فصل

# ۷

# تراکُنش و اسکریپت‌نویسی پیشرفته

## مقدمه

در فصل قبل با مبانی اساسی تراکُنش‌های بیت‌کوین و رایج‌ترین نوع *اسکریپت تراکُنش*، موسوم به اسکریپت P2PKH (پرداخت-به-دَرهم-کلید-عمومی)، آشنا شـــدیم. در این فصل اسکریپت‌نویسی پیشرفته را بررسی می‌کنیم و نشان می‌دهیم که چطور می‌توانید تراکُنش‌هایی با شروط پیچیده‌تر بسازید.

ابتدا اســـکریپت‌های چندامضایی را بررســـی خواهیم کرد، و سپس به تشـــریح دومین نوع اسکریپت تراکُنش رایج، اسکریپت *پرداخت-به-دَرهم-اســـکریپت* (P2SH)، که دریچه‌ای به دنیای اسکریپت‌های پیچیده باز می‌کند، می‌پردازیم. بعد از آن چند عملگر اســـکریپت جدید معرفی می‌کنیم که با آنها می‌توان از طریق *قفل زمانی* بُعد زمان را وارد بیت‌کوین کرد.

## چندامضایی

یک اســـکریپت چندامضایی (multisignature script) دارای این شرط اســـت که از N کلید عمومی موجود در این اســـکریپت دســـتکم M کلید باید امضای لازم برای باز کردن قفل تراکُنش را ارائه کنند. به همین خاطر به اســـکریپت چندامضایی گاه طرح «M-از-N» نیز گفته می‌شـــود، کـــه در آن N تعداد کل کلیدها و M حداقل تعداد امضای لازم برای تأیید اعتبار خروجی اســـت. برای مثال، در یک اســـکریپت چندامضایی «۲-از-۳» سه کلید عمومی (به عنوان امضاکننده‌ی بالقوه) در اســـکریپت وجود دارد، و برای خرج کردن مبلـــغ آن خروجی باید از حداقل دو تا از آنها برای امضای تراکُنش اســـتفاده شود. در حال حاضر، اســـکریپت‌های چندامضایی اســـتاندارد می‌توانند حداکثر ۱۵ کلید عمومی داشـــته باشند، بنابراین در آنها از هر ترکیبی بین ۱-از-۱ تا ۱۵-از-۱۵ می‌توان استفاده کرد. شاید در زمانی که شـــما این کتاب را می‌خوانید، محدودیت حداکثر ۱۵ کلید عمومی در اسکریپت‌های چندامضایی برداشته شده باشد؛ برای آن که در هر زمان از سقف قابل‌قبول شبکه مطلع شوید، بهتر است از تابع `isStandard()` کمک بگیرید.

صورت کلی یک اسکریپت قفل‌کننده با شرط چندامضایی «M-از-N» چنین است:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```

که در آن N تعداد کل کلیدهای عمومی و M آستانه (سقف) امضاهای مورد نیاز برای خرج کردن این خروجی است. برای نمونه، یک اسکریپت قفل‌کننده با شرط چندامضایی «۲-از-۳» به صورت زیر خواهد بود:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

شرط این اسکریپت قفل‌کننده را می‌توان با (مثلاً) اسکریپت بازکننده‌ی زیر که حاوی یک جفت امضا است، برآورده کرد:

```
<Signature B> <Signature C>
```

توجه کنید که هر ترکیب دیگر متشکل از دو امضای ساخته‌شده از کلیدهای خصوصی متناظر با کلیدهای عمومی A، B و C نیز شرط این اسکریپت قفل‌کننده را برآورده خواهد کرد. اسکریپت اعتبارسنجی خروجی مزبور هم از ترکیب این دو اسکریپت ساخته می‌شود:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

اجرای اسکریپت ترکیبی بالا فقط (و فقط) زمانی مقدار TRUE تولید خواهد کرد که اسکریپت بازکننده‌ی قفل با شرط خواسته‌شده در اسکریپت قفل‌کننده مطابقت داشته باشد. در اینجا شرط آن است که اسکریپت بازکننده حاوی حداقل ۲ امضای معتبر ساخته‌شده از کلیدهای خصوصی متناظر با ۳ کلید عمومی اشاره‌شده در اسکریپت قفل‌کننده باشد.

## یک باگ در فرآیند اجرای CHECKMULTISIG

در اجرای CHECKMULTISIG یک باگ کوچک وجود دارد که باید فکری برای آن کرد: وقتی CHECKMULTISIG را اجرا می‌کنید، باید M + N + ۲ درایه‌ی موجود در پشته را به عنوان پارامتر مصرف کند، تا به نتیجه برسد. ولی به خاطر وجود این باگ، CHECKMULTISIG یک مقدار بیش از آنچه انتظار داریم، از پشته بیرون می‌کشد. اجازه دهید این موضوع را به کمک همان مثال قبل بیشتر توضیح دهیم:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

در شروع، CHECKMULTISIG بالاترین درایه‌ی پشته را که در اینجا همان N (با مقدار 3) است، بیرون کشیده و سپس N (در اینجا، ۳) درایه‌ی بعدی، یعنی کلیدهای عمومی A، B و C (که مجاز به امضا کردن خروجی هستند)، را از پشته بیرون می‌کشد. پس از آن، CHECKMULTISIG اقدام به بیرون کشیدن درایه‌ی بعدی می‌کند که همان M (تعداد امضاهای مورد نیاز برای اعتبارسنجی این خروجی؛ در اینجا 2) است. در این مرحله، CHECKMULTISIG باید M (در اینجا، ۲) درایه‌ی بعدی، یعنی به ترتیب امضاهای C و B را (که آخرین درایه‌های موجود در پشته هستند) از پشته بیرون بکشد و اعتبار آنها را بررسی کند. متأسفانه، باگی که در پیاده‌سازی CHECKMULTISIG وجود دارد، باعث می‌شود تا این عملگر یک درایه بیشتر (M + ۱ درایه) از پشته بیرون بکشد. البته CHECKMULTISIG هنگام ارزیابی امضاها این درایه‌ی اضافی را نادیده می‌گیرد، بنابراین تأثیر مستقیمی بر عملکرد آن نخواهد داشت. با این حال، این مقدار اضافی باید در پشته وجود داشته باشد، چون اگر چنین نباشد، اقدام CHECKMULTISIG برای بیرون کشیدن این درایه از پشته‌ی خالی با خطا مواجه می‌شود؛ و همان طور که قبلاً هم گفته‌ایم، اگر در اجرای یک اسکریپت هرخطایی (از جمله خطای پشته) رُخ دهد، آن تراکنش نامعتبر تلقی خواهد شد. از آنجا که این درایه‌ی اضافی هیچ نقشی در ارزیابی اسکریپت ندارد، می‌تواند هر مقداری داشته باشد، ولی به طور سنتی از مقدار 0 برای آن استفاده می‌شود.

از آنجا که این باگ به بخشی از قاعده‌ی اجماع تبدیل شده است، بایستی از این پس برای همیشه تکثیر شود. بنابراین، اسکریپت اعتبارسنجی صحیح به صورت زیر خواهد بود:

```
0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

در نتیجه، اسکریپت بازکننده‌ی قفل، به جای <Signature B> <Signature C>، باید به صورت زیر باشد:

```
0 <Signature B> <Signature C>
```

از این به بعد، اگر یک اسکریپت بازکننده‌ی چندامضایی دیدید که با 0 شروع شده است، تعجب نکنید، چون این همان تمهید لازم برای دور زدن باگی است که به طور تصادفی به یک قاعده‌ی اجماع تبدیل شده است.

## پرداخت-به-دَرهم-اسکریپت (P2SH)

اسکریپت پرداخت-به-دَرهم-اسکریپت (Pay-to-Script-Hash) اولین بار در ۲۰۱۲ به عنوان یک نوع جدید و قدرتمند از تراکُنش معرفی شد که کاربرد اسکریپت‌های پیچیده را تا حد زیادی ساده می‌کند. برای معرفی P2SH بهتر است از یک مثال عملی استفاده کنیم.

در فصل ۱ با محمد [واردکننده‌ی لوازم الکترونیک در دوبی (امارات متحده‌ی عربی)] آشنا شدیم. شرکت محمد در حساب‌های شرکتی خود به طور گسترده از تراکُنش‌های چندامضایی بیت‌کوین استفاده می‌کند. اسکریپت‌های چندامضایی یکی از ویژگی‌های بسیار قدرتمند بیت‌کوین است، و در واقع بیشترین کاربرد را در اسکریپت‌نویسی پیشرفته دارد. این شرکت تمامی پرداخت‌های مشتریان خود [موسوم به **حساب‌های وصولی** (accounts receivable) یا مطالبات] را با استفاده از یک اسکریپت چندامضایی انجام می‌دهد. با این روش، هر پرداختی که توسط مشتریان شرکت انجام شود، به گونه‌ای قفل می‌شود که برای باز (خرج) کردن آن به حداقل دو امضا (یکی امضای محمد و دیگری امضای یکی از شرکای وی یا وکیل شرکت، که کلید پشتیبان را در اختیار دارد) نیاز خواهد بود. اسکریپت‌های چندامضایی به شرکت‌ها امکان کنترل و مقابله‌ی مؤثرتر با دزدی یا اختلاس را می‌دهند. اسکریپت بازکننده‌ی چندامضایی این شرکت نسبتاً طولانی است و چنین ظاهری دارد:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key>
<Partner3 Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

هر چند اسکریپت‌های چندامضایی ویژگی قدرتمندی هستند، ولی استفاده از آنها پُرزحمت است. برای مثال، در مورد اسکریپت بالا، محمد باید قبل از هر پرداخت این اسکریپت را به دست مشتریان خود برساند. هر مشتری هم باید از برنامه‌ی کیف‌پول مخصوصی استفاده کند که قادر به ایجاد اسکریپت‌های سفارشی باشد، و خودش هم روش ایجاد تراکُنش با استفاده از این اسکریپت‌های سفارشی را بداند. علاوه بر آن، تراکُنش حاصل پنج برابر بزرگتر از تراکُنش‌های ساده و معمولی خواهد بود، چون این اسکریپت حاوی پنج کلید عمومی (بسیار طولانی) است، و هزینه‌ی این تراکُنش بزرگ به گردن مشتری می‌افتد (به یاد دارید که کارمزد تراکُنش بر حسب اندازه‌ی آن بر حسب بایت تعیین می‌شود). سرانجام، یک اسکریپت تراکُنش باید تا زمانی که خرج نشده، (به صورت عضوی از «مجموعه‌ی UTXO») در حافظه‌ی تمامی گره‌های کامل بیت‌کوین نگهداری شود، و هر چه این اسکریپت قفل‌کننده بزرگتر باشد، فضای بیشتری اِشغال خواهد کرد. همه‌ی این مشکلات باعث می‌شوند تا استفاده از اسکریپت‌های قفل‌کننده‌ی پیچیده در عمل دشوار باشد.

پرداخت P2SH برای حل همین مشکلات عملی و ساده کردن کاربرد اسکریپت‌های پیچیده (تا حد یک پرداخت ساده به یک آدرس بیت‌کوین) توسعه داده شد. در پرداخت P2SH، اسکریپت‌های قفل‌کننده‌ی پیچیده جای خود را به اثرانگشت دیجیتال (digital fingerprint)، یا دَرهم رمزنگاری (cryptographic hash)، داده‌اند. وقتی یک تراکُنش تلاش می‌کند یک UTXO که بعداً ارائه می‌شود را خرج کند، باید (علاوه بر اسکریپت بازکننده‌ی قفل) حاوی اسکریپت منطبق با این دَرهم باشد. به بیان ساده، P2SH یعنی «پرداخت به اسکریپت منطبق با این دَرهم، اسکریپتی که بعداً در زمان خرج کردن این خروجی ارائه خواهد شد.»

در تراکُنش‌های P2SH، اسکریپت قفل‌کننده جای خود را به یک دَرهم که به آن اسکریپت وصول (redeem script) گفته می‌شود، داده است؛ علت این نام‌گذاری آن است که، در زمان نقد کردن این خروجی، این اسکریپت به جای اسکریپت قفل‌کننده به سیستم ارائه خواهد شد. جدول ۷-۱ اسکریپت اعتبارسنجی مثال قبل (اسکریپت چندامضایی پرداخت به شرکت محمد) بدون استفاده از P2SH را نشان داده است؛ در جدول ۷-۲ همین اسکریپت را با کُدگذاری P2SH مشاهده می‌کنید.

**جدول ۷-۱** اسکریپت پیچیده بدون P2SH

| | |
|---|---|
| اسکریپت قفل‌کننده | `2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG` |
| اسکریپت بازکننده‌ی قفل | `Sig1 Sig2` |

**جدول ۷-۲** اسکریپت پیچیده با کُدگذاری P2SH

| | |
|---|---|
| اسکریپت وصول | `2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG` |
| اسکریپت قفل‌کننده | `HASH160 <20-byte hash of redeem script> EQUAL` |
| اسکریپت بازکننده‌ی قفل | `Sig1 Sig2 <redeem script>` |

همان طور که از این دو جدول می‌بینید، در پرداخت P2SH دیگر خبری از اسکریپت پیچیده‌ای که شروط خرج کردن این خروجی را در بر دارد (همان اسکریپت وصول)، در اسکریپت قفل‌کننده نیست؛ به جای آن فقط دَرهم این اسکریپت (پیچیده) در اسکریپت قفل‌کننده قرار می‌گیرد، و خود این اسکریپت بعداً در زمان خرج کردن خروجی و به عنوان بخشی از اسکریپت بازکننده‌ی قفل ارائه خواهد شد. با این تمهید پیچیدگی تولید و کارمزد تراکُنش‌های پرداخت با شروط پیچیده از دوش فرستنده (خریدار) برداشته شده و به گیرنده (فروشنده/خرج‌کننده) منتقل می‌شود.

اجازه دهید مبادلات بیت‌کوین شرکت محمد، اسکریپت چندامضایی پیچیده‌ی آن، و اسکریپت‌های P2SH حاصل را با جزئیات بیشتر بررسی کنیم. اسکریپت چندامضایی را که این شرکت از آن برای دریافت پول از مشتریان خود استفاده می‌کند، قبلاً دیدیم:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key>
<Partner3 Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

اگر کلیدهای عمومی واقعی (اعداد ۵۲۰-بیتی که با 04 شروع می‌شوند) را در این اسکریپت قرار دهیم، به اسکریپت بسیار طولانی زیر خواهیم رسید:

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395
D7EEC6984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23
```

```
E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D7
8D0E34224858008E8B49047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C905
737812F393DA7D4420D7E1A9162F0279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD
7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D0872274
40645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1ECED3C68D446BCAB69AC0BA7DF5
0D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D137AAB59E0B000EB7E
D238F4D800 5 CHECKMULTISIG
```

کل این اسکریپت را می‌توان با یک دَرهم رمزنگاری ۲۰-بایتی جایگزین کرد؛ برای این منظور ابتدا الگوریتم دَرهم‌سازی SHA256 روی این رشته اعمال شده، و سپس الگوریتم RIPEMD160 روی خروجی الگوریتم SHA256 اعمال می‌شود. حاصل کار دَرهم ۲۰-بایتی (۱۶۰-بیتی) زیر است:

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

تراکُنش P2SH خروجی خود را به جای اسکریپت قفل‌کننده‌ی (طولانی) قبلی به صورت زیر با این دَرهم قفل می‌کند:

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

که به وضوح کوتاه‌تر است. در واقع، به جای گفتن «پرداخت به این اسکریپت چندامضایی ۵-کلیده»، تراکُنش P2SH معادل این است که بگویید، «پرداخت به اسکریپتی با این دَرهم». اکنون، فقط کافی است مشتریان شرکت محمد این اسکریپت قفل‌کننده‌ی بسیار کوتاه‌تر را در پرداخت‌های خود قرار دهند. زمانی که محمد و شرکایش بخواهند یکی از این UTXO ها را خرج کنند، فقط کافی است اسکریپت وصول اولیه (اسکریپتی که از دَرهم آن برای قفل کردن این UTXO استفاده شده) و امضاهای لازم را که به صورت زیر خواهد بود، ارائه کنند:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

این دو اسکریپت در دو مرحله ترکیب می‌شوند. ابتدا، اسکریپت وصول با اسکریپت قفل‌کننده مقایسه می‌شود تا از منطبق بودن دَرهم آنها اطمینان حاصل شود:

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160 <redeem scriptHash> EQUAL
```

در گام بعدی، اگر دَرهم اسکریپت وصول با دَرهم موجود در اسکریپت قفل‌کننده‌ی UTXO منطبق بود، اسکریپت بازکننده‌ی قفل اجرا می‌شود تا اسکریپت وصول را باز کند:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```

تقریباً تمامی اسکریپت‌هایی که در این فصل می‌بینید، فقط به صورت اسکریپت P2SH قابل پیاده‌سازی هستند، و نمی‌توان آنها را به طور مستقیم در اسکریپت قفل‌کننده‌ی یک UTXO به کار برد.

## آدرس‌های P2SH

یکی از ویژگی‌های مهم P2SH این است که دَرهم یک اسکریپت را می‌توان به صورت یک آدرس [بیت‌کوین] که به آن آدرس P2SH گفته می‌شود، کُدگذاری کرد؛ این قابلیت در استاندارد BIP-13 تعریف شده است. درست همان طور که آدرس‌های بیت‌کوین دَرهم ۲۰-بایتی یک کلید عمومی با کُدگذاری Base58Check هستند، یک آدرس P2SH نیز دَرهم ۲۰-بایتی یک اسکریپت با کُدگذاری Base58Check است. آدرس‌های P2SH از پیشوند ویرایش «5» استفاده می‌کنند، که در نتیجه آدرس‌های Base58Check متناظر با آنها همیشه با کاراکتر «3» شروع خواهند شد.

برای مثال، اسکریپت چندامضایی پیچیده‌ی شرکت محمد بعد از تبدیل به آدرس P2SH با کُدگذاری Base58Check معادل 39RF6JqABiHdYHkfChV6USGMe6Nsr66Gzw خواهد بود، که [مانند آدرس‌های بیت‌کوین معمولی] در تقریباً تمامی برنامه‌های کیف‌پول قابل استفاده است، و محمد می‌تواند این «آدرس» را به مشتریان خود بدهد. پیشوند «3» در این آدرس به مشتریان اعلام می‌کند که این یک «آدرس ویژه» است، آدرسی که [به جای یک کلید عمومی] متناظر با یک اسکریپت است؛ ولی از این گذشته، طرز کار و استفاده از آن هیچ تفاوتی با آدرس‌های بیت‌کوین معمولی ندارد.

آدرس‌های P2SH تمامی پیچیدگی اسکریپت‌های چندامضایی را از دید کاربران پنهان می‌کنند، و کاربری که پرداخت را انجام می‌دهد، اصلاً آن اسکریپت‌ها را نخواهد دید.

## مزایای P2SH

مزایای پرداخت P2SH در مقایسه با استفاده از اسکریپت‌های پیچیده برای قفل کردن خروجی‌ها عبارتند از:

- جایگزینی اسکریپت‌های پیچیده با اثرانگشت [دَرهم اسکریپت] کوتاه‌تر در خروجی تراکُنش، که باعث کم شدن حجم تراکُنش می‌شود.
- امکان کُدگذاری اسکریپت‌ها به صورت آدرس، به طوری که فرستنده (و کیف‌پول وی) متوجه پیچیدگی مهندسی پیاده‌سازی P2SH نشود.
- انتقال بار تولید اسکریپت‌های پیچیده به گیرنده، که زحمت فرستنده [خریدار] را کم می‌کند.
- انتقال بار ذخیره‌سازی اسکریپت‌های پیچیده و حجیم مورد نیاز از خروجی (که در مجموعه‌ی UTXO ذخیره می‌شود) به ورودی (که در بلاک‌چین ذخیره خواهد شد).
- انتقال بار ذخیره‌سازی اسکریپت‌های پیچیده و طولانی از حال (زمان پرداخت) به آینده (زمان خرج کردن خروجی).
- انتقال کارمزد تراکُنش‌های حاوی اسکریپت‌های پیچیده و حجیم از فرستنده [خریدار] به گیرنده [فروشنده]، کسی که برای خرج کردن این UTXO باید اسکریپت وصول طولانی آن را ارائه کند.

## اسکریپت وصول و اعتبارسنجی

تا قبل از ویرایش 0.9.2 مشتری هسته‌ی بیت‌کوین، پرداخت-به-دَرهم-اسکریپت، با تابع ()IsStandard، به انواع استاندارد اسکریپت‌های تراکُنش بیت‌کوین محدود بود، یعنی اسکریپت وصولی که باید در تراکُنش خرج کردن یک خروجی ارائه می‌شد، فقط می‌توانست یکی از انواع اسکریپت‌های استاندارد بیت‌کوین باشد: P2PKH، P2PK، اسکریپت چندامضایی (به استثنای کُدهای اجرایی که حاوی عملگر RETURN هستند)، یا خود P2SH. از ویرایش 0.9.2 مشتری هسته‌ی بیت‌کوین به این سو، تراکُنش‌های P2SH می‌توانند حاوی هر نوع اسکریپت معتبری باشند، که این ویژگی باعث انعطاف‌پذیری بیشتر P2SH شده و راه را برای بسیاری انواع پیچیده و بدیع از تراکُنش‌ها باز کرده است.

توجه داشته باشید که همچنان نمی‌توانید یک اسکریپت P2SH را در داخل اسکریپت وصول P2SH قرار دهید، چون P2SH یک استاندارد خودارجاع (recursive) نیست. با آن که از نظر فنی امکان قرار دادن عملگر RETURN در داخل یک اسکریپت وصول وجود دارد [زیرا هیچ قانونی این کار را منع نکرده است]، اما این کار هیچ فایده‌ی عملی در بر ندارد، چون اجرای عملگر RETURN در حین اعتبارسنجی یک تراکُنش باعث نامعتبر شناخته شدن آن خواهد شد.

همچنین دقت داشته باشید که حتی اگر یک خروجی را با دَرهم یک اسکریپت وصول نامعتبر قفل کنید، آن خروجی همچنان پردازش خواهد شد، چون اسکریپت وصول تا لحظه‌ی اقدام برای خرج کردن یک خروجی P2SH به شبکه عرضه نمی‌شود. در واقع، این UTXO با موفقیت قفل می‌شود، ولی گیرنده قادر به خرج کردن آن نخواهد بود، چون تراکُنش خرج کردن این خروجی که حاوی اسکریپت وصول نامعتبر است، پذیرفته نخواهد شد. این یک خطر بالقوه محسوب می‌شود، چون ممکن است بیت‌کوین‌های خود را در یک تراکُنش P2SH قفل کنید که بعداً قادر به باز (خرج) کردن آن نباشید. شبکه‌ی بیت‌کوین هر اسکریپت قفل‌کننده‌ی P2SH را *می‌پذیرد*، حتی اگر این قفل متناظر با یک اسکریپت وصول معتبر نباشد، چون این اسکریپت قفل‌کننده [دَرهم اسکریپت] هیچ نشانه‌ای از معتبر یا نامعتبر بودن اسکریپت وصول متناظر با آن ندارد.

**هشدار**

اسکریپت‌های قفل‌کننده‌ی P2SH فقط حاوی دَرهم یک اسکریپت وصول هستند، که هیچ نشانه‌ای از محتوای واقعی آن اسکریپت وصول در خود ندارد. تراکُنش‌های P2SH همیشه معتبر تلقی شده و پذیرفته می‌شوند، حتی اگر اسکریپت وصول آنها معتبر نباشد. اگر یک خروجی به طور اتفاقی با دَرهم یک اسکریپت وصول نامعتبر قفل شود، آن مبلغ بیت‌کوین دیگر قابل خرج کردن نخواهد بود!

## خروجی ثبت داده (RETURN)

دفتر کل توزیع‌شده و زمان‌دار بیت‌کوین، معروف به بلاک‌چین، کاربردهای بالقوه‌ی دیگری غیر از پرداخت و انتقال نقدینگی نیز دارد. بسیاری از برنامه‌نویسان تلاش کرده‌اند از امنیت و انعطاف‌پذیری سیستم بیت‌کوین برای کاربردهای دیگر مانند خدمات دفاتر اسناد رسمی، گواهی (برگه‌ی) سهام، و قراردادهای هوشمند استفاده کنند. تلاش‌های اولیه شامل استفاده از زبان اسکریپت‌نویسی بیت‌کوین برای ایجاد خروجی‌هایی بود که فقط داده‌ها را در بلاک‌چین ثبت و ضبط کنند؛ برای مثال، ضبط اثرانگشت دیجیتال در یک فایل به گونه‌ای که هر کسی بتواند فقط با ارجاع به آن تراکُنش، وجود چنین فایلی در یک تاریخ مشخص را اثبات کند.

استفاده از بلاک‌چین برای ثبت داده‌های غیرمرتبط با پرداخت‌های بیت‌کوین همواره موضوعی مجادله‌برانگیز بوده است. بسیاری از کاربران و برنامه‌نویسان بیت‌کوین این کاربرد را نوعی سوءاستفاده از سیستم بیت‌کوین می‌دانند و خواهان ممنوعیت آن هستند. در مقابل، عده‌ای هم هستند که این قبیل کاربردها را نمونه‌ای از توانایی‌های فناوری بلاک‌چین می‌بینند و خواهان تشویق چنین تجربیاتی هستند. مخالفان ثبت داده‌های غیرپرداخت در بلاک‌چین استدلال می‌کنند که این کار باعث «تورم بلاک‌چین» می‌شود، و بار پردازشی و هزینه‌ی اضافی (به شکل فضای دیسک و حافظه) بر گره‌های کامل بیت‌کوین تحمیل می‌کند، فشاری که قرار نبوده بر بلاک‌چین تحمیل شود. علاوه بر آن، این قبیل تراکُنش‌ها UTXO هایی تولید می‌کنند که نمی‌توان آنها را خرج کرد، و از فیلد آدرس مقصد بیت‌کوین به عنوان یک فیلد ۲۰-بایتی آزاد استفاده می‌کنند. از آنجا که این فیلد برای ثبت داده‌های دیگری به کار می‌رود (و حاوی یک کلید عمومی [گیرنده] نیست)، UTXO حاصل هرگز نمی‌تواند خرج شود؛ این خروجی یک *پرداخت قلابی* است، و UTXO هایی که هرگز نمی‌توانند خرج شوند، هیچ وقت از «مجموعه‌ی UTXO» خارج نمی‌شوند، که این باعث افزایش تدریجی و دائمی حجم پایگاه داده‌ی UTXO یا «تورم» آن خواهد شد.

در ویرایش 0.9 مشتری هسته‌ی بیت‌کوین، با معرفی عملگر RETURN، مصالحه‌ای بین طرفداران و مخالفان ضبط داده‌های غیرپرداخت در بلاک‌چین صورت گرفت. عملگر RETURN به برنامه‌نویسان اجازه می‌دهد ۸۰ بایت داده‌ی غیرمرتبط با پرداخت به هر خروجی تراکُنش اضافه کنند. با این حال، بر خلاف «UTXO قلابی»، عملگر RETURN یک خروجی «*آشکارا غیرقابل خرج*» تولید می‌کند که لزومی به ذخیره کردن آن در مجموعه‌ی UTXO نیست. خروجی‌های RETURN

در بلاک‌چین ثبت می‌شوند، بنابراین فضای دیسک را اِشغال کرده و حجم بلاک‌چین را افزایش می‌دهند، ولی در مجموعه‌ی UTXO ذخیره نمی‌شوند و مخزن حافظه‌ی UTXO را متورم نمی‌کنند، و (از نظر حافظه‌ی RAM) بار اضافی بر گره‌های کامل تحمیل نخواهند کرد.

ساختار دستوری اسکریپت RETURN بسیار ساده است:

