データ暗号化機能を組み込んだ チェーンコードの実装に関する研究

佐伯 美緒[1,a)]　小島 英春[1,b)]　矢内 直人[1]　土屋 達弘[1]

概要：スマートコントラクトプラットフォームである Hyperledger Fabric は、研究データ管理用基盤である Open Science Chain など様々な場所で利用されている．このバックエンドとなるデータベースは，キー・バリューストアのデータベースとなっておりデータは平文で保存されることから, 誰にでも閲覧が可能である．しかしながら, 保存するデータの閲覧を制限する場合は、データの暗号化を行った上で保存するなど、アクセス制限を設けることが望ましい. そこで，本研究では，Hyperledger Fabric のチェーンコード自体に暗号化および復号の機能を加えることで，アクセス制御を設けたチェーンコードを提案する．直観として，データにアクセス可能な権限を持つユーザのみ、データが利用可能になる．Hyperledger Fabric のサンプルコードに本手法を適用することで，実際にデータが暗号化されていることを確認する．また，提案方法をの性能を評価するために，データベースへの参照とデータの追加の実行時間を評価した．

キーワード：スマートコントラクト，チェーンコード，Hyperledger Fabric，データ暗号化

An Implementation of Chain Code Applications with Data Encryption

Mio Saiki[1,a)]　Hideharu Kojima[1,b)]　Naoto Yanai[1]　Tatsuhiro Tsuchiya[1]

Abstract: In a few years, Hyperledger Fabric as a platform of the smart contract is getting spread in the world to deploy many types of services. The type of its default data base is a key-value store. Stored data is plain texts in a default setting. In this case, stored data is easily taken from data base by any user. However, there are some types of data which only limited users can treat due to privacy, business secret, and so on. In this paper, we propose a method to conceal stored data by encrypting these stored data in chain code. At first, we indicate three types of data encryption for stored data. Then, we implement the proposed method to apply it to a sample code included by fabric-sample. Finally, we evaluate performance of the proposed method comparing with the sample code.

Keywords: Smart Contract, Chain Code, Hyperledger Fabric, Concealing Data

1. はじめに

近年，Ethereum や Hyperledger Fabric[1] などのスマートコントラクトプラットフォームを利用したアプリケーションが多く開発されている．金融機関や食品の追跡などの分野で利用され，学術的なデータを扱う Open Science Chain[2] などが挙げられる．これらのアプリケーションは，実際にデータが保存される台帳と呼ばれるデータベースと，台帳を操作するコントラクトから成り立っており，コントラクトを実行可能なユーザは，事前に登録されたユーザである．コントラクトは台帳の内容を参照する機能と台帳に新しくデータを追加，保存されているデータの更新などの操作を行う．この際に，データの保存や更新の履歴は全て記録され，データがいつ追加され，いつ更新されたかが保証される．コントラクトと台帳はそれらを管理するピアと呼ばれる端末が複数台で管理する分散システムと

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
[a)] m-saiki@ist.osaka-u.ac.jp
[b)] hkojima@ist.osaka-u.ac.jp

なっており，コントラクトの実行と台帳へのデータ追加，更新はピア間の合意により行われる．

本研究では，スマートコントラクトプラットフォームである，Hyperledger Fabric を対象に行う．Hyperledger Fabric ではコントラクトを Chain code と呼ぶため，以降は Chain code と記す．台帳として Key-Value ストアのデータベースを用いている Hyperledger Fabric では，保存されるデータは基本的に平文である．事前に Chain code を実行できるユーザは登録されているため，Chain code を実行可能なユーザは限られているが，データベースへアクセスできるユーザは，Chain code の実行権限を保有していなくとも，保存されたデータベースの内容を閲覧することが可能である．また，同じデータベースを利用していたとしても，保存しているデータを特定のユーザにのみ利用可能にしたい状況が生じることも考えられる．例えば，製品を製造するにあたり，部品のサプライチェーンを複数の企業で共有している場合，新製品の開発を進めるにあたって，新製品の公表後は同じシステムを利用したいが，それまでは，自社のユーザのみが利用できるデータを保存したい状況などが考えられる．

