設計ガイド：

秘密分散ストレージ機能の利用

設計ガイドについて

　ブロックチェーンを利用するアプリケーションは、従来のアプリケーションとは異なる特徴や制約を持ち、それらに対応した設計が必要です。設計ガイドでは、ブロックチェーンを利用するアプリケーションの開発に関する重要なポイントについて、ひとつずつトピックを取り上げ、説明していきます。今回のトピックは、秘密分散ストレージ機能を利用するアプリケーションについてです。

　ベストプラクティスや事例を交えながら、ブロックチェーンの基本原理や概念に沿って効果的にブロックチェーンを活用するための指針を示します。ブロックチェーンに関する一般的な内容は、ブロックチェーンの共通の特性や仕組みに基づいて説明します。東芝独自のブロックチェーン DNCWARE Blockchain+™ (BC+)に関する固有の視点は、このプラットフォームで利用可能な機能に触れながら紹介します。

はじめに

　この設計ガイドでは、BC+固有の機能である、秘密分散ストレージ機能を利用したアプリケーションについて、以下のような観点で説明します。

- 秘密分散ストレージ機能の概要
- 秘密分散について
- ストレージ領域の管理
- アクセス制御について
- ビザンチン障害耐性
- データの整合性
- 利用方法とヒント

秘密分散ストレージ機能の概要

ブロックチェーンは大容量データの保存には向いていません。そのため、大容量のデータが保存できるIPFSなどの外部ストレージと併用して外部にデータを保存することが一般的に行われています。具体的には、大容量データをIPFSに格納し、そのURIをブロックチェーンに記録します。すると、ブロックチェーンに記録されたURIを辿ってIPFSからデータを読み出し、読み出したデータのハッシュ値とURIのコンテンツアドレスとの整合性を確認することで、データの完全性が検証できます。この方法により、ブロックチェーン・アプリケーションで有効に大容量データを取り扱うことができます。

ところが、IPFSに保存されたデータはすべて全世界に公開され、一度公開されたデータは削除することが難しいという特性があり、エンタープライズ用途には向かないケースも多くあります。そのため、企業向けのブロックチェーン・アプリケーションには、機密性を確保できる別のストレージ・ソリューションが求められています。

そこでBC+では、秘密分散ストレージ機能と呼ぶストレージ・ソリューションを2025年から提供します。このソリューションでは、企業向けの高いレベルの情報保護を実現するために、秘密分散によるセキュリティとスマートコントラクトによるアクセス制御機能を組み合わせています。BC+のシステムと一体となって機能を実現しているためブロックチェーンとの親和性が高く、BC+を利用したブロックチェーン・アプリケーションへの導入が容易です。

秘密分散ストレージ機能の特長

・大容量データの保存、読み出しが可能
・秘密分散による機密性および高可用性
・ブロックチェーン・ウォレットによるアクセス認証
・ストレージ領域の管理はスマートコントラクト経由
・ビザンチン障害耐性がある
・データの上書きと削除ができる

秘密分散について

秘密分散とは、情報を複数の断片（シェア）に分割し、そのうちの所定の数のシェアを集めたときに限り、元の情報を復元できる技術の総称です。秘密分散のひとつに、1979 年にアディ・シャミア（Adi Shamir）によって発表された Shamir's Secret Sharing(以下 SSS と呼びます)があります。BC+の秘密分散ストレージ機能ではこの方式を採用しています。本ガイドでは、秘密分散という用語は特にこの方式を指すものとします。

SSS は(T,N)しきい値法に分類されます。ここで N は分散するシェアの数、T は元の情報を復元するために必要十分なシェアの数を意味します。T は 1 以上 N 以下(1≦T≦N)で、しきい値と呼ばれます。T は秘密分散を行うときに任意に設定できますが、T=1 の設定は秘密分散として実質的な意味がないので、通常は 2≦T≦N の範囲で設定します。本ガイドでは、特に断りがない限り 2≦T≦N であるものとします。

