Vol.2020-CSEC-91 No.31
Vol.2020-SPT-40 No.31
Vol.2020-EIP-90 No.31
2020/11/26

ブロックチェーンに基づく分散属性ベース暗号のEthereumにおける実装および評価

佐藤 龍[1]　今村 光良[2]　大東 俊博[1,3]

概要：ブロックチェーン技術は非中央集権的な分散ネットワークにおいて改ざん耐性や障害耐性を実現する．2018 年に Bramm らは ICISC 2008 で提案された分散属性ベース暗号とブロックチェーン技術を組み合わせた Blockchain-based Distributed Attribute based Encryption (BDABE) を提案した．Bramm らは BDABE について Permissioned 型のブロックチェーンを基盤とした Multi-Chain に実装したため，汎用的なプラットフォームとの比較が難しく，また，処理時間・通信時間・必要な費用などの性能評価が十分でない．そこで本研究では，コミュニティにおいてよく知られたプラットフォームである Ethereum を対象に，スマートコントラクトを用いた BDABE の実装および運用時における評価を報告する．

Implementation and Evaluation on a Blockchain-based Distributed Attribute Based Encryption on Ethereum

Ryu Sato[1]　Mitsuyoshi Imamura[2]　Toshihiro Ohigashi[1,3]

1. はじめに

暗号文ポリシー属性ベース暗号 (CP-ABE)[1] は属性ベース暗号 [2] の一種で，属性値 (ID・所属・役職など) の論理式で表現されたアクセスポリシーを暗号文に埋め込むことで，ユーザ属性に基づいたアクセス制御を実現する公開鍵暗号方式である．CP-ABE は属性情報を含む鍵発行を 1 つの鍵発行センターが担う中央集権的に構造で設計されていため，ICISC 2008 において Müller らは複数の鍵発行センタをサポートする手法として分散属性ベース暗号 (DABE)[3] を提案している．

Bramm らは，DABE における課題として，適切なユーザへの秘密鍵配送やアクセスポリシー設計側から見たユーザが保有する属性の透明性を指摘し，Permissioned 型のブロックチェーン基盤である MultiChain[*1] を用いて，分散台帳を利用した鍵配送および属性管理の透明性を提案している．一方で, MultiChain は Unspent Transaction Output (UTXO) 型 によりトランザクションが管理されており，アドレス利用した実装を追加する場合，ユーザとアドレスの接続性に課題がある．また，理論実装を主題としているため，ユースケースへの展開するには，処理時間や通信時間，運用コストといった性能評価を明らかにする必要がある．

そこで，本研究では，アカウント型の基盤における設計を議論するため，ブロックチェーンコミュニティにおいて良く知られた Ethereum[*2] を対象に，Bramm らの手法の実装を行う．そして複数組織間でファイルの暗号化や復号を行うことを想定してクライアント及びサーバでの処理時間，通信時間，必要なコストの評価を実施した．

2. 要素技術

本研究は BDABE を用いたファイルの暗号化・復号のシステムを Ethereum 上に実装し，評価することを目的とし

[1] 東海大学 情報通信学部,
School of Information and Telecom-munication Engineering, Tokai University
[2] 野村アセットマネジメント株式会社
Nomura Asset Management Co., Ltd.
[3] 国立研究開発法人情報通信研究機構
National Institute of Information and Communications Technology

[*1] https://www.multichain.com
[*2] https://ethereum.org/ja/

ている．本章では，要素技術である Ethereum について述べたあとに Bramm らの分散属性ベース暗号について解説を行う．

2.1 Ethereum

Ethereum はブロックチェーン基盤による分散コンピューティングを目的としたプラットフォームで，分散台帳内にプログラムされた内容を，Ethereum Virtual Machine (EVM) が代替実行することで，プログラムを自動実行するスマートコントラクトを実現している．この際，無限ループなどの発生でネットワークが停止してしまうことを防ぐために各オペレーションに Gas と呼ばれる実行コストが設定されており，消費した Gas の量 × Gas Price の分の ETH が必要になる．

2.2 Bramm らによる分散属性ベース暗号

本章では BDABE を Ethereum に実装するにあたって，先行研究として Bramm らの方式 [4] について概説する．まず初めに，このアルゴリズムにおけるユーザの各役割について述べる．その後に，鍵生成に関する処理を説明し，最後にブロックチェーン利用に関する処理についての解説を行う．

2.2.1 役割

本節では Bramm らの方式における役割について述べる．このアルゴリズムには以下の 4 つの役割が存在している．

Root Authority (RA) Root Authority は，パラメーターの設定や鍵の計算などを行い BDABE のシステム全体を初期化する処理，および秘密鍵を計算して AU を作成する処理を行う．RA はネットワークに参加している全ユーザから信頼されている必要がある．

Attribute Authority (AU) Attribute Authority は，鍵ペアを計算して DR を作成する処理，属性の公開鍵を計算して新しく属性を作成する処理，属性の秘密鍵を計算して DR に属性を割り当てる処理，および自身が作成した DR の属性を失効させる処理を行う．AU は自身が作成する DR と属性を割り当てる DR から信頼されている必要がある．

Data Reader (DR) Data Reader は AU から属性の割当を受け，その秘密鍵を用いて暗号化されたファイルを復号することができる．

Data Owner (DO) Data Owner はファイルの所有者で，属性の公開鍵を用いて自身の持つファイルを暗号化する．BDABE のネットワークに参加している必要性はない．

2.2.2 鍵生成に関する処理

続いて本節では，鍵生成に関する処理について述べる．この処理は以下に示す 7 つのアルゴリズムから構成されている．また，これらアルゴリズムで生成される鍵を表 1 にまとめている．

Setup() RA が BDABE のシステムを初期化する際に一度だけ実行するアルゴリズム．はじめにパラメーターや生成元の決定などを行って公開鍵 PK 及びマスターキー MK を出力する．

