Anonify: A Blockchain-Agnostic Execution Environment with Privacy and Auditability

Version 1.0.0

須藤 欧佑 [1]　　恩田 壮恭 [1]　　中村 龍矢 [1]

[1]LayerX Inc.

June 4, 2020

ブロックチェーン技術は, 複数の企業や組織間で状態遷移の正しさを互いに検証し, 暗号学的に改ざん不可能な状態データとして記録, 共有することを可能にする. この特性を活かし, 参加者間の協業を促進する取り組みが活発に行われている. しかし, 異なる参加者間で状態データの共有を行う性質上, 本来は他参加者に公開するべきでないデータも公開してしまうプライバシ問題が大きな課題の一つとしてあげられる. ブロックチェーンにおけるプライバシ問題解決のための取り組みが多く行われている一方, これらの取り組みの多くは秘匿データに対する可用性問題, パフォーマンス課題, 状態遷移表現力の欠如, そして秘匿性と監査性のトレードオフなどの様々な技術的課題が存在する.

本稿では, これらの課題を解決するブロックチェーンのプライバシ保護システムである Anonify を提案する. Anonify は TEE (Trusted Execution Environment) を用いることにより, ブロックチェーンの参加者間で明らかにしたくないデータを保護しながら, 高い可用性と柔軟なビジネスロジックの実行を実現する. 加えて, 特定の主体がデータを読み取ることが可能となる監査機能も提供する.

また, Anonify 上のアプリケーション例として ERC20 規格同等のトークンの状態遷移ロジックを実装し, その評価を行った.

1　はじめに

ブロックチェーン技術はネットワークに参加するノード間で状態遷移ロジックを共有することを可能にする. しかし, ブロックチェーン上のデータは第三者が閲覧することが可能であり, 個人情報・プライバシ保護の課題が存在する. 各ユーザアカウントは固

有のアドレスで識別され, エンタープライズにおけるユースケースではこのアドレスと個人のアイデンティティは紐づいている, もしくは推測されやすい場合が多い. ブロックチェーンを閲覧できる主体にとっては, 特定のトランザクションがどのユーザから送信されたか, 何のデータをどのユーザに向けて送っているのか, という一連のプライバシ情報が明らかになってしまう.

しかしながら, 多くのブロックチェーンではプロトコルとしてデータの秘匿性や送受信者の匿名性は保証されていない. 例えば, 送金アプリケーションにおいて第三者はノードにアクセスすることで特定のトランザクションの送信者アドレス, 受信者アドレス, 送金額を自由に閲覧することが可能である. つまり, ブロックチェーンネットワークを複数企業で横断して構成する場合, 他社に自社の取引情報などがいつでも閲覧可能になってしまう.

このような問題に対処するために, ブロックチェーン上でのプライバシ保護に取り組む研究開発が多く行われている. 例えば, 機密性の高いデータはネットワーク全体にブロードキャストせず, 取引当事者ノードだけで共有するアプローチがある. このような手法は, Quorum[1]や Hyperledger Besu[2]の機能である Private (Group) Transaction, Corda[3] の Non-validating notary, Hyperledger Fabric[4]の Private Data といったエンタープライズ向けブロックチェーンの機能として実装されている. このアプローチでは, 秘匿したいデータをグローバルにアクセス可能なブロックチェーン上で直接扱わずに二者間あるいはグループ間のみで共有することで, 比較的シンプルにプライバシ保護を実現している.

一方で, このようなアプローチの場合, 秘匿化対象のデータを参加者が外部に漏洩することはプロトコルで防ぐことができず, それを検知することもできない. よって, プライバシ保護のためには, 他参加者がデータを漏洩しないことを信用する必要がある. また, トランザクションを共有されていない参加者は状態遷移を検証できないため, あるビジネスロジックの共同利用者間ではデータを隠すことはできない. 例えば, あるユーザが自分に送金されたトークンが正当であるか確かめるには, そのトークンに関する過去のトランザクションを全て検証する必要がある. しかしこのとき, そのトークンの過去の全ての取引に関して, 送受金したユーザのアドレスと送金額が閲覧できてしまう. 以上のように, エンタープライズ向けブロックチェーンに実装されているデータの共有相手を制限する手法は, 他参加者を信用せずにプライバシ保護することや, 利用者間でのプライバシ保護を実現することができない.

この問題に対処しているアプローチとして, 第三者には明らかにしたくない状態デー

[1]https://www.goquorum.com/
[2]https://www.hyperledger.org/projects/besu
[3]https://www.corda.net/
[4]https://www.hyperledger.org/projects/fabric

タもブロックチェーンでグローバルに共有する手法が存在する. ブロックチェーン上に全てのデータが記録されるため, ネットワーク全体で状態遷移の検証が行われ, 状態遷移の履歴としてデータがグローバルに記録されていく. この場合, 第三者が不正に閲覧できないように保護するために, 一般的に状態データは暗号化される. 加えて, 状態遷移が正しく行われたこと, つまり状態データの暗号文が正しく生成されていることを保証する性質も求められる. この性質は, (計算) 完全性と呼ばれる.

完全性をブロックチェーン上で効率的に検証できる主な手段として, 機密性の高いデータを明らかにせずそのデータが正当なことを暗号学的に保証するゼロ知識証明や, ハードウェアレベルのセキュリティ機能である TEE (Trusted Execution Environment) に基づく Attested Execution がある. ゼロ知識証明をベースにした手法については 第 7.1 節で述べるが, 送金以外の汎用的な状態遷移ロジックを実行するには証明生成時の計算オーバーヘッドが依然として大きく, 実際のアプリケーションへの応用は現実的ではないことが課題の一つとしてあげられる [1]. 具体的には, 証券決済における買い注文や配当のロジックのように単なる金額の移動だけではなく, 証券・投資家情報に基づくバリデーションや状態の書き換えなどがロジックに含まれる場合は, 純粋な暗号技術だけでは算術回路の性質上困難である.

そこで, 本稿ではこれらの課題を解決したブロックチェーン上のプライバシ保護システムである Anonify を提案する. Anonify は事前に定義された状態遷移ルールを TEE で実行し, その命令を暗号化しブロックチェーン上に記録する. これにより, プライバシを保護した上で幅広いビジネスロジックの実行を可能とする. 加えて, 特定の監査主体にのみに状態データを開示する仕組みも提供する. Anonify の実装は公開[5]されている.

Anonify でプライバシー保護の技術ベースとなっている TEE はプロセッサレベルのセキュリティ機能であり, TEE によりプライバシ保護と状態遷移ロジックの完全性を実現する. TEE の特徴的な機能として後述する Attestation によって TEE で実行するプログラムが正しいことをブロックチェーン上で検証することが可能になる. 状態遷移は特殊な算術回路で計算されず, 汎用プロセッサでプログラム処理されるので表現性の高い状態遷移をハイパフォーマンスで実行できる.

第 7.2 節で述べている TEE を用いてブロックチェーンのプライバシ保護を行う関連プロジェクトとの大きな違いとして, Anonify では特定の監査主体に監査機能を提供可能であることがあげられる. 多くのユースケースで, 全ての状態データを全ての参加者に対して秘匿化するのではなく, 特定の監査主体が参加者の状態データを閲覧可能であることが求められる. 例えば, 金融取引のそれぞれの取引情報は参加者に対して秘匿化しつつ, 当局には取引情報が開示されるようなケースがあげられる. このようなユースケース要件を満たすために, Anonify では状態データの秘匿化と監査性を両立できる設

[5] https://github.com/LayerXcom/anonify

計となっている.

本稿の構成を述べる. まず, 第 2 章で Anonify を構成する各技術的コンポーネントを紹介する. 次に, 第 3 章で Anonify の概略および満たすべき性質, 基本的なワークフローについて述べ, 第 4 章で各サブプロトコルの技術的な詳細を説明する. 第 5 章で Anonify の実装とパフォーマンス評価の結果について記載している. 第 6 章では, ここまでの章で定義された Anonify プロトコルに対する拡張提案を議論する. 第 7 章では Anonify の関連研究の紹介と, Anonify との比較を行っている. 最後に, 第 8 章で本研究を結論づける.

2 Anonify の要素技術

本章では, Anonify を構成する主要な技術的要素について述べる.