このように，外部からの利用を防ぐ，内部での利用を防ぐことを容易に行うことができれば，より柔軟にスマートコントラクトプラットフォームを利用することが可能であると考えられる．そこで，本研究では，台帳を操作する Chain code に暗号復号処理を取り入れることで外部からのデータの利用や内部でのデータの利用を制限を可能にする．さらに，提案手法を実装し，含まれるデータが適切に暗号復号処理を行われていることを示す．合わせて，Chain code に暗号復号処理を加えることによるデータの追加，参照にかかる計算時間を計測し，提案手法の性能評価も行う．

2. Hyperledger Fabric

Hyperledger Fabric は，Linux Foundation により提供されている，Hyperledger プロジェクトに含まれる 1 つである．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．Fabric は，スマートコントラクトを実現するプラットフォームの 1 つである．Fabric は事前にユーザを登録することで，許可されたユーザしか Chain code の実行ができないことが特徴の 1 つである．コントラクトは Chain code と呼ばれ，様々な言語で実装可能である．Fabric では，Chain code を実行するには，事前にユーザ登録を行い，そのユーザのみ Chain code を実行することができる．また，Fabric ではいくつかの役割を持つサーバから成り立っており，ユーザ登録等，Chain code の実行権限を管理する CA サーバ，Chain code の実行，台帳管理を行うピア，Chain code の実行順序を決定する Orderer，台帳の実態であるデータベースサーバからなる．ピアは複数の端末からなり，Chain code の実行と実行後のデータベースの操作をピア間で合意を行い合意が取れた場合にのみ Chain code の実行やデータベースの更新を行う．

表 1 暗号化する場所とそのデータを利用可能なユーザ

1:	ユーザが手元で暗号化する 暗号化したユーザのみが利用可能
2:	Chain code に鍵とデータを渡して Chain code で暗号化する 鍵を知っているユーザのみが利用可能
3:	Chain code 自身が鍵を持ち暗号化する Chain code にアクセスできるユーザが利用可能

Hyperledger を用いて利用されてい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 (`https://www.hyperledger.org/`)．

3. 提案手法

本節では，提案手法について説明をする．提案手法は，Chain code の実行権限を持つユーザや外部からデータベースへのアクセス権限を持つユーザに対して，保存するデータを暗号化することで，その利用を制限することが可能となる手法である．1 節に示した様に，Chain code の実行権限はないが，データベースへのアクセス権限を持つユーザから保存するデータの秘匿，保護を行う必要がある場合や，Chain code の実行権限があるユーザに対して，特定のユーザのみにデータの利用をする様にデータの秘匿，保護を行う必要がある場合に対して提案手法は対応することが可能である．保存するデータを暗号化して保存するにあたって，表 1 に示すようにいくつか方法が考えられる．単純に，データを秘匿することを考えると，表 1 の 1:のように，保存するデータをユーザが手元で暗号化することで対応することができる．しかし，これでは暗号化したユーザしか復号ができない，もしくは，復号する手段を他のユーザに伝えることで利用可能となる．1:では，各ユーザが個別に暗号アルゴリズムや鍵を設定すると，復号手段の管理が煩雑になり，運用が難しくなると考えられる．

そこで，提案手法は，この 2:と 3:を実現することにより，Chain code 内に暗号復号処理を含めることにより，データの秘匿や保護を行いつつ，Chain code の実行権限を持つユーザもそのデータを利用することを可能とする．2:は，内部でのデータの利用を制限することが可能となり，3:は外部からのデータの利用を制限することが可能となる．

表 1 に示されている 2:と 3:を実現し，Chain code に暗号復号機能を含めることで，ユーザは意識することなく，データをデータベースに暗号化して保存することが可能である．2:では，公開鍵秘密鍵を用いることで，対応する鍵を持つユーザにのみ，復号処理が行えるようにすることが可能である．そのため，Chain code にアクセスする権限を持っていたとしても，その鍵を持たないユーザは保存されているデータの内容を扱うことが不可能である．3:では，Chain

```
 1:async initLedger(ctx) {
 2:  const cars = [
  :    ...
 5:  for (let i = 0; i < cars.length; i++) {
 6:    cars[i].docType = 'car';
 7:    let c = crypto.createCipher('aes-192-cbc', password);
 8:    let encText = c.update(cars[i].owner, 'utf8', 'hex');
 9:    encText += c.final('hex');
10:    cars[i].owner=encText;
11:    await ctx.stub.putState('CAR' + i,
  :            Buffer.from(JSON.stringify(cars[i])));
  :    ....
14:  }
15:}
```

