情報処理学会論文誌 Vol.54 No.6 1807–1822 (June 2013)

# クラウドファイルバックアップシステムにおける複数ポリシに基づいた削除保証手法

手塚 伸[1,a)] 宇田 隆哉[2,b)] 岡田 謙一[1,c)]

受付日 2012年7月29日, 採録日 2013年3月1日

**概要**：日々増加する情報のバックアップ先として，クラウドストレージが注目を集めている．しかし，セキュリティ面の懸念も多いのが現状である．その中でも我々は，不要になった情報の完全な削除をクラウドベンダが保証しないという問題に着目した．既存研究にも，クラウドストレージを利用したファイルの増分バックアップと，削除保証の両立を目的としたものがある．しかし，時間やユーザ，グループといった複数のポリシを組み合わせた，柔軟な復元可能条件の設定はできない．そこで我々は，ファイルの増分バックアップと複数ポリシに基づいた削除保証を実現する “Stratus” を提案する．この特徴は，削除保証を実現するための暗号鍵をハッシュ連鎖と $(k, n)$ 閾値秘密分散法から生成する点である．これにより，たとえば「5 年以上前にバックアップされた版の一括削除」や「共有ファイルに対して，ユーザ全員が権限を失ったときに削除」のように，ファイルの性質に応じたポリシの設定が可能になる．評価結果より，既存のシステムと比較した際のオーバヘッドは，初回フルバックアップで 2.9%，日々の増分バックアップで 1.1%，リストアで 11.4%であった．よって，日常的に行われる増分バックアップのパフォーマンスに対する本手法の影響はわずかであることから，Stratus は有用であると考える．

**キーワード**：削除保証，増分バックアップ，バージョン管理，クラウドストレージ

# A Method of Multiple Policy-based Assured Deletion for Cloud File Backup System

Shin Tezuka[1,a)] Ryuya Uda[2,b)] Ken-ichi Okada[1,c)]

Received: July 29, 2012, Accepted: March 1, 2013

***Abstract:*** Although cloud storage has attracted attention as a location to backup daily increasing information, it also raises various security concerns. In particular, we focused on the problem that complete deletion of unnecessary information is not guaranteed by cloud vendors. Some of the existing studies aim to achieve both incremental file backup using cloud storage and assured deletion. However, they do not enable users to delete files with flexible restore conditions based on multiple policies such as time, user and group. Therefore, we propose “Stratus”, which incrementally backs up files to cloud storage and realizes assured deletion based on multiple policies. The concept of Stratus is to generate an encryption key to realize assured deletion from hash chain and $(k, n)$ threshold secret sharing scheme. This would allow the suit policies based on file types, for instance, such as “batch deleting backed up versions older than 5-year” or “deleting shared files when all users have lost the privileges”. From the result of our evaluation, we have verified that the overhead of full backup for the first time is 2.9%, daily incremental backup is 1.1% and restore is 11.4% compared to the existing system. Therefore, Stratus can be useful since our method requires the minimal performance overhead for everyday incremental backup operations.

***Keywords:*** assured deletion, incremental backup, version control, cloud storage

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University, Yokohama, Kanagawa 223–8522, Japan
[2] 東京工科大学コンピュータサイエンス学部
School of Computer Science, Tokyo University of Technology, Hachioji, Tokyo 192–0982, Japan
a) tezuka@mos.ics.keio.ac.jp
b) uda@cs.teu.ac.jp
c) okada@ics.keio.ac.jp

## 1. はじめに

電子カルテや企業間の取引記録など，センシティブな情報が電子的に扱われている．しかし，近年では日々増加するそれらの情報に対して，改変記録とともに一定の期間保持することが法律により求められており，その管理が課題となっている [1].

他方，次世代の情報基盤として，クラウドコンピューティングが注目を集めている [2]．なかでも，Amazon の Simple Storage Service（S3）[3] に代表されるクラウドストレージは，任意の情報を格納できる，仮想的に無限な領域を提供するサービスである．これにより，個人や企業のユーザが自前で行っていた情報の管理を，従量制でアウトソースできるというメリットを提供している．

これに対して，情報の変更記録をバージョン管理し，クラウドストレージをバックエンドとして増分バックアップを行う研究がなされている [4], [5]．この増分バックアップの手法は，ファイルの各版を単純にフルバックアップする手法と比較して，各版の間で共通する部分を 1 つに集約する．よって，バックアップデータの量が削減されるため，格納されたデータの容量に応じて従量制で料金が課金されるクラウドストレージに対してコスト面でも有益である．しかし，クラウドストレージにはセキュリティに関する懸念も多く存在するのが現状である．なかでも，我々は「情報の秘匿」と「削除保証」という問題に着目した．

クラウドストレージでは，サービスを提供するクラウドベンダが，どのように格納された情報を扱っているかをユーザは知ることができない．ゆえに，その管理体制や仕組みはブラックボックスである．さらに，クラウドベンダ側は格納された情報へ自由にアクセスできるため，意図的であれ，偶発的であれ，情報の流出や改竄に加担する可能性を否定できない．このリスクは，情報がクラウドストレージに保存されている期間が長いほど高まるため，特に不要になった情報は速やかに削除される必要がある．

このような削除に関する問題は，特に行政機関の監査や裁判所の命令により，電子的な情報に対する調査が実施される際に顕在化する．具体的には，調査者や調査に協力するクラウドストレージの管理者は，各国の法律に基づき，ファイルの所持者に対して暗号化に使用した鍵の提出を求めることができる．たとえば，英国の調査権限規制法（Regulation of Investigatory Powers Act：RIPA）などがこれにあたる．つまり，調査者に対する内容の秘匿を目的としたファイルの暗号化は意味をなさなくなる．当然ながら，法令に基づいた調査に対し，各組織は保存義務期間内のファイルについて，その内容を公開すべきである．しかし，前述のとおりクラウドストレージの管理者は，クラウドストレージに保存されたすべてのデータに対してアクセスできる．そのため，正規の調査に加え，ファイルの暗号鍵を得たクラウドストレージの管理者によって，調査範囲を超えて保存義務期間を過ぎたファイルが不正に復号され，情報漏洩につながるという問題がある [6]．よって，保存義務期間を超えた情報は完全に削除され，ファイルの所持者も含めて何人にも復元されないよう，保証される仕組みが必要である．

他方，クラウドストレージを含めた，一般的なファイルシステムにおける「ファイル削除」は，高速化のため実データの完全な削除を行わない．一方，完全なファイルデータの削除には，データが記録された領域に対して，複数回上書きするといった手法が必要となる．しかし，高いレベルで抽象化されたクラウドストレージでは，物理デバイスをユーザが直接操作できないため，これを実行する術はない．これに加えて，クラウドベンダも「完全なファイルの削除」を保証しないという問題がある．

これに対し，クラウドストレージにおける情報の削除保証では，暗号化方式を用いたものが提案されている．前述の上書き方式とは異なり，暗号化方式では対象のファイルを事前に暗号化してから媒体に格納し，削除時に暗号鍵を破棄することでファイル全体の完全な削除を保証する方式である．つまり，記憶デバイスを直接操作できない環境でも用いることができる．

しかし，変更記録をともなう増分バックアップされたファイルは，各版間に共通するデータの実体は 1 つに集約される．そのため，単純にある版のファイルを削除すると，データの一部を共有している他の版も復元できなくなるという，技術的な問題が生じる．よって，一定の保持期間が定められたセンシティブなファイルについて「クラウドストレージにおけるバージョン管理と完全な削除の保証を両立させる手法」の開発は大変重要な課題となる．

この課題を対象とした暗号化方式による削除保証の研究として，たとえば Rahumed らは，クラウドストレージを利用した，ファイルの増分バックアップを実現する FadeVersion [6] を提案している．この特徴は，Layered Encryption とよばれる 2 種類の暗号鍵を用いた手法により，バージョン管理と削除保証を両立させている点，およびファイルに対して「復元可能条件」を設定できる点である．

復元可能条件とは，ユーザやグループに割り当てられたポリシと AND/OR の論理演算により，ファイルへのアクセス権を定めたものである．そして，ファイルの削除とその保証は，この条件を満たさなくなった際に実行される仕組みとなる．なお，この復元可能条件では，ポリシ A およびポリシ B がともに有効な場合は，ファイルを復元可能という条件を「AND 条件」とし，ポリシ A またはポリシ B が有効な場合はファイルを復元可能という条件を「OR 条件」とする．これにより，たとえば複数のユーザやグループによって共有されているファイルに対して，「ユーザ A が在籍中，かつプロジェクト B が進行中」や「ユーザ A ま

たはユーザ B が在籍中」という条件を満たさなくなった際に，関連するファイルの削除が実行され，誰も復元できないことが保証されるように設定できる．

ただし，FadeVersion は AND 条件のみの復元可能条件は設定できるが，OR 条件はサポートされていない．他方，FadeVersion はユーザ，グループのポリシに加え，時間ベースのポリシをファイルに対して設定することで，時限式でファイルの削除を保証できる．しかし，バージョン管理されたファイルについて，削除が保証されるタイミングを示す時間ポリシの割当てが，すべての版で共通となる．そのため，特に追記型のファイルに対して任意の版以前を削除し，それを保証するようなことはできない．つまり，たとえば「直近の状態と変更記録を残して，指定された版より古い版を順次削除する」のような条件は設定できない．他方，ファイルのすべての版に対して別々の暗号鍵を割り当てることで，これを実現する方法も考えられる．しかし，それはファイルの増分に対してセキュアに管理すべき暗号鍵の増大を招くことになる．よって，ファイルの性質に応じて，ポリシどうしの OR 条件を含み，かつ任意の版以前の削除を実現するような，柔軟なポリシを設定できる手法が必要である．

そこで我々は，クラウドストレージへバックアップされるファイルの特性に応じて，複数のポリシに基づいた削除保証を実現する “Stratus” を提案する．Stratus の特徴は，削除保証を実現するための暗号鍵をハッシュ連鎖と $(k, n)$ 閾値秘密分散法から生成する点である．これにより，たとえば「特定のファイルについて，5 年以上経過した版の削除」や「共有ファイルに対して，ユーザ全員が権限を失ったときに削除」といった操作が可能になる．

以下，本稿は次のように構成される．まず，2 章では削除保証とバージョン管理についての関連研究を紹介し，本稿の位置付けを示す．3 章では本研究が想定する環境とベースとなるファイルバックアップシステムについて述べ，対象とする課題を整理する．次に 4 章ではファイルの性質に応じた復元可能条件の設定を実現する手法の詳細を示す．さらに 5 章では提案手法を組み込んだ Stratus の実装について解説する．6 章では提案手法の有効性を確認するため，評価とその結果について考察し，7 章で本稿のまとめとする．

