

(19) 日本国特許庁 (JP)

(12) 特 許 公 報 (B2)

(11) 特許番号

特許第5075236号
(P5075236)

(45) 発行日 平成24年11月21日 (2012.11.21)

(24) 登録日 平成24年8月31日 (2012.8.31)

(51) Int.Cl.	F I
G06F 21/24 (2006.01)	G06F 21/24 166A
G06F 12/00 (2006.01)	G06F 12/00 545A
H04L 9/32 (2006.01)	G06F 12/00 537A
G09C 1/00 (2006.01)	G06F 12/00 531D
	H04L 9/00 675A
請求項の数 9 (全 39 頁) 最終頁に続く	

(21) 出願番号	特願2010-186592 (P2010-186592)	(73) 特許権者	500046438
(22) 出願日	平成22年8月23日 (2010.8.23)		マイクロソフト コーポレーション
(62) 分割の表示	特願2003-405163 (P2003-405163)		アメリカ合衆国 ワシントン州 9805
原出願日	平成15年12月3日 (2003.12.3)		2-6399 レッドモンド ワン マイ
(65) 公開番号	特開2011-8818 (P2011-8818A)		クロソフト ウェイ
(43) 公開日	平成23年1月13日 (2011.1.13)	(74) 代理人	110001243
審査請求日	平成22年8月23日 (2010.8.23)		特許業務法人 谷・阿部特許事務所
(31) 優先権主張番号	10/310,440	(74) 復代理人	100115624
(32) 優先日	平成14年12月5日 (2002.12.5)		弁理士 濱中 淳宏
(33) 優先権主張国	米国 (US)	(74) 復代理人	100129171
前置審査			弁理士 柿沼 健一
		(72) 発明者	ミゲル オーム テムド デ カストロ
			イギリス シービー4 300 ケンブリ
			ッジ リッチモンド ロード 25
			最終頁に続く

(54) 【発明の名称】 サーバーレス分散ファイルシステムにおけるセキュアリカバリ

(57) 【特許請求の範囲】

【請求項 1】

複数の参加コンピュータメンバーが含まれている分散ディレクトリグループ内のコンピュータメンバー上でファイルを安全に更新する前記コンピュータが実行可能な方法であって、前記方法は、

複数の秘密鍵のうち1つの秘密鍵を前記複数のメンバーの各メンバーが、前記グループ内で受信することと、

確立された安全な接続と異なる安全でないチャネル上で、ファイル内容のハッシュと、前記複数の秘密鍵を使用して作成されたメッセージ認証コードのベクトルによりセキュリティ保護された1つまたは複数の更新ファイルを含む転送データを受信することと、

前記分散ディレクトリグループ内の予め定めた数のメンバーにメッセージ認証コードの前記ベクトル内の予め定めた数の前記メッセージ認証コードを検証することを要求することにより前記更新ファイルを認証することと

を含み、

前記メンバーの合計の数は、 $3f + 1$ であり、前記予め定めた数のメッセージ認証コードを検証する必要があるメンバーの数は $f + 1$ であり、前記予め定めた数のメッセージ認証コードは $2f + 1$ であり、 f は前記分散ディレクトリグループにより許容可能なフォールトの個数であることを特徴とする方法を特徴とする方法。

【請求項 2】

前記メッセージ認証コードは、UMACメッセージ認証コードであることを特徴とする

請求項 1 に記載の方法。

【請求項 3】

ユーザーは、前記分散ディレクトリグループの各メンバーとの対称鍵に基づく関係を確立することを特徴とする請求項 1 に記載の方法。

【請求項 4】

プロセッサにより実施されるコンピュータ実行可能命令を格納するコンピュータ可読記録デバイスであって、前記命令は、

複数の秘密鍵のうち 1 つの秘密鍵を前記複数の参加コンピュータメンバーが含まれる分散ディレクトリグループの各メンバーにより前記グループ内で、受信することと、

確立された安全な接続と異なる安全でないチャンネル上で、ファイル内容のハッシュと前記複数の秘密鍵を使用して作成されたメッセージ認証コードのベクトルによりセキュリティ保護された 1 つまたは複数の更新ファイルを含む転送データを受信することと、

前記分散ディレクトリグループ内の予め定めた数のメンバーにメッセージ認証コードの前記ベクトル内の予め定めた数の前記メッセージ認証コードを検証することを要求することにより前記更新ファイルを認証することと

を含む処理を実行し、

前記メンバーの合計の数は、 $3f + 1$ であり、前記予め定めた数のメッセージ認証コードを検証する必要があるメンバーの数は $f + 1$ であり、前記予め定めた数のメッセージ認証コードは $2f + 1$ であり、 f は前記分散ディレクトリグループにより許容可能なフォールトの個数であることを特徴とするコンピュータ可読記録デバイス。

【請求項 5】

前記メッセージ認証コードは、UMAC メッセージ認証コードであることを特徴とする請求項 4 に記載のコンピュータ可読記録デバイス。

【請求項 6】

ユーザーは、前記分散ディレクトリグループの各メンバーとの対称鍵に基づく関係を確立することを特徴とする請求項 4 に記載のコンピュータ可読記録デバイス。

【請求項 7】

プロセッサと、

少なくとも 1 つのメモリとを含み、前記メモリ上に、複数の参加コンピュータメンバーが含まれる分散ディレクトリグループのコンピュータメンバーを実装する複数のプログラム命令を含むコンピュータデバイスであって、

前記プログラム命令は、動作を実施するためにプロセッサにより実施されるように構成され、前記動作は、

複数の秘密鍵のうち 1 つの秘密鍵を受信することであって、複数のメンバーのそれぞれのメンバーが、複数の秘密鍵のうち異なる 1 つの秘密鍵を前記グループ内で、受信することと、

確立された安全な接続と異なる安全でないチャンネル上で、ファイル内容のハッシュと前記複数の秘密鍵を使用して作成されたメッセージ認証コードのベクトルによりセキュリティ保護された 1 つまたは複数の更新ファイルを含む転送データを受信することと、

前記分散ディレクトリグループ内の予め定めた数のメンバーにメッセージ認証コードの前記ベクトル内の予め定めた数の前記メッセージ認証コードを検証することを要求することにより前記更新ファイルを認証することと

を含み、

前記メンバーの合計の数は、 $3f + 1$ であり、前記予め定めた数のメッセージ認証コードを検証する必要があるメンバーの数は $f + 1$ であり、前記予め定めた数のメッセージ認証コードは $2f + 1$ であり、 f は前記分散ディレクトリグループにより許容可能なフォールトの個数であることを特徴とするコンピュータデバイス。

【請求項 8】

前記メッセージ認証コードは、UMAC メッセージ認証コードであることを特徴とする請求項 7 に記載のコンピュータデバイス。

【請求項 9】

ユーザーは、前記分散ディレクトリグループの各メンバーとの対称鍵に基づく関係を確認することを特徴とする請求項 7 に記載のコンピュータデバイス。

【発明の詳細な説明】

【技術分野】

【0001】

本発明は、コンピュータネットワークおよびファイルシステムに関するものであり、特に、サーバーレス分散ファイルシステム (serverless distributed file system) におけるセキュアリカバリ (secure recovery) に関するものである。

10

【背景技術】

【0002】

ファイルシステムは、コンピュータシステム上に格納されているファイルおよびその他のデータオブジェクトを管理する。ファイルシステムは、もともと、ローカルにある常駐の記憶媒体に格納されているファイルにアクセスできるようにコンピュータのオペレーティングシステムに組み込まれていた。コンピュータがネットワーク化されるようになると、一部のファイルストレージ機能が個々のユーザーマシンから取り外され、ユーザーマシンに代わって大量のファイルを格納する特別なストレージサーバーに移された。ファイルが必要になった場合、ユーザーマシンは、ただ単にファイルをサーバーに要求した。このサーバーベースのアーキテクチャでは、ネットワーク上のリモートのストレージサーバーに格納されているファイルの管理およびアクセスを行えるようにファイルシステムの拡張が行われた。

20

【発明の概要】

【発明が解決しようとする課題】

【0003】

今日、ファイルストレージは、ファイルを中央のストレージサーバーに格納するのではなく、ネットワーク化されているさまざまなコンピュータ上に格納するモデルに移行しつつある。サーバーレスアーキテクチャでは、ファイルシステムに新たな問題が生じる。とりわけ問題の 1 つは、ファイルを管理すること、すなわち、ファイルを性をもって格納でき、またその時々に入れ替わり立ち替わりいくつかのコンピュータがアクセス不能になったとしてもファイルにアクセス可能で、しかもそれと同時に不正ユーザーによるファイルへのアクセスを防止するような方法で、多数の異なるコンピュータ上に分散されているファイルを管理することに関係する。特に重要なのは、コンピュータのハッカーや不正ユーザーがネットワーク接続されているコンピュータ内に格納されているデータを改ざんしたり、破壊したりするおそれのある惨事が万が一発生した場合のために、コンピュータに格納されているデータを保護する必要があるという点である。

30

【課題を解決するための手段】

【0004】

本発明は、これらの問題点に対処し、サーバーレス分散ファイルシステムに有効な解決策を提示する。

40

【0005】

サーバーレス分散ファイルシステムに生じた大惨事から復旧するための方法およびシステムについて説明する。より具体的には、オーセンティケータ (authenticator) にセキュアファイル書き込み (secure file writes) が伴う場合に、大惨事発生後、セキュアファイル書き込みを行うためのシステムおよび方法をサーバーレス分散ファイルシステム内の認証されていないチャンネル上で可能にすることができる。オーセンティケータは、期間制限のある委任証明書 (power-of-attorney certificate)、メッセージ認証コード (MAC) のベクトル (vector of message authentication codes)、またはサーバーレス分散ファイルシステムのメンバー間で共有される秘密鍵でセキュリティ保

50

護されている単一メッセージオーセンティケータコード (MAC) である。サーバーレス分散ファイルシステムは、 f をシステムが許容できるフォールトの数を表す値として、少なくとも $3f + 1$ 個の参加コンピュータメンバーを含む。グループは、ファイル作成およびファイルアップロード用に少なくとも 1 つのオーセンティケータを必要とする。

【0006】

一実施形態では、オーセンティケータは、委任証明書であり、この委任証明書には、予め定めた時間の間にファイルアップロードを許可する時間構成要素を含む予め定めた基準を有する。

【0007】

他の実施形態では、オーセンティケータは、秘密鍵によりセキュリティ保護されたメッセージ認証コードである。グループはクライアントマシンからログを受け取る。このログにはファイル書き込みを含み、メッセージ認証コードにより認証される。秘密鍵は、メッセージ認証コードに関連付けられており、そのために、秘密鍵を再構成するためには、グループの $f + 1$ 個のメンバーが秘密鍵の割り当て分を提供する必要がある。

【0008】

一実施形態では、オーセンティケータは、複数の秘密鍵によりセキュリティ保護された MAC のベクトルであり、グループの各メンバーは複数の秘密鍵のうちの 1 つを所有する。この実施形態では、グループは、アップロードを認証するために少なくとも $f + 1$ 個のメンバーを必要とするように構成されている。特に、この実施形態では、各メンバーは n 個のオーセンティケータのうちの 1 つを受け取り、後から、安全でないチャネルで、ファイル内容のハッシュによりセキュリティ保護されている 1 つまたは複数の更新ファイルを含む送信 (データ) および n 個のオーセンティケータを使用して作成された MAC のベクトルとを受け取る。 n 個のオーセンティケータは、UMAC メッセージ認証コードとすることができる。

【0009】

他の実施形態では、オーセンティケータは、サーバーレス分散ファイルシステムにより共有される複数の証明書のうちの 1 つであり、複数の証明書のそれぞれの証明書にシリアル番号が振られ、分散ファイルシステムは順序の狂っている証明書に対する認証を拒絶することで証明書の選択的選択を防止することができる。

【0010】

一実施形態による方法では、ファイル書き込みを、認証されているチャネルの外部にあるフォールトトレラントディレクトリグループに送信し、少なくとも 1 つのオーセンティケータで 1 つまたは複数のファイル書き込みを識別する。少なくとも 1 つのオーセンティケータでは、ディレクトリグループが 1 つまたは複数のファイル書き込みを許可するために少なくとも 1 つの秘密鍵を再構成する必要がある。秘密鍵を再構成するために必要なフォールトトレラントディレクトリグループ内のメンバーの数は、フォールトトレラントディレクトリグループで許容できるフォールトの個数を少なくとも 3 倍して 1 を加えた値である。

【0011】

他の実施形態は、認証されたチャネルの外部にあるフォールトトレラントディレクトリグループへのファイル書き込みの方法を対象とする。この方法では、クライアントマシンにログを作成し、このログはファイル書き込みを含み、メッセージ認証コードと関連付けられている秘密鍵によりセキュリティが保護されているメッセージ認証コードによりこのログを認証する。秘密鍵を再構成するには、フォールトトレラントディレクトリグループの一定数のメンバーが秘密鍵の割り当て分を提供する必要がある、メンバーの数は、少なくとも 1 にフォールトトレラントディレクトリグループの許容可能なフォールトの個数を加えた数である。この方法はさらに、認証されたチャネルの外部でログを送信することを含む。

【0012】

一実施形態は、認証されたチャネルの外部にあるフォールトトレラントディレクトリグ

10

20

30

40

50

ループへのファイル書き込みを可能にするコンピュータ実行可能命令を格納するコンピュータ可読媒体を対象とし、コンピュータ実行可能命令により、ファイル書き込みを含むログを作成する作業などの処理を実行する。ログは、メッセージ認証コードに関連付けられている秘密鍵によりセキュリティ保護されているメッセージ認証コードにより認証される。秘密鍵を再構成するには、フォールトトレラントディレクトリグループの $3f + 1$ の数のメンバーが秘密鍵の割り当て分を提供する必要がある、数 f は、フォールトトレラントディレクトリグループの許容可能なフォールトの数である。命令により、さらに、クライアントマシンは認証されたチャネルの外部でログを送信することができる。本発明の他の特徴および利点は、添付図を参照して進められている例示的な実施形態の以下の詳細な説明から明白であろう。

10

【0013】

付属の請求項で本発明の特徴を詳細に述べているが、本発明は、その目的および利点とともに、添付図面に関する以下の詳細な説明から最もよく理解できよう。

【0014】

類似のコンポーネントおよび／または図を参照するため文書全体を通して同じ番号を使用する。

【図面の簡単な説明】**【0015】**

【図1】 サーバーレス分散ファイルシステムをサポートするネットワーク環境例の図である。

20

【図2】 分散ファイルシステムに参加している図1のデバイスのうちの1つを表すコンピューティングデバイス例の論理コンポーネントの図である。

【図3】 図1の分散ファイルシステムを実装するために使用される、より一般的なコンピュータ環境の図である。

【図4】 複数のサブツリーを持つ名前空間ルートを含む階層型名前空間の例を示す図である。

【図5】 サーバーレス分散ファイルシステム内のファイルおよび対応するディレクトリエントリの格納例を示す図である。

【図6】 サーバーレス分散ファイルシステム内でファイルを格納するプロセスの例を説明する流れ図である。

30

【図7】 サーバーレス分散ファイルシステム内のファイルおよび証明書の格納例を示す図である。

【図8】 委任証明書を使用するプロセスの例を説明する流れ図である。

【図9】 サーバーレス分散ファイルシステム内で秘密鍵を使用するファイルの格納例を示す図である。

【図10】 秘密鍵共有の実施形態による更新ファイルをアップロードするプロセスの例を説明する流れ図である。

【図11】 秘密鍵共有の実施形態により更新ファイルを行えるようにするかどうかを決定するプロセスの例を説明する流れ図である。

【図12】 オーセンティケータのベクトルの実施形態により更新ファイルを行えるようにするかどうかを決定するプロセスの例を説明する流れ図である。

40

【図13】 ファイルの検証を行うプロセスの例を説明する流れ図である。

【図14】 再生攻撃を防止するプロセスの例を説明する流れ図である。

【発明を実施するための最良の形態】**【0016】**

サーバーレス分散ファイルシステム 図1は、サーバーレス分散ファイルシステムをサポートするネットワーク環境100の例を示す図である。4つのクライアントコンピューティングデバイス102、104、106、および108は、データ通信ネットワーク110を介して共に結合されている。4つのコンピューティングデバイスが示されているが、ネットワーク環境100に組み込むことができる数はそれと異なってもよい（4つ