CreateAuthority() RA が AU を作成する際に実行するアルゴリズム．Setup() で計算された PK，MK 及び AU のアドレス a_{AU} を入力し，AU の秘密鍵を計算して SK_{AU} を出力する．

CreateUser() AU が DR を作成する際に実行するアルゴリズム．ここでは公開鍵 PK，AU の秘密鍵 SK_{AU} 及び DR のアドレス a_{DR} を入力し，DR の公開鍵と秘密鍵 $PK_{a_{DR}}, SK_{a_{DR}}$ を出力する．

RequestAttributePK() AU が属性を新しく作る際に実行するアルゴリズム．公開鍵 PK，AU の秘密鍵 SK_{AU}，属性 $\mathcal{A}$ を入力し，属性の公開鍵 $PK_{\mathcal{A}}$ を出力する．

RequestAttributeSK() AU が DR に対して属性を割り当てる際に実行するアルゴリズム．公開鍵 PK，AU の秘密鍵 SK_{AU}，属性 $\mathcal{A}$，DR の公開鍵 $PK_{a_{DR}}$ を入力し，属性の秘密鍵 $SK_{A,a_{DR}}$ を出力する．

Encrypt() DO がアクセスポリシー $\mathcal{A}$ を定義して，ファイルの暗号化を行うアルゴリズム．公開鍵 PK，平文 M，選言標準形 (DNF) で定義されたアクセスポリシー $\mathcal{A}$ と，属性の公開鍵 $PK_{A1} \cdots PK_{AN}$ を入力し，暗号文 CT を出力する．

Decrypt() DR が暗号文 CT を復号するためのアルゴリズム．DR の秘密鍵 $SK_{a_{DR}}$，属性の秘密鍵 $SK_{A,a_{DR}}$，暗号文 CT，アクセスポリシー $\mathcal{A}$ を入力し，平文 M 出力する．

2.2.3 ブロックチェーン利用に関する処理

続いて，本節ではブロックチェーン利用に関する処理について述べる．Bramm らは MultiChain 上で以下の 6 つのアルゴリズムを実装した．

Setup$\mathfrak{B}$() Setup$\mathfrak{B}$() は，RA がブロックチェーンを初期化して鍵をブロックチェーン上で配布することで BDABE プロトコルをセットアップする．この処理では，まず新しいチェーンを生成して genesis block の採掘を行う．続いて，Setup() を実行して公開鍵 PK とマスターキー MK を計算し，PK は誰でも利用できるように公開する．

CreateAuthority$\mathfrak{B}$() CreateAuthority$\mathfrak{B}$() は AU が RA に対して鍵の発行をリクエストした際に実行される．リクエストを受けた RA はブロックチェーン上で用いる RSA キーペア RSA_{priv}, RSA_{pub} とアドレス a_{AU} を計算する．その後，アドレス a_{AU} に対して “Connect”，“Send”，“Recieve”，“issue” のパーミッションを付与する．最後に CreateAuthority() を実行

Vol.2020-CSEC-91 No.31
Vol.2020-SPT-40 No.31
Vol.2020-EIP-90 No.31
2020/11/26

表 1 鍵の種類

記号	名称	説明
PK	公開鍵	殆どの関数で使用する公開鍵．この鍵はブロックチェーン上に書き込まれてシステム全体に公開される．
MK	マスターキー	RA が SK_{AU} 作成する際に使用する鍵．この鍵は RA が管理し，外部からは秘匿される．
SK_{AU}	AU の秘密鍵	AU が $SK_{a_{\mathcal{A},a_{DR}}}$ を作成する際に使用する鍵．RA が計算し，安全な方法で AU へ配布される．
$PK_{a_{DR}}$	DR の公開鍵	AU が $SK_{\mathcal{A},a_{DR}}$ を作成する際に使用する鍵．
$SK_{a_{DR}}$	DR の秘密鍵	DR が暗号文を復号する際に使用する鍵．AU が計算し，安全な方法で DR へ配布される．
$PK_{\mathcal{A}}$	属性の公開鍵	DO が暗号化をする際に使用する鍵．各 AU が保持しており，リクエストすることで誰でも入手可能になっている．
$SK_{\mathcal{A},a_{DR}}$	属性の秘密鍵	DR が暗号文を復号する際に使用する鍵．AU が計算した後，暗号化してブロックチェーン上で配布される．

して SK_{AU} を計算し, RSA キーペア RSA_{priv}, RSA_{pub} と共に安全な方法で AU に配布する．

CreateUser$\mathfrak{B}$() CreateUser$\mathfrak{B}$() は DR が AU に対して鍵の発行をリクエストした際に実行される．リクエストを受けた AU はブロックチェーン上で用いる RSA キーペア RSA_{priv}, RSA_{pub} とアドレス a_{DR} を計算する．その後，アドレス a_{DR} に対して “Connect”，“Send”，“Recieve” のパーミッションを付与する．最後に CreateUser() を実行して DR の秘密鍵 $SK_{a_{DR}}$ と公開鍵 $PK_{a_{DR}}$ を計算し，RSA キーペア RSA_{priv}, RSA_{pub} と DR の秘密鍵 $SK_{a_{DR}}$ が安全な方法で DR に配布する．また，属性の失効を行うために RSA キーペアは AU にも保存しておく．

CreateAttribute$\mathfrak{B}$() CreateAttribute$\mathfrak{B}$() は AU が属性を作成する際に実行される．初めに RequestAttributePK() を実行し，属性の公開鍵 $PK_{\mathcal{A}}$ を計算する．この鍵は暗号化の際に利用するため，リクエストがあれば誰でも入手できるように公開する．続いて，属性の文字列 $\mathcal{A}$ を示すアセットを AU のアドレス宛に発行する．次に，他の AU がその属性 $\mathcal{A}$ を発行できるかを決定して “issue” のパーミッションを付与する．また，属性が DR に発行された履歴を追跡するために AU はそのアセットを “subscribe”[*3]する．

