分散型認証基盤に向けたスマートコントラクトを用いた相互認証方式の提案

掛井 将平[1]　白石 善明[2,3]　毛利 公美[4]　中村 徹[3]
橋本 真幸[3]　齋藤 彰一[1]

概要：IoT デバイスが生成するデータやその分析結果を様々な主体が取引する IoT エコシステムにおいて，主体の本人性を保証する公開鍵基盤（Public Key Infrastructure，PKI）は主体間の信頼関係を構築し，データ源の保証や取引の否認防止の観点から IoT エコシステムの信頼性の向上に寄与する．しかし，単一信頼点である認証局が本人性を保証するので，認証局が侵害されれば全ての信頼関係が損なわれることとなる．本研究では，認証局を各主体に分散しつつ，認証局間で適切な信頼関係が構築されていることを保証する分散型認証基盤（メタ PKI）の構築を目指している．認証局間の信頼関係の構築には相互認証が利用できるが，各認証局の独自の判断に頼るだけでは適切な信頼関係が構築されていることを保証できない．本稿では，スマートコントラクトを利用して統一的な基準で信頼関係を構築するための相互認証方式を提案し，提案方式が認証局の乗っ取りやなりすましに対して安全であることを示す．

キーワード：公開鍵基盤，認証基盤，分散台帳技術，スマートコントラクト，IoT

Proposal of Cross-Certification using Smart-Contract toward Distributed Authentication Infrastructure

Shohei Kakei[1]　Yoshiaki Shiraishi[2,3]　Masami Mohri[4]　Toru Nakamura[3]
Masayuki Hashimoto[3]　Shoichi Saito[1]

Abstract: In IoT ecosystems where various entities deal with IoT data and their analysis results, a public key infrastructure (PKI) constructs trust relationships between entities and improves reliability of the ecosystem from the viewpoint of the data source assurance or prevention of non-repudiation of transactions. However, since a certificate authority (CA), which is a single trust point, guarantees all the identities, if the CA is infringed, all the trust relationships will be impaired. This research aims to construct a distributed authentication infrastructure (Meta-PKI) composed of multiple CAs with mutual trust relationships using cross-certification. However, the appropriate trust relationships are not guaranteed only by relying on the cross-certification according to the CA's judgment. This paper proposes a cross-certification method using smart-contract for constructing the trust relationships based on uniform standards and shows that the proposed method is safe against takeover and spoofing of CAs.

Keywords: Public Key Infrastructure, Authentication Infrastructure, Distributed Ledger Technology, Smart-Contract, IoT

1. はじめに

Internet of Things（IoT）の発展により，多様なデータを取得できる環境が整いつつある一方で，取得されたデータのさらなる有効活用が期待されている．データの所有者や分析者，利用者がお互いに関わり合いながら共存共栄を目指す IoT エコシステムのように，様々な主体がオープンにデータを取り引きできる環境が構築できれば，データのさらなる有効活用を後押しすると期待できる．

データの取引の信頼性を確保するには，データ源の保証や取引の否認防止など，誰がどのデータを取引したのかの保証が考えられる．そのため，取引相手の本人性の保証は，信頼できるデータ取引の基盤を支える要素技術の一つとなる．

主体の本人性を保証する方法として，公開鍵基盤（Public Key Infrastructure, PKI）[1]が広く利用されている．公開鍵基盤は，信頼できる第三者機関である認証局が主体と公開鍵の関連を証明書により保証する基盤であり，電子署名による通信相手の確認に利用されている．認証局を絶対的に信頼することが前提なので，認証局は外部不正／内部不正に対抗できるような高度なセキュリティ対策や人材育成など，技術面や運用面での対策が必要となる．しかしながら，認証局へのクラッキング[2][3]や認証局の不適切な運用[4]により不正な証明書が発行された事例が発生している．

公開鍵基盤による主体間の信頼関係の形成に関して，ある主体 A が信頼する認証局が別の主体 B の本人性を保証している場合，主体 A は主体 B の本人性を信頼する．さらに，別の主体 C／主体 D が認証局から本人性の保証を受け，公開鍵基盤に参加することで，主体 A の信頼関係は主体 C／主体 D へと拡がっていく．このように，認証局には多数の主体の信頼点が集約されていく．

1　名古屋工業大学
Nagoya Institute of Technology
2　神戸大学
Kobe University
3　株式会社国際電気通信基礎技術研究所
Advanced Telecommunications Research Institute International
4　岐阜大学
Gifu University

インターネットのサーバ認証において，主に，主体 A はサービス利用者，主体 B／主体 C／主体 D はサービス提供者であり，サービス利用者はサービス提供者の本人性を信頼してサービスを利用する．認証局を単一信頼点とすることで，サービス提供者はサービス利用者との直接のやり取りなしに信頼関係を構築できる．しかしながら，認証局から不正な証明書が発行された場合，悪意のある主体 X を正規のサービス提供者と見分けることができない．その結果，認証局を基点として形成された全ての主体間の信頼関係が損なわれることとなる．

本研究では，認証局を各サービス提供者に分散させる分散型の認証基盤の構築を目指す．各サービス提供者に分散された信頼点の接続には「相互認証」が利用できる．しかしながら，対向の認証局の評価と相互認証の実施は属人的なものなので，各サービス提供者が各々の判断で相互認証するだけでは，システム全体として適切に信頼関係が構築されていることを検証できない．

本稿では，この属人的な作業を自動化するために，スマートコントラクトを用いた相互認証方式を提案する．スマートコントラクトは，分散台帳技術において，分散台帳に格納された情報や入力値に基づいて分散台帳への操作を制御する機能である．各サービス提供者が統一された規則で相互認証を実行できるように，相互認証を行う枠組みをスマートコントラクトで実現する．