50

よりも多くても少なくてもよい)。

【0017】

ネットワーク110は、さまざまなデータ通信ネットワークを表す。ネットワーク110は、公開部分(たとえば、インターネット)、さらにプライベート部分(たとえば、社内ローカルエリアネットワーク(LAN))、さらに公開部分とプライベート部分の組合せを含むことができる。ネットワーク110は、有線媒体および無線媒体の両方を含むさまざまな従来の通信媒体のうち1つまたは複数を使用して実装することができる。さまざまな通信プロトコルを使用して、公開プロトコルおよび専用プロトコルの両方を含めてネットワーク110を介してデータを通信することができる。このようなプロトコルの例として、TCP/IP、IPX/SPX、NetBEUIなどがある。

10

【0018】

コンピューティングデバイス102~108は、さまざまなコンピューティングデバイスを表し、それぞれのデバイスは同じであっても異なってもよい。たとえば、デバイス102~108として、デスクトップコンピュータ、ラップトップコンピュータ、ハンドヘルドまたはポケットコンピュータ、パーソナルデジタルアシスタント(PDA)、携帯電話、インターネット家電、家庭用電化製品、ゲーム機などがある。

【0019】

デバイス102~108のうち2つまたはそれ以上が、サーバーレス分散ファイルシステムを実装する処理をする。サーバーレス分散ファイルシステムに参加している実際のデバイスは、時間のたつうちに変わり、新しいデバイスがシステムに加わったり、システムから他のデバイスが除去されたりする。分散ファイルシステムを実装する(参加する)各デバイス102~106は、ローカルストレージまたは分散ストレージとして使用するために割り当てられた大容量ストレージデバイス(たとえば、ハードディスクドライブ)の一部を備える。ローカルストレージは、ユーザーが、分散ファイルシステム構造ではなく、自分のローカルマシンに格納したいデータに使用される。分散ストレージ部分は、デバイス(または他のデバイス)のユーザーが分散ファイルシステム構造内に格納したいデータに使用される。

20

【0020】

図1の図の例では、ネットワーク110に接続されているいくつかのデバイスは、分散部分とローカル部分の両方を含む1つまたは複数の大容量ストレージデバイスを備える。分散ストレージまたはローカルストレージに割り当てられる量はデバイスによって異なる。たとえば、デバイス102は、ローカル部分122に比べて分散システム部分120の方が割り当てられるパーセンテージが高く、デバイス104では、分散システム部分124はローカル部分126とほぼ同じであり、デバイス106は、ローカル部分130に比べて分散システム部分128の方が割り当てられるパーセンテージは低い。複数の部分へのストレージ分割は、ストレージデバイス毎に行われ(たとえば、一方のハードドライブを分散システムで使用するよう指定し、他方のハードドライブをローカル専用として指定する)、および/または単一ストレージデバイスで行うことも可能である(たとえば、一方のハードドライブの一部を分散システムで使用するよう指定し、他方の部分をローカル使用に指定する)。分散ストレージまたはローカルストレージに割り当てられる量は時間によっても異なる。コンピューティングデバイス108などのネットワーク110に接続されている他のデバイスは、分散ファイルシステムをどれも実装することはできず、したがって、大容量ストレージデバイスを分散システム用に割り当てられない。したがって、デバイス108はローカル部分132しか持たない。

30

40

【0021】

分散ファイルシステム150は、異なるコンピューティングデバイス102~106にファイルの1つまたは複数のコピーを格納する処理をする。新しいファイルがコンピュータのユーザーによって作成された場合、ユーザーはファイルを自分のコンピューティングデバイスのローカル部分に格納するか、または別法として、分散ファイルシステム内に格納するかを選択することができる。ファイルが分散ファイルシステム150に格納される

50

場合、ファイルはデバイス102～106のうち1つまたは複数のデバイスの大容量ストレージデバイスの分散システム部分に格納されることになる。ファイルを作成したユーザーは、通常、デバイス102～106のどれにファイルを格納するかを制御することはできず、またデバイス102～106のどれにファイルが格納されているかも知ることができない。さらに、ファイルの複製は、通常、保存されるため、ファイルが保存されているコンピューティングデバイス102～106のうちの1つが使用不可能になったとしても（たとえば、停電、故障などが生じて）、ユーザーは後からファイルを取り出すことができる。

【0022】

分散ファイルシステム150は、デバイス102～106のそれぞれに搭載されている1つまたは複数のコンポーネントにより実装され、そのため、中央サーバーがファイルシステムの調整を行う必要がない。これらのコンポーネントは、特定のファイルがどこに格納されるか、異なるデバイスへの格納のためファイルのコピーをいくつ作成するかなどを判別する処理をする。厳密にどのファイルがどのデバイスに格納されるかは、分散ファイルシステム内のデバイスの個数、各デバイスからファイルシステムに割り当てられた記憶領域、保存するファイルのコピー数、暗号でセキュリティ対策した乱数、デバイス上にすでに格納されているファイルの個数など、さまざまな要因に左右される。したがって、分散ファイルシステムを使用すると、ユーザーは、ファイルが他のどのコンピューティングデバイスに格納されているかを正確に知ることなくファイルを（さらにはフォルダまたはディレクトリも）作成し、アクセスすることができる。

【0023】

分散ファイルシステム150は、システム150内のコンピュータの増大に合わせて拡張可能なように設計されている。システム150のデバイス上のコンポーネントによって使用されるプロトコルおよびデータ構造は、設計上、システム内のコンピュータの台数に比例しないため、コンピュータ台数の増大に合わせて容易に拡張することができる。

【0024】

ファイルシステムによって格納されるファイルは、さまざまなデバイス102～106の間に分散され、暗号化の形態で格納される。新規ファイルが作成される場合、その新規ファイルが作成されるデバイスは、ファイルを格納するため他のデバイスに伝達するのに先立ってファイルを暗号化する。新しいファイルのディレクトリエントリ（ファイル名を含む）は、さらに、格納のため他のデバイスにも伝達されるが、暗号化されたファイルが格納されているデバイスと同じデバイスでなくてよいし、（通常は、同じでない）。さらに、新しいフォルダまたはディレクトリが作成される場合、ディレクトリエントリ（フォルダ名またはディレクトリ名を含む）も格納のため他のデバイスに伝達される。本明細書で使用しているように、ディレクトリエントリは、ファイル名とディレクトリ（またはフォルダ）名の両方を含む、ファイルシステムディレクトリに追加することができるエントリを指す。

【0025】

分散ファイルシステム150は、不正ユーザーがデバイス102～106のうちの1つに格納されているデータを読み込めないように設計されている。したがって、デバイス102によって作成され、デバイス104に格納されているファイルは、デバイス104のユーザーからは読み出すことができない（そのユーザーが読み取る権限を与えられていない限り）。このようなセキュリティを実装するために、ファイルの内容だけでなくディレクトリエントリ内のすべてのファイル名およびディレクトリ名も暗号化し、認証ユーザーのみに暗号解読鍵を提供する。したがって、デバイス104は、デバイス102によって作成されたファイルを格納することができるが、デバイス104のユーザーがファイルの認証ユーザーでない場合に、デバイス104のユーザーは、ファイルの内容またはそのディレクトリエントリ内のファイル名のいずれかを暗号解読することはできない（また、読み出すこともできない）。

【0026】

分散ファイルシステム150は、1つまたは複数の名前空間ルートを持つとともに、各名前空間ルートの配下に複数のサブツリーを置く、階層型格納構造を採用している。異なるサブツリーの管理はコンピュータの異なるグループに委託されるので、名前空間ルートまたは特定のサブツリーを管理するコンピュータに負担がかかり過ぎることはない。

【0027】

分散ファイルシステム150はさらに、ファイルおよびそれらのファイルに対応するディレクトリエントリの格納を異なる方法で管理する。システム150に格納されるファイルは複製され、システム内の複数の異なるコンピュータに保存される。さらに、そのファイルに対するディレクトリエントリが作成され、これもまた、システム内の複数の異なるコンピュータ上に保存される。保存されるディレクトリエントリのコピーは保存されるファイルコピーの個数よりも多い。一実施形態では、ディレクトリエントリは、後述の Byzantine-fault-tolerant グループの一部であるコンピュータ上に格納される。

10

【0028】

分散ファイルシステム150は、さらに、ディレクトリおよびファイルの読み書きをする人の管理を行うディレクトリおよびファイルロックメカニズムも採用する。Byzantine グループ内のコンピュータで使用する場合、採用したこのロックメカニズムは、後述のように、ディレクトリグループによるアクションが行われなくてもローカルで実行できるオペレーションの数を増やすことによりパフォーマンスを向上させようとする。

【0029】

20

分散ファイルシステム150内のすべてのコンピュータ102～106には、3つの機能がある、つまり、ローカルユーザーに対するクライアントになれる、システム内に格納されているファイルの暗号化されたコピーに対するリポジトリになれる、および1つまたは複数のディレクトリを保持するコンピュータのグループのメンバーになれるという3つの機能がある。

【0030】

一般に、コンピュータ102～106のユーザーが所与のディレクトリ内のファイルを開く場合、コンピュータは Byzantine-fault-tolerant プロトコルを使用してそのディレクトリ（「Byzantine group」または「directory group」という名前）をまとめて管理する一組のコンピュータ群に要求を送信する。Byzantine グループがコンピュータに対してファイルロックを許可すると、コンピュータはファイルに対してローカル更新を実行して（書き込みロックである場合）、これらの更新を Byzantine グループにプッシュして戻すことができる。コンピュータが最近このファイルにアクセスしている場合、おそらくローカルキャッシュ内にファイルコンテンツの暗号化されたコピーが置かれているであろうから、キャッシュされているコピーを取り出して暗号解読するだけで、そのファイルの読み書きを開始することができる。最近、ファイルの現行バージョンにアクセスしていなかった場合、コンピュータはそのファイルを格納しているコンピュータの1つからファイルの暗号化されているコピーを取り出す。コンピュータが保持している現在のコピーに関する情報は、ロック許可とともに Byzantine グループにより与えられ、ファイル格納コンピュータのうちの1つまたは複数がダウンしている場合には、コンピュータは別のファイル格納コンピュータからファイルを取り出す。Byzantine グループはさらにファイル内容を暗号でハッシュし、コンピュータはその内容を使用して、フェッチするファイルの妥当性を確認する。

30

40

【0031】

ファイル暗号化

ファイルの暗号化には、「収束暗号化 (convergent encryption)」と呼ばれる技術を使用する。収束暗号化には、以下の2つの特性がある。まず第1に、2つまたはそれ以上の暗号化可能オブジェクトが同一である場合、異なる暗号化鍵を使用してオブジェクトを暗号化し、個々の暗号オブジェクトを提供するとしても、暗号化鍵

50

にアクセスして暗号化可能オブジェクトが同一であることを暗号オブジェクトを調べることによって判別する必要はない。第2に、2つまたはそれ以上の暗号化可能オブジェクトが同一であるが、異なる暗号化鍵で暗号化されている場合、暗号オブジェクトすべてを格納するために必要な領域全体は、単一の暗号化可能なオブジェクトを格納するのに必要な領域に、それぞれ異なる暗号化鍵の一定量の記憶を加えた物に比例する。

【0032】

一般に、収束暗号化によれば、ファイルF（または他の種類の暗号化可能オブジェクト）は最初に、一方向ハッシュ関数h（たとえば、SHA、MD5など）を使用してハッシュを実行し、ハッシュ値h（F）を出力する。次に、ハッシュ値を鍵とする対称暗号（たとえば、RC4、RC2など）、つまり、 $E_{h(F)}(F)$ を使用してファイルFを暗号化する。次に、読み取りアクセス制御エントリを、暗号化されたファイルへの読み取りアクセス権が与えられている認証ユーザー毎に作成する。書き込みアクセス制御は、ファイルに対するディレクトリエントリを格納するディレクトリサーバーが扱う。読み取りアクセス制御エントリは、ファイルのハッシュ値h（F）を任意個数の鍵 K_1 、 K_2 、...、 K_m で暗号化し、 $E_{K_1}(h(F))$ 、 $E_{K_2}(h(F))$ 、...、 $E_{K_m}(h(F))$ を得ることにより、形成される。一実施形態では、それぞれの鍵Kは、非対称暗号（たとえば、RSA）の場合の公開／私的鍵ペアのユーザーの持つ公開鍵である。

【0033】

収束暗号化では、ファイルの暗号化された1つのバージョンが、サーバーレス分散ファイルシステム150間で格納され、複製される。ファイルの暗号化バージョンとともに、アクセス権のある認証ユーザーの数に応じて1つまたは複数のアクセス制御エントリが格納される。したがって、分散ファイルシステム150内のファイルは以下の構造を持つ。

$[E_{h(F)}(F), \langle E_{K_1}(h(F)) \rangle, \langle E_{K_2}(h(F)) \rangle, \dots, \langle E_{K_m}(h(F)) \rangle]$

【0034】

収束暗号化の1つの利点は、暗号化されたファイルはファイルシステムによって評価され、暗号解読に頼らずとも（したがって、暗号化鍵を知らなくても）、他のファイルと同一であるかどうかを判別することができるという点である。不要な重複ファイルを削除するには、認証ユーザーアクセス制御エントリを残りのファイルに追加する。他の利点として、暗号化されたファイルは場合によっては数ギガバイトになることに比べ、アクセス制御エントリは数バイト程度でサイズが非常に小さいという点である。そのため、それぞれのファイルに格納されているオーバーヘッド情報の量は少ない。したがって、ファイルを格納するために使用する全領域は単一の暗号化されたファイルを格納するために必要な領域にファイルの各追加読み込み認証ユーザーの記憶域の一定量を加えた物に比例するという特性を持つ。

【0035】

収束暗号化の詳細については、Microsoft Corporationに譲渡された、2000年5月5日にDouceur et al. の名前で出願された「Encryption Systems and Methods for Identifying and Coalescing Identical Objects Encrypted with Different Keys」という表題の米国特許出願を参照されたい。この出願は、参照により本発明に組み込まれている。

【0036】

ディレクトリエントリの暗号化

「排他的(exclusive)暗号化」と呼ばれるプロセスを使用してディレクトリエントリ内のファイルおよびディレクトリ名を暗号化する。排他的暗号化を使用することにより、ディレクトリエントリ内のファイルおよびディレクトリ名を暗号化形式で格納し、不正ユーザーがファイルまたはディレクトリの名前に基づいて情報を不正に取得するのを防止することができる。さらに、排他的暗号化には、以下の3つの特性がある。まず、ディレクトリ内の暗号化されたエントリのどの2つをとっても暗号解読した場合に同じ名前が得られることはない。第2に、ディレクトリ内の暗号化されたすべてのエントリは、暗号解読により構文的に正当な名前が得られる。第3に、ディレクトリを保持しているデ

10

20

30

40

50

ィレクトリグループは、エントリの平文の名前へのアクセス権を持たない。したがって、ファイルシステム150では、ディレクトリ内のどの2つのエントリも同じ名前を暗号化した物ではないこと、および、ディレクトリ内のすべてのエントリは、構文的に正当な名前を暗号化した物であることを保証することができ、それと同時に、ディレクトリを保持しているデバイスがエントリの平文の名前へのアクセス権を持たないことも保証される。

【0037】

一般に、排他的暗号化によれば、平文名（ディレクトリエントリ内のファイル名またはディレクトリ名）は新しい名前にマッピングされる。マッピングされた名前は、任意選択により、ディケシファイドされた（decasified）（大文字と小文字を区別しない）名前および対応する文字小文字区別の情報に変換され、重複名検出で大文字と小文字の区別をなくすることができる。その後、マッピングされた（および任意選択によりディケシファイドされた）名前が符号化され、暗号化される。この暗号化された名前（および任意選択により、随伴する大文字小文字区別情報）は、ディレクトリエントリを管理する役割を持つディレクトリグループに転送される（たとえば、後述のように、パス名に基づいて）。

【0038】

排他的暗号化の詳細については、本願出願人に譲渡された、2001年1月17日にDouceur et al. の名前で出願された「Exclusive Encryption for a Secure Directory Service」という表題の米国特許出願を参照されたい。この出願は、参照により本発明に組み込まれている。

【0039】

ファイル形式

図1のサーバーレス分散ファイルシステム150のファイル形式は、一次データストリームとメタデータストリームの2つの部分からなる。一次データストリームは、複数のブロックに分けられたファイルを含む。各ブロックは、対称暗号（たとえば、RC4）および暗号化鍵としてブロックのハッシュを使用して暗号化される。メタデータストリームは、ヘッダ、一次データストリーム内の暗号化されたブロックにインデックスを付けるための構造、および何らかのユーザー情報を含む。

