Tapyrus Technical Overview

Shigeyuki Azuchi
chaintope, Fukuoka, Japan

Draft

Abstract

2009 年の Bitcoin の登場から、分散環境をベースにした台帳技術が数多く登場した。これらは従来のシステムのような中央の管理者の代わりに暗号学的な仮定に基づいて安全な支払いを可能にする。また、Ethereum のように VM 上で任意のロジックを実行し、その結果を共有する台帳技術も登場し、単純な支払いシステムだけでなく従来の IT システムに代わり企業を横断したビジネス利用のプラットフォームとして注目を集めている。特にエンタープライズ向けのブロックチェーンとしては、Hyperledger Fabric などの参加ノードを限定する許可型のブロックチェーンを利用した多くの PoC が行われている。Bitcoin や Ethereum のような完全にオープンなブロックチェーン技術をエンタープライズ用途で使用するにあたっては、ファイナリティの欠如や、ベースの通貨のボラティリティの高さ、チェーンの運用や追加機能に関するガバナンスの欠如といった課題がある。一方、ネットワークや検証に参加するノードの母数が限定されている許可型のブロックチェーンにおいては、台帳データの正真性の担保も限られた参加者内に限定され、利用者が直接ブロックチェーンのコンセンサスに参加することはなく API などを経由する集中型のシステムになりがちで、RDB など現在の IT システムによって代替可能なものが多い。
我々の提供する Tapyrus は、ブロックチェーンのオープン性を維持しながら、エンタープライズ利用における課題を解消するハイブリット型のブロックチェーンである。ブロックを生成（トランザクションを承認）しチェーンの維持・運営を行うフェデレーション層と、台帳データにオープンにアクセス可能な台帳ネットワークを分離することで、これらの課題を解消する。本文書では、Tapyrus の構成と提供する機能および、Tapyrus 上で構成可能な Layer2 プロトコルについて説明する。

Keywords: Blockchain; DLT.

1 はじめに

2009 年の Bitcoin [1] の登場から、分散環境をベースにした台帳技術が数多く登場した。従来の決済システムは、管理者が運用する集中型のサーバーシステムが取引の承認を行うのに対して、Bitcoin ではそのような集中型のサーバーは存在せず、誰もが参加可能なオープンなネットワークで、取引の承認が行われる。Bitcoin 以前はこのような分散ネットワークで一意の取引を承認するのは困難であるとされてきたが、Bitcoin はこの問題を Proof of Work(PoW) と

呼ばれる計算競争とその結果得られる報酬により解決した。この Bitcoin の発明以降、単純な支払い用途以外にも、任意のトークンの発行や転々流通を可能にするトークン・プロトコルや、VM 上で任意のロジックを実行可能なスマートコントラクトを提供するブロックチェーンプロダクトが多数登場した。このようなブロックチェーン技術は、暗号通貨に限らず広く利用できると考えられ、エンタープライズ用途でのブロックチェーン適用が模索された。エンタープライズ領域でこれらのブロックチェーンを利用するにあたっては次の課題がある。

ファイナリティの欠如　Bitcoin の PoW コンセンサスにはファイナリティが存在しない。トランザクションがブロックに格納されたとしても、チェーンの再編成により後からそのブロックが別のブロックに置き換わることがあり、単にブロックにトランザクションが格納されただけでは、それが確実に処理されたと保証することはできない。ただし、トランザクションが格納されるブロックに後続のブロックが連結するにつれ、再編成が発生する確率は小さくなる。そのため確率的ファイナリティと呼ばれる。また、PoW アルゴリズムを採用していてもハッシュパワーが低く、PoW アルゴリズムが簡単にハッシュパワーを移行可能なチェーンの場合、大規模なチェーンの再編成を行う攻撃が行われる可能性もある [2]。

ボラティリティの高さ　パブリックなブロックチェーンでは、トランザクションの実行にそれぞれ手数料を必要とする。これはオープンなネットワークにおいてトランザクションを処理する際に手数料を設けない場合、トランザクションの実行には参加ノードのコストがかかるのに対し、トランザクションの発行コストは実質 0 であり、大量のトランザクションをネットワークに投入することで、ブロックチェーンネットワークをダウンさせるような Denial-of-service(DoS) 攻撃が可能になる。このためパブリックなブロックチェーンのトランザクションには手数料が必要となるが、その手数料はそのチェーンのベースの通貨で支払われる。しかし暗号通貨は、現状ボラティリティの変動が大きく、エンタープライズ用途で使用する際に、実際に必要となる実コストが読めない。また人気のスマートコントラクトの登場などの外的要因によりトランザクションがネットワークに急増し、ブロックに格納するためのトランザクション手数料が高騰する。

チェーンのガバナンス　管理主体の存在しないブロックチェーンにおいては、コンセンサスに影響する機能拡張や致命的な欠陥が発生した場合の対応に関する合意形成が難しい。Bitcoin では、2017 年に機能追加のソフトフォークを巡ってマイナーと開発者の対立が発生し、ソフトフォークのアクティベーションまで長い時間がかかり、その後の機能追加にも慎重になり

新しい機能（Taproot の導入）に 4 年かかった。また、その間提案されている機能もまだデプロイされる状況にはない。Ethereum では 2016 年にスマートコントラクトのバグを利用した攻撃で大量の資金が盗難にあう The DAO 事件が発生した際、その救済のためにハードフォークが行われた。スマートコントラクトのバグによる障害は他にも存在する中、これだけを救済するのは問題とされ、結果的に反対派はチェーンを分岐し Ethereum Classic という新たなチェーンが誕生することになった。このように管理者不在のチェーンにおいては、チェーンの運営に関するガバナンスが課題になる。

その後、エンタープライズ用途でこのような課題を回避する、参加者を限定した許可型のブロックチェーン Hyperledger Fabric [3] が登場する。コンセンサスアルゴリズムとして Practical Byzantine Fault Tolerance(PBFT) を採用しており、これは合意に至るために参加者の母数が決定している必要があるアルゴリズムで、誰もが自由に参加可能な PoW のようなアルゴリズムとは異なる。許可型のブロックチェーンは、参加可能なノードが限定されており、一般の利用者は基本的にノード運営者が提供する API などを介してサービスを利用する。適切な利用者か、必要な権限を持っているかどうかは API 層で任意に実装されるため、こちらで DoS 攻撃への対策が取られ、ブロックチェーン層でのトランザクション手数料は不要になる。一方、台帳のデータの正真性の証明のスコープも参加ノードに限定される。API ベースのアクセスは、利用者から見るとそのバックエンドがブロックチェーンであるか RDB であるかは直接関係のないことであるため、従来の IT システムで代替可能なことが多く、またその方が従来の IT システムの知見が利用でき、コストも抑えられる傾向にある。