以下，2 章では提案手法における要素技術である公開鍵基盤と分散台帳技術について述べる．3 章では，分散台帳技術を用いた公開鍵基盤に関する先行研究を述べる．そして，本研究が目指す分散型認証基盤である Meta-PKI の概要を 4 章で述べ，提案手法であるスマートコントラクトを用いた相互認証方式を 5 章で述べる．そして，6 章で提案手法の安全性を述べ，最後に 7 章でまとめる．

2. 要素技術

2.1 公開鍵基盤

公開鍵基盤[1]は，信頼できる第三者機関である認証局（Certificate Authority，CA）が識別名（Distinguished Names, DN）[5]で識別される主体と公開鍵の関連を公開鍵証明書により保証する基盤である．この公開鍵証明書の用途として，サーバ証明書としてサーバにインストールされ，クライアントに対するサーバの本人性の証明に利用される．

認証局により形成される信頼関係を図 1 に示す．CA は Trustee から証明書署名要求（Certificate Signing Request, CSR）[6]を受け取ると，Trustee の本人性を確認したうえで，Trustee の公開鍵に対する公開鍵証明書を作成する．そして，この証明書に対して CA の秘密鍵で署名し，発行することで Trustee の本人性を保証する．Trustor は本人性を検証したい Trustee から公開鍵証明書を受け取り，その公開鍵証明書に Trustor が信頼する CA の署名が付いていることと，Trustee

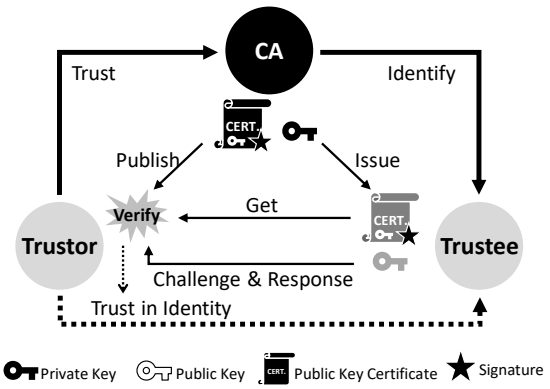

図 1　認証局により形成される信頼関係

Figure 1　Trust Relationships constructed by Certificate Authority.

がその公開鍵証明書に対応する秘密鍵を持っていることを確認できれば，Trustee の本人性を信頼する．公開鍵証明書に対応する秘密鍵を持っているかどうかを確認する手法としてチャレンジ＆レスポンス方式がある．

認証局が複数存在するときのアーキテクチャ[7]として階層型モデルを図 2 に，メッシュ型モデルを図 3 に示す．認証局が別の認証局を信頼し，その信頼関係を辿って，Trustor と Trustee の信頼関係が構築される．

階層型モデルは，上位の CA が下位の CA に対する信頼点となり，信頼関係を拡げていくモデルである．最上位の CA を Root-CA，下位の CA を Sub-CA と表す．Trustor は Root-CA を信頼するだけで，Trustee は任意の Sub-CA から公開鍵証明書の発行を受けるだけで，Trustor と Trustee との信頼関係を構築できる．

メッシュ型モデルでは，Root-CA のような単一信頼点なしに CA が信頼関係を直接構築して，信頼関係を拡げていくモデルである．図 3 では，Trustor は CA_A を信頼しており，CA_A は CA_B を信頼している．信頼関係を辿ることで，Trustor は $Trustee_A$ と $Trustee_B$ との信頼関係が構築されていることが分かる．一方で，CA_C は CA_B を信頼しているが，CA_B は CA_C を信頼していないので，Trustor から $Trustee_C$ への信頼関係は構築されていない．本モデルでは，階層型モデルのような単一信頼点が存在しないので，Trustor との信頼関係を構築するには，Trustor が信頼する CA との信頼関係の構築が必要である．

信頼関係を拡げる方法には，対向の認証局の公開鍵に対して公開鍵証明書を発行する「相互認証」を行う．図 2 の例では，Root-CA が Sub-CA に対して相互認証を，図 3 の例では，CA_B が CA_A と CA_C に対して相互認証を行っている．

階層型モデルとメッシュ型モデルの比較を表 1 に示す．階層型モデルでは，Root-CA が単一信頼点であるので，Root-CA と信頼関係を構築するだけで，その他の CA へと信頼関係を拡げることができる．一方で，メッシュ型モデ

ルでは，単一の信頼点が存在するわけではないので，個別に信頼関係を結ぶこととなり，信頼関係の拡張の容易性は階層型モデルの方が高くなる．しかしながら，階層型モデルは単一信頼点をもつので，メッシュ型モデルと比較して信頼点への依存度が高く，Root-CA で不正が発生すると，システム全体の信頼性が損なわれることとなる．

インターネットにおいては，信頼関係の拡張が容易なことから認証局は階層型モデルで構成されている．ベンダーのサポートにより，主要な Root-CA の公開鍵証明書が信頼できるものとしてブラウザにインストールされており，サーバは主要な Root-CA を上位に持つ Sub-CA から公開鍵証明書の発行を受けているので，クライアントは初めてアクセスするサーバとも信頼関係を構築できる．

2.2 分散台帳技術

分散台帳技術[8]は，分散型のキーバリュー型データベースである分散台帳を中央管理者や集中型のストレージなしに管理する技術である．分散台帳ネットワークへの参加者全員が同じ内容の分散台帳を所持しており，その参加者全員で分散台帳の管理を行う．分散台帳へのデータの書き込みは参加者間で合意された後に行われ，後から合意内容を変更できない．