【0040】

インデックス付けツリー構造により、ブロックのそれぞれのリーフノードが定義される。各リーフノードは、関連付けられているブロックの暗号解読に使用されるアクセス値および他のブロックとは無関係に暗号化されたブロックの検証を行うために使用される検証値からなる。一実施形態では、アクセス値を形成するために、ファイルブロックをハッシュし、対称暗号およびランダム生成鍵を使用して、その結果得られたハッシュ値を暗号化する。次に、鍵は、非対称暗号（たとえば、RSA）および暗号化鍵としてユーザーの公開鍵を使用して暗号化される。検証値は、一方向ハッシュ関数（たとえば、SHA）を使用して、関連付けられた暗号化されているブロックをハッシュすることにより形成される。

【0041】

ファイルのサイズに応じて、インデックス付け構造は、リーフノードをツリーブロックにグループ化し、それぞれのツリーブロックのハッシュ値を計算することにより形成される中間ノードを含むことができる。これらの中間ノードを再び、ブロックにセグメント化し、各ブロックをハッシュして、次のノードを形成することができる。これは、ルートノードに到達するまで、好きなだけ繰り返すことができる。さらに、ルートノードをハッシュし、ハッシュ値をメタデータヘッダおよびユーザー情報とともに使用してファイル全体の検証値を出力する。一実施形態では、ファイル検証値全体にユーザーの署名を入れる。別法として、そのような署名なしでファイルを構成することもできる。

【0042】

ファイル形式は、ランダム生成鍵またはユーザー鍵を知らなくても個別ファイルブロックの検証を行うことができる形式である。ファイルのブロックを検証するには、ファイル

システム側で任意選択により、ファイル全体の検証値（もし存在していれば）に関する署名を評価し、ファイル全体の検証値がルートブロック、メタデータヘッダ、およびユーザー情報のハッシュに一致しているかどうかを確認し、その後、ツリーをトラバースして検証すべきターゲットブロックに関連付けられた適切なリーフノードまでたどる。ファイルシステムは、ターゲットブロックをハッシュし、ハッシュがリーフノードに格納されているアクセス値に一致すれば、ブロックは信頼できる。

【0043】

ファイル形式はさらに、他のブロックに干渉をすることなく、個々のブロックからの読み込みおよび個々のブロックへの書き込みをサポートする。ファイル形式は、さらに、非データの広範な領域を持つ疎ファイルでは役に立つ。

【0044】

ファイル形式の詳細については、本願出願人に譲渡された、2001年3月21日にBolosky et al. の名前で出願された「On-Disk File Format for a Serverless Distributed File System」という表題の米国特許出願を参照されたい。この出願は、参照により本発明に組み込まれている。

【0045】

コンピューティングデバイスのアーキテクチャ

図2は、分散ファイルシステム150に参加している図1のデバイス102～106のうちの1つを表すコンピューティングデバイス200の例の論理コンポーネントの図である。コンピューティングデバイス200は、サーバーコンポーネント202、クライアントコンポーネント204、メモリ206、大容量ストレージデバイス208、および分散ファイルシステムインターフェイス210を備える。コンピューティングデバイス200はさらに、追加コンポーネント（たとえば、プロセッサ）を備えるのが普通であるが、これらの追加コンポーネントは図面をわかりやすくするため図2には示されていない。さまざまなハードウェアおよびソフトウェアコンポーネントを使用するコンピュータアーキテクチャ全般については、図3を参照しながら以下で説明する。

【0046】

メモリ206としては、RAM、ROM、フラッシュメモリなどのさまざまな従来の揮発性および／または不揮発性メモリを使用することができる。大容量ストレージデバイス208としては、磁気ディスク、光ディスク、フラッシュメモリなどのさまざまな従来の不揮発性ストレージデバイスを使用することができる。大容量ストレージデバイス208は、パーティション分割により、分散システム部分とローカル部分に分ける。図2には1つの大容量ストレージデバイス208しか示されていないが、コンピューティングデバイス200は、複数のストレージデバイス208（種類は異なっている、あるいはすべて同じ種類でもよい）を備えることができる。

【0047】

コンピューティングデバイス200は、サーバーレス分散ファイルシステム内で使用されるようになっており、したがって、サーバーコンポーネント202とクライアントコンポーネント204の両方を含む。サーバーコンポーネント202は、デバイス200がストレージデバイス208に格納されている（または格納される予定の）ファイルまたはディレクトリエントリを伴う要求に応答している場合に要求を処理するが、クライアントコンポーネント204は、分散ファイルシステムに格納されている（または格納される予定の）ファイルまたはディレクトリに関するデバイス200による要求の発行を処理する。クライアントコンポーネント204およびサーバーコンポーネント202は、互いに独立に処理する。したがって、サーバーレス分散ファイルシステム150のせいで、クライアントコンポーネント204によって格納されるファイルがサーバーコンポーネント202により大容量ストレージデバイス208に格納される状況が生じる可能性がある。

【0048】

クライアントコンポーネント204は、コンピューティングデバイス150に代わって、ファイルおよびディレクトリの作成、格納、取り出し、読み込み、書き込み、修正、お

10

20

30

40

50

よび検証を行うため、インターフェイス 210 とともに、サーバーレス分散ファイルシステム 150 へのアクセスを管理する、格納および取り出し制御モジュール 220 を備える。制御モジュール 220 は、特定のファイルまたはディレクトリを管理する役割を持つディレクトリグループを識別するディレクトリグループ検索モジュール 222、ファイルを暗号化するファイル暗号化モジュール 226、ディレクトリエントリ内のファイル名およびディレクトリ名を暗号化するディレクトリ暗号化モジュール 228 を使用する。これらのモジュールの処理について、以下で詳述する。

【0049】

サーバーコンポーネント 202 は、分散システム制御モジュール 250、重複識別器 252、およびサブツリー委任モジュール 254 を備える。分散システム制御モジュール 250 は、暗号化ファイル 240 へのアクセスを管理する。これは、大容量ストレージデバイス 208 と通信し、暗号化ファイル 240 の格納および取り出しを行う。分散システム制御モジュール 250 は、さらに、コンピューティングデバイス 200 内に格納されている（あるいは、サーバーレス分散ファイルシステム内のどこかに格納されている）メモリ 206 および／または大容量ストレージデバイス 208 内のディレクトリエントリ（図に示されていない）の記録を保持する。サブツリー委任モジュール 254 は、後述のように、他のディレクトリグループにサブツリーを委任する処理をする。

【0050】

重複識別器 252 は、分散ファイルシステム内の同一の暗号化ファイルを識別するのに使用される。重複識別器 252 がフォールトトレラントを目的とする意図的な複製ではない重複を見つけた場合、重複識別器 252 が制御モジュール 250 に通知し、重複ファイルを除去し、除去されたファイルへのアクセス制御エントリを残りのファイルに追加する。

【0051】

図 3 は、分散ファイルシステムを実装するために使用される、より一般的なコンピュータ環境 300 の図である。コンピュータ環境 300 は、コンピューティング環境の一例にすぎず、コンピュータおよびネットワークアーキテクチャの使用または機能の範囲に関する限定を示唆しないことを意図している。またコンピュータ環境 300 は、コンピュータ環境例 300 に示されているコンポーネントの包含（または除外）またはコンポーネントの結合または組合せに関する要件があると解釈してはならない。

【0052】

コンピュータ環境 300 は、コンピュータ 302 の形の汎用コンピューティングデバイスを備える。コンピュータ 302 のコンポーネントは、それだけに限らないが、1つまたは複数のプロセッサまたは処理ユニット 304、システムメモリ 306、およびシステムメモリ 306 にプロセッサ 304 をはじめとするさまざまなシステムコンポーネントを結合するシステムバス 308 を備える。

【0053】

システムバス 308 は、メモリバスまたはメモリコントローラ、周辺機器バス、グラフィック専用高速ポート、およびさまざまなバスアーキテクチャを使用するプロセッサまたはローカルバスを含む、いくつかのタイプのバス構造のうちの 1つまたは複数のバス構造を表している。たとえば、このようなアーキテクチャとしては、Industry Standard Architecture (ISA) バス、Micro Channel Architecture (MCA) バス、Enhanced ISA (EISA) バス、Video Electronics Standards Association (VESA) ローカルバス、および Mezzanine バスとも呼ばれる Peripheral Component Interconnect (PCI) バスがある。

【0054】

コンピュータ 302 は通常、さまざまなコンピュータ可読媒体を含む。このようなメディアは、コンピュータ 302 からアクセス可能な使用可能媒体であればよく、揮発性および不揮発性媒体、取り外し可能および取り外し不可能媒体の両方を含む。

10

20

30

40

50

【0055】

システムメモリ306は、ランダムアクセスメモリ（RAM）310などの揮発性メモリおよび／または読み取り専用メモリ（ROM）312などの不揮発性メモリの形式のコンピュータ可読媒体を備える。起動時などにコンピュータ302内の要素間の情報伝送を助ける基本ルーチンを含む基本入出力システム（BIOS）314は、ROM 312に格納される。通常、RAM 310には、処理ユニット304に直接アクセス可能な、および／または現在操作されているデータおよび／またはプログラムモジュールを格納する。

【0056】

コンピュータ302はさらに、その他の取り外し可能／取り外し不可能な揮発性／不揮発性コンピュータ記憶媒体を備えることもできる。たとえば、図3は、取り外し不可能な不揮発性磁気媒体の読み書きを行うハードディスクドライブ316（図には示されていない）、取り外し可能な不揮発性磁気ディスク320（たとえば、「フロッピー（登録商標）ディスク」）の読み書きを行う磁気ディスクドライブ318、およびCD-ROM、DVD-ROM、またはその他の光媒体などの取り外し可能な不揮発性光ディスク324の読み書きを行う光ディスクドライブ322を示している。ハードディスクドライブ316、磁気ディスクドライブ318、および光ディスクドライブ322はそれぞれ、1つまたは複数のデータ媒体インターフェイス326によりシステムバス308に接続されている。別法として、ハードディスクドライブ316、磁気ディスクドライブ318、および光ディスクドライブ322はそれぞれ、1つまたは複数のインターフェイス（図に示されていない）によりシステムバス308に接続することもできる。

【0057】

ディスクドライブおよび関連コンピュータ可読媒体は、コンピュータ302用のコンピュータ可読命令、データ構造、プログラムモジュール、およびその他のデータを格納する不揮発性ストレージを備える。例では、ハードディスク316、取り外し可能磁気ディスク320、および取り外し可能光ディスク324を示しているが、磁気カセットまたはその他の磁気ストレージデバイス、フラッシュメモリカード、CD-ROM、デジタル多目的ディスク（DVD）またはその他の光ストレージ、ランダムアクセスメモリ（RAM）、読み取り専用メモリ（ROM）、電氣的消去可能プログラム可能読み取り専用メモリ（EEPROM）などのコンピュータからアクセス可能なデータを格納できる他の種類のコンピュータ可読媒体が、例のコンピューティングシステムおよび環境の実装に使用することも理解されるであろう。

【0058】

ハードディスク316、磁気ディスク320、光ディスク324、ROM 312、および／またはRAM 310には、たとえば、オペレーティングシステム326、1つまたは複数のアプリケーションプログラム328、その他のプログラムモジュール330、およびプログラムデータ332などのプログラムモジュールをいくつでも格納できる。このようなオペレーティングシステム326、1つまたは複数のアプリケーションプログラム328、その他のプログラムモジュール330、およびプログラムデータ332（またはその何らかの組合せ）のそれぞれにより、分散ファイルシステムをサポートする常駐コンポーネントの全部または一部を実装することができる。

【0059】

ユーザーは、キーボード334およびポインティングデバイス336（たとえば、「マウス」）などの入力デバイスを介してコマンドおよび情報もコンピュータ302に入力することができる。他の入力デバイス338（図に特に示されてはいない）としては、マイク、ジョイスティック、ゲームパッド、衛星放送受信アンテナ、シリアルポート、スキャナ、および／または類似の物などがある。これらの入力デバイスやその他の入力デバイスは、システムバス308に結合されている入出力インターフェイス340を介して処理ユニット304に接続されるが、パラレルポート、ゲームポート、またはユニバーサルシリアルバス（USB）などの他のインターフェイスおよびバス構造により接続することもで

きる。

【0060】

モニタ342やその他の種類の表示デバイスも、ビデオアダプタ344などのインターフェイスを介してシステムバス308に接続できる。モニタ342に加えて、他の出力周辺デバイスは、入出力インターフェイス340を介してコンピュータに接続することができるスピーカ（図に示されていない）およびプリンタ346などのコンポーネントを備えることができる。

【0061】

コンピュータ302は、リモートコンピューティングデバイス348などの1つまたは複数のリモートコンピュータへの論理接続を使用してネットワーク環境で処理することができる。たとえば、リモートコンピューティングデバイス348は、パーソナルコンピュータ、携帯型コンピュータ、サーバー、ルーター、ネットワークコンピュータ、ピアデバイス、またはその他の共通ネットワークノードなどが考えられる。リモートコンピューティングデバイス348は、コンピュータ302に関して本明細書で説明している要素および特徴の多くまたはすべてを備えることができる携帯型コンピュータとして示されている。

10

【0062】

コンピュータ302とリモートコンピュータ348との間の論理接続は、ローカルエリアネットワーク（LAN）350および一般的なワイドエリアネットワーク（WAN）352として示されている。このようなネットワーキング環境は、事務所、企業規模のコン

20

【0063】

LANネットワーキング環境に実装する場合、コンピュータ302はネットワークインターフェイスまたはアダプタ354を介してローカルネットワーク350に接続される。WANネットワーキング環境に実装する場合、コンピュータ302は通常、モデム356またはワイドネットワーク352上で通信を確立するためのその他の手段を備える。モデム356は、コンピュータ302に内蔵でも外付けでもよいが、入出力インターフェイス340またはその他の適切なメカニズムを介してシステムバス308に接続できる。図に示されているネットワーク接続は例であり、コンピュータ302と348との間に通信リンクを確立する他の手段を使用できることは理解されるであろう。

30

【0064】

ネットワーク環境では、コンピューティング環境300、コンピュータ302に関して示されているプログラムモジュール、またはその一部とともに示されている物などをリモートのメモリストレージデバイスに格納することができる。たとえば、リモートアプリケーションプログラム358は、リモートコンピュータ348のメモリデバイスに常駐している。説明のため、アプリケーションプログラムおよび、オペレーティングシステムなどの他の実行可能なプログラムコンポーネントはここでは離散ブロックに示されているが、このようなプログラムおよびコンポーネントはさまざまな時期に、コンピューティングデバイス302の異なるストレージコンポーネント内に置かれ、コンピュータのデータプロセッサにより実行されることは理解されるであろう。

40

【0065】

分散ファイルシステム150の一実施形態は、1つまたは複数のコンピュータまたは他のデバイスによって実行される、プログラムモジュールなどのコンピュータ実行可能命令の一般的文脈において説明できる。一般に、プログラムモジュールは、特定のタスクを実行するか、または特定の抽象データ型を実装するルーチン、プログラム、オブジェクト、コンポーネント、データ構造などを含む。通常、さまざまな実施形態で、必要に応じて説明されているようにプログラムモジュールの機能を組み合わせるか、または分散させることができる。

【0066】

暗号化ファイルのファイル形式の実装は、何らかの形式のコンピュータ可読媒体に格納

50

または媒体を使って送信することができる。コンピュータ可読媒体は、コンピュータからアクセスできる使用可能な媒体とすることができる。たとえば、コンピュータ可読媒体は、「コンピュータ記憶媒体」および「通信媒体」を含むことができるが、それらには限定されない。

【0067】

「コンピュータ記憶媒体」は、コンピュータ可読命令、データ構造、プログラムモジュール、またはその他のデータなどの情報を格納する方法または技術で実装される揮発性および不揮発性、取り外し可能および取り外し不可能媒体を含む。コンピュータ記憶媒体としては、RAM、ROM、EEPROM、フラッシュメモリまたはその他のメモリ技術、CD-ROM、デジタル多目的ディスク(DVD)またはその他の光ディスクストレージ、磁気カセット、磁気テープ、磁気ディスクストレージまたはその他の磁気ストレージデバイス、または所望の情報を格納するために使用することができ、コンピュータによりアクセスできるその他の媒体があるが、それらには限定されない。

