Computer Security Symposium 2025
27-31 October 2025

TEEを利用した安全なファイル共有システムの形式的検証

西井 遥紀[†1,a)]　松浦 幹太[†1]

概要：複数のユーザー間で柔軟なアクセス制御のもと大量のファイルを共有する場面では，ファイル共有サービスが幅広く利用されている．しかし外部のサーバーに保存するファイルの機密性の保証を目的とした従来の暗号技術を用いた手法では，アクセス制御情報まで秘匿できず，また大規模ファイル群に対する共有操作のたびに性能低下を招くという課題がある．
本研究では TEE（Trusted Execution Environment）を活用した既存のファイル共有システムを拡張して，任意のストレージを利用して性能を犠牲にすることなくアクセス制御を秘匿化したファイル共有の手法を提案する．本システムにおいて共有に参加するユーザーは自身のマシン上に TEE を構築し，その内部で秘匿すべき共有操作を実行する．また，新規ユーザー参加時に既存ユーザーと行う一連のフローをモデル化し，形式的検証ツールである ProVerif を用いてシステムの安全性を検証した．

キーワード：TEE，形式的検証，ProVerif, ファイル共有システム

Formal Verification of TEE-based Secure File-sharing System

Haruki Nishii[†1,a)]　Kanta Matsuura[†1]

Abstract: File-sharing services on clouds are widely used in scenarios where a large number of files are shared among multiple users under flexible access control policies. However, conventional approaches that employ cryptographic techniques to protect the confidentiality of files stored on external servers cannot conceal access control information and suffer performance degradation when sharing operations are performed on large file sets.
In this paper, we propose a file-sharing system that leverages Trusted Execution Environments (TEEs) to hide access control policies without degrading performance using arbitrary storage backends. In this system, each participant deploys a TEE on their own machine and performs sensitive sharing operations inside it. Furthermore, we model the flow of interactions between an existing user and new user to join the sharing, and verify the security of the system using the formal verification tool ProVerif.

Keywords: TEE, formal verification, ProVerif, file-sharing system

1. はじめに

1.1 背景

クラウド技術の発展とともに，クラウド上に構築されたファイル共有サービスの利用は年々普及している．これらのサービスはファイルとディレクトリの階層構造からなる一般的なファイルシステムとしての機能を持つともに，共有に参加する利用者ごとにデータへの読み書きや削除といった操作に対する柔軟なアクセス制御を可能とする．しかしクラウドに機密性の高いデータを保存することは，偶発的なインシデントや外部の攻撃者によるデータの漏洩，悪意あるクラウドプロバイダーによるデータの利用や改竄につながることから，セキュリティ上のリスクやプライバシーの懸念を引き起こす．そのためこれらのユーザーにとって不本意なアクセスを防ぎ，ファイルを保護することが課題となる．またファイル共有においてはファイルの中身だけでなく，メタデータの保護も重要となる．メタデー

[†1] 現在，東京大学生産技術研究所
Presently with Institute of Industrial Science, The University of Tokyo
[a)] nishii@iis.u-tokyo.ac.jp

タにはファイル名，ファイルサイズ，ファイルの種類といった属性に加え，アクセス権限情報が含まれる．広く利用される AWS の EFS といったクラウドファイルシステムは暗号化によるユーザーのファイル保護を提供するが，ストレージ上のデータに対して特権を持つクラウドプロバイダはこれらを自由に操作できるため，ファイルの機密性や完全性を保証するにはクラウドプロバイダへの信頼が前提となるケースが多い．そこで近年は E2EE（End-to-End Encryption）によりクラウドへの信頼を一切必要とせず，ユーザー間でのみファイル共有を実現する取り組みが活発である．

従来の暗号技術の課題. ファイルの機密性と完全性を保証するファイル共有は共通鍵暗号と公開鍵暗号を併用した E2EE により簡潔に実現できる．まず共有に参加するユーザーごとに公開鍵 pk_i と秘密鍵 sk_i のペアを用意し，ファイル F ごとに Fresh な鍵 k_F を用いてファイルを暗号化する．次にファイルを共有する各ユーザーの公開鍵 pk_i を用いてこの鍵を暗号化して, 生成される暗号データ ck_i すべてを暗号化されたファイルとともにストレージに保存する．これによりストレージにアクセス権を持つ管理者であっても，ストレージに保存された暗号データからファイルを復号することはできず，共有を許したユーザーの秘密鍵 sk_i によってのみファイルの復号が可能となる．このような E2EE のファイル共有の仕組みは Poton Drive[1] 等で実用化されているものの，ファイル共有を失効するたびに再度 Fresh な暗号鍵 k'_F を生成する必要があり，ファイル数やファイルサイズに応じたオーバーヘッドが生じるという問題がある．またこの手法ではディレクトリの階層構造やファイルのアクセス制御情報といったファイルシステム内のメタデータの一部は秘匿できない．

