TEEと秘密分散法を用いた検証可能なSecure Key Release方式の提案

甘田 拓海[1,a)]　石倉 禅[1]　橋本 諒太[1]

概要：Trusted Execution Environment（TEE）は処理中のメモリを保護するが，故障や停止によってその状態は失われるため，TEE 上のアプリケーションが用いる暗号鍵などの秘密情報は TEE の外部で保管する必要がある．そのため，外部の鍵管理サービスが TEE のアテステーションを検証したうえで秘密情報を提供する Secure Key Release（SKR）という技術が実用化されている．しかし既存の SKR は，秘密情報を単一の事業者が保持するため事業者への信頼が必須であり，提供判断の正当性を利用者が検証できない．本稿では，SKR の提供判断をアプリケーションとは別の TEE 上で行い，秘密分散法を用いて秘密情報を複数のシェアに分割して外部へ預ける方式を提案する．提案方式では秘密情報の分散・復元と提供判断をその TEE 上でのみ行うため，いずれの単一主体も秘密情報を復元できない．さらに，利用者は，リモートアテステーションによって，しきい値やシェアの配布先などの設定および提供条件が正しいことを検証できる．また，これらの設定は要求される安全性に応じて任意に選択でき，複数の組織が参加するアプリケーションへの適用も可能である．

A Verifiable Secure Key Release Scheme Using TEEs and Secret Sharing

1. はじめに

近年，クラウド上で機密データを扱う用途が広がり，クラウドの内部犯や侵害されたクラウド基盤から利用者のデータを守ることが求められている [1]．そのため，AMD SEV-SNP [2] や Intel TDX [3] など，CPU が隔離した実行環境（Trusted Execution Environment, TEE）を用いて，処理中のデータもクラウド管理者から保護する Confidential Computing が普及しつつある．TEE は実行環境のメモリを暗号化し，ハイパーバイザやクラウド管理者による読み出しや改ざんを防ぐ．

そのため，ディスクの暗号化鍵や外部サービスのパスワードのように再起動後も使い続ける秘密情報は，TEE の外に保管しておき，起動のたびに TEE へ提供し直す必要がある．このとき，提供先が本当に期待どおりの TEE であるかを検証しなければ，攻撃者の用意した環境に秘密情報を提供してしまう．そこで，秘密情報を保管するサービスが，CPU の署名付きで示される実行環境の構成情報によって提供先を検証し（リモートアテステーション），条件を満たす TEE にだけ秘密情報を提供する Secure Key Release（SKR）が用いられている [4–7]．

しかし SKR では，秘密情報そのものをサービスが保管し，提供するかどうかの判断もサービスの内部で行われる．つまり，クラウド管理者から秘密情報を守るために TEE を使っているにもかかわらず，秘密情報の管理は結局，SKR サービスの事業者に委ねられ，利用者は事業者の判断を信じるしかない．事業者の内部犯やサービスからの漏洩があれば秘密情報はそのまま攻撃者に知られ，誤った判断で不正な環境に提供されても，利用者はそれに気付けない．さらに，複数の組織が 1 つのアプリケーションを共同で使う場合には，全組織が信頼できる運営者を 1 つに決めることが難しい．

これに対し，判断を行うサービス自体を TEE 上で動かせば，その判断が正しいことを外部から検証できる [8]．しかし，TEE の停止に備えて秘密情報の複製をどこかに保管する必要があり，その保管先への信頼が残る．この複製を複数に分けて別々の主体に保管させれば，1 つの保管先

[1] NTT コンピューティングテクノロジーセンタ
NTT Center for Computing Technologies, Musashino, Tokyo 180–8585, Japan
[a)] takumi.amada@ntt.com

ⓒ2026 Information Processing Society of Japan

から漏洩しても秘密情報は漏れない．

本稿では，秘密分散法 [9] を TEE と組み合わせ，いずれの単一の主体も信頼する必要がなく，かつ提供判断の正しさを検証できる SKR の方式を提案する．秘密分散法は，秘密情報をシェアと呼ばれる複数の値に分け，決めた数以上のシェアが集まったときにだけ復元することができる暗号技術である．提案方式では，シェアを集めて秘密情報を復元し，要求元を検証した上で秘密情報を提供する処理を，アプリケーションとは別の TEE の中で実現する．利用者とシェアの保管先は，この TEE が正しいプログラムを事前に合意した設定で動かしていることを検証でき，保管先は検証した場合にだけシェアを返すため，ホストがこの TEE を改ざんした場合，秘密情報は復元されない．TEE が停止してもシェアは保管先に残るため，秘密情報の複製を特定の主体に保管する必要はない．シェアの数と保管先は任意に選ぶことができ，各保管先が自身の基準で要求元を判断できるため，複数の組織が参加するアプリケーションにも適用できる．

我々は提案方式を AMD SEV-SNP 上に実装し，既存の SKR と比較して，実用上十分な性能を持つことを確認した．

以下，2 章では TEE，秘密分散法，SKR について述べ，既存の SKR の課題を整理する．3 章では提案方式の設計と安全性について述べる．4 章では実装について，5 章では評価について述べる．6 章で関連研究に触れ，7 章で本稿をまとめる．

2. 背景と課題

本章では，提案方式の前提となる TEE，秘密分散法，SKR について述べたうえで，既存の SKR が抱える課題と既存研究を整理する．

2.1 TEE

TEE は，CPU の拡張機能によって隔離された実行環境であり，その内部で処理中のコードとデータの機密性と完全性を，OS やハイパーバイザなどの特権ソフトウェア，およびそれらを操作できるクラウド管理者から保護する．具体的に TEE は，隔離と秘匿を実現するメモリ保護と，実行環境が期待した構成であることを外部から検証できるようにするリモートアテステーションの 2 つの機能を提供する．

メモリ保護. CPU 内のメモリ暗号化エンジンは，ソフトウェアから読み出せない TEE ごとの鍵を用いて，主記憶との間でデータを暗号化・復号する．主記憶上のデータは暗号文としてのみ存在し，ハイパーバイザや他の TEE からは内容を読み取れない．さらに CPU 内のメモリ管理ユニットは，アクセス制御によって，各ページの所有者以外からのアクセスやハイパーバイザによるページの差し替えを防ぐ．これにより，データは CPU 内部でのみ平文として扱われ，処理性能を大きく損なうことなくクラウド管理者からの保護が実現される．

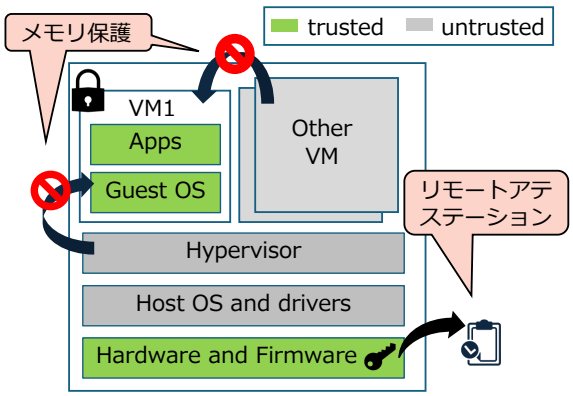

図 1 TEE アーキテクチャ（VM ベース）．

リモートアテステーション. 起動時に実行環境（CPU，ファームウェア，カーネルなど）の構成情報を測定し（その値を測定値と呼ぶ），耐タンパ性を持つ領域に保管された署名鍵でその構成情報に対して署名を行い（アテステーションレポートと呼ぶ），利用者に提供する．利用者は，ハードウェアベンダが提供する署名鍵に対応する検証鍵で署名検証を行い，構成情報が期待する実行環境であることを検証する．これにより，クラウド管理者による環境の改ざん等が行われていないことを確認する．

TEE のアーキテクチャは保護する対象によってプロセス型（Enclave 型）と VM 型の 2 つに分けられる．本稿では，既存のアプリケーションを変更せずに保護できる VM 型の TEE（Confidential VM，CVM）を対象とする（6.1 節）．VM 型 TEE の一般的なアーキテクチャの概略を図 1 に示す．

2.2 秘密分散法

秘密分散法 [9, 10] は，秘密情報を複数パーティで管理し，事前に定めたしきい値以上のパーティが協力した時のみ秘密情報を復元でき，しきい値未満のパーティによる悪用を防ぐ暗号技術である．秘密分散法の標準的な構成として (k, n) しきい値法 [9] が知られている．(k, n) しきい値法は，Share と Reconst の 2 つのアルゴリズムから構成される．

