CL-002

属性ベース暗号を用いたスマートコントラクトのアクセス制御
An Access Control Method Based on Attribute-Based Encryption for Smart Contracts

佐伯　美緒 *　　小島　英春 *　　矢内　直人 *　　土屋　達弘 *
Mio Saiki　Hideharu Kojima　Naoto Yanai　Tatsuhiro Tsuchiya

1. はじめに

1.1. 研究背景

スマートコントラクトはブロックチェーン上でプログラムを作成・利用する技術であり，その主なプラットフォームとして Hyperledger Fabric が様々な場所で利用されている．例えば，研究データ管理用基盤である Open Science Chain [1] や監査データ運用基盤である HF-Audit [2] などが挙げられる．

Hyperlder Fabric を用いて作成されるスマートコントラクトでは，扱うデータが，データベース上に平文で保存されるため，適切にアクセス制御を行う必要がある．例えば，第三者に対し外部からデータベースの閲覧を許してしまうと，保存されているデータの内容が容易に理解できることから，データ漏洩が起こりえる．そこで保存されるデータを暗号化することで，もしデータベース内のデータが窃取されたとしても，内容が理解できない様にすることが望ましい．保存するデータを外部から隠蔽する方法として，ユーザが自身で保存するデータを暗号化し，データベースに保存することが考えられる．この場合，問題となるのは，暗号化したユーザが用いた鍵をどのように管理するかである．例えば，ユーザ 1 が鍵 key1 を利用して暗号化したデータ d_{key1} をデータベースに保存したり，ユーザ 2 が鍵 key2 を用いて暗号化したデータ d_{key2} をデータベースに保存するなどの様に，各ユーザが自身の鍵を用いると，そのデータを復号できるのは，その鍵を保有しているユーザだけである．暗号化したデータを利用したいユーザが暗号化に利用した鍵をどの様に手に入れるかや，どのデータとどの鍵が対応しているのか，利用された鍵をどのように管理するのかなど，鍵の管理が非常に煩雑となる．

1.2. 貢献

本研究では，ユーザの属性情報を通じて暗号化することで同じ属性を持つユーザ間での情報共有を支援する属性ベース暗号を用いることで，Hyperledger Fabric 上でデータの暗号化と復号を効率的に管理できるアクセス制御技術を提案する．これを実現する方法はいくつかあり，サーバ側のチェーンコードで暗号化復号を実現する方法や，クライアント側で暗号化復号を行う方法が考えられる．本研究では，クライアント側での暗号化復号を行うことにより，外部から台帳の情報を窃取されたとしてもその内容を隠蔽することが可能となる．また，実装では，JavaScript を用いて既存の属性ベース暗号ライブラリを利用する方法を示す．これにより，JavaScript 上で属性ベース暗号機能を実現することが可能である．

1.3. 関連研究

ブロックチェーンへ暗号技術を適用することでユーザのプライバシーを保護する研究は多くなされている．例えば，ゼロ知識証明を用いた研究 [3, 4, 5] や Trusted Execution Environment を用いた研究 [6, 7] が知られている．しかしながら，これらの研究ではデータそのものを暗号化した状態で運用するような状況は想定されていない．

本研究に近いものとして，秘密計算を用いることでデータを暗号化した状態でブロックチェーン上に格納する成果 [8, 9] がある．とくに Benhamouda ら [8] はプライベートデータのサポートを Hyperledger Fabric 上に秘密計算を統合実装することで，オークションプロトコルなどプライベートデータの利活用も検討している．しかしながら，Benhamouda らの成果では暗号化したデータを複数人で共有するようなモデルは想定されておらず，本研究ではデータの効率的な運用管理が主な点であることが異なる．

多人数環境で効率的な鍵管理を考慮した研究としてはブロックチェーンを基盤として暗号化状態でデータ検索が可能な鍵集約検索可能暗号 [10] を構成した成果 [11] もある．しかしながら，これはブロックチェーンを要素技術として用いた研究であり，ブロックチェーンでデータを暗号化運用する観点とは異なる．

なお，著者らの以前の研究 [12] では，チェーンコード自体が鍵を持ち，暗号化と復号を行う方法を提案し，実装した．しかし，チェーンコードは外部から閲覧可能なことから，復号方法が外部に漏洩する恐れがある．

2.Hyperledger Fabric およびアクセス制御

本節では本稿における主な問題設定を述べる．まず Hyperledger Fabric の技術的背景を述べたのち，詳細な問題設定を述べる．

2.1. 技術的背景

Hyperledger Fabric は，Linux Foundation により提供されている Hyperledger プロジェクトの 1 つであり，スマートコントラクトを実現するプラットフォームである．Fabric 以外にも，4 つのフレームワークと 5 つのツールが含まれている．コントラクトはチェーンコードと呼ばれ，様々な言語で実装可能である．Fabric では，そのネットワーク内に台帳を操作する Organization と呼ばれる組織があり，その組織は Channel に属している．この Channel に属する Organization は同じ Channel に属する台帳の操作を行うことができる．Channel には，Organization が管理する 1 つ以上の Peer と呼ばれるノードが存在し，この Peer は，Commitment Peer，Endorsement Peer などいくつかの種類がある．Com-