1.1 我々の貢献

我々は、ブロックチェーンはそのオープンな台帳アクセスが重要な要素であると考え、ブロックチェーン層を、チェーンの維持・管理を行うフェデレーションによりブロック生成を行う Signer Network 層と、誰もが自由にノードとして参加可能な台帳ネットワーク層に分離することで、ブロックチェーンのオープン性を維持しながら、エンタープライズ利用における課題を解消するハイブリッド型のブロックチェーン Tapyrus を提案する。

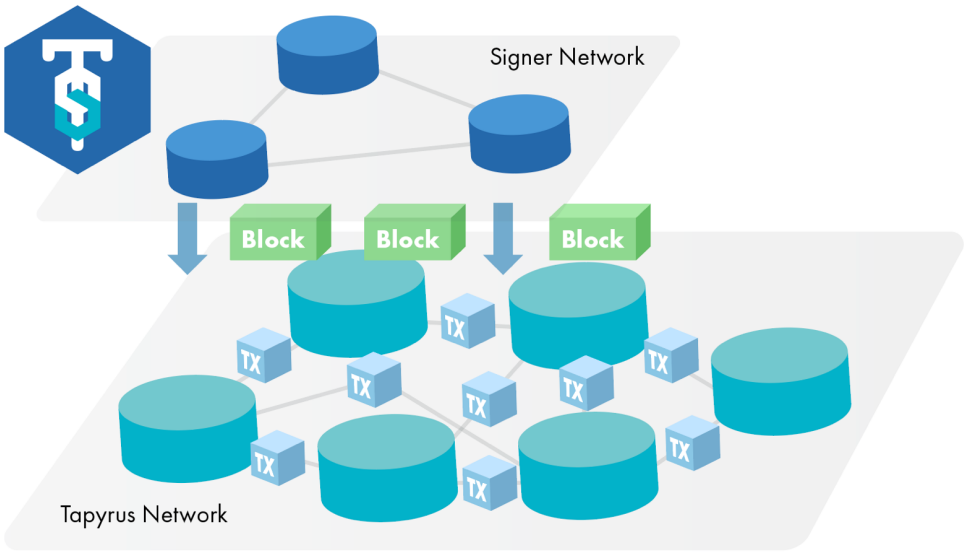

Figure 1: Tapyrus のネットワーク構成

2　Tapyrus の構成

Tapyrus のブロックチェーンは、任意のビジネスユースケース毎（電力証書の売買や CO2 取引、地域通貨、企業コインなど）に独立したネットワークを構成する。各ネットワークは Figure 1 に示すように、主にそのビジネスユースケースの利害関係者で構成されるフェデレーションにより運営される Signer Network と、オープンな台帳ネットワークである Tapyrus Network で構成される。

ファイナリティ　Bitcoin のような PoW では誰もがマイナーになることができるが、Tapyrus ではチェーンのセットアップ時にブロックを作成できる Signer（ノード）が限定される。チェーンの維持・運用を行う定められた Signer が新しいブロックに対してデジタル署名を作成し、ブロックに添付することで Tapyrus Network 上で有効なブロックが作成される。このため PoW のような確率的なファイナリティではなく、ブロックが作成された段階でトランザクションにファイナリティが与えられる。

2.1　ブロック構造

Tapyrus は Bitcoin のブロックチェーンをベースにコンセンサスの変更および機能追加をしている。大きな変更点としてブロック生成のルールを PoW から複数の Signer によるデジタル署名に置き換えており、それに伴い Tapyrus のブロックヘッダーは Table 1 の要素で構成されている。この内、*proof* フィールドに Signer が協力して作成したデジタル署名（Schnorr 署名）がセットされる。

フィールド	型	サイズ	定義
features	int32	4	機能フラグ。現在は 1。
hasPrevBlock	char[32]	32	前のブロックのハッシュ
hashMerkleRoot	char[32]	32	ブロック内の全トランザクションのハッシュで構成されるマークルツリーのルートハッシュ
hashImMerkleRoot	char[32]	32	ブロック内の全トランザクションの TXID で構成されるマークルツリーのルートハッシュ
time	uint32	4	ブロックが作成された時刻の UNIX タイムスタンプ
xfieldType	uint8	1	xfield のタイプ（後述）
xfield	N/A	可変	拡張フィールド。
proof	char[65]	65	ブロックのデジタル署名（Schnorr 署名データとデータ長をプレフィックスとする）

Table 1: Tapyrus のブロックヘッダーの構造

値	名称	型	サイズ	定義
0x00	None	N/A	1	None。xfield は未使用。
0x01	集約公開鍵	char[33]	34	Tapyrus のブロックの検証に使用する公開鍵

Table 2: Tapyrus のブロックヘッダーの構造

現在定義されている *xfieldType* は Table 2 のとおり。

2.2 Tapyrus の Schnorr 署名方式

ブロックに対する署名や、トランザクション内で使用される署名に使用される Scuhorr 署名方式は次の通り。

署名の作成　Tapyrus で使用する楕円曲線は secp256k1 を使用し、位数 n の楕円曲線のジェネレーターを $\mathbb{G}$、暗号学的ハッシュ関数を H とする。

署名の作成は、入力として以下のアイテムを取る。

- 32 バイトの秘密鍵 sk のバイト列

- 32 バイトの署名対象のメッセージ m のバイト列

秘密鍵 sk を使用してメッセージ m に対する署名は以下の手順で作成される。

1. $d' = int(sk)$ とする。
2. $d = 0$ もしくは $d => n$ の場合、失敗する。ここで n は楕円曲線の位数。
3. $P = d'\mathbb{G}$ とする。
4. ランダムな nonce k' を生成し、$R = k'\mathbb{G}$ とする。
5. $jacobi(R.y) = 1$ の場合 $k = k'$ とし、それ以外の場合 $k = n - k'$ とする。
6. R の x 座標を r とする。
7. $e = int(H(bytes(r)||bytes(P)||m)) \mod n$ とする。
8. $s = (k + ed) \mod n$ を計算する。
9. 署名データは (r, s)。

