情報処理学会論文誌　コンピューティングシステム　Vol.5　No.3　57–73 (May 2012)

# 分散ファイルシステムに適した分散アクセス制御機構

荒川 淳平[1,†1,a)]　笹田 耕一[1,†2]

受付日 2011年10月7日, 採録日 2012年1月22日

**概要**：多数のユーザと大量のリソースをかかえる分散ファイルシステムのアクセス制御機構には高い可用性とスケーラビリティが求められる．しかし，従来の集中管理型のアクセス制御機構ではこの要求に応えられない．また，権限証明書を用いる分散管理型のアクセス制御機構の既存研究では，認証方式が公開鍵認証に限定され，権限の変更に副作用があり，自律的に認証情報が更新できないといった問題点が存在する．本稿ではこれらの問題点を解決する分散管理型のアクセス制御機構を提案する．提案機構では，パスワード認証を可能にする公開認証情報を導入し，権限変更や認証情報の更新の問題を解決するために，証明書の更新と証明書の無効化を区別し，ユーザの代わりに機構側が証明書に署名する．提案機構は保持する情報を Consistent Hashing を用いて分散管理する．我々は，提案機構の可用性とスケーラビリティをシミュレーションやプロトタイプ実装によって評価した．

**キーワード**：アクセス制御機構，分散ファイルシステム，権限証明書，セキュリティ

# A Decentralized Access Control Mechanism for Distributed File Systems

Jumpei Arakawa[1,†1,a)]　Koichi Sasada[1,†2]

Received: October 7, 2011, Accepted: January 22, 2012

***Abstract:*** Access control mechanisms of distributed file systems hold thousands of resources with thousands of users require high availability and high scalability. However, existing centralized access control mechanisms are hard to satisfy these requirements. Moreover, existing decentralized access control mechanisms based on authorization certificate have several practical issues; they only support public key authentication; changing access authority causes troublesome side effects; users cannot update authentication information autonomously. To solve these issues, we propose decentralized access control mechanism based on authorization certificate which uses open authentication information instead of public key, AC node private key instead of user's private key for the signature of authorization certificates. Our proposed mechanism stores its data using Consistent Hashing algorithm. We also implemented simulation program and prototype version of our proposed mechanism and proved its availability and scalability.

***Keywords:*** access control mechanism, distributed file system, authorization certificate, security

## 1. はじめに

近年，クラウドコンピューティングの台頭にともない，分散ファイルシステムが再び注目を集めている．これは，サービスとしてコンピューティングリソースを提供するクラウドコンピューティングにとって，分散ファイルシステムがスケーラビリティと高い可用性を提供できる基盤だからである．ユーザ数や利用リソース量の増加に対して，システムを構成するノード数を増やすことで性能や容量の増強が可能なこと，つまりスケールアウトすることは重要な性質である．そして，可用性はそのままサービスの品質に直結する．高い可用性を実現するためには，単一障害点を

[1] 東京大学大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[†1] 現在，株式会社インフォクラフト
Presently with Infocraft, Inc.
[†2] 現在，Heroku, Inc.
Presently with Heroku, Inc.
[a)] jumpei.arakawa@08.alumni.u-tokyo.ac.jp

なくすことが重要なポイントとなる．

したがって，この 2 つのポイント，

- 性能がスケールアウトすること
- 単一障害点を持たないこと

は分散ファイルシステムを評価するうえで最も重要なポイントであると考える．事実，上記の 2 点を満たす分散ファイルシステムが数多く提案・開発されてきている [11], [12], [13]．

一方，アクセス制御機構とはユーザのリソース（ファイルやディレクトリ）に対するアクセス（読み書きや削除といった操作）を制御する機構である．多数ユーザが大量のリソースを利用することを考えた場合，分散ファイルシステム単体ではなく，アクセス制御機構も含めたシステム全体でスケーラビリティや可用性を考えることが必要である．なぜならば，複数のシステムが連携してサービスを提供している場合，1 つのシステムでもスケーラビリティが悪ければ，そこがボトルネックとなりサービス全体の性能を落とすことがありうるし，1 カ所でも単一障害点を持つシステムがあれば，それはサービス全体が単一障害点をかかえていることに等しい．

以上からアクセス制御機構も分散ファイルシステムと同様に，性能がスケールアウトし，単一障害点を持たないことが重要なポイントといえる．

そこで我々は，クラウド時代の分散ファイルシステムに適したアクセス制御機構を提案する．我々の提案するアクセス制御機構（以下，提案機構）は，旧来から多く利用されている集中管理型ではなく，権限証明書を用いた分散管理型のアクセス制御機構である．

Unix に伝統的に実装されている ACL 方式に代表される集中管理型では，可用性とスケーラビリティの 2 点に加えて，ユーザを管理するためのコストの問題があった．この問題を解決する目的で多くの分散管理型のアクセス制御機構が提案されてきた [4], [5], [6], [7], [8], [9]．既存の分散管理型のアクセス制御機構は，スケーラビリティや可用性に優れた性質を示す一方，認証方式が公開鍵認証に限定されており，権限の変更に副作用がともなう等の実用上の問題があった．

提案機構ではこれらの問題を公開認証情報の導入（3.2.2 項）や機構側の秘密鍵による署名（3.2.3 項），CRL（Certificate Revocation List）に加えて CUL（Certificate Update List）の導入（3.2.4 項）によって解決した．また，CRL や CUL が単一障害点や性能上のボトルネックになることを避けるために，CRL および CUL を分散化した（3.3 節）．

本稿では，提案機構の設計を示し，シミュレーションやプロトタイプ実装を用いた評価を示す．

## 2. 問題点

本章では，提案機構で解決する問題点について述べる．ここでは大まかな分類としてとらえた場合の共通する問題点の記述に集中し，個々の具体的な先行研究および既存システムとの比較は 5 章で述べる．

本稿では，スケーラビリティと可用性を評価する目的で，アクセス制御機構をアクセス制御情報（権限やリソースの範囲等）の保持方法によって，集中管理型と分散管理型に以下の定義で分けて議論を進める．

**集中管理型**　機構側でアクセス制御情報をすべて保持する必要がある．

**分散管理型**　機構側でアクセス制御情報の一部しか保持する必要がない．

### 2.1 集中管理型

実用目的（研究目的でない）で利用されているアクセス制御機構のほとんどが集中管理型に分類できる．Linux や Windows が標準的に備えているアクセス制御機構も集中管理型である．集中管理型のアクセス制御機構はすべての情報を機構側がコントロールできるため，自由度が高く実装も容易である．

しかし，すべての情報を機構側で管理するため，ユーザ数やリソース数の増加に比例して管理する情報量が単純に増加する．加えて，単一のノード上での動作を前提に実装されることが多く，その場合アクセス制御機構が単一障害点となってしまう．またその場合は，アクセスの集中も招くこととなり性能上のボトルネックにもなりうる．

単一障害点とボトルネックの問題は，管理情報の複製を複数のノードに持たせることである程度は回避可能である．しかし，上で述べたようにアクセス制御情報をすべて複製・保持する必要があるため，複製するノードの台数を増やせば増やすほど，同期処理のコストや記録コストが増大するため，スケーラビリティに欠ける．

加えて，既存の集中管理型のアクセス制御機構では，ユーザの登録やアクセス権限の変更のために，特別なユーザ（管理者やリソースの所有者）の関与が必要である．このことは，登録・設定コストが集中することを意味しており，実用上の問題となる．

### 2.2 分散管理型

上で述べた集中管理型の問題点を解決する目的で提案されているのが分散管理型のアクセス制御機構である．分散管理型のアクセス制御機構では，アクセス制御情報の大部分を機構側で保持する必要がない．これはアクセス制御時にユーザ側が必要な情報を機構側に提示するからである．権限証明書と呼ぶ，デジタル署名を用いることで検証可能にしたアクセス制御情報[*1]を用いることで，アクセス制御の基本的な処理はユーザ側から提示される権限証明書のみ

[*1] Credential とも呼ばれるが本稿では権限証明書と呼ぶ．

を用いて実施可能となっている．

このため，容易に複数のノードに処理を分散させることができ，スケーラビリティの面で集中管理型に比べて優れているといえる．ただし，機構側で保持する必要のある情報（CRL やルートの秘密鍵）については，分散戦略を別に設計する必要がある．これらの機構側で保持する必要のある情報を冗長化することで，アクセス制御機構から単一障害点を排除することが可能である．

加えて，権限証明書を用いる分散管理型のアクセス制御機構 [4], [5], [6] では，事前にユーザを登録する必要がない．これは権限証明書の委譲（発行）にユーザ登録が実質的に含まれているためである．これにより集中管理型で述べたような登録・設定コストが分散するようになっている．

このように，分散管理型のアクセス制御機構は集中管理型に比べて，スケーラビリティや可用性の面，加えて登録・設定コストの面でも優れている．ところが，以下に述べる実用上の問題があり，このことが分散管理型が広く普及するに至っていない理由であると考える．

1 つ目の問題点は，認証方式に公開鍵認証しか用いれないことである．既存研究 [4], [5], [6] では証明書の署名・検証に用いる公開鍵暗号と認証方式とが深く結びついており，パスワード認証等の一般的な認証方式がサポートされていない．特にパスワード認証は，安全性の面で公開鍵認証には見劣りするものの，多くのユーザにとって理解・利用しやすい認証方法だといえる．事実，主要なすべての OS やほとんどのウェブサービスでユーザ認証にはパスワード認証が利用可能である．このことから，パスワード認証が利用できないことは，分散管理型のアクセス制御機構の普及を妨げている大きな要因の 1 つであると考える．

2 つ目の問題点は，認証情報の更新を自律的に行えないことである．既存研究 [4], [5], [6] では，証明書の署名に委譲を許可したユーザ（許諾者）の秘密鍵を用いるため，証明書の所有者が自由に認証情報（公開鍵認証であれば秘密鍵，パスワード認証であればパスワード）を変更することができない．このため，所有者は新しい認証情報に関する情報を許諾者に送り，許諾者の秘密鍵を用いて新たに証明書を再発行（再委譲）してもらう必要がある．また，古い認証情報を含む証明書は無効化する必要があるため，次に述べる 3 番目と同様の問題点も内包している．

3 つ目の問題点は，権限変更の副作用が大きいことである．ファイルシステムでの利用を考えた場合，1 度設定した権限を後から変更するような利用ケースが考えられる．この場合，既存研究 [4], [5], [6] では，証明書の更新は直接実現できないため，間接的に古い証明書を無効化し新たに新しい証明書を発行する必要がある．しかし，ある証明書を無効化すると，その証明書をもとに委譲されたすべての証明書が無効となってしまう．このことは，権限変更（や上で述べた認証情報の更新）のたびに，配下の関連するすべての設定を，設定に関わったすべてのユーザに，（委譲関係を維持するために）同じ順列で，再度設定（証明書の再発行）してもらう必要があることを意味する．これは受け入れがたい運用上のコストである．

