「マルチメディア，分散，協調とモバイル
(DICOMO2021)シンポジウム」 令和3年6月-7月

分散型台帳への秘密鍵の封入による協同運用可能な公開鍵証明書発行基盤の検討

熊谷 圭太[1] 掛井 将平[1] 白石 善明[2] 齋藤 彰一[1]

概要：複数の組織がコンソーシアムを組むことで，知見や技術を融合し，新たな価値の創出を目指すようなシステムが考案されている．複数の組織が協同運用するようなシステムにおいて，既定のデバイスだけを接続させるような運用には限界があることから，公開鍵基盤により発行される公開鍵証明書に基づくデバイスの真正性の保証が求められる．しかし，公開鍵基盤では真正性の保証は単一信頼点である認証局が担うことから，認証局を運用する組織への依存が必要となる．本稿では，コンソーシアム型のシステムに適した認証局の構築に向けて，複数の組織で協同運用できる公開鍵証明書発行基盤を提案する．各組織による自主的な運用を目指して，個々の組織主導で公開鍵証明書を発行できるようにしつつも，不正な公開鍵証明書の発行を相互に監視できるようにするための要件を整理した．そして，整理した要件をもとに，Intel SGX により秘匿化された分散型台帳を実現できる Hyperledger Fabric Private Chaincode による公開鍵証明書発行基盤の設計を示す．公開鍵証明書の発行に必要な秘密鍵を直接共有せずに，秘密鍵が封入された分散型台帳を共有することで各組織による公開鍵証明書の発行を実現し，さらに，公開鍵証明書の発行と分散型台帳での公開を一つのトランザクションとして設計することで，公開鍵証明書の発行の監視を実現している．

A Study on Public Key Certificate Issuance Infrastructures for Consortiums by Sealing Private Keys into Distributed Ledgers

KEITA KUMAGAI[1] SHOHEI KAKEI[1] YOSHIAKI SHIRAISHI[2] SHOICHI SAITO[1]

1. はじめに

Internet of Things（IoT）の進展によるネットワーク接続デバイスの増加に伴い，多種多様なデータを取得できるようになってきている．自治体や民間企業では，これらのデータを活用するためのデータ連携基盤を構築し，その有効活用に取り組んでいる．また，複数の組織がコンソーシアムを組むことで活用できるデータの多様性を向上させる取り組みも進められている [1]．このような多数のデバイスの利用が進むなか，そのデバイスが正規のデバイスであることの保証は信頼できるサービスの構築において重要である．

インターネットにおけるデバイスの真正性の保証に公開鍵基盤が広く利用されている．公開鍵基盤では，信頼できる第三者機関である認証局（Certificate Authority, CA）が公開鍵とその所有者の情報が記載された公開鍵証明書を発行する．この公開鍵証明書を確認することで，その所有者の情報を知ることができる．CA は公開鍵証明書の内容の真正性に責任を負うことから，信頼できる組織により運用される必要がある．インターネットでの利用においては，社会的に信用された企業がサービスとしてパブリック CA を運用している．

パブリックな用途とは対照的に，独自にプライベート CA を運用して，特定の領域内でのみ有効な公開鍵基盤を構築することもできる．多数の公開鍵証明書を発行する場合や独自の枠組みで公開鍵証明書を発行する場合など，外部サービスの利用が適さない場合などに対応できる．文献 [2] の調査によると，自社内での運用やマネージドサービスによる外部委託などの使用が増加傾向にあることが指摘されている．

1 名古屋工業大学
2 神戸大学

本研究では，コンソーシアム型のシステムに適したプライベート CA の構築に向けて，複数の組織で CA を協同運用できる公開鍵証明書発行基盤を提案する．CA は，サービスへのアクセス可否や認可の判断の根拠となるデバイスの真正性の保証を司ることから，特定の組織に依存することなく，コンソーシアムに参加する全ての組織で協同して運用できることが望ましい．一方で，CA の運用には多くのセキュリティリスクが伴うことから，CA の運用に多数の利害関係者が関わればリスクの増加が懸念される．CA のセキュリティリスクの一つに秘密鍵の漏えいがある．CA の秘密鍵は漏えいすれば，その秘密鍵を使って任意の公開鍵証明書を発行できる．そこで本研究では，CA の協同運用における秘密鍵の管理方法に着目した公開鍵証明書発行基盤を提案する．

本提案では，公開鍵証明書発行基盤の構築に分散型台帳技術を用いる．コンソーシアムに参加する組織が分散型台帳ネットワークを構築し，スマートコントラクトを使って公開鍵証明書を発行・管理する．公開鍵証明書を作成するにはスマートコントラクト上で秘密鍵を扱う必要があるが，その内容は全参加者で共有される．秘密鍵の漏えいを防ぐために，秘密鍵を秘匿した状態でスマートコントラクトを実行する必要がある．本稿では，その方法として Intel SGX を使った分散型台帳の実装である Hyperledger Fabric Private Chaincode（FPC）を用いる方法を示す．