署名の検証　署名の検証は、入力として以下のアイテムを取る。

- 秘密鍵 sk に対応する公開鍵 $P = sk\mathbb{G}$
- 32 バイトの署名対象のメッセージ m のバイト列
- 署名データ (r, s)

これらのデータを使用して、以下の手順で検証を行う。

1. 公開鍵 P が曲線上に存在しないか、無限遠点を指す場合、検証は失敗。
2. r が楕円曲線のフィールドサイズより大きい場合、検証は失敗。
3. s が楕円曲線の位数より大きい場合、検証は失敗。
4. $e = int(H(bytes(r)||bytes(P)||m)) \mod n$ を計算する。
5. $R' = s\mathbb{G} - e\mathbb{P}$ を計算する。
6. R' の x 座標が r と等しい場合、署名検証は成功する。

2.3 Signer Network

Signer Network は、ネットワークのセットアップ時に予め定められたフェデレーションのメンバーによって運営される、ブロックチェーンを維持・運用するためのネットワークである。具体的には、Tapyrus Network で発生した新しいトランザクションを収集、検証し、それらの有効なトランザクションを含む新しいブロックを作成し、Tapyrus ネットワークにそのブロックをブロードキャストする。

2.3.1 ブロックの検証

Signer Network が作成したブロックが有効なブロックかどうかは、ブロックの *proof* が有効な署名か検証することで確認できる。この検証に用いるデータは次の通り。

- ブロックの *proof* にセットされている署名データ (r, s)
- Table 1 のブロックヘッダーから *proof* を除外したデータの double-SHA256 ハッシュ値を署名対象のメッセージ m とする
- 集約公開鍵

この内、集約公開鍵はジェネシスブロックのブロックヘッダーの *xfield* に格納されている値を使用する。この集約公開鍵は、全 Signer の公開鍵を加算し、集約したものである。もし、その後の後続ブロックで、*xfieldType = 0x01* を使用して新しい集約公開鍵が設定されている場合は、以降のブロックはその新しい集約公開鍵を使って署名検証する。これらはいずれも Tapyrus Network ですべての参加者が入手可能なデータであるため、ネットワークの全参加者がブロックの正しさを独立して検証することが可能である。

2.3.2 検証可能な閾値署名方式

Tapyrus Signer は複数の Signer が協力してブロックの署名を作成する。上述した署名方式では簡易的に表現したため集約公開鍵 P に対応する集約秘密鍵 sk が登場したが、集約秘密鍵自体はすべての Signer の秘密鍵を加算したデータであり、この値を計算するためにはすべての Signer が秘密鍵を公開する必要があり、望ましくない。そこで Tapyrus の Signer Network では、 Stinson と Strobl (2001) による検証可能で安全な分散 Schnorr 署名と (t, n) 閾値署名方式 [6] を基に、各 Signer の秘密鍵を明かすことなく、つまり集約秘密鍵 sk を作ることな

く、集約公開鍵に対して有効な Schnorr 署名を作成する。この方式は内部で Feldman 方式の Verifiable Secret Sharing(VSS)[7] を使用し、Pedersen の検証可能な秘密分散法と Schnorr 署名を組み合わせ、n 人の Signer の内、閾値 t 個の部分署名を集めると有効な Schnorr 署名を完成させることができる閾値署名を提供する。すべての Signer の部分署名を収集する必要がなく t 個集めれば良いため、閾値を満たす範囲内の部分的な Signer の故障に対して堅牢である。

閾値は任意に設定することができるが、母数に対して過半数を超える値、もしくはビザンチン耐性を考慮した値を設定することが望ましい。これらの署名方式によって生成される Schnorr 署名は、ネットワークの参加者から見ると単一の Schnorr 署名であるため、Signer の母数が増減しても、各参加者の署名検証コストは一定である。一方、署名生成においては、各 Signer が生成した部分署名の交換が必要になるため、Signer の母数が増えるとこのメッセージ交換コストも比例して増加する。

2.3.3 Signer Network の処理フロー

各 Signer ノードは、それぞれが保持する公開鍵によって識別される。またこの公開鍵データの辞書順で各ノードにはインデックスが割り当てられ、ラウンドマスター（2.3.3 参照）は、このインデックスを基に決定される。

新しいブロックの作成は、各 Signer から選出されたそのラウンドのマスターとその他のメンバーが協調して行い、そのラウンドで閾値を満たす Signer の合意が得られればブロックに対して有効な *proof* が生成され、作成されたブロックが Tapyrus Network にブロードキャストされる。その後、次のブロック生成ラウンドが始まり、ブロック作成が続く。以下に、具体的に Signer ノード間で交換されるメッセージとアルゴリズムについて示す。

メッセージタイプ　各ノード間の通信は、Redis pub/sub にブロードキャストされたメッセージを使って行われる。現在定義されているメッセージタイプは Table 3 のとおり。

ブロック作成ラウンドのプロセス　Signer がメッセージを交換し、１つのブロックを作成するまでの期間をラウンドと呼ぶ。各ラウンド毎に Signer の中からラウンドマスターが選択される。このラウンドマスターは、ラウンドロビンで選択される。ブロック作成ラウンドにおけるラウンドマスターと各 Signer の通信フローを Figure 2 示す。

