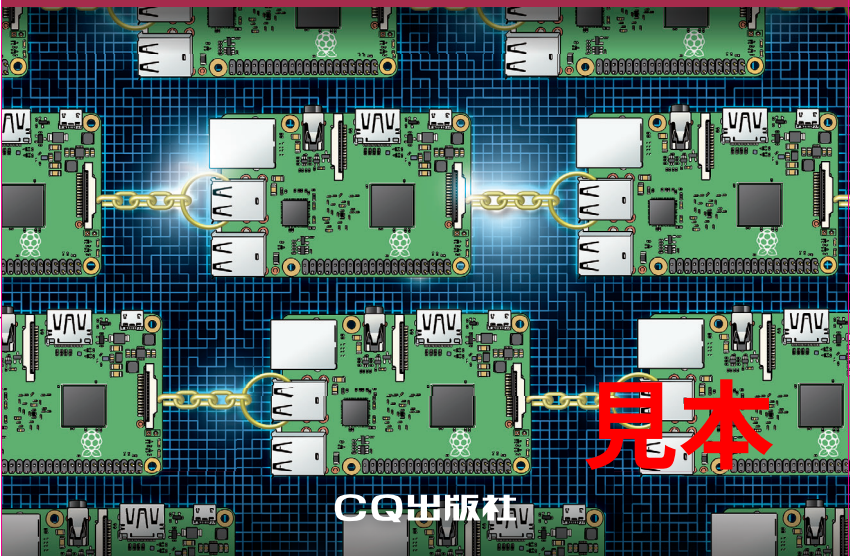
CQ BUNKO SERIES

マイニングや高セキュリティ通信を体験

ラズパイで作るブロックチェーン暗号コンピュータ

佐藤 聖 他 著
Sei Sato



CQ出版社

第2章

基礎知識

■ 知っておかないとマズい理由

ブロックチェーンが話題になる理由は，画期的プラットフォームだからです．このプラットフォームでは，お金の取引だけでなく，あらゆるデータの収集や記録に利用できるため応用範囲が広く，あらゆるビジネスにも影響を与える可能性があります．

プラットフォームは一度普及すると発電事業や道路事業のように長期間インフラとして利用されます．そのため世界中でブロックチェーンによるプラットフォームが開発され，事実上の業界標準として認知されるよう，覇権をめぐって競争が繰り広げられています．

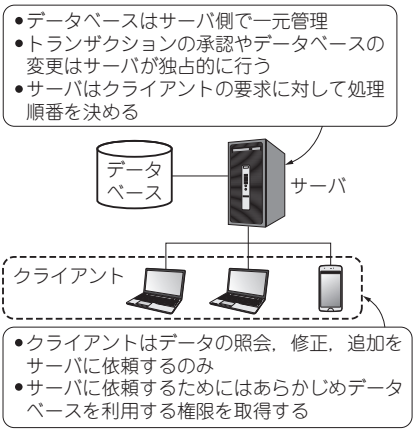
ブロックチェーンはいろんなネットワーク技術を積み重ねたものであるため(詳細は後述)，暗号通貨ではないIoT的な使い方に注目！といいながらも，暗号通貨という言葉が出てきてしまうのは説明上許してください．

■ 一番の特徴…ネットワークにサーバが要らない

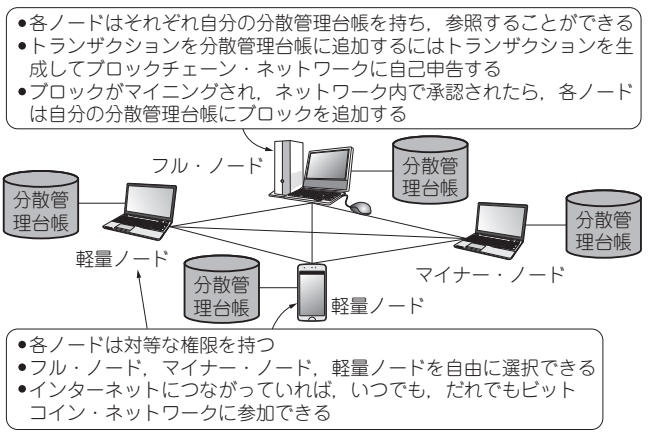
● 従来の中央集権型

従来の中央集権型データベース(**図1**)は，Tポイント，Ponta，楽天ポイント，dポイントなどのポイント・システムや，Suicaなどの電子マネー決済システムに利用されています．こうしたシステムは運用会社が存在し，管理者によってデータが管理されています．

中央集権型データベースの代表例として，リレーショナル



（a）これまでの中央集権型



（b）これからの分散型

図1　これからのネットワークは「分散型」が注目

ータベースがあります．コンピュータでデータを「保存/編集/参照」するためには，Oralce，SQL Server，DB2，MySQL，PostgreSQLなどのデータベース管理ソフトウェアがあります．

データベースの台帳機能は基本的にテーブルが担っており，関係モデルを表現しています．データベースはデータを記録するだけでなく，加工や参照にも利用されます．

中央集権型データベースは，サーバ障害や通信障害によってサービスが停止してしまうと，クライアントからデータ操作ができなくなります．一元的に管理するため障害が発生しても復旧まで短期間で対応されるよう，事前に障害対策が施されていることが一般的です．

● これから注目の分散型

ブロックチェーンのデータはPeer to Peer(P2P)ネットワーク，分散型タイム・スタンプ・サーバによって自律的に管理されます．中央集権型データベースのように管理者がいないのも特徴です．

ブロックチェーンでは，データの取引を認証するための組織や管理者が存在せず，自立的に管理する仕組みが備わっています．

ブロックチェーンにはサーバが存在しないため，サーバ障害がなく，通信障害が発生したとしても一部のノード(マイナーのコンピュータ)にしか影響せず，全体に障害の影響が波及しません．逆に言うとインターネット上のどこかのネットワークに接続しているノードに障害が発生しても，常に全体として機能するよう考慮されています．

万が一，ノードやネットワークに障害が発生しても，正常に稼働しているノードだけでトランザクションの作成，マイニング，トランザクションの認証などが進むので，データの取引全体が停止することがありません．

インターネットによる通信は専用線や地上電話ほど安定していません．ハードウェア障害，トラフィック量の急増，サイバー攻撃などで通信できなくなることがよくあります．ブロックチェーンでは中央管理のデータベースよりも障害に対する高い耐性が

ります.