SSS では、ひとつのシェアのデータサイズは元のデータと同じサイズとなります。シェアは全体で N 個あるので、総データサイズは元のデータサイズの N 倍になります。この冗長性は、元データに乱数係数の変数項を加えた T-1 次の多項式を作り N 個の異なる点での値を求めることにより生成します。T-1 次の多項式の値が T 個の点について分かればその多項式は一意に決まる性質により元データが復元できます(ラグランジュの補間公式)。より詳細な説明は wikipedia(*)にありますので、興味がある方はそちらを参照してください。なお、BC+の秘密分散ストレージ機能では、有限体 GF(256)上で SSS のアルゴリズムを実行することで、バイト単位での秘密分散としています。

(*)https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing

構成と動作の概略

図 1 に秘密分散を使ったストレージシステムの動作の概略を示します。書き込みクライアント A と読み出しクライアント B は同一でも構いませんが、この図では別のコンピュータとしています。書き込みの流れとして、クライアント A 内で元データを秘密分散のアルゴリズムを使って N 個のシェアに分割し（図では N=4)、それぞれのシェアを N 台のサーバに書き込みます。読み込みの流れとして、書き込まれたシェアのうち T 個をクライアント B 内に読み出せば（図では T=2)、秘密分散のアルゴリズムを使って元データを復元できます。一方、T 個より少ないシェアだけでは元データを復元できません。例えばサーバ 1 の管理者がシェア 1 を不正に読みだしても元データを復元できません。

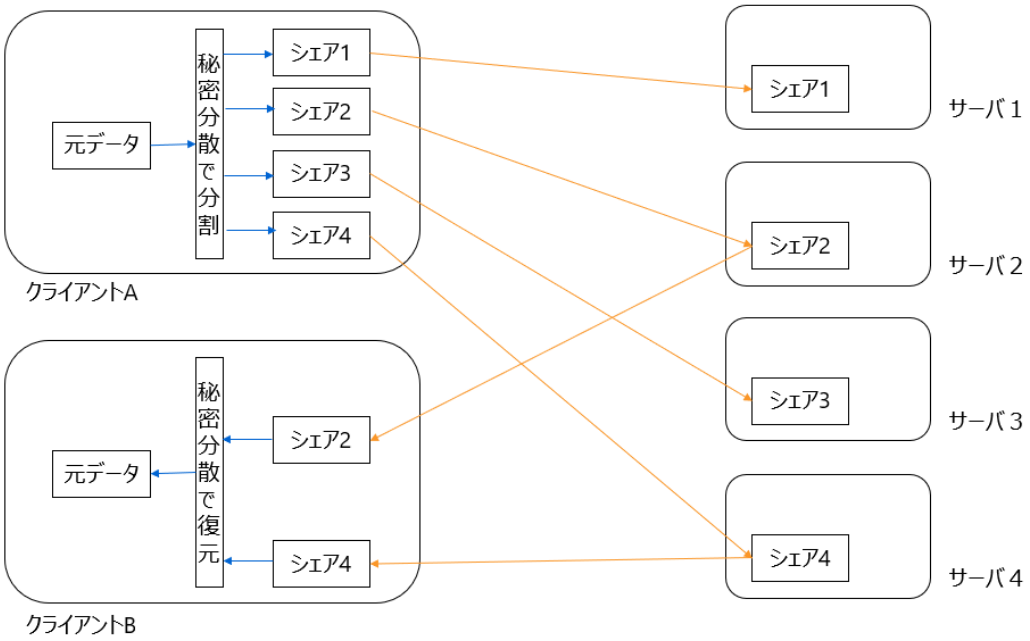

図 1 秘密分散を使ったストレージシステム

分散することによる機密性と高可用性

シェアのそれぞれを異なる独立したサーバに保存することで、1 台のサーバが不正アクセスを許したとしても、元の情報の機密性は守られます。一般には、最大で T-1 台のサーバが不正アクセスを許したとしても、元の情報の機密性は守られます。これは、サーバの管理者に悪意がある場合でも同様の耐性となります。つまり、最大で T-1 台のサーバの管理者に悪意があり結託したとしても、元の情報の機密性は守られます。

また、T<N のときは、１台のサーバが停止していたとしても、元の情報を復元することが出来ます。一般には、最大で N-T 台のサーバが停止していたとしても、残りの T 台のサーバからシェアを読み出せば、元の情報が復元できます。このような性質は高可用性と呼ばれます。つまり、秘密分散は高可用性の性質も持ち合わせています。

