法政大学学術機関リポジトリ
HOSEI UNIVERSITY REPOSITORY

PDF issue: 2025-01-15

Intel SGXを利用したクラウド環境から保護された暗号化共有ファイルシステム

Kobayashi, Jun / 小林, 惇

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
17
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2022-03-24
(URL)
https://doi.org/10.15002/00025260

Intel SGX を利用したクラウド環境から保護された暗号化共有ファイルシステム
Encrypted and shared file system protected from cloud environments using Intel SGX

小林 惇 *
Jun Kobayashi
法政大学大学院　情報科学研究科　情報科学専攻

Abstract—**The platform of cloud services is a system consisting of huge physical machines owned by cloud service providers. Therefore, users of cloud services need to trust the cloud service provider to handle data on the cloud service or handle information protected by encryption or other methods. However, in the former case, there are incidents of privacy violation by cloud service providers, and in the latter case, conventional encryption methods cannot process encrypted data, which limits its functionality and causes delays in encryption and decryption.**

In this research, we propose a protected shared file system that uses the cloud service as storage by using Intel SGX, a type of TEE technology. Intel SGX provides an isolated execution environment by deploying an encrypted memory area called an Enclave. Only encryption-protected data and Enclave-protected applications exist on the cloud service, and the file system can be mounted by the local environment by the formal owner. In addition, it can be mounted in multiple local environments and other SGX applications, and treated like shared storage.

1. 導入

クラウドサービスは情報システムの構築の迅速さや運用コストの低さという利点から IT サービスを提供する事業者に広く普及し、重要な ICT 基盤になっている。クラウドサービスのプラットフォームはクラウドサービス事業者が保有している巨大な物理マシンで構成されたシステムであり、リソースや名前空間を分離し、ユーザーに割り当てることでサービスを提供している。そのため、クラウドサービスの利用者はクラウドサービス事業者を信頼してデータをクラウドサービス上で扱うか、暗号化等の手法で保護された情報を扱う必要がある [1] 。しかし、前者はクラウドプラットフォームが侵害された場合やクラウドサービス事業者に悪意がある場合、ユーザーのデータが侵害される恐れがある。後者は従来の暗号化方式では暗号化されたデータに対して処理ができずクラウドサービスが提供する機能の利用に制限が課されることや暗号化と復号による遅延が発生する課題がある。このような、クラウド環境で扱われるセンシティブなデータのセキュリティやプライバシーに対する関心は高まってきている [2] 。

暗号データをクラウドサービスに置くためには、暗号化を行う場所が重要となる。クライアントで暗号化を行う場合、鍵データと平文のファイルはクライアント上にあり、クラウドサービス上には暗号化データのみが置かれる。これによって保護は実現するが、他のクライアントからファイルを利用する場合は保護された手段で鍵データを共有するコストが発生する問題がある。クラウドサービス上で暗号化する場合、鍵データはクラウドサービス上にあるため、任意のクライアントから利用が可能である。しかし、鍵データはクラウドサービス上に置かれ保護されない。

この問題を解決するために、ホスト環境と分離して処理を行う TEE(Trusted Execution Environment) という技術が登場した。TEE は CPU や専用のチップといったハードウェアが隔離実行環境を提供することで、クリティカルな処理を保護したまま行う事が可能である。これにより、脆弱性によって特権を奪われた OS や信用できないプラットフォームからデータやランタイムを保護することができる。Intel SGX(Software Guard Execution) [3] は TEE 技術の一種で、対応している Intel CPU に組み込まれた命令セット群である。Intel SGX はエンクレーブと呼ばれる暗号化されたメモリ領域を生成して分離実行環境を提供する。Intel SGX を使用した先行研究はあるが、その暗号化処理を実施する環境の場所により利便性と情報の保護に問題点がある。

本研究では Intel SGX が提供する隔離実行環境を利用して、クラウドサービスをストレージとして利用する保護された共有ファイルシステムを提案する。エンクレーブ内に鍵データを置くことにより、クラウドサービス上には暗号化データとエンクレーブによって保護されたアプリケーションのみがある。これにより、情報を保護したまま正式なユーザーのみがファイルシステムを利用することができる。また、クラウドサービスを主体として動作するため、任意のクライアントから共有ファイルシステムが可能利用である。

2. 関連研究

2.1. NeXUS

NeXUS [4] は TEE を利用した暗号化共有ファイルシステムである。クライアント上に TEE の実行環境を展開し、その内部でデータの暗号化やファイルアクセス権限の検証を行ってからクラウドストレージに保存することで保護された共有ファイルシステムを提供している。ローカルのファイルシステム API で受けた命令を TEE で実行されるアプリケーションで処理してからストレージ API に送るため、サーバーサイドのサポートが必要ないという特徴がある。また、クライアント間で TEE を用いた検証を行うことで、クライアントが正当であるか検証し、複数の安全なユーザーのみが共有ファイルシステムを利用できる。

