ブロックチェーンを用いたセキュアな分散コンピューティング

修士論文

ブロックチェーンを用いたセキュアな分散コンピューティング

東京大学大学院 情報理工学系研究科 電子情報学専攻

学籍番号：48-186437 氏名：岩下義明

指導教員：坂井修一

2020 年 1 月 30 日

概 要

ブロックチェーン上でプログラムを実行するスマートコントラクトというシステムがイーサリアム上に用意されている．スマートコントラクトを利用すれば売買などの取引が自動で安全に行われるようになると期待されている．しかし，スマートコントラクトでは実行したプログラムへの入出力を記録した取引が公開されていてデータの秘匿性を保った計算ができない．

本研究ではブロックチェーンやスマートコントラクトの非中央集権的な取引の自動化という理念はそのままに秘匿性を保ったままの計算を実現した．秘匿性を保ったまま分散計算を行う方法としてセキュアマルチパーティ計算を利用した．そしてそれをイーサリアムの Web API である Web3 にて用意されている Whisper という通信プロトコルを用いて実装し，整数の和と積の演算を行った．セキュアマルチパーティ計算ではデータを暗号化して他人のコンピュータに渡し，受け取ったコンピュータが他のコンピュータと協力しながら計算を行う．その際データを受け取ったコンピュータが不正に計算の内容や計算報酬を得る場合に注目し，それらをどのように防いでいくのかを提案した．

目 次

第1章　序論

ビットコインがもたらしたブロックチェーンのシステムは中央集権的な管理者が存在しないながらも安全に価値の移管と保存を可能とした．ブロックチェーンはただ仮想通貨（暗号通貨）への利用にとどまらず，金融システム，さらには金融以外の様々な分野への応用が研究されている．例えばソニーでは著作権の管理においてブロックチェーンを用いる研究を進めている．ブロックチェーンの耐改竄性とタイムスタンプを利用して過去のどの時点にどのような著作権が存在していたかの証明に使われる予定である [49]．また，ウォルマートでは中国から輸入する豚肉をブロックチェーンを用いて管理する計画がある．豚肉がどこからきてどのようなルートを経由したのかをブロックチェーンに登録して問題が起きた時にブロックチェーン上に登録された記録を参照して特定しようというものである [43]．

このようにブロックチェーンが登場するによって改竄されない非中央集権的で分散システムを構築することが可能になった．特に，イーサリアムというブロックチェーンはスマートコントラクトと呼ばれるプログラムをブロックチェーン上に登録することができる．そしてブロックチェーンを利用する人誰もがそのスマートコントラクトを利用することができる．これにより，様々な取引を仲介者が存在しなくて安全に実行したりアプリケーションを動かすなど今まで以上に様々なことがブロックチェーンでできるようになると期待されている．

一方，イーサリアム上のスマートコントラクトでは誰がいつ利用したか・入出力の値などもブロックチェーンに登録されてしまう．そしてブロックチェーンの性質上このブロックチェーンに登録されたデータは世界中で参照が可能である．このようにデータが全世界に公開されている状態はプライベートデータを扱う計算など秘匿性を保ったままの計算を実行するには適していない．

本研究ではブロックチェーンの非中央集権と分散化という理念はそのままに秘匿性を保ったまま計算を行うシステムを提案した．秘匿性を保ち分散して計算を行う方法としてセキュアマルチパーティ計算を利用した．ブロックチェーンとしてイーサリアムを用い，計算をイーサリアム Javascript API である Web3 の Whisper プロトコルを用いて DApps フロントエンドアプリケーションとして実装した．また，Enigma プロジェクトと呼ばれるプライバシーデータの計算と既存のブロックチェーンのスケーラビリティを解決する先行研究を紹介する．そして Enigma プロジェクトの抱える課題とそれを本研究ではどのように解決していくのかを説明する．

本研究において議論したことを以下に述べる．

- イーサリアムシステム上での秘匿性を保った分散計算アプリケーションの提案
- 先行研究 Enigma の課題点とそれを本研究でいかに解決していくかの提案
- DApps ネットワーク上での分散計算の実装
- Whisper プロトコルを利用したシステムの実装
- Whisper プロトコルを利用していく上での課題

- 提案した分散計算システムの応用
- イーサリアムシステム上で動作する分散計算システムを利用してもらう方法の提案

以下に本研究の構成を説明する．第 2 章では秘密計算について具体例を挙げながら説明する．第 3 章ではブロックチェーンの代表例としてビットコインを取り上げ，ブロックチェーンの基本的な技術を説明する．また，本研究で用いたブロックチェーンであるイーサリアムの特徴とそれにより何が実現できるようになるのかを説明する．第 4 章では先行研究を複数紹介する．特に，Enigma プロジェクトは本研究と同じくブロックチェーンを利用しつつも秘匿性を保った計算の実現を目指している．第 5 章では本研究で提案するシステムを説明した．先行研究に存在する課題を挙げ，それらを本研究ではどのように解決していくのかを説明した．第 6 章では本研究で実装したシステムの説明を行う．また，実際に実験を行いどのような結果が得られたのかを示した．第 7 章では本実験において得られた結果を元に考察を行った．さらに本研究で提案したシステムを実際に利用する際にどのような活用方法があるのかや利用してもらうためのアイデアなども示した．第 8 章では本研究の結論を述べた．

第2章　秘密計算

2.1　準同型暗号

準同型暗号とは図 2.1 に示すように暗号化した状態でデータの計算を行い，結果を復号することで元のデータの計算結果が得られる暗号方式のことである．適応可能な計算の範囲において，部分準同型暗号・Somewhat 準同型暗号・完全準同型暗号の三種類に区分される．

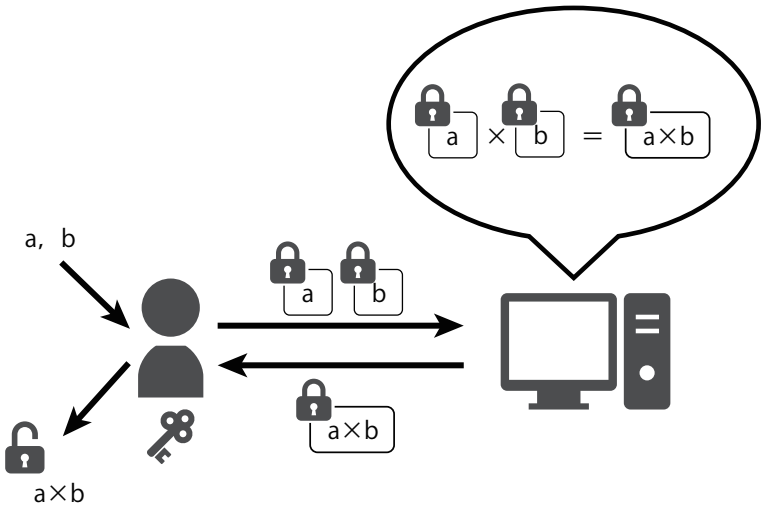


図 2.1: 準同型暗号を用いて計算を行う様子．計算を行っている主体（右のコンピュータ）は暗号化されているデータの中身は分からないまま計算を行う

2.1.1　部分準同型暗号

部分準同型暗号（Partially Homomorphic Encryption）は加算もしくは乗算のどちらかを一回のみ行える準同型暗号である．これは 1978 年に Rivest ら [67] が RSA 暗号 [68] を発表した後に提案した．この今日では公開鍵暗号として知られている RSA 暗号自体も準同型性を有する．

RSA 暗号

RSA 暗号の解読の難しさは桁数が大きい合成数の素因数分解が困難であることに依存する．必要な関数を簡単に示すと以下のものがある：

- KeyGen：適切な正の整数 n, e を生成する
- Encrypt：$\mathsf{Encrypt}(n, e, m) = m^e \mod n$

ここで，n は大きな素数 p, q の積 pq，e は小さな数（通常 $66537(= 2^{16} + 1)$）である．

RSA 暗号が準同型性を持つは以下の場合である．この場合は RSA 暗号は積において準同型性を持つ．

$$\begin{aligned} \mathsf{Encrypt}(n, e, m_1) \cdot \mathsf{Encrypt}(n, e, m_2) &= m_1^e \cdot m_2^e \mod n \\ &= (m_1 \cdot m_2)^e \mod n \\ &= \mathsf{Encrypt}(n, e, m_1 \cdot m_2) \end{aligned} \tag{2.1}$$

このように暗号化した状態で $m_1 \cdot m_2$ が計算できることが分かる．

2.1.2 Somewhat 準同型暗号

部分準同型暗号は演算が一回しかできないため複数回演算を行う場合にはいちいち復号・暗号化を行う必要があり効率が悪い．これを改善したのが Somewhat 準同型暗号（Somewhat Homomorphic Encryption）である．Somewhat 準同型暗号では加算と乗算を回数制限付きではあるが複数回行うことができる．Somewhat 準同型暗号の例として BGN 暗号 [26] がある．これは任意の回数の加算と 1 回の乗算が可能である．

2.1.3 完全準同型暗号

最初の完全準同型暗号（Fully Homomorphic Encryption）は Gentry [44] により提案されたが，これは計算機に実装できないくらい膨大な計算量を必要とした．以後様々な研究により計算量の削減・高速化が行われてきた．

完全準同型暗号においては，暗号の解読困難性を保つためにノイズというランダムな値を入れて計算を行う必要がある．このノイズは加算を行うたびに倍増し，乗算を行うたびに指数的に増加する．ノイズは暗号の解読困難性を保つために必要不可欠なものであるが，このまま計算をし続けてノイズが大きくなりすぎると本来欲しい結果とノイズの分離ができなくなる．それを防ぐために完全準同型暗号ではノイズを削減するブートストラップ（bootstrapping）という作業を行う．しかし，このブートストラップは数十秒〜数分もの計算が必要となる． [74]

2.2 秘密分散法

秘密を分散管理し，また復元する方法は Blakley [24] と Shamir [75] により 1979 年に独立に提案された．この方法は秘密分散法と呼ばれる．特に，元のデータを n 個に分割し，そのうち任意の k 個が集まれば復元できる方法を (k, n) 秘密分散法と呼ぶ．

秘密分散法の代表として Shamir の (k, n) 閾値法がある．これを簡単に説明すると n と k の関係は k 次方程式上の異なる n 個の点である．点が $k-1$ 個までしかわからない状態ではそれらの点を通る k 次方程式を一意に定めることはできないが，点が k 個以上集まると k 次方程式は一意に定まる．

秘密分散法は後述するセキュアマルチパーティ計算のシェアの作成においても使われる．整数 m 未満の数において $n = 3, k = 2$ の秘密分散を利用する際は

単純な $n = 3, k = 2$ 秘密分散

入力：0 以上 m 未満の整数 a

1. $0 < a_0, a_1 < m$ をランダムに選択する．
2. $a_2 := a - a_0 - a_1 (\mod m)$ を計算する．
3. $(a_0, a_1), (a_1, a_2), (a_2, a_0)$ を a のシェアとする．

とすれば簡単に実現可能である．ここでは a_0, a_1, a_2 の三つがそろえば元のデータ a が $a = a_0 + a_1 + a_2 (\mod m)$ より復元可能であるが，それにはシェアのいずれか二つが必要であることが分かる．

2.3 セキュアマルチパーティ計算

セキュアマルチパーティ計算（Secure Multiparty Computation：MPC, sMPC）とは Yao [84, 85] らと Goldreich [45] らにより提案された計算手法である．セキュアマルチパーティ計算では元のデータを分割したシェアを持つノードが複数存在し，ノード同士が互いのシェアを明かさず通信を行い，計算結果を導く方法である（図 2.2）．このときに計算を行うノードのことをパーティ（Party）と呼ぶ．

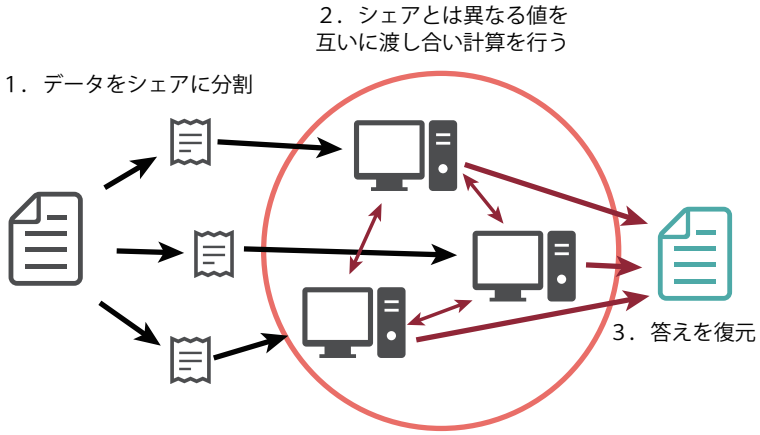


図 2.2: セキュアマルチパーティ計算を行う様子．各パーティはシェアと異なるデータを互いにやりとりし計算を行う．

セキュアマルチパーティ計算において正常な計算を妨げる攻撃者のモデルは二種類あり，malicious モデルと semi-honest モデルに分類される．malicious モデルでは攻撃者は閾値 $t(< n/2)$ 以下のパーティを自由に操作して入力値の取得や演算結果の改竄などの不正を試みる．semi-honest モデルでは攻撃者は正しくプロトコルを実行したパーティの閾値以下のログから入力値を得ようとする．

セキュアマルチパーティ計算は大きく分けて

- シェアの安全な分割
- シェアを利用した計算
- 答えのシェアの復元

の三つの手続きが必要であるが，このうち「シェアの安全な分割」と「答えのシェアの復元」には秘密分散法が用いられることがある．

ここでは Chida ら [29] の提案したモデルをもとにセキュアマルチパーティ計算における加減算と乗算の説明を行う．ここではシェアは節 2.2 にて説明した方法を用いて生成する．

2.3.1 加減算

a のシェア $(a_0, a_1), (a_1, a_2), (a_2, a_0)$ と b のシェア $(b_0, b_1)(b_1, b_2), (b_2, b_0)$ を用意する．ここで，$c_0 := a_0 \pm b_0 \pmod m, c_1 := a_1 \pm b_1 \pmod m, c_2 := a_2 \pm b_2$ とすれば，各パーティにおいて答えのシェア $(c_0, c_1), (c_1, c_2), (c_2, c_0)$ が導きだせる．これらのシェアのうち最低二つが揃えば $c = c_0 + c_1 + c_2 \pmod m$ より $c = a \pm b$ の答え c が求められる．

2.3.2 乗算

乗算においてはパーティ同士の通信が必要となる．ここでは a と b のシェアのうち $(a_0, a_1), (b_0, b_1)$ を持つパーティをパーティX，シェア $(a_1, a_2), (b_1, b_2)$ を持つパーティをパーティY，シェア $(a_2, a_0), (b_2, b_0)$ を持つパーティをパーティZ と呼ぶことにする．計算手順は以下のようになる．

- パーティX は $0 < r_1, r_2, c_0 < m$ を満たす整数 r_1, r_2, c_0 をランダムに生成する
- パーティX は $c_1 := (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 c_0 \pmod m$ を計算してパーティY に (r_1, c_1) を，パーティZ に (r_2, c_0) を送信する．
- パーティY は $y := a_1 b_2 + a_2 b_1 + r_1 \pmod m$ を計算してパーティZ に y を送信する．
- パーティZ は $z := a_2 b_0 + a_0 b_2 + r_2 \pmod m$ を計算してパーティY に z を送信する．
- パーティY，Z はそれぞれ送られてきた値を利用して $c_2 := y + z + a_2 b_2 \pmod m$ を計算する．

以上によりパーティX では $c := ab$ のシェア (c_0, c_1) が，パーティY ではシェア (c_1, c_2) が，パーティZ では (c_2, c_0) が導き出せる．

第3章　ブロックチェーン

ブロックチェーンは Satoshi Nakamoto と呼ばれる人物が 2008 年に発表した論文 [55] に端を発する．そこでは中央集権的なシステムを利用せずに，今まで実現が難しいといわれていた電子取引における二重支払いなどを防ぐかのアイデアが示されている．前半ではブロックチェーンの代表であるビットコインがどのようにブロックチェーンシステムを実現しているのかの説明を行い，後半では本研究で扱うブロックチェーンであるイーサリアムについて説明を行う．

3.1　ビットコイン

ビットコインは中央集権的な管理者をもたず，ビットコインネットワークに参加している人たち自身の相互協力によりネットワーク上で取引を行い，その記録をし，多数の人々が同じ記録をした台帳を共有し不正が行われていないことを保障している．ビットコイン上においては個々人はアドレスと呼ばれる文字列で区別され，アドレス同士の取引のことをトランザクションと呼ぶ．このトランザクションをまとめた記録の塊をブロックと呼び，そのブロックを過去のブロックの後ろにつなげることによりブロックチェーンが形成され，正しくつなげるための鍵を見つける作業をマイニングと呼ぶ．ブロックチェーンのネットワーク上に存在するマイニングなどを行うコンピュータのことをノードと呼ぶ．

3.1.1　アドレス・ウォレット

ビットコインアドレスはビットコインネットワーク上で個々人を区別するために用いられる文字列である．個人を区別するのに用いられるが一人の人間が複数のアドレスを持つことも可能である．既存の金融システムで考えると口座番号のようなものだと考えるとよい．そのビットコインアドレスがいくらのビットコインを所持しているのかを管理しているのがウォレットである．

ウォレットは対応するアドレスのビットコイン残高を管理し，支払いに利用することができるなどまさしく財布の役割を果たすものであるが，ビットコインのウォレットそのものにはビットコインは保存されていない．ビットコインの過去の取引一覧からウォレットに登録されているアドレスが保持する未使用トランザクションアウトプット（UTXO）を集計して合計を利用可能ビットコインとして表示している．すなわち，ウォレットは技術的な側面においてはデータベースの役割を果たす．

ウォレットはインターネットへの接続の可否によってホットウォレットとコールドウォレットに区別される．ホットウォレットとは，インターネットにつながっていていつでも入出金できる状態のウォレットのことである．一方コールドウォレットはインターネットにつながっておらず，アドレスの元となる秘密鍵がオフライン上に存在しているウォレットのことである．コールドウォレットでは署名を行うには鍵そのものあるいは鍵を保管している端末をパソコンなどとつなげる必要がある．ウォレットのスマートフォンアプリケーションやパソコン上のソフトウェアは基本的にホットウォレットである．これらは

入出金が簡単に行えるという利便性はあるものの，秘密鍵がネット上に流出してしまうというリスクを抱えている．コールドウォレットは秘密鍵自体の流出の心配は少ないものの，秘密鍵を保管しているデバイス（専用ハードウェア，ペーパーウォレット等）を紛失してしまったら復元の方法がないことや気軽に入出金することができないなどのデメリットもある． [94, 92]

3.1.2 トランザクション

トランザクションとはビットコインネットワーク上で行われた取引を記したデータのことである．ビットコインシステムはトランザクションをまとめたブロックを既存のブロックチェーンにつなげるのだが，この過去のトランザクションを正確にブロックチェーンにつなげることや，過去のトランザクションの内容が改竄されていないことを保障するシステムこそがビットコインを信頼できるシステムとして形作っている．すなわち，ブロックチェーンの他の要素はこのトランザクションが正しくブロックチェーンに追加されることを補助する役割を果たしている．ブロックチェーンに組み込まれたトランザクションはビットコインネットワーク上の誰からでもいつでも参照できる．すなわち過去の取引は世界中に公開されていることと同義である．

トランザクションはアドレス間のビットコインの移動を示すデータ構造である．このビットコインの移動には未使用トランザクションアウトプット（unspent transaction output : UTXO）を利用して行われる．UTXO は未使用のビットコインの塊であり，任意のアドレスに紐づけられている．そしてアドレスは自身に紐づけられている UTXO の総額を利用できる．トランザクションは送り主・受け取り主とそれぞれの送る・受け取る資金の量が記録されている．送り主も受け取り主も複数記載できる．送金プロセスは以下のようになる．

1. 送る金額以上の送り主が持つ UTXO を集める
2. 受け取る金額の受け取り主名義の UTXO を生成する
3. 集めた送り主名義の UTXO が送る金額よりも多ければ，お釣りを送り主名義の UTXO として生成する

ここで送り主の消費した金額と受け取り主が受け取る金額に差額が生じることがあり，これはトランザクショ手数料と呼ばれる．トランザクションで利用した UTXO は再利用することはできない．また，UTXO ごとに価格もさまざまである．UTXO は一回限りの小切手のようなものである．

トランザクションはトランザクション手数料を設定することができる．これは新しいブロックにトランザクションを追加するマイナーへ支払われる．マイナーにとってみればトランザクション手数料が多いトランザクションをブロックに入れればその分自身への手数料収入が入るわけであるから，トランザクション手数料はトランザクションが早くブロックに追加される（すなわちトランザクションが早く処理される）インセンティブとして働く．トランザクション手数料が少ないトランザクションは後回しにされてなかなかブロックに追加されないということが発生する．しかしトランザクション手数料が少ないとはいえ過去のトランザクションが永遠にブロックにされないままでは承認されないトランザクションがネットワーク上にいつまでも存在してしまう上にトランザクション手数料の一方的な増加を招きかねない．そこで，実際の運用ではトランザクション手数料やトランザクションの年齢を加味した優先度を設定し，手数料が少ないトランザクションもいつかは優先度が十分に上がり処理されるようになっている．

3.1.3　ブロック・ブロックチェーン

ビットコインなどのブロックチェーンはブロックと呼ばれるトランザクションをまとめたオブジェクトを時系列順に数珠つなぎで並べられて作られている．前のブロックを親ブロックと呼び，親ブロックの後ろにつながっているブロックのを子ブロックと呼ぶ．子ブロックはどのブロックが親ブロックかを示すリンクを保持している．すなわち，子ブロックから親ブロックへ，親ブロックからその親ブロックへと辿っていけば親を持たない最初のブロック（genesis ブロックと呼ぶ）へたどり着けるようになっている．genesis ブロックから現在のブロックまでにいくつのブロックが間にあるかをブロックの高さ（height）と呼ぶ．このようにブロックを延々と鎖のようにつなげていくことからブロック「チェーン」と呼ばれる．

genesis ブロックを除くすべてのブロックは親ブロックを一つしか持たないが，親ブロックは子ブロックを一つしか持たないとは限らない．ほぼ同時刻にブロックが追加されるとブロックチェーンが分岐してしまうことがある．この分岐を「フォーク」と呼ぶが，最終的には分岐はより長い方が正当なものとして認められる．

ブロックの中身

ブロックは親ブロックのハッシュ値などを含むブロックヘッダとトランザクションを記したデータ部から構成される．ブロックヘッダにはブロックが内包するトランザクションデータをマークル木を用いて一つのハッシュ値（ルートハッシュ）にしたもの（図 3.1）も含まれている．このルートハッシュはブロック内のトランザクションデータすべてを必須要素として含み，どれか一つでもトランザクションに変化があるとルートハッシュは全く違うものになってしまう．

