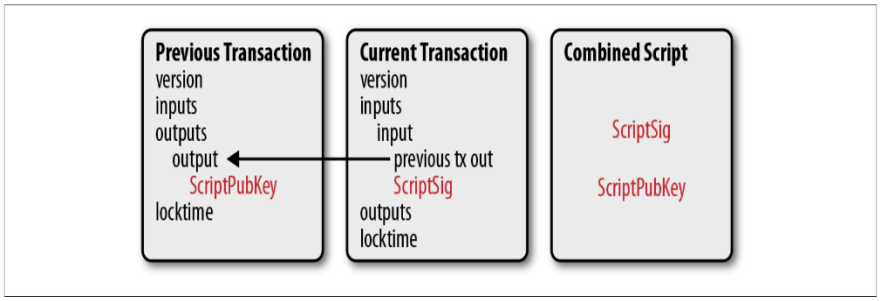
کد نویسی یک تجزیه‌کننده و پیاپی ساز اسکریپت

```
class Script:

    def __init__(self, cmds=None):
        if cmds is None:
            self.cmds = []
        else:
            self.cmds = cmds
    ...
    @classmethod
    def parse(cls, s):
        length = read_varint(s)
        cmds = []
        count = 0
        while count < length:
            current = s.read(1)
            count += 1
            current_byte = current[0]
            if current_byte >= 1 and current_byte <= 75:
                n = current_byte
                cmds.append(s.read(n))
                count += n
            elif current_byte == 76:
                data_length =
little_endian_to_int(s.read(1))
                cmds.append(s.read(data_length))
                count += data_length + 1
            elif current_byte == 77:
                data_length =
little_endian_to_int(s.read(2))
                cmds.append(s.read(data_length))
                count += data_length + 2
            else:
                op_code = current_byte
                cmds.append(op_code)
        if count != length:
            raise SyntaxError('parsing script failed')
        return cls(cmds)
```

1 2 3 4 5 6 7 8 9 10

```
class Script:
...
    def raw_serialize(self):
        result = b''
        for cmd in self.cmds:
            if type(cmd) == int:                                    (1)
                result += int_to_little_endian(cmd, 1)
            else:
                length = len(cmd)
                if length < 75:                                     (2)
                    result += int_to_little_endian(length,      (3)
1)
                elif length > 75 and length < 0x100:
                    result += int_to_little_endian(76, 1)
                    result += int_to_little_endian(length,      (4)
1)
                elif length >= 0x100 and length <= 520:
                    result += int_to_little_endian(77, 1)
                    result += int_to_little_endian(length,      (5)
2)
                else:
                    raise ValueError('too long an cmd')
                result += cmd
        return result

    def serialize(self):
        result = self.raw_serialize()
        total = len(result)
        return encode_varint(total) + result                        (6)
```

ترکیب کردن بخش‌های اسکریپت



شکل ٦-٦: ترکیب ScriptPubKey و ScriptSig

کد کردن مجموعه دستورالعمل‌های ترکیبی

```
class Script:
...
    def __add__(self, other):
        return Script(self.cmds + other.cmds)
```

1

p2pk

```
410411db93e1dcdb8a016b49840f8c53bc1eb68a382e97b1482ecad7b148a6909a5cb2e0eaddfb84c
cf9744464f82e160bfa9b8b64f9d4c03f999b8643f656b412a3ac
```

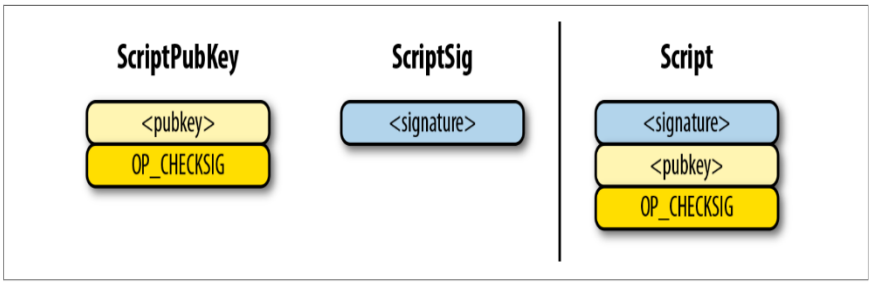
- 41 - length of pubkey
- 0411...a3 - <pubkey>
- ac - OP_CHECKSIG

شکل ۶-۷: Pay-to-pubkey (p2pk) ScriptPubKey

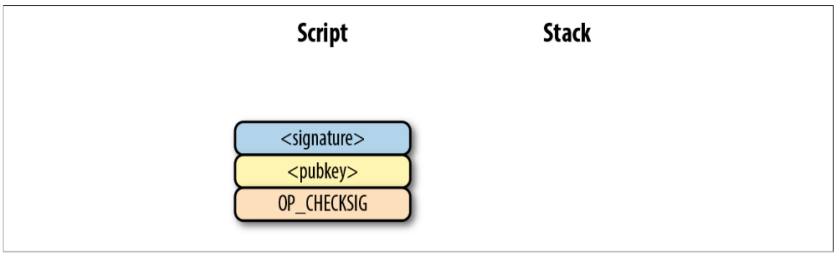
```
47304402204e45e16932b8af514961a1d3a1a25fdf3f4f7732e9d624c6c61548ab5fb8cd410220181
522ec8eca07de4860a4acdd12909d831cc56cbbac4622082221a8768d1d0901
```

- 47 - length of signature
- 3044...01 - <signature>

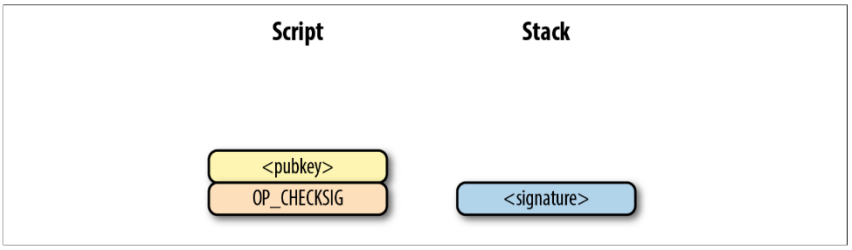
شکل ۶-۸: Pay-to-pubkey (p2pk) ScriptSig



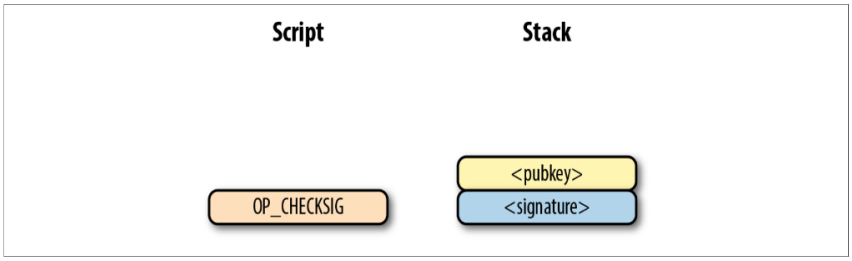
شکل ٦-٩: p2pk ترکیب‌شده



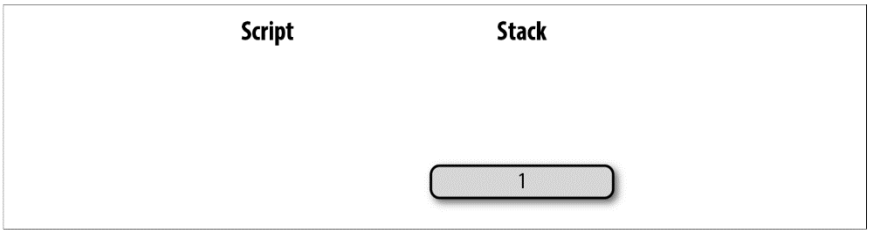
شکل ٦-١٠: شروع p2pk



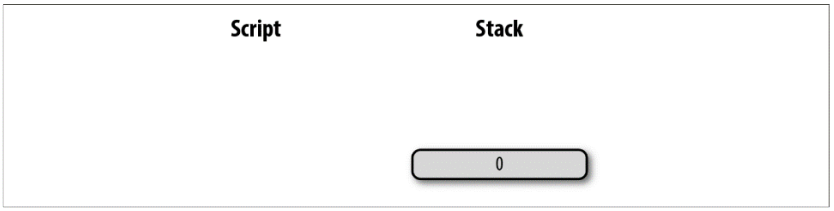
شکل ٦-١١: مرحله‌ی اول p2pk



شکل ٦-١٢: مرحله‌ی دوم p2pk



شکل ٦-١٣: مرحله‌ی سوم p2pk



شکل ٦-١٤: پایان p2pk

کد کردن ارزیابی اسکریپت

```
>>> from script import Script
>>> z =
0x7c076ff316692a3d7eb3c3bb0f8b1488cf72e1afcd929e29307032997a
838a3d
>>> sec =
bytes.fromhex('04887387e452b8eacc4acfde10d9aaf7f6d9a0f975aab
b10d006e\
4da568744d06c61de6d95231cd89026e286df3b6ae4a894a3378e393e93a
0f45b666329a0ae34')
>>> sig =
bytes.fromhex('3045022000eff69ef2b1bd93a66ed5219add4fb51e11a
840f4048\
```

```
76325a1e8ffe0529a2c022100c7207fee197d27c618aea621406f6bf5ef6
fca38681d82b2f06fd\
dbdce6feab601')
>>> script_pubkey = Script([sec, 0xac])
>>> script_sig = Script([sig])
>>> combined_script = script_sig + script_pubkey
>>> print(combined_script.evaluate(z))
True
```

1 2 3

متدی مجموعه دستورات ترکیبی

```
from op import OP_CODE_FUNCTIONS, OP_CODE_NAMES
...
class Script:
...
    def evaluate(self, z):
        cmds = self.cmds[:]
        stack = []
        altstack = []
        while len(cmds) > 0:
            cmd = cmds.pop(0)
            if type(cmd) == int:
                operation = OP_CODE_FUNCTIONS[cmd]
                if cmd in (99, 100):
                    if not operation(stack, cmds):
                        LOGGER.info('bad op:
{}'.format(OP_CODE_NAMES[cmd]))
                        return False
                elif cmd in (107, 108):
                    if not operation(stack, altstack):
                        LOGGER.info('bad op:
{}'.format(OP_CODE_NAMES[cmd]))
                        return False
                elif cmd in (172, 173, 174, 175):
                    if not operation(stack, z):
                        LOGGER.info('bad op:
{}'.format(OP_CODE_NAMES[cmd]))
                        return False
                else:
                    if not operation(stack):
                        LOGGER.info('bad op:
{}'.format(OP_CODE_NAMES[cmd]))
                        return False
            else:
                stack.append(cmd)
        if len(stack) == 0:
            return False
        if stack.pop() == b'':
            return False
        return True
```

1 2 3 4 5 6 7 8 9 10

تشریح کامل عناصر پشته

```
def encode_num(num):
    if num == 0:
        return b''
    abs_num = abs(num)
    negative = num < 0
    result = bytearray()
    while abs_num:
        result.append(abs_num & 0xff)
        abs_num >>= 8
    if result[-1] & 0x80:
        if negative:
            result.append(0x80)
        else:
            result.append(0)
    elif negative:
        result[-1] |= 0x80
    return bytes(result)

def decode_num(element):
    if element == b'':
        return 0
    big_endian = element[::-1]
    if big_endian[0] & 0x80:
        negative = True
        result = big_endian[0] & 0x7f
    else:
        negative = False
        result = big_endian[0]
    for c in big_endian[1:]:
        result <<= 8
        result += c
    if negative:
        return -result
    else:
        return result

def op_0(stack):
    stack.append(encode_num(0))
    return True
```

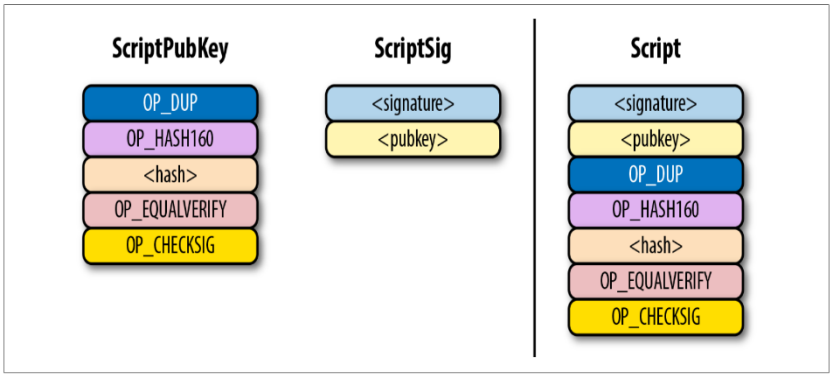
p2pkh

```
76a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac

- 76 - OP_DUP
- a9 - OP_HASH160
- 14 - Length of <hash>
- bc3b...da - <hash>
- 88 - OP_EQUALVERIFY
- ac - OP_CHECKSIG
```

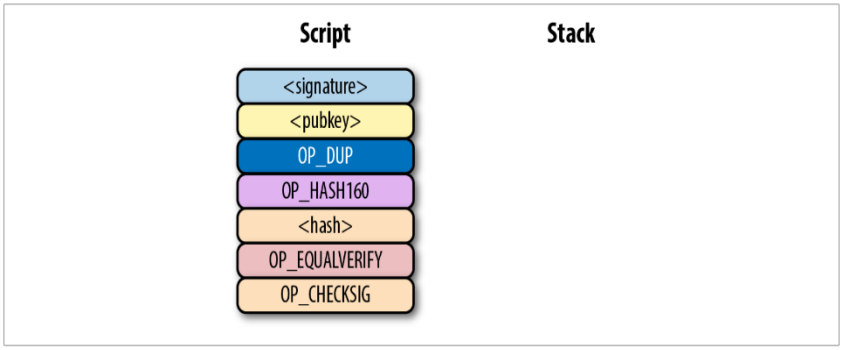
شکل ۶–۱۵: Pay-to-pubkey-hash (p2pkh) ScriptPubKey

```
483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf2
1320b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d7f678
01c31967743a9c8e10615bed01210349fc4e631e3624a545de3f89f5d868
4c7b8138bd94bdd531d2e213bf016b278a

- 48 - Length of <signature>
- 30...01 - <signature>
- 21 - Length of <pubkey>
- 0349...8a - <pubkey>
```

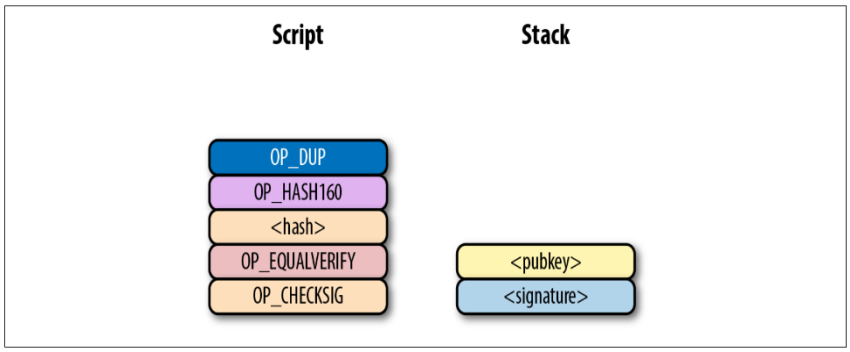
شکل ۶–۱۶: Pay-to-pubkey-hash (p2pkh) ScriptSig



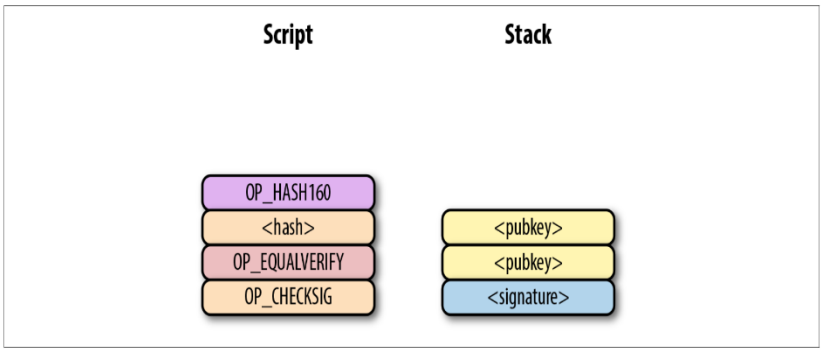
شکل ۶–۱۷: p2pkh ترکیب‌شده



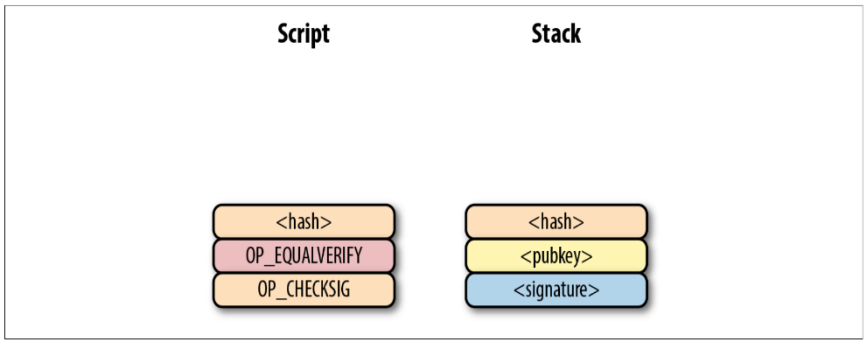
شکل ٦–١٨: شروع p2pkh



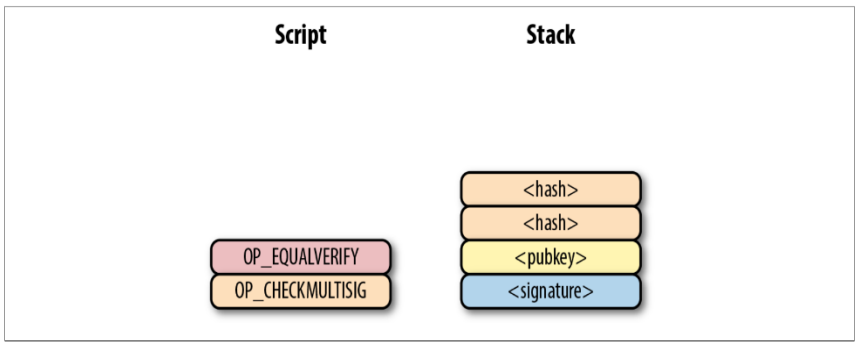
شکل ٦–١٩: مرحله‌ی اول p2pkh



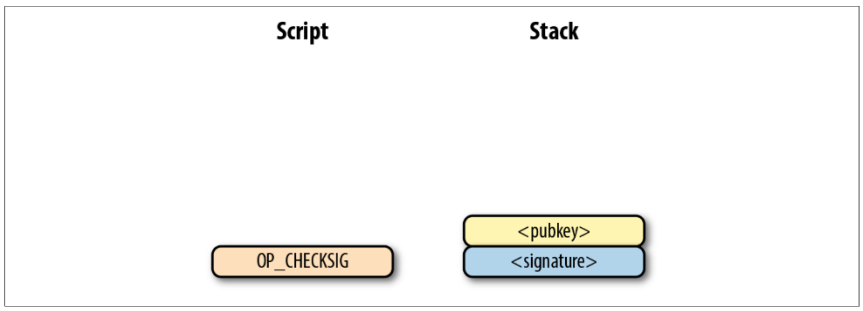
شکل ٦–٢٠: مرحله‌ی دوم p2pkh



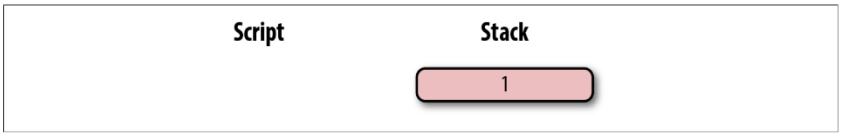
شکل ٦-٢١: مرحله‌ی سوم p2pkh



شکل ٦-٢٢: مرحله‌ی چهارم p2pkh



شکل ٦-٢٣: مرحله‌ی پنجم p2pkh



شکل ٦-٢٤: پایان p2pkh

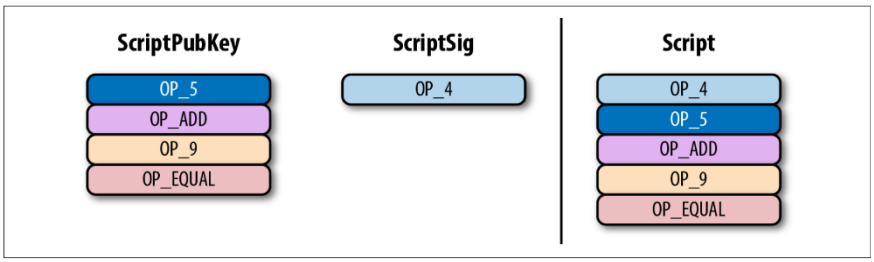
اسکریپت‌ها می‌توانند قراردادی باشند

```
55935987

55 - OP_5
93 - OP_ADD
59 - OP_9
87 - OP_EQUAL
```

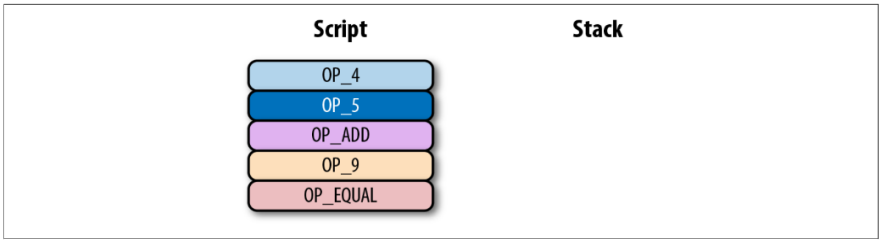
شکل ٦-٢٥: مثالی از ScriptPubKey

```
54

54 - OP_4
```

