修士論文

ビットコインにおけるタイムロックを使用した トランザクション置換に関する研究

A Study on Transaction Replacement Using Timelocks in Bitcoin

指導教員　　松浦幹太　教授

東京大学大学院 情報理工学系研究科 電子情報学専攻

48-186430　　長嶺 隆寛

令和 2 年 1 月 30 日提出

内容梗概

ビットコインは単位時間当たりに処理できるトランザクション数が少ないというスケーラビリティ問題を抱えている．スケーラビリティ問題改善に向けて，トランザクションの処理をブロックチェーンの外で行うオフチェーンのアプローチが数多く提案されている．オフチェーンのアプローチでは，複数のトランザクションをブロックチェーンの外で管理し，これらの処理を集約したトランザクションのみをブロックチェーンに記載する．これにより，ブロックチェーンに記載されるトランザクション数が減少し，スケーラビリティ問題改善に繋がる．本稿では特に，オフチェーンのアプローチの根幹要素である，2 者がブロックチェーンの外で資金交換を行うペイメントチャネルに着目する．ペイメントチャネルで管理されるトランザクションはブロックチェーンプロトコルの範囲外なので，チャネル内で安全なトランザクションの管理を行わなければならない．具体的には，過去の資金残高が反映された過去のトランザクションを，最新の資金残高が反映された最新のトランザクションで置換する必要がある．

先行研究にて，タイムロックを使用してトランザクション置換を行うペイメントチャネルが提案されている．これらの提案はタイムロックを使用した時間制御により，最新のトランザクションのみが有効になる仕組みを構築しているが，時間パラメータの許容誤差やトランザクション手数料の変動を考慮していない．時間パラメータはタイムロックの比較やノードの内部時間に使用され，手数料はトランザクションがブロックに追加される優先度 (時間) を決定付ける．そのため，時間パラメータの許容誤差や手数料変動はタイムロックを使用したトランザクションの管理に影響を及ぼす可能性がある．トランザクション置換が失敗すると，過去の資金残高が反映された過去のトランザクションが有効になるため，ユーザーはペイメントチャネルで安全な資金交換を行うことができない．

本稿ではこれらの問題点を指摘し，解決策を提案する．特に，手数料変動に対応するため，非協力的なペイメントチャネル終了時に追加手数料をユーザー間で公平に負担するプロトコルを提案する．本プロトコルでは，各ユーザーはあらかじめ追加手数料用の資金をペイメントチャネルにデポジットしておき，これを参照する子トランザクションを作成して手数料を追加する (Child Pays for Parent)．そして，余った資金を子トランザクションの 2 つのアウトプットにて各ユーザーに返却する．この 2 つのアウトプットに関して，片方のユーザーが額を決定し，もう一方のユーザーにそのどちらかを選択する権利を与えることで，非協力的な状況下でも追加手数料をユーザー間で折半することができる．本プロトコルを導入することにより，手数料変動に関わらず，ユーザーはペイメントチャネルで公平かつ安全な資金交換を行うことが可能になる．

目次

Chapter 1　序論

ビットコイン [36] は単位時間当たりに処理できるトランザクション数が少ないというスケーラビリティ問題を抱えている [15]．ビットコインは基盤システムにブロックチェーンを使用しているが，ブロックサイズの最大値は 1MB[1]，ブロックの生成間隔は約 10 分により，ビットコインは 1 秒間当たり最大で 10 トランザクション程度しか処理することができない．VISA が 1 秒間当たり 56,000 トランザクション処理できること [27] を考慮すると，ビットコインのスループットが極めて小さいことが分かる．

図 1.1 に，ビットコインの未承認トランザクション数の推移を示す．2017 年 5 月には約 19 万件の未承認トランザクションが存在していたことが読み取れる．短い時間に多くのトランザクションがブロードキャストされた場合，ブロックサイズの上限が設定されていることから，一部のトランザクションはすぐにブロックに追加されない．トランザクションがブロックに追加されないとビットコインの送金情報がネットワークに反映されないため，スケーラビリティ問題はビットコインの決済に時間がかかるといった問題を引き起こす．また，高い手数料を持つトランザクションほど優先的にブロックに追加されるため，トランザクション手数料の相場が上昇するといった問題も発生する．

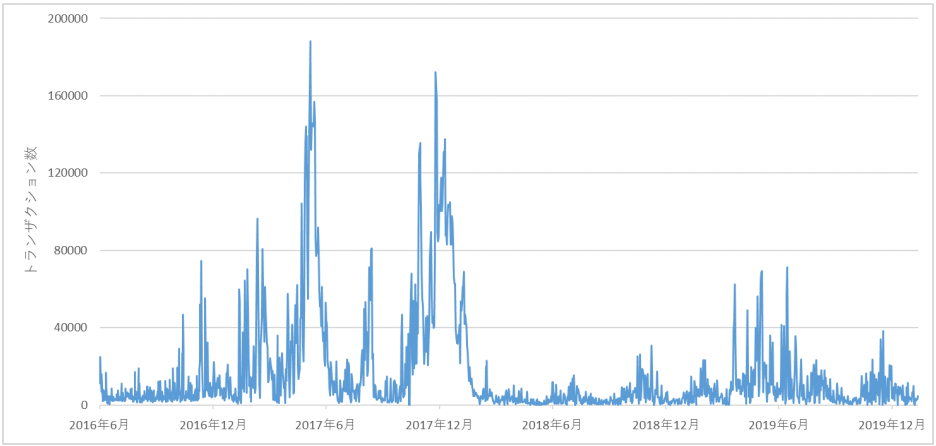

図 1.1: ビットコインの未承認トランザクション数 (データは [11] より取得)

スケーラビリティ問題解決に向けて，長きに渡りブロックサイズを大きくするという議論がなされている [4][24][41] が，これらの提案は問題を伴う．例えば，ブロックサイズを大きくするとノードに高い処理能力が要求されるため，ビットコインネットワークが特定のノードに集中してしまう可能性があることや，ブロック伝播遅延によっ

[1]厳密には 1MB のブロックサイズではなく，Segwit[34] にて導入されたブロックウェイトによって制限される．

て引き起こるチェーンの分岐の増加が問題点として挙げられる [18][25]．また，ブロック生成間隔の短縮においても同様にチェーン分岐の増加によってシステムが不安定になる [21]．このように，ブロックチェーンのプロトコル変更は様々な問題が発生するため，これらのプロトコル変更を必要としない，オフチェーンのアプローチが数多く提案されている [38][32][17][37]．

オフチェーンとはその名の通り，ブロックチェーンの外でトランザクションの処理を行う技術である．オフチェーンのアプローチは，ブロックチェーンの外に新しいレイヤを構築するため，レイヤ 2 やセカンドレイヤなどと呼ばれる．オフチェーンのアプローチでは，全てのトランザクションをブロックチェーン上に記載するのではなく，一部のトランザクションをブロックチェーンの外で管理し，これらの複数の処理を集約したトランザクションのみをブロックチェーンに記載する．このようにすることで，ブロックチェーン上に記載されるトランザクション数が減少し，スケーラビリティ問題の改善に繋がる．本稿では，オフチェーンのアプローチの根幹要素である，2 者がブロックチェーンの外で資金交換を行うペイメントチャネルに着目し，特にタイムロックを使用してトランザクション置換を行うペイメントチャネルを取り上げる [20][13]．ペイメントチャネルにおいて，オフチェーンで管理されるトランザクション (オフチェーントランザクション) はブロックチェーンプロトコルの範囲外なので，チャネル内で安全なオフチェーントランザクションの管理を行わなければならない．具体的には，過去の資金残高が反映された過去のトランザクションを無効にし，最新の資金残高が反映された最新のトランザクションのみが有効となる仕組みが必要となる．

[20][13] では，タイムロックを使用した時間制御により，最新のトランザクションのみが有効となる仕組みを構築している．しかし，ビットコインでは時間パラメータの誤差を許容する設計になっているため，タイムロックをトラストアンカーとして使用する際は，その誤差を考慮しなければならない．また，[20][13] はタイムロックの影響により，トランザクションを作成してからそれが有効になるまで時間差があるという特徴を持つ．ユーザー間で非協力的にペイメントチャネルを終了する場合，この時間内にトランザクション手数料の相場が上昇すると，相対的に低い手数料を持つ最新のトランザクションは優先的にブロックに追加されず，タイムロックを使用したトランザクション置換が失敗する可能がある．本稿ではこれらの要素を考慮し，[20][13] は時間パラメータの許容誤差や手数料変動に対して脆弱であることを指摘する．そして，これらの解決策を提案する．また，手数料変動への対抗策として，トランザクションへの追加手数料をユーザー間で公平に負担するプロトコルを提案する．

本稿の構成は以下の通りである．2 章でビットコインの説明を行い，3 章でペイメントチャネルの説明を行う．4 章で [20][13] の問題提起を行い，5 章で非協力的なペイメントチャネル終了時の公平な手数料追加プロトコルを提案する．6 章でこのプロトコルの評価を行い，最後に 7 章で本稿の結論を述べる．

Chapter 2　ビットコイン

ビットコイン [36] とは，中央集権的な管理者を排除した仮想通貨である．ビットコインはインターネット上の peer to peer ネットワークアーキテクチャとして構築されており，ビットコイントランザクションはブロックチェーンによって管理される．本章では主に，ビットコインの基盤システムであるブロックチェーンと，ビットコインの資金移動を示すトランザクションに関する説明を行う．なお，本章で言及する”ビットコイン”とは，ビットコインの通貨そのもの，もしくはビットコインのネットワークやソフトウェアの総称を指す．

2.1　ブロックチェーン

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 [29].

> 1）「ビザンチン障害を含む不特定多数のノードを用い、時間の経過とともにその時点の合意が覆る確率が 0 へ収束するプロトコル、またはその実装をブロックチェーンと呼ぶ。」
>
> 2）「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。」

ビットコインのトランザクションは，不特定多数のノードが自由に参加することができるパブリックブロックチェーンによって管理される．トランザクションはブロックと呼ばれるデータ単位に格納され，各ブロックは前のブロックのハッシュ値を保持している．ブロックチェーンのデータ構造を図 2.1 に示す．

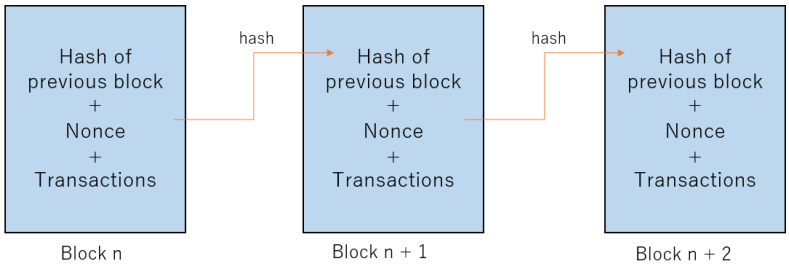

図 2.1: ブロックチェーンのデータ構造

表 2.1: ブロックヘッダの構成

サイズ (byte)	フィールド
4	version
32	previous block header
32	markle root
4	timestamp
4	difficulty target
4	nonce

図 2.1 に示す通り，ブロックのハッシュ値は主に前のブロックのハッシュ値，nonce，格納されるトランザクションによって決定される．トランザクションの集約情報はマークルルートとしてヘッダに記録される．各ブロックには difficulty と呼ばれるパラメータが設定されており，各ブロックのハッシュ値はこの difficulty を下回らなければならない．新しいブロックの生成を試みるノードは，difficulty を下回るハッシュ値を得るため nonce を総当たりで探索する必要がある．Nonce を探索し新しいブロックの生成を試みる作業のことをマイニングと呼び，これを行うノードのことをマイナーと呼ぶ．マイニングに成功したマイナーは，成功報酬としてビットコインを得ることができる．このようなマイニングの作業は Proof of Work(PoW) と呼ばれ，ノード間で合意形成を得るためのコンセンサスアルゴリズムとして使用されている．

悪意のあるユーザーがトランザクションの内容を書き換ると，そのトランザクションが格納されるブロックのハッシュ値が変化するので，ブロックチェーン全体の整合性が保たれなくなる．整合性を保つには，改ざんしたトランザクションが含まれるブロックから現在までの全てのブロックをマイニングする必要があるが，これを達成するには膨大な計算量が必要であり，極めて困難である．もっとも，悪意のあるノードが 51%以上[1] のハッシュレートを保持している場合，マイニングを独占することができるのでトランザクションの書き換えを行うことが可能となる．このような攻撃は 51%攻撃と呼ばれ，PoW の課題となっている．

異なるノードが，同一時刻に異なるブロックをブロックチェーンに追加するとブロックチェーンが分岐 (フォーク) する場合がある．しかし，ブロックチェーンでは最長のチェーンを有効と見なす設計となっているため，最終的にフォークは解消され一本のチェーンに収束する．最新の 1,2 ブロックはフォークによりブロックチェーンに取り込まれない可能性があるので，トランザクションの確定には一般的に 6 ブロック以上の承認を必要とする．

ブロックチェーンは単一の機関が管理するシステムではないので，サーバダウンやデータ紛失，悪意のある管理者がトランザクションを不正に書き換えるといった心配がない．トランザクションは分散して管理されており，コンセンサスアルゴリズムを使用することでノード間で合意形成を得ることができる．このような設計から，ブロックチェーンは耐障害性と耐改ざん性に優れたシステムであると言える．

[1]最近の研究では，51%攻撃は 50%以下のハッシュレートで実現できることが分かっている [22].