ブロックの連結

一定時間に行われたトランザクションを内包したブロックが作られると，今度はそれをブロックチェーンに追加する作業が行われる．この追加する作業はただ親ブロックを指定すればいいわけではなく，親ブロックのハッシュ値・追加するブロックのブロックヘッダ・ノンス（乱数）を含んだ該当ブロックのハッシュ値が特定の条件を満たす解が見つからないと追加できない．この解を見つける作業をマイニングと呼び，マイニングをするブロックチェーンネットワーク参加者をマイナーと呼ぶ．

ビットコインで用いているハッシュ関数は入力の値を少し変えただけでも全く別の出力が出てくる．また，ブロックに記録されているトランザクションを一部でも変えるとそのブロックのハッシュ値が全く変わってしまうだけでなく，そのブロックを親に持つ子ブロックのハッシュ値も全く変わってしまう．ブロックのハッシュ値が変わるということは特定の条件を満たす解を見つけなおさなければ正当なブロックとしてビットコインネットワークに認めてもらえない．しかし世の中の大多数のマイナーが正統なブロックチェーンをマイニングを行い伸ばしているよりも早く改竄したブロックを含むブロックチェーンを長くすることは事実上不可能である．この現実的には改竄が不可能であるという変更不可能性こそがビットコインのようなブロックチェーンが信頼できるものとして扱われている根拠であり，ブロックチェーンに登録されているデータの正当性を保障している．

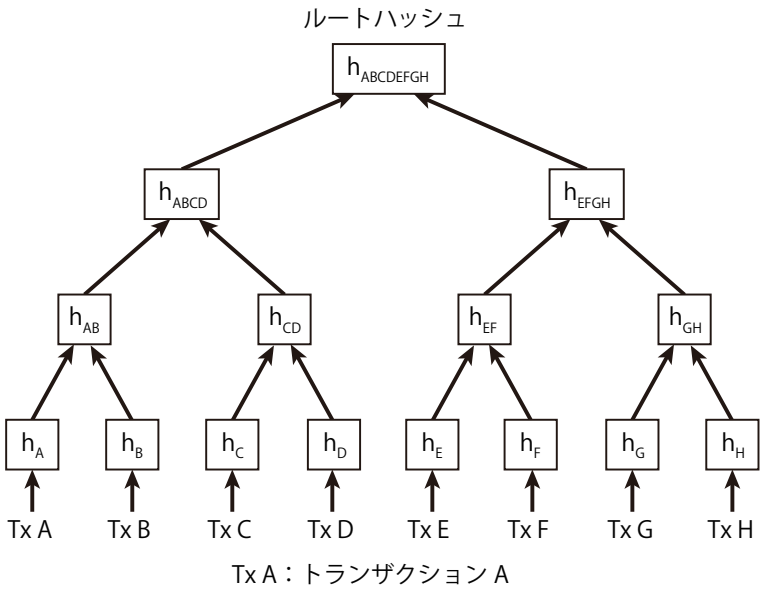


図 3.1: トランザクション A〜H からマークル木を用いてルートハッシュを導出する様子．h_{AB} は h_A と h_B からハッシュ値を求めたことを意味する．ルートハッシュ一つにトランザクション A〜H の要素を含んでいることが分かる．

3.1.4 マイニング

ブロックチェーンに新しいブロックを追加する作業のことをマイニングと呼ぶ．正確には，ブロックを追加するには条件が存在し，その条件を満たす解を見つける計算作業のことをマイニングと呼ぶ．そしてこの解を一番早く見つけた（マイニングに成功した）マイナーにビットコインは報酬を支払う．マイナーが受け取る報酬は新しいブロックを採掘した採掘報酬とブロックに含まれるトランザクション手数料の合計の二種類がある．採掘報酬はブロック数を経るごとに減少していき，最終的に 2140 年にはゼロになることが決まっている．それ以降はマイナーはマイニングに成功してもトランザクション手数料しか得ることができない．

マイニングの計算にはハッシュ関数を用いるのだが，ハッシュ関数の結果を推測することはできず総当たりで条件を満たす解を見つけなければならないためひたすら計算を行うしかない．ビットコインのマイニングにおいては SHA256 ハッシュ関数 [1] が使われる．これは暗号学的ハッシュ関数であり，出力から入力が推測できないだけでなく入力の値が近いもの同士でも全く違う結果を出力するため解に対して入力が近づいているのかなどの推測も行えない．ビットコインのマイニングは基本的にハッシュ値の計算であるので，計算能力の指標として一秒間に何回ハッシュ計算が行えるか（H/s）という単位が存在する．また，マイニングの計算能力のことをハッシュパワーと呼ぶこともある．

現在の target は 612,479 目のブロックにおいて `0x00000000000000000001465f200000000000000000000000000000000000000000` である [79]．二進数に直して 75 個先頭に 0 が並んでいることを考えると完全にランダムな数字の中からこれを見つけるのがいかに大変かが分かる．この target を 2,016 ブロック

ごとにやってくる難易度変更のときに調整し平均して 10 分に一度新しいブロックが見つけられるようにしている．マイナーが増えてハッシュパワーが増加すると早く解かれるようになり，採掘にかかる時間が短くなる．するとビットコインは難易度調整のときに target をより低く設定し，満たすべき条件を難しくする．逆にマイナーが撤退するなどしてハッシュパワーが減少するとブロックの生成時間が長くなる．するとビットコインは難易度調整において target の値を大きくし，条件を緩やかにする．

ビットコインは世界中でマイナーが膨大な計算を行いようやく新しいブロックを追加できるようにしている．このように大量の計算リソースを消費して新しいブロックを追加する新しいブロックを追加するためには大量の計算リソースが必要であり，新しいブロックを見つけたということは逆説的にそれだけの計算リソースを費やしたという証明になる．このブロックチェーンを維持するために計算リソースを費やしたという証明を用いて合意形成を行うアルゴリズム（コンセンサスアルゴリズム）を Proof-of-Work と呼ぶ．

マイナーはマイニング報酬のビットコインを目当てにマイニングを行うが，マイニングの真の目的はビットコインを生成することではなくトランザクションをまとめたブロックを正しくブロックチェーンに追加することである．しかしマイナーがどのような動機であれマイニングを行うことで十分に信頼できる分散化された共通のコンセンサスを作り出しているところがビットコインのよく考えられているところである．

3.1.5 フォーク

ブロックチェーンには仕様変更などでチェーンが分岐する「フォーク」と呼ばれる現象が発生することがあるが，フォーク発生後に収束するかしないかによってソフトフォークとハードフォークに区別される．

- ソフトフォーク
  ソフトフォークはブロックチェーンの仕様が後方互換性を保ったまま変更した際などに発生する．すなわち，新しいプロトコルを採用しつつも，新プロトコル採用以後も過去のプロトコルにより生じたトランザクションを有効とみなせる場合である．このような場合は将来的にはフォークが新しいプロトコルか古いままのプロトコルどちらか優勢な方へと収束する．例としてブロック容量を変えずブロックに入るトランザクションデータを多くした Segwit [20] を導入した場合などである．[31, 21]

- ハードフォーク
  ハードフォークは分岐したあとのチェーンが収束せず，それぞれ別のブロックチェーン・暗号通貨として存在することになる．これはブロックチェーンそのものの仕様変更が起きた際などに発生し，合意がとれるか取れないかでフォーク後の様子が変わる．合意がとれる場合にはハードフォークが行われた後は利用者はほぼ全員新しい仕様を採用し，実質的に古い方が消滅する．これはイーサリアムの大型アップデートなどで発生する．一方，合意がとれない場合のハードフォークは仕様変更を巡り暗号通貨コミュニティ内で対立している場合に起こる．例としてイーサリアムでは 2016 年 6 月に起き 50 億円あまりの被害を出した「The DAO 事件」のハッキングをロールバックして無かったことにするかを巡り，コミュニティ内で対立が発生した．送金後 27 日間は資金を利用できない制度があり，その間にハードフォークを起こして送金の記録を取り消すことで問題を解決した

のが現在まで続くイーサリアムである．しかしコミュニティ内ではこのようなハッキングが起きたからといって過去の取引を中央集権的に無いものとして扱うのは自立分散型組織として矛盾を残すのではないかと考える人たちがそれに反対した．結果としてコミュニティ全体で一つの合意に達することができず，ロールバックした状態のブロックチェーンがイーサリアムとして，ロールバックせずハッキングが起きたことも含めるブロックチェーンがイーサリアムクラシック [7] として存在することになる． [89, 22]

3.2 イーサリアム

3.2.1 概要

イーサリアム (Ethereum) [6, 27] は，ヴィタリック・ブテリン氏によって開発されたプラットフォームの名称である [19]．イーサリアムは単なるブロックチェーンではなく，スマートコントラクトと呼ばれるプログラムをブロックチェーンに登録し，実行することができる．スマートコントラクトが実行されるとイーサリアムの持つ内部の状態（ステート）が変化する．イーサリアムの状態は全世界で統一されており，全体としてみたときに一つのコンピュータのように動作することから「ワールドコンピューター」とも呼ばれている．

このイーサリアムプラットフォーム上でプログラムを実行したり価値のやりとりに用いるのがイーサ（Ether，単位 : ETH）と呼ばれる暗号通貨である．すなわち，イーサは単なるブロックチェーン上で価値が保障される暗号通貨ではなく計算リソースの指標として用いられることもある．

イーサリアムには外部所有アカウント（Externally Owned Account : EOA）とコントラクトアカウントの二種類のアカウントが存在する．EOA はビットコインと同じようにウォレットと結びつけて所有するイーサの管理に用いるアカウントであるが，コントラクトアカウントはイーサリアムブロックチェーン上にデプロイされたスマートコントラクトによりコントロールされるアカウントである． [16][p.65], [48]

イーサリアムの通貨はイーサ（Ether）と呼ばれ，通貨としては ether と表記する．ビットコインが補助通貨として satoshi (1 BTC = 10^8 satoshi) を採用しているように，イーサも補助通貨として wei という単位を採用している．1 ETH は 10^{18} wei である．イーサリアム内部ではイーサの量は常に wei 単位で換算される．ビットコインのブロック生成時間は 10 分であったのに対し，イーサリアムのブロック生成時間は 15 秒となっている．

3.2.2 計画

イーサリアムは開発段階を四つ定めており，それぞれで大きな変化が生じる．四つの主要な開発段階にはそれぞれ「フロンティア」「ホームステッド」「メトロポリス」「セレニティ」という名前がついている．各開発段階内にはサブリリースを行うためのハードフォークが含まれている場合もある． [16][pp.5-6]

- フロンティア
  2015 年 7 月 30 日にリリースされた．これは正式なリリース前の実験段階としてリリースされていたため，バグなどが発生した際にやり直しがきくなどの特徴があった． [59]

- ホームステッド
  2016 年 3 月 14 日にリリースされた．これもフロンティアに引き続き技術者向けのベータ版プラットフォームである．ここでは Gas 手数料が足りない場合の例外の追加，採掘難易度調整方法の修正などが行われた． [60]

- メトロポリス
  2019 年 3 月 1 日にリリースされた．このアップデートは二つに分かれており，前半のビザンチウムアップデートは 2017 年 9 月に完了し，後半のコンスタンティノーブルアップデートが 2019 年 3 月に完了した．ここでは将来行われるセレニティアップデートをふまえ，アップデート後に速やかに新しい体制に移行するために難易度爆弾（difficulty bomb）と呼ばれるシステムが組み込まれてた．これはあらかじめ定められたブロック数に到達すると急激にマイニングが難しくなるシステムのことである．これによりマイナーがいつまでもずっと昔のシステムを使い続けることを防ぎ，新体制に移行することを促す． [83, 91]

- セレニティ
  最終大型アップデートである．ここではコンセンサスアルゴリズムを従来の PoW (Proof-of-Work) から PoS (Proof-of-Stake) に移行することや，シャーディング（Sharding）と呼ばれるマイニングの並列処理を行いトランザクションの処理速度を高速にすること，1000 倍のスケーラビリティを持つことなどが計画されている．イーサリアムの創始者ヴィタリック・ブテリン氏は特に最初のうちから最終的に PoS への移行を言及しているなど，イーサリアムにとって非常に重要なアップデートとなる．前回のメトロポリスアップデート同様複数のハードフォークを繰り返す可能性がある．この大型アップデート後のイーサリアムシステムをイーサリアム 2.0 と呼ぶ． [91, 46]

3.2.3 スマートコントラクト

イーサリアムの大きな特徴の一つに，スマートコントラクトを利用できるということがある．スマートコントラクトとはイーサリアム上で動作するコンピュータ（イーサリアム仮想マシン）によって実行されるプログラムコードのことである．このスマートコントラクトにより，イーサリアムのコントラクトアカウントは制御される．

スマートコントラクトとは元々は暗号学者の Nick Szabo が 1996 年に提唱 [2] した概念である．そこでは契約（コントラクト）を情報技術を用いて適切に処理する仕組みとして提案されている．すなわち，現実世界の契約に発生する様々なリスクや課題を情報技術で解決しようという考え方がスマートコントラクトというものである．このような広義なスマートコントラクトから派生して，イーサリアムにおける狭義のスマートコントラクトは「イーサリアム上で実行されるプログラム」を意味している．以下，本稿での「スマートコントラクト」という言葉は狭義のスマートコントラクトを指す．

スマートコントラクトを用いることで非中央集権的な契約の履行を自動で行うシステムの構築が可能となる．また，スマートコントラクトやその利用記録がイーサリアム上に記録されていることでそのスマートコントラクトの安全性や利用履歴を誰もが確認できる．しかしイーサリアム上に一度登録（デプロイ）されたスマートコントラクトは変更ができず，またスマートコントラクト作成者が削除コードを埋め込まなければ本人にも削除できないためにスマートコントラクトの作成・デプロイに関しては細心の注意が必要である．

3.2.4 DApp・Web3

イーサリアムの開発者達はスマートコントラクトのみではなく，非中央集権化した Web とアプリケーションの実現を目指していた．この非中央集権化した Web は，静的な Web を中心としたインターネット初期の Web（Web1.0），SNS 等によるインタラクティブなコミュニケーションが可能となった Web（Web2.0）に続く第三世代の Web として「Web3.0」（あるいは Web3）と呼ばれている．そしてこの Web3 を実現するアプリケーションが「非中央集権型アプリケーション（Decentralized application : DApp，複数形 : DApps）」である． [88]

DApp

BitAngels[17] の David Johnston 氏は，DApp の定義として以下の 4 点を挙げている [12].

- アプリケーションがオープンソースであること．また，オペレーションが自動で，中央による管理ではないこと．アプリケーションは提案やフィードバックに応じてプロトコルを適合させることができるが，すべての変更においてユーザーのコンセンサスを得る仕組みであること．
- アプリケーションのデータや記録は，公開・暗号化されたブロックチェーンを利用していること．
- オープンに流通可能な暗号化トークンを持ち，アプリケーションを利用する際にはそのトークンを利用すること．また，ユーザーに報酬が発生する場合はそのトークンによって報酬が支払われること．
- アプリケーションへの貢献が暗号アルゴリズムによって証明され，それに従ってトークンが生成されること．

この 4 つの定義が絶対というわけではないが，多くの DApps はこの定義に合致している．広い意味ではビットコインや一部の仮想通貨もこの DApps という定義に合致する．実際の DApps の活用例としては，DEX と呼ばれるような分散取引所 [56] であったり，ゲーム [33] などがある．分散取引所は通常の取引所とは違い，直接ウォレット同士での取引が可能である．取引所に通貨を預ける必要がなくなるので，ハッキングによる盗難リスクが下がる．

DApp のバックエンドとしてスマートコントラクト（及びイーサリアムネットワーク）が利用されることが多い．これは DApp により実現する非中央集権型のアプリケーションとイーサリアムによって実現している非中央集権システムが相性が良いだけでなく，通常のバックエンドのサーバーの代わりとなる機能をスマートコントラクトによって実現できるからである．バックエンドとしてスマートコントラクトを利用することにより，通常の中央集権的なサーバーを用いる場合よりも障害耐性を上げることができるなどのメリットもある．

Web3

DApp のフロントエンドは，ユーザーとイーサリアムネットワークを Web ブラウザを介して繋げるという重要な役割がある．それを行うことによりユーザーのニーズに応じたメッセージの署名やトランザク

ションの送信などが可能にある．フロントエンドにおいては web3.js [9, 80] という Ethereum Javascript API などを通じてイーサリアムネットワークに接続する．web3.js は非中央集権化アプリケーションを作成するために使われる代表的な API であり，eth (Ethereum) プロトコル，bzz (Swarm) プロトコル，shh (Whisper) プロトコルの三つのプロトコルを提供している．

eth (Ethereum) プロトコルはイーサリアムクライアントとして使われるプロトコルで，ブロックチェーンに接続してアカウントの管理を行ったり通貨の送受信を行うプロトコルである．また，スマートコントラクトとのやりとりもこのプロトコルを利用して行える．イーサリアムに関係する全般的な機能を担うプロトコルである．

bzz (Swarm) プロトコル [35] は，IPFS に類似したコンテントアドレッサブル P2P ストレージサービスである．コンテントアドレッサブル（content addressable）とは，コンテンツの各部分のハッシュを計算し，ハッシュがそのコンテンツを識別するために使用することを意味している．Swarm ノードによって配布・複製されるファイルを格納できる，分散ストレージサービスを実現するためのプロトコルである．Swarm を使用すると単一の中央集権的な Web サーバーではなく非中央集権型の P2P システムから Web サイトにアクセスできる．

shh (Whisper) プロトコル [13, 10] は DApp 上に存在する P2P ノード同士のメッセージのやりとりに使われるメッセージングプロトコルである．イーサリアムブロックチェーン上に載せる必要のないメッセージやデータのやり取りに使われる．特徴として以下の点が挙げられる：

- 低レベル
  Whisper API は DApp が利用するものでありユーザーが直接利用するものではない．
- 小容量
  大量のデータ転送のための設計ではない
- 非リアルタイム性
  RTC（リアルタイムコミュニケーション）を目的としたものではない
- 秘匿性
  パケットのトラッキングに利用するものではない

また，単純な双方向通信だけでなくブロードキャストもできる．1 対 1 あるいは 1 対 N の特定の相手に向けてのメッセージ送信においては低レイテンシーだが，1 対*のブロードキャストではレイテンシーが大きくなる．メッセージの容量は 64 KB よりも小さくなければならず，通常は 256 バイト前後で使われる．

3.2.5 イーサリアム仮想マシン（EVM）

イーサリアム仮想マシン（Ethereum Virtual Machine：EVM）は，イーサリアムのシステム上でスマートコントラクトをデプロイ・実行する部分である．EVM のアーキテクチャを図 3.2 に示す．EVM はイーサリアム上でプログラム（スマートコントラクト）を内部に永久に保存し，そこからプログラムを呼び出して実行することのできるコンピュータとして扱うことができる．EVM はスマートコントラクトを実行した結果としてイーサリアムの状態を変化させるが，実行途中で例外などが発生し正常に終了しなかった場合は実行直前の状態までロールバックされる．

EVM は 256 ビットで動作するコンピュータとして扱うことができる．EVM は「疑似」チューリング完全であるが，これはプログラムを実行するのに必要な燃料「gas」が関係している．gas の詳細については後述するが，端的に表現すると「プログラムを実行するたびに支払う必要があるイーサ」である．すなわち計算料のようなものである．EVM 自体はチューリング完全であるが，EVM を用いて命令を実行するにはこの gas が必要でありプログラムを実行する際に利用可能な上限の gas を定めているので無限ループなどが永遠に実行されることはない．その意味でチューリング完全を完全に模倣しているわけではないので疑似チューリング完全である（準チューリング完全とも呼ばれる）．しかしこのような制約をつけることにより悪意のあるプログラマが無限ループなどを実行させて EVM の正常な動作を永遠に阻害することを防ぐことができる．

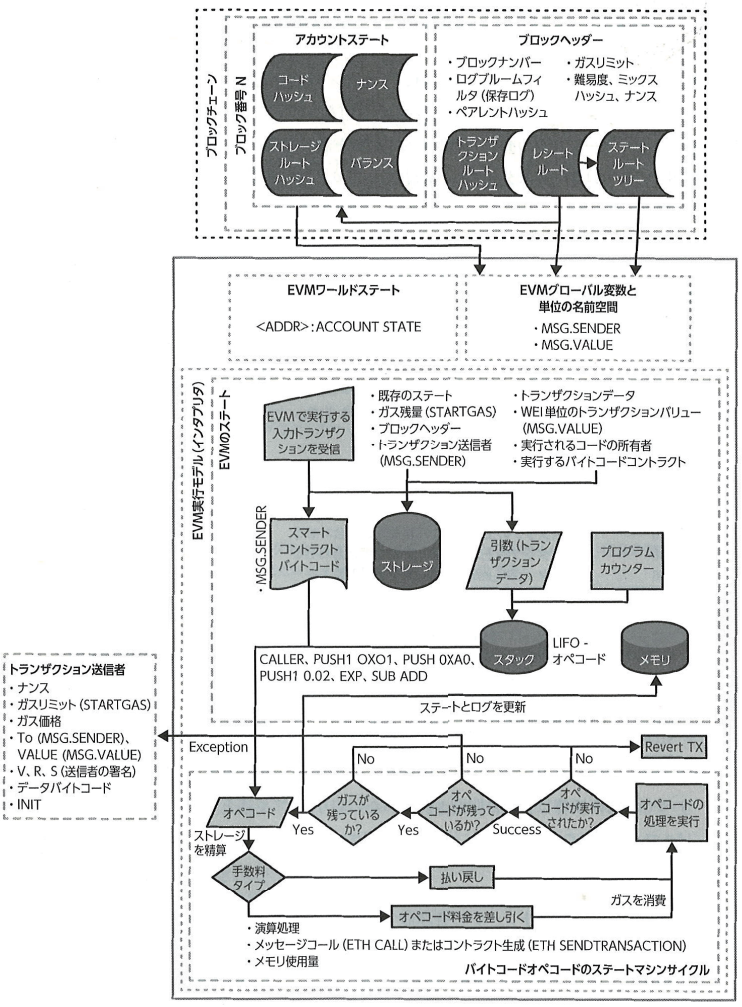


図 3.2: EVM のアーキテクチャ [16]([p.313])

gas

gas（ガス）はイーサリアムにおける基本単位で，EVM 上で計算を行うなどのイーサリアムブロックチェーン上でアクションを起こすのに必要なリソースをお金に変えたものである．ここでのリソースは計算リソースとストレージリソース両方を含む．スマートコントラクトやトランザクションにより行われる各操作はそれぞれ所定の価格の gas が必要であり，スマートコントラクトを正常に終了するにはその合計 gas よりも多いイーサを保有している必要がある．各命令の実行においてはリソースの消費量とある程度の相関を持ち必要な gas の量が決まっている．また，各スマートコントラクト実行者は呼び出し時に支払えるガスの上限（gasLimit）というのを設定する．そうすることによりスマートコントラクトが無限ループなどに陥った際でも gasLimit を超えるスマートコントラクトの実行は行われない．gasLimit を超えたり所持している gas の上限をスマートコントラクトの消費 gas が超えることになった場合には OoG (Out of Gas) 例外が発生し，トランザクションの実行は無かったことになり gas の消費以外のすべての状態は元に戻される．スマートコントラクトの実行により使われた gas は計算を実際に行ったノード（いわゆるマイナー）に支払われる．

gas の単位当たりに費用はイーサ（wei 単位）で規定されるが，その相場は固定ではない．これは gas とイーサを固定相場制にしてしまうとイーサの値上がり等でスマートコントラクトの実行が非実現的な価格になってしまうことを防ぐためである．gas の価格はトランザクションを実行する際にトランザクションの送信者が指定することができる．これにより，マイナーは未処理トランザクションの中から処理すべきトランザクションを金銭的インセンティブを元に自由に選ぶことができる．