TEE を利用したファイル共有と課題. 上記の問題に対処するために，データを保護した状態のまま計算を行う秘密計算の手法を取り入れた安全なファイル共有に関する研究が多くなされている．その中で秘密計算を実現する手法の一つであり近年注目を集めている TEE（Trusted Execution Environment）の技術を利用したファイル共有のシステムを構築する研究が複数ある．これらのシステムはメタデータを含めたシステム上のデータの機密性や完全性を保証するとともに，一般のファイル共有に劣らない性能でのファイル共有を目的とする．しかしこれらの研究で提案されるワークフローによるシステムの安全性に対して厳密な保証を与えているものは少なく，形式的な安全性の証明を与えることが望ましい．

1.2 貢献

本研究では既存研究にならい TEE を利用したファイル共有システムを設計するとともに，システムの安全性を検証自動化ツールである ProVerif を用いて形式的検証を行った．貢献は次のとおりである．

- ファイル共有に参加するユーザーのマシン上で TEE を利用するファイル共有システムを設計して，このシステムの脅威モデルと満たすべきセキュリティ特性を定義した．
- 既存ユーザーが新規ユーザーを共有に招待するフローを ProVerif によりモデル化し，対象のセキュリティ特性を満たすかを検証した．

2. 準備

本節では本稿で用いる技術とツールの説明を行う．

2.1 TEE (Trusted Execution Environment)

TEE はハードウェアの支援により隔離された信頼可能な実行環境，またはこれを構築する技術の総称である．この技術は，機密データを用いた計算の過程において，使用中のデータの保護を目的とする機密コンピューティング（Confidential Computing）[2] において主に利用されている．機密コンピューティングの普及を進める Confidential Computing Consortium による TEE の定義 [3] では，この隔離された実行環境内において，最小要件としてデータの機密性，データの完全性，コードの完全性が保証され，コードの機密性や Attestability（実行環境の構成を第三者に証明できるという属性）等の追加の性質を満たす．ハードウェアの最下層で依存関係を最小限に抑えてセキュリティを提供することで，OS や VM のハイパーバイザー，デバイスドライバのベンダー，計算処理のプラットフォームや周辺機器のベンダー，サービス提供者を信頼できるコンポーネントから除外することが可能となる．一般的な TEE の実装では CPU に搭載された拡張機能を用いて実行環境の隔離を実現している．CPU の拡張命令セットを用いて物理メモリ上に信頼可能とする領域を作成し，信頼可能な領域（TEE 内）と信頼不可の領域（TEE 外）の間での実行の移動は CPU の実行モードの切り替えにより行われる．この実行モードに応じて TEE 内のデータやコードへのアクセスを制御することで隔離領域の保護を実現する．

また実装の多くは Remote Attestation と呼ばれる TEE の構成が正規のものであることを第三者に証明するプロセスを提供する．このプロセスは，RFC 9334 [Remote ATtestation procedureS (RATS)][4] において，自分の環境が信頼できる状態にあることを外部の主体に証明するための共通のフレームワークとして定義されている．TEE における Remote Attestation で，TEE を用いたアプリケーションを起動するプラットフォーム（Attester）が，TEE 内のコードを含む TEE の構成情報と Root of Trust と呼ばれる CPU 固有の値を基に生成される Evidence を作成することから始まる．この Evidence を CPU ベンダなどの検証者（Verifier）とのフローを通して，証明すべきユー

ザー（Relying Party）に提示して，自身のプラットフォーム上の TEE の真正性と完全性を証明することが最終的な目的である．TEE の実装には Intel 製 CPU に搭載された Intel SGX[5]，Intel TDX[6] や AMD 製 CPU に搭載された AMD SEV-SNP の他，Arm TrustZone や RISC-V 向けに提案された Keystonen などがある．これらはそれぞれプロセスや VM といった異なる範囲の実行環境を隔離対象として，ユースケースに応じた Remote Attestation の手法を提供する．また TEE の技術は MPC（Multi-party Computation）や準同型暗号といった秘密計算を実現する他の手法と比較して，Remote Attestation の検証者となるトラストアンカーが存在する反面，複雑な計算に対しても一般的な CPU と同じ速度で処理が可能であるといった特徴がある．

2.2 ProVerif

暗号プロトコルの安全性の形式的検証で使用されるモデルにはシンボリックモデル（Dolev-Yao モデル）[7] と計算モデルの二種類が存在する．シンボリックモデルモデルはプロトコルのフローや満たすべき安全性を固有の記号学的な記法で表現するモデルである．このモデルにおいて暗号プリミティブはブラックボックスの関数として，参加者間でやり取りされるメッセージは関数上の項として表現され，攻撃者は公開された通信路上のメッセージを自由に制御できるものとして想定される．攻撃者が通信路で得られた値に対して，定義された暗号プリミティブを用いて計算を行うことで，与えられたセキュリティ特性が満たされない可能性があるかどうかによってプロトコルの安全性を評価する．