2.1 ブロックチェーン

ブロックチェーン技術はネットワークに参加するノードが状態遷移の正しさを互いに検証し, 暗号学的に改ざん不可能な状態データとして記録, 共有することを可能にする. Anonify では, TEE で実行される状態遷移の完全性を検証する処理を行う基盤として, そして暗号化された状態遷移の履歴を複数主体間で共有するためにブロックチェーンを用いている. したがって, Anonify で利用することが可能なブロックチェーンには, 第 4.3 節で述べる TEE の検証ロジックがスマートコントラクトとして実装可能であること, および TEE がフルノードを介してトランザクションを読み書きできることが求められる. 暗号化された状態遷移の履歴はノード間でレプリケーションされ, 復号鍵を保持している TEE はいつでも合意された最新の状態にアクセスすることが可能である.

2.2 TEE (Trusted Execution Environment)

TEE (Trusted Execution Environment) は特定のアプリケーションが他のソフトウェアから隔離された Enclave で実行されることを保証するプロセッサのセキュリティ機能である. TEE の基本的なセキュリティ要件とインターフェース定義は標準化団体 GlobalPlatform[6]が行なっている.

TEE ではハードウェアレベルの設計により, OS, ハイパーバイザなどの強い権限を持つシステムソフトウェアも保護領域のメモリに対して不正にアクセスできないようになっている. つまり, システムソフトウェアからのマルウェアに対してもアプリケーションが保護される. このようなセキュリティ性質により, 従来より格段に隔離性が高い環境

[6]https://globalplatform.org/

で汎用プログラムを安全に実行できるため, 機密性の高いデータを伴うアプリケーションへの応用が進められている. また, 多くの TEE はメモリの隔離保護に加え Attestation と呼ばれる機能を持ち, この機能により意図した実行バイナリが正規プロセッサで動作していることを保証できる. すなわち, TEE は実行バイナリの秘匿性と完全性を提供する性質を持つ. 代表的な TEE の設計として, Intel SGX [2], ARM TrustZone [3], RISC-V Keystone [4] などがあげられ, AWS Nitro Enclaves[7]も同等の性質を持つ.

一方で, TEE は全てのハードウェアリソースを物理的に隔離していないため, サイドチャネル攻撃が複数報告されている [5, 6, 7, 8, 9]. このような攻撃に対する緩和策は, [10, 11, 12, 13] などで提案され TEE のアーキテクチャ改善が進められている.

また, TPM (Trusted Platform Module) もプロセッサレベルの隔離実行機能の代表的な標準規格である. TPM はプログラムがチップセットにハードコードされているのに対し, TEE は一般開発者に隔離実行するプログラムの実装が拓かれている点で大きく異なる. また, Google Titan[8], Apple T2 [14], Microsoft Pluton[9]なども同様の機能を持つが, 実装とアップデートはあくまでプロダクトベンダのみに制限されている.

2.2.1 Intel SGX

Intel SGX (Software Guard Execution) [2] は第 6 世代の Intel Skylake マイクロアーキテクチャ以降に搭載されているプロセッサにおけるセキュリティ機能である. SGX では, プロセッサ起動時に固定サイズの保護領域が DRAM のサブセットとして確保され, Enclave 領域として仮想メモリが割り当てられる. 保護領域はプロセッサによりアクセスコントロールされ, システムソフトウェアや DMA 経由ですらアクセスすることはできない. また, Enclave 内のソフトウェアはリングプロテクションにおける Ring 3 (ユーザランド) の権限でしか命令を実行することはできず, Ring 0 のような強い権限でアプリケーションを実行することはできない. Enclave 内外へのアクセスをする特有の命令である EENTER や EEXIT も Ring 3 権限の命令である.

SGX の特徴的な機能として Sealing と Attestation があげられる. Sealing は on-chip の鍵を用いて Enclave でデータを暗号化する機能である. また, その暗号文は外部にエクスポートしディスクに保存することができる. 復号する場合は同様に暗号文を Enclave にインポートし, 同じ鍵で Unsealing を行う. もう一つの機能的性質である Attestation については次節で述べる.

[7] https://aws.amazon.com/ec2/nitro/nitro-enclaves/
[8] https://cloud.google.com/titan-security-key
[9] https://azure.microsoft.com/en-us/blog/anatomy-of-a-secured-mcu/

2.2.2 Attestation

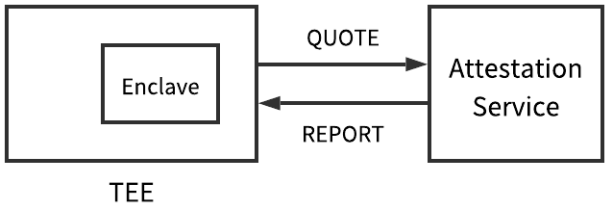

図 1: Remote Attestation

隔離保護されている Enclave 内で実行されるプログラムの完全性を第三者が検証するために, SGX では Remote Attestation [15] という機能が提供されている. Remote Attestation では, 特定の Enclave 内で正しい実行バイナリが動作していることを遠隔の Attestation Service が保証する. なお, 同一マシン上の Enclave 同士が, 同一マシンで動作していることを互いに検証するためのプロトコルである Local Attestation という機能も提供されているが, Anonify では利用していないため詳細は省略する.

Remote Attestation は, マシン外部の Attestation Service に特定の SGX プロセッサのセキュリティステータスや Enclave 実行バイナリを含むデータ構造体 QUOTE を送信し, Attestation Service が保持する秘密鍵により署名された REPORT データを得ることができる. Attestation Service は Intel が運営する IAS (Intel Attestation Service) やサードパーティが運営する DCAP (Data Center Attestation Primitives) を利用することができる. この REPORT には Enclave 内実行バイナリや実行環境に依存したハッシュ値 (MRENCLAVE) が含まれ, 同じソースコードで同じ設定でビルドすれば同じ MRENCLAVE を得ることができる. また, Remote Attestation では, TEE と Attestation Service の間に REPORT の検証を行う Service Provider と呼ばれる主体が仲介する設計が一般的であるが, Anonify では TEE と Attestation Service が直接通信を行う. なぜなら, TEE は受け取った REPORT をブロックチェーンに送信し, スマートコントラクト上で検証を行う設計となっているためである.

2.3 TreeKEM

TreeKEM [16, 17, 18] は, 複数のメンバ間でグループ鍵をバイナリツリー構造に基づき効率的に生成・共有するための KEM (Key Encapsulation Mechanism) である. KEM は, HPKE (Hybrid Public Key Encryption) [19] などの暗号プリミティブを用いて共通鍵を暗号化する仕組みのことを言う. TreeKEM は MLS (Messaging Layer Security) として知られるグループメッセージングの Ends-to-Ends Encryption (E2EE) プロトコル

のコアとなる技術であり, IETF Working Group のドラフトとして標準化が進められている [20].

MLS では, メンバ間の一連の暗号化メッセージのやり取りがあるフローにおいて, ある特定の暗号化メッセージの鍵が漏洩しても, Forward Secrecy によりそのメッセージ以前の暗号文が解読不可能なことを保証し, Post-Compromise Security によりそれ以降の暗号文が解読不可能なことを保証する. つまり, グループ間でのコミュニケーションにおいて, 特定の暗号文に対する鍵が漏洩しても他の暗号文のセキュリティへの影響を最小限にする. この MLS で中心となる技術的要素が, ツリー構造のグループ鍵決定メカニズムである TreeKEM である. 多くのメッセージングアプリケーションで上記の Forward Secrecy と Post-compromise Security の性質を満たすために, double ratchet protocols [21] が応用されているが, これはあくまで二者間メッセージングのためのプロトコルであり, 多くのメンバが存在するグループ内でのメッセージングには不向きである. MLS では, グループへのメンバの加入, 脱退, 鍵ローテーション, E2EE メッセージ送信を double ratchet protocols と比べ効率的に実現する. Anonify では, MLS の主要なサブプロトコルである TreeKEM を応用することで, ブロックチェーンで共有する暗号文に対して Forward Secrecy と Post-Compromise Security の性質を満たしながら, 全ての TEE で使う共通のグループ鍵を外部に漏洩することなく, ブロックチェーンを介した暗号文のやり取りを効率的に行う. さらに, TreeKEM を応用することで参加ノードが増えた場合でも, 鍵ローテーションをノード数 N に対して $O(\log N)$ で効率的に行うことができる.