3.3 コンセンサスアルゴリズム

ここではブロックチェーンにおける様々な合意形成（コンセンサス）アルゴリズムを説明する．

3.3.1 PoW (Proof-of-Work)

PoW (Proof-of-Work) はビットコインやイーサリアムに用いられてるブロックチェーンの中核を担うコンセンサスアルゴリズムである．新しいブロックをブロックチェーンに追加できる条件を満たす解が総当たりでしか計算できないようなものである．それを世界中のマイナーが総当たりで計算して（マイニングして）条件に該当する解を見つける競争をし，最初に解を見つけたマイナーがマイニング報酬を受け取る．

条件を満たす解を見つけるためには大量に計算を行う必要がある．つまり大量に計算を行えば解を見つける確率が高くなるということである．ブロックチェーンのマイニングにおいては SHA-256 ハッシュ関数が使われ，イーサリアムにおいては Ethash と呼ばれるハッシュ関数が使われている．現在は SHA-256 のマイニングに特化した ASIC（特定用途向け集積回路）が存在しており，CPU や GPU，FPGA などのその他プロセッサは電力効率や計算能力において ASIC に太刀打ちできない．しかし ASIC がマイニングにおいて独占的地位を占めるとそれを大量に手に入れられる特定の企業のみにマイニングパワーが集中してしまうという懸念から，ASIC が作りづらい（ASIC 耐性のある）マイニング計算アルゴリズムを用いるブロックチェーンも多い．イーサリアムの Ethash は GPU で一番効率が良く，個人で複数 GPU をつなげてマイニングリグを作成する人も存在する．また，マイニングの計算に必要なキャッシュメモ

リのサイズを大きくして CPU でのマイニングが一番効率良くなるようにした暗号通貨（BitZeny など）も存在する．

PoW ではマイニングを行う必要があり，そのマイニングには電力が必要である．マイニング参加者が増えるほど PoW にて消費される電力は増大する．あくまで推定値であるが，2020 年 1 月 20 日現在のビットコインの電力消費量は 9.17 GW であり，年間消費量に換算すると 74.75 TWh となる．これはチリの年間電力消費量を超える [58]．ビットコインのマイニングだけでこれだけの電力を消費しているということで地球環境に負荷を与えているという批判もある．

3.3.2 PoS (Proof-of-Stake)

PoS (Proof-of-Stake) は通貨の保有量に比例して新しいブロックをブロックチェーンに追加することを承認する権利を得ることができるコンセンサスアルゴリズムである．これは PoW の抱える 51 %攻撃（節 3.4.1 にて説明）への脆弱性・マイニングによる電力消費・検証作業時間がかかることによるスケーラビリティの上限などの問題を解決するために生み出されたアルゴリズムである.「通貨の保有量に比例して」ということであるが，どのように通貨の総保有量を定めるのかについては二種類ある：

- Coin Age
  通貨の保有量に通貨の保有期間を掛けたものを通貨の総保有量として算出する．すなわち昔から通貨の保有に参加しているノードほど有利になる．寡占化を防ぐためマイニングに成功すると Coin Age が減少する．

- Randomized Proof of Stake
  通貨の保有量に比例してランダムに取引承認者を選ぶ仕組みである．すなわち通貨を多く持っているほどマイニングに成功しやすくなる．

このように PoW の抱える様々な問題を解決できる PoS であるが，これにも以下のような課題が存在する：

- 通貨のため込み
  通貨の保有量が多いほどマイニングしやすくなるという仕組み上インセンティブが通貨を使うのではなく貯める方向に働く．その結果通貨の流動性が下がる可能性がある．この対策として古い通貨の持ち分評価額を下げる方法もある．

- 格差の拡大
  PoS では通貨の保有量に比例して報酬を得られるため，通貨をより多く保有している人が多く通貨を手に入れることができるなど格差が拡大してしまう恐れがある．

イーサリアムは次回のセレニティアップデートで PoS への移行を計画している．これによりマイニングによる電力消費の増大を抑えるとともにスケーラビリティを解決するとしている． [4]

3.3.3 dBFT (Delegated Byzantine Fault Tolerant)

dBFT は中国発のスマートコントラクトが利用可能な暗号通貨 NEO [14] が採用しているコンセンサスアルゴリズムである．これは大規模なノードの参加を達成するためのシステムであり，最大多数のノード

にとって利益となるような取引の記録を民主的な方法により行うことができる．また，PoW や PoS と違い NEO トークン（通貨）保持者の全員が報酬を受け取ることができる．また，最新版の dBFT 2.0 [69] においてはトランザクションの不可逆性を保証して二重支払いを防げる上，ユーザーは一回の確認（15 秒）待つだけで取引が完了する．金融用途に適していると言われる次世代のコンセンサスアルゴリズムである．

NEO ブロックチェーンに参加しているノードは Bookkeeper（記帳者）と NEO ホルダー（NEO トークン保持者）に分かれる．Bookkeeper は NEO ホルダーの投票により選出されたブロックチェーンに記帳する権利を有するノードである．NEO ホルダーはブロックチェーンに記帳することはできないが Bookkeeper を選出する権利を有する．Bookkeeper は複数選出され，以下のプロセスで新しいブロックを生成する：

1. Bookkeeper の中からランダムに一人代表者を選出する．それ以外の Bookkeeper は投票者となる．
2. 代表者がそれまでの取引を記録したブロックを生成する．
3. 投票者となった Bookkeeper はそのブロックを認めるか投票を行う．66 %以上の賛成を得ることができればブロックが正当なものとして認められ，新しいブロックがブロックチェーンにつなげられる
4. 66 %以上の賛成を得ることができない場合は別の Bookkeeper が代表者に選出され，再びブロックの生成と投票を行う．
5. 上記プロセスをブロックが認められるまで繰り返し行う．

新たなブロックが生成されると，GAS トークンという新規ブロック生成に応じて発生するトークンが NEO トークンの持ち分に応じて与えられる．NEO ホルダーは Bookkeeper が自分の望む投票をしない場合は別の Bookkeeper を選出することができる．すなわち，NEO ホルダーは所持している NEO トークンの量に応じて自分の望む Bookkeeper を選出する権利が与えられる．

dBFT の利点を以下のようなものがある：

- 新しいブロックをブロックチェーンにつなげる権限があるノードが一つしかないためフォークが発生しない
- 全てのトークン保有者が持ち分に応じてブロックチェーンの維持に関与することができる

[47, 73, 76, 25]

3.4 ブロックチェーンの抱える課題

ブロックチェーンは中央集権的な存在を持たずに運営されているシステムであり，非常に安全性が高いとされている一方で完璧に安全というわけではない．ここでは，ブロックチェーンにはどのような脆弱性が存在するのかについて議論する．

3.4.1 51％攻撃

まず挙げられるのが,「51％攻撃」というものである．これは，悪意を持ったマイナーがブロックチェーンの 51％以上の計算能力を持つことにより可能となる攻撃である．ビットコインを初めとする様々なブロックチェーンは，コンセンサスアルゴリズムとして Proof of Work を採用している．これにより全体で膨大な計算リソースを費やして過半数が正しいと認めたブロックを正式なものとする．悪意あるマイナーが計算リソースを半数以上持つようになるとその時点から本来の正当な取引を認めなかったり不正な取引を正当化できてしまう．これを行うことにより，攻撃が行われた時点からのマイニング報酬を全て獲得したり，二重送金を行ったりなどブロックチェーンの正当性を損ねる様々な不正行為ができてしまう．しかし，このような状態でも攻撃が行われるより前のブロックチェーンの記録を改竄したり，他人のウォレットを操作することはできない．

また，攻撃を行う側にしてみると，市場価値の高い暗号通貨というのはマイニング報酬目的の参加者が多く，そのような競争が激しい暗号通貨の半分以上の計算リソースを用意するには莫大な資金が必要になる．また，せっかく大金を費やして半分以上の計算リソースを確保し 51％攻撃を成功させたとしても，それが行われた時点でその暗号通貨はもはや信頼のおけるブロックチェーンではなく，市場価格の暴落を招き，結果として攻撃に費やした費用に見合う十分な利益が得られないことが予想される．そのような観点から，経済的インセンティブに基づいた 51％攻撃は起こりにくいと考えられる．しかし，ブロックチェーンの信頼性を損ねるという目的で行われる場合にはまだ十分留意しなければならない．

マイニングプールによりこの攻撃が起こりえる状況になることもある．マイニングプールとは，複数のマイナーが参加するマイナーの集合体であり，マイニングプール全体で報酬を得た場合に個々の計算量に応じて報酬を得られるようになっている．マイニング報酬は通常採掘に成功したマイナーが全取りをする．しかし，マイナーが増えたりするなどしてマイニングの難易度が上がってくると，個人では採掘の報酬を全く得られない場合がほとんどになってしまう．そこで多数のマイナーが結託してマイニングプールを構築し，マイニングプール全体でマイニング報酬を狙おうという仕組みである．2013 年に Ghash.io [8] というビットコインのマイニングプールが半分以上の計算能力を持ちそうになったことがある．これに伴いビットコインの価格が下落するなど混乱が見られたが，マイナー達が他のマイニングプールに移ることにより半分以上の計算能力を一つのマイニングプールが占めることが避けられた． [90, 86]

しかし，少数の大規模マイニングプールによる寡占化は依然として問題である．BLOCKCHAIN.COM [71] による 2020 年 1 月 8 日における直近 4 日間のビットコインマイニングプールのシェアを図 3.3 に示す [70]．これによると，F2Pool，Poolin，BTC.com，AntPool の上位四つのマイニングプールを合わせるとハッシュレートが全体の半分を超えてしまう．これらのマイニングプールが結託すれば 51％攻撃も不可能ではない．

3.4.2 セルフィッシュマイニング攻撃

もう一つの代表的な攻撃として Eyal ら [42] が提唱している「セルフィッシュマイニング攻撃」（ブロックウィズホールディングアタック/一時的ブロック隠匿攻撃とも）というものがある．まず，攻撃者はマイニングを他のノードに公開せずに行う．そして新しいブロックを見つけたとしても，それを公開せずにそれに続く新しいブロックをマイニングする．公開されているブロックよりも長いブロックを生成することができれば，それを公開する．ブロックチェーンではより長いブロックが繋がっているチェーン

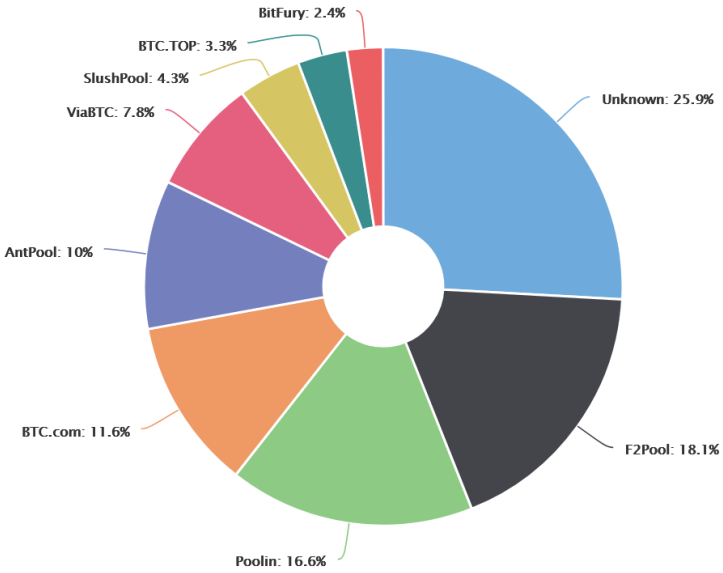
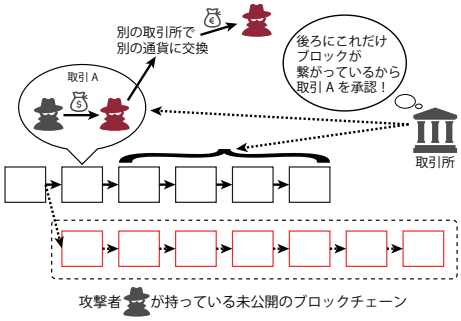


図 3.3: マイニングプールのシェア分布図 [70]．少数のプールでマイニングにおける計算能力の大部分を占めていることが分かる．

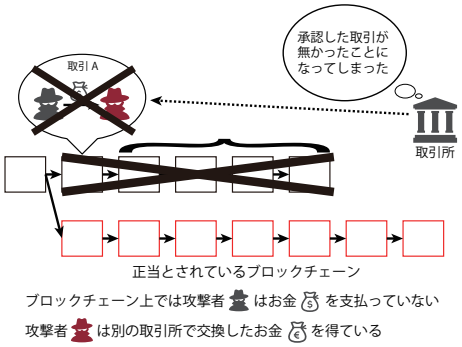
の方を正当なものとして扱ってしまうので，公開されていた短いブロックの方ではなく隠し持っていた長いブロックの方を正しいものとして認識してしまう．すると，公開されていた短いブロックに記載されていたトランザクションが無かったことになってしまい，二重支払いなどの問題を引き起こしてしまう．セルフィッシュマイニング攻撃は最大で 1/3 のマイニングパワーを持っていれば成功してしまうということは Eyal らは 1000 人のマイナーを想定したシミュレーションで示している．また，Sapirshtein ら [72] によると 25 %以下のマイニングパワーでもセルフィッシュマイニング攻撃以上の収益を上げられるという．

実際に，2018 年 5 月 13 日から 15 日にかけてモナコイン [62] がこのセルフィッシュマイニング攻撃の被害にあった [57]．この攻撃はセルフィッシュマイニング攻撃以外にも取引所の確率的ファイナリティーを利用して行われた．ウォレットや暗号通貨の取引所は通常，取引を確定させるまでに n 段階の承認を受けなければならないとする承認数を定めている．承認数分のブロックがネットワークで処理され，取引を記載したブロックの後に繋がるとその取引が確定したものとして扱っている．しかし，これは PoW の仕組みから 100 %確実に承認が確定したというわけではなく,「非常に高い確率で確定したとみなせる状態」である．このことを「確率的ファイナリティー」と呼ぶ．例として，ビットコインでは通常 6 ブロック分つながるとトランザクションが確定したとみなされている．犯人はまずパブリックなブロックチェーン上で他の暗号通貨にモナコインを換金し，承認数分のブロックが処理されるのを待ち決済を確定する（図 3.4a）．次に，隠していたチェーンを流すことで不正な取引が正しいものとみなされてしまい，取引所で使用したモナコインを使用していないこととした（図 3.4b）．このように，承認数分のブロックが処理されていてもトランザクションは確定せず，非常に小さい確率ながらもトランザクションが変更されてしまう可能性がある．これを防ぐためには承認するブロックの数を増やすということも考

えられるが，そうすると送金などの処理の確定までの時間が長くなってしまい，暗号通貨のサービスへの利便性にもかかわってしまう [78, 23].



(a) 取引所に取引が確定されるまでチェーンが長くなるのを待つ．また，取引で用いた通貨を別の通貨に両替する．



(b) 公開されているブロックチェーンよりも長いチェーンを公開し，破棄されたチェーンに含まれていた取引を無かったことにする．すると取引の送信元からお金は失われないまま両替した通貨が手に入る．

図 3.4: セルフィッシュマイニングと確率的ファイナリティーを用いて不正に取引を無かったことにする様子

3.4.3 台帳のデータ容量の増加

ブロックチェーンは稼働開始からのすべてのトランザクションをブロックにまとめ，つなげている．すなわち，すべてのブロックとトランザクションを記録するフルノードに要求される容量は年々増加する一方である．例として，2020 年 1 月の時点においてビットコインのフルノードに必要な容量は 200GB 以上 [18] であり，イーサリアムにおいても約 200GB [93] である．将来においてはフルノードを維持するコストがより増加すると，フルノードの参加者が少なくなることが考えられる．フルノードが少なくなるとフルブロックチェーンの管理が中央集権的になりやすく，改竄されても気づかない・台帳の改竄を検知できるものの正しいトランザクションの中身が無いなどの問題が考えられる．

3.4.4 ブロックのスケーラビリティ

ブロックチェーンに含めるブロックそのものには上限のデータサイズがある．ブロックに入れられなかったトランザクションは次以降のブロックに含められることとなるが，暗号通貨がより活発に利用されて取引量（発行されるトランザクションの量）が増加すると，ブロック生成時間の間に溜まるトランザクションの量がブロック容量を超えてしまうことが考えられる．そうするとトランザクションプールに未承認トランザクションがたまるようになり，なかなかトランザクションが承認されなかったり優先的に承認されるようにするためにはより多くのトランザクション手数料を払わなければならなくなるなど大きく使い勝手が低下してしまう．この問題を解決するにはブロックサイズを大きくするということが考えられる．ビットコインの初期段階のブロックサイズは 1MB だったのだが，このスケーラビリティ

の問題に対処するために Segwit という技術を導入するかブロックサイズを 8MB にするかどちらを採用するかということが問題になった．結果として Segwit を採用しブロックサイズを 2MB にするということで合意が取れたのだが，一部の人々は反発しハードフォークを起こしブロックサイズを 8MB に変更したビットコインキャッシュ [28] という新しい暗号通貨が誕生した． [61]

3.5 P2P メッセージングプロトコル

ここでは既存の P2P 通信プロトコルにはどのようなものがあるのかを説明する．P2P 通信のうち，特に個々のメッセージングに利用できるプロトコルとして節 3.2.4 で紹介した Whisper プロトコル以外の TeleHash プロトコル [11]，WebRTC プロトコル [81] の二つを紹介する．

3.5.1 TeleHash プロトコル

TeleHash プロトコルはリアルタイムコミュニケーション用に開発された通信の内容が漏れない P2P プロトコルである．TeleHash プロトコルは，主に二つの部分からなる．最も重要な部分は「スイッチ」部で，ここでは P2P ネットワークの発見から新しいノードの接続，他のノードの検索，他のノードからの返答待ちなどを行う．相手との接続においては分散ハッシュテーブル（DHT）を利用し，目的のノードまでの参照を繰り返し行う．もう一つの部分はアプリケーション同士の通信を行う部分である．ユーザーにとっては通常の TCP と同じようにコミュニケーションができる．相手ノードの探索やメッセージの暗号化などはすべて TeleHash プロトコルが行うとしている．

TeleHash プロトコルでは暗号化に RSA 暗号と楕円曲線暗号を用いている．また，鍵は使い捨てであり毎回違うものが用意されるので意図しない他のメッセージが復号できることはない．このプロトコルでは二つのノードを直接結ぶのではなく，間を結ぶルーターが存在する．パケット自体は暗号化されているのでルーターは通信の内容を盗み見ることができない．どのようなノードにルーターの役割を任せるのかはアプリケーション側が任意に選択できるとしている．例として管理者的な特定のルーターを設けたり，親しいノード間においてルーターの役割を果たしたりなどである．

このように一見便利な TeleHash プロトコルであるが，これから利用する場合には問題がある．公式サイト [3] が無くなっており，他のドキュメントのページも 2015 年で更新が止まっていて開発が積極的に行われているとは言い難い状況である． [15, 11]

3.5.2 WebRTC プロトコル

WebRTC（Web Real-Time Communication）プロトコルは P2P 通信でブラウザ間のリアルタイムコミュニケーションを実現するためのプロトコルである．音声通話やビデオ通話などリアルタイム性が求められるチャットツールなどコミュニケーションシーンでの利用例が多い．また，特定のプラグインのダウンロードやインストール無しに使え，標準ブラウザに対応しているので非常に気軽に始めることができる．特に，iOS の Safari に対応しているのでアプリを作ることなくモバイルへの対応もできる．

また，このプロトコルはビデオや音声をリアルタイムで通信するために通信プロトコルとして TCP ではなく UDP を使用している． [82, 34]

第4章　先行研究

4.1　Enigma Project

関連研究として，Enigma [87] およびそれを実現させる Enigma プロジェクト [41] を紹介する．

4.1.1　概要

Enigma プロジェクトは現在のブロックチェーンが課題としているプライバシーとスケーラビリティの問題を解決することを目標としている．イーサリアムなどのブロックチェーンはトランザクションの内容がネットに公開されており誰からでも自由に閲覧できる状態となっている．すなわち送金の履歴やスマートコントラクトの取引の中身・実行の履歴などが全部公開されてしまっている．現在のイーサリアムでは中央集権的な存在を持たずに信頼できる契約の自動化プラットフォームが実現できていることは確かであるが，すべての取引の中身が公開されてしまっているということはこのスマートコントラクトの活躍の場を狭めていることにもつながっている．特に個人情報など秘匿する必要がある情報を秘匿したままではスマートコントラクトを利用できない．

イーサリアムのようにブロックチェーンネットワークに参加しているノードが全て同じ状態に同期されるブロックチェーンにおいては，すべてのノードが同じトランザクションを処理しなければいけない．そのため，処理プロセスが煩雑なスマートコントラクトを実行する場合には非常に時間がかかってしまったりスマートコントラクト手数料の高騰などの問題が生じる．これら二つの問題を解決しようする試みが Enigma プロジェクトである． [52, 41]

4.1.2　Enigma プロトコル

Enigma プロトコルは上記の二つの課題を解決するためにシークレットコントラクトとセカンドレイヤーオフチェーンという二つの特徴を持っている．

シークレットコントラクト

シークレットコントラクトでは秘匿性を保ったままスマートコントラクトを実行することができる．Enigma はセカンドレイヤーとして機能し，元データを暗号化させたままスマートコントラクトを実行することによって秘匿性を保っている．Enigma は将来的にはセキュアマルチパーティ計算の実装を目標としているが，現在公開されているテストネット（シークレットコントラクト 1.0）においてはハードウェアベースのプライバシー保護（Trudted Execution Environment，TEE）のみの実装を行っている．これは内部のデータを保護する機能を持ったハードウェアを利用してハードウェア外部からデータを保

護する仕組みである．リモートで操作するときやクラウドを利用するときなどハードウェアが手元にない場合にはハードウェア外部からの物理的なアクセスによってデータが漏れることが考えられるが，これを利用すればそのようなことを防げる．この TEE のトレードオフとしてそのハードウェア自身が汚染されていないことを信用しなければならない．しかし TEE は秘密計算の手法に比べると非常に早く計算ができるというメリットがある．この TEE と分散ネットワークの強みを Enigma 両方生かしていくとしている．テストネットネットワーク 1.0 においてはこの TEE は Intel SGX [32] というセキュアプロセッサを利用して実現されている． [38, 40, 5, 63]