$(S_1, S_2, ..., S_n) \leftarrow \mathbf{Share}(S, n)$ ： ディーラと呼ばれる信頼できるパーティが秘密情報 S から n 個のシェア $S_1, S_2, \ldots, S_n$ を生成し，それぞれを n 個のパーティに配布する．

$S' \leftarrow \mathbf{Reconst}(\{(i, S_i)\}_{i \in I})$ ： n 個のシェアのうち，任意の添字の部分集合 $I \subseteq \{1, \ldots, n\}$ に対するシェアから，秘密情報の復元を試みる．

この時，正当性および安全性は以下の通りである．

ⓒ2026 Information Processing Society of Japan

正当性: $|I| \geq k$ ならば，秘密情報が一意に復元できる（$S' = S$）．

安全性: $|I| \leq k-1$ ならば，秘密情報を一意に復元することができない．

また，安全性において，秘密情報 S に関する情報が一切漏洩しないという強い性質を完全秘匿性と言う．本稿は Shamir の方式 [9] を用いるが，完全秘匿性を満たす他の (k, n) しきい値法でも構成できる．

2.3 Secure Key Release

本稿では，TEE のリモートアテステーションによって実行環境を検証し，所定の提供条件を満たす場合に鍵管理サービスが TEE に秘密情報を提供する仕組みを Secure Key Release（SKR）と総称する．

TEE 上のアプリケーション（以降，App-TEE と呼ぶ）が再起動後も使い続ける秘密情報は，TEE の外部に保管し，起動のたびに TEE 内へ戻す必要がある．ここで重要になるのは，秘密情報の提供先となる App-TEE が期待どおりの環境であることを検証したうえで戻すことである．これを実現する仕組みが SKR である．SKR の主なエンティティは，提供条件を定める利用者，秘密情報を要求する App-TEE，アテステーションレポートを検証する検証サービス，および秘密情報を保管し提供する鍵管理サービス（Key Management Service，KMS）の 4 つである．これらは，リモートアテステーションの標準的なアーキテクチャである RATS [11] の Relying Party Owner，Attester，Verifier，Relying Party にそれぞれ対応する．SKR の実装としては，Confidential Containers の Trustee [5] 等が知られている（6.3 節）．以下では Trustee を例に，その動作を説明する．

Trustee の主な構成要素は，KMS にあたる Key Broker Service（KBS），検証サービスにあたる Attestation Service（AS），および AS が照合に用いる参照値を保管する Reference Value Provider Service（RVPS）である．Trustee は主に，次の 3 つのフェーズで動作する（図 2）．

(1) 利用者は，秘密情報を KBS に，それを提供してよい App-TEE の測定値や TCB のバージョンなどを参照値として RVPS に登録する．本稿ではこれらをまとめて**提供条件**と呼ぶ．

(2) App-TEE は自身のレポートを KBS 経由で AS に提示する．AS はその署名を検証し，内容を RVPS の参照値と照合した結果を KBS に返す．

(3) KBS はその結果を用いて，提供条件を満たす場合に限り秘密情報を App-TEE に返す[*1]．

[*1] 正確には，KBS は結果をトークンとして App-TEE に返し，App-TEE がそのトークンを添えて改めて秘密情報を要求する．

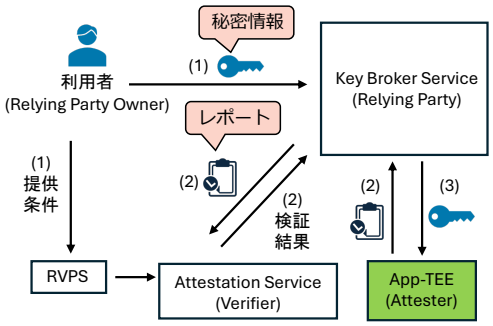

図 2 Trustee の概要．

2.4 既存 SKR の課題

既存の SKR には，秘密情報を保持する KMS の事業者への信頼が必須であること，および提供判断が正しく行われたかを利用者が検証できないこと，の 2 つの課題がある．ここで提供判断とは，検証サービスによるアテステーションレポートの検証から，KMS による提供条件との照合までの一連の過程を指す（前節の (2)(3) が該当）．

課題 1：KMS への信頼. 既存の SKR では，KMS は信頼を前提とするコンポーネントとして構成される．しかし，KMS は秘密情報を保管し，提供判断の結果にかかわらずそれを取り出せるため，その事業者の内部犯や，サービスからの漏洩によって秘密情報が悪用されても，利用者はそれを防ぐ手段を持たない．

課題 2：提供判断の不透明性. 既存の SKR は検証サービスも信頼を前提とするコンポーネントとしているが，提供判断が正しく行われたかを利用者が外部から検証する手段はない．もし検証サービスが不正なレポートを有効と判定しても，Relying Party である KMS は検証サービスの判定を信頼し，正当でない環境に秘密情報を渡してしまう．

上記の課題は，特に次の 2 つの状況で問題となる．

Confidential Computing (CC). CC の目的は，クラウド事業者を信頼せずに機密データを処理することである．しかし既存の SKR を利用する場合，その機密データを保護する秘密情報の管理と提供判断をクラウド事業者に委ねることになり，目的と合致しない．

マルチパーティ. 単一組織で App-TEE を利用する場合は，検証サービスと KMS を自組織の環境に構築するか，自組織が信頼するクラウドサービスを利用すればよい．一方で，複数組織で App-TEE を利用する場合，ある組織が構築した環境を他の組織は検証できず，信頼するクラウドサービスや提供条件も組織ごとに異なるため，全組織が合意できる単一の検証サービスと KMS を置くことが困難である．

2.5 既存研究の限界

Keylime [12] は，TPM を用いてクラウド上のノードにブートストラップ鍵 K_b を導入する方式であり，$K_b = U \oplus V$ と 2 つに分け，U を利用者が，V を検証サービス（Cloud

ⓒ2026 Information Processing Society of Japan

Verifier）が保持する．検証サービスは，ノードの TPM の測定値（PCR 値）を利用者の指定した期待値と照合してから V を渡し，利用者は TPM の識別のみを確認して U を渡す．これにより，検証サービスは利用者の意図なしに単独で鍵を復元できず，課題 1 に対する一つの解になっている．しかし，測定値の照合は検証サービスに集約されており，その提供判断を利用者や第三者が検証する手段はないため，課題 2 は残る．また，シェアの受信と結合をノード上の専用エージェントが行うため，秘密情報を要求する側（App-TEE に相当）に Keylime 固有の実装が必要である．

一方，Edgeless Systems の Contrast [8] は，鍵管理と提供判断を行うコンポーネント（Coordinator）を TEE 上に配置することで課題 2 に対処している．利用者は Coordinator のアテステーションを検証することで，提供条件が正しく設定され，提供判断がそのとおりに行われることを確認でき，Coordinator は正当な App-TEE にのみ秘密情報を提供する．しかし Contrast は，Coordinator の停止や故障に備えて，Coordinator が生成したシード（秘密情報の元）の複製を管理者と呼ばれる信頼できるパーティが保持することを前提としており，課題 1 は解決されていない．

3. 提案方式

本章では，2.4 節の 2 つの課題を解決する提案方式について述べる．

3.1 脅威モデル

本稿が想定する攻撃者は，TEE の脅威モデルと同様に，クラウド管理者の権限を持つ能動的な攻撃者である．攻撃者は，ゲストイメージや OS の差し替え，悪意のあるハイパーバイザによるメモリの読み出し，ストレージのマウントや改ざんなど，TEE の外部で可能なあらゆる操作を行える．さらに本稿では，検証サービスや KMS などのクラウドサービスも攻撃者の配下にあると仮定する．攻撃者の目的は，**(1) 秘密情報を取得すること**，または **(2) 提供条件を満たさない環境に秘密情報を提供させること**である．

一方，TEE のハードウェアによる隔離とアテステーションの署名は破られないものとし，サイドチャネル攻撃，可用性を侵害する攻撃，ハードウェアベンダ（Root-of-Trust）による攻撃，TEE 内部で動作する OS やファームウェアの脆弱性を利用した攻撃，および暗号プリミティブの危殆化は，TEE の脅威モデルと同じく対象外とする．安全性の詳細な議論は，3.5 節で行う．