3 Anonify の概要

Anonify はプライバシを保護した上で, ブロックチェーンを改ざん不可能なデータ共有基盤として用いることを可能にするシステムである. 通常のブロックチェーンのように, 状態遷移の履歴を平文のまま記録していくのではなく, Anonify では TEE で暗号化された命令データをブロックチェーンに記録していく. ここでの命令データには状態遷移ロジックの指定とインプットデータが含まれる. 例えば ERC20 規格[10]同等のトークンの状態遷移ロジックを実装した場合, Transfer 関数のインプットデータは送り手と受け手のユーザ ID (アドレス) と送金額となる. 参加している TEE はブロックチェーンから暗号化された命令データを取得し, Enclave で状態遷移の計算を行い, TEE に記録している状態データを更新する. Enclave 内のデータはハードウェアレベルで外部から保護され, TEE を運営している管理者自身や OS などのシステムソフトウェアもアクセスできないように保護されている.

[10] https://eips.ethereum.org/EIPS/eip-20

ネットワークに参加しているTEE同士はブロックチェーンを介し, 暗号化・復号に用いられるTreeKEMに基づくグループ鍵を共有している. これにより, あるTEEがブロックチェーンにブロードキャストした暗号文を取得し, Enclave内で復号して状態遷移を実行することでTEEの状態を更新することが可能である. つまり, ブロックチェーンには全ての状態遷移の履歴を暗号文として記録し, TEEには最新の状態を平文として記録している.

Enclaveで処理される状態遷移ルールは, TEEのRemote Attestationの仕組みで強制される. ネットワークに参加するTEEは, Remote Attestationの署名付きの結果データ(REPORT)をブロックチェーンに送信し, スマートコントラクトでREPORTの署名検証をすることで改ざんされていないことを検証する. それぞれのTEEは, REPORTに含まれるEnclaveで実行されたプログラムのハッシュ値(MRENCLAVE)がセットアップ時に登録されたものと一致しているか検証する. MRENCLAVEが一致する限り, 全てのTEEは同じ状態遷移ルールを実行していることが保証される.

つまり, 状態遷移の処理自体はそれぞれのTEEが個別に行い, 状態遷移ルールの検証は全ての参加者がブロックチェーン上のスマートコントラクトで行うことになる. これにより, 全ての参加者間でTEEで実行される状態遷移ルールが正しいことを検証し, その状態遷移の履歴は暗号化された状態で全てブロックチェーンに記録することが可能になる.

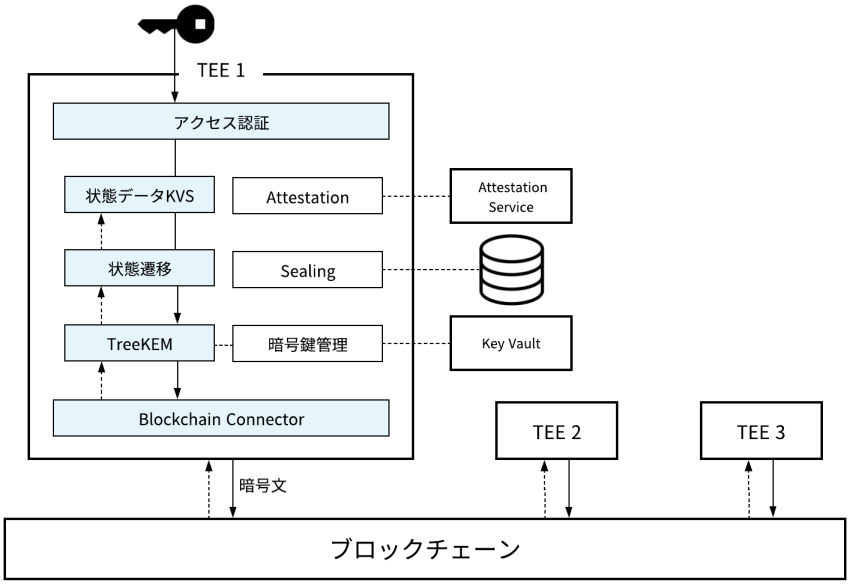

図 2: TEE における状態遷移および, ブロックチェーンで暗号文を共有するワークフロー. 青色のサブプロトコルが状態遷移時に利用される. TEE 外部からのアクセス認証により検証された命令データを TreeKEM で共有されたグループ鍵による暗号化を行い, ブロックチェーンにブロードキャストする. ブロックチェーンに記録された暗号文は TEE に読み取られ, グループ鍵で復号し得られた命令にしたがって状態遷移を実行し結果を状態データに反映する. 特定の認証情報を保持しているクライアントのみがアクセス認証を通過し状態データの KVS にアクセスできる. Sealing は, on-chip の鍵で状態データを暗号化し外部ストレージに記録するための機能を持つ. Attestation は, TEE をブロックチェーンに登録する際に実行プログラムの完全性を検証する.

3.1 セキュリティプロパティ

Anonify のセキュリティプロパティを定義する. 脅威モデルとして, 攻撃者は Anonify ネットワークにおける少なくとも 1 つの TEE を除く, 全ての TEE の OS とネットワークスタックをコントロール可能であると想定する. コントロールされたシステムにおいて, 攻撃者は送受信するメッセージのインターセプト, 並び替え, 遅延を任意に行うことができる. ただし, 各クライアントは攻撃者にコントロールされていない 1 つの TEE を利用可能とする. この脅威モデルの下, Anonify は以下のセキュリティプロパティを満たす.

- Correct Execution: ネットワーク参加者間で事前に合意されたルールに沿って状態遷移が実行されることをここでは Correct Execution と呼ぶ. これは, 第 4.3 節で述べる Attested Execution による TEE の Enclave 内実行プログラムの同一性の保証と, 第 4.2 節で述べる状態 ID による TEE 間の最新の状態データの一致検証によ

り実現される.

- Confidentiality: ハードウェアレベルで隔離保護される TEE での状態データの扱いおよびブロックチェーンには暗号化された命令のみを記録することにより, Anonify で扱うデータは第三者が不正に閲覧することはできないことが保証されている. つまり, 状態データはハードウェアレベルで隔離保護されている TEE にのみ存在し, 外部からアクセスすることはできず, また TEE 外部の通信経路およびブロックチェーン上ではセキュアな暗号アルゴリズムで暗号化されている. ブロックチェーン上に記録されるデータの暗号化には TreeKEM を用いたグループ鍵生成メカニズムが用いられ, 前方秘匿性および効率的な鍵ローテーションの仕組みを導入している. これにより万が一特定の暗号文の鍵が漏洩したとしても他の暗号文に対しての影響を最小限にすることが可能となっている.

- Fault Tolerance: システム全体において正当なノードが少なくとも 1 つ稼働していれば, ノードはブロックチェーンに記録されているトランザクションから全ての状態データを復元し, システムに参加・復旧することができる. 稼働している正当なノードは参加するノードに対し過去の暗号文を復号するための鍵を共有する. つまり, 参加しているノードは全体アーキテクチャにおける機能として全て公平であり, Anonify のシステムとして状態遷移および命令データの記録における単一障害点は存在しない. また, システムが稼働しているとき特定のノードが離脱しても他のノードに影響を与えることはない.

- Data Recoverability: TEE に記録している状態データが全て消失してもブロックチェーンから全ての状態データを復元可能な性質を Data Recoverability と呼ぶ. 全ての状態遷移の命令データは暗号化されブロックチェーンに記録されるため, 復号するための鍵を保持している限り, 最新の状態データをいつでも復元することができる.

- Accountability: 万が一, ネットワークに参加している主体が何らかの異常な操作を行なった場合に, その操作がどの TEE により行われたか検出できる性質をここでは Accountability と呼ぶ. Anonify では, 不正な署名や REPORT の提出などの検証は全てブロックチェーン上で行われ, 署名によりどの TEE が実行した操作であるか全ての参加者間で共有されるため Accountability は保証される.

3.2 機能的プロパティ

第 3.1 節で述べたセキュリティプロパティを前提とし, Anonify の機能的プロパティを定義する.