図 1 `initLedger` での暗号化

code がデータの暗号復号処理を行うため，データベースに保存してあるデータが暗号化されていても，Chain code を通じてデータの追加，参照を行うことが可能である．

4. 実装

提案手法を，Hyperledger Fabric のサンプルコードである Fabcar に実装し，データベースに保存されるデータが適切に暗号化されていることを示す．提案手法は，表 1 に示した 2:と 3:の実現を目標とするが，本論文では，3:の実装を行なった，

Fabcar のソースコードに暗号化の機能を追加したクラス EncFabCar を作成する．暗号化ライブラリは crypto を利用する．EncFabCar は，データベースを初期化するにあたって，Fabcar を踏襲し，メソッド `initLedger()` を実行し，最初にデータベースへ 10 台の自動車データを加える．自動車データは，色 (`color`)，メーカー (`make`)，車種 (`model`)，所有者 (`owner`) などである．この自動車データの所有者を暗号化し，データベースに保存する．

まずは，暗号化するにあたって必要なライブラリを追加し，その際に用いるキーフレーズを定義する．

```
const crypto = require('crypto');
const password='encrypted';
```

次に，図 1 の 5 行目から，データベースへ加えられた 10 台の自動車データに対して，それぞれの所有者のデータを暗号化する．7 行目で暗号化するアルゴリズム AES192 と暗号化に用いるキーフレーズ `password` を与えて，暗号オブジェクト `c` を作成する．8 行目で自動車の所有者を用いて暗号オブジェクトを更新し，9 行目で暗号化された自動車の所有者のデータを取得する．最後に，10 行目で暗号化された自動車の所有者のデータを `cars[i].owner` に代入する．

次に，データを参照する場合の `queryCarDec` メソッド

```
 1:async queryCarDec(ctx, carNumber){
 2:  let carAsBytes = await ctx.stub.getState(carNumber);
  :    ...
 5:  let car = JSON.parse(carAsBytes.toString());
 6:  let d = crypto.createDecipher('aes-192-cbc', password);
 7:  let decText = d.update(car.owner, 'hex', 'utf8');
 8:  decText += d.final('utf8');
 9:  car.owner=decText;
10:  carAsBytes=Buffer.from(JSON.stringify(car));
11:  console.log(carAsBytes.toString());
12:  return carAsBytes.toString();
13:}
```

図 2 自動車データを参照するクエリの復号処理

```
 1:async createCar(ctx,
  :        carNumber, make, model, color, owner) {
 2:  let c = crypto.createCipher('aes-192-cbc', password);
 3:  let encText = c.update(owner, 'utf8', 'hex');
 4:  encText += c.final('hex');
 5:  const car ={
 6:    color,docType: 'car',
 7:    make,model,
 8:    owner:encText,
 9:  };
10:  await ctx.stub.putState(carNumber,
  :        Buffer.from(JSON.stringify(car)));
11:}
```

図 3 自動車データを追加するメソッドでの暗号化

について説明する．このメソッドではデータベースから `carNumber` で示される自動車データを取得し，そこに含まれる暗号化された所有者情報を復号し．結果を返す．まず，図 2 の 6 行目で復号するアルゴリズム AES192 と復号処理に用いるキーフレーズを与えて復号オブジェクトを作成する．次に，7 行目で暗号化された自動車の所有者のデータを用いて復号オブジェクトを更新し，8 行目で復号された自動車の所有者のデータを取得する．最後に，9 行目で `car.owner` に復号された自動車の所有者のデータを代入し，12 行目でその結果を返す．

図 3 の自動車データを追加する `createCar` メソッドでの暗号化の手順は `initLedger` での暗号化の手順と同様である．

5. 評価

実装を行なったのちに，実際にどのようなデータが保存されているかを示す．図 4 は，EncFabCar を動作させ，内容を表示している．1 行目から 4 行目までは，`query.js` を実行し，保存されている情報を取り出して，表示している．3 行目は，`car.owner` を復号せずに表示をしており，4 行目は，図 2 のメソッドを用いて，`car.owner` を復号して表示している．このとき，取り出している自動車データは，`CAR1` の情報である．データベースに保存されているデー