セカンドレイヤーオフチェーン

Enigma はパブリックブロックチェーンのようなファーストレイヤーだけではなく，その上にあるセカンドレイヤーとして機能する．また，トランザクション処理を全てオフチェーンで行うことにより各ノードがデータの異なる部分を計算することが可能となる．このようにオフチェーンネットワークを使用することによりブロックチェーン技術だけでは解決が難しい以下の課題を解決できるとしている：

- ストレージ Enigma では分散ハッシュテーブル（DHT）を利用してデータを保存している．これによりブロックチェンを利用してアクセスでき，データ自体は参照せずにデータへのポインタを格納する．秘匿化が必要なプライベートデータはストレージ側にアクセスする前にクライアント側で暗号化する．
- プライベートな演算 Enigma ネットワークは個別のノードによる計算を行うので正確な実行を保障しながらいずれのノードにもデータを漏らさずにコードを実行することができる．
- 重い処理の演算秘匿化が必要でない場合でもオフチェーンによる個別のノードによる演算はブロックチェーンへの負荷をかけずに大容量データの計算などを可能にする．これによりイーサリアムなどが直面しているスケーラビリティの問題を解決できる．

[52, 87]

4.1.3 ロードマップ

エニグマの計画しているロードマップを図 4.1 に示す．これはあくまで計画であり，実際には昨年 12 月 24 日に最初のテストネットが稼働したばかりである [37]．ロードマップによると，Enigma には四つの大きなアップデートが存在する．

- Discovery
  Discovery はシークレットコントラクトを実装した最初のリリースとなる．ここでのシークレットコントラクトは TEE を用いたものであり，悪意のあるホストからデータを保護することができる．また，開発者の利便性を考えスマートコントラクト開発はプライバシー保護に関する部分を除きイーサリアムに準拠するように作られている．

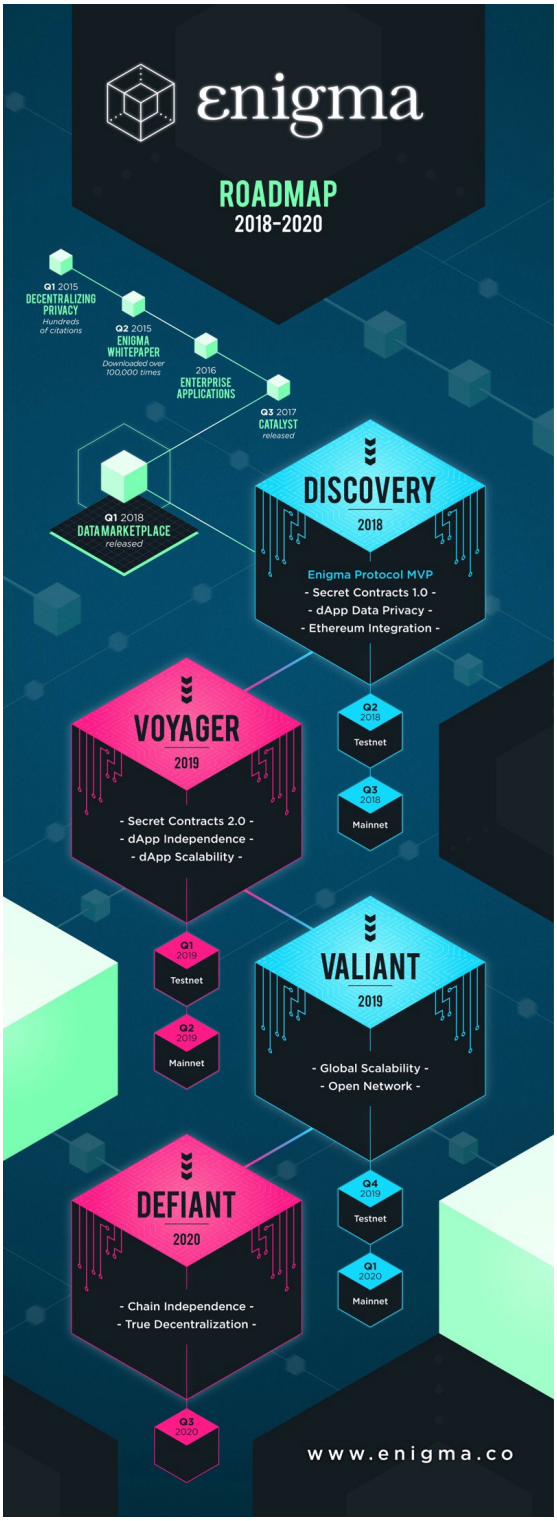


図 4.1: Enigma プロジェクトのロードマップ [39]

- Voyager
  Voyager は DApps のセキュリティ強化に焦点を当てたものとなる．ここではセキュアマルチパーティ計算を利用できるようになる．開発者は TEE かセキュアマルチパーティ計算のどちらを用いるか選択できる．また，Enigma は独自のチェーン（単純なコンセンサスアルゴリズムと限定的な機能を有する）とネットワークを立ち上げ，すべての DApps をその上で動かすようにする．これによりスケーラビリティの向上を図る．最終大型アップデートである Defiant まではセキュリティ強化のためにイーサリアムを親チェーンとして利用する．

- Valiant
  Valiant ではスケーラビリティと分散化を中心に行う．このネットワークアップデートでは Enigma のチェーンをパフォーマンスを低下させずに広く公開されつつも秘匿されたコンセンサスを持つことを目標としている．

- Defiant
  Defiant では他のネットワークからは完全に独立した Enigma ネットワークおよびチェーンを構築する．これにより Enigma での処理は他のプラットフォームに依存しなくなる．セキュリティと分散化を強化する暗号学的プロトコル（特にセキュアマルチパーティ計算周り）のメジャーアップデートもこれに含まれる．

[39, 50]

4.1.4 Enigma アーキテクチャ

ここでは Enigma がどのようにシークレットコントラクト 1.0 を実現させているのかについて述べる．

Enigma ノード

Enigma のノードには計算を行う Enigma ノードとどの Enigma ノードを選ぶかを選択する主ノード（Principal Node）の二種類が存在する．主ノードは一時的な中央集権的ノードであり次の二つの役割を果たす：Enigma ノードを選択するために乱数を生成することとネットワークに他のノードが参加するために鍵を伝搬させることである．この主ノードは開発バージョンのみに存在すし，将来的には完全に非中央集権化したプロセスにより計算ノードを選択する予定である．

Enigma ノードは表層（Surface）とコア（Core）の二つの部分から成る．Enigma ノードの構成図を図 4.2 に示す

表層は Enigma ノードにおいて情報が漏れる可能性がある信頼されていない部分であり，Enigma コントラクトと核の間の計算内容の調整の大部分を担う．表層は計算ノードの選択や計算処理において作用する．コアは Enigma ノードにおいて情報が漏れないと信頼されている部分であり，計算処理の実行を行う．コアは Intel SGX に対応した CPU の SGX Enclave Core 内で動作する．ここで計算や登録，暗号化，検証などを行う．Enigma コントラクトはこれらが正しくコア内で行われたかをチェックする機能をもち，正しく計算が行われたということを保障する． [63]

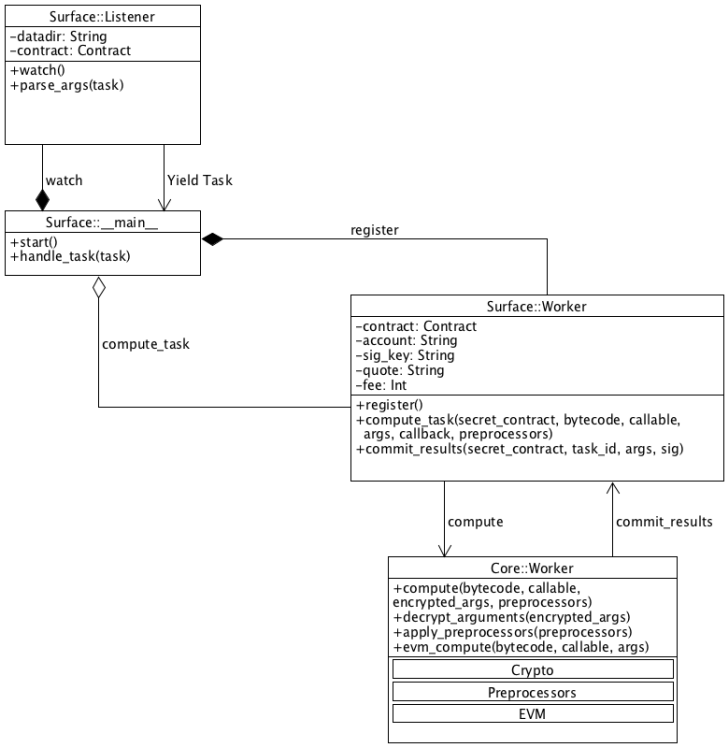


図 4.2: Enigma ノードの構成 [65]

論理構成

Enigma システムは単独で動作するシステムではなく七つの論理的な構成要素からなる.

1. Enigma Library (EnigmaP.js) DApp 内に存在する Enigma の機能を呼び出す Javascript ライブラリ.
2. DApp Contract DApp 作成者により作成された，暗号化されたデータ・計算式・呼び出し等を含むスマートコントラクト.
3. Enigma Contract イーサリアムネットワーク上にデプロイされた Enigma ネットワークのチェーン上の命令をコントロールするスマートコントラクト.
4. Surface Enigma ノード内で信頼されていない部分．Enigma コントラクトとコアを結びつける働きを主に担う.
5. Core Enigma ノード内で信頼されている部分．SGX Enclave 内で動作し，計算の実行を担う.
6. Principal Node ネットワーク上の他のノードに乱数を渡すための一時的な中央集権的ノード
7. Attenstation Service ローカルネットワークには存在しない見積りを行う独立したサービス

このうち 1 から 5 までの様子を図 4.3 に示す.

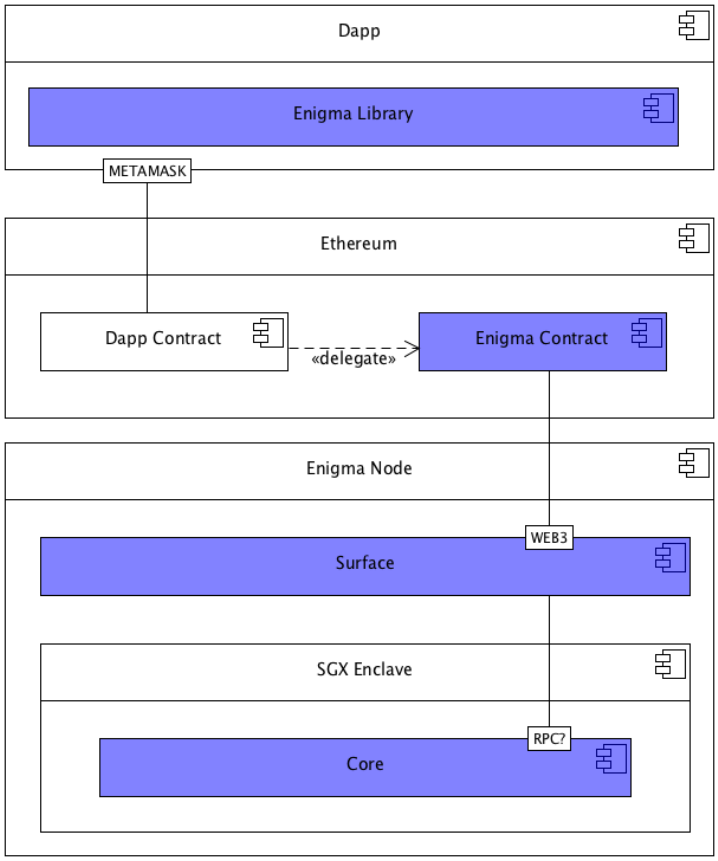


図 4.3: Enigma の論理構成図 [64]

計算の様子

ここでは Enigma がどのように分散計算を行うのかを説明する．計算を投げたいエンドユーザーがいたとする．まずはエンドユーザーがブラウザ上の DApp を利用し，その中で使われている Enigma Library を通してデータの暗号化とエンドユーザーが正しく信頼されているハードウェア（Intel SGX）を使っているかの認証を行う．次にエンドユーザーが Enigma コントラクトを利用して Enigma ネットワーク上に計算要求を投げる．計算要求を受け取ると登録されている各計算ノードは信頼されているハードウェア内の真の乱数生成モジュールを利用して乱数を作成する．その乱数を利用して計算するノードを選択し，エンドユーザーから暗号化された計算タスクを受け取る．これらの様子を図 4.4 に示す．

選択された計算ノードは内部の信頼できるハードウェア内でデータを復号し計算を行う．実行後は入力・出力・命令をハッシュ化し信頼できるハードウェア内にのみ存在する秘密鍵で署名を行う．それをチェーン上に載せることで正しく計算を行ったことを証明する．最後に Enigma ネットワークを通じてエンドユーザーまで暗号化した答えが届けられ，計算ノードは報酬を得る．これらの様子を図 4.5 に示す． [63]

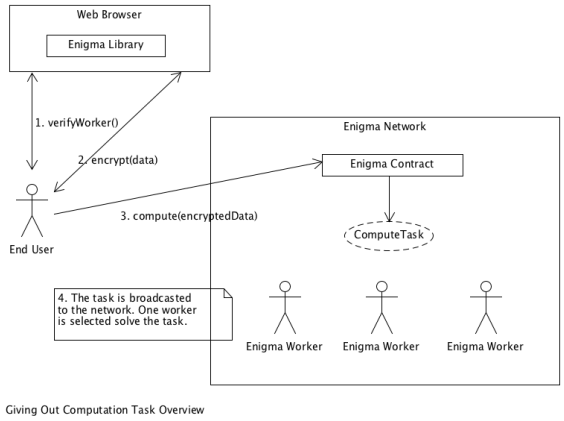


図 4.4: エンドユーザーが計算を投げる様子 [64]

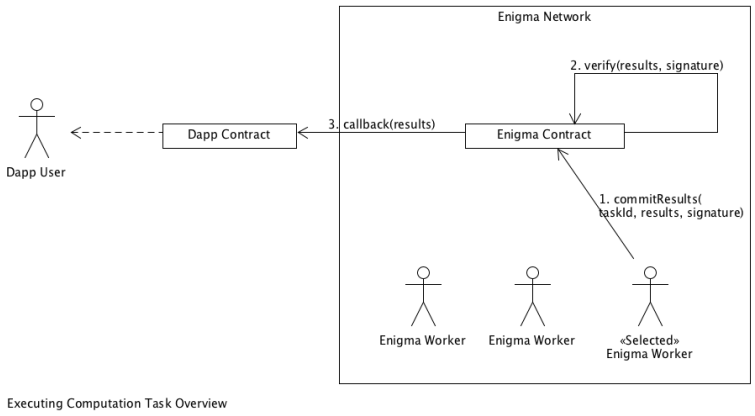


図 4.5: 計算ノードから計算結果を受け取る様子 [64]

4.1.5 まとめ

Enigma プロジェクトはプライバシー保護に配慮した分散計算ということで今回私が行った研究と方向性が非常に似ている．その意味においても Enigma プロジェクトがどのようにセキュアな分散コンピューティングを実現しようとしているのを見ることは非常に重要である．しかし Enigma にも課題があり，私の研究ではそれらをどのように解決できるかということに関しては次章で述べる．

4.2 ブロックチェーンによるセキュアな分散処理を実現するニューラルネットワークの実装

Johjima ら [51] は，ブロックチェーンネットワーク上で異常検知可能なニューラルネットワークの分散処理を行うシステムのシミュレーションを行った．

4.2.1 概要

ここでは全世界に大量に存在する IoT デバイスの余剰計算資源に機械学習で広く使われているニューラルネットワークの計算を分散させて行わせようとしている．ニューラルネットワークとは人間の脳の仕組みを模した計算モデルであり，レイヤーと呼ばれる複数の計算ノード（ニューロンと呼ぶ）の集合体が重なり構成されている．あるレイヤーのニューロンは前のレイヤーの複数ニューロンから計算を受け取り内部で計算を行い，次のレイヤーの複数ニューロンに渡す．入力から値を受け取るレイヤー・結果を出力するレイヤー・間のレイヤーをそれぞれ入力レイヤー・出力レイヤー・隠れレイヤーと呼ぶ．

基本的にニューラルネットワークはニューロン数が多くなるほど高い学習精度が得られる．しかし，ニューラルネットワークの内部は機械により自動で学習が行われるためブラックボックスとなっており，どのニューロンが強い影響力を持つのかなどの情報を得ることができない．ニューロンは同じ学習デー

タでも学習ごとに異なる学習結果を得るためにニューロンごとの正しさを判断するための指標が存在しない．そのため外部からニューロン間の値のやりとりを確認したとしてもそのニューロンが正しく計算しているかの判別ができない．このような性質がニューラルネットワークの分散処理を実現する際に問題となる．

そこでこの先行研究ではニューラルネットワークをニューロンに分けた分散処理を行い，各ニューロンとして割り当てられたノードが正しく計算を行っているのかをブロックチェーンを用いて監視・検証を行うシステムを提案した．

4.2.2 実験手法

この実験においてはニューラルネットワークの設計は python で行い，入力レイヤーのニューロン数は４・隠れレイヤーのニューロン数は６・出力レイヤーのニューロン数は３である．ニューラルネットワークにおけるニューロンのパラメータをイーサリアム上のスマートコントラクトを利用して取得し，その結果を確認している．

この実験でのニューロンの異常検知を行う動作順序を説明する．

1. １ニューロンを１デバイスとしてニューラルネットワークの計算を分散して行う．
2. 異常行動をしているか確認するデバイス（監査対象デバイス）を選択する．
3. 監査対象デバイスはスマートコントラクトを通じて計算の入力と出力をイーサリアムネットワーク上で共有する．
4. 監査対象デバイスに入力を提供したデバイスと監査対象デバイスから計算結果を受け取ったデバイスも監査対象デバイスとやりとりした値をイーサリアムネットワーク上で値を共有する．
5. これら監査対象デバイス周りのデバイスで通信を行った値に関して検証を行う．監査対象デバイスが正常に値を受信しているか・入力値を用いて正常に計算しているか・正常に値を送信しているかの判定を行う．
6. 異常検知を行うデバイスを監査ノードと呼ぶ．この監査ノードはイーサリアムネットワーク上の値を一定時間ごとに自動で取得し，イーサリアムネットワーク上に検証に必要な値がそろったらそれらの値を利用して５の条件を満たしているかを確認する．

これらの仕組みを図 4.6 に表す．

実験では意図的にデバイス上でニューロンに異常行動を起こし，その異常を監査ノードで検知することができること・異常が起きてない状況で誤検知することがないことを確認した．実験環境は一台の PC 上で仮想的にニューラルネットワークの計算を行い，並行して監査ノードによる監査を行う環境下で実験を行った．

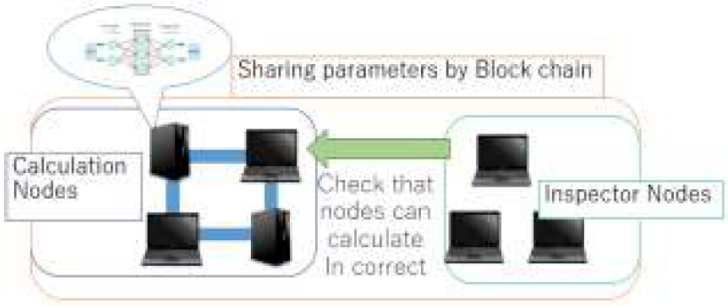


図 4.6: ニューラルネットワークの計算を監視している様子 [51]

4.2.3 実験結果

誤動作するニューロンを含まない正常なニューラルネットワークの計算においては監査ノードがチェックを行った結果正しく正常であると判定し，ランダムに値を送信するような悪いニューロンを組み込んだニューラルネットワークにおいては，監査ノードが異常動作を正く検出できた．

4.2.4 まとめ

この実験では，イーサリアムのスマートコントラクトを利用してニューラルネットワークの分散計算を正しく行えるのか試している．分散計算では計算を任せた相手が正しく計算してくれるという保障が無いだけにどのように計算の正しさを保障するのかが問題となる．ここでは計算結果を公開してよいという前提のもと，計算プロセスをブロックチェーン上に公開しその検証を行うという手法をとった．これから増大する IoT 機器の余剰計算能力でも実現可能という点で本研究は非常に魅力的である．

第5章　提案手法

本研究ではイーサリアムのスマートコントラクトのみでは不可能であったデータの秘匿性を保ったままの分散計算を，イーサリアム DApp プラットフォーム上でセキュアマルチパーティ計算を用いて実現する手法を提案する．

5.1　関連研究の問題点

ここでは「秘匿性を保った分散コンピューティング」という観点から本研究に非常に近い Enigma に関しどのような点がセキュアな分散コンピューティングを実現する際の懸念点として挙げられるかを述べる．

5.1.1　Secrect Contract

データの秘匿性を保ったまま計算を行うシークレットコントラクトに関する懸念点を以下に述べる：

- 独自の通信プロトコルを用いている
  Enigma がノード間の通信で用いるプロトコルの詳細は公開されていない．場合によって脆弱性などが見つかりプライバシーデータの漏洩につながる可能性がある．Enigma からの情報が漏れないと信頼して使うしかないことになっている．

- 現在の構成では Intel SGX のみ対応している
  Intel SGX を搭載している CPU は Intel の Skylake アーキテクチャ以降の CPU のみである．スマートフォンなどに搭載されている CPU の多くは非対応であり Enigma システムを利用できるノードが限られている．

- 主ノードという一時的な中央集権的ノードを設定している
  開発者バージョンのみにおいてであるが，計算ノードの選択の際に主ノードという一時的に中央集権的ノードを設定している．この主ノードが本当に信頼できるノードなのかという懸念がある．主ノードが汚染されていたら恣意的にノード選択が行われる可能性が存在する．

- 将来セキュアマルチパーティ計算を使用する際に SPDZ プロトコルの利用も示されている
  将来のセキュアマルチパーティ実装に関して，SPDZ という外部から検証可能なセキュアマルチパーティ計算のプロトコルについて言及されている．この SPDZ は準同型暗号を利用しているため計算量が大きくなってしまうことが考えられる．

5.1.2 独自チェーンの構築

Enigma は独自のチェーンやオフチェーンレイヤーを利用しスケーラビリティを解決するとしている．独自のチェーンを利用することに関する懸念点を挙げる：

- 独自のチェーンを利用してくれる人の規模に大きく依存する
  イーサリアム等とは独立して独自のチェーンを利用するということは，Enigma システムを使う人が少ないことはすなわち Enigma ネットワークの検証を行う人が少ないということを意味する．ネットワークの検証を行うマイナーが少ないと 51 %攻撃などの計算量を利用した攻撃が成功しやすくなるということであり，ブロックチェーンの信頼性が脅かさることとなる．