10

【0068】

「通信媒体」は、通常、コンピュータ可読命令、データ構造、プログラムモジュール、またはその他のデータを搬送波またはその他のトランスポートメカニズムなどの変調データ信号で実現する。通信媒体は、任意の情報配信媒体も含む。「変調データ信号」という用語は、信号内の情報を符号化する方法でその特性のうち1つまたは複数を設定または変更した信号を意味する。たとえば、通信媒体としては、有線ネットワークまたは直接配線接続などの有線媒体、および、音響、RF、赤外線、およびその他の無線媒体などの無線媒体があるが、それらには限定されない。上記のいずれかの組合せもコンピュータ可読媒体の範囲に収まる。

20

【0069】

階層型格納構造

分散ファイルシステム150は、1つまたは複数の名前空間ルートを含み、それぞれのルートがディレクトリまたはフォルダの1つまたは複数のサブツリーをサポートすることができる、それぞれのサブツリーは1つまたは複数の追加サブツリーをサポートすることができる、階層型ファイル格納構造を採用している。ディレクトリは、シミュレートされたファイルフォルダとみなすことができ、0個またはそれ以上のファイルおよび/または0個またはそれ以上の他のディレクトリを保持することができる。サブツリーは、1つまたは複数のディレクトリのことで、ルート（さらにこれは名前空間ルートも含むことができる）を含み、サブツリーのルートからそのサブツリーのすべてのメンバーへの経路がそのサブツリー自体の中にあるという特性を持つ。図4は、ディレクトリA、B、C、D、E、F、G、H、J、I、M、K、およびLを含む複数のサブツリーを備える名前空間ルートを含む階層型名前空間400の例を示している。通常、さらに多くのディレクトリが名前空間ルートのサブツリーに含まれるが、説明しやすくするため、図4にはごくわずかのディレクトリしか示されていない。

30

【0070】

それぞれのサブツリーは、ディレクトリグループと呼ばれる1つまたは複数のコンピュータのグループにより管理される。本明細書では主に、サブツリーを管理するディレクトリグループとして説明しているが、別法として、1つまたは複数のディレクトリグループで名前空間内の任意の集まりのディレクトリを管理することができる。コンピュータの1つまたは複数のモジュールに対し、図2の制御モジュール250など、割り当てられているサブツリーを管理するためのディレクトリサービスを実装する役割が与えられる。一実施形態では、各ディレクトリグループは、後述のように、Byzantine-fault-tolerantグループ（または単にByzantineグループと呼ぶ）である。しかし、ディレクトリグループは、Byzantine-fault-tolerantグループである必要はなく、他のグループを使用することもできる。

40

【0071】

図4の実線は、どのディレクトリが他のどのディレクトリのサブディレクトリであるか

50

を識別する、ディレクトリ間の関係を示している。たとえば、ディレクトリ C は、ディレクトリ B のサブディレクトリである。ディレクトリは、サブディレクトリから見てサブディレクトリの「親」ディレクトリとも呼ばれる。たとえば、ディレクトリ B は、ディレクトリ C の親ディレクトリと呼ばれる。

【0072】

図4のそれぞれの点線のボックスは、特定の点線で囲まれているディレクトリを管理するディレクトリグループを示している。したがって、名前空間400の例では、ルートの名前空間はディレクトリグループ402により管理され、ディレクトリA、B、C、FおよびGはディレクトリグループ404により管理され、ディレクトリDおよびEはディレクトリグループ406により管理され、ディレクトリHおよびJはディレクトリグループ408により管理され、ディレクトリK、I、L、およびMはディレクトリグループ410により管理される。

10

【0073】

特定のディレクトリまたは名前空間を管理するディレクトリグループは、そのディレクトリに格納されている各ファイルのディレクトリエントリだけでなく、ディレクトリ内の各サブディレクトリのディレクトリエントリを保持する役割を持つ。ファイルの各ディレクトリエントリにより、ファイルが格納されている分散ファイルシステム150内の1つまたは複数のコンピュータが識別される。サブディレクトリの各ディレクトリエントリにより、そのサブディレクトリを管理する役割を持つディレクトリグループが識別される。ディレクトリエントリは、さらに、作成、修正、およびアクセスタイムスタンプ、読み込みおよび書き込みアクセス制御リスト、複製の場所の集まり、ファイルのサイズなどの追加情報も格納できる。

20

【0074】

各ディレクトリグループは、名前空間ルートおよび／または名前空間内の1つまたは複数のサブツリーを管理する役割を持つ。各ディレクトリグループは、さらに、1つまたは複数の追加サブツリーを識別し、それらの追加サブツリーの管理責任を別のディレクトリグループに委任することができる。たとえば、ディレクトリDおよびEは、もともと、ディレクトリグループ404により管理されていたが、後から、ディレクトリグループ406に委任することができる。

【0075】

30

ディレクトリグループでは、いつでも、サブツリーを他のディレクトリグループに委任することを決定できる。一実施形態では、この決定は作業負荷に基づいて行われ、ディレクトリグループは、過負荷になりつつあるとグループが判断した場合にサブツリーを委任する決定を下す。グループでは、過負荷になりつつあることを決定するためにさまざまなファクターを使用することができ、一実施形態では、各ディレクトリグループが、マシン1台当たりの予想ディレクトリ数の平均値にほぼ等しいサイズのサブツリーを管理しようとする（たとえば、10,000のオーダーで）。

【0076】

サブツリーの委任先となるディレクトリグループを、さまざまな方法で決定できる。一実施形態では、委任を実行するディレクトリグループが、認識される分散ファイルシステム150からランダムにコンピュータを選択し、その選択されたコンピュータをサブツリーの委任先となる新しいディレクトリグループとして使用する。他のさまざまなファクターが選択プロセスに関わる（たとえば、可用性の低いコンピュータを選択しない、最近サブツリーを委任したコンピュータを選択しない、など）。

40

【0077】

ディレクトリグループは、ディレクトリグループの1つまたは複数のメンバーにより電子署名された委任証明書を生成することにより特定のサブツリーを委任することができる。複数のメンバーが委任証明書に署名する状況では、署名プロセスはさまざまな形態を取りうる。一実施形態では、各メンバーは委任証明書の各自のコピーに署名する。他の実施では、委任証明書に再帰的に署名する（たとえば、あるメンバーが証明書に署名し、次に

50

その電子署名された証明書に他のメンバーが署名するというように行う)。異なるメンバーが再帰的に証明書に署名する順序は、電子署名を検証する場合にベリファイヤ (verifier) にその順序が知られている限り重要ではない (たとえば、署名の順序でベリファイヤを事前にプログラムしたり、順序を識別する情報を証明書に挿入しておくことができる)。4つのサイナー (signers) が再帰的に署名する証明書の例を以下に示す。

$$\sigma_{s4}(\sigma_{s3}(\sigma_{s2}(\sigma_{s1}(DC))))$$

ただし、DCは、委任証明書が電子署名されることを表し、 $\sigma_{si}()$ は()の内容がサイナー*i*によって電子署名されたことを示す。

【0078】

一実施形態では、ディレクトリグループ内のメンバー (コンピュータ) の数は、設計者が耐えられるようにしたいと考える障害のあるコンピュータの数に依存する。本明細書で使用しているように、障害のあるコンピュータとは、アクセス不可能な (たとえば、コンピュータの電源がオフになっている、あるいは故障している) コンピュータまたは破損しているコンピュータのことである (たとえば、悪意のあるユーザーまたはプログラムはコンピュータへのアクセス権を取得していると、適切な応答を与えないことにより、または不適切なデータを与えることにより、クエリに対し不適切な形で応答することができる)。一具体例では、*f* 台の障害のあるコンピュータを許容するために、ディレクトリグループは、 $3f + 1$ 台のコンピュータで構成する。さらに、この例では、少なくとも $f + 1$ 台のコンピュータは委任証明書に電子署名する。

【0079】

各名前空間ルートは、証明機関 (CA) から取得した証明書に関連付けられている。証明機関は、名前空間の作成を検証する信頼できる機関である。サブツリーと関連するそれぞれの委任証明書は、現在のサブツリーから遡って0個またはそれ以上の他のサブツリーを通りCAによって署名された名前空間ルート証明書まで辿る証明書連鎖を含む。したがって、それぞれの委任証明書は、サブツリーを管理する認定ディレクトリグループであることを証明する複数の証明書に関連付けられている (証明書連鎖をCAによって署名された証明書まで遡って確定することにより)。

【0080】

委任証明書は、異なるコンポーネントを含むことができ、一実施形態では、委任証明書は、(1) 委任を実行するディレクトリグループにより管理されているサブツリーのルートの下にある委任されている経路の識別、(2) 委任を実行するディレクトリグループに委任されているサブツリーのルートの識別、(3) 委任されているサブツリーの識別、および(4) サブツリーの委任先となっているグループのメンバーの識別を含むことができる。サブツリーおよび経路メンバーの識別は変化する場合があります、実際のディレクトリ名 (たとえば、ディレクトリA、B、C、Dなどの名前) または別法として、識別名 (たとえば、グローバル一意識別子 (GUID)) とすることができる。識別番号を使用することにより、ディレクトリ名が変更された場合でも委任証明書を作成し直す必要がない。

【0081】

委任証明書の例は、図4に示されている。ディレクトリグループ402は、CAから、グループ402が名前空間ルートを管理する権限のあることを示す証明書を取得する。この証明書は次のような形式である。

$$\sigma_{OurCA}(\text{"Root", GUID}_{Root}, DG_{402}) \quad (1)$$

ただし、 σ_{OurCA} はCA「OurCA」が証明書に署名したことを示し、「Root」は名前空間ルートの名前であり、 $GUID_{Root}$ は名前空間ルートのグローバル一意識別子であり、 DG_{402} はディレクトリグループ402のメンバーの名前 (または他の識別子) を表す。

【0082】

ディレクトリグループ402がディレクトリAで始まるサブツリーをディレクトリグループ404に委任することに決めた場合、ディレクトリグループ402はディレクトリグ

10

20

30

40

50

ループ404のメンバーに受け渡す委任証明書を生成する。この委任証明書は、上の証明書(1)とともに、以下の証明書を含む。

$$\sigma_{DG402}(\text{GUID}_{\text{Root}}/A, \text{GUID}_A, DG_{404}) \quad (2)$$

ただし、 σ_{DG402} は証明書がディレクトリグループ402のメンバーによって署名されたことを示し、 $\text{GUID}_{\text{Root}}/A$ はディレクトリグループ404($/A$)に委任される経路とともにディレクトリグループ402($\text{GUID}_{\text{Root}}$)に委任されるサブツリーのルート GUID であり、 GUID_A は委任されるサブツリーのグローバル意識別子(つまり、ディレクトリAで始まるサブツリー)であり、 DG_{404} はディレクトリグループ404のメンバーの名前(または他の識別子)を表す。

【0083】

同様に、ディレクトリグループ404がディレクトリDで始まるサブツリーをディレクトリグループ406に委任することに決めた場合、ディレクトリグループ404はディレクトリグループ406のメンバーに受け渡す委任証明書を生成する。この委任証明書は、上の証明書(1)および(2)とともに、以下の証明書を含む。

$$\sigma_{DG404}(\text{GUID}_A/B/C/D, \text{GUID}_D, DG_{406}) \quad (3)$$

ただし、 σ_{DG404} は証明書がディレクトリグループ404のメンバーによって署名されたことを示し、 $\text{GUID}_A/B/C/D$ はディレクトリグループ406($/B/C/D$)に委任される経路とともにディレクトリグループ404(GUID_A)に委任されるサブツリーのルート GUID であり、 GUID_D は委任されるサブツリーのグローバル意識別子(つまり、ディレクトリDで始まるサブツリー)であり、 DG_{406} はディレクトリグループ406のメンバーの名前(または他の識別子)を表す。

【0084】

図の例では、委任証明書は、特定のサブツリー内のディレクトリ毎にではなく、委任時点で発行される。たとえば、委任証明書は、 $/A/B$ または $/A/B/C$ ではなくA(サブツリー内の上位ディレクトリ)について発行される。

【0085】

図4では、分散ファイルシステム150内の各コンピュータは、名前空間内のパス名のあるサブセットをそのパス名を管理するディレクトリグループにマッピングするローカルキャッシュ(たとえば、図2のキャッシュ260)を保持する。たとえば、特定のコンピュータのキャッシュは、パス名 $/A$ 、 $/A/B$ 、 $/A/B/C$ 、 $/A/F$ 、および $/A/F/G$ のそれぞれからディレクトリグループ404へのマッピングを含むことができる。異なるコンピュータは、キャッシュ内に異なるマッピングを持つが、それぞれは通常、少なくとも名前空間ルートからその管理しているディレクトリグループ(ディレクトリグループ402)へのマッピングを含む。

【0086】

管理しているディレクトリグループのマッピングへのパス名を保持することにより、コンピュータは、常に名前空間ルート(およびおそらく他のディレクトリグループ)を管理しているディレクトリグループにアクセスしなくても、ディレクトリグループの検索プロセスの少なくとも一部を自動的にローカルで実行することができる。たとえば、コンピュータ側で、パス名が $/A/B/f o o . t x t$ である「 $f o o . t x t$ 」という名前のファイルにアクセスしようとし、コンピュータは、そのローカルキャッシュ内で、ディレクトリグループ404のパス名のマッピングを備えると仮定する。この例では、コンピュータは、自前のローカルキャッシュから、ディレクトリB内のファイルを管理するディレクトリグループ404のメンバーを、したがってファイル $f o o . t x t$ を容易に識別することができる。そこで、ファイル「 $f o o . t x t$ 」の場所を判別するためにアクセスするコンピュータ(つまり、どのコンピュータがパス名 $/A/B$ に対するディレクトリエントリを管理するか)の判別は、判別を行うためにディレクトリグループ402または404のいずれかにアクセスしなくても、キャッシュ内の情報に基づきコンピュータが実行する。

【0087】

コンピュータは、ローカルキャッシュ内に、パス名全体をディレクトリグループにマッピングするための十分な情報がない場合、キャッシュ内に存在するパス名の最も長いプレフィックスに対するマッピングを見つける。その後、コンピュータは、その最も長いプレフィックスで最後のディレクトリを管理しているディレクトリグループにアクセスし、パス名の残り部分とその委任証明書をできるだけ多く管理するディレクトリグループを判別する。ディレクトリグループにアクセスし、委任証明書を取得するこのようなプロセスは、適切なマッピングが見つかるまで続けられる。

【0088】

たとえば、コンピュータ側で、パス名が／A／B／C／D／foo2.txtである「foo2.txt」という名前のファイルにアクセスしようとし、コンピュータは、そのローカルキャッシュ内で、ディレクトリグループ406ではなく、ディレクトリグループ404のパス名のマッピングを備えると仮定する。コンピュータは、そのパス名を見て、パス名（／A／B／C）内にあるキャッシュに含まれる最長のプレフィックスのマッピングを見つけ、ディレクトリグループ404である、そのディレクトリを管理する役目を負っているディレクトリグループにアクセスする。コンピュータは、ディレクトリグループ404のメンバーに対し、ディレクトリグループ406の委任証明書である、パス名／A／B／C／D／foo2.txtの関連するサブツリーに対する委任証明書を問い合わせる。ディレクトリグループ404のメンバーは、この委任証明書を問い合わせを行っているコンピュータに返し、このコンピュータはその委任証明書を検証することができる（たとえば、署名する側のコンピュータの公開鍵に基づいて）。受け取った委任証明書により、ディレクトリ／Dを管理する責任のあるディレクトリグループを識別し、コンピュータはファイル「foo2.txt」の場所を決定するためにそのディレクトリグループにアクセスすることを知る。したがって、ファイル「foo2.txt」の場所を判別するためにどのコンピュータにアクセスすべきかを判別する作業にディレクトリグループ404のメンバーへのアクセスが伴ったが、決定を下すためにディレクトリグループ402のメンバーへのアクセスは必要なかった。

【0089】

ディレクトリおよびファイルの複製および格納

図1の分散ファイルシステム150は、ディレクトリエントリおよびそれらのエントリに対応するファイルの格納を異なる方法で管理する。システム150に格納されるファイルは複製され、システム150内の複数の異なるコンピュータに保存される。さらに、そのファイルに対するディレクトリエントリが作成され、これもまた、Byzantine-fault-tolerantグループの一部であるシステム150内の複数の異なるコンピュータ上に保存される。ディレクトリエントリの保存先コンピュータは、さらに以下で詳しく説明するように、ファイルの保存先コンピュータよりも多い。

【0090】

明細書で説明しているファイルおよびディレクトリエントリの格納の異なる取り扱いを、上述の階層型格納構造と関連して使用できる。しかし、明細書で説明しているファイルおよびディレクトリエントリの格納の異なる取り扱いは、階層型格納構造を採用していないシステムでも使用できる。

