Vol.2021-CSEC-95 No.17
Vol.2021-SPT-45 No.17
Vol.2021-EIP-94 No.17
2021/11/9

# Confidential Program Executionの提案と安全性評価

奥田 哲矢[1]　中林 美郷[1]　荒井 研一[2]　菊池 亮[1]　千田 浩司[1]

**概要**：本研究では，TEE (Trusted Execution Environment) を応用したクラウドサービス群である Confidential Computing について，データおよびプログラムの両者を秘匿したまま利用できる Confidential Program Execution を提案し，その安全性を評価する．前提として，Intel SGX, AMD SEV のようなサーバサイドにおける TEE を使えば，クラウド事業者に対してデータを秘匿しつつ，クラウドサービスを利用することができる．さらにその発展として，Felsen らは，データを有するユーザとプログラムを有するユーザが，互いにそれぞれのデータとプログラムを自身以外（クラウド事業者を含む）には秘匿したまま，プログラムの実行結果を享受できる方式を提案している．しかし Felsen らの方式は，実行毎にデータとプログラムをクラウド事業者にアップロードする必要があり，かつ方式の安全性証明は与えられていなかった．本稿では，Felsen らと同様にデータとプログラムを秘匿しつつ実行結果を得られ，且つ実行毎にデータとプログラムをクラウド事業者にアップロードする必要がない方式を提案し，その方式の安全性を，形式検証ツールである ProVerif を用いて評価した．評価の結果，本研究の提案プロトコルが，各データおよびプログラムの秘匿の要件，および各エンティティの認証の要件を充足することが分かった．また，本研究の提案および評価を通じて分かった，TEE 応用プロトコル設計時に，TEE がユーザとは独立したエンティティとしてふるまう点，および，TEE を含めたマルチパーティの攻撃者モデルを想定すべき点は，今後多くの TEE 応用プロトコルが設計される際に，プロトコル設計者の参考になると期待される．

# Proposition and Security Evaluation of Confidential Program Execution

## 1. はじめに

近年，Confidential Computing が産業界で注目されている．Confidential Computing は，クラウド利用者が，クラウド事業者に対して利用者の機密データなどを秘匿しつつ，クラウドサービスを利用する技術群である．また，その要素技術としては，信頼できる領域を提供する TEE/Enclave や，それらが適切に動作していることを遠隔で検証できる Remote Attestation が用いられる．産業界の注目は高く，Linux Foundation 傘下に Confidential Computing Consortium が発足して，大手 IT ベンダが多く参画している．特に，Microsoft は，積極的にアカデミックにおける発信と，研究開発成果の事業導入を進めている．例えば，事例 [17,19] では，事前に信頼関係の無いエンティティ間で，双方の有するデータを利用した分析結果を，互いのデータを秘匿したまま得る技術とサービスを提案している．ただし，本事例では，各エンティティが，各々のデータとプログラムを互いに持ち寄る利用シーンは想定されていない様である．

本稿では，TEE/Enclave を活用して，事前に信頼関係のないパートナー企業間で，互いの有するプログラムとデータを秘匿したまま，プログラム実行結果のみを享受できるプログラム秘匿実行の仕組みを提案する．既存研究としては Felsen ら [11] が上記を実現する方式を提案している．しかし Felsen らの方式は，実行毎にデータとプログラムをクラウド事業者にアップロードする必要があり，かつ方式の安全性評価は与えられていなかった．

### 1.1 本論文の貢献

本稿では，データおよびプログラムの両者を秘匿したまま実行する，Confidential Program Execution を提案し，その安全性を評価する．提案方式は Felsen らの方式と異なり，実行毎にデータとプログラムをクラウド事業者にアップロードする必要がないという利点を有する．さらに提案方式の安全性を，形式検証ツールである ProVerif を用いて評価した．評価の結果，本研究の提案プロトコルは，

[1] NTT 社会情報研究所
NTT Social Informatics Laboratories
[2] 長崎大学
Nagasaki Uniersity

各データおよびプログラムの秘匿の要件，および，各エンティティの認証の要件を充足することが分かった．また，本研究の提案および評価を通じて，TEE 応用プロトコル設計時に，TEE がユーザとは独立したエンティティとしてふるまうこと，および，TEE を含めたマルチパーティの攻撃者モデルを想定すべきことが分かった．これは今後多くの TEE 応用プロトコルが設計される際に，プロトコル設計者の参考になるのではないかと期待している．

## 2. 前提とする技術

### 2.1 TEE

サーバサイドで利用される TEE は，Enclave（日本語訳で「飛び地」）と呼ばれることも多い領域で，クラウド等のサーバ上に，クラウドを利用するユーザの視点で，信頼された“Trusted”な領域を提供する．本領域は，他のアプリケーションから論理的に隔離されており，機密なプログラムやデータを，同じサーバ上の他のアプリケーションから隔離して実行できる．