- General-purpose: 状態遷移を TEE での汎用プロセッサ上で実行するため, 柔軟な状態遷移ルールを定義可能である. また, ブロックチェーン上のスマートコントラクトでは非現実的であった計算コストの大きな処理も可能である.
- Blockchain-agnostic: Anonify は, 第 4.3 節で述べる Attested Execution における署名検証などのロジックがスマートコントラクトとして実装可能であり, 外部からトランザクションの読み書きが可能なあらゆるブロックチェーンに適用することが可能である.
- Auditability: Anonify 上の全ての状態データはプライバシ保護される一方で, 第 4.5 節で述べる監査機能により, 許可された主体が状態データを閲覧し, 監査することが可能となる. 監査主体はブロックチェーン上に記録された暗号文を復号する鍵を所有し, その鍵を用いて平文の状態データを参照することができる.

3.3 TEE での状態データ管理

状態データである *state* は Enclave 上に記録される. それぞれのユーザに対する *state* は以下のようなシンプルな Key-Value ストアで管理されている.

$$(userid, id_m) \rightarrow (state, d)$$

userid はアクセス認証の検証情報から得られるユーザ ID であり, ユーザ公開鍵から一意に定まる値である. この場合, 対応する秘密鍵による署名を提示することでのみ, *state* にアクセスすることが可能である. id_m は同じ *userid* に対する *state* の識別子であり, id_m をキーに指定することで同じ *userid* に対して複数の *state* を Enclave 内で管理でき, 複数の状態遷移を定義することができる. d は特定のキーの更新数を表す値であり, これにより第 4.2 節で述べる状態 ID が一意に決定する.

3.4 ワークフロー

本節では, Anonify ネットワークに TEE が参加する登録フェーズ, 外部から Anonify へトランザクションを送信する状態遷移フェーズ, そして暗号化に用いられるグループ鍵をローテーションする鍵ローテーションフェーズのワークフローを述べる.

ここでは, 外部からトランザクションを送信する主体は, Enclave へアクセス可能な鍵などの認証情報を保持していることを前提とする. それぞれのワークフローにおける要素技術の詳細は次章で述べる.

3.4.1 登録フェーズ

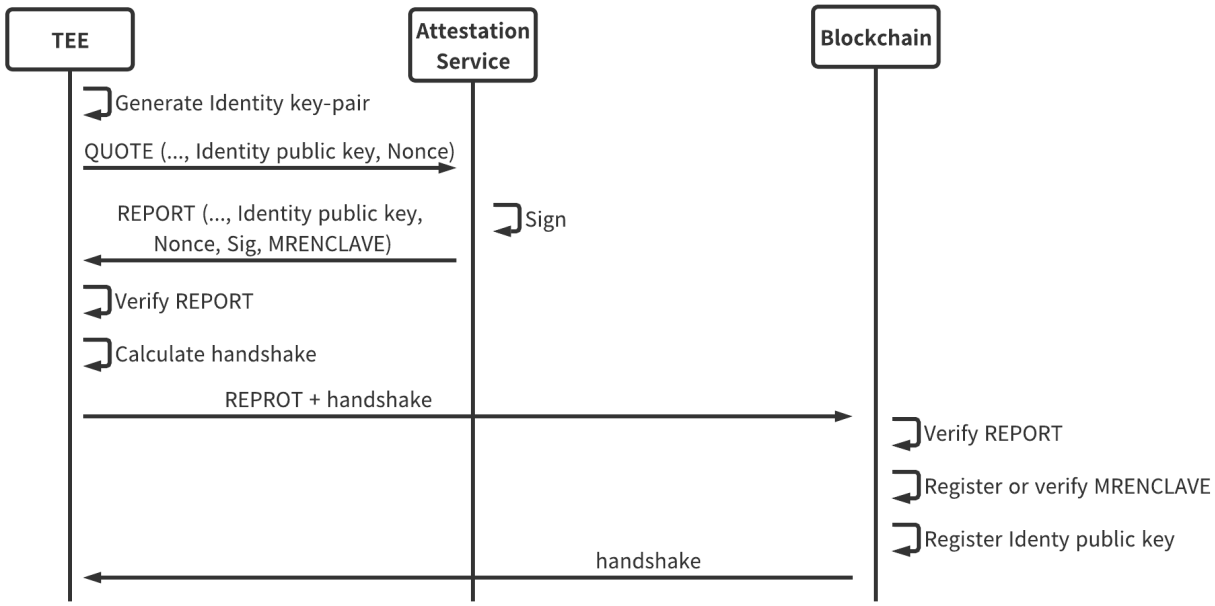

図 3: 登録フェーズ

TEE が正規の状態遷移プログラムを実行することを検証した上でブロックチェーンに公開鍵を登録するフェーズである. Remote Attestation により得られた REPORT のオンチェーン検証とそれに含まれている Enclave Identity 公開鍵のオンチェーンへの登録を行う. REPORT の検証には, REPORT の署名検証や MRENCLAVE の一致検証などが含まれる. 登録フェーズは, 新しい Anonify ネットワークを生成するとき, TEE が既存の Anonify ネットワークに参加するとき, REPORT タイムスタンプの有効期限が切れた場合に実施されるフェーズである. また, 新しいノードの参加によるグループ鍵の更新をするためにハンドシェイクパラメタのブロードキャストも行う. TEE が Enclave を生成してから, Anonify 上の特定のグループに参加するまでのワークフローは以下のようになる.

1. Enclave 生成時に Enclave に閉じた Identity 鍵ペアを生成
2. Remote Attestation を行う. QUOTE に Identity 公開鍵や Nonce を含め, Attestation Service に送信し署名付き REPORT のレスポンスを受け取る
3. グループ鍵を共有するための TreeKEM グループ状態に基づくハンドシェイクを生成する.

4. REPORT とハンドシェイクを含むトランザクションを生成しブロックチェーンにブロードキャスト.

5. ブロックチェーン上での REPORT の一連の検証を通して, Identity 公開鍵をブロックチェーンに記録することで TEE の参加が完了する. 新しい Anonify ネットワークを生成する場合は, ブロックチェーンで REPORT の検証後に MRENCLAVE を記録する. 一方, 既存の Anonify ネットワークに参加する場合は, REPORT の検証後にすでにブロックチェーンに記録されている MRENCLAVE と提示している MRENCLAVE が一致するか検証を行う.

3.4.2 状態遷移フェーズ

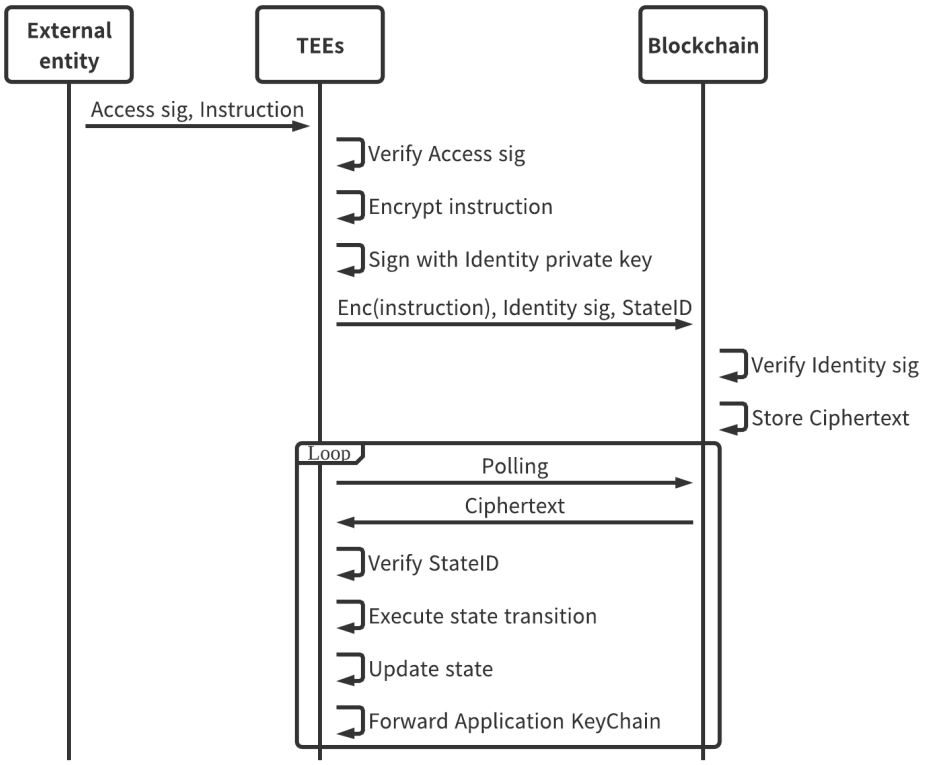

図 4: 状態遷移フェーズ

暗号化された状態遷移の命令を共有するためにトランザクションをブロックチェーンネットワークでブロードキャストするフェーズである. ユーザが Anonify 上でトランザクションを送信し, ブロックチェーンで共有され他の TEE の状態に反映されるワークフローは以下のようになる. また, ユーザとは認証情報を持つ外部のエンティティを指し, TEE の運営者が管理している場合もある.