秘密分散では、T の設定を大きくとれば、機密性が高まりますが、可用性が低くなります。T の設定を小さくとれば、機密性は低くなりますが、可用性が高くなります。したがって、アプリケーションの目的にあわせて T の値を適切に設計することが求められます。

ブロックチェーンへの格納との比較

ブロックチェーンでは、すべてのノードに同一の内容が記録されます。どのノードを見ても同じ内容が記録されているので、最大の可用性を実現します。また、このデータ構造は改ざんに対しても耐性があります。実は、このブロックチェーンの分散構造は秘密分散での T=1 の設定に相当します。つまり、秘密分散はブロックチェーンの自然な拡張に位置づけることができ、ブロックチェーン・システムとの親和性があります。

BC+の秘密分散ストレージ機能では、秘密分散のシェアをブロックチェーンのノードと同じサーバに格納します。ブロックチェーンのサーバは互いに独立して管理されるため、秘密分散で利用するのに適しています。ただし、シェアの内容はサーバごとに異なるため、シェアがブロックチェーンのハッシュチェーンに繋がれることはありません。そのため、シェアがブロックチェーンに記録されているわけではありません。「ブロックチェーンのノードと同じサーバに格納」という回りくどい言い方をしたのはそのためです。なお、同じ理由でスマートコントラクトからシェアの内容を参照することはできません。

上述したように、ブロックチェーンの分散構造は T=1 の設定の秘密分散と同等なので、プライベート・ブロックチェーンであっても、１台のノードが不正アクセスを許しただけでブロックチェーンに格納されたデータの機密性は破られます。一方、2≦T の設定の秘密分散によって格納されたデータは、それよりも高い機密性を持っています。したがって、機密性が特に重要な場合には、データサイズの大小にかかわらず秘密分散ストレージ機能を利用することが適切です。

ストレージ領域の管理

BC+の秘密分散ストレージ機能では、シェアを格納するためのサーバ側の領域をストレージ領域と呼びます。ストレージ領域はシェアを書き込む前にあらかじめブロックチェーン上のスマートコントラクトを実行して作成しておく必要があります。また、不要になったストレージ領域の削除もこのスマートコントラクトから行います。スマートコントラクトはすべてのブロックチェーン・サーバ上で多重実行されるので、ストレージ領域の管理は自然にすべてのブロックチェーン・サーバ上で連動して行われます。なお、ストレージ領域を管理するスマートコントラクトをストレージコントラクトと呼びます。

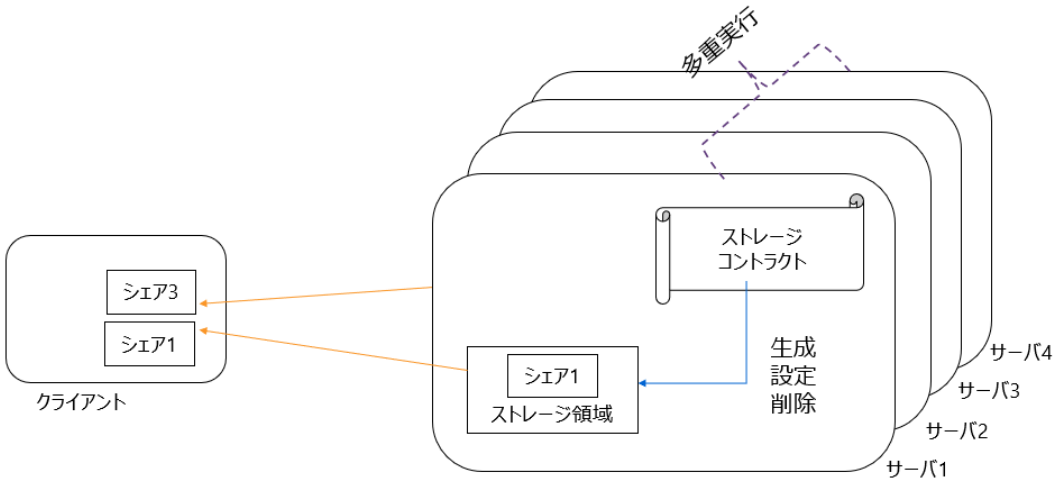

図 2 ストレージコントラクトから行うストレージ領域の管理

ストレージ領域は複数作成することができ、その領域は互いに独立しています。ストレージ領域の作成時には、領域の名前やサイズの他にアクセス権や有効期限も設定します。設定されたアクセス権をクライアントが持っている場合に限り、そのストレージ領域へのアクセスが許可されます（アクセス制御の詳細については後述します)。これらの設定は、アプリケーションの目的に応じて適切に行うことが重要です。