2. 関連技術

2.1 公開鍵基盤

公開鍵基盤は，公開鍵とその所有者の情報を紐づける枠組みであり，通信相手が確かにその公開鍵の所有者であることを公開鍵暗号方式に基づいて検証できる仕組みを提供する [3]．公開鍵の所有者の保証は単一信頼点である CA が担っており，CA は公開鍵の所有者を確認したうえでその所有者に公開鍵証明書を発行する．CA はその記載内容を保証するために，自身の秘密鍵で公開鍵証明書に署名を付す．公開鍵の所有者は，この公開鍵証明書を通信相手に提示することで，自身の真正性を証明することができる．

公開鍵証明書のフォーマットは，国際標準化団体である ITU（国際電気通信連合）によって標準化されており，代表的なものに X.509 がある．図 1 に示すように，X.509 の公開鍵証明書は署名前証明書，署名アルゴリズム，デジタル署名で構成されている．署名前証明書には，公開鍵証明書の発行者や主体者の情報，公開鍵などの基本的な情報が記載されており，追加の情報は拡張領域に格納することになっている．

公開鍵証明書はその用途に応じてその内容を分ける必要があるが，その大きな分類として CA 用の公開鍵証明書かそうでないかがある．CA 用の公開鍵証明書は，公開鍵の所有者を保証する以外に，公開鍵証明書の発行・検証にも使用される．そこで，拡張領域の基本制約の拡張内に CA 用の公開鍵証明書であることを表すフラグを立てる．

<table>
<tr><td rowspan="12">署名前証明書</td><td colspan="2">バージョン</td></tr>
<tr><td colspan="2">シリアル番号</td></tr>
<tr><td colspan="2">アルゴリズム識別子</td></tr>
<tr><td colspan="2">発行者</td></tr>
<tr><td rowspan="2">有効期間</td><td>開始時刻</td></tr>
<tr><td>終了時刻</td></tr>
<tr><td colspan="2">主体者</td></tr>
<tr><td rowspan="2">主体者公開鍵情報</td><td>アルゴリズム</td></tr>
<tr><td>主体者公開鍵</td></tr>
<tr><td colspan="2">発行者ユニーク識別子</td></tr>
<tr><td colspan="2">主体者ユニーク識別子</td></tr>
<tr><td rowspan="3">拡張領域</td><td>識別子</td></tr>
<tr><td>重要度</td></tr>
<tr><td>拡張子</td></tr>
<tr><td colspan="3">署名アルゴリズム</td></tr>
<tr><td colspan="3">電子署名</td></tr>
</table>

図 1　X.509 証明書の構造

公開鍵基盤における公開鍵証明書の申請・発行・検証の流れを図 2 に示す．まず，申請において，証明書所有者は自身の鍵ペアを生成し，これに対して証明書発行要求（Certificate Signing Request, CSR）を作成する（Step 1-1）．CSR とは，証明書発行に必要となる情報のことであり，コモンネーム，組織名，申請者の公開鍵等の情報が含まれる．次に，作成された CSR を登録局（Registration Authority, RA）に送信する（Step 1-2）．RA は，証明書所有者の本人確認を行い，証明書発行要求を行った者が本人であることを確認する（Step 1-3）．本人であることが確認できれば，公開鍵証明書の発行に移る．RA は CSR を CA に送信し，公開鍵証明書の作成を依頼する（Step 2-1）．依頼を受けたら，まず，CA は CSR に基づいて署名前証明書を作成する．そして，CA の秘密鍵でこれに署名することで公開鍵証明書を作成し（Step 2-2），この公開鍵証明書を証明書所有者に発行する（Step 2-3）．CA は，発行した公開鍵証明書や証明書失効リスト（Certificate Revocation List, CRL）を公開するリポジトリを管理しており，適宜，リポジトリの更新を行っている．最後に，検証において，証明書所有者は自身の身元を証明するために，CA から発行された公開鍵証明書を証明書利用者に提示する（Step 3-1）．証明書利用者は，必要に応じてリポジトリを参照しながら，自身が信頼する CA から発行された公開鍵証明書であることを検証（Step 3-2）する．検証が成功すれば，証明書を提示した者が提示された証明書に記載されている者であることを信頼できる．