- Enigma 通貨が他の通貨と切り離されて独自の価値を持つ
  Enigma は独自のチェーンと通貨を利用するとしているが，それはすなわち Enigma 通貨がビットコインやイーサリアムと独立して価値を持つということである．Enigma に計算ノードとして計算リソースを提供した人が受け取る Enigma 通貨に価値が無いと計算リソースが提供されにくくなる．通貨の流通量が少なければ価格の変動が大きくなったり安定性を欠いたりし，計算を任せたい側と計算を行う側での価格の合意が難しくなることも考えられる．

5.1.3 本研究での改善案

Enigma に関して挙げた懸念点に関し，本研究では（未実装のものも含めて）どのように解決していくかについて述べる．

- 独自の通信プロトコルを用いている → web3 の whisper プロトコルを使用する
  通信プロトコルとして公開されている web3 の whisper プロトコルを用いることで，独自に安全な通信プロトコルを開発する必要がない．また，脆弱性があったとしても whisper 側が対応してくれることを期待できる．さらに，イーサリアム Javascript API の web3 を用いているので本研究システムとイーサリアムネットワークやクライアントとの接続が容易である．すなわち本研究を利用したアプリケーションの開発がしやすいということである．

- Enigma の現在の構成では Intel SGX のみ対応している → ブラウザが動く端末に対応
  セキュアマルチパーティ計算を各計算ノードのブラウザ上で動作する DApp で行うのでブラウザが動くスマートフォンなどの軽量な端末などでも動作させることができる

- 主ノードという一時的な中央集権的ノードを設定している → 個別に計算ノードを設定
  これは本研究では実装まで至ってないが，whisper のブロードキャストを用いて計算要求を投げ，それに応じた計算ノードに計算を託すことを予定している．詳細は節 5.2 で説明する．

- セキュアマルチパーティ計算で SPDZ を使用する可能性 → 準同型暗号を使用しないセキュアマルチパーティ計算を採択
  本研究で用いているセキュアマルチパーティ計算では準同型暗号を使用しないため，計算能力が高くない端末にも計算を行わせることができる．

- 独自のチェーンを利用してくれる人の規模に依存・Enigma 通貨が他の通貨と切り離されて独自の価値を持つ → イーサリアム上のシステムを利用し，やりとりする通貨もイーサとする
  仮想通貨時価総額ランキング 2 位（約 2.1 兆円）[30] という仮想通貨では超大手のイーサリアムのシステムを利用できるので利用者が参加しやすい．イーサを計算システムの取引通貨として利用すれば価格のボラティリティが少ないため計算を託す人・計算を行う人が計算料のやりとりを行いやすい．また，詳細は節 7.3.2 にて説明するがマイナーにこの DApp を利用してもらえれば常時計算ノードが確保できることになる．

5.2 提案プロトコル

提案するシステムの動作プロトコルを以下に示す：

1. 計算を託すノード（以下クライアントノード）がブロードキャストでアクティブな計算可能なノード（以下計算ノード）に計算要求を呼びかける
2. 計算要求を受け取った計算ノードは，自身の計算の受け入れやすさに応じた待機時間に変化を付けて応答
3. 複数の計算ノードから返答を受け取ったクライアントノードは，各計算ノードの成績をブロックチェーン上のデータから参照して計算を託すノードを選ぶ
4. クライアントノードは計算を託すノードを決め，それらの計算ノードでグループを組む
5. クライアントノードは各計算ノードに計算内容と各々が所属するグループの名簿を送る
6. 計算内容と名簿を受け取った計算ノードは名簿に登録された他の計算ノードと協力してセキュアマルチパーティ計算を行い，結果をクライアントノードへ送る
7. 複数グループのセキュアマルチパーティ計算の結果を受け取ったクライアントノードは，それぞれが正しく計算を行っているか検証する
8. クライアントノードは答えを取得し，各計算ノードの成績をブロックチェーン上に登録する
9. クライアントノードは紐づけられているアカウントから計算料を各計算ノードに対応するアドレスに支払う

これらの一連の流れを図 5.1 に示す．

5.3 新規性

通常のセキュアマルチパーティ計算では計算を行うパーティに紛れ込む不正なノードとして semi-honest モデルと malicious モデルを考えている．本研究のように計算ノードに報酬を支払うモデルでは計算を行ったふりをして計算報酬を受け取る不正なノードが存在することが考えられる．このようなノードを lazy（怠け者）ノードと呼び，このノードが混入したモデルを lazy モデルと呼ぶことにする．

5.3.1 lazy ノードの動作

lazy ノードの動作はなるべく計算を行わずに報酬を得ようとするため，以下のような動作をすると仮定する：

- 入力を受け取ったら無作為な値を返す
- パーティ間の連携についても無作為な値を渡す

すなわち内部で計算を行わず，あたかも計算を行ったかのように外部に（見せかけの）計算結果を渡すノードのことである．

5.3.2 新規性

上記の悪意を持った三種類のノードが存在していても高い確率で正常に動作するために，以下の三点を本研究では新しく提案する：

- セキュアマルチパーティ計算を別個に行うグループを複数作成する
- グループごとにシェアの内容を分ける
- 計算結果の成績を登録

以下にこれらの提案を行う理由を示す．

セキュアマルチパーティ計算を別個に行うグループを複数作成する

本研究で採用しているセキュアマルチパーティ計算のプロトコルではエラーを検知することはできるもののエラーを検知した段階でそのグループの計算を止めてしまう．すなわち，グループの中に計算を止めてしまうような悪意のあるノードが一つでも混入していたら計算ができない．また，計算が最後まで実行されて答えが導出できたとしてもそれが本当に正しい答えなのかの保障が存在しない．そこで，複数グループにセキュアマルチパーティ計算を行わせることで計算が止まった場合の耐障害性を確保する．また，複数グループの計算結果が一致していればそれを正しく計算が行われた結果導出された解答とする．

グループごとにシェアの内容を分ける

複数グループにセキュアマルチパーティ計算を行わせるということは計算を行うノードが増えるということである．その際考えられる懸念として，攻撃者の支配下にあるノードが複数のノードに混入してしまうことがある．その際に複数グループでシェアの分割方法が同じならば，複数グループのパーティからシェアの復元が可能となってしまう（図 5.2）．計算の正しさを確保するためにセキュアマルチパーティ計算の利点である計算の秘匿性を下げることがないように熟慮した結果，グループごとにシェアの内容を分けることとした．シェアは乱数を用いて作成しているため，グループごとに別の乱数を用いればシェアが全く異なるものとなる．別々のグループに攻撃者の支配下にある悪いノードが混入していた

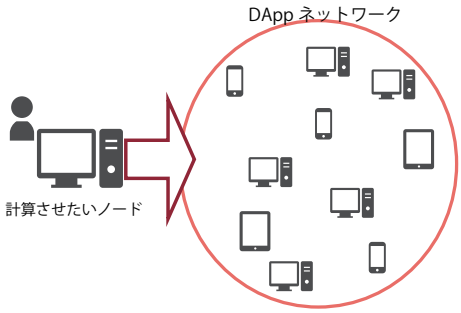
としても，別グループのシェアは元データの復元に役に立たない.(図 5.3)．したがって，元のセキュアマルチパーティ計算にて保障していた閾値 $(t < n/2)$ 以下の場合における計算の安全性を損なうことはないまま lazy ノードの混入への耐性を上げることが可能となる．

計算結果の成績を登録

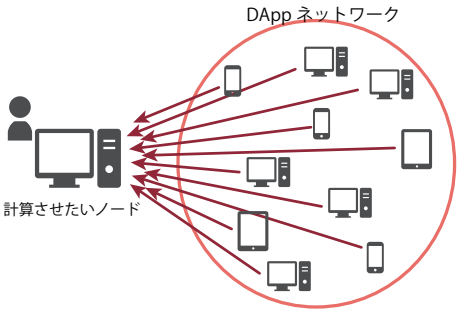
今回のセキュアマルチパーティ計算は実際の運用では計算を行ったノードに計算料を支払うということを想定している．各計算ノードが過去に行った計算の成績を登録することで，計算するノードの際にその成績を参照し常に間違いを出し続けている lazy ノードの採択を避けたり常に正しい計算を行う正直なノードの採択率を上げるということが可能となる．

計算ノードにおいて計算の受け入れやすさを設定

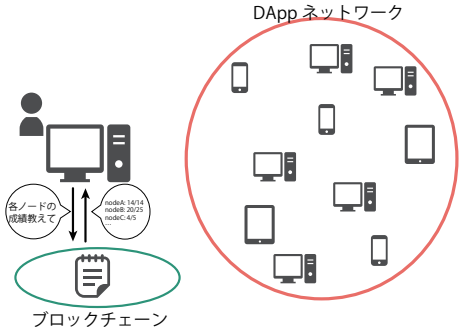
計算ノードの選択対象となるアクティブなノードは様々なスペックのノードが混在することが考えられる．また，計算ノードの中にはバッテリーで動作するものもあり，アクティブなノードすべてにランダムに計算を投げていたらノードの利用者が意図しないうちにバッテリーの残存電力を消費したり限りある計算リソースを他人の計算で奪うことにつながりかねない．そのような事態を避け，かつ計算を行いたいノードがより積極的に計算を受け入れられるように計算の受け入れやすさをノードごとに設定できるようにする．そうすることにより計算ノードを所持している人が状況に応じて適切に余剰計算能力をこのシステムに提供することが可能となる．



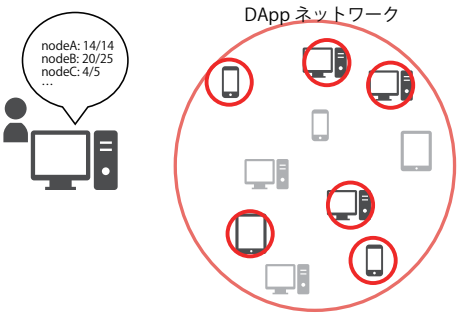
(a) 計算させたいノード（クライアントノード）が DApp ネットワーク上でアクティブなノードに計算要求をブロードキャストで投げる．



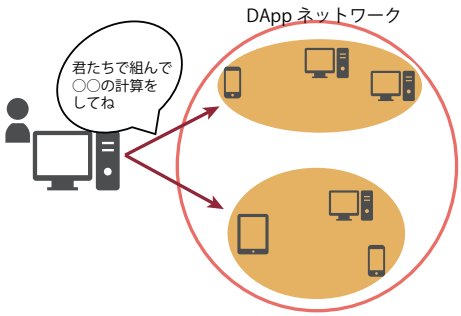
(b) アクティブなノードのうち計算を行いたいノードが返答する．



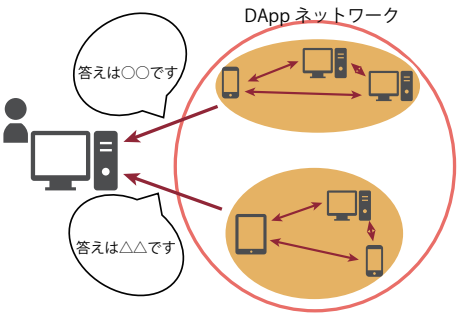
(c) クライアントノードは返答があったノードの成績をブロックチェーン上で参照する．



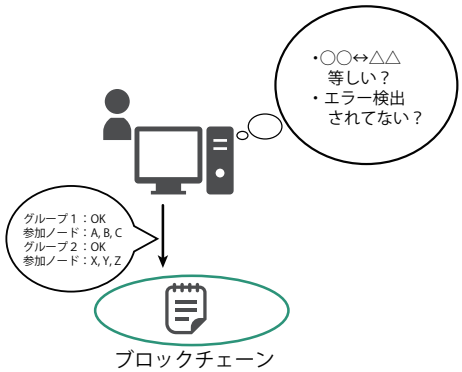
(d) 返答があったノードの中から過去の成績を参考に今回計算をさせるノード（計算ノード）を選択する．



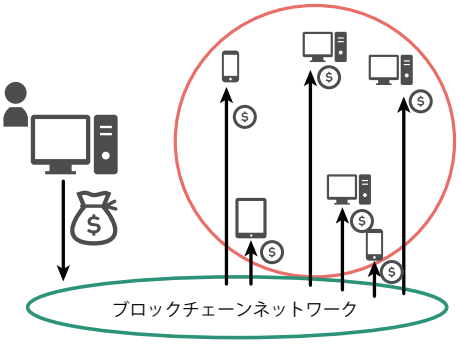
(e) クライアントノードは各計算ノードに計算内容と組んだグループの情報を送信する



(f) 各計算ノードは決められたグループ内で通信を行いセキュアマルチパーティ計算を実行する．答えが導出できたらクライアントノードに送る．



(g) 複数グループから送られてきた答え同士を比較して検証する．検証した結果を元に今回の計算ノードの成績をブロックチェーン上に登録する．



(h) ブロックチェーンネットワークを用いてクライアントノードは各計算ノードに計算料を支払う．

図 5.1: 提案するシステムの動作概略図（その 2）

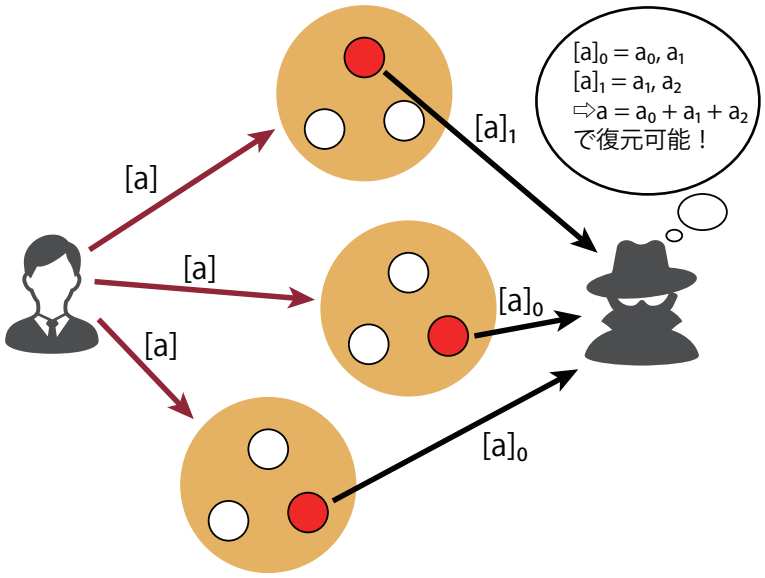


図 5.2: シェアの分割をグループごとに分けない場合．複数グループに攻撃者の支配下にあるノードが散逸して存在していた場合に攻撃者が元のデータを復元できてしまう．

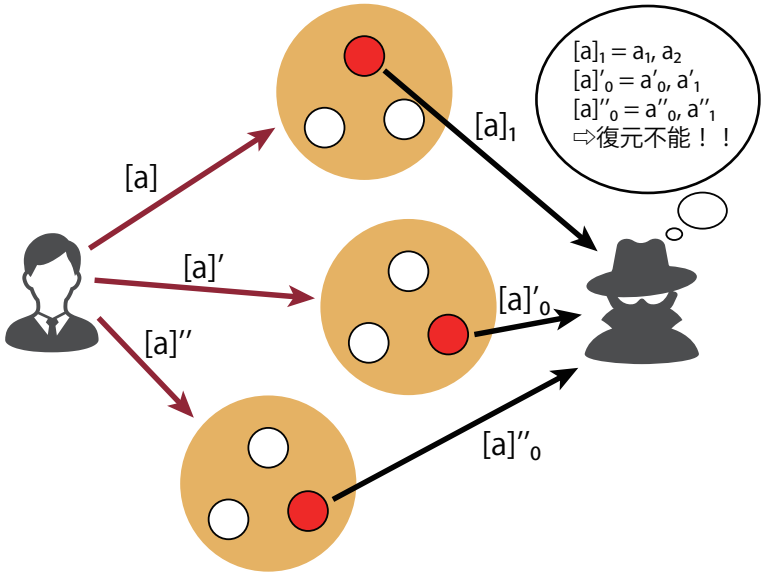
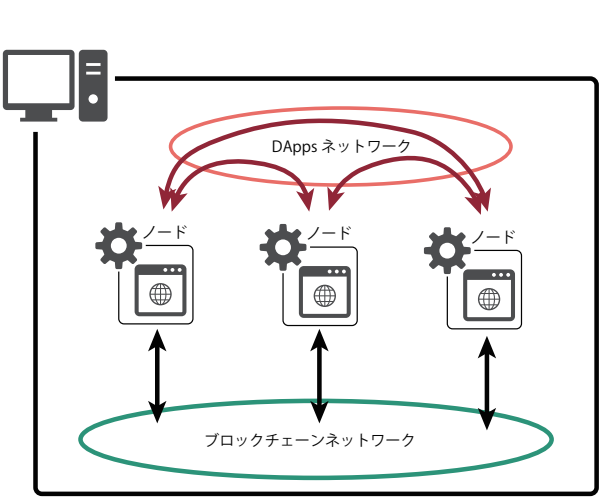


図 5.3: シェアの分割をグループごとに変えた場合．攻撃者が複数グループ上のパーティから値を取ってきてもそれらは互いに意味を持たないので元のデータの復元ができない

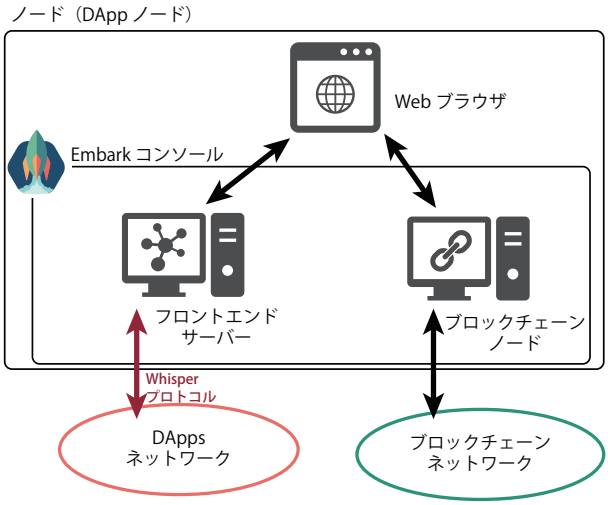
第6章　実験

6.1　実装

節 5.2 にて説明したプロトコルのうち，5. から 8. の部分の実装をフロントエンドアプリケーションとして Javascript を用いて行った．実装にあたっては DApps 開発フレームワークである Embark を用いた．また，DApp 間の通信プロトコルとしてイーサリアム Javascript API である Web3 [9] の whisper という通信プロトコルを用いた．一つのパソコン内でイーサリアムプライベートネットワークを立ち上げ，そこに複数の Embark で起動したそれぞれのノードを接続している（図 6.1）．あらかじめどのノードがどの役割を担うのかは固定して実験を行った．



(a) 実装したシステムの全体図．複数ノードを立ち上げてそれらの間で通信を行っている．



(b) システム内のノードの様子．Embark を用いて Web ブラウザとフロントエンドアプリケーション・ブロックチェーンノードを接続している．

図 6.1: 実装したシステム概略図

6.1.1　Embark

Embark [36] は DApp のフロントエンド開発に注力しており，最新版の whisper や swarm に正式に対応している．また，スマートコントラクトを所定のフォルダに置くだけで自動でコンパイル・デプロイをするのでテストネットワークの稼働を自分でする必要がない点で非常に便利である．Embark ネイティ

ブでフロントエンドのウェブサーバーを稼働させているので react [66] や meteor [54] など外部のフロントエンドサーバーを利用することなく DApp の動作を確認できる．

この embark はスマートコントラクトのデプロイに Ganache [77] というイーサリアムローカル開発環境を利用している．これとスマートコントラクト開発言語 Solidity のコンパイラ solc を合わせることにより，DApp のディレクトリ内に配置したスマートコントラクトを自動でコンパイルしイーサリアムブロックチェーン上にデプロイすることを可能としている．また，自動でデプロイだけでなくマイニングも行ってくれるのでデプロイすることやスマートコントラクトを実行することで gas が要求されても必要に応じてマイニングを行い手元に十分なイーサを確保してくれる．

6.1.2 非同期処理

本実験で実装したフロントエンドアプリケーションで利用した Web3 はバージョン 1.x.x 系以降では非同期処理を採用している．これによりイーサリアムブロックチェーン等からの返答を待つことができる上にシングルスレッドのメインプロセスにおいては待つことなく別の処理を進めることができる．

クライアントノードが計算を行うということは自身が計算前のシェアを計算ノードに投げ，計算ノードから計算結果のシェアを受け取る必要がある．Whisper で用意されている受信関数 subscribe はイベントを検知して子プロセスを実行する仕様になっている．最初に subscribe を呼び出して待機させておき，計算が進んで計算ノードからメッセージを受信したら条件に応じた subscribe の子プロセスが実行されて答えを導く仕様となっている．このようにクライアントノードは計算を行い外部に送信し，別の関数上で結果を受け取るという構造になっている．そのため通常の関数みたいに引数に計算する内容を渡した関数を呼び出すことで計算を実行し，返り値で答えを返すということの実装が難しい．

6.1.3 Whisper

Whisper の送信関数 post においては引数に以下のオブジェクトを設定できる：

- `symKeyID - String`(optional):メッセージの暗号化に必要な共通鍵の ID（`symKeyID` または `pubKey` のいずれかが必要）
- `pubKey - String`(optional)：メッセージの暗号化に必要な公開鍵（`symKeyID` または `pubKey` のいずれかが必要）
- `sig - String`(optional)：署名鍵の ID
- `ttl - Number`：Time-to-Live，メッセージの生存時間（秒）
- `topic - String`：メッセージのトピック（4 バイト長）
- `payload - String`：暗号化するメッセージの内容
- `padding - Number`(optional)：パディング（任意の長さのバイト列）
- `powTime - Number`(optional)：Proof-of-Work に費やす時間の最大値（秒）

- `powTarget - Number`(optional)：このメッセージに必要な Proof-of-Work のターゲットの最小値
- `targetPeer - Number`(optional)：相手の Peer ID（P2P メッセージの場合のみ）

セキュアマルチパーティ計算でパーティ間の通信は個別の通信であるから `targetPeer` を利用したかったのだが，今回構築した環境では接続されている Peer の数を数える getPeerCount 関数を用いて調べても 0 となり構築している複数の embark 実行環境が Peer として認識されなかった．そのため全ノード共通の共通鍵を用いて暗号化し，受信関数 subscribe でも指定する `topic` を用いてお互いのノードが受信するべきメッセージを区別した．

共通鍵とその ID の作成は専用の initialize ノードを用意して行った．この intialize ノードは起動すると Whisper の newSymKey 関数を利用して新しい共通鍵を作成する．その共通鍵を引数に addSymKey 関数を実行すると共通鍵の ID が入手できる．そのように作成した共通鍵と ID を全ノードで共有することによりメッセージの暗号化を行った．