## 2. 関連研究

ファイルの完全な削除を保証する戦略には，大別して「上書き方式」と「暗号化方式」がある．上書き方式は，磁気で情報を記録する媒体に対して有効な方式であり，Gutmann method [7] や DoD 5220.22-M [8] がある．これらは，削除対象となるファイルが記録された領域に対し，特定のパターンやランダムなビット列で複数回上書きすることで，残留磁気を完全に消去する．しかし，記憶デバイスに対する操作をユーザが直接行うことはできないため，クラウドストレージへは適用できない．

他方，暗号化方式は保存されるファイルをある暗号鍵で事前に暗号化し，削除が要求された際は鍵のみを破棄することで，ファイル全体の削除保証を実現する手法である．暗号化方式を利用したものには，たとえば ext3 ファイルシステムに対して，時刻を巻き戻してアクセスする機能や，ある時点以前の全ブロックの削除保証を 1 つの操作で実現した Ext3cow [9] がある．また Perlman は，ファイルへ設定された有効期限に達した時点で，鍵マネージャから暗号鍵を削除する，時限式の手法 [10] を提案した．

クラウドストレージ上へバックアップされたファイルの削除保証に特化した研究として，Tang らの FADE [11] があげられる．FADE はバックアップ対象のファイルを，ファイルサーバ上でデータ鍵とよばれる暗号鍵により暗号化し，クラウドストレージへアップロードする．そして，データ鍵は制御鍵により暗号化される．なお，制御鍵はファイルサーバを運用する組織により，Key-manager 上で管理され，ファイルの復元可能条件を定めたポリシと結び付いている．ポリシが無効になった際は，対応する制御鍵を Key-manager から破棄することで，削除保証が実現される．また，FADE では復元可能条件を定めたポリシどうしに，AND/OR 条件を付加する手法もあわせて実装されている．たとえばあるファイルが複数のグループに共有されている場合，AND 条件の実現には複数ポリシに対応する制御鍵で多重にデータ鍵を暗号化する．また，OR 条件はそれぞれのグループに対応する制御鍵でデータ鍵を個別に暗号化することで実現される．ただし，FADE は単一版の扱いを想定しているため，ファイルの変更記録をバージョン管理，あるいは増分バックアップするようなシステムを対象としていない．そのため本研究が対象とするような，ファイルが複数の版にわたってバックアップされるべき環境において，一部の版に対する削除とその削除を保証する仕組みは実現されていない．

また，FADE では，制御鍵の暗号化に公開鍵暗号方式の Rivest-Shamir-Adleman algorithm（RSA）[12] を用いるが，これに対して田中らは ElGamal 暗号 [13] を利用し，高速化を図っている [14]．

他方，我々はバージョン管理と削除保証を実現する，ADEC [15] を提案した．他の研究とは異なり，ADEC の特徴はクラウドストレージをバックエンドとする仮想ファイルシステムとして動作する点である．さらに，ファイルの変更記録の不正な入れ替えを防ぐため，各ファイルの版間で連鎖的に電子署名を生成する手法を導入している．ただし，複数ポリシに基づく復元可能条件の設定は考慮されていない．

既存のファイルサーバに保存されたファイルをクラウドストレージ上へバックアップし，削除保証とバージョン管

理機能を両立させた研究として，Rahumed らの FadeVersion [6] があげられる．これは，Vrable らが提案した増分バックアップシステムの Cumulus [4], [5] をベースに，削除保証の機能を加えたものである．暗号化方式による削除保証とバージョン管理の両立を考えた場合，ある版のデータを削除すると，そのデータを共有している他の版も復元できなくなるという問題がある．これに対して，FadeVersion は Cumulus に由来する重複除去機構に FADE の手法を取り入れることで，これを解決している．また，ポリシに対応する制御鍵でデータ鍵を暗号化することで，ポリシどうしの AND 条件の設定や，ファイルが削除されずに復元できる有効期限の設定をサポートする．

しかし，ポリシの割当てがファイル単位であるため，ポリシに基づいた削除を実行した際は，そのファイルのすべての版が失われる．つまり，一定の期間が経過した任意の古い版から順次削除するような操作はできない．ファイルのすべての版に対して別々の暗号鍵を割り当てることで，これを実現する方法も考えられるが，それはファイルの増分に対してセキュアに管理すべき暗号鍵の増大を招くことになる．また，複数人でファイルを共有するような環境においては，「いずれかのユーザがアクセス権を持っている間はファイルを削除しない」といった，OR 条件を含む復元可能条件をポリシどうしに設定できる必要性がある．しかし，FadeVersion が対応しているのは AND 条件のみであり，いずれかのポリシが破棄された時点で削除が実行される．

また，文献内 [6] では言及されていないが，FADE のように複数の制御鍵でデータ鍵を個別に暗号化することで，OR 条件を実装できると考えられる．しかしこの方法では，あるグループに対応するポリシを破棄すると，そのグループが利用していたすべてのファイルの全版が削除されてしまう．したがって，本研究が対象とする保存義務期間を超えたファイルの版から順次削除するという目的には合致しない．

これに対して本研究で開発された Stratus は，ポリシに対応した複数の制御鍵でデータ鍵を暗号化する FadeVersion とは異なり，ポリシ鍵から制御鍵を生成する新たな手法を導入している．これはハッシュ関数の不可逆性を利用した連鎖鍵，および廣田らにより提案された「$(k, n)$ 閾値秘密分散法 [16] により，複数の異なる暗号鍵から同じ復号結果を得る手法」[17] を用いて実現される．これにより，既存研究の FadeVersion ではサポートされていない，復元可能条件が複数のポリシによる AND/OR 条件で設定されているファイルに関して，任意の時点より前の版を削除し，その削除を保証することが可能となる．

## 3. クラウド型ファイルバックアップシステム

本研究では，既存のファイルサーバに格納されているファイルを，クラウドストレージへバックアップし，クラウドベンダに対するバックアップデータの秘匿，バージョン管理，削除保証を実現する “Stratus” を開発した．本章では，想定する環境，および 4 章で解説する手法のベースシステムについて述べる．

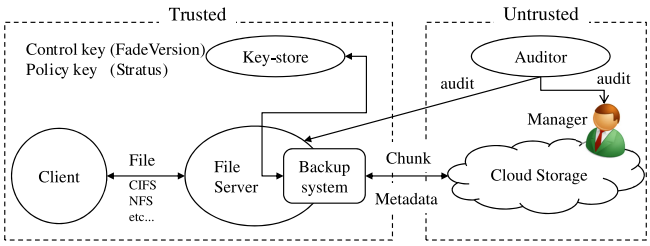


図 1 想定環境
**Fig. 1** Assumed environment.

### 3.1 想定環境

本研究の対象は，**図 1** のようにオンプレミスな環境で運用されているファイルサーバのコンテンツを，クラウドストレージへ定期的にバックアップすることを計画している組織である．なお，本研究に関連するエンティティは，ユーザへのサービスを行う「クライアント」，ファイルの格納とバックアップ動作の主体となる「ファイルサーバ」，ファイルのバックアップ先となる「クラウドストレージ」，暗号鍵を管理する「Key-store」，後述の調査を実施する「調査者（図中 Auditor）」，およびクラウドストレージを管理する「管理者（図中 Manager）」である．対象のファイルコンテンツは，特に法律や組織の内規により保存義務期間が定められており，機密情報を含むセンシティブなものを想定している．なお，具体的には日々増加する医療カルテや企業間の取引記録など，秘匿と完全な削除が求められるものがこれに該当する．

各組織はこのようなファイルについて，行政機関の監査や裁判所による調査に備えて，保存義務が発生する期間中は最新の版のみではなく，すべての版に対する変更記録を保持することが求められる．しかし，クラウドストレージを利用した増分バックアップシステムでは「行政機関の監査や裁判所の命令による電子データの調査時に，組織にとって第三者となる調査者とクラウドベンダの管理者が協力し，ユーザによってすでに削除された対象外の古いファイルの復元を調査の範囲を超えて不正に試みる」という攻撃が考えられる．これは，調査者や調査に協力するクラウドストレージの管理者はクラウド上のすべてのデータにアクセスできるが，それは調査対象ではないデータも容易に取得できることを意味している．さらに調査者は，調査対象の組織に対して，1 章で述べたように，強制的にファイルを復号するための暗号鍵の提出を求めることができる．つまり，たとえクラウドストレージへバックアップされた

ファイルが暗号化された状態であっても，提出された鍵によって復号される可能性がある．よって，クラウドストレージへバックアップされたファイルの版のうち，保存義務期間を過ぎて不要になったものは，漏洩防止のために順時削除し，その削除が完全になされたことを保証する仕組みが求められる．

以上のような環境において，本研究では保存義務期間が定められたファイルを対象に，削除保証とバージョン管理を両立させた増分バックアップシステムを作成する．さらに，先行研究の FadeVersion では不可能であった「指定された任意の版以前の削除保証」，および「ポリシどうしの OR 条件の設定」の実現を目的とする．基本的な仕組みは FadeVersion の手法を踏襲し，最終的なファイルの秘匿性と削除保証は図 1 中の Key-store に格納された，次節で述べる暗号鍵に依存する．また，Key-store の実装は本研究の対象としないが，4.4 節で述べるファイルサーバ内と同じ筐体内のデバイスや外部のサーバを利用する方式が考えられる．

ただし，ユーザへの直接的なサービスは対象外とし，ファイルサーバに格納されたファイルへアクセスする際は，適切な権限管理の下で NFS や CIFS，Samba など，既存のネットワークファイルシステムの利用を前提とする．また，「削除保証」とは，一般的な削除操作（完全なファイル内容の削除を保証しない）に加えて，第三者による検証時や不正な復元を試みた場合でも，元の内容を得られないことを指す．

また「柔軟な復元可能条件」とは，ファイルに対して「ユーザ」，「グループ」に準じたポリシに加え，さらに「時間（増分バックアップの版数）」に関するポリシを用い，それらを AND/OR 演算で組み合わせて復元可能条件が設定できることを意図する．つまり，この復元可能条件が満たされなくなったときに，削除保証が実施されることになる．また，本来の論理演算には AND/OR に加えて NOT が必要となる．しかし，ファイルに対するアクセス権を失い，復元可能条件を満たさなくなることは，次章で述べる「ポリシ鍵の破棄」を意味するため，本稿の提案範囲では NOT 演算は含めないものとする．