2.2　トランザクション

2.2.1　概要

ビットコインのトランザクションとは，ビットコインの資金移動をビットコインネットワークに示すものである．トランザクションは主にインプットとアウトプットから構成され，インプットにはビットコインの送金元情報が，アウトプットには送金先情報がビットコインアドレスとして記載される．例として，5BTC 保持しているアリスが，ボブに 2BTC 送金するトランザクションを図 2.2 示す．図 2.2 では省略しているが，インプットとアウトプットの差額が，このトランザクションを含むブロックをマイニングしたマイナーが受け取る手数料となる．マイナーは最大限の利益を得るために，単位データサイズ当たりの手数料の高いトランザクションを優先的にブロックに追加する [31].

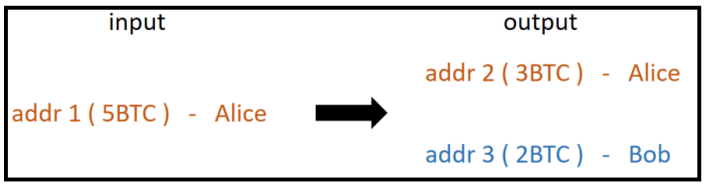

図 2.2: ビットコイントランザクション

トランザクションは，インプットの資金が全てアウトプットに移動する構成をとる．即ち，アリスのお釣りはアウトプットに記載される．このような未使用のアウトプット (UTXO:Unspent Transaction Output) は，将来のトランザクションのインプットとなり得るので，ビットコインの資金そのものである．

アウトプットには，そのアウトプットの所有者の公開鍵の情報が格納される．そして，そのアウトプットを使用する際に所有者は秘密鍵でトランザクションに対し署名を行い，その署名をトランザクションに格納する．署名が施されたトランザクションはビットコインネットワークにブロードキャストされ，各ノードによって検証されながらネットワーク内を伝播する．

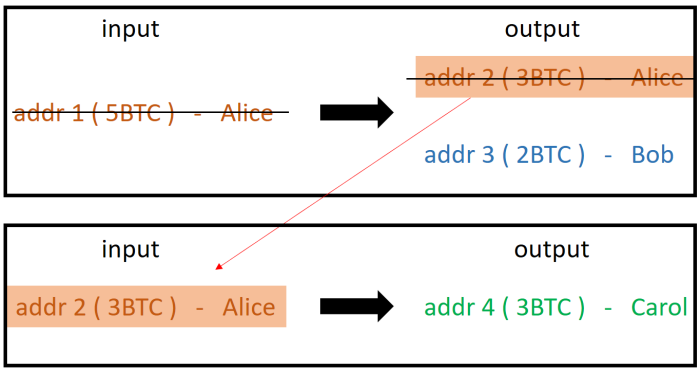

図 2.3: UTXO が使用される様子

2.2.2　構成

トランザクションの構成は，2017 年 8 月にアクティベートされた Segregated Witness(segwit)[34] の導入前後で異なる．本章では segwit 導入前の初期の構成を説明する．Segwit 導入後の構成に関しては 2.3 章を参照されたい．初期のトランザクションの構成を，以下に示す．

表 2.2: 初期のトランザクション構成

サイズ (byte)	フィールド
4	version
1-9	input counter
variable	inputs
1-9	output counter
variable	outputs
4	locktime

表 2.3: インプットの構成

サイズ (byte)	フィールド
32	transaction hash
4	output index
1-9	unlocking script size
variable	unlockcing script
4	sequence number

表 2.4: アウトプットの構成

サイズ (byte)	フィールド
8	amount
1-9	locking script size
variable	locking script

表 2.2 に示す通り，トランザクションは主にインプットとアウトプットから構成される．インプットに関して，トランザクションのハッシュ値はトランザクション ID(TXID) となり，アウトプットのインデックスと組み合わせて UTXO の参照先として使用される．Unlockcing script(scriptSig) には署名を含むスクリプトが格納される．アウトプットはトランザクションの送金額と，公開鍵の情報を保持する locking script(scriptPubKey) を保持している．署名の範囲は，SIGHASH フラグ [10] を使用して柔軟に選択することができる．デフォルトは SIGHASH_ALL であり，unlocking script を除く全てのトランザクションデータが署名される．

トランザクションがビットコインネットワークにブロードキャストされると，各ノードは unlocking script と locking script が連結されたスクリプトを検証し，トランザクションが有効であるか確認を行う．有効なトランザクションのみがリレーされ，後にブロックに追加される．

2.2.3　Script 言語

Script 言語とは，ビットコイントランザクションのスクリプト言語である．Script 言語はスタックベースの言語であり，locking script や unlocking script に使用される．unlocking script と locking script が連結されたスクリプトが検証され，スクリプトの実行が終わった段階でスタック上に TRUE が残っていれば，そのスクリプトは有効と見なされる．例として，以下のスクリプトを考える．

```
1  2  OP_ADD  3  OP_EQUAL
```

$1+2=3$ であり，最終的に TRUE が返ってくるのでこのスクリプトは有効である．検証されるスクリプトは unlocking script と locking script が連結されたものなので，このスクリプトは以下のように unlocking script と locking script に分割することができる．

- unlocking script: 1
- locking script: 2 OP_ADD 3 OP_EQUAL

この locking script は $x + 2 = 3$ となる x は何か，というロック条件を示している．locking script に対して，適切な unlocking script を使用することで有効なスクリプトを得ることができる．

実際には，locking script には公開鍵の情報が格納され，unlocking script には署名が格納される．この公開鍵に対応する秘密鍵を所有する者だけが有効な unlocking script を作成することが可能であり，ビットコインを使用することができる．Pay-to-Public-Key-Hash(P2PKH) というスクリプト形式の unlockcing script と locking script を以下に示す．

- locking script: OP_DUP OP_HASH160 <20 バイトの公開鍵のハッシュ> OP_EQUALVERIFY OP_CHECKSIG
- unlocking script: <署名> <公開鍵>

上記のスクリプトは図 2.4 に示すプロセスで評価される．

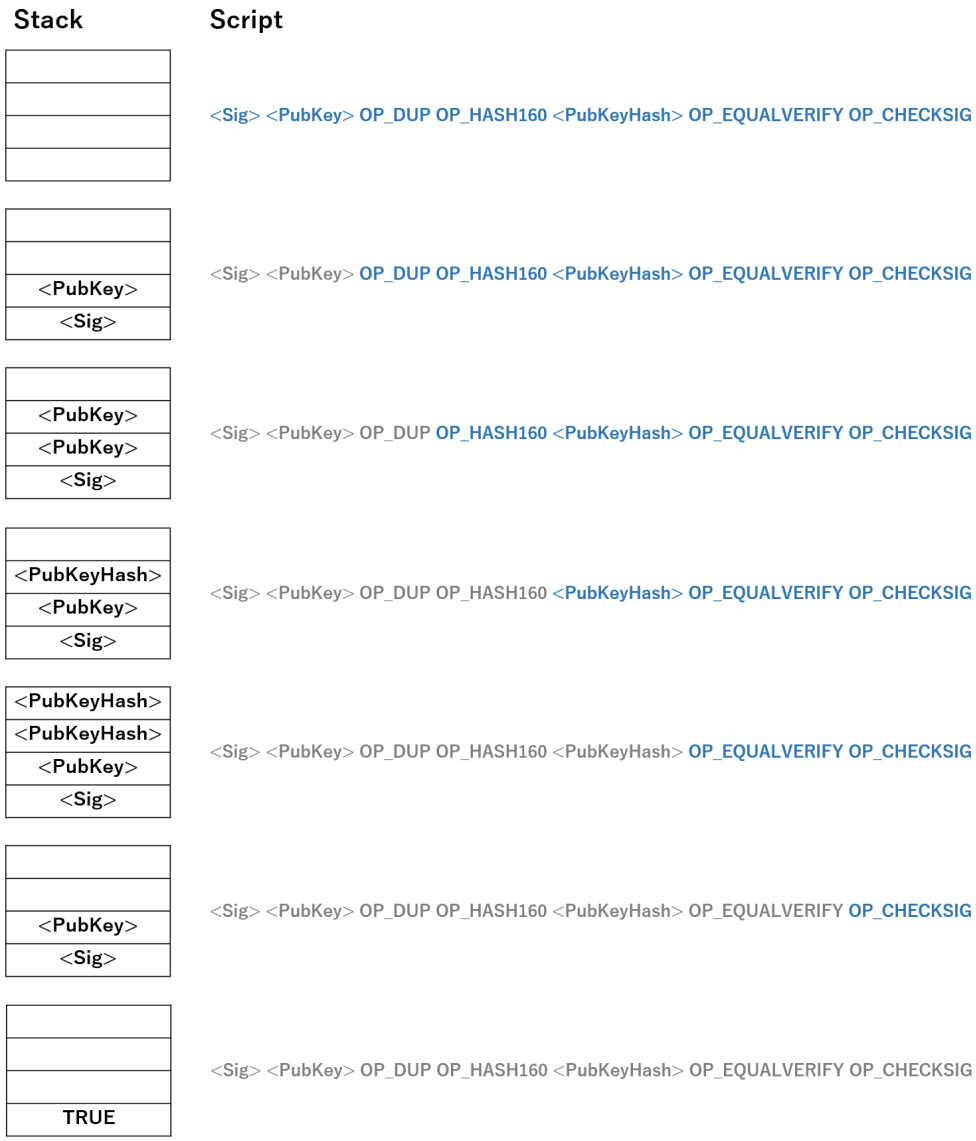

図 2.4: スクリプトが評価される様子

Locking script にロック条件を示すスクリプトに対して，locking script にスクリプトのハッシュ値を設定する Pay-to-script-hash(P2SH) がある．ロック条件を示すスクリプトは redeem script と呼ばれ，locking script には redeem script のハッシュ値が，unlocking script には redeem script が記載される．P2SH を使用することにより，RAM 内の UTXO セットのサイズを減らすことや，送金人が複雑なスクリプトを持つトランザクションを作成する負担を減らすことができる．P2SH の構成は以下の通りである．

- locking script: OP_HASH160 <20 バイトの redeem script のハッシュ> OP_EQUAL
- unlocking script: <アンロック条件> <redeem script>

P2SH は複雑なスクリプトを記載する際に使用されるが，P2SH の使用例としてマルチシグが挙げられる．マルチシグとは，ビットコインの送金に 1 つ以上の署名を必要とする技術である．マルチシグは秘密鍵のバックアップや，送金に複数人の合意を必要とする場合に使用される．特に，n 個の割り当てられた公開鍵の内，m 個の公開鍵に対応した秘密鍵を使用して署名を行うことで UTXO をアンロックするようなマルチシグのことを m-of-n マルチシグと呼ぶ．P2SH を使用しない場合，m 個の公開鍵全てを locking script に記載する必要があるが，P2SH を使用することで 20 バイトのハッシュ値に置き換えることができる．

2.2.4　タイムロック

タイムロックとは，指定された時間 (Unix 時間) もしくはブロック高に達するまでビットコインの送金を制限する機能である．この指定された時間は，トランザクションが追加されるブロックのタイムスタンプではなく，過去 11 ブロックのタイムスタンプの中央値である median time past[30] と比較される．これは，マイナーが虚偽のタイムスタンプをブロックに記録することで，本来ブロックに追加するべきではないトランザクションがブロックに追加されることを防ぐためである．

タイムロックは絶対時間/相対時間で制御するものと，トランザクションレベル/スクリプトレベルで制御するものに分類される．相対時間とは，トランザクションが参照する UTXO がブロックに追加されてからの相対時間を指す．タイムロックが設定されたトランザクションは，その指定された時間もしくはブロック高に達するまで，ブロックに追加されない．ビットコインで使用されるタイムロックを表 2.5 に示す．