3.2 設計方針

2.4 節の課題 1 と課題 2 をともに解決する方式を考える．

3.2.1 基本アイディア

提案方式は，提供判断を TEE 上に置く Contrast [8] の考え方と，Keylime [12] のように秘密を複数の主体に分けて単独では復元できないようにする考え方を組み合わせる．提供判断と秘密情報の分割・復元を行う TEE を SKR-TEE と呼ぶ．提案方式では，Contrast における Coordinator と，秘密分散法におけるディーラおよび復元を行う主体の役割を SKR-TEE が担う．SKR-TEE は，秘密情報を (k, n) しきい値法でシェアに分けてシェア保有者に配布する．App-TEE から要求があるたびに，SKR-TEE は App-TEE を検証したのち，k 個のシェアをシェア保有者から集めて TEE 内で復元し，提供する．この時，シェア保有者は，SKR-TEE を検証した上で，シェアを返却するか判断する．これにより，正当な SKR-TEE を除くいずれの単一主体も秘密情報を取得できず（課題 1），提供判断と提供条件を運用者以外も検証できる（課題 2）．

3.2.2 設計上の特徴

両者を組み合わせるにあたり，提案方式は次の 4 点を設計上の特徴とする．

分散条件の任意性. しきい値 k，シェアの分割数 n，および各シェアの保有者を，以降では**分散条件**と呼ぶ．Keylime の鍵分割は 2 者固定の XOR であり，配布先も利用者と検証サービスに固定されていた．提案方式はこれを (k, n) しきい値法と任意のシェア保有者に一般化する．保有者は，レポートの検証とシェアの送受信および管理ができる主体であれば良いため，様々な選択肢を取ることができる．また秘密分散のパラメータである n と k（$1 < k \leq n$）も任意に指定することができ，用途に応じて定められる．具体的な分散条件の例は 3.4 節で述べる．

シェア保有者による判断. 保有者は，SKR-TEE のレポートによって，SKR-TEE の測定値と，分散条件および提供条件が自身の合意した内容と一致することを確認したうえでシェアを返す．さらに SKR-TEE は，自身のレポートに加えて App-TEE のレポートも保有者に提示する．これにより保有者は，利用者が SKR-TEE に登録した提供条件とは別に，保有者自身の提供条件を App-TEE に課し，それを満たす場合に限りシェアを返すことができる．

SKR-TEE のステートレス化. Keylime では検証サービス自身がシェアを保持するため，検証サービスを常時稼働させ，シェアを失わないよう保全する必要があった．提案方式では，SKR-TEE 自身をシェア保有者に含めるか否かを分散条件で定められる．含めなければ，SKR-TEE は秘密情報とシェアのいずれも長期的には保持しない．残る分散条件と提供条件も，その正しさは保有者によるレポートの検証で保証されるため，SKR-TEE が保持する必要はなく，信頼できない外部のストレージに置いて要求のたびに取得する構成も取れる．この構成では SKR-TEE を常駐させる必要がなく，再起動後は設定を取得し直すだけで動作を再

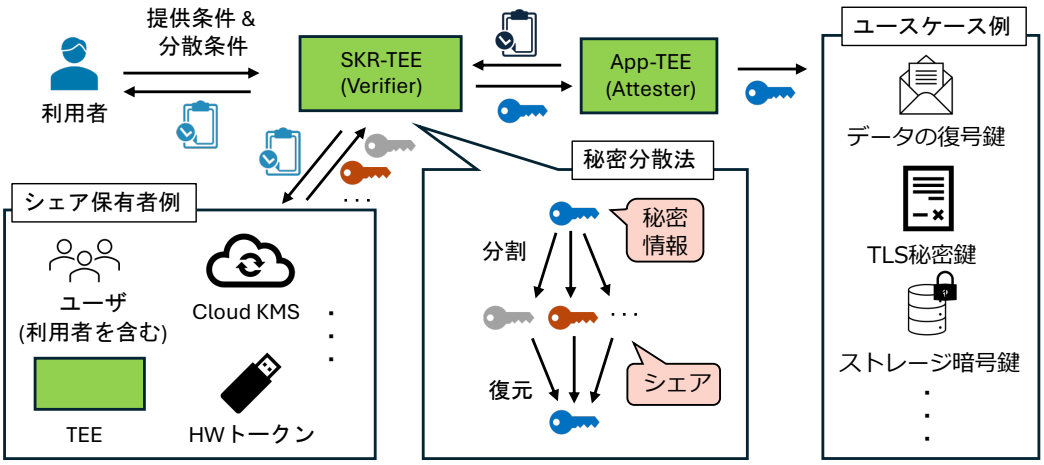

図 3 提案方式の構成.

開でき，負荷分散のための複製も容易である．

条件の動的な変更. 提案方式では，秘密情報を変えずに，分散条件と提供条件を運用中に変更できる．SKR-TEE が保有者からシェアを集めて秘密情報を復元し，新しい条件で分け直して配布する（3.3.4 節の Reshare）．これにより，保有者やしきい値を変更できるほか，シェアを失った保有者がいても，残りが k 個以上あれば n 個のシェアを持つ状態に戻せる．また，新しいシェアは新たな乱数で生成されるため，それまでに漏洩した $k-1$ 個以下のシェアは，新しいシェアと組み合わせても復元に使えない．

3.3 設計

提案方式の概要を図 3 に示す．

3.3.1 記法

$[n] = \{1, \ldots, n\}$ とする．$\mathrm{Enc}(pk, m)$ と $\mathrm{Dec}(sk, c)$ は公開鍵暗号による暗号化と復号，$\mathrm{H}(\cdot)$ は暗号学的ハッシュ関数，$\|$ は連結を表す．TEE X のアテステーションレポートを Report_X と書く．レポートには，起動時に定まる測定値のほかに，実行時に TEE 内のプログラムが指定した値を格納する領域（SEV-SNP の REPORT_DATA など）があり，これに格納された値をユーザデータと呼ぶ．値 $v_1, \ldots, v_m$ をレポートに含めるときは $\mathrm{H}(v_1 \| \cdots \| v_m)$ をユーザデータとする．検証者が Report_X で値 $v_1, \ldots, v_m$ を**検証する**とは，検証者が次のすべてを確認することをいう．

- 署名が TEE ハードウェアの証明書チェーンのもとで正しいこと．
- レポートの測定値が，検証者が保持する X の参照値（Reference Value）と一致すること．
- レポートのユーザデータが，検証者が $v_1, \ldots, v_m$ から計算した $\mathrm{H}(v_1 \| \cdots \| v_m)$ と一致すること．

検証に成功すれば，検証者の期待するコードが動作する X が $v_1, \ldots, v_m$ を含むレポートを発行したことが保証される．$\perp$ は失敗を表す．秘密情報 S はそれぞれ固有の識別子 ID_S を持ち，ID_S は Setup で SKR-TEE が乱数として生成する．SKR-TEE が保持する内部状態 state と別の主体からの入力 x から出力 y を得るアルゴリズム f を，$y \leftarrow f(x; \mathsf{state})$ と表す．

3.3.2 構成要素

提案方式のエンティティとその役割は以下のとおりである．

App-TEE: 秘密情報を利用するアプリケーションが動作する TEE である．既存の SKR と同じ手順（3.3.3 節）で秘密情報 S を要求する．

SKR-TEE: 提供判断と，秘密情報の分割・配布および復元を行う TEE である．利用者が登録した分散条件 $dc = (k, n, (P_1, \ldots, P_n))$ と提供条件 pol および識別子 ID_S を保持する．

利用者: SKR-TEE に dc と pol を登録し，$\mathsf{Report}_{\mathsf{skr}}$ によってそれらが正しく反映されたことを確認してから秘密情報を渡す主体である．

シェア保有者 P_i: SKR-TEE から配布されたシェア S_i を保管し，要求に応じて返却する．SKR-TEE および利用者を含む任意の主体を指定できる（3.4.1 節）．

3.3.3 前提

利用者が指定する分散条件 dc と提供条件 pol はシェア保有者とは事前に合意されており，SKR-TEE の測定値の参照値とともに共有されているものとする．条件を変更する場合も，新しい (dc', pol') は変更の前に利用者と各保有者の間で合意しておくものとする．秘密情報 S は，利用者や App-TEE が指定しても，SKR-TEE が内部で生成してもよい．前者の場合，秘密情報は SKR-TEE 内で生成された公開鍵で暗号化して送る．送信先が検証済みの SKR-TEE であることと機密性を保証できれば，RA-TLS [13] などの他の手段を用いてもよい．全コンポーネント間の通信路は TLS などで保護されているものとする．