なお，組織が開示義務を負う保存義務期間内のファイルに対して，正規に開示を求められた場合は，その要求に従うことを想定している．また，攻撃者となりうるのは前述のとおり，第三者となる調査者とクラウドストレージの管理者（図 1 中破線枠の Untrusted）であるが，調査時においてファイルサーバを管理する組織側に背任者はいないことを前提とする．そのため，「調査者に対して意図的に調査範囲外のファイルや暗号鍵を流出させることはない」という点において，クライアント，ファイルサーバの管理者，および Key-store（図 1 中破線枠の Trusted）は信頼できるものとする．さらに，通信経路上やファイルサーバに対する不正侵入などの攻撃は，Secure Socket Layer（SSL）やファイアウォール，不正侵入検知システムなど，他の技術で十分に保護可能であり，本研究では考慮しない．

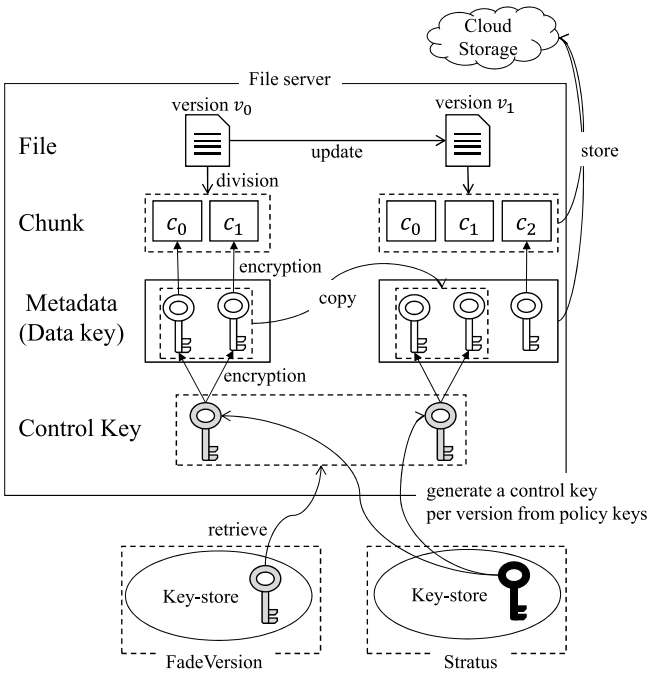


図 2 ファイル分割と Layered Encryption
**Fig. 2** File division and Layered Encryption.

本研究が想定する調査は，監査による定期的なものと，インシデント発生時など不定期に行われるものに大別される．定期的な監査は，たとえば「1 年に 1 度」や，より重要な情報を扱う環境であれば「1 カ月に 1 度」など，バックアップされたファイルに対する検証作業を示す．不定期な調査は，問題が発生した際に「ある特定のユーザやグループに関連した一定期間のファイル」に対して行われるものであり，「数年に 1 度」といった，より低い頻度の検証作業を示す．ファイルサーバの管理者は調査者の要求に対して，対象となるファイルを復元できるよう，次節で述べるポリシ鍵を開示することで，その要求に対応する．

### 3.2 ベースシステムと課題

本研究で開発された Stratus は，FadeVersion と同様に Layered Encryption の考え方に基づいた手法を用いる．

**図 2** は FadeVersion と Stratus におけるファイルバックアップ時の動作を模式的に表している．図中の実線で囲まれた “File Server” の枠は，ファイルサーバの内部で行われる処理を示し，FadeVersion と Stratus に共通する部分である．なお，図下段の破線で囲まれた 2 つの “Key-store” は，FadeVersion と Stratus で Key-store に格納すべき暗号鍵が異なることを示す．

初版のバックアップは，まず図 2 上段で示すように $v_0$ のファイルをファイルサーバ上で複数の「Chunk」とよばれる固定長の断片，たとえば図中 $c_0$，$c_1$ のように分割する．

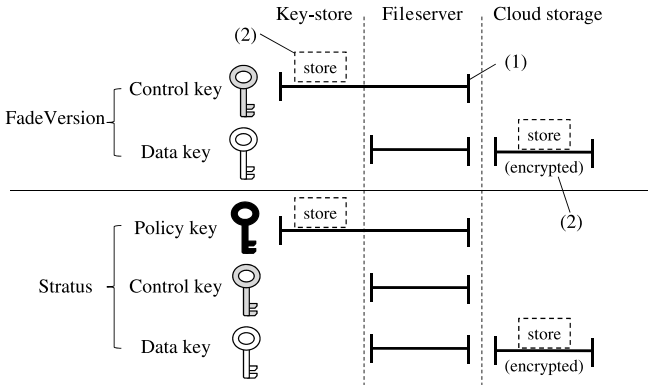


図 3 暗号鍵が扱われる範囲と格納先
**Fig. 3** Handling range and storage location of encryption keys.

この Chunk は，ファイルサーバ上で生成された，それぞれ異なるランダムな共通鍵の「データ鍵」(図中の白抜きの鍵）により暗号化される．さらに各データ鍵は，「制御鍵」(図中の灰色の鍵）とよばれる共通鍵によって暗号化される．最後に，各 Chunk のハッシュ値と暗号化されたデータ鍵は，ファイルごとかつ版ごとに生成される「Metadata」に記録され，暗号化された Chunk とともにクラウドストレージへ格納される．なお，ファイルを Chunk へ分割する際の方式は，FadeVersion と同じく Stratus でも 1 MB の固定長とし，本稿では議論の対象としない．

他方，FadeVersion における制御鍵はセキュアなデバイスやシステムによって構成された，鍵を確実に削除する機能を持つ「Key-store」上に格納される．なお，Stratus における制御鍵の扱いについては，ベースとなる FadeVersion での管理法に対して，次章で述べる手法によりポリシ鍵を導入することで改良を加える．

ここで，制御鍵，データ鍵，ポリシ鍵の各鍵がどのエンティティによって扱われるのかをまとめる．**図 3** の上段は本研究のベースとなる FadeVersion，下段は Stratus において，各暗号鍵が扱われる範囲と格納先を示したものである．図中 (1) の太実線は，その線の左側の鍵について，縦の破線で区切られた Key-store，ファイルサーバ，クラウドストレージのうち，どのエンティティが扱うことができるかという範囲を示している．また，図中 (2) の破線枠で囲まれた “store” は，その鍵がどのエンティティに格納されるのかを示している．

まず，FadeVersion における制御鍵（図中灰色の Control key）は Key-store 上に格納され，データ鍵を暗号化するためにファイルサーバが知ることができる．他方，Stratus における制御鍵は，前述のとおり次章で詳述する手法により，ファイルサーバ上で複数のポリシ鍵から生成され，使用後はいかなる記憶媒体にも永続化せずにメモリ上から破棄される．

次に，データ鍵（図中白抜きの Data key）は，ファイルサーバ上で制御鍵により暗号化され，Metadata に含まれた状態（図中 (3) の “encrypted” の箇所）でクラウドストレージへ格納される．よって，ファイルサーバはデータ鍵を生成し，Chunk を暗号化するためにこの鍵を扱うことができる．しかし，クラウドストレージの管理者は，Metadata から暗号化済みのデータ鍵を取得できたとしても，これを復号することはできない．

最後に，ポリシ鍵（図中黒色の Policy key）は Stratus で新たに追加されたポリシごとに割り当てられる鍵であり，ファイルサーバ上で制御鍵を生成するために用いられ，Key-store に格納される．

次に，更新されたファイルをバックアップする際の動作について述べる．図 2 中の更新されたファイル $v_1$ は，同様に Chunk へ分割されるが，変化があった Chunk のみ，たとえば図中 $c_2$ のように新たに生成されたデータ鍵で暗号化される．他方，$c_0$，$c_1$ のように，過去の版の中に同じ内容の Chunk が存在する場合は，過去の Metadata から対応するデータ鍵を複製し，新しい制御鍵で再度暗号化して Metadata へ記録する．これにより，複数バージョンで Chunk を共有する重複除去とバージョン管理を実現する．

過去の版の復元が要求された場合は，まず指定された版の Metadata から，ファイルを構成する Chunk のハッシュ値と対応するデータ鍵を取得する．次に，ファイルサーバ上で動作する FadeVersion により Key-store から取得，あるいは Stratus によりポリシ鍵から生成された制御鍵によって各データ鍵を復号する．最後に，クラウドストレージから回収された各 Chunk は，対応するデータ鍵によって復号され，元のファイルへ復元される流れとなる．

ファイルの削除保証については，FadeVersion では制御鍵を，Stratus ではポリシ鍵を Key-store から削除することで実現される．つまり，削除された制御鍵あるいはポリシ鍵から生成される制御鍵で暗号化されたデータ鍵は復元不可となり，結果的にファイル全体の削除が保証される．また平時において，たとえクラウドストレージの管理者が，暗号化済みの Chunk をクラウドストレージの磁気媒体などから取得できても，これを復号することはできず，元のファイルの閲覧も不可能となる．

このように，ファイルの復元には Chunk の復号が不可欠であり，そのためにはデータ鍵が必要となる．そして，FadeVersion におけるデータ鍵の復号には，Key-store に格納された制御鍵が必要であり，Stratus では Key-store に格納されたポリシ鍵から制御鍵を生成する必要がある．よって，調査時に FadeVersion における制御鍵，あるいは Stratus におけるポリシ鍵の提出が求められても，Key-store からこれらが適切に削除されていれば，調査者やクラウドストレージの管理者へこの鍵を渡すことは不可能であり，Key-store, ファイルサーバ，クラウドストレージ，調査者のどのエンティティであっても，暗号化されたデータ鍵を復号することはできない．したがって，削除されたものに

ついては，データ鍵を用いて Chunk を復号することができないため，平時/調査時ともクラウドストレージの管理者はファイルを復元することはできない．

以上のような仕組みをベースに，既存研究が有する以下の課題の解決を目指す．

- 指定された版より過去の版の一括削除
- 複数ポリシに基づいた，OR 条件を含む復元可能条件の設定

## 4. 連鎖鍵および複数ポリシに基づく制御鍵の生成手法

本章では，クラウドストレージ上にバックアップされたファイルの削除保証に関して，Stratus へ新たに導入された手法について述べる．

前章で述べた課題を解決するため，我々は FadeVersion における制御鍵に着目した．まず FadeVersion において，ある時点より過去の版に対する削除を保証する操作が難しい原因は，ファイルに関連付けられたポリシに対応する制御鍵が，すべての版で共有されているためである．つまり，版ごとに制御鍵を削除するといった細かい操作を行うことができない．ファイルの各版に対して 1 つ 1 つ別の有効期限を定めたポリシを割り当てることも可能であるが，それはポリシ数の増大を招き，管理を困難にする．そこで，我々はハッシュ関数を用いて，過去の鍵から連鎖的に，順次新しい制御鍵を生成する手法を提案する．