*大阪大学大学院情報科学研究科, Graduate School of Information Science and Technology, Osaka University

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

mitment Peer は台帳を保有しており，Peer 間で合意が得られたトランザクションを自身の台帳に反映する．Endorsement Peer は，Commitment Peer の機能に加えて，チェーンコードの実行や Endorsement Policy を保有している．Endorsement Policy では，トランザクションがどの Organization によって承認されるかについて定義されている．また，Channel には，Orderer と呼ばれるトランザクションの実行順序を決めるノードが存在している．台帳は，データベースとして存在しているが，データの更新や読込などは，Peer 間で合意が得られたトランザクションによって行われる．Channel に所属する Organization に登録されたユーザは，チェーンコードを介して，その Channel に存在する台帳を操作することができ，操作のトランザクションが Endorsement Peer によって合意が得られた場合に台帳の操作が行われる．各 Organization には，CA と MSP と呼ばれる Peer や Orderer，ユーザが各 Organiztion が所属する Channel に参加する際に認証を行うノードが存在する．

Hyperledger を用いて利用されるアプリケーションは，様々なものがあり，スーパーマーケットや金融企業，サプライチェーン，健康情報管理などの領域で利用されている†．

2.2. 問題設定

本研究では，外部に漏れると問題となるプライベートデータを Hyperledger Fabric 上で暗号化することで，データの窃取や Endorsement Policy の不適切な設定によるデータの漏洩を防ぐことを目的とする．

一般に，データ自体は Peer に存在するデータベースに保存されており，そのデータベースを閲覧する権限を持っていれば，誰でもデータベース内のデータを閲覧することが可能である．このとき，データベースが攻撃を受け，内部の情報を窃取されたとしても，その内容を解読できない様にしてデータの漏洩を防ぐことが必要である．さらにいえば，同じ Channel に所属する異なる Organization に自身の Organization のデータを閲覧されることも防ぐべきである．

直観的な対策は暗号技術によりデータそのものを保護することであるが，このような状況において鍵情報を適切に管理することは，処理の煩雑さの観点からしばしば容易ではない．例えば，サプライチェーンに参加する複数の企業が複数の Organization として同じ Channel に所属している場合，製品毎に，どの企業がどのデータを操作することを可能にするかを設定する必要があるが，製品や参加企業が多くなると Endorsement Policy の変更やその管理も多岐に渡ることが予想される．このとき 1.1 節で述べた様に，単純に各ユーザや各企業が独自に鍵を利用して暗号化すると，どのデータがどの鍵を持つユーザと紐づいているか，また，各ユーザにどの鍵を配布したかの管理が煩雑になると考えられる．

すなわち，本稿で取り上げる技術要件は，Hyperledger Fabric 上で以下を実現することにある．

データの秘匿性: 各 Organization により Hyperledger Fabric 上に保存されるデータは暗号化されており，該当する Organization に属するユーザ以外が平文データを取得・閲覧することはできない．

データ管理の効率性: 各 Organization により Hyperledger Fabric 上に保存されるデータは，複製・再暗号化されることなく，該当する Organization に属する任意のユーザがアクセスできる．

なお，本稿では内部結託者がデータを意図的に漏洩させるような攻撃，すなわち，ある Organization に属するユーザが暗号化されたデータを復号し他の Organization に横流しするような攻撃は考えない．また，使われている暗号技術が危殆化するような状況は想定しないものとする．

3. 提案手法

本節では前節で述べた要件を満たすアクセス制御技術を提案する．主な着想はデータを暗号化する際に属性ベース暗号 [13] を用いることである．まず属性ベース暗号の概要について述べたのち，プロトコルの構成について述べる．

3.1. 要素技術：属性ベース暗号

属性ベース暗号 (Attribute-Based Encryption) [13] は, 暗号化する際に暗号文を復号できる対象としてユーザが持つ属性情報を利用する技術である．このとき，復号できるか条件を指定するアクセス構造を鍵に紐づける KP-ABE [14] と暗号文に紐づける CP-ABE [15] があるが，いずれの方式においてもアクセス構造に含まれるような属性を満たさないユーザは，その暗号文を復号することができない．すなわち，ある属性条件を満たす不特定多数のユーザのみに復号することを許可することが可能となる．このような背景から属性ベース暗号の研究は盛んにおこなわれている [16, 15, 17, 18, 19, 20]．