表 2.5: タイムロックの分類

	トランザクションレベル	スクリプトレベル
絶対時間	nLocktime	OP_CLTV[40]
相対時間	nSequence(BIP68)[23]	OP_CSV[12]

トランザクションレベルのタイムロックは将来の支払いを可能にするが，必ずしもその支払いは保証されない．これは，トランザクションにタイムロックは設定されているが，トランザクションが参照する UTXO にはタイムロックが設定されていないからである．例として，アリスがボブに将来支払いを行う，タイムロックが設定されたトランザクション TX_{ab} を考える．ボブは将来 TX_{ab} で資金を受け取ることを期待する．しかし，アリスは TX_{ab} が参照する UTXO を使用して，キャロルへの支払いを行う別のトランザクション TX_{ac} も作成することが可能である．TX_{ac} がブロックに追加されると，TX_{ab} は無効となりボブは資金を受け取ることができない．

これに対し，スクリプトレベルのタイムロックは，アウトプットにタイムロックが設定されいる．指定された時間に達するまで，タイムロックが設定されたアウトプットは使用することができない．従って，上記の問題点はスクリプトレベルのタイムロックを使用することで解決することができる．

2.3 Segregated Witness

Segregated Witness(Segwit)[34] とは，トランザクションの署名をインプットから witness と呼ばれる別のデータ領域に移す仕様である．Segwit 導入前の初期の構成では，署名を含む unlocking script が TXID に算出に関わっていたため，署名の妥当性を失わずに TXID の変更が可能となる malleability 問題 [19] を抱えていた．例えば，unlocking script に何もしないというオペレータである OP_NOP を加えることで，スクリプトの実行に影響を与えずに TXID を変更することができる．Segwit が導入され TXID の計算方法が変更となり，この malleability 問題は解決された．

また，segwit の導入に伴いブロックサイズの計算ルールが変更され，より多くのトランザクションをブロックに含めることが可能になった．本章では segwit 導入によるビットコインの変更点を紹介する．

2.3.1 トランザクションの構成

Segwit 導入後のトランザクション構成を，表 2.6 に示す．

表 2.6: Segwit 導入後のトランザクション構成

サイズ (byte)	フィールド
4	version
1	marker
1	flag
1-9	input counter
variable	inputs
1-9	output counter
variable	outputs
variable	witness
4	locktime

Segwit の構成では，marker，flag，witness フィールドが新たに追加される．署名はインプットではなく，witness に格納されることが主な変更点となる．Segwit トランザクションも初期のトランザクションと同様に TXID が計算されるため，segwit トランザクションは TXID の決定に署名が影響を与えない．

2.3.2 スクリプト

Segwit の導入により，pay-to-witness-pubkey-hash(P2WPKH) と pay-to-witness-script-hash(P2WSH) の 2 つの新しいスクリプト形式が追加される．これらを使用する場合，署名は witness に格納されるため scriptSig は空となる．P2WPKH，P2WSH の scriptPubKey と witness の構成は以下の通りである．

- P2WPKH
  - scriptPubKey: 0 <20 バイトの公開鍵のハッシュ>
  - witness: <署名> <公開鍵>
- P2WSH
  - scriptPubKey: 0 <32 バイトの redeem script のハッシュ>
  - witness: <アンロック条件> <redeem script>

2.3.3　ブロックウェイト

Segwit の導入に伴い，ブロックサイズの計算ルールが変更された．以下のブロックウェイトという概念が新たに導入される．

$$Block\,weight = Base\,size * 3 + Total\,size$$

ベースサイズとは，witness に関連しないデータを既存のシリアライゼーション形式でシリアライズしたバイト数である．表 2.6 に関して，marker，flag，witness を除くデータが対象となる．トータルサイズとは，witness に関連するデータを含む，全てのデータを Segwit 用のシリアライゼーション形式 [35] でシリアライズしたバイト数である．1MB のブロックサイズのルールは，4MB のブロックウェイトのルールへと変更になり，各ブロックのウェイトは 4MB 以下になる必要がある．

ブロック内のトランザクションが全て segwit 未対応だと仮定すると，ベースサイズとトータルサイズが変わらないため，$Total\,size * 4 <= 4MB$ となり，1MB のブロックサイズのルールと変わらない．これに対し，segwit 対応のトランザクションはベースサイズが比較的小さいため，より多くのトランザクションをブロックに格納することができる．例えば，ブロック内のトランザクションが全て segwit 対応であり，ベースサイズとトータルサイズの割合が 1:2 と仮定すると，$Total\,size * \frac{5}{2} <= 4MB$ となり，1 つのブロックに約 1.6MB のトランザクションを格納することが可能となる．

Chapter 3　ペイメントチャネル

3.1　概要

ペイメントチャネルとは，2 者がブロックチェーンの外で複数のトランザクションを管理する (資金の交換を行う) 手法である．複数のトランザクションをブロックチェーンの外で管理し，それらの結果を集約したトランザクションのみをブロックチェーンに記載する．このようにすることで，ブロックチェーンに記載されるトランザクション数が減少し，ユーザーは手数料削減や送金時間短縮，ビットコインはスケーラビリティ問題改善に繋がる．本章では特に，2 者間で双方向の送金を可能にする双方向ペイメントチャネルの説明を行う．ペイメントチャネル構築から終了までのプロセスを以下に示す．2 の実装はペイメントチャネルにより異なるが，ここでは簡略化した基本的なモデルを紹介する．

1. チャネルの構築：2-of-2 マルチシグにチャネル内で使用する資金をデポジットするファンディングトランザクションを作成し，ビットコインネットワークにブロードキャストする.

2. チャネル内での資金交換：2-of-2 マルチシグをインプットにとり，各ユーザーをアウトプットにとるコミットメントトランザクションを作成する．アウトプットの額が各ユーザーの残高となる．

3. チャネルの終了：最終的な残高を反映したセトルメントトランザクションを作成し，ビットコインネットワークにブロードキャストする．

1 のチャネルの構築に関して，片方のユーザーが音信不通になった場合にデポジットした資金が永久にロックされるので，ファンディングトランザクションを作成する前に，最初のコミットメントトランザクションかリファンドトランザクション[1]を作成する．ユーザーはファンディングトランザクションの署名前に，ファンディングトランザクションを参照するコミットメントトランザクションを作成するので，署名前後でファンディングトランザクションの TXID が変化しないことが必要になる．Segwit が導入され署名が TXID の算出に関わらなくなったので，双方向ペイメントチャネルを実装することが可能になった．

[1] ファンディングトランザクションでデポジットした資金をユーザーに返却するトランザクションである．リファンドトランザクションにはタイムロックが設定されており，この指定された時間に達する前にチャネルを終了しなければならない．

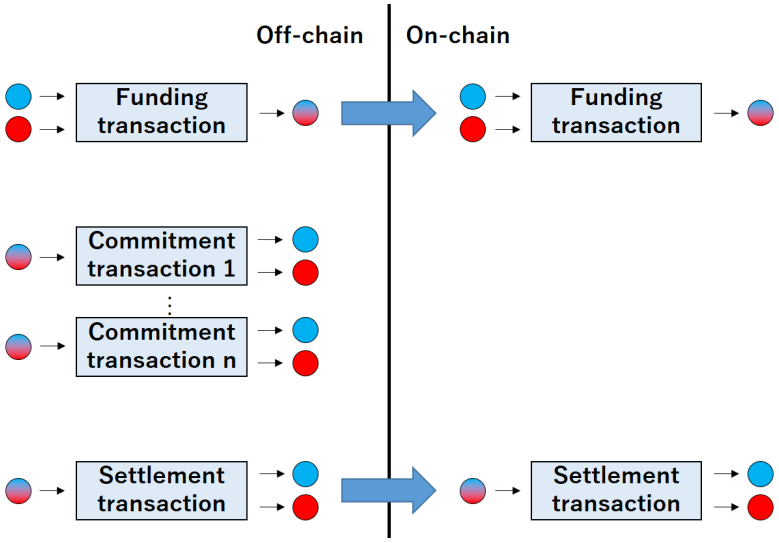

図 3.1: ペイメントチャネルの概要

資金の交換は，コミットメントトランザクションを発行することで行われる．各ユーザーのアウトプットの額を更新したコミットメントトランザクションを発行することで，更新された額だけ資金の交換が行われることになる．例として，アリスとボブがペイメントチャネル内で資金交換を行う状況を考える．両者が共にペイメントチャネルで 0.5BTC 使用したいと考えている場合，ファンディングトランザクションは図 3.2 のようになる．なお，赤のインプット/アウトプットはアリス，青のインプット/アウトプットはボブ，赤と青が混ざったインプット/アウトプットは 2-of-2 マルチシグを表している．

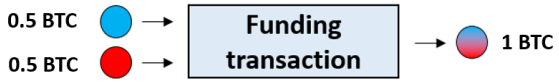

図 3.2: ファンディングトランザクションの例

次に，アリスがボブに 0.4BTC 送金することを考える．0.4BTC 送金後，アリスの資金残高は 0.1BTC，ボブの資金残高は 0.9BTC に変化する．同様に，続いてボブがアリスに 0.8BTC 送金する場合，両者の資金残高はアリスが 0.9BTC，ボブが 0.1BTC となる．1 回目の送金を反映したコミットメントトランザクション 1，2 回目の送金を反映したコミットメントトランザクション 2 は図 3.3 のようになる．

図 3.3: コミットメントトランザクションの例

コミットメントトランザクションは 2-of-2 マルチシグをインプットにとるので，資金の交換は必ず両者の合意のもと (署名のもと) 行われる．複数回トランザクションを発行しても，ブロックチェーン上にはファンディングトランザクションとセトルメントトランザクションの 2 つしか記載されない．片方のユーザーが合意せずセトルメントトランザクションを作成できない場合は，最新のコミットメントトランザクションをブロードキャストすることでチャネルを終了することができる．

ペイメントチャネルを使用する上で注意するべきことは，オフチェーンで管理されるコミットメントトランザクションの取り扱いである．具体的には，**最新のコミットメントトランザクションのみを有効にする仕組み**が必要となる．過去のコミットメントトランザクションには過去の残高が反映されているので，これが有効になると最新の取引が反映されず，最新のコミットメントトランザクションで資金を受け取るユーザーは相手に資金を盗まれる事態が発生する．

	Balance(BTC)		description
	Alice	Bob	
initial	0.5	0.5	-
commitment transaction 1	0.1	0.9	Alice sends 0.4BTC to Bob
commitment transaction 2	0.9	0.1	Bob sends 0.8BTC to Alice

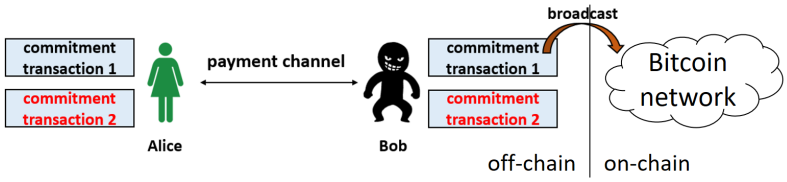

図 3.4: ペイメントチャネルで資金が盗まれる様子

図 3.3 の資金交換の様子を図 3.4 に示す．コミットメントトランザクション 2 に現在の残高が反映されているので，これが有効になることが期待される．この時の両者の残高はアリスが 0.9BTC, ボブが 0.1BTC である．しかし，悪意を持ったボブがコミットメントトランザクション 1 をビットコインネットワークにブロードキャストし，これがブロックに追加されると，ブロックチェーン上にはアリスの残高が 0.1BTC, ボブの残高が 0.9BTC という情報が記載される．この時，アリスはボブに 0.8BTC 奪われる事態が発生するので，先述の通りペイメントチャネルでは最新のコミットメントトランザクションのみを有効にする仕組みが必要となる．

3.2　タイムロックの使用

3.1 章にて，ペイメントチャネルでは過去のコミットメントトランザクションを無効にし，最新のコミットメントトランザクションのみを有効にする仕組みが必要であることを述べた．トランザクションを無効にする仕組みとして，タイムロックがある．なお，ここで言及するタイムロックとは，トランザクションレベルのタイムロックである．

