卒業研究

メタチェーンを用いた
軽量ブロックチェーンアーキテクチャの開発

創生学部創生学修課程
伊勢田　氷琴

目次

1　緒言

1.1　はじめに

近年、インターネット上を始めとする信頼できない参加者が介在するネットワークにおいて、中央集権的な管理者を想定することなく、第三者との間で経済的価値を移転させることができる技術として、ブロックチェーン技術が注目されている。

ブロックチェーン技術は、ビットコインやイーサリアムをはじめとする暗号通貨の基幹技術であり、ハッシュ関数や電子署名、特徴的なデータ構造を駆使することによって、インターネット上におけるデータの真正性・完全性を保証することができる技術である。加えて、多くのブロックチェーンシステムが想定するネットワークは、システムへの各参加者（以下ノードと呼ぶ）が 1 対 1 の関係で接続される分散型ネットワークであるため、単一障害点を持たず、高い可用性を兼ね備えている。

このような特徴を持つブロックチェーン技術は、近年ではより広範な概念である「分散型台帳」を実現する技術の一つとして、金融分野を中心に様々な分野での応用が期待されている。代表的な応用例としては、ビットコインを始めとする暗号通貨（Satoshi Nakamoto 2008）、医療データの管理(Li et.al 2018)、IoT(Seyoung et.al 2017)、トレーサビリティの確保(Feng 2017)、電子投票(Khan et.al 2018)などがある。一方で、ブロックチェーン技術には様々なデメリットも存在する。その代表的なものがチェーンの伸長に伴うストレージの圧迫である。ブロックチェーンは、全ての取引記録を時系列に繋げていき、かつその全てのデータをネットワーク全員で保持することが前提とされている。従って、システムの利用者が増え、取引量が大幅に増加したり、システムが何十年と存続するに従って、全ての取引記録を保持するために大量のストレージと計算資源が必要となる。例えば、2009 年から運用が開始されているビットコインのブロックチェーンは、2021 年 12 月 3 日現在約 377.95GB に達している。これは 2020 年 12 月 3 日地点で 314.31GB であったため、1 年で 63.64GB 増えたことになる[1]。

このようなストレージの圧迫は様々な問題を引き起こす。主要な問題としては、第 1 にシステムの中央集権化、第 2 にスケーラビリティの低下、第 3 に応用可能性の低下、第 4 にセキュリティ性の低下が挙げられる。

前述の通り、巨大なストレージ容量とこれを処理できるだけの計算資源が必要となるため、参入障壁が高くなり、システムの中央集権化が起きる可能性がある。実際に、ビットコインは一日に約 27 万件の取引が実行されるが[2]、これを処理するフルノード（全ての取引記

---

[1] YCHARTS, Bitcoin Blockchain Size. https://ycharts.com/indicators/bitcoin_blockchain_size（2021/12/03 確認）

[2] Blockchain.com, Confirmed Transaction Per Day. https://www.blockchain.com/charts/n-transactions（2021/12/03 確認）

録を持つノード）は、2021 年 12 月 3 日現在 14885 ノードのみである[3]。このような状況が続けば、一部のノードによってシステム全体が支配されることとなり、分散型ネットワークの利点が生かされなくなる。また、このようなシステムに新しく参加するノードは、大量の計算資源を用意する必要があるため、計算資源を多く持たないノードは参入できず、処理できるトランザクションの量に制限がかかるなど、スケーラビリティも低下する。スケーラビリティの低下は、ブロックチェーン技術の応用可能性を著しく下げると考えられる。また、悪意のあるノードが高い計算能力を持っている場合には、これらに加えて相対的に悪意のあるノードの影響力が増すことで、データの改竄などが起きやすくなるため、セキュリティ上の懸念も生じる。

以上の理由から、長期間に渡って大量のトランザクションを処理してもストレージを圧迫しないような、軽量なブロックチェーンアーキテクチャの開発が必要である。

1.2　研究背景

ブロックチェーンのストレージ容量の増加に対し、これまで取られてきた解決策を整理する。

まず、ブロックチェーンの保存を一部の豊富な計算資源を持つノードに任せる方法がある（Nakamoto 2009）。これはノードをフルノードと SPV（Simplified Payment Verification）ノードに分ける方法である。前者は全取引を保持する一方で、後者はブロックハッシュのみを保持する軽量なノードであり、約 1,000 分の 1 程度の容量で運用できる（アントノプロス 2018）。SVP ノードは、フルノードに対してブロックの内容を問い合わせることで UTXO など必要な情報を集め、トランザクションの発行等を行うことができる。

また、近年では sharding という技術がブロックチェーンの文脈で積極的に利用されている。シャーディングとは、本来トランザクションをシャード（破片）に分け、切り分けられた複数のグループで処理し、結果を同期することでスループットを向上させる技術である（鳩田、生永　2020）。Bin et al(2020)はこれを応用し、既存の暗号通貨のネットワークの上にノードを距離に応じてグルーピングし、そのグループ間でオーバレイネットワークを形成しストレージを分散させる手法を提案している。また、Zamani et al(2020)は、committee と呼ばれる小グループにトランザクション処理やブロックチェーンの管理を分散させる手法を提案している。また、チェーン自体を複数に切り分けて、保存を分散させる手法も存在する。Yibin & Yangyu（2020）は、ブロックチェーンを複数のセグメントに切り分け、各ノードにそのセグメントの 1 片の保存を任せる方法を提案している。

---

[3] BITNODES, Global Bitcoin Nodes Distribution. https://bitnodes.io/（2021/12/03 確認）

1.3　研究目的

以上の研究は、確かにノードが負担するブロックチェーンのストレージ負担を一部軽減するものである。しかし、アーキテクチャを変更したために、非中央集権性、セキュリティ性のいずれかを犠牲にしている側面がある。

まず、ビットコインで導入されている手法は、非中央集権性を著しく損なう。何故なら、一部のノードによるデータの独占を生じさせるためである。また、Bin Qu et al(2020)やM.Zamani et al(2020)の手法は、ブロックを保存する特定のグループが攻撃された場合、データの完全性が失われるという意味で、セキュリティ性に問題がある。Yibin＆Yangyu（2020）の手法も同様に、悪意のあるノードが身分を偽って全チェーンを取得し、その情報を削除した場合、チェーンの完全性が失われるという問題がある。また、同手法は各ノードがセグメントの１片を保持するのみであり、もしノードが攻撃された場合、当該セグメントが恒久的に失われ、データの完全性が損なわれる懸念がある。

以上より、既存手法の問題点は次の２点に集約できる。即ち、第１にシステムの非中央集権を確保できないこと、第２にデータの完全性が失われることである。従って、本研究ではビットコインのような比較的シンプルなブロックチェーンを想定し、各ノードが負担するストレージの容量を減らしつつ、非中央集権性・セキュリティ性も同時に保証できるブロックチェーンアーキテクチャ（以下本システムと呼ぶ）を提案する。本論文では、開発したブロックチェーンアーキテクチャを詳細に解説すると共に、本システムの振る舞いを理論的計算によって解析し、ノード数とストレージ削減効果の関係性を明らかにする。また、プログラミング言語「Python」で実装し、小型コンピュータの「ラズベリーパイ」4 台で構成された小規模ネットワークで原理実証実験を行い、その結果を報告する。

2　提案手法

2.1　システムの概要

本システムの概要は図１に示す通りである。図１から分かる通り、本システムでは従来のブロックチェーシステムに次の２点の改良を加える。

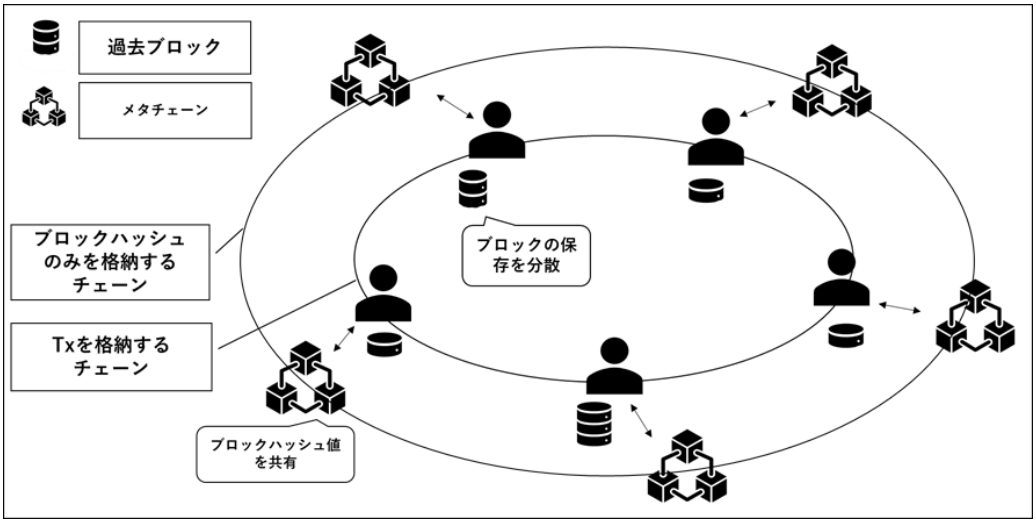

図 1　システム概要

第１に、ブロック保存の分散である。全ブロックを全員で保持するのではなく、各ブロック生成のタイミングで保存するノードと保存しないノードに振り分けるように変更する。ただし、単に振り分けただけでは非中央集権が失われるため、ブロックを保持するノードはランダムに選ばれることとする。従って、例えば従来のブロックチェーンであれば、図２に示すように、全員が同じブロックを全て保持していたが、本システムでは図３に示すように、ノードによって保持しているブロックと保持しないブロックが存在し得る状況が生まれる。

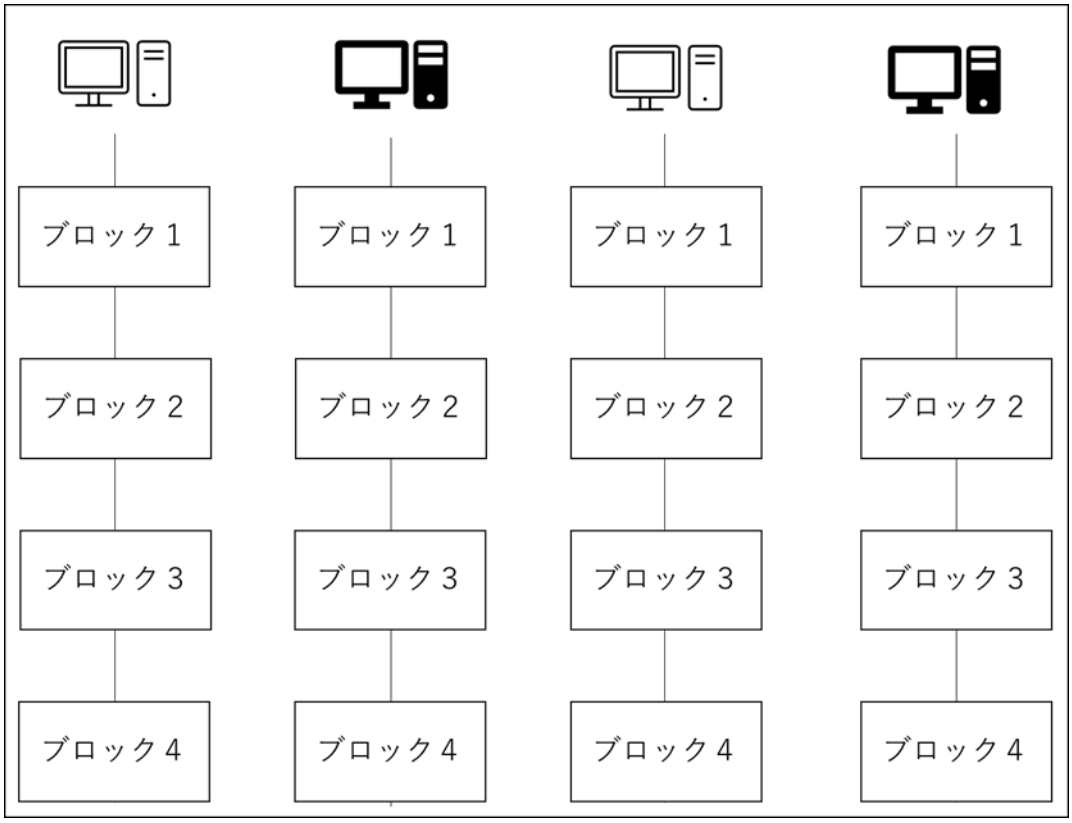

図 2　従来のブロックチェーン

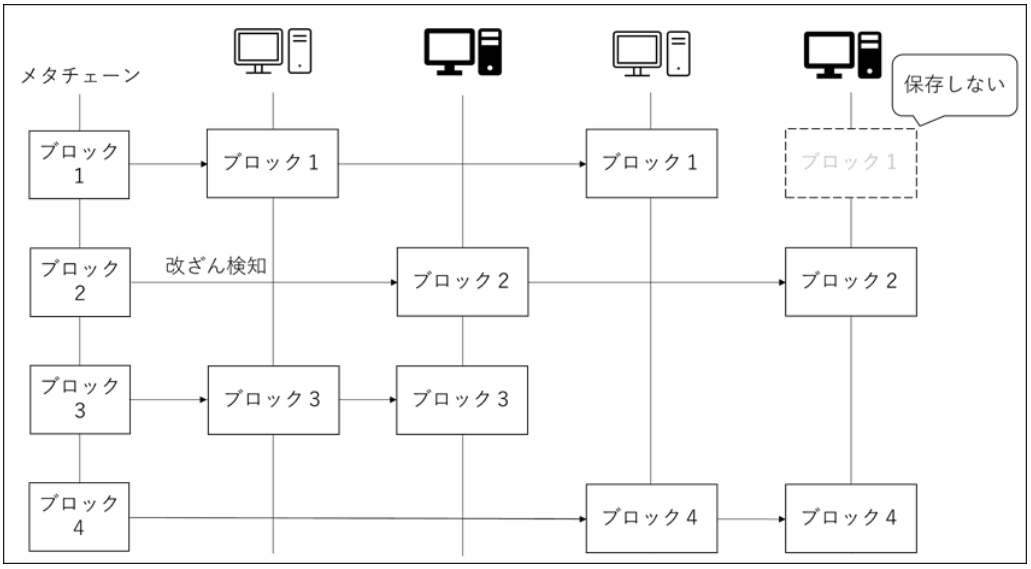

図 3　本システムにおけるブロックチェーン

第２に、メタチェーンの導入である。本システムでは悪意のあるノードが存在することを想定しているため、単にブロックの保存を分散させただけでは、悪意のあるノードによる改ざんを検知できない。そこで、「メタチェーン」と定義する、ブロックハッシュのみを格納

したブロックチェーンを、通常のチェーンと並行して全員で保持するようにする。メタチェーンは非常に軽量であるため、メタチェーンを同時に保持するとしても、ストレージの削減効果は期待できる。

2.2　ストレージ削減効果

ネットワークに存在するノードの数を$N_{node}(\geq 1)$とし、ブロック保存するノード数を$k(1 \leq k \leq N_{node})$、1 ブロック当たりのサイズを$B_{block}$バイト、1 ブロック当たりのメタチェーンのブロックのサイズを、B_{meta}バイト、累積ブロック数をN_{block}とする。また、ノードがネットワークから離脱したり、新たにに加入することは無いと仮定する。従ってノードの数は時間に対し一定である。加えて、トランザクションプールには、常にブロックサイズを十分に上回るトランザクションがプールされていると仮定し、ブロックを割り当てられたノードは必ずブロックを保存するものとする。

このとき、必要なストレージ容量は、従来のブロックチェーンであれば$B_{block}N_{block}$バイトとなる。しかし、本システムの場合はあるノードがブロックを保存する確率は

$$\frac{k}{N_{node}}$$

となるため、ある時間における保存するブロックのサイズの期待値は、

$$\frac{k}{N_{node}}B_{block}N_{block} + B_{meta}N_{block}$$
$$= B_{block}N_{block}\left(\frac{k}{N_{node}} + \frac{B_{meta}}{B_{block}}\right)$$

となる。ここで、本システムではB_{block}とB_{meta}の値について、

$$\frac{B_{meta}}{B_{block}} \ll 1$$

という関係が成り立っている。即ちB_{block}と比較してB_{meta}は十分小さいため、N_{Block}が非常に大きいとき、$B_{meta}N_{Block}$が$B_{block}N_{block}\left(\frac{k}{N_{node}} + \frac{B_{meta}}{B_{block}}\right)$に占める割合は小さくなる。従って、近似的に

$$B_{block}N_{block}\left(\frac{k}{N_{node}}+\frac{B_{meta}}{B_{block}}\right) \quad \cong \quad \frac{k}{N_{node}}B_{block}N_{Block}$$

が成り立つ。以上より、従来のブロックチェーンと比較して、本システムでは必要なストレージ容量が近似的に$\frac{k}{N_{node}}$分削減されると言える。尚、$k = N_{node}$のとき、メタチェーンに関する処理を無視すれば、本システムは従来のブロックチェーンと等価な動きをする。

2.3　アルゴリズム

本システムでは、第１章で述べたトランザクション処理の過程に、乱数によるブロック保存ノードの分散化とメタチェーンへのハッシュ値の格納が加わる。具体的なアルゴリズムは以下の通り。

尚、mining()はマイニングを実行しつつ、自身が一番早くブロックを生成できればTrueを、他のノードが先にブロックを生成すればFalseを返す関数である。random(a,b)はa以上b以下の乱数を返す関数である。store_block_in_blockchain()はメインのブロックチェーンにブロックを格納する関数であり、store_block_in_metachain()はメタチェーンにブロックを格納する関数である。変数blockには、mining()実行時にグローバル変数として生成されたブロックが格納されている。

アルゴリズム 1

```
Main(n,k)
1 while True:
2     if mining() == True:
3         if random(1,n) >= k :
4             store_block_in_blockchain(block)
5             store_block_in_metachain(block)
6         elseif random(1,n) < k:
7             store_block_in_metachain(block)
8     elseif mining() == False:
9         if random(1,n) >= k :
10            store_block_in_blockchain(block)
11            store_block_in_metachain(block)
12        elseif random(1,n) < k:
13            store_block_in_metachain(block)
```

2.4　セキュリティ

2.4.1 51%攻撃

本システムではPoWの使用を想定しており、ブロックの保存を分散させる以外の仕組みはビットコインに準じている。従って、51%の脅威はビットコインと同様に存在する。Satoshi Nakamoto（2008）によれば、誠実なノードが次のブロックを見つける確率をp、攻撃者が次のブロックを見つける確率をqとして、攻撃者がz個前のブロックを生成しているとすると、攻撃者が遅れから追いつく確率q_zは、

$$q_z = \begin{cases} 1 & if\ p \leq q \\ \left(\frac{q}{p}\right)^z & if\ p > q \end{cases}$$

と与えられる。

2.4.2 ビザンティン障害体制

ビザンティン障害耐性とは、「なりすましや悪意のあるノード、ネットワーク上障害などが発生しても全体として正しい情報に基づき動作する性質」と定義されている（鳩田, 生永 2020）。本システムは、基本的な土台をビットコインにおけるブロックチェーンとしているため、PoWに基づいて合意を取ることができる。

2.4.3 データの完全性

本システムでは、あるブロックを保持するのは一部のノードのみである。従って、ネットワークに存在する悪意のあるノードの数によって、51%攻撃の他にもブロックを改ざんできる可能性がある。この場合、本システムではメタチェーンを使って改ざんを検知でき、悪意を持ったノードがk人より一人でも少なければ、改ざんを訂正することができる。

仮に、悪意を持ったノードが$l(\leq N_{node})$人おり、彼ら全員がブロックを保存するノードとして選ばれ、結託して自らに有利なようにデータを書き換える状況、即ち$l = k$が成立した場合、情報の完全性が失われることになる。また、これは悪意のあるl人のノードが、乱数の結果に関わらずブロックを保存し続け、これを訂正する誠実なノードの数kを上回る状況が発生する確率と同等である。以上より、このシステムがデータの完全性を保証できる条件は、ノード数が十分に大きければ、

$$l < k$$

となる。

3　シミュレーション

本章では、システムの有効性を実証するため、シミュレーションによって従来のブロックチェーンと比較して削減可能なストレージ容量の時間的推移および、攻撃等によってデータの完全性が失われる確率を定量的に評価する。

3.1　理論値の算出

まず、ネットワークに存在するノードの数を$N_{node}(\geq 1)$とし、ブロック保存するノード数を$k(1 \leq k \leq n)$、1 ブロック当たりのサイズをB_{block}バイト、1 ブロック当たりのメタチェーンのブロックのサイズを、B_{meta}バイト、累積ブロック数をN_{Block}とする。また、本シミュレーションでは、ノードがネットワークから離脱したり、新たにに加入することは無いと仮定する。従ってノードの数は時間に対し一定である。加えて、トランザクションプールには常にブロックサイズを十分に上回るトランザクションがプールされていると仮定し、ブロックを割り当てられたノードは必ずブロックを保存するものとする。