直観として，データの暗号化および復号に属性ベース暗号を取り入れることで，鍵の管理を容易にすることが可能となる．属性ベース暗号では属性に基づいて暗号化復号を行うことから，各ユーザが属する Organization を属性として扱うことで，各ユーザや企業が独自に鍵の管理を行う必要がなくなる．さらに，各ユーザにおいて暗号化と復号を行うために必要な属性として複数の属性を設定できるため，単一のユーザで複数の Organization のデータを利用することも可能となる．例えば，(Org1 or Org2) という属性を用いて暗号化した場合，Org1 の Organization と Org2 の Organization がそのデータを復号することができる．この様に，復号できる Organization を属性として指定することによって，鍵とユーザの紐づけが不要になることや，単一の属性を通じて不特定多数のユーザに対しアクセス制御が可能になることから，効率的な管理が期待できる．

3.2. アクセス制御技術の構成

属性ベース暗号を用いて暗号化と復号を行うことで，煩雑な鍵の管理を容易にするアクセス制御技術を以下に述べる．大きな流れとしては，データを追加，更新を行う際に，そのデータを属性と共に暗号化し，読込みを行う際には，そのデータを復号する．このとき，ク

†https://www.hyperledger.org/

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

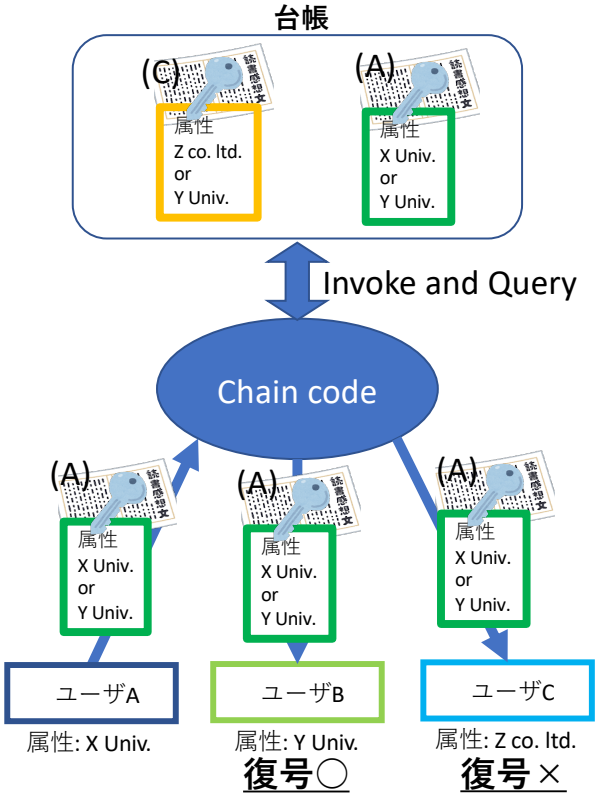

図 1: 提案手法の動作例

ライアント側で暗号化と復号を行うことで，サーバ側を変更することなく属性ベース暗号を利用することが可能となる．

図 1 に提案方法の動作例を示す．ユーザ A，ユーザ B，ユーザ C は，それぞれ属性 X Univ.，Y Univ.，Z co. ltd. を保有している．また，台帳に存在するデータ (A) とデータ (C) はそれぞれ属性 (X Univ. or Y Univ.) と属性 (Z co. ltd. or Y Univ.) を用いて暗号化されたデータである．ユーザ A が属性ベース暗号の属性 (X Univ. or Y Univ.) を用いてある平文を暗号化する際，チェーンコードを実行し，台帳にデータ (A) を保存する．のちに，ユーザ B がデータ (A) を読込んだ場合，データ (A) はユーザ B 自身の属性を含む属性を利用して暗号化されているため，ユーザ B は復号することが可能である．一方，ユーザ C はデータ (A) を読込んだとしても，属性が異なるため復号することは不可能である．同様に，データ (C) に関してはユーザ A の属性が含まれていないため，ユーザ A にとって復号することは不可能である．この様に，暗号化する際に，単一の暗号文から復号できるユーザの属性を指定することが可能になるため，前述した鍵の管理の煩雑さは解消できる．

なお，属性ベース暗号ではマスタ鍵を用いて各ユーザの鍵を生成する中央管理サーバが必要となる．このマスタ鍵を用いた鍵生成処理は Channel の管理者が行うものとする．すなわち，Hyperledger Fabric は管理者にアクセスが認められたユーザのみが利用できる仕組みとなっており，本提案手法では管理者が鍵生成も兼ねる．

```
1:#include <napi.h>
2:#include <string>
3:#include <openabe/openabe.h>
4:#include <openabe/zsymcrypto.h>
5:class JsOABE:public Napi::ObjectWrap<JsOABE>{
6: public:
7:  static Napi::Object Init(Napi::Env env,
8:           Napi::Object exports);
9:  JsOABE(const Napi::CallbackInfo &info);
10:  ~JsOABE();
11:  Napi::Value encrypt(
12:           const Napi::CallbackInfo &info);
13:  Napi::Value decrypt(
14:           const Napi::CallbackInfo &info);
15:  Napi::Value keygen(
16:           const Napi::CallbackInfo &info);
17: protected:
18:  oabe::OpenABECryptoContext *abe;
19: private:
20:  static Napi::FunctionReference constructor;
21:};
```