【0091】

Byzantine-fault-tolerantグループは、ある一定の台数のコンピュータに障害が生じている（危うい状態にあるかまたは利用できない状態にある）としても情報の格納および／または他のアクションの実行に使用できるコンピュータのグループである。コンピュータは、悪意のあるユーザーがコンピュータを操作している、悪意のあるプログラムがコンピュータ上で実行されているなどのさまざまな異なる形で危うい状態に置かれる場合がある。要求への応答に拒絶する、わざと正しくない情報またはゴミ情報などで要求に応答するなど、危うい状態にあるコンピュータについてあらゆる種類の処理が観察できる。Byzantine-fault-tolerantグループは、このような危うい状態にあるコンピュータが存在していても、情報の格納および／またはそ

の他のアクションの実行を正確に行うことができる。Byzantineグループは、当業者にはよく知られており、したがって、本発明に関係している場合を除きこれ以上説明しない。

【0092】

当業者には、一定数 f の故障コンピュータがあってもある種の計算を正しく行えるようにするために（故障コンピュータはパワーダウン状態にあるなど危うい状態にあるかまたは何らかの形で利用できない状態にある）、Byzantine-fault-tolerantグループは少なくとも $3f + 1$ 台のコンピュータを備えていなければならないことが知られている。分散ファイルシステム150では、ディレクトリエントリはByzantine-fault-tolerantグループの $3f + 1$ 台のコンピュータ上に格納されるが、ファイル自体は $f + 1$ 台のコンピュータ上に格納される（ディレクトリエントリが格納されるのと同じコンピュータの1つまたは複数であってよい）。

10

【0093】

図5は、サーバーレス分散ファイルシステム内のファイルおよび対応するディレクトリエントリの格納例を示す図である。ファイルシステム500（たとえば、図1のサーバーレス分散ファイルシステム150）は、12台のコンピュータ502、504、506、508、510、512、514、516、518、520、522、および524を備える。システム500の設計者は、2つのコンピュータ障害を許容できるようにすることを望んでいると仮定すると、Byzantine-fault-tolerantグループは少なくとも7（ $(3 \times 2) + 1$ ）台のコンピュータを備える必要がある。Byzantineグループ526は、502～514を含むように図に示されている。

20

【0094】

ファイル528をファイルシステム500に格納する場合、適切なディレクトリグループ内のコンピュータにより対応するディレクトリエントリ530が格納される（ディレクトリファイルを管理する役割を持つディレクトリグループは、ファイル528のパス名に基づいて格納される）。ディレクトリエントリ530に対する図5のディレクトリグループは、Byzantineグループ526であり、ディレクトリエントリ530はByzantineグループ526内の正常に機能しているそれぞれのコンピュータ502～514に格納される。したがって、ディレクトリエントリ530は最大7台までの異なるコンピュータ上に格納される。一方、ファイル528は、複製され、3台のコンピュータのそれぞれ（コンピュータ516、520、および524）に格納される。図に示されているように、ファイル528が格納されているコンピュータは、Byzantineグループ526になくてもよく、通常はない（しかしファイル528が格納されるコンピュータの1つまたは複数を任意選択によりByzantineグループ526内に置くことも可能である）。

30

【0095】

各ディレクトリエントリは、対応するファイルの名前、ファイルの格納先であるコンピュータの識別、およびファイルの内容をディレクトリエントリに対応する物として検証できるようにするファイル検証データを含む。ファイル検証データはさまざまな形態をとることができるが、一実施形態では、MD5（Message Digest 5）、SHA-1（Secure Hash Algorithm-1）などの暗号化セキュアハッシュ関数をファイルに適用することにより生成されるハッシュ値である。記憶域からファイルを取り出す場合、取り出し側コンピュータは、ハッシュ値を再生成し、その値をディレクトリエントリ内のハッシュ値と比較して、コンピュータが正しいファイルを受信したことを検証することができる。他の実施では、ファイル検証データは、ファイル識別番号（たとえば、ファイルの一意的識別子）、ファイルバージョン番号、およびファイルに付けられている署名のユーザーの名前の組合せである。

40

【0096】

図6は、サーバーレス分散ファイルシステム内でファイルを格納するプロセスの例を説明する流れ図である。最初に、新しいファイル格納要求がクライアントコンピューティン

50

グデバイスに届く（処理602）。クライアントは、ファイルとファイル名を暗号化し、ファイル内容のハッシュを生成する（処理604）。クライアントは、暗号化されたファイル名およびファイル内容ハッシュを適切なByzantine-fault-tolerantディレクトリグループに、ディレクトリエントリを作成する要求とともに送信する（処理606）。ディレクトリグループは、ファイル名が既存の名前と衝突していないか、クライアントに要求している内容を実行する許可が与えられているかを検証するなどして、要求の妥当性を確認する（処理608）。要求の妥当性が確認されない場合、要求は失敗する（処理610）。しかし、要求の妥当性が確認された場合、ディレクトリグループは新しいファイルのディレクトリエントリを生成する（処理612）。ディレクトリグループは、さらに、新しいファイルの複製セットを決定し、その複製セットを新規生成されたディレクトリエントリに追加する（処理614）。ファイルの複製も生成され（処理616）、ファイルシステム内の複数のコンピュータに保存される（処理618）。

【0097】

Byzantineグループ内にディレクトリエントリを格納し、ファイル検証データをエントリに挿入することにより、フォールトトレランスが保持される（最大f件までの障害）。しかし、格納領域要件およびByzantineオペレーションは、Byzantineオペレーションを使用してアクセスせず、ディレクトリから別々にファイルを格納すると低減される。たとえば、ディレクトリエントリは100バイトのオーダーになることもあるが、ファイル自体は数千さらには数十億バイトのオーダーとなる場合がある。

【0098】

ディレクトリおよびファイルのロックメカニズム

図1の分散ファイルシステム150内の各オブジェクト（たとえば、ディレクトリおよびファイル）は、リースされたロックの集まりと関連付けられている。これらのロックは、アプリケーションで実行したいオペレーションの種類に基づいて、アプリケーションがそのオペレーションを実行するためディレクトリを開くのか、ファイルを開くのかを判別するために使用される。ロックは、ロックの種類および争奪のレベルによって決まる特定のタイムスパンを持つリースとみなすことができる。たとえば、書き込みロックのタイムスパンは数分程度であるが、読み込みロックのタイムスパンは数日程度と長い場合がある。アプリケーション側でオブジェクトにオペレーションを実行したい場合、アプリケーションが実行されているクライアントコンピュータは、オペレーションを実行するために必要なロックがすでにかかっているかどうかを調べる。ロックがまだかかっていなければ、そのオブジェクトを管理する役割を持つディレクトリグループに適切なロックを要求する。アプリケーションは、所望のオペレーションの実行を完了した後、任意選択により、取得されているロックを開放するか、または自動的にタイムアウトになるか、または管理するディレクトリグループによりリコールされるまで保持することができる。

【0099】

特定のディレクトリについて、ディレクトリを実装するByzantine-fault-tolerantグループは、そのディレクトリ内のすべてのファイル、ディレクトリのサブディレクトリの名前、およびディレクトリ自体を削除する権限に対するロックを管理する。このロックメカニズムは、適切なファイルおよびディレクトリ上の広範な（粒度が粗い）ロックを要求側クライアントコンピュータに付与し、クライアントコンピュータが多数の読み込みおよび／または更新を、ロック取得に対してByzantineメッセージを何回も要求せずに1回のByzantineロック取得で処理できるようにする。

【0100】

図の例では、ロックメカニズムは10種類の異なるロック、つまり、Read、Write、Open Read、Open Write、Open Delete、Not Shared Read、Not Shared Write、Not Shared Delete、Insert、およびExclusiveを採用している。ReadおよびWriteロックは、オブジェクト内のデータへのアクセスを制御する場合に使用する

(たとえば、ファイルの内容)。Open Read、Open Write、Open Delete、Not Shared Read、Not Shared Write、およびNot Shared Deleteロックは、オブジェクトを開く操作を制御する場合に使用する。InsertおよびExclusiveロックは、専用ロックである。これら10個のロックについて、以下で詳述する。アプリケーション側で実行したいオペレーションに応じて、これらのロックのうちの適切な物がアプリケーションから要求される。

【0101】

Readロック。Readロックは、アプリケーションが関連するファイルを読み込むようにアプリケーションによって要求される。Readロックを、Writeロックと

10

【0102】

Writeロック。Writeロックは、アプリケーションが関連するファイルに書き出せるように（更新ともいう）アプリケーションによって要求される。Writeロックを、Readロックとともに使用すると、ディレクトリグループでファイル内のデータの一貫性を保持することができる。

【0103】

アプリケーション側でオブジェクトを開きたい場合、ディレクトリグループは、（1）モードはアプリケーションがすでにそのオブジェクトを開いてしまっている他のアプリケーションと衝突することを要求しているか、（2）オペレーションは、他のアプリケーションがすでに開いてしまっていて、共有する意志のあることを示しているそのオブジェクトの内容と衝突することになるということについてオブジェクトを共有するつもりであるオペレーションであるかという2つのチェックを実行する。Open Read、Open Write、Open Delete、Open Not Shared Read、Open Not Shared Write、およびOpen Not Shared Deleteという10個のロックのうちの6個がこのチェックをサポートすることを対象としている。これらのロックは、オブジェクトを開く機能をアプリケーションに付与するために使用されるが、必ずしも、そのオブジェクトに対するデータを取得できることを保証していない（ReadロックまたはWriteロック（アプリケーション側で実行したいオペレーションの種類による）を取得してデータにアクセスする）。

20

30

【0104】

Open Readロック。Open Readロックは、アプリケーションが読み込みのため関連するオブジェクトを開けるようにアプリケーションから要求される。

【0105】

Open Writeロック。Open Writeロックは、アプリケーションが書き込みのため関連するオブジェクトを開けるようにアプリケーションから要求される。

【0106】

Open Deleteロック。Open Deleteロックは、アプリケーションが削除のため関連するオブジェクトを開けるようにアプリケーションから要求される。

40

【0107】

Open Not Shared Readロック。Open Not Shared Readロックは、アプリケーションが他のアプリケーションとオブジェクト読み込み機能を共有するつもりがない場合にアプリケーションによって要求される。

【0108】

Open Not Shared Writeロック。Open Not Shared Writeロックは、アプリケーションが他のアプリケーションとオブジェクト書き込み機能を共有するつもりがない場合にアプリケーションによって要求される。

【0109】

Open Not Shared Deleteロック。Open Not Shar

50

ed Delete ロックは、アプリケーションが他のアプリケーションとオブジェクト削除機能を共有するつもりがない場合にアプリケーションによって要求される。

【0110】

サポートされている他の2つのロックはInsert ロックおよびExclusive ロックである。

【0111】

Insert ロック。Insert ロックは、ディレクトリ内のオブジェクト用に特定の名前を作成するようにアプリケーションによって要求される。Insert ロックを付与すると、アプリケーションは特定の名前でオブジェクトを作成する権限が与えられる。Insert ロックは、同じオブジェクト名を持つ他のInsert ロックおよびそのディレクトリ上のExclusive ロックと衝突する。

【0112】

Exclusive ロック。Exclusive ロックは、ディレクトリ内に存在する（が、すでに存在しているわけではない）それぞれの使用可能な名前に対するInsert ロックを含む、前記の9つのロックのすべてを取得するように、アプリケーションによって要求される。ディレクトリに対するExclusive ロックは、ディレクトリ内のファイルまたはサブディレクトリに対するExclusive ロックを暗黙のうちに含むことはないが、むしろ、ディレクトリの名前空間のみに対するExclusive ロックを含む。Exclusive ロックは、前記の9つのロックのそれぞれと衝突する。

【0113】

さまざまな異なるロックの間に、いろいろな衝突が存在する。表Iは、一実施形態のロック間の衝突を示す衝突マトリックスである。表Iでは、Ins (Insert)、Excl (Exclusive)、O-R (Open Read)、O-W (Open Write)、O-D (Open Delete)、O-!R (Open Not Shared Read)、O-!W (Open Not Shared Write)、およびO-!D (Open Not Shared Delete) という略記を使用する。表Iのセル内の「X」は、対応する2つのロックの間の衝突を示すが、たとえば、Open ReadはOpen Not Shared Readと衝突するが、Open Not Shared Writeとは衝突しない。

【0114】

【表1】

表 I

	Ins	Read	Write	Excl	O-R	O-W	O-D	O-!R	O-!W	O-!D
Ins	X	X	X	X						
Read	X		X	X						
Write	X	X	X	X						
Excl	X	X	X	X	X	X	X	X	X	X
O-R				X				X		
O-W				X					X	
O-D				X						X
O-!R				X	X					
O-!W				X		X				
O-!D				X			X			

【0115】

1つのクライアントコンピュータのみが名前空間のある領域にアクセスした場合にパフォーマンスを高めるため、ファイルシステム150は、アプリケーション（またはクライアント）が近い将来追加関連ロックを要求する可能性が高いという仮定のもとで、クライアント要求に対し実行されているアプリケーションよりも広い範囲のロックを発行することができる。たとえば、アプリケーションがファイル/A/B/C/foo.txtを開

く場合、クライアントはこのファイルのロックを要求する。ディレクトリグループがロックを付与する場合、これはロックを／A／B／Cのディレクトリロックにアップグレードすることができる（たとえば、過去のパフォーマンスに基づいて、ディレクトリグループがそのディレクトリ上の衝突がまれであると判断する）。アプリケーションが同じディレクトリ内の他のファイルを開く場合、クライアントはディレクトリグループに他のロックを要求することなく、ファイルを開くことができる。

【0116】

クライアントのロック要求が他のクライアントに付与されている既存のロックと衝突する場合、ディレクトリグループは、以前発行したロックを、要求を拒絶するのではなく、新しい要求と衝突しないロックにダウングレードしようとする。ロックアップグレードを行うと、クライアントは要求しなかったロックを保持し、ロックダウングレードの成功の確率は、通常、非自明な値となる。ロックのリコールが失敗した場合、その要求は拒否される。

【0117】

さまざまなオペレーションをファイルシステム内のオブジェクトに対して実行することができる。以下の表 I I は、より一般的なオペレーションのうちのいくつかと、それらのオペレーションを実行するためアプリケーションによって要求されるロックをまとめた。

【0118】

【表 2】

表 I I

オペレーション	説明
Read Object	ディレクトリまたはファイルの読み込みの要求。オブジェクトのOpen Readロックの後にReadロックが必要である。任意選択により、必要ならば、アプリケーション側でOpen Not Sharedロックを要求することができる。
Write/Update Object	ファイルへの書き込みの要求。オブジェクトのOpen Writeロックの後にWriteロックが必要である。任意選択により、必要ならば、アプリケーション側でOpen Not Sharedロックを要求することができる。
Delete File	ディレクトリ内のファイルの削除要求。Open DeleteおよびWriteロックを必要とする。通常、アプリケーション側でも、Open Not Sharedロックをすべて要求する。
Delete Directory	ディレクトリの削除要求。ディレクトリに対するExclusiveロックを必要とする。ディレクトリは、空の場合のみ削除できる。
Rename Directory	ディレクトリの名前変更の要求。親ディレクトリ（名前変更されるディレクトリがサブディレクトリとなっているディレクトリ）に対するExclusiveロックおよびデスティネーションディレクトリ内の新しいディレクトリ名に対するInsertロックが必要である。複数のディレクトリにまたがる名前変更の場合、新しい親ディレクトリに対してInsertロックが必要になる。
Rename File	ディレクトリ内のファイルの名前変更の要求。ファイルに対するWriteロック、およびディレクトリ（名前変更が複数のディレクトリにまたがる場合には異なるディレクトリである）内の新しい名前に対するInsertロックが必要である。
Create Object	新しいファイルまたはディレクトリの作成の要求。新しい名前に対するInsertロックを必要とする。

【0119】

ファイルに変更を加える場合、変更は、コンピュータによりローカルで行われ、その後、ファイル（暗号化された後に）はそのファイルを管理する役割を持つディレクトリグループにプッシュされ戻される。この情報はディレクトリグループ内のさまざまなコンピュータに格納され、更新されたファイルは適切なコンピュータに格納される。

【0120】

セキュアクラッシュリカバリ (Secure Crash Recovery)