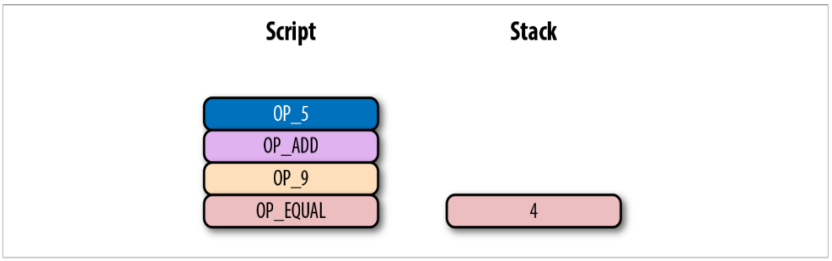
شکل ٦-٢٦: مثالی از ScriptSig



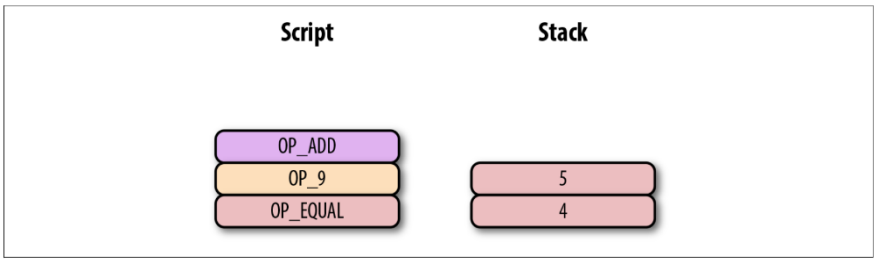
شکل ٦-٢٧: مثال ترکیب‌شده



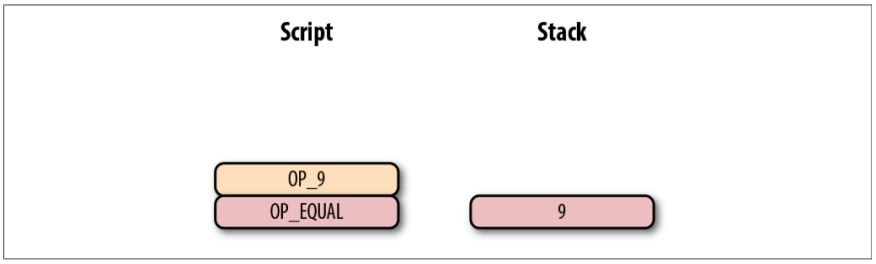
شکل ۶-۲۸: شروع مثال



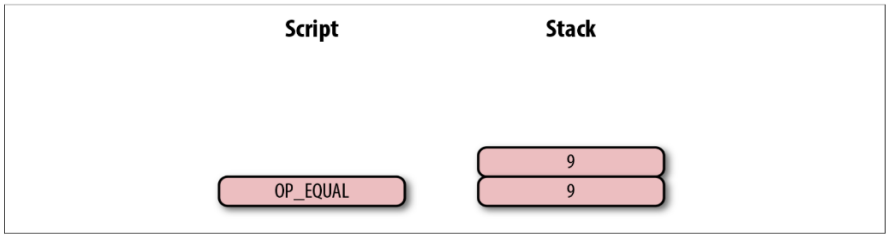
شکل ۶-۲۹: مرحله‌ی اول مثال



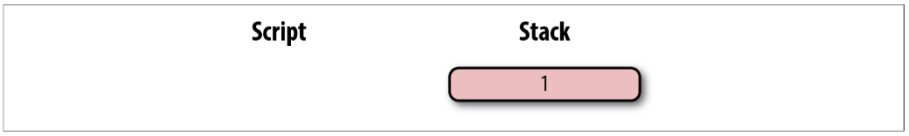
شکل ۶-۳۰: مرحله‌ی دوم مثال



شکل ۶-۳۱: مرحله‌ی سوم مثال



شکل ٦-٣٢: مرحله‌ی چهارم مثال



شکل ٦-٣٣: پایان مثال

فصل هفتم

ایجاد و اعتبارسنجی تراکنش‌ها

بررسی مجموع ورودی‌ها در مقابل مجموع خروجی‌ها

شبه کد کلاس fee در کد زیر مشاهده می‌شود:

```
class Tx:
    ...
    def fee(self):
        '''Returns the fee of this transaction in satoshi'''
        input_sum, output_sum = 0, 0
        for tx_in in self.tx_ins:
            input_sum += tx_in.value(self.testnet)
        for tx_out in self.tx_outs:
            output_sum += tx_out.amount
        return input_sum - output_sum
```

```
>>> from tx import Tx
>>> from io import BytesIO
>>> raw_tx =
('0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71
bf830\
3c6a989c7d1000000006b483045022100ed81ff192e75a3fd2304004dcad
b746fa5e24c5031ccf\
cf21320b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d7f
67801c31967743a9c8\
e10615bed01210349fc4e631e3624a545de3f89f5d8684c7b8138bd94bdd
531d2e213bf016b278\
afeffffff02a135ef01000000001976a914bc3b654dca7e56b04dca18f25
66cdaf02e8d9ada88a\
c99c39800000000001976a9141c4bc762dd5423e332166702cb75f40df79
fea1288ac19430600')
>>> stream = BytesIO(bytes.fromhex(raw_tx))
>>> transaction = Tx.parse(stream)
>>> print(transaction.fee() >= 0)
True
```

1

بررسیِ امضا

```
>>> from ecc import S256Point, Signature
>>> sec =
bytes.fromhex('0349fc4e631e3624a545de3f89f5d8684c7b8138bd94b
dd531d2e\
213bf016b278a')
>>> der =
bytes.fromhex('3045022100ed81ff192e75a3fd2304004dcadb746fa5e
24c5031c\
cfcf21320b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d
7f67801c31967743a9\
```

```
c8e10615bed')
>>> z =
0x27e0c5994dec7824e56dec6b2fcb342eb7cdb0d0957c2fce9882f715e8
5d81a6
>>> point = S256Point.parse(sec)
>>> signature = Signature.parse(der)
>>> print(point.verify(z, signature))
True
```

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
06b483045022100ed81ff192e75a3fd2304004dcadb746fa5e24c5031ccfcf21320b0277457c98f02
207a986d955c6e0cb35d446a89d3f56100f4d7f67801c31967743a9c8e10615bed01210349fc4e631
e3624a545de3f89f5d8684c7b8138bd94bdd531d2e213bf016b278afeffffff02a135ef0100000000
1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c398000000000001976a9141c4bc
762dd5423e332166702cb75f40df79fea1288ac19430600
```

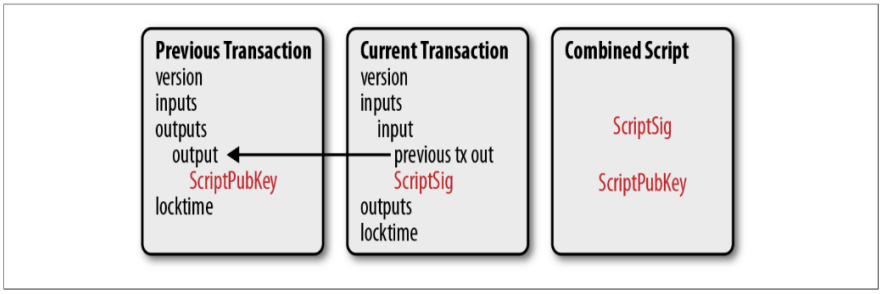
شکل ۷-۱: امضا در بخش مشخص شده‌ی زرد رنگ در سیستم هگزادسیمال قرار دارد

گام اول: خالی کردن تمام Scriptsigها

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
000feffffff02a135ef0100000000 1976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac
99c398000000000001976a9141c4bc762dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۷-۲: صفر کردن ورودی‌های ScriptSig

گام دوم ۲: جایگزینیِ ScriptSig ورودی امضاشده با ScriptPubKey قبلی



شکل ۷-۳: ترکیب کردن scriptPubKey وscriptsig

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
01976a914a802fc56c704ce87c42d7c92eb75e7896bdc41ae88acfeffffff02a135ef0100000000019
76a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c3980000000000001976a9141c4bc76
2dd5423e332166702cb75f40df79fea1288ac19430600
```

شکل ۷-٤: جایگزینی ScriptSig برای یکی از ورودی‌ها با ScriptPubKey تراکنش قبلی

گام سوم: اضافه کردن متغیر نوع hash

```
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf8303c6a989c7d10000000
01976a914a802fc56c704ce87c42d7c92eb75e7896bdc41ae88acfeffffff02a135ef0100000000019
76a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88ac99c3980000000000001976a9141c4bc76
2dd5423e332166702cb75f40df79fea1288ac1943060001000000
```

شکل ۷-۵: اتصال نوع تابع هش (رنگ قهوه‌ای)

شبه کدِ تراکنشِ تغییریافته به z:

```
>>> from helper import hash256
>>> modified_tx =
bytes.fromhex('0100000001813f79011acb80925dfe69b3def355fe914
\
```

```
bd1d96a3f5f71bf8303c6a989c7d1000000001976a914a802fc56c704ce8
7c42d7c92eb75e7896\
bdc41ae88acfeffffff02a135ef01000000001976a914bc3b654dca7e56b
04dca18f2566cdaf02\
e8d9ada88ac99c39800000000001976a9141c4bc762dd5423e332166702c
b75f40df79fea1288a\
c1943060001000000')
>>> h256 = hash256(modified_tx)
>>> z = int.from_bytes(h256, 'big')
>>> print(hex(z))
0x27e0c5994dec7824e56dec6b2fcb342eb7cdb0d0957c2fce9882f715e8
5d81a6
```

```
>>> from ecc import S256Point, Signature
>>> sec =
bytes.fromhex('0349fc4e631e3624a545de3f89f5d8684c7b8138bd94b
dd531d2e\
213bf016b278a')
>>> der =
bytes.fromhex('3045022100ed81ff192e75a3fd2304004dcadb746fa5e
24c5031c\
cfcf21320b0277457c98f02207a986d955c6e0cb35d446a89d3f56100f4d
7f67801c31967743a9\
c8e10615bed')
>>> z =
0x27e0c5994dec7824e56dec6b2fcb342eb7cdb0d0957c2fce9882f715e8
5d81a6
>>> point = S256Point.parse(sec)
>>> signature = Signature.parse(der)
>>> point.verify(z, signature)
True
```

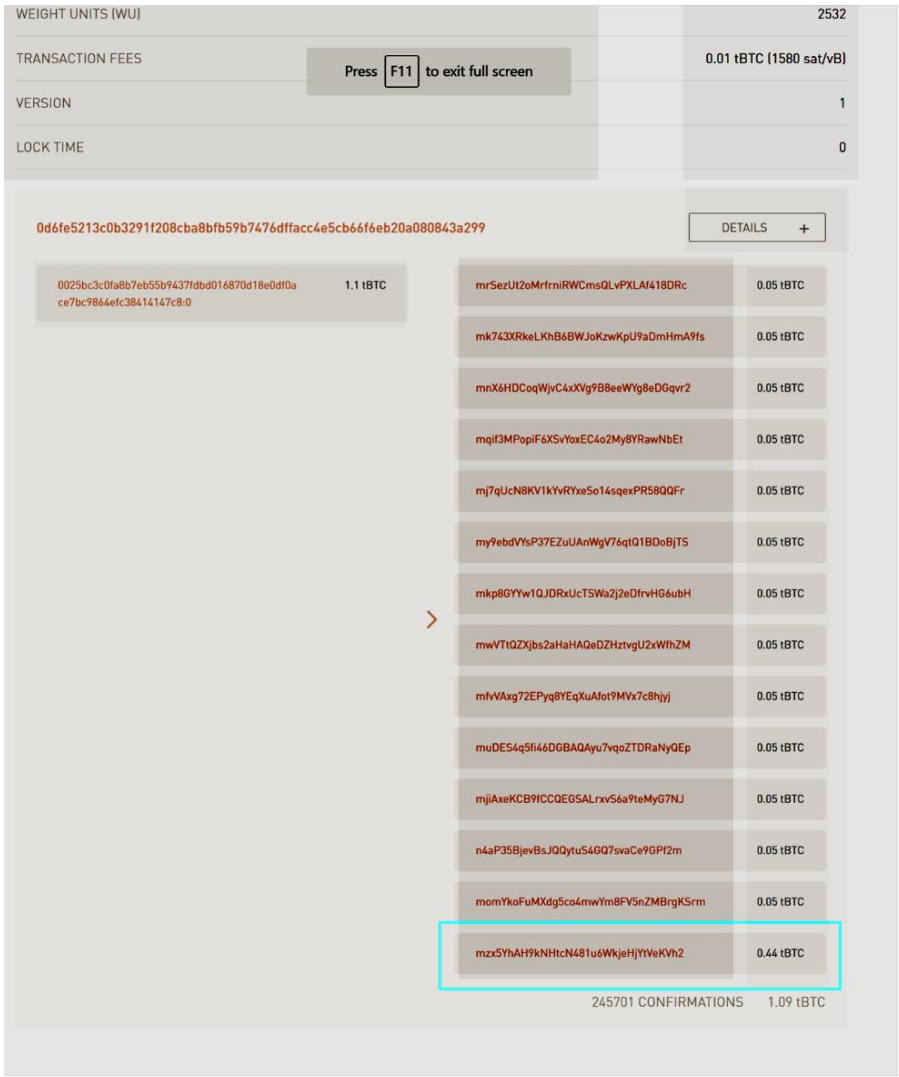
تائید یک تراکنش کامل

```
class Tx:
...
    def verify(self):
        '''Verify this transaction'''
        if self.fee() < 0:
            return False
        for i in range(len(self.tx_ins)):
            if not self.verify_input(i):
                return False
        return True
```

1

2

ساخت یک تراکنش



شکل ٧-٦: UTXO که خرج می‌شود

ساختن تراکنش

```
def decode_base58(s):
    num = 0
    for c in s:
        num *= 58
        num += BASE58_ALPHABET.index(c)
    combined = num.to_bytes(25, byteorder='big')
    checksum = combined[-4:]
    if hash256(combined[:-4])[:4] != checksum:
        raise ValueError('bad address: {}
{}'.format(checksum,
            hash256(combined[:-4])[:4]))
    return combined[1:-4]
```

1 2 3

```
def p2pkh_script(h160):
    '''Takes a hash160 and returns the p2pkh ScriptPubKey'''
    return Script([0x76, 0xa9, h160, 0x88, 0xac])
```

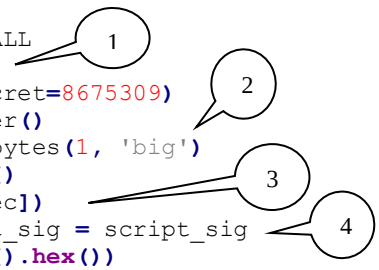
```
>>> from helper import decode_base58, SIGHASH_ALL
>>> from script import p2pkh_script, Script
>>> from tx import TxIn, TxOut, Tx
>>> prev_tx =
bytes.fromhex('0d6fe5213c0b3291f208cba8bfb59b7476dffacc4e5cb
66f6\
eb20a080843a299')
>>> prev_index = 13
>>> tx_in = TxIn(prev_tx, prev_index)
>>> tx_outs = []
>>> change_amount = int(0.33*100000000)
>>> change_h160 =
decode_base58('mzx5YhAH9kNHtcN481u6WkjeHjYtVeKVh2')
>>> change_script = p2pkh_script(change_h160)
>>> change_output = TxOut(amount=change_amount,
script_pubkey=change_script)
>>> target_amount = int(0.1*100000000)
>>> target_h160 =
decode_base58('mnrVtF8DWjMu839VW3rBfgYaAfKk8983Xf')
>>> target_script = p2pkh_script(target_h160)
>>> target_output = TxOut(amount=target_amount,
script_pubkey=target_script)
>>> tx_obj = Tx(1, [tx_in], [change_output, target_output],
0, True)
>>> print(tx_obj)
```

1 1 2

```
tx:
cd30a8da777d28ef0e61efe68a9f7c559c1d3e5bcd7b265c850ccb406859
8d11
version: 1
tx_ins:
0d6fe5213c0b3291f208cba8bfb59b7476dffacc4e5cb66f6eb20a080843
a299:13
tx_outs:
33000000:OP_DUP OP_HASH160
d52ad7ca9b3d096a38e752c2018e6fbc40cdf26f OP_EQUALVE\
RIFY OP_CHECKSIG
10000000:OP_DUP OP_HASH160
507b27411ccf7f16f10297de6cef3f291623eddf OP_EQUALVE\
RIFY OP_CHECKSIG
locktime: 0
```

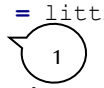
امضا کردن تراکنش

```
>>> from ecc import PrivateKey
>>> from helper import SIGHASH_ALL
>>> z = transaction.sig_hash(0)
>>> private_key = PrivateKey(secret=8675309)
>>> der = private_key.sign(z).der()
>>> sig = der + SIGHASH_ALL.to_bytes(1, 'big')
>>> sec = private_key.point.sec()
>>> script_sig = Script([sig, sec])
>>> transaction.tx_ins[0].script_sig = script_sig
>>> print(transaction.serialize().hex())
0100000001813f79011acb80925dfe69b3def355fe914bd1d96a3f5f71bf
8303c6a989c7d10000\
00006a47304402207db2402a3311a3b845b038885e3dd889c08126a8570f
26a844e3e4049c482a\
11022010178cdca4129eacbeab7c44648bf5ac1f9cac217cd609d216ec2e
bc8d242c0a01210393\
5581e52c354cd2f484fe8ed83af7a3097005b2f9c60bff71d35bd795f54b
67feffffff02a135ef\
01000000001976a914bc3b654dca7e56b04dca18f2566cdaf02e8d9ada88
ac99c3980000000000\
1976a9141c4bc762dd5423e332166702cb75f40df79fea1288ac19430600
```



ایجاد تراکنش‌های شخصی در testnet

```
>>> from ecc import PrivateKey
>>> from helper import hash256, little_endian_to_int
>>> secret = little_endian_to_int(hash256(b'Jimmy Song
secret'))
>>> private_key = PrivateKey(secret)
>>> print(private_key.point.address(testnet=True))
mn81594PzKZa9K3Jyy1ushpuEzrnTnxhVg
```



فصل هشتم

Pay-to-Script Hash

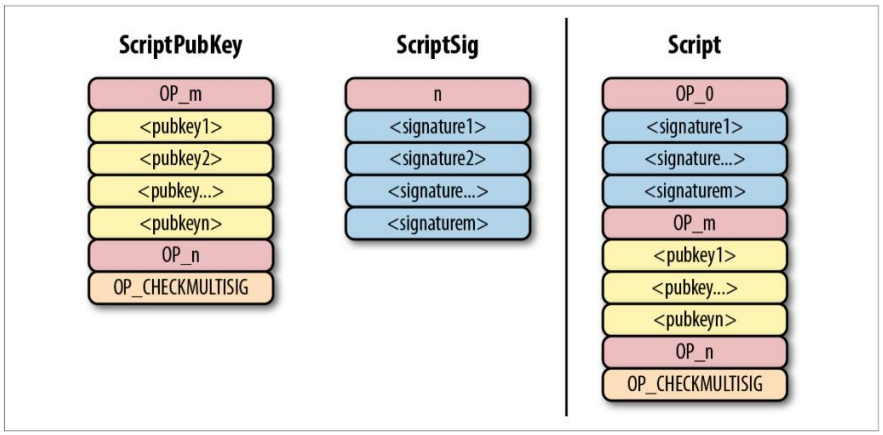
Bare multisig

```
5141 04fcf07bb1222f7925f2b7cc15183a40443c578e62ea17100aa3b44b
a66905c95d4980aec4cd2f6eb426d1b1ec45d76724f26901099416b9265b
76ba67c8b0b73d21 0202be80a0ca69c0e000b97d507f45b98c49f58fec66
50b64ff70e6ffccc3e6d00 52ae

        - 51 - OP_1
        - 41 - Length of <pubkey1>
        - 04fc...3d - <pubkey1>
        - 21 - Length of <pubkey2>
        - 0202...00 - <pubkey2>
        - 52 - OP_2
        - ae - OP_CHECKMULTISIG
```

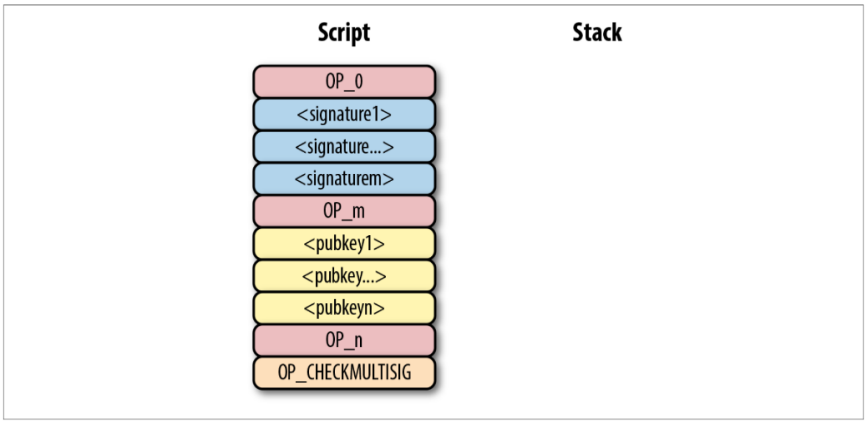
شکل ۸–۱: Bare multisig ScriptPubKey

```
00 483045022100e222a0a6816475d85ad28fbeb66e97c931081076dc9655
da3afc6c1d81b43f9802204681f9ea9d52a31c9c47cf78b71410ecae6188
d7c31495f5f1adfe0df5864a7401

        - 00 - OP_0
        - 48 - Length of <signature1>
        - 3045...01 - <signature1>
```