6.2 プロトコル

実装したセキュアマルチパーティ計算のプロトコルは Chida ら [29] のプロトコルをベースとした．複数グループでセキュアマルチパーティ計算を行っているが，違いはシェアの分割方法だけでありプロトコルに違いはないので以下の説明では一つのグループのパーティ P_i $(i = 0, 1, 2)$ で説明する．入力 $a \in \mathbb{Z}/m\mathbb{Z}$ の秘密分散は，P_i がもつシェアを $[a]_i = (a_i, a_{i+1})$ とする．なお，$i \pm 1$ は 3 で割った余りとみなす．すなわち $i = 2$ であれば $a_{i+1} = a_0$ である．Chida らのプロトコルとは異なる部分を下線で表す．

ここでは，計算命令・計算する内容を投げ，計算結果を受け取るノードをクライアントノードとし，クライアントノードから数値を受け取り実際の計算を行うノードを計算ノード，もしくはパーティとする．

6.2.1 クライアントノード（前半）

シェアの作成

まずは計算ノードへ送信するシェアを作成する．a をクライアントノードの入力としてパーティ P_i に $[a]_i$ を送るため以下の makeShare プロセスによりシェアを作成する：

makeShare：シェアの作成

入力：$a \in \mathbb{Z}/m\mathbb{Z}$

1. a_0，$a_1 \in \mathbb{Z}/m\mathbb{Z}$ をランダムに生成する．
2. $a_2 := a - a_0 - a_1$ を計算する．
3. $i = 0, 1, 2$ について，$[a]_i := (a_i, a_{i+1})$ とする．

例として和 $a + b$ を行う場合には上記プロセスを a と b 両方に対して行い，a のシェア a_0, a_1, a_2 と b のシェア b_0, b_1, b_2 を作成する．

シェアと命令の送信

次に作成したシェアと計算命令を各パーティに送信する：

sendOrder：シェアと命令を送信

入力：$a_0, a_1, a_2, b_0, b_1, b_2, m$, 算術命令

1. 各 P_i に算術命令, $m, [a]_i, [b]_i$ をリスト化したものを Whisper プロトコルで送信

クライアントノードが自発的に行うプロセスはここまでである．これ以降は各パーティから受信したデータを元に以下のプロセスが実行されるので，計算ノードの動作を説明したあとに記載する．

6.2.2 計算ノード（パーティ）

計算ノードはクライアントノードから上記 sendOrder により送られたデータを基に計算を行う．行う演算に関係なくまずはシェアの一致性検証を行う．

シェアの一致性検証

パーティにおけるシェアのエラー検出手続きとして以下の EqTest プロセスを行う．

EqTest：複数のパーティが共有しているシェアの一致性検証

入力：$P_i([a]_i)$
出力：$0/1$

1. P_i は $\alpha_{i+1} := a_{i+1}$ を P_{i+1} に送信する．
2. P_i は $\alpha_i = a_i$ であれば 1 を，そうでなければ 0 をクライアントノードに送る．

次に，送られてきた算術命令に応じて以下の加算もしくは積算を行う．

加算

加算の場合は以下の Add プロセスを行う．

Add：a, b のシェアから $a + b$ のシェアを計算

入力：$P_i([a]_i, [b]_i)$
出力：$P_i([a + b]_i)$

1. P_i は $[a + b]_i := (a_i + b_i, a_{i+1} + b_{i+1})$ を計算する．
2. P_i は $[a + b]_i$ をクライアントノードに送る．

乗算

乗算の場合は以下の Mul プロセスを行う．

Mul：a, b のシェアから ab のシェアを計算

入力：$P_i([a]_i, [b]_i)$
出力：$P_i([ab]_i)$

1. P_0 は $r_1, r_2, c_0 \in \mathbb{Z}/m\mathbb{Z}$ をランダムに選択してから，$c_1 := (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 - c_0$ を計算して P_1, P_2 にそれぞれ $(r_1, c_1), (r_2, c_0)$ を送信し，$[ab]_0 := (c_0, c_1)$ とする．
2. P_1, P_2 はそれぞれ $y := a_1b_2 + a_2b_1 + r_1, z := a_2b_0 + a_0b_2 + r_2$ を計算して P_2, P_1 に送信する．
3. P_1, P_2 は $c_2 := y + z + a_2b_2$ を計算してそれぞれ $[ab]_1 := (c_1, c_2), [ab]_2 := (c_2, c_0)$ とする．
4. P_i は $[ab]_i$ をクライアントノードに送る．

6.2.3 クライアントノード（後半）

ここでは計算ノードで演算が行われたあとにクライアントノードで行うプロセスを示す．受信するデータにはヘッダを付加していて受信したデータが何を意味しているのか分かるようになっている．

シェアの一致性検証後

malicious モデルにおけるシェアのエラー検出として使われるシェアの一致性検証 EqTest を各パーティで行った後に，クライアントノード側ですべてのパーティにおいて EqTest の結果が 1 であるか確認する EqTestCheck を行う．

EqTestCheck：全パーティにおけるシェアの一致性検証の確認

入力：0 or 1
出力：true or $\perp$

1. P_0, P_1, P_2 から受け取る値が全て 1 ならば true を返して以下の復元プロセスに進み，そうでなければ異常を示す $\perp$ を返して該当パーティの計算を終了する．

この一致性検証について Chida らは malicious モデルにおいてエラー検出が可能であるとしているが，lazy モデルにおいても有効である．lazy ノードは EqTest の 2 において偽の判定を送ることは外部からは不正をしているかわからずに行うことができるが，これは結局自身が保持するシェアを書き換える操作に等しく EqTest におけるエラーとは見なさない．隣のシェアに送る値は正しいものを送らないと隣のシェアでエラーが検知されてしまうため，完全にランダムな値を送る lazy ノードはこの段階で検知できる．

シェアの復元

各パーティから送られてきた答えのシェアを以下の Dec プロセスで復号する．

Dec：2-out-of-3 秘密分散のエラー検出を含む復元

入力：$P_i([a]_i)$
出力：a or $\perp$

1. P_i は $(\alpha_i, \beta_i) := (a_i, a_{i+1})$ を開示する．
2. $i = 0, 1, 2$ について $\beta_i \neq \alpha_{i+1}$ となる β_i, α_{i+1} が存在すれば，異常を示す $\perp$ を返して終了する．
3. $a = \alpha_0 + \alpha_1 + \alpha_2$ を計算する．

このように答えのシェアから答えを復元し，グループごとに行ったセキュアマルチパーティ計算の答えが一致すればそれを正答として扱う．シェアの一致性検証をくぐり抜るような lazy ノードがいるグループはこの Dec プロセスにおいてエラーが検出される．

6.3 実験環境

本研究における実験環境ハードウェアを表 6.1 に，ソフトウェアを表 6.2 に示す．

CPU	Intel Xeon E5-1603 v3
メモリ	DDR4 ECC 1866 MHz 4GB × 4
ディスク 0	SSD 256 GB
ディスク 1	HDD 2 TB
GPU	NVIDIA Quadro K620

表 6.1: 使用したハードウェア構成

6.4 実験結果

6.4.1 計算の様子

実際に計算を行った様子を図 6.2〜図 6.6 に示す．加算の様子は図 6.2 と図 6.3，乗算の様子は図 6.4〜図 6.6 である．それぞれ各ノードの Web ブラウザ上のコンソールに結果を表示するようにしたため，そのコンソールのログをスクリーンショットで撮っている．

6.4.2 計算時間

和と積のセキュアマルチパーティ計算を二つのグループで実行し，それぞれの経過時間各プロセスにおける所要した時間を計測した．また，比較のためにクライアントノード上で Javascript にて実行した

機能	ソフトウェア名	バージョン
OS	Windows10	バージョン 1909
Web ブラウザ	Firefox	72.0.2 (64bit)
開発環境	embark	4.2.0
フロントエンド API	web3.js	1.2.4
通信プロトコル	whisper	v6
イーサリアムローカル開発環境	Ganache	v2.1.2
スマートコントラクト使用言語・コンパイラ	Solidity・solc	v0.5.12
フロントエンド言語	Javascript	ES2016 (ES7)
サーバーサイド Javascript 環境	node.js	v11.13.0
パッケージ管理ソフトウェア	npm	6.9.0

表 6.2: 使用したソフトウェアおよびそのバージョン

演算の所要時間も載せる．所要時間の計測においては，各プロセスの節目に Unix 時間のミリ秒を返す関数 Date.now() を利用し，それらの差分を取ることで所要時間を算出した．途中で通信がロストし計算に失敗した場合を含めず 10 回計測を行い，それぞれの結果と平均を載せる．

計測したノードはクライアントノード（以下 CL と表記）と，二つのグループのうちの一つ（グループ A とする）の P_0, P_1（以下 X，Y と表記，P_2 も Z と表記）の計三つである．

加算

加算においては $58 + 39$ の計算を行った．各プロセスまでの合計所要時間を表 6.3 に，各プロセスごとの所要時間を表 6.4 に示す．

プロセス	計測したノード	経過時間の平均 [ms]
シェア作成まで	CL	21
送信プロセス開始まで	CL	23
ノード受信	X	296
ノード間シェアチェック完了	X	775
ノード間シェアチェック完了	Y	734
計算・送信プロセス完了	X	784
計算・送信プロセス完了	Y	775
受信して復元完了（最初のグループ）	CL	1340
受信して復元完了（二つ目のグループ）	CL	1398
グループ A 復元完了	CL	1367

表 6.3: 加算における各プロセスまでの合計所要時間（平均）

プロセス名	計測したノード	所要時間 [ms]
シェア作成	CL	21
送信プロセス開始	CL	3
ノード受信	X	272
シェアチェック	X	480
シェアチェック	Y	439
計算・送信プロセス完了	X	9
計算・送信プロセス完了	Y	41
グループ A 復元完了	CL	584

表 6.4: 加算における各プロセスの所要時間（平均）

乗算

乗算においては 58×39 の計算を行った．各プロセスまでの合計所要時間を表 6.5 に，各プロセスごとの所要時間を表 6.6 に示す． 各プロセスの進行する様子を図 6.7 に示す．

プロセス	該当するノード	経過時間の平均 [ms]
シェア作成まで	CL	24
送信プロセス開始まで	CL	30
ノード受信	X	433
ノード間シェアチェック完了	X	964
ノード間シェアチェック完了	Y	908
計算・送信プロセス完了	X	974
X からの受信完了	Y	1361
Z からの受信完了	Y	2130
計算・送信プロセス完了	Y	2141
受信して復元完了（最初のグループ）	CL	2397
受信して復元完了（二つ目のグループ）	CL	2937
グループ A 復元完了	CL	2660

表 6.5: 乗算における各プロセスまでの合計所要時間（平均）

単一ノードでの計算時間

後の考察にて比較するために，クライアントノード内部の Javascript で同様の計算を行った際の計算時間を表 6.7 に示す．

計測開始地点	計測終了地点	計測したノード	図 6.7 上の該当箇所	所要時間 [ms]
計算開始	シェア作成	CL	a	24
シェア作成	送信プロセス開始	CL	b	6
送信プロセス開始	ノード受信	X	c	402
ノード受信	シェアチェック完了	X	d	531
ノード受信	シェアチェック完了	Y	e	475
シェアチェック完了	計算・送信プロセス完了	X	f	10
X の送信プロセス完了	X からの受信完了	Y	g	388
X の送信プロセス完了	Z からの受信完了	Y	h	1157
Z からの受信完了	計算・送信プロセス完了	Y	i	10
Y の送信プロセス完了	グループ A 復元完了	CL	l	520