3.2.1　タイムロックを使用したトランザクション置換

過去のトランザクションを無効にするには，最新のトランザクションのタイムロックに，過去のトランザクションのタイムロックに設定した値よりも早い値を設定すれば良い．

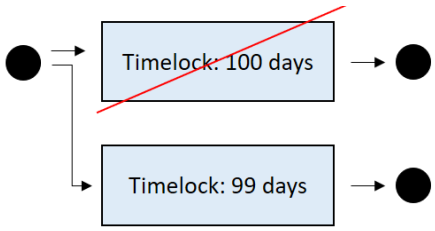

図 3.5: タイムロックを使用したトランザクション置換

図 3.5 に，タイムロックを使用してトランザクションを無効にする様子を示す．図 3.5 では，100 日のタイムロックが設定されたトランザクションを，99 日のタイムロックが設定されたトランザクションで無効にしている．これは，99 日後に下のトランザクションは有効になるが，その時点で上のトランザクションはまだ有効ではないからである．

3.2.2　ペイメントチャネルへの応用

タイムロックを使用したトランザクション置換を使用して，コミットメントトランザクションの管理を行うペイメントチャネルが提案されている [20][13]．絶対タイムロッ

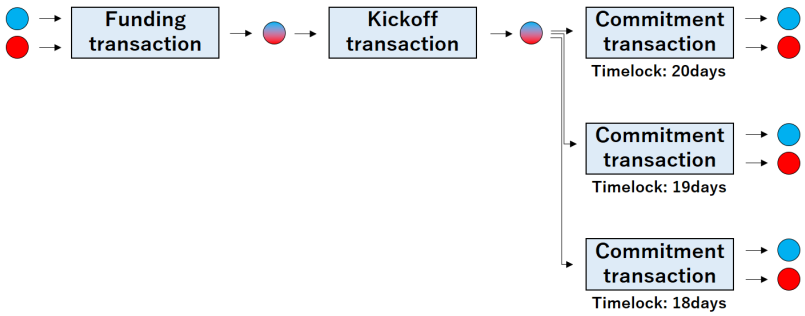

図 3.6: 相対タイムロックを使用したコミットメントトランザクションの置換

クを使用すると，その絶対時間に達した時点でチャネルを終了しなければならないので，相対タイムロックを使用したペイメントチャネルが提案されている．相対タイムロックを使用する場合は，相対値の起点となるキックオフトランザクションを作成する．キックオフトランザクションが承認されてからの相対時間によってタイムロックは計算されるので，キックオフトランザクションをブロードキャストしない限り，有効期限のないチャネルを作成することができる．チャネル内で使用するタイムロックの最大値を T_{max}，最小値を T_{min}，間隔を ΔT とすると，コミットメントトランザクションの更新回数は最大で $(T_{max} - T_{min})/\Delta T$ となる．一般的に，二重支払い防止のためにトランザクションは 6 承認以上を必要とするため，ΔT の最小値は 1 時間とされている [20]．

図 3.7 のように，タイムロックで無効化するコミットメントトランザクションの層を追加することで，コミットメントトランザクションの更新回数を増やすことができる．このような多層のコミットメントトランザクションの構造は invalidation tree と呼ばれる [20]．図 3.7 では，タイムロックが 1 番早いトランザクションの枝 (Timelock:18days - Timelock:18days) のみが有効となる．

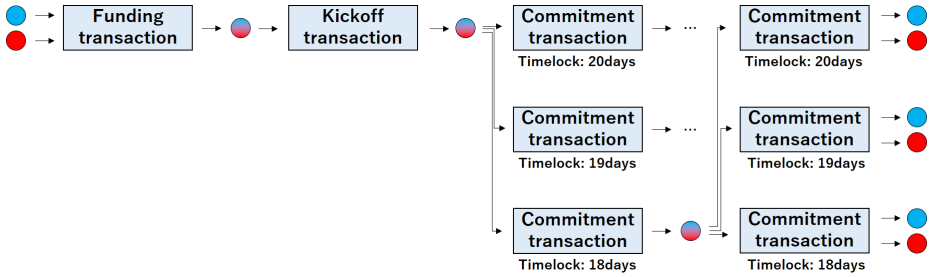

図 3.7: 相対タイムロックを使用した invalidation tree

チャネルの終了は，2 者が協力する場合と，協力しない場合で異なる．協力する場合は最新の残高を反映したセトルメントトランザクションを作成し，これをビットコインネットワークにブロードキャストする．協力しない場合は，キックオフトランザクションと最新のコミットメントトランザクションをブロードキャストする．この時，コミットメントトランザクションにはタイムロックが設定されているため，タイムロックが有効になるまで 2 者は資金を使用することができない．

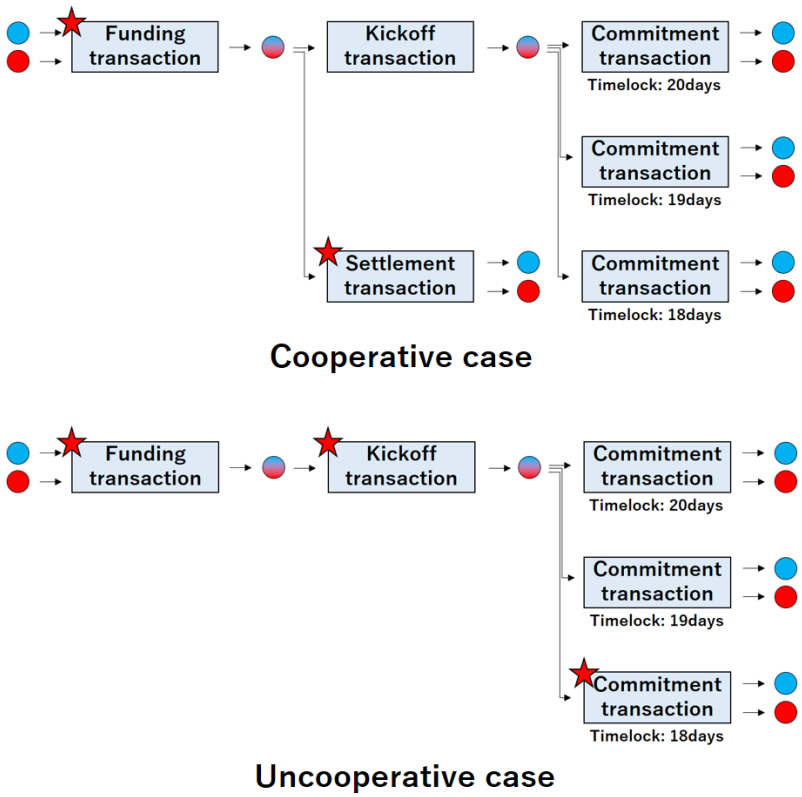

図 3.8: 協力的，非協力的なペイメントチャネル終了の様子．星マークのトランザクションがブロードキャストされる．

3.3　関連研究

3.1 章で言及した通り，ペイメントチャネルではコミットメントトランザクションの管理が設計の肝となる．関連研究として，ペナルティを設定することで過去のコミットメントトランザクションをブロードキャストする動機を与えないペイメントチャネルと，Trusted Execution Environment(TEE) を使用してトランザクションの管理を行うペイメントチャネルを紹介する．また，ペイメントチャネルを拡張し複数人との資金交換を行うための技術として，Hashed Time-Locked Contract を紹介する．

3.3.1　Lightning Network

Lightning Network[38][39] とは，オフチェーン上で不特定多数のユーザーと資金交換を行う技術である．本章では特に Lightning Network で使用されるペイメントチャネルに着目し，便宜上このペイメントチャネルを Lightning Channel と名付ける．Lightning Channel では，悪意のあるユーザーが過去のコミットメントトランザクションを不正にブロードキャストした際に，取引相手はそのユーザーの資金を入手できるような設計となっている．このようなペナルティを設定することで，ユーザーに不正を行う動

機を与えないことができる．Lightning Channel の概要図を図 3.9 に示す．

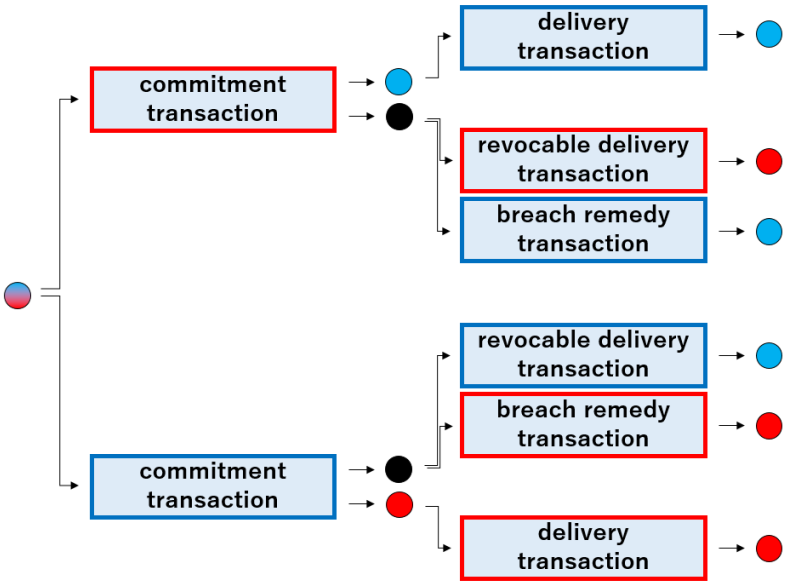

図 3.9: Lightning Channel の概要図

各ユーザーは異なるアウトプットを持つコミットメントトランザクションを保持する．図 3.9 において，赤枠で囲まれたトランザクションはアリスがブロードキャストするもの，青枠で囲まれたトランザクションはボブがブロードキャストするものを表している．ここではアリスの視点から各トランザクションの説明を行う．アリスのコミットメントトランザクションを参照するトランザクションは，以下の通りである．なお，アリスのシークレットを Sa，ボブのシークレットを Sb とおく．

- delivery transaction : ボブへの支払いを行うトランザクションである．ボブはボブの資金を直ぐに入手することができる
- revocable delivery transation : アリスへの支払いを行うトランザクションである．タイムロックが設定されており，アリスは一定時間後[2]にアリスの資金を入手することができる．
- breach remedy transaction : アリスが過去のトランザクションを不正にブロードキャストした際に，ペナルティとしてボブがその資金を受け取るトランザクションである．ボブは Sa を使用し，アリスの資金を入手することができる．

両者は i 回目の資金交換を行うコミットメントトランザクション i の作成時に，コミットメントトランザクション $i-1$ で使用したシークレット Sa_{i-1}，Sb_{i-1} を取引相手に公開し，Sa_i，Sb_i のハッシュ値を交換する．Sa_i，Sb_i のハッシュ値はコミットトランザクション i を作成する際に使用される．両者は取引相手の過去のシークレット

[2]コミットメントトランザクションがブロックに追加されてからの相対時間

を保持しているので，取引相手が不正に過去のコミットメントトランザクションをブロードキャストした場合，シークレットを利用して breach remedy transaction を作成，ブロードキャストし取引相手の資金を入手することができる．このペナルティにより，各ユーザーは過去のコミットメントトランザクションをブロードキャストする動機がなくなる．

非協力的にチャネルを終了する場合は，セトルメントトランザクションを作成できないので最新のコミットメントトランザクションをブロードキャストする．アリスが最新のコミットメントトランザクション i ブロードキャストしても，ボブは Sa_i を知らないのでアリスの資金を得ることはできない．アリスは一定時間後に，revocable delivery transaction を作成しブロードキャストすることで，自身の資金を得ることができる．

3.3.2　Teechan

Trusted Execution Environment(TEE) を使用した，Teechan[33] というペイメントチャネルが提案されている．他のペイメントチャネルとは異なり，Teechan はトランザクションの管理を TEE が担っている．Teechan は Intel の Software Guard Extensions(SGX)[28][14] の enclave を TEE として使用している．

Intel SGX とは，TEE を備えた CPU の命令セットである．SGX を使用することで，攻撃者がユーザのデータにアクセスすることを防ぐことができる．Teechan では主に SGX が提供する enclave と remote attetation という機能を使用している．それぞれの概要は以下の通りである．

- Enclave：重要なデータを格納するメモリ内の領域である．Enclave 内のデータは特権モードでもアクセスすることが不可能であり，特定のプログラムからのアクセスしか許可されない．
- Remote attestation：リモートの enclave 同士で認証を行う機能である．所望のコードが通信相手の enclave 内で動作していること確認することができる．