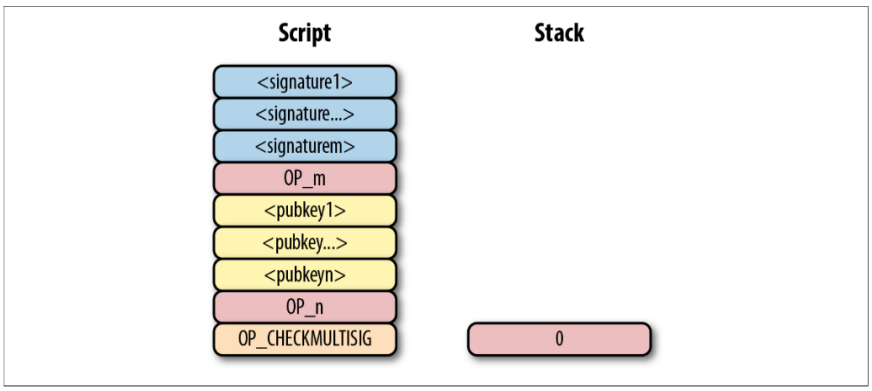
شکل ۸–۲: Bare multisig ScriptSig



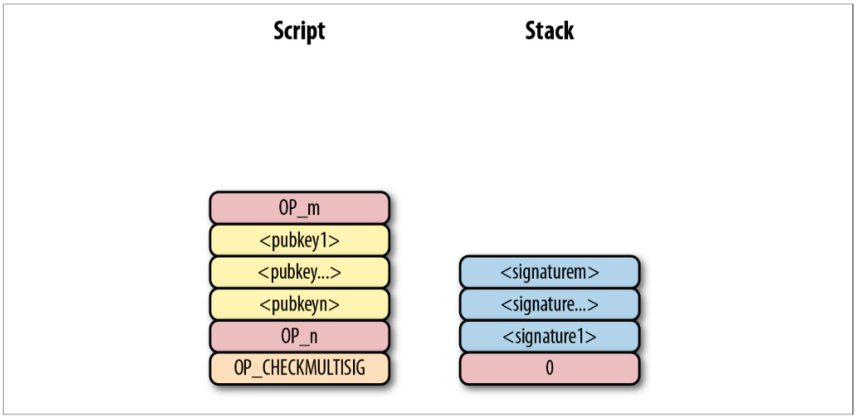
شکل ۸–۳: Bare multisig combined script



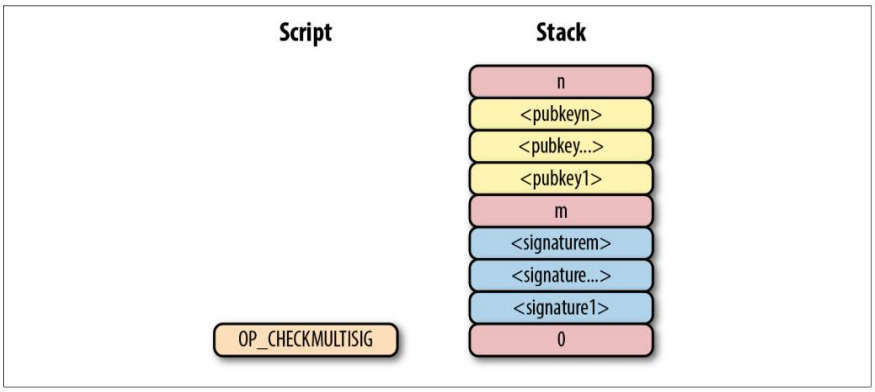
شکل ۸-٤: شروع Bare multisig



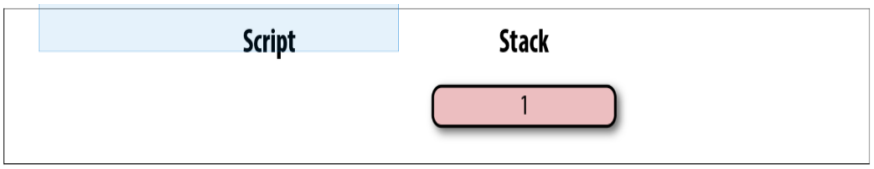
شکل ۸-۵: مرحله‌ی اول Bare multisig



شکل ۸–۶: مرحله‌ی دوم Bare Multisig



شکل ۸–۷: مرحله‌ی سوم Bare multisig



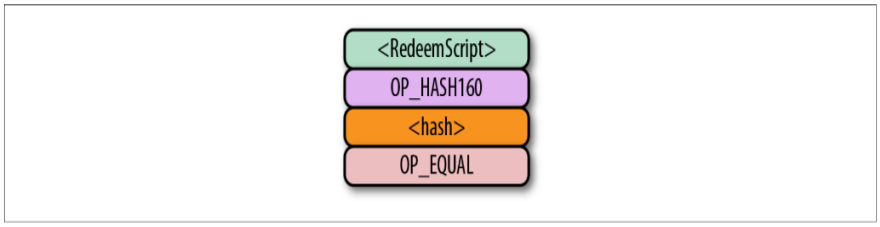
شکل ۸–۸: پایان Bare multisig

کد کردن OP_CHECKMULTISIG

```
def op_checkmultisig(stack, z):
    if len(stack) < 1:
        return False
    n = decode_num(stack.pop())
    if len(stack) < n + 1:
        return False
    sec_pubkeys = []
    for _ in range(n):
        sec_pubkeys.append(stack.pop())
    m = decode_num(stack.pop())
    if len(stack) < m + 1:
        return False
    der_signatures = []
    for _ in range(m):
        der_signatures.append(stack.pop()[:-1])
    stack.pop()
    try:
        raise NotImplementedError
    except (ValueError, SyntaxError):
        return False
    return True
```

1

Pay-to-Script-Hash (p2sh)



شکل ۸-۹: الگوی p2sh که قانون خاصی را اجرا می‌کند.

```
5221022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff129
5d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec45
3e1ffac7fbdbd4bb7152ae

        - 52 - OP_2
        - 21 - Length of <pubkey1>
        - 02...db70 - <pubkey1>
        - 21 - Length of <pubkey2>
        - 03...bb71 - <pubkey2>
        - 52 - OP_2
        - ae - OP_CHECKMULTISIG
```

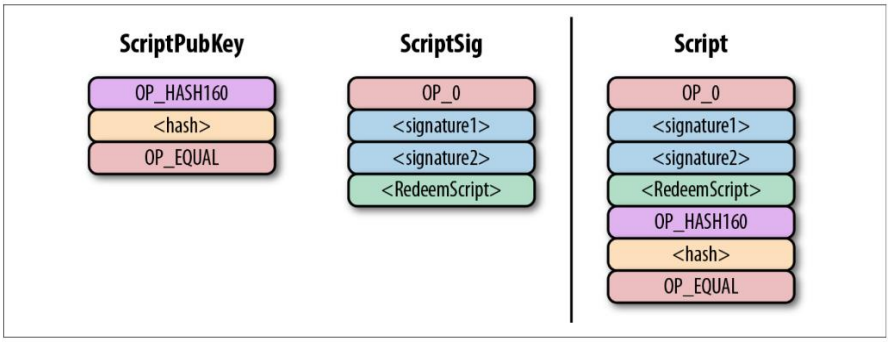
شکل ۸–۱۰: Pay-to-script-hash (p2sh) RedeemScript

```
a91474d691da1574e6b3c192ecfb52cc8984ee7b6c5687

        - a9 - OPHASH160
        - 14 - Length of <hash>
        - 74d6...56 - <hash>
        - 87 - OP_EQUAL
```

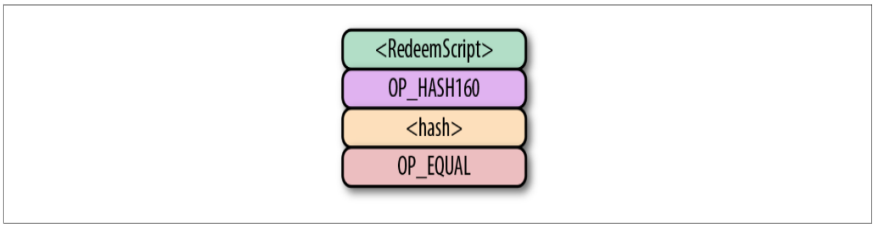
شکل ۸–۱۱: Pay-to-script-hash (p2sh) ScriptPubKey

```
00483045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e022
05a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a8993701483045022100da
6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dcadce
2121ea07f233115c6f09034c0be68db99980b9a6c5e7540220147 5221022626e955ea6ea6d98850c9
94f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed804
5c8b98bec453e1ffac7fbdbd4bb7152ae

        - 00 - OP_0
        - 48 - Length of <signature1>
        - 3045...3701 - <signature1>
        - 48 - Length of <signature2>
        - 3045...2201 - <signature2>
        - 47 - Length of <RedeemScript>
        - 5221...ae - <RedeemScript>
```

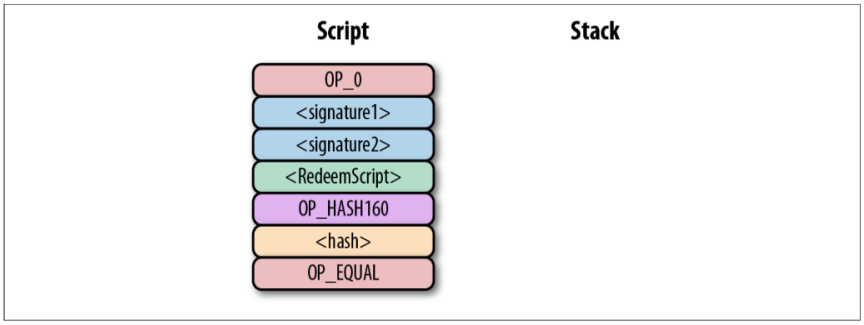
شکل ۸–۱۲: Pay-to-script-hash (p2sh) ScriptSig



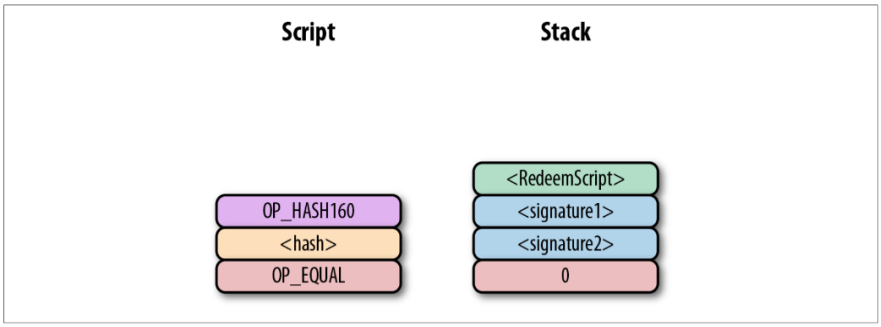
شکل ۸–۱۳: اسکریپتِ ترکیب‌شده p2sh



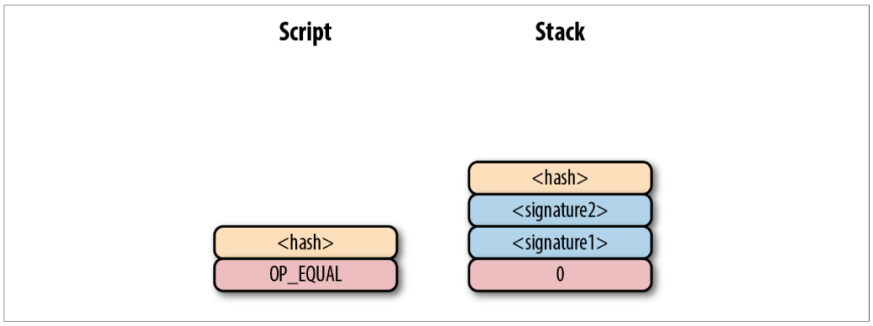
شکل ۸–۱٤: الگوی p2sh که قانون خاصی را اجرا می‌کند.



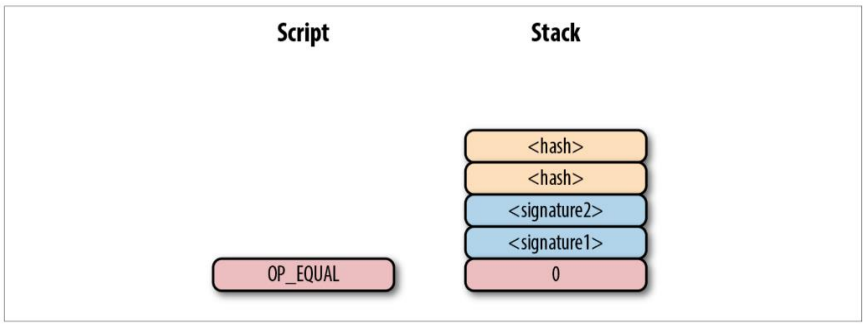
شکل ۸–۱۵: شروع p2sh



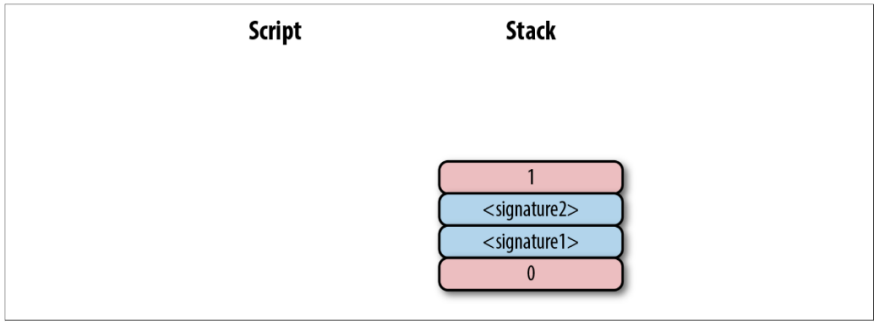
شکل ۸–۱۶: مرحله‌ی اول p2sh



شکل ۸–۱۷: مرحله‌ی دوم p2sh



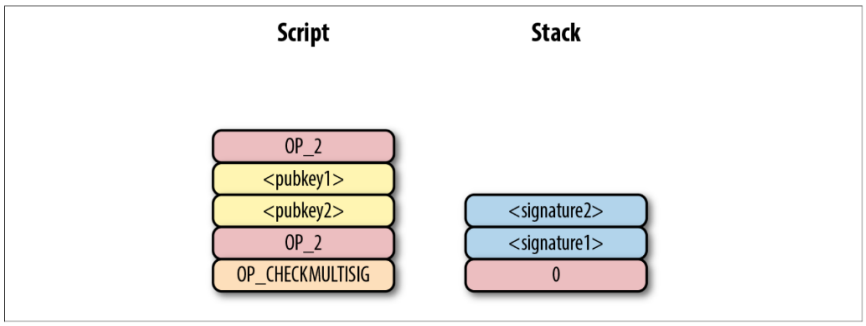
شکل ۸–۱۸: مرحله‌ی سوم p2sh



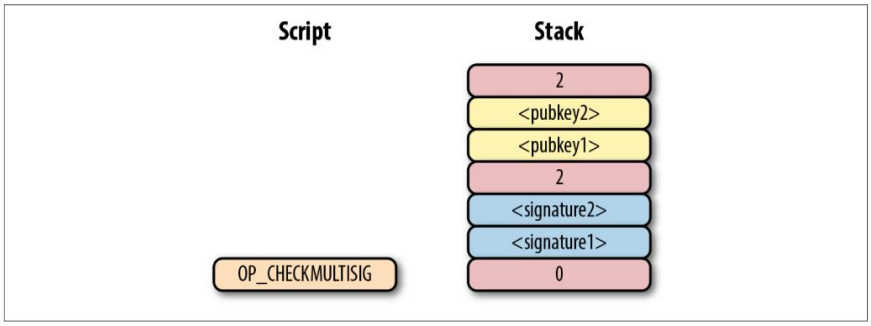
شکل ۸–۱۹: اگر ارزیابی با نرم‌افزارِ pre-BIP0016 صورت گیرد، p2sh خاتمه می‌یابد.

```
5221022626e955ea6ea6d98850c994f9107b036b1334f18ca8830bfff129
5d21cfdb702103b287eaf122eea69030a0e9feed096bed8045c8b98bec45
3e1ffac7fbdbd4bb7152ae

        - 52 - OP_2
        - 21 - Length of <pubkey1>
        - 02...db70 - <pubkey1>
        - 21 - Length of <pubkey2>
        - 03...bb71 - <pubkey2>
        - 52 - OP_2
        - ae - OP_CHECKMULTISIG
```

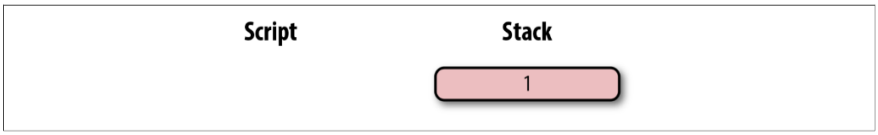
شکل ۸–۲۰: p2sh RedeemScript



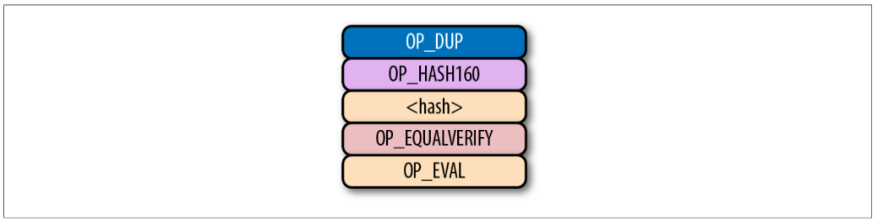
شکل ۸–۲۱: مرحله‌ی چهارم p2sh



شکل ۸-۲۲: مرحله‌ی پنجم p2sh



شکل ۸-۲۳: p2sh برای نرم‌افزار post-BIP0016 پایان می‌یابد.



شکل ۸-۲٤: OP_EVAL می‌تواند دستوری باشد که دستورات اضافی را بر اساس عنصر بالا اضافه می‌کند.

کد کردن p2sh

```
class Script:
...
    def evaluate(self, z):
...
        while len(commands) > 0:
            command = commands.pop(0)
            if type(command) == int:
...
            else:
                stack.append(cmd)
                if len(cmds) == 3 and cmds[0] == 0xa9 \
```

```
                    and type(cmds[1]) == bytes and
len(cmds[1]) == 20 \
                    and cmds[2] == 0x87:                1
                2   cmds.pop()
                    h160 = cmds.pop()
                    cmds.pop()
                    if not op_hash160(stack):           3
                        return False
                    stack.append(h160)
                    if not op_equal(stack):
                4       return False
                    if not op_verify(stack):
                        LOGGER.info('bad p2sh h160')
                        return False
                5   redeem_script = encode_varint(len(cmd))
+ cmd
                    stream = BytesIO(redeem_script)
                    cmds.extend(Script.parse(stream).cmds)
```

آدرس‌ها

```
>>> from helper import encode_base58_checksum
>>> h160 =
bytes.fromhex('74d691da1574e6b3c192ecfb52cc8984ee7b6c56')
>>> print(encode_base58_checksum(b'\x05' + h160))
3CLoMMyuoDQTPRD3XYZtCvgvkadrAdvdXh
```

تائید امضای p2sh

```
0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a385aa0836f01d5e4789e6bd304d87221a00000
0db00483045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e
02205a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a899370148304502210
0da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dca
dce2121ea07f233115c6f09034c0be68db99980b9a6c5e754022014752210226 26e955ea6ea6d9885
0c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed
8045c8b98bec453e1ffac7fbdbd4bb7152aeffffffff04d3b1140000000000001976a914904a49878c0
adfc3aa05de7afad2cc15f483a56a88ac7f4009000000000001976a914418327e3f3dda4cf5b908932
5a4b95abdfa0334088ac722c0c0000000001976a914ba35042cfe9fc66fd35ac2224eebdafd1028a
d2788acdc4ace020000000017a91474d691da1574e6b3c192ecfb52cc8984ee7b6c5687 00000000
```

شکل ۸–۲۵: اعتبارسنجی ورودی‌های p2sh

گام اول: خالی کردن scriptsigs

شکل ۸–۲۶: خالی کردن هر scriptsig ورودی

گام دوم: جایگزینیِ scriptsig ورودیِ p2sh امضا شده با RedeemScript

شکل ۸–۲۷: جایگزینی scriptsig ورودی که با RedeemScript بررسی می‌شود.

گام سوم: اضافه کردن نوع هش

شکل ۸–۲۸: Append the hash type (SIGHASH_ALL), 01000000

```
>>> from helper import hash256
>>> modified_tx =
bytes.fromhex('0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a38
\
5aa0836f01d5e4789e6bd304d87221a0000004752210226 26e955ea6ea6d
98850c994f9107b036\
b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9fee
d096bed8045c8b98be\
c453e1ffac7fbdbd4bb7152aeffffffff04d3b11400000000001976a9149
04a49878c0adfc3aa0\
5de7afad2cc15f483a56a88ac7f40090000000001976a914418327e3f3d
da4cf5b9089325a4b9\
5abdfa0334088ac722c0c00000000001976a914ba35042cfe9fc66fd35ac
2224eebdafd1028ad2\
788acdc4ace020000000017a91474d691da1574e6b3c192ecfb52cc8984e
e7b6c5687000000000\
1000000')
>>> s256 = hash256(modified_tx)
>>> z = int.from_bytes(s256, 'big')
>>> print(hex(z))
0xe71bfa115715d6fd33796948126f40a8cdd39f187e4afb03896795189f
e1423c
```

```
0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a385aa0836f01d5e4789e6bd304d87221a00000
0db00483045022100dc92655fe37036f47756db8102e0d7d5e28b3beb83a8fef4f5dc0559bddfb94e
02205a36d4e4e6c7fcd16658c50783e00c341609977aed3ad00937bf4ee942a899370148304502210
0da6bee3c93766232079a01639d07fa869598749729ae323eab8eef53577d611b02207bef15429dca
dce2121ea07f233115c6f09034c0be68db99980b9a6c5e75402201475221022626e955ea6ea6d9885
0c994f9107b036b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9feed096bed
8045c8b98bec453e1ffac7fbdbd4bb7152aeffffffff04d3b11400000000001976a914904a49878c0
adfc3aa05de7afad2cc15f483a56a88ac7f40090000000001976a914418327e3f3dda4cf5b908932
5a4b95abdfa0334088ac722c0c00000000001976a914ba35042cfe9fc66fd35ac2224eebdafd1028a
d2788acdc4ace020000000017a91474d691da1574e6b3c192ecfb52cc8984ee7b6c568700000000
```

شکل ۸-۲۹: DER signature and SEC pubkey within the p2sh ScriptSig and Redeem Script

```
>>> from ecc import S256Point, Signature
>>> from helper import hash256
>>> modified_tx =
bytes.fromhex('0100000001868278ed6ddfb6c1ed3ad5f8181eb0c7a38
\
5aa0836f01d5e4789e6bd304d87221a0000004752210226 26e955ea6ea6d
98850c994f9107b036\
b1334f18ca8830bfff1295d21cfdb702103b287eaf122eea69030a0e9fee
d096bed8045c8b98be\
```

```
c453e1ffac7fbdbd4bb7152affffffff04d3b11400000000001976a9149
04a49878c0adfc3aa0\
5de7afad2cc15f483a56a88ac7f400900000000001976a914418327e3f3d
da4cf5b9089325a4b9\
5abdfa0334088ac722c0c00000000001976a914ba35042cfe9fc66fd35ac
2224eebdafd1028ad2\
788acdc4ace020000000017a91474d691da1574e6b3c192ecfb52cc8984e
e7b6c5687000000000\
1000000')
>>> h256 = hash256(modified_tx)
>>> z = int.from_bytes(h256, 'big')
>>> sec =
bytes.fromhex('022626e955ea6ea6d98850c994f9107b036b1334f18ca
8830bfff\
1295d21cfdb70')
>>> der =
bytes.fromhex('3045022100dc92655fe37036f47756db8102e0d7d5e28
b3beb83a\
8fef4f5dc0559bddfb94e02205a36d4e4e6c7fcd16658c50783e00c34160
9977aed3ad00937bf4\
ee942a89937')
>>> point = S256Point.parse(sec)
>>> sig = Signature.parse(der)
>>> print(point.verify(z, sig))
True
```

فصل نهم

بلوک‌ها

تراکنش‌های سکه‌پایه

```
010000000100000000000000000000000000000000000000000000000000000000ffffffff
5e03d71b07254d696e656420627920416e74506f6f6c20626a31312f4542312f4144362f43205914
293101fabe6d6d678e2c8c34afc36896e7d9402824ed38e856676ee94bfdb0c6c4bcd8b2e5666a040
0000000000000c7270000a5e00e00ffffffff01faf20b5800000000 1976a914338c84849423992471
bffb1a54a8d9b1d69dc28a88ac00000000
```

- 01000000 - version
- 01 - # of inputs
- 000...00 - previous tx hash
- ffffffff - previous tx index
- 5e0...00 - ScriptSig
- ffffffff - sequence
- 01 - # of outputs
- faf20b58...00 - output amount
- 1976...ac - p2pkh ScriptPubKey
- 00000000 - locktime

شکل ۹-۱: جزییات تراکنش ارز پایه.