IssueAttribute$\mathfrak{B}$() IssueAttribtue$\mathfrak{B}$() は AU が DR に属性を発行する際に実行される．属性を発行する前には必ず CreateAttribute$\mathfrak{B}$() で属性を作成する必要がある．初めに，DR に属性 $\mathcal{A}$ を発行するのが適切かを何らかの手順で判定する．属性を発行できると判定されると，RequestAttributeSK() を実行して属性の秘密鍵 $SK_{\mathcal{A},a_{DR}}$ を計算する．計算された属性の秘密鍵 $SK_{\mathcal{A},a_{DR}}$ は DR の公開鍵 RSA_{pub} で暗号化された後に，アセットの残高を AU から DR に送金するトランザクションに含めて送信する．アセットの総金額は “1” だけ DR に送られ，アセットの残高を参照することで属性が割り当てられているか否かを判定するために利用する．

RevokeAttribute$\mathfrak{B}$() RevokeAttribute$\mathfrak{B}$() は属性失効の際に実行される．失効の操作は RSA キーペアを知る AU だけが実行できる．つまり，CreateUser$\mathfrak{B}$() を実行した AU は，他の AU が発行した属性であってもその DR がもつ全ての属性に対しての責任がある．ブロックチェーンを用いた属性の失効は鍵失効とは異なるため，暗号論的に前方秘匿性はない．プログラムを書き換えて秘密鍵をエクスポートすることで，後からブロックチェーンの外で暗号文を復号できる．

[*3] MultiChain のクライアント上でアセットを追跡する機能

3. 実装

本章では，はじめに分散属性ベース暗号の鍵生成などについて説明し，次に，Ethereum 上での構成法についての解説を行う．

3.1 分散属性ベース暗号

今回，分散属性ベース暗号の鍵生成アルゴリズムを実装するにあたって，開発言語として C 言語を用いた．また，Bramm らの方式は非対称ペアリングから構成されるため，ペアリング暗号ライブラリである PBC Library[*4]の曲線である Type F curve [*5]を用いて実装した．

Setup(): Setup() ではまず初めに素数 $p > 3$ となる線形写像群 $\mathbb{G}_1$，$\mathbb{G}_2$ とペアリング関数 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ 選択する．続いて，$g_1 \in \mathbb{G}_1$，$g_2 \in \mathbb{G}_2$ を満たす生成元 g_1, g_2, $P_1 \in \mathbb{G}_1$, $P_2 \in \mathbb{G}_2$ を満たす無作為な点 P_1, P_2, および $\mathrm{y} \in \mathbb{Z}_p$ を満たす無作為な整数 y を選択する．そして，グローバルハッシュ関数 $H : \{0,1\}^* \to \mathbb{Z}_p$ を定義する．このとき，公開鍵 PK は以下の式で与えられる．

$$PK = \left\{ \begin{array}{l} \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, H, \\ e, g_1, g_2, P_1, P_2, \\ e(g_1, g_2)^{\mathrm{y}} \end{array} \right\}$$

また，マスターキー MK は以下のように与えられる．

$$MK = \{\mathrm{y}\}$$

[*4] https://crypto.stanford.edu/pbc/

[*5] 128 ビット安全性を確保するために，PairingParametersGenerator API を用いて素数位数を 455-bit に定義した BN 曲線 (Type F curve) を生成して使用している．

Vol.2020-CSEC-91 No.31
Vol.2020-SPT-40 No.31
Vol.2020-EIP-90 No.31
2020/11/26

CreateAuthority(PK, MK, a_{AU})**:** CreateAuthority() ではまず初めに $r_{AU} \in \mathrm{Z}_p$ を満たす一様な乱数 r_{AU} を選択する．この乱数はそれぞれのオーソリティーでランダムオラクルとして仮定され，グローバルハッシュ関数 H の出力を無作為化するために用いられる．続いて MK を $\alpha, \beta \in \mathbb{Z}_p$ と $\alpha + \beta = \mathrm{y} \equiv MK$ の条件を満たす無作為な α, β に分離する．この α, β を指数として ${g_1}^{\alpha}, {g_2}^{\beta}$ を計算し，無作為因子である r_{AU} と AU のアドレス a_{AU} とを加えて AU の秘密鍵とする．SK_{AU} は以下の式で与えられる．

$$SK_{AU} = \left\{ \begin{array}{ll} SK_{AU,I} = {g_1}^{\alpha}, & SK_{AU,II} = {g_2}^{\beta}, \\ SK_{AU,III} = r_{AU}, & SK_{AU,IV} = a_{AU} \end{array} \right\}$$

CreateUser(PK, SK_{AU}, a_{DR})**:** CreateUser() ではまず初めに $r_{a_{DR}} \in \mathbb{Z}_p$ を満たす無作為な $r_{a_{DR}}$ を選択して秘密の値とする．このとき，DR の公開鍵 $PK_{a_{DR}}$ は以下の式で与えられる．

$$PK_{a_{DR}} = \left\{ \begin{array}{l} PK_{a_{DR},I} = {g_1}^{r_{a_{DR}}}, \\ PK_{a_{DR},II} = {g_2}^{r_{a_{DR}}}, \\ PK_{a_{DR},III} = a_{DR} \end{array} \right\}$$

また，DR の秘密鍵 $SK_{a_{DR}}$ は SK_{AU} を利用して計算され，以下の式で与えられる．

$$SK_{a_{DR}} = \left\{ \begin{array}{l} SK_{a_{DR},I} = {g_1}^{\alpha} \cdot {P_1}^{r_a}, \\ SK_{a_{DR},II} = {g_2}^{\beta} \cdot {P_1}^{r_a} \end{array} \right\}$$