次に，複数のポリシに基づいた復元可能条件の構成法について考える．FadeVersion では，単純に複数のポリシに対応する制御鍵で，データ鍵を多重に暗号化する．これは，AND 条件のみが想定される環境では問題ないが，OR 条件を設定することはできない．そこで，$(k, n)$ 閾値秘密分散法 [16] を応用し，AND 条件および OR 条件で構成された複数のポリシから，版ごとに制御鍵を生成する手法を提案する．

なお，$(k, n)$ 閾値秘密分散法の本来の目的は，ある秘密情報を $n$ 個のシェアとよばれる情報の断片に分割し，元の秘密情報を秘匿することである．元の秘密情報は，$n$ 個のシェアのうち，$k$ 個以上を結合することで復元される．一方，$k$ 個未満のシェアからは情報量的安全性により，元の秘密情報に関していっさいの情報を得ることはできないという特徴がある．

以下の節では，2 つの提案手法の詳細，およびポリシ鍵の更新・管理法について述べる．

### 4.1 連鎖鍵による制御鍵の生成

Stratus によりバックアップされる各ファイルには，System time based Policy (SP) と File time based Policy (FP) が割り当てられる．SP は，たとえば「10 年」など，バックアップシステムの運用方針により，過去の版を保持する期間を定めた時間ベースのポリシである．つまり，どのようなファイルであっても，このポリシで定められた時間が経過した後は確実な削除が保証される．なお，SP はバックアップシステム全体で 1 つであり，全ファイルがこれを共有する．

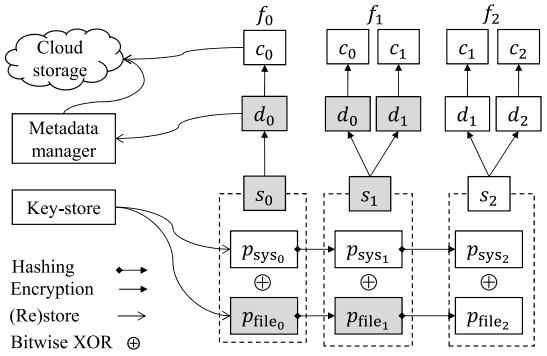


図 4 ポリシ鍵と鍵連鎖に基づく制御鍵の生成

**Fig. 4** Control key generation based on policy key and chained-Key.

他方，FP は SP と同様に時間ベースのポリシであるが，ファイルごとの最大保持期間を定める．なお，組織の方針によっては，これらのポリシで定める最大保持期間を無限に設定することも可能である．また，各ポリシには，ランダムな共通鍵の「ポリシ鍵」が割り当てられ，Key-store に格納される．

**図 4** は，あるファイル $f$ の初版から第 3 版までが Stratus によってバックアップされる様子を示している．図中上段の $c_i$ は分割されたファイルの $i$ 番目の Chunk を示し，$d_i$ は各 Chunk を暗号化するためのデータ鍵である．なお，初版のファイル $f_0$ は $c_0$，第 2 版 $f_1$ は $c_0$，$c_1$ の Chunk から構成される．また，第 3 版の $f_2$ 版は $c_0$ を含まず，$c_1$，$c_2$ からなる．なお，これはたとえばログファイルなど，追記型のファイルを想定している．下段の $p_{\mathrm{sys}_v}$ は，第 $v$ 版の SP に対応するポリシ鍵，$p_{\mathrm{file}_v}$ は FP のポリシ鍵を示す．また，中段の $s_v$ は，第 $v$ 版の制御鍵である．

ここで，初版 $v = 0$ のファイルがバックアップされる過程に着目する．初版のファイルに含まれる Chunk $c_0$ は，$d_0$ によって暗号化され，$d_0$ を暗号化するための制御鍵 $s_0$ は $p_{\mathrm{sys}_0} \oplus p_{\mathrm{file}_0}$ より生成される．なお，$x \oplus y$ は暗号鍵のビット列 $x$，$y$ に対して，ビットごとに排他的論理和を求めることを示す．つまり，Key-store に格納された $p_{\mathrm{sys}_0}$ と $p_{\mathrm{file}_0}$ の両方が揃わないとファイルは復元できず，いずれかのポリシ鍵が破棄された時点で，初版の完全な削除が保証される．なお，暗号化済みの $d_0$ は，Metadata の一部として記録される．

第 2 版以降のバックアップも，基本的なプロセスは初版の場合と同様である．ただし，制御鍵 $s_v$ の生成に必要なポリシ鍵は，それぞれ 1 つ前のポリシ鍵から，式 (1) のよ

うに連鎖的に求められる．なお，$\mathrm{H}(x)$ は $x$ のハッシュ値を生成する関数である．

$$s_v = \mathrm{H}(p_{\mathrm{sys}_{v-1}}) \oplus \mathrm{H}(p_{\mathrm{file}_{v-1}}) \quad (1)$$

次に，FP で定められた保存期間を過ぎ，初版 $f_0$ と第 2 版 $f_1$ の削除保証が要求された場合を考える．この場合，第 3 版未満の制御鍵 $s_0$，$s_1$ が回復できなくなるような処置を行えばよい．そこで，まず Key-store に格納された $p_{\mathrm{file}_0}$ から $\mathrm{H}(\mathrm{H}(p_{\mathrm{file}_0}))$ で $p_{\mathrm{file}_2}$ を生成し，メモリ上に保持する．次に，Key-store から $p_{\mathrm{file}_0}$ を破棄し，先に生成した $p_{\mathrm{file}_2}$ をファイル $f$ に対する FP のポリシ鍵として Key-store へ格納する．あるいは，Key-store の実装方式（4.4 節参照）によっては，2 章で述べた DoD 5220.22-M [8] を用い，$p_{\mathrm{file}_0}$ を破棄した後に $p_{\mathrm{file}_2}$ で更新する．これにより，図 4 中の灰色の背景で示された鍵が回復できなくなる．具体的には，まず $p_{\mathrm{file}_0}$ はもちろん，ハッシュ関数の不可逆性により $p_{\mathrm{file}_0}$ から連鎖的に導出される $p_{\mathrm{file}_1}$ も生成することができない．よって，制御鍵 $s_0$，$s_1$ も生成できず，最終的にファイル $f$ の初版および第 2 版の復元が不可能となる．

以上のように，削除対象とする時刻に応じて，特定の版番号未満のポリシ鍵を破棄することで，Stratus は時間ベースの削除保証を実現する．

### 4.2 複数ポリシによる復元可能条件の設定

前節のとおり，制御鍵は SP および FP から生成される．本節では，これに加えてユーザやグループといった，ファイルに対して任意に設定できる複数の Attribute based Policy（AP）を組み合わせることで，AND/OR 条件を含む復元可能条件を実現する手法について述べる．

まず，2 つの AP である $P_a$，$P_b$ の第 $v$ 版におけるポリシ鍵を，それぞれ $p_{a_v}$，$p_{b_v}$ とする．このとき「ポリシ A およびポリシ B がともに有効な場合はファイルを復元可能」という AND 条件を $P_a \wedge P_b$ とする．前節のとおり，この AND 条件は $p_{a_v} \oplus p_{b_v}$ から制御鍵 $s_v$ を生成すればよい．また，FP とあわせて $p_{\mathrm{file}_v} \wedge p_{a_v} \wedge p_{b_v}$ のような復元可能条件では，$t_0 = p_{a_v} \oplus p_{b_v}$ を求めた後に $s_v = p_{\mathrm{file}_v} \oplus t_0$ として，制御鍵を生成すればよい．なお，$t_y$ は制御鍵の生成過程で y 番目に導出される，ポリシ鍵と同じビット長の一時鍵を示す．

次に，「ポリシ A またはポリシ B が有効であればファイルを復元可能」という OR 条件を $P_a \vee P_b$ とする．このようなポリシどうしの OR 条件を実現するためには，2 つのポリシ鍵のうち，どちらか一方からでも一時鍵 $t_0$ を導出できるような仕組みが必要となる．

そこで，$(k, n)$ 閾値秘密分散法の性質を逆に利用した廣田らの手法 [17] を用いることで，複数の暗号鍵のうちのいずれか 1 つで暗号文を復号するという OR 条件を実現する．これは，秘密情報をシェアに分割するのではなく，いくつかの定められたシェアと秘密情報から残りのシェアを求め，これを複数の暗号鍵に割り当てる手法である．しかし，この手法は単一の版のファイルに対してのみ適用が想定されており，複数版のファイルを扱うことはできない．よって本提案では，この手法を前節の連鎖鍵を用いた手法と融合させることで，ファイルの版に対して OR 条件を含むポリシ設定に基づいた削除保証を実現する．

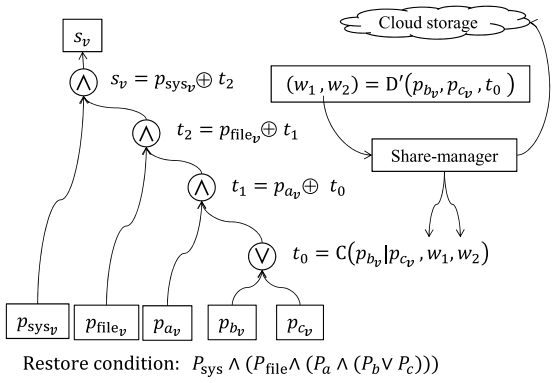


**図 5** 複数ポリシに基づく制御鍵の生成
**Fig. 5** Control key generation based on multiple policy.

具体的には，**図 5** の右上段に示す分散関数 D′ を用い，廣田らの手法により 2 つの暗号鍵から一時鍵を導出する．なお，この分散関数 D′ は，次の手順に従いランダムに生成された一時鍵 $t_0$，および 2 本のポリシ鍵 $p_{a_v}$，$p_{b_v}$ から，シェア $w_1$，$w_2$ を生成するものである．

まず，通常の $(k, n)$ 閾値秘密分散法は，式 (2) のように秘密情報 $S$ とガロア体上の独立な $k-1$ 個の乱数 $R_1, R_2, \cdots, R_{k-1}$ から構成される分散関数 $D(x)$，$1 \le x \le n$ により，シェア $w_1, \cdots, w_n$ を生成する．

$$w_x = \mathrm{D}(x) = S + R_1 x + R_2 x^2 + \cdots + R_{k-1} x^{k-1} \quad (2)$$

これに対して，$t_0$ を秘密情報 $S$，$p_{a_v}$，$p_{b_v}$ がシェアの一部 $w_3$，$w_4$ となるように，式 (3) から乱数 $R_1$，$R_2$ を掃き出し法などにより，代数的に決定する．そして，得られた乱数から $\mathrm{D}(1)$，$\mathrm{D}(2)$ を求めて $w_1$，$w_2$ を得る．

$$\begin{cases} p_{a_v} = \mathrm{D}(3) = t_0 + R_1 x + R_2 x^2 \\ p_{b_v} = \mathrm{D}(4) = t_0 + R_1 x + R_2 x^2 \end{cases} \quad (3)$$