ScriptSig

```
>>> from io import BytesIO
>>> from script import Script
>>> stream =
BytesIO(bytes.fromhex('4d04ffff001d0104455468652054696d657332
03033\
2f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b20
6f66207365636f6e64\
206261696c6f757420666f722062616e6b73'))
>>> s = Script.parse(stream)
>>> print(s.cmds[2])
b'The Times 03/Jan/2009 Chancellor on brink of second
bailout for banks'
```

پروتکل BIP0034

```
>>> from io import BytesIO
>>> from script import Script
>>> from helper import little_endian_to_int
>>> stream =
BytesIO(bytes.fromhex('5e03d71b07254d696e656420627920416e745
06f6f\
6c20626a31312f4542312f4144362f43205914293101fabe6d6d678e2c8c
34afc36896e7d94028\
```

```
24ed38e856676ee94bfdb0c6c4bcd8b2e5666a0400000000000000c72700
00a5e00e00'))
>>> script_sig = Script.parse(stream)
>>> print(little_endian_to_int(script_sig.cmds[0]))
465879
```

سرآیندهای بلوک‌ها

```
020000208ec39428b17323fa0ddec8e887b4a7c53b8c0a0
a220cfd0000000000000000005b0750fce0a889502d4050
8d39576821155e9c9e3f5c3157f961db38fd8b25be1e77a
759e93c0118a4ffd71d

        - 02000020 - version, 4 bytes, LE
        - 8ec3...00 - previous block, 32 bytes, LE
        - 5b07...be - merkle root, 32 bytes, LE
        - 1e77a759 - timestamp, 4 bytes, LE
        - e93c0118 - bits, 4 bytes
        - a4ffd71d - nonce, 4 bytes
```

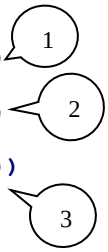
شکل ۹-۲: سرآیند بلوک

```
>>> from helper import hash256
>>> block_hash =
hash256(bytes.fromhex('020000208ec39428b17323fa0ddec8e887b4a
7\
c53b8c0a0a220cfd0000000000000000005b0750fce0a889502d40508d39
576821155e9c9e3f5c\
3157f961db38fd8b25be1e77a759e93c0118a4ffd71d'))[::-1]
>>> block_id = block_hash.hex()
>>> print(block_id)
0000000000000000007e9e4c586439b0cdbe13b1370bdd9435d76a644d04
7523
```

```
class Block:

    def __init__(self, version, prev_block, merkle_root,
timestamp, bits, nonce):
        self.version = version
        self.prev_block = prev_block
        self.merkle_root = merkle_root
        self.timestamp = timestamp
        self.bits = bits
        self.nonce = nonce
```

نسخه

```
>>> from io import BytesIO
>>> from block import Block
>>> b =
Block.parse(BytesIO(bytes.fromhex('020000208ec39428b17323fa0
ddec8e887b\
4a7c53b8c0a0a220cfd00000000000000000005b0750fce0a889502d40508
d39576821155e9c9e3\
f5c3157f961db38fd8b25be1e77a759e93c0118a4ffd71d')))
>>> print('BIP9: {}'.format(b.version >> 29 == 0b001))
BIP9: True
>>> print('BIP91: {}'.format(b.version >> 4 & 1 == 1))
BIP91: False
>>> print('BIP141: {}'.format(b.version >> 1 & 1 == 1))
BIP141: True
```

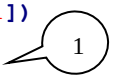


اثبات کار

```
020000208ec39428b17323fa0ddec8e887b4a7c53b8c0a0a220cfd000000
0000000000005b0750fce0a889502d40508d39576821155e9c9e3f5c3157
f961db38fd8b25be1e77a759
e93c0118a4ffd71d

>>> from helper import hash256
>>> block_id =
hash256(bytes.fromhex('020000208ec39428b17323fa0ddec8e887b4a
7c5\3b8c0a0a220cfd00000000000000000005b0750fce0a889502d40508d
39576821155e9c9e3f5c31\57f961db38fd8b25be1e77a759e93c0118a4f
fd71d'))[::-1]
>>> print('{}'.format(block_id.hex()).zfill(64))
0000000000000000007e9e4c586439b0cdbe13b1370bdd9435d76a644d04
7523
```

هدف

```
>>> from helper import little_endian_to_int
>>> bits = bytes.fromhex('e93c0118')
>>> exponent = bits[-1]
>>> coefficient = little_endian_to_int(bits[:-1])
>>> target = coefficient * 256**(exponent - 3)
>>> print('{:x}'.format(target).zfill(64))
0000000000000000013ce900000000000000000000000000000000000000
0000
```



```
>>> from helper import little_endian_to_int
>>> proof =
little_endian_to_int(hash256(bytes.fromhex('020000208ec39428
b17323\
fa0ddec8e887b4a7c53b8c0a0a220cfd0000000000000000005b0750fce0
a889502d40508d3957\
6821155e9c9e3f5c3157f961db38fd8b25be1e77a759e93c0118a4ffd71d
')))
>>> print(proof < target)
True
```

1

سختی شبکه

```
>>> from helper import little_endian_to_int
>>> bits = bytes.fromhex('e93c0118')
>>> exponent = bits[-1]
>>> coefficient = little_endian_to_int(bits[:-1])
>>> target = coefficient*256**(exponent-3)
>>> difficulty = 0xffff * 256**(0x1d-3) / target
>>> print(difficulty)
888171856257.3206
```

تنظیم سختی شبکه

```
>>> from block import Block
>>> from helper import TWO_WEEKS
>>> last_block =
Block.parse(BytesIO(bytes.fromhex('00000020fdf740b0e49cf75bb
3\
d5168fb3586f7613dcc5cd89675b0100000000000000002e37b144c0bace
d07eb7e7b64da916cd\
3121f2427005551aeb0ec6a6402ac7d7f0e4235954d801187f5da9f5')))
>>> first_block =
Block.parse(BytesIO(bytes.fromhex('000000201ecd89664fd205a37
\
566e694269ed76e425803003628ab0100000000000000000bfcade29d080d
9aae8fd461254b0418\
05ae442749f2a40100440fc0e3d5868e55019345954d80118a1721b2e'))
)
>>> time_differential = last_block.timestamp -
first_block.timestamp
>>> if time_differential > TWO_WEEKS * 4:
...     time_differential = TWO_WEEKS * 4
>>> if time_differential < TWO_WEEKS // 4:
...     time_differential = TWO_WEEKS // 4
```

1

2

3

```
>>> new_target = last_block.target() * time_differential //
TWO_WEEKS
>>> print('{:x}'.format(new_target).zfill(64))
0000000000000000076150000000000000000000000000000000000000000000
0000
```

تبدیل Target به دست آمده به بیت:

```
def target_to_bits(target):
    '''Turns a target integer back into bits'''
    raw_bytes = target.to_bytes(32, 'big')
    raw_bytes = raw_bytes.lstrip(b'\x00')
    if raw_bytes[0] > 0x7f:
        exponent = len(raw_bytes) + 1
        coefficient = b'\x00' + raw_bytes[:2]
    else:
        exponent = len(raw_bytes)
        coefficient = raw_bytes[:3]
    new_bits = coefficient[::-1] + bytes([exponent])
    return new_bits
```

2 1 4 3 5

فصل دهم

شبکه‌سازی

پیام‌های شبکه

```
f9beb4d976657273696f6e0000000065000000 5f1a69d2 7211010001000000000000000bc8f5e540
0000000010000000000000000000000000000000000000ffffc61b6409208d0100000000000000000000
00000000000000ffffcb0071c0208d128035cbc97953f80f2f5361746f7368693a302e392e332fcf0
5050001

        - f9beb4d9 - network magic (always 0xf9beb4d9 for mainnet)
        - 76657273696f6e0000000000 - command, 12 bytes, human-readable
        - 65000000 - payload length, 4 bytes, little-endian
        - 5f1a69d2 - payload checksum, first 4 bytes of hash256 of the
                     payload
        - 7211...01 - payload
```

شکل ۱۰–۱: یک پیام شبکه – پاکت نامه‌ای که حاوی *بارداده* واقعی است.

```
NETWORK_MAGIC = b'\xf9\xbe\xb4\xd9'
TESTNET_NETWORK_MAGIC = b'\x0b\x11\x09\x07'

class NetworkEnvelope:

    def __init__(self, command, payload, testnet=False):
        self.command = command
        self.payload = payload
        if testnet:
            self.magic = TESTNET_NETWORK_MAGIC
        else:
            self.magic = NETWORK_MAGIC

    def __repr__(self):
        return '{}: {}'.format(
            self.command.decode('ascii'),
            self.payload.hex(),
        )
```

تجزیه‌ی بارداده

```
7f110100000000000000000ad17835b00000000000000000000000000000000000000000ffff0
0000000d20000000000000000000000000000000000ffff00000000d20f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f0000000001

7f110100 - Protocol version, 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37
```

شکل ۱۰-۲: نسخه تجزیه شده *بارداده*

کلاس VersionMessage:

```
class VersionMessage:
    command = b'version'

    def __init__(self, version=70015, services=0,
timestamp=None,
                 receiver_services=0,
                 receiver_ip=b'\x00\x00\x00\x00',
receiver_port=8333,
                 sender_services=0,
                 sender_ip=b'\x00\x00\x00\x00',
sender_port=8333,
                 nonce=None,
user_agent=b'/programmingbitcoin:0.1/',
                 latest_block=0, relay=False):
        self.version = version
        self.services = services
        if timestamp is None:
            self.timestamp = int(time.time())
        else:
            self.timestamp = timestamp
        self.receiver_services = receiver_services
        self.receiver_ip = receiver_ip
        self.receiver_port = receiver_port
        self.sender_services = sender_services
        self.sender_ip = sender_ip
        self.sender_port = sender_port
```

```
        if nonce is None:
            self.nonce = int_to_little_endian(randint(0,
2**64), 8)
        else:
            self.nonce = nonce
        self.user_agent = user_agent
        self.latest_block = latest_block
        self.relay = relay
```

اتصال به شبکه

```
>>> import socket
>>> from network import NetworkEnvelope, VersionMessage
>>> host = 'testnet.programmingbitcoin.com'
>>> port = 18333
>>> socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
>>> socket.connect((host, port))
>>> stream = socket.makefile('rb', None)
>>> version = VersionMessage()
>>> envelope =
NetworkEnvelope(version.command,version.serialize())
>>> socket.sendall(envelope.serialize())
>>> while True:

...     new_message = NetworkEnvelope.parse(stream)
...     print(new_message)
```

1 2 3 4 5

کلاسِ پیامِ verack:

```
class VerAckMessage:
    command = b'verack'

    def __init__(self):
        pass
    @classmethod
    def parse(cls, s):
        return cls()

    def serialize(self):
        return b''
```

```
class SimpleNode:

    def __init__(self, host, port=None, testnet=False,
logging=False):
        if port is None:
            if testnet:
                port = 18333
            else:
                port = 8333
        self.testnet = testnet
        self.logging = logging
        self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.socket.connect((host, port))
        self.stream = self.socket.makefile('rb', None)

    def send(self, message):
        '''Send a message to the connected node'''
        envelope = NetworkEnvelope(
            message.command, message.serialize(),
testnet=self.testnet)
        if self.logging:
            print('sending: {}'.format(envelope))
        self.socket.sendall(envelope.serialize())

    def read(self):
        '''Read a message from the socket'''
        envelope = NetworkEnvelope.parse(self.stream,
testnet=self.testnet)
        if self.logging:
            print('receiving: {}'.format(envelope))
        return envelope

    def wait_for(self, *message_classes):
        '''Wait for one of the messages in the list'''
        command = None
        command_to_class = {m.command: m for m in
message_classes}
        while command not in command_to_class.keys():
            envelope = self.read()
            command = envelope.command
            if command == VersionMessage.command:
                self.send(VerAckMessage())
            elif command == PingMessage.command:
                self.send(PongMessage(envelope.payload))
        return
command_to_class[command].parse(envelope.stream())
```

1

2

3

```
>>> from network import SimpleNode, VersionMessage
>>> node = SimpleNode('testnet.programmingbitcoin.com',
testnet=True)
>>> version = VersionMessage()
>>> node.send(version)
>>> verack_received = False
>>> version_received = False
>>> while not verack_received and not version_received:
...     message = node.wait_for(VersionMessage,
VerAckMessage)
...     if message.command == VerAckMessage.command:
...         verack_received = True
...     else:
...         version_received = True
...         node.send(VerAckMessage())
```

دریافت سرآیندهای بلوک

```
7f11010001a35bd0ca2f4a88c4eda6d213e2378a5758dfcd6af437120000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000

        - 7f110100 - Protocol version, 4 bytes, little-endian, 70015
        - 01 - Number of hashes, varint
        - a35b...00 - Starting block, little-endian
        - 0000...00 - Ending block, little-endian
```

شکل ۱۰-۳: Parsed getheaders

```
class GetHeadersMessage:
    command = b'getheaders'

    def __init__(self, version=70015, num_hashes=1,
start_block=None, end_block=None):
        self.version = version
        self.num_hashes = num_hashes
        if start_block is None:
            raise RuntimeError('a start block is required')
        self.start_block = start_block
        if end_block is None:
            self.end_block = b'\x00' * 32
        else:
            self.end_block = end_block
```

پاسخ سرآیندها

```
>>> from io import BytesIO
>>> from block import Block, GENESIS_BLOCK
>>> from network import SimpleNode, GetHeadersMessage
>>> node = SimpleNode('mainnet.programmingbitcoin.com',
testnet=False)
>>> node.handshake()
>>> genesis = Block.parse(BytesIO(GENESIS_BLOCK))
>>> getheaders =
GetHeadersMessage(start_block=genesis.hash())
>>> node.send(getheaders)
```

```
02000000020df3b053dc46f162a9b00c7f0d5124e2676d47bbe7c5d0793a500000000000000ef445fef
2ed495c275892206ca533e7411907971013ab83e3b47bd0d692d14d4dc7c835b67d8001ac157e67000
00000002030eb2540c41025690160a1014c577061596e32e426b712c7ca00000000000000768b89f07
044e6130ead292a3f51951adbd2202df447d98789339937fd006bd44880835b67d8001ade0920400

        - 02 - Number of block headers
        - 00...67 - Block header
        - 00 - Number of transactions (always 0)
```

شکل ۱۰-٤: تجزیه سرآیندها

```
class HeadersMessage:
    command = b'headers'

    def __init__(self, blocks):
        self.blocks = blocks

    @classmethod
    def parse(cls, stream):
        num_headers = read_varint(stream)
        blocks = []
        for _ in range(num_headers):
            blocks.append(Block.parse(stream))
            num_txs = read_varint(stream)
            if num_txs != 0:
                raise RuntimeError('number of txs not 0')
        return cls(blocks)
```

1 2 3

دانلود سرآیندها، بررسی اثبات کار آن‌ها و تائید تنظیمات سختی بلوک:

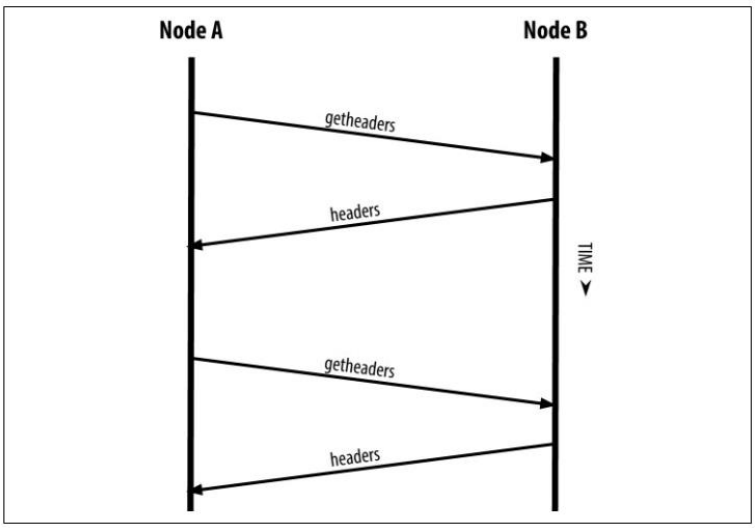
```
>>> from io import BytesIO
>>> from network import SimpleNode, GetHeadersMessage,
HeadersMessage
>>> from block import Block, GENESIS_BLOCK, LOWEST_BITS
>>> from helper import calculate_new_bits
>>> previous = Block.parse(BytesIO(GENESIS_BLOCK))
>>> first_epoch_timestamp = previous.timestamp
>>> expected_bits = LOWEST_BITS
>>> count = 1
>>> node = SimpleNode('mainnet.programmingbitcoin.com',
testnet=False)
>>> node.handshake()
>>> for _ in range(19):
...     getheaders =
GetHeadersMessage(start_block=previous.hash())
...     node.send(getheaders)
...     headers = node.wait_for(HeadersMessage)
...     for header in headers.blocks:
...         if not header.check_pow():
...             raise RuntimeError('bad PoW at block
{}'.format(count))
...         if header.prev_block != previous.hash():
...             raise RuntimeError('discontinuous block at
{}'.format(count))
...         if count % 2016 == 0:
...             time_diff = previous.timestamp -
first_epoch_timestamp
...             expected_bits =
calculate_new_bits(previous.bits, time_diff)
...             print(expected_bits.hex())
...             first_epoch_timestamp = header.timestamp
...         if header.bits != expected_bits:
...             raise RuntimeError('bad bits at block
{}'.format(count))
...         previous = header
...         count += 1
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
ffff001d
6ad8001d
28c4001d
71be001d
```

1 2 3 4 5

فصل یازدهم

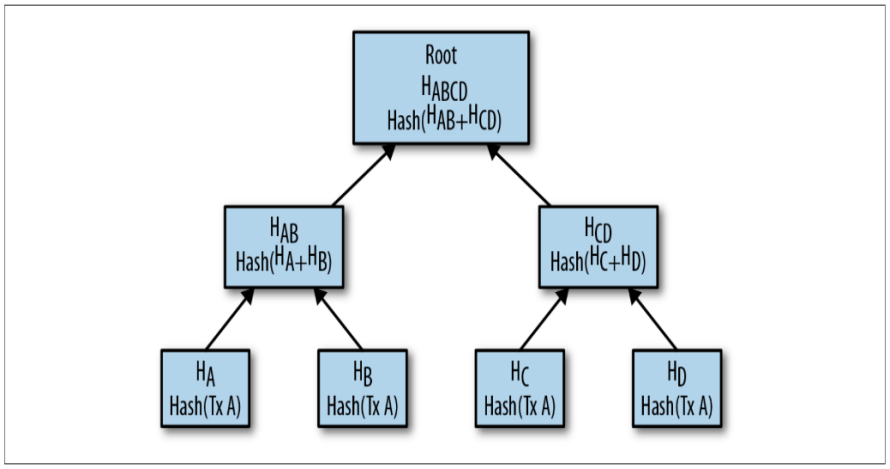
تائید مختصر پرداخت

تائید مختصر پرداخت



شکل ۱۱-۱: گره‌ی SPV سرآیندهای بلوک را هماهنگ می کند.