■ 技術の階層

ブロックチェーンは，分散型台帳技術，分散型ネットワーク技術と呼ばれ，用途や機能の違いでブロックチェーン1.0，2.0，3.0があります(**表1**)．各バージョンの定義は組織やプロジェクトで異なりますがおおよそ次の通りです.

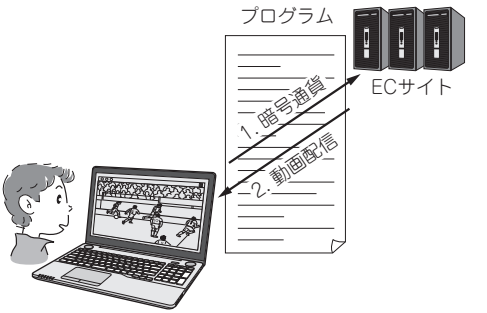
- ブロックチェーン1.0…暗号通貨への応用で支払い機能に特化

表1　ブロックチェーンはいろんなネットワーク技術を組み合わせて実現する

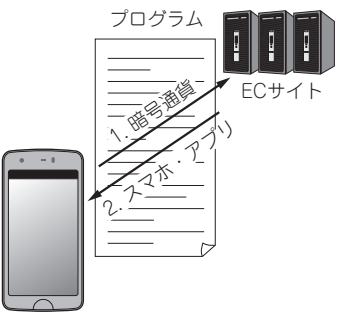
バージョン	主な用途	主要アプリケーション	主要ブロックチェーン技術	機能
1.0	暗号通貨	ビットコインの取引をサポートするプラットフォーム	ブロックチェーン	暗号通貨のP2P取引
		アルトコインの取引をサポートするプラットフォーム	アルトチェーン	
2.0	デジタル情報取引	スマートコントラクトに代表されるデジタルアセット	カラードコイン	Open Assets Protocol(権利情報)など
			パーミッションチェーン	参加者の権限設定など
			サイドチェーン	異なるブロックチェーンを連携させる
			マイクロペイメント	トランザクションはブロックチェーン外で処理し、ブロックチェーンにコミットすることでパフォーマンスとコスト効果を得る
3.0	ブロックチェーンのプラットフォーム化やサービス化	ブロックチェーンサービス(BaaS)	上記を含む	ブロックチェーン上にアプリケーションを構築

- ブロックチェーン2.0…暗号通貨以外への決済用途等で利用される．契約の自動実行(スマート・コントラクト)機能(**図2**)など
- ブロックチェーン3.0…金融以外の分野でのIoTや人工知能と組み合わせた応用(ブロックチェーン2.0を含む)

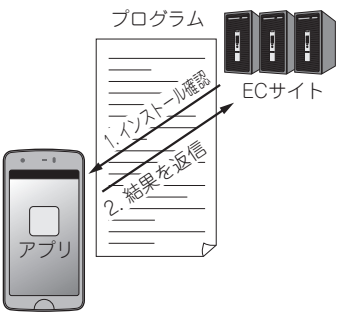
暗号通貨だけでも1500種類以上あり，暗号通貨以外のブロッ



(a) 契約の定義…あらかじめ暗号通貨の入金があったら動画を配信するような契約をプログラムで定義する



(b) 契約の自動執行…契約を定義したプログラムによって利用者からの入金をトリガにしてスマホ・アプリを自動的にスマホへ送信



(c) 実行結果の監査…契約が正しく実行されたか確認

図2　契約の自動実行機能(スマート・コントラクト機能)**をベースにすると簡単に安全なやりとりが可能になる**

クチェーンを活用したプラットフォームもあるので，バージョンによってコンセンサスを得ることが難しいと思います．

● 始めるならシンプル構造のブロックチェーン1.0から

ブロックチェーン技術を理解するには，暗号通貨の仕組みを理解するのが早道です．ブロックチェーン1.0は一番シンプルな構造なので，初めてブロックチェーンを学習するときに最適です．

ブロックチェーンは1つの技術ではなく，ネットワーク，データベース，セキュリティ，ハードウェアなどの技術を複合的に組み合わせた技術(**図3**)ですので，一見すると複雑に見えます．1つ1つの技術は従来からあるものなのでご存じかもしれません．今後もコンピュータ技術の進歩や新しいアーキテクチャの開発によってブロックチェーン技術は進化していくと思いますが，基本的な概念は変わらないと思います．

自動車業界は早い時期からブロックチェーンの活用に取り組ん

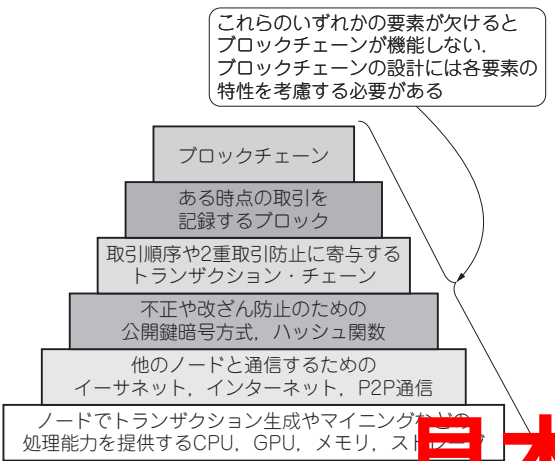


図3　ブロックチェーンの技術階層

でおり，日本では金融業界よりも活用方法の研究が進んでいると思います．例えば自動車の管理や制御，充電システム，自動車部品販売，中古自動車販売などにブロックチェーン技術を応用するような使い方です．自動車業界でブロックチェーン技術の活用が始まれば他の業界でも同様のブロックチェーン・プラットフォームが普及していくと感じています(1)(2)．

■ ネットワークへの参加者

ブロックチェーンの利用者はさまざまです(**図4**)．個人，ユーザ・グループ，一般企業，金融機関，中央銀行，政府機関，国際機関，NPO団体など，国境を越えて多様な利用者があり得ます．