本稿では，サーバサイドで利用される TEE を，下記の 3 点の特徴を有するメモリ領域と定義する．

(1) TEE/Enclave 内のデータが外部から閲覧できないこと，TEE/Enclave 内のアプリが外部から改ざんできないこと．
(2) 耐タンパ性を有する秘密鍵格納モジュール (Root of Trust) が TEE/Enclave と共に存在すること．
(3) TEE/Enclave が動作するハードウェア，および，TEE/Enclave 上で動作するソフトウェアを，外部から同定するための属性情報を，TEE/Enclave 利用者が遠隔から検証できること (Remote Attestation). それらの検証情報に基づいて，TEE/Enclave 利用者が TEE/Enclave とセキュアチャネルを確立できること (Attested TLS).

サーバサイドで利用される TEE の具体例としては，Intel SGX, AMD SEV, Intel TDX, ARM CCA 等が存在する．本稿では，Intel SGX を応用したプロトコル設計を行ったため，Intel SGX について次節で概説する．

### 2.2 Intel SGX

Intel SGX (Software Guard Extensions) は，サーバサイドで利用される TEE の一形態である．近年は，ハイパースケールサーバ (Xeon Scalable Processor) 向けにシフトしており，最大 1TB の Enclave を生成できる [15]．Intel は，並行して，Intel TDX (Trust Domain Extensions) という AMD SEV に相当するメモリ暗号化機能に特化した技術も提供しており，今後の方向性を注視する必要がある．本研究では，MRSIGNER，MRENCLAVE というプロトコル設計に有用なレジスタを備える SGX をベースに，プロトコル設計および安全性評価を行った結果を示す．Intel SGX に関連する用語説明を以下に記す．

- TA (Trusted Application)：Enclave 内で動作させるアプリケーション．
- HA (Host Application)：Enclave 外で動作するアプリケーションで，TA をラップする役割を有する．
- Enclave：他プロセスと隔離されたセキュアな領域．Enclave 内では，TA のみが実行され，HA は実行されない．
- MRSIGNER：Intel SGX の内部レジスタの一種で，Enclave で動作する TA へのコードサイン署名に対応する公開鍵の値を格納する．
- MRENCLAVE：Intel SGX の内部レジスタの一種で，Enclave で動作する TA のコードハッシュ値に対応する値を格納する．
- Attestation: Remote attestation．2.4 節を参照．
- Quoting: Attestation Key による署名権限を有する Intel の特権的な Enclave に対して，署名依頼を行う処理．

なお，Intel SGX と連動する Intel MEE (Memory Encryption Engine) により，Intel SGX ではメモリ暗号化が実施される．Intel TDX では，Intel (MK)TME ((Multi-Key) Total Memory Encryption) など，別の用語で呼ばれる．

### 2.3 Confidential Computing

Confidential Computing は，ハードウェアおよびそれらのベンダに対する Confidence（信頼）を礎に，ハードウェアおよびソフトウェアの Integrity（一貫性）が検証されたクラウド環境を，クラウド事業者および他利用者に対する Confidentiality（秘匿性）を保ちながら，利用することができるクラウドサービスである．

Confidentiality（秘匿性）を守る前提となる脅威/攻撃者として，クラウド上には悪い管理者が居るかも知れない，クラウド自体が乗っ取られているかも知れない状況を想定する．この場合，従来から利用される通信の暗号化 (SSL/TLS) やストレージの暗号化による対策では十分ではない．なぜなら，メモリ上では秘匿したいデータが平文で処理されており，上記想定する脅威/攻撃者はメモリへの読出/書込/実行権限を有するためである．

そこで，Confidential Computing では，TEE/Enclave が有するメモリ隔離あるいはメモリ暗号化の仕組みを利用する．さらに，CPU の階層的な権限制御機構（リングプロテクション）を使って実行権限の隔離を行う．これにより，CPU が特権的に確保した Enclave と呼ぶメモリ領域は，CPU ベンダのみが制御可能であり，クラウド事業者からは制御不可となる．これにより，上記で想定したクラウド上の攻撃者に対して，メモリ上のデータの Confidentiality（秘匿性）を守ることができる．Linux Foundation 傘下の Confidential Computing Consortium の定義で言えば [8],

data in transit (通信), at rest (ストレージ) に加えて，in use (メモリ), すなわち，実行中も Confidentiality (秘匿性) を守ることが可能となる．

### 2.4 Remote Attestation

サーバサイドの TEE/Enclave の重要な特徴として，前節までのクラウド上のデータ保護に関する情報を，遠隔のクラウド利用者に対して保証あるいは主張できる点がある．これが，Remote Attestation（遠隔検証）と呼ばれる仕組みである．