درخت مرکل



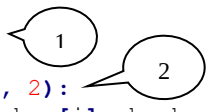
شکل ۱۱-۲: درخت مرکل

والد مرکل

```
>>> from helper import hash256
>>> hash0 =
bytes.fromhex('c117ea8ec828342f4dfb0ad6bd140e03a50720ece40169
ee38b\
dc15d9eb64cf5')
>>> hash1 =
bytes.fromhex('c131474164b412e3406696da1ee20ab0fc9bf41c8f05fa
8ceea\
7a08d672d7cc5')
>>> parent = hash256(hash0 + hash1)
>>> print(parent.hex())
8b30c5ba100f6f2e5ad1e2a742e5020491240f8eb514fe97c713c31718ad7
ecd
```

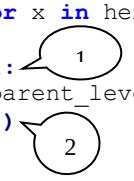
سطح والد مرکل

```
>>> from helper import merkle_parent
>>> hex_hashes = [
...
'c117ea8ec828342f4dfb0ad6bd140e03a50720ece40169ee38bdc15d9eb64
cf5',
...
'c131474164b412e3406696da1ee20ab0fc9bf41c8f05fa8ceea7a08d672d7
cc5',
...
'f391da6ecfeed1814efae39e7fcb3838ae0b02c02ae7d0a5848a66947c072
7b0',
...
'3d238a92a94532b946c90e19c49351c763696cff3db400485b813aecb8a13
181',
...
'10092f2633be5f3ce349bf9ddbde36caa3dd10dfa0ec8106bce23acbff637
dae',
... ]
>>> hashes = [bytes.fromhex(x) for x in hex_hashes]
>>> if len(hashes) % 2 == 1:
...     hashes.append(hashes[-1])
>>> parent_level = []
>>> for i in range(0, len(hashes), 2):
...     parent = merkle_parent(hashes[i], hashes[i+1])
...     parent_level.append(parent)
>>> for item in parent_level:
...     print(item.hex())
8b30c5ba100f6f2e5ad1e2a742e5020491240f8eb514fe97c713c31718ad7e
cd
7f4e6f9e224e20fda0ae4c44114237f97cd35aca38d83081c9bfd41feb9078
00
3ecf6115380c77e8aae56660f5634982ee897351ba906a6837d15ebc3a225d
f0
```



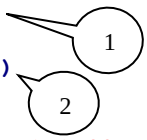
ریشه‌ی مرکل

```
>>> from helper import merkle_parent_level
>>> hex_hashes = [
...
'c117ea8ec828342f4dfb0ad6bd140e03a50720ece40169ee38bdc15d9eb
64cf5',
...
'c131474164b412e3406696da1ee20ab0fc9bf41c8f05fa8ceea7a08d672
d7cc5',
...
'f391da6ecfeed1814efae39e7fcb3838ae0b02c02ae7d0a5848a66947c0
727b0',
...
'3d238a92a94532b946c90e19c49351c763696cff3db400485b813aecb8a
13181',
...
'10092f2633be5f3ce349bf9ddbde36caa3dd10dfa0ec8106bce23acbff6
37dae',
...
'7d37b3d54fa6a64869084bfd2e831309118b9e833610e6228adacdbd1b4
ba161',
...
'8118a77e542892fe15ae3fc771a4abfd2f5d5d5997544c3487ac36b5c85
170fc',
...
'dff6879848c2c9b62fe652720b8df5272093acfaa45a43cdb3696fe2466
a3877',
...
'b825c0745f46ac58f7d3759e6dc535a1fec7820377f24d4c2c6ad2cc55c
0cb59',
...
'95513952a04bd8992721e9b7e2937f1c04ba31e0469fbe615a78197f68f
52b7c',
...
'2e6d722e5e4dbdf2447ddecc9f7dabb8e299bae921c99ad5b0184cd9eb8
e5908',
...
'b13a750047bc0bdceb2473e5fe488c2596d7a7124b4e716fdd29b046ef9
9bbf0',
... ]
>>> hashes = [bytes.fromhex(x) for x in hex_hashes]
>>> current_hashes = hashes
>>> while len(current_hashes) > 1:
...     current_hashes = merkle_parent_level(current_hashes)
>>> print(current_hashes[0].hex())

acbcab8bcc1af95d8d563b77d24c3d19b18f1486383d75a5085c4e86c86b
eed6
```



ریشه‌ی مرکل در بلوک‌ها

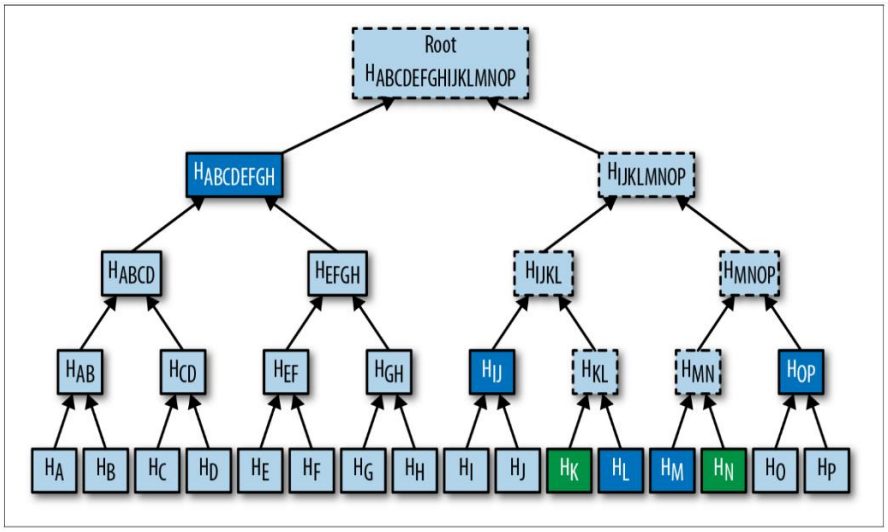
```
>>> from helper import merkle_root
>>> tx_hex_hashes = [
...
'42f6f52f17620653dcc909e58bb352e0bd4bd1381e2955d19c00959a221
22b2e',
...
'94c3af34b9667bf787e1c6a0a009201589755d01d02fe2877cc69b929d2
418d4',
...
'959428d7c48113cb9149d0566bde3d46e98cf028053c522b8fa8f735241
aa953',
...
'a9f27b99d5d108dede755710d4a1ffa2c74af70b4ca71726fa57d68454e
609a2',
...
'62af110031e29de1efcad103b3ad4bec7bdcf6cb9c9f4afdd5869817955
16577',
...
'766900590ece194667e9da2984018057512887110bf54fe0aa800157aec
796ba',
...
'e8270fb475763bc8d855cfe45ed98060988c1bdcad2ffc8364f783c9899
9a208',
... ]
>>> tx_hashes = [bytes.fromhex(x) for x in tx_hex_hashes]
>>> hashes = [h[::-1] for h in tx_hashes]

>>> print(merkle_root(hashes)[::-1].hex())


654d6181e18e4ac4368383fdc5eead11bf138f9b7ac1e15334e4411b3c47
97d9
```



اضافه کردن پارامتر tx_hashes:

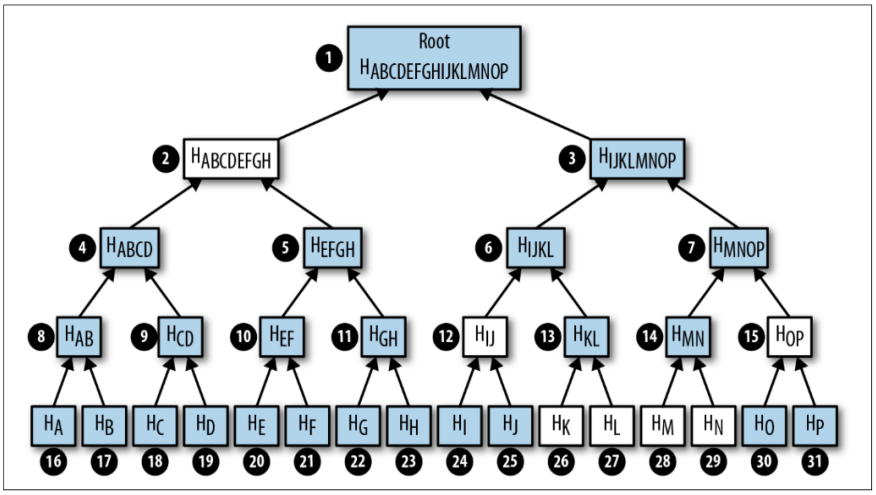
```
class Block:

    def __init__(self, version, prev_block, merkle_root,
                 timestamp, bits, nonce, tx_hashes=None):
        self.version = version
        self.prev_block = prev_block
        self.merkle_root = merkle_root
        self.timestamp = timestamp
        self.bits = bits
        self.nonce = nonce
        self.tx_hashes = tx_hashes
```

استفاده از درخت مرکل

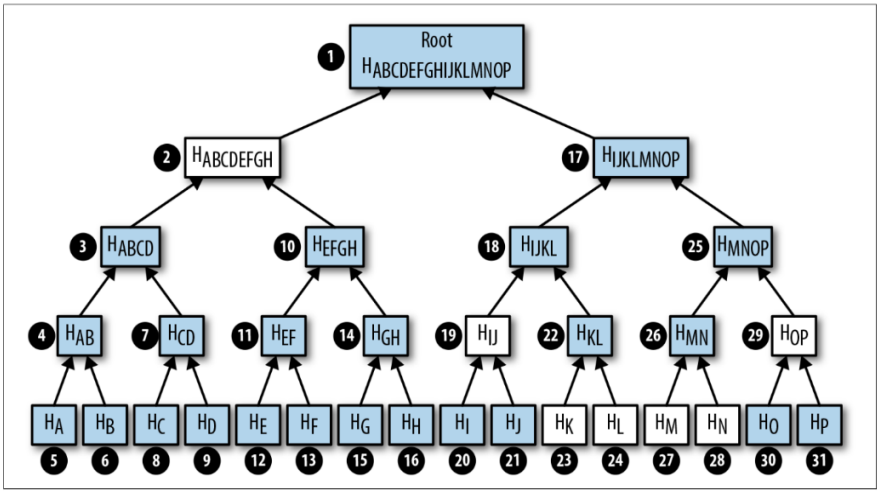


شکل ۱۱-۳: اثبات مرکل

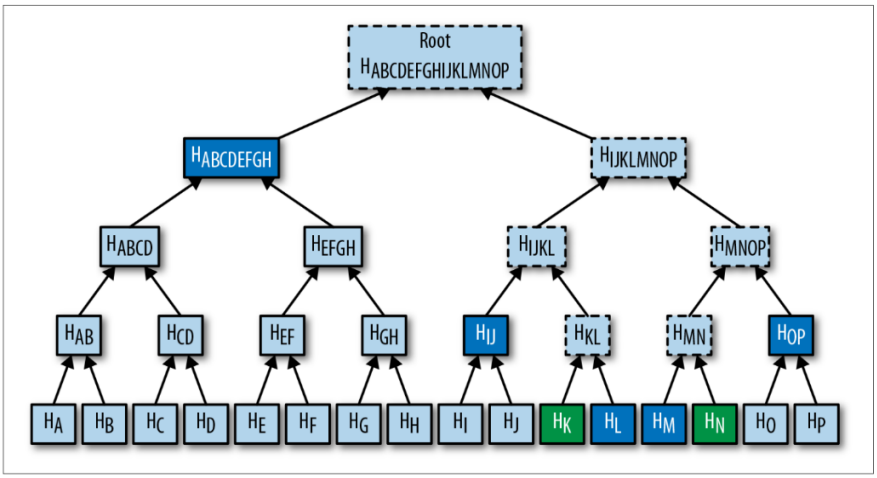
بلوک مرکل



شکل ۱۱-٤: پیمایش اول-عرض



شکل ۱۱-۵: پیمایش اول-عمق



شکل ۱۱-٦: اثبات مرکل

ساختار درخت مرکل

```
>>> import math
>>> total = 16
>>> max_depth = math.ceil(math.log(total, 2))

>>> merkle_tree = []
>>> for depth in range(max_depth + 1):
...     num_items = math.ceil(total / 2**(max_depth -
depth))
...     level_hashes = [None] * num_items
```

1
2
3
2
5

```
...     merkle_tree.append(level_hashes)
>>> for level in merkle_tree:
...     print(level)                                    6
[None]
[None, None]
[None, None, None, None]
[None, None, None, None, None, None, None, None]
[None, None, None, None, None, None, None, None, None, None,
None, None, None,\
 None, None, None]
```

کد کردن درخت مرکل

```
class MerkleTree:

    def __init__(self, total):
        self.total = total
        self.max_depth = math.ceil(math.log(self.total, 2))
        self.nodes = []
        for depth in range(self.max_depth + 1):
            num_items = math.ceil(self.total /
2**(self.max_depth - depth))
            level_hashes = [None] * num_items
            self.nodes.append(level_hashes)
        self.current_depth = 0
        self.current_index = 0

    def __repr__(self):
        result = []
        for depth, level in enumerate(self.nodes):
            items = []
            for index, h in enumerate(level):
                if h is None:
                    short = 'None'
                else:
                    short = '{}...'.format(h.hex()[:8])
                if depth == self.current_depth and index ==
self.current_index:
                    items.append('*{}*'.format(short[:-2]))
                else:
                    items.append('{}'.format(short))
            result.append(', '.join(items))
        return '\n'.join(result)
```

1

2

محاسبه ریشه‌ی مرکل در صورت داشتن هش هر برگ:

```
>>> from merkleblock import MerkleTree
>>> from helper import merkle_parent_level
>>> hex_hashes = [
...
"9745f7173ef14ee4155722d1cbf13304339fd00d900b759c6f9d58579b5
765fb",
...
"5573c8ede34936c29cdfdfe743f7f5fdfbd4f54ba0705259e62f3991706
5cb9b",
...
"82a02ecbb6623b4274dfcab82b336dc017a27136e08521091e443e62582
e8f05",
...
"507ccae5ed9b340363a0e6d765af148be9cb1c8766ccc922f83e4ae6816
58308",
...
"a7a4aec28e7162e1e9ef33dfa30f0bc0526e6cf4b11a576f6c5de585938
98330",
...
"bb6267664bd833fd9fc82582853ab144fece26b7a8a5bf328f8a059445b
59add",
...
"ea6d7ac1ee77fbacee58fc717b990c4fcccf1b19af43103c090f601677f
d8836",
...
"457743861de496c429912558a106b810b0507975a49773228aa788df407
30d41",
...
"7688029288efc9e9a0011c960a6ed9e5466581abf3e3a6c26ee317461ad
d619a",
...
"b1ae7f15836cb2286cdd4e2c37bf9bb7da0a2846d06867a429f654b2e7f
383c9",
...
"9b74f89fa3f93e71ff2c241f32945d877281a6a50a6bf94adac002980aa
fe5ab",
...
"b3a92b5b255019bdaf754875633c2de9fec2ab03e6b8ce669d07cb5b188
04638",
...
"b5c0b915312b9bdaedd2b86aa2d0f8feffc73a2d37668fd9010179261e2
5e263",
...
"c9d52c5cb1e557b92c84c52e7c4bfbce859408bedffc8a5560fd6e35e10
b8800",
...
"c555bc5fc3bc096df0a0c9532f07640bfb76bfe4fc1ace214b8b228a129
7a4c2",
...
"f9dbfafc3af3400954975da24eb325e326960a25b87fffe23eef3e7ed2f
b610e",
... ]
>>> tree = MerkleTree(len(hex_hashes))
```

```
>>> tree.nodes[4] = [bytes.fromhex(h) for h in hex_hashes]
>>> tree.nodes[3] = merkle_parent_level(tree.nodes[4])
>>> tree.nodes[2] = merkle_parent_level(tree.nodes[3])
>>> tree.nodes[1] = merkle_parent_level(tree.nodes[2])
>>> tree.nodes[0] = merkle_parent_level(tree.nodes[1])
>>> print(tree)
*597c4baf.*
6382df3f..., 87cf8fa3...
3ba6c080..., 8e894862..., 7ab01bb6..., 3df760ac...
272945ec..., 9a38d037..., 4a64abd9..., ec7c95e1...,
3b67006c..., 850683df..., \
d40d268b..., 8636b7a3...
9745f717..., 5573c8ed..., 82a02ecb..., 507ccae5...,
a7a4aec2..., bb626766..., \
ea6d7ac1..., 45774386..., 76880292..., b1ae7f15...,
9b74f89f..., b3a92b5b..., \
b5c0b915..., c9d52c5c..., c555bc5f..., f9dbfafc...
```

سایر متدهای مفید:

```
class MerkleTree:
...
    def up(self):
        self.current_depth -= 1
        self.current_index //= 2

    def left(self):
        self.current_depth += 1
        self.current_index *= 2

    def right(self):
        self.current_depth += 1
        self.current_index = self.current_index * 2 + 1

    def root(self):
        return self.nodes[0][0]

    def set_current_node(self, value):
        self.nodes[self.current_depth][self.current_index] =
value

    def get_current_node(self):
        return
self.nodes[self.current_depth][self.current_index]

    def get_left_node(self):
        return self.nodes[self.current_depth +
1][self.current_index * 2]

    def get_right_node(self):
        return self.nodes[self.current_depth +
1][self.current_index * 2 + 1]

    def is_leaf(self):
```

1

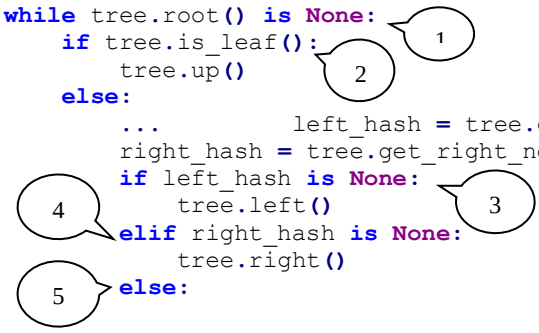
2

```
        return self.current_depth == self.max_depth

    def right_exists(self):
        return len(self.nodes[self.current_depth + 1]) > \
            self.current_index * 2 + 1
```

پر کردن درخت از طریق پیمایش عمق اول با استفاده از متدهای left، right و up:

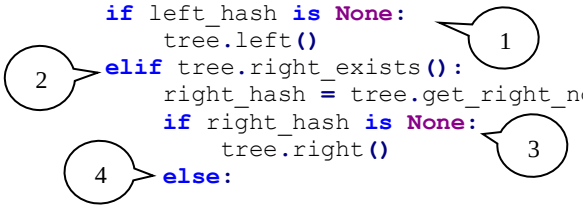
```
>>> from merkleblock import MerkleTree
>>> from helper import merkle_parent
>>> hex_hashes = [
...
"9745f7173ef14ee4155722d1cbf13304339fd00d900b759c6f9d58579b5
765fb",
...
"5573c8ede34936c29cdfdfe743f7f5fdfbd4f54ba0705259e62f3991706
5cb9b",
...
"82a02ecbb6623b4274dfcab82b336dc017a27136e08521091e443e62582
e8f05",
...
"507ccae5ed9b340363a0e6d765af148be9cb1c8766ccc922f83e4ae6816
58308",
...
"a7a4aec28e7162e1e9ef33dfa30f0bc0526e6cf4b11a576f6c5de585938
98330",
...
"bb6267664bd833fd9fc82582853ab144fece26b7a8a5bf328f8a059445b
59add",
...
"ea6d7ac1ee77fbacee58fc717b990c4fcccf1b19af43103c090f601677f
d8836",
...
"457743861de496c429912558a106b810b0507975a49773228aa788df407
30d41",
...
"7688029288efc9e9a0011c960a6ed9e5466581abf3e3a6c26ee317461ad
d619a",
...
"b1ae7f15836cb2286cdd4e2c37bf9bb7da0a2846d06867a429f654b2e7f
383c9",
...
"9b74f89fa3f93e71ff2c241f32945d877281a6a50a6bf94adac002980aa
fe5ab",
...
"b3a92b5b255019bdaf754875633c2de9fec2ab03e6b8ce669d07cb5b188
04638",
...
"b5c0b915312b9bdaedd2b86aa2d0f8feffc73a2d37668fd9010179261e2
5e263",
```

```
...
"c9d52c5cb1e557b92c84c52e7c4bfbce859408bedffc8a5560fd6e35e10
b8800",
...
"c555bc5fc3bc096df0a0c9532f07640bfb76bfe4fc1ace214b8b228a129
7a4c2",
...
"f9dbfafc3af3400954975da24eb325e326960a25b87fffe23eef3e7ed2f
b610e",
... ]
>>> tree = MerkleTree(len(hex_hashes))
>>> tree.nodes[4] = [bytes.fromhex(h) for h in hex_hashes]
>>> while tree.root() is None:
...     if tree.is_leaf():
...         tree.up()
...     else:
...         ...         left_hash = tree.get_left_node()
...         right_hash = tree.get_right_node()
...         if left_hash is None:
...             tree.left()
...         elif right_hash is None:
...             tree.right()
...         else:
...
tree.set_current_node(merkle_parent(left_hash, right_hash))
...             tree.up()
>>> print(tree)
597c4baf...
6382df3f..., 87cf8fa3...
3ba6c080..., 8e894862..., 7ab01bb6..., 3df760ac...
272945ec..., 9a38d037..., 4a64abd9..., ec7c95e1...,
3b67006c..., 850683df..., \
d40d268b..., 8636b7a3...
9745f717..., 5573c8ed..., 82a02ecb..., 507ccae5...,
a7a4aec2..., bb626766..., \
ea6d7ac1..., 45774386..., 76880292..., b1ae7f15...,
9b74f89f..., b3a92b5b..., \
b5c0b915..., c9d52c5c..., c555bc5f..., f9dbfafc...
```



```
>>> from merkleblock import MerkleTree
>>> from helper import merkle_parent
>>> hex_hashes = [
...
"9745f7173ef14ee4155722d1cbf13304339fd00d900b759c6f9d58579b5
765fb",
...
"5573c8ede34936c29cdfdfe743f7f5fdfbd4f54ba0705259e62f3991706
5cb9b",
...
"82a02ecbb6623b4274dfcab82b336dc017a27136e08521091e443e62582
e8f05",
...
"507ccae5ed9b340363a0e6d765af148be9cb1c8766ccc922f83e4ae6816
58308",
...
"a7a4aec28e7162e1e9ef33dfa30f0bc0526e6cf4b11a576f6c5de585938
98330",
...
"bb6267664bd833fd9fc82582853ab144fece26b7a8a5bf328f8a059445b
59add",
...
"ea6d7ac1ee77fbacee58fc717b990c4fcccf1b19af43103c090f601677f
d8836",
...
"457743861de496c429912558a106b810b0507975a49773228aa788df407
30d41",
...
"7688029288efc9e9a0011c960a6ed9e5466581abf3e3a6c26ee317461ad
d619a",
...
"b1ae7f15836cb2286cdd4e2c37bf9bb7da0a2846d06867a429f654b2e7f
383c9",
...
"9b74f89fa3f93e71ff2c241f32945d877281a6a50a6bf94adac002980aa
fe5ab",
...
"b3a92b5b255019bdaf754875633c2de9fec2ab03e6b8ce669d07cb5b188
04638",
...
"b5c0b915312b9bdaedd2b86aa2d0f8feffc73a2d37668fd9010179261e2
5e263",
...
"c9d52c5cb1e557b92c84c52e7c4bfbce859408bedffc8a5560fd6e35e10
b8800",
...
"c555bc5fc3bc096df0a0c9532f07640bfb76bfe4fc1ace214b8b228a129
7a4c2",
...
"f9dbfafc3af3400954975da24eb325e326960a25b87fffe23eef3e7ed2f
b610e",
...
"38faf8c811988dff0a7e6080b1771c97bcc0801c64d9068cffb85e6e7aa
caf51",
... ]
>>> tree = MerkleTree(len(hex_hashes))
```

```
>>> tree.nodes[5] = [bytes.fromhex(h) for h in hex_hashes]
>>> while tree.root() is None:
...     if tree.is_leaf():
...         tree.up()
...     else:
...         left_hash = tree.get_left_node()
...         if left_hash is None:
...             tree.left()
...         elif tree.right_exists():
...             right_hash = tree.get_right_node()
...             if right_hash is None:
...                 tree.right()
...         else:
...
tree.set_current_node(merkle_parent(left_hash, right_hash))
...                 tree.up()
...         else:
...
tree.set_current_node(merkle_parent(left_hash, left_hash))
...                 tree.up()
>>> print(tree)
0a313864...
597c4baf..., 6f8a8190...
6382df3f..., 87cf8fa3..., 5647f416...
3ba6c080..., 8e894862..., 7ab01bb6..., 3df760ac...,
28e93b98...
272945ec..., 9a38d037..., 4a64abd9..., ec7c95e1...,
3b67006c..., 850683df..., \
d40d268b..., 8636b7a3..., ce26d40b...
9745f717..., 5573c8ed..., 82a02ecb..., 507ccae5...,
a7a4aec2..., bb626766..., \
ea6d7ac1..., 45774386..., 76880292..., b1ae7f15...,
9b74f89f..., b3a92b5b..., \
b5c0b915..., c9d52c5c..., c555bc5f..., f9dbfafc...,
38faf8c8...
```