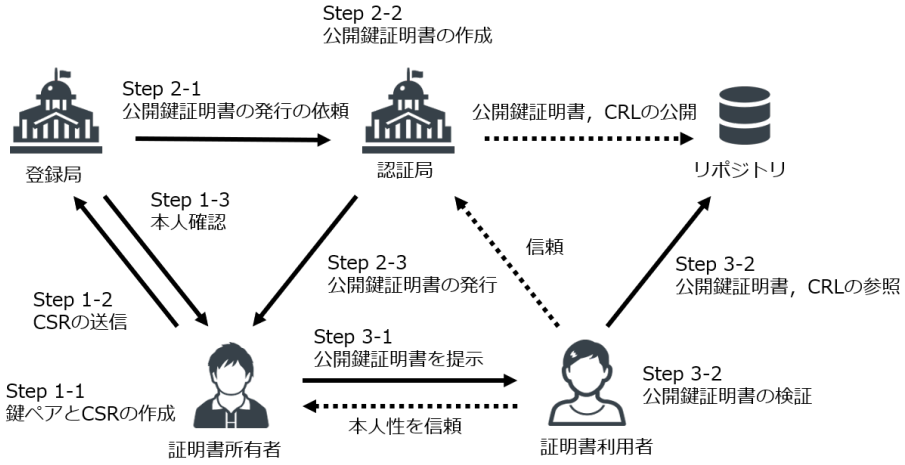

図 2　公開鍵基盤における公開鍵証明書の申請・発行・検証の流れ

公開鍵証明書の署名・検証の処理について図 3 に示す．公開鍵証明書の署名処理は図 3 の左側に示すように，ハッシュ化した署名前証明書を CA の秘密鍵で暗号化することで行われる．つまり，公開鍵証明書に付すデジタル署名は CA しか作成できないこととなるので，公開鍵証明書の内容に対して CA による保証が得られる．一方で，図 3 の右側に示すように，署名前証明書のハッシュ値とデジタル署名を CA の公開鍵で復号した値が一致することを確認することで，検証が行われる．CA の公開鍵証明書は公開されており，誰でも検証処理を行うことができる．公開鍵証明書の内容に対する信頼は，CA が適切に秘密鍵を管理している点に集約される．つまり，CA の秘密鍵が不正に利用されれば，公開鍵証明書の内容を偽ることができるので，秘密鍵の利用は厳重に行われる必要がある．

2.2　Hyperledger Fabric Private Chaincode

Hyperledger Fabric[5] とは，オープンソースのブロックチェーンプラットフォームである Hypyerledger[4] のフレームワークの一つである．Hyperledger Fabric はブロックチェーンの中でも分散型台帳技術に分類される．Hyperledger Fabric では，Peer，Orderer の 2 つのノードで構成されている．クライアントは自身の情報をブロックチェーンネットワークに登録することで，このネットワークに参加することができる．クライアントはあるトランザクションを実行したい場合，そのトランザクションを Peer へ送信する．Peer はクライアントから送られてきたトランザクションの実行と検証，ブロックへの書き込みを行う．トランザクションの実行は複数の Peer で行われ，トランザクションとその実行結果をクライアントへ送信する．クライアントが Peer から送信されたトランザクションと実行結果を集め Orderer へ送信すると，Orderer はクライアントから送られてきたトランザクションを順序付けし，1 つのブロックにまとめ，全ての Peer に送信する．Peer では，トランザクションの検証は事前に定められたポリシーによって実行結果が正しいか判断を行う．ポリシーには過半数が同じ結果といったように各 Peer の実行結果がどれほど一致しているかという基準が定められている．そして，実行結果が正しいと判断した場合に，ブロックへ書き込みを行う．Hyperledger Fabric では，実際にスマートコントラクトを実行するのは Peer である．したがって，クライアントが何を引数とし，どのようなスマートコントラクトを実行し，どのような実行結果が出たかということをクライアントだけでなく Peer も知ることができる．つまり，クライアントにとって他者に知られたくない情報であってもスマートコントラクトを実行すると Peer に知られることになり，この点は Hyperledger Fabric の欠点といえる．

Hyperledger Fabric Private Chaincode[6] は，Hyperledger Fabric をより発展させたもので前述した Hyperledger Fabric の欠点を克服している．欠点を克服するために Intel SGX が使用されている．

Intel SGX とは，データを保護しつつプログラムを実行するための CPU の拡張機能のことである．メモリ上に Enclave と呼ばれる領域を生成することで，データを暗号的に厳重に保護している．

Hyperledger Fabric Private Chaincode では，Peer の内部が Intel SGX で保護されている．Enclave 内でスマートコントラクトは実行されるため，Peer 自身であっても引数，スマートコントラクトの内容，実行結果は知ることができない．

3.　関連研究

公開台帳に公開鍵証明書が発行されたログを記録することで不正に発行された公開鍵証明書を検知するシステムが提案されている．このシステムは，公開鍵証明書を検証する際に公開台帳を確認し，検証対象の公開鍵証明書が記録されているならば正しい証明書と判断し，記録されていな

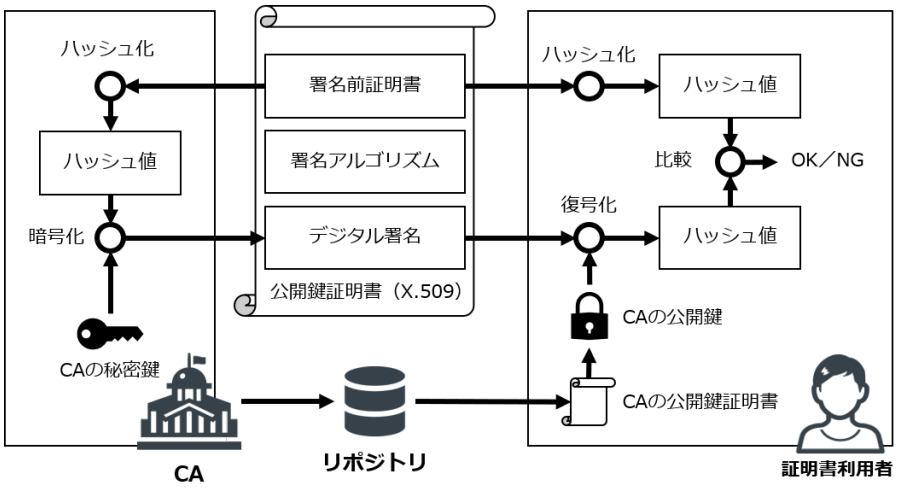

図 3　公開鍵証明書の署名・検証処理

いならば不正な証明書と判断する．

CertChain[7] では，ブロックチェーンを用いた公開鍵証明書の監査システムを構築することで，公開鍵証明書の正当性を保証しようとしている．この手法では，認証局が発行した公開鍵証明書はブロックチェーンに登録される．クライアントがサーバの公開鍵証明書を検証する際にブロックチェーン上にある公開鍵証明書を参照し，ブロックチェーン上に該当する公開鍵証明書が存在するならば，その公開鍵証明書を正しい証明書と判断するという手法である．