Teechan の構成を図 3.10 に示す．

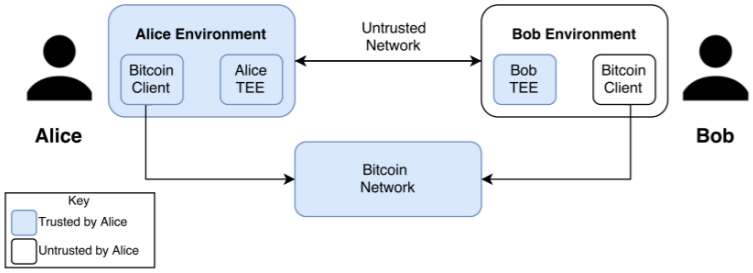

図 3.10: Teechan の構成 [33]

アリスとボブは互いに自身のマシン上に TEE を立ち上げる．この時アリスは自身の環境に加え，ボブが所有するマシンの中にある TEE_B も信頼している．これは，enclave によりボブは TEE_B 内の情報へ不正にアクセスできないからである．TEE_A は remote attestation を使用し，TEE_B 内で Teechan のアプリケーションが正常に動作していることを確認する．TEE_B 内のプログラムが悪意をもって改ざんされていないことを確認した後，アリスは TEE_A を介して TEE_B に，ファンディングトランザクションにデポジットする資金の UTXO 情報，金額，UTXO をアンロックする秘密鍵を送信する．ボブも同様に，TEE_B を介して TEE_A にこれらの情報を送信する．

TEE_B はファンディングトランザクションを作成し，ボブにこれを送信する (TEE_A が作成し，アリスに送信してもよい)．ボブはファンディングトランザクションをビットコインネットワークにブロードキャストし，チャネルを構築する．アリスは TEE_A からファンディングトランザクションの TXID を受け取っており，チャネルが構築されたことを確認することができる．

信頼できる第三者として振る舞う TEE_A と TEE_B は，アリスとボブの両者の秘密鍵を保持している．TEE はファンディングトランザクションのアウトプットである 2-of-2 マルチシグをアンロックすることができるので，各ユーザは TEE にリクエストを送ることで，有効なトランザクションを得ることができる．

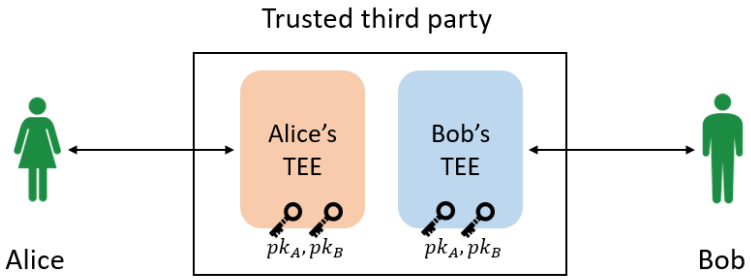

図 3.11: 信頼できる第三者として振る舞う TEE

チャネル内での支払いの更新は，TEE に更新リクエストを送信することで行われる．アリスは TEE_A へ，ボブは TEE_B へ支払の更新リクエストを行い，リクエストを受け取った TEE はアリスとボブの最新の取引を更新する．

チャネルの終了は，アリスもしくはボブが任意のタイミングで TEE に終了リクエストを送信することで行われる．リクエストを受け取った TEE は両者の秘密鍵を用いて最新の取引を反映したトランザクションに署名を行い，リクエストを送信したユーザーにセトルメントトランザクションを送信する．ユーザーがこれをビットコインネットワークにブロードキャストすることで，チャネルを終了することができる．

3.3.3　Hashed Time-Locked Contract

Hashed time-locked contract(HTLC)[7] とは，ハッシュとタイムロックを使用して，送金人が受取人に対して支払いを保証する手法である．受取人が受け取りの意思を示し

た場合は必ず支払いが行われ，反対に意思を示さなかった場合は送金人に資金が返却される．HTLC は中間者を経由して資金の支払いを行う場合や，異なるブロックチェーン間で仮想通貨の交換を行う atomic swap[26] などで使用される．本章では，アリスがボブを経由してキャロルに送金する例を紹介する．

中間者を経由して送金する場合，送金プロセスの途中でユーザーが不正を行うことができない仕組みが必要となる．例えば，中間者であるボブが悪意を持っている場合，アリスから受けとった資金をキャロルに送金せずに持ち逃げるという事態も考えられる．HTLC は各ユーザーの支払いを保証するので，上記のような事態を防ぐことができる．HTLC を使用した送金の流れを以下に示す．

1. キャロルは自身しか知らないシークレットを生成し，そのハッシュ値をアリスに送信する．
2. アリスは以下のアンロック条件を持つトランザクション TX1 を作成し，ビットコインネットワークにブロードキャストする．
    - タイムロックに指定した時間経過後，アリスの署名
    - シークレットとボブの署名
3. ボブも同様に，以下のアンロック条件を持つトランザクショTX2 を作成し，ビットコインネットワークにブロードキャストする．
    - タイムロックに指定した時間経過後，ボブの署名
    - シークレットとキャロルの署名
4. キャロルは TX2 のアウトプットを参照するトランザクションを作成，ブロードキャストし，資金を入手する．この時点で，シークレットの値は公開される．
5. シークレットの値を入手したボブは，TX1 のアウトプットを参照するトランザクションを作成，ブロードキャストし，資金を入手する．

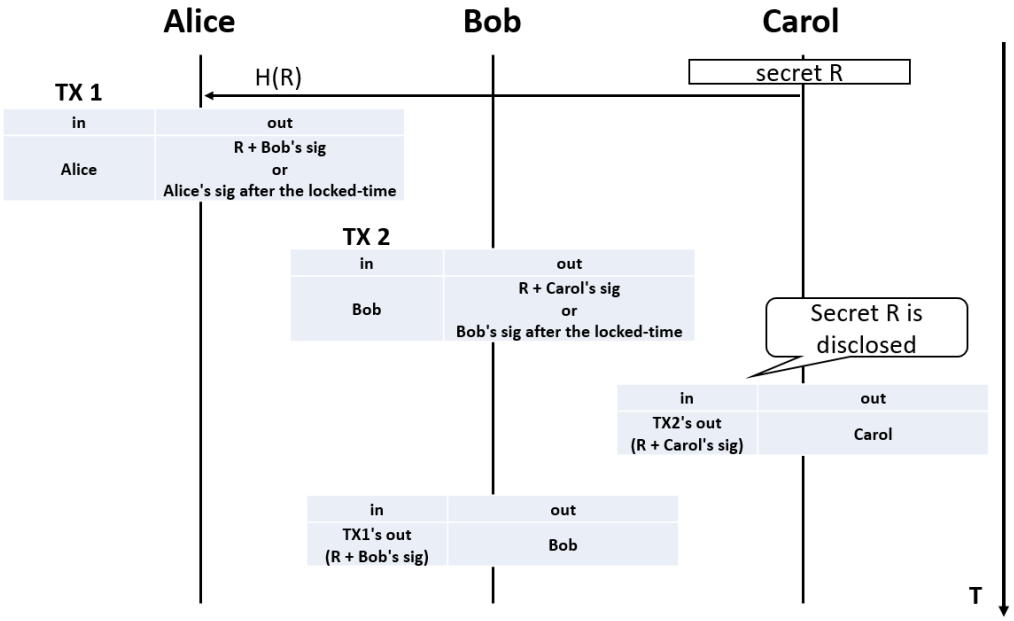

図 3.12: HTLC を使用した送金の流れ

キャロルが資金を受け取るためにトランザクションをブロードキャストすると，シークレットの値が公開されるため，TX1 と TX2 において受取人は資金を入手することが可能となる．反対に，キャロルが受け取りを放棄した場合，シークレットが公開されないので，TX1 と TX2 において送金人は一定時間後に資金を回収することができる．ボブが TX2 をブロードキャストしなかった場合においても，アリスは一定時間後に TX1 の資金を回収することが可能である．

このように，シークレットとタイムロックを使用した制御により，送金人が受取人に対して支払いの保証を行うことができる．また，HTLC を使用することで複数のペイメントチャネルを繋ぐことが可能になり，様々なユーザーへの送金を可能にするペイメントネットワークを構築することができる．

Chapter 4　問題提起

3.2 章で紹介した，タイムロックを使用してトランザクション置換を行うペイメントチャネル [20][13] は，時間を基準にしてコミットメントトランザクションの管理を行っている．しかし，トラストアンカーとして使用している時間パラメータの許容誤差や，トランザクションの優先度 (承認時間) を決定づける手数料を考慮していない．本章では，これらの要素に対し [20][13] は脆弱であることを指摘する．

4.1　時間パラメータの許容誤差に対する脆弱性

タイムロックを使用したトランザクション置換において，トランザクションに設定するタイムロックの間隔の最小値は 1 時間とされていた [20]．これは，トランザクションの確定に (ビットコインの決済に)6 ブロックの承認を必要とするからである．本章ではこの 1 時間という値に着目し，時間パラメータの許容誤差を考慮するとこれが不十分であることを指摘する．

4.1.1　時間パラメータの許容誤差

ビットコインネットワークには実時間とは別にネットワーク特有の内部時間がある．これを network-adjusted time と呼ぶ．Network-adjusetd time は，自身のノードと接続されている全てのノードから送られてくるタイムスタンプの中央値によって決定される．ただし，network-adjusted time が自身のシステム時間と 70 分以上離れている場合は無視される [6]．接続ノードから送信されるタイムスタンプで network-adjusetd time が決定されることから，悪意を持った接続ノードが，虚偽のタイムスタンプを送信し犠牲者の network-adjusted time を操作する timejacking[3] という攻撃が指摘されている．

また，ブロックチェーンのブロックにはタイムスタンプが格納されているが，このタイムスタンプは信頼できる第三者ではなくマイナーが発行するために信頼できる値ではない．そのため，ブロックのタイムスタンプは以下の条件を満たす場合にのみ有効と判断される [6]．

- 過去 11 ブロックのタイムスタンプの中央値より大きい
- network-adjusted time に 120 分追加した値より小さい

ブロックチェーンは時系列順にブロックが繋がっていくが，これらのブロックのタイムスタンプは時系列となる必要はない．例えば，ビットコインにおける 480048 番目

のブロックのタイムスタンプ[1] は，480047 番目のタイムスタンプ[2] よりも早い値となっている．

4.1.2　攻撃モデル

Network-adjusetd time やブロックタイムスタンプは誤差を許容する設計になっているため，攻撃者はこれらの時間パラメータを操作することができる．以下，時間パラメータを操作することでペイメントチャネル内の資金を奪う攻撃モデルを示す．

非協力的にチャネルを終了する場合を考える．前提として，コミットメントトランザクションのタイムロックの間隔 ΔT は最小値の 1 時間とし，犠牲者は network-adjusetd time でトランザクションをブロードキャストする時間を管理していると仮定する．また，攻撃者は 2 番目に新しいコミットメントトランザクション (TX_{n-1}) で犠牲者から資金を受け取り，最新のコミットメントトランザクション (TX_n) で犠牲者に資金を送るものとする．例えば，チャネル内で 1BTC の資金交換を行うと図 4.1 のようになる．

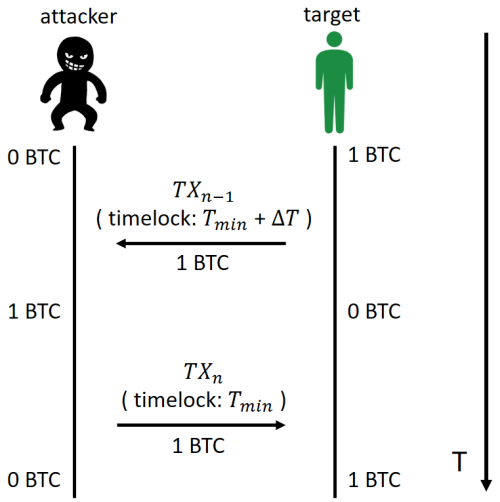

図 4.1: ペイメントチャネルでの資金交換の様子

図 4.1 において，攻撃者と犠牲者の資金残高はそれぞれ，TX_{n-1} 終了時は 1BTC, 0BTC、TX_n 終了時は 0BTC、1BTC である．TX_n 終了時に，攻撃者が TX_{n-1} をブロードキャストしこれがブロックチェーンに追加されると，攻撃者は犠牲者から 1BTC の資金を奪うことに成功する．

この攻撃を成功させるためには，(a) 犠牲者に TX_n をブロードキャストさせる時間を遅らせ，(b)TX_{n-1} のタイムロック $(T_{min} + \Delta T)$ が有効になる時間を早めることが必要になる．(a) を達成するには timejacking を使用して犠牲者の network-adjusted time を早めればよい．また，タイムロックは median time past と比較されることから，(b) を達成するには過去 11 ブロックに格納されるブロックのタイムスタンプを遅くし，