この Signer の処理を Rust で実装したのが Tapyrus Signer[8] である。

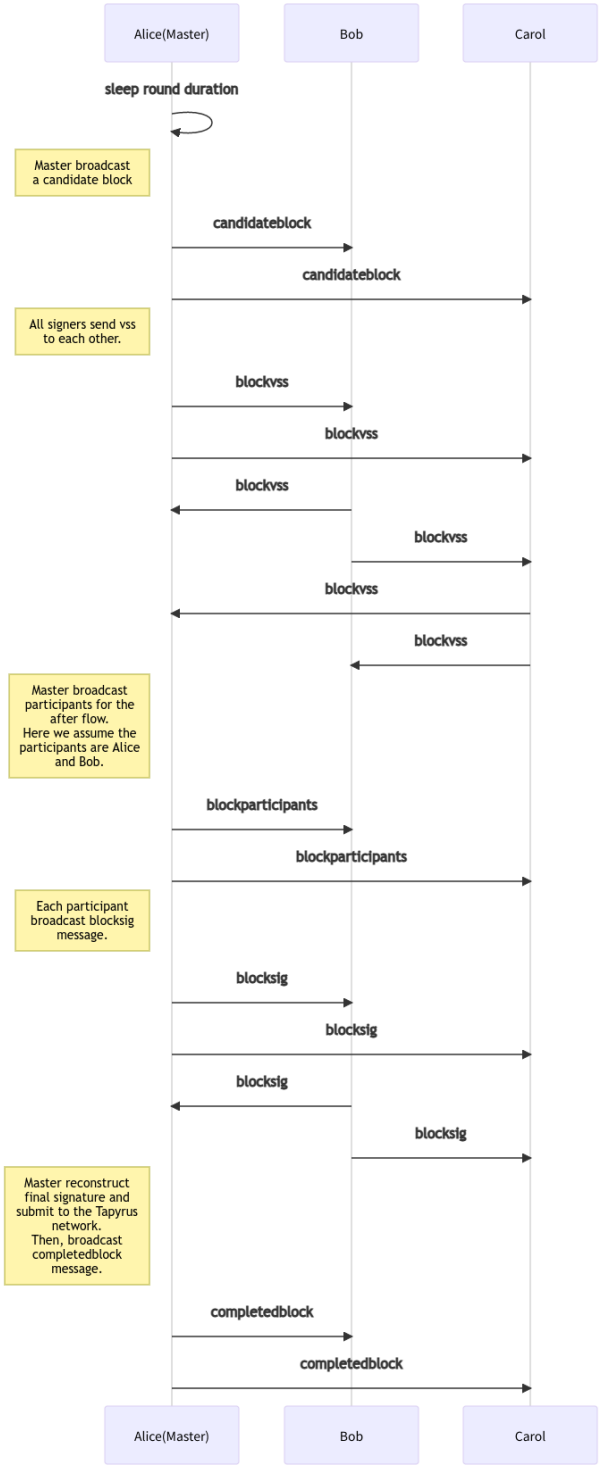

Figure 2: 署名生成ラウンドの処理シーケンス

メッセージ	ペイロード	定義
candidateblock	Block	ラウンドマスターが候補ブロックを送信
blockvss	BlockVSS	ランダムシークレットの VSS を送信
blockparticipants	Vec< $PublicKey$ >	ラウンドマスターによる署名プロトコルの通知
blocksig	LocalSig	ローカルの部分署名の送信
completedblock	Block	ラウンドマスターが完成したブロックを送信

Table 3: Signer 間で通信されるメッセージタイプ

2.4 Tapyurs Network

Tapyrus Network は誰もが参加可能なブロックチェーンネットワークで、ジェネシスブロックから最新のブロックまでブロックチェーンのすべてのオンチェーンデータの提供、新規トランザクションの発行、伝播をサポートする。Tapyrus のフルノード Tapyrus Core[9] は、Bitcoin の参照実装である Bitcoin Core[10] をフォークして実装されている。

Bitcoin ベースのブロックチェーンとしたのは、以下の理由による:

- ブロックサイズの制限やブロック内のトランザクションで使用される高コストの opcode の最大値などの制約により、データの増加スピードやスクリプトの検証コストが予測可能である。これは安定してシステムを運用する上で重要な要素となる。
- 2009 年の稼働からさまざまな攻撃にさらされながら止まることなく運用されてきた P2P コード。コンセンサスの仕様ではないが、P2P メッセージを交換してブロックチェーンを構築するコード実装には、攻撃に対するさまざまな措置が実装されている。

2.4.1 Tapyrus Core

Tapyrus は基本的に Bitcoin のトランザクション [4] や、Script [5] システムをベースにいくつか改良を加えている。

コンセンサスアルゴリズムの変更　PoW と最長チェーンルールのコンセンサスアルゴリズムを、Signer Network により生成されるデジタル署名をベースにしたコンセンサスアルゴリズムへ変更。Bitcoin のブロック生成間隔は難易度調整アルゴリズムにより約 10 分になるよう調整されているが、Tapyrus のブロック生成間隔は、Signer が任意に調整可能である。

TXID の malleability の修正　トランザクションには、そのトランザクションをブロックチェーンで一意に識別するための TXID と呼ばれる識別子を持つ。この識別子は明示的にトランザクションに設定される値ではなく、トランザクションデータの double-SHA256 ハッシュを計算し、それをリトルエンディアンで表現した値である。そのため、トランザクションデータの一部でも変更されると、TXID も変化する。通常、トランザクションのデータはデジタル署名により保護されているため、トランザクションデータの変更は行えないが、唯一、デジタル署名のデータだけは署名できないため、このデータは変更可能である。署名データを変更すると通常は無効なデジタル署名になるが、署名アルゴリズムとして採用されている ECDSA 自体の malleability[1]や署名をセットする際の opcode の指定方法を変えることで、デジタル署名は有効なままそのデータの表現を変えることで、TXID を変更することが可能であった。Bitcoin ではこのため、チェーン上にブロードキャストしたトランザクションの TXID が意図せず悪意あるユーザーによって変更される問題が存在した。単純なオンチェーン支払いであれば大きな問題ではないが、Lightning Network[11] のような Layer 2 プロトコルでは、親子関係のあるトランザクションを構築し、それぞれに事前署名する必要があるため、TXID の malleability の修正が必要で、Bitcoin では 2017 年にアクティベートされた Segregated Witness でその対応を行った。Tapyrus では、トランザクションハッシュと TXID を分離し、TXID の計算に署名データを含めないことで Bitcoin に元々存在するこの脆弱性を修正している。

Schnorr 署名のサポート　ブロックの検証に加え、トランザクションの Script システムで使用する署名検証アルゴリズムに ECDSA に加え、Schnorr 署名をサポートしている。具体的には、以下の 4 つの opcode で ECDSA と Schnorr 署名が評価される。