前章までの議論と同様に、必要なストレージ容量は、従来のブロックチェーンであれば

$$B_{block}N_{Block}\text{バイト}$$

となる。一方、本システムの場合は、保存するブロックのサイズの期待値は累積で

$$= B_{block}N_{block}\left(\frac{k}{N_{node}} + \frac{B_{meta}}{B_{block}}\right)\text{バイト}$$

となる。

ここではシミュレーションの単純化のために、誠実なノードのみでネットワークが構成されていると仮定する。

3.2　シミュレーション結果

本シミュレーションはストレージの削減効果がノード数N_{node}や、ブロックを保存するノード数kによってどのように変化するのか評価することを目的としている。従って、これら以外の変数は全て所与のものとして扱う。パラメータの設定は表 1 の通り。

表 1　パラメータの設定

パラメータ	設定値
B_{block}	10kB
B_{meta}	150B
N_{node}	10000

表 1 のパラメータを設定し、ブロックを保存するノードkの値を、N_{node}に対して 10%から 90%まで変化させ、また各kについて累積ブロック数N_{block}を最大で 10000 ブロックまで動かしたときの削減ストレージ容量を示したグラフを図に示す。

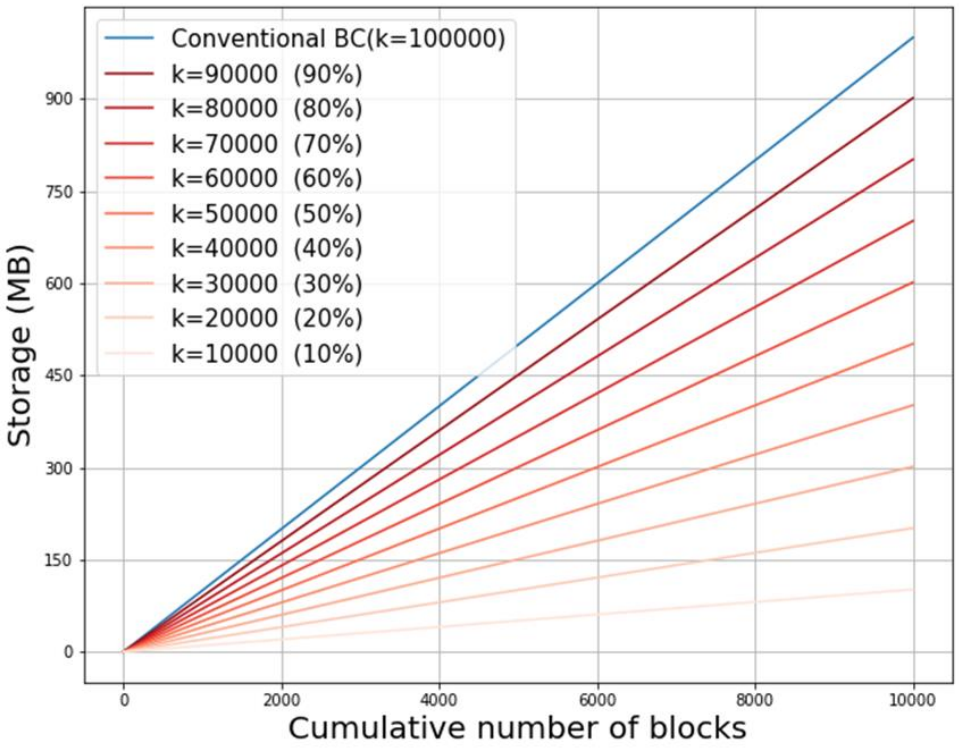

図 4　必要ストレージ容量の変化

図から分かる通り、k の値が小さくなるほど大きくストレージを削減できている。10000 目のブロックが作成されたときの必要ストレージ容量と、従来のブロックチェーンと比較した削減率は表 2 の通りである。

表 2　1000 ブロック地点での必要ストレージ容量と削減率

k	必要ストレージ（MB）	削減率（%）
100000	1000MB	0%
90000	901.5MB	9.85%
80000	801.5MB	19.85%
70000	701.5MB	29.85%
60000	601.5MB	39.85%
50000	501.5MB	49.85%
40000	401.5MB	59.85%
30000	301.5MB	69.85%
20000	201.5MB	79.85%
10000	101.5MB	89.85%

表から、削減率は最大で 89.85%となる。また、ブロックを保存するノードが 1 割であれば、10000 ブロック作成地点でも 100MB 程度の使用に抑えることができる。

4　原理実証実験

第 3 章でのシミュレーションに加えて、本システムの有効性を検証するため、従来のブロックチェーンと本システムをそれぞれプログラミング言語「Python」で実装し、小型コンピュータの「ラズベリーパイ」4 台を用いて構築したネットワーク上で実証実験を行なった。本章では実証実験を行なった条件と結果について報告する。

4.1　実験機材

本実証実験で使用したコンピュータは「Raspberry Pi 3 model B+」2 台と「Raspberry Pi 3 model B V1.2」2 台、OS は「Raspberry Pi OS(32-bit)」である。それぞれ仕様は表 4 の通り。

表 3 使用コンピュータの仕様

	Raspberry Pi 3 Model B+	Raspberry Pi 4 Model B
CPU	Broadcom BCM2837B0, Cortex-A53（ARMv8）64-bit SoC	Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
CPU クロック	1.4GHz	1.2GHz
メモリ	1GB LPDDR2 SDRAM	1GB RAM[4]

4.2　実験系

ラズベリーパイ 4 台を 5GHz の無線 LAN に接続し、TCP/IP を用いて P2P のオーバレイネットワークを構築した。P2P ネットワーク上では生成されたブロック及びトランザクション、一定時間おきの生存確認パケットのみが共有される。ネットワークトポロジは図 5 に示す通りである。コンピュータの負担を軽減するために、トランザクションは追加で用意したもう一台のコンピュータから各コンピュータにブロードキャストする形で、外生的に与えられるものとした。また、トランザクションは本来送金データであるが、実装を簡略化するためにダミーデータを入れたダミートランザクションとしている。ダミーデータは表 5 に、ダミートランザクションの例は表 6 に示している。尚、「time」はトランザクションが発行された日付と時刻、「publisher」はトランザクションを発行したノードの公開鍵、「data」はダミーデータ、「TxID」は以上の文字列にハッシュ関数を適用したハッシュ値、「sig」はこのハッシュ値に対し、「publisher」に対応する秘密鍵で付した電子署名である。ハッシュ関数や電子署名等に関する技術的説明は付録 1 を参照のこと。また、ダミーデータのサイズは 10 バイト、ダミートランザクションのサイズは 458 バイトとなっている。

表 4　ダミーデータ

aaaaaaaaaa

[4] RaspberryPi, Raspberry Pi 3 Model B+.
https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/（2021/12/14 確認）
及び RaspberryPi, Raspberry Pi 3 Model B.
https://www.raspberrypi.com/products/raspberry-pi-3-model-b/（2021/12/14 確認）
より作成

表 5　ダミートランザクション


```
  {
    "TxID": "cadf946bd324b2a250743a863768de4a02bc3615809d02f4eceaeffa6219c2ce",
    "time": "2021-11-29 17:02:39.912743",
    "publisher": "faac7aa1381c22a780e15c1e62930c904827a3585d91aeda9fb05080ed7643d379c5c498e4c6732eec7
9403887c7f7c32c1680fcc389dce83649f172af78fb2c",
    "data": "aaaaaaaaaa",
    "sig": "b88939540ac458ec0e0b65ff9ccc46fa1b13f0a6c90edd7c5d80647b93309309c0a8ab9ebd56eb553b9019b69
f1183440165f158855e6a17e8c83baab22a3c08"
  }
```


また、ブロックチェーンの中身の例は表 7 に、メタチェーンの中身の例は表 8 に示す通りである。尚、「nonce」は PoW の結果として見つけた値、「timestamp」はブロックが生成された日付と時刻、「previous_hash」は前のブロックのハッシュ値、「height」は繋がっているブロックの数、「tx_hash」は格納されるトランザクション全体のハッシュ値、「hash」は以上のデータ全てのハッシュ値である。尚、PoW やノンスに関する詳しい説明は、付録 1 を参照のこと。

表 6　ブロックチェーン


```
[
  {
    "hash": "00004115bc7d51e6c6d0d65e30ab1e1649b38da0ade623fe9e8157bb65ed8588",
    "timestamp": "2021-12-14 10:49:48.042670",
    "nonce": 14669,
    "previous_hash": "",
    "tx_hash": "125a154b01587d99ea0d34d5916c5d34ce27f01ca522bd526a3ba599c5f2a084",
    "height": 0,
    "size": 5,
    "tx": [
      {
        "TxID": "1a8f88409a7d34040216944d1e2f58b7902c83c0017f65f04f31b6d89102d893",
        "time": "2021-12-14 10:49:30.555416",
        "publisher": "73f97070e4ea475c5b79104ea03eb584fac35deb295104e6c5d9d84909b83a4b7ae7923da0b9b72
3336009976ac04735b36ceef131513dad6964842a54cfa33a",
        "data": "aaaaaaaaaa¥n",
        "sig": "527a4c43cb44ed6836fc9372e1502e089fef9c774733e4f1946c5a53e5f92116dac150ef7d411a0a1499fd
b2ffe9021eed57dc14adf421c1ef9a4e44b915300d"
```

```
        },
        {
          "TxID": "8654b9f9bffba9971667054e5aa1e242bd16a016f2798efeee6604223146f42a",
          "time": "2021-12-14  10:49:33.598429",
          "publisher": "73f97070e4ea475c5b79104ea03eb584fac35deb295104e6c5d9d84909b83a4b7ae7923da0b9b72
3336009976ac04735b36ceef131513dad6964842a54cfa33a",
          "data": "aaaaaaaaaa¥n",
          "sig": "2ac8584e52c8f3d96cbb3d57d8f600fb6a5f91f8ad04dd169041f0aeb691cc864f514b00527ddcfb211bd1
175b3d7b1b859723342b316fed817b6e131be572aa"
        },
        .....
```


表 7　メタチェーン


```
[
  {
    "hash": "00002783f0ff8a7e1983f0c20e1af764ce20068ffe0529178601e98ef8cba187",
    "timestamp": "2021-12-17  18:30:24.293309"
  },
  {
    "hash": "0000a3f9eca73bb13d6804c0430bcd5e92e6bc0a3b99fb02a173e4f80f133acc",
    "timestamp": "2021-12-17  18:30:47.259501"
  },
  {
    "hash": "0000df43021f6b255b4e07d6060693b8c3d2291dcd93b66a46fe4eb069b6102d",
    "timestamp": "2021-12-17  18:31:07.272298"
  },
  {
    "hash": "0000046a624962823604ca4a82ea409b41bdceca0311579460f87437b04b8162",
    "timestamp": "2021-12-17  18:31:50.876592"
  },     .....
```

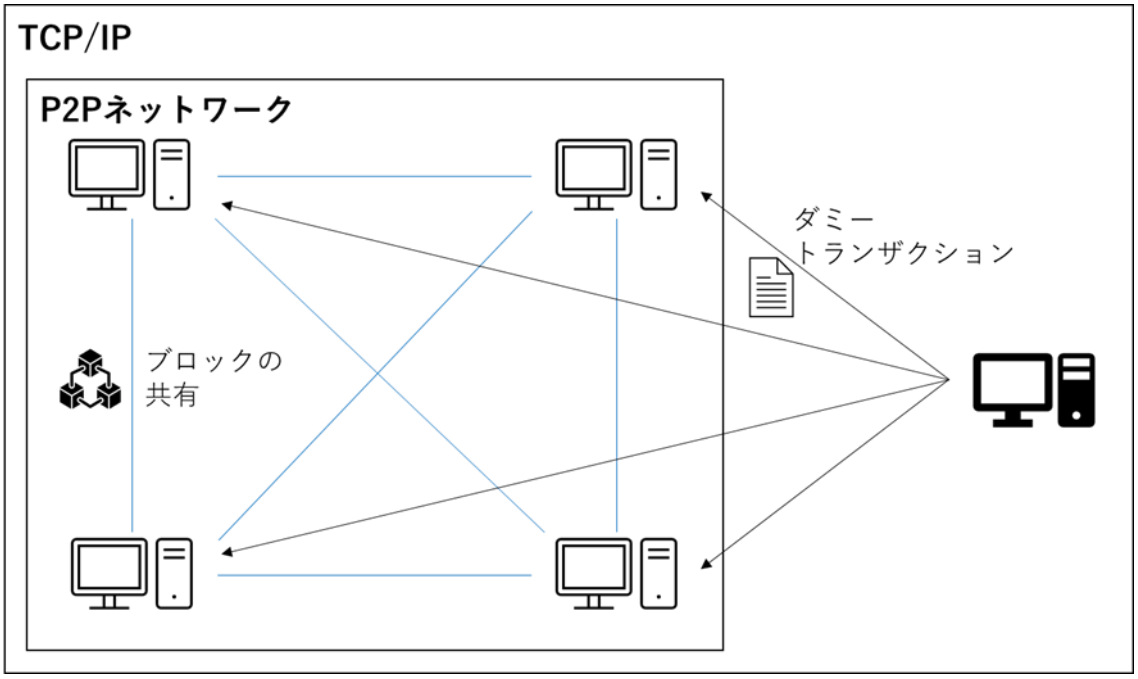

図 5　ネットワークトポロジ

本実証実験ではトランザクションは 0 秒から 3 秒に 1 件発行され、また 1 ブロックに格納されるトランザクションの数は 20 個としている。

実証実験では、ブロックが 25 個繋がるまでを 1 回の実験として、ブロックを保存するノード数であるkを 4（従来のブロックチェーンと等価）から 1 まで値を変えながら、ネットワークや乱数等の諸条件の誤差を小さくするために各 5 回試行を行なった。

4.3　実験結果

実験結果を表 9 に示す。表から、kの値を小さくしていくほど大きくストレージを削減できることが分かる。$k = 4$のとき、本システムは従来のブロックチェーンと同じストレージ容量を必要とする。$k = 4$のときと比較して、$k = 3$、即ち全 4 ノード中 3 ノードがブロックを保存するときには 8.35%、$k = 2$のときは 63.28%、$k = 1$のときは 77.39%必要なストレージ容量を削減可能である。

表 8　実証実験の結果[5]

k	ノード1	ノード2	ノード3	ノード4	平均	平均削減率
4	165.67KB	165.67KB	165.67KB	165.67KB	165.67KB	0%
3	151.08KB	164.28KB	139.94KB	152.06KB	151.84KB	8.35%
2	55.37KB	57.79KB	72.63KB	57.77KB	60.89KB	63.28%
1	40.04KB	41.78KB	31.89KB	36.08KB	37.45KB	77.39%

5　考察

シミュレーションの結果から、本システムは非中央集権性とデータの完全性を確率的に保証しつつ、必要ストレージ容量の削減が可能であると言える。ブロックの保存は分散されるため、特定のノードにデータが集中することは無く、大規模なネットワークにおいては、kの値を変えることで、システムのセキュリティ性とストレージ削減量を柔軟に調整することができる。

一方、本研究では解決できなかった課題が大きく3つある。

第1に、小規模なネットワークでは適用が難しいことである。実証実験の結果から、必要ストレージ容量の削減は確かに可能であるものの、実験に用いた小規模なネットワークでは、システムの振る舞いが不安定になることが分かった。本システムでは、ブロックを保存するノードの割当を、各ノードが発生させた乱数に従って決めるというナイーブな実装となっているため、4ノード程度では、ノード全員がkより小さい乱数を出力してしまう確率が無視できない水準で存在する。実際に、全てのノードがブロックを保存しない確率を考えると、全てのノードが独立に生成した乱数がkを下回る確率は

$$\frac{k}{N_{node}}$$

と与えられるため、逆に上回る確率は

$$1-\frac{k}{N_{node}}$$

となる。これが全ノードについて発生するから、求める確率は

$$\left(1-\frac{k}{N_{node}}\right)\left(1-\frac{k}{N_{node}}\right)\left(1-\frac{k}{N_{node}}\right)\cdots\left(1-\frac{k}{N_{node}}\right)=\left(1-\frac{k}{N_{node}}\right)^{N_{node}}$$

[5] 小数第3位を四捨五入している

となる。

仮に、$N_{node} = 4$、$k = 2$とすると、この値は 0.0625 となり、無視できる水準とは言えない。しかし、例えば$N_{node} = 100$、$k = 10$とすれば 0.000027 と、0.1%以下の確率となり、ほぼ無視しても良い水準まで下がる。以上のことから、本システムは大規模なネットワークでは十分に機能するものの、小規模なネットワークでは有効ではないと言える。解決方法としては、確実にブロックを保存できるよう、ブロックの保存に関して事前にノード間でコンセンサスを取るようにランダム化アルゴリズムを変更するといったことが考えられるが、ノードの振る舞いが予測できず、信頼もできないという前提に立つならば、根本的な解決策にはなり得ないと考えられる。

第 2 に、インセンティブ・システムの欠落である。本論文では十分に議論することができなかったが、インセンティブ・システムも考慮する必要があるだろう。本システムは、各ノードが発生させた乱数の結果に基づいてブロックの保存を割り振っている。ブロックの保存は強制されるものではなく、保存しなかった場合のペナルティも考慮されていない。従って、各ノードが可能な限りストレージを温存しながら、システムには参加したいと考えたとき、ブロックの保存を無視するインセンティブが発生することになる。解決策としては、例えば本論文と近いアプローチを行なっている Yibin＆Yangyu（2020）は、ブロックの保存を証明することで、マイニングへの参加権を付与する方式を提案している。その他にも、ブロックの参照時に僅かな手数料を付与するといった方法が考えられる。このようなインセンティブを適切に付与することが今後の課題と言える。

第 3 に、帯域幅の問題である。暗号通貨システムであれば、全てのブロックが揃わなければ理論的に自分の残高を確認することはできない。従って、ユーザは取引の都度、全ノードにブロックを要求するメッセージをブロードキャストし、さらに全ブロックを受信する必要がある。このような大容量のメッセージが多数ネットワークを行き来すると、帯域幅を圧迫する可能性がある。対処法としては、全てのブロックをキャッシュするノードを配置したり、ブロックを保存する各ノードから、要求の UTXO のみを計算して渡す仕組みを実装するといった方法が考えられる。

本システムは、目的であるストレージの削減を達成することは可能だが、小規模ネットワークでの運用方法、インセンティブ・システムの構築、帯域幅の制御などは今後の課題と言える。

6　結言

暗号通貨「ビットコイン」により、世界で初めて実装されたブロックチェーン技術は、今後金融を始めとして、IoT や医療データの管理、電子投票やトレーサビリティの確保など、データの完全性が求められ、かつ中央集権的な管理が馴染まない多くの領域で応用されていくだろう。しかし、ブロックチェーン技術によって保証されるデータの真正性や完全性は、システムを維持する高負荷な計算や、各ノードが負担する莫大なストレージ容量に依存し

ている。ブロックチェーンは時間に対して単調に伸びていくため、パーソナル・コンピュータや IoT デバイスなど、計算資源が限られたノードはシステムの維持に貢献できず、中央集権化が発生したり、システムの持続可能性が失われることとなる。

本論文では、全ノードが全てのブロックを保持する従来のブロックチェーンアーキテクチャに対し、ブロックの保存をランダム化することで、非中央集権性とセキュリティ性を犠牲にすることなく、ストレージ容量を削減可能な軽量ブロックチェーンアーキテクチャを提案した。また、シミュレーションと実証実験のそれぞれの結果から、既存のブロックチェーンと比較してストレージ容量を大幅に削減できることを示した。一方、本論文はストレージ容量の削減を主眼に置いているため、インセンティブ・システムや小規模ネットワークへの適用、帯域幅の圧迫等に対する対処は今後の課題と言える。

7　謝辞

最後に、本研究を進めるに当たり、技術的側面から卒業論文の構成に至るまで、適切な指導を賜った指導教官の熊野英和教授に感謝いたします。

8　参考文献

・Bin Qu, Li-E Wangand, Peng Liu, Zhenkui and Shi, Xianxian Li. 2020. "GCBlock: A Grouping and Coding Based Storage Scheme for Blockchain System," in IEEE Access, vol. 8, pp. 48325-48336.

・Feng Tian. 2017 "A supply chain traceability system for food safety based on HACCP, blockchain & Internet of things," 2017 International Conference on Service Systems and Service Management, pp. 1-6.

・Khan Kashif Mehboob, Junaid Arshad, Muhammad Mubashir Khan. 2018 "Secure Digital Voting System Based on Blockchain Technology," International Journal of Electronic Government Research（IJEGR）14, no.1: 53-62.

・Nakamoto, Satoshi. 2008.“Bitcoin: A Peer-to-Peer Electronic Cash System”. https://bitcoin.org/bitcoin.pdf.