App-TEE は，既存の SKR である Trustee [5] のクライ

ⓒ2026 Information Processing Society of Japan

アント [14] と同じ以下のインタフェースを持つものとする．

(1) 提供元に ID_S を指定して秘密情報を要求し，ナンス (nonce) を受け取る．
(2) 鍵ペア $(pk_{\mathsf{app}}, sk_{\mathsf{app}})$ を生成し，pk_{app} とナンスを含む自身のレポート $\mathsf{Report}_{\mathsf{app}}$ を提供元に提示する．
(3) 暗号文 ct を受け取り，$S \leftarrow \mathrm{Dec}(sk_{\mathsf{app}}, ct)$ によって秘密情報を得る．

3.3.4 プロトコル

提案方式は，(Setup, KeyRec, Reshare, Restore) の 4 つの対話アルゴリズムから構成される（Algorithm 1）．レポートごとに異なるナンス（乱数）r を含むことで，レポートの新鮮性（freshness）を保証する．SKR-TEE は各アルゴリズムにおいて利用した S，各シェア，鍵ペア等は消去し，秘密情報に関する状態を残さない（自身がシェア保有者である場合の自身のシェアを除く）．

Setup. 秘密情報ごとに初回に一度だけ実行し，利用者が SKR-TEE に分散条件 dc，提供条件 pol を登録して，秘密情報 S（S は SKR-TEE が生成する場合は省略する）をシェアに分けて配布する．dc と pol を SKR-TEE のレポートに含めることで，SKR-TEE 自身の測定値と両条件が TEE のハードウェアによる 1 つの署名で結び付けられる．利用者は，このレポートで dc と pol を検証することで，期待するコードが自身の指定した条件のもとで動作していることを確認する．S を指定する場合は 3.3.3 節の手段で送る．利用者は得た ID_S を App-TEE に設定する．各 P_i は，シェアを受け取る際にレポートを検証し，合意した条件 (dc, pol) で動作する SKR-TEE からのシェアだけを受け取って，ID_S と (dc, pol) を記録する．

KeyRec. App-TEE が秘密情報を要求するたびに実行し，SKR-TEE が App-TEE を検証したうえでシェアを回収し，秘密情報を復元して提供する．失敗した場合，$\perp$ を出力する．App-TEE 側の手順は 3.3.3 節のとおりである．SKR-TEE は新しい鍵ペア $(pk_{\mathsf{skr}}, sk_{\mathsf{skr}})$ を生成して各シェア保有者にシェアを要求し，ナンスを受け取って，その公開鍵とナンスを含むレポートを P_i に示す．P_i は，$\mathsf{Report}_{\mathsf{skr}}$ で ID_S，dc，pol，pk_{skr}，r_i，$\mathrm{H}(\mathsf{Report}_{\mathsf{app}})$ を検証し，シェアを pk_{skr} で暗号化して返す．このとき SKR-TEE は $\mathsf{Report}_{\mathsf{app}}$ も P_i に送り，そのハッシュ値を $\mathsf{Report}_{\mathsf{skr}}$ に含めることで差し替えを防ぐ．これにより P_i は，秘密情報の提供先が自身の提供条件を満たすかを独自に判断してもよい．不正なシェアによる復元結果の誤りの検出は本稿では扱わない (3.5.3 節)．

Reshare. 分散条件 dc，提供条件 pol を新しい分散条件 dc'，提供条件 pol' に変更する際に実行する．SKR-TEE は各 P_i に新旧両方の設定を含むレポートを示し，各 P_i はそのレポートで新旧いずれの設定も検証できた場合に限りシェアを返す．復元した秘密情報を新しい分散条件で分け直して新しい保有者 P'_j に配布した後，各 P_i は SKR-TEE の指示に従って古いシェアを削除する．

Restore. SKR-TEE の再起動や負荷分散のための複製の際に実行し，SKR-TEE に分散条件と提供条件を設定し直す．SKR-TEE は秘密情報に関する状態を持たないため，(ID_S, dc, pol) を設定し直すだけで KeyRec を再開できる．

3.4 分散条件の設計とユースケース

分散条件，すなわち k，n，およびシェア保有者は，要求される安全性と可用性に応じて利用者が設計する．本節では，分散条件の設計と，代表的な 2 つのユースケースにおける分散条件の例を示す．

3.4.1 分散条件の設計

分散条件の設計では，しきい値 k と，シェア保有者の種類が重要である．

しきい値 k. k は安全性と可用性の釣り合いを決める．k を大きくすると，漏洩や結託があっても秘密情報を復元できない保有者数の上限（$k-1$）が増えて安全性が高まる一方，復元に必要なシェア数も増える．k を小さくするとその逆になり，特に $k=1$ は単一主体で復元できる状態である．可用性の側から見ると，$n-k$ は紛失してもよい保有者の数であり，$k=n$ では紛失への耐性がない．たとえば $k=n$ で SKR-TEE が保有者を兼ねていれば，SKR-TEE の停止がそのまま秘密情報の喪失になる．

シェア保有者の種類. シェアに触れられる管理者と，応答性の 2 点が重要となる．保有者には任意の主体を指定できるため，複数のクラウドの KMS，検証サーバと連携した HSM やハードウェアトークン，他の組織のサーバなど，管理者の異なる主体を組み合わせればよく，利用者自身や SKR-TEE を加えることもできる．ただし，2 つの保有者が別の組織であっても，いずれも同じクラウドの KMS にシェアを保管していれば，そのクラウドの運用者は 2 つのシェアを取得できることに注意する．また応答性の観点からは，KeyRec には k 個のシェアが必要であるため，要求時に応答できる保有者が k 以上なければならない．応答性の低い保有者は，k を満たすための主要な保有者ではなく，紛失時のバックアップとして $n-k$ の側であることが望ましい．

3.4.2 ユースケース 1：単一組織での利用

単一の組織が App-TEE を利用する場合は，SKR-TEE 自身と KMS をシェア保有者に含めることで，通常の提供に利用者が介在せず，SKR-TEE の故障や停止にも対応できる構成となる．一例として，$k=2$，$n=3$ において，SKR-TEE，クラウドの KMS，利用者の三者をシェア保有者とする場合を考える．初回は，SKR-TEE をデプロイした後，利用者が Setup で dc と pol を登録して秘密情報を分割・配布する．以後の提供は SKR-TEE と KMS の 2 者で KeyRec が完結するため，利用者は介在しない．SKR-TEE が故障や停止によって保持するシェアを失っても，KMS と

Algorithm 1: 提案方式のプロトコル

$(\mathsf{ID}_S, \{S_i\}_{i\in[n]}) \leftarrow$ **Setup**(dc, pol, S)

1: **利用者**: dc，pol，r を SKR-TEE に送る．
2: **SKR-TEE**: 識別子 ID_S を乱数として生成する．
3: **SKR-TEE**: ID_S に対して dc と pol を自身に適用する．
4: **SKR-TEE**: ID_S，dc，pol，r を含む $\mathsf{Report}_{\mathsf{skr}}$ を生成して利用者に返す．
5: **利用者**: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証し，ID_S を得る．
6: **SKR-TEE**: S が未指定の場合は生成する．
7: **SKR-TEE**: $(S_1, \ldots, S_n) \leftarrow \mathsf{Share}(S, n)$ を行い，各 P_i に ID_S のシェアの受け取りを要求する．
8: P_i: r_i を返す．
9: **SKR-TEE**: ID_S，dc，pol，r_i を含む $\mathsf{Report}_{\mathsf{skr}}$ を取得し，S_i とともに P_i に送る．
10: P_i: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証し，S_i を保存する．

ct **or** $\perp \leftarrow$ **KeyRec**$(\mathsf{ID}_S;\ dc, pol)$

1: **SKR-TEE**: r を送り，pk_{app} と r を含む $\mathsf{Report}_{\mathsf{app}}$ を受け取る．
2: **SKR-TEE**: ID_S に登録された (dc, pol) を参照し，$\mathsf{Report}_{\mathsf{app}}$ で pk_{app}，r を検証して pol と照合する．ID_S が未登録か照合に失敗すれば $\perp$ を返す．
3: **SKR-TEE**: 鍵ペア $(pk_{\mathsf{skr}}, sk_{\mathsf{skr}})$ を生成し，各 P_i に ID_S のシェアを要求する．
4: P_i: r_i を返す．
5: **SKR-TEE**: ID_S，dc，pol，pk_{skr}，r_i，$\mathrm{H}(\mathsf{Report}_{\mathsf{app}})$ を含む $\mathsf{Report}_{\mathsf{skr}}$ を取得し，$\mathsf{Report}_{\mathsf{app}}$ とともに P_i に送る．
6: P_i: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証し，成功すれば ID_S のシェア S_i について $ct_i \coloneqq \mathrm{Enc}(pk_{\mathsf{skr}}, S_i)$ を返す．
7: **SKR-TEE**: 応答した P_i の添字集合を I とし，各 ct_i を $S_i \leftarrow \mathrm{Dec}(sk_{\mathsf{skr}}, ct_i)$ で復号する．
8: **SKR-TEE**: $S \leftarrow \mathsf{Reconst}(\{(i, S_i)\}_{i\in I})$ を得る．失敗なら $\perp$ を返す．
9: **SKR-TEE**: $ct \leftarrow \mathrm{Enc}(pk_{\mathsf{app}}, S)$ を App-TEE に返す．