PoliCert[8] では，ログベースの公開鍵基盤システムを提案している．クライアントとサーバが通信する際にサーバは自身の公開鍵証明書を CA から受け取り，クライアントに送ることで自身の安全性を示している．PoliCert ではサーバは自身の公開鍵証明書を CA から受け取った後，ログサーバに公開鍵証明書を受け取ったというログの記録を行う．そして，クライアントがサーバの公開鍵証明書を検証する際にログサーバを参照し，該当するログが存在するかどうかによって正しい証明書か否かを判断する．

公開台帳により公開鍵証明書を公開することで，認証局による不正な公開鍵証明書の発行を監視できる．しかし，これは特定の組織が秘密鍵を安全に保管していることが前提であり，本研究で想定するような複数の組織での協同運用を前提とした基盤では，不正の検知ができない．公開台帳による，公開鍵証明書の発行の記録にあわせて，CA の秘密鍵を複数の組織で安全に利用できる必要がある．

4. 提案基盤の設計方針

本研究では，複数の組織で協同運用できる公開鍵証明書発行基盤を提案する．複数の組織で協同運用する単純な方法としては，公開鍵証明書の発行に必要な秘密鍵を各組織の管理者で共有して，それぞれの責任のもとで公開鍵証明書を発行する方法がある．しかし，各管理者に強い信頼を仮定する必要があり，悪意のある管理者が不正に秘密鍵を使用すれば秘密裏に公開鍵証明書を発行できることからこの方法は現実的でない．つまり，複数の組織が公開鍵証明書発行基盤を協同運用するには，必要に応じて各組織の管理者主導で公開鍵証明書を発行できる一方で，基盤が適正に運用されるように不正な公開鍵証明書の発行は漏れなく検知できる必要がある．そこで本提案基盤は，以下の要件を満たすように設計する．

- 要件 1：特定の組織に依存せずに公開鍵証明書を発行できること
- 要件 2：発行された公開鍵証明書は全て記録に残ること
- 要件 3：公開鍵証明書を発行した事実を消去できないこと

要件 1 に関して，公開鍵証明書を作成する処理を特定のデバイス上で実行するのではなく，複数のピアが合意しながら処理を実行するスマートコントラクト上で実行することで実現する．要件 2 に関して，秘密鍵に直接アクセスできれば公開鍵証明書を作成できるので，スマートコントラクトからしか秘密鍵を利用できないようにすることで実現する．公開鍵証明書の作成に利用する特定の秘密鍵を特定のスマートコントラクトでのみ利用できるようにすることを本研究では「封入」と呼ぶ．要件 3 に関して，作成された公開鍵証明書はスマートコントラクトの処理結果として出力されるとともに，分散型台帳に記録することで実現する．

以上に示したように，スマートコントラクトを用いることで，秘密鍵を使った公開鍵証明書の作成とその証明書の分散型台帳への記録を単一のトランザクションとして設計する．その結果，分散型台帳を確認することで公開鍵証明書を発行した事実を確認できるようになるが，検証者が正しく分散型台帳に記録されている内容を確認できる必要がある．そこで，以下の要件を追加する．

- 要件 4：公開鍵証明書が発行された事実を検証者が確かに確認できること

本研究では，ブロックチェーン技術で上記 4 要件を満たす公開鍵証明書発行基盤を設計する．ブロックチェーン技術はパブリック型とプライベート型に分類できるが，本提案基盤ではプライベート型を利用する．その理由として二点挙げられる．一つ目は，コンソーシアムに参加する組織間だけで運用可能な基盤を設計できることが挙げられる．ブロックチェーン技術を使って CA の機能を実装することから，その機能にアクセスできる主体を限定する必要がある．パブリック型の場合，不特定多数からのアクセスを許してしまうことから，プライベート型で提案基盤を実現することが望ましい．二つ目は，不特定多数の参加者で形成されるパブリック型では，トランザクションのファイナリティが得られない点が挙げられる．ファイナリティが得られないことによって，公開鍵証明書の発行事実が取り消される潜在的リスクが生じる．そこで，決められた参加者間で合意しながらトランザクションを確定させていくことができるプライベート型の利用が望ましい．

以上より，本研究ではプライベート型のブロックチェーン技術の一つである Hyperledger Fabric を利用する．さらに，Enclave によりスマートコントラクトの処理内容と分散型台帳に記録された情報を秘匿可能な FPC を用いることで，上記 4 つの要件を満たす公開鍵発行基盤を提案する．

5. 提案基盤の設計

5.1 提案基盤の概要

提案基盤の構成を図 4 に示す．FPC ネットワーク上のスマートコントラクトで CA の機能を実装し，RA に相当する複数の組織が Peer を管理しており，この Peer が FPC ネットワークを形成している．各 RA は，Peer を介してスマートコントラクトを実行することで CA の機能を利用できる．FPC ネットワークにおいて，各 Peer が持つ台帳は同期されており，全ての台帳に同じ情報が書き込まれている．各組織は Intel SGX を利用可能なデバイス上で Peer を動作させる．それにより，Peer が実行するスマートコントラクトの処理内容と入出力，台帳への入出力が秘匿される．