● その1：マイナー

▶センサの場合

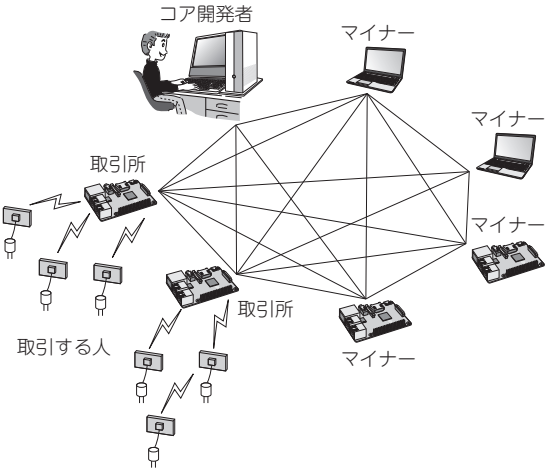
自作のブロックチェーンや，バージョン2.0/3.0のブロックチェーン・プラットフォームでは，ラズベリー・パイがマイナー・ノードとして利用できます．

ASICやGPUなどを搭載していなくても，IoT端末でマイニングが可能になります．筆者としてもラズベリー・パイを利用してセンサ・データを取引するような実験も試してみたいと考えています．

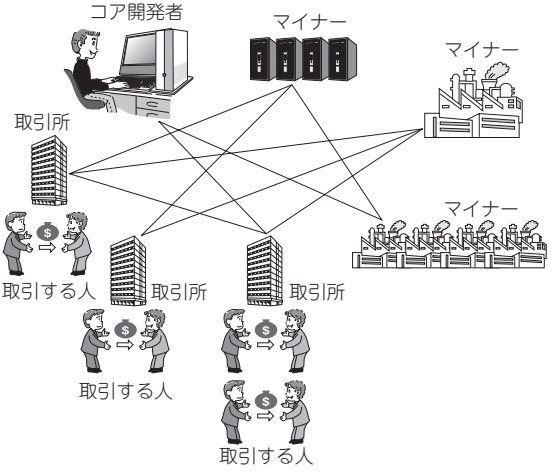
▶ちなみに…通貨の場合

取引は何かしらの方法で管理しなければ不正が横行して信頼できる取引が不可能になります．マイナーは取引を検証してブロックに記録する重要な役割を果たしています．マイニングによって生成された不正なトランザクションや暗号通貨の2重使用などを防ぐ役割を担っています．

通貨の場合はマイニングの専用工場があります．[redacted]のマイニング・ファームは以前からたくさんあるので有名です．



（a）センサ・データの場合



（b）暗号通貨の場合

図4　ブロックチェーンへの参加者にはいくつか種類がある

第3章

台帳のデータ構造

■ チェーンのつながり方

ブロックチェーンはトランザクションを格納し記録する台帳です(**図1**)．この台帳は各ノードがそれぞれ保有しており参照や更新を行います(**図2**)．ブロックチェーンは，複数のブロックがチ

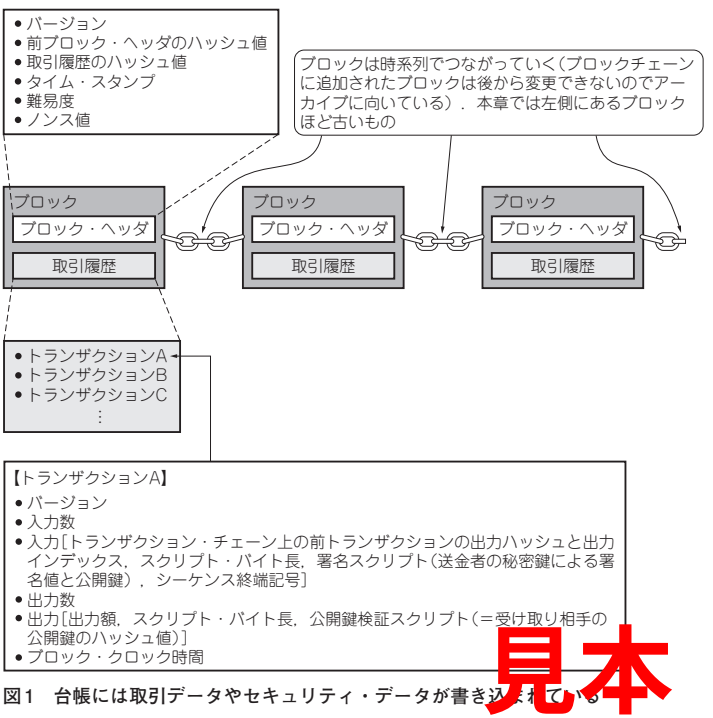


図1　台帳には取引データやセキュリティ・データが書き込まれている

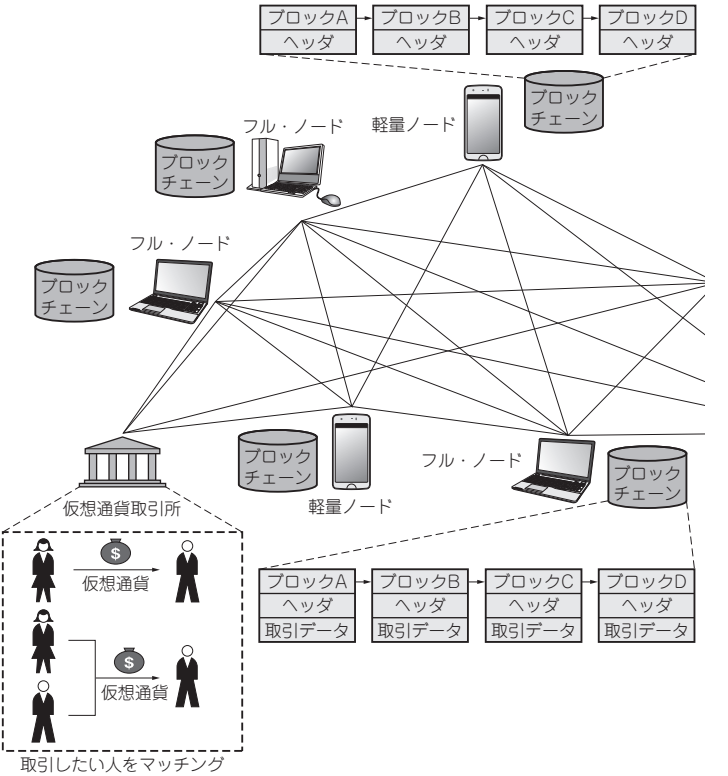


図2　台帳は各ノードがそれぞれ保有しており参照や更新を行う

ェーンのようにつながったイメージです．例えば「Proof of Work：PoW」(暗号通貨によく利用される)による計算で作られるブロック［**図3**(**a**)］は主に，ブロック・ヘッダと取引履歴とでできています．