$\{S'_j\}_{j\in[n']}$ **or** $\perp \leftarrow$ **Reshare**$(\mathsf{ID}_S, dc', pol';\ dc, pol)$

1: **利用者**: ID_S，$dc' \coloneqq (k', n', (P'_1, \ldots, P'_{n'}))$，$pol'$，$r$ を SKR-TEE に送る．
2: **SKR-TEE**: ID_S に登録された (dc, pol) を参照し，ID_S，dc，pol，dc'，pol'，r を含む $\mathsf{Report}_{\mathsf{skr}}$ を取得して利用者に返す．
3: **利用者**: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証する．
4: **SKR-TEE**: 鍵ペア $(pk_{\mathsf{skr}}, sk_{\mathsf{skr}})$ を生成し，dc の各 P_i に ID_S のシェアを要求する．
5: P_i: r_i を返す．
6: **SKR-TEE**: ID_S, dc, pol, dc', pol', pk_{skr}, r_i を含む $\mathsf{Report}_{\mathsf{skr}}$ を取得し，P_i に送る．
7: P_i: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証し，成功すれば $ct_i \coloneqq \mathrm{Enc}(pk_{\mathsf{skr}}, S_i)$ を返す．
8: **SKR-TEE**: 応答した P_i の添字集合を I とし，各 ct_i を $S_i \leftarrow \mathrm{Dec}(sk_{\mathsf{skr}}, ct_i)$ で復号する．
9: **SKR-TEE**: $S \leftarrow \mathsf{Reconst}(\{(i, S_i)\}_{i\in I})$ を得る．失敗なら $\perp$ を返す．
10: **SKR-TEE**: ID_S に対して dc' と pol' を自身に適用し，$(S'_1, \ldots, S'_{n'}) \leftarrow \mathsf{Share}(S, n')$ を行い，各 P'_j にシェアの受け取りを要求する．
11: P'_j: r_j を返す．
12: **SKR-TEE**: ID_S，dc'，pol'，r_j を含む $\mathsf{Report}_{\mathsf{skr}}$ を取得し，S'_j とともに P'_j に送る．
13: P'_j: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証し，S'_j を ID_S の新しいシェアとして保存する．
14: **SKR-TEE**: dc の各 P_i からナンス r_i を受け取り，ID_S，dc'，pol'，r_i を含む $\mathsf{Report}_{\mathsf{skr}}$ を添えて古いシェアの削除を指示する．
15: P_i: $\mathsf{Report}_{\mathsf{skr}}$ でこれらを検証し，S_i を削除する．

$\phi \leftarrow$ **Restore**(ID_S, dc, pol)

1: **利用者**: ID_S，dc，pol を SKR-TEE に送る．
2: **SKR-TEE**: ID_S に対して dc と pol を自身に適用する．

利用者のシェアで KeyRec は成功する．ただし，その後も利用者が介在しない運用に戻すために，再起動後に Restore に続けて利用者が Reshare を実行し，秘密情報を再分割して SKR-TEE に新しいシェアを持たせる．

3.4.3 ユースケース 2：マルチパーティでの利用

複数の組織が 1 つの App-TEE を共同で利用する場合は，参加する各組織をシェア保有者とすることで，各組織が SKR-TEE を検証したうえでシェアを渡す，特定の主体を信頼しない運用を構成できる．具体的には，所定の数の組織が合意したときに限り復号（しきい値暗号 [15]）や署名（しきい値署名 [16]）を行いたいユースケースで利用できる．

一例として，組織 A，B，C，D，E をシェア保有者とする $(3, 5)$ の場合を考える．利用者は 5 組織のいずれでも，組織外の第三者でもよい．初回は，利用者が SKR-TEE をデプロイして Setup で dc と pol を登録し，秘密情報を分割・配布する．以後の KeyRec では，5 組織のうち 3 組織以上がシェアを返せば秘密情報を復元できる．各組織は，シェアを渡す前に SKR-TEE のレポートで dc と pol を検証し，さらに App-TEE のレポートを自身の提供条件と照らし合わせてシェアを返すか否かを判断する．いずれか 2 組織が結託しても，または 2 組織からシェアが漏洩しても秘密情報は復元されず，2 組織が紛失や離脱をしても提供を継続できる．SKR-TEE は秘密情報に関する状態を一切持たないため，故障や停止からの再起動でも，負荷分散のための複製でも，同じ (dc, pol) で Restore を実行するだけでよい．

3.5 安全性

3.1 節の攻撃者が，k 個未満のシェア保有者と，App-TEE および SKR-TEE のホストを制御できるとき，$k \geq 2$ のもとで，攻撃者の目的である **(1) 秘密情報を取得すること**，**(2) 提供条件を満たさない環境に秘密情報を提供させること**のいずれも達成できないことを示す．厳密な証明や，形

式検証は今後の課題とする．

攻撃者によって制御されていないシェア保有者を正当な保有者と呼ぶ．Setup で登録する条件と Reshare で変更した後の条件は，利用者と保有者がプロトコルの外で合意していると仮定する．また，変更前と変更後における攻撃者が制御する保有者は，それぞれのしきい値未満とする．これは保有者の集合を変更できる秘密分散法（Proactive Secret Sharing） [17] と同じ仮定である．また，Setup で S を指定する利用者は正当なパーティであるとし，それ以外の利用者は攻撃者の制御下にあるとする．以下では，SKR-TEE が正規のコードと合意した条件で動作する場合と，攻撃者の制御下にある場合，すなわち SKR-TEE のコードが改ざんされている場合や，攻撃者が各アルゴリズムを任意に実行できる場合に分けて議論する．

3.5.1 秘密情報の機密性

攻撃者が，S そのものを直接取得することも，k 個以上のシェアを集めて復元することもできないことを示す．しきい値未満のシェアからは，完全秘匿性（2.2 節）により S の情報は一切得られない．また，S を利用者が指定する場合は，3.3.3 節の手段により検証済みの SKR-TEE にのみ送られる．

SKR-TEE が正規の場合. S とシェアが平文で存在するのは TEE の内部だけであり，攻撃者はホストから読み出せない．シェアは SKR-TEE と保有者の間の通信路を流れるが，配布では保有者を認証した TLS で，返却では保有者が $\mathsf{Report}_{\mathsf{skr}}$ で検証した pk_{skr} による暗号化で保護される．S 本体は，SKR-TEE と App-TEE の間の通信路を流れるが，SKR-TEE が $\mathsf{Report}_{\mathsf{app}}$ で検証した pk_{app} による暗号化で保護される．App-TEE のレポートを偽造，再送，中継して S を得る攻撃は，3.5.2 節で扱う．

SKR-TEE が攻撃者の制御下にある場合. 攻撃者は，改ざんした SKR-TEE で S を直接受け取るか，正当な保有者から少なくとも 1 つのシェアを得る必要がある．Setup において，改ざんされた SKR-TEE が攻撃者が知る S のシェアを配布しようとしても，攻撃者が合意と異なる条件で登録しようとしても，正当な保有者は測定値と条件を検証するため，そのシェアを保存しない．KeyRec において，改ざんされた SKR-TEE では $\mathsf{Report}_{\mathsf{skr}}$ の測定値が，Restore で合意と異なる条件を設定された SKR-TEE では $\mathsf{Report}_{\mathsf{skr}}$ のユーザデータが，保有者の期待値と異なるため，保有者はシェアを返さない．Reshare において，未合意の条件への変更は正当な保有者が拒否する．また，変更後のシェアは新たな乱数で生成されるため，変更前に漏洩したシェアと組み合わせても S の復元には使えない．