図1を再び参照すると、上述のシステムは、不正ユーザーおよびマシンによる変更またはアクセスを防止して、分散ファイルシステム150内にファイルを格納し、ファイルおよびディレクトリエントリの格納を管理する。ファイルがシステム内の複数の異なるコンピュータ上のシステム150内に複製され、格納されているとしても、ディレクトリエントリ、およびシステム内の複数の異なるコンピュータ上に保存されているファイルに対して保護が実施される。ファイルの書き込みでは、いくつかの故障／安全でないコンピュータ f （安全でない／故障コンピュータは、停電など危うい状態にあるかまたは他の何らかの形で利用できない状態にある）にも関わらず連携する必要があるコンピュータの台数について、Byzantine-fault-tolerantグループは少なくとも $3f+1$ 台のコンピュータを含んでいる必要がある。分散ファイルシステム150では、ディレクトリエントリはByzantine-fault-tolerantグループの $3f+1$ 台のコンピュータ上に格納されるが、ファイル自体は $f+1$ 台のコンピュータ上に格納される（ディレクトリエントリが格納されるのと同じコンピュータの1つまたは複数であってよい）。

【0121】

ファイル書き込みでは、ローカルマシンであってもクラッシュなどの致命的な障害の後、またはユーザーがマシンからログアウトした後に保護を実施する必要がある。より具体的には、ユーザーがマシンにログインした場合、そのマシンがユーザーに代わってファイルを更新するということである。分散ファイルシステムの一部であるファイルに更新を加えた場合、更新はシステムに即座にアップロードされるわけではない。むしろ、システムリソースに関しては、即時アップロードはひどくコストが高くつく。ログインして更新を実行した後、ユーザーがマシンからログアウトするか、またはマシンでクラッシュした場合、クラッシュまたはログアウトイベントに先立ってユーザーが正当に加えた変更をアップロードできる状態をマシンが保持している間に、ユーザーがログインできなくなった場合に、ユーザーに代わって処理できる状態をマシンが保持することを防止する必要がある。それぞれの書き込みにユーザーのRSA秘密鍵で署名させると、マシンがデータとともに書き込みを認証する署名を格納しているため後からアップロードすることが可能である。マシンが書き込みの認証のセキュアディレクトリグループを納得させる必要があった場合、できるならマシンはデータとともに証明書を送信する。しかし残念なことに、この解決策もまた、過剰なマシンリソースが要求される。たとえば、RSA秘密鍵演算に基づく署名に対するコストは、現在のディスクの約6~8msのディスク待ち時間と比較して1GHzプロセッサ上でコンピュータ処理(CPU)時間は約6.5msである。好ましくは、システムでは、ファイル書き込みのクリティカルパスではこのようなコスト高を回避することになる。本明細書で提示している実施形態によれば、システム150などのシステムに対する保護は効率がよく、ユーザーの秘密鍵が証明提示の際に利用可能でない場合にファイル書き込みに対する保護が行われる。

【0122】

委任証明書

図7および図8の組合せを参照すると、本発明の一実施形態は、委任証明書を使用してファイル書き込み保護を処理する。この実施形態によれば、ユーザーがまずファイル710を書き込む場合、ユーザーは、委任証明書に示されている予め定めた基準に従ってマシン106上のクライアントソフトウェア204がユーザーに代わりファイル710を更新するのを許可するユーザーの秘密署名鍵で委任証明書720に署名する（処理810）。たとえば、この基準により、委任証明書を限られた時間に限定すること、および／または

新しいバージョン番号のファイル 710 にのみ変更を許可することが可能である。クライアントマシン 106 は、暗号化されたファイルデータとともにローカルディスクの分散ファイル部分 128 に証明書を格納する（処理 820）。同じファイルにその後書き込みを行うと、マシン 106 はデータを分散ファイル部分 128 に書き込む。マシン 106 がクラッシュして再起動し、ネットワーク 110 に対し更新ファイルがある場合、マシン 106 は更新のログをディレクトリグループ内の複製 750 に送信し、委任証明書 720 を挿入する（処理 840）。複製 750 は、委任証明書をチェックし、正しいバージョン番号を識別する指示、証明書が期限切れになっておらず、正しいマシンの名前を指定しているという指示などの予め定めた基準を満たした場合のみ（処理 850）変更を受け付ける。通常の場合は、マシンがクラッシュしない場合、ファイルは正常に閉じられ、ディレクトリ内に格納されているファイル内容のハッシュ表現を、たとえば、すでに存在している可能性のあるセキュリティ保護されたマシン間接続を使用して、複製 750 に送信することができる。ユーザーが、変更を複製グループ 750 にアップロードする前にマシン 106 からログアウトした場合、修正されたファイルのハッシュに対しユーザーの秘密鍵で署名する。ユーザーは、ユーザーがログアウトし、複製 750 がユーザーの代わりにマシン 106 からもうそれ以上要求を受け付けることができなくなった場合に、委任証明書を取り消す（処理 860）。取り消しでは、取り消し証明書を格納するために複製されたオペレーションを実行する必要がある。

10

【0123】

一実施形態では、委任証明書は、ディレクトリグループによってチェックされるいくつかのセキュリティ項目のうちの 1 つである。たとえば、ファイル 710 を閉じて、ディレクトリに更新を送信する前に、ローカルマシンにクラッシュが発生した場合、更新がディレクトリグループに届くと、委任証明書 720 上のユーザーの署名とファイル 710 の内容のマシン 106 による署名とを比較することで、ファイルの有効性をチェックすることができる。

20

【0124】

図 7 および 8 で説明している方法は、マシン 106 がクラッシュしてもユーザーの鍵を損なうことはないが、いくつかの欠点がある。特に、マシン 106 上でユーザーがファイル 710 を書き込み、マシン 106 がクラッシュして、マシンが損なわれ、たとえば、ファイル 710 がクラッシュ時点で開いていると仮定する。ユーザーからの委任証明書がマシン 106 上に存在し、委任証明書がユーザーによって取り消されていないため、マシン 106 は、再起動の後任意の方法でファイル 710 を変更することができる。したがって、Outlook のデータベースファイルなど常時開いているある種のファイルでは、このような脆弱性は望ましくないリスクとなりうる。そこで、不本意なシャットダウンが発生した場合、通常であれば、Outlook の .pst ファイルに委任証明書が使用されることになり、攻撃者がその時を狙ってこの脆弱性につけ込む可能性がある。しかし、委任証明書法では、委任証明書の脆弱性はクラッシュ時に行われる修正を受けるファイルにのみ制限されるという意味でファイルが保護されるのである。したがって、その証明書の影響を受けないファイルは修正されず安全である。

30

【0125】

一実施形態では、委任証明書は、有効期限による期間制限が課せられている。しかし、委任証明書の都合の悪い時期が期限切れの日だと、ローカルマシンがデータアップロードを行えなくなった場合にさらに問題が大きくなる可能性がある。一方、データアップロードを行えるように委任証明書のタイムアウトを長くすると、マシン 106 が脆弱な状態に置かれている期間が長くなり攻撃者に付け込まれるおそれがある。この弱点に対処するため、一実施形態では、そうすることが可能であれば復旧したマシンにクラッシュ後直ちにグループとコンタクトを取らせることを目的としている。たとえば、マシン 106 は再起動直後に破損ファイルを保持している可能性があるか、またはマシンは再起動に長い時間がかかりファイル 710 をネットワーク 110 に送り返すことができない。

40

【0126】

50

一実施形態では、委任証明書 720 は最初の書き込みの後に生成される。別法として、書き込み許可でファイル 710 を開き、最初に書き込むまでの間のある時点に委任証明書を生成することができる。生成時に署名された委任証明書 720 をマシン 106 上のローカルログに追加することができる。

【0127】

ディレクトリグループがユーザーに代わってより強力な委任証明書を発行する場合、すべてのファイルに対し委任証明書の署名を回避することができる。たとえば、その委任証明書を（ユーザーに書き込み権限がある）ディレクトリ内のすべてのファイルまたは限られたバージョン番号の集まりに対する他の何らかのファイル群に適用することができる。もちろん、このようなことを行くと、ユーザーの脆弱性が増す、つまり、マシンはクラッシュ後の集まりの中の（ユーザーに修正の権限がある）ユーザーのファイルのすべてについてユーザーに代わって処理することが可能である。

10

【0128】

秘密鍵共有を使用する権限委任

図 9 と図 10 をまとめて参照すると、他の実施形態では、ファイルの書き込み後クライアントマシンを信頼しなくても、ファイル書き込みのクリティカルパスで秘密鍵署名を回避するアプローチを対象としている。図 9 に示されているように、ファイルシステム 900 は、7 台のコンピュータ 902、904、906、908、910、912、および 914 を含む Byzantine-fault-tolerant グループ 926 である。システム 900 の設計者が、2 つのコンピュータ障害を許容できるようにすることを望んでいると仮定すると、Byzantine-fault-tolerant グループは少なくとも 7（（3×2）+1）台のコンピュータを備える。ディレクトリエントリ 930 は、7 台のコンピュータ 902～914 と共有される。

20

【0129】

一般に、クライアントコンピュータ 916 が Byzantine グループ 926 に更新 918 などの更新ファイルに対する書き込みロックを要求する場合、ユーザーに代わってその作業が行われる。ロックが発行された場合、ユーザーは新しいバージョンのファイルを生成することが許可される。バージョン番号は、クライアントコンピュータ 916 が書き込みロックを解除し、実際にファイルが修正されると増分される。

30

【0130】

秘密鍵共有を使用する方法によれば、クライアントコンピュータ 916 がディレクトリグループ 926 にコンタクトを取り、たとえば、ユーザーに代わってファイル書き込みを実行したい場合、秘密鍵 K が作成される（処理 1001）。ユーザーが M を、秘密鍵を決定するために必要な、秘密鍵の割り当て分の個数または秘密鍵の異なる部分を表す物と決定し、M を $f+1$ とすることにより決定され（処理 1002）、ユーザーは再構成しきい値 M とともに秘密鍵 K を N 個の割り当て分に分割する（処理 1004）。グループ 926 については、M は 3 でなければならない。したがって、秘密鍵のユーザー／作成者は、N を、ディレクトリエントリ（ $3f+1$ ）および

40

【0131】

【数 1】

$$M = \lfloor N/3 \rfloor + 1 = f + 1$$

【0132】

を保持する複製グループ 926 のサイズとなるように選択しなければならない。

【0133】

最初にディレクトリグループとコンタクトを取る場合、ユーザーはクライアント 916 を介して、N 個の割り当て分を、この場合には 7 個の割り当て分をグループ 926 のそれぞれのメンバーに 1 つずつ受け渡す（処理 1006）（しかし、一部のグループメンバーがグループ内でアクティブでない場合、クライアント 916 は、メンバーがアクティブになるまで、またアクティブにならない限り、それらのグループメンバーの割り当て分を送

50

信しないが、いつでもせいぜい f 個のメンバーが非アクティブ状態にあると仮定している)。コンピュータ 902~914 のうち 3 台のコンピュータは、対称鍵などの鍵である秘密鍵 K を復元する必要がある。秘密鍵 K は、クライアントコンピュータ 916 上のユーザーに知られており、ユーザーはこの秘密鍵 K を使用して、ディレクトリグループ 926 に送信されるファイル書き込みおよび更新ファイルを認証する。特に、クライアント 916 上のユーザーがファイルを更新する場合、秘密鍵 K を更新ファイルとともに鍵として使用してメッセージ認証コード (MAC) を作成する (処理 1008)。図からわかるように、MAC は、対称鍵、この場合は秘密鍵 K により暗号化されているファイルデータの SHA-1 タイプのハッシュなどの一方向ハッシュ関数とすることができる。システム要件に応じて、秘密鍵 K をファイルデータの一部と組み合わせて、MAC を作成したり、更新全体と組み合わせて MAC を形成することもできる。MAC 形成のコストを制限要因とみなすシステムでは、この方法をデータの一部に対して適用し、コンピューティングリソースを節約することができる。逆に、システムが MAC 形成のコストをセキュリティに比べてあまり重要でないとき、この方法を MAC 形式に適用して、ファイル全体を含むようにできる。

【0134】

秘密鍵 K およびその N 個の割り当て分は、クライアントマシン 916 上に格納されており、マシンがクラッシュしたり、ユーザーがログアウトした場合 (処理 1010)、これらは忘れ去られる。したがって、マシンがクラッシュしたり、ユーザーがログアウトした場合、マシンは MAC がもう添付されていないログエントリの MAC を生成することはできないが、それは、このような MAC を生成する作業には忘れられた秘密鍵 K を知っている必要があるからである。クラッシュもログアウトイベントもない場合、マシンはユーザーの信用情報を保持しており、これを使用すれば、サーバーグループ 926 とのセキュリティ保護された接続を確立し、このサーバーグループ上でグループが添付されている MAC を検証しなくてもログを受け付ける。逆に、クラッシュまたはログアウトがある場合、マシンは、ユーザーの信用情報を保持せず、サーバーグループ 926 との適切なセキュリティ保護された接続を確立することはできないが、その代わりに、たとえば、これは再起動に基づくことができるが、クライアントコンピュータ 916 が可能である場合、たとえば、クライアントコンピュータ 916 は一実施形態により、自動的に、セキュリティ保護されていないチャネル上で更新および関連する MAC をグループに送信する (処理 1010)。

【0135】

ディレクトリグループ 926 は、MAC と更新をログの形式で受け取る。しかし、ログは認証セッションの外部から送られるので、MAC は秘密鍵 K を使用して作成されるため、秘密鍵共有法で更新を実行することができる。特に、図 11 に示されているように、ディレクトリグループは更新ログの形で MAC および更新を受け取る (処理 1110)。ディレクトリグループは、ログのシリアル番号をチェックし、欠損している更新がないか (処理 1112)、グループのそれぞれのアクティブな、正しく機能しているメンバーが秘密鍵 K の割り当て分を公開しているかを検証する (処理 1120)。ディレクトリグループ 926 は、 K の M 個の正しい割り当て分を受け取るまで、または $2f + 1$ 個の割り当て分を受け取るまで、受け取った割り当て分が正しいかろうとなかろうと、待機する (処理 1130)。ディレクトリグループは、受け取った割り当て分が正しいかどうかを判別する (処理 1140)。さらに具体的には、受け取った割り当て分が何らかの秘密鍵 K の割り当て分の正しい集まりになっていない場合、ディレクトリグループでは、割り当て分を生成した際にクライアントが破損していると想定し、ログを破棄する (処理 1140 および 1150)。秘密鍵 K が再構成される場合、ディレクトリグループ 926 は、秘密鍵 K を使用して、更新ログ内の MAC を検証する (処理 1160)。MAC が正しい場合、ディレクトリグループは、ログを認証されている物として受理し、ログ内の更新をそれが認証チャネル上で注目しているユーザーからきた物であるかのように適用する (処理 1170)。MAC がチェックしない場合、ログは破棄される (処理 1150)。一実施形態によ

れば、ログを破棄する決定は、ログ内の更新に割り当てられたシリアル番号をチェックする作業を含んでいる。たとえば、ディレクトリグループが秘密鍵Kの割り当て分を公開する処理を実行するのに先立って、ディレクトリグループに届いたそれぞれのログのチェックを、シリアル番号のリストまたは他の順序チェックメカニズムと突き合わせて行い、受け取ったログが予め定めた順序であることを確認する（処理1112）。そのようにして、順序の狂っている更新は実行できなくなっており、悪意のあるマシンは検出を免れずに有効なログの真ん中からオペレーションを削除できず、残りを送信することはできない。順序チェックを行った後、メンバーは秘密鍵Kの割り当て分を公開することができ（処理1120）、その割り当て分により秘密鍵Kが再作成されない場合または $2f+1$ 個の正しい割り当て分が届いていない場合（処理1130および1140）、ディレクトリグループはログを破棄する（処理1150）。

10

【0136】

一実施形態では、秘密鍵Kは、同じユーザー、クライアントマシン、およびディレクトリグループを共有するすべての書き込みについて関連付けられている単一の鍵である。この実施形態では、MACはディレクトリグループ内の各メンバーに送信され、秘密鍵Kの割り当て分を持つ少なくとも $f+1$ 個のメンバーは鍵を再構成する必要がある。したがって、この実施形態では、鍵は、特定のByzantineグループについて関連するすべての書き込みを対象とすることができる。