具体的な手順としては，まず，TEE/Enclave 機能を提供する CPU に付属している RoT (Root of Trust) と呼ばれるコプロセッサに、CPU ベンダは出荷時に事前共有鍵を OTP (One Time Programmable) に焼き付ける．この事前共有鍵は，CPU ベンダの公開する Attestation Service で同様に管理されている．クラウド利用者が，クラウド上の CPU に対して Remote Attestation 要求を行うと，Report/Evidence などと呼ばれる署名付き属性情報が得られる．この Report/Evidence を，CPU ベンダの公開する Attestation Service に対して送信して検証要求を行うと，Attestation Service は事前共有鍵で署名を検証して，Report/Evidence に含まれる属性情報の真正性を検証する．この検証結果をクラウド利用者に対して送信することで，クラウド上で動作するハードウェアおよびソフトウェアの真正性が，クラウド利用者に対して保証される．より具体的には，TEE/Enclave がクラウド上で適切に動作しており，クラウドに預けたデータの Confidentiality （秘匿性）が適切に守られていることを，クラウド利用者は遠隔から検証できる．本章の TEE/Enclave の定義により，Report/Evidence に含められる属性情報は，TEE/Enclave により安全にメモリ隔離されており改ざんされないことが保証される．Remote Attestation で遠隔検証可能な属性情報として，Intel SGX では，前述の MRSIGNER, MRENCLAVE の値があり，それら属性情報の真正性を，Remote Attestation により遠隔から検証可能となる．

### 2.5 ProVerif

本研究では，提案プロトコルの安全性評価に ProVerif を利用することとした．ProVerif は，暗号利用プロトコルの安全性評価に利用されている，暗号利用プロトコルの形式検証ツールのデファクトスタンダードである [5]．ProVerif では Dolev-Yao モデル [9] と呼ばれる暗号プリミティブを理想化したモデルに対して，様々な安全性要件を検証することができる．

#### 2.5.1 安全性要件

本論文では，秘密情報の秘匿性と通信相手との認証を対象とする安全性要件とする．ProVerif では，安全性要件は以下で説明するクエリと呼ばれる文で表現される．

秘密情報 $m$ の秘匿性（すなわち，攻撃者は情報 $m$ を手に入れることができない）は述語 `attacker` を用いて次のように表現される：

- query `attacker` ($m$).

このクエリに対し，プロトコルの実行中に攻撃者が情報 $m$ を何らかの方法で手に入れることができる場合，false（攻撃発見）が出力され，そうでない場合は true（攻撃発見無し）が出力される．

通信相手の認証は，イベントと呼ばれる概念を用いて行われる．イベントは各パーティのプロセスに設置されるフラグのようなもので，各イベントの対応関係を用いて認証要件を検証することができる．例えば，パーティ A が通信相手であるパーティ B を認証するプロトコルの認証要件を検証する場合（すなわち，攻撃者は A に対して B になりすますことができないことを検証したい場合）を考える．イベント `beginAuth` をパーティ B のプロトコル開始時に，イベント `endAuth` をパーティ A のプロトコル終了時（認証完了時）に記述した上で，クエリは以下のように表現される：

- query i:bitstling;
  inj-event (`endAuth`(i)) ==> inj-event (`beginAuth`(i)).

上記のクエリは，イベント `endAuth` (i) が発生したときに，それに対応するイベント `beginAuth` (i) が必ず発生しているかどうかを検証する．ここで，i は認証用のパラメータ（例えばセッション番号や乱数など）である．また，event の前に inj-を付けることでイベントの対応関係が 1 対 1 であることを検証できる．これにより，再送攻撃の検出が可能となる．対応するイベントが 1 対 1 の関係で発生している場合，true（攻撃発見無し）が出力され，そうでない場合は false（攻撃発見）が出力される．対応するイベントが存在するということは通信相手も同じパラメータを用いて通信を行っていることを意味するため，このクエリが成り立つかどうかで認証要件を検証することができる．

## 3. 問題設定

本研究では，TEE/Enclave を活用して，事前に信頼関係のないパートナー企業間で，互いの有するプログラムとデータを秘匿したまま，プログラム実行結果のみを享受できる，プログラム秘匿実行の仕組みを，TEE/Enclave を利用して提供することを目指す．

本稿の問題設定を下記に記述する．

データ $D$ を有する USERD（データホルダ）と，プログラム $P$ を有する USERP（プログラムオーサ）が存在して，$D$ と $P$ を互いに明かさず $P(D)$ を計算する．USERP または USERD が実行結果である $P(D)$ を得る．$P(D)$ を計算する基盤となるクラウドサービスを提供するのは，PF 事業者である．PF 事業者に対して，機密データであ