RequestAttributePK$(PK, SK_{AU}, \mathcal{A})$**:**
RequestAttributePK() を実行する前に AU は鍵計算を行う前に入力された属性 $\mathcal{A}$ を作成する権限があるかを判定し，AU が属性を作成する権限を持っていない場合はこのアルゴリズムは実行されずに NULL を返す．属性の作成をする権限があればこのアルゴリズムを実行して，$PK_{\mathcal{A}}$ を計算する．また AU に属性の管理権限がある場合，他の AU に属性の発行を許可するかどうかを設定することもできる．

$PK_{\mathcal{A}}$ の計算アルゴリズムでは初めに SK_{AU} を用いて指数 $\varepsilon(A, a_{AU}) = H(A) \cdot r_{AU} \cdot H(a_{AU})$ を計算する．このとき，$PK_{\mathcal{A}}$ は以下の式で与えられる．

$$PK_{\mathcal{A}} = \left\{ \begin{array}{l} PK_{\mathcal{A},I} = {g_1}^{\varepsilon(A,AU)}, \\ PK_{\mathcal{A},II} = {g_2}^{\varepsilon(A,AU)}, \\ PK_{\mathcal{A},III} = e(g_1, g_2)^{\mathrm{y} \cdot \varepsilon(A,AU)} \end{array} \right\}$$

この $PK_{\mathcal{A}}$ は SK_{AU} の無作為因子 r_{AU} を含むため，それぞれの AU にしか計算できない．そのため，一度リクエストされた属性の $PK_{\mathcal{A}}$ は，誰もが利用できるように公開しておく．

RequestAttributeSK$(PK, SK_{AU}, \mathcal{A}, PK_{a_{DR}})$**:**
RequestAttributeSK() は DR が属性の秘密鍵の発行をリクエストした際に AU によって実行される．AU が属性を発行する権限を持っていない場合，このアルゴリズムは実行されず NULL を返す．属性を発行をする権限を持っていればこのアルゴリズムを実行して，$SK_{A,a_{DR}}$ を計算する．

このアルゴリズムでは初めに SK_{AU} を用いて指数 $\varepsilon(A, a_{AU}) = H(A) \cdot r_{AU} \cdot H(a_{AU})$ を計算する．属性の秘密鍵 $SK_{A,a_{DR}}$ は指数 $\varepsilon(A, a_{AU})$，公開鍵 PK，AU の秘密鍵 SK_{AU}，属性 $\mathcal{A}$，DR の公開鍵 $PK_{a_{DR}}$ を用いて計算され，以下の式で与えられる．

$$SK_{\mathcal{A},a_{DR}} = \left\{ \begin{array}{l} SK_{\mathcal{A},a_{DR},I} = (PK_{a_{DR},I})^{\varepsilon(A,AU)}, \\ SK_{\mathcal{A},a_{DR},II} = (PK_{a_{DR},II})^{\varepsilon(A,AU)} \end{array} \right\}$$
$$= \left\{ g1^{r_{AU} \cdot \varepsilon(A,AU)}, g2^{r_{AU} \cdot \varepsilon(A,AU)} \right\}$$

Encrypt$(PK, M, \mathbb{A}, PK_{\mathcal{A}_1}, ...PK_{\mathcal{A}_N})$**:** Encrypt() は DO が暗号化を行う際に実行される．このアルゴリズムでは初めに，$r \in \mathbb{G}_T$ となる乱数 r を選択し，SHA256 アルゴリズムを用いて共通鍵 $K = SHA256(r)$ を計算する．続いて DNF で入力されたアクセスポリシー $\mathbb{A}$ を複数の AND で表される条件式 $\{S_1, S_2, ...S_n\}$ に分解し，分解された条件式の数 $j = 1, 2...n$ だけ乱数 $r_j \in \mathbb{Z}$ を選択する．そして，CT_j を条件式の数だけ繰り返し計算する．このとき，CT_j は以下の式で与えられる．

$$CT_j = \left\{ \begin{array}{l} E_{j,I} = r \cdot \left(\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A},III} \right)^{r_j}, \\ E_{j,II} = P_1^{r_j}, \\ E_{j,III} = P_2^{r_j}, \\ E_{j,IV} = \left(\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A},I} \right)^{r_j}, \\ E_{j,V} = \left(\prod_{\mathcal{A} \in S_j} PK_{\mathcal{A},II} \right)^{r_j} \end{array} \right\}$$

前の工程で計算した $\{CT_1, CT_2, ..., CT_n\}$ と平文 M を K を鍵として AES256-GCM で暗号化したものを結合し，以下の式で表される暗号文 CT とする．

$$CT = \{CT_1, CT_2, ..., CT_n, AES\_ENC(M)\}$$

Decrypt$(CT, \mathbb{A}, SK_{\mathcal{A}_1,a_{DR}}, ...SK_{\mathcal{A}_N,a_{DR}})$**:** A Decrypt() は DR が複号を行う際に実行される．このアルゴリズムでは初めに，DNF で入力されたアクセスポリシー $\mathbb{A}$ を複数の AND で表される条件式 $\{S_1, S_2, ...S_n\}$ に分解する．そして DR が復号に必要な属性を持っていない場合は NULL を返す．属性を持っている場合は以下に示す式で r を計算し，r を SHA256 アルゴリズムで計算して共通鍵 K を得て暗号文 CT に含まれる $AES\_ENC(M)$ を復号することで平文 M を取り出す．

$$r = E_{j,I} \cdot \frac{e\left(E_{j,II}, \prod_{\mathcal{A} \in S_j} SK_{\mathcal{A},a_{DR},II}\right) \cdot e\left(E_{j,III}, \prod_{\mathcal{A} \in S_j} SK_{\mathcal{A},a_{DR},I}\right)}{e\left(E_{j,IV}, SK_{a,II}\right) \cdot e\left(E_{j,V}, SK_{a,I}\right) \cdot}$$