ストレージ領域の識別

ストレージコントラクトは複数作成できます。ストレージコントラクト下で管理されるストレージ領域は名前（key 文字列）によって区別されます。異なるストレージコントラクトの下で管理されるストレージ領域は互いに独立しています。つまり、ストレージ領域はストレージコントラクトの ID と key 文字列の組によって、システム内で一意に識別されます。

key 文字列は再利用することはできません。同じストレージコントラクト下の同じ key 文字列に対して 2 つめのストレージ領域を作成しようとするとエラーとなります。いったん削除した後や有効期限が切れた後に再作成することもできません。ただし、別のストレージコントラクトの同じ key 文字列は、互いに独立しているため同時に利用可能です。

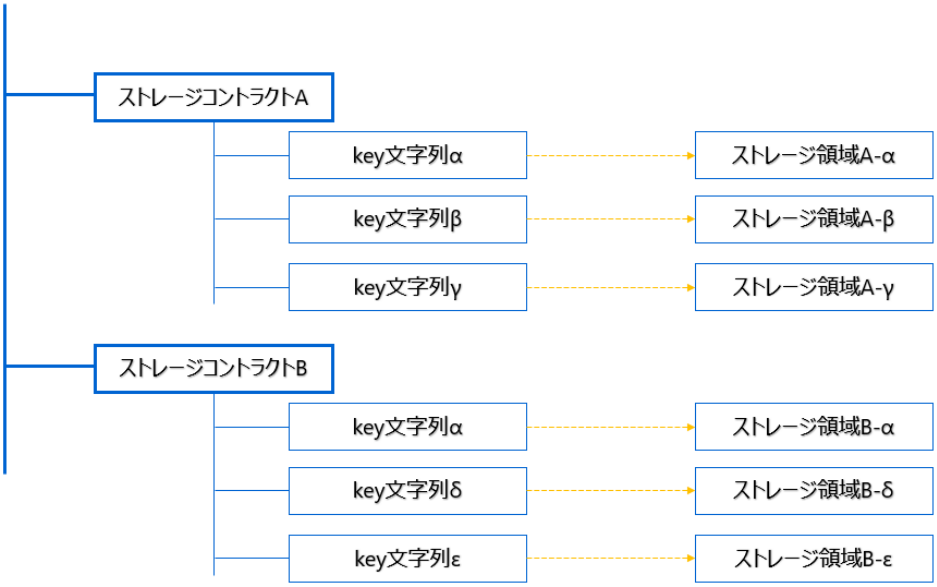

図 3 ストレージ領域の名前空間

有効期限

有効期限が過ぎたストレージ領域は自動的に無効化されデータの読み書きはできなくなります。有効期限はストレージ領域の作成時に指定するようになっており、作成後に変更することはできません。また有効期限を延長する方法はありません。有効期限が切れた後に同じ key 文字列を使ってストレージ領域を作成することもできません。なお、有効期限が切れた後のストレージ領域の削除は成功します。

アクセス制御について

ストレージ領域からの読み出しにはアクセス制限が設けられています。アクセス制限がない状況では、誰でもシェアを読み出して元データを復元できてしまい、秘密分散の目的が果たせなくなるためです。BC+の秘密分散ストレージ機能は、このアクセス制御をブロックチェーンのアクセス制御機能を活用して実現しています。

具体的には、アクセスの認証にはブロックチェーン・ウォレットを使用します。ウォレットを使ってデジタル署名を付けたアクセス要求をクライアントからサーバに送ることにより認証を要求します。要求を受けたサーバ側では、ウォレットアドレスをもとにアクセスを要求しているユーザを特定します。そして、ストレージ領域の作成時にあらかじめ設定されているアクセス制御リストを参照し、アクセス要求を行ったユーザがそこに含まれている場合に限り、そのストレージ領域へのアクセスが許可されます。つまり、アクセス権を持つウォレットがあるときに限り、ストレージ領域へのアクセスができます。