1. 外部エンティティが TEE に対してアクセスに必要な認証情報と状態遷移の命令と競合を回避しなければならない状態に関する状態 ID (第 4.2 節) を含む状態遷移のインプットを送信する. 送金アプリケーションの場合は, 状態遷移のインプットに命令として送金額と送り手, 受け手の $userid$ を入れ, 状態 ID に送り手の $userid$ と残高に基づくハッシュ値を入れる.

2. Enclave 内で認証情報の検証を行い, TEE は状態遷移の命令を暗号化, Enclave Identity 秘密鍵による署名, 第 4.2 節で述べる状態 ID の計算を行い, ブロックチェーンノードにこれらのデータを含めたトランザクションをブロードキャストする.

3. ブロックチェーン上のスマートコントラクトで Enclave Identity 公開鍵による署名検証後, 暗号文を記録する.

4. ネットワーク内の全ての TEE はブロックチェーンに記録されたトランザクションデータから状態 ID を検証後, Enclave 内で暗号文を復号し状態遷移ロジックを実行, 状態データの更新をする. 送金アプリケーションの場合は, 送り手の残高が送金額分減り, 受け手の残高がその分増える. Application KeyChain は TreeKEM に基づく鍵の算出ロジックであり一連の処理が反映後, KeyChain の更新を行う.

3.4.3 鍵ローテーションフェーズ

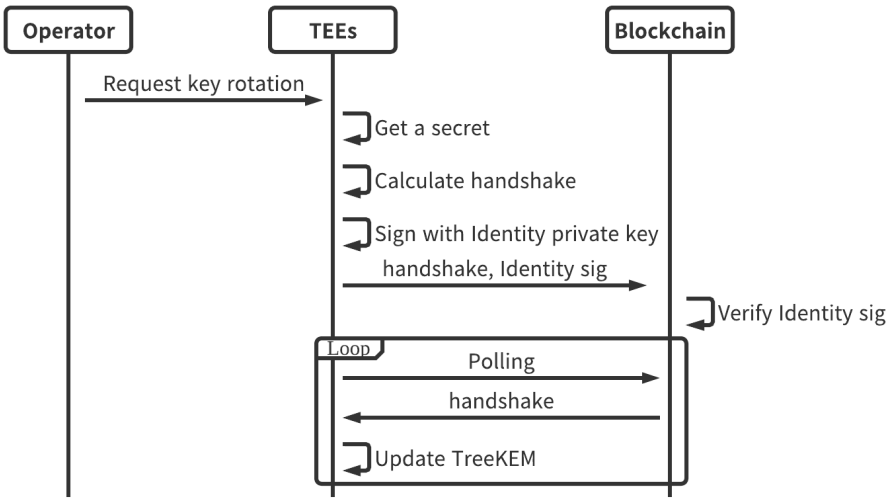

図 5: 鍵ローテーションフェーズ

全ての TEE で共有しているグループ暗号鍵をローテーションするフェーズである. Anonify ネットワークに参加している TEE の運営者が第 2.3 節で述べた暗号鍵の Post-compromise Security の性質を満たすために行う. このフェーズを実施するタイミング

はそれぞれの運営者が柔軟に設定可能である．具体的なワークフローは以下のようになる．

1. TEE の運営者は, 自身のクライアントを介して鍵ローテーションのリクエストを TEE へ送信する．
2. リクエストに合わせて新しい鍵を内部で生成あるいは外部から取得を行う．
3. 新しい鍵を用いて TreeKEM のグループ状態を更新するために必要なハンドシェイクを計算する．
4. 全ての TEE で TreeKEM のグループ鍵を更新するために参加している TEE の公開鍵を用いて暗号化しブロックチェーンへブロードキャストする．
5. それぞれの TEE はフェッチした暗号文を自身の秘密鍵で復号し, TreeKEM のグループ状態をアップデートし, グループ鍵算出に用いられる共通のグループシード値が得られる．

4 Anonify のサブプロトコル

本章では, Anonify を構成するサブプロトコルについて説明する．なお, Σ は安全な署名スキームとする．

4.1 Enclave へのアクセス認証

特定のユーザ ID に対する状態遷移の実行や状態の取得はそのアクセス権限を保持する主体のみが実行可能である．ここでは, 署名検証に基づくチャレンジ・レスポンス認証を用いる．まず, クライアントからの TEE へのアクセスリクエストに対し, TEE は擬似乱数 c をレスポンスする．続いて, クライアントは秘密鍵 sk_u で c に署名し, TEE へ送信する．TEE はクライアントが c に対して署名したことを, クライアントの公開鍵 pk_u により検証する．この署名検証に成功した場合, TEE は pk_u からユーザ ID $userid$ を算出し, そのクライアントが $userid$ に対応する Enclave 内の状態データにアクセスすることを許可する．ただし, クライアントの公開鍵からユーザ ID は一意に定まるものとする．

Algorithm 1 Challenge-response authentication

1: **Prover $\mathcal{P}$:**
2: $(pk_u, sk_u) \coloneqq \Sigma.\textsc{keygen}(1^\lambda)$
3: receive c from $\mathcal{V}$
4: $\sigma_c \coloneqq \Sigma.\textsc{sign}(sk_u, c)$
5: send σ_c to $\mathcal{V}$
6: **Verifier $\mathcal{V}$:**
7: $c \leftarrow \{0,1\}^\lambda$
8: send c to $\mathcal{P}$
9: receive σ_c from $\mathcal{P}$
10: assert $\Sigma.\textsc{verify}(pk_u, \sigma_c, c)$

4.2 状態 ID

第 3.1 節で定義された脅威モデルにおいて攻撃者は, TEE のネットワーク I/O のインターセプトや, 不正なブロックチェーンノードの稼働により, 不正な状態データから状態遷移の命令を送信することが可能である. これにより, 例えば以下のような攻撃が考えられる.

- ブロックチェーンに記録されたトランザクションの TEE への反映を遅延もしくはスキップすることで, ある状態遷移が実行される前の状態において命令を生成する攻撃. 送金アプリケーションでは, 攻撃者は自身の TEE において, 自身の送金トランザクションの反映を遅らせ, 送金額が残高から引かれる前に本来の残高以上の金額を送金する命令の作成が可能となる.
- ブロックチェーンに記録されていないトランザクションを TEE へ反映することで, 不正な状態遷移が実行された後の状態において命令を生成する攻撃. 送金アプリケーションでは, 攻撃者が管理する 2 つのユーザ A と B に関し, A から B への送金トランザクションをネットワークにブロードキャストしないまま, 自身の TEE に反映することで, B が本来の残高以上の金額を送金する命令の作成が可能となる.

これらの攻撃により生成されたトランザクションがブロードキャストされた場合に, その他の TEE が状態遷移フェーズにおいて検出できるよう, Anonify は状態 ID という仕組みを提供する. 具体的には, 遷移前の状態データから導出される状態 ID $stateid$ を以下のように定義し, 上記の攻撃で攻撃者が利益を被る状態に対する状態 ID をトランザクションに含める. 例えば, 残高以上の金額を送金が実行されないように送金者の状態 ID は含め, 受金者の状態 ID は含めない.

$$stateid = \textsc{hash}(userid \parallel id_m \parallel state \parallel d)$$

id_m は, 同じ $userid$ に紐付く状態の識別子であり, $state$ が状態データを表す. そして, ブロックチェーンに記録されたトランザクションにより TEE に新しい状態を反映する際, Enclave 内で更新しようとしている状態データが状態 ID と適合するか検証する. これにより, 不正な状態データに対する状態遷移の命令を他の TEE が検出でき, トランザクションの署名からどの TEE で不正が起きたのか特定が可能である.