3.2 スマートコントラクト

Bramm らは MultiChain のアクセスコントロール機能とトークン発行機能であるアセット機能を利用して属性管理を実装している．今回, Ethereum におけるアクセスコントロールおよびトークンの発行はスマートコントラクトによる実装で実現している．開発言語には Solidity(0.7.1)[*6] を利用し，以下の 5 つの登録関数を実装した．

constructor(Txid$_{PK}$) コントラクタはスマートコントラクトをデプロイする際に一度だけ実行される．この関数では，デプロイを行ったアドレスを RA として登録し，PK を送信したトランザクションのトランザクション IDTxid$_{PK}$ をスマートコントラクト内に保存する．

create_authority() create_authority() は AU のアドレス a_{AU} をスマートコントラクトに登録するために実行する．登録処理を行う前にコントラクト実行のトランザクションの署名を検証し，実行者が RA として登録されているかを確認し，RA ではない場合登録を拒否する．入力されたアドレス a_{AU} が既に役割を持っている場合も登録が行えない．アドレスに対して AU であるというフラグを立てることで登録が行われ，正常に登録が終わると AU のアドレス a_{AU} を Created イベントで返す．

create_user() create_user() は DR のアドレス a_{DR} をスマートコントラクトに登録するために実行する．登録処理を行う前にコントラクト実行のトランザクションの署名を検証し，実行者が AU として登録されているかを確認し，AU ではない場合登録を拒否する．アドレス a_{DR} が何らかの役割として登録されていた場合も登録は行われない．アドレスに対して DR であるというフラグを立てて，作成した AU のアドレスを親として追加することで DR の登録が行われ，正常に登録が終わると DR のアドレス a_{DR} を Created イベントで返す．

create_attribute($\mathcal{A}$) create_attribtue() は属性の作成を行う際に実行する．属性を作成する前にコントラクト実行のトランザクションの署名を検証し，実行者が AU として登録されているかを確認し，AU ではない場合作成を拒否する．属性 $\mathcal{A}$ が作成済みの場合も同じ属性を作成することはできない．属性の作成者と，属性を発行可能な AU に自分のアドレスを追加することで登録が行われ，正常に属性の作成が終わると属性 $\mathcal{A}$ を AttributeCreated イベントで返す．

[*6] https://github.com/ethereum/solidity

delegate_authority(a_{AU},$\mathcal{A}$) delegate_authority() は属性の発行権限を他の AU に委任する場合に実行する．属性を委任する前にコントラクト実行トランザクションの署名を検証し，実行者が $\mathcal{A}$ の作成者であることを確認し，条件に一致しない場合委任を拒否する．すでに a_{AU} に委任が行われている場合も属性の委任は実行できない．$\mathcal{A}$ を発行可能な AU として a_{AU} を追加することで属性の委任が行われ，正常に属性の委任が終わると属性 $\mathcal{A}$ と委任先の AU のアドレス a_{AU} を IssuerAdded イベントで返す．

issue_attribute(a_{DR},$\mathcal{A}$) issue_attribute() は属性を DR に発行する場合に実行する．発行する前にコントラクト実行のトランザクションの署名を検証し，実行者が属性 $\mathcal{A}$ の発行権限を持つかを確認し，発行権限がない場合発行を拒否する．DR に属性発行のフラグを立てることで属性の発行を行われ，正常に属性の発行がされると属性 $\mathcal{A}$ と発行先の DR のアドレス a_{AU} を AttributeIssued イベントで返す．

revoke_attribute(a_{DR},$\mathcal{A}$) revoke_attribute() は DR の持つ属性を失効させ場合に実行する．失効の前にコントラクト実行のトランザクションの署名を検証し, 実行者が DR の作成者であることを確認し，条件を満たさない場合は実行を拒否する．また，属性が DR に未発行だった場合も実行を拒否する．属性の失効は，発行フラグを消去することで行われる, 正常に失効が終わると属性 $\mathcal{A}$ と DR のアドレス a_{AU} を AttributeRevoked イベントで返す．

これらの関数で登録された情報は自動的に生成もしくは手動で実装したゲッターメソッドから参照することができ，暗号化や復号の際に用いられる．

3.3 Ethereum での鍵の配布

Ethereum での鍵の配布などを実装するにあたって，開発言語として Python を用いた．また，実装にあたっては Ethereum で用いられる ECDSA 鍵ペアの生成の為に cryptos(1.36) 及び eciespy(0.3.6) を，$SK_{\mathcal{A},a_{DR}}$ を暗号化して Ethereum 上にアップロードするために pycryptodome(3.9.8) を，スマートコントラクトをデプロイするために py-solc(3.2.0) を，geth との通信のために web3(5.12.1) を，そして，サーバとして稼働させるために Flask(1.1.2) のライブラリを使用した．

スマートコントラクトを利用する為に Bramm らのアルゴリズムを以下のように変更し，実装を行った．

Setup$\mathfrak{B}$() BDABE のシステムを初期化する関数．図 1 に示すように以下の処理を行う．はじめに, setup() を実行して PK, MK を生成しその中の PK を自分宛ての送金トランザクションに埋め込んで 0wei 送金する．その後，PK が埋め込まれたトランザクションのトラ

ンザクション ID$Txid_{PK}$ を引数としてスマートコントラクトをデプロイする．Setup 実行後 RA は web サーバとして稼働し，AU からの HTTPS リクエストを受け付ける．