2.2. TrustFS

TrustFS [5] はスタッカブルファイルシステムに TEE を用いた処理コンポーネントを容易に組み込むことが可能な FUSE ベースファイルシステムのフレームワークである。ファイルシステム操作を FUSE カーネルモジュールが取得して複数のモジュール化された処理層で処理を行うようにファイルシステムを構成することで柔軟なファイルシステム開発を可能にする SAFEFS を元に、処理層に TEE　で実行されるアプリケーションを

* Supervisor: Prof. Toshio Hirotsu

組み込むことをサポートする。処理層ではプロキシを介して TEE を用いた処理コンポーネントにリダイレクトすることができ、コンテンツを考慮した安全なストレージ機能をもつモジュール化されたスタッカブルファイルシステムを容易に実装することを可能にしている。

2.3. 問題点

これらの研究では、TEE を利用することで信頼できないクラウド環境上でのデータの保護を可能にしている。しかし、利便性や安全性にいくつか問題点がある。NeXUS はクライアントに展開された TEE でデータを暗号化してクラウド環境に渡すことで保護を実現している。クライアントが TEE 対応である必要があり、復号に使用されるデータを別の保護された手段で共有する必要があるため、利用できるクライアントに制約がある。また、通常環境と比較して処理が重い TEE をクライアントで実行するため、クライアントのリソースを消費する。TrustFS はリモート環境内の FUSE の処理を TEE で実行されるアプリケーションにプロキシすることで保護をしているが、クライアントからプロキシに至るまでのアプリケーションや通信路は保護されておらず、ホスト環境から攻撃を受ける恐れがある。これらの問題点を解決するために、本研究では任意のクライアントと安全な手段で接続され、クラウド環境の内の保護された領域で実行される共有ファイルシステムを提案する。

3. Intel SGX

Intel SGX は Intel CPU がエンクレーブと呼ばれる隔離実行環境を提供する技術である。エンクレーブは CPU によって生成された暗号化メモリ領域であり、専用の命令セットでのみアクセスが可能なため、内部に展開された情報は外部から取得することはできない。また、作成時に生成する署名情報を元にエンクレーブコードとその実行処理の保護が保障される。このようなセキュリティモデルにより、Intel SGX はマルウェアや悪意のあるホスト等のソフトウェア攻撃とメモリを対象としたハードウェア攻撃に対して耐性を持つ。

Intel SGX の機能を利用したアプリケーション、SGX アプリケーションは信頼コンポーネントと非信頼コンポーネントに分離して実装する必要がある。信頼コンポーネントとはエンクレーブ内で実行されるコンポーネントである。保護したいリソースや処理機能、ライブラリを信頼コンポーネントに置くようにアプリケーションの実装を行うことで、これらを保護したまま実行することができる。しかし、エンクレーブはユーザーモードで実行されており、保護のために任意のシステムコールを発行できないという特徴がある。システムコールが発生する処理を、エンクレーブを使用しない非信頼コンポーネントに分けて実装することで SGX アプリケーションに組み込むことができる。

エンクレーブへのアクセスは専用の ECALL、OCALL というシステムコールでのみ許可されている (図 1)。ECALL とはエンクレーブのインターフェイスを呼び出すシステムコールで、OCALL はエンクレーブが外部のインターフェイスを呼び出すシステムコールである。このシステムコールを利用することで 2 つのコンポーネントを連携させることができる。開発者は連携のために、これらのインターフェイス定義を明示することと、非信頼コンポーネントにおける保護を実現するアプリケーション設計が求められる。