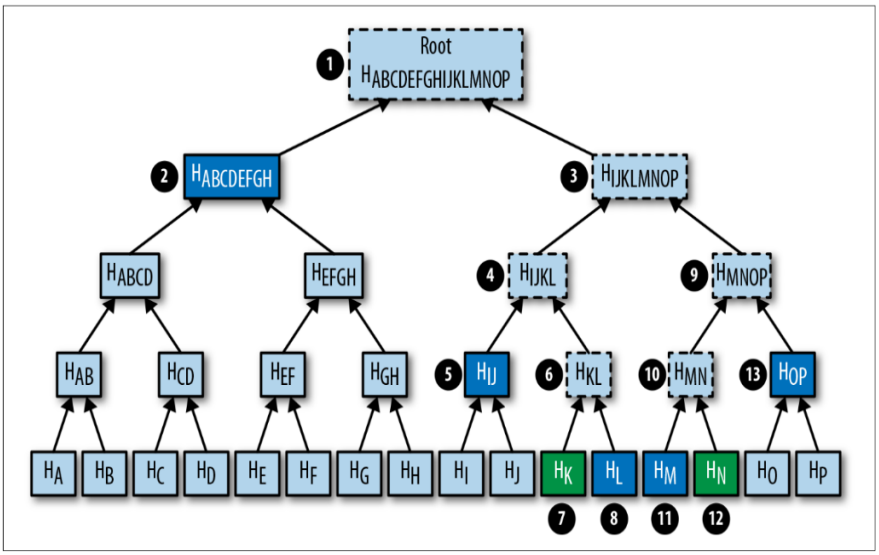
فرماندهی بلوک مرکل

```
00000020df3b053dc46f162a9b00c7f0d5124e2676d47bbe7c5d0793a500000000000000ef445fef2
ed495c275892206ca533e7411907971013ab83e3b47bd0d692d14d4dc7c835b67d8001ac157e670bf
0d00000aba412a0d1480e370173072c9562becffe87aa661c1e4a6dbc305d38ec5dc088a7cf92e645
8aca7b32edae818f9c2c98c37e06bf72ae0ce80649a38655ee1e27d34d9421d940b16732f24b94023
e9d572a7f9ab8023434a4feb532d2adfc8c2c2158785d1bd04eb99df2e86c54bc13e1398628972174
00def5d72c280222c4cbaee7261831e1550dbb8fa82853e9fe506fc5fda3f7b919d8fe74b6282f927
63cef8e625f977af7c8619c32a369b832bc2d051ecd9c73c51e76370ceabd4f25097c256597fa898d
404ed53425de608ac6bfe426f6e2bb457f1c554866eb69dcb8d6bf6f880e9a59b3cd053e6c7060eea
caacf4dac6697dac20e4bd3f38a2ea2543d1ab7953e3430790a9f81e1c67f5b58c825acf46bd02848
384eebe9af917274cdfbb1a28a5d58a23a17977def0de10d644258d9c54f886d47d293a411cb62261
03b55635

        - 00000020 - version, 4 bytes, LE
        - df3b...00 - previous block, 32 bytes, LE
        - ef44...d4 - Merkle root, 32 bytes, LE
        - dc7c835b - timestamp, 4 bytes, LE
        - 67d8001a - bits, 4 bytes
        - c157e670 - nonce, 4 bytes
        - bf0d0000 - number of total transactions, 4 bytes, LE
        - 0a - number of hashes, varint
        - ba41...61 - hashes, 32 bytes * number of hashes
        - 03b55635 - flag bits
```

شکل ۱۱-۷: تجزیه‌ی بلوک مرکل

```
def bytes_to_bit_field(some_bytes):
    flag_bits = []
    for byte in some_bytes:
        for _ in range(8):
            flag_bits.append(byte & 1)
            byte >>= 1
    return flag_bits
```

استفاده از بیت‌های پرچم و هش ها



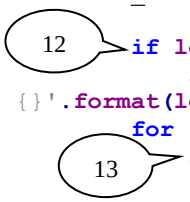
شکل ۱۱-۸: پردازش بلوک مرکل

محاسبه ریشه‌ی درخت مرکل با استفاده از بیت‌های پرچم مناسب و هش ها:

```
class MerkleTree:
...
    def populate_tree(self, flag_bits, hashes):
        while self.root() is None:
            if self.is_leaf():
                flag_bits.pop(0)
                self.set_current_node(hashes.pop(0))
                self.up()
            else:
                left_hash = self.get_left_node()
                if left_hash is None:
                    if flag_bits.pop(0) == 0:
                        self.set_current_node(hashes.pop(0))
                        self.up()
                    else:
                        self.left()
                elif self.right_exists():
                    right_hash = self.get_right_node()
                    if right_hash is None:
                        self.right()
                    else:
                        self.set_current_node(merkle_parent(left_hash,
                                              right_hash))
                        self.up()
                else:
```

1 2 3 4 5 6 7 8 9 10 11

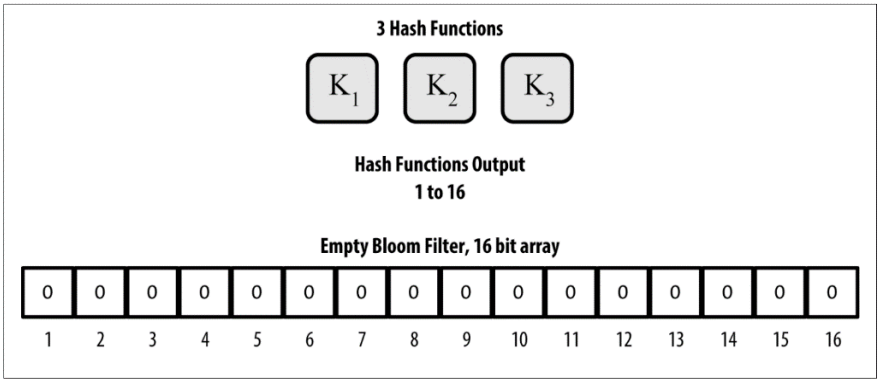
```
self.set_current_node(merkle_parent(left_hash, left_hash))
                    self.up()
        if len(hashes) != 0:
            raise RuntimeError('hashes not all consumed
{}'.format(len(hashes)))
        for flag_bit in flag_bits:
            if flag_bit != 0:
                raise RuntimeError('flag bits not all
consumed')
```



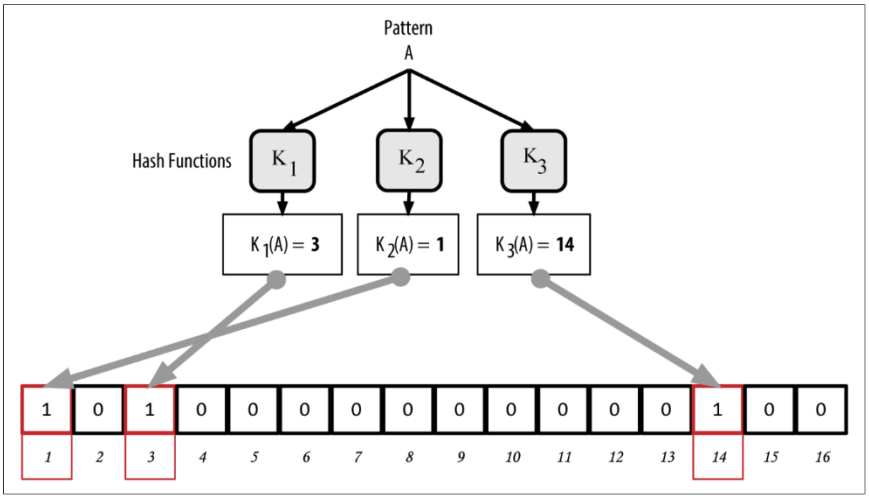
فصل دوازدهم

بلوم‌فیلترها

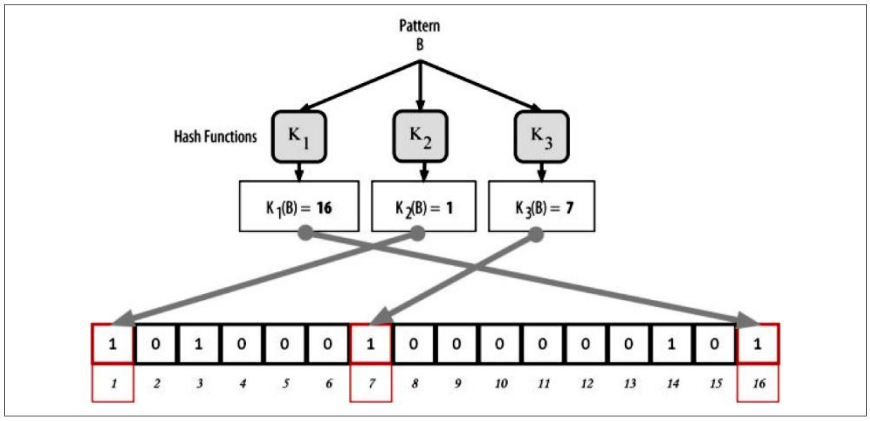
بلوم‌فیلتر چگونه کار می‌کند؟



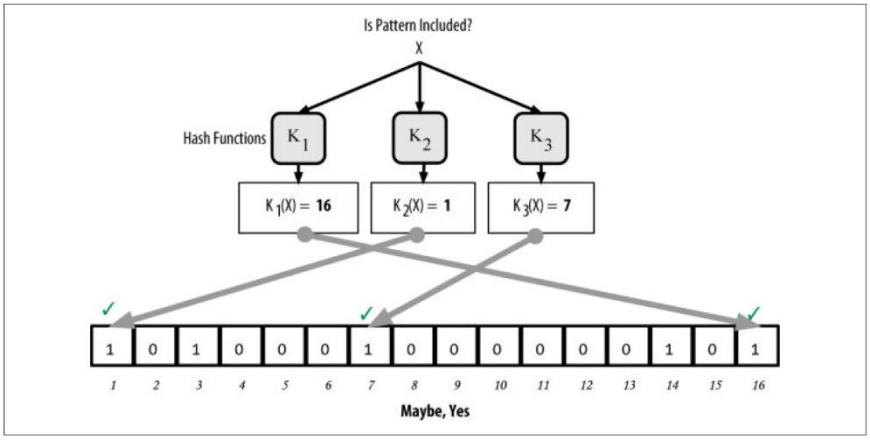
شکل ۱۲-۱: ساختار کلی یک بلوم فیلتر



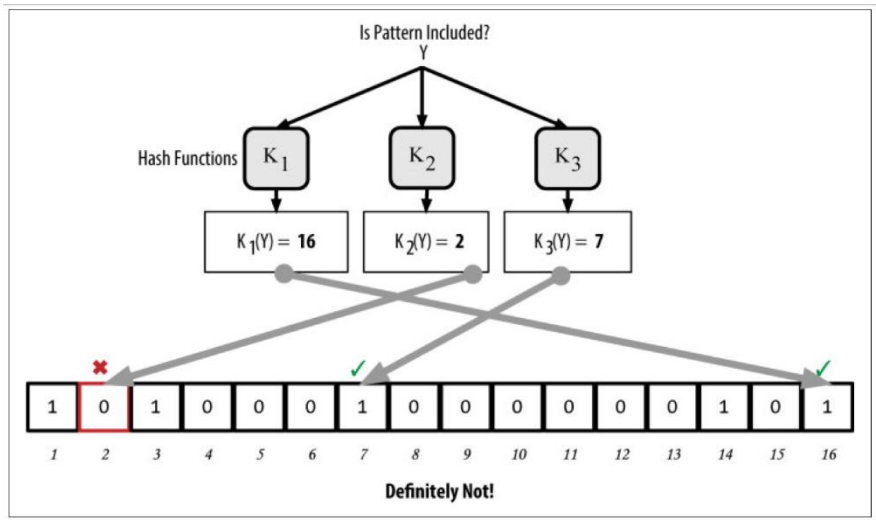
شکل ۱۲-۲: افزودن الگوی "A" به بلوم فیلتر ساده



شکل ۱۲-۳: افزودن الگوی "B" به بلوم فیلتر ساده



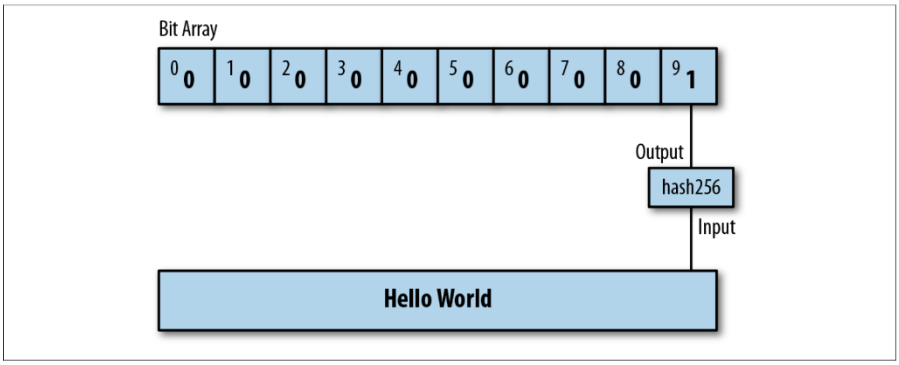
شکل ۱۲-٤: آزمایش وجود الگوی x در بلوم فیلتر ساده



شکل ۱۲-۵: آزمایش وجود الگوی Y در بلوم فیلتر ساده

```
>>> from helper import hash256
>>> bit_field_size = 10
>>> bit_field = [0] * bit_field_size
>>> h = hash256(b'hello world')
>>> bit = int.from_bytes(h, 'big') % bit_field_size
>>> bit_field[bit] = 1
>>> print(bit_field)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

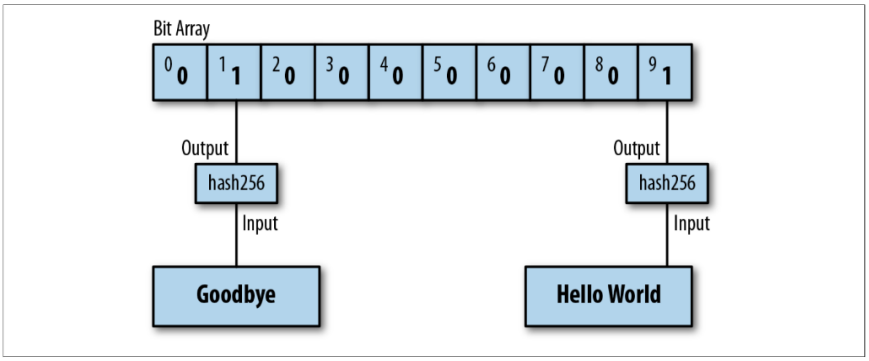
1 2 3 4



شکل ۱۲-٦: بلوم فیلتر ۱۰ بیتی با یک عنصر

```
>>> from helper import hash256
>>> bit_field_size = 10
>>> bit_field = [0] * bit_field_size
>>> for item in (b'hello world', b'goodbye'):
...     h = hash256(item)
...     bit = int.from_bytes(h, 'big') % bit_field_size
...     bit_field[bit] = 1
>>> print(bit_field)
[0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
```

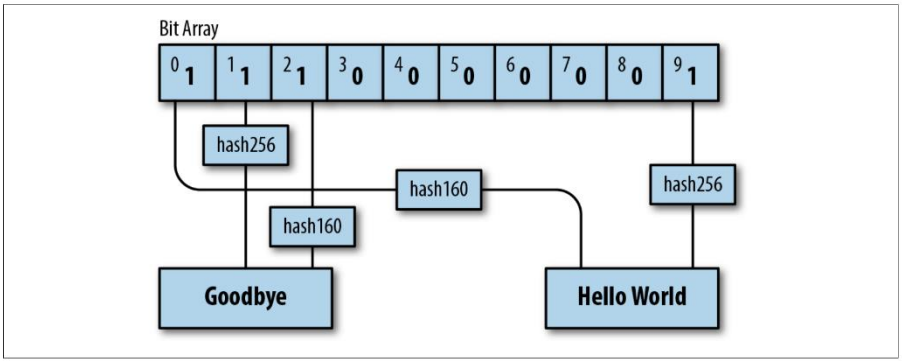
1



شکل ۱۲-۷: *بلوم فیلتر* ۱۰ بیتی با دو عنصر

```
>>> from helper import hash256, hash160
>>> bit_field_size = 10
>>> bit_field = [0] * bit_field_size
>>> for item in (b'hello world', b'goodbye'):
...     for hash_function in (hash256, hash160):
...         h = hash_function(item)
...         bit = int.from_bytes(h, 'big') % bit_field_size
...         bit_field[bit] = 1
>>> print(bit_field)
[1, 1, 1, 0, 0, 0, 0, 0, 0, 1]
```

1



شکل ۱۲-۸: بلوم فیلتر ۱۰ بیتی با دو عنصر و دو تابع هش

بلوم‌فیلترهای BIP0037

```
>>> from helper import murmur3
>>> from bloomfilter import BIP37_CONSTANT
>>> field_size = 2
>>> num_functions = 2
>>> tweak = 42
>>> bit_field_size = field_size * 8
>>> bit_field = [0] * bit_field_size
>>> for phrase in (b'hello world', b'goodbye'):
...     for i in range(num_functions):
...         seed = i * BIP37_CONSTANT + tweak
...         h = murmur3(phrase, seed=seed)
...         bit = h % bit_field_size
...         bit_field[bit] = 1
>>> print(bit_field)
[0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0]
```

1 2 3 4 5 6

کد نویسیِ کلاس BloomFilter:

```
class BloomFilter:

    def __init__(self, size, function_count, tweak):
        self.size = size
        self.bit_field = [0] * (size * 8)
        self.function_count = function_count
        self.tweak = tweak
```

بارگذاری[1] فیلتر بلوم

```
0a4000600a080000010940050000006300000000

- 0a4000600a080000010940 - Bit field, variable field
- 05000000 - Hash count, 4 bytes, little-endian
- 63000000 - Tweak, 4 bytes, little-endian
- 00 - Matched item flag
```

شکل ۱۲–۹: تجزیه‌ی بارگذار فیلتر

دریافت بلوک‌های مرکل

```
02030000003 0eb2540c41025690160a1014c577061596e32e426b712c7ca000
0000000000030000001049847939585b0652fba793661c361223446b6fc410
89b8be00000000000000

- 02 - Number of data items
- 03000000 - Type of data item (tx, block, filtered
      block, compact block), little-endian
- 30...00 - Hash identifier
```

شکل ۱۲–۱۰: getdata تجزیه‌شده

```
class GetDataMessage:
    command = b'getdata'

    def __init__(self):
        self.data = []

    def add_data(self, data_type, identifier):
        self.data.append((data_type, identifier))
```

1

2

[1] Loading

دریافت تراکنش‌های مورد نظر

```
>>> from bloomfilter import BloomFilter
>>> from helper import decode_base58
>>> from merkleblock import MerkleBlock
>>> from network import FILTERED_BLOCK_DATA_TYPE,
GetHeadersMessage, GetDataMe\
ssage, HeadersMessage, SimpleNode
>>> from tx import Tx
>>> last_block_hex =
'00000000000538d5c2246336644f9a4956551afb44ba47278759ec55\
ea912e19'
>>> address = 'mwJn1YPMq7y5F8J3LkC5Hxg9PHyZ5K4cFv'
>>> h160 = decode_base58(address)
>>> node = SimpleNode('testnet.programmingbitcoin.com',
testnet=True, logging=\
False)
>>> bf = BloomFilter(size=30, function_count=5, tweak=90210)
>>> bf.add(h160)

>>> node.handshake()
>>> node.send(bf.filterload())
>>> start_block = bytes.fromhex(last_block_hex)
>>> getheaders = GetHeadersMessage(start_block=start_block)