ブロック・ヘッダ［**図3**(**b**)］の中には，前ブロック・ヘッダのハッシュ値，取引履歴のハッシュ値，ノンス値，その他ヘッダが含まれます．

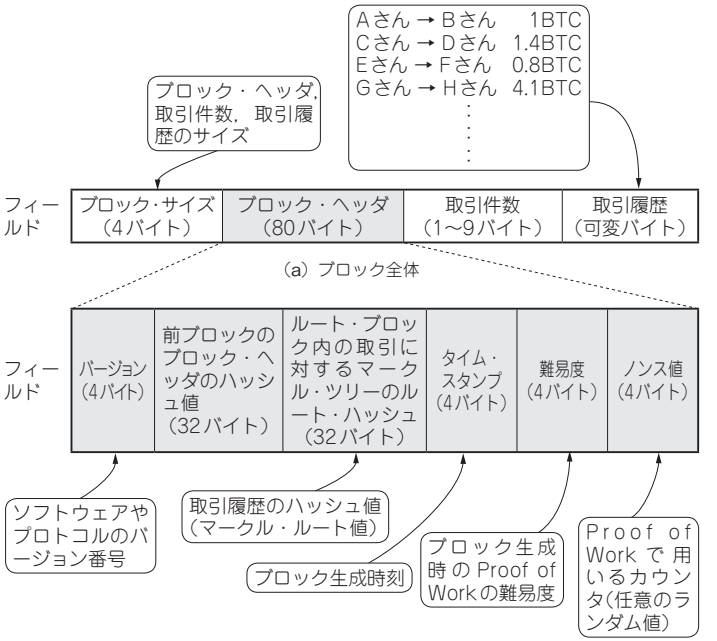


図3　台帳（ブロック）の構造

■ ヘッダの中身①…前ブロック・ヘッダのハッシュ値

「前ブロック・ヘッダのハッシュ値」は，前ブロックとの論理的なつながりを保持します．これによって前ブロックと現ブロックとのつながりの正しさを検証できます．

■ ヘッダの中身②…取引履歴ツリーのルート・ハッシュ値

● 取引そのものはブロック内に残っていなくてもよい

ブロック・ヘッダには，取引データから算出したハッシュ値［**図4**(c)］[注1]が格納されていて，マークル・ツリーのルート・ハッ

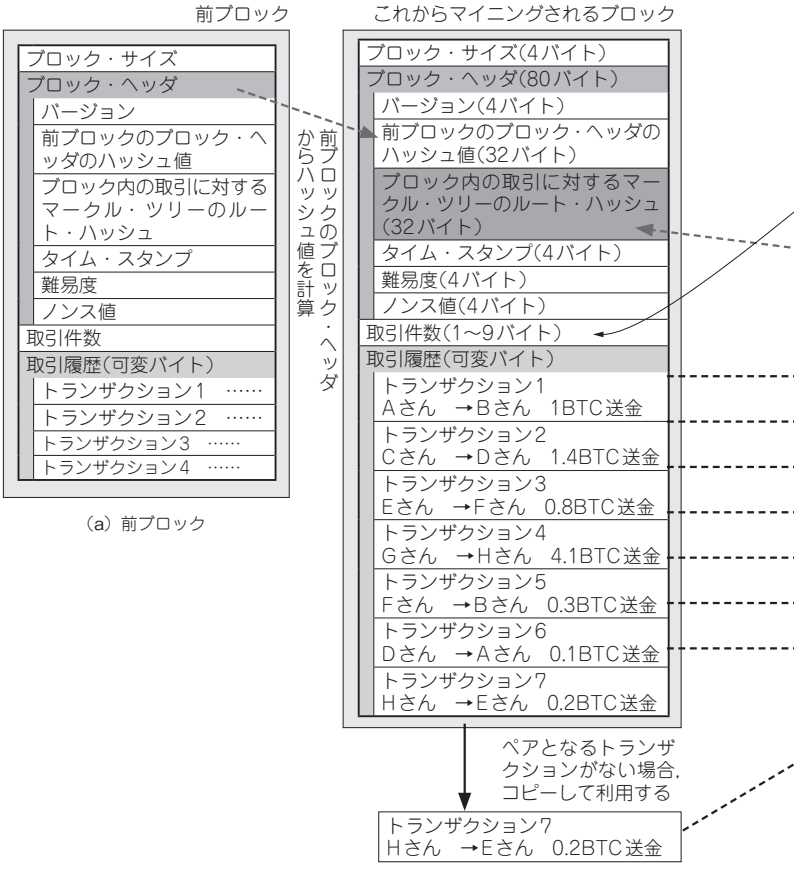
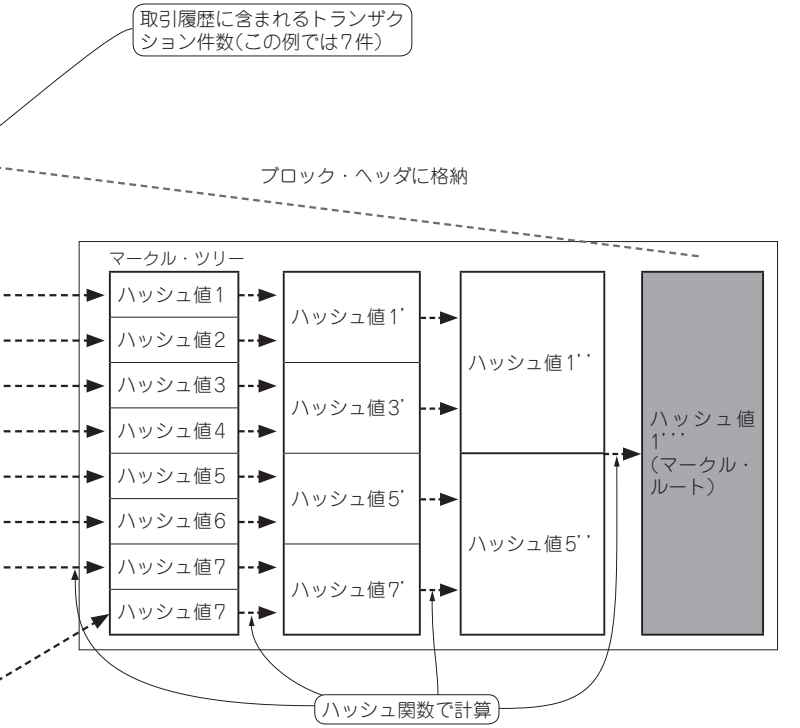