・Li Hongyu, Zhu Liehuang, Shen Meng, Gao Feng, Tao Xiaoling, Liu Sheng. 2018.“Blockchain-Based Data Preservation System for Medical Data”Med Syst. 2018, 42, 141.

・Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018.“RapidChain: Scaling Blockchain via Full Sharding.”In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security（CCS ’18）. Association for Computing Machinery, New York, NY, USA, 931–948.

・P. Hoffman, VPN Consortium, B. Schneier. 2005.“Attacks on Cryptographic Hashes in Internet Protocols”, https://www.ipa.go.jp/security/rfc/RFC4270EN.html（2021 年 12 月 16 日確認）

・Seyoung Huh, Sangrae Cho, Soohyung Kim. 2017 "Managing IoT devices using blockchain platform" 19th International Conference on Advanced Communication Technology（ICACT), pp. 464-467.
・Xu, Yibin & Huang, Yangyu,. 2020.“Segment Blockchain: A Size Reduced Storage Mechanism for Blockchain.”IEEE Access. 1109. 10. pp.1-1.
・アンドレアス・M・アントノプロス（著），今井崇也, 鳩貝淳一郎（訳）(2018)「ビットコインとブロックチェーン　暗号通貨を支える技術」NTT 出版株式会社
・赤羽喜治, 愛敬真生（2017)「ブロックチェーン　仕組みと理論」株式会社リックテレコム
・株式会社 FLOC（2019)「Python で動かして学ぶ！新しいブロックチェーンの教科書」翔泳社
・滝沢誠, 榎戸智也（2014)「分散システム：P2P モデル」コロナ社
・鳩田康史, 生永雄輔（2020)「分散型台帳テクノロジー　ブロックチェーン/DLT の基礎知識と Corda 入門」朝倉書店
・松浦健一郎, 司ゆき（2018)「入門　仮想通貨の作り方　プログラミングで学ぶブロックチェーン技術・ハッシュ・P2P のしくみ」秀和システム
・一般社団法人日本ブロックチェーン協会,「ブロックチェーンの定義」を公開しました. https://jba-web.jp/news/642（2021 年 12 月 12 日確認）
・理化学研究所, 富士通株式会社（2021)「スーパーコンピュータ「富岳」TOP500、HPCG、HPL-AI において 3 期連続の世界第 1 位を獲得」, https://www.riken.jp/pr/news/2021/20210628_2/（2021 年 12 月 16 日確認）
・結城浩（2017)「暗号技術入門　第 3 版」SB クリエイティブ株式会社

付録１　ブロックチェーン技術の詳細

付録１では、本文では触れなかったブロックチェーン技術の詳細について説明する。まず、ブロックチェーン技術とはどのような技術なのか、概要を整理する。その上で、ブロックチェーン技術が開発され、現在様々な応用が模索されている背景について説明し、以降はブロックチェーン技術の技術的詳細について説明する。

ブロックチェーン技術は、それ自体が全く新しい設計思想に基づいたデータ構造だが、その中で使用されている要素技術は古くから存在するものが大半である。その中でも特にブロックチェーン技術の理解において重要となる「一方向ハッシュ関数」、「電子署名」、「P2P」について、それぞれ解説する。その後、トランザクションの構造、およびブロックチェーンの構造について解説する。最後に、自律的なノードの集合であるブロックチェーンシステムが機能する理由であるコンセンサス・アルゴリズムについて説明する。

A.1　ブロックチェーン技術の概要

本節では、技術的内容に踏み込む前に、ブロックチェーン技術全体の概要を説明する。尚、本文で使用される「ブロックチェーン」はビットコインにおけるブロックチェーンを想定している。

一般社団法人日本ブロックチェーン協会によると、「ブロックチェーン」は以下のように定義されている。

電子署名とハッシュポインタを使用し改竄検出が容易なデータ構造を持ち、且つ、当該データをネットワーク上に分散する多数のノードに保持させることで、高可用性及びデータ同一性等を実現する技術を広義のブロックチェーンと呼ぶ（日本ブロックチェーン協会　2016）。

即ち、ブロックチェーンとはデータ構造の一種である。データ構造は、一般的に検索を高速化したり、特定のアルゴリズムに従った演算を高速化するといった目的を持つ。ブロックチェーンの場合は「改ざんを検知できること」、「改ざんを防ぐこと」に特化したデータを構造であると言える。

ブロックチェーンでは、改ざんを検知するために、各データを時系列に整理する。ここでいう「データ」とは、ビットコインであれば「いつ」、「誰が」、「誰に」、「いくら送金したのか」を示す「取引データ」（以後トランザクションと呼ぶ）である。各トランザクションは、電子署名を付与することで内容の正当性を確保する。

各データを時系列に整理しただけでは、まだ改ざんが容易である。そこで、次に各データを一定の間隔で区切り、まとまったデータの塊とする。これを「ブロック」と呼ぶ。そして、ブロックに対しハッシュ関数を適用する。ハッシュ関数とは、データを固定長のデータに対して一意に定まる文字列に変換する関数であり、変換された文字列から変換前の文字列を

特定するのが困難な一方向性関数である。ハッシュ関数の出力は「ハッシュ値」と呼ばれる。このハッシュ関数を各ブロックについて適用し、改ざんがあった場合にハッシュ値が合わなくなることで、改ざんを検知することができる。

しかし、ブロックごと改ざんされた場合には改ざんを検知できない。そこで、各ブロックについて、前のブロックのハッシュ値も含めてハッシュ関数にかけることにする。このようにすることで、例えばt番目のブロックを改ざんした時に、$t+1$番目のブロックや、$t+2$番目のブロックなど、後続の全てのハッシュ値が書き換わるため、改ざんを確実に検知できる。以上がブロックチェーンの定義における「電子署名とハッシュポインタを使用し改竄検出が容易なデータ構造を持ち」が意味するところである。

以上の仕組みに加えて、ブロックチェーンを多数のノードで分散的に保持する。これにより、ある特定のノードが攻撃を受けたり、ネットワークから離脱するようなことがあっても、システム全体としては動き続けることになるため、高い可用性を実現できる。また、多数のノードによる監視があるため、データの改ざんが他のノードによって迅速に発見・処理されることとなる。以上がブロックチェーンの定義における「当該データをネットワーク上に分散する多数のノードに保持させることで、高可用性及びデータ同一性等を実現する」の意味するところである。

加えて、実際のブロックチェーンシステムにおいては、ブロックの書き換えをさらに難しくするため、ブロックの生成に高負荷な計算を必要とする PoW（Proof of Work）を始めとする「コンセンサス・アルゴリズム」が実装されている。

最後に、一般的なブロックチェーンシステムにおけるトランザクションの処理の流れを説明する。まず、ネットワークに参加している各ノードからトランザクションが発行される。トランザクションはネットワーク参加者全員に送付される。これを「ブロードキャスト」という。ブロードキャストされたトランザクションを受け取ったノードは、これを「トランザクションプール」と呼ばれる領域に保持する。そして、このトランザクションプールから、各ブロックに格納できる上限までトランザクションを選択し、これらのハッシュ値を取得し、ブロックを生成する。PoW であれば、一定の条件を満たす数値「ナンス」を発見するまで総当りの計算を繰り返し、一番初めにナンスを見つけたノードがブロックを繋げることができる。ブロックは全ノードにブロードキャストされ、各ノードはそのブロックを検証し、問題がなければ再びマイニングを開始する。

A.2　ブロックチェーン技術の背景

このような現在「ブロックチェーン技術」と呼ばれる技術を世界で初めて提案したのは、Satoshi Nakamoto[6]である。彼は、自身が所属する暗号技術メーリングリストにて、2008 年

[6]Satoshi Nakamoto 本人に関することは、2021 年現在一切明らかになっていない。この名前が本名か、偽名か、そもそも一人なのかグループなのかも不明である。

に「Bitcoin: A Peer-to-Peer Electronic Cash System」[7]という論文を発表した。これが現在のビットコインの主要なアイデアとなる。2009 年には同論文を実装した Bitcoin-Qt[8]の運用が開始されており、現在でも幾つかの改良が加えられながら、同じシステムが運用されている。

ビットコインは、信頼できるノードが存在しないネットワークにおいて、「価値の移転」を可能にしたという意味で画期的なシステムであった。例えば、銀行を始めとする現在の通貨システムや、「Suica」や「PayPay」、「LinePay」といった電子通貨は、信頼できる第三者が取引に介在している。信頼できる第三者がいることで、通貨の偽装や、同じ通貨を同時に別の 2 人に支払うといったことができない。これは、信頼できる第三者が取引を監視し、データの正当性を保証し、残高の管理を厳密に行うためである。

しかし、このような「信頼」に基づいた中央集権型のモデルには多くの欠点も存在する。Satoshi Nakamoto(2008)は、例として次の 2 つの問題を挙げている。第 1 に、中央集権型のモデルは完全に不可逆な取引を提供できないこと、第 2 に多額の手数料が発生するために少額決済ができないことである。例えば、銀行を始めとする金融機関は、預金者の A さんが詐欺師の B さんに誤って送金したとき、A さんと B さんの争議を仲裁して、A さんの送金を取り消す権限を持っている。このように、取引は「覆る」ことが、中央集権モデルではあり得る。このことを「不可逆な取引を提供できない」と言い表している。また、このような例外的な手続きは、金融機関では日々多数発生する。これらへの対処にかかるコストが手数料とし徴収されることで、少額決済が難しくなる。実際に、「ゆうちょ銀行」では 2021 年 12 月 19 日現在、国際送金には 7500 円もの手数料が発生し、加えて調査請求や取り消し、事故訂正には 3000 円の手数料が発生する[9]。例えば、1000 円程度の少額決済では、最大で 10 倍程度のコストが発生する可能性があり、殆ど採算を取ることは難しい。このような送金コストの存在がインターネットを介した価値の移転が妨げてきた歴史がある。

このような問題意識を受けて、ビットコインは信頼できる第三者の存在を仮定しない、非中央集権型の自律的な通貨システムとして提案された。

しかし、信頼できる第三者の存在を仮定しないデジタル通貨では、通貨の偽造や、ダブルスペンディングと呼ばれる、同じ通貨を同時に 2 人の相手との決済に使うという不正行為が可能になる。これらはいずれも、デジタルデータのコピーが容易であるという性質から生じる問題点である。デジタルデータはコピーが容易であるため、真贋を見分けることが原理的に不可能であり、また意味も無いのである。そして、そのために同じデータを同時に 2 人に送りつければダブルスペンディングを行うことができる。

---

[7] 実は、この論文では「ブロックチェーン」という言葉は使われていない。

[8] 現在の BitCore である

[9] ゆうちょ銀行 HP, その他の料金. https://www.jp-bank.japanpost.jp/ryokin/rkn_others.html（2021 年 12 月 19 日確認）

しかし、ビットコインでは、ネットワーク上に存在するノード間で同じデータを共有しつつ、そのデータの改ざんを難しくすることで、上の２つの問題を解消している。

ブロックチェーン技術は、前述した通り、各ノードが 1 対 1 の関係で接続される P2P というネットワーク形態を取る。その中で、時系列に整列されたデータをブロックと呼ばれる単位で格納し、前のブロックの情報を埋め込むことで、前後のブロックに情報的な依存性を付けることで、改ざん検知を容易にしている。そして、ブロックの追加にマイニングと呼ばれる高負荷な計算を必要とすることで、不正なデータを含んだブロックを簡単に生成できないように工夫されている。

以上の仕組みは、インターネット上における通貨システムを実装するために構築されたものだが、データの真正性や完全性を分散自立型ネットワークで確保できるという点は、他の多くの分野でも応用できるものである。例えば電子投票は、行政を信用できない国家では、投票結果の改ざん等を防ぐために有用である。また、ブロックチェーンシステムの分散性を利用して IoT デバイスを管理したり、医療データを改ざんから防ぐような応用も考えられている。近年ブロックチェーン技術が注目されている背景には、このようなブロックチェーン技術の高い応用可能性がある。

以下の節ではこのようなブロックチェーン技術について、主要な要素技術である「一方向性ハッシュ関数」、「電子署名」、「P2P」について説明した後、これらを組み合わせた「トランザクション」や「ブロックチェーン」、「コンセンサス・アルゴリズム」の概念について詳細に説明する。

A.3　一方向性ハッシュ関数

一方向ハッシュ関数は、任意のデータを固定長の数列に変換する関数である。ただし、その変換には次の 4 つの性質がある。第 1 に決定性、第 2 に連続性、第 3 に一方向性、第 4 に衝突耐性、第 5 に高速に計算できることである（結城 2017; 株式会社 FLOC 2019）。

A.3.1 決定性

決定性とは、入力が同じとき、必ず同じ値を返す性質のことである。例えば、

I am sleepy.

という文字列を一方向ハッシュ関数の一種である「SHA-256」に入力し、見やすいように 16 進数に変換すると、次の数列が出力される。

3ac505c3ab3f0607ef8aafd0f8f37079873c76bcb4092cb9d14e9834d63c7c10

これが「I am sleepy.」の「ハッシュ値」である。当然、これを再び「SHA-256」に入力しても、同じ結果が得られる。当たり前のようにも見えるが、例えば二次関数のように、一つのx に対し、２つ値が定まるというようなことが、起きないということが重要である。

A.3.2 機密性

連続性とは、入力と入力の相関関係が無いように見える性質である。例えば、先程の入力である「I am sleepy.」を、「I an sleepy.」と、m を n に変えて再び SHA-256 に入力すると

fa5ed535e5ec37362d5b5111d09cf363aa900cd20fd11cc58b6c7c3490094fc2

という文字列が出力される。「I am sleepy.」と比較すると、1 文字変えただけで全く異なる値が出力されていることが分かる。このように、全体の一部が少しでも変わると全く違う値が出力されることも、一方向ハッシュ関数の特徴の一つである。

A.3.3 衝突耐性

衝突耐性とは、同じハッシュ値を出力する異なる入力を探すことが困難である性質である。これは例えば、先程計算した「I am sleepy.」のハッシュ値「3ac505c3ab3f0607ef8aafd0f8f37079873c76bcb4092cb9d14e9834d63c7c10」と同じ値を出力する「I am sleepy.」以外の入力を見つけるのが非常に難しいという性質である。

ただし、耐性が高いだけであって、見つけることが不可能なわけではない。例えば、この世にある全ての文字列について、ハッシュ値を求めていけばいつかは見つかる可能性はある。このような攻撃は「ブルートフォース攻撃」あるいは「総当り攻撃」、「現像攻撃」などと呼ばれており、理論的には出力長がLの一方向ハッシュ関数であれば、$2^{\frac{L}{2}}$回未満の試行回数で発見できることが知られている（Hoffman et.al 2005）。従って、例えば SHA-256 であれば、ビット長が 256 であるため、$2^{\frac{256}{2}} = 2^{128} \cong 10^{38}$回の試行回数で計算可能である。これは、例えば 442.01PFLOPS の計算能力を持つ日本最高峰のスーパーコンピューター「富岳」を使っても約2.26×10^{29}秒、あるいは7.17×10^{21}年程度かかる計算となる（理化学研究所 2021）。これは宇宙の始まりから現在までの経過時間とされている1.38×10^{10}年（138 億年）より遥かに長く、事実上計算不可能である。

A.3.4 一方向性

一方向性とは、あるハッシュ値が与えられたとき、そのハッシュ値から元の入力を逆算するのが困難であるという性質である。例えば、「3ac505c3ab3f0607ef8aafd0f8f37079873c76bcb

4092cb9d14e9834d63c7c10」というハッシュ値から、入力である「I am sleepy.」を見つけるのは非常に難しいということである。

あるハッシュ値から元の入力を見つけ出すことも、逆算方法が確立され、現在は使用されていない一方向ハッシュ関数を除けば、一般的にはブルートフォース攻撃しか方法はない。理論的には、出力長がLの一方向ハッシュ関数であれば、2^L回未満の試行回数で発見できることが知られている（Hoffman et.al 2005）。従って、例えば SHA-256 であれば、ビット長が 256 であるため、$2^{256} \cong 10^{77}$回の試行回数で計算可能である。これは、例えば 442.01PF LOPS の計算能力を持つ日本最高峰のスーパーコンピュータ「富岳」を使っても約2.26×10^{68}秒、あるいは2.94×10^{61}年程度かかる計算となる。

A.3.5 計算速度

一方向ハッシュ関数は、一般的に高速に計算できる必要がある。ハッシュ関数はデータに対し、一意に定まる要約を与えるため、データのインデックスにも使用されることがある。大量のデータを処理するためには、インデックスの作成にかかる時間が長ければ使い物にならない。

A.4　電子署名

電子署名とは、あるデジタルデータが確実に署名者本人によって作成されたものであり、同時に書き換えられていないことを証明する技術である。即ち、現実の紙媒体の文書における「はんこ」や「署名」に当たるものを、デジタルに実装したものであると言い換えることができる。

電子署名を理解するためには、まず公開鍵暗号方式を理解する必要がある。本節では暗号化に関する用語を整理した後、公開鍵暗号方式および電子署名について解説する。

A.4.1 暗号技術に関する用語整理

一般的に、ある秘密にしたいデータがあるとする。これを暗号に変換するために、特定の「暗号化アルゴリズム」を用いて異なる文字列に変換する。この変換を「暗号化」と呼び、変換前のデータは「平文」と呼ぶ。そして、この変換時にアルゴリズムに入力するパラメータを「鍵」と呼ぶ。「暗号化」されたデータから平文を求める作業は「復号」と呼ぶ（結城 2017）。以上を図示したものが図である。

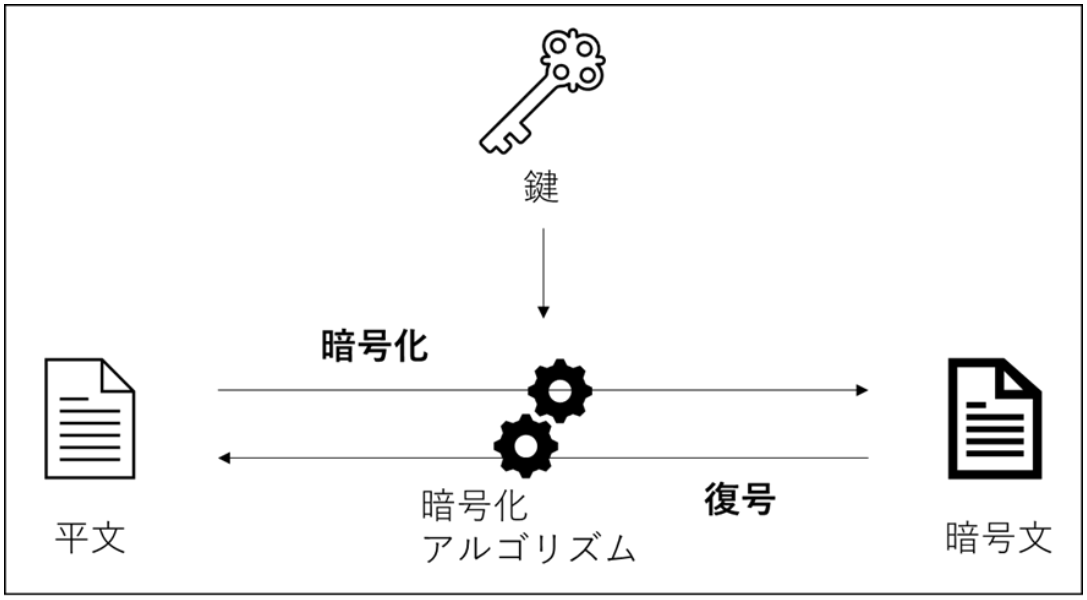

図 6　暗号化の流れ

暗号化は、その手法に応じて大きく対象暗号と公開鍵暗号に分類できる。対称暗号とは、暗号化に使う鍵と復号に使う鍵が同一の暗号であり、具体的な暗号化アルゴリズムとしては、シーザー暗号、単一換字暗号などがある。公開鍵暗号は、暗号化と復号を別の鍵で行う方式で、暗号化に使う鍵は「公開鍵」、復号に使う鍵は「秘密鍵」と呼ばれている。具体的な暗号化アルゴリズムとしては、楕円曲線暗号や RSA などがある。

対称暗号は、暗号化も復号もシンプルに実装でき、処理速度が早い一方で、「鍵配送問題」が発生する。例えば、データを第三者に秘密にして送信したい者、即ち送信者が、暗号文を受け取り手である受信者に暗号文を送付するとする。この暗号文を復号するためには、相手に鍵と暗号化アルゴリズムを教える必要がある。しかし、一般的にインターネットでは流れるデータを当事者に知られることなく窃取する「盗聴」が非常に容易である。従って、盗聴者の存在を仮定すると、対象暗号のみではこの 2 者間で秘密のやり取りをするのが不可能になる。このような問題を鍵配送問題と呼ぶ。