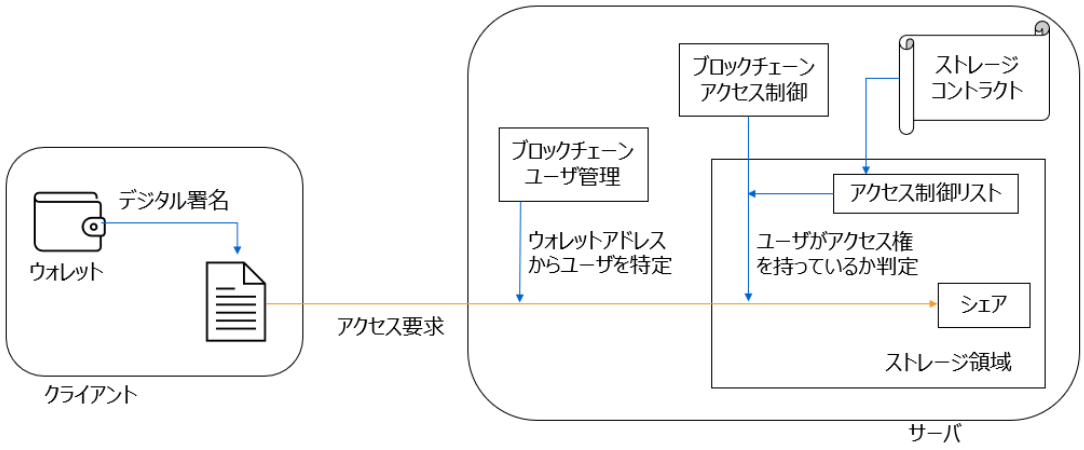

図 4 ストレージ領域のアクセス制御

アクセス制御リストに含まれるユーザにはアクセス権が与えられます。アクセス制御リストにはドメインやグループも指定することができ、ドメインやグループに含まれる全てのユーザに対して一括してアクセス権を付与することができます。後からドメインやグループのメンバーが変更された場合、アクセス権の付与は自動的に追随します。アクセス制御リストは、読み出しと書き込みそれぞれに対して個別に設定が可能です。

例：企業間でファイルを安全に渡す

A 社から B 社へ機密性のあるファイルを渡す場合を考えてみましょう。A 社は、機密情報を安全に共有するために、秘密分散ストレージ機能を利用します。まず、A 社は書き込みのアクセス制御リストに A 社のユーザ ID を指定し、読み出しのアクセス制御リストに B 社のユーザ ID を指定して、ストレージ領域を作成します。次に、A 社は渡すファイルをストレージ領域に秘密分散して書き込みます。これにより、指定された B 社のみがストレージ領域から読み出して元のファイルを復元できます。一方、指定されなかった C 社がサーバを 1 台管理していてもこの管理権限だけでは秘密分散された元のファイルを復元することはできません。

例：チーム内に限定して情報共有

あるプロジェクトチームが機密情報をチーム内で共有したい場合、プロジェクトメンバーだけがアクセスできるストレージ領域を作成し、そこに秘密分散して機密情報を格納し、プロジェクトメンバーだけが参照できるようにします。具体的には、プロジェクトメンバー全員が所属するグループをブロックチェーン上に作成し、そのグループをストレージ領域の読み出しアクセス制御リストに設定します。プロジェクトの進行にともない新しいプロジェクトメンバーが追加される場合、そのグループにプロジェクトメンバーのユーザ ID を追加すると、このプロジェクトメンバーも同じストレージ領域にアクセスできます。

ビザンチン障害耐性

ビザンチン障害耐性は、一部のノードに誤動作あるいはノード管理者の不正行為があったとしても、正常なノードには影響が及ばず、全体として正しい動作を継続できる性質です。ブロックチェーンは一定のビザンチン障害耐性を持ちます。一方で、秘密分散は基本的にビザンチン障害耐性をもちません。これは、T 個のシェアを集めて元データを復元するときに、そのうちの 1 つのシェアが壊れているだけで復元した元データも壊れてしまうからです。