分散台帳に書き込まれるデータはブロックという単位で管理されている．そのブロックは暗号学的ハッシュ関数で計算された前ブロックのハッシュ値を含むことで，ブロックが連鎖構造を持つ．そのため，ブロックの改ざんや偽造は，ハッシュ値の不一致による連鎖構造の途絶として検知できる．

分散台帳技術の概要を図 4 に示す．ブロック B_n は，前のブロック B_{n-1} のハッシュ値 H_{n-1} とブロック･データ D_n を含む．D_n は，分散台帳への書き込みを示すトランザクション T^n をまとめたものが含まれている．ブロックのデータ構造は，分散台帳技術の実装に応じて異なる．本研究では，Hyperledger Fabric[9]を対象とする．

スマートコントラクトは，分散台帳に対するデータの読み書きのためのインタフェースであり，その処理内容は事前に定義される．分散台帳への書き込み処理はトランザクションとして，分散台帳ネットワークの各参加者により検証されたうえで実行される．

例えば，数値演算のように特定の入力を受け取りその演算結果を分散台帳に書き込むようなスマートコントラクトを定義した場合，任意の演算結果を分散台帳に書き込むことはできず，分散台帳ネットワークの各参加者により入力値から計算された値が分散台帳に格納される．

処理の実行の可否条件を事前に定義することで，一連の処理を契約の条件確認と履行とみなして自動で実行させることができる．本研究では，このスマートコントラクトの特徴を相互認証に利用する．

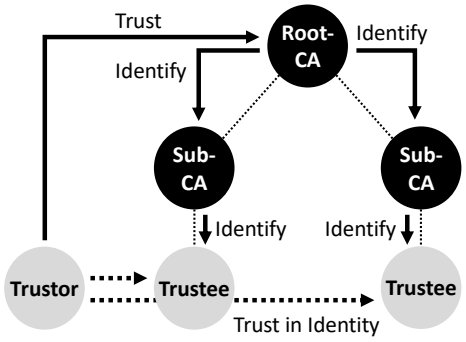

図 2　認証局の階層型モデル

Figure 2　Hierarchical models of Certificate Authority.

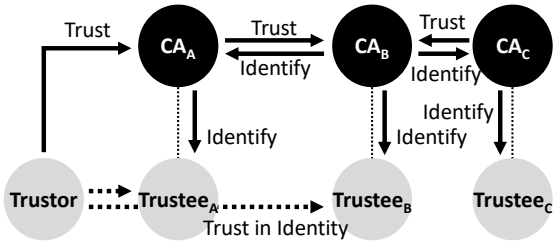

図 3　認証局のメッシュ型モデル

Figure 3　Mesh models of Certificate Authority.

表 1　階層型モデルとメッシュ型モデルの比較

Table 1　Comparison between hierarchical models and mesh models.

	Hierarchy-Model	Mesh-Model
信頼関係の拡張容易性	高	低
信頼点への依存度	高	低

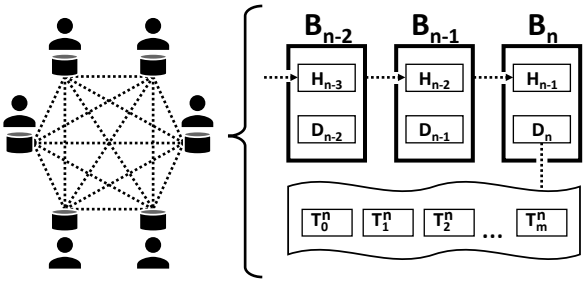

図 4　分散台帳技術の概要

Figure 4　An overview of Distributed Ledger Technology.

3. 先行研究

3.1 Public Key Infrastructure と分散台帳技術

認証局は申請に応じて公開鍵証明書を発行するだけであり，申請者の意図は考慮されない．例えば，クライアントを騙す意図で，正規のサーバにインストールされた公開鍵証明書の DN（例えば，xxx-service.co.jp）に似た DN（例えば，xxx-service.net）で公開鍵証明書を申請することができる．また，1 章で挙げた認証局の内部不正の事例[4]や申請者へのなりすましにより同名の DN を持つ公開鍵証明書を

発行するようなことも考えられる．しかしながら，不正な公開鍵証明書が発行されたことを正規の申請者が直接検知する手段はない．このような状況をうけて，発行された公開鍵証明書の透明性を図る Certificate Transparency（CT）[10]が提案されている．CT は，認証局が公開鍵証明書を発行したことをログとしてログサーバに記録し，公開する．公開鍵証明書の所有者は，ログを照会することで，不正な発行が行われていないかを確認できる．

CT は，不正な公開鍵証明書の発行を不特定多数の主体により監視するための技術であるが，特定の機関におけるログサーバの運用が必要となる．Madala ら[11]は，認証局が公開鍵証明書を発行するときに分散台帳への登録を行う手法を提案している．分散台帳に格納されたデータは第三者が確認できる特徴を CT に利用している．

3.2 Pretty Good Privacy と分散台帳技術

公開鍵暗号を利用したシステムに Pretty Good Privacy（PGP）[12]がある．PGP は，中央集権的な機関が存在せず，鍵ペアを所有する各個人が相手の公開鍵に対して署名することで信頼関係を構築する．さらに，信頼する個人が別の個人を信頼する場合に，その個人へと信頼関係が拡張される．PGP では，このように，特定の認証局なしに信用の輪（web of trust）によって信頼関係が拡張される．公開鍵証明書の作成に関して，対面での受け渡しや両者が信頼できる輸送手段などで公開鍵を受け取り，証明書を作成し署名するため，PKI における CA のような鍵ペアの所有者を保証する単一信頼点が存在しないことが特徴である．また，インターネットにおいて公開鍵を受け渡しする基盤として，事前に公開鍵を keyserver と呼ばれる公開鍵の保管庫に登録しておく方法がある．事前に公開鍵のハッシュ値を相手から受け取り，そのハッシュ値に該当する公開鍵を keyserver から取得する．以上のように，単一信頼点なしに公開鍵の所有者の正当性を確認できるが，公開鍵証明書作成時の公開鍵の所有者の確認は，個人で適切に行うことが求められる．