表 6.6: 乗算における各プロセスの所要時間（平均）

計算内容	所要時間
加算	<1 ms
乗算	<1 ms

表 6.7: クライアントノード単独で計算を行った際の所要時間

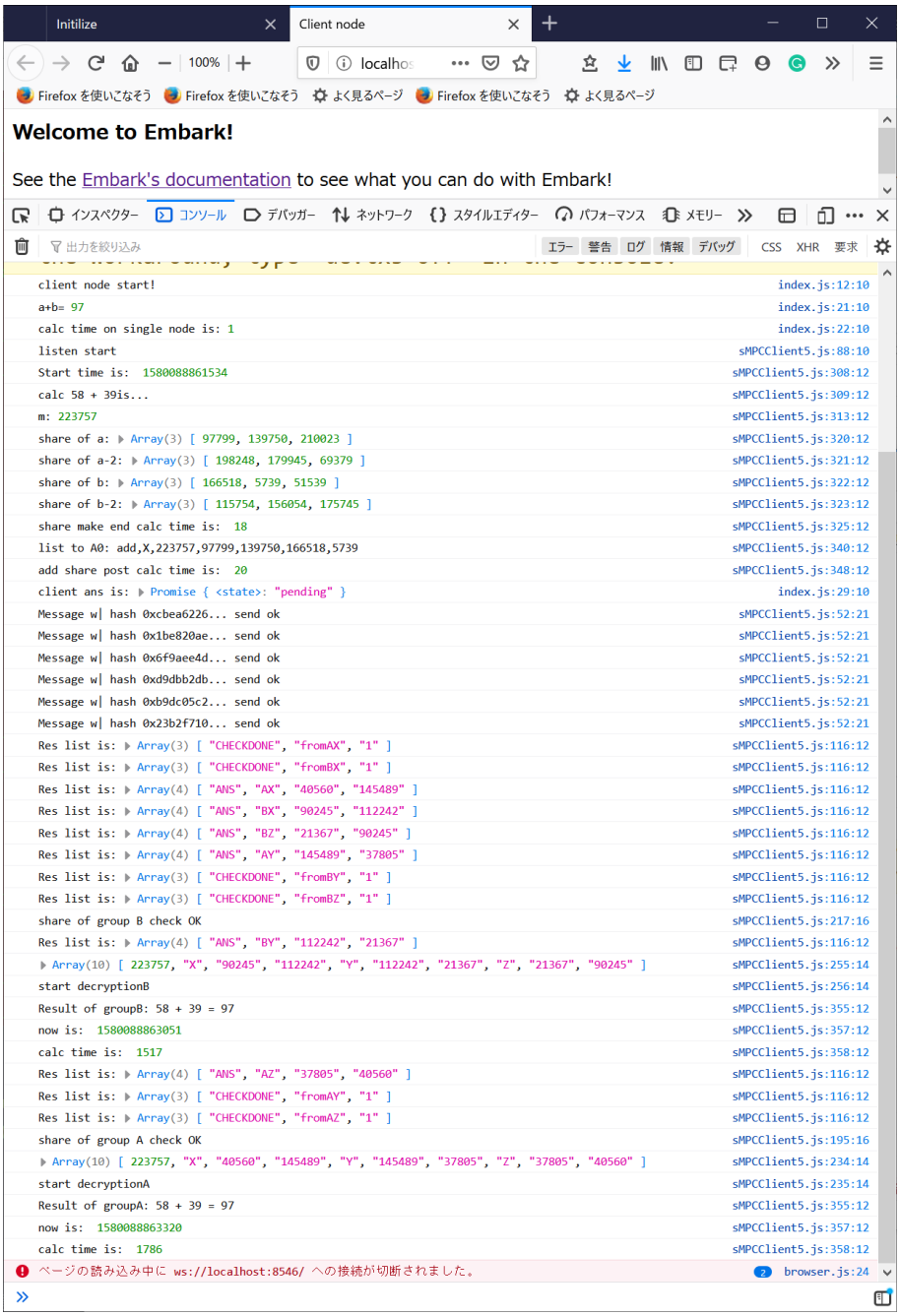


図 6.2: 加算におけるクライアントノードの様子．二つのグループで正しく計算が行われたことが分かる．

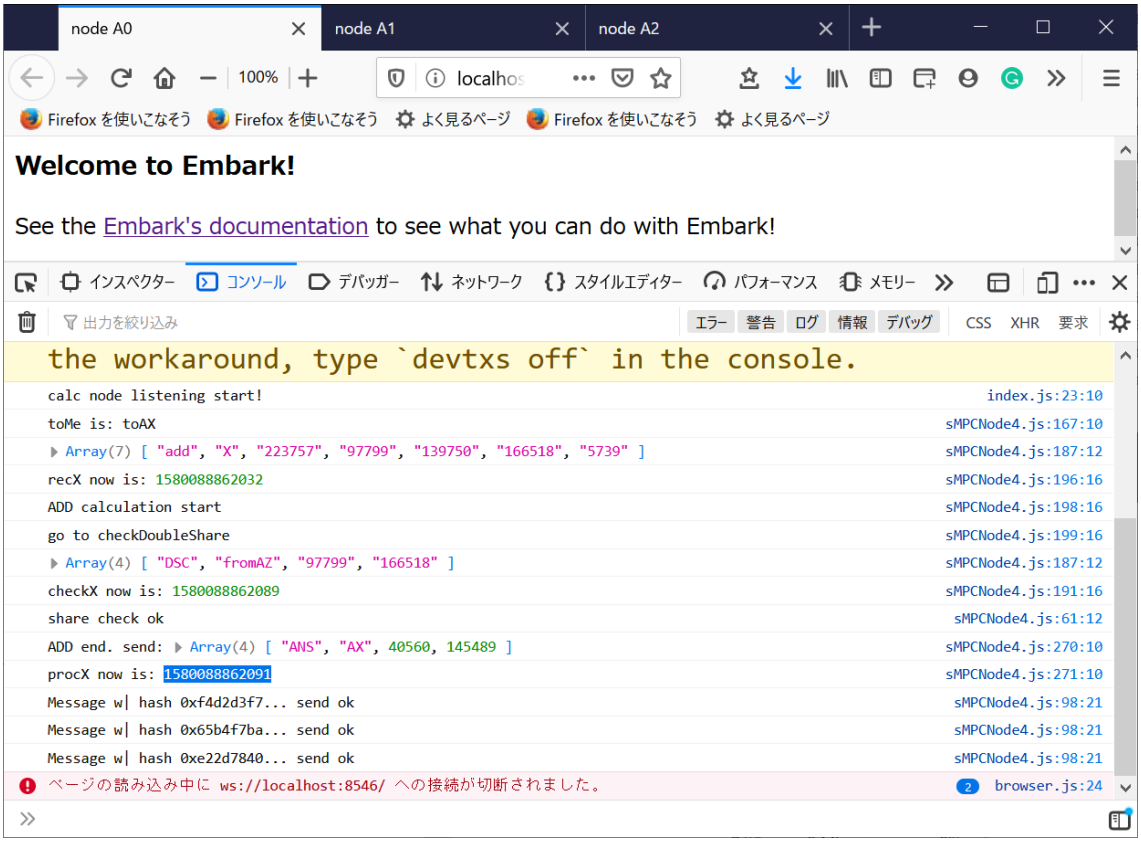


図 6.3: 加算における計算ノードの様子．シェアチェックと計算を行ったことが分かる．

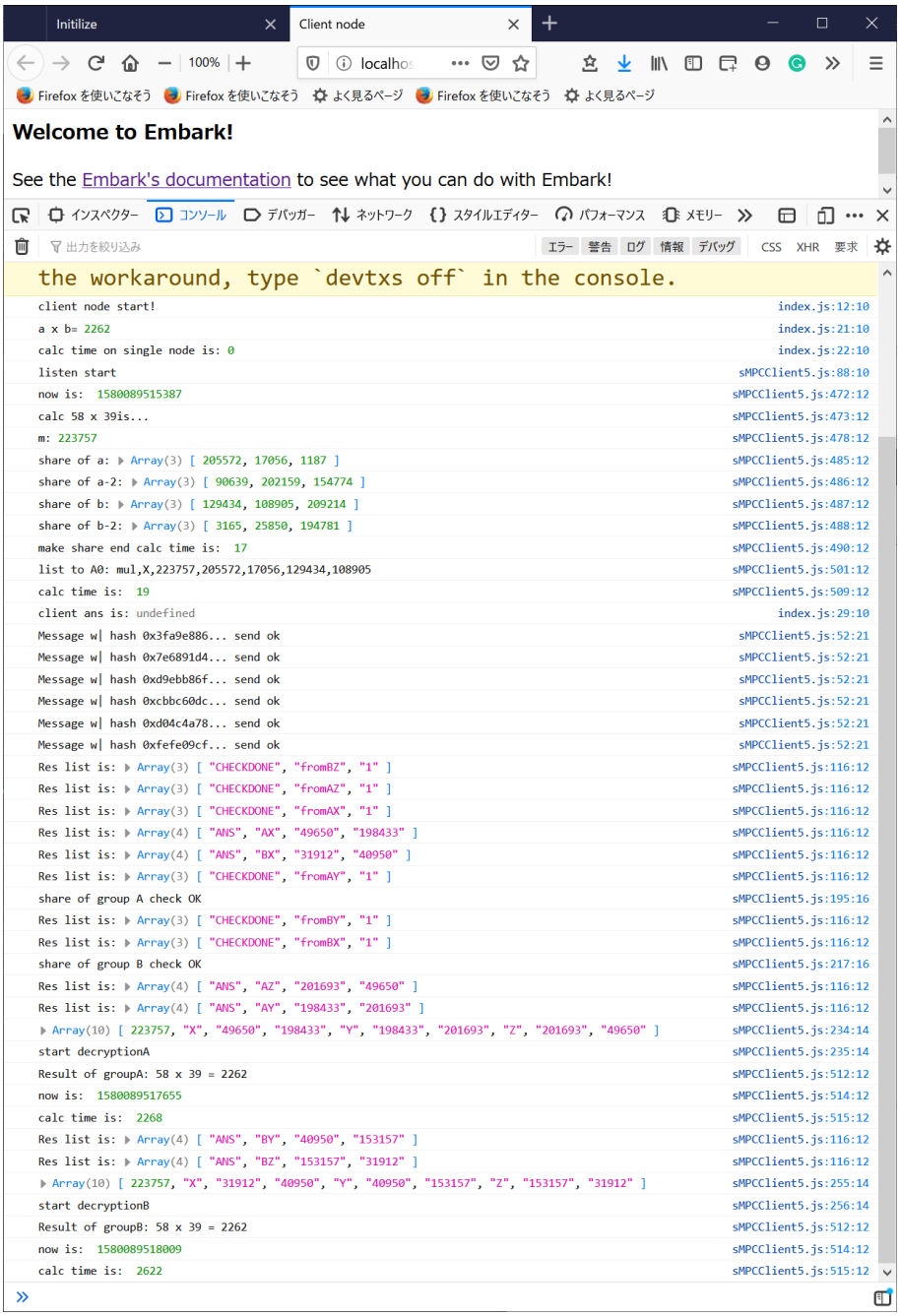


図 6.4: 乗算におけるクライアントノードの様子．二つのグループで正しく計算が行われたことが分かる．

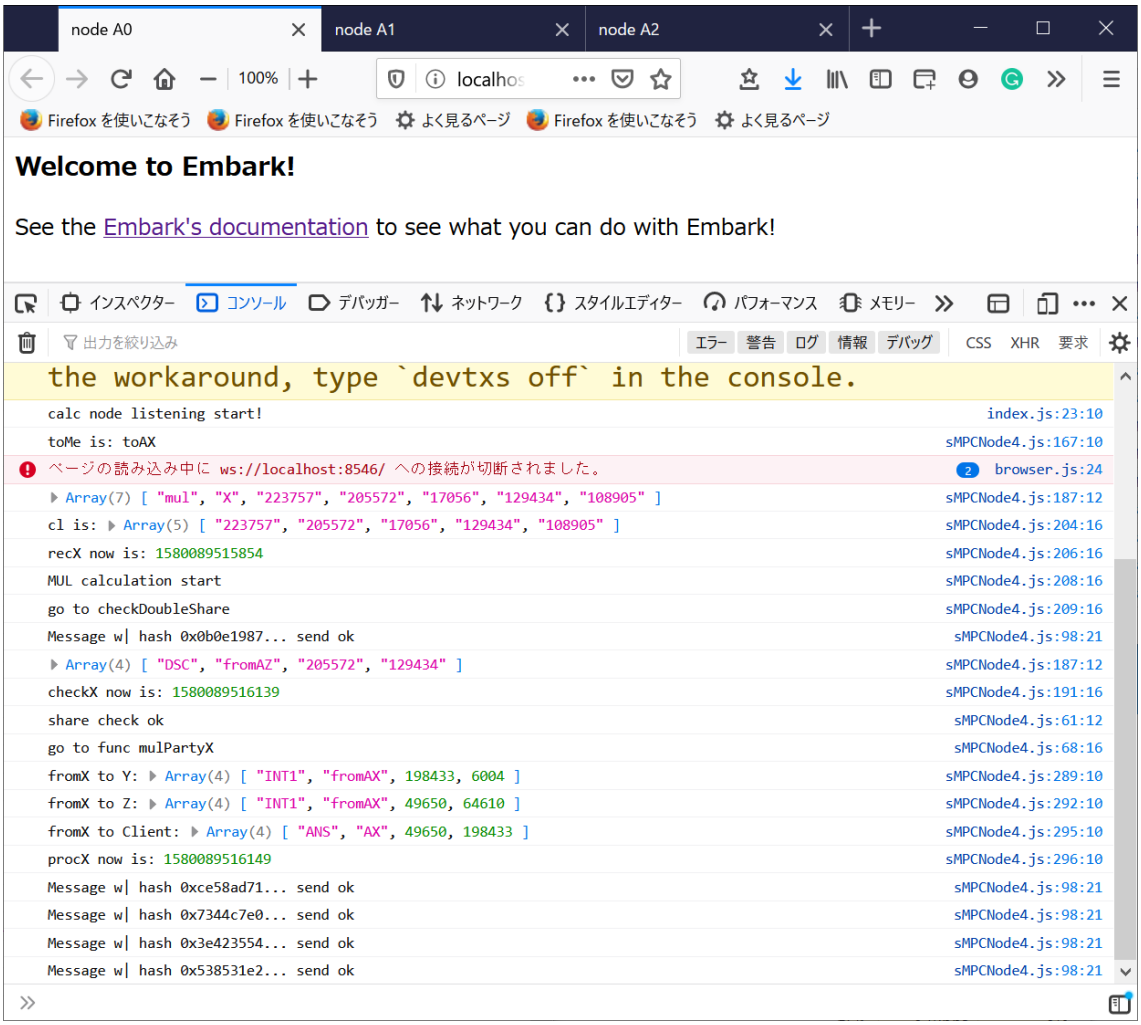


図 6.5: 乗算における計算ノード X の様子．シェアチェックやクライアントノードへの返答だけでなくノード Y や Z に値を送信していることが分かる．

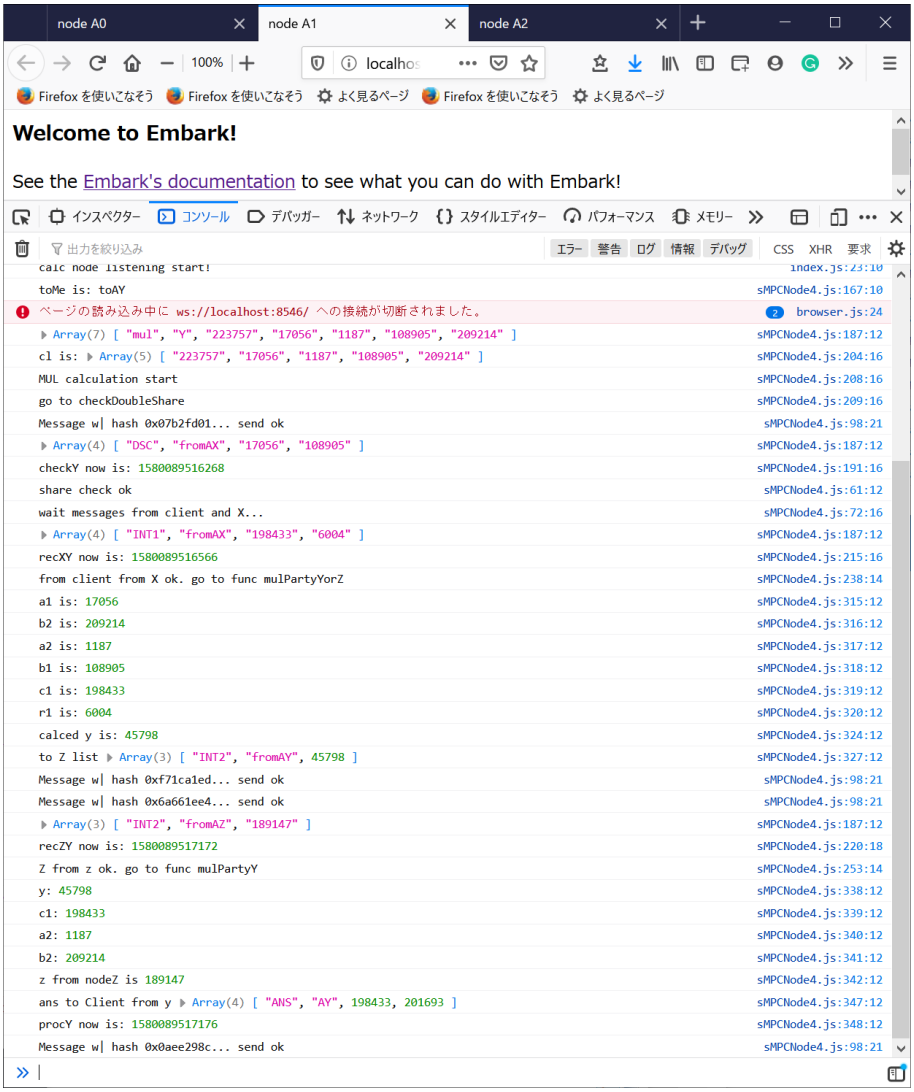


図 6.6: 乗算における計算ノード Y の様子．シェアチェックだけでなく X や Z から受信した値を利用して計算を行ったことが分かる．

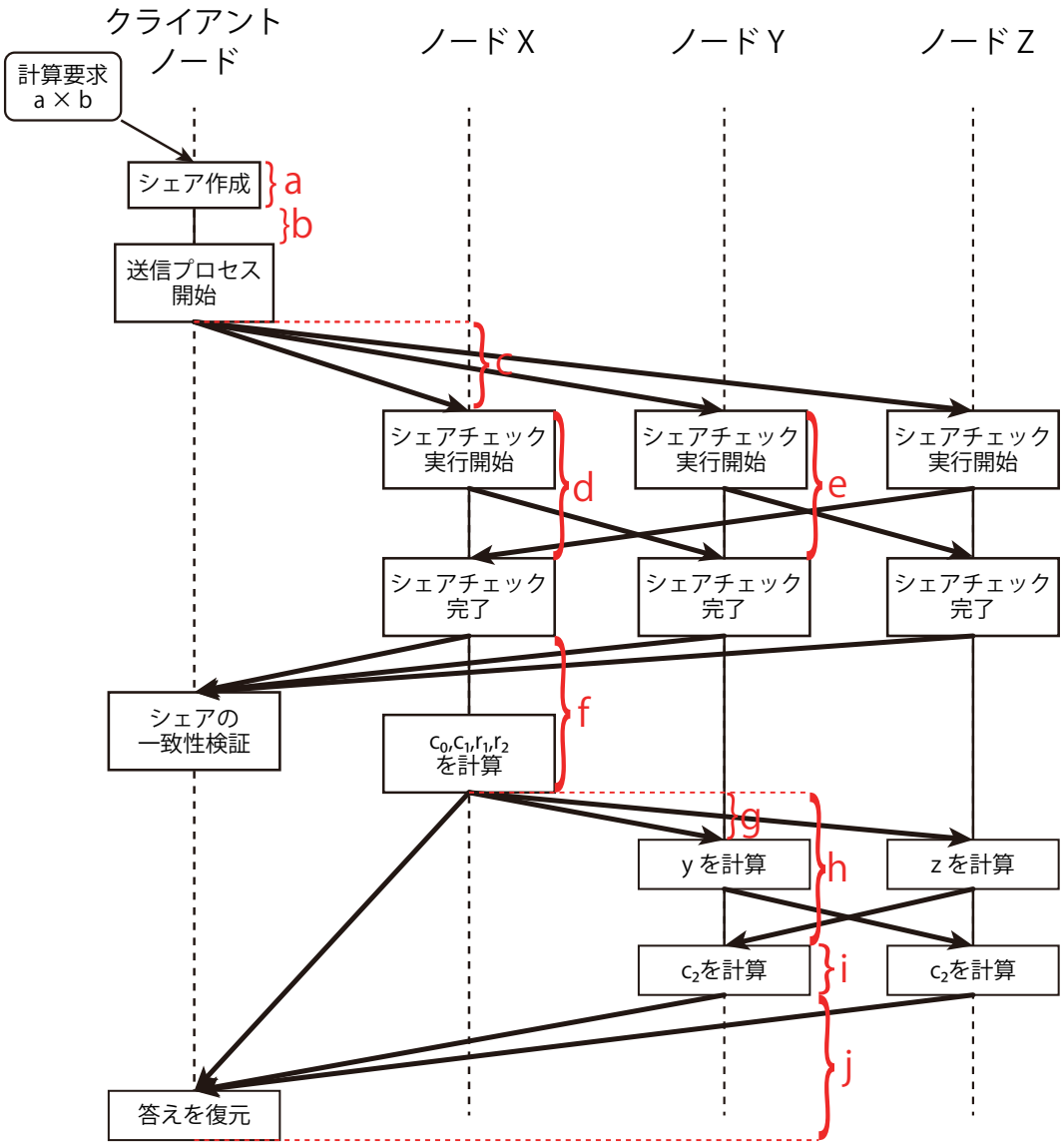


図 6.7: 乗算の計算プロセスが進行する様子．ノード間で通信を行い協力しながら計算を進めている．

第7章　評価・考察

7.1　計算時間の比較

7.1.1　単一ノードで行う場合との比較

セキュアマルチパーティ計算を利用した場合の所要時間は加算においては表 6.3 によると平均 1398 ms であり，乗算は表 6.5 によると平均 2937 ms である．一方クライアントノード単独で計算を行った場合の平均計算時間は表 6.7 によると加算と乗算いずれも 1 ms 未満である．セキュアマルチパーティ計算を通常の計算の代わりとして使うには効率が良くないということが分かる．

7.1.2　単独グループにセキュアマルチパーティ計算を行う場合との比較

今回実装したセキュアマルチパーティ計算においては，複数のグループにセキュアマルチパーティ計算を行わせているため，すべてのグループからの計算が出揃うのが単独グループでの計算に比べて時間がかかることが考えられる．今回の実験において最初に計算が終了したグループと最後に計算が終了したグループでどの程度所要時間のオーバーヘッドがあるのかに関してまとめたものを表 7.1 に示す．

計算内容	最初のグループの所要計算時間 [ms]	二番目のグループの所要計算時間 [ms]	所要時間比
加算	1340	1398	+4.3 %
乗算	2397	2937	+23 %

表 7.1: 複数グループ計算によるオーバーヘッド

これによると計算によってはやや大きめのオーバーヘッドが発生しているように見える．しかし各試行一回一回でこのオーバーヘッドが発生しているわけではなく，二つのグループの計算時間の差が 3 ms しか無かった場合もあれば 1 秒以上遅れて二つ目のグループの計算結果がクライアントノードに到着した場合もあった．また，試行ごとに計算時間の差がありある試行の両方のグループが計算するのにかかった時間が別の試行の片方のグループの計算が終わるより早いということもあった．すなわち，複数グループに計算を任せたから必ず遅くなるというわけではなく，ネットワーク上で早く通信が行われるかどうかの方が大きい．グループが多いと全部がネットワーク上で素早く通信されるとは限らない確率が高くなるので結果としてオーバーヘッドが発生する．

7.1.3　各プロセス間の計算時間

図 6.7 および表 6.6 によると，各ノード内で計算を行うプロセスは数 ms〜数十 ms で終わるのに対し，各ノード間で通信を行うプロセスは数百 ms 単位となっていて．この通信を複数回行うことにより単一ノード内部で計算を行う場合と比べて計算時間が著しく遅くなっている．今回通信に利用した Whisper プロトコルはそもそもがリアルタイムコミュニケーションを前提としたものでないため多少の遅延を前提としている．また，節 6.1.3 で説明した通り今回は P2P 通信ではなくブロードキャスト通信を用いて通信している．この Whisper は P2P では低遅延，ブロードキャストではそれよりも大きい遅延が発生する仕様であるのでこのように通信で遅延が発生した．同じ Whisper プロトコルであってもノード間で P2P 通信を用いて実装すればこれよりも計算時間を短縮することが期待できる．

7.1.4　複合計算のオーバーヘッド

実際のセキュアマルチパーティ計算を利用したい状況において，和や積を一回行うだけで計算が完了するとは考えにくい．そこで，各計算におけるクリティカルパスを考え，複合計算においては独立した単独の計算の連続と比べどの程度計算時間の短縮が望めるかを評価する．ここでは簡単のためクリティカルパス上のノード間の通信の回数で計算にかかる時間を評価する．また，非同期処理を用いてクリティカルパス以外の部分で行う通信はノード間の通信回数に含めない（例：加算において計算ノードが ShareCheck を行ったあとにクライアントノードに返答する際の通信は，計算ノード内の計算＋答えのシェアをクライアントノードに送信することと並行に行われているとみなし ShareCheck 後の通信回数を合わせて一回とする）．

加算におけるクリティカルパスを図 7.1 に，乗算におけるクリティカルパスを図 7.2 に示す．

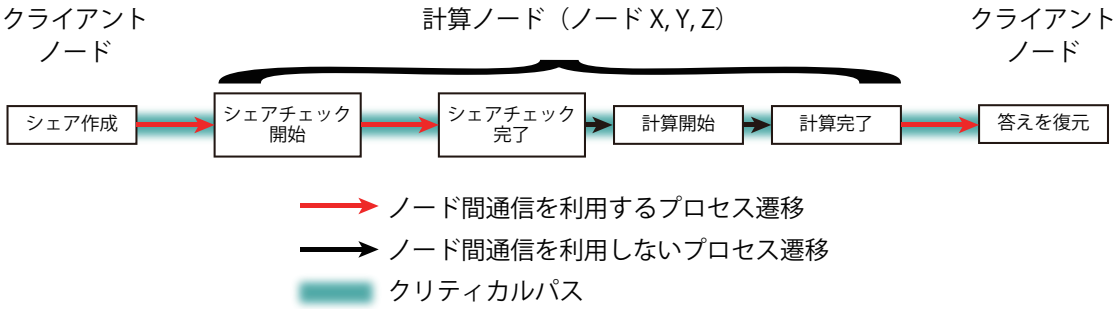


図 7.1: 加算におけるクリティカルパス．ノード間で通信を行う場面がクリティカルパス上で三回あることが分かる

これらの図から分るように加算では 3 回・乗算では 5 回クリティカルパス上でノードの通信を行っている．

次に，複合計算において省略できる通信を考える．複合計算を同一のグループで行う場合，クライアントノードから計算ノードへのデータの受け渡しあるいはその逆は複合計算の最初と最後だけで済む．また，シェアの一致性検証は一回行えばグループ内のノードに lazy ノードや malicious ノードが居ないこ

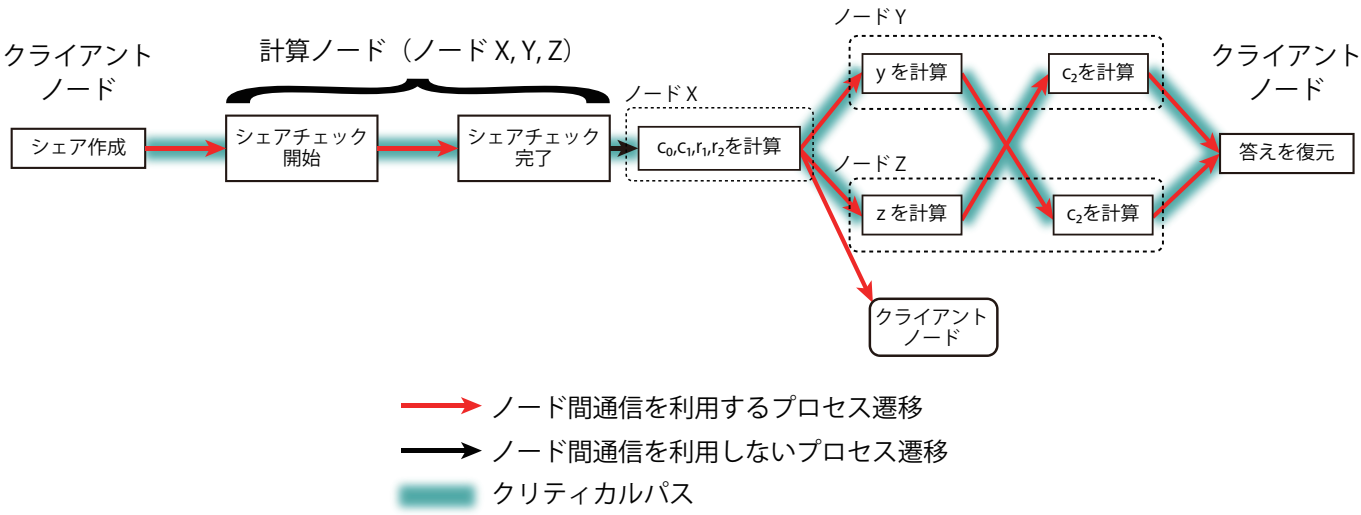


図 7.2: 乗算におけるクリティカルパス．計算プロセスが並列に進む場面があるがいずれにせよ計算の完了までに並列化できないノード間通信を 5 回行う必要がある．

とが確認できる．これは，一度で分からないよう擬態する相手の場合は複数回行っても擬態される可能性が十分考えられるためである．

整理すると，複合計算上で省略不可能な通信は加算では 0 回・乗算では 2 回あることが分かる．すなわち，複合計算において加算を m 回，乗算を n 回行う際に必要な通信回数 t は

$$t = 3 + 2n \tag{7.1}$$

となる．

7.2 現実的な実装に向けて

本実験においては，節 5.2 に示した提案手法のプロトコルの一部分しか実装していない．実用的な実装に向けて今回の実装以外何が必要かを整理すると以下のような機能が挙げられる．

- 計算要求を受け取ったクライアントノードがブロードキャストで協力してくれる計算ノードを見つける．
- アクティブな計算ノードは自身の計算の受け入れやすさに応じた待機時間後にクライアントノードに返答する．
- 複数の計算ノードから返答を受け取ったクライアントノードはブロックチェーン上に登録されている計算ノードごとの成績を参照して採用する計算ノードを選ぶ．

- クライアントノードは選んだ計算ノード複数をまとめてグループを作る．
- 複数グループの計算結果を受け取ったクライアントノードはそれぞれのグループの結果が正しく行われているか検証する．
- クライアントノードは検証結果に合わせて各計算ノードの成績をブロックチェーン上に登録する．
- クライアントノードは紐づけれているイーサリアムアドレスから計算料を各計算ノードに対応するアドレスへ支払う．

7.3 応用例

ここでは，本研究において提案したシステムがどのように応用されうるかについて考察を行う．

7.3.1 プライバシーデータ計算への応用

セキュアマルチパーティ計算の応用先として注目されているのはプライバシーデータを利用した計算である．データを秘匿したまま計算を行う計算として準同型暗号も存在するが，準同型暗号暗号は計算量や計算回数の制限という課題がある．計算回数に制限がない完全準同型暗号に比べるとセキュアマルチパーティ計算は複数回の乗算を行ってもデータ容量や必要な計算量が小さく抑えられる．

セキュアマルチパーティ計算の計算量以外の利点として，計算の結果が欲しい人に計算元のデータが分からないまま計算が可能であるということがある．これは複数の第三者からデータを提供してもらいそれらを組み合わせて計算を行うシチュエーションが考えられる．このように構築するシステムを図 7.3 に示す．このように元のデータ所有者が外部にデータそのものを公開することなくデータを提供してもらい，演算の結果のみ利用するということができる．

また，プライバシーデータを計算に利用する場合は和と積だけでなく他に様々な計算を行うことが必要となることが考えられる．例として，プライバシーデータを読み出す際のテーブル引きなどがある．これに関しては，セキュアマルチパーティ計算のルックアップテーブルへの応用を Launchbury ら [53] が提唱している．

また，本研究において参考にしたセキュアマルチパーティ計算は論理回路演算が可能である．$a, b \in 0, 1$ のシェアの組み合わせ $[a], [b]$ について否定 NOT は

NOT：$a \in 0, 1$ のシェアから a の否定 $\bar{a} := 1 - a$ のシェアを生成

入力：$P_i([a]_i)$
出力：$P_i([\bar{a}]_i)$

1. P_0 は $[\bar{a}]_0 := (1 - a_0, -a_1)$ を計算する．
2. P_1 は $[\bar{a}]_1 := (-a_1, -a_2)$ を計算する．
3. P_2 は $[\bar{a}]_2 := (-a_2, 1 - a_0)$ を計算する．

のようになる．他の論理回路演算については加減算 Add/Sub・定数倍 CoMul・乗算 Mul を用いると以下

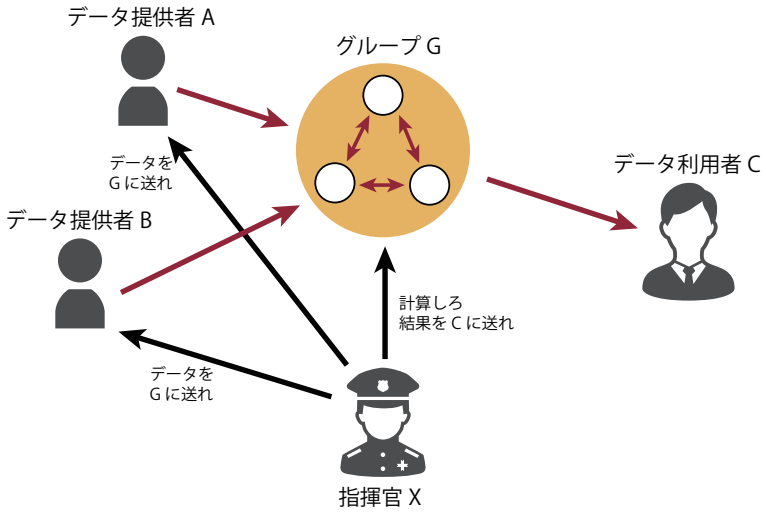


図 7.3: 元データを秘匿した状態でセキュアマルチパーティ計算を行う様子．ここにおける指揮官（各ノードやデータ提供者に指示を出す人）とデータ利用者は同一人物でも成立する．