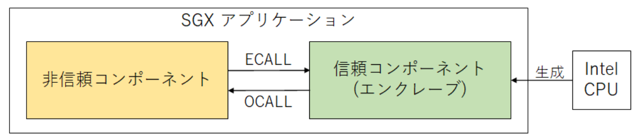

図 1. Intel SGX アプリケーションモデル

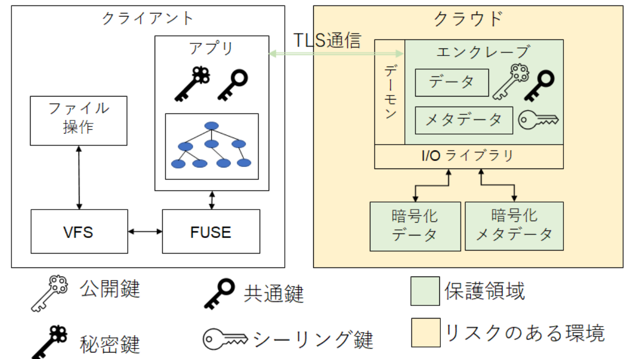

図 2. 設計

エンクレーブは信頼できない環境で実行されるため、エンクレーブがなりすまし攻撃を受ける可能性がある。そのため、外部アプリケーションはアクセス前にエンクレーブが正当であるか検証を行う必要がある。Intel SGX はエンクレーブの正当性検証のための機能として、アテステーションを提供している。アテステーションはエンクレーブ内に置かれた公開鍵とクライアントに置かれた秘密鍵を用いて実行され、エンクレーブが正当な Intel CPU によって生成されていることと、エンクレーブが想定された環境であることを検証する。

データを永続化するためには揮発性メモリから非揮発性メモリに情報を保存する必要があるが、情報を保護したまま非揮発性メモリに保存するためには暗号化処理が必要である。Intel SGX ではエンクレーブ内で情報を暗号化するシーリングという機能を持つ。また、シーリングされた情報を復号する機能をアンシーリングと呼ぶ。エンクレーブ内のデータをシーリングしてからストレージに保存することでデータの保護が可能である。シーリングに用いられる鍵はエンクレーブの固有情報を元にエンクレーブ内で生成される。そのため、シーリングされたデータをシーリングしたエンクレーブ以外で復号することはできない。

4. 設計

提案するファイルシステムは安全なクライアント上にあるクライアントアプリケーションと安全でないクラウド環境上にある SGX アプリケーションで構成される。クライアントアプリケーションはクライアント環境でユーザーと SGX アプリケーションの仲介を行い、SGX アプリケーションはファイルシステムの主体としてクラウドで動作して、保護された共有ファイルシステムを提供する。図 2 に構成を示す。

4.1. セキュリティモデル

本研究では安全なクライアント環境と安全でないクラウド環境を想定する。クライアント環境はユーザーが管理している環境であり、内部のアプリケーションやデータが攻撃を受けることはない。クラウド環境はユー

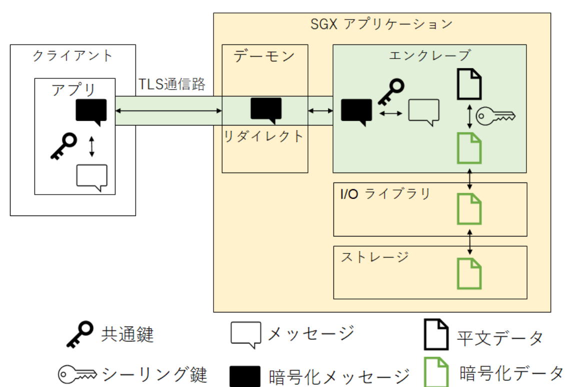

図 3. デーモンと I/O ライブラリの概要

ザー以外が管理する環境であり、内部のデータやアプリケーションはホスト環境や他のアプリケーションから攻撃を受ける可能性がある。クラウド環境上で保護されている領域は、エンクレーブ内部と暗号データである。また、クライアントアプリケーションと SGX アプリケーション間の通信は通信路は暗号化されていれば保護がなされている。共有ファイルシステムの正当なユーザーはエンクレーブに署名したユーザーとそのユーザーに許可された他のユーザーのみとする。つまり、本研究のモデルで平文ファイルを扱うことのできる箇所はクライアントアプリケーションとエンクレーブである。

4.2. SGX アプリケーションの分離

エンクレーブは一般的なシステムコールを利用できないため、通信とストレージアクセスの機能を非信頼コンポーネントに分離する。概要を図 3 に示す。デーモンは通信を行うコンポーネントであり、クライアントアプリケーションとセッションを生成して通信を行う。デーモンはメッセージやデータを受信するとエンクレーブが提示しているインターフェイスを介してエンクレーブにリダイレクトする。デーモンを通過するメッセージとデータは暗号化されており、デーモンでは平文データを取得することができない。つまり、通信はデーモンとクライアントアプリケーション間で行われ、暗号データの交換はエンクレーブとクライアントアプリケーションで行われる。エンクレーブのインターフェイスは汎化されており、インターフェイスの使用方法や引数からファイルシステムの動作を推測することは困難である。I/O ライブラリはストレージアクセスを行うコンポーネントである。エンクレーブがストレージアクセスを行う際に I/O ライブラリを呼び出して使用する。I/O ライブラリに渡されるデータはエンクレーブ内でシーリングによる暗号化がされており、I/O ライブラリは取得したデータからデータの平文を取得することはできない。