ProVerif[8] は Bruno らによって開発されたシンボリックモデルを利用した暗号プロトコルの安全性の検証自動化ツールである．無制限のプロトコルの実行セッションに対して，秘匿性，認証，観測上の等価性といったセキュリティ特性の完全に自動化された検証を可能とする．ProVerif の記述は暗号プリミティブの宣言，プロトコルのフローを表すプロセス，満たすべき安全性を示すクエリからなる．拡張π計算によって記述されたプロトコルを Horn 節に変換した上で，クエリ形式で与えられた攻撃可能性を自動定理証明器の手法で探索する．この手法は近似的であるため，攻撃が不可能であると判定された場合は実際に安全であるが，攻撃可能と判定された場合でも実際に攻撃可能かは保証されず偽陽性の可能性がある．そのため確率的多項式時間（PPT）の攻撃者に対して優位性があるか否かで安全性を検証する計算モデルと比べて，厳密な保証は与えられないが，効率的に脆弱性を検出することに特化している．

3. 関連研究

本節では TEE に関する形式的検証や TEE を利用したファイル共有を扱った関連研究を示す．

3.1 TEE を利用したファイル共有システム

前述の従来の E2EE によるファイル共有の問題に対処するため，ファイルやメタデータの機密性と完全性を保証するとともに，共有失効を含めた各処理のパフォーマンス向上を目的として，Djoko らは TEE を利用した共有ファイルシステムである NeXUS[9] を提案した．このファイルシステムは任意のストレージサーバーと，TEE の実装の一つである Intel SGX を搭載したマシンを保有する複数のクライアントで構成される．各クライアントはマシン上に構築した保護領域内でクライアントの認証，暗号化された状態でストレージに保存されるファイルやメタデータの復号，アクセス制御といった処理を行う．またファイル共有に参加しているクライアントが他のクライアントを共有に招待する際は，Remote Attestation を通して互いの TEE の構成を検証したうえで，ファイルシステムへのアクセスに必要な情報を共有する．これにより共有に参加する全クライアントが前述の処理を適切に行う TEE を起動することが保証される．また TEE の特性により，TEE 内部で扱われる認証やアクセス制御に関する情報はマシンを保有するクライアントであっても取得できないため，複雑な暗号処理を要することなく秘匿性の高い共有を実現した．

また Burke らによって提案, 開発された BFS[10] は TEE を活用したセキュアかつ高速なクラウド共有ファイルシステムであり，ファイルおよびメタデータの機密性と完全性を TEE で保護することを目的とする．NeXUS と異なりクラウド上で TEE を展開し，クライアントが接続時に Remote Attestation により TEE の構成を検証することで，クラウドプロバイダを信頼することなくファイルシステムへの安全なアクセスを可能とする．本研究ではこれらの設計を参考に，参加するユーザー間での安全なファイル共有を実現するシステムを構築する．

3.2 TEE を利用したアプリケーションの形式的検証

TEE に関連する形式的検証も数多くなされている．その一つに Google の TEE 開発フレームワークである Asylo のプロジェクトにおける，EKEP（Enclave Key Exchange Protocol）[11] の ProVerif を利用した形式的検証 [12] がある．EKEP は互いに TEE を展開したクライアント・サーバー間での計 6 メッセージからなる鍵交換プロトコルである．このプロトコルは Remote Attestation を通した Ephemeral DH 鍵交換によるセキュアな通信路の確立を目的としている．検証では相互認証や確立した通信路上で送受されるデータの秘匿性に加え，前方秘匿性（PFS）やリプレイ攻撃の防止といった高度なセキュリティ特性も対象とする．また Remote Attestation そのものの安全性に関する研究も複数存在する．例として Sardar[13] らは TEE の実

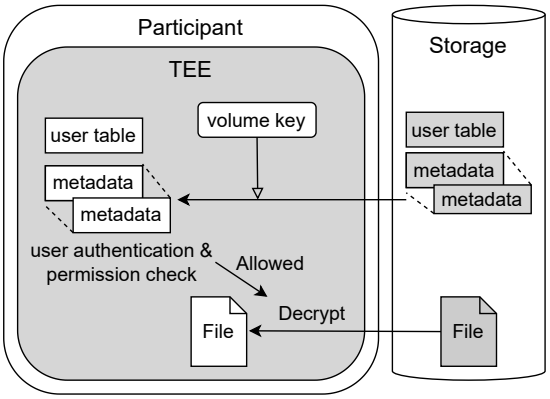

図 1 ファイル共有システムの設計
Fig. 1 Design of file-sharing system

装である Arm CCA と Intel TDX を対象として，Remote Attestation のフローを定式化し，ProVerif や Tamarin といったツールを用いて形式的検証した．本研究ではこれらの先行研究で用いられる TEE の ProVerif によるモデルを参考にして，Remote Attestation を含む一連のプロトコルフローおよびセキュリティ特性のモデル化と形式的検証を目指した．