>>> node.send(getheaders)
>>> headers = node.wait_for(HeadersMessage)
>>> getdata = GetDataMessage()
>>> for b in headers.blocks:
...     if not b.check_pow():
...         raise RuntimeError('proof of work is invalid')
...     getdata.add_data(FILTERED_BLOCK_DATA_TYPE, b.hash())
>>> node.send(getdata)
>>> found = False
>>> while not found:
...     message = node.wait_for(MerkleBlock, Tx)
...     if message.command == b'merkleblock':
...         if not message.is_valid():
...             raise RuntimeError('invalid merkle proof')
...     else:
...         for i, tx_out in enumerate(message.tx_outs):
...             if
tx_out.script_pubkey.address(testnet=True) == address:
...                 print('found:
{}:{}'.format(message.id(), i))
...                 found = True
...                 break
found:
e3930e1e566ca9b75d53b0eb9acb7607f547e1182d1d22bd4b661cfe18dc
ddf1:0
```

1 2 3 4 5 6 7 8 9 10

فصل سیزدهم

Segwit

تراکنش‌های p2wpkh

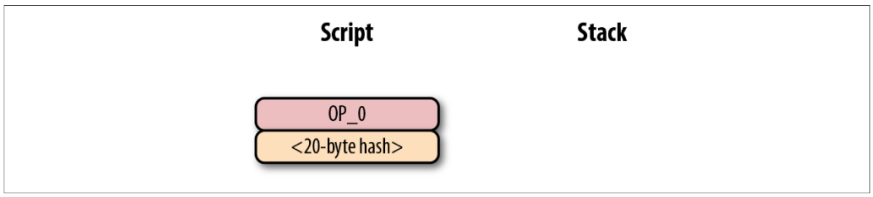
```
010000000115e180dc28a2327e687facc33f10f2a20da717e5548406f7ae8b4c811072f8560100000
000ffffffff0100b4f505000000001976a9141d7cd6c75c2e86f4cbf98eaed221b30bd9a0b92888ac
00000000

        - 01000000 - version
        - 01 - # of inputs
        - 15e1...56 - previous tx hash
        - 01000000 - previous tx index
        - 00 - ScriptSig
        - ffffffff - sequence
        - 01 - # of outputs
        - 00b4...00 - output amounts
        - 1976...ac - ScriptPubKey
        - 00000000 - locktime
```

شکل ۱۳-۱: p2wpkh به‌گونه‌ای که توسط نرم‌افزار pre-BIP0141 مشاهده می‌شود.

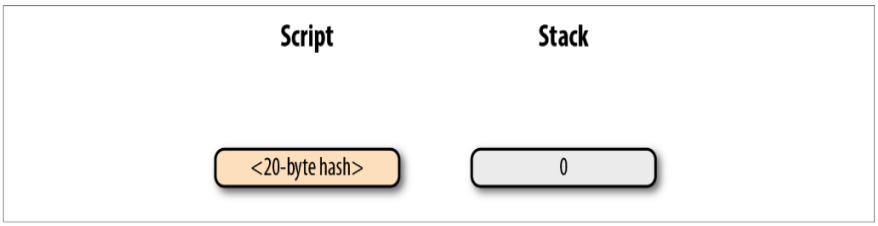
```
010000000001010115e180dc28a2327e687facc33f10f2a20da717e5548406f7ae8b4c811072f856010
00000000ffffffff0100b4f505000000001976a9141d7cd6c75c2e86f4cbf98eaed221b30bd9a0b928
88ac02483045022100df7b7e5cda14ddf91290e02ea10786e03eb11ee36ec02dd862fe9a326bbcb7f
d02203f5b4496b667e6e281cc654a2da9e4f08660c620a1051337fa8965f727eb19190121038262a6
c6cec93c2d3ecd6c6072efea86d02ff8e3328bbd0242b20af3425990ac00000000

        - 01000000 - version
        - 00 - Segwit marker
        - 01 - Segwit flag
        - 01 - # of inputs
        - 15e1...56 - previous tx hash
        - 01000000 - previous tx index
        - 00 - ScriptSig
        - ffffffff - sequence
        - 01 - # of outputs
        - 00b4...00 - output amounts
        - 1976...ac - ScriptPubKey
        - 0248...61 - witness
        - 00000000 - locktime
```

شکل ۱۳-۲: p2wpkh به‌گونه‌ای که توسط نرم‌افزار post-BIP0141 مشاهده می‌شود.

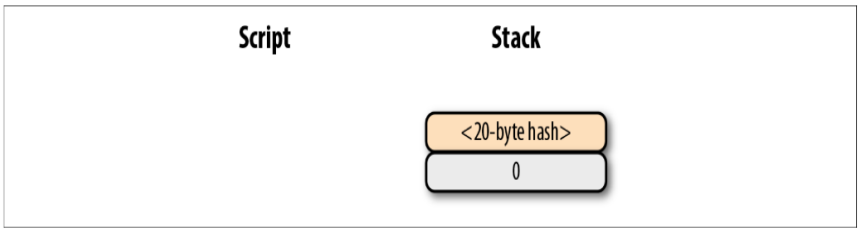
شکل ۱۳-۳: p2wpkh ScriptPubKey



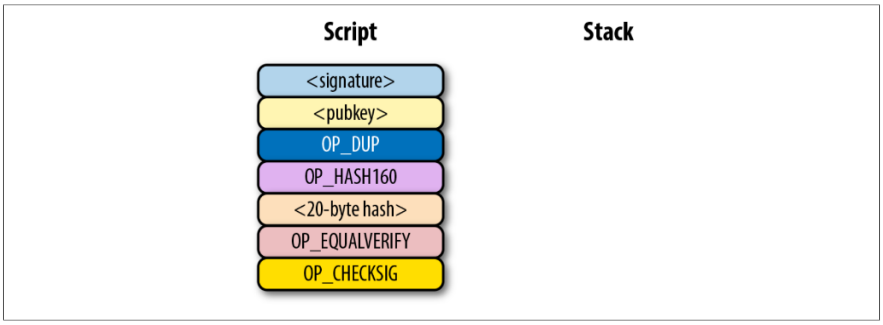
شکل ۱۳–٤: شروع p2wpkh



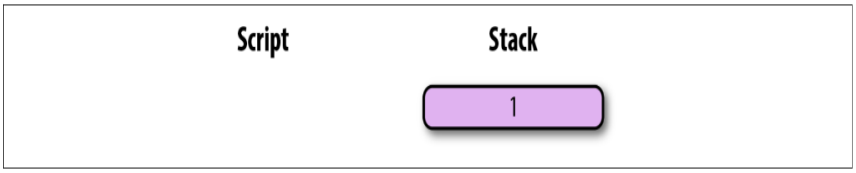
شکل ۱۳–٥: مرحله‌ی اول p2wpkh



شکل ۱۳–٦: مرحله‌ی دوم p2wpkh



شکل ۱۳-۷: مرحله‌ی سوم p2wpkh



شکل ۱۳-۸: مرحله‌ی چهارم p2wpkh

p2sh-p2wpkh

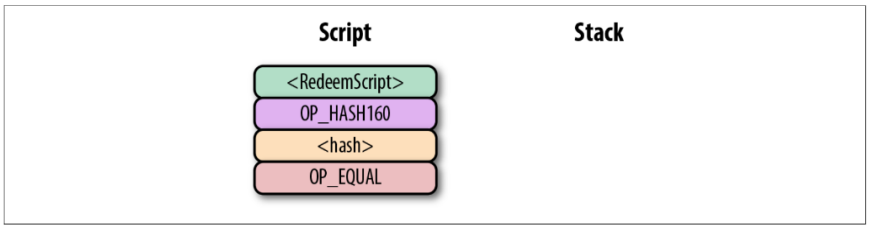
```
0100000001712e5b4e97ab549d50ca60a4f5968b2225215e9fab82dae4720078711406972f0000000
017160014848202fc47fb475289652fbd1912cc853ecb0096feffffff02323600000000000001976a9
14121ae7a2d55d2f0102ccc117cbcb70041b0e037f88ac1027000000000000001976a914ec0be509516
51261765cfa71d7bd41c7b9245bb388ac075a0700

        - 01000000 - version
        - 01 - # of inputs
        - 712e...2f - previous tx hash
        - 00000000 - previous tx index
        - 1716...96 - ScriptSig
        - feffffff - sequence
        - 02 - # of outputs
        - 3236...00 - output amounts
        - 1976...ac - ScriptPubKey
        - 075a0700 - locktime
```

شکل ۱۳-۹: p2sh-p2wpkh to pre-BIP0141 software

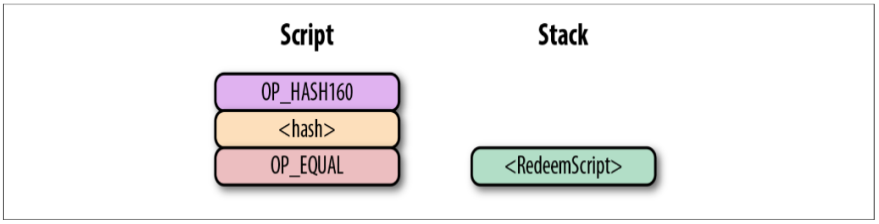
```
01000000000101712e5b4e97ab549d50ca60a4f5968b2225215e9fab82dae4720078711406972f000
000001716001484820 2fc47fb475289652fbd1912cc853ecb0096feffffff0232360000000000001 9
76a914121ae7a2d55d2f0102ccc117cbcb70041b0e037f88ac1027000000000000 1976a914ec0be50
951651261765cfa71d7bd41c7b9245bb388ac02483045022100926 3c7de80c297d5b21aba846cf6f0
a970e1d339568167d1e4c1355c7711bc1602202c9312b8d32fd9c7acc54c46cab50eb7255ce3c0122
14c41fe1ad91bccb16a13012102ebdf6fc448431a2bd6380f912a0fa6ca291ca3340e79b6f0c1fdaf
f73cf54061075a0700

        - 01000000 - version
        - 00 - Segwit marker
        - 01 - Segwit flag
        - 01 - # of inputs
        - 712e...2f - previous tx hash
        - 00000000 - previous tx index
        - 1716...96 - ScriptSig
        - feffffff - sequence
        - 02 - # of outputs
        - 3236...00 - output amounts
        - 1976...ac - ScriptPubKey
        - 0248...61 - witness
        - 075a0700 - locktime
```

شکل ۱۳-۱۰: p2sh-p2wpkh to post-BIP0141 software

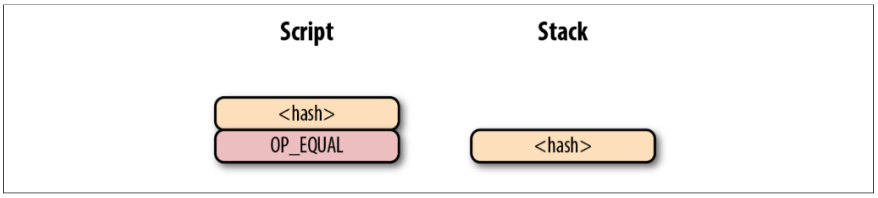
شکل ۱۳-۱۱: p2sh-p2wpkh ScriptPubKey شبیه همان p2sh ScriptPubKey نرمال است.



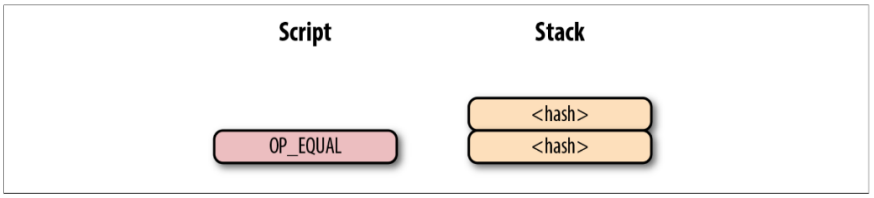
شکل ۱۳-۱۲: شروع p2sh-p2wpkh



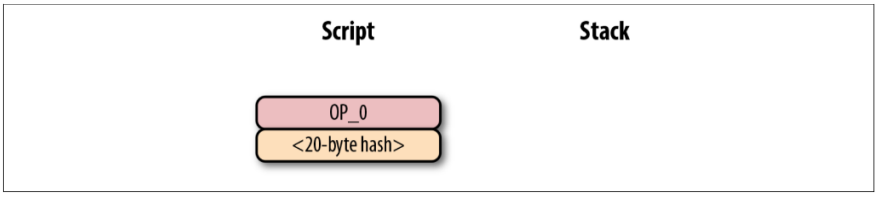
شکل ۱۳-۱۳: مرحله‌ی اول p2sh-p2wpkh



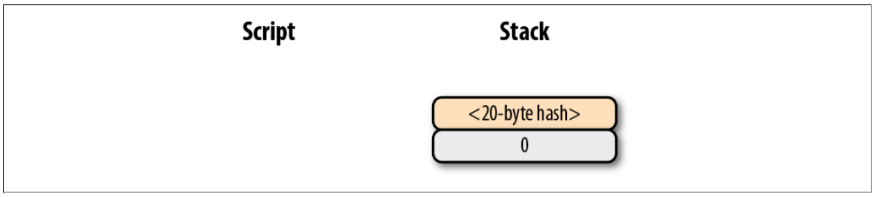
شکل ۱۳-۱٤: مرحله‌ی دوم p2sh-p2wpkh



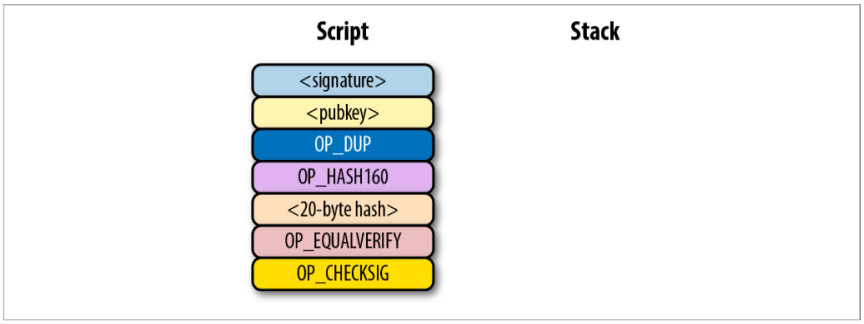
شکل ۱۳-۱۵: مرحله‌ی سوم p2sh-p2wpkh



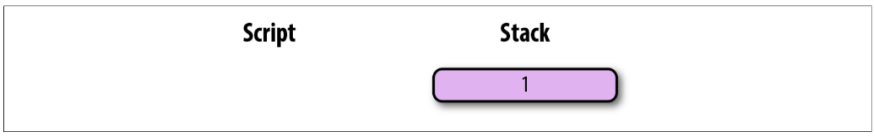
شکل ۱۳-۱٦: مرحله‌ی چهارم p2sh-p2wpkh



شکل ۱۳-۱۷: مرحله‌ی پنجم p2sh-p2wpkh



شکل ۱۳-۱۸: مرحله‌ی ششم p2sh-p2wpkh



شکل ۱۳-۱۹: پایان فرایند p2sh-p2wpkh

کدگذاری p2wpkh و p2sh-p2wpkh

```
class Tx:
    command = b'tx'

    def __init__(self, version, tx_ins, tx_outs,
        locktime, testnet=False, segwit=False):
        self.version = version
        self.tx_ins = tx_ins
        self.tx_outs = tx_outs
        self.locktime = locktime
        self.testnet = testnet
        self.segwit = segwit
        self._hash_prevouts = None
```

```
        self._hash_sequence = None
        self._hash_outputs = None
```

تغییر متد parse:

```
class Tx:
...
    @classmethod
    def parse(cls, s, testnet=False):
        s.read(4)
        if s.read(1) == b'\x00':
            parse_method = cls.parse_segwit
        else:
            parse_method = cls.parse_legacy
        s.seek(-5, 1)
        return parse_method(s, testnet=testnet)

    @classmethod
    def parse_legacy(cls, s, testnet=False):
        version = little_endian_to_int(s.read(4))
        num_inputs = read_varint(s)
        inputs = []
        for _ in range(num_inputs):
            inputs.append(TxIn.parse(s))
        num_outputs = read_varint(s)
        outputs = []
        for _ in range(num_outputs):
            outputs.append(TxOut.parse(s))
        locktime = little_endian_to_int(s.read(4))
        return cls(version, inputs, outputs, locktime,
                   testnet=testnet, segwit=False)
```

بخش تجزیه‌کننده یا parser، برای پیاپی‌سازی Segwit:

```
class Tx:
...
    @classmethod
    def parse_segwit(cls, s, testnet=False):
        version = little_endian_to_int(s.read(4))
        marker = s.read(2)
        if marker != b'\x00\x01':
            raise RuntimeError('Not a segwit transaction
{}'.format(marker))
        num_inputs = read_varint(s)
        inputs = []
        for _ in range(num_inputs):
            inputs.append(TxIn.parse(s))
        num_outputs = read_varint(s)
        outputs = []
        for _ in range(num_outputs):
            outputs.append(TxOut.parse(s))
        for tx_in in inputs:
            num_items = read_varint(s)
            items = []
```

```
            for _ in range(num_items):
                item_len = read_varint(s)
                if item_len == 0:
                    items.append(0)
                else:
                    items.append(s.read(item_len))
            tx_in.witness = items
        locktime = little_endian_to_int(s.read(4))
        return cls(version, inputs, outputs, locktime,
                   testnet=testnet, segwit=True)
```

تغییرات مربوط به روش‌های پیاپی‌سازی:

```
class Tx:
...
    def serialize(self):
        if self.segwit:
            return self.serialize_segwit()
        else:
            return self.serialize_legacy()

    def serialize_legacy(self):
        result = int_to_little_endian(self.version, 4)
        result += encode_varint(len(self.tx_ins))
        for tx_in in self.tx_ins:
            result += tx_in.serialize()
        result += encode_varint(len(self.tx_outs))
        for tx_out in self.tx_outs:
            result += tx_out.serialize()
        result += int_to_little_endian(self.locktime, 4)
        return result

    def serialize_segwit(self):
        result = int_to_little_endian(self.version, 4)
        result += b'\x00\x01'
        result += encode_varint(len(self.tx_ins))
        for tx_in in self.tx_ins:
            result += tx_in.serialize()
        result += encode_varint(len(self.tx_outs))
        for tx_out in self.tx_outs:
            result += tx_out.serialize()
        for tx_in in self.tx_ins:
            result +=
int_to_little_endian(len(tx_in.witness), 1)
            for item in tx_in.witness:
                if type(item) == int:
                    result += int_to_little_endian(item, 1)
                else:
                    result += encode_varint(len(item)) +
item
        result += int_to_little_endian(self.locktime, 4)
        return result
```

1

2

3

```
class Tx:
...
    def hash(self):
        '''Binary hash of the legacy serialization'''
        return hash256(self.serialize_legacy())[::-1]
```

```
class Tx:
...
    def verify_input(self, input_index):
        tx_in = self.tx_ins[input_index]
        script_pubkey =
tx_in.script_pubkey(testnet=self.testnet)
        if script_pubkey.is_p2sh_script_pubkey():
            command = tx_in.script_sig.commands[-1]
            raw_redeem = int_to_little_endian(len(command),
1) + command
            redeem_script =
Script.parse(BytesIO(raw_redeem))
            if redeem_script.is_p2wpkh_script_pubkey():
                z = self.sig_hash_bip143(input_index,
redeem_script)
                witness = tx_in.witness
            else:
                z = self.sig_hash(input_index,
redeem_script)
                witness = None
        else:
            if script_pubkey.is_p2wpkh_script_pubkey():
                z = self.sig_hash_bip143(input_index)
                witness = tx_in.witness
            else:
                z = self.sig_hash(input_index)
                witness = None
        combined_script = tx_in.script_sig +
tx_in.script_pubkey(self.testnet)
        return combined_script.evaluate(z, witness)
```

1

2

3

4

```
def p2wpkh_script(h160):
    '''Takes a hash160 and returns the p2wpkh
ScriptPubKey'''
    return Script([0x00, h160])
...
    def is_p2wpkh_script_pubkey(self):
        return len(self.cmds) == 2 and self.cmds[0] == 0x00
\
            and type(self.cmds[1]) == bytes and
len(self.cmds[1]) == 20
```

1

2

```
class Script:
...
    def evaluate(self, z, witness):
    ...
        while len(commands) > 0:
        ...
            else:
                stack.append(command)
 ...            1
                if len(stack) == 2 and stack[0] == b'' and
len(stack[1]) == 20:
                    h160 = stack.pop()
                    stack.pop()
                    cmds.extend(witness)
                    cmds.extend(p2pkh_script(h160).cmds)
```

Pay-to-Witness-Script-Hash (p2wsh)

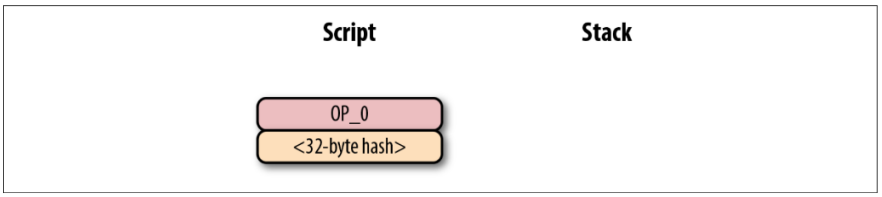
```
0100000001593a2db37b841b2a46f4e9bb63fe9c1012da3ab7fe30b9f9c974242778b5af898000000
0000ffffffff01806fb30700000000001976a914bbef244bcad13cffb68b5cef3017c7423675552288a
c00000000