[1]https://www.blockchain.com/btc/block/00000000000000000e614b038b284b938503e70704c4cc7fb8fbb43206d89f8
[2]https://www.blockchain.com/btc/block/00000000000000000012f25df12067a39f184c57c340137c742da82aec95ea398

median time past を遅らせれば良い．以下，攻撃のプロセスを示す．なお，各ブロックは 10 分毎にマイニングされるものとする．

1. 攻撃者は犠牲者に対し timejacking を仕掛け，犠牲者の network-adjusted time を実時間から 70 分早める．

2. 攻撃者は，犠牲者の network-adjusted time に対し 120 分遅いタイムスタンプを含んだブロックをマイニングする．

3. Median time past が $T_{min} + \Delta T$ より遅くなったタイミングで，攻撃者は TX_{n-1} をブロードキャストする．

4. TX_{n-1} がブロックに追加されると，攻撃者は犠牲者から資金を奪うことに成功する．

1 と 2 に関して，前章で紹介した通りこれらの値は許容誤差の範囲内なので，犠牲者は上記の network-adjusted time やブロックタイムスタンプを有効なものとして受け入れる．具体例として，T_{min} が 12:00 に設定されており，現在の実時間が 13:10 で，攻撃者が過去 11 ブロックの内最後の 6 ブロックを連続してマイニングに成功した状況を考える．この時の実時間，ブロックに格納されるタイムスタンプ，犠牲者の network-adjusted time は以下のようになる．犠牲者は network-adjusted time で時刻を管理していると

表 4.1: 実時間，ブロックタイムスタンプ，犠牲者の network-adjusted time

real time	block timestamp (target's network-adjusted time + 120)	target's network-adjusted time (real time - 70)
12:10	13:00	11:00
12:20	13:10	11:10
12:30	13:20	11:20
12:40	13:30	11:30
12:50	13:40	11:40
13:00	13:50	11:50
13:10	13:10	12:00

仮定しているので，犠牲者が現在時刻は 12:00(T_{min}) であると思っている時，実時間は 13:10 である．この時，ブロックチェーンの過去 11 ブロックのタイムスタンプは順に 11:20, 11:30, 11:40, 11:50, 12:00, ”13:00”, ”13:10”, ”13:20”, ”13:30”, ”13:40”, ”13:50” である (””で囲まれたタイムスタンプは攻撃者が生成したもの)．犠牲者が TX_n をブロードキャストする前，既に median time past は 13:00($T_{min} + \Delta T$) であり，この時点で TX_{n-1} は有効である．即ち，攻撃者がこのタイミングで TX_{n-1} をブロードキャストしこれがブロックに追加されると，犠牲者の資金を奪うことに成功する．このように，犠牲者が network-adjusted time で時刻を管理しており，タイムロックの間隔 ΔT が 1 時間であると，時間パラメータの許容誤差 (システム正常) の範囲内でトランザクション置換が失敗する場合がある．従って，1 時間という値は不十分である．

実際には，連続して複数のブロックをマイニングすることは極めて難しい．しかし，連続してマイニングすることが出来なくても一定数 median time past を遅くすることは可能であり，ユーザーが期待する正常な動作を妨げることができる．

4.1.3　解決策

4.1.2 章で言及した攻撃は，Unix 時間ではなくブロック高でタイムロックを設定することや，median time past の値を監視すること，タイムロックの間隔 ΔT を大きくすることなどで解決できる．タイムロックの間隔 ΔT の値は，トランザクションの確定に 1 時間かかることに加え，network-adjusted time が最大 70 分の誤差を許容していること，ブロックタイムスタンプは上限値に関して network-adjusted time ＋ 2 時間未満であれば受け入れられることなどを考慮して設定しなければならない．

実際には，少額取引をメインとするペイメントチャネルに対して，膨大なハッシュパワーやノード数を必要とするこの攻撃が実現する可能性は極めて低い．しかし，攻撃モデルが示したように，network-adjusted time やブロックタイムスタンプはトランザクション置換の失敗を引き起こす程度の誤差を許容している．ユーザーはこれらを考慮して，時刻管理にどの時間パラメータを使用するのか，タイムロックの間隔 ΔT をどのような値にするのか決定するべきである．

4.2　変動する手数料に対する脆弱性

4.2.1　概要

[20][13] は，コミットメントトランザクションを作成してからそれが有効になるまで時間差があるという特徴を持つ．非協力的にペイメントチャネルを終了する場合，最新のコミットメントトランザクションがブロードキャストされ，ブロックに追加されることが期待される．しかし，コミットメントトランザクション作成後にビットコインネットワークが混雑し手数料相場が上昇すると，相対的に低い手数料を持つ最新のコミットメントトランザクション (以下，TX_n) はマイナーに優先的に選択されず，ブロックに追加されない可能性がある．

ビットコイントランザクションの手数料の推移を図 4.2 に示す．トランザクション手数料の相場の変動が極めて激しいことが読み取れる．図 4.2 において，各色はトランザクションが n ブロック以内にマイニングされるために必要な手数料を表している．手数料が高いトランザクションほど優先度が高いので，1 ブロック以内にマイニングされるために必要な手数料 (青) は，6 ブロック以内にマイニングされるために必要な手数料 (橙) よりも高いことが分かる．

ブロックに追加されないままタイムロックの間隔 ΔT 経過すると，2 番目に新しいコミットメントトランザクション (以下，TX_{n-1}) も有効になる．TX_{n-1} がブロックに追加されると，TX_n での資金移動はブロックチェーンに反映されず，TX_n で資金を受け取るユーザーは取引相手に資金が盗まれる事態が発生する．一般的に，TX_{n-1} に TX_n

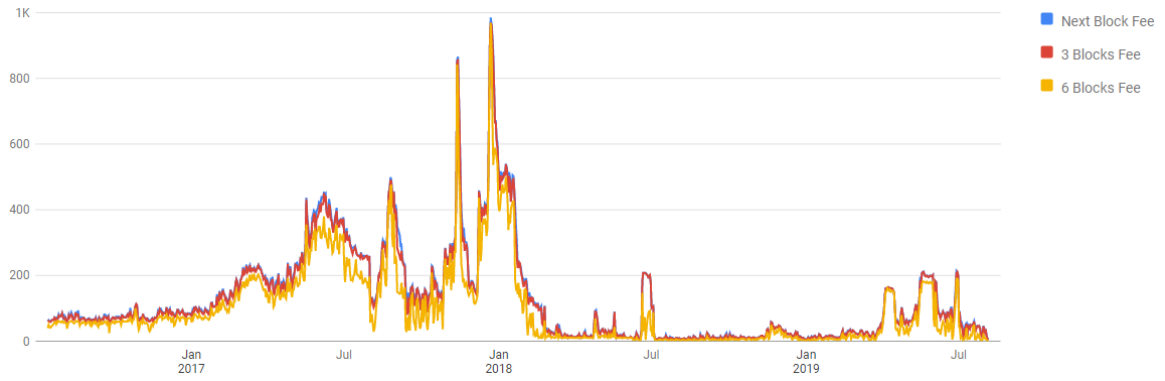

図 4.2: n ブロック以内にマイニングされるためのトランザクション手数料．x 軸は日付，y 軸はトランザクション手数料 (satoshis/byte) を表している [5]．

よりも高い手数料が設定されていない限り，メモリプールにおいて TX_n は TX_{n-1} に置換されない [16]．しかし，TX_{n-1} は確かに有効な状態であり，TX_n で送金するユーザーが意図的にマイニングを行い TX_{n-1} をブロックに追加することも考えられるので，一定のリスクは残る．

通常，手数料が不足してトランザクションが承認されない場合は，手数料を高めに変更したトランザクションを作り直せばよい．しかし，コミットメントトランザクションは 2-of-2 トランザクションを参照しているため，非協力的なペイメントチャネル終了時において，ユーザーは単独でトランザクションを作り直すことができない．

4.2.2　手数料追加の解決策

非協力的なペイメントチャネル終了時にコミットメントトランザクションを作り直すことはできないが，一方のユーザーが単独でこれに手数料を追加することはできる．一方のユーザーが単独で手数料を追加する手法として，以下の 2 つが挙げられる．

- Chid Pays for Parent(CPFP)：　CPFP[9] とは，なかなか承認されないトランザクションの優先度を高める手法である．未承認のトランザクションである親トランザクションに対して，親トランザクションのアウトプットを参照する子トランザクションを作成しこれに高い手数料を設定する．マイナーは高い手数料が設定された子トランザクションを承認したいと考えるが，これを承認するためには親トランザクションも承認する必要がある．結果的に，親トランザクションは優先度が高くなり承認されやすくなる．
  コミットメントトランザクションのアウトプットを参照するトランザクションを作成し，これに高い手数料を設定することで，コミットメントトランザクションの優先度を高めることが可能になる．

- SIGHASH_ANYONECANPAY：　SIGHASH_ANYONECANPAY とは，インプットの追加を許可する SIGHASH である．

SIGHASH_ALL|SIGHASH_ANYONECANPAY のように，他の SIGHASH を組み合わて使用される．コミットメントトランザクションを SIGHASH_ANYONECANPAY を使用して署名することで，ユーザーは単独で手数料を追加することが可能になる．

上記の手法を適用することでTX_nがブロックに追加されない問題を解決することができるが，非協力的なケースにより，一方のユーザーが全ての追加手数料を負担しなければならない．ここで新たに，手数料の公平性の問題が生じる．

4.2.3　手数料の公平性の問題

前章にて，トランザクションの優先度を高めるために手数料を追加する手法を述べたが，手数料を追加する人はその手数料分，資金を余分に支払わなければいけないという問題が発生する．手数料を追加する人とは，TX_{n-1}がブロックに追加されることを阻止したい人，即ちTX_nで資金を受け取るユーザー（受取人）である．ここでは，コミットメントトランザクションで交換する資金は追加手数料よりも高く，受取人はTX_nをブロックに追加する動機が十分にあると仮定する．

非協力的にペイメントチャネルを終了する場合，TX_nで資金を送るユーザー（送金人）は追加手数料を支払う必要がない．TX_n作成後に手数料が高騰し，セトルメントトランザクションを作成する際に高額な手数料が必要な場合，送金人は非協力的にチャネルを終了する方が総手数料を安く抑えられることがある．例として，ファンディングトランザクション，キックオフトランザクション，TX_nに手数料X satoshis が設定され，その後手数料相場が高騰しトランザクションに手数料$10X$ satoshis が必要になった状況を考える．なお，各トランザクションに設定する手数料はユーザー間で折半するものとする．この時点でチャネルを終了する場合，各ユーザーが支払う総手数料は表 4.2 のようになる．TX_n有効時に手数料相場が下がれば問題ないが，$10X$ satoshis

表 4.2: 各ユーザーが支払う総手数料

	送金人	受取人
協力時	$(X+10X)/2$	$(X+10X)/2$
非協力時	$(X+X+X)/2$	$(X+X+X)/2$ + 追加手数料

のままだとTX_nへの追加手数料は$9X$ satoshis となる．この時，送金人は手数料を節約できる一方で，受取人は多くの手数料負担を強いられることになり，手数料の公平性の問題が生じる．

このような手数料の不公平性は，送金人にチャネルを非協力的に終了する動機を与える可能性がある．手数料相場が高騰した際，送金人は非協力的にチャネルを終了することで自身が負担する手数料を安く抑えることができる．このように，手数料変動を考慮すると，ユーザー間で安定的にチャネルを維持することが難しくなると考えられる．

Chapter 5　提案手法

4.2 章にて，最新のコミットメントトランザクションに手数料を追加することで過去のコミットメントトランザクションが有効になること防ぐことができるが，新たに追加手数料の公平性の問題が生じることを述べた．本章では，この公平性の問題を解決するために，追加手数料をユーザー間で折半することを可能にするプロトコルを提案する．

5.1　トイモデル

プロトコルを説明する前に，トイモデルとして，1 つのケーキをアリスとボブで公平に分ける (切る) 方法を考える．両者はより大きく切られたケーキを欲しているものとする．ケーキを公平に分けるには，アリス (またはボブ) がケーキを 2 つに切り，ボブ (またはアリス) がどちらかの切られたケーキを選択すれば良い．切られたケーキの選択権はボブにあるため，アリスはより大きく切られたケーキを得るためにケーキを均等に分けるよう動機付けされている．