## 3. 提案機構

本章では提案機構の設計について述べる．提案機構は，分散管理型のアプローチをとりつつ，前章で述べた問題点を解決する．具体的には，以下の点を実現する．

(1) 性能がスケールアウトする．
(2) 単一障害点を持たない．
(3) 登録・設定コストが分散する．
(4) パスワード認証が利用できる．
(5) 自律的に認証情報が更新できる．
(6) 副作用なく権限変更ができる．

本章ではまず 3.2 節で提案機構の構成と基本となる権限証明書を用いたアクセス制御について述べる．ついで，3.2.1 項で権限委譲によって (3) が解決されていることを確認し，そのうえで提案機構で (4) から (6) を実現する方法について 3.2.2 項から 3.2.4 項で述べる．そして 3.3 節では，提案機構を分散・冗長化する方法について述べ，(1) と (2) が解決できることを示す．

### 3.1 提案機構の構成

個々の機能の実現方法を示す前に，提案機構の全体的な構成について述べる．提案機構は**図 1** に示すような環境を想定している．

提案機構は分散ファイルシステムをターゲットとしたアクセス制御機能を提供する分散管理型のアクセス制御機構である．提案機構は対等な複数の AC ノード（Access Control ノード）で構成され，それぞれの AC ノードがアクセス制御に必要なすべての機能を等しくユーザ[*2]に対して提供する．図からも分かるように，分散ファイルシステムの構成ノードと AC ノードが同一の計算機であっても問

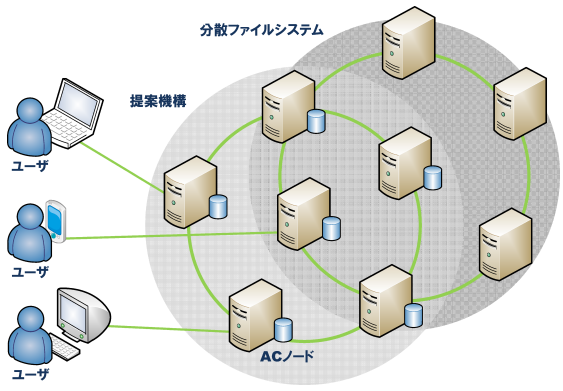


**図 1**　提案機構と分散ファイルシステムの構成（想定環境）
**Fig. 1**　Our ACM and distributed file system.

[*2] ユーザには人間の利用者以外にも機器やプログラムが含まれる．

表 1　AC ノードが保持する情報
**Table 1**　Information held by AC nodes.

| 項目 | 内容 |
|---|---|
| ノード ID | 提案機構全体でユニークな AC ノードの ID |
| ノード経路表 | 全 AC ノードの ID と接続情報のマップ |
| ノード秘密鍵 | AC ノードごとに異なる鍵ペアの秘密鍵（鍵ペアは定期的に更新される） |
| ノード公開鍵 | AC ノードごとに異なる鍵ペアの公開鍵（鍵ペアは定期的に更新される） |
| ノード公開鍵リスト | 全 AC ノードの過去から現在までの公開鍵のリスト |
| CRL | 無効化された証明書のリストの一部（詳細は 3.3 節） |
| CUL | 更新された証明書のリストの一部（詳細は 3.3 節） |

題はない．

各 AC ノードは**表 1** に示す情報を保持する．提案機構は分散管理型であるため，アクセス権限やリソース範囲といった代表的なアクセス制御情報は保持しない．ノード秘密鍵は権限証明書の署名に利用し，ノード公開鍵は署名の検証や公開認証情報の作成に利用する．AC ノードが保持する情報は，ノード公開鍵も含め，ユーザに対して公開されない性質のものとする．特にノード秘密鍵についてはより厳重に管理されているものとする．

ノード公開鍵リストは，ノード ID ごとに利用開始日時とノード公開鍵と漏洩フラグ[*3]の三つ組をエントリとして持つリストで，権限証明書の検証時やユーザの認証時に利用する．

ノード経路表は，AC ノードの ID とその AC ノードに接続するための情報[*4]を対応させた表で，AC ノードが他の AC ノードと通信するときに利用する．

ユーザはリクエストごとまたはセッションごとにロードバランサや DNS ラウンドロビン等の仕組みによって，1 台の AC ノードに割り振られ，その 1 台とのみ通信を行う．同時に複数台の AC ノードとは直接通信を行わない．また，ユーザと AC ノードとの通信や AC ノードどうしの通信は TLS（Transport Layer Security）/SSL（Secure Sockets Layer）等によって秘匿性が担保されているものとする．

### 3.2　権限証明書によるアクセス制御

まず証明書の細かい記述に移る前に，権限証明書を用いたアクセス制御の流れを**図 2** に示す．前提として，ユーザは AC ノードを介してのみ分散ファイルシステムにアクセスできるものとする．つまり，AC ノードを介さずに直接分散ファイルシステムにアクセスすることはできないものとする．

ユーザは分散ファイルシステムのリソースに対してアクセスする場合，操作要求（たとえばファイル読み込みやディレクトリ作成等）に所有する権限証明書を添えて，AC ノードに対して送信する．AC ノード側では，1) 提示された証明書が有効であり，2) 要求された操作が提示された証明書において許可されており，3) ユーザが提示された証明書の所有者であることが認証された，場合にのみ対象の分散ファイルシステムに対して操作要求を仲介する．この際，1) の証明書の検証処理において，AC ノードは他の AC ノードと通信を行い CRL や CUL 等を参照する．また，3) の認証処理は操作要求のたびに毎回行う必要はなく，セッション単位等で結果をキャッシュすることも可能である[*5]．

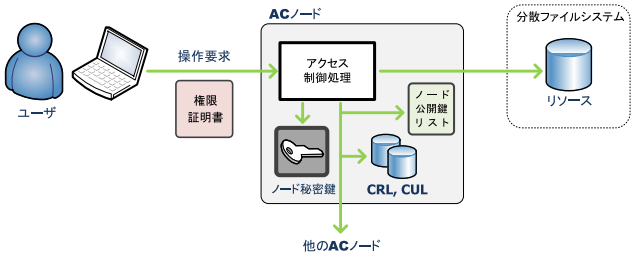


図 2　アクセス制御機構の振舞い
**Fig. 2**　Behavior of our ACM.

#### 3.2.1　証明書を用いた権限委譲による登録・設定コストの分散

提案機構で利用する権限証明書は**表 2** に示す情報で構成される．

提案機構では既存研究 [4], [5], [6] と同様に，ユーザは所有する権限証明書をもとに，そのサブセット権限を有する新たな権限証明書を発行することができる[*6]．

委譲による証明書発行の前提として，連鎖証明書リストが空で委譲のルートとなる証明書（ルート証明書）が少なくとも 1 つ以上存在するものとする．この前提の実現方法としてはいくつかの選択肢がある（ただし，提案機構は特に実現方法を限定はしない）．たとえば，初期状態で全リソースに対してすべての操作が可能なルート証明書を管理者が所有している状態にする，という方法もあれば，管理者は特権操作で AC ノードに対して任意のルート証明書を発行させられるようにする，という方法もある．

**図 3** に権限証明書を用いた権限委譲の例を示す．証明

*3　対応するノード秘密鍵の漏洩が疑われる場合にこのフラグを立てる．
*4　たとえば URL や IP アドレスとポート番号等．
*5　ユーザの利用するクライアントプログラムが一時的に認証情報（パスワード等）をキャッシュすることも可能である．
*6　サブセットなので所有する権限とまったく同じ権限を与えることも可能である．

表 2　権限証明書の構成情報
**Table 2**　Information contained in authorization certificates.

| 項目 | 内容 |
|---|---|
| 証明書 ID | 提案機構全体でユニークな証明書の ID[*7] |
| 対象リソース | 対象となる分散ファイルシステム上のリソースの集合 |
| 許可する操作 | 対象リソースに対して許可する操作の集合 |
| 有効期間 | 証明書の有効期間（開始日時と終了日時） |
| 連鎖証明書リスト | 権限の委譲元を示す証明書のリスト |
| 認証方式 | 利用できる認証方式（公開鍵認証またはパスワード認証） |
| 公開認証情報 | 認証時に利用される公開可能な情報（詳細は 3.2.2 項） |
| 発行日時 | 証明書の発行日時（更新の場合は更新日時） |
| 発行者ノード ID | 証明書を発行した AC ノードのノード ID |
| 署名 | 証明書の内容に対するデジタル署名 |

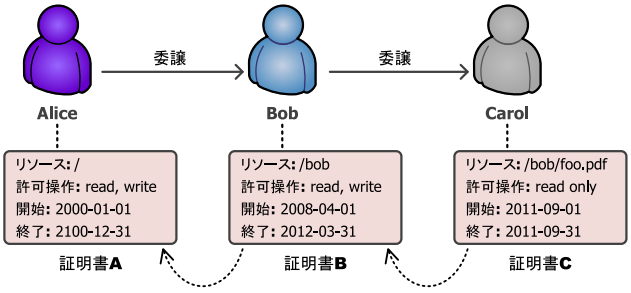


図 3　権限証明書を用いた権限委譲
**Fig. 3**　Delegation by using authorization certificates.