【0137】

オーセンティケーター

20

他の実施形態によれば、秘密鍵共有の代わりに、メッセージ認証コード（MAC）のベクトルを使ってファイル書き込みを保護している。この実施形態では、クライアントマシンからの追加信用なしで、メッセージ認証コードのベクトルに権限が委譲される。図12を参照すると、クライアントマシンは、ディレクトリグループのメンバー間に分散するように構成される複数の秘密鍵を作成し、ディレクトリグループのメンバー毎に秘密鍵を1つ作成する（処理1210）。複数の秘密鍵には、ディレクトリグループの各メンバーにより個別に確立される複数の秘密対称セッション鍵を含めることができ、したがって、各メンバーマシンと個々のユーザーの間に対称鍵関係を確立できる。ユーザーは、ファイルを作成し、そのファイルを複数の秘密鍵とともにディレクトリグループに送信するか、またはファイルを送信することなくオフラインでメンバーのそれぞれとの対称鍵関係を確立することができる。いかなる場合も、クライアントマシンは、1つの秘密鍵をディレクトリグループメンバーのそれぞれに送信する（処理1220）。

30

【0138】

一般に、ユーザーが更新を生成し、後で送信するためその更新をログに格納する場合、ログには複数のメッセージが格納され、それぞれのメッセージにはMACのベクトルが添付され、それぞれのMACは処理1210で生成された鍵のうちの1つで鍵合わせしたベクトル内に置かれている（処理1230）。クライアントマシンが、一実施形態により認証チャネルセッションをいきなり終了させるなどクラッシュまたはその他の致命的イベントのせいで使用不能になった場合、クライアントは鍵を忘れ（処理1240）、ログ内の更新は認証チャネルの外部に送信される（処理1260）。当業者であれば、MACのベクトル内の各MACは、対称鍵、または他の多数のMACアルゴリズムのうちの1つで暗号化された更新ログ記録（それ自体、ファイル内容のセキュアハッシュを含む）の一方向ハッシュ関数であることを理解するであろう。この場合、1つのMACを送信するのではなく、複数のMACを送信し、各MACではクライアントとディレクトリグループのうちの1つのメンバーにしか知られていない異なる秘密鍵を使用している。ディレクトリグループのそれぞれの正常に機能しているメンバーは鍵を受け取る。一実施形態では、別々の対称鍵をユーザー、クライアントマシン、およびディレクトリグループメンバー毎に設定する。たとえば、ユーザーは、複製／ディレクトリグループメンバー毎に1つのMACエントリによりオーセンティケーターを計算することにより各書き込みを認証することができる。クラッシュなどに続いてクライアントマシンが再起動した後（処理1240）、復旧

40

50

すると、マシンはオーセンティケータをディレクトリグループに送信することができる（処理1260）。グループの各メンバーは、対応するMACを別々に検証し、更新を受け入れるかまたは更新を拒絶する（処理1270、1280、および1290）。一実施形態では、更新を許可するためには（処理1280）MACの $f+1$ 回の検証が必要であり（処理1270）、そうでなければ、更新は拒絶される（処理1290）。

【0139】

一実施形態では、MACのベクトル内のMACの少なくとも1つを作成するために使用される対称鍵の1つまたは複数は、ディレクトリグループの1つまたは複数のメンバーに対する要求を認証するためなどの他の目的にも使用される鍵である。上述の秘密鍵共有法などの他の実施形態では、それぞれの更新をシリアル番号に関連付けて、順序の狂っている更新を許可しないようにできる。

10

【0140】

都合のよいことに、秘密鍵共有法とMACベクトルを使用する方法の両方において、MACを計算するための暗号法が十分に強いと仮定すると、攻撃者は、 $f+1$ 個のByzantineグループメンバーに対する制御権を取得した場合またはクライアントマシンから秘密鍵を学習できる場合に書き込み要求を改ざんすることぐらいしかできない。クライアントマシン内の鍵は、ページングファイルに入り込むのを回避するために、メインメモリ内に、またピン留めページに保持し、マシンがクラッシュした場合に鍵が破棄されることが好ましい。さらに、ユーザーは、いつでも鍵を破棄することができ、特に、一実施形態ではユーザーは、ログアウトしたい場合に鍵を破棄することが必要である。

20

【0141】

バージョン検証

上述の委任証明書、秘密鍵共有、およびMACベクトル法は、それぞれ、更新および/またはファイル署名をByzantineグループによって格納されるのに先立って、他の保護手段を含めることができる。

【0142】

特に、図13とともに図9を参照すると、バージョン検証を対象とする方法がある。クラッシュした後、クライアントコンピュータ916は、秘密Kタイプ鍵および関連する情報に基づき、ファイルを署名とともに、検証のためByzantineグループ926に送信することができる（処理1310）。Byzantineグループ926は、上述の方法のうちの1つに従って処理し、すべての複製でlock-secret鍵を再構成し、それを使用してMACを検証し、委任証明書をチェックし、またはMACのベクトルを検証する。いずれにせよ、秘密鍵共有法では、秘密鍵はもはや秘密ではない。これらの方法のそれぞれにおいて、認証がチェックされる（処理1330）。特に、委任証明書法では、証明書のチェックが行われ、その証明書が有効であればディレクトリグループはファイルのハッシュを格納する。秘密鍵共有法では、サーバーが、ファイルのMACを検証し、MACが正しければファイルのハッシュを格納する。MACベクトルオーセンティケータ法では、サーバーは、MACのベクトルを検証し、MACベクトルが確認されればファイルのハッシュを格納する。

30

【0143】

この時点以降、この方法によれば、提供されたオーセンティケータ（証明書、共有秘密鍵、またはMACベクトル）を使用して更新されたファイルの他の検証をクライアント916が要求しないようにできる（処理1340）。つまり、これらのオーセンティケータは、1回の使用のみ有効である。したがって、再構成された後に不良複製から鍵または鍵の割り当て分が漏れても、更新のファイルハッシュがすでに複製に公開されているため、またその特定のハッシュのみを検証するため、公開された鍵によりダメージを受けることはない。たとえば、不良複製が、現在障害のあるクライアント916と共謀し、ファイルの破損したバージョンに対して新しいハッシュを生成しようとするのを仮定する。その場合、複製にはすでにオリジナルのハッシュがあるため、ファイルのそのバージョンに対する異なるハッシュを受け付けない、つまりクライアント916は、Byzantineグル

40

50

ープによって秘密鍵再構成が開始されるのに先立って、すでに署名済みデータを Byzantine グループ 926 に「コミット」している。この方法は、データを公開するのに先立ってデータに対するハッシュをコミットするある種の乱数発生プロトコルに似ている。

【0144】

図14を参照すると、一実施形態は、選択的再生攻撃を防止するため検証を使用することを対象とする。特に、秘密鍵共有のそれぞれにおけるオーセンティケータ、オーセンティケータと委任証明書実施形態は、特定のファイルの内容の認証を行う。したがって、クラッシュしてから、ログをディレクトリグループに対して再生するのに先立ってマシンが破損した場合、攻撃者はディレクトリグループに送信する更新および無視する更新を選ぶことが可能になる。

10

【0145】

たとえば、3つのファイルF1、F2、およびF3を更新しているマシンを考える。ユーザーは、まずF1を書き込み、次にF2、そして最後にF3を書き込む。クライアントマシンは、これらの書き込みをファイルに適用し、この3つの新しいファイル（上述の実施形態で生成された）に対してオーセンティケータA1、A2、A3を生成し、ログに格納する。その後クライアントマシンがクラッシュし、再起動に先立って破損している場合、攻撃者は有効なオーセンティケータとともに3つのファイルの更新バージョンを見つけることができる。攻撃者は、新しいオーセンティケータを作成することはできないが、ユーザーがF1およびF2の後にF3が更新されていると確信していたとしても、F1またはF2なしのF3のみの変更をディレクトリグループに報告することができる。

20

【0146】

このような攻撃を防御するため、図14を参照して一実施形態を説明する。まず、特定の更新されたファイルだけでなく、ログ内のその時点までの更新のシリーズ全体について生成オーセンティケータを構成する（処理1410）。上の例では、ディレクトリグループはA1を計算し、次にA1、2、次にA1、2、3と計算し、ファイル1までの更新を認証し、その後、ファイル1と2の両方に対する更新、最後に3つすべての更新を認証する。クライアントマシンが破損した場合、マシンはこれら3つのオーセンティケータの1つを自由に送り返すことができるが、マシンは、ファイル3が更新されたが、ファイル1と2は更新されていないということをディレクトリグループに納得させることができない。したがって、ディレクトリグループはログ内のファイルの順序をチェックする（処理1420）。

30

【0147】

上述の3つの実施形態はすべて、何らかのデータ上の電子署名、たとえば、A（データ）などの「署名」関数の形を取る。ユーザーが新しいログを開始し、ディレクトリグループが第1の更新を受け取った場合、第1のファイルに対する変更を認証することは、 $A(h(F1))$ と表すことができる。ファイル内容ハッシュは、データだけでなく、ファイルIDを含むメタデータについても計算される。したがって、一方のファイルのオーセンティケータを使用して他のファイルの内容を変更することはできない。第2のファイルが更新された場合、ディレクトリグループは $A(h(h(F1); h(F2)))$ を生成する。特に、ディレクトリグループは、第1のファイルの内容ハッシュを第2のファイルの内容ハッシュと連結し、まとめてハッシュし、その結果を認証する。第3のファイルについては、ディレクトリグループは $A(h(h(h(F1); h(F2)); h(F3)))$ を生成する。したがって、ディレクトリグループはメモリ内の付加された1つのハッシュを追跡し、それを新規更新されたファイル内容とともにハッシュしてから、新しいオーセンティケータを計算する。

40

【0148】

一実施形態では、ディレクトリグループはファイル内容の変更と、さらに解除されたロック、アクセス制御リストに対する変更、ファイルの名前変更／削除などの他のオペレーションを含む特定のグループに対する更新の単一ログを構築する。最終的に、ユーザーは

50

、ディレクトリグループにまで遡る他のオペレーションを含めたログをアップロードし、更新をディレクトリ情報のグループのコピーに適用する。この統合ログ内のエントリでは、上述の方法の1つを使用してオーセンティケータを構成しており、したがって、クラッシュした後、安全にアップロードすることができる。クラッシュまたはログアウトに先立ってログをアップロードする場合、クライアントは新しいログ（前にアップロードしたログの内容を含まない）に対して新しいオーセンティケータを開始することができるが、それは、クライアントがもはや第1のログが存在していないことをディレクトリグループに納得させることができないからである。

【0149】

秘密鍵共有およびオーセンティケータ法のコスト

10

秘密鍵共有法とMACベクトルオーセンティケータ法の違いはシステム要件によって決まる。特に、MをSHA-1ハッシュのMACを生成する時間とし、SSGを秘密ロック鍵の割り当て分を生成して複製に送信する時間とし、SSRを割り当て分から秘密ロック鍵を再構成する時間とする。他のパラメータとしては、クライアントがログアウトするまでのディスクへの遅延書き込みの平均数Wに対するByzantineグループが許容するフォールトの数fおよびリカバリ数と書き込み数との比Rがある。

【0150】

秘密鍵共有の実施形態では、Byzantineグループへの第1のオペレーションがユーザーに代わって実行された場合にクライアントマシン上のユーザーが秘密ロック鍵の割り当て分を生成する必要があるが、SSG秒かかる。遅延書き込みのクリティカルパスでは、この実施形態によりMACの計算が付加され、これはM秒を要する。復旧時に、ファイルハッシュの妥当性確認に先立って複製により鍵が再構成されるが、これはSSR秒を要する。

20

【0151】

オーセンティケータベースの実施形態では、ユーザーが最初にByzantineグループメンバーと通信する場合に对称セッション鍵を生成する必要がある場合がある。これらの鍵を生成するコストは小さく、これらの鍵をクライアントマシンとディレクトリグループ内の複製との間の既存のセキュアチャネルで送信することができる。遅延ファイル書き込みのクリティカルパスでは、オーセンティケータベースの実施形態は、 $3f+1$ 回のMAC計算が必要であり、オーセンティケータの生成にM秒を要する。実際には、 $f+1$ 個の正しい複製を納得させ、これらの複製はさらに他の複製を納得させることができるため、 $2f+1$ 回の暗号化で十分である。復旧時に、各複製はその対称鍵を使用して、ファイルハッシュの妥当性を確認し、これにM秒を要するが、余分な再構成作業は行われない。

30

【0152】

秘密鍵分割実施形態では、ユーザーが最初にByzantineグループとコンタクトを取る際に前払いコストを支払い、さらに、クラッシュに先立ってディレクトリグループに更新をアップロードしていなかったファイルにクラッシュが発生した場合に、再構成コストペナルティを支払う。しかし、秘密鍵分割実施形態は、クリティカルパスではコストが低い。ページ修正がログに書き込まれた場合、秘密鍵共有アプローチでは、ファイルハッシュを計算し、ハッシュのMACを計算する必要がある。MACベクトルベースの実施形態では、ファイルハッシュを計算し、ハッシュの $2f+1$ 個のMAC（つまり、 $f=3$ で7つのMAC）を計算する必要がある。したがって、秘密鍵共有ベースのスキームのコストは $CSS = SSG + W * M + W * R * SSR$ となる。

40

したがって、オーセンティケータベースのスキームのコストは $CA = W * (2f + 1) * M + W * R * M$ となる。

【0153】

損益分岐点は、 $W = SSG / (2f * M + R * (M - SSR))$ に対応する。

したがって、トレードオフの関係は、ロック秘密鍵の割り当て分の生成、RSAを使用したハッシュの署名、ハッシュのMACの計算、および割り当て分からの秘密鍵の再構成

50

の相対的コストによって決まる。

【0154】

これらの実施形態ではRC2を128ビット鍵とともに使用して、1GHzのPentium（登録商標）でMACを計算すると仮定している場合、MACの計算に $M = 8 \mu s$ を要する。 $f = 3$ および $R = 0$ とし、秘密鍵共有スキームではファイルキャッシュからディスクへの書き込みのクリティカルパスでMAC計算を1回実行すると仮定すると、オーセンティケータベースのアプローチだと、約 $56 \mu s$ のコストに対して7回の等価MAC計算を実行することになる。したがって、クリティカルパスの待ち時間に関して、スキームは両方とも、ディスク書き込み待ち時間（ $6 \sim 8 ms$ ）とファイルのハッシュを計算する時間に比べて時間のオーバーヘッドがごく小さい。

10

【0155】

秘密鍵共有アプローチでは、約24バイトの1つのデータレコードをログに記録しなければならないが、オーセンティケータベースの実施形態では、7個のレコード（168バイト）をログに記録しなければならない。オーセンティケータベースのアプローチにおけるバイトオーバーヘッドは高いが、168バイトという絶対値は非常に低い。このオーバーヘッドは、ディレクトリグループに対してまだハッシュがフラッシュされていない、修正済みのファイルのみである。

【0156】

128ビットのロック秘密鍵の10個の割り当て分を生成するSSGは $1.5 ms$ [1]であり、署名を生成する時間は $6.5 ms$ であると仮定すると、損益分岐点は $W = 31$ である。

20

【0157】

たとえば、MAC計算では、HMAC-SHA1の計算はPIII 1GHzで約 $2 \mu s$ のコストがかかり、したがって、増益分岐点は $W = 125$ となる。おそらくセキュリティ保護されているMAC構成であるUMAC2/8を使用すると、PIII Katmai 600MHzで32バイトについてMACを生成するコストは $0.62 \mu s$ であり、損益分岐点は $W = 403$ となる。このメッセージサイズでは、UMACはCBCモードでRC6を使用してMACを計算する。UMAC2/8は、当面すべての複製に再利用できる8バイトと、複製毎に用意される8バイトタグを持つ。したがって、合計64バイトをログに記録するが、秘密鍵共有実施形態よりもわずか48バイト多いだけである。

30

【0158】

復旧後、秘密鍵分割実施形態とオーセンティケータベースの実施形態は両方とも、該当するデータすべてをサーバーすべてに伝送する。送出される量は両方のスキームで等しいが、秘密鍵分割実施形態では、同じレコードを全員に送信し、オーセンティケータベースのスキームでは、異なるがサイズは等しいレコードを全員に送信する。前者は10台のサーバーのそれぞれにおいて秘密鍵を解読する必要がある。 $10/7$ の数については、コストは $25 \sim 30 ms$ の間であり、おそらく $10/4$ についてはより小さいであろう。オーセンティケータベースの実施形態では、各サーバーでファイルハッシュについてMACを計算する必要がある。

したがって、 $R = 1e-4$ とし、HMAC-SHA1を使用して復旧を考慮すると、損益分岐点は $W = 158$ となる。これは、 $R = 1e-5$ で $W = 128$ である。