- OP_CHECKSIG
- OP_CHECKSIGVERIFY
- OP_CHECKDATASIG
- OP_CHECKDATASIGVERIFY

どちらの署名かは提供される署名データのサイズで決まる。Script で Schnorr 署名が利用可能になると、署名データ自体を小さくでき、コインの使用に複数人のユーザーの署名を必

[1]ECDSA 署名値 (r, s) は、$(r, -s \mod n)$ としても有効な署名データである。

要とするマルチシグの署名データを単一の署名データに集約したり、Script を使用せずにデジタル署名自体で条件付きのコントラクトの実装を可能にする Scriptless Script[12] のような技術がより簡単に導入できるといったメリットがある。また、Schnorr 署名は正式な安全性評価があり、ECDSA のような署名の malleability も存在しない。

オラクルのサポート　ブロックチェーン上で実行されるスマートコントラクトは、基本的にサンドボックス環境で実行されるため、外部のデータに依存したコントラクトの実行はできない。これはそれらのデータが正しいデータか、ブロックチェーンのコンセンサスで検証する術がないためである。これに対し、データの提供者という信頼点（オラクルと呼ぶ）を設けることで、外部データに依存したコントラクトを実現することが可能である。ブロックチェーン上のコントラクトでは、あくまでオラクルによって提供されたデータであることを検証するだけで、データの正しさ自体を検証するものではない。Tapyrus では、このオラクルが利用できるよう、次の 2 つの opcode を追加している。

- OP_CHECKDATASIG
- OP_CHECKDATASIGVERIFY

これらは、任意のメッセージに対する署名検証を行う opcode である。これらの opcode は、以下の 3 つの要素を入力として取る。

1. 公開鍵
2. 任意のメッセージ
3. 署名（ECDSA もしくは Schnorr）

続いて、2 のメッセージの SHA-256 ハッシュ値を計算し、それをメッセージダイジェストとして、3 の署名が 1 の公開鍵に対して有効なデジタル署名かを検証する。オラクルを利用したコントラクトを実行する場合、1 の公開鍵はオラクルの公開鍵で、2 のメッセージはオラクルが公開したコントラクトの条件に使用するデータで、3 の署名がオラクルによって生成された署名になる。両 opcode の参照実装を Listing 1 に示す。

Listing 1: OP_CHECKDATASIG と OP_CHECKDATASIGVERIFY の参照実装

```
1 case OP_CHECKDATASIG:
2 case OP_CHECKDATASIGVERIFY: {
3     // (sig message pubkey -- bool)
```

```
4       if (stack.size() < 3) {
5           return set_error(
6               serror, SCRIPT_ERR_INVALID_STACK_OPERATION);
7       }
8
9       valtype &vchSig = stacktop(-3);
10      valtype &vchMessage = stacktop(-2);
11      valtype &vchPubKey = stacktop(-1);
12
13      //check signature encoding without hashtype byte
14      if ( (vchSig.size() != CPubKey::COMPACT_SIGNATURE_SIZE - 1
15           && !CheckECDSASignatureEncoding(vchSig, serror, true))
16          || !CheckPubKeyEncoding(vchPubKey, flags, sigversion, serror)) {
17          // serror is set
18          return false;
19      }
20
21      bool fSuccess = false;
22      if (vchSig.size()) {
23          valtype vchHash(32);
24          //Hash message
25          CSHA256().Write(vchMessage.data(), vchMessage.size())
26              .Finalize(vchHash.data());
27          //no hashtype in signature. call VerifySignature not CheckSig
28          fSuccess = checker.VerifySignature(vchSig, CPubKey(vchPubKey),
                uint256(vchHash));
29      }
30
31      if (!fSuccess && (flags & SCRIPT_VERIFY_NULLFAIL) && vchSig.size()) {
32          return set_error(serror, SCRIPT_ERR_SIG_NULLFAIL);
33      }
34
35      popstack(stack);
36      popstack(stack);
37      popstack(stack);
38      stack.push_back(fSuccess ? vchTrue : vchFalse);
39      if (opcode == OP_CHECKDATASIGVERIFY) {
40          if (fSuccess) {
41              popstack(stack);
42          } else {
43              return set_error(serror, SCRIPT_ERR_CHECKDATASIGVERIFY);
44          }
45      }
46  } break;
```

OP_CHECKDATASIG と OP_CHECKDATASIGVERIFY の違いは、OP_CHECKSIG と OP_CHECKSIGVERIFY の違いと同様で、結果をスタックにプッシュするか、失敗しにすぐに Script を失敗と評価するかである。

トークン　ブロックチェーンのユースケースにおいては、任意のトークンの発行や、転々流通はよく知られる機能である。Bitcoin では 80 バイトの任意のデータを記録可能な OP_RETUN opcode を利用して早くから Open Assets Protocol[13] や、Counterparty Protocol[14] など多くのオーバーレイプロトコルが開発された。特に Ethereum では ERC20 Token のシェアが大きい。

多くのプロトコルがオーバーレイの Layer2 プロトコルであるのに対し、Tapyrus ではネイティブトークン（TPC/tapyrus）に加え Layer 1 で任意のトークンの発行、転送をサポートする。オーバーレイプロトコルは任意のデータをブロックチェーンに記録する機能を利用して、ネイティブトークンとは別の価値移転を可能にする Layer2 プロトコルだが、それらは Layer 1 のコンセンサスとして正しさを検証されておらず、Client Side Validation と呼ばれるのそのプロトコルを独自に解釈するウォレットによってトークンを認識している。つまり、それらを解釈しないウォレットにデータが渡ると、それらはトークンではなく、Layer 1 のネイティブトークンとして解釈される。また、これらのトークンの正しさを検証するのに、トークンが発行された起点のトランザクションから、すべての転送トランザクションが必要となり、既に使用済みの TXO（Transaction Output）をずっと保持しなければならない。これはノードの使用済みの TXO データを削除するプルーニング機能が利用できないことを意味し、フルノードのディスクサイズを圧迫する。また履歴トランザクションが必要なことから、スマートフォンや IoT デバイスなどの軽量デバイスでの検証にも不向きである。Tapyrus では Layer 1 の機能としてトークン機能をサポートすることで、使用済み TXO のプルーニングや軽量デバイスでのトークンの識別を容易にする。Tapyrus で発行可能なトークンのタイプは以下の 3 種類である：