のようになる．

- 論理積：AND$([a],[b])$:=Mul$([a],[b])$
- 論理和：OR$([a],[b])$:=Sub(Add$([a],[b])$,Mul$([a],[b])$)
- 排他的論理和：XOR$([a],[b])$:=Sub(Add$([a],[b])$,CoMul(Mul$([a],[b])$, 2))

さらに，これらを利用すれば否定論理積 NAND が作成できる：

- 否定論理積：NAND$([a],[b])$:=NOT(AND$([a],[b])$)

となる．NAND ゲートが作れることから，このセキュアマルチパーティ計算はチューリング完全である．

7.3.2 実際の利用方法への考察

実際に提案したシステムを利用してもらうためにどのようなインセンティブがあるかを考察する．ここではプライバシーデータをやりとりすると考えてデータ提供者へのインセンティブ・計算ノードとして立候補してくれるノードを提供する計算ノード提供者へのインセンティブの二種類について議論する．

前提として，節 7.3.1 セキュアマルチパーティ計算では計算結果を利用したい人や計算を行うノードに元データそのままを渡す必要がない．

データ提供者となるためのインセンティブ

データ提供者とは，図 7.3 におけるパーティにデータを提供する側の人である．現実世界においてはそれがプライバシーデータを保有する企業のサーバであったり個人のスマートフォンであることが考えら

れる．セキュアマルチパーティ計算で安全に計算ができることが保障されていたとしてもプライバシーデータの保有者がそのままデータの提供に応じるとは限らない．そこで，各データ保有者は自身の持つデータの利用料を設定できるようにする．そのようにすればデータ保有者は安全が保障されつつデータを提供し，その代わりにデータ利用料を徴収できるようになる．このデータ利用料は計算結果を受け取る利用者の計算料金に上乗せすればよい（図 7.4 ）．

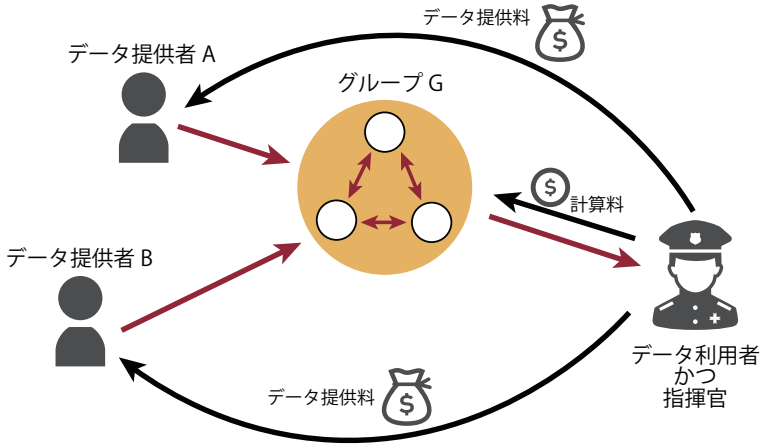


図 7.4: 指揮官兼データ利用者がデータ提供者にデータ提供料を支払う様子．これと計算ノードへの計算料の支払いは実際はブロックチェーンネットワーク上で行われる．

計算ノード提供者となるためのインセンティブ

次に計算ノードを提供してくれるためにどのような仕組みが可能かを考える．いくらデータ提供者とデータを活用したい人が存在していても間でセキュアマルチパーティ計算を行うノードが居ないと実際の計算は行えない．また，計算ノードが居たとしても，不正を働こうとするノードばかりでは計算システムが機能しなくなってしまう．すなわち，どのように多数の不正を行わない市井の人々に計算ノードを提供してもらうかが肝要である．

計算ノードは計算を行いその対価として報酬を得る．そして計算はいつ要求されるかわからないのでできる限り常時イーサリアムネットワークにつながっていることが望ましい．すなわち，イーサリアムネットワーク上に存在する数多のマイニングノードが計算ノードして機能することが理想である．マイニングノードを稼働させている人は自身の計算能力をイーサリアムに提供しその報酬としてイーサを受け取る．そのようにマイナーはもともと報酬目当てで計算能力を提供している人であるから，計算ノードとして計算能力を提供してもらいその報酬として対価を払うようにすれば計算ノードを提供してもらうことが十分考えられる．特に本計算システムにおいてはイーサリアムネットワーク上で動作しイーサをやりとりするため，イーサをマイニングしているマイナーの要求に合致する．

また，本研究で用いたセキュアマルチパーティ計算における各ノードで行う計算は軽量であるから，イーサリアムの主要なマイニング機械である GPU ではなく CPU を利用すればマイニングの邪魔をすることなく計算ノードとしての稼働が可能である．

さらに，本計算システムの実用的な実装を考える．本計算システムでは計算ノードは Web ブラウザ上で計算を行う．現状多くのマイナーがマイニングプールに参加してマイニングを行っている．そしてマイニングプールの多くが分析画面としてのダッシュボードを Web ブラウザ上で提供している．すなわちマイニングプールのダッシュボード上に本計算システムを実装すればマイニングプールに参加しているマイナーに計算ノードを提供してもらうことが可能となる．

第8章　結論

本研究ではイーサリアムのシステム上で動作する秘匿性を保った分散計算を提案した．今回はその中でも分散計算を行い答えが導き出す部分を実装した．分散計算はセキュアマルチパーティ計算を採用し，実装にはイーサリアム Javascript API である Web3 の Whisper プロトコルを利用してフロントエンドアプリケーション上で分散計算が行えることを示した．ただ，提案で示したブロックチェーンを利用しての計算ノードの選択・成績の参照・報酬の支払いは実装に至らなかった．

実際に通信に Whisper プロトコルを用いてセキュアマルチパーティ計算をした結果，実行速度にはノード間の通信が大きく関わってくることが分かった．これは構築した環境の特性上 Whisper プロトコルの中でも遅延の大きいブロードキャストを用いているからであると考えられる．今後の課題として複数のコンピュータを接続したブロックチェーンネットワークで同様の実験が行えるか・Whisper の P2P 通信を用いればどの程度早くなるのか・Whisper 以外の P2P 通信プロトコルが利用できないかなどの解明が挙げられる．

本研究では実際にネットワーク上でシステムが稼働するまでに至らなかったが，もし提案したシステムが実現すれば大規模な秘匿性を持つ分散コンピューティングネットワークとして大いに役立つであろう．

発表

- 2019 年並列／分散／協調処理に関する『北見』サマー・ワークショップ (SWoPP2019)
  2019 年 7 月 24 日〜2019 年 7 月 26 日
  北見市民会館
  『ブロックチェーンを用いたセキュアな分散コンピューティング』岩下義明・入江英嗣・坂井修一

謝辞

本研究を進めていくうえで，指導教員の坂井修一教授と入江英嗣准教授からは多大なお力添えを賜りました．ずっと環境構築がうまくいかず悩んでいたりシステムのアイデアを提案したときに適切な指導をしてくださったことにより本研究をこのようにまとめ上げることができました．また新規のアイデアを考え付いたときに入江先生とともに耳を傾けて不備がないかなどの考察をしてくださった小泉透さんにも感謝申し上げます．ブロックチェーンという共通の研究テーマを持ち議論を重ねて理解を深め合った浅野泰輝君，林リウヤ君にも感謝申し上げます．

関連図書

[1] https://web.archive.org/web/20130526224224/http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf(参照 2020-01-11).

[2] http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html(参照 2020-01-08).

[3] http://telehash.org/(参照 2020-01-09).

[4] block-chain.jp - proof of stake とは何か？ https://block-chain.jp/blockchain/proof-of-stake/(参照 2020-01-20).

[5] Eniglma protocol. https://enigma.co/discovery-documentation/(参照 2020-01-15).

[6] Ethereum. https://ethereum.org/ja/(参照 2020-01-15).

[7] Ethereum classic. https://ethereumclassic.org/(参照 2020-01-13).

[8] Ghash.io. http://ghash.io/(参照 2020-01-07).

[9] Github - ethereum/web3.js: Ethereum javascript api. https://github.com/ethereum/web3.js/(参照 2020-01-14).

[10] Github - ethereum/wiki whisper. https://github.com/ethereum/wiki/wiki/Whisper(参照 2020-01-09).

[11] Github - telehash/telehash.github.io/v3/spec v3.0.0-stable. https://github.com/telehash/telehash.github.io/blob/master/v3/spec/v3.0.0-stable.pdf(参照 2020-01-09).

[12] Github - the general theory of decentralized applications, dapps. https://github.com/DavidJohnstonCEO/DecentralizedApplications(参照 2020-01-13).

[13] Github - whisper v6 rpc api - ethereum/go-ethereum. https://github.com/ethereum/go-ethereum/wiki/Whisper-v6-RPC-API(参照 2020-01-).

[14] Neo smart economy. https://neo.org/(参照 2020-01-20).

[15] .. A Day in the Life of. Telehash: an encrypted p2p network for your apps. https://adayinthelifeof.nl/2013/11/12/telehash-an-encrypted-p2p-network-for-your-apps/(参照 2020-01-09).

[16] Gavin Wood Andreas M. Antonopoulos. 『マスタリング・イーサリアム』. オライリー・ジャパン, 2019 年 11 月 28 日. 宇野 雅晴，鳩貝 淳一郎 監訳 中城 元臣，落合 渉悟 技術監修 落合庸介，小林 泰男，土屋 春樹，祢津 誠晃，平山 翔，三津澤 サルバドール将司，山口 和輝 訳.

[17] BitAngels. https://www.bitangels.network/(参照 2020-01-13).

[18] Bitcoin. Bitcoin core のダウンロード. https://bitcoin.org/ja/download(参照 2020-01-13).

[19] bitFlyer. イーサリアム（ethereum）とは？https://bitflyer.com/ja-jp/ethereum(参照 2020-01-06).

[20] bitFlyer. ビットコイン（bitcoin）用語集 segwit. https://bitflyer.com/ja-jp/glossary/segwit(参照 2020-01-13).

[21] bitFlyer. ビットコイン（bitcoin）用語集 ソフトフォーク. https://bitflyer.com/ja-jp/glossary/soft_fork(参照 2020-01-13).

[22] bitFlyer. ビットコイン（bitcoin）用語集 ハードフォーク. https://bitflyer.com/ja-jp/glossary/hard_fork(参照 2020-01-13).

[23] Blockchain Biz. モナコインへのセルフィッシュマイニング攻撃. https://gaiax-blockchain.com/monacoin-selfish-mining(参照 2020-01-07).

[24] G. R. Blakley. Safeguarding cryptographic keys. *1979 International Workshop on Managing Requirements Knowledge, MARK 1979*, pp. 313–317, 1979.

[25] Max Boddy. Cointelegraph - ブロックチェーンプラットフォーム「ネオ」，金融に最適とうたうコンセンサスアルゴリズム最新版「dbft 2.0」を実装. https://jp.cointelegraph.com/news/neo-announces-new-consensus-mechanism-for-its-new-mainnet(参照 2020-01-20).

[26] Dan Boneh, Eu Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. *Lecture Notes in Computer Science*, Vol. 3378, pp. 325–341, 2005.

[27] Vitalik Buterin. A next-generation smart contract and decentralized application platform. *Etherum*, No. January, pp. 1–36, 2014.

[28] Bitcoin Cash. Peer-to-peer electronic cash. https://www.bitcoincash.org/(参照 2020-01-13).

[29] Koji Chida, Dai Ikarashi, Koki Hamada, and Katsumi Takahashi. A Lightweight Three-party Secure Function Evaluation with Error Detection and Its Experimental Result. Vol. 52, No. 9, pp. 2674–2685, 2011.

[30] CoinMarketCap. 仮想通貨時価総額上位 100. https://coinmarketcap.com/ja/(参照 2020-01-19).

[31] CoinPost. ソフトフォークとハードフォーク問題/両者の意味と違いを解説. https://coinpost.jp/?p=3143(参照 2020-01-13).

[32] Victor Costan and Srinivas Devadas. Intel SGX Explained. *Cryptology ePrint Archive, Report 2016/086*, p. 108, 2016.

[33] CryptoKitties. Collect and breed furrever friends! https://www.cryptokitties.co/(参照 2020-01-24).

[34] MDN Web docs. Webrtc api. https://developer.mozilla.org/ja/docs/Web/API/WebRTC_API(参照 2020-01-09).

[35] Swarm 0.5 Documentation. Introduction. https://swarm-guide.readthedocs.io/en/latest/(参照 2020-01-14).

[36] Embark. Embark into the ether. https://embark.status.im/(参照 2020-01-22).

[37] Enigma. Announcing the launch of enigma ' s first networked testnet! https://blog.enigma.co/announcing-the-launch-of-enigmas-first-networked-testnet-359fd816cb69(参照 2020-01-15).

[38] Enigma. Defining secret contracts. https://blog.enigma.co/defining-secret-contracts-f40ddee67ef2(参照 2020-01-15).

[39] Enigma. Enigma ' s ambition — our latest roadmap. https://blog.enigma.co/enigmas-ambition-our-latest-roadmap-8d50107ad314(参照 2020-01-15).

[40] Enigma. Secret contracts: Now new and improved! https://blog.enigma.co/secret-contracts-now-new-and-improved-df742393d6d8(参照 2020-01-15).

[41] Enigma. Securing the decentralized web. https://enigma.co/(参照 2020-01-).

[42] Ittay Eyal and Emin Gün Sirer. Majority Is Not Enough: Bitcoin mining is vulnerable. *Communications of the ACM*, Vol. 61, No. 7, pp. 95–102, 2018.

[43] Fortune. Walmart and ibm are partnering to put chinese pork on a blockchain. https://fortune.com/2016/10/19/walmart-ibm-blockchain-china-pork/(参照 2020-01-27).

[44] Craig Gentry. a Fully Homomorphic Encryption Scheme. *PhD Thesis*, No. September, pp. 1–209, 2009.

[45] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. *Networks*, 1987.

[46] HEDGE GUIDE. イーサリアムは大型アップデート「セレニティ」で1,000倍の処理速度へ. https://hedge.guide/news/ethereum-joseph-lubin-serenity-1000-201905.html(参照 2020-01-13).

[47] NEO Developer Guide. The dbft algorithm. https://docs.neo.org/developerguide/en/articles/consensus/consensus_algorithm.html(参照 2020-01-20).

[48] Individua1. 外部所有アカウントとコントラクトアカウント、そしてマークルツリー. https://individua1.net/ethereum-guide/externally-owned-accounts-contract-accounts/(参照 2020-01-13).

[49] Sony Japan. ブロックチェーン基盤を活用したデジタルコンテンツの権利情報処理システムを開発. https://www.sony.co.jp/SonyInfo/News/Press/201810/18-1015/(参照 2020-01-27).

[50] EnigmaJapan のブログ. Enigma の野望 - 最新のロードマップ. https://www.enigmajapan.blog/entry/2018/03/26/212205(参照 2020-01-15).

[51] Shota Johjima, Kosuke Kaneko, Yuki Nishida, Kazuya Nakayama, Yusuke Tsutsumi, and Kouichi Sakurai. Implementation for Neural Network Executed in Secure Distributed Processing using Blockchain. pp. 4–8, 2018.

[52] Stir Lab. Enigma プロジェクトの概要とプライバシー保護に関する社会的背景. https://lab.stir.network/2018/12/20/about-enigma-and-gdpr/(参照 2020-01-15).

[53] John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. *ACM SIGPLAN Notices*, Vol. 47, No. 9, pp. 189–200, 2012.

[54] Meteor. Build apps with javascript. https://www.meteor.com/(参照 2020-01-22).

[55] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Www.Bitcoin.Org*, p. 9, 2008.

[56] Kyber Network. The on-chain liquidity protocol for the tokenized world. https://kyber.network/(参照 2020-01-24).

[57] ITmedia NEWS. モナコインのブロックチェーン、攻撃受け「巻き戻し」 国内取引所も警戒. https://www.itmedia.co.jp/news/articles/1805/18/news071.html(参照 2020-01-07).

[58] University of Cambridge. Cambridge bitcoin electricity consumption index. https://www.cbeci.org/comparisons/(参照 2020-01-20).

[59] COIN OTAKU. イーサリアム (ethereum) のアップデートフロンティアの解説. https://coin-otaku.com/topic/2291(参照 2020-01-13).

[60] COIN OTAKU. イーサリアム (ethereum) のアップデートホームステッドの解説. https://coin-otaku.com/topic/2315(参照 2020-01-13).

[61] Coin Otaku. ビットコインのブロックサイズとは？問題点や解決策をわかりやすく解説. https://coinotaku.com/posts/14653(参照 2020-01-13).

[62] Monacoin Project. Monacoin. https://monacoin.org/(参照 2020-01-07).

[63] Enigma Protocol. Enigma protocol 0.1 documentation. https://enigma.co/protocol/index.html(参照 2020-01-15).

[64] Enigma Protocol. Enigma protocol 0.1 documentation - software architecture. https://enigma.co/protocol/SoftwareArchitecture.html(参照 2020-01-20).

[65] Enigma Protocol. Enigma protocol 0.1 documentation - system design. https://enigma.co/protocol/SystemDesign.html(参照 2020-01-20).

[66] React. ユーザインターフェース構築のための javascript ライブラリ. https://ja.reactjs.org/?no-cache=1(参照 2020-01-22).

[67] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, Vol. 4, No. 11, pp. 120–126, 1978.

[68] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *National Science Foundation*, Vol. 7, No. 26, pp. 611–613, 1977.

[69] Jeff S. Neo ' s dbft 2.0 — single block finality with improved availability. https://medium.com/neo-smart-economy/neos-dbft-2-0-single-block-finality-with-improved-availability-6a4aca7bd1c4(参照 2020-01-20).

[70] Blockchain Luxembourg S.A. ハッシュレート分布大きなのマイニングプールのハッシュレート分布の推定. https://www.blockchain.com/pools?(参照 2020-01-08).

[71] Blockchain Luxembourg S.A. ブロックチェーン - 最も信頼されている仮想通貨企業. https://www.blockchain.com/(参照 2020-01-08).

[72] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal selfish mining strategies in bitcoin. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9603 LNCS, pp. 515–532, 2017.

[73] Masahiro Sasaki. dbft -民主的なコンセンサスアルゴリズム-. https://medium.com/@masahiro.sasaki/dbft-%E6%B0%91%E4%B8%BB%E7%9A%84%E3%81%AA%E3%82%B3%E3%83%B3%E3%82%BB%E3%83%B3%E3%82%B5%E3%82%B9%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0-5c36b80cac2b(参照 2020-01-20).

[74] Hiroki Sato. 完全準同型暗号のデータマイニングへの利用に関する研究動向 A Survey on the use of Fully Homomorphic Encryption Scheme in Data Mining. pp. 165–172, 2016.

[75] Adi Shamir. How to Share a Secret. *Communications of the ACM*, Vol. 22, No. 11, pp. 612–613, 1979.

[76] steemit. Neo's consensus protocol: How delegated byzantine fault tolerance works. https://steemit.com/neo/@basiccrypto/

neo-s-consensus-protocol-how-delegated-byzantine-fault-tolerance-works(参照 2020-01-20).

[77] Truffle Suite. Ganache. https://www.trufflesuite.com/ganache(参照 2020-01-22).

[78] Crypto Times. イチからわかるマイニング事情【第 4 回】: セルフィッシュマイニング. https://crypto-times.jp/mining-selfishmining-4/(参照 2020-01-07).

[79] Greg Walker. learn me a bitcoin - target. https://learnmeabitcoin.com/guide/target(参照 2020-01-12).

[80] web3.js 1.0.0 documentation. web3.js - ethereum javascript api. https://web3js.readthedocs.io/en/v1.2.4/(参照 2020-01-14).

[81] WebRTC. https://webrtc.org/(参照 2020-01-09).

[82] WebRTC. Architecture. https://webrtc.org/architecture/(参照 2020-01-09).

[83] Gavin Wood. Ethereum: a secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, pp. 1–32, 2014.

[84] Andrew C. Yao. Protocols for Secure Computations. *Annual Symposium on Foundations of Computer Science - Proceedings*, pp. 160–164, 1982.

[85] Andrew Chi-Chih Yao. How to generate and exchange secrets. *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, No. 1, pp. 162–167, 1986.

[86] ZOOM. ブロックチェーンの安全性を脅かす「51 %攻撃」の仕組みと実態. https://zoom-blc.com/51-percent-attack(参照 2020-01-07).

[87] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized Computation Platform with Guaranteed Privacy. pp. 1–14, 2015.

[88] あたらしい経済. Web3 時代のブラウザとは. https://www.neweconomy.jp/features/web3a/30980(参照 2020-01-13).

[89] 仮想通貨部かそ部. The dao 事件とは？ イーサリアムを支える重要技術について徹底解説. https://kasobu.com/the-dao-ethrerum/(参照 2020-01-13).

[90] コインペディア. 51 %攻撃を受けるとどうなる？ 仕組み・実例・攻撃対策. https://coinpedia.cc/fiftyonepercentattack(参照 2020-01-07).

[91] コインペディア. イーサリアム大型アップデート「メトロポリス」が完了！実装内容と今後. https://coinpedia.cc/eth-metropolis(参照 2020-01-13).

[92] ハードウェアウォレット. コールドウォレットとホットウォレットの違い. https://hardwarewallet.jp/hardwarewallet/difference.html(参照 2020-01-11).

[93] GMO インターネット次世代システム研究室. Ethereum のフルノードを立てる 2019 年秋. https://recruit.gmo.jp/engineer/jisedai/blog/ethereum-fullnode-autumn-2019/(参 照 2020-01-26).

[94] 里丸. 【ビットコインウォレット】の仕組み「コールドウォレット」「ホットウォレット」とは. https://salestechnologylab.com/%E3%83%93%E3%83%83%E3%83%88%E3%82%B3%E3%82%A4%E3%83%B3%E3%82%A6%E3%82%A9%E3%83%AC%E3%83%83%E3%83%88-%E3%81%AE%E4%BB%95%E7%B5%84%E3%81%BF-%E3%82%B3%E3%83%BC%E3%83%AB%E3%83%89%E3%82%A6%E3%82%A9%E3%83%AC%E3%83%83%E3%83%88-%E3%83%9B%E3%83%83%E3%83%88%E3%82%A6%E3%82%A9%E3%83%AC%E3%83%83%E3%83%88-%E3%81%A8%E3%81%AF-7cf0e1926134(参照 2020-01-11).