ここで，$w_1$，$w_2$ のみから一時鍵 $t_0$ を生成できないよう，$n = 4$，$k = 3$ とすることが重要な点である．元の秘密情報である $t_0$ は $p_{a_v}$，$w_1$，$w_2$ または $p_{b_v}$，$w_1$，$w_2$ から結合関数 C により生成される．これにより，$p_{a_v}$ または $p_{b_v}$ から一時鍵 $t_0$ を求めることができ，ポリシどうしの OR 条件が実現される．

なお，$(k, n)$ 閾値秘密分散法における秘密情報は，図 5 中の一時鍵 $t_0$ に相当する．この一時鍵とは，ファイルサーバが Key-store に格納された複数のポリシ鍵から，メモリ上で制御鍵を生成する過程で導出されるものである．また，生成された図 5 中の $w_1$，$w_2$ の 2 個のシェアは，Key-store

に格納された 2 つのポリシ鍵のうち，ファイルサーバ上でいずれかの鍵から連鎖的に生成された鍵（図 5 中の $p_{b_v}$，$p_{c_v}$）と合わせることで，結合関数 C により秘密情報の一時鍵 $t_0$ を生成するシェアである．つまり，ポリシ鍵から連鎖的に生成された鍵も，$t_0$ を $(k, n)$ 閾値秘密分散法で生成するためのシェアの 1 つである．

$w_1$，$w_2$ は Share マネージャ（図 5 中の Share-manager）によってファイルサーバにキャッシュされた後，最終的に Chunk とともにクラウドストレージへ格納される．また，複数のファイルに対して，同じ組合せの OR 条件でポリシが割り当てられていた場合は，ファイルサーバに障害が発生しない限りシェアの再計算を行わず，Share マネージャから取得される．一時鍵に関しても，バックアップの処理中はファイルサーバのメモリ上でキャッシュすることで高速化を図り，ファイルサーバの記憶媒体や Key-store への永続化はしない．これは，一時鍵は前述のとおり Key-store に格納されたポリシ鍵と，Share マネージャに格納されたシェアから，ハッシュ連鎖と $(k, n)$ 閾値秘密分散法により復元できるためである．

他方，先に述べた閾値 $\mathrm{k} = 3$ より，第 $v$ 版の一時鍵の復元には 2 個のシェア $w_1$，$w_2$ に加えて，3 個目のシェアであるポリシ鍵から連鎖的に生成された図中 5 の $p_{b_v}$，$p_{c_v}$ が必要となる．つまり，$w_1$，$w_2$ はファイルサーバに加えてクラウドストレージの管理者が得ることもできるが，これらのみでは閾値に達しない．なお，$p_{b_v}$，$p_{c_v}$ および $w_1$，$w_2$ の生成は，ファイルサーバのみが行える．よって，ポリシ鍵が適切に Key-store 内で格納/削除され，ファイルサーバのメモリ上のみで扱われていれば，第三者は一時鍵を復元することができないため，本研究が想定する環境においては，これら $w_1$，$w_2$ のシェアは秘匿する必要はない．

次に，図 5 の計算木は，式 (4) のようなポリシが設定されたファイルについて，第 $v$ 版を復元するための制御鍵 $s_v$ を求める過程を示している．なお，制御鍵 $s_v$ はポリシ鍵 $p_{\mathrm{sys}_v}$，$p_{\mathrm{file}_v}$，$p_{a_v}$，$p_{b_v}$，$p_{c_v}$ から生成される．また，各第 $v$ 版のポリシ鍵は $v-1$ のポリシ鍵から前節のハッシュ関数（$v \geq 1$）により，連鎖的に生成される．

$$P_{\mathrm{sys}} \wedge (P_{\mathrm{file}} \wedge (P_a \wedge (P_b \vee P_c))) \tag{4}$$

まず，図中の右下段に示すように，$(k, n)$ 閾値秘密分散法の結合関数 C を用いて，$p_{b_v}$，$p_{c_v}$ および Share マネージャに格納された $w_1$，$w_2$ から一時鍵 $t_0$ を生成する．このとき，ポリシ $P_b$，$P_c$ ともに破棄されておらず有効な場合は，結合関数 $C$ に与えるポリシ鍵は，$k = 3$ より $p_{b_v}$ または $p_{c_v}$ のいずれかでよい．

次に，AND 条件で結合されたポリシについて $p_{a_v}$ と $t_0$，$p_{\mathrm{file}_v}$ と $t_1$ の順で排他的論理和をとり，一時鍵 $t_2$ を生成する．そして最後に，ポリシ鍵 $p_{\mathrm{sys}_v}$ と $t_2$ の排他的論理和から制御鍵 $s_v$ が生成される．

以上の手法により，本研究では AND/OR で結合された複数のポリシと版番号を基づいた復元可能条件の設定を実現する．

ところで，本研究で用いる $(k, n)$ 閾値秘密分散法は Shamir の多項式補間法に基づいたものである．ただし，たとえば $(k, d, n)$ ランプ型閾値秘密分散法や，その応用である廣田らの「部分情報の復元を制御できる手法」[18] など，あらかじめ定められた複数のシェアと秘密情報から，多項式補間法により残りのシェアを求めることができるアルゴリズムであれば，Shamir の手法以外でも本研究に適用できる．ただし，これらの閾値秘密分散法はシェアの容量効率が Shamir の手法に比べて良い反面，閾値以下のシェアでも秘密情報の一部を復元できる．よって，本研究では秘密情報を暗号鍵とし用いるため，その一部であっても漏洩することは許容できず，Secret Perfect な Shamir の方式を用いるのが妥当である．

### 4.3 ポリシ鍵の開示と更新

3.1 節で述べた調査に対して，調査者がファイルの復元と検証ができるよう，暗号鍵を開示する必要がある．このとき，調査者にファイル単位での調査を要求された場合は，制御鍵やデータ鍵を開示する方法も考えられる．しかし，これらの鍵は後述の連鎖性を有しておらず，開示された鍵の正当性を証明できないため，偽造することが容易である．よって，調査者に対しては，その要求に対応するポリシ鍵を開示する．これにより，調査者はポリシ鍵から制御鍵の生成とデータ鍵の復号を行うことで，クラウドストレージにバックアップされたファイルを復元することができる．

しかし，ポリシ鍵を得た調査者が，ハッシュ連鎖により「調査終了時点より未来のポリシ鍵を生成することができる」という弊害が発生する．つまり調査者は，取得したポリシ鍵からハッシュ連鎖により未来のポリシ鍵を生成し，そこから制御鍵を得ることができる．よって，クラウドベンダと結託することで，調査範囲を超えた，調査期間より後にバックアップされたファイルを不正に復元される可能性がある．

この問題に対して，調査が終了した時点で，以降のポリシ鍵が Forward Secure となるよう，ポリシ鍵を更新する手法を導入する．具体的には，開示した版より後方の，調査期間が終了する時点のポリシ鍵に乱数を加えることで，鍵の連鎖性を維持しつつ，ポリシ鍵を新しい鍵系列へと更新する．たとえば，調査者に開示した，あるユーザ A に対応する $v$ 版のポリシ鍵を $p_{a_v}$ とし，調査期間が終了する $e$ 版の時点のポリシ鍵を $p_{a_e}$ とする．このとき，$p_{a_e}$ の次の版の鍵 $p_{a_{e+1}}$ はポリシ鍵の鍵長と等しい bit 数の乱数 $r$ を用い，式 (5) から生成される．なお，$r$ と鍵の更新を行った版番号 $e$ は，ポリシ鍵とともに Key-store へ格納される．また，$p_{a_{e+1}}$ より後方の版の鍵は 4.1 節のとおり，1 つ前の

版の鍵から連鎖的に生成する．

$$p_{a_{e+1}} = \mathrm{H}(p_{a_e} \oplus r) \tag{5}$$

これにより，調査者は乱数 $r$ を知ることができないため，所持しているポリシ鍵を連鎖するのみでは，調査終了後の新しい鍵系列のポリシ鍵を生成できない．よって，調査者が未来の鍵を生成できる権利を剥奪することになり，調査期間より後にバックアップされたファイルを復元することは不可能となる．また，新しい鍵系列であっても，過去の鍵からの連鎖性は維持された状態である．そのため，乱数 $r$ を知ることができるファイルサーバは，最新の鍵を生成できるとともに，4.1 節で述べたポリシ鍵を上書きする手法により，過去の版を抹消できる．

ポリシ鍵を更新するタイミングは，3.1 節で述べた調査方法に依存する．たとえば，毎月定期的に監査を行うような環境では，そのつどポリシ鍵を更新する必要がある．また，インシデント発生時に調査が行われた際は，調査者に開示したポリシ鍵を更新する．他方，ポリシ鍵を更新することは，新たに Key-store で管理すべきデータが増加することを意味する．したがって，高頻度で調査が行わる場合は，Key-store に対する容量コストが懸念されるが，6.6 節で詳述のとおり現実的な頻度では実用上問題はないと考えられる．

他方，利益に反するファイルを隠蔽する，あるいは内容を偽造するために，ファイルサーバを管理する組織が意図的にポリシ鍵を捏造し，調査者に開示する可能性が考えられる．しかし，ポリシ鍵は過去の鍵から連鎖的に生成されており，組織が過去に調査を受けていれば，その調査を担当した調査者は古い鍵系列の基となるポリシ鍵を所持していると想定される．つまり，組織が新旧両方の鍵系列を完全に捏造するためには，過去に遡り第三者である調査者が所持している鍵も含めて整合性を維持する必要がある．よって，調査者に対して組織側がポリシ鍵を捏造し，正当なものであると偽証することは現実的に困難である．

### 4.4 ポリシ鍵の管理

4.1，4.2 節で述べた本研究の中核となる手法では，複数の暗号鍵を用いる．このうち，クラウドベンダに対するファイル内容の秘匿と削除保証のために，ポリシの鍵の管理が重要な課題となる．したがって，Key-store の実装方式は本稿の対象外とするが，本節では考えられる設計方針について説明する．

本研究では，すべてのポリシ鍵は Key-store に格納/管理されることを前提とする．Key-store には，ポリシ鍵を格納する機能，および Stratus の指示によってポリシ鍵を完全に削除する機能が求められる．この Key-store の実装にはたとえば下記のようなものが考えられ，ファイルサーバと同じ筐体内のデバイスを用いる方式や，外部の複数のサーバを利用する方式があげられる．