図 2: OpenABE を利用するためのクラス

```
1:JsOABE::JsOABE(const Napi::CallbackInfo& info):
2:    Napi::ObjectWrap<JsOABE>(info){
3: auto env = info.Env();
4: oabe::InitializeOpenABE();
5: std::type=
6:    info[0].As<Napi::String>().ToString();
7: std::mpk=
8:    info[1].As<Napi::String>().ToString();
9: std::msk=
10:    info[2].As<Napi::String>().ToString();
11: this->abe=
12:    new oabe::OpenABECryptoContext(type);
13: this->abe->importPublicParams(mpk);
14: this->abe->importSecretParams(msk);
15:}
```

図 3: コンストラクタ

4. 実装

本研究では，属性ベース暗号を C++で実装した OpenABE[21] を利用して，チェーンコードによって保存されるデータに属性ベース暗号を適用して保存する方法を実装する．Hyperledger Fabric 2.1.1 のサンプルコードである Fabcar を対象にし，使用言語は JavaScript である．Fabcar では，台帳に保存されている情報を取り出すための query.js，新しく情報を追加する invoke.js があり，それぞれを対象に，query.js には，取り出した情報を復号する実装，invoke.js には保存する情報を暗号化する実装を行う．まずはじめに，JavaScript から OpenABE を利用するための JavaScript モジュール `napi-abe` を作成し，その後に，`napi-abe` を利用して，暗号化，復号の処理を行うことができる invoke.js と query.js の実装を行う．次節では，どのように JavaScript から OpenABE を利用するかについて説明する．

4.1.napi-abe モジュール作成

JavaScript を用いて OpenABE の暗号化，復号などの機能を利用するために，JavaScript から C++で作成されたライブラリを利用することが必要となる．そこで，N-API ライブラリを利用して，図 2 のような OpenABE ライブラリを利用するクラス `JsOABE` を作成し，JavaScript から `napi-abe` モジュールとして OpenABE

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

```
1:Napi::Value JsOABE::keygen(
2:   const Napi::CallbackInfo &info){
3: auto env = info.Env();
4: std::string attrs =
5:   info[0].As<Napi::String>().ToString();
6: std::string key =
7:   info[1].As<Napi::String>().ToString();
8: this->abe->keygen(attrs,key);
9:}
```

図 4: 鍵の生成を行うメソッド

```
 1:Napi::Value JsOABE::encrypt(
 2:   const Napi::CallbackInfo &info){
 3: auto env = info.Env();
 4: std::string attrs =
 5:   info[0].As<Napi::String>().ToString();
 6: std::string plain =
 7:   info[1].As<Napi::String>().ToString();
 8: std::string cipher="";
 9: this->abe->encrypt(attrs, plain, cipher);
10: return Napi::String::New(env, cipher);
11:}
```

図 5: 暗号化するメソッド

```
 1:Napi::Value JsOABE::decrypt(
 2:   const Napi::CallbackInfo &info){
 3: auto env = info.Env();
 4: std::string attrs =
 5:   info[0].As<Napi::String>().ToString();
 6: std::string cipher =
 7:   info[1].As<Napi::String>().ToString();
 8: std::string plain="";
 9: if(this->abe->decrypt(attrs, cipher, plain)){
10:  return Napi::String::New(env, plain);
11: }
12: return Napi::String::New(env, "");
13:}
```

図 6: 復号するメソッド

を利用する．このクラスで利用されている型 Napi::Env や Napi::CallbackInfo，Napi::Value などを利用して，JavaScript で使用する型と C++で使用する型を変換する．コンストラクタや鍵作成，暗号化，復号に加えて，JavaScript においてこのライブラリが読み込みされる際に初期化するメソッド Init が必要である．Init メソッドでは，JavaScript からこのクラスで定義したメソッドを公開する処理を行っている．クラスメンバとして，OpenABE をライブラリの機能を実際に提供するメンバ abe が定義されている．

図 3 のコンストラクタでは，5 行目から 10 行目までの処理で，引数 info に含まれる 3 つの文字列を取り出している．1 つは，OpenABE ライブラリで利用される型，残りの 2 つは，その型にあったパラメータである．11 行目，12 行目においてクラスメンバ abe をインスタンス化し，13 行目，14 行目でパラメータを設定している．