Vol.2021-CSEC-95 No.17
Vol.2021-SPT-45 No.17
Vol.2021-EIP-94 No.17
2021/11/9

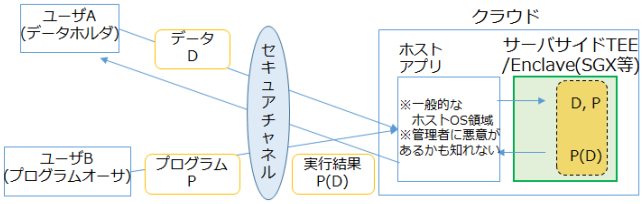


図 1: 提案プロトコルの概要

る $D, P, P(D)$ は一切渡らないこととする．また、本稿では，特に USERD が実行結果である $P(D)$ を得る設定を想定する．この場合，USERP に対して，機密データである $D, P(D)$ は一切渡らないこととする．また，USERD に対して，機密データである $P$ は一切渡らないこととする．

これら問題設定を実現するための補助的なエンティティとして，TEE (Intel SGX)，TA (Enclave)，Verifier (Intel Attestation Service) を用意する．

## 4. 提案プロトコル

前節の問題設定を解決する提案プロトコルを下記に示す．

### 4.1 事前利用

本プロトコルの開始前に，データホルダおよびプログラムオーサは，当該クラウドサービスをデータおよびプログラムを保管するストレージとして利用している想定とする．データホルダは，事前ストレージ利用時に，クラウド事業者に対してデータを秘匿するため，暗号化鍵 $PK_d$ でデータ $D$ を暗号化した $\mathrm{HybEnc}(D, PK_d)$ をクラウドサービスに送信して保管する．プログラムオーサは，事前ストレージ利用時に，クラウド事業者に対してプログラムを秘匿するため，暗号化鍵 $PK_p$ でプログラム $P$ を暗号化した $\mathrm{HybEnc}(P, PK_p)$ を，$HA, TA$ と，さらにそれらのハッシュ値に署名した $\mathrm{Sign}(\mathrm{Hash}(TA||\mathrm{HybEnc}(P, PK_p)), SK_p)$ を一緒にクラウドサービスに送信して保管する．ここで，$HA, TA$ については，プログラムオーサが $P$ をパッケージ出来るように，事前に PF 事業者から SDK/ライブラリ形式で提供されている想定とする．また，本 SDK/ライブラリは OSS で提供されていることとする．

0. PF $\rightarrow$ USERP: $HA$, $TA$
1. $SK_d, PK_d \leftarrow \mathrm{KeyGen}_{\mathrm{USERD}}$
1. $SK_p, PK_p \leftarrow \mathrm{KeyGen}_{\mathrm{USERP}}$
2. USERD $\rightarrow$ PF: $\mathrm{HybEnc}(D)_{PK_d}$
2. USERP $\rightarrow$ PF: $HA$, $TA$, $\mathrm{HybEnc}(P)_{PK_p}$, $SIGN_P$

   $SIGN_P \coloneqq \mathrm{Sign}(\mathrm{Hash}(TA, \mathrm{HybEnc}(P)_{PK_p}))_{SK_p}$

### 4.2 初期設定 (Enclave 起動)

プログラムオーサは，自身の秘密鍵に対応する自己署名証明書 $(PK_p, \mathrm{Sign}(PK_p, SK_p))$ を送信する．これは，これから起動する Enclave の権限を本プログラムオーサの秘密鍵に紐付けるためであり，一般的に回避されることの多い自己署名証明書を用いる点については，Enclave 起動時の秘密鍵の紐付けは TOFU (Trust On First Use) で良く，TTP (Trusted Third Party) による証明を必須としないためである．その後の Enclave 起動は，SGX が実施する．

Enclave 起動後，Enclave は，新たなエンティティとして，TA に記述された処理に従ってふるまう．TA には，Attestation 付き ECDHE 鍵交換の処理が記述されている．Attestation の実体は署名であり，Intel SGX が管理する Attestation Key による署名生成が可能な，Intel の特権的な Enclave (Quoting Enclave) によって，一般 Enclave の署名要求 (Quoting) に対して署名生成して応答する．本 Attestation 生成処理に従って，Enclave はプログラムオーサと Attestation 付き ECDHE 鍵交換を行い，Enclave とプログラムオーサは共通鍵 $K_p$ を得る．同様に，Enclave はデータホルダと Attestation 付き ECDHE 鍵交換を行い，Enclave とデータホルダは共通鍵 $K_d$ を得る．Attestation は，Intel Attestation Sevice に問い合わせることで，含まれる MRSIGNER や MRENCLAVE の値の真正性を検証可能となる．

1. TEE $\leftarrow PK_p, \mathrm{Sign}(\mathrm{Hash}(PK_p))_{SK_p}$
2. TEE $\leftrightarrow$ USERP: SignedDH

   SignedDH $\coloneqq$ TEE $\rightarrow$ USERP: $g^x, \mathrm{Sign}(g^x)_{\mathrm{Attestation}}$

   SignedDH $\coloneqq$ TEE $\leftarrow$ USERP: $g^y, \mathrm{Sign}(g^y)_{SK_p}$
2. TEE $\leftrightarrow$ USERD: SignedDH