4. 設計

本節ではファイル共有システムの構成および想定すべき脅威について扱う．

4.1 構成

システムの構成を**図 1** に示す．本ファイル共有システムは先行研究である NeXUS にならい，共有に参加するユーザーのマシン上で TEE を展開するものとする．またストレージはファイルを格納できる任意のものを想定する．システムに登場するエンティティは以下である．

参加者: ファイルの共有に参加するユーザー．各参加者は自身のマシン上で TEE を展開する．またユーザーが新規に共有に参加するには，既存の参加者とのフローを通して招待される必要がある．

ストレージ: ファイルを保存するエンティティ．共有の参加者を含めた任意のユーザーが接続可能とする．ファイルの作成，変更，削除といった基本的な操作を可能とするが，アクセス制御を含む追加の操作は要求しない．

検証者: Remote Attestation のプロセスにおいて，TEE の Evidence を検証するエンティティ．TEE の実装の多くは CPU のベンダーを指す．

またストレージには以下のデータを格納する．

ファイル: 共有するファイル本体．各ファイルは後述のメタデータに含まれる鍵によって認証付き暗号化される．また本稿では対象としないが，ファイルサイズを秘匿するために，固定長のブロック単位で分割することも可能である．

メタデータ： ファイル共有システム内でファイル本体以外の付随するデータ．ボリューム鍵と呼ばれるシステム全体で利用される鍵によって認証付き暗号化される．各ファイル・ディレクトリの名前を含む階層構造や，アクセス制御に関する情報が含まれる．またファイルを暗号化するための鍵もここに含まれる．

ユーザーテーブル： 認証やアクセス制御に用いられる，共有参加者の情報を格納するテーブル．各エントリは参加者の識別子となるホスト名，公開鍵と署名からなりボリューム鍵で暗号化される．

参加者は自身のマシン上に構築した保護領域内でユーザーの認証やストレージ上のファイルやメタデータの暗号化と復号，アクセス制御といった操作を行う．全参加者がこれらの操作を適切に行っていることが要求されるため，新規参加者を招待する際には互いに保護領域の完全性を検証する必要がある．

4.2 脅威モデル

本ファイル共有システムは複数の種別の攻撃を想定して，ファイルおよびメタデータの機密性と完全性を確保することを目的とする．始めにサーバー上のストレージに対する特権を所持する管理者が，悪意をもってストレージに操作を加えることを想定する．この操作はストレージ内のファイルの読み書きや，削除，置換に加え，共有参加者とストレージ間のネットワークでやり取りされるメッセージの取得や改ざんといった管理者が可能な任意の操作を含む．また共有に参加するユーザーが悪意をもって，アクセス権限を逸脱したファイルやメタデータの読み書きを試みる場合も想定する．加えて共有に参加していない外部のユーザーがマシン上に悪意のあるコードを含む TEE を展開して，生のファイルやメタデータを取得，改ざんしようとする状況も考慮する．

また，本研究では攻撃者や参加者が公開された通信路上のメッセージや保護領域を出入りするデータをセッション終了後も保持することを想定する．これは TEE の実装に固有の問題等により，セッション終了後に保護領域内でのみアクセス可能にすべき機密データが漏洩する可能性を考慮してのものである．特に TEE は実装ごとに隔離された保護領域内の値を漏洩させる攻撃がたびたび報告されている．そこでこのような状況に対処し保護領域内のデータの一部が危殆化した場合も，他の秘匿すべき情報が復元できないという前方秘匿性（Perfect Forward Secrity）が保証されることを目指す．

ただし本研究ではシステムの実装に固有の問題については考慮の対象としないものとする．例として TEE を用い

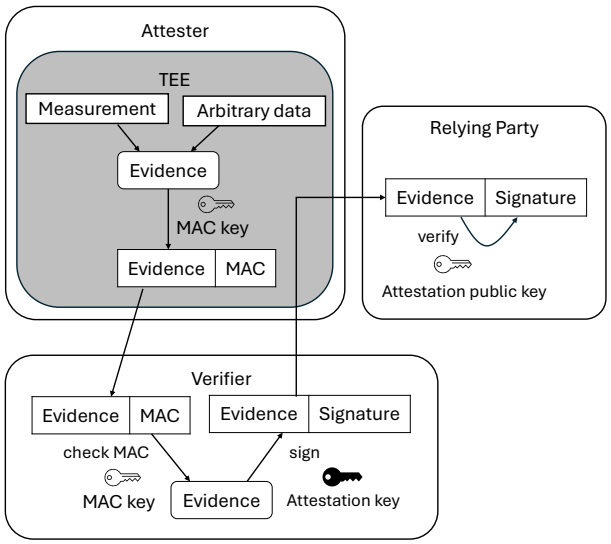

図 2 Remote Attestation のフロー
Fig. 2 Flow of remote attestation

```
let VerifierForNewcomerA(NewComerMacKey:
    mackey, AttestationPrivKey: sskey) =
  in(c, (evidence: bitstring, tag: mactag));
  let (=NewcomerKind, _: bitstring) =
      evidence in
  if macverify(evidence, NewComerMacKey, tag
      ) then
  out(c, sign(evidence, AttestationPrivKey))
      .

let VerifierForInviterB(InviterMacKey: mackey
    , AttestationPrivKey: sskey) =
  (* 省略*)

let VerifierForOther(AttestationPrivKey: sskey
    ) =
  in(c, arbitrary_data: bitstring);
  out(c, sign((OtherKind, arbitrary_data),
      AttestationPrivKey)).
```