PGP は，keyserver を使った公開鍵の配布ができるが，特定のサーバで公開鍵が管理されるので，当該サーバに求められる信頼性が高まる．Yakubov ら[13]は，サーバがダウンしているときは公開鍵を取得できないことや，keyserver への公開鍵の取得のリクエストが中間者攻撃によりすり替えられ偽の公開鍵が返されることなどを指摘しており，分散台帳技術を用いた keyserver を提案している．また，Al-Bassam[14]は，PGP の公開鍵と任意の属性を分散台帳で管理する手法を提案している．

3.3 先行研究との比較

先行研究では，多数の主体に台帳を分散して安全に管理できる分散台帳技術の特徴を利用して，PKI や PGP におけるデータ管理の分散化が行われている．特定の信頼点に依存しないデータ管理とすることで，データの改ざんの防止やシステムの可用性の向上が行われている．

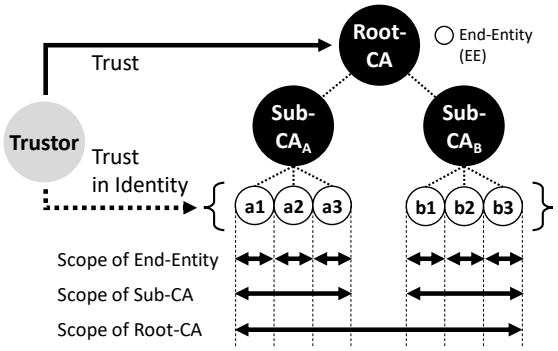

図 5　認証局の信頼点の範囲

Figure 5　Scopes of trust anchors of Certificate Authority.

本研究では，多数の主体に台帳を分散して安全に管理できる特徴のほかに，中央管理者なしに契約の履行を保証できるスマートコントラクトの特徴に着眼して，システム全体として統一された規則で信頼関係が構築されていることを保証する相互認証方式を提案する．階層型モデルの単一信頼点を解消するとともに，メッシュ型モデルでの認証局間の信頼関係の構築方法を示す．

4. 分散型認証基盤：Meta-PKI

4.1 目的

図 5 に示すように，階層型モデルでは，上位の認証局が下位の認証局や端末（End-Entity, EE）に対する公開鍵証明書を発行する．その証明書の発行範囲が信頼点の範囲（以降，スコープ）であり，認証局は自身のスコープにおける単一信頼点となる．そして，下位の主体のスコープは上位の認証局のスコープに集約されていくことから，最上位の認証局のスコープはシステム全体へと拡がる．

Root-CA で不正が発生した場合，Root-CA を基点とする信頼関係が損なわれることとなる．Root-CA はシステム全体へのスコープを持つので，その影響はシステム全体に及び，全ての主体間の信頼関係が損なわれる．一方で，全ての Sub-CA が一定の安全性を担保できる水準のセキュリティ対策を施していることの保証はなく，セキュリティ対策が相対的に弱い Sub-CA が存在する．そのような Sub-CA から不正な公開鍵証明書が発行された場合，Trustor は Root-CA の信頼点を基点にその不正な公開鍵証明書を信頼してしまう．例えば，EE を IoT エコシステムにおけるサービス提供サーバとした場合，これらの要因により IoT エコシステムの信頼性が損なわれることが懸念される．

本研究では，下位の主体のスコープが上位の主体のスコープに集約されることで，認証局の単一信頼点が形成されていることを着眼点として，認証局の信頼点をスコープの最小単位である EE に分散する分散型認証基盤の構築を目指している．具体的には，EE をサービス提供者が運用するサーバとして，サービス提供者がそのサービス内での利用者の本人性を保証する目的で認証局を運用する．各サービ

ス提供者に分散された認証局間の信頼関係を構築するために，サービス提供者同士が相互認証により認証局間の信頼関係を構築する．

4.2 課題

相互認証では，対向の認証局の信頼性を評価し，相手の公開鍵に対して, 自身の秘密鍵で公開鍵証明書を発行する．公開鍵証明書を検証することで，信頼関係が構築されていることを確認できる．対向の認証局の信頼性を評価して公開鍵証明書を発行する一連のプロセスは，専門家による属人的な判断のもとで行われるので，サービス提供者が各々の判断で相互認証するだけでは，システム全体としての信頼性を保証できない．

本研究では, 相互認証を実施するための枠組みを構築し，それに従って処理を自動化することで，属人的な作業を排除する．中央管理者なしに統一された方法で相互認証するために，契約の条件確認から履行までを自動化できるスマートコントラクトを用いる．

4.3 Meta-PKI の概要

本研究の目的である分散型認証基盤である Meta-PKI を図 6 に示す．Meta-PKI は Meta-CA 層と CA 層，End-Entity（EE）層の三つの層から構成される．