A.4.2 公開鍵暗号方式

公開鍵暗号方式とは、公開鍵暗号を使った暗号文による通信の方式である。公開鍵暗号は、前述したように秘密鍵と公開鍵を用いる。この秘密鍵と公開鍵に関しては、図 7 に示すような 4 つの性質がある。第 1 に、秘密鍵から公開鍵は生成できても、公開鍵から対になる秘密鍵を逆算して求めることは、事実上できないこと、第 2 に公開鍵は公開してもよいが、秘密鍵は他者に共有してはならないこと、第 3 に公開鍵で暗号化した文書は、対になる秘密鍵でのみ復号が可能であること、第 4 に秘密鍵で暗号化した文書は、対になる公開鍵でのみ復号が可能であることである。

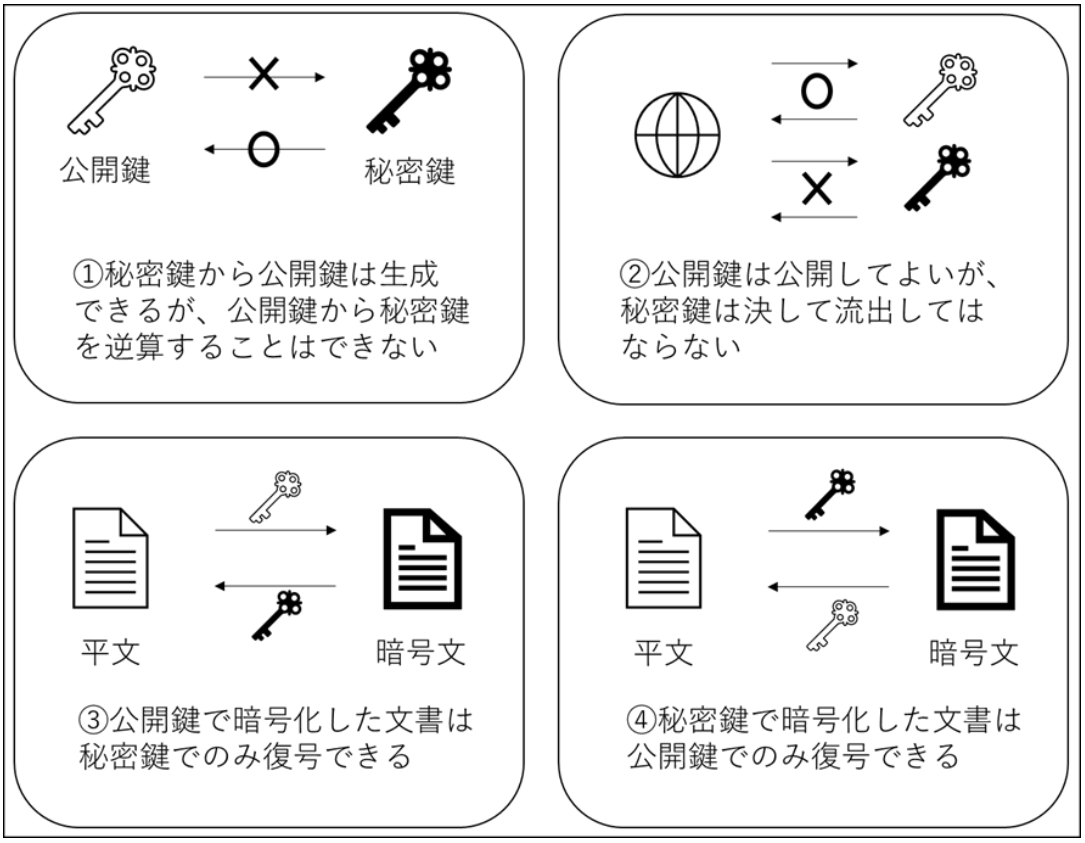

図 7　秘密鍵と公開鍵の性質

以上を前提として、ここでは送信者が受信者に対し、図８の手順で文書を送ることを想定する。

まず、受信者は秘密鍵と公開鍵のペアを生成する。送信者は、受信者の公開鍵を取得し、これを用いて暗号文を作り出す。この暗号文を受信者に送信し、受信者は自身の秘密鍵を使って平文に復号することで秘密通信が可能になる。

仮に、図８の②・④の地点で盗聴が行われたとする。盗聴者は、受信者の公開鍵と暗号文を持っているが、暗号文は対になる秘密鍵でしか復号できないため、中身を見ることはできない。また、公開鍵から秘密鍵を逆算することは難しいため、秘密鍵を逆算して暗号文を復号することも不可能である。以上より、送信者と受信者の秘密通信が成立する。

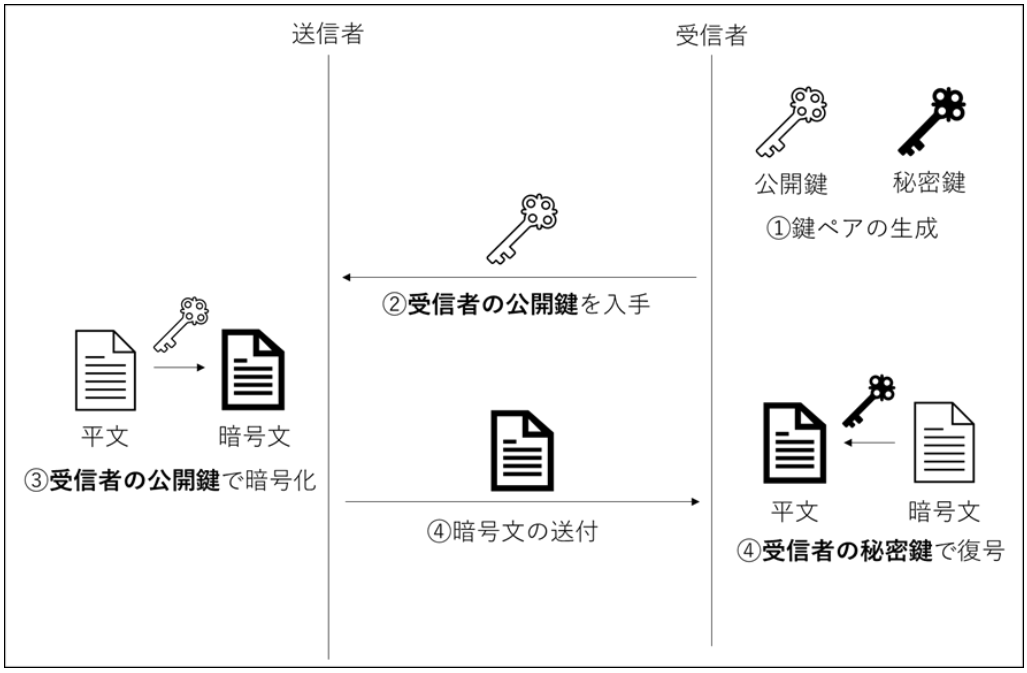

図 8　公開鍵暗号方式による通信

A.4.3 電子署名

次に、電子署名について説明する。電子署名は、いわば公開鍵暗号の性質を「逆に」利用したものである。電子署名を利用することで、受信者は、送信者から送られた文書が、確実に送信者によって作成されたものであること、また途中で文書の「改ざん」が行われていないことを確認できる。ここでは、送信者が受信者に電子署名を付した文書を公開鍵暗号方式で送付することを想定して説明を行う。

電子署名の生成と確認は、大きく「送信者側の前処理」、「通信」、「受信者側の後処理」の３つの段階に分けることができる。

まず、送信者側の前処理として、図９に示す次の３つの処理を行う。

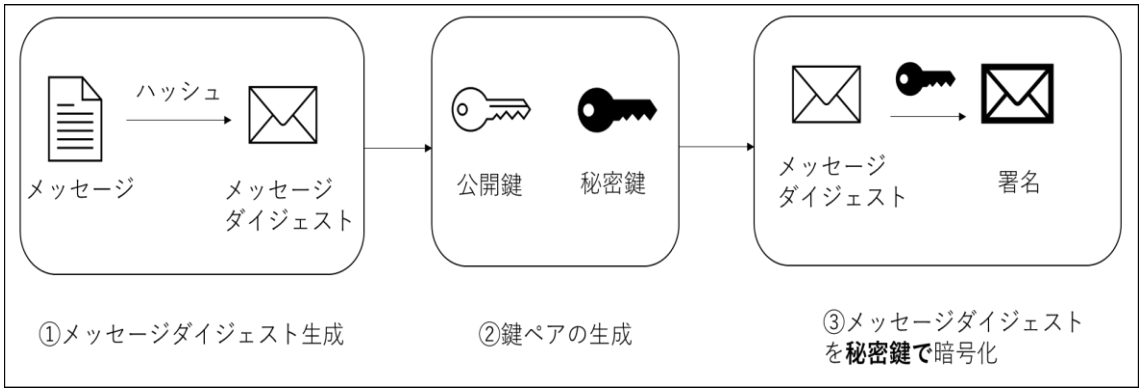

図 9　送信者側の前処理

まず、電子署名を施す文書（以下メッセージと呼ぶ）を一方向ハッシュ関数にかけ、ハッシュ値を得る。このハッシュ値は、この文書に対して一意に定まるという意味で、メッセージの要約と言える。従って、専門的にはこれを「メッセージダイジェスト」と呼ぶ。次に、公開鍵と秘密鍵のペアを生成し、メッセージダイジェストを秘密鍵で暗号化する。この暗号化されたメッセージダイジェストを「署名」と呼ぶ。

次に、図１０に示すように前述した公開鍵暗号方式で通信を行う。

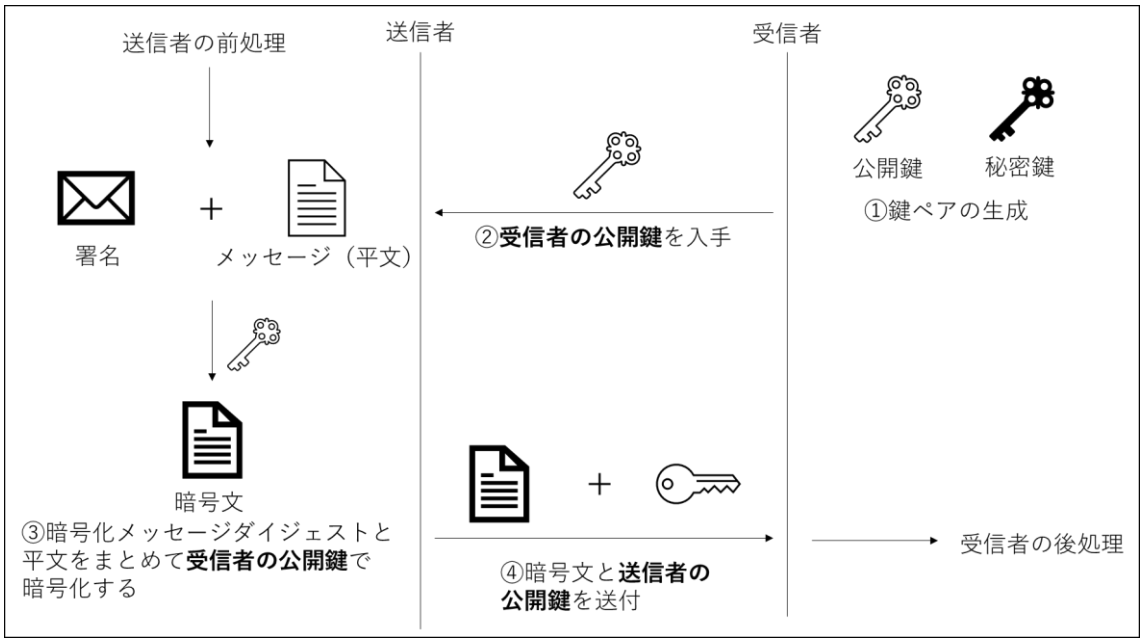

図 10　公開鍵暗号方式による通信

まず、送信者は受信者の公開鍵を入手する。次に、署名と平文のメッセージをまとめて受信者の公開鍵で暗号化する。そして、送信者の公開鍵と暗号文を受信者に送付する。

最後に、図 11 に示す受信者の後処理について説明する。

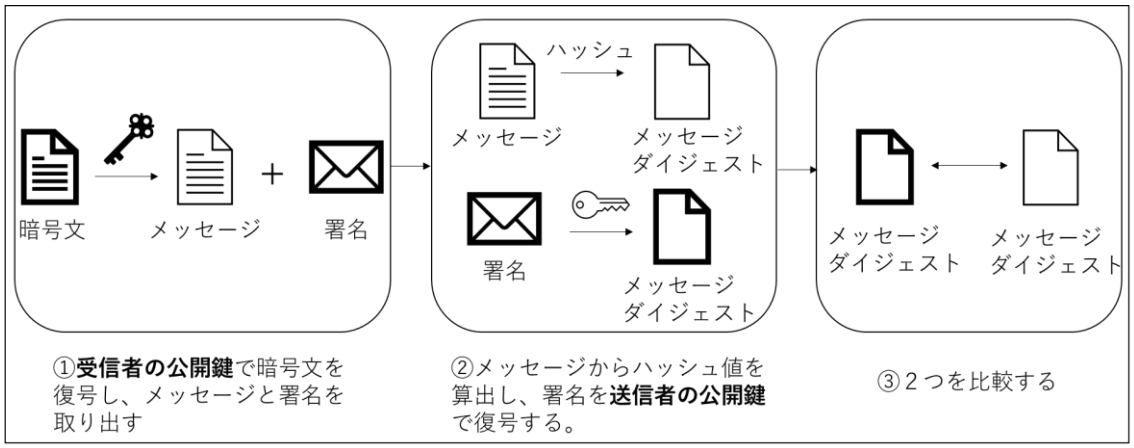

図 11　受信者側の後処理

まず、送付されてきた暗号文を自身の秘密鍵で復号し、メッセージと署名を取り出す。次に、取り出したメッセージを、「受信者側の前処理」時と同じ一方向ハッシュ関数にかけ、ハッシュ値、即ちメッセージダイジェスト A を得る。もし、復号が失敗すれば、それはメッセージが送信者本人から送付されていないこと、即ち「なりすまし」を意味する。

次に、署名を同封されていた送信者の公開鍵で復号し、メッセージダイジェスト B を取り出す。最後に、メッセージダイジェスト A・B を比較する。同じであれば、メッセージは改ざんされておらず、確かに送信者が作成したものであると確認できる。しかし、異なれば改ざんが行われたことが分かる。

何故、電子署名の復号とメッセージダイジェストの照合で改ざんとなりすましを検知できるのか。そもそも、メッセージダイジェストは、もし文書が改ざんされれば、一方向ハッシュ関数の性質から全く異なる値を出力する。また、送信者の公開鍵で復号できるのは、送信者の秘密鍵で暗号化されたメッセージのみである。送信者の秘密鍵が漏洩する以外は、秘密鍵を知っているのは送信者本人だけである。従って、署名が送信者の公開鍵で復号できれば、それは送信者がメッセージを作成したことの証明となり、受信者側で改めて計算したメッセージダイジェストが同じであれば、それは文書が改ざんされていないことの証明となるのである。

A.5　P2P モデル

P2P モデルとは、「中央コントローラが存在せずに、構成要素のプロセス（ピアと呼ばれる）が自律的に他のプロセスと協調動作する完全分散型のモデル」と定義されている（滝沢, 榎戸　2014）。本節では、P2P の対象となるアーキテクチャである「クライアント・サーバモデル」について説明した後、P2P モデルについて説明する。

A.5.1 クライアント・サーバモデル

P2P モデルは、中央集権的なサーバがシステムを管理する「クライアント・サーバモデル」と対象となる考え方である。クライアント・サーバモデルでは、図に示すようにサーバとクライアントが分離している。

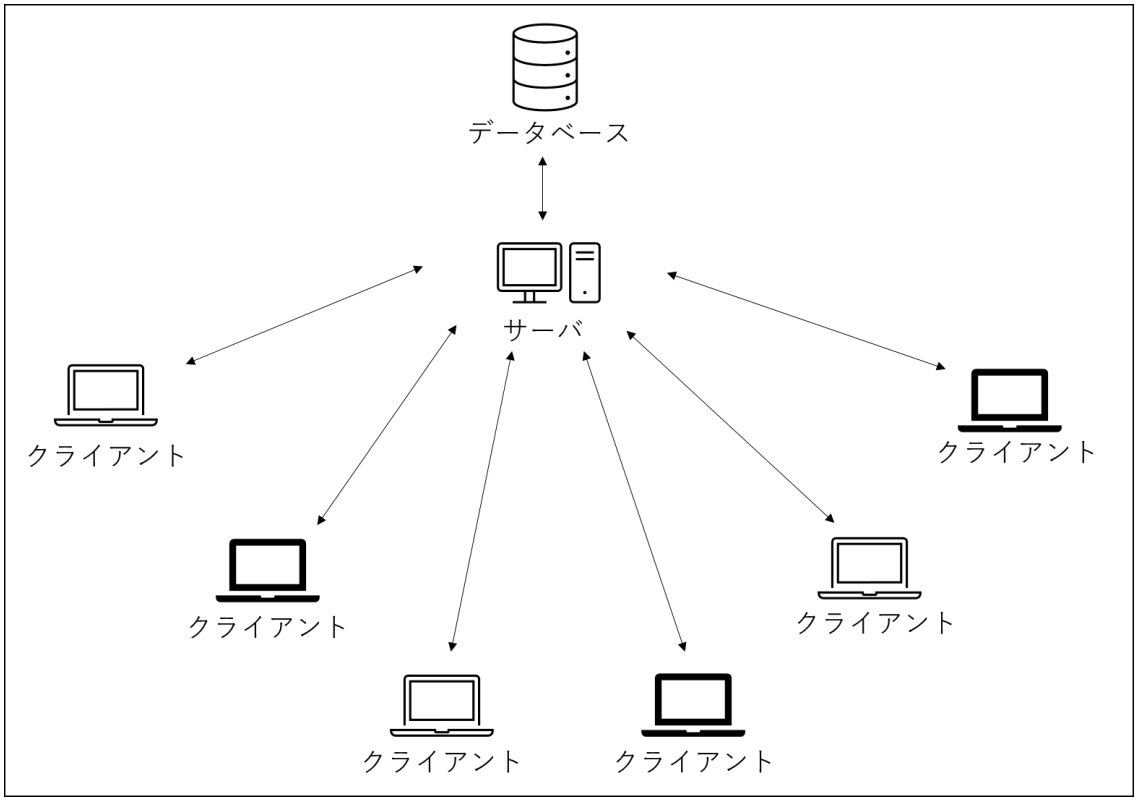

図 12　クライアント・サーバモデル

「クライアント」は主にデータを問い合わせるコンピュータであり、「サーバ」はデータを提供するコンピュータである。例えば、銀行のシステムであれば ATM がクライアントであり、残高等を管理する勘定系システムがサーバとデータベースに当たる存在である。

このようにコンピュータ間に異なる役割を持たせることで、処理の簡素化・高速化や、システムの管理が容易になるというメリットがある。一方で、図１２から明確に分かるように、サーバが何らかの理由でダウンした場合、システム全体が停止することになる。このような、システムのある１点がダウンしたとき、システム全体がダウンするようなポイントを「単一障害点」と呼ぶが、クライアント・サーバモデルでは単一障害点の存在が主要な問題とされている。例えば、みずほ銀行の勘定系システムの障害[10]などは、システムの中央集権的な管

[10] JIJI.COM, 「外為送金、最大９１件遅延　１２日のシステム障害―三井住友銀」https://www.jiji.com/jc/article?k=2021101300543&g=eco（2021 年 12 月 19 日確認）

理による弊害の一つと言える。加えて、多数のノードを管理するためには高性能なサーバが必要となり、ネットワーク回線への負担も大きくなる。このような理由から、クライアント・サーバモデルはスケーラビリティに限界があるとされている。

A.5.2 P2P モデル

一方、図１３に示す P2P モデルでは、システムの管理者が存在せず、全てのコンピュータが対等な関係で接続されている。従って、P2P モデルでは、各コンピュータの役割という側面から見れば、全てのコンピュータが「サーバ」にも「クライアント」にもなり得るということになる。このような P2P モデルの形態は特に「ピュア P2P」と呼ばれている。