   SignedDH $\coloneqq$ TEE $\rightarrow$ USERD: $g^z, \mathrm{Sign}(g^z)_{\mathrm{Attestation}}$

   SignedDH $\coloneqq$ TEE $\leftarrow$ USERD: $g^w, \mathrm{Sign}(g^w)_{SK_d}$
3. $K_p \coloneqq g^{xy}$
3. $K_d \coloneqq g^{zw}$

### 4.3 プログラムおよびデータ復号

プログラムオーサは，Enclave 上でプログラム $P$ を復号するために，$SK_p$ を $K_p$ で暗号化した $\mathrm{Enc}(SK_p, K_p)$ を，メッセージ認証コード (MAC) として $\mathrm{Mac}(\mathrm{Hash}(\mathrm{Enc}(SK_p, K_p)), K_p)$ を付与して，Enclave に送信する．Enclave は，事前ストレージ利用時に保管されていた $\mathrm{HybEnc}(P, PK_p)$ を Enclave 内に取得して，$K_p$ で MAC 検証および復号された $SK_p$ を用いて，$P$ を復号する．同様に，データホルダは，Enclave 上でデータ $D$ を復号するために，$SK_d$ を $K_d$ で暗号化した $\mathrm{Enc}(SK_d, K_d)$ を，メッセージ認証コード (MAC) として $\mathrm{Mac}(\mathrm{Hash}(\mathrm{Enc}(SK_d, K_d)), K_d)$ を付与して，Enclave に送信する．Enclave は，事前ストレージ利用時に保管されていた $\mathrm{HybEnc}(D, PK_d)$ を Enclave 内に取得して，$K_d$ で MAC 検証および復号された $SK_d$ を用いて，$D$ を復号する．

1. TEE ← USERP: $\mathrm{Enc}(SK_p)_{K_p}, MAC_p$
   $MAC_p \coloneqq \mathrm{Mac}(\mathrm{Hash}(\mathrm{Enc}(SK_p)_{K_p}))_{K_p}$
2. $SK_p \coloneqq \mathrm{Dec}(\mathrm{Enc}(SK_p)_{K_p})_{K_p}$
3. $P \coloneqq \mathrm{Dec}(\mathrm{HybEnc}(P)_{PK_p})_{SK_p}$
1. TEE ← USERD: $\mathrm{Enc}(SK_d)_{K_d}, MAC_d$
   $MAC_d \coloneqq \mathrm{Mac}(\mathrm{Hash}(\mathrm{Enc}(SK_d)_{K_d}))_{K_d}$
2. $SK_d \coloneqq \mathrm{Dec}(\mathrm{Enc}(SK_d)_{K_d})_{K_d}$
3. $D \coloneqq \mathrm{Dec}(\mathrm{HybEnc}(D)_{PK_d})_{SK_d}$

### 4.4 プログラム実行結果送信

上記までの手順で，Enclave 上にプログラム $P$ およびデータ $D$ が格納されたので，$P(D)$ を計算可能となる．$P(D)$ は，今回設定ではデータホルダに返すため，データホルダ向けの共通鍵 $K_d$ で暗号化した $\mathrm{Enc}(P(D), K_d)$ をデータホルダに送信する．データホルダは，$K_d$ を使って $P(D)$ を復号する．

1. $P(D) \coloneqq \mathrm{ProgramExecution}(P, D)$
2. TEE → USERD: $\mathrm{Enc}(P(D))_{K_d}$
3. $P(D) \coloneqq \mathrm{Dec}(\mathrm{Enc}(P(D))_{K_d})_{K_d}$

本提案プロトコルのシーケンス図を図 2 に示す．

本提案プロトコルの効果として，主にプログラムオーサの視点で，$P$, $D$ を PF 事業者に対して秘匿しつつ，$P$ をデータホルダに対して秘匿しつつ，$D$, $P(D)$ をプログラムオーサが与り知ることなく，$P(D)$ の実行結果をデータホルダに提供できる．

## 5. 提案プロトコルの成立に必要な工夫点

### 5.1 SDK/ライブラリの OSS 化

プロトコル手順 (事前利用) で，プログラムオーサが $P$ をパッケージング出来るように事前に TA,HA を PF 事業者が SDK/ライブラリ形式で提供することとした．本 SDK/ライブラリは OSS 化されることで，本 SDK/ライブラリが不正にデータやプログラムを取得や送信することが無いか，外部から検証可能となる．これにより，プログラムオーサは，$P$ のみを実装すれば良く、本サービスを利用する開発者のハードルが下がると想定される．