このトイモデルでは，一方のユーザーが単独で，1 つのケーキを均等に分割している．我々が提案するプロトコルは，このモデルをペイメントチャネルに適用する．

5.2　公平な手数料追加プロトコルの概要

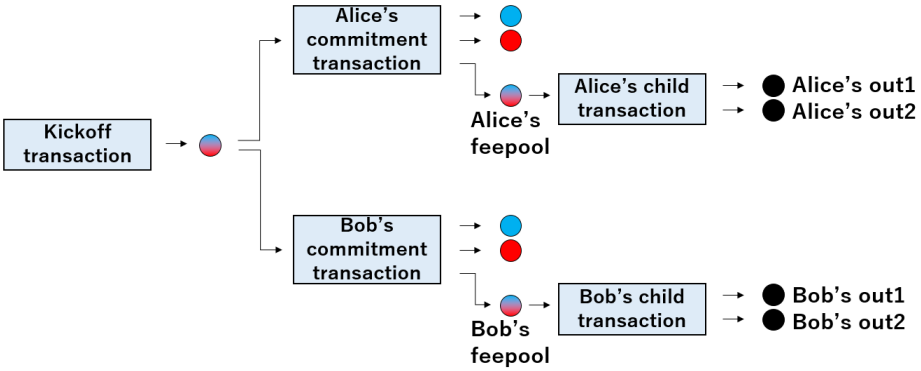

図 5.1: 公平な手数料追加プロトコルの概要図

本プロトコルの概要図を図 5.1 に示す．本プロトコルでは，CPFP を使用して手数料を追加する．子トランザクションはコミットメントトランザクション作成時に同時に作成する．コミットメントトランザクションは各ユーザーへの支払いを行うアウト

プットに加え，追加手数料用の資金を保持するアウトプット (feepool) も保持する．追加手数料は feepool を参照する子トランザクションにて支払われる．feepool の資金はチャネル開設時に 2 者からデポジットされ，追加手数料を引いた余った資金は子トランザクションのアウトプット (out1, out2) で 2 者に返却される．また，コミットメントトランザクションのブロードキャスターを特定するために，各ユーザーは異なる feepool のアンロック条件を持つコミットメントトランザクションをそれぞれ保持する．本プロトコルを実現する上で必要となる，子トランザクションに関する条件を以下に示す．

1. 手数料を追加する時に，out1 と out2 の額を決定できる．
2. feepool が必ず out1 と out2 に使用される．

1 に関して，将来の手数料相場が予測できないため，最新のコミットメントトランザクションが有効になった時点で手数料を追加したい．これは非協力的なチャネル終了時を想定しているため，out1 と out2 の額は一方のユーザーが単独で設定できる必要がある．また，2 に関して，一方のユーザーが feepool の資金を持ち逃げする事態を防ぐため，feepool の資金は必ず out1 と out2 に使用されることが要求される．

1 の条件を満たすには feepool の設定にシングルシグを使用すればよいが，2 の条件が満たされない．また，2 の条件を満たすには 2-of-2 マルチシグを使用すればよいが，反対に 1 の条件が満たされなくなる．本稿では，1 と 2 の条件を同時に満たすために，新しい SIGHASH である SIGHASH_WITHOUT_OUTPUT_VALUE を導入することを提案する．

5.2.1　SIGHASH_WITHOUT_OUTPUT_VALUE

SIGHASH_WITHOUT_OUTPUT_VALUE とは，アウトプットの額を署名範囲から除外する SIGHASH である．[1] にて提案されたが，具体的なユースケースについては言及されなかった．SIGHASH_WITHOUT_OUTPUT_VALUE を導入することで，上記の問題を解決することができる．厳密には，[1] の提案は SIGHASH の設計に変更を加えているため，SIGHASH_WITHOUT_OUTPUT_VALUE をそのまま現在のビットコインに導入できるわけではないが，ここでは実装の詳細は無視する．

feepool には 2-of-2 マルチシグを使用するが，各ユーザーは取引相手の feepool に対して SIGHASH_WITHOUT_OUTPUT_VALUE を使用して署名を行う．署名を受け取ったユーザーは，後にアウトプットの額を変更することが可能になる．また，アウトプットの額以外に関しては 2 者で合意しているため，アウトプットの宛先を変更することはできない．

5.3　各アウトプットのスクリプト

追加手数料をユーザー間で均等に負担するには，out1 と out2 に等しい額が設定されており，それぞれが 2 者に返却されれば良い．トイモデルで説明したように，片方のユーザーが out1 と out2 の額を決定し，もう片方のユーザーが out1 と out2 のどちらか

を選択することができれば，追加手数料をユーザー間で均等に負担することができる．本章では，feepool，out1，out2 の各アウトプットのスクリプトに関して述べる．各アウトプットは，Hashed time-locked contract のように，シークレットとタイムロックを使用して使用条件が制御されている．

アリスとボブのシークレットをそれぞれ S_a，S_b とおき，署名をそれぞれ $Sig_a(SIGHASH)$，$Sig_b(SIGHASH)$ とおく．例えば，$Sig_a(ALL)$ の場合，アリスが SIGHASH_ALL を使用して施した署名を指す．また，out1 と out2 のスクリプトに関して，num1 は num2 より大きな値である．本章では具体例として，num1 を 48 時間，num2 を 24 時間とする．ここでは，アリスがコミットメントトランザクションに手数料を追加するケースを考える．ボブが手数料を追加するケースでは，後述する条件に関してアリスとボブの立場を対照的に考えればよい．アリスは，アリスのコミットメントトランザクションとアリスの子トランザクションをブロードキャストする．

5.3.1　Alice's feepool

Alice's feepool の redeem script を以下に示す．

redeem script 5.1: Alice's feepool

```
OP_HASH160 <digest Sa> OP_EQUALVERIFY 2
<Alice's PubKey> <Bob's PubKey> 2 OP_CHECKMULTISIG
```

アンロック条件は以下の通りである．

- $Sig_a(ALL)$，$Sig_b(WITHOUT\_OUTPUT\_VALUE)$，$S_a$

5.3.2　Alice's out1

Alice's out1 の redeem script を以下に示す．

redeem script 5.2: Alice's out1

```
OP_HASH160 <digest Sa> OP_EQUAL
OP_IF
    OP_HASH160 <digest Sb> OP_EQUALVERIFY
    <Alice's PubKey>
OP_ELSE
    <num1> OP_CSV OP_DROP <Bob's PubKey>
OP_ENDIF
OP_CHECKSIG
```

アンロック条件は以下の通りである．

- (アリスへ)：$Sig_a(ALL)$，S_b，S_a

または

- (ボブへ)：48 時間後[1]，$Sig_b(ALL)$

[1]子トランザクションが承認されてからの相対時間．タイムロックには OP_CSV[12] を使用する．

5.3.3　Alice's out2

Alice's out2 の redeem script を以下に示す.

redeem script 5.3: Alice's out2

```
OP_HASH160 <digest Sa> OP_EQUAL
OP_IF
    OP_HASH160 <digest Sb> OP_EQUALVERIFY
    <Bob's PubKey>
OP_ELSE
    <num2> OP_CSV OP_DROP <Alice's PubKey>
OP_ENDIF
OP_CHECKSIG
```

アンロック条件は以下の通りである.

- (アリスへ) : 24 時間後[1]，$Sig_a(ALL)$

または

- (ボブへ) : $Sig_b(ALL)$，S_b，S_a

5.3.4　タイムロックとシークレットによる制御

アリスが子トランザクションをブロードキャストすると，S_a が公開される．S_a を入手したボブは，out1 と out2 共にアンロックすることが可能になる．ボブは out1 と out2 どちらを選択するか，24 時間以内に決定しなければならない．これは，24 時間後にアリスが out2 を使用することが可能になるからである.

ボブが out1 を使用する場合は，48 時間待機し $Sig_b(ALL)$ を使用して out1 をアンロックする．24 時間後，アリスは out2 を使用できるようになる．ボブが out2 を使用する場合は，24 時間以内に S_a，S_b，$Sig_b(ALL)$ を使用して out2 をアンロックする．この時 S_b が公開されるので，S_b を入手したアリスは out1 を使用できるようになる．なお，アリスは 48 時間以内に out1 を使用しなければならない.

このように，ボブが out1 を選択するとアリスは out2 を選択できるようになり，ボブが out2 を選択するとアリスは out1 を選択できるようになる．また，タイムロックとシークレットによる制御により，各ユーザーは out1 と out2 両方を選択することができない．out1 と out2 の選択権がボブにあることから，アリスは out1 と out2 の額を均等に設定する動機付けがされている.

5.4　子トランザクションの手数料

各ユーザーは自由に子トランザクションの手数料を設定することができるが，ユーザーが自身の利益を最大化するために行動すると仮定すると，子トランザクションには必要最低限の手数料が設定されると考えられる．また，TX_n で資金を送るユーザー (送

金人) が，TX_n がブロックに追加されることを妨害するため子トランザクションに相場よりも低い手数料を設定しブロードキャストすることも考えられるが，その場合は TX_n で資金を受け取るユーザー (受取人) が適切な手数料を設定した子トランザクションをブロードキャストすればよい．メモリプールにおいて，前者のトランザクションは，より高い手数料持つ後者のトランザクションに置換される [16].

Chapter 6　評価

5 章で提案したプロトコルを導入することで，追加手数料をユーザーが間で折半することが可能になるが，同時にトランザクションサイズも増加する．トランザクションサイズの増加は手数料の増加を意味する．厳密には，Segwit 対応のトランザクションは，トランザクションサイズではなく仮想サイズ[1]によって手数料が決定される．本章では仮想サイズに着目し，プロトコルの導入前後において，4.2 章で言及した最新のコミットメントトランザクション TX_n で資金を受け取るユーザー (受取人) が負担する手数料がどのように変化するか評価する．

6.1　受取人が負担する手数料

本章では，アリスを受取人と想定する．プロトコルを使用しない場合，アリスは追加手数料を全額負担する．手数料を追加する手法として，CPFP と SIGHASH_ANYONECANPAY が考えられるが，本章では柔軟に手数料を設定できる CPFP を使用するケースを考える．これは，SIGHASH_ANYONECANPAY を使用して手数料を追加する場合，お釣り用のアウトプットの追加ができないため，アリスは追加資金を過不足なく保持する UTXO を必要とするからである．

プロトコル導入前後に関して，それぞれコミットメントトランザクションと子トランザクションを作成し，仮想サイズを調査した．プロトコル導入前の CPFP を使用するケースに関して，トランザクションの構成を以下に示す．括弧内は使用するスクリプトを表している．

- コミットメントトランザクション
  - input: キックオフトランザクションのアウトプット (P2WSH)
  - output1:　ボブへの支払い (P2WPKH)
  - output2:　アリスへの支払い (P2WPKH)
- 子トランザクション
  - input: コミットメントトランザクションの output2
  - output:　アリスへの支払い (P2WPKH)

プロトコル導入後のトランザクションの構成に関しては，図 5.1 を参照されたい．なお，feepool，out1，out2 は P2WSH を使用する．署名サイズは DER エンコーディン

[1] トランザクションウェイトを 4 で割ったもの

グ [2] 後 70 バイト，SIGHASH フラグを加えて 71 バイトで統一している．シークレットのサイズは 32 バイトとした．それぞれのトランザクションの各要素のサイズは以下の通りである．なお，表 6.2，表 6.5 における OP_0 は，OP_CHECKMULTISIG の実装のバグを補うためのものである．OP_CHECKMULTISIG は実行時に，処理に関係のないスタック上の要素を 1 つ余分にポップするバグがあるので，それの調整のために OP_0 が必要になっている [8]．

表 6.1: コミットメントトランザクションの各要素のサイズ

フィールド	サイズ (byte)	
	プロトコル導入前	プロトコル導入後
version	4	4
marker	1	1
flag	1	1
input counter	1	1
inputs	41	41
output counter	1	1
outputs	62	105
witness	218	218
locktime	4	4
合計	333	376

表 6.2: コミットメントトランザクションの witness の各要素のサイズ

フィールド	サイズ (byte)
witness counter	1
OP_0	1
signature1 size	1
signature1	71
signature2 size	1
signature2	71
redeem script size	1
redeem script	71
合計	218

表 6.3: 子トランザクションの各要素のサイズ

フィールド	サイズ (byte)	
	プロトコル導入前	プロトコル導入後
version	4	4
marker	1	1
flag	1	1
input counter	1	1
inputs	41	41
output counter	1	1
outputs	31	86
witness	107	274
locktime	4	4
合計	191	413

表 6.4: プロトコル導入前の子トランザクションの witness の各要素のサイズ

フィールド	サイズ (byte)
witness counter	1
signature size	1
signature	71
pubkey hash size	1
pubkey hash	33
合計	107