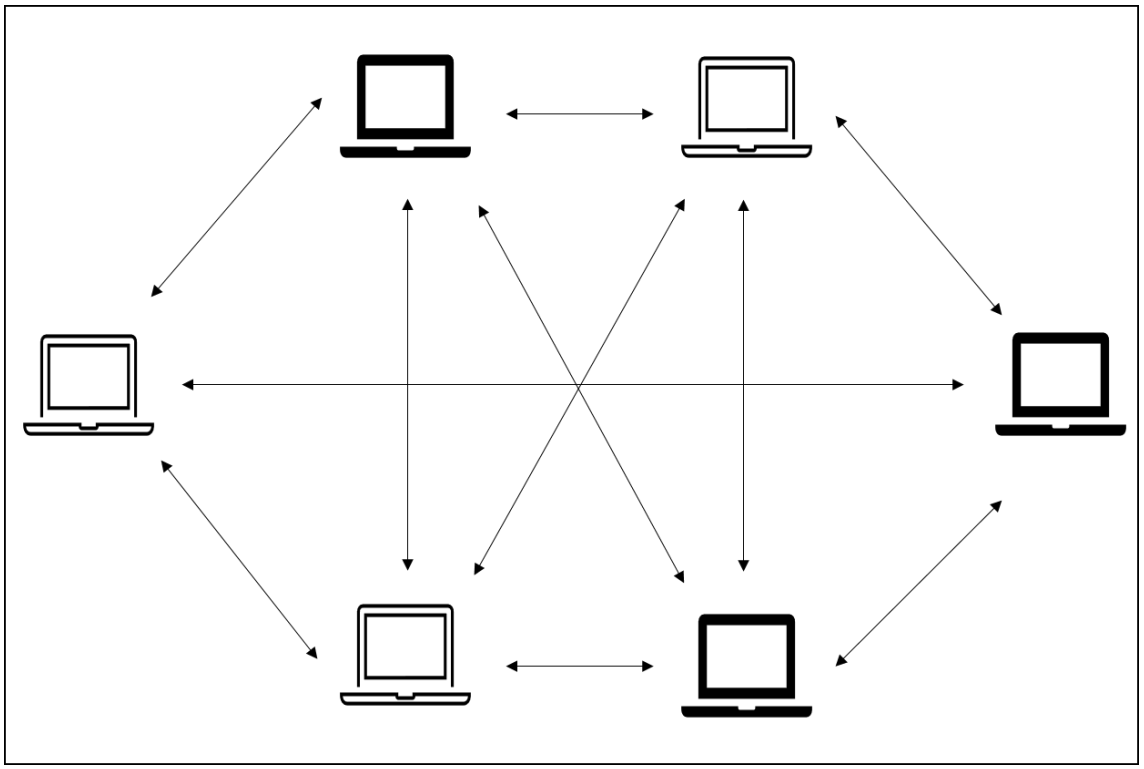

図 13　P2P モデル

ピュア P2P モデルの最大の特徴は単一障害点を持たず、そのためにシステムの可用性が非常に高いことである。また、多数のコンピュータに処理を分散することで、スケーラビリティも確保できる。一方で、参加も離脱も自由に行える P2P モデルでは、安全性・確実性・パフォーマンス等に課題が存在する。安全性の面で言えば、ネットワークトポロジが頻繁に変化し、ネットワークの分断が発生する懸念もあり、これらがシステムへの攻撃に繋がる可能性もある。確実性に関しては、ブロードキャストがきちんとネットワーク全体に行き渡るのか保証できない点、データの同期や到達保証に関しても、クライアント・サーバモデルほど厳密には行えないという課題がある。また、パフォーマンス面では、ネットワークの遅延

によるデータ同期の遅れが発生するために、リアルタイム性が求められるサービスの提供が難しいといった課題がある（赤羽, 愛敬 2017)。これらの問題は、第 7 節で説明する「ビザンティン将軍問題」の直接の要因となっている。

A.6　トランザクション

本節では、「トランザクション」の概念について解説する。ビットコインにおけるトランザクションとは、取引データを意味する。即ち、「誰が」、「誰に」、「いつ」、「いくら」送金するのかまとめたデータである。ビットコインにおけるトランザクションの構成は複雑なため、本論文ではダミーデータを格納したトランザクションを使用したが、本節では実際のトランザクションについても詳細に解説する。

A.6.1 トランザクションの構成

ビットコインにおけるトランザクションは、表１０に示すような構成となっている。このようなデータの構造は「json」と呼ばれており、「キー：値」という関係が各要素で成立している。json は人間に分かりやすく、かつコンピュータでも処理しやすい形式である。

表 9　ビットコインのトランザクション


```
"tx": [
    {
      "txid": "083eb2937a2b068c558f02fcaa4d7923ef3caa99e7555067f9feac6d47265eb2",
      "hash": "3f4b222fb8c22db6a081bde2647b0218f47085e52b60b825aef5b440ca1b318d",
      "version": 1,
      "size": 228,
      "vsize": 201,
      "weight": 804,
      "locktime": 0,
      "vin": [
        {
          "coinbase": "0396d21800045739fa5d043a0d29290c9218f95ded000000000000000a636b706f
6f6c1b2f7e7e20626173696362697463682e736f667477617265207e7e2f",
          "sequence": 4294967295
        }
      ],
      "vout": [
        {
          "value": 0.39147111,
```

```
            "n": 0,
            "scriptPubKey": {
              "asm": "0 eecb235403933d7e325573f19d82cddc3288f5b8",
              "hex": "0014eecb235403933d7e325573f19d82cddc3288f5b8",
              "reqSigs": 1,
              "type": "witness_v0_keyhash",
              "addresses": [
                "tb1qam9jx4qrjv7huvj4w0cemqkdmseg3adc2l0htw"
              ]
            }
          },
          {
            "value": 0.00000000,
            "n": 1,
            "scriptPubKey": {
              "asm": "OP_RETURN aa21a9ed59a31cf19a1e87351378e342ba50bf86e9c36026c9961d2b
088e376d56732347",
              "hex": "6a24aa21a9ed59a31cf19a1e87351378e342ba50bf86e9c36026c9961d2b088e376d5
6732347",
              "type": "nulldata"
            }
          }
        ],
        "hex": "010000000001010000000000000000000000000000000000000000000000000000000000
000000ffffffff3f0396d21800045739fa5d043a0d29290c9218f95ded000000000000000a636b706f6f6c1b2f
7e7e2062617369636269746368 2e736f667477617265207e7e2fffffffff0267565502000000001600 14eecb2
35403933d7e325573f19d82cddc3288f5b8000000000000000266a24aa21a9ed59a31cf19a1e87351378e3
42ba50bf86e9c36026c9961d2b088e376d5673234701200000000000000000000000000000000000000000
0000000000000000000000000000000000"
      },
```


ここでは重要な要素のみ完結に説明する。まず「txid」は各トランザクションに一意に定められた ID である。「vin」は出力、即ち送金者が「いくら」送金するのかを示している。後述するように、ビットコインでは複数の送金先を同時に選べるため、この欄はいくらでも長くなり得る。「address」はネットワーク内のノードを一意に定める ID である。「scriptPubKey」はトランザクションへの署名に使用した秘密鍵の対となる公開鍵で、署名の検証に用

いる。「amount」は送金量を意味する。(松浦, 司　2017; 株式会社 FLOC　2019; アントロノプス　2018)。

A.6.2 UTXO モデル

ビットコインでは、残高の管理に UTXO（Unspent Transaction Output）という考え方を利用する。UTXO とは何かを説明する前に、まずあるユーザが持つ残高を確認するにはどういった方法が考えられるのか考察する。

初めに思いつくのはアカウントベースの管理である。例えば、あるユーザの残高をアカウントと一意に紐付けて管理すれば処理を簡略化できる。これは、銀行口座における残高の管理方法と殆ど同じと言ってよいだろう。誰かに送金するときは、送金者の残高を減らして、宛先の残高を増やすだけでよい。

しかし、この方法は非中央集権型のネットワークには馴染みにくい。アカウントと残高を紐付けることは、各預金者の特定に繋がり、匿名性を著しく損なう。また、残高が多い者の特定も容易であり、攻撃を受けやすくなるというセキュリティ上の懸念もある。

そこで、アカウントベース以外の残高の管理方法が必要となる。そもそも、残高が変化するのは口座に入出金があった場合のみである。従って、入出金の記録のみを保持すれば、口座という概念を使う必要は無くなる。即ち、「自身が持っているコインを誰かに送金した」という「事実」をかき集めれば口座からの出金記録を作ることができるし、「誰かから入金された」という「事実」をかき集めれば、口座への入金記録を作ることができ、これらをまとめれば残高が一意に定まる。ここでは説明の簡略化のために「口座」という言葉を使ったが、これは他者から観察して一意に定まる必要もない。つまり、例えば口座番号「××○○△△□□」というものではなく、例えば毎回異なる口座番号を作り出して、そこに入金させるようにしたとしても、過去の口座番号を全て自分だけが把握していれば、自分の残高は簡単に分かる。

しかし、例えばこの状況を実装して、実体の無い「コイン」のデータを受け渡しするだけでは簡単にコインの偽装ができてしまう。従って、コインのデータを偽装できない形に変える必要がある。

ここで、第 3 節で説明した「電子署名」を使う。「コイン」の偽装が簡単にできるのは、「コイン」のデータを簡単に複製できるためであった。簡単に複製できる「コイン」には、その「コイン」所持者にコインを使用する「正当性」が無いことと同義である。コインを使う「正当性」がある場合とは即ち、そのコインが確実に前の所有者の同意の元で、自分へ送金されたものであるという「証拠」がある場合のみである。ビットコインではこの「証拠」に当たるものを「電子署名」で実装している。即ち、自分の所持するコイン（正確には UTXO）に、前の所有者の電子署名が付されており、かつこの電子署名が検証の結果有効である場合にのみ、コインを使う「正当性」が与えられることとなる。

以上のことから、コインを使う（送金する）際には、前の所有者の電子署名の検証作業が要求されるため、各トランザクションには必ず前の所有者の電子署名が付されている。一連の流れを図示すると図１４のようになる。

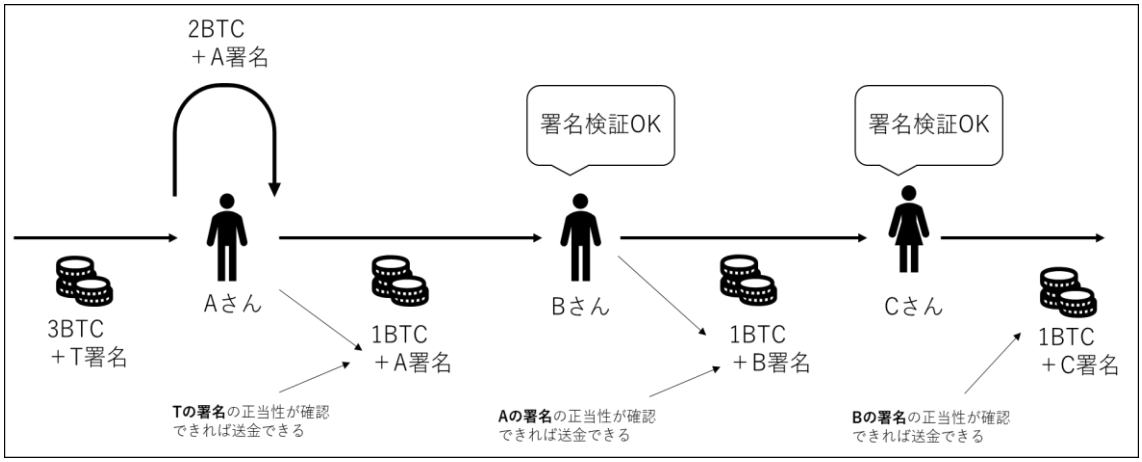

図 14　UTXO モデルによる送金フロー

図１４の通り、A さんは自身のコインを第三者に送金する際には、前の所持者である T さんの署名が必要となり、検証の結果問題なければ送金可能となる。この検証はビットコインに参画する多くのノードによって行われるため、コインの偽装（＝署名の偽装）を行なったとしても、そのトランザクショは無効化される。

また、前述の通り、ビットコインでは「口座」という概念は存在しない。あるのは自身の公開鍵のみであり、そこに紐付けられた（送金された）コインを送ることができるだけである。従って、ビットコインでは、一回の取引で全財産を使い切らなければならない。余った残高は再び「自分に送金」することで、残高を管理するのが UTXO モデルである。例えば、上の図の状態で A さんが B さんに送金する前後の UTXO の変化は図１５の通りである。

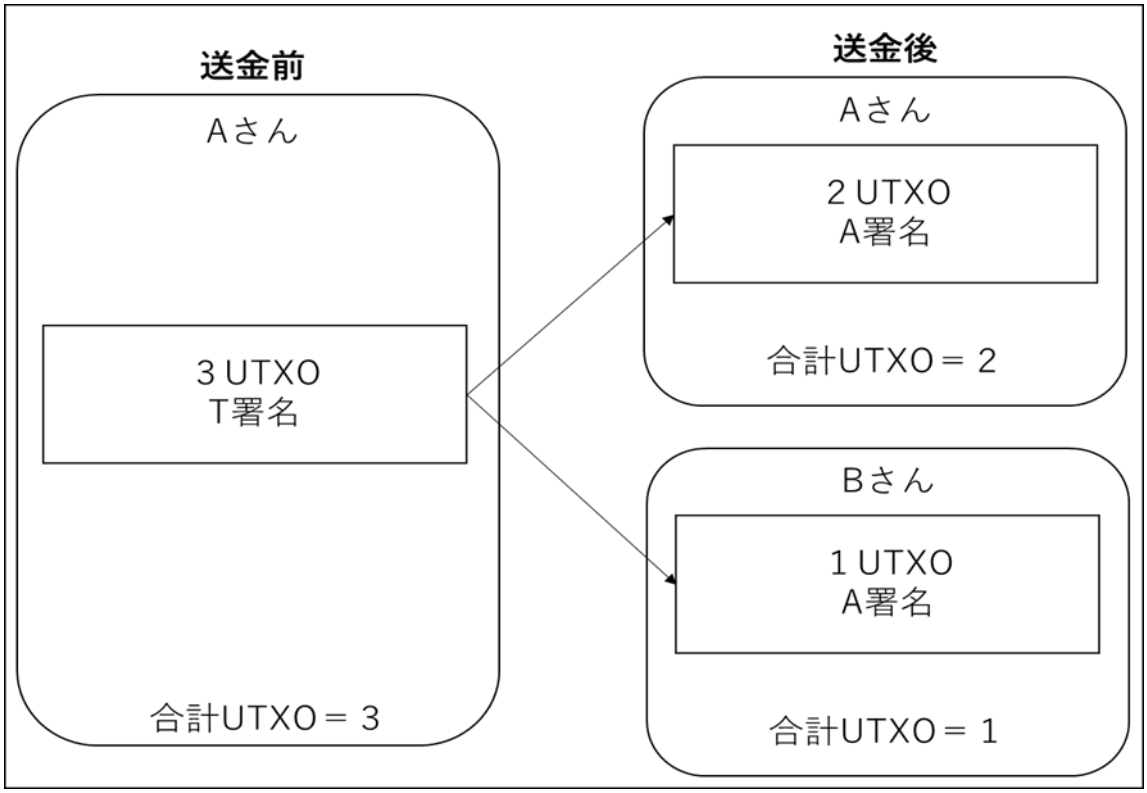

図 15　UTXO の変化

A さんの UTXO は当初、T さんから送付された 3 であったが、送金後は自身に送金した 2 UTXO のみになり、B さんは 1 UTXO を手にしている。このように UTXO は一度使用すると消え、新たに生成される。UTXO を全てかき集めたものが残高であるため、トランザクションを全て確認すれば残高は一意に定まる。

A.7　ブロックチェーン

前節で説明した通り、ビットコインではトランザクションの流れが各ノードの資産に直結するため、データが書き換えられたり、失われたりすることを防ぐ必要がある。従って、ブロックチェーンはデータの真正性・完全性を保証できるような仕組みとなっている。

まず、参考にビットコインにおけるブロックチェーンの中身を json 形式で表１１に示す。

表 10　ブロックチェーンの中身


```
{
  "hash": "000000000001ea421d38c4d86aa629eb2316a95ee1071da91bea8d84a81e6f8b
",
  "confirmations": 42977,
  "strippedsize": 753,
  "size": 898,
  "weight": 3157,
  "height": 1626035,
  "version": 536870912,
  "versionHex": "20000000",
  "merkleroot": "4f5269006f191dbc769adfc44c7a6a0494401889d27e621e25bc544241d1
bf69",
  "tx": [
 ....
  ],
  "time": 1576672957,
  "mediantime": 1576672899,
  "nonce": 440101750,
  "bits": "1b0ffff0",
  "difficulty": 4096,
  "chainwork": "0000000000000000000000000000000000000000000001348dc5fe29169
44b97",
  "nTx": 3,
  "previousblockhash": "00000000000ce46a9a22ce9e7bd27c337f6693768d69974a9cc37
8328c7245b5",
  "nextblockhash": "000000000002b41bfb62bb3346fb9b08f5724d6821f692842c5d3ffb9
4ececa6"
}
```


「hash」はブロック全体のハッシュ値を意味する。ブロックハッシュとも呼ばれている。「confirmations」は承認数を意味する。「size」はブロックのサイズ、「height」は何ブロック目かを意味する。「merkleroot」は、マークルツリーによって計算されたトランザクション全体のハッシュ値である。「tx」には前節で説明したトランザクションが格納される。「time」

はブロック生成日時、「ノンス」は第 7 節で説明する PoW によって求める値、「difficulty」は PoW の難易度、「previousblockhash」は 1 つ前のブロックのブロックハッシュである。

しかし、これでは構造が読み取りにくいので、重要な要素のみをピックアップし、図示したものを図１ 6 に示す。

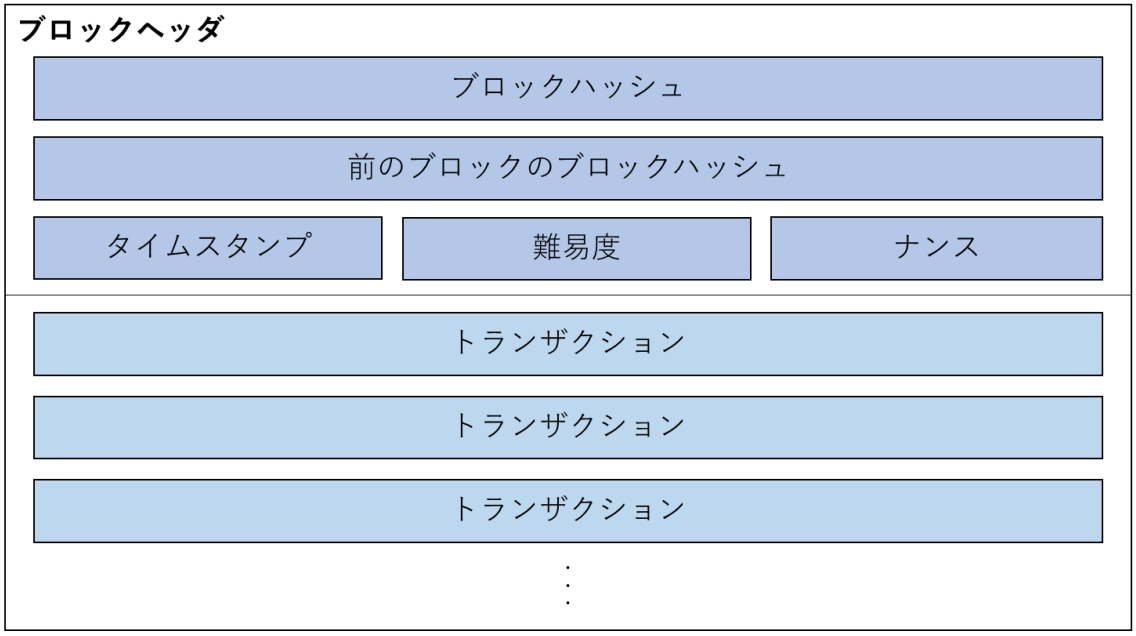

図 16　ブロックの中身

ブロックチェーンにおいて、最大の特徴と言えるのが前のブロックのハッシュ値を含んでいる点である。このようにすることで、あるt番目のブロックの中身を改ざんしたとき、$t+1$番目、$t+2$番目と、後続する全てのブロックのハッシュ値が変わることになる。即ち、t番目のブロックの改ざんを行うためには、n個のブロックがあるとき、$n-t$個のブロック全てのハッシュ値を計算し直し、ブロックをつなぎ直す必要がある。しかし、次節で述べるように、この「ブロックを繋ぐ」作業には莫大な計算資源が求められる。ブロックチェーンの改ざんが難しいとされているのは、このためである。

A.8　コンセンサス・アルゴリズム

本節ではコンセンサス・アルゴリズムについてその概略を説明し、その代表的な例として PoW（Proof of Work）、PoS（Proof of Stack）を取り上げる。

コンセンサス・アルゴリズムとは、分散的なネットワークにおいてノードが合意を形成するためのアルゴリズムである。ビットコインが想定する P2P モデルのネットワークにおいては、情報伝達の遅れや、悪意のあるノードが介在しているため、合意形成が非常に難しい

とされている。このような問題は「ビザンティン将軍問題」と呼ばれており、これを（一部）解決可能とされているのがコンセンサス・アルゴリズムである。

A.8.1 PoW

A.8.1.1 PoW の仕組み

PoW（Proof of Work）は、ノードに膨大なハッシュ演算を課し、ある特定の値「ナンス」を見つけたノードがブロックを繋げることができるアルゴリズムである。総当りでナンスを見つける作業が金を採掘する作業に似ていることから、「マイニング」とも呼ばれ、条件を満たしたナンスは「ゴールデンナンス」とも呼ばれる。