### 5.2 秘密鍵のアップロード処理 (Dan-Twist)

本プロトコルでは，事業者からの要望があったため，プログラムオーサとデータホルダが，元々クラウドサービスをストレージとして利用している状況からプロトコルを開始した．すなわち，$\mathrm{HybEnc}(P, PK_p)$ と $\mathrm{HybEnc}(D, PK_d)$ は，PF 事業者のストレージに事前に保管済みの想定とした．そして，後から復号鍵である $SK_p$, $SK_d$ を投入する構成とした．秘密鍵である $SK_p$, $SK_d$ をクラウド/サーバにアップロードする処理は，一般的なクラウドの利用シーンにおいては特異である．しかし，SGX/Enclave を活用することで，クラウド事業者に対して，送信した秘密鍵 $SK_p$, $SK_d$ の秘匿性が守られているために，本プロトコルの安全性は充足されている．我々は，本工夫点を，発案者の愛称を取って “Dan-Twist” と呼び交わしている．

Dan-Twist の効果について，さらに以下で述べる．Dan-Twist が無い場合を想定すると，秘匿プログラム実行の都度，プログラムとデータを Enclave に送信する必要があり，プログラムやデータのサイズが大きい場合に，アップロード時間などのユーザ待ち時間が大きくなっていた．本提案の Dan-Twist を用いた構成であれば，ユーザは PF 事業者のストレージサービスを既に利用している等，暗号化したプログラムやデータが PF 事業者のストレージに事前に保管済みの想定として，各復号鍵のアップロードのみで秘匿プログラム実行が可能となり，アップロード時間などのユーザ待ち時間を抑えることが出来る．

### 5.3 スクリプトインタプリタ (Dan-Engine)

プロトコル手順（プログラム復号）で，Enclave 上で復号したプログラム $P$ は，実際は Enclave 上でそのまま実行できない．なぜなら，SGX における Enclave は，起動時に TA のコードが固定される機構となっており，後から Enclave に投入された $P$ をプログラムとして解釈・実行できる機構とはなっていないためである．TCB (Trusted Computing Base) を最小限に固定して，そのコードハッシュ値は SGX 内部レジスタ MRENCLAVE に格納され，Remote Attestation で外部から検証可能となるため，本機構の採用は妥当であるが，一方，実行可能なプログラムの自由度が下がる課題に繋がる．本課題への対策として，本提案では，TA に簡易なスクリプトインタプリタを含める構成とした．これにより，Enclave 内で復号した $P$ を，プログラムとして解釈・実行可能となる．我々は，本工夫点を，発案者の愛称を取って “Dan-Engine” と呼び交わしている．

## 6. 安全性評価

本節では，ProVerif（バージョン 2.02）を用いて提案プロトコルの安全性評価を行う．

### 6.1 想定する攻撃者

本論文では, ProVerif による検証において標準的な Dolev-Yao モデルの攻撃者を想定する．Dolev-Yao モデルでは，各パーティの鍵の情報や通信路に流れるプロトコルメッセージは文字列ではなく項として扱われ，公開鍵暗号方式などの暗号プリミティブは「暗号文は復号鍵を持っているときにのみ復号できる（復号鍵を持っていないといかなる平文の情報も得られない)」という理想的なものとして形式