4.3. ファイルの暗号化とメタデータの隠蔽

クラウド環境のストレージにファイルを保存する際はファイルの暗号化と、ファイルデータの推測を避けるためにディレクトリ構成等のメタデータを隠蔽を行う必要がある。クライアントアプリケーションとエンクレーブの内部にのみ平文のファイルデータがあり、クラウド環境のストレージには平文のファイルデータから生成された暗号化データブロックと暗号化メタデータを置くことで、ファイルの保護と隠蔽を行う。エンクレーブ内部にファイルのメタデータとファイルから生成された暗号データブロックを紐付ける管理構造を実装し、この管理構造を使用してファイルの管理を行う。ストレージにファイルを保存する場合、まず、クライアントアプリケーションからエンクレーブにファイルが渡される。エンクレーブはファイル本体をデータブロックに分割して暗号化を行い、そのファイルメタデータを元に生成したハッシュをファイル名として、暗号データブロックの保存を行う。同時に、ファイルメタデータとハッシュの対応付けを管理構造に追記する。ストレージに保存されたデータを読み出す場合、ファイルメタデータと管理構造の対応付けから対象ファイルから生成された暗号データブロックを特定する。暗号データブロックはエンクレーブ内に読み込まれて復号処理が行われ、結合を行うことで平文のファイルが復元される。エンクレーブ内部の管理構造は暗号ファイルとしてストレージに逐次保存される。SGX アプリケーションの初期化時にこの暗号ファイルをエンクレーブにロードすることで終了以前と同様の状態を復元している。このようにクラウド環境上で暗号データを管理することで、クライアント環境の制約がなくなり、公開鍵を登録した任意のクライアントから同一の共有ファイルシステムを利用することが可能である。

4.4. ファイルアクセス方式

ファイルアクセスを行うインターフェースとして、ブロックベース方式とストリームベース方式を用意した。ブロックベース方式では、ユーザー空間ファイルシステムを定義できる FUSE を使用してクライアント環境に定義されたアプリケーションとクラウド環境の SGX アプリケーションが連携する共有ファイルシステムを提供する。概要を図 4 に示す。この共有ファイルシステムはクライアント環境にマウントされる。ファイルを OPEN すると、VFS が FUSE に命令を渡し、FUSE がクライアントアプリケーションに命令を渡す。クライアントアプリケーションはエンクレーブから対象となるファイルのファイルディスクプリタを取得する。クライアントアプリケーションはこのファイルディスクプリタを元にファイルハンドルを行う。READ によってデータブロックが参照される場合、アプリケーションがエンクレーブに参照されたデータブロックの情報を送信して、エンクレーブ内でデータブロックは復号され返信としてアプリケーションに取得される。WRITE でデータブロックが更新される場合、データブロックがエンクレーブに送信されて、エンクレーブ内の管理構造の更新と暗号化データブロックの更新が行われる。

ストリームベース方式では、クライアント環境上に置かれたアプリケーションとクラウド環境上に置かれた SGX アプリケーションで構成される。ユーザーはクライアントカル環境のアプリケーションを使用することで共有ファイルシステムにアクセスすることができる。ユーザーは対話型のインターフェイスを用いて、アプリケーションに任意のファイル命令と対象となるファイル情報を与えることで共有ファイルシステムに対して操作を行う。アプリケーションがファイル命令を発行すると、暗号化されたメッセージがリモートアプリケーションに送出され、エンクレーブ内で復号をしてメッセージを解釈して処理が行われる。また、処理の結果も暗号化メッセージとして返信されて、アプリケーションが復号を行い、メッセージの解釈と処理を行いユーザー結果を提示する。READ や WRITE といったファイルの読み込みや書き込みが発生する命令の場合、そのファイルはアプリケーションとリモートアプリケーション間の