3.5.2 不正な提供の防止

pol を満たす正規の App-TEE のレポートを $\mathsf{Report}_{\mathsf{app}}$，*pol* を満たさない攻撃者の環境 App-TEE* のレポートを $\mathsf{Report}^{*}_{\mathsf{app}}$ とする．攻撃者が，App-TEE* に S を提供させることができないことを示す．

SKR-TEE が正規の場合. 攻撃者は，App-TEE* を正規の App-TEE に見せかけて S を提供させようと試みる．しかし，$\mathsf{Report}_{\mathsf{app}}$ を偽造すれば署名の検証に失敗し，過去の $\mathsf{Report}_{\mathsf{app}}$ を再送すれば SKR-TEE の発行したナンスと一致しない．正規の App-TEE の新しい $\mathsf{Report}_{\mathsf{app}}$ を中継しても，S はその pk_{app} で暗号化されるため，復号できるのは正規の App-TEE だけであり，攻撃者も App-TEE* も S を得られない．

SKR-TEE が攻撃者の制御下にある場合. SKR-TEE は $\mathsf{Report}^{*}_{\mathsf{app}}$ を受理しうるが，3.5.1 節で示したとおり，正当な保有者が各アルゴリズムで $\mathsf{Report}_{\mathsf{skr}}$ を検証するため，S 本体もしきい値以上のシェアも得られない．

3.5.3 安全性における課題

提案方式の安全性には，次の 2 つの限界があり，今後の課題とする．

秘密分散法の課題. 提案方式の安全性は，攻撃者が制御する保有者がしきい値未満であることに依存する．攻撃者が k 個以上の保有者を支配または結託させれば，SKR-TEE を介さずに S を復元できる．これは分散条件の妥当性や，条件の変更をどのように合意するかという課題であり，本稿の範囲外である．

シェアの正当性. 攻撃者の制御する保有者が誤ったシェアを返した場合，SKR-TEE は復元に失敗するか，誤った S' を復元して提供する．これは機密性を損なわないが，可用性と復元結果の正しさを損なう．対策として，Setup で各保有者に $\mathrm{H}(S)$ をシェアとともに保管させ，KeyRec で応答の過半数が示す $\mathrm{H}(S)$ と S' を照合する方法が考えられるが，応答する保有者の過半数が正当であることを要する．SKR-TEE がシェアに署名する方法や Message Authentication Code（MAC）は，秘密鍵を SKR-TEE 内で保持する必要があり，SKR-TEE のステートレス化を妨げる．

4. 実装

本章では，提案方式の実装について述べる．

4.1 実装環境

SKR-TEE，シェア保有者，利用者用のコマンドラインツールの 3 つを Go で実装した．TEE には AMD SEV-SNP の CVM を用い，SKR-TEE と App-TEE はそれぞれ別の CVM 上で動作する．シェア保有者と利用者ツールは TEE の外で動作する通常のプロセスである．App-TEE は Rust で書かれた Confidential Containers の既存のクライアント [14] を用いた．実装と評価には，AMD EPYC 7543P（32 コア），Ubuntu 22.04.5（Linux 6.16.0, AMDSEV 版），QEMU 10.0.0（KVM），SNP ファームウェア API 1.58 のホストを用い，CVM は 4 vCPU，2 GB のメモリ

ⓒ2026 Information Processing Society of Japan

を割り当てた．CVM のゲストは Ubuntu 22.04.5（Linux 6.16.0）であり，カーネル，initrd，起動オプションを測定値に含めて起動した．ただし本試作では，SKR-TEE，AA，CDH をゲストの仮想ディスク上のファイルとして置いたため，これらは起動時の測定値に含まれない．実運用では，SNPGuard [18] のように，ゲストで動作するコード全体を測定値の対象にする必要がある．

4.2 App-TEE と SKR-TEE

SKR-TEE と App-TEE との通信は，Trustee の KBS Attestation Protocol（Version 0.4.0） [19] に準拠させた．これにより App-TEE 側では，Trustee 向けの既存クライアントである Attestation Agent（AA）と Confidential Data Hub（CDH）[14] を宛先の変更のみでそのまま利用できる．KBS Attestation Protocol のトークン（2.3 節）は，JWT の形式のまま，その識別子（乱数）を，有効期限付きでメモリ上の $\mathsf{Report}_{\mathsf{app}}$ と pk_{app} を参照する Handle [20] として扱った．

レポートの検証と提供条件との照合には go-sev-guest [21] を用い，提供条件 pol は測定値や TCB のバージョンなどの参照値を記述した JSON として表現した．秘密分散には Shamir の方式 [9] を用い，実装には HashiCorp Vault の shamir パッケージ [22] を用いた．暗号化には，App-TEE 向けには JWE（ECDH-ES，P-256），保有者向けには Go 標準の HPKE [23]（X25519）を用いた．SKR-TEE 自身のレポートには，3.3.1 節の値を，用途のラベルと各値の長さを前置して連結し，その SHA-512 ハッシュ値（64 バイト）を SEV-SNP の REPORT_DATA フィールドに格納した．

4.3 シェア保有者と利用者ツール

シェア保有者は，ナンスの発行，シェアの受け取り，レポートの検証とシェアの返却，シェアの削除の 4 つの API を HTTPS で提供するサービスとして実装した．期待する SKR-TEE の測定値と合意済みの (dc, pol) を設定として持ち，レポートの検証には SKR-TEE と同じ go-sev-guest を用いる．シェアの保存先はローカルファイルと Azure Key Vault の 2 種類を実装しており，保存先を置き換えれば他の KMS や HSM も保有者として利用できる．

5. 評価

本章では，提案方式の実装を用いて，各アルゴリズムの処理時間を分散条件を変えて測定して既存の SKR と比較し，あわせてコード規模を評価する．

5.1 実験方法

SKR-TEE，App-TEE，シェア保有者はすべて同一ホスト上で動作させ，保有者はシェアをローカルファイルに保存するホスト上のプロセス（以下，ホスト内の保有者）とした（UC1 の KMS にあたる保有者のみ Azure Key Vault（AKV）に保存した）．往復遅延（ping）は，App-TEE から Trustee の KBS と SKR-TEE でそれぞれ 0.11 ms と 0.21 ms，SKR-TEE から保有者で 0.1 ms であった．秘密情報には，利用者が生成した 32 バイトの乱数を用い，Setup で SKR-TEE のレポートに含まれる公開鍵により暗号化して渡した．測定値はいずれも 10 回の試行の平均である．

比較対象. 既存の SKR である Trustee [5] と，提供判断と秘密情報の保管を 1 つの TEE のみで行う構成（以下，baseline）の 2 つと比較した．Trustee は docker compose の構成で同じホスト上に配置し，App-TEE との通信を提案方式と同じく HTTPS とした．Trustee の値のみ，KBS の応答で生じる遅延 ACK の待ち（約 40 ms）を避けるため，App-TEE で TCP の quickack を有効にして測定した．baseline は Contrast [8] の Coordinator を想定しており，SKR-TEE のコードから秘密分散の部分を省いて実現した．

分散条件. SKR-TEE，クラウドの KMS，利用者を保有者とするユースケース 1（3.4.2 節）の $(2,3)$（以下，UC1）と，SKR-TEE を含まずしきい値を過半数として分割数を増やした $(2,3)$，$(3,5)$，$(6,10)$ を用いた．$(3,5)$ はユースケース 2 にあたる．

5.2 鍵提供の処理時間

App-TEE 内で，CDH のクライアントツールが秘密情報を要求してから受け取るまでの時間とその内訳を図 4 に示す．ただし，Trustee は内部を計測していないため，合計のみを示す．Trustee は 72.9 ms，baseline は 66.0 ms であったのに対し，提案方式は $(2,3)$，$(3,5)$，$(6,10)$ でそれぞれ 87.5，98.4，130.9 ms，UC1 で 137.0 ms であった．App-TEE 内の処理はいずれの構成でも約 54 ms，App-TEE のアテステーション（AA のレポート生成の待ちを含む）は約 12.5 ms であり，baseline と変わらない．提案方式の増分は保有者との通信によるもので，ホスト内の保有者では 1 台あたり約 10 ms であり，その過半は SKR-TEE 自身のレポートの生成（約 6 ms）である．UC1 ではこれに AKV からの読み出し（平均 58.7 ms）が加わり，ばらつきも大きい．HPKE の復号（0.1 ms）と Shamir の復元（$(6,10)$ でも 0.2 ms）は無視できる．参考として，同じ App-TEE 内で LUKS2 のボリュームを開く時間（cryptsetup 2.4.3 の既定の argon2id 設定）は 2057 ms であり，提案方式の鍵提供の時間はその 4〜7% にあたる．