前述の通り、ブロックチェーンは悪意のあるノードでも簡単にブロックを繋げることができた場合、不正が蔓延りシステムが機能しなくなる。PoW では、ブロックを繋げることに多大な労力がかかるようにすることで、ブロックの改ざんを防いでいる。

PoW のアルゴリズムは、アルゴリズムに示すように、非常にシンプルである。尚、hash() は引数のハッシュ値を返す関数、変数 difficulty には 1 以上の整数値が設定されている。また変数 tx_hash にはトランザクションのハッシュ値、previous_hash には前のブロックのハッシュ値、time には現在時刻が格納されている。

アルゴリズム 2　mining()

```
mining()
1 nonce = 0
2 while True:
3       hash = hash(nonce, tx_hash, previous_hash, time)
4       if difficulty >= hash:
5             return nonce
```

他にも方法は考えられるが、最も単純な実装が上のアルゴリズムである。即ち、変数 nonce に 0 から順に変えながら前のブロックハッシュなどと共にハッシュ関数に入力し、それが条件を満たすか判定するものである。difficulty はゴールデンナンスの条件であり、その条件を満たしたナンスを発見できた場合に、このアルゴリズムは停止する。difficulty の値は、一般的に大きいほど条件を満たすのが難しくなり、大量のハッシュ演算が必要となる。第 2 節でも述べた通り、ハッシュ値は入力によって大きく値を変えるため、次の値を予測することは不可能であり、総当たりで計算する以外にゴールデンナンスを見つける方法は無い。

一方、ゴールデンナンスを発見したノードが、ゴールデンナンスと共に新たなブロックをブロードキャストしたとする。このとき、ゴールデンナンスとその他の情報を使って、そのゴールデンナンスが本当に正しいのか検証するのは非常に簡単である。何故なら、第 2 節 5 項でも述べた通り、ハッシュ演算は非常に高速にできるためである。従って、ゴールデンナ

ンスを見つけるということは、そのノード必要な計算をしたという証明となり、ブロックを繋ぐ権限があると言える。

そして、ゴールデンナンスを見つけたノードには一定のコインが報酬として支払われる。この報酬のために多くのノードがマイニングに参加し、システムの維持に積極的に貢献することとなる。これが PoW におけるインセンティブ・システムである。

A.8.1.2 フォーク

PoW のアルゴリズムは前述した通りだが、このやり方ではゴールデンナンスを同時に見つけるということが起こり得る。これは、ゴールデンナンスの演算は各ノードが同期して行なっているわけではなく、好き勝手なタイミングで始めているためである。

同時にゴールデンナンスが見つかった場合、図に示すようにチェーンの分岐が発生する。このことを「フォーク」と呼び、このような状態が長く続けば、トランザクションの処理に支障が発生する。

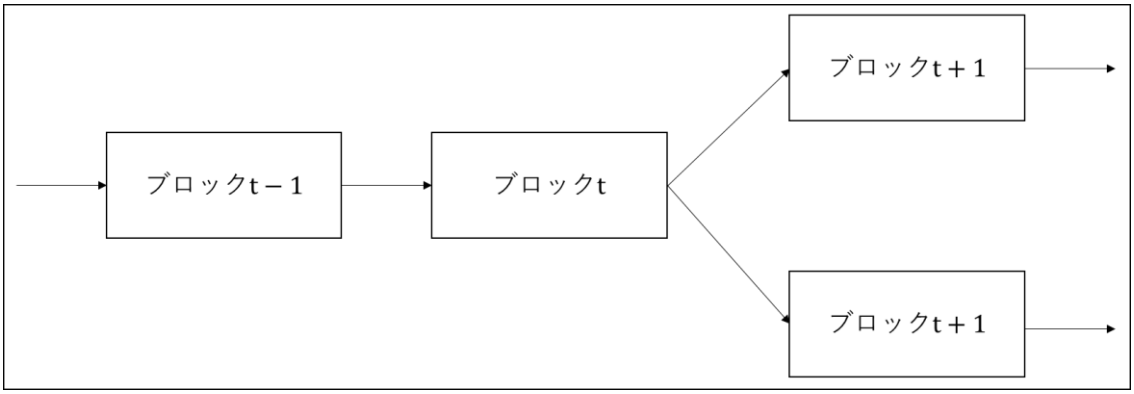

図 17　チェーンのフォーク

そこで PoW では一番長いチェーンを正当なものと見なすことになっている。前述の通り、P2P ではデータが同期されていないため、ノードによって持つデータが異なるということが生じる。従って、同じタイミングでゴールデンナンスをブロードキャストしたとしても、ノード A が作ったブロックを支持するノード群と、ノード B が作ったブロックを支持するノード群に分かれるということが起きる。PoW のアルゴリズムでは、計算資源を多く持つノードが有利である。従って、仮に複数のチェーンに分岐したとしても、やがて最も演算能力を持っているグループのチェーンが最も伸びることで、長期的にはチェーンが 1 本に収束する。最も長いチェーンにならなかったものは、一般的には破棄されることとなる。このため、ビットコインではブロックのフォークを想定して、自分が発行したトランザクションが含まれるブロックの上に、さらに 6 個以上ブロックが繋がらない限りは、トランザクションが確定しない仕様になっている。

ただし、ノード間の思想の違いなどから、故意にチェーンを分岐させ、互換性のない新しい通貨として独立させるということが、実際にはたびたびに起きている。このようなフォークは特に「ハードフォーク」と呼ばれ、「ビットコインキャッシュ」や「ビットコインゴールド」などは、ビットコインからのハードフォークによって誕生した全く異なる暗号通貨である。

A.8.2 PoS

PoW は実装がシンプルであり、ブロックの改ざんは非常に難しい。加えて、インセンティブ・システムとして公平であるため、堅牢な仕組みと言える。しかし、大半のノードのハッシュ演算が無駄になったり、システムが大規模になるほど大きい演算能力が必要となるなど、スケーラビリティに乏しい点が課題とされている。

そこで、演算能力ではなく、コインを多く持っているノードのマイニングの難易度を下げる、或いはマイニングをすること無くブロックを繋げる権限を付与するアルゴリズムである、PoS（Proof of Stake）も提案されている。PoS は「イーサリアム」への実装が予定されており、ハッシュ演算の大半を削減できることが期待されている。

PoS は「多くのコイン（利害関係）を持つノードは不正をしない可能性が高い」という仮定に基づいており、PoW とは大きく異なるアルゴリズムである。計算資源を有効活用できる一方で、PoW より堅牢でないといった課題もある。

付録２　原理実証実験用プログラムの解説

付録２では実証実験に使用したプログラムの簡単な解説を行う。尚、本システムはネットワークを介した非同期システムであり、内部では乱数を使用しているため、環境によっては本論文で提示した結果と異なる場合がある。

B.1　原理実証実験用プログラムの処理

プログラム実行の流れを図に示した。まず、MetaChainSystem.py を実行すると、実験回数が保存された iter.csv を読み出し、「files 実験回数」というディレクトリを作成し、実験回数を 1 増やして iter.csv を閉じる。その後 IP アドレスや ID、ポートの設定など、p2p の初期化を行なった後、システムは待ち状態となる。そして、別のコンピュータで実行された tx_generator.py から送られたシステム開始合図に応じてマイニングを始める。自分がゴールデンナンスを見つければ、繋げたブロックをブロードキャストし、ブロックハッシュと生成時間をメタチェーンに保存し、ブロック本体は randomize()の結果に応じて保存する。他のノードが先にゴールデンナンスを見つければ、マイニングは中断され、受け取ったブロックの検証作業を行う。そして、ブロックハッシュをメタチェーンに追加し、同様に randomize()の結果に応じてブロックを保存する。

また、tx_generator.py から送られたトランザクションは、受信するたびに検証され、trans.txt に保存される。マイニング中に発生したトランザクションは pending_trans.txt に保存され、マイニング終了のタイミングで trans.txt に移される。

以上の全ての動作の最中で、p2p 基盤部分は一定時間おきに生存確認パケットを送受信し、接続を確かめ合う。もし、ネットワークから外れたノードがいれば、ノードをリストから除外する。本システムではノードの新規加入や離脱を想定していないため、ノードがネットワーク障害などにより、システムから離脱した場合、システムは強制終了する仕様となっている。

また、tx_generator.py は MetaChainSystem.py と同様の p2p 基盤を持つが、処理はトランザクションを一定時間おきに生成し、ブロードキャストするのみである。

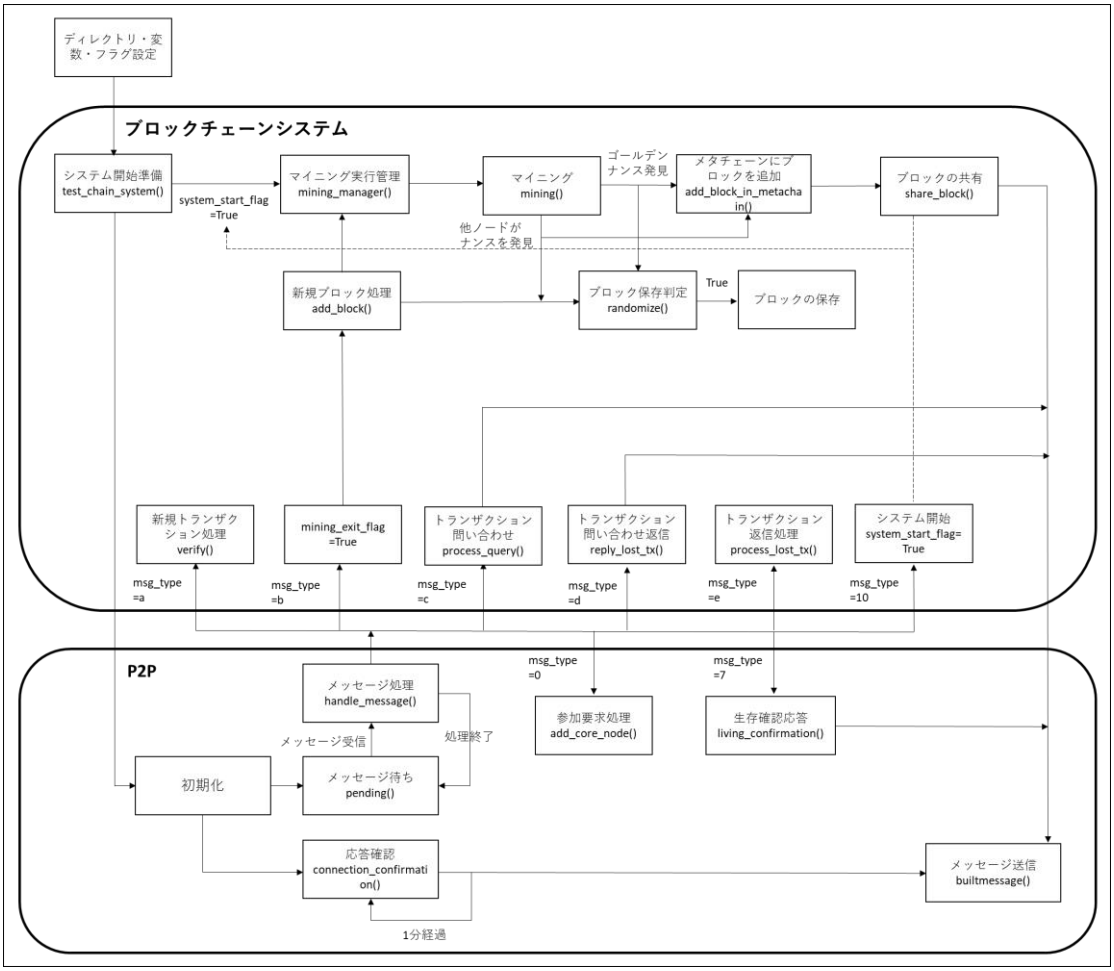

図 18　実証実験用プログラムの処理の流れ

B.2　実験手順

原理実証実験では次のようにプログラムを実行した。

まず、ラズベリーパイ 4 台に MetaChainSystem.py と iter.csv を配布し、同じディレクトリに置く。次に、別に用意したコンピュータから SSH 接続を行い、２つのファイルがあるディレクトリを開き、ラズベリーパイ 4 台それぞれで MetaChainSystem.py を実行する。次に、SSH 接続を行なっている本体のコンピュータに、予め 4 台のラズベリーパイの情報が格納された peer_list.txt と tx_generator.py を同じディレクトリに起き、tx_generator.py を実行する。tx_generator.py から 4 台のラズベリーパイ全てにシステム開始の合図が送られるため、その後は予め MetaChainSystem.py にハードコードされたシステム終了ブロック数に到達するまで待機する。