提案基盤で管理される情報を表 1 に示す．台帳は Key-Value 型で情報を保持するようになっており，提案基盤の実行に必要な情報や発行した公開鍵証明書，失効した公開鍵証明書の情報などが格納されている．台帳には，CA の秘密鍵 sk_{CA} とそれに対応する公開鍵証明書のシリアル番号のリスト $List_{CACerts}$，検証処理で使用する秘密鍵 sk_{Audit} とそれに対応する公開鍵証明書のシリアル番号のリストが格納できるようになっている．加えて，公開鍵証明書はそのシリアル番号をキーにして管理されるようになっており，割り当てるシリアル番号のカウンタ値も管理されている．また，失効した公開鍵証明書はキー `crl` で，そのシリアル番号のリストが管理されている．

表 1 台帳で管理されるデータ

Key	Value
`CAKey`	CA の秘密鍵 sk_{CA}
`CACerts`	CAKey に対応する公開鍵証明書のシリアル番号のリスト $List_{CACerts}$
`AuditKey`	公開鍵証明書の登録確認に使用する秘密鍵 sk_{Audit}
`AuditCerts`	AuditKey に対応する公開鍵証明書のシリアル番号のリスト $List_{AuditCerts}$
`SerialNumber`	カウンタ値
公開鍵証明書のシリアル番号	公開鍵証明書
`crl`	失効した公開鍵証明書のシリアル番号のリスト

5.2 エンティティ

本提案基盤に関係するエンティティを以下に示す．

- 証明書所有者（End Entity, EE）: 証明書所有者は，提案基盤に対して証明書発行要求を行うエンティティである．
- 証明書検証者（Verifier, V）: 証明書検証は，証明書を検証するエンティティである．
- 証明書発行者（Issuer, I）: 証明書発行者は証明書を発行するためにスマートコントラクトを呼び出すエンティティである．
- 基盤管理者（Manager, M）: 基盤管理者は，提案基盤をセットアップするエンティティである．FPC ネットワークの立ち上げ，スマートコントラクトのデプロイ，公開鍵証明書の作成に使用する鍵ペアの作成と封入を行う．セットアップ後は，証明書発行者として振る舞う．
- スマートコントラクト（Smart Contract, SC）: スマートコントラクトは，ブロックチェーンで動くプログラムのことであり，台帳上のデータに対して読み書きを行うことができる．FPC ではスマートコントラクトは事前に作成しなければならず，FPC ネットワークがの起動後は，スマートコントラクトを書き換えることはできない．
- 台帳（Ledger, L）: 台帳は，データを格納する場所である．

5.3 トランザクション

本提案基盤ではセットアップ，証明書発行，証明書検証，証明書失効の 4 つのトランザクションがある．

5.3.1 セットアップ

セットアップでは，基盤管理者主導で FPC ネットワークの立ち上げと各組織で合意されたスマートコントラクトのデプロイを行ったうえで，図 5 に示すシーケンスを実行