- 再発行可能なトークン
- 再発行不可能なトークン
- NFT（Non-Fungible Token）

トークン仕様　Tapyrus Script には、Script でトークンを識別するための OP_COLOR opcode が追加されており、その仕様を以下に示す。

OP_COLOR opcode が Script 内に現れると、スタックの一番上の要素が COLOR 識別子として解釈される。スタックに 1 つも要素が残っていない場合、もしくは COLOR 識別子が後述するルールに従っていない場合、Script は失敗する。 COLOR 識別子がルールに準拠す

る場合、そのUTXOのコインはそのCOLOR識別子のコインの量を表す。OP/_COLORの含まれないScriptはネイティブトークンTPC（tapyrus）の量を指す。

トランザクションにおいて、インプットのカラー識別子が持つトークンの総量と、アウトプットのカラー識別子のトークンの総量は基本的に一致する（トークンの焼却を除いて)。つまりトランザクション内の各トークンのバランスを維持する。尚、現時点ではSignerがブロック生成の手数料としてトークンを受け取ることはないものとする。

Script内でOP_COLORを使用する場合、以下の制約が適用される。

- Script内に含められるOP_COLORの数は１つのみである。
- OP_COLORはOP_IFなどの制御opcodeの分岐内に記述することはできない。
- 上記のカウントと制約のため、各アウトプットのカスタムトークンのscriptPubkeyには必ずOP_COLORが含まれる。 このため、P2SHのredeem scriptにOP_COLORを含めることはできない。そのようなスクリプトを構成した場合、 スクリプトインタプリタのスタックに複数のOP_COLORが含まれることになり、P2SHを使用する際に必ずエラーとなりコインの喪失に繋がる。

結果、既存のP2PKHとP2SHをカラーリングした以下のタイプのscriptPubkeyをサポートする。

- CP2PKH(Colored P2PKH)：
  $< COLOR\ identifier > OP\_COLOR\ OP\_DUP\ OP\_HASH160 < H(pubkey) > OP\_EQUALVERIFY\ OP\_CHECKSIG$
- CP2SH(Colored P2SH)：
  $< COLOR\, identifier > OP\_COLOR\, OP\_HASH160 < H(redeemscript) > OP\_EQUAL$

上記のように、トークンのUTXOには必ずそのトークンのCOLOR識別子が含まれているため、これによりUTXO単体でトークンの識別が可能になる。これにより、ブロックチェーンの使用済みデータのプルーニングを行っても未使用のトークンは確実に保持され、軽量ノードも受信したUTXOのみでそのトークンを認識することが可能になる。

COLOR識別子は1バイトのTYPEと、32バイトのPAYLOADで構成される。COLOR識別子が現在サポートするタイプとペイロードをTable 4に示す。トークンの発行トランザクションでは、アウトプットのOP_COLORが指定するCOLOR識別子が、インプットのデー

タに対してこのルールを満たしているか検証する。 さらにタイプ 0xC3 の場合、追加で、発行量が 1 であることを検証しなければならない。

タイプ	定義	ペイロード
0xC1	再発行可能なトークン	発行インプットの scriptPubkey の SHA256 値である 32 バイトのデータ
0xC2	再発行不可能なトークン	発行インプットの OutPoint の SHA256 値である 32 バイトのデータ
0xC3	NFT	発行インプットの OutPoint の SHA256 値である 32 バイトのデータ

Table 4: COLOR 識別子のタイプ

上記のルールにより、再発行不可能なトークンおよび NFT は、その COLOR 識別子が発行トランザクションのインプットが参照する OutPoint から生成されるため、同じ COLOR 識別子のトークンは二度と発行できないことが保証される。

トークンの発行トランザクション　トークンを新規に発行する場合、トークン発行用の UTXO をインプットにセットし、その UTXO から上記のルールをベースに COLOR 識別子を導出する。その COLOR 識別子と OP_COLOR opcode を使用した scriptPubkey(CP2PKH, CP2SH, etc) をトランザクションアウトプットにセットした発行トランザクションを作成する。

この時、新規トークン発行のアウトプットには任意のトークンの量を value をセットできる。また、インプットで指定した UTXO の TPC は別の非トークンアウトプットを作成し、回収する。

トークンの転送トランザクション　トークンを送付する場合は、トークンを持つ UTXO をインプットにしたトランザクションを作成し、送金先のアドレスに対して、インプットのトークンと同じ COLOR 識別子と OP_COLOR opcode を付与したアウトプットを追加する。この時、インプットの数やアウトプットの数、トークンの種類は複数設定できるが、インプットとアウトプットでトークンの種類毎の総量が保持されている必要がある。

トークンの焼却　トークンを焼却する場合は、焼却するトークンを持つ UTXO と手数料用の TPC を持つ UTXO をインプットにしたトランザクションを作成し、手数料を差し引いた TPC のお釣りを受け取るアウトプットを追加する。トークンの UTXO の value 値は全てトー

クンの量であるため、 手数料を設定するためには必ず、TPC の UTXO をセットする必要がある。

また、上記３つのトークン処理を１つのトランザクションを組み合わせることは可能である。

各組み合わせおよび有効/無効のパターンについて、以下の資料に掲載する:

https://docs.google.com/spreadsheets/d/1hYEe5YVz5NiMzBD2cTYLWdOEPVUkTmVIRp8ytypdr2g/

3　拡張プロトコル

Tapyrus は Bitcoin や Ethereum、Fabric などと同じ Layer 1 のブロックチェーン実装である。シンプルな送金やトークンの交換などは Layer 1 のプロトコルで十分実現できるが、P2P ベースの分散ネットワークが土台の Layer 1 で高スループットな決済アプリケーションを実現する場合や、取引に秘匿性を持たせたいといった要求に対しては Layer 1 のプロトコルのみで実現するのは難しく、アプリケーションに応じて上位レイヤーのプロトコル設計などの拡張が必要となる。これは TCP/IP の上に HTTP がさらに TLS が拡張されたのと似ている。Tapyrus ではこれまでのユースケースや Paradium のようなプロダクトを実装するにあたって、必要に応じて拡張プロトコルを設計しており、本セクションではそれらのプロトコルの一部を紹介する。