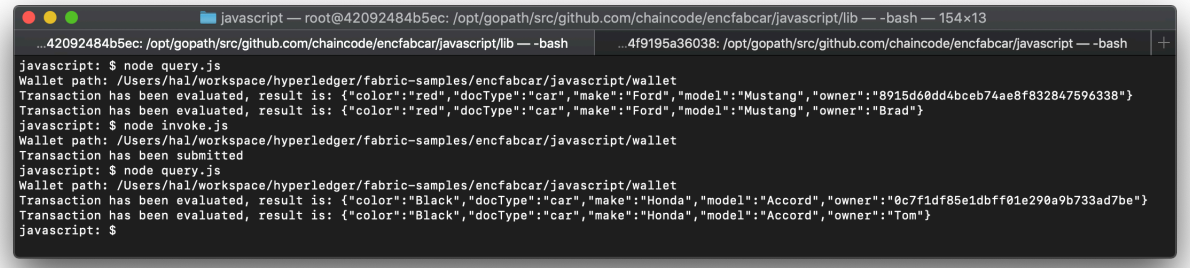

図 4　クエリ実行と自動車データ追加

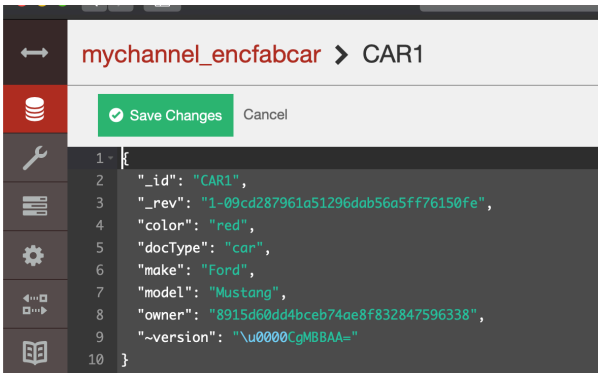

図 5　データベースに保存されているデータ例

タは，図 5 に示されるように，owner の暗号化が行われている．図 5 は，EncFabCar をインストールしたピアに対して，外部からデータベースへアクセスし，その内容を表示する Fauxton を利用している．5 から 7 行目は，新しく，CAR12 を追加している．図 3 のメソッドを用いて，CAR12 を地下しており，8 行目以降は，CAR12 を取り出している．CAR1 と CAR12 のどちらも，owner が正しく暗号化されていることがわかる．

次に，実行時間について評価を行う．表 2 は暗号復号処理がある場合とない場合の実行時間を示している．実行環境は以下の通りである．

OS:	MacOS X Mojave
CPU:	Core i5
MEM:	16GB
Hyperledger Fabric	ver. 1.4
Node.js	ver. 12.8.1

暗号復号処理ありの実行時間は，図 2 と図 3 を 10 回実行した平均である．暗号復号処理なしの実行時間は，fabric-sample に含まれる fabcar を利用している．fabcar に含まれるメソッド queryCar と createCar は，図 2 と図 3 の暗号復号処理がないこと以外は同じ動作をする．これも暗号復号処理ありの場合と同じく 10 回実行した平均である．

表 2 が示すように，2 つの実行時間に大きな差はなく，暗号復号処理を Chain code に含めたとしても，実行時間に大きな影響はないと言える．

表 2　参照と追加にかかる実行時間 [ms]

暗号復号処理	なし	あり
参照	24.26	27.52
追加	2225.48	2221.59

6. まとめ

本論文では，Chain code に暗号復号処理機能を実装し，対象となる Chain code を利用するユーザが扱う情報を暗号化して保存することを可能とする手法を提案した．暗号化したデータをデータベースに保存する方法を 3 種類示し，その 1 つを実装し，実際に保存されるデータが暗号化されており，読み出す際に復号されていることを示した．また，提案手法の性能評価を行い，暗号復号処理を加えたとしても，実行時間に大きな差はないことを示した．

今後の課題としては，表 1 に示してある 2:の手法を実装し，その評価を行う必要があると考えている．本論文での実装の元となった Fabcar では，保存されるデータは，名前や色などのテキスト情報であったが，画像を扱う様にすることで，医療画像等のプライバシーを考慮する必要があるデータを対象にすることが可能となることが考えられる．よって，様々な種類のデータを扱う方法も検討することが必要である．

謝辞
本研究は，JSPS 科研費 18K11262，18K18049 の助成を受けたものです．

参考文献

[1] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A. D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S. W. and Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains, *Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018*, pp. 30:1–30:15 (2018).

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