書 C の連鎖証明書リストには証明書 B と証明書 A が含まれ，証明書 B の連鎖証明書リストには証明書 A が含まれ，証明書 A の連鎖証明書リストは空（つまりルート証明書）である．

この権限委譲を行う際，機構側に事前にユーザを登録する必要はない．提案機構のおけるユーザとは，認証時に本人しか知りえない知識（秘密鍵やパスワード）を検証可能な形で提示できる存在でさえあればよい．

なお，提案機構は既存研究 [4], [5], [6] とは異なり，必ず提案機構の AC ノードを通じて新たな権限証明書を発行する．以下に委譲元となる証明書 $\mathcal{C}_p$ を使って新しく権限証明書を発行する手順を示す．

**手順 A：権限証明書の発行**

**step 1**　ユーザは AC ノードに対して委譲元の証明書 $\mathcal{C}_p$ および新たに発行する証明書の権限情報（対象リソース，許可する操作，有効期限）と初期パスワードを送る．

**step 2**　AC ノードは委譲元の証明書 $\mathcal{C}_p$ の検証を行う（詳細は手順 F）．

**step 3**　AC ノードはユーザに対して委譲元の証明書 $\mathcal{C}_p$ の所有者であることの認証を行う（詳細は手順 B）．

**step 4**　AC ノードは委譲元の証明書 $\mathcal{C}_p$ と対比して，新しい証明書の権限情報に不正（権限の拡大や期限の延長）がないかを確認する．

**step 5**　AC ノードは新しい証明書の証明書 ID を生成して設定する[*8]．

**step 6**　AC ノードは初期パスワードから公開認証情報を計算して設定する（詳細は 3.2.2 項の式 (1)）．

**step 7**　AC ノードは新しい証明書の連鎖証明書リストに委譲元の証明書 $\mathcal{C}_p$ の連鎖証明書リストに $\mathcal{C}_p$ 自身を加えたリストを設定する．

**step 8**　AC ノードは新しい証明書の発行日時に現在時刻を，発行ノード ID に自身のノード ID をそれぞれ設定する．

**step 9**　AC ノードは自身のノード秘密鍵を用いて署名し，新しい証明書を発行する．

上記の手順では許諾者が初期パスワードを設定し，その初期パスワードを対象ユーザに伝達することを想定している．ただし，その手段については特に制限を設けていない[*9]．別の選択として，初期パスワードの設定や伝達を AC ノード（機構側）に委ねることもできる．この場合，機構側から対象ユーザに直接データをやりとりを可能とする補足情報（たとえばメールアドレス）を証明書に含めればよい．AC ノードは step 1 で初期パスワードを許諾者から受け取る代わりにランダムに生成し，上記手順の完了後にメール等で対象ユーザに初期パスワードを伝達すればよい．権限委譲（証明書の新規発行）以降は，証明書の所有者が後の手順 C で示す手順で認証情報の更新（パスワードの変更）を必要に応じて行う．

これにより，管理者やオーナ等の特定のユーザに限らず，自分の有する権限の範囲内で，アクセス制御の設定を行うことが可能になり，人的な登録・設定コストを分散することができる．

### 3.2.2　公開認証情報によるパスワード認証の実現

分散管理型のアクセス制御機構における問題点の 1 つが

*7　後述する証明書の更新を行っても証明書 ID は変わらない．
*8　UUID 等の提案機構全体で一意な識別子を生成する．
*9　許諾者が口頭やメール，その他所属組織の定める手段等で対象ユーザに伝達することを想定している．

パスワード認証が利用できないことであることは先に述べた．Miltchev らの調査 [2] からも分散管理型のアクセス制御機構はすべて，認証方式として公開鍵認証のみが利用可能とされている．既存研究 [4], [5], [6] の権限証明書には所有者の公開鍵が含まれており，機構側はその公開鍵のみを用いて認証（チャレンジ・レスポンス認証）を行うことができる．また，公開鍵は元より公開されることを前提としているため，その情報が漏洩した場合においても安全性が担保されている．

しかし，公開鍵認証のみが，上で述べた性質を満たすものではない．提案機構では，公開鍵に代えて，新たに公開認証情報と呼ぶ概念を導入した．公開認証情報とは以下の条件を満たす情報である．

i) アクセス制御機構がその情報のみを用いて認証を行うことができる．
ii) その情報が漏洩した場合においても認証の安全性が担保できる．

ii) が満たされることで権限証明書にその情報を含めることが可能となり，i) が満たされることでアクセス制御機構は認証情報を保持する必要がなくなる．また定義から明らかなように，公開鍵認証に用いられる公開鍵は公開認証情報の一種であるといえる．ここでのポイントは，i) の条件において，「アクセス制御機構が」と対象者を限定している点である*10．

提案機構では，パスワードの平文 *password* に対して，証明書の新規発行や認証情報の更新ごとに乱数を生成し，それらを結合したうえで一方向性ハッシュ関数を用いて一方向化し，その結果を各 AC ノードが保持するノード秘密鍵に対応するノード公開鍵 $K_{npb}$ を用いて暗号化することでパスワード認証用の公開認証情報 *oauth* を作成する．

$$oauth = r_1 \parallel r_2 \parallel \mathcal{E}_{K_{npb}}\Big(r_2 \parallel \mathcal{H}\left(r_1 \parallel password\right)\Big),\quad r_1, r_2 \text{ は乱数} \tag{1}$$

なお，$c = a \parallel b$ は情報 $a$ と情報 $b$ を連結した情報 $c$ を，$h = \mathcal{H}(M)$ はメッセージ $M$ に対して一方向性ハッシュ関数を適応した結果 $h$ を，$c = \mathcal{E}_K(M)$ はメッセージ $M$ を公開鍵 $K$ を用いて暗号化した結果 $c$ をそれぞれ示す．

式 (1) の値を用いて，提案機構では以下の手順でパスワード認証を実現する．

**手順 B：権限証明書に対する認証（パスワード認証の場合）**

**step 1**　ユーザは AC ノードに対して証明書およびパスワード $p$ を送る．
**step 2**　AC ノードは証明書の認証方式としてパスワード認証が可能であることを確認する．
**step 3**　AC ノードは証明書から公開認証情報 *oauth* を取得する．
**step 4**　AC ノードはノード公開鍵リストから証明書の発行時に用いられたノード公開鍵 $K_{npb}$ を取得する（詳細は手順 E）．
**step 5**　AC ノードは $c = \mathcal{E}_{K_{pub}}\Big(r_2 \parallel \mathcal{H}\left(r_1 \parallel p\right)\Big)$ が成立することを検証する．ただし $r_1 \parallel r_2 \parallel c = oauth$*11．

以上のように提案機構はユーザから提示された情報（証明書とパスワード）のみを用いてパスワード認証を実現することができる．

次に式 (1) の値の安全性について，攻撃者が他人の証明書を入手した場合を考える．攻撃者は公開認証情報から乱数 $r_1$ と $r_2$ を入手することはできる*12．しかし，公開認証情報 *oauth* を求める過程で一方向化されているため，逆向きにパスワードを求めることは困難であるといえる．また，一方向化の際に乱数が結合されているため辞書攻撃も行えない．加えて，通常，攻撃者は暗号化や復号化に必要なる鍵ペア（ノード秘密鍵とノード公開鍵）を知りえないため，*oauth* の値を計算したり，効率的な攻撃を行うことはできない*13．

以上より，式 (1) の値は公開認証情報であり，パスワードは公開認証情報として，安全性が担保された状態で証明書内に保存されることとなる．したがって，提案機構においては，証明書の管理がパスワードの管理（保管）を意味するため，LDAP 等の外部にパスワードの保管手段を必要としない．提案機構は分散管理型のアクセス制御機構であり，証明書はユーザ側で管理されるため，パスワード管理を行う必要はない．提案機構（AC ノード）はユーザが提示する証明書とパスワードのみを用いてパスワード認証を実現できる．

#### 3.2.3　ノード秘密鍵を用いた署名による自律的な認証情報の更新

既存研究 [4], [5], [6] では，権限証明書の署名に許諾者の秘密鍵を用いるため，所有者が許諾者の介在なく自律的に認証情報（秘密鍵やパスワード）を更新することができない．

そこで提案機構では，許諾者の秘密鍵に代えて，AC ノードごとに鍵ペアを用意し，ノード秘密鍵を用いて権限証明書に署名する．権限委譲による新しい証明書の発行については 3.2.1 項ですでに述べた．**図 4** に認証情報の更新の概要と以下にその手順を示す（CUL については次節で詳しく述べる）．

*10 公開鍵認証の場合，通常「誰でも」認証（検証）を行うことができる．

*11 $a \parallel b = c$ は情報 $c$ を分割して得た情報 $a$ と情報 $b$ を示す．つまり，*oauth* から $r_1$，$r_2$，$c$ を分割して取り出している．

*12 より安全を期すため，式 (1) の値をさらにノード秘密鍵で暗号化する等の対策をとることで，攻撃者による乱数の入手を防ぐことも可能ではある．ただし，効率的な総当たり攻撃（選択平文攻撃）が可能になるのは，この対策によらず，ノード公開鍵を攻撃者が入手できた場合のみである．

*13 AC ノードが保持する情報が漏洩した場合の影響と対策については 4.1 節で議論する．

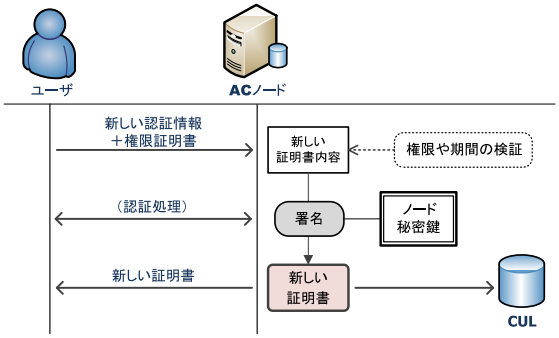


図 4　ノード秘密鍵による署名（認証情報の更新）
**Fig. 4**　Signing using node private key.

**手順 C：認証情報の更新**

**step 1**　ユーザは更新したい証明書 $\mathcal{C}$ と新しい認証情報（パスワード）を AC ノードに送る．

**step 2**　AC ノードは証明書 $\mathcal{C}$ の検証を行う（詳細は手順 F）．

**step 3**　AC ノードはユーザに対して証明書 $\mathcal{C}$ の所有者であることの認証を行う（詳細は手順 B）．

**step 4**　AC ノードは証明書 $\mathcal{C}$ の内容をコピーして新しい証明書 $\mathcal{C}'$ を作成する．

**step 5**　AC ノードは新しい認証情報から公開認証情報を生成して，新しい証明書 $\mathcal{C}'$ に設定する．

**step 6**　AC ノードは新しい証明書 $\mathcal{C}'$ の発行日時に現在時刻を設定する．

**step 7**　AC ノードは自身のノード秘密鍵を用いて署名し，新しい証明書 $\mathcal{C}'$ を発行する．

**step 8**　AC ノードは新しい証明書 $\mathcal{C}'$ をユーザに返し，同時に CUL に登録する．

このように，機構側がノード秘密鍵を用いて署名することで，所有者は許諾者によることなく，自由に認証情報を更新することができる．なお，この手順でかかるコストについては 4.4.1 項で述べる．

また，機構側が署名する副次的なメリットとして，検証コストの低減がある．既存研究 [4], [5], [6] ではユーザが自由に，証明書を作成・発行できるため，証明書の検証時に権限・範囲の拡大や期限の延長がないかをチェックしなければならないが，提案機構では機構側が証明書を発行するため，その際に必要なチェックを行えるため，検証コストが少なくて済む．

提案機構では利用するパスワードの数について特に制約は存在しない．つまり，ユーザが複数の証明書を保持している場合，個々の証明書に別々のパスワードを設定することも可能であるし，共通するパスワードを設定することも可能である．ユーザが所有するすべての証明書のパスワードを共通して運用したい場合，上で述べた認証情報の更新は所有する証明書の数の分，手順 C を繰り返し行う必要がある．ただ，これらの処理はクライアントプログラム側で一括して行う等の工夫も可能である．

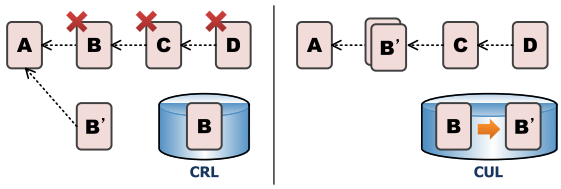


図 5　証明書の無効化と更新の区別
**Fig. 5**　Revocation and update of certificates.