表 6.5: プロトコル導入後の子トランザクションの witness の各要素のサイズ

フィールド	サイズ (byte)
witness counter	1
OP_0	1
signature1 size	1
signature1	71
signature2 size	1
signature2	71
secret size	1
secret	32
redeem script size	1
redeem script	94
合計	274

表 6.1 に関して，プロトコル導入により P2WSH のアウトプット (feepool) が 1 つ増えるので，43 バイト増えることが読み取れる．また，表 6.3 に関して，プロトコル導入前後で保持するアウトプットが P2WPKH * 1 つから P2WSH * 2 つに変化するので，$43 + (43 - 21) = 55$ バイト増加することが読み取れる．2.3.3 章で言及したウェイトの算出式から，各トランザクションの仮想サイズは以下の通りである．

表 6.6: 各トランザクションの仮想サイズ (vbyte)

	プロトコル使用	プロトコル未使用
コミットメントトランザクション	211	168
子トランザクション	206	110
合計	417	278

プロトコルの導入後，コミットメントトランザクションと子トランザクションの仮想サイズは増えるが，これらの手数料はユーザー間で折半することができる．コミットメントトランザクション作成時に設定された手数料を x satoshis/vbyte，有効時の手数料相場を nx satoshis/vbyte とすると，アリスが負担する総手数料は以下のようになる．なお，コミットメントトランザクションの手数料はユーザー間で折半し，追加手数料はコミットメントトランザクション有効時の相場を適用するものとする．

- プロトコル使用：$\frac{417nx}{2}$ satoshis
- プロトコル未使用：$(278n - \frac{168}{2})x$ satoshis

$n > 1.21$ で，プロトコル使用時の方がアリスが負担する総手数料が安くなることが分かる．即ち，コミットメントトランザクション作成時の手数料と比較して，有効時の手数料相場が 1.21 倍以上になれば，アリスはプロトコルを導入することで負担する手数料を安く抑えることができる．4.2 章で言及した TX_n が優先的にブロックに追加されない問題は，手数料相場が大きく上昇したときに発生するので，追加手数料が必要な場合において，アリスはプロトコルを導入することで自身が負担する手数料を安く抑えることができる．

また，各ユーザーは資金交換の際に，プロトコルを導入したコミットメントトランザクションと，導入しないコミットメントトランザクションの 2 つを同時に作成することも可能である．手数料相場が上昇した場合は前者のコミットメントトランザクションをブロードキャストし，上昇しなかった場合は後者のコミットメントトランザクションをブロードキャストすればよい．

6.2　out1/out2 を参照するトランザクション

out1/out2 を参照するトランザクションは，witness に out1/out2 のアンロック条件と redeem script が含まれるため，通常のトランザクションに比べ仮想サイズは大きくなる．しかし，out1 と out2 の額が等しく設定され，ユーザーが witness のサイズを小さ

くしたいと考えているのであれば，ユーザーはシークレットを使用しないアンロック条件を持つアウトプットを選択する可能性が高い．例えばアリスが手数料を追加する場合，ボブは Alice's out1 を選択する高い．手数料相場が高騰している時に提案したプロトコルが使用されると仮定すると，手数料相場が高騰している時に (タイムロックの設定時間より前に)out1/out2 を使用する必要がないので，これらを参照するトランザクションの手数料は大きな負担にならないと考えられる．

Chapter 7　結論

本稿はタイムロックを使用してトランザクション置換を行うペイメントチャネルに着目し，これは時間パラメータの許容誤差や手数料変動に対して脆弱であることを指摘した．そして，手数料変動に対応するために，非協力的なペイメントチャネル終了時の公平な手数料追加プロトコルを提案した．

タイムロックを使用したトランザクション置換は，時間パラメータを基準にトランザクションの管理を行っている．しかし，Network-adjusted time は最大で 70 分ノードのシステム時間との誤差を許容しており，ブロックタイムスタンプは上限値に関して network-adjuseted time + 2 時間未満であれば有効と見なされる．これらの許容誤差を考慮すると，[20] で述べられたタイムロックの間隔の最小値は 1 時間であるという主張は，システム正常の範囲内でトランザクション置換が失敗する可能性があるため不十分である．さらに，コミットメントトランザクションはタイムロックの影響により，作成されてから有効になるまで時間差があるという特徴を持つ．コミットメントトランザクション作成後に手数料相場が上昇すると，相対的に低い手数料を持つコミットメントトランザクションは優先的にブロックに追加されない．ブロックに追加されないままタイムロックの間隔経過すると，過去のコミットメントトランザクションも有効になり，タイムロックを使用したトランザクション置換が失敗する可能性がある．本稿ではこれらの問題提起を行い，解決策を提案した．前者の問題は median time past の値を監視することやタイムロックをブロック高で設定すること，後者の問題は SIGHASH_ANYONECANPAY や CPFP などを使用して，最新のコミットメントトランザクションに手数料を追加することで解決することができる．

手数料追加の解決策によりトランザクション置換を正常に行うことが可能になるが，追加手数料は片方のユーザーが単独で負担しなければならない．本稿ではこれを解決するために，非協力的なペイメントチャネル終了時に，追加手数料をユーザー間で折半することを可能にするプロトコルを提案した．本プロトコルは CPFP を使用するが，ファンディングトランザクション作成時に追加手数料用の資金を余分に投入し，追加手数料を引いた余った資金を子トランザクションにて各ユーザーに返却する．返却される 2 つの資金に関して，片方のユーザーが額を設定し，もう一方のユーザーがそのどちらかを選択することで，非協力的な状況下でも追加手数料をユーザー間で折半することが可能となる．本プロトコルを導入すると，ユーザーは非協力的にペイメントチャネルを終了する動機がなくなり，結果として，手数料変動に関わらずユーザー間でペイメントチャネルを安定的に維持することが可能となる．

ペイメントチャネルはセカンドレイヤの技術であるが，セカンドレイヤだけで完結する技術ではなく，ブロックチェーンやビットコインネットワークの状態も考慮しなければならない．また，信頼していないユーザーとのチャネル構築も目指していることから，手数料などのインセンティブも考慮して設計されるべきである．

謝辞

本研究を行うにあたり，熱心にご指導いただいた生産技術研究所の松浦幹太教授に深く感謝いたします．また，研究室での活動において研究のアドバイスをくださった今井先生，Mihaljevic 先生，細井さん，大畑さん，小林さん，角田さん，宮前さん， 碓井さん，林田さん， Phalakarn さん， 黄さん，石井さん， 宮里さん，石さん，石坂さん，Piriou さん，その他関係者の皆様に感謝いたします．そして，大学院での生活をサポートしていただいた秘書の仲野さん，鶴山さん，小倉さんに感謝いたします．

分散台帳とその応用技術特別研究会 (RC-94) では，研究に関する多くの知識を得ることができました．参加企業様をはじめとする全ての関係者に感謝いたします．最後に，大学院生活を応援してくれた両親に感謝します．

参考文献

[1] Build your own nHashType. https://github.com/scmorse/bitcoin-misc.

[2] BIP66: Strict DER signatures, 2015. https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki.

[3] Alex Boverman. Timejacking & Bitcoin, 2011. http://culubas.blogspot.com/2011/05/timejacking-bitcoin_802.html.

[4] Gavin Andresen. BIP101: Increase maximum block size, 2015. https://github.com/bitcoin/bips/blob/master/bip-0101.mediawiki.

[5] billfodl.com. Bitcoin Transaction Fees. https://billfodl.com/pages/bitcoinfees.

[6] Bitcoin Wiki. Block timestamp. https://en.bitcoin.it/wiki/Block_timestamp.

[7] Bitcoin Wiki. Hash Time Locked Contracts. https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts.

[8] Bitcoin Wiki. Script. https://en.bitcoin.it/wiki/Script.

[9] Bitcoin.org. Child Pays For Parent. https://bitcoin.org/en/glossary/cpfp.

[10] Bitcoin.org. Signature Hash Types. https://bitcoin.org/en/transactions-guide#signature-hash-types.

[11] blockchain.com. Mempool Transaction Count. https://www.blockchain.com/charts/mempool-count.

[12] BtcDark, Mark Friedenbach, and Eric Lombrozo. CHECKSEQUENCEVERIFY, 2015. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki.

[13] Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of Bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.

[14] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[15] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. On Scaling Decentralized Blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[16] David A. Harding and Peter Todd. BIP125: Opt-in Full Replace-by-Fee Signaling, 2015. `https://github.com/bitcoin/bips/blob/master/bip-0125.mediawiki`.

[17] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A simple layer2 protocol for bitcoin. *White paper: https://blockstream. com/eltoo. pdf*, 2018.

[18] Christian Decker and Roger Wattenhofer. Information Propagation in the Bitcoin Network. In *IEEE P2P 2013 Proceedings*, pages 1–10. IEEE, 2013.

[19] Christian Decker and Roger Wattenhofer. Bitcoin Transaction Malleability and MtGox. In *European Symposium on Research in Computer Security*, pages 313–326. Springer, 2014.

[20] Christian Decker and Roger Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[21] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-NG: A Scalable Blockchain Protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, 2016.

[22] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[23] Mark Friedenbach, BtcDrak, Nicolas Dorier, and kinoshitajona. BIP68: Relative lock-time using consensus-enforced sequence number, 2015. `https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki`.

[24] Jeff Garzik. BIP102: Block size increase to 2MB, 2015. `https://github.com/bitcoin/bips/blob/master/bip-0102.mediawiki`.

[25] Johannes Göbel and Anthony E Krzesinski. Increased block size and bitcoin blockchain dynamics. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. IEEE, 2017.

[26] Maurice Herlihy. Atomic Cross-Chain Swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254, 2018.

[27] Visa Inc. Visa Inc. at a Glance. https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf.

[28] Intel Corporation. Intel Software Guard Extension. https://software.intel.com/sgx.

[29] Japan Blockchain Association. ブロックチェーンの定義, 2016. https://jba-web.jp/archives/2011003blockchain_definition.

[30] Thomas Kerin and Mark Friedenbach. BIP113: Median time-past as endpoint for lock-time calculations, 2015. https://github.com/bitcoin/bips/blob/master/bip-0113.mediawiki.

[31] David Koops. Predicting the confirmation time of bitcoin transactions. *arXiv preprint arXiv:1809.10596*, 2018.

[32] Joshua Lind, Ittay Eyal, Florian Kelbert, Oded Naor, Peter Pietzuch, and Emin Gun Sirer. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454*, 2017.

[33] Joshua Lind, Ittay Eyal, Peter Pietzuch, and Emin Gün Sirer. Teechan: Payment Channels Using Trusted Execution Environments. *arXiv preprint arXiv:1612.07766*, 2016.

[34] Eric Lombrozo, Johnson Lau, and Pieter Wuille. BIP141: Segregated Witness, 2015. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[35] Lombrozo, Eric and Wuille, Pieter. BIP144: Segregated Witness (Peer Services), 2016. https://github.com/bitcoin/bips/blob/master/bip-0144.mediawiki.

[36] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008.

[37] Alejandro Ranchal Pedrosa, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Scalable lightning factories for bitcoin. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 302–309, 2019.

[38] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. *https://lightning.network/lightning-network-paper.pdf*, 2016.

[39] Rusty Russell. Reahing The Ground With Lightning (draft0.2), 2015. https://github.com/ElementsProject/lightning/blob/master/doc/deployable-lightning.pdf.

[40] Peter Todd. OP_CHECKLOCKTIMEVERIFY, 2014. https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki.

[41] Pieter Wuille. BIP103: Block size following technological growth, 2015. https://github.com/bitcoin/bips/blob/master/bip-0103.mediawiki.

発表文献

国際会議

i Takahiro Nagamine and Kanta Matsuura. A New Protocol for Fair Addition of a Transaction Fee When Closing a Payment Channel Uncooperatively. Financial Cryptography and Data Security, Poster Session, 2020.

国内会議

ii 長嶺隆寛，松浦幹太．ビットコインにおける手数料を考慮したオフチェーントランザクションの管理．コンピュータセキュリティシンポジウム (CSS2019) 論文集，pages 562-568，2019.

iii 長嶺隆寛，松浦幹太．非協力的なペイメントチャネル終了時の公平な手数料追加プロトコル．暗号と情報セキュリティシンポジウム (SCIS2020) 論文集，2020.

国内研究会

iv 長嶺隆寛，松浦幹太．ビットコインにおける手数料を考慮したオフチェーントランザクションの管理．分散台帳とその応用技術特別研究会，2019.