- 耐タンパ性を備えたセキュアデバイス
- $(k, n)$ 閾値秘密分散法を用いた複数 Key-store サーバ

シンプルな実装方式としては，耐タンパ性を備えたセキュアデバイスに対して，ポリシ鍵を格納することが考えられる．ただし，上記のようなデバイスは内部に格納できる鍵の本数が少ないことも予想される．そこで，セキュアデバイス上で新たにマスタ鍵となる共通鍵を生成/管理し，これを用いてポリシ鍵を暗号化する．さらに，暗号化されたポリシ鍵をファイルサーバ上に格納する．この仕組みにより，セキュアデバイスが管理すべき鍵はマスタ鍵 1 つのみとなる．ただし，ポリシ鍵が破棄される際は，上書き方式（2 章参照）を併用し，破棄対象のポリシ鍵に対して複数回の上書き処理を行う必要がある．

上記のセキュアデバイスを用いた方式は，ファイルサーバに対する内部犯による不正侵入および情報漏洩を考慮した場合，破棄されていないポリシ鍵を守る手段として有効である．他方，このような攻撃を他の方法で防ぐことを前提とした場合は，上記セキュアデバイスを用いる方式の派生として，5 章で述べるプロトタイプで採用したようなファイルサーバのハードディスクを利用した実装でもよい．これは，Stratus が動作するファイルサーバと同じ筐体内のハードディスク上にポリシ鍵を格納し，削除する機能を有するような，よりシンプルな実現方式である．ただし，3.1 節で述べた調査者による攻撃に対して，前述のセキュアデバイスと同様に，ポリシ鍵を削除するために複数回の上書き処理を行う必要がある．

次に $(k, n)$ 閾値秘密分散法を用いた方式について述べる．この方式では，ポリシ鍵を $(k, n)$ 閾値秘密分散法により複数のシェアへ分割し，Key-store の機能を実装した複数のサーバへ分散して保存する．あるポリシ鍵が必要な場合は Key-store のサーバ群から，定められた個数のシェアを回収し，ポリシ鍵を復元する．このような仕組みは，鍵管理の権限が単一のサーバや管理者へ依存することを防ぐとともに，障害対策としても有効である．

## 5. 実装

我々は，クラウドストレージを利用した増分バックアップシステムである Cumulus [4] を拡張し，4 章で述べた手法を実装することで，バージョン管理と複数ポリシに基づいた削除保証を両立させる，Stratus のプロトタイプを開発した．以下，本章では Stratus を構成するコンポーネントおよび Metadata の構造について示す．

なお，Stratus では共通鍵暗号方式として，鍵長 256 bit の Advanced Encryption Standard（AES）[20] を使用し，暗号モードは Counter-mode を用いた．各種ハッシュ値の生成および連鎖鍵の生成には，ハッシュ値長 256 bit の Secure Hash Algorithm-256（SHA-256）[21] を採用した．これら

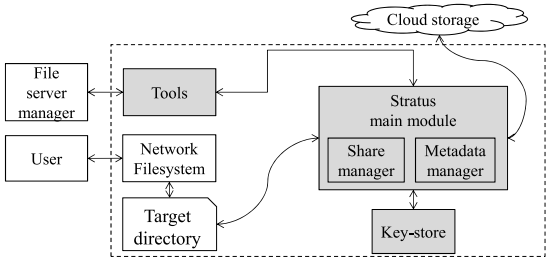


図 6 コンポーネント構成
**Fig. 6** Component structure.

暗号関連の処理には，Crypto++ [22] を用いている．

また，今回の実装では 4.4 節で述べた，ポリシ鍵を磁気媒体であるファイルサーバのハードディスクへファイルとして格納する方式を Key-store の実装方式として用いる．そのため，暗号化方式による削除保証に加え，Key-store から完全にポリシ鍵を削除するために上書き方式を併用し，残留磁気を消去するというアプローチをとっている．ただし，本来の上書き方式による削除保証は，ファイルのデータ全体に対して行われるものであるのに対し，あくまでデータ量の小さいポリシ鍵に対してのみ行っている．そのため，6.5 節のとおり Key-store からポリシ鍵を削除するために必要な時間的コストは低い．

### 5.1 コンポーネント

Stratus は**図 6** に示すコンポーネントから構成される．以下では，これらの詳細について述べる．なお，図中の破線内がファイルサーバ内を示し，特に灰色の背景のものがプロトタイプとして作成されたものである．

- **Stratus-main モジュール．** Stratus の中核をなすモジュールであり，対象のディレクトリ以下のバックアップおよびリストアを行う．
  サブモジュールの Metadata マネージャは，Metadata の管理を担当し，Share マネージャはポリシ間の OR 条件から制御鍵を生成するための，4.2 節で述べたシェアを格納する．
  なお，Metadata およびシェアは，最終的にファイルとして出力され，Chunk とともにクラウドストレージへアップロードされるが，バックアップ/リストアの高速化を図るため SQLite [23] を用い，ファイルサーバ上でキャッシングされる．
- **Key-store.** 制御鍵の生成に必要なポリシ鍵の管理を担当する．4.4 節で述べたような実装方式が考えられ，耐タンパ性を有するなどセキュアな必要がある．なお，現時点ではプロトタイプのため，ファイルサーバ上で Stratus-main モジュールとは独立して動作する単純なプロセスとして作成されており，ポリシ鍵を専用のディレクトリ内に保管する機能，破棄が要求された際に DoD 5220.22-M [8] を用いて複数回上書き/抹消する機能を有する．
- **ツール群．** Stratus のフロントエンドとなるツール群である．バックアップの作成には stratus-store，リストアには stratus-restore コマンドを用いる．さらに，ポリシの生成/破棄に利用される stratus-policy や，ファイルへのポリシの割当てを行う stratus-mod などが用意されている．
- **クラウドストレージ．** バックアップされたファイルの格納先となるバックエンドのストレージである．オブジェクトに対して GET，PUT，DELETE，LIST など基本的な操作を受付けるものであればベンダは問わない．
- **ネットワークファイルシステム．** 組織内のクライアントがファイルサーバを利用できるような機能を提供する．なお，クライアントに対して，どのような方式でサービスを提供するかは本研究の対象外とする．他方，ファイルシステム上のパーミッションや所有者，所有グループ，アクセスコントロールリストと Stratus のポリシ管理機構が連携/同期され，適切なアクセス制御と削除保証が実施されることが望ましい．ただし，この点については今後の検討課題である．

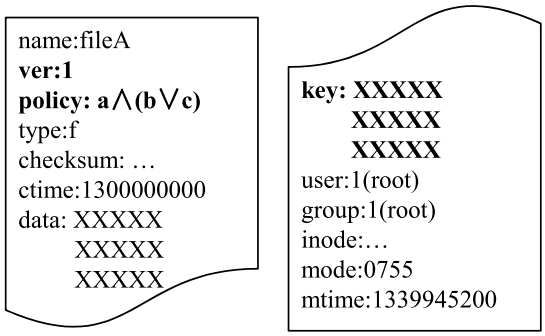


図 7 Metadata の構成
**Fig. 7** Metadata structure.

### 5.2 Metadata の構成

Stratus は Cumulus と同様に，バックアップされたファイルの版ごとに**図 7** に示すような Metadata を生成する．この Metadata には，ファイル名やパーミッション，所有者など，Linux におけるファイルシステム（たとえば ext3）がファイルごとに保持している i-node 構造体の一部，およびファイルを構成する各 Chunk の識別子（Chunk のハッシュ値）が含まれる．これに加えて，Stratus では図中の太字で示された版番号，ファイルへ割り当てられたポリシ，データ鍵が含まれる．

## 6. 評価実験

本章では Stratus のプロトタイプに関する評価について述べる．この評価実験の目的は，既存のクラウドファイル

バックアップシステムに対して，4 章で示した削除保証の手法を適用したことによる，バックアップ/リストアのオーバヘッド，削除保証に要する処理時間，および Key-store に格納されるポリシ鍵のデータサイズを明らかにし，有用性を確認することである．

ところで，本稿の執筆時点では FadeVersion は一般に公開されておらず，直接的な比較が難しいのが現状である．そこで，Rahumed らの論文 [6] で比較対象としている Cumulus [4] を用い，間接的に FadeVersion と Stratus を比較した．

### 6.1 実験環境

最終的なバックアップ先のクラウドストレージには Amazon S3 を使用し，格納先のコンテナとなる “Bucket” のリージョンは “東京” とした．なお，バックアップ元となるファイルサーバの仕様は次のとおりであり，大学内に設置されている．OS: CentOS 6.2（Kernel-2.6.32-220，64 bit），CPU: Intel Xeon（3.1 GHz，4-cores），RAM: 8 GB，HDD: 160 GB（SATA 7,200 rpm）．

本研究の目的より，現実の医療カルテや企業の取引記録を用いて評価を行うことが理想であるが，現状はそれらの入手が困難である．そこで，Rahumed らおよび Vrable らの文献 [4], [6] と同様に，実運用されているファイルサーバ上の，home ディレクトリの内容を 61 日間（2012 年 4 月 1 日–5 月 31 日）にわたり rdiff-backup [24] で記録し，これをワークロードとした．なお，上記の home ディレクトリにはプログラムのソースコードやバイナリファイル，画像，学術系のドキュメント，E メールなどが含まれており，ユーザ数は 7 名である．

Cumulus および Stratus による増分バックアップの実験は，まず rdiff-backup を用いて最新のスナップショットを取得し，その後このスナップショットに対して行う．さらに，OS によるキャッシュの影響を排除するため，直前にページキャッシュ，ダーティキャッシュ，i-node キャッシュの解放を行っている．

ワークロードについて，**表 1** は初日および最終日における基本統計量を，**図 8** はファイルサイズによる累積分布を計測したものである．また，**図 9** は評価期間中に rdiff-backup を用いて取得された，1 日あたりの増分量の推移である．結果より，平日と休日で増分量に差があるが，全体の格納量は日々増加する傾向にあり，初日と最終日を比較すると中央値やファイルサイズの分布に大きな変化はない．なお，1 日あたりの増分量の平均は 31.48 MB であった．

Stratus による増分バックアップでは，すべてのファイルに対して $P_{\text{sys}} \wedge (P_{\text{file}} \wedge (P_a \vee P_b))$ というポリシを設定した．これは各ファイルに必ず付加される SP と FP に加えて，ファイルサーバの管理者が任意に決めることができる AP の $P_a$，$P_b$ が設定されることを示す．これは，たとえば組織内で進行中である，2 つのプロジェクトに割り当てられたポリシなどを想定しており，両方のポリシが破棄された場合にファイルの削除が保証される．なお，$P_{\text{sys}}$ は前述のとおり，システム全体で 1 つであるが，$P_{\text{file}}$，$P_a$，$P_b$ はすべてのファイルに対して，互いに独立したものが割り当てられる．