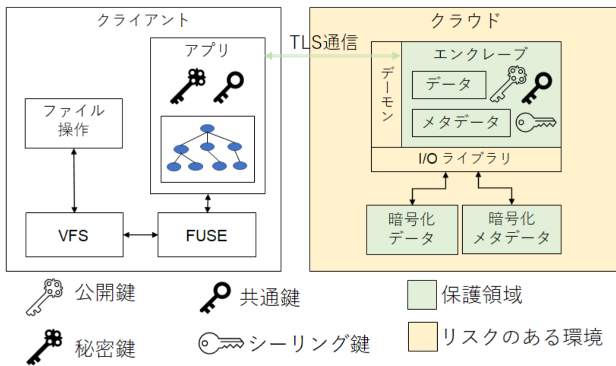

図 4. FUSE を用いたブロックベース方式

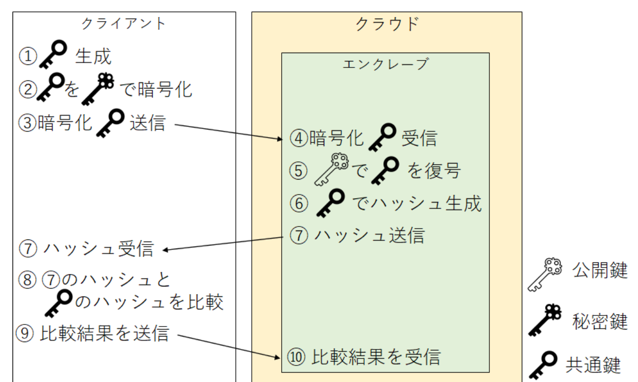

図 5. エンクレーブの正当性検証

TLS セッションを通じて、ストリーミングデータとして送信される。

4.5. 検証と暗号通信

クライアントアプリケーションがエンクレーブを利用するためには、エンクレーブの正当性検証と通信路の暗号化が必要である。エンクレーブの正当性検証は RSA 署名で行う。通信路の暗号化に使用する共通鍵の交換は RSA 暗号で行う。クライアントアプリケーションと正当なエンクレーブには、事前に生成されたキーペアを与える。4.2 章で説明したようにデーモンは非信頼コンポーネントであるため、暗号通信はクライアントアプリケーションとエンクレーブ間の保護を提供する必要がある。そこで、クライアントアプリケーションとエンクレーブ間で検証と鍵共有を行う独自のプロトコルを使用する。この検証と鍵共有の手順について図 5 に示す。最初にクライアントアプリケーションで AES 暗号方式の共通鍵を生成する。生成された共通鍵は秘密鍵で暗号化される。暗号化された共通鍵は SGX アプリケーションのエンクレーブに送信される。暗号化された共通鍵を受信したエンクレーブは埋め込まれた公開鍵を使用することで平文の共通鍵を取得する。エンクレーブは共通鍵のハッシュを生成して、それをクライアントアプリケーションに送信する。クライアントアプリケーションは、受信したハッシュと共通鍵のハッシュの比較を行う。比較の結果が一致していれば検証は正常に終了したとみなし、エンクレーブに結果を送信してファイルシステムを利用可能な状態に遷移する。遷移後は検証で共有された共通鍵を用いて TLS セッションで通信を行う。検証が不正であれば、エンクレーブが不正であるとみなしクライアントアプリケーションは終了する。

5. 実装

提案手法を、C++を用いて Ununtu20.04.3 LTS 上で開発した。ブロックベース方式は、FUSE 越しに通知されるファイル操作が、ストリーム方式で構成される通信セッションでのデータ転送に変換される。そこで、本節ではストリームベース方式の共有から説明する。

5.1. クライアントサイド

クライアントアプリケーションでは、検証を含む TLS セッションの生成処理とファイルアクセスの処理を行う。TLS セッションに関わる暗号化と復号処理は OpenSSL 1.1.1.11 [6] のライブラリを使用している。

クライアントアプリケーションを実行すると、検証を含む TLS セッションの生成処理が開始する。まず、TLS セッションを生成するために必要な AES 構造体と RSA 構造体を生成する。AES 構造体では共通鍵となる文字列を入力として、128 ビットの CTR モードで動作する暗号化と復号に使用される構造体をそれぞれ初期化する。RSA 構造体では、秘密鍵をファイルから読み込んで初期化を行う。次に RSA 構造体を使用して共通鍵を暗号化する。その後、ソケットを用いてリモートアプリケーションと検証セッションを開始する。検証セッションは 4.5 章で説明された通り処理され、内部のハッシュは SHA-256 シーリングに用いられる鍵はエンクレーブの固有情報を元にエンクレーブ内で生成される。検証の結果が正常な場合のみファイルアクセスの処理を行う状態に移行する。

ファイル命令は対話型のインターフェイスと通して発行される。ユーザーが OPEN 命令、 READ 命令、WRITE 命令に相当するコマンドとファイル名を入力すると、その内容に応じて処理が行われる。命令情報は共通鍵で暗号化され、エンクレーブに送信することで SGX アプリケーションに命令を処理するように通知する。通知後は SGX アプリケーションからの返信を待機して、受信したら命令に応じた結果をユーザーに提示する。READ 命令と WRITE 命令が入力された場合、通知を送信する通信に加えて、ファイルデータをストリーミングデータとして送受信を行う。