CreateAuthority$\mathfrak{B}$() AU からリクエストがあった際に鍵を発行する関数．図 2 に示す手順で AU に鍵を発行する．はじめに AU の ECDSA キーペア $ECDSA_priv, ECDSA_pub$ を選択し，この鍵ペアから Ethereum アドレス a_{AU} を計算する．次に AU のアドレスをスマートコントラクトに登録し，正常に登録ができたら CreateAuthority() を実行して，SK_{AU} を生成する．SK_{AU} はレスポンスとして HTTPS のセキュアチャネル上で AU に送信する．鍵を発行された後，AU は Web サーバとして稼働し DR からのリクエストを HTTPS で受け付ける．

CreateUser$\mathfrak{B}$() DR からリクエストがあった際に鍵を発行する関数．図 3 に示す手順で DR に鍵を発行する．はじめに DR の ECDSA キーペア $ECDSA_priv, ECDSA_pub$ を選択し，この鍵ペアから Ethereum アドレス a_{AU} を計算する．次に CreateUser() を実行して $SK_{a_{DR}}, PK_{a_{DR}}$ を生成する．このうち $PK_{a_{DR}}$ は DR 宛のトランザクションに埋め込んでアップロードする．アップロードが終わったらトランザクション ID$Txid_{PK_{a_{DR}}}$ とアドレスをスマートコントラクトに登録して，最後に $SK_{a_{DR}}$ とトランザクション ID$Txid_{PK_{a_{DR}}}$ をレスポンスとして HTTPS のセキュアチャネル上で DR に送信する．Bramm らの方式とは異なり，作成した ECDSA キーペアは保存せずとも属性の失効が行える．

CreateAttribute$\mathfrak{B}$($\mathcal{A}$) 属性 $\mathcal{A}$ を作成する関数．はじめに RequestAttributePK() を実行して $PK_{\mathcal{A}}$ を生成し，その後属性 $\mathcal{A}$ をスマートコントラクトに登録する．生成された $PK_{\mathcal{A}}$ は暗号化の際に必要になるので AU に HTTPS リクエストを送ることで取得できるようにしておく．

IssueAttribute$\mathfrak{B}$($\mathcal{A}$,a_{DR},$ECDSA_pub$,$Txid_{PK_{a_{DR}}}$) DR からのリクエストで属性を割り当てる関数．図 4 に示す手順で DR に属性を発行する．はじめにスマートコントラクトから $Txid_{PK_{DR}}$ を取得し DR の公開鍵 PK_{DR} を Ethereum からダウンロードする．続いて requestAttributeSK() を実行して $SK_{\mathcal{A},a_{DR}}$ を生成し，スマートコントラクトに属性 $\mathcal{A}$ をスマートコントラクトに登録する．その後 $SK_{\mathcal{A},a_{DR}}$ を DR の公開鍵 $ECDSA_pub$ で暗号化し，DR 宛の送金トランザクションに埋め込んでアップロードする．最後にトランザクション ID $Txid_{SK_{\mathcal{A},a_{DR}}}$ を DR にレスポンスとして送信する．

RevokeAttribute$\mathfrak{B}$() DR に割り当てた属性を失効させる関数．スマートコントラクトに書き込まれた属性の発行フラグを消去する．鍵失効とは異なるため，暗号論的に前方秘匿性はない．

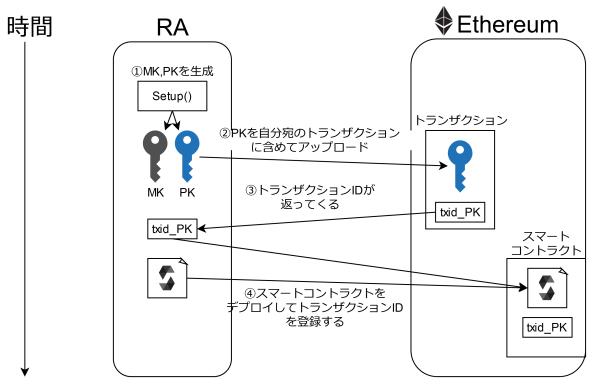

図 1 Setup$\mathfrak{B}$ の概要

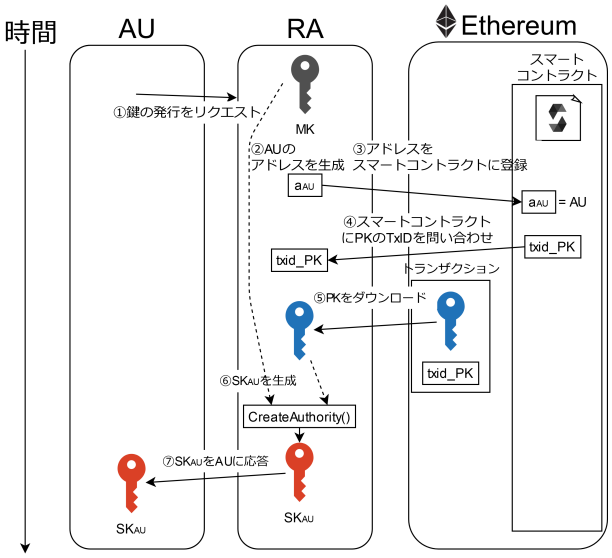

図 2 CreateAuthority$\mathfrak{B}$ の概要

4. 評価

本章では初めに処理時間や通信時間について計測を行い，続いて Ethereum の mainnet を用いた際の運用コストについて確認する．

4.1 処理時間の評価

一般にブロックチェーン技術はブロックの生成に時間がかかることが知られており，etherscan[*7]の公開している情報によると，今年の Ethereum のブロック生成時間の平均は 13.1 秒であった．ブロックチェーンの利用で処理時間が極端に遅くなってしまうことは望ましくないため，ブロックチェーン処理，計算処理，通信に分けて処理時間の計測を行った．VULTR[*8]のクラウドコンピューティングサービスを利用して表 2 に示す通りインスタンスを設置した．