3.1　トラッキングプロトコル

ブロックチェーンのユースケースの１つに、モノの移動履歴をトラッキングするサプライチェーン・トレーサビリティがある。誰もが自由にアクセス可能な台帳に、モノの移動を記録することで、企業や国をまたぐ移動の記録を誰もがいつでも自由に検証でき、オープンなブロックチェーンのユースケースの１つとして適している。一方、ブロックチェーンに記録可能なデータサイズは限定的で、データサイズを増やすことが可能だったとしても、分散ネットワークにおいてデータの増加は参加者のノード運用コストの上昇とトレードオフになる。このため、大量のモノの移動をブロックチェーンに記録する場合、データのトレースが証明可能な状態でいかコンパクトにするかが、ブロックチェーンのスループットやディスクスペースにおいて重要な課題である。本節では、これらの課題に対処する拡張プロトコルについて紹介する。

3.1.1 トラッキングトランザクション

UTXO 型のブロックチェーンである Tapyrus では、トランザクションのインプットで既存の有効な UTXO を消費して（送金元、送金の原資）、トランザクションのアウトプットで新しい UTXO を生成（送金先）する。トレーサビリティのようなユースケースでは、インプットをモノの移動元、アウトプットを移動先として表現することが可能である。

トラッキングペイロード　トラッキングトランザクションは、送信先のアウトプットに加えて、移動するモノの情報を記録するトラッキングペイロードを持つアウトプットを余分に 1 つ保持する。このアウトプットが存在することで、このトランザクションがトラッキングトランザクションであることを示す。トラッキングペイロードのアウトプットは OP_RETURN を使用した以下の形式になる。

$$OP\_RETURN < Tracking\ Payload >$$

トラッキングペイロードは Table 5 のデータで構成される。

フィールド	サイズ	内容
マーカー	2 バイト	常に 0x5450。トラッキングプロトコル= TP を表す Hex 値
バージョン	1 バイト	現在は 0x01
ペイロードサイズ	CompactSize [16]	ペイロードのサイズ
ペイロード	可変	移動するアイテムの一意な識別子で作成されたアキュムレーター値（3.1.4 参照）

Table 5: トラッキングペイロードのデータ

3.1.2 トレースルール

トラッキングプロトコルにおける、移動元と移動先は以下のルールにより決定する。

移動元　はトランザクションインプットで表現され、次のルールにより決定する。

- 移動を新規開始する場合

  トランザクションインプットの内、先頭のインプットを移動元のアドレスとする。

- 移動の途中である場合
  インプットが参照する UTXO の内、トラッキングペイロードが適用される UTXO をすべて移動元のアドレスとする。

移動先　はアウトプットの内、トラッキングペイロードのアウトプットの直後にセットされている UTXO を移動先のアドレスと規定する。なお、１つのトラッキングトランザクションに対して、複数の移動先を定義することを可能とする。これはトランザクションにトラッキングペイロードが複数定義できることを意味する。各ペイロードの適用先はその直後の UTXO とする。

3.1.3　ウォレットのサポート

本トラッキングプロトコルに対応するウォレットは、以下の機能を提供する必要がある。

- トラッキングトランザクションのパース
  トラッキングペイロードを含むトラッキングトランザクションを受信した場合、そのトランザクションをトラッキングトランザクションとしてパースすること。

- トラッキング UTXO の管理
  トラッキングペイロードが割り当てられている UTXO はトラッキング用の UTXO として管理する必要がある。これらの UTXO はモノの移動を行う際のインプットとして消費される。

3.1.4　アキュムレーターを使用したデータの圧縮

トラッキングペイロードには、移動するモノの識別子を単純に記録するのではなく、識別子を含むアキュムレーターの値が記録される。これは一意の識別子をそのまま登録すると、その識別子の数に比例してペイロードデータサイズが増えるのを避けるためである。アキュムレーターとは、任意の数のアイテムの追加、削除かつ、任意のアイテムがそのアイテムに含まれているか照会が可能だが、アキュムレーター内に追加されたアイテムのリストは取得できないという特性を持つデータアルゴリズムである。

このようなアルゴリズムには、ハッシュの二分木を利用するアルゴリズムや、RSA 暗号を利用するアルゴリズム、多項式コミットメントを利用するアルゴリズムなどさまざまなのも

がある。Tapyrus のトラッキングプロトコルでは、この内 RSA 暗号を利用した RSA アキュムレーターを使用する。これは RSA アキュムレーターの以下の特性を考慮した結果である。

- アキュムレーターにどれだけアイテムを追加してもアキュムレーターの値は一定である。
- アキュムレーターに対してデータを追加、削除するにあたってアイテムの順番を考慮しなくて良い。

3.1.5 RSA アキュムレーターの基本アルゴリズム

RSA アキュムレーターは次の手順で初期化する。

1. 巨大な素数 p, q をランダムに選択し、$N = p * q$ を計算する。この時 N のビット長は十分安全な値にする。現在の推奨値は 3072 ビット。
2. N を法とした群を形成し、その中からジェネレーター g を選択する。
3. $A_0 = g$ としアキュムレーターを初期化する。

初期化したアキュムレーター A_0 に対して、次の手順で ID=x のアイテムを追加することができる。

1. x のハッシュ値 $H_p(x)$ を計算する。この時 H_p は素数を出力するハッシュ関数である。
2. アキュムレーター A_0 に対し 1 で計算した値の冪剰余 $A_1 = A_0^{H_p(x)} \mod N$ を計算する。
3. A_1 がアイテム x を追加して更新されたアキュムレーター値となる。

上記の手順で、さまざまなアイテムをアキュムレーターに追加することができる。アキュムレーターの更新は N を法とした冪剰余の計算であるため、どれだけアイテムを追加しようともアキュムレーターの値 N を超えることはない。これが RSA アキュムレーターが一定値のデータになる理由である。

アイテムがアキュムレーター内に含まれているかどうかは、包含証明とアキュムレーター値から計算することが可能である。上記の例では、アキュムレーター A_1 に x が含まれていることの包含証明は A_0 である。包含証明 A_0 とアイテム x が提供された検証者はアキュムレーターに対して、$A_1 = A_0^{H_p(x)} \mod N$ が成立するか検証し、成功すれば x がアキュムレーター A_1 に含まれていることが証明される。包含証明はアキュムレーターに追加されたすべてのア