EE 層 : CA に対して証明書を申請する端末（EE）群からなるレイヤである．

CA 層 : Meta-CA が生成する CA 群からなるレイヤである．EE の本人性の保証を行い, EE に対する信頼点となる．

Meta-CA 層 : スマートコントラクトを用いた相互認証を行うメタ CA（Meta-CA, mCA）からなるレイヤであり，他の mCA との信頼関係を構築する．分散台帳ネットワークとの接続点であり，CA に対する信頼点であり，サービスごとの信頼点となる. mCA は自身の運用ポリシーを定めており，そのポリシーの水準に応じて信頼関係が構築されるようにスマートコントラクトを定義する．ポリシーの水準は 0.0 を最低値とした 1.0 までの値として表される．提案方式では，この値をスコア（Score）と呼び，自身のスコアより高い mCA を信頼するものとする．

図 6 では，mCA_trustor は mCA_trustee を信頼し，mCA_trustee は mCA_trustor に対して公開鍵証明書を発行することで mCA_trustor の本人性を保証している．これにより, EE_trustor は mCA_trustee を信頼点として, EE_trustee の本人性を信頼することができる．

4.4 Meta-PKI のユースケース

Meta-PKI のユースケースの一つとして，IoT エコシステムの構築が考えられる．図 7 に IoT エコシステムの例を示す．IoT エコシステムでは，IoT デバイスの所有者が IoT デバイスから生成されるデータを所有するデータ所有者（Data Owner, DO）であり，DO がデータをデータ分析者（Data Analyst, DA）に提供する．DA は提供されたデータ

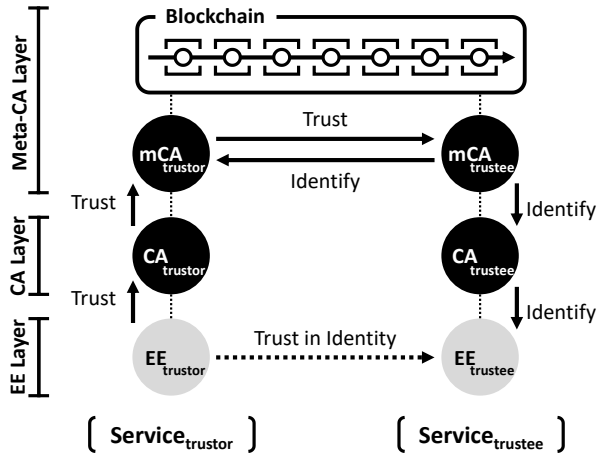

図 6　Meta-PKI の概要

Figure 6　An overview of Meta-PKI.

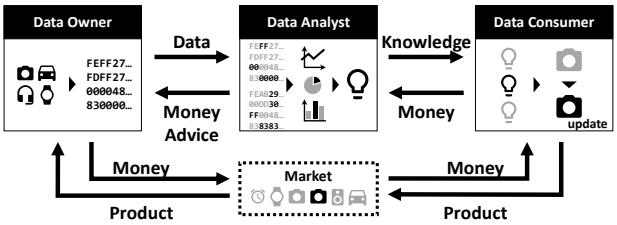

図 7　IoT エコシステムの例（[15]を基に作成）

Figure 7　Examples of IoT eco-systems (based on [15]).

を分析し，データ提供の見返りとして DO に分析結果や対価を還元する．さらに，DA は多数の分析結果を統合して新たに得られた知見をデータ購入者（Data Consumer, DC）に販売し, DC はその知見を利用して, 新たなデバイスを開発する．

DO や DA，DC それぞれをデータの提供／分析／購入を行うサービスとみなし，それぞれで多様なサービスが展開されていることを想定する．このような多様な主体が関連し合うシステムにおいて，各サービス提供者の本人性の保証を行う場合，認証局の単一信頼点の解消が課題となる．そこで，Meta-PKI のユースケースとして，サービスを提供するサーバ端末やサービスを利用するクライアント端末を EE として，その EE の本人性を保証する目的で mCA，CA を運用する．そして，mCA を介してサービス提供者間の信頼関係を構築する．

5. スマートコントラクトを用いた相互認証方式

Meta-PKI では，相互認証の実施や信頼関係の検証など下記に示す 7 つのトランザクション（Tx）が定義されている．Tx の発行に伴い分散台帳に格納されるデータを表 2 に示す. 分散台帳はキーバリュー型のデータベースであるので，任意のキーを指定して，任意のデータを格納することができる．提案手法において，主に，キーを mCA の識別名を用い，バリューとして mCA に紐づく各種情報を含めるために，再帰的なキーバリュー構造としており，そのデータ形

式として JSON オブジェクト[16]を使用している．

トランザクションの説明における主体は，図 6 に示した主体に準ずる．また，主体 E_X は X として識別される主体 E を，変数 V_E は主体 E が所有する変数 V を表す．特に，変数のうち，公開鍵証明書を$Cert^{E_a}_{E_b}$と表し，E_a から E_b に対して発行された公開鍵証明書であることを表す．

5.1 Initialization Tx（INI-Tx ）

mCA は INI-Tx により，Meta-PKI の初期化を行う．ここで，Meta-PKI の初期化を行う mCA を他の mCA と区別して mCA_initiator と表す．

この mCA_initiator は Meta-PKI で利用される認証局のポリシーの仕様 PolicySpec を定義し，文字列 “policy-spec” をキーとして分散台帳に格納する．表 2 では，ポリシーの例として，mCA と CA の公開鍵証明書に関するポリシー（MCA_CERT, CA_CERT）がセクション名 CERTIFICATE として指定されており，CERTIFICATE セクションの内容が”SPEC”内に定義されている．

CERTIFICATE セクションの例として，公開鍵証明書のパラメータである署名アルゴリズム（SIGN_ALGO），鍵アルゴリズム（KEY_ALGO），有効期間（VALIDITY_PERIDO）から構成されている．各ポリシーには 0.0～1.0 の範囲で mCA_initiator がスコアを定義する．SPEC と SECTION は Meta-PKI の適用先に応じて定義する必要があり，その指標を与えることは今後の課題である．