```
RETURN <data>
```

که در آن بخش data به ۸۰ بایت محدود بوده، و در اکثر مواقع حاوی یک رشته‌ی دَرهم (مانند خروجی ۳۲-بایتی الگوریتم SHA256) است. بسیاری از برنامه‌های کاربردی برای کمک به شناسایی کاربرد این داده از یک پیشوند استفاده می‌کنند. برای مثال، فایل «اثبات وجود» مورد استفاده در خدمات اسناد رسمی دیجیتال از پیشوند ۸-بایتی DOCPROOF، که با کُدگذاری اَسکی به صورت 44 4f 43 50 52 4f 4f 46 در می‌آید، استفاده می‌کند (برای اطلاعات بیشتر به سایت http://proofofexistence.com/ مراجعه کنید).

به خاطر داشته باشید که هیچ اسکریپت بازکننده‌ی قفل متناظر با یک اسکریپت RETURN وجود ندارد، که بتوان از آن برای خرج کردن یک خروجی RETURN استفاده کرد. اسکریپت RETURN هم به صورتی طراحی شده که نتوان خروجی آن را خرج کرد، و بنابراین نیازی به نگه داشتن آن در مجموعه‌ی UTXO نیست؛ همان طور که گفتیم، اسکریپت RETURN یک خروجی «*آشکارا غیرقابل‌خرج*» است. اسکریپت RETURN معمولاً یک خروجی با مبلغ ۰ بیت‌کوین است، چون هر بیت‌کوینی که در این خروجی قرار گرفته باشد، برای همیشه از دست خواهد رفت. اگر در ورودی یک تراکُنش به یک خروجی RETURN ارجاع شده باشد، «موتور اعتبارسنجی اسکریپت» اجرای این اسکریپت اعتبارسنجی را متوقف کرده و آن تراکُنش را به عنوان یک تراکُنش نامعتبر علامت‌گذاری خواهد کرد؛ به عبارت دیگر، اجرای عملگر RETURN همیشه باعث توقف اجرای اسکریپت و «بازگرداندن» مقدار FALSE خواهد شد. بنابراین، اگر به طور اتفاقی در ورودی یک تراکُنش به یک خروجی RETURN ارجاع کنید، آن تراکُنش نامعتبر خواهد شد.

هر تراکُنش استاندارد فقط یک خروجی RETURN می‌تواند داشته باشد (خروجی که با مقدار برگشتی تابع isStandard منطبق باشد). با این حال، یک خروجی واحد RETURN را می‌توان با خروجی‌هایی از هرنوع دیگر ترکیب کرد.

از ویرایش 0.10 دو گزینه‌ی خط-فرمان جدید به هسته‌ی بیت‌کوین اضافه شده است. گزینه‌ی datacarrier انتشار و استخراج تراکُنش‌های RETURN را کنترل می‌کند؛ مقدار پیش‌فرض این گزینه «1» است که انتشار و استخراج این تراکُنش‌ها را مجاز می‌کند. گزینه‌ی datacarriersize یک آرگومان عددی می‌گیرد که حداکثر اندازه‌ی اسکریپت RETURN را بر حسب بایت مشخص می‌کند؛ مقدار پیش‌فرض این گزینه ۸۳ بایت (حداکثر مقدار مجاز) است که از ۸۰ بایت داده‌ی RETURN، یک بایت کُد اجرایی عملگر RETURN، و دو بایت کُد اجرایی PUSHDATA تشکیل می‌شود.

> وقتی عملگر RETURN پیشنهاد شد، محدودیت ۸۰ بایت برای آن در نظر گرفته شده بود، ولی در اولین پیاده‌سازی این عملگر مقدار مجاز آن به ۴۰ بایت کاهش داده شد. در فوریه ۲۰۱۵، با انتشار ویرایش 0.10 هسته‌ی بیت‌کوین، این محدودیت به همان ۸۰ بایت افزایش یافت. اختیار تصمیم‌گیری درباره‌ی انتشار یا استخراج تراکُنش‌های RETURN، یا حجمی از داده که مایل به پردازش آن هستند (۸۰ بایت یا کمتر)، بر عهده‌ی خود گره‌ها گذاشته شده است.

## قفل‌زمانی

قفل‌زمانی (timelock) یک ساز و کار محدودکننده برای تراکُنش‌ها یا خروجی‌ها است که اجازه‌ی خرج کردن آنها را فقط بعد از زمانی معین می‌دهد. بیت‌کوین از ابتدا دارای ساز و کار قفل‌زمانی در سطح-تراکُنش بوده است که به وسیله‌ی فیلد nLocktime در تراکُنش پیاده‌سازی می‌شود. از اواخر ۲۰۱۵ و اواسط ۲۰۱۶ دو قفل‌زمانی جدید برای قفل کردن خروجی‌ها در سطح-UTXO، به نام CHECKLOCKTIMEVERIFY و CHECKSEQUENCEVERIFY، به بیت‌کوین اضافه شده است.

قفل‌زمانی برای پرداخت‌های مدت‌دار و قفل کردن یک مبلغ بیت‌کوین تا زمانی مشخص در آینده کاربرد دارد. از آن مهم‌تر، قفل‌زمانی بُعد زمان را وارد زبان اسکریپت‌نویسی بیت‌کوین کرده و راه را برای قرادادهای هوشمند چندمرحله‌ای پیچیده باز می‌کند.

## زمان‌قفل تراکُنش (nLocktime)

همان طور که گفتیم، بیت‌کوین از همان ابتدا دارای ساز و کار قفل‌زمانی در سطح-تراکُنش بود. قفل‌زمانی تراکُنش یک گزینه در سطح-تراکُنش (یک فیلد در ساختمان داده‌ی تراکُنش) است که زمان معتبر شدن یک تراکُنش، و امکان انتشار آن در شبکه‌ی بیت‌کوین و اضافه شدن به بلاک‌چین، را مشخص می‌کند. قفل‌زمانی که در هسته‌ی بیت‌کوین با متغیر nLocktime تعریف شده، گاه به همین نام یعنی زمان‌قفل (locktime) نیز شناخته می‌شود. در اکثر تراکُنش‌ها این فیلد (متغیر) مقدار ۰ دارد، یعنی تراکُنش می‌تواند بلافاصله منتشر و اجرا شود. اگر nLocktime مقداری غیرصفر و کمتر از ۵۰۰ میلیون داشته باشد، این عدد به عنوان *ارتفاع بلاک* تعبیر می‌شود، یعنی این تراکُنش معتبر نیست و نبایستی منتشر شود، یا باید در بلاک‌چین قبل از ارتفاع بلاک تعیین‌شده قرار گیرد. اگر مقدار nLocktime بیشتر از ۵۰۰ میلیون باشد، این فیلد به عنوان یک *برچسب زمانی* شبه-یونیکس (تعداد ثانیه‌های سپری‌شده از تاریخ اول ژانویه ۱۹۷۰) تلقی شده و تراکُنش تا پیش از آن زمان معین معتبر نخواهد بود. تراکُنش‌هایی که در آنها فیلد nLocktime به یک بلاک یا تاریخ آینده اشاره می‌کند، بایستی توسط سیستم مبدأ نگه داشته شده، و *نباید* تا قبل از معتبر شدن (فرا رسیدن آن بلاک یا تاریخ) در شبکه‌ی بیت‌کوین منتشر شوند. اگر یک تراکُنش قبل از زمان مشخص‌شده در فیلد nLocktime در شبکه‌ی بیت‌کوین منتشر شود، توسط اولین گره به عنوان یک تراکُنش نامعتبر پس زده شده و به گره‌های دیگر فرستاده نخواهد شد. کاربرد nLocktime درست مثل چک *مدت‌دار* است.

### محدودیت‌های زمان‌قفل تراکُنش

با آن که قفل‌زمانی nLocktime اجازه‌ی خرج کردن یک خروجی را به آینده موکول می‌کند، ولی خرج کردن این خروجی قبل از آن زمان تعیین‌شده را غیرممکن نمی‌کند. اجازه دهید با یک مثال نشان دهیم چگونه ممکن است چنین اتفاقی بیفتد.

فرض کنید آلیس با خرج کردن یکی از خروجی‌های خود به آدرس باب، یک تراکُنش را امضا کرده و nLocktime این تراکُنش را ۳ ماه بعد تعیین می‌کند. آلیس این تراکُنش را به باب می‌فرستد تا آن را نزد خود نگه دارد (درست مثل صدور یک چک مدت‌دار در وجه فروشنده). با این تراکُنش، آلیس و باب می‌دانند که:

- باب نمی‌تواند تا ۳ ماه آینده این تراکُنش را روی شبکه‌ی بیت‌کوین منتشر کرده و آن را نقد (وصول) کند.
- باب می‌تواند ۳ ماه دیگر (بعد از پایان سررسید) این تراکُنش را روی شبکه‌ی بیت‌کوین منتشر کند.

با این حال:

- هیچ منعی وجود ندارد که آلیس یک تراکُنش دیگر ایجاد کرده و همان خروجی را یک بار دیگر (این بار بدون قفل‌زمانی) در آن خرج کند. به عبارت دیگر، آلیس می‌تواند قبل از پایان سررسید همان UTXO را دوباره خرج کند.
- باب هیچ تضمینی ندارد که آلیس دست به چنین کاری نخواهد زد.

توجه به این محدودیت تراکُنش‌های nLocktime بسیار مهم است. تنها چیزی که در این تراکُنش قطعی است، عدم توانایی باب برای وصول آن قبل از سررسید ۳-ماهه است. در واقع، هیچ تضمینی وجود ندارد که باب به پول خود برسد! برای دستیابی به این ضمانت، محدودیت قفل‌زمانی باید (به جای تراکُنش) روی خود UTXO قرار داده شود و بخشی از اسکریپت قفل‌کننده باشد. این کار با استفاده از نوع جدید قفل‌زمانی، موسوم به «بررسی-زمان-قفل-اعتبارسنجی»، ممکن است.

## عملگر بررسی-زمان-قفل-اعتبارسنجی (CLTV)

در دسامبر ۲۰۱۵، نوع جدیدی از قفل‌زمانی (در قالب یک انشعاب نرم) به بیت‌کوین اضافه شد. بر اساس استاندارد تعریف‌شده در BIP-65، یک عملگر اسکریپت جدید به نام CHECKLOCKTIMEVERIFY، یا CLTV، به زبان اسکریپت‌نویسی بیت‌کوین اضافه شد. عملگر CLTV به جای آن که مانند nLocktime در سطح-تراکُنش عمل کند، قفل‌زمانی را روی خروجی قرار می‌دهد. این ویژگی انعطاف‌پذیری بسیار بیشتری به قفل‌های زمانی داد. به بیان ساده، با استفاده از عملگر CLTV در اسکریپت وصول یک خروجی می‌توانید زمان نقد شدن آن خروجی را به تعویق بیندازید، ولی کل تراکُنش همچنان بلافاصله قابل انتشار در شبکه‌ی بیت‌کوین است.

nLocktime یک قفل‌زمانی سطح-تراکُنش، ولی CLTV یک قفل‌زمانی سطح-خروجی است.

CLTV جایگزین nLocktime نیست، بلکه فقط یک UTXO (خروجی) خاص را محدود می‌کند، به طوری که امکان خرج کردن آن خروجی در یک تراکُنش آینده با nLocktime بزرگتر یا مساوی همچنان وجود دارد. عملگر CLTV یک پارامتر ورودی (ارتفاع بلاک یا زمان سررسید)، با فرمت مشابه پارامتر ورودی عملگر nLocktime، می‌گیرد. همان طور که پسوند VERIFY در نام CLTV نشان می‌دهد، این عملگر از آن دسته عملگرهایی است که مقدار TRUE یا FALSE برمی‌گردانند: مقدار برگشتی FALSE بلافاصله اجرای اسکریپت اعتبارسنجی را قطع می‌کند؛ مقدار برگشتی TRUE اجرای این اسکریپت را ادامه می‌دهد.

برای قفل کردن یک خروجی با CLTV، این عملگر در داخل اسکریپت وصول خروجی تولیدکننده‌ی آن UTXO قرار داده می‌شود. برای مثال، اگر آلیس بخواهد یک پرداخت به باب انجام دهد، این خروجی معمولاً حاوی یک اسکریپت P2PKH (پرداخت-به-دَرهم-کلید-عمومی) به صورت زیر خواهد بود:

```
DUP HASH160 <Bob's Public Key Hash> EQUALVERIFY CHECKSIG
```

برای قفل کردن این پرداخت برای زمان سررسید معین (مثلاً، ۳ ماه دیگر)، این تراکُنش باید به یک تراکُنش P2SH با اسکریپت وصول به صورت زیر تبدیل شود:

```
<now + 3 months> CHECKLOCKTIMEVERIFY DROP DUP HASH160 <Bob's Public Key Hash>
EQUALVERIFY CHECKSIG
```

که در آن <now + 3 months> ارتفاع بلاک یا زمان تخمینی از لحظه‌ی استخراج این تراکُنش است، که می‌توان آنها را چنین محاسبه کرد: ارتفاع بلاک فعلی + ۱۲,۹۶۰ بلاک (تعداد تقریبی بلاک‌هایی که در ۳ ماه استخراج می‌شوند)، یا زمان فعلی (با فرمت برچسب زمانی شبه-یونیکس) + ۷,۷۶۰,۰۰۰ ثانیه (تعداد ثانیه‌های موجود در ۹۰ روز، پس از کسر زمان تقریبی استخراج این بلاک). معنای عملگر DROP را که بعد از CHECKLOCKTIMEVERIFY آمده، در ادامه توضیح خواهیم داد.

برای خرج کردن این UTXO، باب یک تراکُنش با ارجاع به این UTXO به عنوان ورودی ایجاد می‌کند؛ امضا و کلید عمومی خود را در اسکریپت بازکننده‌ی قفل این ورودی قرار می‌دهد؛ به متغیر nLocktime این تراکُنش مقداری بزرگتر یا مساوی قفل‌زمانی تعیین‌شده در عملگر CHECKLOCKTIMEVERIFY تراکُنش آلیس می‌دهد؛ و سپس این تراکُنش را روی شبکه‌ی بیت‌کوین منتشر می‌کند.

تراکُنش باب به صورت ذیل ارزیابی (اعتبارسنجی) خواهد شد. اگر مقداری که آلیس در عملگر CHECKLOCKTIMEVERIFY تراکُنش خود مشخص کرده، کوچکتر یا مساوی فیلد nLocktime تراکُنش باب باشد، اجرای اسکریپت اعتبارسنجی (درست مثل اجرای یک عملگر NOP: «هیچ عملی اتفاق نیفتاده») ادامه می‌یابد؛ در غیر این صورت، اجرای اسکریپت اعتبارسنجی قطع شده و این تراکُنش به عنوان نامعتبر علامت‌گذاری خواهد شد.

به بیان دقیق، عملگر CHECKLOCKTIMEVERIFY در موارد زیر مقدار FALSE برگردانده و باعث قطع شدن اجرای اسکریپت اعتبارسنجی و نامعتبر شدن تراکُنش می‌شود (منبع: BIP-65):

۱. پشته خالی باشد؛ یا

۲. مقدار درایه‌ی بالای پشته کوچکتر از ۰ باشد؛ یا

۳. نوع قفل‌زمانی (ارتفاع بلاک یا برچسب زمانی) در درایه‌ی بالای پشته و فیلد nLocktime یکسان نباشد؛ یا

۴. مقدار درایه‌ی بالای پشته بزرگتر از فیلد nLocktime تراکُنش باشد؛ یا

۵. فیلد nSequence (شماره ترتیب) ورودی مقدار 0x FFFFFFFF داشته باشد.

**نکته**

عملگرهای CLTV و nLocktime هر دو از یک فرمت واحد برای بیان زمان‌قفل استفاده می‌کنند: ارتفاع بلاک یا زمان (تعداد ثانیه‌ها) سپری‌شده از اول ژانویه ۱۹۷۰. وقتی از این دو عملگر در یک تراکُنش استفاده می‌کنید، بسیار مهم است که هر دو فرمت یکسانی داشته باشند؛ به بیان دیگر، مقدار این دو فیلد یا باید ارتفاع بلاک باشد یا زمان بر حسب ثانیه.

بعد از اجرای اسکریپت اعتبارسنجی، اگر عملگر CLTV با موفقیت اجرا شود (یعنی مقدار TRUE برگرداند)، پارامتر زمان ماقبل آن همچنان در بالای پشته باقی می‌ماند و بایستی دور انداخته شود تا اختلالی در اجرای عملگرهای بعدی اسکریپت اعتبارسنجی ایجاد نکند؛ عملگر DROP برای همین منظور به کار رفته است. همان طور که در ادامه خواهید دید، در اغلب اسکریپت‌های این فصل عملگر DROP بعد از CHECKLOCKTIMEVERIFY می‌آید.

با استفاده‌ی همزمان از CLTV و nLocktime، سناریویی که در قسمت قبل دیدیم (امکان خرج کردن مجدد UTXO مورد اشاره در تراکُنش باب توسط آلیس) دیگر رُخ نخواهد داد. از آنجا که آلیس اکنون خودِ UTXO را قفل کرده، خرج کردن این UTXO قبل از منقضی شدن سررسید ۳-ماهه دیگر برای هیچ کس (حتی خود آلیس) ممکن نخواهد بود.

عملگر CLTV در استاندارد BIP-65 (CHECKLOCKTIMEVERIFY) تعریف شده است. CLTV با ایجاد امکان استفاده از قفل‌های زمانی در زبان اسکریپت‌نویسی بیت‌کوین، به برنامه‌نویسان اجازه می‌دهد تا اسکریپت‌های بسیار پیچیده و جالب بنویسند.

## قفل‌زمانی نسبی

عملگرهای nLocktime و CLTV هر دو *قفل‌زمانی مطلق* هستند، یعنی به یک زمان مشخص در آینده اشاره می‌کنند. در این قسمت دو *قفل‌زمانی نسبی* معرفی می‌کنیم که زمان را به عنوان شرطی از خرج شدن یک خروجی می‌سنجند، یعنی زمان سپری‌شده از لحظه‌ی تأیید آن خروجی در بلاک‌چین.

قفل‌زمانی نسبی ابزاری بسیار سودمند است چون اجازه می‌دهد دو یا چند تراکُنش وابسته همچنان ارتباط خود را حفظ کنند، و در عین حال با آن می‌توان محدودیت زمانی روی یک تراکُنش اِعمال کرده و آن را وابسته به زمان سپری‌شده از تأیید تراکُنش قبلی کرد. به عبارت دیگر، ساعت این قبیل تراکُنش‌ها فقط از لحظه‌ای ثبت شدن UTXO مربوطه در بلاک‌چین شروع به شمارش می‌کند. این ویژگی به خصوص در *کانال حالت دو-طرفه* و *شبکه‌ی آذرخش* (که در فصل ۱۲ خواهید دید) کاربرد زیادی دارد.

قفل‌زمانی نسبی هم (مانند قفل‌زمانی مطلق) می‌تواند در دو سطح - تراکُنش و سطح - اسکریپت پیاده‌سازی شود. *قفل‌زمانی نسبی سطح-تراکُنش* [که در استاندارد BIP-68 تعریف شده است] به عنوان یک اجماع درباره‌ی مقدار فیلد nSequence (شماره ترتیب)، فیلدی که در تمام ورودی‌های تراکُنش وجود دارد، پیاده‌سازی می‌شود. *قفل‌زمانی نسبی سطح-اسکریپت* [که در استاندارد BIP-112 تعریف شده] را هم می‌توان با استفاده از عملگر CHECKSEQUENCEVERIFY، یا CSV، پیاده‌سازی کرد. استانداردهای BIP-68 و BIP-112 در می ۲۰۱۶ به عنوان یک انشعاب نرم در قواعد اجماع بیت‌کوین فعال شدند.

## قفل‌زمانی نسبی با nSequence

با مقدار دادن به فیلد nSequence هر یک از ورودی‌های تراکُنش، می‌توان به هر ورودی آن یک قفل‌زمانی نسبی اختصاص داد.

### مفهوم اولیه‌ی nSequence

هدف اولیه از طراحی فیلد nSequence این بود که بتوان تراکُنش‌ها را در *مخزن حافظه* دستکاری کرد (ولی این عملکرد هرگز به درستی پیاده‌سازی نشد). در این عملکرد، اگر مقدار فیلد nSequence در ورودی‌های یک تراکُنش کوچکتر از $2^{32}$ (0x FFFFFFFF) باشد، به معنای آن است که این تراکُنش هنوز «نهایی نشده است.» یک تراکُنش «نهایی‌نشده» تا زمانی که تراکُنش دیگری همان ورودی‌ها را با مقدار nSequence بالاتر خرج نکرده باشد، در مخزن حافظه نگه داشته خواهد شد. همین که یک تراکُنش که مقدار فیلد nSequence در ورودی‌های آن معادل $2^{32}$ باشد، دریافت شود، این تراکُنش «نهایی شده» و استخراج خواهد شد.

همان طور که گفتیم، عملکرد اولیه‌ی nSequence هرگز به درستی پیاده‌سازی نشد، و به فیلد nSequence تراکُنش‌هایی که از قفل‌زمانی استفاده نمی‌کنند، به طور سنتی مقدار $2^{32}$ داده می‌شود. اگر یک تراکُنش دارای قفل‌زمانی nLocktime یا CLTV باشد، به فیلد nSequence باید مقداری کوچکتر از $2^{32}$ داده شود تا قفل‌زمانی غیرفعال نشود. به فیلد nSequence این قبیل تراکُنش‌ها به طور سنتی مقدار $2^{32}-1$ (0xFFFFFFFE) داده می‌شود.

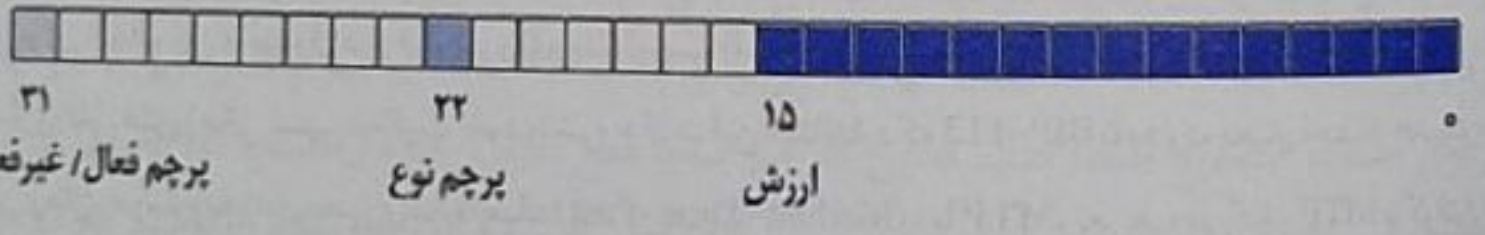
## فیلد nSequence به عنوان یک قفل‌زمانی مبتنی بر اجماع

بعد از فعال شدن BIP-68، برای تراکُنش‌هایی که فیلد nSequence یکی از ورودی‌های آنها مقداری کوچکتر از $2^{31}$ داشته باشد (یعنی، بیت‌های ۱ تا ۳۱ این فیلد مقدار ۰ داشته باشند)، قواعد اجماع جدیدی اِعمال شد. از لحاظ برنامه‌نویسی، این بدان معنا است که اگر بیت منتهی‌الیه سمت چپ فیلد nSequence مقدار ۰ داشته باشد، به عنوان پرچمی تلقی می‌شود که به معنای وجود «قفل‌زمانی نسبی» است. در غیر این صورت (یعنی، اگر بیت‌های ۱ تا ۳۱ این فیلد مقدار ۱ داشته باشند)، از مقدار فیلد nSequence برای کاربردهای دیگر مانند فعال‌سازی CHECKLOCKTIMEVERIFY، nLocktime، OIRBF، و سایر کاربردهایی که ممکن است در آینده توسعه داده شوند، کنار گذاشته می‌شود.

یک تراکُنش با ورودی دارای nSequence کمتر از $2^{31}$ به عنوان یک تراکُنش با «قفل‌زمانی نسبی» تعبیر می‌شود. چنین تراکُنشی فقط زمانی معتبر است که زمان سررسید ورودی آن (تاریخ انقضای قفل‌زمانی) گذشته باشد. برای مثال، یک تراکُنش با قفل‌زمانی نسبی ۳۰ بلاک فقط بعد از استخراج شدن ۳۰ بلاک از زمانی که در UTXO این ورودی اشاره شده، معتبر خواهد شد. از آنجا که هر ورودی تراکُنش یک فیلد nSequence مجزا دارد، تراکُنش‌ها می‌توانند روی هر ورودی خود یک قفل‌زمانی مستقل داشته باشند؛ برای معتبر شدن تراکُنشی با قفل‌های زمانی مختلف روی ورودی‌های متفاوت، بایستی قفل‌زمانی تمامی ورودی‌ها منقضی شوند. یک تراکُنش می‌تواند شامل ورودی‌هایی دارای قفل‌زمانی ($nSequence < 2^{31}$) و ورودی‌هایی بدون قفل‌زمانی نسبی ($nSequence \geq 2^{31}$) باشد.

مقدار nSequence هم بر حسب بلاک یا ثانیه بیان می‌شود، ولی فرمت آن قدری با آنچه در مورد nLocktime دیدیم، متفاوت است. برای تشخیص این که مقدار این فیلد بر حسب بلاک است یا ثانیه، از یک پرچم-نوع استفاده می‌شود. این پرچم-نوع بیت بیست و سوم فیلد nSequence از سمت راست است، و بیت‌های ۱ تا ۲۲ این فیلد مقدار آن را تشکیل می‌دهند. اگر این پرچم-نوع (بیت ۲۳) مقدار ۱ داشته باشد، آنگاه مقدار فیلد nSequence به عنوان مضربی از ۵۱۲ ثانیه تعبیر خواهد شد. اگر مقدار بیت پرچم-نوع ۰ باشد، این فیلد به عنوان تعداد بلاک‌ها در نظر گرفته می‌شود.

وقتی فیلد nSequence به عنوان یک قفل‌زمانی نسبی تلقی شود، فقط ۱۶ بیت سمت راست آن به کار برده می‌شوند. بعد از ارزیابی پرچم‌ها (بیت‌های ۳۲ و ۲۳)، معمولاً مقدار این فیلد با یک ماسک ۱۶-بیتی (مثلاً، به صورت nSequence & 0x0000FFFF) فیلتر می‌شود. شکل ۷-۱ چیدمان باینری مقدار فیلد nSequence در استاندارد BIP-68 را نشان می‌دهد. [برای اطلاعات بیشتر درباره‌ی این استاندارد به آدرس https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki مراجعه کنید.]