        - 01000000 - version
        - 01 - # of inputs
        - 593a...98 - previous tx hash
        - 00000000 - previous tx index
        - 00 - ScriptSig
        - ffffffff - sequence
        - 01 - # of outputs
        - 806f...00 - output amounts
        - 1976...ac - ScriptPubKey
        - 00000000 - locktime
```

شکل ۱۳-۲۰: pay-to-witness-script-hash به‌گونه‌ای که توسط نرم‌افزار pre-BIP0141 مشاهده می‌شود.

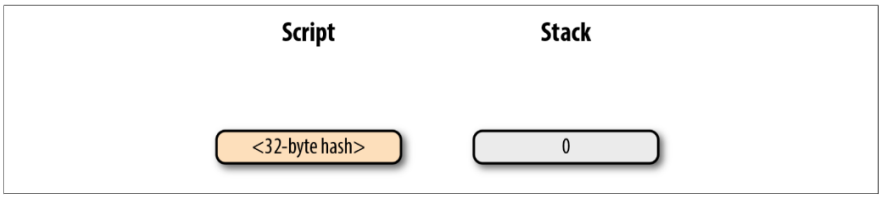
```
010000000001015 93a2db37b841b2a46f4e9bb63fe9c1012da3ab7fe30b9f9c974242778b5af898000000000ffffffff01806fb307000000001976a914bbef244bcad13cffb68b5cef3017c7423675552288ac040047304402203cdcaf02a44e37e409646e8a506724e9e1394b890cb52429ea65bac4cc2403f1022024b934297bcd0c21f22cee0e48751c8b184cc3a0d704cae2684e14858550af7d01483045022100feb4e1530c13e72226dc912dcd257df90d81ae22dbddb5a3c2f6d86f81d47c8e022069889ddb76388fa7948aaa018b2480ac36132009bb9cfade82b651e88b4b137a01695221026ccfb8061f235cc110697c0bfb3afb99d82c886672f6b9b5393b25a434c0cbf32103befa190c0c22e2f53720b1be9476dcf11917da4665c44c9c71c3a2d28a933c352102be46dc245f58085743b1cc37c82f0d63a960efa43b5336534275fc469b49f4ac53ae00000000

        - 01000000 - version
        - 00 - Segwit marker
        - 01 - Segwit flag
        - 01 - # of inputs
        - 593a...98 - previous tx hash
        - 00000000 - previous tx index
        - 00 - ScriptSig
        - ffffffff - sequence
        - 01 - # of outputs
        - 806f...00 - output amounts
        - 1976...ac - ScriptPubKey
        - 0248...61 - witness
        - 00000000 - locktime
```

شکل ۱۳–۲۱: pay-to-witness-script-hash، به‌گونه‌ای که توسط نرم‌افزار post-BIP0141 مشاهده می‌شود.

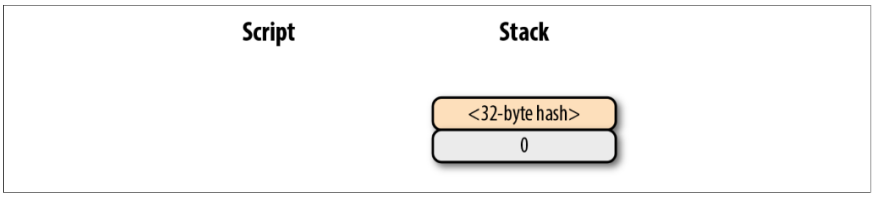
شکل ۱۳–۲۲: p2wsh ScriptPubKey



شکل ۱۳–۲۳: شروع p2sh



شکل ۱۳-۲٤: مرحله‌ی اول p2wsh



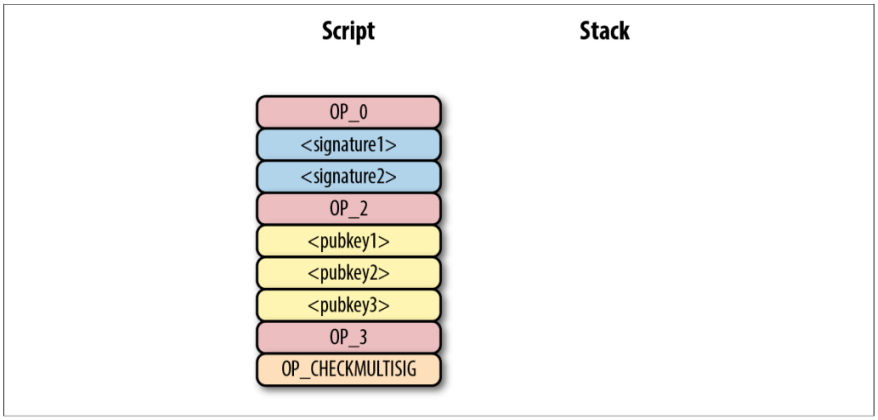
شکل ۱۳-۲۵: مرحله‌ی دوم p2wsh

```
0400473044022203cdcaf02a44e37e409646e8a506724e9e1394b890cb52429ea65bac4cc2403f1022
024b934297bcd0c21f22cee0e48751c8b184cc3a0d704cae2684e14858550af7d0148304502210 0fe
b4e1530c13e72226dc912dcd257df90d81ae22dbddb5a3c2f6d86f81d47c8e022069889ddb76388fa
7948aaa018b2480ac36132009bb9cfade82b651e88b4b137a0169522102 6ccfb8061f235cc110697c
0bfb3afb99d82c886672f6b9b5393b25a434c0cbf32103befa190c0c22e2f53720b1be9476dcf1191
7da4665c44c9c71c3a2d28a933c352102be46dc245f58085743b1cc37c82f0d63a960efa43b533653
4275fc469b49f4ac53ae

        - 04 - Number of witness elements
        - 00 - OP_0
        - 47 - Length of <signaturex>
        - 3044...01 - <signaturex>
        - 69 - Length of WitnessScript
        - 5221...ae - <WitnessScript>
```

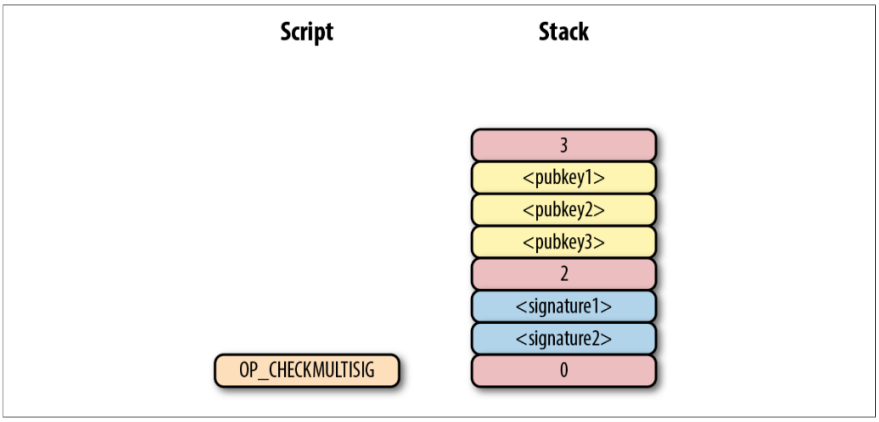
شکل ۱۳-۲٦: p2wsh witness

```
5221026ccfb8061f235cc110697c0bfb3afb99d82c886672f6b9b5393b25a434c0cbf32103befa190
c0c22e2f53720b1be9476dcf11917da4665c44c9c71c3a2d28a933c352102be46dc245f58085743b1
cc37c82f0d63a960efa43b5336534275fc469b49f4ac53ae

        - 52 - OP_2
        - 21 - Length of <pubkeyx>
        - 0...01 - <pubkeyx>
        - 53 - OP_3
        - ae - OP_CHECKMULTISIG
```

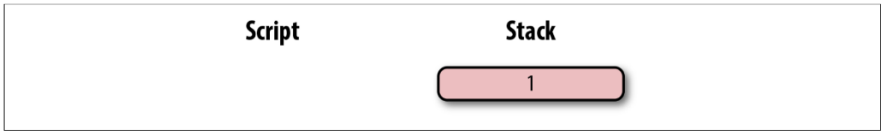
شکل ۱۳-۲۷: p2wsh WitnessScript



شکل ۱۳-۲۸: مرحله‌ی سوم p2wsh



شکل ۱۳-۲۹: مرحله‌ی چهارم p2wsh



شکل ۱۳-۳۰: مرحله‌ی پنجم p2wsh

p2sh-p2wsh

```
010000000170825 6c5896fb3f00ef37601f8e30c5b460dbcd1fca1cd7199f9b56fc4ecd5400
000000232200206 15ae01ed1bc1ffaad54da31d7805d0bb55b52dfd3941114330368c1bbf69
b4cffffffff01603edb0300000000160014bbef244bcad13cffb68b5cef3017c74236755522
00000000

        - 01000000 - version
        - 01 - # of inputs
        - 7082...54 - previous tx hash
        - 00000000 - previous tx index
        - 2322...4c - ScriptSig
        - ffffffff - sequence
        - 01 - # of outputs
        - 603e...00 - output amounts
        - 1600...22 - ScriptPubKey
        - 00000000 - locktime
```

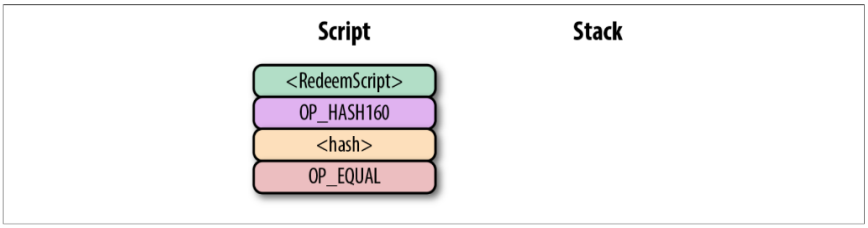
شکل ۱۳-۳۱: p2sh-p2wsh به نرم‌افزار pre-BIP0141

```
010000000001017082 56c5896fb3f00ef37601f8e30c5b460dbcd1fca1cd7199f9b56fc4ecd540000
00002322002061 5ae01ed1bc1ffaad54da31d7805d0bb55b52dfd3941114330368c1bbf69b4cfffff
fff01603edb0300000000160014bbef244bcad13cffb68b5cef3017c74236755522040047304402 20
010d2854b86b90b7c33661ca25f9d9f15c24b88c5c4992630f77ff004b998fb802204106fc3ec8481
fa98e07b7e78809ac91b6ccaf60bf4d3f729c5a75899bb664a501473044022046d66321c6766abcb1
366a793f9bfd0e11e0b080354f18188588961ea76c5ad002207262381a0661d66f5c39825202524c4
5f29d500c6476176cd910b1691176858701695221026ccfb8061f235cc110697c0bfb3afb99d82c88
6672f6b9b5393b25a434c0cbf32103befa190c0c22e2f53720b1be9476dcf11917da4665c44c9c71c
3a2d28a933c352102be46dc245f58085743b1cc37c82f0d63a960efa43b5336534275fc469b49f4ac
53ae00000000
```

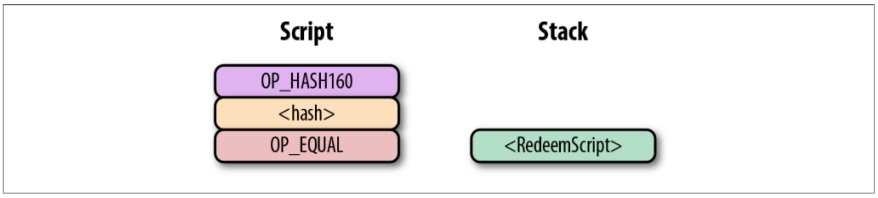
- 01000000 - version
- 00 - Segwit marker
- 01 - Segwit flag
- 01 - # of inputs
- 7082...54 - previous tx hash
- 00000000 - previous tx index
- 2322...4c - ScriptSig
- ffffffff - sequence
- 01 - # of outputs
- 603e...00 - output amounts
- 1600...22 - ScriptPubKey
- 0400...ae - witness
- 00000000 - locktime

شکل ۱۳-۳۲: p2sh-p2wsh به نرم‌افزار post-BIP0141

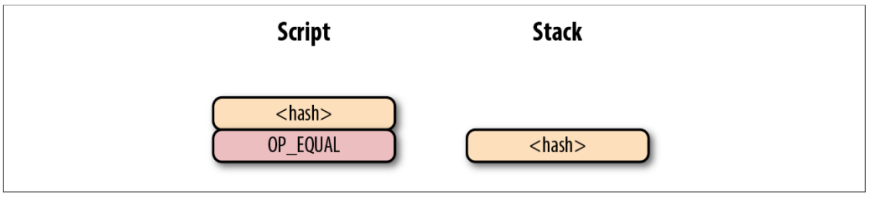
شکل ۱۳-۳۳: p2sh-p2wsh ScriptPubKey



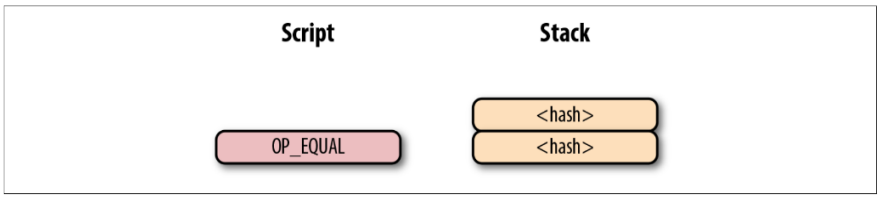
شکل ۱۳-۳٤: شروع p2sh-p2wsh



شکل ۱۳–۳۵: مرحله‌ی اول p2sh-p2wsh



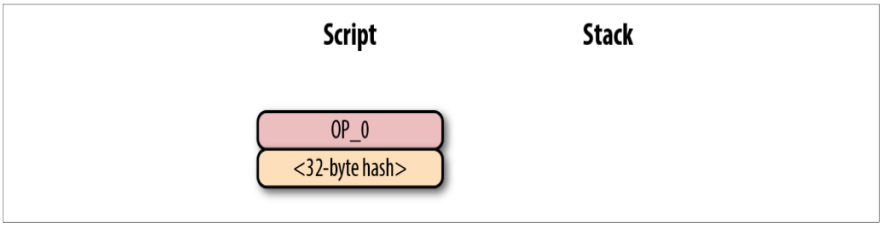
شکل ۱۳–۳۶: مرحله‌ی دوم p2sh-p2wsh



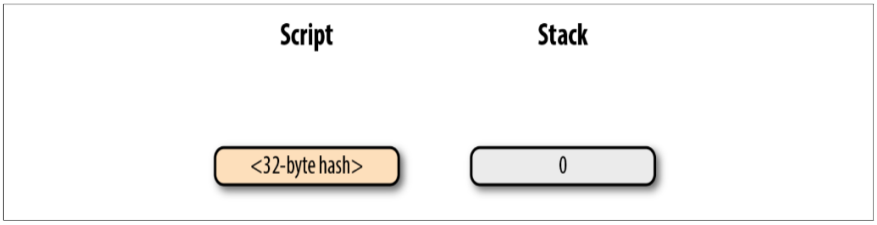
شکل ۱۳–۳۷: مرحله‌ی سوم p2sh-p2wsh

```
220020615ae01ed1bc1ffaad54da31d7805d0bb55b52dfd3941114330368c1bbf69b4c

        - 22 - Length of <RedeemScript>
        - 0020...4c - <RedeemScript>
```

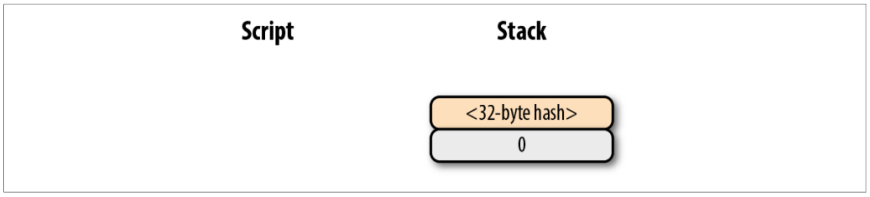
شکل ۱۳–۳۸: p2sh-p2wsh RedeemScript



شکل ۱۳-۳۹: مرحله‌ی چهارم p2sh-p2wsh



شکل ۱۳-٤۰: مرحله‌ی پنجم p2sh-p2wsh



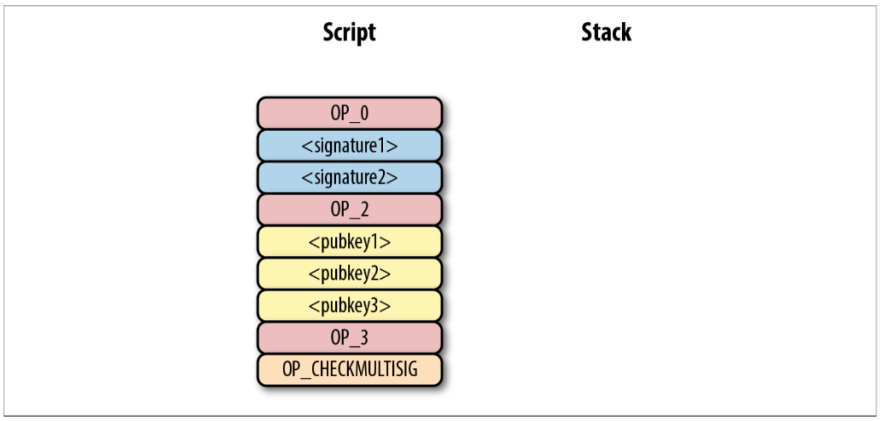
شکل ۱۳-٤۱: مرحله‌ی ششم p2sh-p2wsh

```
040047304402200 10d2854b86b90b7c33661ca25f9d9f15c24b88c5c4992630f77ff004
b998fb802204106fc3ec8481fa98e07b7e78809ac91b6ccaf60bf4d3f729c5a75899bb6
64a5014730440220 46d66321c6766abcb1366a793f9bfd0e11e0b080354f18188588961
ea76c5ad002207262381a0661d66f5c39825202524c45f29d500c6476176cd910b16911
76858701695221026ccfb8061f235cc110697c0bfb3afb99d82c886672f6b9b5393b25a
434c0cbf32103befa190c0c22e2f53720b1be9476dcf11917da4665c44c9c71c3a2d28a
933c352102be46dc245f58085743b1cc37c82f0d63a960efa43b5336534275fc469b49f
4ac53ae

        - 04 - Number of witness elements
        - 00 - OP_0
        - 47 - Length of <signaturex>
        - 3044...01 - <signaturex>
        - 69 - Length of WitnessScript
        - 5221...ae - <WitnessScript>
```

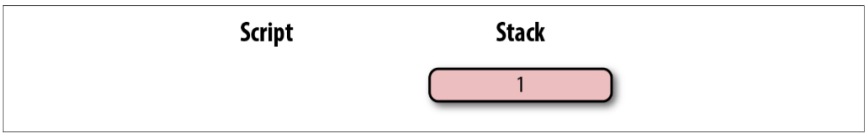
شکل ۱۳-۴۲: p2sh-p2wsh witness

```
5221026ccfb8061f235cc110697c0bfb3afb99d82c886672f6b9b5393b25a434c0cbf32103befa190
c0c22e2f53720b1be9476dcf11917da4665c44c9c71c3a2d28a933c352102be46dc245f58085743b1
cc37c82f0d63a960efa43b5336534275fc469b49f4ac53ae

        - 52 - OP_2
        - 21 - Length of <pubkeyx>
        - 0...01 - <pubkeyx>
        - 53 - OP_3
        - ae - OP_CHECKMULTISIG
```

شکل ۱۳-۴۳: p2sh-p2wsh WitnessScript



شکل ۱۳-۴۴: مرحله‌ی هفتم p2sh-p2wsh



شکل ۱۳-۴۵: پایان p2sh-p2wsh

Coding p2wsh and p2sh-p2wsh

تجزیه و پیاپی‌سازی دقیقاً مانند قبل است. تغییرات اصلی مربوط به `verify_input` در tx.py و `evalute` در script.py است:

```
class Tx:
...
    def verify_input(self, input_index):
        tx_in = self.tx_ins[input_index]
        script_pubkey =
tx_in.script_pubkey(testnet=self.testnet)
        if script_pubkey.is_p2sh_script_pubkey():
            command = tx_in.script_sig.commands[-1]
            raw_redeem = int_to_little_endian(len(command),
1) + command
            redeem_script =
Script.parse(BytesIO(raw_redeem))
            if redeem_script.is_p2wpkh_script_pubkey():
                z = self.sig_hash_bip143(input_index,
redeem_script)
                witness = tx_in.witness
            elif redeem_script.is_p2wsh_script_pubkey():
                command = tx_in.witness[-1]
                raw_witness = encode_varint(len(command)) +
command
                witness_script =
Script.parse(BytesIO(raw_witness))
                z = self.sig_hash_bip143(input_index,

witness_script=witness_script)
                witness = tx_in.witness
            else:
                z = self.sig_hash(input_index,
redeem_script)
                witness = None
        else:
            if script_pubkey.is_p2wpkh_script_pubkey():
                z = self.sig_hash_bip143(input_index)
                witness = tx_in.witness
            elif script_pubkey.is_p2wsh_script_pubkey():
                command = tx_in.witness[-1]
                raw_witness = encode_varint(len(command)) +
command
                witness_script =
Script.parse(BytesIO(raw_witness))
```

1

2

```
                z = self.sig_hash_bip143(input_index,
witness_script=witness_script)
                witness = tx_in.witness
            else:
                z = self.sig_hash(input_index)
                witness = None
        combined_script = tx_in.script_sig +
tx_in.script_pubkey(self.testnet)
        return combined_script.evaluate(z, witness)
```

روشی برای شناساییِ p2wsh در script.py:

```
def p2wsh_script(h256):
    '''Takes a hash160 and returns the p2wsh ScriptPubKey'''
    return Script([0x00, h256])
...
class Script:
...
    def is_p2wsh_script_pubkey(self):
        return len(self.cmds) == 2 and self.cmds[0] == 0x00
\
            and type(self.cmds[1]) == bytes and
len(self.cmds[1]) == 32
```

استفاده از قانون ویژه‌ی p2wsh:

```
class Script:
...
    def evaluate(self, z, witness):
    ...
        while len(commands) > 0:
        ...
        else:
                stack.append(command)
 ...
                if len(stack) == 2 and stack[0] == b'' and
len(stack[1]) == 32:
                    s256 = stack.pop()
                    stack.pop()
                    cmds.extend(witness[:-1])
                    witness_script = witness[-1]
                    if s256 != sha256(witness_script):
                        print('bad sha256 {} vs {}'.format
                            (s256.hex(),
sha256(witness_script).hex()))
                        return False
                    stream =
BytesIO(encode_varint(len(witness_script))
                        + witness_script)
```

```
                    witness_script_cmds =
Script.parse(stream).cmds
                    cmds.extend(witness_script_cmds)
```