5.2 Participation Tx（PAR-Tx）

mCA は PAR-Tx により，Meta-PKI に参加する．mCA は PolicySpec に則したポリシー$Policy_{mCA}$と自己署名による公開鍵証明書$Cert^{mCA}_{mCA}$を作成する．分散台帳には，ID_{mCA}をキーとして，$Policy_{mCA}$と$Cert^{mCA}_{mCA}$が格納される．すでに，同一の ID が使用されていた場合は失敗とする．

5.3 Construction Tx（CON-Tx）

mCA_trustor は CON-Tx により，指定した mCA_trustee との信頼関係を構築する．mCA_trustor は信頼関係を構築したい mCA_trustee に対して CSR を発行し，相互認証を要求する．mCA_trustee は CSR に対する公開鍵証明書$Cert^{mCA_trustee}_{mCA_trustor}$を作成して，両 mCA の ID（$ID_{mCA\_trustor}$，$ID_{mCA\_trustee}$）と$Cert^{mCA_trustee}_{mCA_trustor}$で CON-Tx を実行する．CON-Tx では，両 mCA のポリシーのスコア（$Score_{mCA\_trustor}$，$Score_{mCA\_trustee}$）を計算して，その大小関係を比較し，$Score_{mCA_trustee}$が $Score_{mCA_trustor}$以上であれば，表 2 に示すように $ID_{mCA\_trustee}$ をキーに $ID_{mCA\_trustor}$ と$Cert^{mCA_trustee}_{mCA_trustor}$を”CrossCert”として分散台帳に格納する．

スコアは，各セクションに定義されたスコアを加算平均することで，”SPEC”内のそれぞれのスコアを算出し，その計算結果をさらに加算平均することで単一の値として算出する．結果として，スコアは 0.0 から 1.0 の値となる．

5.4 Destruction Tx（DES-Tx）

mCA_trustor は DES-Tx により，指定した mCA_trustee と

表 2　分散台帳に記録されるデータ

Table 2　Data recorded in Distributed Ledger.

Tx	Key	Value（JSON Object）
INI-Tx	“policy-spec”	{ “SPEC”: { “MCA_CERT”: “CERTIFICATE”, “CA_CERT”: “CERTIFICATE” }, “SECTION”: { “CERTIFICATE”: { “SIGN_ALGO”: { “MD5_WITH_RSA”: 0, “SHA1_WITH_RSA”: 0.5, “SHA256_WITH_RSA”: 0.8, “SHA512_WITH_RSA”: 1.0 }, “KEY_ALGO”: { “RSA_512”: 0, “RSA_1024”: 0.25, “RSA_2048”: 0.8, “RSA_4096”: 1.0 }, “VALIDITY_PERIOD”: { “1440”: 0.2, “720”: 0.4, “360”: 0.6, “180”: 0.8, “90”: 1.0 } } } }
PAR-Tx	ID_{mCA}	{ “Policy”: { “MCA_CERT”: { “SIGN_ALGO”: “SHA512_WITH_RSA”, “KEY_ALGO”: “RSA_2048”, “VALIDITI_PERIOD”: “90” }, “CA_CERT”: { “SIGN_ALGO”: “SHA512_WITH_RSA”, “KEY_ALGO”: “RSA_2048”, “VALIDITI_PERIOD”: “90” } }, “Cert”: $Cert^{mCA}_{mCA}$ }
CON-Tx	$ID_{mCA\_trustee}$	{ “CrossCert”: { $ID_{mCA\_trustor}$: { “Cert”: $Cert^{mCA\_trustee}_{mCA\_trustor}$ } } }
DES-Tx	$ID_{mCA\_trustee}$	{ “CrossCert”: { $ID_{mCA\_trustor}$: { “Cert”: <...>, “Destructed”: <date> } } }
DEP-Tx	ID_{mCA}	{ “CaCert”: { ID_{CA}: { “Cert”: $Cert^{mCA}_{CA}$ } } }
ELI-Tx	ID_{mCA}	{ “CaCert”: { ID_{CA}: { “Cert”: <...>, “Eliminated”: <date> } } }
VAL-Tx	-	-

の信頼関係を解消する．$ID_{mCA\_trustor}$ をキーに，信頼関係を解消した日時を対応する $ID_{mCA\_trustee}$ に関連付けて分散台帳に格納する．

5.5 Deployment Tx（DEP-Tx）

mCA は DEP-Tx により, CA を mCA の配下に配備する．mCA は CA の鍵ペアと公開鍵証明書$Cert_{CA}^{mCA}$を作成して，表 2 に示すように，ID_{mCA} をキーに ID_{CA} と$Cert_{CA}^{mCA}$を”CaCert”として分散台帳に格納する．

5.6 Elimination Tx（ELI-Tx）

mCA は ELI-Tx により，指定した CA を mCA の配下から除去する. ID_{mCA} をキーに, 除去した日時を対応する ID_{CA} に関連付けて分散台帳に格納する．

5.7 Validation Tx（VAL-Tx）

EE_trustor から EE_trustee に対して信頼関係が構築されているかを VAL-Tx により検証する．つまり，EE_trustor の信頼点が mCA_trustor であり，EE_trustee の信頼点が mCA_trustee であることを確認して，mCA_trustee が mCA_trustor に対して公開鍵証明書を発行していることを確認する．

処理の手順は次のようになる．まず，検証者は EE，CA の公開鍵証明書を EE から取得し, $Cert_{EE}^{CA}$が CA により発行されていることを以下のように検証する．ここで，Verify_Cert(A, B)は公開鍵証明書 B で公開鍵証明書 A を検証することを表す.