5.2. サーバーサイド

SGX アプリケーションはデーモンと I/O ライブラリとエンクレーブで構成される。エンクレーブはユーザーレベルで実行されるため、通信やストレージに対する処理は行うことができない。そのため、それらの機能を分割して実装を行っている。

エンクレーブは 4.5 章で説明されたローカルアプリケーションとの検証、 4.3 章で説明されたファイルシステムの暗号化と復号を行う。RSA 暗号方式と AES 暗号方式に関わる処理は Intel SGX SSL ライブラリ [7] を利用した。Intel SGX SSL ライブラリは、エンクレーブ内にホスト環境の OpenSSL を安全に利用できる API を提供するライブラリである。エンクレーブは初期化時に RSA 構造体とメタデータ管理構造の復元を行う。RSA 構造体はエンクレーブコードに直接記述された秘密鍵をロードして初期化される。メタデータ管理構造はファイル名やディレクトリ情報やハッシュ情報を保持する単方向リストとして定義される。初期時には、この管理構造の状態を保存した暗号ファイルを読み込み管理

構造の復元を行う。その後、検証を行い正常と判断されれば AES 構造体が初期化されてファイルアクセスを処理する状態に遷移する。

ファイルアクセスを処理する状態ではデーモンから取得した暗号メッセージを AES 共通鍵で復号して命令の内容を把握する。取得した命令によって、エンクレーブ内での処理と返信内容とデーモンの通信方法を決定する。取得したものが OPEN 命令であれば、管理構造を検索してファイルが存在するか確認する。その後、結果から暗号メッセージを生成して、メッセージだけを送信を指定するフラグとともにデーモンに渡して、ローカルアプリケーションに返信を行う。READ であれば、管理構造から対象となる暗号ブロックファイルを特定して I/O ライブラリを介して暗号ブロックファイルを読み込み、エンクレーブ内で復号する。その後、復号データからファイルを復元して共通鍵で暗号化する。暗号メッセージと暗号データサイズと暗号化されたファイルをフラグとともにデーモンに渡して返信を行う。WRITE であればデーモンに暗号データを受信するようにフラグを返す。その後、デーモンから受け取った暗号データを復号して平文データを取得し、ブロックへの分割とシーリングでの暗号化と管理構造の更新、I/O ライブラリを使用してストレージへの書き込みを行う。

6. 評価

本研究が提案する暗号化共有ファイルシステムの性能を示すために、既存の共有ファイルシステムと同様の環境と比較し、実環境を用いて 3 種類のファイル操作の評価を行う。既存の共有ファイルシステムとして NFS と同様の環境を使用する。 NFS は Kerberos 認証や SSL トンネルと合わせて使用することでセキュア化を図っている。通信は暗号化によって保護されているが、ファイルシステムサーバー内でデータは平文である。そのため、通信路の暗号化を行い、ファイルの暗号化を行わない共有ファイルシステムを比較対象とする。

1 つ目の評価としてクライアントアプリケーションから OPEN 命令を発行して結果が得られるまでの実行時間を計測し、暗号化共有ファイルシステムに存在するファイル数が与える影響について評価する。計測はクライアントアプリケーションに OPEN 命令を入力してからクライアントアプリケーションで結果が出力されるまでの時間で実施され、10000 回の平均実行時間を計算する。この評価では 100 個のファイルメタデータを所持する提案手法の暗号化共有ファイルシステムと 10000 個のファイルメタデータを所持する提案手法の暗号化共有ファイルシステムにおいて、その平均実行速度を計測する。OPEN 命令の対象となるファイルのメタデータは、管理構造の終端に保存されるようにした。これにより、OPEN 命令におけるファイル検索の最悪実行時間を計測することができる。表 1 に OPEN 命令の平均実行時間を示す。ファイルシステムが管理するファイル数が 100 の場合は 429.4 マイクロ秒であり、ファイル数が 10000 の場合は 444.9 マイクロ秒である。また、それぞれの平均実行時間の内、通信に使われた時間はそれぞれ 99.4% と 96.9% であった。ファイルシステムが管理するファイル数の増加によってオーバーヘッドが発生するが、全体の実行時間に対してごく僅かであり与える影響は小さい。