図4 マークル・ツリーのルート・ハッシュは取引データから生成する

注1：ハッシュ値は，ハッシュ関数を使って作ります．ハッシュ関数は，任意長のビット列から規則性のない固定長のビット列を生成してくれる関数です．詳しくは，○章で解説しています．



(c) 取引履歴からハッシュ値を求める

シュといいます.

ハッシュ値を格納しておくことで,取引データそのものはブロック内に格納しておかなくても,取引データがブロックの要素として正しい組み合わせであることを検証できます.

取引履歴のハッシュ値は,取引の不正や整合性を確認するため

に利用されます．ブロック・ヘッダと取引履歴とをつなげる重要な情報です．ブロック・ヘッダにルート・ハッシュを書き込むことで，軽量ノードは一部の取引履歴だけを保持していればよく，過去から現在までの全ての取引データを保持する必要はありません．

● 過去の取引を検証できるメカニズム

軽量ノード利用者が，過去の取引データを検証する方法を通貨の例で示します．まず，P2P通信によってウォレット内の取引履歴を他ノードに伝えます．フル・ノードA（例えばAとしただけ）は，ブルーム・フィルタ注2に合致するトランザクションを確認し，見つけるとブロック・ヘッダとマークル・パス（マークル・ツリーの取引データやハッシュ値をつなげているパス）を軽量ノードに返します．軽量ノードは，マークル・パスを使って検証したい取引データがブロック内に含まれていることを確認します．

こうすることでビットコイン・アドレスをネットワーク上に送信することなく取引データを検証できます．暗号通貨で利用されるブロックチェーンで単なるデータから通貨としての役割が果たせるようになるために重要な仕組みの1つです．

図5の軽量ノードでGさんが「Gさん→Hさん4.1BTC送金」の取引を検証しようとした場合，検証（概要）の流れは以下の通りです．

1. Gさんの軽量ノードはウォレットが持っている全てのビットコイン・アドレスをリストにします．
2. リストからHさんへ送金した際のビットコイン・アドレスにひも付けられたトランザクション・アウトプットか

注2：ウォレットに入っている特定のビットコイン・アドレスを隠す目的で生成される探索パターン．ビットコイン・アドレスを相手に伝えずに済みます．あるブロックに検索パターンが含むか否かを調べるのに利用します．

注3：軽量ノードがP2SHアドレスの残高をトラッキングしている場合，ペイー・パブリック・ハッシュ・スクリプトを作成します．

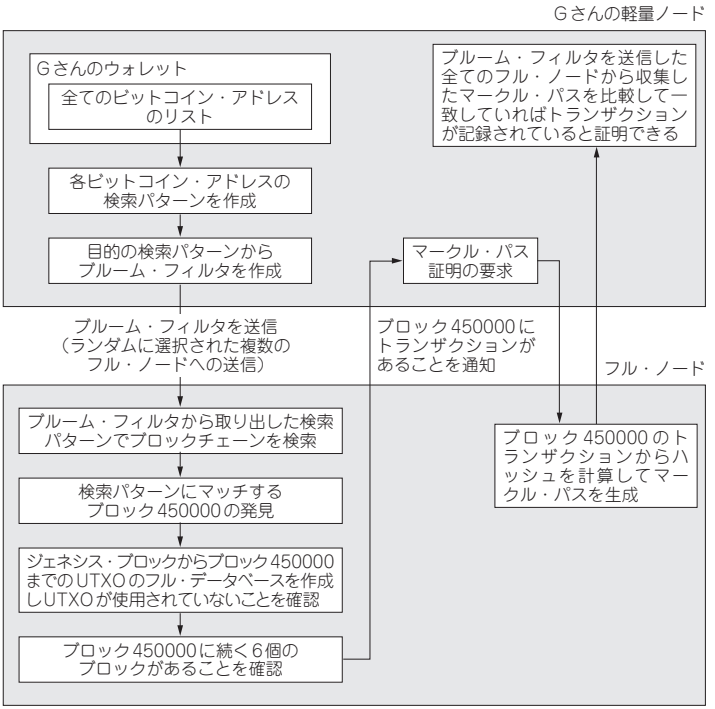


図5　取引を検証する流れ
「Gさん→Hさん4.1 BTC送金」の例

ら，検索パターン（パブリック・ハッシュ・スクリプト）を作成します[注3].

3. 探索パターンから「ブルーム・フィルタ」（N個のビット列とM個のハッシュ関数で構成）を作成します.
4. 軽量ノードからブルーム・フィルタを送ります．ランダムに選択した幾つかのフル・ノードに，P2P通信によって送信し，探索パターンにマッチするトランザクションの調査を依頼します.
5. フル・ノードで検索パターンにマッチするブロック

450000が見つかったら，ジェネシス・ブロック（最初のブロック）からブロック450000までを結びつけるUTXO（特定の所有者にロックされた分割不可能なビットコインの固まり）のフル・データベースを作成し，トランザクションの検証のためUTXOで使用されていないことを確認します．これによりウォレットに残高として残っているビットコイン・アドレスを特定できます．