**شکل ۷-۱** کُدگذاری فیلد nSequence در استاندارد BIP-68 (منبع: BIP-68).

## قفل‌زمانی نسبی با CSV

درست مثل CLTV و nLocktime، برای nSequence هم یک عملگر اسکریپت وجود دارد که آن را به یک قفل‌زمانی نسبی سطح-اسکریپت ارتقا می‌دهد. این عملگر که CHECKSEQUENCEVERIFY (*بررسی-توالی-اعتبارسنجی*)، یا به اختصار CSV نام دارد، در استاندارد BIP-112 تعریف شده است. وقتی از عملگر CHECKSEQUENCEVERIFY در اسکریپت وصول یک UTXO استفاده می‌شود، آن UTXO فقط در تراکُنشی می‌تواند خرج شود که مقدار nSequence آن بزرگتر یا مساوی پارامتر این CSV باشد. در حالت کلی، این بدان معنا است که UTXO مزبور تا قبل از استخراج بلاک‌ها یا سپری شدن زمان مشخص‌شده (بر حسب ثانیه) نسبت به زمان استخراج آن UTXO نمی‌تواند خرج شود مانند CLTV، فرمت مقدار CSV هم باید با مقدار nSequence متناظر منطبق باشد؛ به عبارت دیگر، مقدار هر دو فیلد یا باید بر حسب ارتفاع بلاک بیان شده باشد یا بر حسب زمان (ثانیه).

قفل‌زمانی نسبی با CSV به خصوص برای مواردی سودمند است که بخواهیم چندین تراکُنش (زنجیره‌ای) ایجاد و امضا کنیم، ولی انتشار آنها را به تعویق بیندازیم، و آنها را خارج از زنجیره نگه داریم. در این حالت، تراکُنش فرزند نمی‌تواند تا زمان انتشار تراکُنش مادر (والد) خرج و استخراج شود، و قفل‌زمانی نسبی آن هم فعال نخواهد شد. یکی از کاربردهای این نوع تراکُنش زنجیره‌ای را می‌توان در *کانال پرداخت* و *کانال حالت* و همچنین *کانال پرداخت هدایت‌شده* (*شبکه‌ی آذرخش*) مشاهده کرد (فصل ۱۲ را ببینید). [برای کسب اطلاعات بیشتر درباره‌ی CSV و استاندارد BIP-112 می‌توانید به آدرس https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki مراجعه کنید.]

## زمان-سپری‌شده-میانه (MTP)

به عنوان بخشی از ساز و کار فعال‌سازی قفل‌های زمانی نسبی، تغییری در چگونگی محاسبه‌ی «زمان» قفل‌های زمانی (مطلق و نسبی) نیز ایجاد شده است. در بیت‌کوین یک تفاوت ظریف، ولی بسیار مهم، بین زمان رسمی (معمولی) و زمان اجماع وجود دارد. بیت‌کوین یک شبکه‌ی غیرمتمرکز است، که در آن هر کسی تصور خاص را خود از زمان دارد. در این شبکه، رویدادها به طور همزمان اتفاق نمی‌افتند، و همه‌ی گره‌ها باید **زمان اختفا (تکوین) شبکه** (network latency time) را در محاسبات خود در نظر بگیرند. اما برای ایجاد یک دفتر کل مشترک همه رویدادها باید سرانجام همزمان شوند. شبکه‌ی بیت‌کوین در هر ۱۰ دقیقه به یک اجماع درباره‌ی حالت این دفتر کل، به صورتی که در *گذشته* وجود داشته، دست می‌یابد.

وظیفه‌ی مقدار دادن به فیلد برچسب زمانی در سرآیند بلاک بر عهده‌ی معدنچی است. قواعد اجماع بیت‌کوین مقدار مشخصی اختلاف بین دقت ساعت در میان گره‌های غیرمتمرکز شبکه را مجاز می‌دانند. با این حال، این بازه‌ی زمانی مجاز یک انگیزه‌ی شوم و نامیمون به معدنچیان داد تا در مورد زمان یک بلاک دروغ بگویند و بتوانند با تجمیع تراکُنش‌های زمان‌داری که هنوز موعد سررسید آنها نرسیده، پول بیشتری (بابت کارمزد تراکُنش) به جیب بزنند. در قسمت بعد توضیحات بیشتری درباره‌ی این قبیل شیوه‌های کلاهبرداری خواهیم داد.

برای حذف این مشوق دروغ‌گویی و تقویت امنیت قفل‌های زمانی، یک BIP مخصوص پیشنهاد شد و همزمان با فعال‌سازی قفل‌های زمانی نسبی فعالیت خود را شروع کرد. این استاندارد که BIP-113 نام دارد، معیار اجماع جدیدی برای سنجش زمان، موسوم به زمان-سپری‌شده-میانه (Median-Time-Past)، یا **MTP**، تعریف می‌کند. MTP با گرفتن میانه [که با میانگین فرق دارد] از برچسب زمانی ۱۱ بلاک آخر محاسبه می‌شود. سپس از این زمان میانه به عنوان زمان اجماع برای محاسبه و پردازش تمام قفل‌های زمانی به کار گرفته خواهد شد. از آنجا که برای محاسبه‌ی ۱۱ بلاک به حدود ۲ ساعت

زمان نیاز است، با در نظر گرفتن نقطه‌ی میانه‌ی این بازه‌ی ۲-ساعته، تأثیر برچسب زمانی هر بلاک منفرد کاهش می‌یابد. با یکپارچه‌کردن ۱۱ بلاک، دیگر هیچ معدنچی نمی‌تواند به تنهایی چنان تأثیری روی برچسب‌های زمانی بگذارد که قادر به پردازش تراکُنش‌های زمان‌دار سررسیدنشده (و به جیب زدن کارمزد آنها) باشد.

با معرفی MTP، پیاده‌سازی محاسبه‌ی زمان برای nLocktime، CLTV، nSequence و CSV نیز دستخوش تغییر شده است. زمان اجماع محاسبه‌شده با MTP همیشه حدود یک ساعت از زمان واقعی (ساعت رسمی) عقب‌تر است. هنگام تخمین زمان مورد نیاز برای قفل‌های nLocktime، CLTV، nSequence و CSV در تراکُنش‌های زمان‌دار بایستی این اختلاف را هم در نظر بگیرید. [برای کسب اطلاعات بیشتر درباره‌ی MTP و استاندارد BIP-113 می‌توانید به آدرس https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki مراجعه کنید.]

## دفاع قفل‌زمانی در مقابل کارمزدرُبایی

کارمزدرُبایی (fee-sniping) یک سناریوی حمله‌ی نظری است که در آن معدنچی تلاش می‌کند از طریق بازنویسی بلاک‌های گذشته، تراکُنش‌های دارای کارمزد بالاتر را از بلاک‌های آینده برباید و سود خود را بیشینه کند. برای مثال، فرض کنید بالاترین بلاک موجود در شبکه بلاک 100,000# باشد، ولی تعدادی از معدنچی‌ها تصمیم می‌گیرند به جای تلاش برای استخراج بلاک 100,001# و توسعه‌ی بلاک‌چین، همان بلاک 100,000# را دوباره استخراج کنند. این معدنچی‌ها می‌توانند هر تراکُنش معتبری (تراکُنشی که هنوز استخراج نشده) را در این بلاک قرار دهند، و مجبور نیستند این بلاک را با همان تراکُنش‌های قبلی استخراج کنند. در واقع این به نفع آنها است که سودآورترین تراکُنش‌ها (تراکُنش‌هایی که حجم و در نتیجه کارمزد بیشتری دارند) را گلچین کرده و در بلاک جدید خود قرار دهند. البته آنها می‌توانند تراکُنش‌های موجود در بلاک 100,000# «قدیمی»، و همچنین هر تراکُنش دیگری را از مخزن حافظه به بلاک «جدید» خود اضافه کنند. در حقیقت، آنها در بازتولید بلاک 100,000# حتی اجازه دارند تراکُنش‌هایی را از «حال» به این «گذشته»ی بازنویسی‌شده بکشند.

در حال حاضر، این ترفند چندان سودآور نیست، چون جایزه‌ی استخراج بلاک‌های جدید بسیار بیشتر از کارمزد کل موجود در هر بلاک است. ولی در آینده‌ای نزدیک، کارمزد تراکُنش به جایزه‌ی اصلی (یا حتی به تنها جایزه) تبدیل خواهد شد؛ در آن زمان، این سناریو اجتناب‌ناپذیر می‌شود.

برای جلوگیری از «کارمزدرُبایی»، وقتی هسته‌ی بیت‌کوین یک تراکُنش جدید ایجاد می‌کند، به طور پیش‌فرض با استفاده از nLocktime آن را به «بلاک بعدی» محدود می‌کند. در مثال بالا، هسته‌ی بیت‌کوین به طور خودکار به هر تراکُنشی که ایجاد کند، مقدار 100,001 خواهد داد. در شرایط عادی این nLocktime بی‌تأثیر است، چون به هر حال این تراکُنش‌ها تنها می‌توانند به بلاک 100,001#(که بلاک بعدی است) اضافه شوند.

ولی در حمله‌ی انشعاب بلاک‌چین، معدنچی‌های متقلب نخواهند توانست تراکُنش‌های دارای کارمزد بالا را از مخزن حافظه بیرون بکشند و به بلاک‌های قدیمی اضافه کنند، چون همه‌ی این تراکُنش‌ها دارای یک قفل‌زمانی به بلاک 100,001# هستند. آنها فقط می‌توانند بلاک 100,000# را با تراکُنش‌هایی که در آن لحظه هنوز معتبر هستند، استخراج مجدد کنند، که این کار هم اساساً هیچ کارمزد جدیدی نصیب آنها نخواهد کرد. برای این منظور، هسته‌ی بیت‌کوین یک واحد به شماره‌ی بلاک فعلی اضافه کرده و این مقدار را به فیلد nLocktime تمام تراکُنش‌ها می‌دهد، و سپس با دادن مقدار 0xFFFFFFFE به فیلد nSequence همه‌ی ورودی‌های تراکُنش، قفل nLocktime را فعال می‌کند.

# کنترل جریان در اسکریپت (عبارت‌های شرطی)

یکی از قدرتمندترین ویژگی‌های زبان اسکریپت‌نویسی بیت‌کوین کنترل جریان (flow control) است، که به *عبارت‌های شرطی* نیز معروف است. اگر قبلاً برنامه‌نویسی کرده باشید، به احتمال زیاد با کنترل جریان که با ساختار IF...THEN...ELSE انجام می‌شود، آشنا هستید. عبارت‌های شرطی بیت‌کوین در اساس همان هستند، ولی ظاهر آنها کمی متفاوت است.

در ساده‌ترین سطح، *عملگرهای شرطی* بیت‌کوین به شما اجازه می‌دهند بسته به TRUE یا FALSE بودن نتیجه‌ی ارزیابی یک شرط منطقی، یک اسکریپت وصول بسازید که قفل آن بتواند به دو روش باز شود. برای مثال، اگر شرط x مقدار TRUE داشته باشد، از اسکریپت A به عنوان بازکننده‌ی قفل استفاده می‌شود، و در غیر این صورت [FALSE بودن شرط x] اسکریپت B بازکننده‌ی قفل خواهد بود.

عبارت‌های شرطی بیت‌کوین را می‌توان به *طور نامحدود* در داخل یکدیگر قرار داد، یعنی در داخل یک عبارت شرطی از یک عبارت شرطی دیگر استفاده کرد، که به آن عبارت‌های شرطی «تو در تو» (nested) گفته می‌شود. بدین ترتیب، امکان ساخت اسکریپت‌های بسیار پیچیده‌ای وجود دارد که صدها یا حتی هزاران مسیر اجرایی محتمل داشته باشند. هر چند تعداد سطوح تو در تو کردن عبارت‌های شرطی بیت‌کوین هیچ محدودیت نظری دارد، ولی قواعد اجماع بیت‌کوین برای اندازه‌ی اسکریپت‌ها یک سقف (بر حسب بایت) تعیین کرده‌اند.

بیت‌کوین برای کنترل جریان اسکریپت از عملگرهای IF، ELSE، ENDIF و NOTIF استفاده می‌کند. علاوه بر این چهار عملگر، عبارت‌های شرطی را می‌توان با استفاده از *عملگرهای منطقی*، مانند BOOLAND، BOOLOR و NOT، با یکدیگر ترکیب کرد.

از آنجا که زبان اسکریپت‌نویسی بیت‌کوین یک زبان پشته- محور است، شاید فکر کنید کنترل جریان چه معنایی می‌تواند داشته باشد. همان طور که عبارت 1 + 1 در زبان اسکریپت‌نویسی بیت‌کوین تبدیل به 1 1 ADD می‌شود، عبارت‌های شرطی بیت‌کوین هم از «آخر به اول» خوانده و اجرا می‌شوند. در زبان‌های برنامه‌نویسی سنتی، کنترل جریان به این صورت است:

```
if (condition):
  code to run when condition is true
else:
  code to run when condition is false
code to run in either case
```

در یک زبان اسکریپت‌نویسی پشته- محور (مثل بیت‌کوین)، شرط منطقی قبل از عملگر IF می‌آید تا بتواند از «آخر به اول» اجرا شود:

```
condition
IF
  code to run when condition is true
ELSE
  code to run when condition is false
ENDIF
code to run in either case
```

وقتی یک اسکریپت بیت‌کوین را می‌خوانید، به یاد داشته باشید که شرط منطقی (که مسیر جریان اسکریپت را کنترل می‌کند) *قبل از* عملگر IF می‌آید.

## عبارت‌های شرطی و عملگر VERIFY

در اسکریپت‌های بیت‌کوین، هر کُد اجرایی یا عملگری که به VERIFY ختم شود، یک عبارت شرطی خواهد بود. پسوند VERIFY یعنی اگر این شرط مقدار TRUE برنگرداند، اجرای اسکریپت بلافاصله قطع شده و آن تراکنش نامعتبر تلقی شود. بر خلاف عبارت IF، که در آن یک مسیر اجرای مستقل برای TRUE نبودن شرط پیش‌بینی شده است، عملگر VERIFY نقش یک عبارت نگهبان را بازی می‌کند و اجرای اسکریپت فقط (و فقط) در صورتی ادامه می‌یابد که آن (پیش)شرط محقق شود.

برای مثال، اسکریپت زیر ارائه‌ی امضای باب و یک پیش-تصویر (کلید سرّی) که دَرهم خاصی تولید می‌کند، می‌طلبد. برای باز کردن قفل این اسکریپت برآورده کردن هر دو شرط ضروری است:

```
HASH160 <expected hash> EQUALVERIFY <Bob's Pubkey> CHECKSIG
```

برای وصل این اسکریپت، باب باید یک اسکریپت بازکننده‌ی قفل بسازد که حاوی یک پیش-تصویر (pre-image) معتبر و امضای خودش باشد:

```
<Bob's Sig> <hash pre-image>
```

بدون ارائه‌ی این پیش-تصویر (پیش‌شرط)، باب نمی‌تواند به آن بخش از اسکریپت بازکننده که امضای او را ارزیابی می‌کند، برسد. اسکریپت بالا را با استفاده از یک عملگر IF هم می‌توان پیاده‌سازی کرد:

```
HASH160 <expected hash> EQUAL
IF
  <Bob's Pubkey> CHECKSIG
ENDIF
```

در سمت باب، اسکریپت بازکننده‌ی قفل هیچ تفاوتی با قبل ندارد:

```
<Bob's Sig> <hash pre-image>
```

این اسکریپت IF دقیقاً همان کار پسوند VERIFY را انجام می‌دهد؛ آنها هر دو به عنوان عبارت نگهبان عمل می‌کنند. با این حال، همان طور که می‌بینید، استفاده از عملگر VERIFY یک اسکریپت فشرده‌تر و کارآمدتر ایجاد کرده است. پس، اگر عملگر VERIFY کارایی بهتری دارد، چرا باید از IF استفاده کرد؟ اگر همه‌ی چیزی که لازم دارید، اضافه کردن یک پیش‌شرط (عبارت نگهبان) است، بهتر است از همان VERIFY استفاده کنید؛ ولی اگر می‌خواهید چندین مسیر اجرایی داشته باشید (و جریان اجرای اسکریپت را در مسیرهای مختلف کنترل کنید)، به یک ساختار کنترلی IF... ELSE نیاز خواهید داشت.

**توجه**

کُدهای اجرایی مانند EQUAL نتیجه‌ی ارزیابی هر عملگر (TRUE/FALSE) را به داخل پشته برمی‌گردانند، تا توسط عملگر بعدی خوانده و ارزیابی شود. بر خلاف آن، پسوند اجرایی EQUALVERIFY هیچ چیزی در پشته باقی نمی‌گذارد. کُدهای اجرایی که به VERIFY ختم می‌شوند، نتیجه‌ی ارزیابی خود را به پشته برنمی‌گردانند.

## استفاده از کنترل جریان در اسکریپت

یکی از رایج‌ترین کاربردهای کنترل جریان در زبان اسکریپت‌نویسی بیت‌کوین، ایجاد اسکریپت‌هایی است که راه‌های متعددی برای وصول یک UTXO پیشنهاد می‌کنند. اجازه دهید به یک مثال ساده نگاه کنیم: آلیس و باب هر دو امضاکننده‌ی مجاز یک اسکریپت هستند و می‌توانند آن را وصول کنند. اگر بخواهیم از اسکریپت‌های چندامضایی برای این کار استفاده کنیم، باید یک اسکریپت «۱-از-۲» بسازیم. اما در اینجا آن را با استفاده از یک ساختار IF... ELSE انجام خواهیم داد:

```
IF
  <Alice's Pubkey> CHECKSIG
ELSE
  <Bob's Pubkey> CHECKSIG
ENDIF
```

در همان نگاه اول به این اسکریپت وصول، از خود می‌پرسید: «پس شرط اسکریپت کجاست؟ چرا چیزی قبل از عملگر IF نیامده است؟» این شرط بخشی از خود اسکریپت وصول نیست، بلکه در اسکریپت بازکننده‌ی قفل ارائه خواهد شد. این کار اجازه می‌دهد تا آلیس و باب مسیر اجرای این اسکریپت را «انتخاب کنند.» برای مثال، آلیس می‌توانند برای وصول این UTXO از اسکریپت بازکننده‌ی قفل زیر استفاده کند:

```
<Alice's Sig> 1
```

مقدار 1 که در انتهای این اسکریپت آمده، به عنوان شرط عبارت IF (در اسکریپت وصول) عمل خواهد کرد و چون مقدار آن TRUE است، باعث می‌شود تا اسکریپت وصول بخش IF (مقایسه با کلید عمومی آلیس) را در پیش بگیرد. اگر باب بخواهد برای وصول این UTXO یک اسکریپت بازکننده‌ی قفل بسازد، می‌تواند از اسکریپت زیر استفاده کند:

```
<Bob's Sig> 0
```

اجرای این اسکریپت باعث می‌شود تا مقدار 0 به داخل پشته فرستاده شود، و چون این مقدار معادل FALSE است، بخش ELSE اسکریپت وصول (مقایسه با کلید عمومی باب) در پیش گرفته خواهد شد.

از آنجا که عبارت‌های شرطی IF را می‌توان «تو در تو» کرد، می‌توان اسکریپت‌هایی با مسیرهای اجرایی متعدد ساخت که در آنها انتخاب مسیر اجرای مورد نظر بر عهده‌ی اسکریپت بازکننده‌ی قفل است:

```
IF
  script A
ELSE
  IF
    script B
  ELSE
    script C
  ENDIF
ENDIF
```

در این سناریو، سه مسیر اجرایی وجود دارد: script A، script B یا script C. اسکریپت بازکننده‌ی قفل با ارائه‌ی یک توالی معین از دو مقدار TRUE یا FALSE مسیر اجرای دلخواه را انتخاب می‌کند. برای مثال، اگر بخواهیم مسیر اجرای script B را انتخاب کنیم، اسکریپت بازکننده‌ی قفل باید به 0 1 (به ترتیب از چپ به راست TRUE و FALSE) ختم شود. وقتی این دو مقدار به داخل پشته فرستاده می‌شوند، مقدار دوم [0 (FALSE)] دربالای پشته قرار می‌گیرد. وقتی اولین عملگر IF این مقدار را از پشته بیرون می‌کشد، مسیر اجرای اسکریپت را به اولین بخش ELSE می‌سپارد. از آنجا که این بخش خود با یک عملگر IF شروع شده است، این عملگر مقدار بعدی [1 (TRUE)] را از پشته بیرون کشیده و در نتیجه دستور بعد از IF دوم، که script B است، را اجرا می‌کند.

با استفاده از این قبیل ساختارها می‌توان اسکریپت‌هایی با دهها یا صدها مسیر اجرایی ساخت که هر کدام مسیری متفاوت برای وصول یک UTXO در پیش می‌گیرند. برای خرج کردن این UTXO کافی است یک اسکریپت بازکننده‌ی قفل با توالی TRUE و FALSE مناسب بسازید که اسکریپت وصل را به مسیر دلخواه هدایت کند.

## یک اسکریپت پیچیده

در انتهای این فصل تعداد زیادی از مفاهیم گفته‌شده را در یک مثال واحد به کار می‌گیریم. در این مثال، محمد [واردکننده‌ی لوازم الکترونیک در دوبی (امارات متحده‌ی عربی)] می‌خواهد با توجه به مقررات شرکت، حساب اصلی آن را با استفاده از اسکریپت‌های وصول زمان‌دار با سطوح مجوز مختلف کنترل کند. کسانی که در این شرکت حق امضای اسکریپت‌ها را دارند، عبارتند از: خود محمد، دو شریکش (سعید و زایر)، و وکیل شرکت (عبدل). تصمیمات بر اساس رأی اکثریت (دو از سه) گرفته می‌شوند. با این حال، اگر مشکلی برای کلید هر یک از این سه شریک پیش بیاید، وکیل شرکت باید بتواند با کلید عمومی خودش به همراه امضای یکی از شرکا آن مبلغ را وصول کند. سرانجام، اگر هیچ کدام از سه شریک در دسترس نباشند، وکیل شرکت باید بتواند این حساب را به طور مستقیم مدیریت کند. اسکریپتی که بتواند این خواسته‌ها را برآورده کند، چنین است:

```
IF
  IF
    2
  ELSE
    <30 days> CHECKSEQUENCEVERIFY DROP
    <Abdul the Lawyer's Pubkey> CHECKSIGVERIFY
    1
  ENDIF
  <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's Pubkey> 3 CHECKMULTISIG
ELSE
  <90 days> CHECKSEQUENCEVERIFY DROP
  <Abdul the Lawyer's Pubkey> CHECKSIG
ENDIF
```

اسکریپتی که محمد نوشته، با استفاده از یک ساختار IF . . . ELSE «تو در تو»، سه مسیر اجرا به وجود می‌آورد. در مسیر اول، شامل خطوط ۳ و ۹، این اسکریپت به عنوان یک اسکریپت چندامضایی «۲-از-۳» ساده عمل می‌کند. خط ۳ حد نصاب چندامضایی را مشخص می‌کند. برای انتخاب این مسیر اجرا فقط کافی است در انتهای اسکریپت بازکننده‌ی قفل دو مقدار متوالی TRUE TRUE (یا 1 1) قرار دهید:

```
0 <Mohammed's Sig> <Zaira's Sig> TRUE TRUE
```

مقدار 0 که در ابتدای این اسکریپت بازکننده‌ی قفل می‌بینید، به دلیل باگ عملگر CHECKMULTISIG در بیرون کشیدن یک مقدار اضافی از پشته است. همان طور که در قسمت‌های قبلی همین فصل توضیح دادیم، این مقدار [که به طور سنتی 0 انتخاب می‌شود] تأثیری در اجرای CHECKMULTISIG ندارد، ولی اگر وجود نداشته باشد، اجرای اسکریپت با خطا متوقف شده و تراکُنش را نامعتبر خواهد کرد.

مسیر دوم یک اسکریپت وصول زمان‌دار با سررسید ۳۰-روزه می‌سازد، که در انتهای این مدت برای باز کردن آن به امضای عبدل (وکیل شرکت) و امضای یکی از شرکا (چندامضایی «۱-از-۳») نیاز است. مقدار 1 که در خط ۷ می‌بینید، حد نصاب چندامضایی این اسکریپت را مشخص می‌کند. برای انتخاب این مسیر فقط کافی است در انتهای اسکریپت بازکننده‌ی قفل دو مقدار متوالی TRUE FALSE (یا 1 0) قرار داده شود:

```
0 <Saeed's Sig> <Abdul's Sig> FALSE TRUE
```

چرا TRUE FALSE؟ مگر نباید برعکس باشد؟ از آنجا که این دو مقدار به داخل پشته فرستاده می‌شوند، TRUE بالای FALSE قرار خواهد گرفت. به عبارت دیگر، اولین IF ابتدا TRUE را از بیرون می‌کشد و بعد FALSE را.

سرانجام، سومین مسیر به عبدل اجازه می‌دهد این UTXO را به تنهایی خرج کند، ولی فقط بعد از گذشت ۹۰ روز. برای انتخاب این مسیر اسکریپت بازکننده‌ی قفل باید به یک مقدار FALSE (یا 0) ختم شود:

```
<Abdul's Sig> FALSE
```

سعی کنید با اجرای این اسکریپت روی کاغذ، درستی عملکرد آن را امتحان کنید.

این اسکریپت نکات زیادی را به ذهن متبادر می‌کند. برای مثال، ببینید آیا می‌توانید به پرسش‌های زیر پاسخ دهید:

- آیا وکیل شرکت (عبدل) نمی‌تواند با قرار دادن یک مقدار FALSE در انتهای اسکریپت بازکننده همیشه مسیر سوم را انتخاب کند؟
- بعد از گذشت ۵، ۳۵ و ۱۰۵ روز از استخراج این UTXO، چند مسیر ممکن وجود دارد؟
- اگر وکیل شرکت کلید خود را گم کند، آیا مبالغ دریافتی شرکت دیگر قابل‌وصول نخواهند بود؟ بعد از ۹۱ روز چطور؟
- این سه شریک چگونه می‌توانند ساعت را در هر ۲۹ یا ۸۹ روز «ریست» کنند تا مانع از دسترسی انحصاری عبدل به حساب شرکت شوند؟
- چرا برخی از عملگرهای CHECKSIG در این اسکریپت پسوند VERIFY دارند، و برخی دیگر خیر؟