2 つ目の評価としてクライアントアプリケーションから READ 命令と WRITE 命令を発行して結果が得られるまでの実行時間を計測して、共有ファイルシステムに Intel SGX を導入することで発生するオーバーヘッドについて評価を行う。計測はクライアントアプリケーションに READ 命令または WRITE 命令を入力してからクライアントアプリケーションで結果が出力されるまでの時間で実施され、10000 回の平均実行時間を計算する。NFS と同等の環境を既存手法として比較する。提案手法は Intel SGX の提供するエンクレーブによるデータの保護を導入することで、ホスト環境からの攻撃を考慮してよりセキュリティレベルを向上させるファイルシステムである。4 KB と 8 KB のファイルについて READ 命令と WRITE 命令を生成して、その平均実行時間を計測することで、セキュリティレベルの向上とパフォーマンスのトレードオフを評価する。表 2 に計測結果を示し、表 3 にそれぞれの平均実行時間のうちで通信に使用された時間の割合を示す。ファイルサイズの増加よる遅延の増加とエンクレーブを導入することによるオーバーヘッドの発生が見られる。また、エンクレーブの導入によって、通信に使用される割合が低下して暗号化処理にかかる時間が増加していることが分かる。

表 1. OPEN 命令の平均実行時間

管理ファイル数	平均実行時間（μ s)	通信時間の割合 (%)
100	429.4	99.4
10000	444.9	96.9

表 2. READ 命令と WRITE 命令の平均実行時間

ファイル命令	サイズ (KB)	既存手法の実行時間（μ s)	提案手法の実行時間（μ s)
READ	4	1398.3	2201.3
READ	8	1406.6	2821.6
WRITE	4	1398.3	1782.3
WRITE	8	1434.6	1849.6

7. 考察

本研究では、Intel SGX の提供するエンクレーブによる保護を受けることで発生するオーバーヘッドについて実験を行い、オーバーヘッドが許容されるかを評価するために 6 章でエンクレーブが含まれる環境と含まれない環境でファイル操作命令の平均実行時間を評価した。1 つ目の評価では、提案手法におけるファイスシステムサイズが増加することによるオーバーヘッドの増加について示した。提案手法の暗号共有ファイルシステムではファイルのメタデータを単方向リストの管理構造で管理しており、OPEN 命令、READ 命令、WRITE 命令等の命令が実行されるたびに管理構造の探索処理が行われる。ファイル数が増加すると管理構造のエントリが増加するため、探索に遅延が発生する。一方で OPEN 命令における通信時間を除いた、クライアントアプリケーションと SGX アプリケーションの処理時間はそれぞれ 0.6% と 3.1 % 程度であり、 全体の実行時間においてごく僅かである。これにより、通信にかかる時間にオーバーヘッドが吸収されて、ファイル数が増加したとしてもユーザーに影響を与える可能性は低いと考えられる。2 つ目の評価では、共有ファイルシステムに Intel SGX を導入することによるオーバーヘッドの増加について示した。エンクレーブを導入することで、

表 3. READ 命令と WRITE 命令の通信時間割合

ファイル命令	サイズ (KB)	既存手法の通信時間 (%)	提案手法の通信時間 (%)
READ	4	99.8	63.4
READ	8	99.5	49.6
WRITE	4	99.8	78.3
WRITE	8	99.4	77.5

READ 命令では 4KB の場合に 57.4%、 8KB の場合に 100.6 % のオーバーヘッドが発生している。WRITE 命令では 4KB の場合に 27.5%、 8KB の場合に 28.9 % のオーバーヘッドが発生している。これは既存手法にはない、エンクレーブを用いたシーリングとアンシーリングといった暗号化機能やファイルの分割、管理構造の操作が追加されることで発生する遅延だと考えられる。また、ファイルサイズが大きいほどこれらの処理量は増加するため、オーバーヘッドも増加する。既存手法において、通信にかかる実行時間の割合は、99 % を超えており、提案手法においては 63.4 〜 78.3 % である。これは、エンクレーブを導入した場合においても実行時間において通信が占める割合が高い事を示している。この評価は同一ネットワーク内に配置されたマシンで行っているが、クラウドサービスを想定すると通信が占める割合がより高くなり、エンクレーブを導入することによるオーバーヘッドが与える影響が小さくなると考えられる。一方で、通信ではメッセージとファイルデータとファイルデータサイズを分離して交換されているため、1 つの命令処理で複数回通信が発生している。そのため、通信プロトコルを変更して通信回数を抑制することで平均実行時間の改善できると考えられる。