図 3 Remote Attestation における検証者のプロセスの ProVerif コード
Fig. 3 ProVerif code of process of verifiers in remote attestation

たアプリケーションの実装によってはサイドチャネルにより保護領域内の値を推測する攻撃が成り立つ場合がある．しかしこれらは包括的に扱うのが困難であるため，本稿ではこれらを考慮しない．また，ストレージ管理者によるストレージへのアクセスの遮断や保存されているファイルの削除，または攻撃者による DoS 攻撃といった可用性を損なう攻撃についても対象としないものとする．

5. ファイル共有システムの安全性検証

本節ではファイル共有システムの安全性の形式的検証を行う．新規ユーザーが共有に参加するフローおよび満たすべきセキュリティ特性をそれぞれ ProVerif によりモデル化し，検証を実行した．

5.1 Remote Attestation のモデル

TEE の Remote Attestation は RFC 9334 に従い，TEE を起動する証明者（Attester），証明すべき相手ユーザー（Relying Party），検証者（Verifier）の三者からなるプロセスとしてモデル化する．なお一般的に検証者は CPU のベンダを指すが，TEE の実装によっては複数に分かれる場合がある．例として Intel SGX や Intel TDX における Remote Attestation[6], [14] は Intel が作成する Quoting Enclave と呼ばれる保護領域を証明者のマシン上に構築して行う．この Quoting Enclave 内部で Evidence に相当する Quote と呼ばれる構造体を作成し，これを受け取ったユーザーが Intel の提供する検証サービスや証明書チェーンを使ってこれを検証する．本研究では一般化のために，この Quoting Enclave も含めて一つの検証者として扱う．

TEE の Remote Attestation のフローを**図 2** に示す．始めに証明者は TEE 内部で TEE の構成情報を表す測定値や任意のデータを含めた Evidence を作成する．次にこの真正性を証明すべき Evidence に MAC 鍵を用いて MAC を付与して検証者に送信する．この MAC 鍵は事前にマシンごとに異なる TEE の Root of Trust（CPU 製造時に焼かれた値など）から生成され，証明者と検証者の間でのフローを通して外に漏れない形で共有される．検証者は受け取った Evidence から MAC 鍵を用いて MAC を検証したうえで，Attestation 鍵を用いて Evidence に署名を加えて証明者に返す．Attestation 鍵は検証者のみが保有する秘密鍵であり，対応する公開鍵は誰でも取得できるため，Evidence を証明者から受け取った相手は署名を検証することで Evidence の真正性と完全性を確認できる．

検証者を表すプロセスの ProVerif による記述を**図 3** に示す．コード中の `in`，`out` はそれぞれ公開された通信路 `c` 上でのデータの受信と送信を表す．Evidence には証明すべき任意の値に加え，TEE の測定値や起動するマシンを識別する値が含まれ，証明者から Evidence と MAC を受け取った検証者はこの値を確認して対応する MAC 鍵で検証を行う．そのためプロトコルに登場する証明者の TEE の種類の数だけ対応する検証者のプロセスを記述する必要がある．また，前述の悪意を持った TEE を持つユーザーがいる脅威モデルに対応するために，異なる識別値を含む任意の Evidence に対して Attestation 鍵による署名を行う検証者のプロセスも同様に定義した．

5.2 ユーザーの共有参加のモデル

次にユーザーが新しくファイル共有に参加するフローを**図 4** に示す．このフローは新しく共有に参加する新規参加者（Newcomer），既に共有に参加している招待者（Inviter）および Remote Attestation における検証者の間で行われる．フローの目的は招待者と新規参加者が互いの TEE の

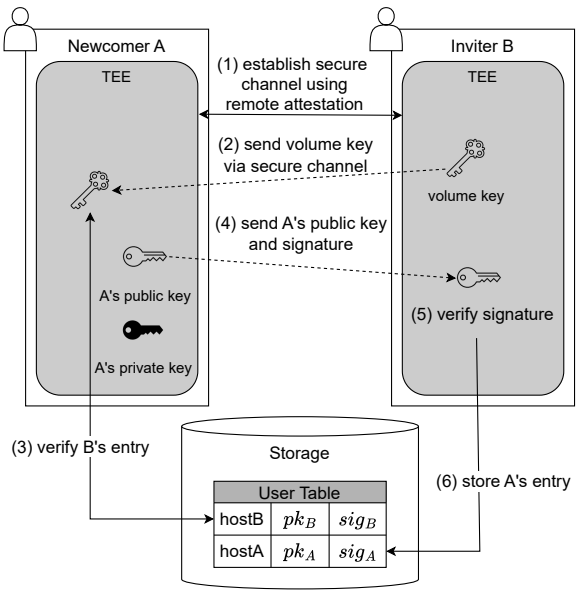

図 4 新規ユーザーがファイル共有に参加するフロー概略（検証者は省略）

Fig. 4 Flow of new user to participate file sharing (verifiers omitted)