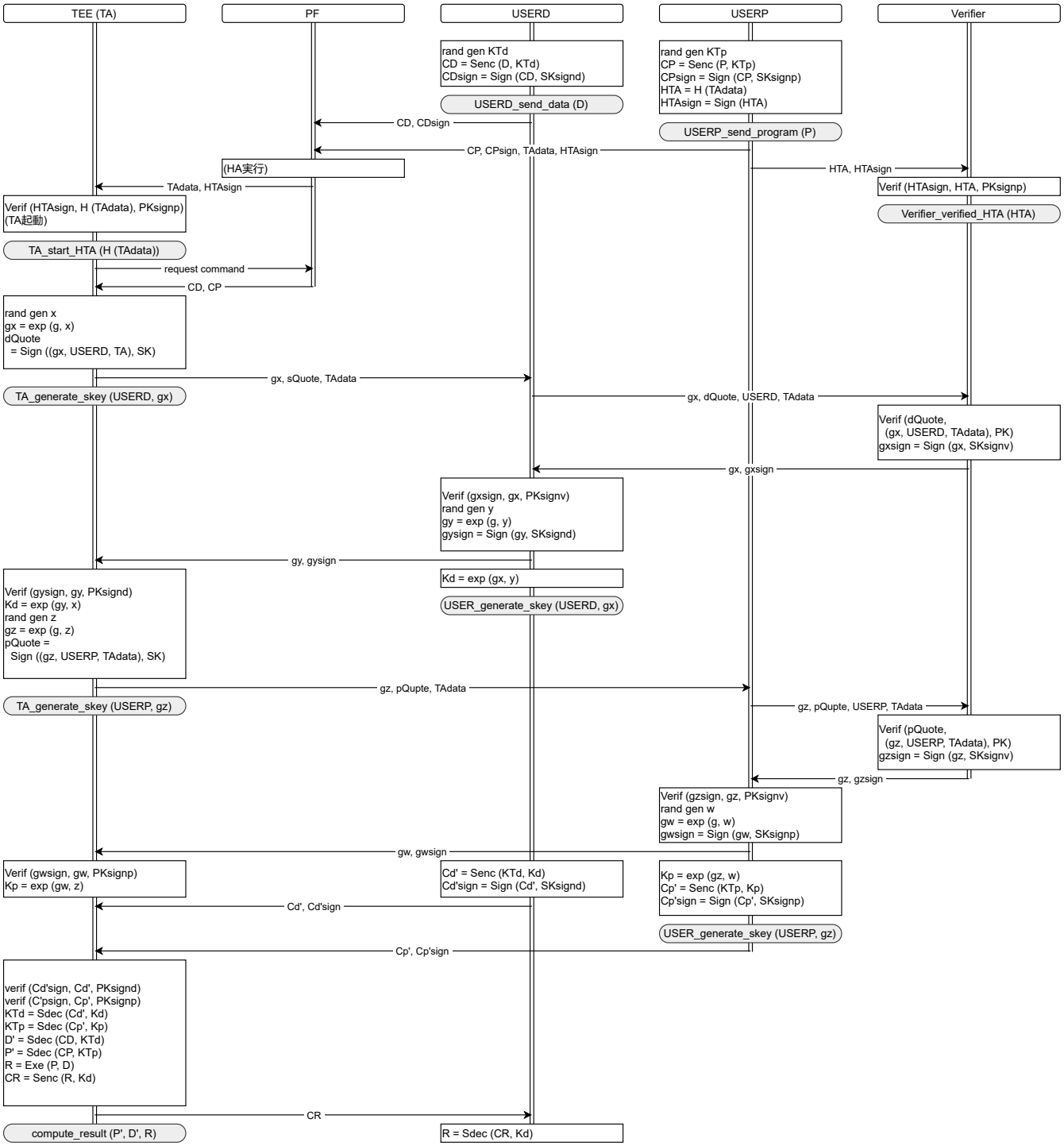


図 2: 提案プロトコルのシーケンス図

化される．その上で攻撃者は，通信路に流れる任意のメッセージに対し，盗聴・改ざん・削除・再送の操作が行える能動的な攻撃者としてモデリングされる．

### 6.2 安全性要件

本論文では, 以下の 11 点の安全性要件を検証対象とする．

- S1: プログラム $P$ の秘匿性
- S2: データ $D$ の秘匿性
- S3: データ $D$ を入力としたプログラム $P$ の実行結果 $P(D)$ の秘匿性
- S4: USERD に対する $P$ の秘匿性
- S5: USERP に対する $D$ の秘匿性
- S6: USERP に対する $P(D)$ の秘匿性
- A1: Verifier による TA の認証
- A2: USERD による TA の認証
- A3: USERP による TA の認証
- A4: USERD による USERP の認証
- A5: USERP による USERD の認証

ここで，S1 から S6 はそれぞれ秘匿したい情報の秘匿性（Secrecy）である．S4 は USERD に対してプログラム $P$ が秘匿されていることを意味する．同様に，S5 と S6 はそれぞれ USERP に対してデータ $D$ と実行結果 $P(D)$ が秘匿されていることを意味する．

A1 から A5 は認証要件（Authentication）であり，「A による B の認証」は「A が通信相手は B と想定してプロトコルを実行しているならば，その通信相手は B である」という要件を意味する．A1 から A3 はそれぞれ攻撃者が TA になりすましできないことを意味し，A4 と A5 はユーザ同士の互いの認証を意味する．

上記の安全性要件を形式化し，ProVerif の検証クエリとして表現すると次のようになる．

- S1, S4: query attacker (P).
- S2, S5: query attacker (D).
- S3, S6: query attacker (P(D)).
- A1: query TA: bitstring; inj-event (TA_start_HTA(TA)) ==> inj-event (Verifier_verified_HTA(TA)).
- A2, A3: query id: user_id, gx: bitstring; inj-event(USER_generate_skey (id, gx)) ==> inj-event(TA_generate_skey (id, gx)).
- A4: query pr: bitstring, da:bitstring, res:bitstring; inj-event (compute_result (pr, da, res)) ==> inj-event (USERP_send_program(pr)).
- A5: query pr: bitstring, da:bitstring, res:bitstring; inj-event (USERP_send_program(pr)) ==> inj-event (compute_result (pr, da, res)).