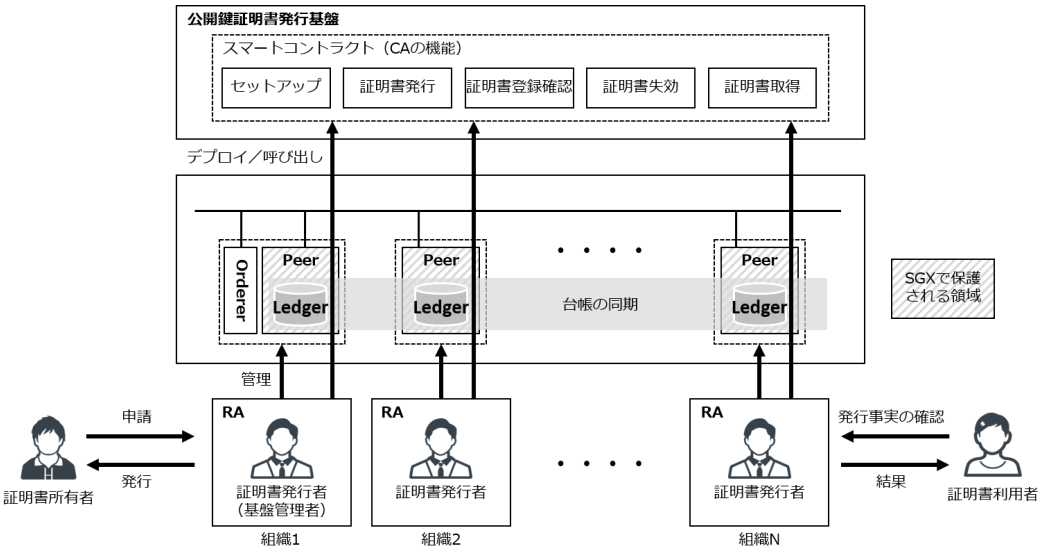

図 4　提案基盤の構成

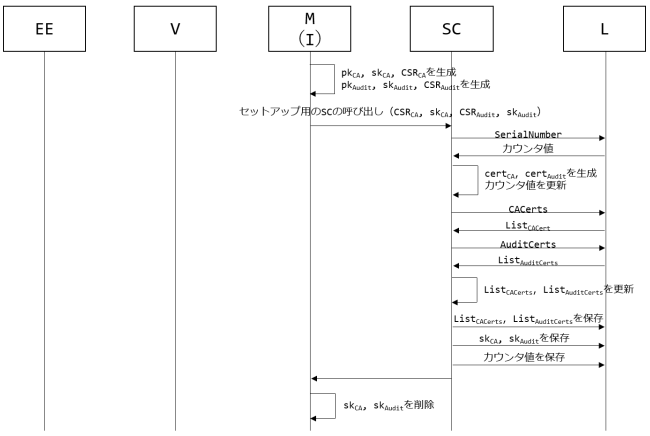

図 5　セットアップのシーケンス

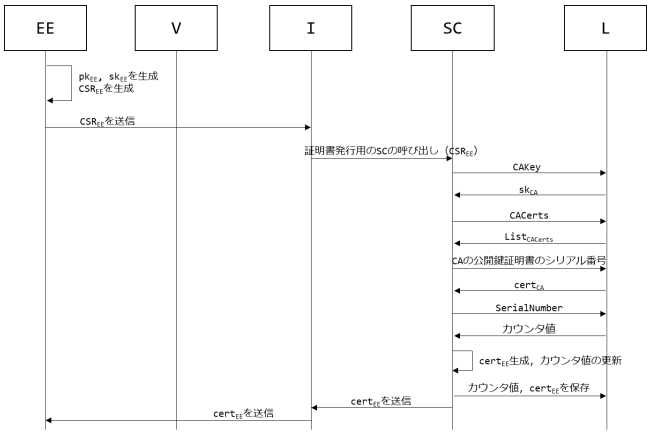

図 6　証明書発行のシーケンス

する．なお，FPC ネットワークの立ち上げやスマートコントラクトのデプロイに関しては本稿では割愛する．

セットアップでは，まず基盤管理者は公開鍵証明書の作成に使用する公開鍵 pk_{CA} と秘密鍵 sk_{CA} を生成し，これらを使って証明書署名要求 CSR_{CA} を生成する．また，公開鍵証明書の登録確認用に公開鍵 pk_{Audit} と秘密鍵 sk_{Audit} を生成し，これを使って証明書署名要求 CSR_{Audit} を生成する．そして，CSR_{CA} と CSR_{Audit} に対応する公開鍵証明書 $cert_{CA}$ と $cert_{Audit}$ の作成と sk_{CA} と sk_{Audit} を封入するスマートコントラクトを実行する．$cert_{Audit}$ は sk_{CA} で署名することで，$cert_{CA}$ を基点に $cert_{Audit}$ を検証できるようにしている．一連の処理が完了したら，最後に sk_{CA} と sk_{Audit} ををを削除する．

5.3.2　証明書発行

証明書発行のシーケンスを図 6 に示す．証明書発行では，まず証明書所有者が公開鍵 pk_{EE} と秘密鍵 sk_{EE} を生成し，これらから証明書署名要求 CSR_{EE} を生成する．そして，証明書の発行を申請するためにいずれかの証明書発行者に CSR_{EE} を送信する．CSR_{EE} を受けた証明書発行者は証明書発行を行うスマートコントラクトを呼び出す．スマートコントラクトは台帳に格納されている sk_{CA} と $cert_{CA}$ を使って，CSR_{EE} の内容に基づいて公開鍵証明書 $cert_{EE}$ を生成する．生成される公開鍵証明書には，台帳で管理されるシリアル番号のカウンタ値が割り当てられ，そのシリアル番号をキーとして台帳に $cert_{EE}$ が格納される．最後に，証明書発行者はスマートコントラクトの処理結果として $cert_{EE}$ を受け取り，これを証明書所有者に送信する．

5.3.3　証明書検証

証明書検証の処理のシーケンスを図 7 に示す．証明書検証では，証明書検証者が任意の証明書発行者に検証したい公開鍵証明書 $cert'_{EE}$ と乱数 $r1$ を送信することで検証を要求する．証明書発行者は受け取った $cert'_{EE}$ と $r1$ を引数に証明書登録確認用のスマートコントラクトを呼び出す．スマートコントラクトでは，$cert'_{EE}$ のシリアル番号をキーに台帳から $cert_{EE}$ を取り出し，データが一致するかを比較する．また，$cert_{EE}$ が失効されていないこともあわせて