真正性と完全性を検証したうえで，両者間でファイル共有に必要な情報を共有することにある．これは主に次の情報からなる．

- 参加者の公開鍵：参加者がファイルアクセス時に認証やアクセス制御に利用する鍵．新規参加者は共有の参加時に TEE 内で署名方式の秘密鍵と公開鍵を作成する．このうち秘密鍵は所持者の TEE 内でのみアクセス可能であり，公開鍵は参加者の TEE 内で共有される．
- ボリューム鍵：ストレージに格納されるユーザーテーブルやメタデータを復号する鍵．TEE 内でのみアクセス可能で，全参加者の TEE 内で共有される．

共有参加時の新規参加者と招待者の TEE 内におけるプロセスの Remote Attestation を行うフローの ProVerif による記述を**図 5** に示す．始めに新規参加者は Ephemeral DH 鍵ペアを作成し，自身のファイル共有における識別子となるホスト名と生成した DH 公開鍵を Evidence として MAC を付与して検証者に送信し，署名付きの Evidence を得て招待者に提示する．招待者はこれを受け取ると，署名とマシンの識別子を検証するとともに，自身も Ephemeral DH 鍵ペアを生成し，前述した Evidence の内容に受け取ったメッセージのハッシュ値を加えたものを Evidence として，同様に検証者に送り，署名付きの Evidence を新規参加者に提示する．新規参加者は同様に署名とマシンの識別子を検証し，互いに自身の DH 秘密鍵と受け取った DH 公開鍵から後の通信に利用する共有鍵を作成する．

```
let newcomerA(hostA: host, MacKeyA: mackey,
    AttestationPubKey: spkey) =
    (* 省略*)
    new skeA: Z;
    let pkeA = exp(g, skeA) in

    let MessageA = (NewcomerKind, (hostA, pkeA
        )) in
    let TagA = mac(MessageA, MacKeyA) in
    out(c, (MessageA, TagA));

    in(c, SignA: signature);
    let (=MessageA) = checksign(SignA,
        AttestationPubKey) in
    out(c, SignA);

    in(c, SignX: signature);
    let MessageX = checksign(SignX,
        AttestationPubKey) in
    let (=InviterKind, (hostX: host, pkeX: G,
        =SignA)) = MessageX in

    let SharedKey = G2nonce(exp(pkeX, skeA))
        in
    event newcomerTerm(hostA, hostX, SharedKey
        );
    (* 省略*)

let inviterB(hostB: host, MacKeyB: mackey,
    AttestationPubKey: spkey, ...) =
    (* 省略*)
    in(c, SignX: signature);
    let MessageX = checksign(SignX,
        AttestationPubKey) in
    let (=NewcomerKind, (hostX: host, pkeX: G)
        ) = MessageX in

    new skeB: Z;
    let pkeB = exp(g, skeB) in
    let SharedKey = G2nonce(exp(pkeX, skeB))
        in
    event inviterAccept(hostX, hostB,
        SharedKey);

    let MessageB = (InviterKind, (hostB, pkeB,
         SignX)) in
    let TagB = mac(MessageB, MacKeyB) in
    out(c, (MessageB, TagB));

    in(c, SignB: signature);
    let (=MessageB) = checksign(SignB,
        AttestationPubKey) in
    out(c, SignB);
    (* 省略*)
```

図 5 Remote Attestation における新規参加者と招待者のプロセスの ProVerif コード

Fig. 5 ProVerif code of process of new participant and host in remote attestation