B.3　ソースコード

ソースコードは以下の通り。

ソースコード 1　MetaChainSystem.py

```
1 #モジュール類のインポート
2 import base58
3 import ecdsa
4 import filelock
5 import json
6 import time
7 from time import sleep
8 import datetime
9 import threading
10 import filelock
11 import hashlib
12 import re
13 import sys
14 from concurrent.futures import ThreadPoolExecutor
15 import socket
16 import os
17 import pickle
18 import random
19 import csv
20
21 #事件結果を保存するディレクトリの設定
22 iteration_file_name = 'iter.csv'
23 with open(iteration_file_name) as f:
```

```
24        a = list(csv.reader(f))
25 n = int(a[-1][0])
26 b = [n+1]
27 with open(iteration_file_name, 'a', newline='') as f:
28        writer = csv.writer(f)
29        writer.writerow(b)
30
31 DIR = 'files' + str(n) + '/'
32 os.mkdir(DIR)
33
34 #各種ファイル名の設定
35 key_file_name1 = DIR + "key.lock"
36 key_file_name2 = DIR + "key.txt"
37 trans_file_name1 = DIR + "trans.lock"
38 trans_file_name2 = DIR + "trans.txt"
39 block_file_name1 = DIR + "block.lock"
40 block_file_name2 = DIR + "block.txt"
41 pending_trans_file_name1 = DIR + 'pending_trans.lock'
42 pending_trans_file_name2 = DIR + 'pending_trans.txt'
43 meta_chain_file_name1 = DIR + "MetaChain.lock"
44 meta_chain_file_name2 = DIR + "MetaChain.txt"
45 peer_list_file_name1 = DIR + 'peer_list.lock'
46 peer_list_file_name2 = DIR + 'peer_list.txt'
47
48 #パラメータの設定
49 k = 3
50 system_stop_point = 25
51 num_of_tx_to_store = 20
52 num_of_node = 4
53
54 #内部パラメータの初期化
55 num_of_new_tx = 0
56 num_of_pending_tx = 0
57 num_of_blocks = 0
58
59 #内部フラグの初期化
```

```
60 mining_exit_flag = False
61 mining_flag = False
62 pending_tx = True
63 system_start_flag = False
64 pending_tx = False
65 request_OK_flag = False
66 system_stop_flag = False
67 node_list = []
68
69 #システム開始準備
70 def start_test_chain_system():
71     global system_start_flag
72     global request_OK_flag
73
74     print("Waiting for setting up tx generator....")
75     p2p_start()
76     while True:
77         sleep(1)
78         if system_start_flag == True:
79             print("Start system")
80             sleep(15)
81             break
82
83     mining_manager()
84
85 #鍵の生成
86 def generate_key():
87     global key_file_name1
88     global key_file_name2
89
90     private_key = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
91     public_key = private_key.get_verifying_key()
92
93     private_key = private_key.to_string()
94     public_key = public_key.to_string()
95
```

```
96      private_b58 = base58.b58encode(private_key).decode('ascii')
97      public_b58 = base58.b58encode(public_key).decode('ascii')
98
99
100     with filelock.FileLock(key_file_name1, timeout=10):
101         try:
102             with open(key_file_name2, 'r') as file:
103                 key_list = json.load(file)
104         except:
105             key_list = []
106
107         key_list.append({
108             'private': private_b58,
109             'public' : public_b58
110         })
111
112         with open(key_file_name2, 'w') as file:
113             json.dump(key_list, file, indent=2)
114     return public_b58, private_b58
115
116 #ブロックの検証
117 def verify_block(block):
118     sha = hashlib.sha256()
119     sha.update(bytes(block['nonce']))
120     sha.update(bytes.fromhex(block['previous_hash']))
121     sha.update(bytes.fromhex(block['tx_hash']))
122     hash = sha.digest()
123
124     if hash.hex() == block['hash']:
125         return True
126     else:
127         return False
128
129 #ブロックの保存
130 def add_block(block,tx_id_list,block_maker_ID):
131     global num_of_new_tx
```

```
132     global num_of_blocks
133     global system_stop_point
134     global mining_exit_flag
135     global pending_tx
136     global block_file_name1
137     global block_file_name2
138     global trans_file_name1
139     global trans_file_name2
140     global tx_generator_ip
141     global tx_generator_port
142     global reqester_ip
143     global reqester_port
144     process_pending_tx()
145     if verify_block(block) == True and randomize() == True:
146         add_block_in_metachain(block)
147         with filelock.FileLock(block_file_name1, timeout=10):
148             try:
149                 with open(block_file_name2, 'r') as file:
150                     block_list = json.load(file)
151             except:
152                 block_list = []
153
154         with filelock.FileLock(trans_file_name1, timeout=10):
155             try:
156                 with open(trans_file_name2, 'r') as file:
157                     file_tx_list = json.load(file)
158             except:
159                 file_tx_list = []
160
161         unprosessed_tx_list = []
162         tx_list = []
163         #省略 ID
164         file_tx_id_list = []
165         lost_tx_list = []
166
167         for tx in file_tx_list:
```

```
168                 if tx['TxID'][:10] in tx_id_list:
169                     tx_list.append(tx)
170                     file_tx_id_list.append(tx['TxID'][:10])
171                 else:
172                     unprosessed_tx_list.append(tx)
173
174             if int(block['size']) != len(tx_list):
175                 for TxID in tx_id_list:
176                     if TxID not in file_tx_id_list:
177                         require_lost_tx(TxID,block_maker_ID)
178
179             block["tx"] = tx_list
180             block_list.append(block)
181
182             with filelock.FileLock(block_file_name1,timeout=10):
183                 with open(block_file_name2, 'w') as file:
184                     json.dump(block_list, file, indent=2)
185
186             with filelock.FileLock(trans_file_name1,timeout=10):
187                 with open(trans_file_name2,"w") as file:
188                     json.dump(unprosessed_tx_list,file,indent=2)
189
190             num_of_new_tx = len(unprosessed_tx_list)
191             num_of_blocks += 1
192             print(num_of_blocks,"th block has made.")
193             mining_exit_flag = False
194         else:
195             add_block_in_metachain(block)
196             #with filelock.FileLock(block_file_name1, timeout=10):
197             #     try:
198             #         with open(block_file_name2, 'r') as file:
199             #             block_list = json.load(file)
200             #     except:
201             #         block_list = []
202
203             with filelock.FileLock(trans_file_name1, timeout=10):
```

```
204                 try:
205                     with open(trans_file_name2, 'r') as file:
206                         file_tx_list = json.load(file)
207                 except:
208                     file_tx_list = []
209
210             unprosessed_tx_list = []
211             tx_list = []
212             #省略 ID
213             file_tx_id_list = []
214             lost_tx_list = []
215
216             for tx in file_tx_list:
217                 if tx['TxID'][:10] in tx_id_list:
218                     tx_list.append(tx)
219                     file_tx_id_list.append(tx['TxID'][:10])
220                 else:
221                     unprosessed_tx_list.append(tx)
222
223             if int(block['size']) > len(tx_list):
224                 for TxID in tx_id_list:
225                     if TxID not in file_tx_id_list:
226                         require_lost_tx(TxID,block_maker_ID)
227
228             with filelock.FileLock(trans_file_name1,timeout=10):
229                 with open(trans_file_name2,"w") as file:
230                     json.dump(unprosessed_tx_list,file,indent=2)
231
232             num_of_new_tx = len(unprosessed_tx_list)
233             num_of_blocks += 1
234             print(num_of_blocks,"th block has made.")
235             mining_exit_flag = False
236
237         if num_of_blocks == system_stop_point:
238             print("Number of block has reached system to stop value.")
```

```
239             build_message(100,ID,tx_generator_ip,tx_generator_port,host,port,0,0,0,0,
0,0,0)
240             build_message(100,ID,reqester_ip,reqester_port,host,port,0,0,0,0,0,0,0)
241             exit()
242
243 #自身のトランザクションプールにないトランザクションの要求
244 def require_lost_tx(TxID,block_maker_ID):
245     global peer_list_file_name1
246     global peer_list_file_name2
247     global ID
248     with filelock.FileLock(peer_list_file_name1,timeout=10):
249         try:
250             with open(peer_list_file_name2,'r') as file:
251                 peer_list = json.load(file)
252         except:
253             peer_list = []
254     #print("Sending Request of lost tx")
255     for peer in peer_list:
256         if ID != peer:
257             build_message("d",ID,peer_list[peer]['IP address'],peer_list[peer]['Po
rt'],host,port,TxID,0,0,0,0,0,0)
258
259 #要求されたトランザクションの返信
260 def reply_lost_tx(TxID,reqester_ID):
261     global block_file_name1
262     global block_file_name2
263     global peer_list_file_name1
264     global peer_list_file_name2
265     global ID
266
267     with filelock.FileLock(block_file_name1, timeout=10):
268         try:
269             with open(block_file_name2, 'r') as file:
270                 block_list = json.load(file)
271         except:
272             block_list = []
```

```
273
274     with filelock.FileLock(peer_list_file_name1,timeout=10):
275         try:
276             with open(peer_list_file_name2,'r') as file:
277                 peer_list = json.load(file)
278         except:
279             peer_list = []
280
281     for block in block_list:
282         for tx in block['tx']:
283             height = block['height']
284             if TxID == tx['TxID'][:10]:
285                 build_message('e',ID,peer_list[reqester_ID]['IP address'],peer_lis
t[reqester_ID]['Port'],host,port,tx['TxID'],"new tx",tx["time"],tx["data"],tx["sig"],tx["pu
blisher"],height)
286                 break
287
288 #返信されたトランザクションの処理
289 def process_lost_tx(tx,height):
290     global block_file_name1
291     global block_file_name2
292     global trans_file_name1
293     global trans_file_name2
294
295     with filelock.FileLock(block_file_name1, timeout=10):
296         try:
297             with open(block_file_name2, 'r') as file:
298                 block_list = json.load(file)
299         except:
300             block_list = []
301
302     with filelock.FileLock(trans_file_name1, timeout=10):
303         try:
304             with open(trans_file_name2, 'r') as file:
305                 file_tx_list = json.load(file)
306         except:
```

```
307                 file_tx_list = []
308
309         tx_list = []
310         for tx_ in file_tx_list:
311             if tx['TxID'] != tx_['TxID']:
312                 tx_list.append(tx_)
313
314         for block in block_list:
315             if block['height'] == height:
316                 if tx not in block['tx']:
317                     block['tx'].append(tx)
318             elif block['height'] > height:
319                 break
320
321         with filelock.FileLock(block_file_name1,timeout=10):
322             with open(block_file_name2, 'w') as file:
323                 json.dump(block_list, file, indent=2)
324
325         with filelock.FileLock(trans_file_name1,timeout=10):
326             with open(trans_file_name2,"w") as file:
327                 json.dump(tx_list,file,indent=2)
328
329 #ブロック保存の判定
330 def randomize():
331         global k
332         global num_of_node
333         global ID
334
335         seed = int(ID[1:])
336         random.seed(seed+time.time())
337
338         if k >= random.randint(1,num_of_node):
339             print("I saved the block.")
340             return True
341         else:
342             False
```

```
343
344 #メタチェーンへハッシュ値を格納
345 def add_block_in_metachain(block):
346     global meta_chain_file_name1
347     global meta_chain_file_name2
348
349     with filelock.FileLock(meta_chain_file_name1, timeout=10):
350         try:
351             with open(meta_chain_file_name2, 'r') as file:
352                 meta_block_list = json.load(file)
353         except:
354             meta_block_list = []
355
356     meta_block_list.append({
357         'hash' : block['hash'],
358         'timestamp':block['timestamp']
359     })
360
361     with filelock.FileLock(meta_chain_file_name1,timeout=10):
362         with open(meta_chain_file_name2, 'w') as file:
363             json.dump(meta_block_list, file, indent=2)
364
365 #トランザクションの検証
366 def verify(new_tx_id,new_tx):
367     global mining_flag
368     global pending_tx
369     global num_of_new_tx
370     global num_of_pending_tx
371     global num_of_tx_to_store
372     global trans_file_name1
373     global trans_file_name2
374     global pending_trans_file_name1
375     global pending_trans_file_name2
376
377     if mining_flag == True:
378         verify_pending_tx(new_tx)
```

```
379     elif mining_flag == False:
380         with filelock.FileLock(trans_file_name1, timeout=10):
381             try:
382                 with open(trans_file_name2, 'r') as file:
383                     file_tx_list = json.load(file)
384             except:
385                 file_tx_list = []
386         tx_publisher = bytes.fromhex(new_tx['publisher'])
387         time = new_tx['time']
388         data = new_tx['data']
389         tx_sig = bytes.fromhex(new_tx['sig'])
390 
391         sha = hashlib.sha256()
392         sha.update(tx_publisher)
393         sha.update(time.encode())
394         sha.update(data.encode())
395         hash = sha.digest()
396         key = ecdsa.VerifyingKey.from_string(tx_publisher, curve=ecdsa.SECP25
6k1)
397         if key.verify(tx_sig, hash) == True:
398             file_tx_list.append(new_tx)
399 
400         with filelock.FileLock(trans_file_name1, timeout=10):
401             with open(trans_file_name2,"w") as file:
402                 json.dump(file_tx_list, file, indent=2)
403         num_of_new_tx += 1
404 
405         print("Peer made ",num_of_new_tx,"th tx.")
406 
407 #マイニング中に発生したトランザクションの処理
408 def process_pending_tx():
409     global pending_tx
410     global num_of_new_tx
411     global num_of_pending_tx
412     global pending_trans_file_name1
413     global pending_trans_file_name2
```

```
414     global trans_file_name1
415     global trans_file_name2
416
417     with filelock.FileLock(trans_file_name1, timeout=10):
418         try:
419             with open(trans_file_name2, 'r') as file:
420                 file_tx_list = json.load(file)
421         except:
422             file_tx_list = []
423
424     with filelock.FileLock(pending_trans_file_name1, timeout=10):
425         try:
426             with open(pending_trans_file_name2, 'r') as file:
427                 file_pending_tx_list = json.load(file)
428         except:
429             file_pending_tx_list = []
430
431     for tx in file_pending_tx_list:
432         file_tx_list.append(tx)
433
434     with filelock.FileLock(trans_file_name1, timeout=10):
435         with open(trans_file_name2,"w") as file:
436             json.dump(file_tx_list, file, indent=2)
437
438     with filelock.FileLock(pending_trans_file_name1, timeout=10):
439         with open(pending_trans_file_name2,"w") as file:
440             json.dump([], file, indent=2)
441
442     pending_tx = False
443     num_of_new_tx += num_of_pending_tx
444     print(num_of_pending_tx,"Pending tx has processed.")
445     num_of_pending_tx = 0
446
447 #マイニング中に発生したトランザクションの検証
448 def verify_pending_tx(new_tx):
449     global pending_tx
```

```
450     global num_of_pending_tx
451     global pending_trans_file_name1
452     global pending_trans_file_name2
453
454     with filelock.FileLock(pending_trans_file_name1, timeout=10):
455         try:
456             with open(pending_trans_file_name2, 'r') as file:
457                 file_tx_list = json.load(file)
458         except:
459             file_tx_list = []
460
461     tx_publisher = bytes.fromhex(new_tx['publisher'])
462     time = new_tx['time']
463     data = new_tx['data']
464     tx_sig = bytes.fromhex(new_tx['sig'])
465
466     sha = hashlib.sha256()
467     sha.update(tx_publisher)
468     sha.update(time.encode())
469     sha.update(data.encode())
470     hash = sha.digest()
471
472     key = ecdsa.VerifyingKey.from_string(tx_publisher, curve=ecdsa.SECP256k1)
473     if key.verify(tx_sig, hash) == True:
474         file_tx_list.append(new_tx)
475
476     with filelock.FileLock(pending_trans_file_name1,timeout=10):
477         with open(pending_trans_file_name2, 'w') as file:
478             file_tx_list = json.dump(file_tx_list,file,indent=2)
479     pending_tx = True
480     num_of_pending_tx += 1
481     print("Peer made ",num_of_pending_tx,"th pending tx.")
482
483 #マイニングの実行
484 def mining():
485     global mining_flag
```

```
    global mining_exit_flag
    global pending_tx
    global num_of_blocks
    global system_stop_point
    global num_of_new_tx
    global block_file_name1
    global block_file_name2
    global trans_file_name1
    global trans_file_name2
    global tx_generator_ip
    global tx_generator_port
    global reqester_ip
    global reqester_port
    global meta_chain_file_name1
    global meta_chain_file_name2
    global num_of_pending_tx

    process_pending_tx()
    print('-----Start mining-----')

    DIFFICULTY = 4
    public_key = generate_key()[0]

    with filelock.FileLock(block_file_name1, timeout=10):
        try:
            with open(block_file_name2, 'r') as file:
                block_list = json.load(file)
            previous_hash = block_list[-1]['hash']
        except:
            block_list = []

    with filelock.FileLock(meta_chain_file_name1, timeout=10):
        try:
            with open(meta_chain_file_name2, 'r') as file:
                meta_block_list = json.load(file)
                previous_hash = meta_block_list[-1]['hash']
```

```
522             except:
523                 meta_block_list = []
524                 previous_hash = ''
525
526         with filelock.FileLock(trans_file_name1, timeout=10):
527             try:
528                 with open(trans_file_name2, 'r') as file:
529                     file_tx_list = json.load(file)
530             except:
531                 file_tx_list = []
532
533         tx_id_list = []
534         tx_list = []
535
536         sha = hashlib.sha256()
537         for tx in file_tx_list:
538             tx_id_list.append(tx['TxID'][:10])
539             sha.update(bytes.fromhex(tx['TxID']))
540             tx_list.append(tx)
541             if len(tx_list) == num_of_tx_to_store:
542                 break
543
544         tmp = file_tx_list
545         file_tx_list = []
546
547         for tx in tmp:
548             if tx['TxID'][:10] not in tx_id_list:
549                 file_tx_list.append(tx)
550
551         tx_hash = sha.digest()
552         for nonce in range(random.randint(0,99999),100000000):
553             if mining_exit_flag == True:
554                 mining_exit_flag = False
555                 mining_flag = False
556                 return
557             sha = hashlib.sha256()
```

```
558             sha.update(bytes(nonce))
559             sha.update(bytes.fromhex(previous_hash))
560             sha.update(tx_hash)
561             hash = sha.digest()
562
563             if re.match(r'0{' + str(DIFFICULTY) + r'}', hash.hex()):
564                 break
565
566         block_list.append({
567             'hash' : hash.hex(),
568             'timestamp':str(datetime.datetime.now()),
569             'nonce': nonce,
570             'previous_hash': previous_hash,
571             'tx_hash': tx_hash.hex(),
572             'height': num_of_blocks,
573             'size':len(tx_list),
574             'tx'    : tx_list
575         })
576
577         if randomize() == True:
578             with filelock.FileLock(block_file_name1, timeout=10):
579                 with open(block_file_name2, 'w') as file:
580                     json.dump(block_list, file, indent=2)
581
582         with filelock.FileLock(trans_file_name1, timeout=10):
583             with open(trans_file_name2, 'w') as file:
584                 json.dump(file_tx_list, file, indent=2)
585         share_block(block_list[-1],tx_id_list)
586         add_block_in_metachain(block_list[-1])
587
588         num_of_blocks += 1
589         mining_flag = False
590         num_of_new_tx -= len(tx_list)
591         print('-----End mining-----')
592         print("found golden nonce!",nonce)
593         print(num_of_blocks,"th blocks has made.")
```

```
594
595         if num_of_blocks == system_stop_point:
596             print("Number of block has reached system to stop value.")
597             build_message(100,ID,tx_generator_ip,tx_generator_port,host,port,0,0,0,0,
0,0,0)
598             build_message(100,ID,reqester_ip,reqester_port,host,port,0,0,0,0,0,0,0)
599             exit()
600         process_pending_tx()
601
602 #生成したブロックのブロードキャスト
603 def share_block(block,tx_id_list):
604         global ID
605         global peer_list_file_name1
606         global peer_list_file_name2
607         with filelock.FileLock(peer_list_file_name1,timeout=10):
608             try:
609                 with open(peer_list_file_name2,'r') as file:
610                     peer_list = json.load(file)
611             except:
612                 peer_list = []
613         #def build_message(msg_type,ID,addr_to,port_to,my_addr,my_port,sp1,sp2,sp
3,sp4,sp5,sp6):
614         for peer in peer_list:
615             if peer != ID:
616                 build_message('b',ID,peer_list[peer]['IP address'],peer_list[peer]['Po
rt'],host,port,block['hash'],block['nonce'],block['previous_hash'],block['tx_hash'],tx_id_li
st,block['height'],block['timestamp'])
617
618 #マイニング実行管理
619 def mining_manager():
620         global mining_flag
621         global num_of_new_tx
622         global system_stop_flag
623         global num_of_tx_to_store
624         global num_of_blocks
625         global system_stop_point
```

```
626
627     print("mining manager start...")
628     while True:
629         sleep(1)
630         if mining_flag == False and num_of_blocks <= system_stop_point:
631             mining_flag = True
632             mining_exit_flag = False
633             mining()
634         elif system_stop_flag == True or num_of_blocks > system_stop_point:
635             exit()
636         else:
637             sleep(3)
638
639 #トランザクションの参照要求処理
640 def process_query(TxID,requester_ip,requester_port):
641     global block_file_name1
642     global block_file_name2
643
644     with filelock.FileLock(block_file_name1, timeout=10):
645         try:
646             with open(block_file_name2, 'r') as file:
647                 block_list = json.load(file)
648             previous_hash = block_list[-1]['hash']
649         except:
650             block_list = []
651
652     for block in block_list:
653         for tx in block['tx']:
654             if tx['TxID'] == TxID:
655                 build_message('d',ID,requester_ip,requester_port,host,port,tx['Tx
ID'],tx['time'],tx['publisher'],tx['data'],tx['sig'],0,0)
656                 return
657     build_message('d',ID,requester_ip,requester_port,host,port,0,0,0,0,0,0,0)
658
659 #P２Pシステムの初期化
660 def p2p_start():
```

```
661     global host
662     global port
663     global ID
664     global my_socket
665     host = get_myip()
666     my_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
667     port = port_set()
668     ID = "N" + host[-2:] + str(port)[-2:]
669     initialize_peer_list(ID,host,port)
670     show_my_info()
671     print(ID)
672
673     #他のノードからのリクエストを常に待機する
674     accepting_request = threading.Thread(target=pending)
675     accepting_request.setDaemon(True)
676     accepting_request.start()
677
678     #20 秒置きに peer の生存確認を行う
679     conn_confirmation = threading.Thread(target=connection_confirmation)
680     conn_confirmation.setDaemon(True)
681     conn_confirmation.start()
682
683 #ポートが使用されているかどうかの判定
684 def port_check(port):
685     try:
686         my_socket.bind((host,port))
687         return 1
688     except:
689         return 0
690
691 #ポートの設定
692 def port_set():
693     port = 50030
694     while True:
695         if port_check(port) == 1:
696             return port
```

```
697             else:
698                 port += 1
699
700 #自分の IP アドレスの取得
701 def get_myip():
702     s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
703     s.connect(('8.8.8.8',80))
704     return s.getsockname()[0]
705
706 #メッセージの処理
707 def pending():
708     global system_stop_flag
709     my_socket.listen(10)
710     while True:
711         conn, addr = my_socket.accept()
712         handle_message(conn)
713         if system_stop_flag == True:
714             exit()
715
716 #ノードリストの初期化
717 def initialize_peer_list(ID,host,port):
718     global peer_list_file_name1
719     global peer_list_file_name2
720     edge_node_list = {}
721     my_info = {
722                 ID:{
723                         "time":time.ctime(),
724                         "IP address":host,
725                         "Port":port}
726                 }
727
728     with filelock.FileLock(peer_list_file_name1,timeout=10):
729         try:
730             with open(peer_list_file_name2,'w') as file:
731                 json.dump(my_info,file,indent=2)
732         except:
```

```
733                 exit()
734
735 #自分の IP アドレス、ポート番号、ID の表示
736 def show_my_info():
737     print("Your address is :",host)
738     print("Your port is :",port)
739     print("Your ID is:",ID)
740
741 #受信メッセージの処理
742 def handle_message(message):
743     global host
744     global port
745     global node_list
746     global system_start_flag
747     global mining_exit_flag
748     global mining_flag
749     global request_OK_flag
750     global tx_generator_ip
751     global tx_generator_port
752     global reqester_ip
753     global reqester_port
754
755     conn = message
756     msg = conn.recv(4096)
757
758     msg = json.loads(msg)
759     msg_type = msg['msg_type']
760     peer_ID = msg['ID']
761     peer_addr = msg['addr']
762     peer_port = msg['port']
763
764     if msg_type == 0:
765         request_OK_flag = True
766         share_node_list(peer_addr,peer_port,peer_ID)
767         add_core_node(peer_addr,peer_port,peer_ID)
768     elif msg_type == 1:
```

```
769             print("Request for connection was called.")
770             add_core_node(peer_addr,peer_port,peer_ID)
771             build_message(0,ID,peer_addr,peer_port,host,port,0,0,0,0,0,0,0)
772         elif msg_type == 2:
773             renew_core_list(msg["sp1"])
774         elif msg_type == 3:
775             living_confirmation(peer_addr,peer_port)
776         elif msg_type == 4:
777             node_list.append(peer_ID)
778         elif msg_type == 6:
779             send_core_node_list(peer_addr,peer_port)
780         elif msg_type == 7:
781             living_confirmation(peer_addr,peer_port)
782         elif msg_type == 8:
783             node_list.append(peer_ID)
784         elif msg_type == 10:
785             system_start_flag = True
786         elif msg_type == 11:
787             print("New peer added.")
788             add_core_node(msg["sp1"],msg["sp2"],msg["sp3"])
789         elif msg_type == 'a':
790             tx_generator_ip = peer_addr
791             tx_generator_port = peer_port
792             tx = {"TxID":msg["sp1"],
793                   "time":msg["sp3"],
794                   "publisher":msg["sp6"],
795                   "data":msg["sp4"],
796                   "sig":msg["sp5"]}
797             verify(msg["sp1"],tx)
798         elif msg_type == 'b':
799             mining_exit_flag = True
800             print('-----End mining-----')
801             print(peer_ID,"has found golden nonce,",msg['sp2'])
802             block = {"hash":msg['sp1'],
803                      'timestamp':msg['sp7'],
804                      'nonce':msg['sp2'],
```

```
805                         'previous_hash':msg['sp3'],
806                         'tx_hash':msg['sp4'],
807                         'height':msg['sp6'],
808                         'size':len(msg['sp5']),
809                         'tx':[]}
810             add_block(block,msg['sp5'],peer_ID)
811         elif msg_type == 'c':
812             reqester_ip = peer_addr
813             reqester_port = peer_port
814             process_query(msg['sp1'],peer_addr,peer_port)
815         elif msg_type == 'd':
816             reply_lost_tx(msg['sp1'],peer_ID)
817         elif msg_type == 'e':
818             tx = {"TxID":msg["sp1"],
819                   "time":msg["sp3"],
820                   "publisher":msg["sp6"],
821                   "data":msg["sp4"],
822                   "sig":msg["sp5"]}
823             process_lost_tx(tx,msg["sp7"])
824
825 #生存確認への返信
826 def living_confirmation(peer_addr,peer_port):
827         build_message(8,ID,peer_addr,peer_port,host,port,0,0,0,0,0,0,0)
828
829 #生存確認の送信
830 def connection_confirmation():
831         global node_list
832         global ID
833         global host
834         global port
835         global peer_list_file_name1
836         global peer_list_file_name2
837         global system_stop_flag
838
839         while True:
840             if system_stop_flag == True:
```

```
841                 exit()
842             sleep(30)
843             node_list = []
844             #print("Start connection confirmation...")
845             peer_list = []
846             remove_list = []
847             is_change = False
848             with filelock.FileLock(peer_list_file_name1,timeout=5):
849                 try:
850                     with open(peer_list_file_name2,'r') as file:
851                         peer_list = json.load(file)
852                 except:
853                     peer_list = []
854
855             for peer in peer_list:
856                 if peer != ID:
857                     try:
858                         build_message(7,ID,peer_list[peer]['IP address'],peer_list[p
eer]['Port'],host,port,0,0,0,0,0,0,0)
859                     except:
860                         pass
861             sleep(30)
862             node_list.append(ID)
863
864             for peer in peer_list:
865                 if peer in node_list:
866                     continue
867                 else:
868                     remove_list.append(peer)
869                     print("peer ",peer," was disconnected.")
870                     is_change = True
871
872             if is_change:
873                 print("There was a chenge in network topology.")
874                 remove_node(remove_list)
875
```

```
876 #メッセージの組み立て
877 def build_message(msg_type,ID,addr_to,port_to,my_addr,my_port,sp1,sp2,sp3,sp4,
sp5,sp6,sp7):
878     message = {'msg_type':msg_type,
879                'ID':ID,
880                'addr':my_addr,
881                'port':my_port,
882                'sp1':sp1,
883                'sp2':sp2,
884                'sp3':sp3,
885                'sp4':sp4,
886                'sp5':sp5,
887                'sp6':sp6,
888                'sp7':sp7}
889     message = json.dumps(message)
890     my_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
891     my_socket.connect((addr_to,port_to))
892     my_socket.sendall(message.encode('utf-8'))
893
894 #ノードリストに新規ノードを追加
895 def add_core_node(peer_addr,peer_port,peer_ID):
896     global peer_list_file_name1
897     global peer_list_file_name2
898     add_info = {
899         peer_ID:{
900                  "time":time.ctime(),
901                  "IP address":peer_addr,
902                  "Port":peer_port}
903             }
904
905     with filelock.FileLock(peer_list_file_name1,timeout=10):
906         try:
907             with open(peer_list_file_name2,'r') as file:
908                 peer_list = json.load(file)
909         except:
910             peer_list = []
```

```
911             peer_list.update(add_info)
912
913             with open(peer_list_file_name2,'w') as file:
914                 json.dump(peer_list,file,indent=2)
915
916 #更新されたノードリストの共有
917 def share_node_list(new_addr,new_port,new_ID):
918     global peer_list_file_name1
919     global peer_list_file_name2
920     with filelock.FileLock(peer_list_file_name1,timeout=10):
921         try:
922             with open(peer_list_file_name2,'r') as file:
923                 peer_list = json.load(file)
924         except:
925             peer_list = []
926
927     for peer in peer_list:
928         if peer != new_ID and peer != ID:
929             build_message(11,ID,peer_list[peer]['IP address'],peer_list[peer]['Po
rt'],host,port,new_addr,new_port,new_ID,0,0,0,0)
930         if peer != ID:
931             build_message(11,ID,new_addr,new_port,host,port,peer_list[peer]["IP
 address"],peer_list[peer]['Port'],peer,0,0,0,0)
932
933 #切断されたノードをノードリストから除外
934 def remove_node(remove_list):
935     global peer_list_file_name1
936     global peer_list_file_name2
937     with filelock.FileLock(peer_list_file_name1,timeout=10):
938         try:
939             with open(peer_list_file_name2,'r') as file:
940                 peer_list = json.load(file)
941         except:
942             peer_list = []
943
944     for peer in remove_list:
```

```
945             peer_list.pop(peer)
946         with filelock.FileLock(peer_list_file_name1,timeout=10):
947             with open(peer_list_file_name2,'w') as file:
948                 json.dump(peer_list,file,indent=2)
949
950 #ノードリストを更新
951 def renew_core_list(new_peer_list):
952     global peer_list_file_name1
953     global peer_list_file_name2
954     with filelock.FileLock(peer_list_file_name1,timeout=10):
955         try:
956             with open(peer_list_file_name2,'r') as file:
957                 peer_list = json.load(file)
958         except:
959             exit()
960         peer_list.update(new_peer_list)
961     with filelock.FileLock(peer_list_file_name1,timeout=10):
962         with open(peer_list_file_name2,'w') as file:
963             json.dump(peer_list,file,indent=2)
964
965 start_test_chain_system()
```