実際に鍵生成，暗号化，復号を行う実装を図 4，図 5，図 6 に示している．図 4 では，引数 info に含まれる値は，2 つあり，属性ベース暗号の属性を表す文字列とその鍵 ID を表す文字列である．それぞれを 4 行目から 7 行目で変数 attrs と key に取り出し，8 行目で鍵生成を行う．図 5 では，引数 info に含まれる値は，属性を表す文字列と暗号化したい平文である．それぞれを 4 行目から 7 行目までで変数 attrs と plain に取り出す．9 行目の abe->encrypt(attrs,plain,cipher); が実行されると変数 cipher に，plain を暗号化したものが格納される．その後，JavaScript で利用できる型 Napi::Value に変換して cipher を返す．図 6 では，暗号化とは逆の作業をするため，引数 info には，属性と暗号化されたものが含まれる．それぞれを 4 行目から 7 行目までで変数 attrs と cipher に取り出し，9 行目で this->abe->decrypt(attrs, cipher, plain) を実行する．返り値が真であれば，復号ができたことを表し，変数 plain に格納されている元の平文を返す．そうでなければ，空の文字列を返す．

4.2. 属性ベース暗号機能の追加

本節では，JsOABE クラスをチェーンコードにアクセスするクライアントで利用し，属性ベース暗号を用いてエントリの暗号化，復号を行う方法について述べる．対象とする Fabcar には台帳から情報を取り出すための query.js と台帳にエントリを追加する invoke.js があり，それらに保存するエントリの暗号化と復号を行う機能を追加する．それぞれの JavaScript ファイルの内容について簡単に説明し，エントリの読込や保存をする際にどのように機能を追加するかについて述べる．JsOABE クラスを利用するためには，各スクリプトファイルにおいて，

```
const napi = require("bindings")("napi-abe")
```

を用いて，napi-abe モジュールを呼び出し，JsOABE クラスを図 7 や図 8 で利用される変数 abe としてインスタンス化を行う必要がある．

invoke.js は台帳に新しくエントリを追加したり，更新したりする際に利用されるクライアント側のスクリプトである．エントリには，Key-Value ストアにおける Key を表す carNumber があり，Value を表す情報は JSON 形式で

```
{
 color:色
 make:メーカー
 model:モデル
 owner:所有者
}
```

となっている．エントリを追加するにはメソッド

```
await contract.submitTransaction('createCar',
'CAR12', 'Maker1', 'newModel', 'Black', 'Tom');
```

を実行する．これは，'Tom' が所有するメーカーが'Maker1' の色が'Black' の'newModel という車を CAR12 としてエントリを保存することを意味している．実際には，サーバ側のチェーンコードである createCar が呼び出されている．これらの情報のうち，所有者を暗号化するために，所有者'Tom' を暗号化する機能を追加する場合は図 7 の通りとなる．図 7

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

```
1:var keys = JSON.parse(
2:  fs.readFileSync('./keys.json'));
3:var type="KP-ABE";
4:const abe=new napi.JsOABE(type,
5:  keys[type]["mpk"],keys[type]["msk"]);
6:abe.keygen("(attr1 or attr2)", "key0");
7:var enc=abe.encrypt("attr1",'Tom');
8:await contract.submitTransaction('createCar',
9:  'CAR12', 'Maker1', 'newModel', 'Black', enc);
```

図 7: 暗号化を行う処理の例

```
 1:var keys = JSON.parse(
 2:  fs.readFileSync('./keys.json'));
 3:var type="KP-ABE";
 4:const abe=new napi.JsOABE(type,
 5:  keys[type]["mpk"],keys[type]["msk"]);
 6:abe.keygen("(attr1 or attr2)", "key10");
 7:var result =
 8:  await contract.evaluateTransaction(
 9:    'queryCar','CAR12');
10:var car = JSON.parse(result);
11:var owner=abe.decrypt('key10',car['owner']);
```

図 8: 復号を行う処理の例

は，1 行目，2 行目において属性ベース暗号に必要なパラメータをファイルから読み込んでいる．4 行目，5 行目において，属性ベース暗号を扱うための変数 abe をインスタンス化し，6 行目で鍵の生成を行う．この時，属性が attr1 か attr2 が与えられる場合に限り復号が可能とするため，属性には (attr1 or attr2) を与えている．7 行目において，属性 attr1 を与え，文字列 Tom を暗号化し，変数 enc に代入している．その後に，8 行目，9 行目においてエントリを保存している．これにより，所有者が暗号化されたエントリが保存される．

query.js　は台帳に保存してある情報を取り出す際に利用されるクライアント側のスクリプトである．実際には，サーバ側のチェーンコードに定義されているメソッド queryCar が呼び出されており，そのメソッドが実行された結果が返ってくる．queryCar は引数として Key-Value ストアにおける Key である carNumber を取り，その carNumber と紐づけられている情報を返すことによってエントリを取り出すことができる．例えば，carNumber が CAR0 のエントリを取り出す場合は，

```
var result =
  await contract.evaluateTransaction(
  'queryCar','CAR0');
```

というメソッドを用いることで result に情報が得られる．得られた情報は JSON 形式であるため

```
var car = JSON.parse(result);
```

を実行することにより car を連想配列として扱うことが可能となる．もし，所有者情報である car['owner'] が前述した **invoke.js** を用いて暗号化されていた場合，