5.3 Setup，Restore，Reshare の処理時間

利用者ツールが各操作を要求してから完了の応答を受け取るまでの時間を表 1 に示す．Reshare は分散条件を変えずに実行した．Restore は，SKR-TEE のプロセスを強制終了して再起動した後に実行した（CVM の起動時間は含まない）．Restore は保有者と通信しないため，分散条件によ

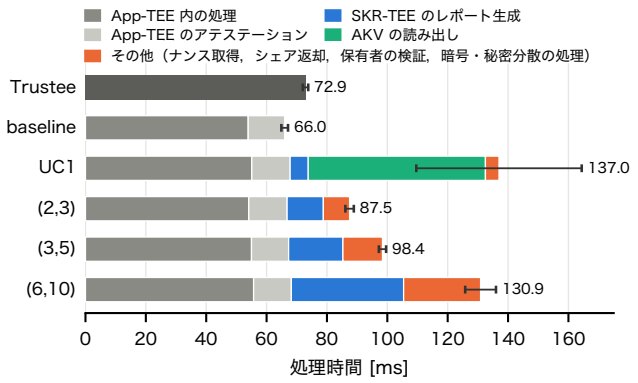

図 4 鍵提供の処理時間の合計とその内訳．

表 1 Setup，Restore，Reshare の処理時間 [ms]．

分散条件	Setup	Restore	Reshare
UC1	245.9	2.2	217.4
$(2,3)$	41.6	2.2	60.3
$(3,5)$	59.8	2.3	93.6
$(6,10)$	107.9	2.2	173.6

らず約 2 ms で完了した．Setup と Reshare は，ホスト内の保有者の構成では，保有者の数とともに増え，UC1 では AKV との通信が支配的であった．

5.4 コード規模

保有者は SKR-TEE の測定値を検証してシェアを返すため，その測定値が表すコード，すなわち信頼の基盤（Trusted Computing Base，TCB）は小さく，第三者が確認できることが望ましい．SKR のコード行数を表 2 に示す．Trustee では，TEE の外で動作する KBS，AS，RVPS とその運営者を信頼する必要があり，その規模は外部ライブラリを含めて約 225 万行である[*2]．これに対し提案方式で信頼する必要があるのは TEE 内の SKR-TEE の約 5.6 万行であり，baseline との本体の差は秘密分散と保有者との通信等の約 1,100 行である．

シェア保有者は 553 行，利用者ツールは 321 行である．App-TEE 側の AA と CDH（本体約 2.3 万行，外部ライブラリ約 158 万行）は Trustee の場合と共通であり，変更せずに用いた．

5.5 考察

保有者 1 台と SKR-TEE の通信では，TCP と TLS 1.3 の確立，ナンスの取得，シェアの返却の 4 回の往復が生じる．そのため，往復遅延が平均 d の保有者が分散する環境における通信の待ち時間は，保有者に 1 台ずつ要求する場合（本実装）で約 $4dk$，全保有者に同時に要求する場合で k 番目に速い保有者の往復遅延の約 4 倍となる．ただし同時に要求しても，レポートの生成はゲストのドライバが 1 件ずつ処理するため，約 6 ms の生成が k 回続く．全保有者のナンスを 1 枚のレポートにまとめれば，この部分も k によらず一定にできるが，その安全性の検討は今後の課題である．

[*2] Rust の外部ライブラリは x86_64 の Linux 向けのファイルだけを ライブラリ（crate）単位で数えたため，やや多めの値である．

表 2 SKR のコード行数比較（空行・コメント・テストと標準ライブラリを除く）．

構成	実行場所	本体	外部ライブラリ
Trustee	TEE 外	18,844	2,230,317
baseline	TEE 内	1,629	52,873
SKR-TEE	TEE 内	2,734	53,031

6. 関連研究

6.1 TEE

TEE は，隔離の単位によってプロセス型と VM 型に大別される．プロセス型には，Intel SGX [29]，Arm TrustZone [30]，RISC-V の Keystone [31] がある．VM 型には AMD SEV(-ES/SNP) [2]，Intel TDX [3]，Arm CCA [32] がある．プロセス型は TCB が小さい一方で，既存のアプリケーションを専用の SDK [33] で移植するか，ライブラリ OS [34,35] を用いて TCB を増やす必要がある．VM 型は VM 全体を隔離するため既存のアプリケーションを変更せずに保護できる一方で，ゲスト OS を含むため TCB が大きい．さらに，NVIDIA H100 の GPU [36] や DPU [37] など，アクセラレータにも TEE の機能が広がっている．これらはいずれもリモートアテステーションを備えており，提案方式は SKR-TEE と App-TEE のレポートを検証できる TEE であれば適用できる．

6.2 検証サービス

TEE のレポートを検証し，その結果をトークンとして発行する検証サービスは，主要なクラウドがそれぞれ提供している．Azure は Azure Attestation [4] を，GCP は Confidential Space の検証サービス [6] を用い，AWS KMS は Nitro Enclaves のアテステーション文書を自ら検証する [7] が，いずれも主に自社のクラウド上の TEE を対象とする．クラウドに依存しない検証器には，サービスとして動作する Trustee の AS [5]，Intel Trust Authority [38]，Veraison [39] と，ライブラリやツールとして用いる go-sev-guest [21]，snpguest [40] がある．提案方式では，各保有者が自ら選んだ検証器で SKR-TEE のレポートを検証するため，全員が単一の検証サービスとその運営者を信頼する必要はない．

6.3 Secure Key Release

アテステーションで TEE を検証してから秘密情報を供給する手順は，Intel SGX の登場時から TEE の基本的な機能として示されてきた [41]．Haven [34] は，利用者が

ⓒ2026 Information Processing Society of Japan

表 3 既存方式との比較．課題 1，2 は 2.4 節の課題，CPU 非依存は秘密情報の復元が同一の CPU に縛られないことを表す．

方式	課題 1	課題 2	CPU 非依存
Trustee [5]	–	–	✓
クラウド SKR [4, 6, 7]	–	–	✓
SGX Sealing [24, 25]	✓	✓	–
eHSM-KMS [26]，KMSGX [27]	–	✓	–
MultiSGX-KMS [28]	–	✓	–
HashiCorp Vault [22]	–	–	✓
Keylime [12]	✓	–	✓
Contrast [8]	–	✓	✓
提案方式	✓	✓	✓

enclave の測定値を確認したうえで，レポートに含まれる公開鍵でディスクの暗号化鍵を包んで渡す手順を示した．現在は，Trustee [5]，Azure SKR [4]，AWS KMS [7]，GCP の Cloud KMS [6] など，検証サービスと KMS が提供判断を担う SKR が実用化されている．これらは提供先の TEE を検証する仕組みであり，提供判断を行う側と秘密情報を保持する側の信頼を前提とする．

一方で，TEE を用いずに秘密分散で鍵を管理する HashiCorp Vault の unseal [22] は，複数の運用者にシェアを持たせるが，復元を Vault のサーバ上で行う．

KMS 自体を TEE で保護する研究として，eHSM-KMS [26] や KMSGX [27] は KMS を SGX の enclave 上で動作させ，MultiSGX-KMS [28] は複数の enclave 上で鍵のシェアを持たせる．しかし，いずれも鍵やシェアの永続化に SGX の Sealing [24] を用いるため，同一の CPU でしか復元できず，Sealing を TDX の CVM から使えるようにした Loose SEAL [25] も同じ制約を受ける．本稿に最も近いのは，提供判断を TEE に置く Contrast [8] と，鍵を 2 つに分けて持たせる Keylime [12] であり，それぞれ課題 2 と課題 1 に対処する（2.5 節）．既存方式と提案方式の比較を表 3 に示す．既存方式にはいずれも，特定の主体への信頼か同一の CPU への依存が残るが，提案方式は分割・復元と提供判断を検証可能な SKR-TEE で行うことで，その両方を不要にする．

7. おわりに

本稿では，SKR の提供判断をアプリケーションとは別の TEE で行い，秘密情報を秘密分散法で複数のシェア保有者へ預けることで，単一主体を信頼せずに検証可能な SKR を実現する方式を提案した．AMD SEV-SNP 上に実装し，既存の SKR のクライアントを無改変のまま利用でき，実用上十分な性能を持つことを確認した．今後は，SKR-TEE を含むゲスト全体を測定値の対象とする実装の改善と，プロトコルの形式的な安全性証明に取り組む．

参考文献

[1] Microsoft: Storm-0558 によるキーの取得に関する主な技術調査の結果について，`https://msrc.microsoft.com/blog/2023/09/results-of-major-technical-investigations-for-storm-0558-key-acquisition-ja/`（参照 2026-10-26）．

[2] AMD: AMD Secure Encrypted Virtualization (SEV), `https://www.amd.com/ja/developer/sev.html` (Accessed on 2026-10-26).