ソースコード 2　tx_generator.py

```
1 import base58
2 import ecdsa
3 import filelock
4 import json
5 import time
6 from time import sleep
7 import datetime
8 import threading
9 import filelock
10 import hashlib
11 import re
12 import sys
13 from concurrent.futures import ThreadPoolExecutor
```

```
14 import socket
15 import os
16 import pickle
17 import random
18 import csv
19
20 #Set internal variables
21 num_of_new_tx = 0
22 num_of_pending_tx = 0
23 num_of_blocks = 0
24 system_stop_point = 10
25 num_of_tx_to_store = 11
26 count = 0
27
28 #Set system flags
29 system_start_flag = False
30 request_OK_flag = False
31 system_stop_flag = False
32 node_list = []
33
34 iteration_file_name = 'iter2.csv'
35 with open(iteration_file_name) as f:
36     a = list(csv.reader(f))
37 n = int(a[-1][0])
38 b = [n+1]
39 with open(iteration_file_name, 'a', newline='') as f:
40     writer = csv.writer(f)
41     writer.writerow(b)
42
43 DIR = "./files" + str(n) + "/"
44 try:
45     os.mkdir(DIR)
46 except:
47     pass
48
49 key_file_name1 = DIR + "key.lock"
```

```
50 key_file_name2 = DIR + "key.txt"
51 trans_id_file_name1 = DIR + "trans_id.lock"
52 trans_id_file_name2 = DIR + "trans_id.txt"
53 peer_list_file_name1 = DIR + 'peer_list.lock'
54 peer_list_file_name2 = DIR + 'peer_list.txt'
55
56
57 def start_test_chain_system():
58     print("Start tx generator....")
59     p2p_start()
60     prepare()
61
62     #トランザクションの生成のみを行う
63     generate_tx_ = threading.Thread(target=generate_tx_manager)
64     generate_tx_.start()
65
66     sleep(10)
67
68 def prepare():
69     global system_start_flag
70     global request_OK_flag
71     global peer_list_file_name1
72     global peer_list_file_name2
73
74     print("Setting up transaction generator...")
75
76     with filelock.FileLock('peer_list.lock',timeout=10):
77         try:
78             with open('peer_list.txt','r') as file:
79                 peer_list = json.load(file)
80         except:
81             peer_list = []
82
83     for peer in peer_list:
84         if peer != ID:
85             #build_message(1,ID,addr_,port_,host,port,0,0,0,0,0,0)
```

```
86                  try:
87                      build_message(1,ID,peer_list[peer]['IP address'],peer_list[peer]['
Port'],host,port,0,0,0,0,0,0,0,0)
88                  except:
89                      build_message(1,ID,peer_list[peer]['IP address'],peer_list[peer]['
Port']+1,host,port,0,0,0,0,0,0,0,0)
90
91      send_system_start_signal()
92      print("Start system")
93
94 def generate_key():
95      global key_file_name1
96      global key_file_name2
97
98      private_key = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
99      public_key = private_key.get_verifying_key()
100
101      private_key = private_key.to_string()
102      public_key = public_key.to_string()
103
104      private_b58 = base58.b58encode(private_key).decode('ascii')
105      public_b58 = base58.b58encode(public_key).decode('ascii')
106
107
108      with filelock.FileLock(key_file_name1, timeout=10):
109          try:
110              with open(key_file_name2, 'r') as file:
111                  key_list = json.load(file)
112          except:
113              key_list = []
114
115          key_list.append({
116              'private': private_b58,
117              'public' : public_b58
118          })
119
```

```
120             with open(key_file_name2, 'w') as file:
121                 json.dump(key_list, file, indent=2)
122     return public_b58, private_b58
123
124 def generate_tx_manager():
125     global num_of_new_tx
126     global num_of_pending_tx
127     global num_of_blocks
128     global mining_flag
129     global system_stop_flag
130
131     while True:
132         sleep(random.randint(0,5))
133         generate_tx()
134         num_of_new_tx += 1
135         if system_stop_flag == True:
136             exit()
137
138 def generate_tx():
139     global key_file_name1
140     global key_file_name2
141     global trans_file_name1
142     global trans_file_name2
143     global pending_tx
144
145     generate_key()
146
147     with open('data.txt','r') as file:
148         data = file.read()
149
150     with filelock.FileLock(key_file_name1,timeout=10):
151         try:
152             with open(key_file_name2,'r') as file:
153                 key_list = json.load(file)
154         except:
155             key_list = []
```

```
156
157     time = str(datetime.datetime.now()).encode()
158     pub_key = base58.b58decode(key_list[0]['public'])
159     pri_key = base58.b58decode(key_list[0]['private'])
160
161     sha = hashlib.sha256()
162     sha.update(pub_key)
163     sha.update(time)
164     sha.update(data.encode())
165     hash = sha.digest()
166
167     key = ecdsa.SigningKey.from_string(pri_key, curve=ecdsa.SECP256k1)
168     sig = key.sign(hash)
169
170     with filelock.FileLock(trans_id_file_name1,timeout=10):
171         try:
172             with open(trans_id_file_name2,'r') as file:
173                 tx_list = json.load(file)
174         except:
175             tx_list = []
176
177     tx_list.append({
178         'TxID': hash.hex(),
179         'time':time.decode(),
180         'publisher': pub_key.hex(),
181         'data': data,
182         'sig': sig.hex()
183     })
184     with filelock.FileLock(trans_id_file_name1,timeout=10):
185         with open(trans_id_file_name2,'w') as file:
186             json.dump(tx_list,file,indent=2)
187
188     share_tx(tx_list[-1],hash.hex())
189
190 def share_tx(tx,TxID):
191     global peer_list_file_name1
```

```
192     global peer_list_file_name2
193     with filelock.FileLock(peer_list_file_name1,timeout=10):
194         try:
195             with open(peer_list_file_name2,'r') as file:
196                 peer_list = json.load(file)
197         except:
198             peer_list = []
199
200     for peer in peer_list:
201         if peer != ID:
202             build_message("a",ID,peer_list[peer]['IP address'],peer_list[peer]['Port'],host,port,TxID,"new tx",tx["time"],tx["data"],tx["sig"],tx["publisher"],0)
203
204 def build_message(msg_type,ID,addr_to,port_to,my_addr,my_port,sp1,sp2,sp3,sp4,sp5,sp6,sp7):
205     message = {'msg_type':msg_type,
206                'ID':ID,
207                'addr':my_addr,
208                'port':my_port,
209                'sp1':sp1,
210                'sp2':sp2,
211                'sp3':sp3,
212                'sp4':sp4,
213                'sp5':sp5,
214                'sp6':sp6,
215                'sp7':sp7}
216     message = json.dumps(message)
217     my_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
218     my_socket.connect((addr_to,port_to))
219     my_socket.sendall(message.encode('utf-8'))
220
221 def p2p_start():
222     global host
223     global port
224     global ID
225     global my_socket
```

```
226     host = get_myip()
227     my_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
228     ID = "N0000"
229     port = port_set()
230     initialize_peer_list(ID,host,port)
231     show_my_info()
232
233     #他のノードからのリクエストを常に待機する
234     accepting_request = threading.Thread(target=pending)
235     accepting_request.setDaemon(True)
236     accepting_request.start()
237
238     #20 秒置きに peer の生存確認を行う
239     #conn_confirmation = threading.Thread(target=connection_confirmation)
240     #conn_confirmation.setDaemon(True)
241     #conn_confirmation.start()
242
243 def port_check(port):
244     try:
245         my_socket.bind((host,port))
246         return 1
247     except:
248         return 0
249
250 def port_set():
251     port = 50030
252     while True:
253         if port_check(port) == 1:
254             return port
255         else:
256             port += 1
257
258 def get_myip():
259     s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
260     s.connect(('8.8.8.8',80))
261     return s.getsockname()[0]
```

```
262
263 def pending():
264     global system_stop_flag
265     my_socket.listen(10)
266     while True:
267         conn, addr = my_socket.accept()
268         handle_message(conn)
269         if system_stop_flag == True:
270             exit()
271
272 def initialize_peer_list(ID,host,port):
273     global peer_list_file_name1
274     global peer_list_file_name2
275
276     my_info = {
277                 ID:{
278                     "time":time.ctime(),
279                     "IP address":host,
280                     "Port":port}
281                 }
282
283     with filelock.FileLock('peer_list.lock',timeout=10):
284         try:
285             with open('peer_list.txt','r') as file:
286                 peer_list = json.load(file)
287         except:
288             peer_list = []
289     peer_list.update(my_info)
290
291     with filelock.FileLock(peer_list_file_name1,timeout=10):
292         with open(peer_list_file_name2,'w') as file:
293             json.dump(peer_list,file,indent=2)
294
295 def show_my_info():
296     print("Your address is :",host)
297     print("Your port is :",port)
```

```
298         print("Your ID is:",ID)
299
300 def handle_message(message):
301         global host
302         global port
303         global node_list
304         global system_start_flag
305         global request_OK_flag
306         global num_of_blocks
307         global num_of_new_tx
308         global count
309
310         conn = message
311         msg = conn.recv(4096)
312
313         msg = json.loads(msg)
314         msg_type = msg['msg_type']
315         peer_ID = msg['ID']
316         peer_addr = msg['addr']
317         peer_port = msg['port']
318
319         if msg_type == 0:
320             request_OK_flag = True
321             print("peer",peer_ID,"OK.")
322             share_node_list(peer_addr,peer_port,peer_ID)
323             add_core_node(peer_addr,peer_port,peer_ID)
324         elif msg_type == 1:
325             print("Request for connection was called.")
326             add_core_node(peer_addr,peer_port,peer_ID)
327             build_message(0,ID,peer_addr,peer_port,host,port,0,0,0,0,0,0,0)
328         elif msg_type == 2:
329             renew_core_list(msg["sp1"])
330         elif msg_type == 3:
331             living_confirmation(peer_addr,peer_port)
332         elif msg_type == 4:
333             node_list.append(peer_ID)
```

```
334         elif msg_type == 6:
335             send_core_node_list(peer_addr,peer_port)
336         elif msg_type == 7:
337             #print("living confirmation recieved")
338             living_confirmation(peer_addr,peer_port)
339         elif msg_type == 8:
340             #print("living confirmation recieved2")
341             node_list.append(peer_ID)
342         elif msg_type == 10:
343             system_start_flag = True
344         elif msg_type == 11:
345             print("New peer added.")
346             add_core_node(msg["sp1"],msg["sp2"],msg["sp3"])
347         elif msg_type == 100:
348             count += 1
349             if count == 3:
350                 exit()
351         elif msg_type == 'a':
352             tx = {"TxID":msg["sp1"],
353                   "time":msg["sp3"],
354                   "publisher":msg["sp6"],
355                   "data":msg["sp4"],
356                   "sig":msg["sp5"]}
357             verify(msg["sp1"],tx)
358         elif msg_type == 'b':
359             num_of_blocks += 1
360             num_of_new_tx -= len(msg['sp5'])
361
362 def living_confirmation(peer_addr,peer_port):
363     build_message(8,ID,peer_addr,peer_port,host,port,0,0,0,0,0,0,0)
364
365 def connection_confirmation():
366     global node_list
367     global ID
368     global host
369     global port
```

```
370     global peer_list_file_name1
371     global peer_list_file_name2
372     global system_stop_flag
373
374     while True:
375         if system_stop_flag == True:
376             exit()
377         sleep(30)
378         node_list = []
379         #print("Start connection confirmation...")
380         peer_list = []
381         remove_list = []
382         is_change = False
383         with filelock.FileLock(peer_list_file_name1,timeout=5):
384             try:
385                 with open(peer_list_file_name2,'r') as file:
386                     peer_list = json.load(file)
387             except:
388                 peer_list = []
389
390         for peer in peer_list:
391             if peer != ID:
392                 try:
393                     build_message(7,ID,peer_list[peer]['IP address'],peer_list[p
eer]['Port'],host,port,0,0,0,0,0,0,0)
394                 except:
395                     pass
396         sleep(30)
397         node_list.append(ID)
398
399         for peer in peer_list:
400             if peer in node_list:
401                 continue
402             else:
403                 remove_list.append(peer)
404                 print("peer ",peer," was disconnected.")
```

```
405                     is_change = True
406
407             if is_change:
408                 print("There was a chenge in network topology.")
409                 remove_node(remove_list)
410
411 def add_core_node(peer_addr,peer_port,peer_ID):
412     global peer_list_file_name1
413     global peer_list_file_name2
414     add_info = {
415         peer_ID:{
416                 "time":time.ctime(),
417                 "IP address":peer_addr,
418                 "Port":peer_port}
419             }
420
421     with filelock.FileLock(peer_list_file_name1,timeout=10):
422         try:
423             with open(peer_list_file_name2,'r') as file:
424                 peer_list = json.load(file)
425         except:
426             peer_list = []
427         peer_list.update(add_info)
428
429         with open(peer_list_file_name2,'w') as file:
430             json.dump(peer_list,file,indent=2)
431
432 def share_node_list(new_addr,new_port,new_ID):
433     global peer_list_file_name1
434     global peer_list_file_name2
435     with filelock.FileLock(peer_list_file_name1,timeout=10):
436         try:
437             with open(peer_list_file_name2,'r') as file:
438                 peer_list = json.load(file)
439         except:
440             peer_list = []
```

```
441
442     for peer in peer_list:
443         if peer != new_ID and peer != ID:
444             build_message(11,ID,peer_list[peer]['IP address'],peer_list[peer]['Port'],host,port,new_addr,new_port,new_ID,0,0,0,0)
445         if peer != ID:
446             build_message(11,ID,new_addr,new_port,host,port,peer_list[peer]["IP address"],peer_list[peer]['Port'],peer,0,0,0,0)
447
448 def send_system_start_signal():
449     global peer_list_file_name1
450     global peer_list_file_name2
451
452     with filelock.FileLock(peer_list_file_name1,timeout=10):
453         try:
454             with open(peer_list_file_name2,'r') as file:
455                 peer_list = json.load(file)
456         except:
457             peer_list = []
458
459     for peer in peer_list:
460         if peer != ID:
461             build_message(10,ID,peer_list[peer]['IP address'],peer_list[peer]['Port'],host,port,0,0,0,0,0,0,0)
462
463 def remove_node(remove_list):
464     global peer_list_file_name1
465     global peer_list_file_name2
466     with filelock.FileLock(peer_list_file_name1,timeout=10):
467         try:
468             with open(peer_list_file_name2,'r') as file:
469                 peer_list = json.load(file)
470         except:
471             peer_list = []
472
473     for peer in remove_list:
```

```
474             peer_list.pop(peer)
475         with filelock.FileLock(peer_list_file_name1,timeout=10):
476             with open(peer_list_file_name2,'w') as file:
477                 json.dump(peer_list,file,indent=2)
478
479 def renew_core_list(new_peer_list):
480         global peer_list_file_name1
481         global peer_list_file_name2
482         with filelock.FileLock(peer_list_file_name1,timeout=10):
483             try:
484                 with open(peer_list_file_name2,'r') as file:
485                     peer_list = json.load(file)
486             except:
487                 exit()
488             peer_list.update(new_peer_list)
489         with filelock.FileLock(peer_list_file_name1,timeout=10):
490             with open(peer_list_file_name2,'w') as file:
491                 json.dump(peer_list,file,indent=2)
492 start_test_system()
```

付録３　シミュレーションプログラムの解説

付録３では、3.2 で行ったシミュレーションのソースコードについて解説する。storage_simulation.py では、ブロックの容量やネットワークに存在するノードの数、k の値等のパラメータを設定し、3.1 で導出した式によって計算を行い、matplotlib を使用してグラフを描画している。ソースコードは以下の通り。

ソースコード 3　storage_simulation.py

```
1 import matplotlib.pyplot as plt
2 import matplotlib.ticker as ticker
3
4 DIR = './'
5
6 #パラメータ設定
7 B_block = 100000
8 B_meta = 150
9 p_attack = 0.01
```

```
10 max_N_block = 10001
11 N_node = 100000
12
13 #変数の準備
14 result = []
15 result2 = []
16 file_name = DIR + "storage_simulation" + ".png"
17
18 #ストレージ容量の計算
19 for k in range(1,10):
20     k = int((k/10)*N_node)
21     tmp = []
22     for N_block in range(1,max_N_block):
23         improved_BC = ((k/N_node)*B_block*N_block + B_meta*N_block)/(10**6)
24         tmp.append(improved_BC)
25     result2.append(tmp[-1])
26     result.append(tmp)
27 nomal_BC_storage = [B_block*N_block/(10**6) for N_block in range(1,max_N_block)]
28
29 #横軸の設定
30 N_block_list = [int(i) for i in range(1,max_N_block)]
31
32 #グラフの描画
33 plt.figure(figsize=(10,8))
34 plt.gca().get_yaxis().set_major_locator(ticker.MaxNLocator(integer=True))
35 cm = plt.get_cmap("Reds")
36 colors = [cm(0.1), cm(0.2), cm(0.3), cm(0.4), cm(0.5),cm(0.6), cm(0.7), cm(0.8), cm(0.9), cm(1.0)]
37 plt.xlabel('Cumulative number of blocks')
38 plt.ylabel('Strage (MB)')
39 i = 1
40 con_BC_label="Conventional BC(k="+str(N_node)+")"
41 plt.plot(N_block_list,nomal_BC_storage,label=con_BC_label)
42 for res in result:
```

```
43        label_k = int((i/10)*N_node)
44        label_ = "k=" + str(label_k)
45        plt.plot(N_block_list,res,label=label_,color=colors[-i])
46        i += 1
47  plt.grid()
48  plt.legend()
49  plt.savefig(file_name)
50
51  #シミュレーション結果の出力
52  print("Strage Required when 10000th block made.¥n")
53  print("Conventional BC:",nomal_BC_storage[-1],"MB")
54  for res in result2:
55        print("k =",(result2.index(res)+1)*10000,":storage:",res,"MB ",round(100-(res/
nomal_BC_storage[-1])*100,2),"% reduced.")
```