表 1 初日および最終日の統計量
**Table 1** Statistics on the initial state and the state on the final day.

| | Initial state | Final state |
|---|---|---|
| # of files | 140,475 | 155,167 |
| Average | 54 KB | 60 KB |
| Median | 19 KB | 20 KB |
| Largest | 235 MB | 235 MB |
| Total | 7.6 GB | 9.4 GB |

### 6.2 初回バックアップ

日々の増分バックアップとは異なり，初回は指定されたディレクトリ以下に含まれる全ファイルがバックアップの対象となる．そこで，まず初回のフルバックアップにおいて，ファイルサーバ上でバックアップデータを生成する

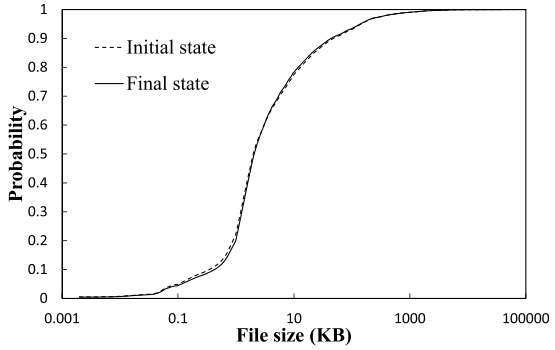


図 8 初日および最終日におけるファイルサイズの累積分布
**Fig. 8** Cumulative distribution of file sizes in the initial and the final day.

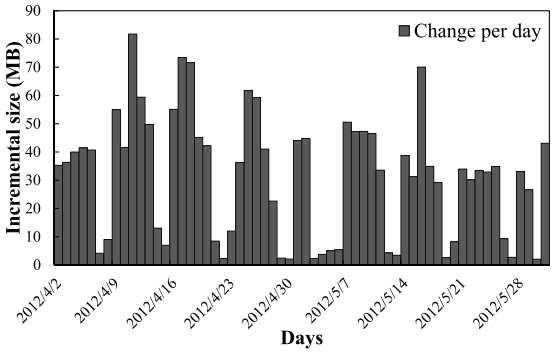


図 9 rdiff-backup の報告による増分サイズ変化
**Fig. 9** Size of incremental changes as reported by rdiff-backup.

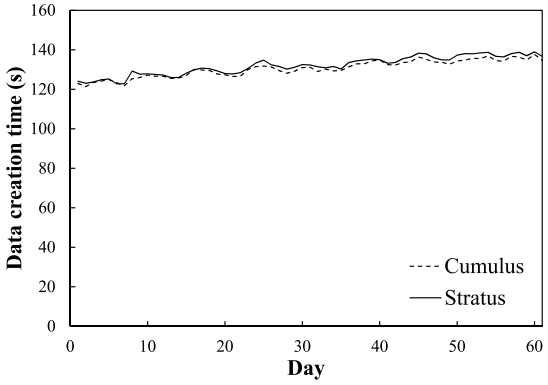


図 10 増分バックアップデータの生成に要する時間の推移
**Fig. 10** Changes in the time required for creation of incremental backup data.

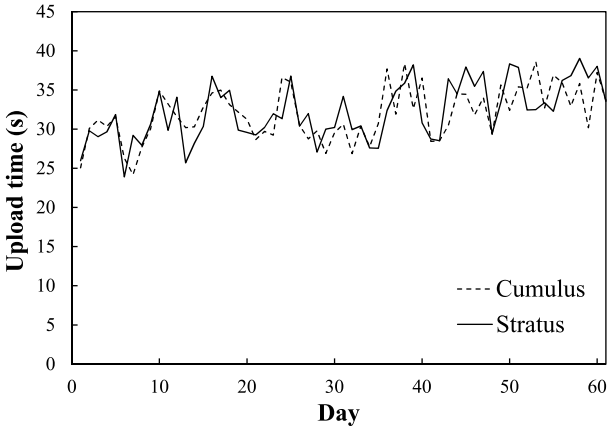


図 11 増分バックアップデータのアップロードに要する時間の推移
**Fig. 11** Changes in the time required for upload incremental backup data.

フェーズに要する時間について評価した．

その結果，Cumulus が 812 秒，Stratus が 855 秒であり，その差は 5.2%増であった．なお，増加分の内訳は 67.2%がポリシから制御鍵を生成する鍵生成部，29.7%が Chunk の暗号化，3.1%がその他の処理であった．FadeVersion では，フルバックアップを行った際の Cumulus との差が 3.2%と報告されている．したがって，間接的に FadeVersion と比較すると，その差は 2.0%である．このオーバヘッドの差は，$(k, n)$ 閾値秘密分散法による鍵生成に起因すると考察される．

次に，生成されたバックアップデータをクラウドストレージへアップロードするフェーズについて評価を行った．なお，生成されたバックアップデータの容量は Cumulus が 6.38 GB，Stratus が 6.41 GB であった．この差は提案手法によって新たに Metadata へ加えられた要素（5.2 節参照），およびポリシどうしの OR 条件を実現するために生成されたシェアによるものである．この結果，バックアップデータのアップロードに要する時間は Cumulus が 771 秒，Stratus が 775 秒であった．

以上より 2 つのフェーズを合わせた，全体の時間は Cumulus が 1,583 秒，Stratus が 1,630 秒であり，2.9%増であった．この結果より，Cumulus に対する Stratus のオーバヘッドはわずかであり，またフルバックアップは初回のみの動作となるため，運用へ影響を及ぼすことはないと考える．

### 6.3 増分バックアップ

6.1 節で述べたワークロードを基に，日単位の増分バックアップを Cumulus と Stratus で実施した．

まず，ローカル上でバックアップデータを作成するフェーズについて評価を行った．**図 10** は毎日 1 回行われる増分バックアップにおいて，バックアップデータの生成に要した時間の推移を表している．結果より，平均で Cumulus は 130.4 秒，Stratus は 131.9 秒を要し，1.2%増となった．なお，このオーバヘッドは前節のとおり Chunk の暗号化および鍵生成に必要な時間に起因する．初回バックアップに比べてオーバヘッドが小さいのは，フルバックアップ時のファイル数に対して，日々バックアップすべきファイル数が少なく，鍵生成の回数が抑えられるためである．

次に**図 11** は，生成されたバックアップデータをクラウドストレージへアップロードするのに要した時間の推移を示している．ネットワークのスループットに起因する揺れも大きいと考えられるが，平均は Cumulus が 31.9 秒，Stratus が 32.2 秒であり，その差は 0.9%の増加であった．

以上より，日々の増分バックアップにおけるオーバヘッドの平均は 1.1%とわずかであり，実用的な範囲と考える．また，今回の実験ではすべてのファイルに対して独立した AP を割り当てた．他方，実際の環境では複数のファイルに対して，同じポリシが割り当てられることが多いと考えられる．そのようなケースでは，キャッシュによって鍵生成の回数が減り，よりオーバヘッドは低減される．

### 6.4 リストア

最終日となる 61 日目にバックアップされた状態をリストアする実験を行った．なお，リストアについてもバックアップ時と同様に，クラウドストレージからバックアップデータをダウンロードするフェーズと，バックアップデータからファイルを復元するフェーズに分けられる．ただし，ダウンロードのフェーズにおける Cumulus と Stratus の差は，アップロード時と同様である．したがって，ローカルストレージ上のバックアップデータからのリストアについて評価した．

その結果，Cumulus は 577 秒であったのに対し，Stratus は 11.4%増の 643 秒であった．なお，最終日のバックアップデータの容量は Cumulus が 8.13 GB，Stratus が 8.16 GB である．初回フルバックアップと比較したオーバヘッドの差は，扱うファイル数が時間経過とともに増加したこと

も一因である．加えて，Cumulus に由来するリストアプログラムが Python により記述されている．よって，特に Stratus では暗号処理に関する外部のライブラリを多く呼び出すため，これがオーバヘッドになったと考えられる．他方，リストアに関する Cumulus 対 FadeVersion の 55.1%のオーバヘッドと比較すると，Cumulus 対 Stratus の差は圧縮されている．これは，評価環境のサーバの CPU が AES に関する命令セットを有していることや，暗号ライブラリが異なることが要因であると考察する．

### 6.5 削除保証

ファイルの削除を保証するために必要な処理は，ポリシ鍵の破棄とクラウドストレージからの Chunk の削除に大別される．ただし，ポリシ鍵の破棄により制御鍵が回復不能になった時点で，クラウド上の Chunk から元のファイルを復元することはできない．また，Chunk の削除はクラウドストレージ側の性能に大きく依存し，Cumulus，FadeVersion，Stratus のいずれのでも削除すべき Chunk 数に差はない．

そこで，ワークロードの最終日の状態において，1 カ月前の 4 月 30 日以前にバックアップされた全ファイルの版を，ポリシに基づいて削除保証するために必要な時間を計測した．なお，全ファイルにわたる削除保証の処理は，Key-store で管理されている SP のポリシ鍵 $p_{\mathrm{sys}_0}$ を DoD 5220.22-M（7 回上書き）で抹消し，5 月 1 日の版 $p_{\mathrm{sys}_{30}}$ で更新することで実現される．

このポリシ鍵の破棄/更新に必要な処理時間を 1,000 回測定した結果，平均 62 ミリ秒であった．以上より，バックアップされた複数のファイルおよび版に対する削除保証がわずかな時間で実現されることを確認した．

### 6.6 Key-store

ポリシ鍵を保持するために Key-store に対して要求される容量を明らかにするため，ポリシ鍵のデータサイズに関して述べる．今回のプロトタイプで実装された Key-store は，5.1 節のとおりファイルサーバ上で動作するプロセスであり，各ポリシ鍵をファイルとして保持する．また，1 本のポリシ鍵は 32 Byte（256 bit）であり，付与される識別子は 8 Byte である．なお，FadeVersion については 4 章の冒頭で述べたとおり，版ごとの削除保証を可能とするため，ファイルの各版に対して 1 つの新しいポリシを割り当てるものと仮定した．つまり，各版につき 1 つの制御鍵が Key-store に格納されることになる．

**図 12** は Stratus と FadeVersion において，ユーザ数，グループ数，バックアップされたファイル数を複数仮定し，Key-store に格納されるデータの容量を示したものである．結果より，Stratus では 100 ユーザ，10 グループ，100,000 ファイルの場合に 4.0 MB，最もポリシ数が多い