イテムの内、証明したいアイテムを除いたアイテムのハッシュ値の内積から計算することも可能である。

上記の RSA アキュムレーターの基本原理に加えて、アイテムがアキュムレーターに含まれていないことを証明する非包含証明や、スタンフォード大学の Dan Boneh および Benedikt B ¨ unz、Ben Fisch らによって紹介されたバッチ処理化 [17] が可能である。

トラッキングプロトコルでの利用　トラッキングプロトコルでは上記のように１回の移動で移動するモノを１つのアキュムレーターにセットし、トラッキングトランザクションに記録する。ブロックチェーン上に保持するデータは一定サイズのアキュムレーター値に圧縮され、包含証明のデータはオフチェーンで各当事者が分散管理することとなる。これにより移動の記録はブロックチェーン上にコミットされながら、必要なデータは各当事者に分散されるため、分散ネットワークのディスク負荷を利用者のノードに分散することができる。

3.2　クロスチェーンスワップ

WIP：別紙「Tapyrus 技術資料」参照。

3.3　Credential プロトコル

WIP：別紙「Tapyrus 技術資料」参照。

4　他のプロダクトとの比較

WIP：別紙「ブロックチェーン比較資料」参照。

5　関連プロダクト

Tapyrus では、Singer Network および Tapyrus Network に加えて、ブロックチェーンアプリケーションを構成する上で必要とされるソフトウェアをいくつか提供する。

Tapyrus Explorer　https://github.com/chaintope/tapyrus-explorer/
Tapyrus Network のブロックチェーンのデータを照会可能なブロックチェーン・エクスプローラ。

Esplora Tapyrus　https://github.com/chaintope/esplora-tapyrus
アドレスやトークン等をキーに、それらに関連するトランザクションや UTXO を高速に取得するためのインデックスサーバー。ブロックチェーン自体は記録されるデータの検索には向かず、高速にデータを検索するためには、それらを検索するためのインデックスを作成する必要があるが、これらはインデックスの作成およびそのデータスペースと検索スピードのトレードオフ関係にある。Tapyrus Core では最低限 TXID をキーとしたインデックスは保持するが、アドレスやトークンをキーにしたデータ取得を高速にするためには別途インデックス化が必要で、Esplora Tapyrus はそのインデックスサーバーである。Tapyrus Explorer のバックエンドでもある。

tapyrusjs-lib　https://github.com/chaintope/tapyrusjs-lib
Tapyrus プロトコルに対応した JavaScript ライブラリ。

tapyrusrb　https://github.com/chaintope/tapyrusrb
Tapyrus プロトコルに対応した Ruby ライブラリ。

rust-tapyrus　https://github.com/chaintope/rust-tapyrus
Tapyrus プロトコルに対応した Rust ライブラリ。

tapyrusjs-wallet　https://github.com/chaintope/tapyrusjs-wallet
スマートフォンアプリ向けの JavaScript で記述された Tapyrus のウォレット実装。

[WIP]Glueby　https://github.com/chaintope/glueby
ブロックチェーンを利用したアプリケーション開発においては、鍵管理や、ステート（UTXO）のハンドリング、プライバシーレベルの設定、スケーリングといった課題に適切に対応する必要があり、必要に応じて Layer 2, 3 といった上位プロトコルの設計が必要になる。これらの設計には、ブロックチェーンのアーキテクチャや暗号技術に関する深い知識を必要とするケースが多く、迅速なアプリケーション開発における課題となる。Glueby は、Tapyrus を利用したブロックチェーンアプリケーションを極力ブロックチェーンに関する知識を必要とせずに素早く構築できるようこれらの課題のソリューションを内包し、ユースケースレベルの API をブロックチェーンアプリケーション開発者に提供する Ruby ライブラリである。

[WIP]Tapyrus API　https://github.com/chaintope/tapyrus-api (Private Access only)
Gluby を言語に依存することなく幅広いプラットフォームで利用可能にする REST API。利用者はブロックチェーンを意識することなく API コールのみでブロックチェーンアプリケーションの構築が可能。

rsa-accumulator　https://github.com/chaintope/rsa-accumulatorrb
Dan Boneh および Benedikt B ¨ unz、Ben Fisch らによるバッチ化をサポートする RSA アキュムレーター [17] の Ruby ライブラリ実装。

References

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” http://bitcoin.org/bitcoin.pdf, 2008.

[2] “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, 2018

[3] Hyperledger Fabric, https://www.hyperledger.org/use/fabric

[4] Bitcoin Transaction, https://en.bitcoin.it/wiki/Transaction

[5] Bitcoin Script, https://en.bitcoin.it/wiki/Script

[6] DR. Stinson and R. Strobl, “Provably Secure Distributed Schnorr Signatures and a (t, n) Threshold Scheme for Implicit Certificates http://cacr.uwaterloo.ca/techreports/2001/corr2001-13.ps, 2001

[7] “Verifiable secret sharing” https://en.wikipedia.org/wiki/Verifiable_secret_sharing https://ja.overleaf.com/project/606fb0ab21f88266

[8] Tapyrus Signer, https://github.com/chaintope/tapyrus-signer

[9] Tapyrus Core, https://github.com/azuchi/tapyrus-core

[10] Bitcoin Core, https://github.com/bitcoin/bitcoin/

[11] Thaddeus Dryja, Joseph Poon, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments"
d http://lightning.network/lightning-network-paper.pdf, 2016

[12] Andrew Poelstra, "Mimblewimble and scriptless scripts"
http://diyhpl.us/wiki/transcripts/realworldcrypto/2018/
mimblewimble-and-scriptless-scripts/, 2018

[13] Flavien Charlon, "Open Assets Protocol"
https://github.com/OpenAssets/open-assets-protocol, 2014

[14] Counterparty Protocol
https://github.com/CounterpartyXCP/Documentation/blob/
master/Developers/protocol_specification.md

[15] "EIP-20: ERC-20 Token Standard", https://eips.ethereum.org/EIPS/eip-20

[16] Compact Size, https://en.bitcoin.it/wiki/Protocol_documentation#Variable_length_integer

[17] Dan Boneh, Benedikt B ¨ unz, Ben Fisch, Stanford University
"Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains"
https://eprint.iacr.org/2018/1188.pdf