4.3 Attested Execution

Anonify では, 状態遷移の処理はオフチェーンの TEE で行い, オンチェーンで状態遷移が正しく行われていることの検証を行う. このように, ブロックチェーンネットワークに参加する全ての TEE が確かに正しい状態遷移ロジックを実行していることを保証する仕組みを Attested Execution と呼ぶ. Attested Execution は TEE の登録時に行う Remote Attestation の仕組みを利用し, トランザクションをブロードキャストするたびに実行されるサブプロトコルである. Remote Attestation は, プロセッサ製造時に組み込まれた固有の鍵を用い, プロセッサが生成するデータ構造体 (MRENCLAVE を含む) を署名し, Attestation Service に送信する. Attestation Service は署名検証を行い, CPU のセキュリティバージョンや CRL (リボケーションリスト) の検証により CPU の正当性状態や MRENCLAVE を含むデータ構造体を署名し, TEE へレスポンスする.

Remote Attestation により, 状態遷移プログラムの完全性を保証する. TEE が Attestation Service に送信した QUOTE リクエストに対し, REPORT とその署名 σ_{att} が返される. Enclave を初期生成する時に Enclave Identity 鍵ペア (pk_e, sk_e) を生成し, pk_e を $w_{v,c}$ に含め, ブロックチェーン上に登録する. これにより, Attested Execution は pk_e による署名検証に置き換えることができる. なぜなら, (pk_e, sk_e) は Enclave の実行プログラムに依存するため, sk_e による署名が Remote Attestation と同じ完全性を与えることができる. この Remote Attestation はオンチェーンへの TEE の登録時のみに行われる.

Algorithm 2 Setup Attested Execution

1: **Prover $\mathcal{P}$:**
2: $(pk_e, sk_e) \coloneqq \Sigma_{ident}.\textsc{keygen}(1^\lambda)$
3: $rn \leftarrow \{0,1\}^\lambda$
4: // Remote Attestation
5: $(w_{v,c}, \sigma_{att}) \coloneqq \textsc{attestation}(pk_e, rn)$
6: send $(w_{v,c}, \sigma_{att})$ to $\mathcal{V}$
7: **Verifier $\mathcal{V}$:**
8: receive $(w_{v,c}, \sigma_{att})$ from $\mathcal{P}$
9: assert $\Sigma_{att}.\textsc{verify}(pk_{att}, \sigma_{att}, w_{v,c})$
10: register pk_e

4.4 TreeKEM

ブロックチェーンを介し複数の TEE 間で, 暗号化に使用するグループ鍵を効率的に管理するために TreeKEM [16] を応用する. TreeKEM は, バイナリツリーのインデックスを i で表現し, 自身のリーフノードは鍵ペア用いて $(pk_{i,e}, sk_{i,e})$, 自身以外のリーフノードは $(pk_{i,e}, \cdot)$ と表現できる. e は, グループのエポックであり, グループ鍵のローテーションやメンバの追加, 脱退によりエポックはインクリメントされていく. ここで, ノードの秘密鍵 $sk_{i,e}$ と親ノードの $secret_{i,e}$ は以下のように導出される.

$$sk_{i,e} \coloneqq \mathrm{KDF}(\text{“}node\text{”}, secret_{i,e})$$
$$secret_{i,e} \coloneqq \mathrm{KDF}(\text{“}path\text{”}, secret_{i\pm1,e})$$

このようにして自身の鍵ペアと他参加者の公開鍵から導出されたルートノードをシードとすることで, グループに参加するメンバーに共通の鍵を生成することができる. 鍵ローテーションによりエポック e はインクリメントされ $secret_{i,e+1}$ が自身の新しいリーフノードにセットされ, 新しいグループ鍵を導出するためのパラメタがブロックチェーンを介してブロードキャストされる (詳細は [16] を参照). 以上のように, 鍵ローテーション操作あるいはメンバーの追加・削除によりルートノードが更新され, TreeKEM のグループ状態が $\mathcal{S}_0, \mathcal{S}_1, ..., \mathcal{S}_n$ と遷移していく.

一方, それぞれのグループ状態のルートノードから鍵導出関数により Application Key である $k_{e,f,g}$ (f: グループ内におけるメンバーのインデックス, g: Application Keychain のインデックス) が導出される. メッセージをブロックチェーンから受け取り, 復号する際に Application Keychain が以下のように進む.

$$k_{e,f,g} \coloneqq \mathrm{KDF}(f, k_{e,f,g-1})$$

ブロックチェーンに記録する命令データに対する暗号文は以下のように表記される.

$$C_{e,f,g} \coloneqq Enc[k_{e,f,g}](id_{fn} \parallel input \parallel padding)$$

id_{fn} は事前に決められた状態遷移ロジックに対する識別子である. $padding$ により $input$ のサイズの違いを吸収し, 全ての暗号文を同じサイズにすることで暗号文サイズから実行する状態遷移の情報が漏洩しないようにする.

4.5 暗号鍵管理と監査

鍵 $secret_{i,e}$ を生成・管理する手法は, 監査機能の提供有無により以下の 2 つに分類できる.

- 監査機能を提供しない場合: TEE で $secret_{i,e}$ を生成・管理するためハードウェアレベルで外部からのアクセスを防ぎ, 監査機能の提供は行わない.

- 監査機能を提供する場合: 監査主体が外部で $secret_{i,e}$ を生成・管理する.

前者の方法では, TEE での擬似乱数をシードに $secret_{i,e}$ を生成し, Sealing など TEE に閉じた鍵で $secret_{i,e}$ を暗号化して外部ストレージに記録し管理する. 後者の方法では, 特定の監査主体が状態遷移を監査できるよう, 監査主体が外部の鍵管理サービスで $secret_{i,e}$ を生成し, 利用時に TEE がそれをリクエストする. 監査主体は $secret_{i,e}$ を TEE の外部で保持しているため, ブロックチェーンにアクセスして取得した $C_{e,f,g}$ を自身が管理している $secret_{i,e}$ とブロックチェーン上の e, f, g の情報から $k_{e,f,g}$ を生成し, 暗号文を復号できる.

5 実装とパフォーマンス評価

本章では, Anonify の実装とそのパフォーマンス評価を述べる. Anonify は Rust SGX SDK[11] を用いて Rust で実装し, パフォーマンス評価は Microsoft Azure で 2 コアの Intel Xeon E-2176G 3.70 GHz CPU, 8GB RAM, Ubuntu 18.04.4 LTS 上で SGX が動作可能な VM 環境で行った. また, Ethereum のトークン規格である ERC20[12] を参考に, transfer, transfer_from, approve, mint, burn の状態遷移の命令を実装した. これらの命令および, 鍵ローテーションの命令 key_rotation と, TEE の登録の命令 register の SGX マシン上でのトランザクション生成時間をそれぞれ測定したところ, 以下の結果となった.

関数	transfer	transfer_from	approve	mint	burn	key_rotation	register
処理時間 [ms]	0.305	0.335	0.313	0.351	0.339	0.328	415

表 1: 各メソッドに対するトランザクション生成時間

register は Remote Attestation が外部の Attestation Service にリクエストを送信するため, 他のメソッドと比較して処理に長い時間がかかった. その他のメソッドに関しては SGX マシン上でトランザクション生成をする.

6 Anonify プロトコルの拡張

6.1 オンチェーンデータとのインターオペラビリティ

Anonify プロトコルにより秘匿化されている平文の状態データは全て TEE に記録されており, Anonify プロトコル外でスマートコントラクトにより管理されているアセットデータと相互に変換はできない. 例えば, スマートコントラクトで管理されているトー

[11] https://github.com/apache/incubator-teaclave-sgx-sdk
[12] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol

クンを Anonify プロトコルに反映し, 秘匿化されたトークンとして扱うことは現状できない. そこで, このようなブロックチェーン上の既存のアセットデータと Anonify プロトコル内のデータのインターオペラビリティを可能にするプロトコルを提案する. このインターオペラビリティは以下で述べるインポート機能とエクスポート機能に分けられ, これら 2 つの機能が第 3.1 節で述べた Anonify のセキュリティプロパティを満たしながら安全に実現できることを目標とする.