40

【0159】

本質的に、秘密鍵分割実施形態のセットアップおよび復旧の計算オーバーヘッドと、主流の非キャッシュ書き込み経路におけるオーセンティケータベーススキームの計算オーバーヘッドは釣り合っている。増益分岐点は、MAC計算と秘密鍵共有に使用される技術によって変化し、また同じロック秘密鍵で認証された書き込みの回数および復旧から復旧までの間の書き込みの回数によっても異なる。

【0160】

結論

上の説明では構造的機能および／または方法論的ステップに固有の言語を使用している

50

が、付属の請求項で定められている発明は、説明した特定の機能またはステップに限られていないことは理解されるであろう。むしろ、特定の機能およびステップは請求されている発明を実施する例として開示されている。

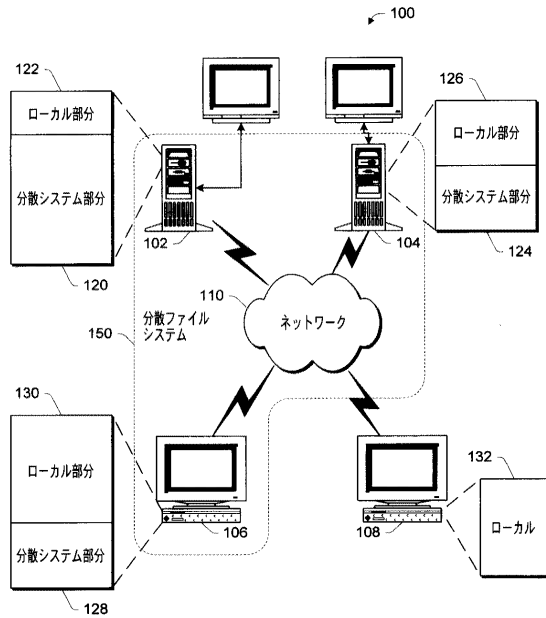
【符号の説明】

【0161】

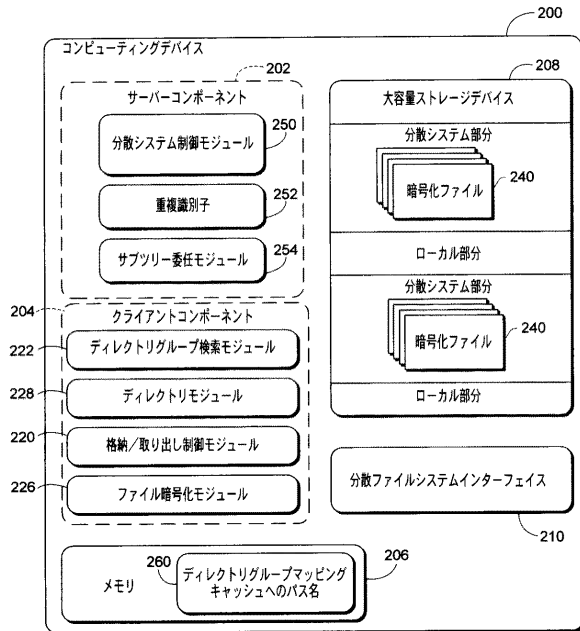
100	ネットワーク環境	
102	デバイス	
104	デバイス	
106	デバイス	
108	コンピューティングデバイス	10
110	ネットワーク	
120	分散システム部分	
122	ローカル部分	
124	分散システム部分	
126	ローカル部分	
128	分散システム部分	
130	ローカル部分	
132	ローカル部分	
150	分散ファイルシステム	
200	コンピューティングデバイス	20
202	サーバーコンポーネント	
204	クライアントコンポーネント	
206	メモリ	
208	大容量ストレージ	
210	分散ファイルシステムインターフェイス	
220	制御モジュール	
222	ディレクトリグループ検索モジュール	
226	ファイル暗号化モジュール	
228	ディレクトリ暗号化モジュール	
240	暗号化ファイル	30
250	分散システム制御モジュール	
252	重複識別器	
254	サブツリー委任モジュール	
300	コンピュータ環境	
302	コンピュータ	
304	処理ユニット	
306	システムメモリ	
308	システムバス	
310	ランダムアクセスメモリ (RAM)	
312	読み取り専用メモリ (ROM)	40
314	基本入出力システム (BIOS)	
316	ハードディスクドライブ	
320	取り外し可能な不揮発性磁気ディスク	
318	磁気ディスクドライブ	
324	取り外し可能な不揮発性光ディスク	
326	データ媒体インターフェイス	
328	プログラム	
330	プログラムモジュール	
332	プログラムデータ	
334	キーボード	50

3 3 6	ポインティングデバイス	
3 3 8	入力デバイス	
3 4 0	入出力インターフェイス	
3 4 2	モニタ	
3 4 6	プリンタ	
3 4 8	リモートコンピューティングデバイス	
3 5 0	ローカルエリアネットワーク (LAN)	
3 5 2	ワイドエリアネットワーク (WAN)	
3 5 4	ネットワークインターフェイスまたはアダプタ	
3 5 6	モデム	10
3 5 8	リモートアプリケーションプログラム	
4 0 0	名前空間	
4 0 2	ディレクトリグループ	
4 0 4	ディレクトリグループ	
4 0 6	ディレクトリグループ	
4 0 8	ディレクトリグループ	
4 1 0	ディレクトリグループ	
5 0 0	ファイルシステム	
5 0 2、5 0 4、5 0 6、5 0 8、5 1 0、5 1 2、5 1 4、5 1 6、5 1 8、5 2 0		
、5 2 2、5 2 4	コンピュータ	20
5 2 6	Byzantine グループ	
5 2 8	ファイル	
5 3 0	ディレクトリエントリ	
7 1 0	ファイル	
7 2 0	委任証明書	
7 5 0	複製	
9 0 0	ファイルシステム	
9 0 2、9 0 4、9 0 6、9 0 8、9 1 0、9 1 2、9 1 4	7 台のコンピュータ	
9 1 6	クライアントコンピュータ	
9 1 8	更新	30
9 2 6	Byzantine-fault-tolerant グループ	
9 3 0	ディレクトリエントリ	

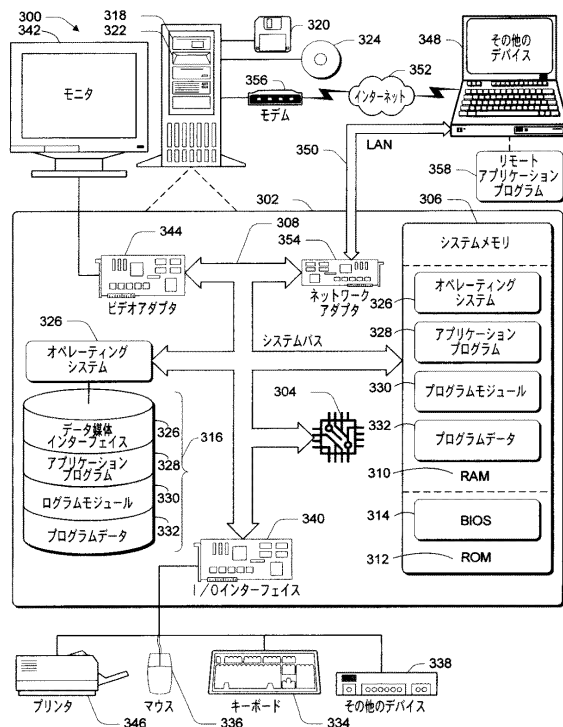
【図 1】



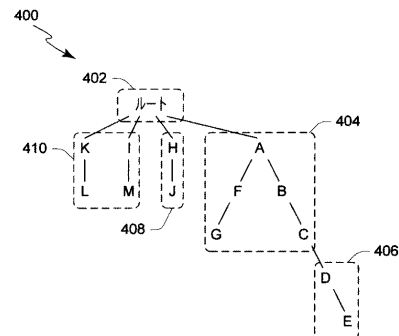
【図 2】



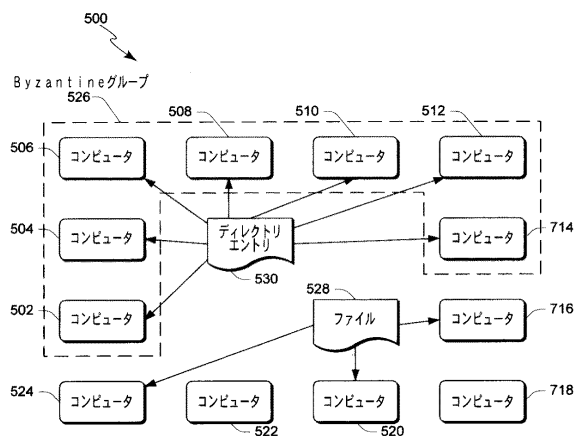
【図 3】



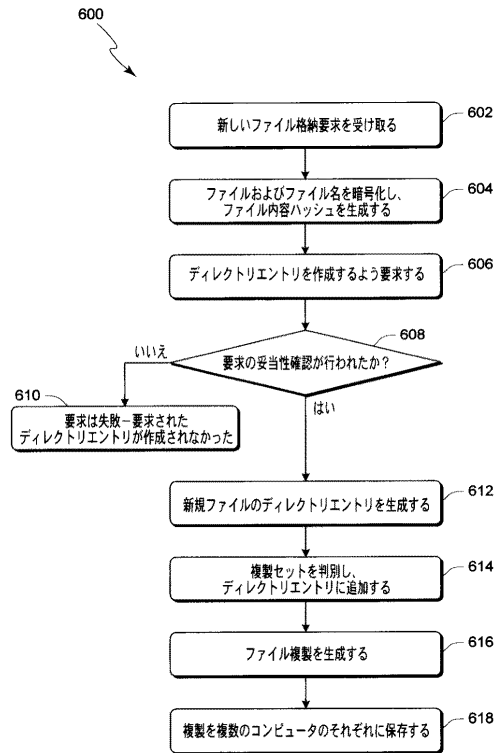
【図 4】



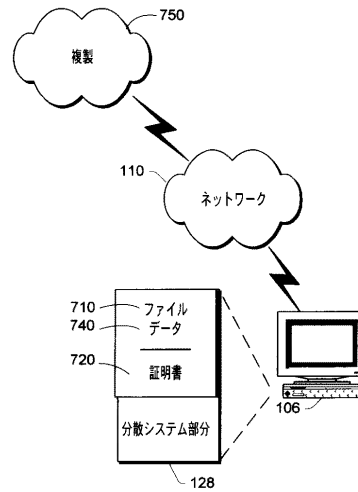
【図 5】



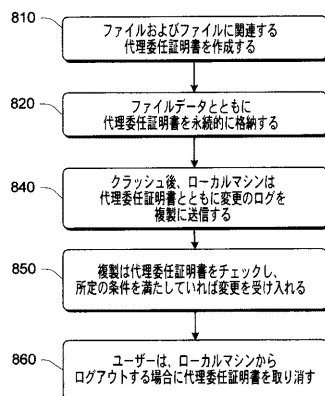
【図 6】



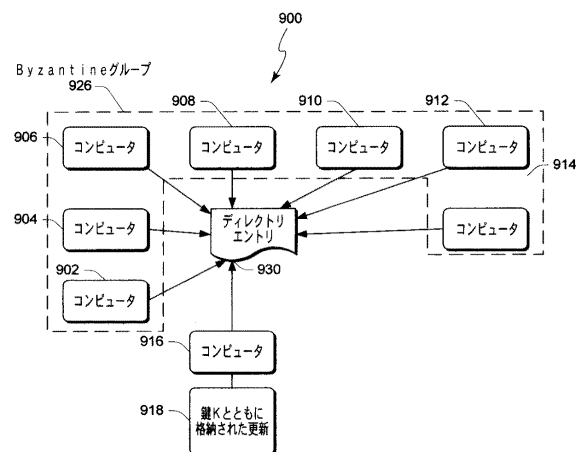
【図 7】



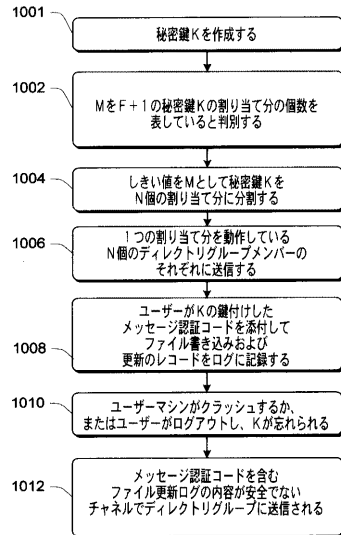
【図 8】



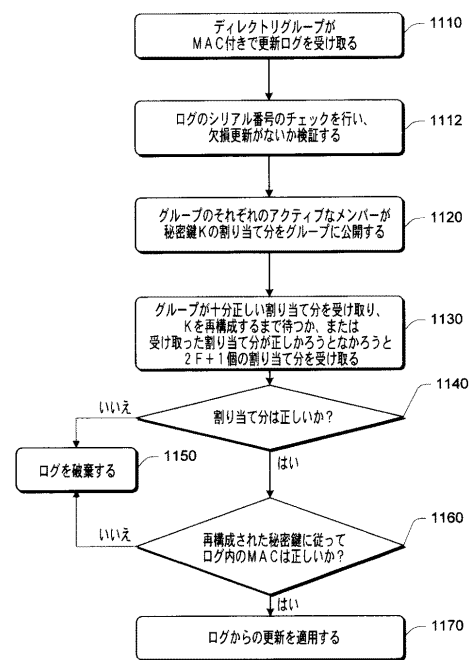
【図 9】



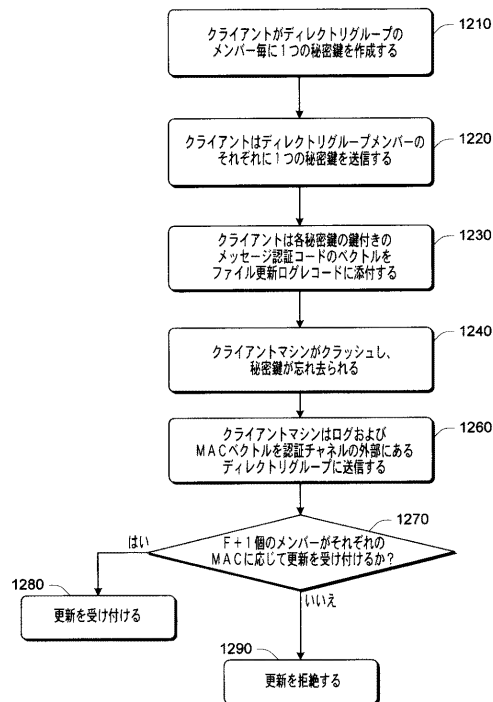
【図 10】



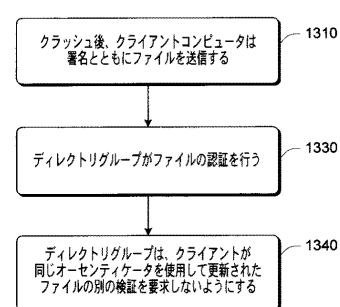
【図 11】



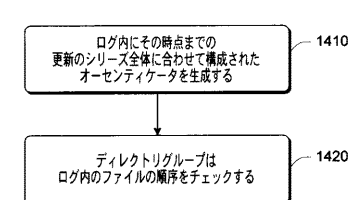
【図 12】



【図 13】



【図 14】



フロントページの続き

(51)Int.Cl. F I
G 0 9 C 1/00 6 4 0 D

(72)発明者 アトゥル アジャ
アメリカ合衆国 9 8 0 0 7 ワシントン州 ベルビュー ノースイースト 3 5 ストリート
1 4 5 9 0 アパートメント ディー 2 0 4

(72)発明者 ジョン アール. ドーサー
アメリカ合衆国 9 8 0 0 7 ワシントン州 ベルビュー ノースイースト 1 6 ストリート
1 4 7 0 5

(72)発明者 ウィリアム ジェイ. ボロスキー
アメリカ合衆国 9 8 0 2 7 ワシントン州 イサコア 3 1 6 プレイス サウスイースト 8
4 2 6

審査官 和田 財太

(56)参考文献 Miguel Castro and Barbara Liskov, Practical Byzantine Fault Tolerance, Proceedings of
the Third Symposium on Operating Systems Design and Implementation, 米国, 1 9 9 9 年
2 月, p.1-p.14

(58)調査した分野(Int.Cl., D B 名)
G 0 6 F 2 1 / 2 4
G 0 6 F 1 2 / 0 0
G 0 9 C 1 / 0 0
H 0 4 L 9 / 3 2