$$\text{Verify\_Cert}(Cert_{EE}^{CA}, Cert_{CA}^{mCA}) \quad (1)$$

次に，検証者は EE から受け取った$Cert_{CA}^{mCA}$が DEP-Tx により mCA が CA を配備したときに発行された証明書であることを以下のように検証する．ここで，Verify_DepTx($Cert_{B}^{A}$)は，$Cert_{B}^{A}$の発行者 A の ID_A をキーに分散台帳内の”CaCert”を取得し，その中に$Cert_{B}^{A}$が格納されていることを確認する．

$$\text{Verify\_DepTx}(Cert_{CA}^{mCA}) \quad (2)$$

次に，検証者は ID_{mCA} をキーに分散台帳内から$Cert_{mCA}^{mCA}$を取得し，$Cert_{CA}^{mCA}$が mCA により発行されていることを以下のように検証する．

$$\text{Verify\_Cert}(Cert_{CA}^{mCA}, Cert_{mCA}^{mCA}) \quad (3)$$

以上より，EE _trustee と EE_trustor それぞれにおいて，EE, CA, mCA の信頼関係が確認される. 最後に, $ID_{mCA\_trustee}$ をキーに分散台帳内の”CrossCert”を取得し，その中から $ID_{mCA\_trustor}$ に関連付いた $Cert_{mCA\_trustor}^{mCA\_trustee}$ を取得し，mCA_trustor と mCA_trustee 間で CON-Tx が実行されたことを以下のように検証する．

$$\text{Verify\_Cert}(Cert_{mCA\_trustor}^{mCA\_trustee}, Cert_{mCA\_trustee}^{mCA\_trustee}) \quad (4)$$

$$\text{Verify\_Cert}(Cert_{mCA\_trustor}^{mCA\_trustor}, Cert_{mCA\_trustee}^{mCA\_trustee}) \quad (5)$$

全ての検証に成功したら，VAL-Tx は成功を返す．なお，VAL-Tx において，分散台帳に格納されるデータはない．

6. 安全性

6.1 攻撃者の設定

攻撃者は Meta-PKI への攻撃により，攻撃者が管理する認証局 CA_attacker を Meta-PKI に不正に参加させることを考える．まず，初めに，正規の Meta-CA と CA_attacker の信頼関係を構築するために，正規の Meta-CA を乗っ取って CA_attacker に対して公開鍵証明書の発行を考える．次に，正規の Meta-CA になりすまして Meta-PKI に参加することで，正規の Meta-CA から CA_attacker に公開鍵証明書が発行されたように偽ることを考える．ここで，前提条件として，Meta-PKI を構成する各主体は不正を行わず，攻撃者と結託しないものとする．

6.2 Meta-CA の乗っ取り

攻撃者が mCA を乗っ取り，攻撃者が用意した CA_attacker に対して公開鍵証明書$Cert_{CA_attacker}^{mCA}$を不正に発行することで, CA_attacker が信頼されるような攻撃を考える．本攻撃の特徴として，mCA は CA を配備するために公開鍵証明書を発行するが，その際の公開鍵証明書への署名に利用される秘密鍵を攻撃者が利用できる点にある．ここで, CA_attacker が信頼されることを攻撃の目的とするので, VAL-Tx において CA_attacker が検知されなければ，攻撃成功とする．

まず, 攻撃者は秘密裏に攻撃を行うために, DEP-Tx を行わずに mCA から CA_attacker に$Cert_{CA_attacker}^{mCA}$を発行することを考える．DEP-Tx では，mCA が CA の公開鍵証明書$Cert_{CA}^{mCA}$を作成し，分散台帳に格納することで，mCA が CA を配備したことを保証している．つまり, DEP-Tx の実行なしに分散台帳に$Cert_{CA}^{mCA}$を格納できない．そのため， VAL-Tx により信頼関係を検証する際に，分散台帳に$Cert_{CA_attacker}^{mCA}$が格納されていることを検証者が確認することで，DEP-Tx が実行されていないことが式(2)により検知され，VAL-Tx は失敗となる．

また, 攻撃者が DEP-Tx を実行して$Cert_{CA_attacker}^{mCA}$を分散台帳に記録することで，VAL-Tx を成功させるように試みることを考える．DEP-Tx により分散台帳に格納された公開鍵証明書は各主体で共有されるので，mCA の管理者は$Cert_{CA_attacker}^{mCA}$が発行されたことを知ることができる．不正な公開鍵証明書を検知した場合，DES-Tx により当該証明書を無効にすることで VAL-Tx を失敗させることができる．

以上より，Meta-CA を乗っ取る攻撃者に対して安全である．

6.3 Meta-CA のなりすまし

攻撃者が mCA と同じ ID を持つ mCA’を偽造し，攻撃者が用意した CA_attacker に対して公開鍵証明書$Cert_{CA_attacker}^{mCA'}$を不正に発行することで, CA_attacker が信頼されるような攻撃を考える．本攻撃の特徴として，Meta-PKI において mCA の識別に利用される識別名を利用する点である. 識別

名は公開されているので，攻撃者がなりすましの対象の識別名で自身を偽ることで，mCA を乗っ取ることなく攻撃を試みる．攻撃の成否に関して，6.2 節と同様に，VAL-Tx において CA_attacker が検知されず成功となれば，攻撃成功となる．