*7 https://etherscan.io
*8 https://www.vultr.com/

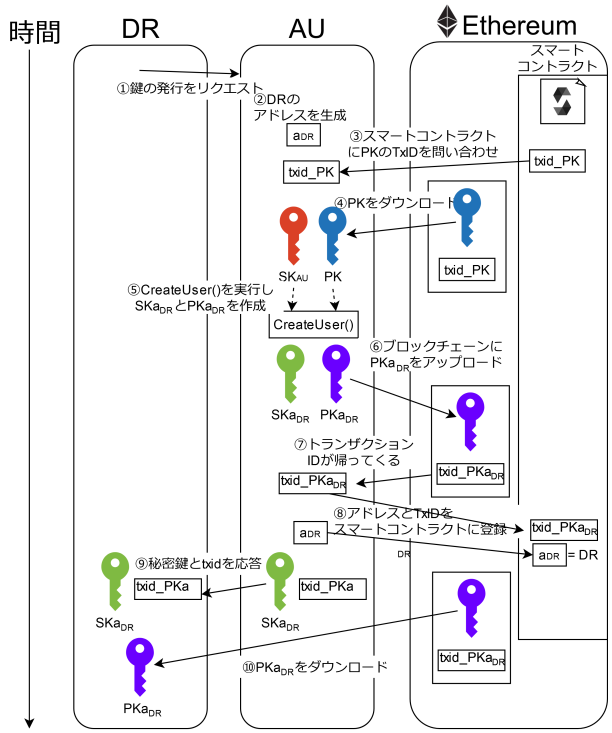

図 3 CreateUserℬ の概要

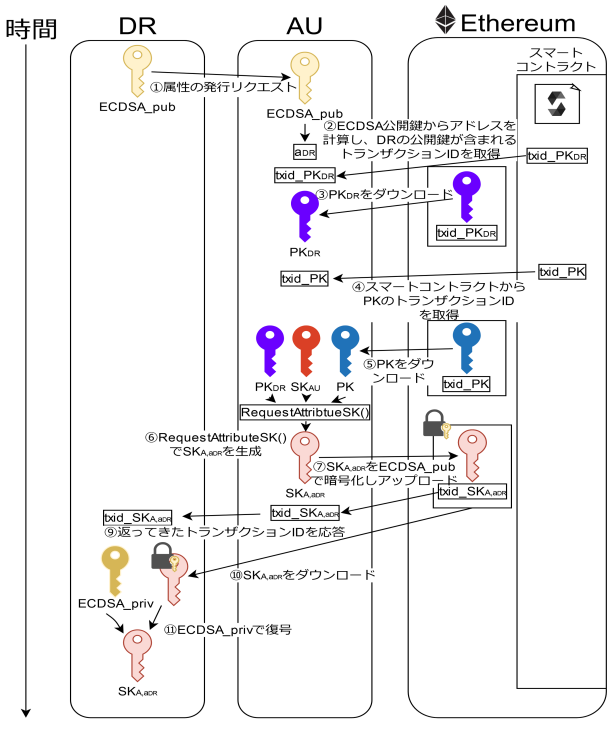

図 4 IssueAttributeℬ の概要

各インスタンスに表 3 の環境を構築し，それぞれ 10 回の計測を行った結果を表 4 に示す．この結果からブロックチェーンに最も多くアクセスを行うセットアップの処理においても平均 27 秒で処理が終わるということが分かった．プライベートチェーンにおけるブロック採掘速度は平均 4.6 秒と mainnet の 2.8 倍程度高速であったため，mainnet での処理時間はさらに長くなると予想される．一方で実行頻度の高い暗号化や復号の処理に関しては，ブロックの採掘を待たずに結果が得られるゲッターメソッドを用いることで他の処理と比べて高速に動作し，現実的な時間で暗号化や復号が行えることも分かった．

表 2 ノードの配置

RA	東京
AU1	東京
AU2	ニュージャージー
DR	東京
DO	東京

表 3 計測に使用した機器の仕様

CPU	Intel Skylake CPU 2 Core
RAM	4GB
OS	CentOS Linux release 8.2.2004 (Core)

表 4 処理時間 ($\bar{x}$:平均、s:標準偏差)

	ブロックチェーン		計算		通信	
処理	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
セットアップ	25.107	6.754	2.004	0.100	-	-
AU 作成 (日)	12.168	3.607	1.171	0.061	0.015	0.001
AU 作成 (米)	10.850	5.158	1.121	0.049	0.469	0.009
DR 作成 (日)	10.257	7.905	1.059	0.064	0.012	0.001
DR 作成 (米)	9.425	4.656	1.154	0.121	0.517	0.154
属性作成	6.710	2.998	1.205	0.187	-	-
属性委任	9.037	4.835	9.3×10^{-5}	2.5×10^{-5}	-	-
属性発行 (日)	6.426	3.153	1.068	0.046	0.008	0.001
属性発行 (米)	6.581	3.795	1.186	0.081	0.460	0.012
属性失効	11.683	7.659	0.957	0.050	-	-
暗号化 (日)	-	-	0.230	0.013	0.008	0.00045
暗号化 (米)	-	-	0.697	0.020	0.457	0.009
復号	1.094	0.176	0.445	0.024	-	-

4.2 運用コストの評価

Ethereum などのブロックチェーンを利用したシステムをパブリックチェーン上で運用する場合，トランザクションの送信やスマートコントラクトの実行を行う際に手数料を支払う必要がある．ここ最近 Ethereum の手数料の高騰が問題となっているという背景もあり，運用に多額の手数料が必要になることは望ましくないため，手数料の見積もりを行った．3.3 節で示す処理で送信されたトランザクションの Gas Used を調査し日本円に換算したものを表 5 に示す．実験の結果，スマートコントラクトのデプロイが最も手数料が高く 17,704 円で，次いでスマートコントラクトの実行が平均で 902 円，最も手数料が安いのが鍵のアップロードで表 6 に示す鍵長の場合は平均 360 円となり，また Encrypt と Decrypt に関しては手数料が発生しないということが分かった．