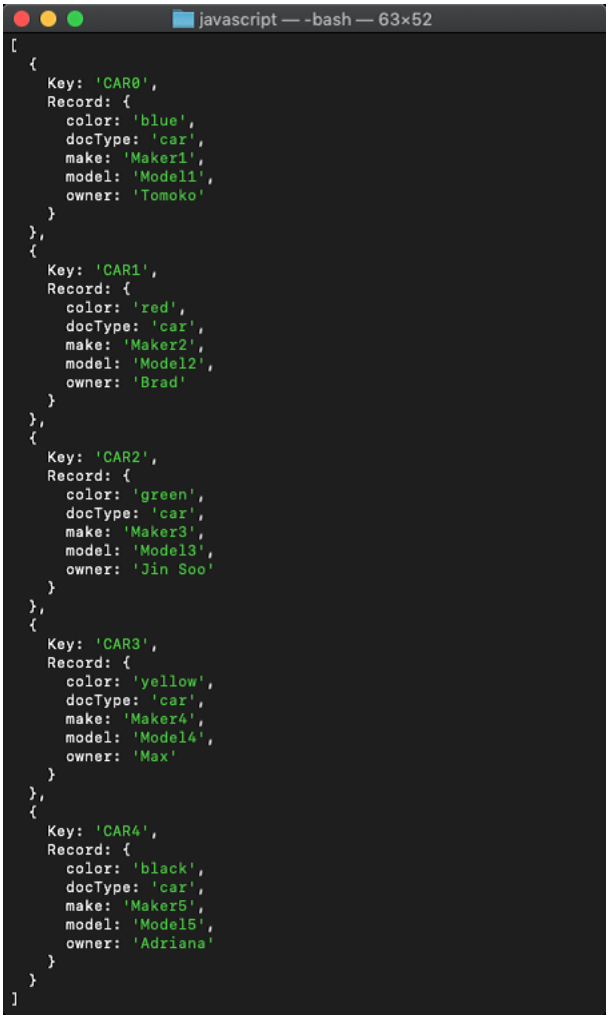

図 9: Fabcar に保存されている初期情報

それを復号する必要がある．その復号の機能を追加するプログラムは図 8 の通りである．1 行目から 6 行目までは，**invoke.js** と同じ処理であり，パラメータの読込から鍵の生成を行う．7 行目から 10 行目まででは，carNumber が'CAR12' のエントリを取得して変数 car に代入している．この時，car['owner'] は暗号化されているため，11 行目において，復号し変数 owner に代入している．この時，6 行目での鍵生成において，属性が同じであることが重要である．

5. 実験

本節では，実際にエントリーの追加や読込を行った際に，どのような情報が保存されるかについて説明を行う．Fabcar で保存されている JSON 形式のデータは図 9 の通りである．図 9 に示されている Key は Key-Value ストアの Key であり，**query.js** や **invoke.js** で利用されている carNumber を表している．Record は，その Key に紐づけられている情報を表している．前節では，この一部である owner のみを暗号化した例を挙げたが，本節では，この情報をまとめて暗号化し，carNumber を紐づけて保存する．こうすることで，外部から台帳の情報を窃取されたとしてもすべてが暗号化されてい

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

図 10: 初期情報を暗号化した後の保存されている情報

るため，情報の取得が不可能となる．まずは，Fabcar で初期値として保存されている情報を一旦暗号化する JavaScript ファイルを実行し，保存されているデータを平文から全て暗号化した情報に更新する．その際，データを更新するにあたって，サーバ側のチェーンコードに新しくメソッド addCar を追加した．

5.1. 保存されている情報の変更

Fabcar に初期値として保存されている情報は，平文であるため，それらを暗号化された情報に更新する．すべての情報を一旦取り出し，それぞれの情報を JSON.stringify メソッドと Buffer.from メソッドを用いて文字列に変換し，それを暗号化する．その手続きを **initABE.js** として作成し，それを実行することで初期値の情報を暗号化する．図 11 のプログラムは，図 9 として保存されている情報を暗号化する．1 行目から 6 行目までは，暗号化を行う準備を行っている．7 行目から 16 行目までは，情報を読込それを暗号化して保存している．14 行目，15 行目ではサーバ側のチェーンコードのメソッド addCar を使用して，暗号化した情報を保存している．addCar は，引数として Key である carNumber と暗号化された情報を取り，それを図 10 のように保存する．図 9 の Key:’CAR0’ の Record を暗号化したものは，図 10 の Key:’CAR0’ の Record である．

```
1:var keys = JSON.parse(
2:  fs.readFileSync('./keys.json'));
3:var abetype="KP-ABE";
4:const abe=new napi.JsOABE(type,
5:   keys[type]["mpk"],keys[type]["msk"]);
6:abe.keygen("(attr1 or attr2)", "key10");
7:for (let i = 0; i < 10; i++) {
8:  var result =
9:     await contract.evaluateTransaction(
10:       'queryCar','CAR'+i);
11:  var car = JSON.parse(result);
12:  var data=Buffer.from(JSON.stringify(car));
13:  var enc=abe.encrypt("attr1",data);
14:  await contract.submitTransaction(
15:     'addCar', 'CAR'+i, enc);
16:}
```