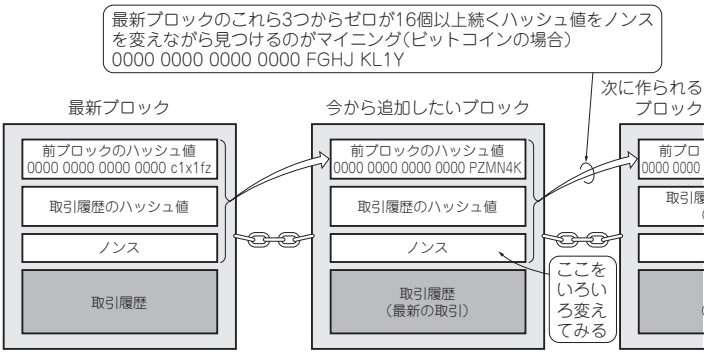
6. ビットコイン・ネットワーク上の他のノードでもブロック450001～450006を確認して目的のブロックに6個のブロックが接続されていることを事実としてトランザクションが2重に使用されていないことを確認します．
7. フル・ノードは軽量ノードに検索パターンにマッチするブロック450000を伝え，軽量ノードはブロックに目的のトランザクションが含まれるかを検証するため，ランダムに選択した複数のフル・ノードへマークル・パス証明を要求します．
8. 軽量ノードは全てのフル・ノードからマークル・パス証明を受け取り，比較することでそのブロックにトランザクションの記録が存在することを証明できます．

■ ヘッダの中身③…重要なランダム値「ノンス値」

● 見つけたらマイニング成功

ブロックを生成するために必要な任意のランダム値が格納されます．前ブロック・ヘッダのハッシュ値，取引履歴のハッシュ値，その他ヘッダにノンス値を加えて，次ブロック・ヘッダに格納されるハッシュ値を算出します［**図6(a)**］．

ビットコインでは，ブロックのハッシュ値を一定以下の数値にさせるノンス値を発見しなければ［**図6(b)**］，ブロックチェーンにブロックを格納することができません．つまりノンス値が発見



(a) 今追加したいブロックのノンスをあれこれ変える

前のブロックのハッシュ値	今のブロックのハッシュ値	取引データ	ノンス	次のブロックのハッシュ値
0000000000000000 g84uy90gqh13kjs9	0000000000000000 tyu53b7i9fr56mlp	aabbccddeeffgghh	00000000	b8ci9d77hag4a7l2jj4groi3gal33j
0000000000000000 g84uy90gqh13kjs9	0000000000000000 tyu53b7i9fr56mlp	aabbccddeeffgghh	00000001	89jfiogeo1314kgoqk90ko093d
0000000000000000 g84uy90gqh13kjs9	0000000000000000 tyu53b7i9fr56mlp	aabbccddeeffgghh	00000002	b74gl5ajgag984u4qgqklgqegoq3ijtgqo
0000000000000000 g84uy90gqh13kjs9	0000000000000000 tyu53b7i9fr56mlp	aabbccddeeffgghh	⋮	6hr9gaskfasfasgasgakjglakjglkajglka
0000000000000000 g84uy90gqh13kjs9	0000000000000000 tyu53b7i9fr56mlp	aabbccddeeffgghh	⋮	uu8gegagjaiglkagnaket3tqt34q4gr
0000000000000000 g84uy90gqh13kjs9	0000000000000000 tyu53b7i9fr56mlp	aabbccddeeffgghh	1gb78dko	00000000000000009x13vci1kld90op

少しずつ変えていく

このノンスだ！

0が16個続くハッシュ値が登場！

(b) ノンスが見つかるまで

図6　新規ブロックを追加するためにはその次のブロックに格納されるべきハッシュ値を求める

できないとマイニングに成功しません.

正解のノンス値を見つける問題が解けると，過去の全取引履歴を格納したブロックチェーンを更新できるの[redacted]な[redacted]な権限です．マイニングが改ざんの抑止力として働き[redacted]成功[redacted]ばマ

第1章

Jupyter Notebookで ステップ・バイ・ステップ

■ 試す準備

第2部 Appendix1のように，Jupyter Notebookの実行プログラムを格納してあるイメージ・ファイル「Interface201808_RPi.img」を用意します．

ラズベリー・パイが立ち上がったら，コマンド・プロンプトから以下の4つのコマンドを実行してライブラリをインストールします．

```
$ sudo apt-get update && sudo apt-get upgrade↵
$ pip3 install graphviz↵
$ pip3 install merkletools↵
$ sudo reboot↵
```

ラズベリー・パイ上のウェブ・ブラウザで「http://192.168.0.108:8888/tree」[注1]にアクセスして，パスワード「interface」を入力してログインします．

IF201808フォルダ配下に「2章.ipynb」が見えるのでクリックします．

■ 体験1…ハッシュ値の計算

ハッシュ値を理解するには実際に計算してみるのがよいと思います．ビットコインを利用するためのオープンソース・ソフトウ

注1：接続できないときは，http://localhost:8888/tree で試してください．または，ブラウザのブックマークのJupyter Notebookで接続します．

ェアにBitcoin Core（GitHub：bitcoin/bitcoin，https://github.com/bitcoin/bitcoin）があります．これを利用すれば実際に取引されている取引データやブロックを使って調べることができます．ビットコインではさまざまな情報からハッシュ値を算出して利用しているので，ここではハッシュ値の計算だけでなく特徴を見てみます．

IPythonやJupyter NotebookでSHA-256アルゴリズムを使ってハッシュ値を計算してみます．2章.ipynbのプログラムを実行するとハッシュ値の計算を体験できます．実際に手を動かしてプログラムを実行してみることでハッシュ関数の特徴を体感でき，Bitcoin Coreなどのツールを使うようになっても理解が深まると思います．

なお，Jupyter Notebookの使い方はAppendix2で紹介します．

● ライブラリの読み込み，関数宣言

In[1]は，ハッシュ値計算用ライブラリを読み込んでおり，In[2]でtext2hash関数を宣言しています．独自の関数を用意しておけば繰り返しハッシュ値を計算するのが簡単になります．

```
In [1]
import hashlib
In [2]
def text2hash(mytext):
    hash_object = hashlib.sha256(mytext.encode())
    print("HASH:    " + hash_object.hexdigest())
    print("LETTERS: " + str(len(hash_object.hexdigest())))
```

● ハッシュ値の出力

データ長にかかわらず64桁のハッシュ値が出力されることを確認してみます．In[3]～In[5]では，データとしてa，ab，abcをtext2hash関数に渡して，ハッシュ値を計算しています．

出力は「HASH:」で始まる行がハッシュ値，「LETTERS:」で始まる行がハッシュ値の文字列の長さです．実行結果はいずれも

64桁のハッシュ値が表示されました.