- インポート機能: オンチェーンのスマートコントラクトで管理されているデータを Anonify 内の状態データとして取り込む機能である. スマートコントラクトで管理されているアセットデータを Anonify プロトコルにインポートするために, エスクローコントラクト[13]などをデプロイし, Anonify プロトコルがそのコントラクトのイベントを読み取るように稼働させておく. クライアントがエスクローコントラクトの deposit 関数を実行するトランザクションを送信すると, トークンはエスクローコントラクトにロックされ, TEE の状態データに $userid$, トークン量, コントラクトアドレスが記録される. この状態データは Anonify プロトコル内で状態遷移させることが可能である.
- エクスポート機能: Anonify プロトコルで管理されている状態データを Anonify 管理外のスマートコントラクト上の状態データとして取り出す機能である. TEE がエスクローコントラクトの withdraw 関数などを実行するトランザクションを送信することで, 指定したトークン量を $userid$ に対して取り出すことができる. TEE はそのトランザクションを読み取り, $userid$ が保持するトークンをその分減少させる.

これらの機能により, 例えば証券の DvP 決済において, オンチェーンのスマートコントラクトで管理されている法定通貨建てのトークンと, Anonify で秘匿化して管理する証券データをアトミックに交換することが可能となる.

エクスポート機能を実現するためには, エクスポートのトランザクションの正しさがオンチェーンで検証可能である必要がある. ここで, 第 4.2 節で述べた手法により, 攻撃者は自身の TEE の状態データを不正に書き換えることができるため, 不正な状態データからエクスポートのトランザクションを生成することが可能である. 例えば, Anonify 内で実際には受け取っていないトークン, または, 他ユーザーに既に移転したトークンをエクスポートするトランザクションが生成可能である. このようなトランザクションは第 4.2 節の状態 ID により, 他の TEE では事後的に検出可能であるが, オンチェーンで検出することはできない.

[13] https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/payment/escrow/Escrow.sol

そこで, 不正な状態データのエクスポートをオンチェーンで防止するために, TEE 内のライトクライアント機能とオンチェーンでの状態 ID の重複検証を加える. ここで, ライトクライアント機能とは, ブロックチェーンのデータを全て保持せずに, あるトランザクションが記録されたことを検証する仕組みである. ライトクライアント機能を TEE 内に実装することで, 攻撃者はブロックチェーンにトランザクションを記録する以外の方法で不正に TEE の状態データを更新することが不可能となる. なお、TEE 内にライトクライアント機能を実装する仕組みは Ekiden [22] において Proof of Publication として議論されており, PoW ベースの Nakamoto コンセンサス用のライトクライアント機能が提案されている. 次に, オンチェーンでの状態 ID の重複検証とは, エクスポートのトランザクションが含む状態 ID が既に他のトランザクションに含まれている場合に, そのエクスポートを無効化することである. エクスポートのトランザクションに, エクスポートする状態データに紐づく状態 ID を強制的に含ませることで, 攻撃者が過去の状態データをエクスポートすることは不可能となる. 以上の仕組みにより, 正当かつ最新の状態データのエクスポートのみが可能となることを保証できる.

6.2 送受金の競合の回避

Anonify ではある状態データに関して, その保有者であるユーザ自身による状態遷移と, その他のユーザによる状態遷移とが競合してしまう場合がある. 例えば, トークン管理等のユースケースにおいて, ユーザ A が送金をするトランザクション T_A と, ユーザ B がユーザ A に送金をするトランザクション T_B が同時に生成されたとする. このとき, T_A より前に, T_B がブロックチェーンに記録されると, TEE においても T_B が T_A より先に反映される. その結果, ユーザ A の残高が T_A の生成時点から変化しているため, T_A の状態 ID は不正となり, T_A の状態遷移は実行されなくなる. このような状態遷移の競合は, ユースケースによっては実用上の課題となりうる.

このような問題を解決するため, Zether [23] で提案されている Pending state と同様の仕組みを導入できる. 例えばトークン管理等のユースケースでは, 各ユーザは自分が送金に使うトークン用のメインの状態データと, 他ユーザから送金された金額の合計を一時的に記録する状態データを分け, 定期的にメインの状態データに合算する等が可能である.

7 関連プロジェクト

本章では, Anonify に関連するブロックチェーンにおけるプライバシ保護に取り組むプロジェクトとして, ゼロ知識証明と TEE を用いたものについて説明する.

7.1 ゼロ知識証明

ブロックチェーンのプライバシ保護を保護するために, 送信するトランザクションデータを暗号化し, 相手に復号するための鍵を共有するアプローチがある. ゼロ知識証明は, この時の計算の正しさ（送金額の範囲制限や暗号化処理の完全性）を送信者の秘密鍵や送金額などの平文を明らかにせずグローバルに保証することができる. つまり, ブロックチェーンにブロードキャストされたバイト列が特定の “正しい” 暗号文であることを保証する.

ユーザの匿名性を保護するためのナイーブな施策としては, 利用するアドレスをその都度, 新しく生成する方法があげられる. 本方式は, UTXO 型ブロックチェーンに対する Wallet で実装されている. 個人と紐付くアドレスが毎回新しくなるため, 一見, 匿名性が保護されるよう思えるが, 状態に対するトランザクション同士の繋がりが特定のアドレス間の繋がり (Linkability) を明らかにしてしまう [24, 25]. これにより, 結局それぞれの新しいアドレスと特定の主体がヒューリスティックに紐付けられてしまう. これを緩和するために, CryptoNote Protocol [26] ではリング署名を用いて, 複数のメンバ公開鍵をインプットとし, 第三者からはどのメンバ公開鍵に対応する秘密鍵によって署名されているのか識別できないようにしているが, 依然として多くの Linkability 攻撃が提案されている [27, 28, 29].

Zcash Protocol では, トランザクション同士の Linkability が一切明らかにならないように, 未使用の状態データを Append-only な Merkle Tree のリーフに記録し, 消費した状態データを別のプールに Merkle Tree のリーフインデックスと紐づかない形で記録していく. Merkle Tree に記録されている特定の状態を消費するときに, 自身がその状態を保持していること, 確かに Merkle Tree のリーフに含まれていること, 消費済みプールに記録されていないこと（二重支払い防止）などをゼロ知識証明を用いて保証する. 加えて, これらのプロトコルは特定の監査主体に対してアドレスと紐付く特定の復号鍵を渡すことで, ブロックチェーン上のそのアドレスが送信元となっている暗号文を復号可能にする.

ゼロ知識証明によるプライバシ保護の取り組みは上記のように UTXO 型のブロックチェーンプロトコルにおいてだけではなく, Ethereum[14]などのスマートコントラクトをベースにしたアプリケーションレイヤにおいても実装が進められている. 代表的なプロジェクトとしては, 従来の zk-SNARKs(ゼロ知識証明) において問題視されていたパラメタ生成 (Trusted Setup) を算術回路非依存にすることが可能な Plonk [30] をベースにしてプライバシ保護のフレームワークを提供する Aztec[15]がある. その他にも EY,

[14]https://ethereum.org/
[15]https://www.aztecprotocol.com/

ConsenSys, Microsoft らが連携して取り組む Baseline Protocol[16]もエンタープライズ向けの基幹業務システムなどをゼロ知識証明を用いて機密データの保護をしようとしている. また以上の取り組みは現状, 比較的シンプルな送金などの状態遷移の秘匿化, 匿名化に対するアプローチであったが, Hawk [31] や Zexe [1] のように任意の状態遷移に対しゼロ知識証明を適用しようとする取り組みも行われている.

以上のように, ゼロ知識証明を用いて状態遷移の完全性を与えることでブロックチェーンにおけるプライバシ保護を行うアプローチの開発プロジェクトは活発に進んでいる一方で, 証明時の計算オーバーヘッドや監査の困難さなど現実のアプリケーションに適用するためには多くの技術的課題が存在する. 特に, 表現力の高い状態遷移ロジックのプライバシ保護は算術回路における任意の状態遷移ロジックの計算や再帰的な実行 (ゼロ知識証明の検証を算術回路で処理) が原因でより多く証明時オーバーヘッドが生じゼロ知識証明を応用することは困難である.