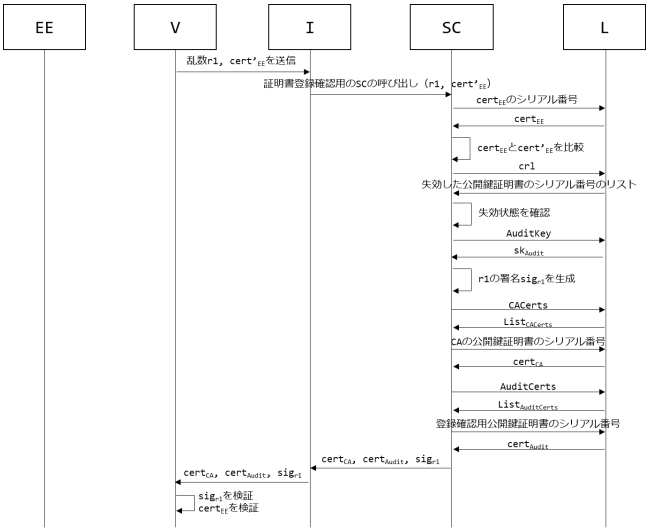

図 7　証明書検証のシーケンス

確認する．$cert'_{EE}$ と一致する $cert_{EE}$ が格納されていることと失効されていないことが確認できれば，その確認の証拠として sk_{Audit} で $r1$ の署名 sig_{r1} を作成する．なお，この確認が失敗すればここで処理を中断する．次に，$cert_{CA}$ と $cert_{Audit}$ を台帳から取り出し，sig_{r1} とあわせて，証明書発行者に送信する．証明書発行者は，これらを証明書検証者に送信し，証明書検証者は sig_{r1} と $cert_{EE}$ の検証を行う．この検証に成功すれば $cert_{EE}$ が正しく提案基盤で発行されたことを確認できる．

5.3.4　証明書失効

証明書失効の処理のシーケンスを図 8 に示す．まず，証明書所有者は失効したい公開鍵証明書 $cert_{EE}$ のシリアル番号を任意の証明書発行者に送信することで，証明書失効を要求する．証明書発行者は，受け取ったシリアル番号をキーに台帳から $cert_{EE}$ を取り出す．$cert_{EE}$ を取り出すことができれば，失効要求が $cert_{EE}$ の所有者本人によるものであることを確認するために，乱数 $r2$ の署名を証明書所有者に要求する．なお，$cert_{EE}$ が台帳に格納されていなければ，ここで処理を中断する．証明書所有者は，sk_{EE} で $r2$ の署名 sig_{r2} を生成し，これを証明書発行者に送信する．証明書発行者は，sig_{r2} の検証が成功すれば，失効の申請は $cert_{EE}$ の所有者本人のものであると確認できる．なお，この署名の検証が失敗すれば，ここで処理を中断する．次に，証明書発行者は $cert_{EE}$ を失効するために，$cert_{EE}$ のシリアル番号を引数に証明書失効用のスマートコントラクトを呼び出す．スマートコントラクトでは，$cert_{EE}$ が台帳に登録されていることを確認したうえで，crl のリストに $cert_{EE}$ のシリアル番号を追記する．

6.　定性評価

本章では，設計した公開鍵証明書発行基盤が設計方針で示した 4 つの要件を満たしているかを評価する．

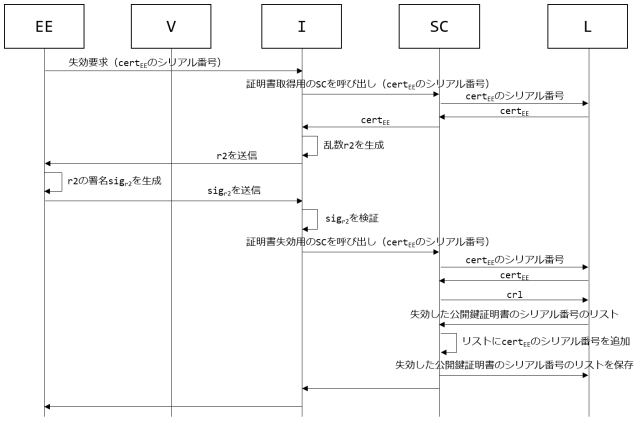

図 8　証明書失効のシーケンス

6.1　要件 1 に対する評価

要件 1 は「特定の組織に依存せずに公開鍵証明書を発行できること」である．公開鍵証明書の発行処理が特定の組織に依存するようになっていれば，その組織が信頼点となる．協同運用する基盤上で各組織主導で公開鍵証明書の発行を行うには，このような信頼点は障害点になりうる．

公開鍵証明書の発行には CA の秘密鍵が必要であるが，セットアップシーケンスで示すようにその秘密鍵を FPC ネットワーク上の台帳にのみ保存されるように設計している．つまり，その秘密鍵の利用を証明書発行時にのみ利用できるようにスマートコントラクトを実装することで，FPC ネットワークの参加者ならば誰でも実行できるようになっている．したがって，提案基盤は要件 1 を満たす．

6.2　要件 2 に対する評価

要件 2 は「発行された公開鍵証明書は全て記録に残ること」である．適正な公開鍵証明書発行基盤を協同運用するには，発行された公開鍵証明書は後から確認可能であることが求められる．どのような公開鍵証明書が発行されたかを後から確認できることで，公開鍵証明書を不正に発行することに対する抑止力となる．

公開鍵証明書の作成には CA の秘密鍵が必要であり，その秘密鍵は台帳にのみ保存されるようになっている．したがって，公開鍵証明書を作成するにはスマートコントラクトを実行しなければならない．証明書発行シーケンスに示すように，公開鍵証明書の作成と同時にその公開鍵証明書が台帳に記録されるようになっており，発行と記録を一つのトランザクションとして実現している．これにより，提案基盤で発行された公開鍵証明書は全て台帳に記録されるようになっている．一方で，セットアップシーケンスに示すように，基盤管理者は適切に CA の秘密鍵を削除することが求められる．削除せずに CA の秘密鍵を持っている場合，基盤管理者は秘密裏に公開鍵証明書を作成できる．しかしながら，スマートコントラクトを実行せずに作成した公開鍵証明書は台帳に記録されないので，正規に発行さ