次に招待者が新規参加者のファイル共有に用いる公開鍵を追加するフローの ProVerif による記述を**図 6** に示す．招待者は前述の共有鍵を用いてボリューム鍵を暗号化して新規参加者に送信し，ボリューム鍵を取得した新規参加者は，ストレージに格納されたユーザーテーブルから招待者のホスト名を引くことで暗号化されたエントリを得るため，これをボリューム鍵で復号する．このエントリは招待者の公開鍵と，ホスト名とマシンの識別子とボリューム鍵のハッシュ値に対する署名からなっており，署名を検証することで相手が正当な参加者であることを確認する．次に自身も鍵ペアを作成し，公開鍵と秘密鍵による同様の署名を共有

```
let newcomerA(hostA: host, MacKeyA: mackey,
    AttestationPubKey: spkey) =
    (* 省略*)
    in(c, encrootkey:bitstring);
    let (rootkey: nonce, rootmk: mackey) =
        sdecrypt(encrootkey, SharedKey) in
    event newcomerReceiveRootkey(rootkey);

    get user_entries(=hostX, entX: bitstring)
        in
    let (pkX: spkey, signX: signature) =
        sdecrypt(entX, rootkey) in
    let (=hostX, =InviterKind) = checksign(
        signX, pkX) in

    new skA: sskey;
    let pkA = spk(skA) in
    let signA = sign((hostA, NewcomerKind),
        skA) in
    out(c, sencrypt((pkA, signA), SharedKey));
    (* 省略*)

let inviterB(hostB: host, MacKeyB: mackey,
    AttestationPubKey: spkey, skB: sskey,
    ...) =
    (* 省略*)
    event inviterSendRootkey(rootkey);
    out(c, sencrypt((rootkey, rootmk),
        SharedKey));

    in(c, message1: bitstring);
    let (pkX: spkey, signX: signature) =
        sdecrypt(sdecrypt(message1, SharedKey)
        , rootkey) in
    if checksign(signX, pkX) = (hostX,
        NewcomerKind) then
    insert user_entries(hostX, (pkX, signX));
    (* 省略*)
let KeyRegister =
        in(c, (h: host, k: spkey, s: signature
            ));
        insert user_entries(h, (k, s)).

let KeyLeaker =
        in(c, h: host);
    get user_entries(=h, ent: bitstring) in
    out(c, ent).
```

図 6 新規ユーザーの共有追加の ProVerif コード
Fig. 6 ProVerif code of flow of registering new participant

鍵を用いて暗号化して招待者に送り，これを検証することでユーザーテーブルに新規参加者のエントリを追加して共有追加のフローを終える．また，前述の脅威モデルより，ストレージ管理者はユーザーテーブルを自由に取得，変更できるため，それに対応したプロセスもそれぞれ定義した．

5.3 安全性の検証

5.3.1 セキュリティ特性のモデル化・検証

上記で定義した全プロセスを並行実行する場合に，システムの安全性が満たされるか検証した．ProVerif において，検証すべきセキュリティ特性はクエリの形式で記述される．検証したクエリのコードを**図 7** に示す．各クエリはそれぞれ以下のセキュリティ特性に対応する．

新規参加者と招待者の相互認証 二者間での Remote Attestation を用いた鍵共有のフローを終えた時に，互いに相手が自分の想定する相手である必要がある．この認証に関するセキュリティ特性はプロセス中に出現する `event` の前後関係によって記述される．コード中の `inj-event(A) ==> inj-event(B)` はイベント `A` が発生したとき，それ以前にイベント `B` が発生していて，セッション数が複数ある場合にこの関係が単射であということを示す．図中の二つのクエリはそれぞれ招待者の新規参加者への認証と，新規参加者の招待者への認証を表し，これらの関係が常に成り立たないと，自分の想定しない相手と通信する可能性が生じ，認証の失敗を意味する．

```
(* Query 1 *)
query hostX: host, hostY: host, sk: nonce;
  (inj-event(newcomerTerm(hostX, hostY, sk))
       ==> inj-event(inviterAccept(hostX,
       hostY, sk))).
query k: nonce; inj-event(
    newcomerReceiveVolkey(k)) ==> inj-event(
    inviterSendVolkey(k)).
(* Query 2 *)
query attacker(new volkey).
(* Query 3 *)
query secret pkA.
query secret pkB.
```

図 7 安全性クエリの ProVerif コード
Fig. 7 ProVerif code of security queries

ボリューム鍵の秘匿性 ボリューム鍵は共有に参加するユーザーの TEE 内でのみアクセス可能な鍵であり，これを取得すればストレージ上のファイルやメタデータを復号できるため，決して攻撃者にわたってはならない．`attacker(A)` は攻撃者が公開された通信路上のメッセージを操作することで `A` を取得することはできないことを表す．

公開鍵の秘匿性 共有に参加する全ユーザーの公開鍵はストレージ上のユーザーテーブルに暗号化された状態で格納される．これを取得されてもファイルを復元，改ざんすることはできないが，アクセス制御に関する情報を秘匿するためにもこれらは機密であることが望ましい．

ProVerif の実行結果より，新規参加者の招待者への認証のクエリのみ `cannot be proved` と判定され，その他のクエリはいずれも `true` と判定された．そのためデータの秘匿性については満たされることが分かったが，認証に関しては ProVerif による完全な検証は達成できず，検証者に到達可能性の判断が委ねられる結果となった．

5.3.2 前方秘匿性の検証

セッション終了後に一部の機密データが漏洩した場合も，上記のセキュリティ特性が満たされるかについても検証した．ProVerif ではセッション後にデータ `data` が漏洩する状況は，上記の並行するプロセスに加えて，`(phase 1; out(c, data))` を加えることで再現される．これは `phase 0` となるセッション終了後に `data` を公開された通信路上に流すことを意味する．このデータ部分を変