提案手法の READ 命令と WRITE 命令を比較すると、READ 命令でより大きなオーバーヘッドが発生していることが分かる。これは暗号に関する処理時間が発生するためである。エンクレーブ内にキャッシュを導入して、アンシーリングしたデータをキャッシュに登録することで、次回以降はキャッシュを参照することで実行速度を改善できると考えられる。キャッシュをエンクレーブに導入することは、シーリングとアンシーリングの頻度を減らして実行時間を改善する以外にも利点がある。提案手法の共有ファイルシステムはリモート環境上にファイル本体とメタデータを保持しているため、公開鍵の情報とクライアントアプリケーション保有して入れば任意の環境から利用することができる。つまり、多数のローカル環境やリモート環境の他のエンクレーブアプリケーションから共有ファイルシステムを利用するが可能となっている。頻繁にアクセスされるファイルについてのキャッシュをエンクレーブが保持していれば、この共有ファイルシステムにアクセスするユーザー全てが恩恵を受けることができる。一方で、ChongChong Zhao らの研究 [8] では、エンクレーブ内で膨大なメモリを確保して利用するとオーバヘッドが急激に増加することが示されている。そのため、エンクレーブに保持しておくキャッシュ量は実行速度の低下が許容される程度に抑えるように考慮しなければならない。

提案手法では、リモートアプリケーションのエンクレーブは秘密鍵で署名された共有オブジェクトファイルとしてリモート環境に置かれており、リモート環境をアプリケーションからこの共有オブジェクトを参照することでリモートアプリケーションを起動する。Intel SGX の提供するエンクレーブによってランタイムコードの整合性とデータの機密性は保たれるが、共有オブジェクトは保護しないため、コードの機密性は提供されていない。そのため、共有ファイルオブジェクトをリバースエンジニアリングすることで、エンクレーブコードおよび秘密鍵が漏洩する恐れがある。この問題に対して Intel SGX Protected Code Loader(PCL) が提供されている。Intel SGX PCL は共有オブジェクトを暗号化して生成する。また、暗号化された共有オブジェクトをエンクレーブ起動時に復号して読み込むことが可能となり、コードの機密性も保障することが可能になる。

現在の実装では、検証に使用される RSA 公開鍵はエンクレーブコードに直接埋め込まれている。RSA キーペアは定期的に更新することでセキュリティを向上することができるが、埋め込まれた RSA 公開鍵を変更するためにはエンクレーブの再生成が必要になる。シーリングのポリシーによっては、エンクレーブコードのハッシュが一致する場合、つまり完全に同一なエンクレーブでなければシーリングが行うことができないため、一度ファイルをすべて取得するなどの手間が生じる問題がある。この問題を解決するために、埋め込まれた RSA 鍵以外のキーペアを暗号化して登録と管理することのできる機構が必要である。

8. 結論

本研究では Intel SGX を用いた暗号共有ファイルシステムを提案した。提案した手法ではファイルシステムの主体をクラウド環境に配置し、信頼できる領域のみでファイルシステムを実行する。これにより、共有ファイアシステムを利用するクライアントの制約を回避し、クラウド環境からの攻撃から保護したまま共有ファイルシステムを利用することを可能にした。また、既存の手法と提案手法の実行速度を評価し、Intel SGX が提供する機密性とオーバヘッドについて考察してオーバヘッドが許容範囲であることを示した。今後の課題としてはキャッシュによる高速化やクラウド環境上のアプリケーションコードの保護を行うことが挙げられる。

参考文献

[1] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, “Depsky: Dependable and secure storage in a cloud-of-clouds,” *ACM Trans. Storage*, vol. 9, no. 4, nov 2013. [Online]. Available: https://doi.org/10.1145/2535929

[2] C. A. Ardagna, R. Asal, E. Damiani, and Q. H. Vu, “From security to assurance in the cloud: A survey,” *ACM Comput. Surv.*, vol. 48, no. 1, jul 2015. [Online]. Available: https://doi.org/10.1145/2767005

[3] Intel, “インテル® ソフトウェア・ガード・エクステンションズ (インテル® SGX),” https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/software-guard-extensions.html.

[4] J. B. Djoko, J. Lange, and A. J. Lee, “Nexus: Practical and secure access control on untrusted storage platforms using client-side sgx,” in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 401–413.

[5] T. Esteves, R. Macedo, A. Faria, B. Portela, J. Paulo, J. Pereira, and D. Harnik, “Trustfs: An sgx-enabled stackable file system framework,” in *2019 38th International Symposium on Reliable Distributed Systems Workshops (SRDSW)*, 2019, pp. 25–30.

[6] “OpenSSL,” https://www.openssl.org/.

[7] Intel, “Intel® Software Guard Extensions SSL,” https://github.com/intel/intel-sgx-ssl.

[8] C. Zhao, D. Saifuding, H. Tian, Y. Zhang, and C. Xing, “On the performance of intel sgx,” in *2016 13th Web Information Systems and Applications Conference (WISA)*, 2016, pp. 184–187.