図 11: 保存されている初期情報を暗号化する処理

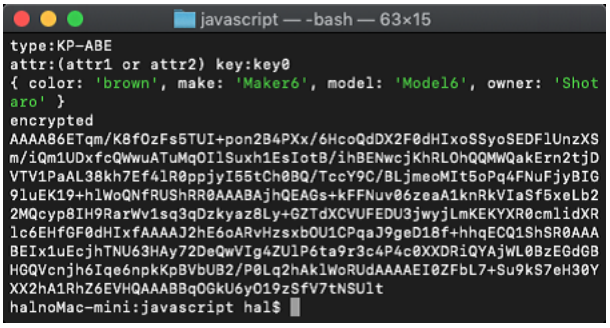

図 12: 新しく情報を追加

5.2. 情報の読込と更新

ここでは，暗号化した情報を更新する例とそれを復号する処理を行う例を挙げる．それぞれの処理を行うプログラムを **addABE.js** と **readABE.js** として作成し，実行結果を図 12 と図 13 に示す．図 12 は新しく carNumber が CAR6 の情報を追加した結果を示している．追加する情報は図 12 の 3 行目から 4 行目に示されており，以下の通りである．

```
{
 color:'brown'
 make :'Maker6'
 model:'Model6'
 owner:'Shotaro'
}
```

この情報を暗号化した結果は，図 12 の 5 行目 encrypted 以降の文字列である．この文字列を CAR6 の Record として保存する．その保存した値を読込み，復号した結果が図 13 である．CAR6 を読込み，その Record は図 13 の 3 行目以降の文字列であり，図 12 の 6 行目以降の文字列と同じである．これを復号したものが図 12 の decrypted 以降の文字列である．

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

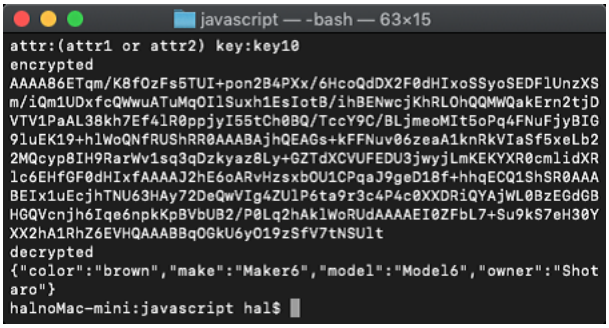

図 13: 情報の読込み

6. 考察

本節では 2.2 節で述べた要件を満たすことを以下に述べる．また，更なる拡張について述べる．

6.1. 技術要件の確認

まず提案手法がデータの秘匿性を満たすことを確認する．本稿の提案手法では，台帳に保存されたデータを窃取されたとしてもその内容を保護するために暗号化を行っている．この処理は図 5 における 9 行目が該当している．ここで用いられた `attrs` に該当する鍵を持つユーザでない限り，図 6 の復号処理を実行できない．すなわち，Hyperledger Fabric でのトランザクションの承認を設定している Endorsement Policy が不適切な設定であった場合に，本来はアクセスすることができない組織のユーザが台帳にアクセスすることが可能になったとしても，暗号化により平文そのものを知ることができず内容が保護される．これによりデータの秘匿性は保証される．

次に，データ管理の効率性を満たすことを確認する．台帳に保存されるデータは図 5 における 9 行目で，`attrs` を用いて暗号化される．このとき，図 4 で該当する `attrs` の鍵を所有する任意のユーザは，図 6 を通じて復号が可能となる．このとき，各ユーザにおいて共通な単一の暗号文を使用したまま，各ユーザは復号可能である．これにより，データ管理の効率性も保証される．

6.2. 更なる拡張の可能性

属性ベース暗号を Hyperledger Fabric に適用する方法のもうひとつの方法としてサーバ側でチェーンコード内に実装する方法が挙げられる．本研究において，クライアント側で暗号化を行う方法を選択した理由として実装が容易であることが挙げられる．各クライアントでは，暗号化と復号に用いるパラメータが同じにすることができれば，暗号化と復号は属性ベース暗号の実装である OpenABE が行う．しかし，サーバ側で暗号化を行う場合，Endorsement Peer による合意がなされない可能性がある．Endorsement Peer は，出力される情報に対して合意をとり，その結果合意できたものは台帳に保存されるが，合意が取れない場合は保存されない．そのため，Peer で暗号化を実行した際に，乱数を用いて暗号化を行う場合には，同じパラメータ，同じ平文を用いても暗号化した結果が異なる場合がある．そのような場合，合意が取れないため，台帳に保存されない．サーバ側で本手法を実現するためには，暗号化を乱数を用いずに，同じパラメータ，同じ平文の場合には，常に同じ暗号化した情報を出力する必要がある．この機能をサーバ側で行うにあたっては，今後の課題である．