そこで、BC+の秘密分散ストレージ機能では、基本となる SSS のアルゴリズムはそのままとして、シェアが壊れているかを判別できるように冗長なデータ構造を外付けし、ビザンチン障害耐性を持たせています。その結果、T-1 台までのビザンチン障害に対して耐性を持つようになっています。つまり、最大で T-1 台のノードに不正がある場合でも、クライアント側で秘密分散の読み出しが正常完了した場合、その読み出されたデータは壊れていないことが保証されます。

ブロックチェーン自身が持つビザンチン障害耐性と合わせ、管理を含めた全体としてビザンチン障害耐性を持つストレージシステムは、外部からの攻撃や内部の不正からデータを守ることができ、全体としての信頼性が上がり、データの安全性を一層高めることができます。

なお、ビザンチン障害耐性を持たせるために冗長なデータ構造を追加したので、そのぶんシェアのデータサイズは少し増えています。その増加分を考慮してストレージ領域の割り当てサイズを計算する必要があるところが注意点になります。（具体的な計算方法は後述します）

データの整合性

ブロックチェーンに記録されたデータは上書きや削除ができません。しかし、秘密分散ストレージ機能ではそれが可能です。具体的には、上書きは重ねて書き込むことで、削除はストレージコントラクトからストレージ領域を削除することで行えます。

　このように、秘密分散ストレージ機能はデータの変更ができる仕様のため、最初に書き込まれていたデータが正しいものであったとしても、あとから不正なデータを上書きすることによりに改ざんされる可能性を考慮する必要があります。そこで、正しいデータのハッシュ値をブロックチェーンに別途記録しておき、秘密分散ストレージからデータを読み出した後にブロックチェーンに記録されているハッシュ値と整合しているかをクライアント側で確認するなどして、改ざんされていないデータであるかを検証することが重要になります。

同時アクセスにおける整合性ついて

・同時に複数の書き込みが行われた場合には、データが混ざる可能性があり、データの整合性は保証されません。
・同時に書き込みと読み出しが行われた場合には、新旧データが混ざって読み出される可能性があり、データの整合性は保証されません。
・同様に、書き込みが中断した後に読み出しが行われた場合には、新旧データが混ざって読み出される可能性があり、データの整合性は保証されません。

バージョン番号による整合性の強化

　同時アクセスにおける整合性を強化するために、書き込み時にバージョン番号を指定するオプションがあります。すべてのクライアントが必ずバージョン番号を指定して書き込んでいる場合には、整合性に関して下記が成立します。
・同時に複数の書き込みが行われた場合に、書き込みが正常完了したデータについては整合性が保証されます。
・同時に書き込みと読み出しが行われた場合に、読み出しが正常完了したデータについては整合性が保証されます。
・同様に、書き込みが中断した後に読み出しが行われた場合に、読み出しが正常完了したデータについては整合性が保証されます。

バージョン番号は上記の整合性を保証するために指定するオプションであり、過去のバージョンを記憶する変更履歴機能を実現するものではありません。バージョン番号の管理はアプリケーション側で行う必要があり、上書き時には、すでに書き込まれているバージョン番号より大きい値を指定する必要があります。そうでない場合には、上書きはエラーとなります。バージョン番号の管理はアプリケーション側で任意の方法で行うことができますが、一例として、書き込むデータのハッシュ値を記録するトランザクションのトランザクション番号をバージョン番号とすることができます。

ストレージ領域への書き込みが１回しかないアプリケーションの場合では、形式的にバージョン番号＝１として書き込むことが推奨です。この場合、読み出し時にもバージョン番号＝１を指定して読み出すことにより、上書きされていないことが簡単に確認できます。