皆さんのPCで同じプログラムを実行すると同じハッシュ値が計算されますので試してみてください．関数に任意の長さのデータ(文字，数字，記号の組み合わせでもよい)を渡すだけです．ハッシュ関数アルゴリズムとデータが同じなら同一のハッシュ値が得られることが確認できるはずです.

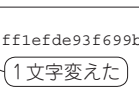
```
In [3]
text2hash("a")
Out [3]
HASH:     ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785
af
ee48bb
LETTERS: 64
In [4]
text2hash("ab")
Out [4]
HASH:     fb8e20fc2e4c3f248c60c39bd652f3c1347298bb977b8b4d590
3b85055
620603
LETTERS: 64
In [5]
text2hash("abc")
Out [5]
HASH:     ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61
f20015ad
LETTERS: 64
```

● ハッシュ値の出力2…1文字変えてみる

次にデータ量が多い場合に，1文字だけを変更してハッシュ関数で計算するとハッシュ値が本当に変化するか確認してみましょう．In[6]～In[7]を用意して実行してみます．In[6]はデータの最終文字が「c」ですがIn[7]では「0」に変えてあります.

実行するとデータがたった1文字違うだけで，算出されるハッシュ値が大きく異なることが分かります．このように一部のデータが変わっただけでもハッシュ値が変化し，データの改ざんを簡単に検出できます．ブロックチェーンにはハッシュ関数が信頼性の向上に大きく貢献しています.

```
In [6]
text2hash("abcabcabcabcabcabc")
Out [6]
HASH:     916f4626f2d02e07085873c17f8115790840519094e94114b706573c
9749331f
LETTERS: 64
In [7]
text2hash("abcabcabcabcabcab0")
Out [7]
HASH:     ed1a5a15e462296479d13c0ff1efde93f699bb2a288038b1f705ce4f
7df5327c
LETTERS: 64
```



▶やってみよう問題

任意の300文字以上の文字列(改行のない日本語の文章でも大丈夫)を使って，1文字だけが違う場合には本当にハッシュ値が異なるか，ハッシュ値の長さが一緒か確認してみましょう．

■ 体験2…取引履歴ツリーのルート・ハッシュ値の算出

マークル・ツリーのルート・ハッシュ値を求めます．マークル・ツリーのルート・ハッシュは，1ブロックに数百から数千の取引履歴を格納しても，どんな取引の中身でも効率的に検証することが可能となります．また，個々の取引履歴のハッシュ値をそのまま格納しようとすると膨大なデータ量になってしまいますが，マークル・ツリーのルート・ハッシュだけを格納すれば，容量を大幅に削減できます．

● 利用するPythonライブラリの主な機能

マークル・ツリーを作ってみます．Pythonのmerkletoolsライブラリ[注1]によって簡単にハッシュ木を作ることができます．ライブラリは

```
$ pip install merkletools
```

でインストールできます．主な機能を以下に示します．

注1：https://github.com/Tierion/pymerkletools

表1　merkletoolsライブラリのメソッド一覧

メソッド	説　明
add_leaf(value, do_hash)	ツリーにリーフまたはリーフのリストとして値を追加する．文字列を値として渡す場合にはdo_hashオプションにTrueを指定する
get_leaf_count()	ツリーに現在追加されている葉の数を返す
get_leaf(index)	指定されたインデックスにあるリーフの値を16進文字列として返す
reset_tree()	ツリーから全ての葉を削除し，新しいツリーの作成を開始する準備をする
make_tree()	追加された葉を使ってマークル・ツリーを生成する
is_ready	ツリーが構築されてルートとプルーフを供給する準備ができているかどうかを示すブール値のプロパティ
get_merkle_root()	ツリーのマークル・ルートを16進文字列として返す
get_proof(index)	指定されたインデックスでのリーフのハッシュ・オブジェクトの配列としての証拠を返す
validate_proof(proof, target_hash, merkle_root)	証明が有効でtarget_hash(ハッシュ値)とmerkle_root(マークル・ルート)が正しく接続されているかどうかを検証して結果をブール値(True / False)で返す

- マークル・ツリーの作成
- マークル証明の生成
- マークル証明の検証

ライブラリには**表1**のようなメソッドが用意されています．merkletoolsライブラリを使って以下の取引をハッシュ値にして，最終的にマークル・ルート(全取引履歴のハッシュ値)を算出してみます．

＊＊取引履歴一覧＊＊
Aさん → Bさん 1BTC
Cさん → Dさん 1.4BTC
Eさん → Fさん 0.8BTC
Gさん → Hさん　4.1BTC

● ツリーの作成

Jupyter Notebookで**In[8]**～**In[12]**を実行すると，マークル・ツリーが作成されます．**In[10]**ではリストを変数`leavies`に格納していますが，リストの文字列は取引データ(トランザクション・データ)に見立てています．これらの文字列からハッシュ値を作成し，**図1**のように全取引履歴のハッシュ値(マークル・ルートのハッシュ値)までを算出しています．

```
In [8]
import merkletools as mk
In [9]
mt = mk.MerkleTools(hash_type="md5")
In [10]
leavies = ["A → B 1BTC", "C → D 1.4BTC", "E → F 0.8BTC", "G → H
4.1BTC"]
In [11]
mt.add_leaf(leavies, True)
In [12]
mt.make_tree()
```