### 3.2.4　無効化と更新の区別による副作用の抑制

既存研究 [4], [5], [6] では，明示的に証明書を更新する手段は提供されておらず，証明書の更新は証明書の無効化と再発行で実現されている．このことが，権限の委譲によって連鎖している一連の証明書の無効化という予期せぬ副作用を生むことは先に述べた．

そこで提案機構では，明示的に証明書の無効化と更新を区別する．具体的には，無効化された証明書を格納する CRL（Certificate Revocation List）とは別に，更新された証明書を格納する CUL（Certificate Update List）を新たに導入する．

ある証明書もしくはその連鎖証明書リスト内の証明書のどれか 1 つでも CRL に含まれていた場合，その証明書は無効と見なされる．それに対して，対象の証明書が CUL に含まれていた場合，その古い証明書のみ（連鎖する証明書は対象外）が無効となり，CUL に保持されている新しい証明書に置き換えられる（**図 5**）．以下に権限証明書 $\mathcal{C}$ を更新する手順を示す．

**手順 D：権限証明書の更新（権限変更）**

**step 1**　ユーザは AC ノードに対して更新したい古い証明書 $\mathcal{C}$ と新しい証明書の権限情報，そして所有する証明書 $\mathcal{C}_u$ を送る．

**step 2**　AC ノードは証明書 $\mathcal{C}$ および $\mathcal{C}_u$ の検証を行う（詳細は手順 F）．

**step 3**　AC ノードはユーザの証明書 $\mathcal{C}_u$ が，対象の証明書 $\mathcal{C}$ の連鎖証明書リストの中に存在すること（$\mathcal{C}$ の祖先であること）を確認する．

**step 4**　AC ノードはユーザの証明書 $\mathcal{C}_u$ と対比して，新しい証明書の権限情報に不正（権限の拡大や期限の延長）がないかを確認する．

**step 5**　AC ノードはユーザに対して証明書 $\mathcal{C}_u$ の所有者であることの認証を行う（詳細は手順 B）．

**step 6**　AC ノードは証明書 $\mathcal{C}$ の内容をコピーして新しい証明書 $\mathcal{C}'$ を作成する．

**step 7**　AC ノードは新しい権限情報を新しい証明書 $\mathcal{C}'$ に設定する．

**step 8**　AC ノードは新しい証明書 $\mathcal{C}'$ の発行日時に現在時刻を設定する．

**step 9**　AC ノードは自身のノード秘密鍵を用いて署名

し，新しい証明書 $\mathcal{C}'$ を発行する．

**step 10**　AC ノードは証明書 $\mathcal{C}$ の証明書 ID と新しい証明書 $\mathcal{C}'$ のペアを適切な AC ノードの CUL に格納する[*14]．

上で示した手順により，更新された証明書が CUL に格納される．提案機構では，証明書の検証時に CUL を参照し，証明書が古い場合，CUL に登録されている最新の証明書に置き換える．

証明書の検証手順を示す前に，検証時に必要となるノード公開鍵をノード公開鍵リストから取得する手順を以下に示す．

**手順 E：権限証明書に対応するノード公開鍵の取得**

**step 1**　AC ノードは証明書 $\mathcal{C}$ から発行日時と発行者ノード ID を取得する．

**step 2**　AC ノードはノード公開鍵リストから発行者ノード ID に対応するエントリを検索する．

**step 3**　AC ノードは該当するエントリのうち，「（証明書の）発行日時 >（ノード鍵ペアの）利用開始日時」を満たす最も利用開始日時の大きなエントリを取得する．

**step 4**　AC ノードは該当エントリのノード公開鍵と漏洩フラグを返す．

ノード公開鍵リストには過去から現在までの全 AC ノードのノード公開鍵が格納されている．このため，上に示した手順によって，発行や更新を担当していない AC ノードであっても，証明書検証時やユーザ認証時に必要となるノード公開鍵を取得することができる．

それでは，権限証明書を検証する手順を次に示す．

**手順 F：権限証明書の検証**

**step 1**　AC ノードは現在時刻が証明書 $\mathcal{C}$ の有効期限の範囲内であることを確認する．

**step 2**　AC ノードは証明書 $\mathcal{C}$ とその連鎖証明書リストに含まれるすべての証明書がいずれも無効でないことを確認する（詳細は手順 H）．

**step 3**　AC ノードはノード公開鍵リストから証明書 $\mathcal{C}$ の発行時に用いられたノード公開鍵 $K_{npb}$ と漏洩フラグを取得する（詳細は手順 E）．もし対応するノード秘密鍵が漏洩していた場合，検証は失敗する．

**step 4**　AC ノードはノード公開鍵 $K_{npb}$ を用いて証明書 $\mathcal{C}$ の署名を検証する．

**step 5**　AC ノードは証明書 $\mathcal{C}$ が更新されているか（証明書 ID が CUL に含まれているか）を確認し，含まれていた場合，CUL に登録されている最新の証明書 $\mathcal{C}'$ を用いて以降の処理を継続する．

証明書の検証は，ユーザからの操作要求の可否をチェックするときはもちろんのこと，証明書の発行や更新等の権限証明書が利用される操作で必ず実行される．このため，遅くとも検証時には，すべての証明書の無効化（上記 step 2）および更新（上記 step 5）が検出されることを保証できる．

たとえば，Alice がパスワードの変更にともなう証明書の更新を行った場合を考える．攻撃者が Alice の古い証明書とそのパスワードを不正に入手してアクセスを試みた場合，CUL に Alice の証明書の ID に対応する新しい証明書が登録されているため，証明書が最新のものに置き換えられ，認証に失敗する．

別の例として，Bob が Carol に権限委譲しているケースを考える（図 3 と同じケース）．最初 Bob は Carol に Bob 自身が全権限を有する /bob/foo.doc に対して読み込み権限のみを与えて権限委譲していたが，後に書き込み権限も与える必要が発生したとする．この場合，Bob の証明書は Bob が Carol に対して発行した証明書の親（つまり Carol の証明書の連鎖に Bob の証明書が含まれている）であり，Bob の証明書には書き込み権限が含まれているため，書き込み権限を加えた証明書を Carol の新しい証明書として CUL に登録できる．Carol は Bob による証明書の更新を意識する必要はなく，自身の保有している古い証明書を使って，書き込み処理を要求すれば，証明書が最新のものに置き換えられ，書き込み処理が許可される．

### 3.3　アクセス制御機構の分散化

権限証明書を用いたアクセス制御を行う場合，アクセス制御に必要な情報は権限証明書に含まれている．このため，基本的に個々の AC ノードで情報を保持する必要がなく，個々の AC ノードは独立してアクセス制御が行える．この性質により，アクセス制御を行うノードを増やすことで，容易に単一障害点を排除し，性能をスケールアウトすることができる．

しかし，例外的にこの性質を満たさないのが，無効化した証明書を保持する CRL や更新した証明書を保持する CUL である．CRL および CUL はアクセス制御機構側で保持し，ユーザから送られてきた権限証明書が無効でないか等を確認するために全 AC ノードから参照できる必要がある．仮に，CRL や CUL を特定のノード上に構築した場合，その部分がアクセス制御機構全体にとっての単一障害点となり，また同時にボトルネックとなって，性能のスケールアウトを妨げる．

これらの情報以外にもノード経路表やノード公開鍵リストもノード間通信や証明書検証・ユーザ認証時に必要になるため，全 AC ノードで利用できる必要がある．しかし，ノードの追加・離脱やノードの鍵ペア更新の頻度は，証明書の無効化や更新の頻度と比べて低いと考える．

提案機構では，ノード間で接続関係を構築する手段につ

[*14] 同じ ID に対してすでに CUL に証明書が登録されていた場合，発行日時の新しい方で上書きする．

いて特に制限を持たない．AC ノードがノード経路表を用いて他の AC ノードとメッセージを交換できさえすればよい．ただし，通信コストの面では，提案機構では，ノードの頻繁な追加や離脱の発生しない，比較的静的な環境を想定し，余分な仲介なしで直接目的の AC ノードと通信できるものとする．したがって，各 AC ノードはすべての AC ノードの ID と接続情報が含まれたノード経路表を持つ．つまり，接続関係の構築は単純には各 AC ノードにノード経路表を配布することで実現できる．

より大規模でかつ高頻度のノードの追加・離脱が発生する環境を想定する場合であれば，Chord [14] 等の大規模で動的な P2P 環境に適したアルゴリズムを用いることで，提案機構を活用することは可能と考える[*15]．

AC ノードの数は無効化/更新される証明書の数と比べて十分に小さいと考えられるため，ノード経路表やノード公開鍵リストの保存に必要なデータ量も十分に小さいといえる．このため，ノード経路表とノード公開鍵リストの管理は分散化せずに単純な複製の仕組みでも十分に機能すると考える．たとえばノード経路表であれば，ノードの追加や離脱が検出されたタイミングでファイルとして全 AC ノードにコピーで配布する，等である．したがって，本稿ではノード経路表やノード公開鍵リストの分散化については言及せず，AC ノードに備わっている情報として扱う．以降では CRL および CUL を分散化する仕組みについて述べる．

提案機構では，Consistent Hashing を用いて，CRL や CUL を全 AC ノードにまたがって分散構築する．各 AC ノードにはノード ID を割り振られており，それを環状のハッシュ空間の上に配置する．そして，証明書の内容からハッシュ関数[*16]によってハッシュ値を求め，その値に（順方向で）最も近い AC ノードに証明書を保存し，その近傍の AC ノードに証明書の複製を保存する．証明書は冗長数を示すパラメータ $k$ 台の AC ノードに保存する（**図 6**）．