5. 考察

Bramm らの BDABE を Ethereum に実装するにあたっ

表 5 関数の実行コスト

関数	処理	Gas	円
Setup()	PK の送信	42,744	477 円
	SmartContract のデプロイ	1,602,746	17,704 円
CreateAuhtority()	AU の登録	47,053	518 円
CreateAttribute()	属性の作成	111,823	1,239 円
CreateUser()	PK_a の送信	27,148	297 円
	DR の登録	158,674	1,758 円
IssueAttribute()	属性の発行	71,247	784 円
	$SK_{\mathcal{A},a_{DR}}$ の送信	28,256	307 円
RevokeAttribute()	属性失効	19,269	213 円
Encrypt()	-	-	-
Decrypt()	復号可否の照会	0	0 円

表 6 BDABE における鍵長

鍵	鍵長
MK	61 バイト
PK	1368 バイト
SK_{AU}	446 バイト
PK_{DR}	385 バイト
SK_{DR}	342 バイト
$PK_{\mathcal{A}}$	1058 バイト
$SK_{\mathcal{A},a_{DR}}$	374 バイト

て，Bramm らの方式にはない独自の実装を行っている部分がある．そこで本章では，独自の実装を行った部分についての解説と考察を行う．

5.1 暗号化・復号

Bramm らの暗号化と復号の実装は plaintext $M \in G_T$ を暗号化や復号する物で，ファイルなどを扱う際の実装については触れられていなかった．そのため本研究では，$M \in G_T$ の代わりに共有値 $r \in G_T$ を無作為に選択し，ファイルの暗号化の際には共有値の SHA256 ダイジェストを一時的な鍵として AES256-GCM アルゴリズムを使用するハイブリッド暗号として実装を行った．これにより，アクセスポリシーに論理和が含まれる際の暗号文サイズが肥大化を防ぐことができ，また GCM モードの利用によって暗号文の完全性が確認できるようになった．

5.2 DR の公開鍵の管理方法について

DR の公開鍵 $PK_{a_{DR}}$ は **CreateUser()** により生成されるが，この公開鍵をどのように管理・配布するかについて文献 [4] では十分な記載が無かった．もし $PK_{a_{DR}}$ を DR が管理し，**RequestAttributeSK()** で属性の秘密鍵を生成する際にユーザから受け取った $PK_{a_{DR}}$ に対応する秘密鍵を AU が生成する実装をした場合，複数の DR による結託攻撃が可能となる．**Decrypt()** では，同一の $PK_{a_{DR}}$ から生成された属性鍵の集合を使わないと復号できないようにしており，それによって異なるユーザの属性鍵を組み合わせて権限を越える復号ができないようにしている．しかしながら，ユーザが自身の $PK_{a_{DR}}$ を提示する実装にした場合，ユーザが別のユーザ DR' の公開鍵 $PK_{a_{DR'}}$ を送信し，その後 DR' の属性鍵と組み合わせて復号処理をすることで DR と DR' の任意の属性の AND の条件で復号が可能となってしまう．本研究では **CreateUser()** で生成した際にユーザに $PK_{a_{DR'}}$ を渡すのではなく，Ethereum 上に送信して管理する実装とした．これにより，**RequestAttributeSK()** の際に DR に正規の公開鍵 $PK_{a_{DR}}$ に対応する属性鍵しか発行されることが無くなり，上記の結託攻撃を防ぐことができる．

5.3 属性失効

Bramm らは属性毎に Asset と呼ばれるトークン発行機能を利用して，その残高を確認することで DR がどの属性を持っているかを管理していた．そのため，属性を執行させる際には DR の秘密鍵を用いて Asset の残高を AU に返却する必要があり，その処理のために DR の RSA キーペアを AU が管理する必要があった．しかし，本研究ではスマートコントラクトを用いてユーザや属性の管理を行っており DR とそれを作成した AU は 1 対 1 で記録されている．そのため，AU は AU の秘密鍵のみで自身が管理する DR の属性を執行させることができ，DR の ECDSA キーペアを保持する必要がなくなった．安全性に関しては Bramm らの方式と同様で，この鍵執行方式には暗号論的に前方秘匿性が無い．

6. 結論

本稿ではブロックチェーンに基づく分散属性ベース暗号を Ethereum 上で実現する方法について検討し，実装・評価を行った．その結果，BDABE の方式を Ethereum 上で運用する際にかかるコストや処理時間が現実的であることがわかった．

今後の課題は Müller らの分散属性暗号以外の複数の鍵発行センタをサポートする方式，特に中央機関を必要としない方式について同様にブロックチェーン上で実装して評価することが挙げられる．

参考文献

[1] Bethencourt, J., Sahai, A. and Waters, B.: Ciphertext-Policy Attribute-Based Encryption, *2007 IEEE Symposium on Security and Privacy (SP '07)*, pp. 321–334 (online), DOI: 10.1109/SP.2007.11 (2007).
[2] Sahai, A. and Waters, B.: Fuzzy identity-based encryption, *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, pp. 457–473 (2005).
[3] Müller, S., Katzenbeisser, S. and Eckert, C.: Distributed attribute-based encryption, *International Conference on Information Security and Cryptology*, Springer, pp. 20–36 (2008).
[4] Bramm, G., Gall, M. and Schütte, J.: BDABE-Blockchain-based Distributed Attribute based Encryption., *ICETE (2)*, pp. 265–276 (2018).