7. まとめ

本研究では，属性ベース暗号を用いて，Hyperledger Fabric のサンプルコードである Fabcar を対象に，保存するデータを暗号化し，復号する手段を示した．また，C++で実装された属性ベース暗号ライブラリを JavaScript から利用する方法も示した．これらを利用して，実際に暗号化した情報の更新や読込を行い復号する方法を実装し，それらの実行結果を示し，暗号化，復号を行うことができることを示した．今後の課題としては，本研究で行ったクライアント側での暗号化，復号ではなく，サーバ側での暗号化，復号の実装と評価を行う予定である．また，暗号化したデータはそのままでは検索する際にその内容を直接調べることができないため，検索可能暗号を用いて，暗号化されたデータを検索することを可能にすることも考えている．

参考文献

[1] : Open Science Chain, `https://www.opensciencechain.org`.

[2] Lu, N., Zhang, Y., Shi, W., Kumari, S. and Choo, K.-K. R.: A secure and scalable data integrity auditing scheme based on hyperledger fabric, *Computers & Security*, Vol. 92, p. 101741 (2020).

[3] Poelstra, A., Back, A., Friedenbach, M., Maxwell, G. and Wuille, P.: Confidential Assets, *Proc. of FC 2019*, LNCS, Vol. 10958, Springer, pp. 43–63 (2019).

[4] Cecchetti, E., Zhang, F., Ji, Y., Kosba, A., Juels, A. and Shi, E.: Solidus: Confidential Distributed Ledger Transactions via PVORM, *Proc. of CCS 2017*, ACM, p. 701–717 (2017).

[5] Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P. and Vechev, M.: Zkay: Specifying and Enforcing Data Privacy in Smart Contracts, *Proc. of CCS 2019*, ACM, p. 1759–1776 (2019).

[6] Kosba, A., Miller, A., Shi, E., Wen, Z. and Papamanthou, C.: Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts, *Proc. of S&P 2016*, IEEE, pp. 839–858 (2016).

[7] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A. and Song, D.: *Proc. of EuroS&P 2019*.

[8] Benhamouda, F., Halevi, S. and Halevi, T. T.: Supporting private data on Hyperledger Fabric with secure multiparty computation, *IBM Journal of Research and Development* (2019).

[9] Zyskind, G., Nathan, O. and Pentland, A. S.: Decentralizing Privacy: Using Blockchain to Protect Personal Data, *Proc. of SPW 2015*, IEEE, pp. 180–184 (2015).

[10] Cui, B., Liu, Z. and Wang, L.: Key-Aggregate Searchable Encryption (KASE) for Group Data

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

Sharing via Cloud Storage., *IEEE Transactions on Computers*, Vol. 65, No. 8, pp. 2374–2385 (2016).

[11] Niu, J., Li, X., Gao, J. and Han, Y.: Blockchain-Based Anti-Key-Leakage Key Aggregation Searchable Encryption for IoT, *IEEE Internet of Things Journal*, Vol. 7, No. 2, pp. 1502–1518 (2020).

[12] Kojima, H. and Yanai, N.: A Chain Code Mechanism with Data Encryption onHyperledger Fabric, *Proc. ESORICS 2019, Poster* (2019).

[13] 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).

[14] Goyal, V., Pandey, O., Sahai, A. and Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data, *Proceedings of the 13th ACM conference on Computer and communications security*, pp. 89–98 (2006).

[15] Bethencourt, J., Sahai, A. and Waters, B.: Ciphertext-policy attribute-based encryption, *2007 IEEE symposium on security and privacy (SP'07)*, IEEE, pp. 321–334 (2007).

[16] Ostrovsky, R., Sahai, A. and Waters, B.: Attribute-based encryption with non-monotonic access structures, *Proceedings of the 14th ACM conference on Computer and communications security*, pp. 195–203 (2007).

[17] Chase, M.: Multi-authority attribute based encryption, *Theory of Cryptography Conference*, Springer, pp. 515–534 (2007).

[18] Chase, M. and Chow, S. S.: Improving privacy and security in multi-authority attribute-based encryption, *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 121–130 (2009).

[19] Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization, *International Workshop on Public Key Cryptography*, Springer, pp. 53–70 (2011).

[20] Lewko, A. and Waters, B.: Decentralizing attribute-based encryption, *Annual international conference on the theory and applications of cryptographic techniques*, Springer, pp. 568–588 (2011).

[21] Zeutro LLC.: The OpenABE library - open source cryptographic library with attribute-based encryption implementations in C/C++ (https://github.com/zeutro/openabe).

Copyright © 2020 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.