### 3.3.1　CUL の分散化

CRL の分散化について登録・参照の両側面から述べる．まず，証明書 $\mathcal{C}$ を無効化する手順を下に示す．

**手順 G：権限証明書の無効化（CRL への登録）**

**step 1**　ユーザは AC ノード（以降，処理ノード）に対して無効化したい証明書 $\mathcal{C}$ と所有する証明書 $\mathcal{C}_u$ を送る．

**step 2**　処理ノードは証明書 $\mathcal{C}$ および $\mathcal{C}_u$ の検証を行う（詳細は手順 F）．

**step 3**　処理ノードはユーザの証明書 $\mathcal{C}_u$ が，対象の証明書 $\mathcal{C}$ と同じか，その連鎖証明書リストの中に存在すること（$\mathcal{C}$ の祖先であること）を確認する．

**step 4**　処理ノードはユーザに対して証明書 $\mathcal{C}_u$ の所有者であることの認証を行う（詳細は手順 B）．

**step 5**　処理ノードは証明書 $\mathcal{C}$ のハッシュ値を求める．

**step 6**　処理ノードは証明書 $\mathcal{C}$ のハッシュ値から最も近傍の利用可能な AC ノード（以降，担当ノード）に証明書 $\mathcal{C}$ を転送する．

**step 7**　担当ノードは証明書 $\mathcal{C}$ を自身の CRL および複製保存用の $k-1$ 台の AC ノードの CRL に登録する．

**step 8**　担当ノードは無効化完了を転送元の処理ノードに通知する．

**step 9**　処理ノードは無効化完了をユーザに通知する．

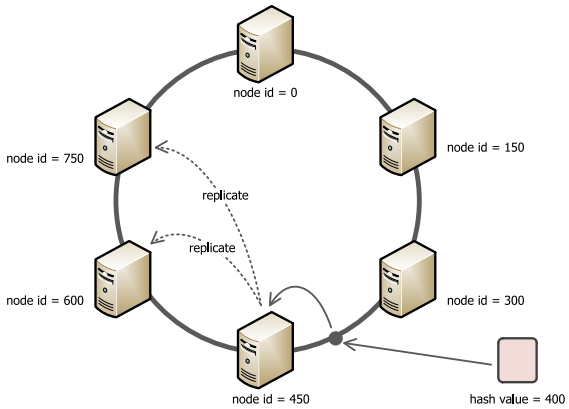


**図 6**　Consistent Hashing による CRL，CUL の分散管理（$k = 3$ のとき）

**Fig. 6**　Decentralized CRL and CUL by using consistent hashing ($k = 3$).

図 6 は証明書のハッシュ値が 400 で $k = 3$ の場合の例であり，担当ノードはノード ID が 450 の AC ノードで，複製を保存する 2 つの AC ノードのノード ID は 600 と 750 である．担当ノードは step 7 において，$k-1$ 回の通信を行い，CRL にエントリの複製を保存しているため，step 9 で無効化の完了を通知した時点では，$k$ 台のノードが CRL の構成情報（証明書）を保持することが保証されている．これにより，同時に任意の $k-1$ 台の AC ノードが故障した場合においても，提案機構の CRL はデータ損失が発生せず機能することができる．

次いで，証明書の検証の際に CRL を各 AC ノードから参照する手順を以下に示す．

**手順 H：権限証明書の無効化確認（CRL の参照）**

**step 1**　AC ノード（以降，処理ノード）は証明書 $\mathcal{C}$ のハッシュ値を求める．

**step 2**　処理ノードは証明書 $\mathcal{C}$ のハッシュ値から，複製も含めて対応する CRL を保持している AC ノード（保持ノード）[*17]を求める．

[*15] ただしこの場合，目的の AC ノードと通信するために他の AC ノードを介することがあるため，通信コストが本稿での評価よりは増加すると考える．

[*16] このハッシュ関数は一方向性を満たさなくても，一様性が満たされていればよい．

[*17] 図 6 の場合，保持ノードにはノード ID が 450，600 および 750 の AC ノード 3 台が該当する．

**step 3**　処理ノードは保持ノードのうちの利用可能な任意の1台に対して証明書 $\mathcal{C}$ の証明書 ID を送る．

**step 4**　保持ノードは送られてきた証明書 ID に一致する証明書が CRL に登録されているかを確認して，その結果を処理ノードに返す．

**step 5**　処理ノードは保持ノードからの結果をそのまま無効化確認の結果として返す．

上の手順によって，どの AC ノードからであってもすべての証明書の無効化および確認をすることができる．証明書の検証時には，対象の証明書に加えて，その連鎖証明書リストに含まれる証明書の無効化も確認するため，上記の手順が連鎖する証明書の数だけ繰り返されることとなる．ただし，同じ AC ノードに対する複数の証明書の CRL 参照はまとめて行うことで，通信回数を抑えることとする．たとえば図 6 のケースで，10 個の連鎖する証明書のハッシュ値がすべて 400 近辺の値だった場合，ノード ID が 450（600 や 750 でもよい）のノードの CRL を 1 度参照すれば 10 個すべての証明書の無効化確認をすることができる．

また，step 3 において処理ノードは利用可能な保持ノードに対して CRL 参照を行っている．このため，$k-1$ 台の AC ノードが故障している場合においても，少なくとも 1 台は接続可能な保持ノードが存在する．これにより，$k-1$ 台以下のノード故障時も，通常と同じように証明書の無効化確認を行うことができる．

#### 3.3.2　CUL の分散化

ここまで CRL の分散化について述べたが，CUL の分散化も基本的に CRL の場合と同じである．権限証明書の更新（CUL への登録）については，登録時に証明書 ID と証明書のペアを保存する以外は，CRL への登録（手順 G）と同じ手順である．権限証明書の更新確認（CUL の参照）についても，保持ノードが証明書 ID に対応するエントリがあった場合，登録されている最新の証明書を処理ノードに返す以外は，CRL の参照（手順 H）と同じ手順である．

#### 3.3.3　権限証明書からハッシュ値の算出方法

ここで，証明書からハッシュ値を求める方法について，分散の偏りを制御する方法と合わせて述べる．証明書 $\mathcal{C}$ からハッシュ値を求める方法はいくつも考えることができるが，以下に 2 つの方法を示す．

**方法 a**　証明書 $\mathcal{C}$ の証明書 ID をハッシュ化する．

**方法 b**　証明書 $\mathcal{C}$ の連鎖証明書リスト内の証明書をルートから順番に並べ，先頭（ルート）から $l$ 番目の証明書の証明書 ID をハッシュ化する．ただし，連鎖証明書リストの長さが $l$ より小さい場合は，証明書 $\mathcal{C}$ の証明書 ID をハッシュ化する．

方法 a は単純な[*18]方法であり，すべての証明書に対して偏りのないハッシュ値を得ることができる．方法 b も最終的にはハッシュ関数が適用されるので一様性は期待できるが，特定の条件で恣意的に偏りが発生するようになっている．それは，$l$ 回以上の委譲によって発行されている証明書について，$l$ 回目の委譲で委譲元となった証明書が同じであれば必ず同じハッシュ値になる，というものである．この偏りによって，委譲によってつながれた，いかなる長さの証明書の連鎖の無効化確認を行う場合においても，たかだか $l$ 台の AC ノードの CRL の参照で実現することができる．なぜならば，$l$ 回目以降の委譲で生成された証明書はすべて同じハッシュ値を持つため，（無効化されているとすれば）同一の AC ノード上の CRL に保存されているはずだからである．

本章では提案機構で保持する情報である CRL・CUL について Consistent Hashing を用いて分散化する方法について述べた．提案機構の CRL および CUL は，どの AC ノードからも登録・参照が行え，ノード故障に対しても耐性を持つことを示した．

## 4.　評価

本章では，提案機構のセキュリティ，登録・設定コスト，可用性とスケーラビリティの 4 点についての評価を行う．

### 4.1　セキュリティ

本節では提案機構の安全性の評価として，ノード秘密鍵が漏洩した場合の影響とその対策について述べる．

#### 4.1.1　秘密鍵漏洩時の影響

AC ノードが保持しているノード秘密鍵が漏洩した場合，攻撃者はすべてのリソースに任意の操作が可能な権限証明書を発行することができてしまう．これはアクセス制御機構としての完全破綻である．また，対となるノード公開鍵で計算された公開認証情報については，漏洩したノード秘密鍵を用いて暗号化を解除することが可能なため，パスワードの保護が一方向性ハッシュ関数のみとなってしまう[*19]．

ただし，既存研究においてもルートとなる権限証明書の所有者である管理者（たち）の秘密鍵が 1 つ以上存在する．既存研究では，ユーザは任意の鍵ペアを利用し，それを用いて権限証明書に署名するため，単に署名が正しいというだけでは有効性を判断できない[*20]．そこで既存研究では，管理者の秘密鍵で署名された権限証明書が連鎖のルートに含まれていることを，あらかじめ機構側に登録されている管理者の公開鍵を用いて検証することで，権限証明書の有効性を判断する．

このため，これらの秘密鍵が 1 つでも漏洩した場合，提

[*18] 方法 b が一般ケースで，方法 a は方法 b において $l=\infty$ の特殊なケースと考えることもできる．

[*19] ただし，乱数と結合したうえで一方向化しているため効率的な辞書攻撃は利用できない．

[*20] 攻撃者が勝手に生成した鍵ペアで署名した証明書である可能性がある．

案機構のノード秘密鍵が漏洩した場合と同様に，攻撃者はすべてのリソースに任意の操作が可能な権限証明書を発行することができる．なお攻撃者は管理者の秘密鍵を得ているため，権限証明書を偽造しなくても，管理者として公開鍵認証での認証をパスし，管理者の権限証明書を使って全権限を行使することもできる[*21]．

以上から，証明書の発行にユーザの秘密鍵を用いる既存研究と比べて，漏洩時の影響範囲が大きいとはいえない．しかし一方で，既存研究ではトータルブレイクを招く秘密鍵の数は制御可能なのに対して，提案機構では AC ノードの台数と同じだけノード秘密鍵が存在するため，秘密鍵が漏洩するリスクはより高いといわざるをえない[*22]．そこで以下では，提案機構でのノード秘密鍵の漏洩への対策について述べる．

#### 4.1.2　定期的な鍵ペアの更新

ノード秘密鍵が漏洩した場合の別の影響として，漏洩発覚後の証明書の無効化がある．漏洩したノード秘密鍵で署名された証明書は無効にする必要があり，結果的にそれらから委譲された証明書も影響を受ける．提案機構では，この無効化の範囲を限定的にとどめるために，いくつかの対策が用意されている．

まず前提として，AC ノードごとに異なる鍵ペアを用いて証明書を発行しているため，漏洩していないノード秘密鍵によって発行された証明書を無効化する必要はない．逆に既存研究で 1 つの秘密鍵で署名された証明書をルートにすべての委譲が行われた場合，漏洩発覚後にはすべての証明書を無効化しなければならない．