え，以下の値が漏洩した場合の前方秘匿性を検証した．

Remote Attestation の MAC 鍵　新規参加者と招待者が Evidence を検証者に送る際に使用するそれぞれの MAC 鍵を漏洩しても，上記のセキュリティ特性の検証結果に変更はなかった．これは MAC 鍵が相互認証のために一時的に用いられるものであり，認証後にやり取りされるデータの秘匿性には影響がないことからも明らかである．

参加ユーザーの秘密鍵　招待者が自身の秘密鍵を漏洩した場合に，対応する公開鍵の秘匿性が破られる以外の結果に変更はなかった．また新規参加者の秘密鍵が漏洩した場合も同様の結果が得られた．そのため本システムにおいては，参加者の秘密鍵の漏洩により他の参加者の情報やボリューム鍵が漏洩する危険性がないことが検証された．

6. まとめ

本研究では TEE を利用したファイル共有システムを設計するとともに，新規ユーザーがファイル共有に参加するフローをモデル化し，ProVerif による安全性の形式的検証を行った．検証により本システムにおいて対象としたセキュリティ特性は一部判定できないものを除いて満たされることが確認された．TEE の技術は年々発達し，これを用いて安全なアプリケーションやプロトコルを開発するプロジェクトも数多く進行している．また複数のユーザー間で安全にファイルを共有するシステムの需要も充分にあると考えられるため，本研究が安全で広く利用されるシステムの構築の一助になることを期待する．

今後の課題としては，本研究においては値の秘匿性や認証といった最低限のセキュリティ特性のみを扱ったが，プロセス間の同値性の検証によるアクセス制御やデータの区別不可能性といった複雑なセキュリティ特性の実現にも焦点をあてて，さらなる安全なファイル共有システムの構築を目指したいと考えている．

謝辞　本研究の一部は，JST 戦略的創造研究推進事業（CREST）JPMJCR22M1 の支援を受けたものである．

参考文献

[1] Proton: The Proton Drive security model. `https://proton.me/blog/protondrive-security` [Accessed on: 2025-08-19] (2022).

[2] Consortium, C. C.: A Technical Analysis of Confidential Computing v1.3. https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf [Accessed: 2025-04-29], Technical report (2022).

[3] Consortium, C. C.: Confidential Computing: Hardware-Based Trusted Execution for Applications and Data 1.3. `https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC_outreach_whitepaper_updated_November_2022.pdf` [Last Access: 2025-04-29], Technical report (2022).

[4] Birkholz, H., Thaler, D., Richardson, M., Smith, N. and Pan, W.: Remote ATtestation procedureS (RATS) Architecture, RFC 9334 (2023).

[5] Costan, V. and Devadas, S.: Intel SGX Explained, Cryptology ePrint Archive, Paper 2016/086 (2016).

[6] Intel: Intel® Trust Domain Extensions. `https://www.intel.co.jp/content/www/jp/ja/content-details/856790/intel-trust-domain-extensions.html` [Last Access: 2025-07-19], Technical report (2025).

[7] Dolev, D. and Yao, A.: On the security of public key protocols, *IEEE Transactions on Information Theory*, Vol. 29, No. 2, pp. 198–208 (online), DOI: 10.1109/TIT.1983.1056650 (1983).

[8] Blanchet, B.: *Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif*, Vol. Now Foundations and Trends (2016).

[9] Djoko, J. B., Lange, J. and Lee, A. J.: NeXUS: Practical and Secure Access Control on Untrusted Storage Platforms using Client-Side SGX, *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 401–413 (online), DOI: 10.1109/DSN.2019.00049 (2019).

[10] Burke, Q., Beugin, Y., Hoak, B., King, R., Pauley, E., Sheatsley, R., Yu, M., He, T., Porta, T. F. L. and McDaniel, P.: Securing Cloud File Systems With Trusted Execution, *IEEE Transactions on Dependable and Secure Computing*, Vol. 22, No. 3, pp. 1976–1992 (online), DOI: 10.1109/TDSC.2024.3474423 (2025).

[11] Asylo: Enclave Key Exchange Protocol (EKEP). `https://asylo.dev/docs/concepts/ekep.html` [Accessed on: 2025-07-25] (2018).

[12] google/ekep analysis, G.: A Formal Analysis of EKEP. `https://github.com/google/ekep-analysis` [Accessed on: 2025-07-25] (2018).

[13] Sardar, M. U., Fossati, T., Frost, S. and Xiong, S.: Formal Specification and Verification of Architecturally-Defined Attestation Mechanisms in Arm CCA and Intel TDX, *IEEE Access*, Vol. 12, pp. 361–381 (online), DOI: 10.1109/ACCESS.2023.3346501 (2024).

[14] Intel: Supporting Third Party Attestation for Intel® SGX with Intel® Data Center Attestation Primitives. https://www.intel.com/content/www/us/en/content-details/671314/supporting-third-party-attestation-for-intel-software-guard-extensions-data-center-attestation-primitives.html [Last Access: 2025-07-19], Technical report (2019).