まず，攻撃者は秘密裏に攻撃を行うために，PAR-Tx を行わずに mCA'を用意することを考える．PAR-Tx では，mCA の公開鍵証明書$Cert_{mCA}^{mCA}$が分散台帳に登録される．そして，この登録された公開鍵証明書は，VAL-Tx により信頼関係を検証する際に利用される．ここで，mCA'は mCA と同じ識別名 ID_{mCA} を持つが，mCA'は攻撃者が用意したものなので mCA とは異なる鍵ペアを持つので，公開鍵証明書は$Cert_{mCA'}^{mCA'}$となる．さらに，mCA'は CA_attacker に対して公開鍵証明書$Cert_{CA_attacker}^{mCA'}$を発行している．そのため，VAL-Tx による信頼関係の検証では，ID_{mCA} をキーとして保存された$Cert_{mCA}^{mCA}$で，式(3)が Verify_Cert($Cert_{CA\_attacker}^{mCA'}$，$Cert_{mCA}^{mCA}$)として実行されるので，VAL-Tx は失敗となる．

また，攻撃者が PAR-Tx を実行して mCA'を Meta-PKI に登録することで，VAL-Tx を成功させるように試みたとしても，すでに登録済みの ID で PAR-Tx を実行できないようにトランザクションが定義されているので，攻撃は失敗となる．

以上より，Meta-CA になりすます攻撃に対して安全である．

7. まとめ

本研究では，認証局がカバーする信頼点の範囲がサービス提供者のサービス提供範囲を包含することで，システム全体における単一信頼点が認証局に形成されることを着眼点として，各サービス提供者に信頼点である認証局を分散させる分散型認証基盤メタ PKI を目指している．

分散された認証局間の信頼関係の構築には，対向の認証局の信頼性を人手で評価したうえで公開鍵証明書を発行する相互認証が利用できる．しかし，一連の手続きは属人的であるので，各サービス提供者が独自に相互認証するだけでは，メタ PKI のシステム全体において適切な信頼関係が構築されていることを保証できない．そこで本稿では，分散台帳で管理された認証局の運用ポリシーから認証局の信頼性をスコア化するための枠組みとそのスコアに基づくスマートコントラクトを用いた相互認証方式を提案した．契約の条件確認と履行を自動化するスマートコントラクトにより，相互認証の一連の処理を自動化している．

安全性評価では，Meta-PKI に参加する主体は不正を行わないことを前提として，外部の攻撃者による認証局 Meta-CA（mCA）の乗っ取りと偽造に対して安全であることを示した．

提案手法では，mCA は不正を行わないことを前提としていたが，不適切な運用により認証局の信頼性が損なわれるような事件が発生していることを考慮すると，mCA がポリシー通りに運用されていることを検証する仕組みが必要である．また，IoT デバイのような小型のデバイスでは，処理能力が低く，単純に暗号アルゴリズムの強度でポリシーを定義し，認証局の信頼性を測るだけでは，様々なサービスに適用するにあたっては不十分であり，さらなる指標の策定が必要である．本研究ではこれらを今後の課題とする．

参考文献

[1] 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. RFC 5280, https://tools.ietf.org/html/rfc5280, 2008, (参照 2019-08-13).

[2] Comodo group, Report of incident on 15-MAR-2011: Update 31-MAR-2011. https://www.comodo.com/Comodo-Fraud-Incident-2011-03-23.html, 2011, (参照 2019-08-13).

[3] Prints, J.R., DigiNotar Certificate Authority breach - Operation Black Tulip. http://tweakimg.net/files/upload/Operation+Black+Tulip+v1.0.pdf, Fox-IT, 2011, (参照 2019-08-13).

[4] O'Brien, D., Sleevi, R. and Whalley, A., Chrome's Plan to Distrust Symantec Certificates. https://security.google-blog.com/2017/09/chromes-plan-to-distrust-symantec.html, 2017, (参照 2019-08-13).

[5] Kille, S., A String Representation of Distinguished Names. RFC 1779, https://tools.ietf.org/html/rfc1779, 1995, (参照 2019-08-15).

[6] Nystrom, M., Kaliski, B., PKCS #10: Certification Request Syntax Specification Version 1.7. RFC 2986, https://tools.ietf.org/html/rfc2986, 2000, (参照 2019-08-15).

[7] Shimaoka, M., Hastings, N., and Nielsen, R. Memorandum for Multi-Domain Public Key Infrastructure Interoperability. RFC 5217, https://tools.ietf.org/html/rfc5217, 2008, (参照 2019-08-13).

[8] 佐藤雅史，長谷川佳祐，佐古和恵，並木悠太，梶ヶ谷圭祐，松尾真一郎，ブロックチェーン技術の教科書. C&R 研究所, 2018 年.

[9] The Linux Foundation, Hyperledger Fabric. https://www.hyperledger.org/projects/fabric, (参照 2019-08-19).

[10] Laurie, B., Langley, A., and Kasper, E., Certificate Transparency. RFC 6962, https://tools.ietf.org/html/rfc6962, 2013, (参照 2019-08-15).

[11] Madala, D. S. V., Jhanwar, M. P., and Chattopadhyay, A., Certificate Transparency Using Blockchain. IEEE International Conference on Data Mining Workshops (ICDMW), p. 71-80, 2018.

[12] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and Thayer, R., OpenPGP Message Format. RFC 4880, https://tools.ietf.org/html/rfc4880, 2007, (参照 2019-08-15).

[13] Yakubov, A., Shbair, W. and State, R. BlockPGP: A Blockchain-based Framework for PGP Key Servers. 2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW), 2018.

[14] Al-Bassam, M., SCPKI: A Smart Contract-based PKI and Identity System. Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, p. 35-40, 2017.

[15] Perera, C., Sensing as a service (S2aaS): Buying and selling IoT data. IEEE Internet of Things eNewsletters, 2016.

[16] JSON. https://www.json.org/, (参照 2019-08-15).