提案機構では，AC ノードの鍵ペアを定期的に更新することで，漏洩発覚時の無効化の影響範囲を時区間的に限定することが可能である．提案機構では定期的に AC ノードの鍵ペアを更新する．この際，古いノード秘密鍵は破棄される．このため攻撃者の AC ノードへの侵入時期が特定できれば，ノード公開鍵リストのうち，侵入された AC ノードの，該当する期間のエントリに対してのみ漏洩フラグを立てればよい．これにより，無効化の対象となる証明書をより限定的にすることが可能である．

#### 4.1.3　複数ノードによる署名

ノード秘密鍵の漏洩に対する根本的な対策として複数の AC ノードによる署名について述べる．方法としては，証明書の発行時や更新時に $m$ 台の AC ノードが同様にチェックを行い，$m$ 個の発行者ノード ID と署名を最終的な証明書に含める，というものである[*23]．そして，証明書の検証時には $m$ 個の署名が含まれており，かつすべての署名が妥当である場合にのみ検証をパスさせる．

このようにすることで，証明書の発行や更新のコストは $m$ 倍になることと引き替えに，$m$ 台の AC ノードから秘密鍵が漏洩しない限り，トータルブレイクには陥らない．仮に攻撃者が $m-1$ 台以下の AC ノードからノード秘密鍵を入手して証明書を偽造しようとしたとする．しかしながら，証明書の発行時には少なくとも 1 台のノード秘密鍵の漏洩していない正常な AC ノードが参加するため，不正な内容で証明書を発行することができない．

さらにいえば，$m$ 台の AC ノードでの署名はほぼ同時に行われるため，当然同じ時間帯のノード秘密鍵が用いられる．このため，攻撃者が鍵ペアの更新周期をまたいで別々に $m$ 台の AC ノードからノード秘密鍵を入手したとしても証明書を偽造することはできない．AC ノードごとに鍵ペアの更新周期やタイミングをずらしておくことでさらに攻撃を成立しにくくすることができる．また，$m$ 台の AC ノードの選び方に一定の制約[*24]を設けることでさらに強力にすることも可能と考える．

また，$m-1$ 台以下の AC ノードからのノード秘密鍵の漏洩であれば，漏洩発覚後も既存の証明書を無効化する必要はない．さらに仮に $m$ 台以上からノード秘密鍵が漏洩が発覚した場合も，すべて漏洩した秘密鍵で $m$ 個の署名がされている証明書以外は無効化する必要がなく，漏洩発覚後の影響範囲もきわめて限定的にすることが可能と考える．

なお，この対策はユーザが直接証明書に署名して発行する既存研究 [4], [5], [6] で実現することは難しいと考える．複数の管理者がいる場合に，ユーザがそれぞれの管理者に対して署名を求める必要があるため，発行が完了までに多くの人的コストと時間がかかるためである．これに対して，提案機構では機構側の AC ノードが署名することで，ユーザに対して透過的に，複数の署名を付けるという処理を自動的に行うことができる．

これらの対策によって，同時に，$m$ 台以上の AC ノードからノード秘密鍵が漏洩しない限り，提案機構の安全性を保つことができると考える．

### 4.2　登録・設定コスト

提案機構の登録・設定コストが分散するようになっているかについて述べる．Miltchev らは，ユーザ登録や権限の設定といったオペレーションコストが分散できる，理想的な権限委譲は以下の性質を満たすと定義している [2].

**自律性（*Autonomy*）**：（管理者等の）第三者に頼らずに権限を委譲できること．

**説明性（*Accountability*）**：誰が誰に権限を委譲したかを説明（追跡）できること．

**推移性（*Transitivity*）**：連鎖的に（委譲された権限をさらに別のユーザに）委譲できること．

[*21] 提案機構でパスワード認証が使われている場合，既存の証明書を利用するには正しいパスワードを入手する必要がある．

[*22] ノード秘密鍵を共有することは可能だが侵入される可能性が台数倍あることに変わりはない．

[*23] ルート証明書も当然 $m$ 台の AC ノードによって署名されている．

[*24] ノード ID の値がお互いに一定数以上離れていること，等．

**粒度調整性（*Fine granularity*）**：自分の権限のサブセットが他人に委譲できること．

**組織非依存性（*Organizational independence*）**：自分と異なる組織のユーザにも権限が委譲できること．

提案機構における権限委譲が推移性，粒度調整性を満たすことは明らかである．説明性についても証明書には連鎖証明書リストが含まれており，署名により改ざんができないことから満たされている．組織非依存性は，既存研究では公開鍵認証を用いることでユーザ登録なしでユーザ認証を実現することで満たされている．提案機構では，公開認証情報を用いることでパスワード認証においても事前のユーザ登録なしでユーザ認証を実現しており，組織非依存性を満たしている．

残る自律性について，一見満たされているように見えるが，提案機構では「人的な」自律性を満たしているというのが公平と考える．これは機構側で証明書を発行するようにしたことのトレードオフである．ユーザの秘密鍵を用いて証明書を発行する場合，アクセス制御機構が利用できない場合であっても，ユーザの保有する情報のみを用いて，新しく証明書を作成・発行することができる．

これに対して提案機構の場合，人的には管理者やリソース所有者等の特定のユーザを必要とせず権限を委譲することができ，自律的であるといえる．しかしその一方で，機会的には提案機構が利用できない状況では権限を委譲することができず，自律的であるとはいえない．

ただし，提案機構で掲げている登録・設定コストの分散とは，特定のユーザにユーザ登録や権限設定のコストが集中することを解決することが目的である．このことから，人的な自律性が満たされることで，目的は満たされていると考える．

加えて，提案機構は次節で示すように高可用性を実現しており，機構側の問題でアクセス制御機構が利用できない状況はほとんど発生しないと考える．このため，上で述べたアクセス制御機構が利用できない状況とは，ネットワークが利用できない状況にある等のユーザ側の問題に起因する場合が大半であると考えられる．ユーザがオフラインである場合等は，そもそもアクセス制御の対象となる分散ファイルシステム自体も利用できないため，実用上の問題はないと考える．

以上から，提案機構では登録・設定コストが分散するといえる．

### 4.3　可用性

#### 4.3.1　耐障害性

提案機構の耐障害性について，データと処理の観点から述べる．

まず主たるアクセス制御情報である権限証明書については，機構側で保持する必要がないため，冗長化の必要がない．次に，CRL および CUL については，3.3 節で述べたように $k$ 台の AC ノードへの保存が保証される．したがって，同時に $k-1$ 台の AC ノードが失われた場合にも，CRL および CUL のデータが失われることはない．さらにいえば，提案機構では環状のハッシュ空間でノード ID が連続した $k$ 台の AC ノードに複製を保存するため，$k$ 台の AC ノードが故障した場合も，環状ハッシュ空間で連続した $k$ 台でない限りは，データが失われない．

処理の観点から眺めると，権限証明書の発行や検証，無効化や更新，それらに含まれるユーザ認証等，提案機構で定義されているすべての処理において，必要な情報はユーザ側から提示されるか機構側で冗長化されて保存されている．したがって，いずれの処理も提案機構を構成するどの AC ノードにおいても実行可能である．これにより，ある AC ノードが故障した場合，ユーザは故障した AC ノードの代わりに，別の AC ノードを利用することができる．ロードバランサやユーザの利用するクライアントプログラム等によってノード故障をユーザに意識させずにノードを切り替えることも可能である．

CRL や CUL への登録の最中に関連する AC ノードに障害が発生した場合，特にユーザが通信する AC ノード（処理ノード）から保存を担当するノードに証明書を転送した直後に担当ノードが故障した場合（手順 G の step 6 の直後），その後の処理が行われずに証明書が保存されない．しかし，処理ノードは担当ノードの応答を待ってユーザに操作完了を通知する．したがって，処理ノードは担当ノードから応答が得られない場合，次の候補（次にハッシュ値の値に近い AC ノード）に対して処理を行う等で単一障害点を回避することが可能である．

以上から，提案機構において単一障害点は存在せず，高い可用性を実現しているといえる．

#### 4.3.2　障害からの復帰

次にノード障害が発生した後に CRL および CUL が $k$ 台に保持されている状態（以後，正常状態）に復帰する方法についていくつかの選択肢を述べる．ノード障害が発生した場合に，以下に示すようなリカバリを行うことで，冗長数を維持し，データの喪失を防ぐことができる．

一般的な方法としては，代替となる新しい AC ノードを古い AC ノードと同じノード ID でネットワークに参加させることである．新しい AC ノードは，そのノード ID を持つ AC ノードがハッシュリング上で担当するべきデータ（必ず他の $k-1$ 台ノードが保持している）を周囲の AC ノードから複製して取得する[*25]．そして，ノード経路表やノード公開鍵リストを新しい AC ノードの情報に置き換えて更新することで，提案機構を正常状態に復帰させること

[*25] この処理は，実際のノード故障以外にも，ネットワーク障害等によって一時的に AC ノードどうしが正しく通信ができなくなった場合等に行うことで，正常状態を維持することができる．

ができる．

別の選択肢として，Consistent Hashing におけるリングを縮退させる方法がある．これは長期間にわたってノードが離脱する場合や，性能をスケールダウンさせる場合等に用いる．具体的にはノード経路表から該当する AC ノードの ID のエントリを取り除き，以降の処理から当該 AC ノードをなかったものとして扱う[*26]．ただし，この処理の事前には，取り除く AC ノードが保持していたデータが正しく冗長度 $k$ を維持するために，当該 AC ノードの周辺の AC ノードどうしが通信し，CUL や CRL で該当するエントリをコピーして転送する．

また，ユーザ側（クライアントプログラム側）の障害に対しても，提案機構ではその操作が冪等[*27]の性質を満たすことが可能なため，回復が容易と考える．提案機構への直接的な操作としては，証明書の発行・無効化・更新と認証情報の更新の 4 つがある．まず，新規に証明書を発行する処理は提案機構の状態を変更しないため冪等といえる．また，証明書の無効化も繰り返し無効化を行っても結果は変わらないため冪等といえる．証明書の更新に関しては，最も新しい時刻の証明書が採用されるので，同じ内容で更新処理を繰り返しても結果は同じとなり，冪等と見なすことができる．認証情報の更新（パスワード変更）の場合は，すでに更新が成功していれば，古い証明書は新しいものに置き換わっているため，古いパスワードをもとにしたパスワード変更は失敗するが，状態としては変化しないため，失敗の扱いを工夫する等することで冪等と見なすことができる．これにより，たとえばユーザが複数の証明書のパスワードをまとめて更新している最中に直接通信している AC ノード（処理ノード）がクラッシュした場合も，クライアントプログラム側で新しい AC ノードを選択してパスワード変更の処理を最初から行うことで，期待する状態を得ることが可能となる．