れた公開鍵証明書のみが記録に残るようになっている．また，証明書検証シーケンスにおいて，公開鍵証明書が台帳に登録されているかを確認しており，台帳に記録されないような秘密裏に作成された公開鍵証明書は証明書検証シーケンスで検知できる．したがって，提案基盤は要件 2 を満たす．

6.3 要件 3 に対する評価

要件 3 は「公開鍵証明書を発行した事実を消去できないこと」である．公開鍵証明書の発行事実を消去できると，不都合な発行事実を後から消すことができる．公開鍵証明書の発行事実は台帳への公開鍵証明書の保存として記録されているので，公開鍵証明書の発行事実の消去は，台帳に保存されている公開鍵証明書の消去を意味する．

分散型台帳技術において，スマートコントラクトを介してのみ台帳の内容を更新できるようになっている．つまり，公開鍵証明書の発行事実を消去するには，不正に台帳内の情報を書き換える必要があるが，各組織が管理する台帳を書き換えることは事実上不可能である．また，台帳に記録された公開鍵証明書を削除するスマートコントラクトは無く，正規の機能を悪用して公開鍵証明書を削除することもできない．したがって，提案基盤は要件 3 を満たす．

6.4 要件 4 に対する評価

要件 4 は「公開鍵証明書が発行された事実を証明書検証者が確かに確認できること」である．証明書の検証では，証明書検証シーケンスに示すように，証明書の検証は CA による署名が付されていることと，その証明書が台帳に記録されていることの両方を確認する．CA による署名が付されているかの確認は CA の公開鍵証明書で検証すればよいが，検証対象の公開鍵証明書が台帳に記録されているかは証明書発行者に問い合わせるしかない．要件 2 で述べたように，台帳に記録されない公開鍵証明書が作成されるリスクがあることから，単純に証明書発行者に発行事実の有無を問い合わせだけでなく，確かに発行事実が台帳に記録されていることを証明書検証者が確認できる必要がある．

そこで，証明書検証シーケンスに示すように，公開鍵証明書が台帳に登録されていることを表す署名をスマートコントラクト上で作成することで，証明書検証者が発行事実を確認できるようにしている．この署名に使用する秘密鍵は，CA の秘密鍵と同様にセットアップシーケンスにおいて，台帳内に封入されるので，全ての証明書発行者でこの検証の問い合わせに応えることができる．ただし，基盤管理者は秘密鍵を削除せずに隠し持っているリスクがあるので，証明書検証者は基盤管理者以外の証明書発行者に証明書検証を依頼することで，要件 4 を満たすことができる．

7. おわりに

本稿では，複数の組織で協同運用できる公開鍵証明書発行基盤を提案した．認証局による公開鍵証明書の不正な発行の検知において，公開台帳を使う手法が知られているが，秘密鍵の利用が課題となる協同運用を目的とした認証局には適さない．そこで本研究では，分散型台帳に封入された CA の秘密鍵を利用するスマートコントラクトを組織間で共有することで，複数の組織で協同運用可能な公開鍵証明書発行基盤を提案した．協同運用における 4 要件について評価を行い，本提案基盤が要件を満たしていることを確認した．

FPC のスマートコントラクトは C 言語を使用して開発できる．ただし，全ての機能を使えるわけではなく，非決定的なアルゴリズムを実装することはできない．現在，提案基盤の各シーケンスの実装に必要な個々の機能をスマートコントラクトで実装しているところである．これらの機能を結合し，提案基盤の性能を評価することが今後の課題である．

参考文献

[1] 統合ビッグデータ研究センター：研究紹介，国立研究開発法人情報通信研究機構（オンライン），入手先 ⟨https://www2.nict.go.jp/bidal/research.html⟩（参照 2021-05-10）.

[2] Entrust：2020 世界の PKI および IoT 動向調査，Entrust(オンライン) 入手先 ⟨https://www.entrust.com/lp/ja/global-pki-iot-trends-study⟩（参照 2021-05-10）.

[3] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R. and Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC5280, 2008.

[4] Hyperledger：Hyperledger – Open Source Blockchain Technologies，The Linux Foundation（オンライン），入手先 ⟨https://www.hyperledger.org/⟩（参照 2021-05-10）.

[5] Hyperledger：Hyperledger Fabric – Hyperledger，The Linux Foundation（オンライン），入手先 ⟨https://www.hyperledger.org/use/fabric⟩（参照 2021-05-10）.

[6] Brandenburger, M., Cachin, C., Kapitza, R. and Sorniotti, A.：Hyperledger Fabric Private Chaincode – Hyperledger Lab，Hyperledger（オンライン），入手先 ⟨https://labs.hyperledger.org/labs/fabric-private-chaincode.html⟩（参照 2021-05-10）.

[7] Chen, J., Yao, S., Yuan, Q., He, K., and Ji, S. and Du, R., CertChain: Public and efficient certificate audit based on blockchain for TLS connections, *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, pp. 2060–2068, 2018.

[8] Szalachowski, P., Matsumoto, S. and Perrig, A.: Policert: Secure and flexible TLS certificate management, *Proc. of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 406–417, 2014.