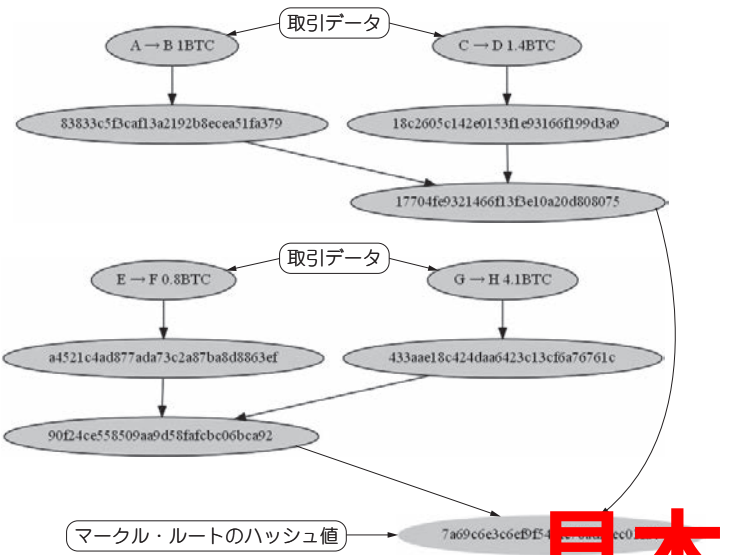


図1　全取引履歴のハッシュ値

● マークル・ツリーのグラフを作成する

次にPython3のGraphvizライブラリ[注2]で，マークル・ツリーのグラフを作成します．このライブラリは木構造を描画するのに特化しており，DOT言語というグラフ記述言語のスクリプトも用意されています．グラフ構造を描画するライブラリとしてはnetworkxの方が有名ですが，状態遷移図や木構造の描画はGraphvizの方が簡単に扱えます．また，今回は単純なグラフなのでGraphvizライブラリのDOT言語は使用しません．

Linux系OSでは，

```
$ pip install graphviz
```

でインストールできます．

Windowsは，ダウンロード・ページ[注3]からStable 2.38 Windows install packages[注4]からインストーラをダウンロードしてインストールできます．graphvizフォルダ配下にbinフォルダが作成されるのWindowsでは環境変数のPATHに「C:¥Program Files (x86)¥Graphviz2.38¥bin」[注5]を追加します．

次のプログラムを実行するとマークル・ツリーのグラフを出力します．グラフはpngファイル(D:¥merklegraph.png)として生成されます．graphvizライブラリを読み込み(In[13])，グラフ作成と出力ファイルのフォーマット指定(In[14])，追加するノードの形などのスタイルを指定(In[15])，木構造に設定する取引データの数を変数leaviesに格納(In[16])します．

In[17]で枝(エッジ)と葉(ノード群)を追加します．ノード追加はedgeメソッドで取引データ，そのハッシュ値，2つのハッシュ値から算出したハッシュ値，マークル・ツリーのルート・ハッシュ値の順番になるよう指定しました．マークル・ツリーの頂点

注2：https://www.graphviz.org/
注3：https://www.graphviz.org/download/
注4：数字は執筆時点のバージョン番号．
注5：インストール・ディレクトリを指定してください．

のノードはピンク色を指定して追加(In[18])，グラフをファイルに出力(In[19])し，出力されるとOut[19]のように出力先が表示されます．

```
In [13]
from graphviz import Digraph
In [14]
g = Digraph(format="png")
In [15]
g.attr("node", style="filled")
In [16]
leaves = mt.get_leaf_count()
In [17]
for i in range(0, leaves):
    leaf = mt.get_proof(i)
    key_0= list(leaf[0].keys())
    key_1= list(leaf[1].keys())
    g.edge(leavies[i], mt.get_leaf(i))
    g.edge(mt.get_leaf(i), leaf[1][key_1[0]])
    g.edge(leaf[1][key_1[0]], mt.get_merkle_root())
In [18]
g.node(mt.get_merkle_root(), color="pink")
In [19]
g.render("./merklegraph")
Out [19]
'./merklegraph.png'
```

● マークル・ルートのノードのハッシュ値を確認してみる

マークル・ツリーを使って特定の取引データが改ざんされていないかを確認することがあります．ここでは，1つ1つの取引データの，マークル・ツリー内での整合性を確認します．通常はマイニングで生成されたブロックの検証をブロックチェーンを持つノードがそれぞれ自動的に処理されていますが，手作業でノードからマークル・ルートまでの整合性を確認してその仕組みを体験してみます．この確認によってマークル・ルートの値のみ比較することでブロックに含まれる取引に改ざんがあることを検知できるか検証してみます.ハッシュ関数の演算にSHA265を利用するとハッシュ値が64ビット長になり確認の際に比較しづらいので，今回はハッシュ値が短いmd5を利用します．

作成したグラフ(**図1**)からマークル・ツリーの全体構造と確認

してみると，

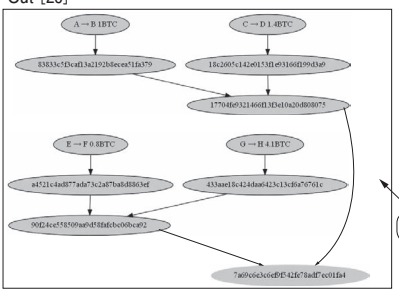
1段目：4つの取引データに見立てた文字列
2段目：ハッシュ関数で文字列から算出したハッシュ値
3段目：2つのハッシュ値から算出したハッシュ値
4段目：マークル・ルートのハッシュ値

が表示されています．マークル・ルートのハッシュ値を In [21] コマンドで確認して，グラフ（**図1**）と一致しているか確認します．同じハッシュ値ならば問題ありません．

In [20]

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
im = Image.open("./merklegraph.png")
im_list = np.asarray(im)
plt.figure(figsize=(100,80))
plt.imshow(im_list)
plt.show()
```

Out [20]



In [21]

```
print("root:", mt.get_merkle_root())
```

Out [21]

```
root: 7a69c6e3c6ef9f542fe78adf7ec01fa4
```

In [22]

```
import hashlib
```

In [23]

```
print(leavies[0])
```

Out [23]

```
A → B 1BTC
```

In [24]

```
a = hashlib.md5(leavies[0].encode())
```

In [25]

```
a.hexdigest()
```

ISBN978-4-7898-5028-5
C3055 ￥1200E
CQ出版社

9784789850285

定価：本体1,200円（税別）

1923055012004

ブロックチェーンは，参加する個々の端末が同じデータを共有します．このデータは時系列にたどることができ，また，データの提供者，データの確からしさを証明することもできます．
改ざんできない強固なデータ交換プラットフォームとして，通貨以外の用途でも注目を集めています．