こられの方法によって提案機構においてノードに障害が発生後に正常状態での運用に戻すことが可能となる．

### 4.4　スケーラビリティ

提案機構のスケーラビリティについて，計算量とシミュレーションおよびプロトタイプ実装による実測結果から述べる．

#### 4.4.1　通信コスト

はじめに提案機構の処理の通信コスト（通信回数）について述べる．権限証明書の無効化および更新については，CRL または CUL の保存対象となる AC ノードを証明書から算出して，複製の保存を含めてもたかだか $k$ 回の通信で済む[*28]．

証明書のパスワード変更（認証情報の更新）も証明書の更新と同じ CUL への登録処理であるためたかだか $k$ 回の通信で実現できる．所有する複数の証明書のパスワードを一括して変更する場合，証明書の数に応じた通信スコトが発生する．しかし，ユーザが所有する証明書の数とパスワード変更の頻度を考えた場合，この通信コストは限定的と考える．まず，表 2 でも示したように，証明書には複数のリソースを対象として含めることが可能である．加えて，ディレクトリに権限を設定することで，下位のリソースに同じ権限を設定することが可能である（アクセス権限の継承）[1]．したがって，ACL のようにファイルやディレクトリの数で権限証明書が必要にはならないと考える．そして，パスワード変更の頻度についても，後述で評価の対象とする証明書の検証がすべてのアクセス制御の操作で発生することに比べれば，十分に限定的と考える．

権限証明書の発行については，内包されている検証処理を除くと，他の AC ノードとの通信は発生しない．4.1.3 項で述べた方法で複数台の AC ノードで署名する場合であってもたかだか $m$ 回の通信で済む．証明書の新規発行は，たとえばリソースの所有者が他のユーザにそのリソースを新たに共有する，というような場合に行われるため，その頻度も限定的といえる．

いずれも，これらの処理は AC ノード数によらず一定の通信回数であり，定数コストと見なせる．またその頻度からも実用上の問題とはならないと考える．

ここで，権限証明書の検証処理の通信コストについて考える．権限証明書の検証処理において，通信が発生しうるのは，CRL および CUL のチェックのための通信である．CUL のチェック（証明書の更新確認）は該当の証明書の CRL を保持する AC ノードとのみ通信を行えばよいため，たかだか 1 回の通信であり定数コストと見なせる．したがって，検証処理の計算量としては CRL のチェック（証明書の無効化確認）に必要な通信回数を評価すればよい．

ここで CRL のチェックに計算量に影響するパラメータとして，

- AC ノード数 $n$
- CRL のデータ冗長数 $k$
- 証明書の最大分散数 $d$

の 3 つを考える．このとき，CRL のチェックに必要な通信回数の最悪値は $n-1$ または $d$ のいずれか小さい方の値になる．これはそれぞれ，自分以外の他の AC ノードすべての CRL をチェックする回数（$=n-1$）と，証明書およびその連鎖する証明書が分散保存されている AC ノードの CRL をすべてチェックする回数（$=d$）である．なお，$d$ の値は 3.3.3 項で述べたハッシュ値を求める方法として，

[*26] ただし，当該 AC ノードが発行した証明書が継続して利用できるようにするため，ノード公開鍵リストには変更を加えない．

[*27] 冪等とはその操作を複数回実行した場合であっても，1 度実行したのと同じ結果（状態）となる性質．

[*28] 処理をしている AC ノードが対象 AC ノードに含まれていた場合は通信回数が減る．

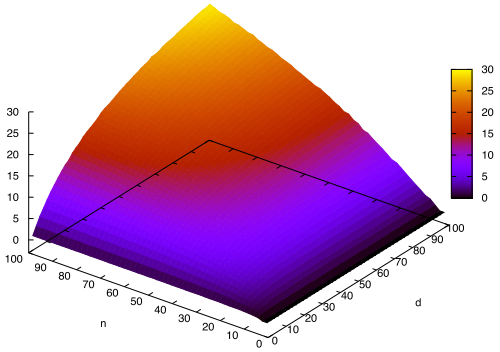


図 7　シミュレーション結果（$k = 3$）
**Fig. 7**　Simulation of the number of communications ($k = 3$).

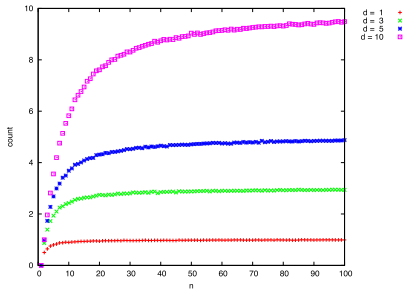


図 8　シミュレーション結果（$k = 1$）
**Fig. 8**　Simulation results ($k = 1$).

方法 a の場合は $d =$ 連鎖証明書リストの長さ $+1$ であり，方法 b の場合は $d = l$ である．

しかし，実際の平均的な通信回数は，$n$ と $d$ の関係，また CRL のデータ冗長数である $k$ によって異なる．たとえば，$k = n$ のときは，各 AC ノードに全証明書の CRL が割り当てられている（ミラーリング状態）ため，通信の必要がなくなる．

### 4.4.2　シミュレーション

計算量の期待値を求める目的で，3.3 節の手順 H で示した手順によって証明書の無効化確認のために発生する通信回数のシミュレーションプログラムを作成して評価を行った．なお，証明書は各 AC ノードに一様に分布するものとし[*29]，ランダムに選んだ $d$ 個の証明書が連鎖していたものとして，その無効化確認のために通信する必要がある AC ノードの個数を計測する処理を 1,000 回試行し，その平均値を計算量として評価した．

**図 7** が $k = 3$ の場合のシミュレーション結果である．$n$ および $d$ の増加にともない通信回数は増加することが確認できるが，$n = 100$，$d = 100$ の場合であっても，平均の通信回数は 29.42 と最悪値よりも少ないことが分かる．また，実際には証明書の連鎖の深さを表す $d$ がこれほど大きい値をとるとは考えにくいため，$d = 1, 3, 5, 10$ の範囲でシミュレーションした結果を**図 8**，**図 9**，**図 10**，**図 11** に示す．

*29 担当 AC ノードは証明書 ID のハッシュ値をもとに決定するため，十分な数の証明書が存在すれば，各 AC ノードに対して一様に分布することになる．

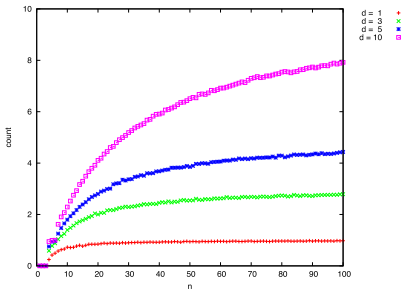


図 9　シミュレーション結果（$k = 3$）
**Fig. 9**　Simulation results ($k = 3$).

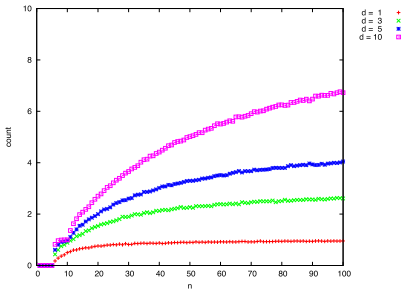


図 10　シミュレーション結果（$k = 5$）
**Fig. 10**　Simulation results ($k = 5$).

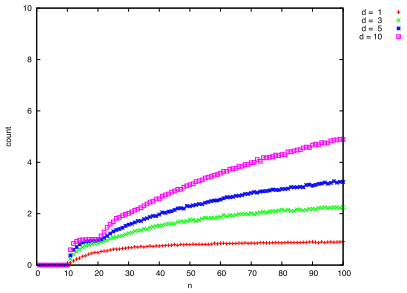


図 11　シミュレーション結果（$k = 10$）
**Fig. 11**　Simulation results ($k = 10$).

結果から AC ノード数 $n$ を増加させていくと，$d$ に漸近的に近づいていることが分かる．ただし図 8（$k = 1$）から図 11（$k = 10$）への変化からも分かるように，CRL のデータ冗長数 $k$ を増やすことで，計算量の増加を抑えることが可能である．

CRL のデータ冗長数 $k$ を増やすことについていえば，ある程度の規模までは $k = n$ つまりすべての AC ノードで全証明書分の CRL を保存することは妥当と考える．なぜならば，CRL に登録される証明書は通常ごくわずかだからである．証明書を無効化するのは，悪意あるユーザが関与した権限委譲をすべて無効とするような場合のみで，より頻度の高い権限変更や認証情報の更新は権限証明書の更新，つまり CUL で実現可能だからである．

### 4.4.3　プロトタイプによる実測

シミュレーション結果を検証する目的で，我々は Apache 上で動作する PHP スクリプトとして提案機構のプロトタイプ yassac を実装した．対象となる分散ファイルシステム

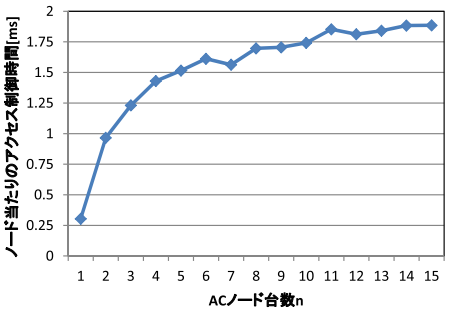


図 12　プロトタイプでの実測結果（$d = 1$）
**Fig. 12**　Measurement results with the prototype ($d = 1$).

としては，単一 PHP スクリプトとして動作する分散ファイルシステム yass [15] を利用した．yass を利用した理由としては，コードセットが非常にコンパクトで拡張が容易であり，Apache+PHP 環境を利用することで記述するコード量を抑えられるからである．プロトタイプでは，yass と yassac で別の計算機を利用することはせず，同一の計算機上に AC ノードと yass のノードが動作する形でシステムを構成した．

評価実験では，Amazon EC2 上の 20 台の仮想インスタンスを利用した．インスタンスタイプはマイクロインスタンス（最大仮想 2 コア，613 MB のメモリを搭載）で，OS には 64 ビット版の CentOS 5.5 を利用した．Apache は 2.2.3（OpenSSL 0.9.8e）を，PHP は 5.1.6 をそれぞれ利用した．全インスタンスはネットワーク遅延を抑えるために，同一リージョン（US East）の同一ゾーン（us-east-1d）上で実行した．なお，ノード間の通信遅延は平均 1 [ms] 程度であった．

20 台のノードのうち 5 台をクライアントに割り当て，1 台から 15 台を AC ノード（兼 yass のノード）として動作させた場合の性能測定を行った．ベンチマークとしては，AC ノードをランダムに選択し，ディレクトリを 1 つ作成するという処理とした．計測対象はノード側でアクセス制御にかかった時間，つまり権限証明書の検証および認証と権限のチェックにかかった時間を測定した．また，計測時間はクライアントごとに 500 個のディレクトリを作成させて，1 回あたりのアクセス制御にかかる時間の平均をとった．結果は図 **12** のようになった．同条件のシミュレーション結果（図 **13**）と比較するとシミュレーション結果は実測値を良く相似しており，妥当であるといえる．

以上の通信コスト，シミュレーション結果，プロトタイプの実測値，いずれからも提案機構は，AC ノード数の増加による通信コストの増加は限定的であり，性能がスケールアウトするといえる．

#### 4.4.4　証明書検証時の通信コストのゼロ化

ここまでで，シミュレーション結果やプロトタイプ実装による実測結果をもとに，提案機構の証明書検証時の通信コストの増大はノード数の増加に対して緩慢であることを

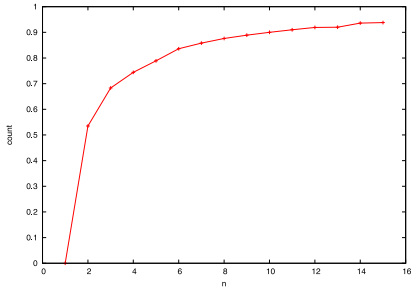


図 13　シミュレーション結果（比較用）
**Fig. 13**　Simulation results ($d = 1$).