利用方法とヒント

ストレージ領域の作成

ストレージ領域は、ストレージコントラクトに引数 cmd='create'を指定して実行することで作成されます。ストレージ領域は、チャンクと呼ばれる一定のサイズの領域に分割されています。チャンクには 0 番から通し番号がついています。0 番のチャンクは、秘密分散のメタデータ領域として用いられ、分散するノードのピア ID のリストや、しきい値 T の値、ファイルサイズ、バージョン番号などが格納されます。1 番からのチャンクには、秘密分散のシェアが書き込まれます。なお、ストレージ領域の作成時のパラメータには下記があります。

key: ストレージ領域を識別するための文字列
expiry: 有効期限(UNIX タイム)
chunksize: チャンクのサイズ
chunks: チャンクの数
readable: ストレージ領域からの読み出しアクセス制御リスト
writable: ストレージ領域への書き込みアクセス制御リスト
peers: ストレージ領域を確保するピアのリスト(省略時はすべてのピアに確保される)
info: アプリケーションで自由に設定できる情報

ストレージ領域に必要な割り当てサイズ

ストレージ領域作成時に指定する chunksize は、システムで許容される最大の値に設定するともっとも効率が良いです。具体的には chunksize=40000 とします（システム設定がデフォルトの場合)。また、ストレージ領域作成時に指定する chunks は下記式で計算できます。

---

chunks = 1 + Math.ceil(fsize / (chunksize - 32(N+1)))*

ここで、fsize は元ファイルのサイズです。

ストレージ領域の削除

ストレージコントラクトに引数 cmd='delete'と key 文字列を指定して実行することで、ストレージ領域を削除できます。

秘密分散ライブラリをセットアップする

秘密分散ライブラリを使うためには、はじめに分散するブロックチェーン・ノードのリストを指定する必要があります。たとえば、

```
var pid_sockets = [
    [ 'p11111111', 'https://peer1.dncware-blockchain.com' ],
    [ 'p22222222', 'https://peer2.dncware-blockchain.com' ],
    [ 'p33333333', 'https://peer3.dncware-blockchain.com' ],
];
```

というように、ノードの ID とエンドポイントの URL を組にして並べます。そして、

```
var sss = new api.StorageSSS(chainID, N, T, pid_sockets);
```

として、秘密分散ライブラリをセットアップします。

データを秘密分散して書き込む

ファイルの書き込みは以下のように 1 行でできます。

```
await sss.writeFile(wallet, cid, { key, ver }, wdata);
```

ここで、wallet はブロックチェーン・ウォレット、cid はストレージコントラクトの ID、key はストレージ領域を識別する key 文字列、ver はバージョン番号、wdata は書き込みデータです。

秘密分散したデータを復元する

ファイルの読み込みは以下のように 1 行でできます。

```
var rdata = await sss.readFile(wallet, cid, { key });
```

ここで、wallet はブロックチェーン・ウォレット、cid はストレージコントラクトの ID、key はストレージ領域を識別する key 文字列、rdata は復元されたデータです。正常な場合、rdata は、書き込み時のデータ wdata と一致します。

書き込みと読み出しのライブラリ設定の関係

一般には書き込みと読み出しのクライアントは別々のコンピュータなので、秘密分散ライブラリのセットアップのパラメータが不一致となる可能性があります。その場合の条件は下記のとおりです。

・しきい値 T の値は一致していなければなりません。

・pid_sockets の pid の一覧は集合として一致していなければなりません。順序は異なっていても構いません。

アクセス性能を上げる為のヒント

アクセス先のブロックチェーン・ノードが遠方にある場合、アクセス速度が遅くなります。これはネットワークの遅延が大きいことが原因の一つです。そのような場合には、ライブラリの並列度を調整することにより、アクセス性能が向上する場合があります。具体的には、以下のように並列度を変更します。

```
sss.setConcurrency(p)
```

p の初期状態は 4 なので、たとえば倍にして p=8 としてまずは様子を見ます。p を変えながら、p のちょうどよい値を探します。p の値をあまり大きくとりすぎるとかえって性能が劣化します。経験的には、ブロックチェーン・ノードとクライアントがすべて日本国内にあれば p=4 の初期設定で問題ないでしょう。

余談ですが、p=0 に設定すると一時停止となり、p>0 で再設定するまで動作が保留されます。

まとめ

秘密分散ストレージ機能の設計ガイドとして、秘密分散、ストレージ領域の管理とアクセス制御、障害耐性とデータの整合性を説明し、具体的な利用方法を記述しました。