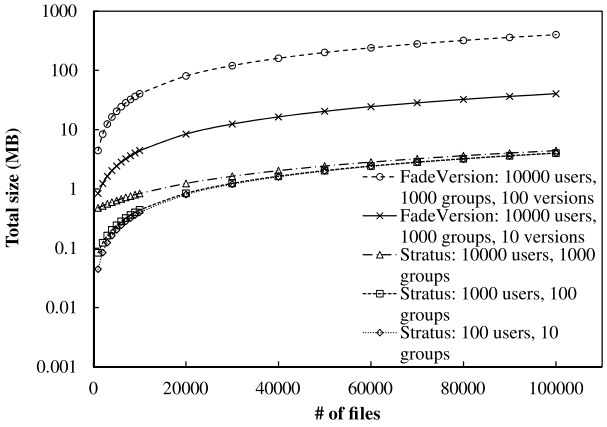


**図 12** Key-store に格納されるデータ量
**Fig. 12** Size of the stored data in the Key-store.

10,000 ユーザ，1,000 グループの場合は 4.4 MB であった．他方，FadeVersion では各ファイルに対して 100 版の状態を保持すると，10,000 ユーザ，1,000 グループ，100,000 ファイルで 400.0 MB の容量が Key-store 上で消費される．

なお，4.3 節で述べたポリシ鍵の開示を調査者に要求された場合，調査者が未来のポリシ鍵を生成できないよう，調査終了後に開示したポリシ鍵を新しい鍵系列に更新する必要がある．この鍵系列の更新は，1 つのポリシ鍵あたり，ポリシ鍵の識別子で 8 Byte，乱数で 32 Byte（256 bit），更新した時点の版番号で 8 Byte が消費される．よって，たとえば 1 カ月に 1 度の頻度で監査が行われると想定し，前述の最もポリシ数が多い 10,000 ユーザ，1,000 グループ，100,000 ファイルの場合，1 回の調査にともなう鍵更新で 5.3 MB を消費する．また，同様の頻度で 1 年間にわたり監査を行った場合は 63.9 MB を消費する．

3.1 節で述べた本研究が想定する「1 日単位で増分バックアップを行う」ような環境では，第三者による監査の間隔は最も短くて 1 日単位であるうえ，1 カ月より短い間隔で行うことは金銭的なコスト面で現実的ではない．また，インシデントにともなう調査については，場合によっては短期間に高い頻度でファイルを調査する必要が生じるが，同一インシデントの調査を行う調査者が一定期間内に頻繁に変更されるとは考え難く，一定期間はポリシ鍵の鍵系列を変更しなくても問題は生じない．他方，確かにその期間内は調査者が未来に作成されるポリシ鍵を生成可能である．しかし，次にポリシ鍵の鍵系列が変更されるまでの間は，その調査者は任意のタイミングでファイルの鍵を要求する権利を元々有している．以上より，Key-store の容量コストについて，実用面で問題はないと考えられる．

このように，Stratus が FadeVersion と比較して格段に消費量が低いのは，FadeVersion が版ごとのポリシに対応した制御鍵を Key-store に格納する必要があるのに対して，Stratus はファイルごとのポリシ鍵から版ごとの制御鍵を生成するため，版数に対して影響を受けないためである．

よって，FadeVersion と比較して，版数が多いほど Stratus は Key-store の容量コストにおいて優位性があるといえる．

## 7. まとめ

本稿では，クラウドストレージ上へ格納されたファイルについて，完全な削除が保証されないという問題に着目した．これに対して，我々はバージョン管理と複数のポリシに基づいた削除保証を実現するファイルバックアップシステム，Stratus を提案した．Stratus は既存の研究と比較して，バックアップされたファイルの復元に必要な制御鍵を，ハッシュ連鎖と $(k, n)$ 閾値秘密分散法に基づいて生成するという特徴がある．これにより，バックアップされたファイルのうち，特定の版以降を残し，それ以前を一括して削除するといった操作を可能にした．また，複数ポリシを AND/OR で組み合わせることで，ファイルの性質に応じた柔軟な復元可能条件の設定を実現した．

評価より，既存のバックアップシステムである Cumulus と比較して，Stratus のオーバヘッドは，初回フルバックアップで 2.9%，増分バックアップで 1.1%，リストアで 11.4%であった．また，現実における調査の頻度を考慮した場合，調査にともなうポリシ鍵の更新を月単位で行ったとしても，Key-store に与えるコストのインパクトは実用上問題はないと考えれられる．

以上より本研究が対象とする，ファイルの性質に応じて過去の版を連続的に削除する必要がある環境において Stratus は有用であると考える．また，今後の展望として，定期的な増分バックアップではなく，ADEC [15] のような，仮想ファイルシステムを用いたリアルタイムにバージョン管理と削除保証を行うシステムへも，Stratus の手法を適用させることを計画している．

### 参考文献


[1] 木村道弘，宮崎一哉，前田陽二：電子文書保存のしくみと実務，中央経済社 (2008).

[2] 独立行政法人情報処理推進機構：クラウド・コンピューティング社会の基盤に関する研究会報告書 (2010).

[3] Amazon.com: Amazon Simple Storage Service (S3), Amazon.com (online), available from ⟨http://aws.amazon.com/s3/⟩ (accessed 2012-07-17).

[4] Vrable, M., Savage, S. and Voelker, G.M.: Cumulus: Filesystem Backup to the Cloud, *Proc. 7th USENIX Conf. on File and Storage Technologies*, pp.225–238 (2009).

[5] Vrable, M., Savage, S. and Voelker, G.M.: Cumulus: Filesystem backup to the cloud, *ACM Trans. Storage*, Vol.5, No.4, pp.14:1–14:28 (2009).

[6] Rahumed, A., Chen, H.C.H., Tang, Y., Lee, P.P.C. and Lui, J.C.S.: A Secure Cloud Backup System with Assured Deletion and Version Control, *Proc. Intl. Conf. on Parallel Processing*, pp.380–397 (2011).

[7] Gutmann, P.: Secure Deletion of Data from Magnetic and Solid-State Memory, *Proc. 6th USENIX Security Symposium*, pp.77–89 (1996).

[8] U.S. Department of Defense: Industrial Security Manual for Safeguarding Classified Information (1984).

[9] Peterson, Z.N.J. and Burns, R.: Ext3cow: A Time-Shifting File System for Regulatory Compliance, *ACM Trans. Storage*, Vol.1, No.2, pp.190–212 (2005).

[10] Perlman, R.: File System Design with Assured Delete, *Proc. 3rd IEEE Intel. Security in Storage Workshop*, pp.83–88 (2005).

[11] Tang, Y., Lee, P.P.C., Lui, J.C.S. and Perlman, R.: FADE: Secure Overlay Cloud Storage with File Assured Deletion, *Proc. 6th Intel. Conf. on Security and Privacy in Communication Network*, pp.380–397 (2010).

[12] Rivest, R.L., Shamir, A. and Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM*, Vol.21, pp.120–126 (1978).

[13] Elgamal, T.: A Public Key Cryptosystem and a SignatureScheme Based on Discrete Logarithms, *IEEE Trans. Inf. Theory*, Vol.31, No.4, pp.469–472 (1985).

[14] 田中敏之，西出隆志，櫻井幸一：安全なクラウドストレージを実現する FADE への改良の提案，情報処理学会コンピュータセキュリティシンポジウム 2011 論文集，Vol.2011, No.3, pp.167–172 (2011).

[15] Tezuka, S., Uda, R. and Okada, K.: ADEC: Assured Deletion and Verifiable Version Control for Cloud Storage, *Proc. 26th IEEE Intel. Conf. on Advanced Information Networking and Applications*, pp.22–30 (2012).

[16] Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance, *J. ACM*, Vol.36, No.2, pp.335–348 (1989).

[17] 廣田啓一，北原 亮，高本正信：複数鍵暗号化装置，複数鍵復号装置，複数鍵暗号化システム及びプログラム，特許第 4587452 号 (2010).

[18] 廣田啓一，北原 亮，遠藤雅和，山室雅司：ランプ型閾値秘密分散法における部分情報の復元制御，電子情報通信学会技術研究報告 ISEC，情報セキュリティ，Vol.103, No.416, pp.57–64 (2003).

[19] 大津一樹，井上亮文，宇田隆哉，松下 温：P2P ファイル共有システムにおける鍵管理効率化手法の実装評価，情報処理学会論文誌，Vol.47, No.8, pp.2464–2476 (2006).

[20] Daemen, J., Rijmen, V. and Leuven, K.U.: AES Proposal: Rijndael, *AES Algorithm Submission*, pp.343–348 (1999).

[21] U.S. National Institute of Standards and Technology: Secure hash standard (2002).

[22] Dai, W.: Crypto++ (online), available from ⟨http://www.cryptopp.com⟩ (accessed 2012-07-17).

[23] Hipp, D.R.: SQLite (online), available from ⟨http://www.sqlite.org⟩ (accessed 2012-07-17).

[24] Tridgell, A., Mackerras, P. and Davison, W.: rdiff-backup (online), available from ⟨http://www.nongnu.org/rdiff-backup/⟩ (accessed 2012-07-17).

**手塚 伸** （学生会員）

2006 年東京工科大学情報工学科卒業，2008 年同大学院バイオ・情報メディア研究科修士課程修了．現在，慶應義塾大学大学院理工学研究科開放環境科学専攻後期博士課程に在学中，慶應義塾大学 ITC 本部助教．ネットワークセキュリティの研究に従事．2006 年情報処理学会 DICOMO 2006 優秀プレゼンテーション賞受賞．2005 年度下期未踏ソフトウェア創造事業採択（共同開発者）．

**宇田 隆哉** （正会員）

1998 年慶應義塾大学理工学部計測工学科卒業．2000 年同大学院理工学研究科計測工学専攻前期博士課程修了．2002 年同大学院理工学研究科開放環境科学専攻後期博士課程修了．博士（工学）．現在，東京工科大学コンピュータサイエンス学部講師．ネットワークセキュリティの研究に従事．2002 年 IFIP/SEC 2002 Best Student Paper Award 受賞．電子情報通信学会会員．

**岡田 謙一** （フェロー）

慶應義塾大学理工学部情報工学科主任教授，工学博士．専門は，CSCW，グループウェア，HCI．情報処理学会理事，情報処理学会誌編集主査，論文誌編集主査，GN 研究会主査，日本 VR 学会理事等を歴任．現在，情報処理学会論文誌：デジタルコンテンツ編集長，電子情報通信学会 HB/KB 幹事長．情報処理学会論文賞（1996，2001，2008 年），情報処理学会 40 周年記念論文賞等を受賞．日本 VR 学会フェロー，IEEE，ACM，電子情報通信学会，人工知能学会各会員．