[3] Cheng, P. et al.: Intel TDX Demystified: A Top-Down Approach, *ACM Comput. Surv.*, Vol. 56, No. 9, pp. 238:1–238:33 (2024).

[4] Microsoft: Secure Key Release feature with AKV and Azure Confidential Computing (ACC), `https://learn.microsoft.com/en-us/azure/confidential-computing/concept-skr-attestation` (Accessed on 2026-10-26).

[5] Confidential Containers: Trustee, `https://github.com/confidential-containers/trustee` (Accessed on 2026-10-26).

[6] Google: Confidential Space security overview, `https://docs.cloud.google.com/docs/security/confidential-space` (Accessed on 2026-10-26).

[7] AWS: Cryptographic attestation support in AWS KMS, `https://docs.aws.amazon.com/kms/latest/developerguide/cryptographic-attestation.html` (Accessed on 2026-10-26).

[8] Edgeless Systems: Contrast, `https://docs.edgeless.systems/contrast/` (Accessed on 2026-10-26).

[9] Shamir, A.: How to Share a Secret, *Commun. ACM*, Vol. 22, No. 11, pp. 612–613 (1979).

[10] Blakley, G. R.: Safeguarding cryptographic keys, *1979 International Workshop on Managing Requirements Knowledge, MARK 1979, New York, NY, USA, June 4-7, 1979*, IEEE, pp. 313–318 (1979).

[11] Birkholz, H. et al.: Remote ATtestation procedureS (RATS) Architecture, RFC 9334, `https://www.rfc-editor.org/info/rfc9334/` (Accessed on 2026-10-26) (2023).

[12] Schear, N. et al.: Bootstrapping and maintaining trust in the cloud, *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016* (Schwab, S., Robertson, W. K. and Balzarotti, D., eds.), ACM, pp. 65–77 (2016).

[13] Knauth, T. et al.: Integrating Remote Attestation with Transport Layer Security, arXiv:1801.05863 (2018).

[14] Confidential Containers: Trustee: guest-components, `https://github.com/confidential-containers/guest-components` (Accessed on 2026-10-26).

[15] Desmedt, Y. and Frankel, Y.: Threshold Cryptosystems, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings* (Brassard, G., ed.), Lecture Notes in Computer Science, Vol. 435, Springer, pp. 307–315 (1989).

[16] Desmedt, Y. and Frankel, Y.: Shared Generation of Authenticators and Signatures, *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings* (Feigenbaum, J., ed.), Lecture Notes in Computer Science, Vol. 576, Springer, pp. 457–469 (1991).

[17] Maram, S. K. D. et al.: CHURP: Dynamic-Committee Proactive Secret Sharing, *Proceedings of the 2019 ACM*

ⓒ2026 Information Processing Society of Japan

SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019 (Cavallaro, L., Kinder, J., Wang, X. and Katz, J., eds.), ACM, pp. 2369–2386 (2019).

[18] Wilke, L. and Scopelliti, G.: SNPGuard: Remote Attestation of SEV-SNP VMs Using Open Source Tools, *IEEE European Symposium on Security and Privacy Workshops, EuroS&PW 2024, Vienna, Austria, July 8-12, 2024*, IEEE, pp. 193–198 (2024).

[19] Confidential Containers: KBS Attestation Protocol, `https://github.com/confidential-containers/trustee/blob/main/kbs/docs/kbs_attestation_protocol.md` (Accessed on 2026-10-28).

[20] Lodderstedt, T., McGloin, M. and Hunt, P.: OAuth 2.0 Threat Model and Security Considerations, RFC 6819, `https://www.rfc-editor.org/info/rfc6819/` (Accessed on 2026-10-26) (2013).

[21] Google: go-sev-guest, `https://github.com/google/go-sev-guest` (Accessed on 2026-10-26).

[22] HashiCorp: HashiCorp Vault: Shamir Secret Sharing Implementation, `https://github.com/hashicorp/vault/tree/main/shamir` (Accessed on 2026-10-26).

[23] Barnes, R. et al.: Hybrid Public Key Encryption, RFC 9180, `https://www.rfc-editor.org/info/rfc9180/` (Accessed on 2026-10-26) (2022).

[24] Intel: *Intel Software Guard Extensions Developer Guide* (2022).

[25] Mahhouk, M.: Loose SEAL: Restoring Sealed Storage Capabilities in Intel TDX Through SGX Sidecar, Confidential Computing Summit 2025, `https://www.confidentialcomputingsummit.com/e/ccs25/page/2025posters` (Accessed on 2026-10-26) (2025).

[26] Intel: eHSM-KMS, `https://github.com/intel/ehsm` (Accessed on 2026-10-26).

[27] Bouamama, J., Hedabou, M. and Erradi, M.: Cloud Key Management using Trusted Execution Environment, *Proceedings of the 18th International Conference on Security and Cryptography, SECRYPT 2021, July 6-8, 2021* (di Vimercati, S. D. C. and Samarati, P., eds.), SCITEPRESS, pp. 560–567 (2021).

[28] Abdulsalam, Y. S. et al.: Decentralized SGX-Based Cloud Key Management, *Network and System Security - 17th International Conference, NSS 2023, Canterbury, UK, August 14-16, 2023, Proceedings* (Li, S., Manulis, M. and Miyaji, A., eds.), Lecture Notes in Computer Science, Vol. 13983, Springer, pp. 327–341 (2023).

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

[30] Ngabonziza, B. et al.: TrustZone Explained: Architectural Features and Use Cases, *2nd IEEE International Conference on Collaboration and Internet Computing, CIC 2016, Pittsburgh, PA, USA, November 1-3, 2016*, IEEE Computer Society, pp. 445–451 (2016).

[31] Lee, D. et al.: Keystone: an open framework for architecting trusted execution environments, *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020* (Bilas, A., Magoutis, K., Markatos, E. P., Kostic, D. and Seltzer, M. I., eds.), ACM, pp. 38:1–38:16 (2020).

[32] Arm: Confidential Compute Architecture, `https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture` (Accessed on 2026-10-26).

[33] Intel: Intel SGX SDK, `https://www.intel.co.jp/content/www/jp/ja/content-details/671054/intel-software-guard-extensions-intel-sgx-sdk-for-windows-os-installation.html` (Accessed on 2026-10-26).

[34] Baumann, A., Peinado, M. and Hunt, G. C.: Shielding Applications from an Untrusted Cloud with Haven, *ACM Trans. Comput. Syst.*, Vol. 33, No. 3, pp. 8:1–8:26 (2015).

[35] Tsai, C., Porter, D. E. and Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX, *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA, USENIX Association, pp. 645–658 (2017).

[36] NVIDIA: Confidential Compute on NVIDIA Hopper H100, `https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/HCC-Whitepaper-v1.0.pdf` (Accessed on 2026-10-26).

[37] Lal, R., Anderson, J. B. and Jackson, A.: Data Processing Unit's Entry into Confidential Computing, *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP 2023, Toronto, Canada, 29 October 2023*, ACM, pp. 56–63 (2023).

[38] Intel: Intel Trust Authority, `https://docs.trustauthority.intel.com/` (Accessed on 2026-10-30).

[39] Confidential Computing Consortium: Veraison: VERificAtIon of atteStatiON, `https://github.com/veraison` (Accessed on 2026-10-30).

[40] Confidential Computing Consortium: VirTEE, `https://virtee.io/` (Accessed on 2026-10-26).

[41] Anati, I. et al.: Innovative Technology for CPU Based Attestation and Sealing, *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP)* (2013).

ⓒ2026 Information Processing Society of Japan