示した．以下では，ルート証明書を無効化する際の通信コストとのトレードオフと，ユーザと応対する AC ノードの割り振りに一定のルールを課することで，証明書の通信回数をゼロに抑えることが可能であることを示す．

まず，3.3 節で示した証明書からハッシュ値を求める方法として方法 b を採用し，かつ $l = 2$ とする．これにより，CRL の参照時に発生する通信はたかだか 2 回（2 番目の証明書とルート証明書の CRL チェック）となる．次に，ルート証明書に限って冗長数 $k = n$ とする．つまり，全 AC ノードにすべてのルート証明書の CRL が割り当てられることになる．これにより，ルート証明書の無効化時には $n - 1$ 回の通信が発生する代わりに，すべての証明書がたかだか 1 回（2 番目の証明書の CRL チェックのみ）の通信で検証できるようになる．最後に，ユーザがリクエストを送る際に，ロードバランサやクライアントプログラムが利用する証明書から 2 番目の証明書の CRL を担当する AC ノードを割り振るようにする．以上により，他の AC ノードとの通信を発生させることなく，すべての証明書を検証できる．

ルート証明書の CRL をミラーリングしているので，ルート証明書の無効化処理はスケールアウトしないが，その無効化処理が実際に起こる頻度の低さを考えると意味のあるトレードオフだと考える．また，方法 b の $l = 2$ でハッシュ値を求めるため，分散の偏りが大きくなると予想される．しかし，大量の証明書が存在する状況を想定すれば，最終的にはハッシュ関数の性質により一様分布に近づくため，大きな問題にはならないと考える．逆に証明書の量が少ない状況を想定すれば，CRL や CUL の記憶容量やアクセス負荷は十分に小さく，問題とならないと考える．

以上で述べた工夫により，証明書の検証時の通信コストをゼロにするという，有用な効果を得ることができる．実際，提案機構をベースにしたアクセス制御機構を採用している実システム[*30]では，ここで述べた工夫が実施されている．

[*30] 数万人のユーザをかかえる企業向けオンラインストレージサービスのシステム．

表 3　関連研究との比較結果
**Table 3**　Comparison result.

| | Unix ACL | GSFS | DRBAC | DisCFS | 提案機構 |
|---|---|---|---|---|---|
| 登録設定コストの分散 | × | △ | △ | ○ | ○ |
| パスワード認証の利用 | ○ | ○ | ○ | × | ○ |
| 自律的な認証情報更新 | ○ | ○ | ○ | × | ○ |
| 副作用のない権限変更 | ○ | ○ | ○ | × | ○ |
| 単一障害点を持たない | × | × | – | ○ | ○ |
| 性能がスケールアウト | × | × | – | ○ | ○ |

## 5. 関連研究

本章では関連研究について述べる．また，章の最後で比較結果を**表 3** にまとめる．

伝統的な UNIX 系 OS に実装されている ACL によるアクセス制御機構は，分散の仕組みを持たないため，単一障害点を持ち，性能もスケールアウトしない．また，管理者やリソースの所有者でないと権限設定が行えず，管理運用コストが集中する．一方で，パスワード認証は利用可能であり，自律的な認証情報の更新も可能である．また，権限の変更による副作用も発生しない．

GSFS [3] は ACL を拡張して権限の変更自体を権限化することで権限委譲を実現したシステムである．しかし，GSFS の権限委譲は推移性と粒度調整性を同時に満たすことはできていない．また，可用性とスケーラビリティについては旧来の ACL 実装と同様の問題をかかえている．

Crampton らの提案（DRBAC）[10] では，ロールベースのアクセス制御で権限委譲を実現している．これにより，権限設定の管理運用コストが集中することは避けられる．しかしながら，依然として事前に機構側へのユーザ登録（やロール設定）が必要であり，そのコストは管理者等に集中すると考えられる．また，分散化に関する記述はなく，実装によると考えられる．加えて，分散化した場合を考えても，機構側で管理する情報が多いため，スケーラビリティの面で不利になると考えられる．

DisCFS [4] は本提案機構と同様に権限証明書を用いた分散型のアクセス制御機構である．機構側へのユーザの登録が不要であり，権限委譲により権限設定のコストも分散する．単一障害点がなく性能がスケールアウトすると考えられる．問題点としては，利用できる認証方式が公開鍵認証のみであり，自律的な認証情報の更新は行えない．また，権限変更には委譲されている証明書すべての無効化という副作用が発生する．

## 6. おわりに

本稿では分散ファイルシステムに適用する観点から，単一障害点を持たず，性能がスケールアウトするアクセス制御機構を提案した．可用性やスケーラビリティの面で優れる分散管理型のアクセス制御機構も，既存研究では実用上の問題点（公開鍵認証以外が利用できない，自律的な認証情報が更新できない，権限変更に副作用がともなう）をかかえていた．

そこで我々は，提案機構で，新たに公開認証情報を用いた認証，ノード秘密鍵による機構による署名，証明書の無効化と更新の区別を盛り込むことで，これらの問題を解決した．また，提案機構で CRL および CUL を分散管理する方法についても設計を示し，高い可用性とスケーラビリティの実現方法を述べた．そして，提案機構のシミュレーションプログラムおよびプロトタイプを実装して評価を行い，ノード数の増加によって提案機構の性能が向上すること（スケールアウトすること）を確認した．

提案機構はすでに実世界のシステムで採用され始めているが，今後は実運用から得られたデータ等をもとに分析・改良を行う予定である．

### 参考文献


[1] 荒川淳平，笹田耕一，竹内郁雄：実用的な分散アクセス制御機構，情報処理学会コンピュータシステム・シンポジウム論文集（ComSys 2010），Vol.2010, No.13, pp.47–56 (2010).

[2] Miltchev, S., Smith, M.S., Prevelakis, V., Keromytis, A. and Ioannidis, S.: Decentralized Access Control in Distributed File System, *ACM Computing Serveys*, Vol.40, No.3, Article 10 (2008).

[3] Kaminsky, M., Savvides, G., Mazieres, D. and Kaashoek, M.F.: Decentralized user authentication in a global file system, *Proc. 19th ACM Symposium on Operating Systems Principles* (*SOSP*), pp.60–73 (2003).

[4] Miltchev, S., Prevelakis, V., Ioannidis, J., Keromytis, A. and Smith, J.: Secure and flexible global file sharing, *Proc. Annual USENIX Technical Conference*, Freenix Track, pp.165–178 (2003).

[5] Ioannidis, J., Ioannidis, S., Keromytis, A.D. and Prevelakis, V.: Fileteller: Paying and getting paid for file storage, *Proc. 6th International Conference on Financial Cryptography* (*FC'02*), pp.282–299, Springer-Verlag (2002).

[6] Levine, A., Prevelakis, V., Ioanndis, J., Ioannidis, S. and Keromytis, A.D.: Webdava: An administrator-free approach to web file-sharing, *Proc. IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (*WETIC*), *Workshop on Distributed and Mobile Collaboration*, pp.59–64 (2003).

[7] Leung, W.A., Miller, L.E. and Jones, S.: Scalable Security for Petascale Parallel File Systems, *Proc. ACM/IEEE Conference on Supercomputing*, Article No.16 (2007).
[8] Regan, J. and Jensen, C.: Capability file names: Separating authorization from user management in an Internet file system, *Proc. USENIX Security Symposium*, pp.211–233 (2001).
[9] Petersen, K., Spreitzer, M., Terry, D. and Theimer, M.: Bayou: Replicated database services for world-wide applications, *Proc. 7th Workshop on ACM SIGOPS European Workshop* (1996).
[10] Crampton, J. and Khambhammettu, H.: Delegation in role-based access control, *International Journal of Information Security*, Vol.7, No.2, pp.123–136 (2008).
[11] Ghemawat, S., Gobioff, H. and Leung, S.-T.: The Google File System, *19th ACM Symposium on Operating Systems Principles* (2003).
[12] Tatebe, O., Hiraga, K. and Soda, N.: Gfarm Grid File System, *New Generation Computing*, Vol.28, No.3, pp.257–275, Ohmsha, Ltd. and Springer (2010).
[13] Weil, S., Brandt, S.A., Miller, E.L., Long, D.D.E. and Maltzahn, C.: Ceph: A Scalable, High-Performance Distributed File System, *Proc. 7th Conference on Operating Systems Design and Implementation* (*OSDI*) (2006).
[14] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, *ACM SIGCOMM 2001*, pp.149–160 (2001).
[15] 荒川淳平，笹田耕一，竹内郁雄：yass: Yet another simple storage，情報処理学会研究報告システムソフトウェアとオペレーティング・システム，Vol.2010-OS-113, No.11 (2010).

**荒川 淳平**　（正会員）

2012 年東京大学大学院情報理工学系研究科創造情報学博士課程を修了．博士（情報理工学）．2005 年株式会社インフォクラフトを設立．現在，同社代表取締役．ソフトウェアデザイン，システムソフトウェア，セキュリティに関する研究に興味を持つ．

**笹田 耕一**　（正会員）

2004 年東京農工大学大学院工学研究科博士前期課程情報コミュニケーション工学専攻修了．2006 年同大学院工学教育部博士後期課程電子情報工学専攻退学．博士（情報理工学）（東京大学情報理工学系研究科 2007 年）．2006 年東京大学情報理工学系研究科助手，2008 年同講師，2012 年 Heroku, Inc. にて Ruby 処理系の開発に従事（現職）．システムソフトウェア，特に並列処理システム，言語処理系に関する研究に興味を持つ．