7.2 TEE

CCF [32] は, Microsoft が進めている TEE をプロトコルに組み込んだオープンソースフレームワークである. 秘匿化すべきデータは TEE で暗号化し, その他のデータは平文でブロックチェーンに記録する. CCF は TEE をブロックチェーンのプロトコルに組み込んだ設計になっているため Anonify のようにバックエンドとなるブロックチェーンは選択できない. また, 暗号化に用いる鍵はそれぞれのノードで秘密分散管理されているため特定の監査機関に対して監査性の機能を与えることはできない.

Hyperledger Avalon [33] は EEA (Enterprise Ethereum Alliance) が策定を進めている Off-Chain Trusted Compute [34] の仕様を元に実装をしているプロジェクトである. Avalon は TEE だけではなくゼロ知識証明や sMPC などのソフトウェアベースで完全性を保証する手法に対しても適用可能なように, より一般的に開発されているフレームワークである. Avalon では, クライアントから TEE を介してブロックチェーンへ送る暗号文に対応する鍵を, それぞれワンタイムキーとしてクライアントが生成・管理している. よって, TEE がブロックチェーンに記録された暗号文から状態を取得するためには全てのワンタイムキーを複数の TEE で同期・保持する必要がある. この制約から, ブロックチェーンに記録した過去の状態に基づき連続的に状態遷移を計算するような使い方はできない. 一方, Anonify ではネットワーク全体で一意に定まるブロックチェーンのトランザクションの順序にしたがって, それぞれの TEE で管理するグループ鍵が決定的に求まる. そのため, 参加している全ての TEE でブロックチェーンに記録されている過去の状態も永続的に活用することが可能である.

[16] https://www.baseline-protocol.org/

Ekiden [22] は, TEE を用いることによりプライバシとスケーラビリティに優れた特徴を持つ独自のブロックチェーンを目指す. Ekiden はパブリックチェーンを前提としており, 暗号化鍵を管理する TEE が経済的インセンティブにより分散運営される. Anonify がエンタープライズ向けブロックチェーンをバックエンドに用いることを想定しているのに対し, Ekiden はスケーラブルかつプライバシ志向なパブリックチェーンを目指している点で大きく異なる. また, Anonify は状態データに対する外部監査の仕組みを提供しているのに対し, Ekiden は分散された TEE クラスタで鍵が秘密分散され管理されているため監査機能は存在しない.

LucidiTEE [35] は, Visa Research が公表している TEE とブロックチェーンを用いたマルチパーティー計算 (sMPC) のプロトコルである. 複数のプレイヤーが機密性の高いデータを他者には秘匿にしつつ TEE を用いて協力して計算を行う. この計算自体は TEE で行われるが, TEE は外部との I/O 部分を保護することはできない（インターセプトなどが悪意ある OS により潜在的に可能）ため, 過去の状態に依存したマルチパーティー計算や全てのパーティに対する公平性の保証が TEE 単体では困難であるためブロックチェーンを用いる. 参加パーティ全員がアクセス可能なブロックチェーンに対して計算のインプット, アウトプットあるいは更新状態値のハッシュ値をコミットしておくことで上記のポリシーに従うようにマルチパーティ計算を行うことを可能にする. つまり, Anonify はブロックチェーンとしてのロバスト性のある状態データの保持を目指しているのに対し, LucidiTEE は公平性を満たした秘密計算の大規模処理を目指している点で異なる. LucidiTEE は復元可能な状態を全員に可用なブロックチェーンにおかず, 秘密計算のための暗号化された更新状態としてローカルに記録する.

8 結論

ブロックチェーンのプライバシを保護するシステムである Anonify の性質とアーキテクチャを述べた. TEE をベースに状態データの秘匿性と完全性を保証することで, 効率的かつ汎用的な状態遷移のプライバシ保護を実現する. 加えて, 特定の監査主体が復号のための鍵を管理することで監査機能を提供するスキームも持つ. 今後は, TEE に記録している状態データを暗号化して外部ストレージに保存するスナップショット機能, 自身が管理しているアカウントに対する状態データのエクスプローラ機能, クライアントの新しいアクセス認証方法の対応などの機能拡充や実装レベルでのセキュリティ・パフォーマンス向上を進めていく.

謝辞

BI-SGX[17]の開発者であり, 2019 年度未踏スーパークリエータである櫻井碧氏には Intel SGX に関する多くの有益なご助言をいただき感謝申し上げます.

参考文献

[1] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 820–837, 2018.

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

[3] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

[4] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. Keystone: A framework for architecting tees. *arXiv preprint arXiv:1907.10119*, 2019.

[5] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[6] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 1041–1056, 2017.

[7] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 557–574, 2017.

[8] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[17] https://bi-sgx.net/

[9] Kit Murdock, David Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel sgx. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[10] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, Urs Müller, and Ahmad-Reza Sadeghi. Dr. sgx: hardening sgx enclaves against cache attacks with data location randomization. *arXiv preprint arXiv:1709.09917*, 2017.

[11] Yangchun Fu, Erick Bauman, Raul Quinonez, and Zhiqiang Lin. S gx-l apd: Thwarting controlled side channel attacks via enclave verifiable page faults. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 357–380. Springer, 2017.

[12] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 217–233, 2017.

[13] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *NDSS*, 2017.

[14] us-19-davidov-inside-the-apple-t2.pdf. `https://i.blackhat.com/USA-19/Thursday/us-19-Davidov-Inside-The-Apple-T2.pdf`. (Accessed on 05/29/2020).

[15] Attestation service for intel(r) software guard extensions: Api documentation. `https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf`. (Accessed on 04/11/2020).

[16] Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. Treekem: Asynchronous decentralized key management for large dynamic groups, 2018.

[17] Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the ietf mls standard for group messaging. Technical report, Cryptology ePrint Archive, Report 2019/1189, 2019. https://eprint. iacr. org ⋯.

[18] Joel Alwen, Margarita Capretto, Miguel Cueto, Chethan Kamath, Karen Klein, Guillermo Pascual-Perez, Krzysztof Pietrzak, and Michael Walter. Keep the dirt: Tainted treekem, an efficient and provably secure continuous group key agreement protocol.

[19] Richard Barnes, Karthikeyan Bhargavan, and Christopher A. Wood. Hybrid Public Key Encryption. Internet-Draft draft-irtf-cfrg-hpke-03, Internet Engineering Task Force, May 2020. Work in Progress.

[20] Richard Barnes, Benjamin Beurdouche, Jon Millican, Emad Omara, Katriel Cohn-Gordon, and Raphael Robert. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-09, Internet Engineering Task Force, March 2020. Work in Progress.

[21] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–158. Springer, 2019.

[22] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *arXiv preprint arXiv:1804.05141*, 2018.

[23] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. *IACR Cryptology ePrint Archive*, 2019:191, 2019.

[24] Dmitry Ermilov, Maxim Panov, and Yury Yanovich. Automatic bitcoin address clustering. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 461–466. IEEE, 2017.

[25] Michael Fleder, Michael S Kester, and Sudeep Pillai. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657*, 2015.

[26] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *European Symposium on Research in Computer Security*, pages 456–474. Springer, 2017.

[27] Dimaz Ankaa Wijaya, Joseph Liu, Ron Steinfeld, and Dongxi Liu. Monero ring attack: Recreating zero mixin transaction effect. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1196–1201. IEEE, 2018.

[28] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In *European Symposium on Research in Computer Security*, pages 153–173. Springer, 2017.

[29] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 2018(3):143–163, 2018.

[30] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Technical report, Cryptology ePrint Archive, Report 2019/953, 2019.

[31] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[32] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, et al. Ccf: A framework for building confidential verifiable replicated services. Technical report, Technical Report MSR-TR-2019-16, Microsoft, 2019.

[33] hyperledger/avalon: Hyperledger avalon. `https://github.com/hyperledger/avalon`. (Accessed on 04/16/2020).

[34] Enterprise ethereum alliance off-chain trusted compute specification v1.1. `https://entethalliance.github.io/trusted-computing/spec.html`. (Accessed on 04/21/2020).

[35] Rohit Sinha, Sivanarayana Gaddam, and Ranjit Kumaresan. Luciditee: Policy-based fair computing at scale. *IACR Cryptology ePrint Archive*, 2019:178, 2019.