S4, S5, S6 はクエリとしてはそれぞれ S1，S2，S3 と同様であるが，S4 の場合は USERD の秘密情報（署名鍵，共通鍵，一時秘密鍵）を，S5 と S6 の場合は USERP の秘密情報をそれぞれプロトコルの中で漏洩させる．そうすることで，USERD，USERP の視点からの $P$, $D$, $P(D)$ の秘匿性を検証することが可能になる．A1 のクエリは，直訳すると「任意の *TAdata* に対し，*TAdata* という命令で TA を新しく起動したならば，その *TAdata* はそのイベントよりも前に Verifier によって認証されている」という意味を持ち，すなわち攻撃者が Verifier の認証なしに新しく TA を起動することができないことを検証する．A2，A3 のクエリは「ユーザが鍵 $K_d$ を $gx$ という値を用いて生成した場合，その値は TA によってユーザに送られたものである」という意味を持ち，ユーザによる TA の認証を意味する．A4 は「USERD がデータ *da* とプログラム *pr* を用いて計算された計算結果 *res* を受け取った場合，その値は USERP から PF に送られたプログラム *pr* を用いて計算されている」という意味であり，USERD による USERP（のプログラム）の認証を意味する．A5 はその逆で「USERP が PF にプログラム pr を送った場合，そのプログラムを用いた計算結果を USERD は受け取っている」という意味であり，USERP による USERD の認証を意味する．

各イベントの挿入位置については，図 2 のシーケンス図を参照されたい．シーケンス図内の丸枠で囲まれた項目がイベントである．

### 6.3 評価結果

提案プロトコルの ProVerif による安全性評価結果として，前節の安全性評価の検証項目について，いずれも True（攻撃発見無し）が出力され，安全性が検証された．

この結果から，プログラム $P$, データ $D$, 実行結果 $P(D)$ といった秘密情報が攻撃者に対して秘匿されること，プログラムオーサに対してデータ $D$ と実行結果 $P(D)$ が，データホルダに対してプログラム $P$ がそれぞれ秘匿されたままプロトコルの実行が行われること，攻撃者がそれぞれのエンティティに対してなりすましができないこと（再送攻撃も不可）が保証されていることがわかる．

## 7. 関連研究

データを互いに秘匿して分析する技術として，暗号化したままで一度も復号することなく分析する秘密計算がある．秘密計算にはいくつかの構成方法が知られており，秘密分散 [4,20,23] や garbled circuit [13,22]，（完全）準同型暗号 [10,12,18] などを用いて構成される．これら既存技術は，いずれもデータを秘匿する目的であり，本稿で提案するようなプログラム自体の秘匿実行は想定されていない．計算するプログラムを秘匿する試みとしては汎用回路 [1,21] が挙げられるが，汎用回路を用いるには，まず実行したいプログラムをバイナリ回路で記述する必要があり，さらに回路情報を秘匿するためのオーバーヘッドがかかるため，本稿で達成している，通常のプログラミングと同様に記述でき，且つ平文での実行の数倍程度のオーバーヘッドで実行可能なプログラム秘匿は現状困難である．

また，TEE を含むハードウェアを秘密計算に応用する先行研究が多く存在する．TEE の登場以前で，耐タンパハードウェアを用いて，秘密計算を実現する研究として，[6,16] 等が挙げられる．これらの研究では，CRS (Common Reference String) や trusted public-key registration services（PKI に相当）を，すなわち何らかの trust-based assumption を低減するために，耐タンパデバイスという phisical assumption を採用したが，結果として，secure channel と secure token distribution を仮定しており，trust-based assumption を完全には排除できていなかった．その後，Intel SGX を中心とするサーバサイド TEE が実用化されると，ハードウェアを応用する秘密計算に関する研究が多く発表されるようになった．先行研究 [2,3] は，サーバサイド TEE の特徴的な機能である Remote Attestation を援用して，Attested Computation と呼ぶプロトコルを実現して，

秘密計算に応用している．これら研究の目的は，公開された固定プログラム P に，各 Party が複数の秘匿データを入力することであり，各 Party が秘匿データおよび秘匿プログラムを入力する構成ではないため，我々の研究とは問題設定が異なる．先行研究 [7,11,14] は，TEE による秘密計算の drawback として，サイドチャネル対策を課題として，garbled circuit などを用いた Secure Function Evaluation (SFE) を組み合わせて，サイドチャネル耐性と高速性を両立することを目指した研究である．特に，先行研究 [11] では，公開プログラム P に秘匿データ D を複数入力する SFE に加えて，秘匿プログラム P に秘匿データ D を入力する Private Function Evaluation (PFE) を，汎用回路を使えば PFE は SFE に帰着できることを利用して，TEE を応用したプロトコルとしては初めて PFE を提案している．ただし，安全性評価はなされていなかった．本研究は，データとプログラムを互いに持ち寄って，互いに秘匿したまま，データに対してプログラムを実行する PFE に相当するプロトコルについて，提案と共に ProVerif による安全性評価を行った，初の研究であると言える．

## 8. まとめ

本稿では，データを有するデータホルダと，プログラムを有するプログラムオーサが，互いにデータとプログラムを秘匿したまま，データに対してプログラムを実行する，Confidential Program Execution プロトコルを提案した．本プロトコルは，TEE および Remote Attestation の技術をベースに設計されたものである．さらに，本プロトコルを形式検証ツールである ProVerif を用いて安全性評価を行った．安全性評価の結果，本プロトコルが設定した安全性の検証項目について，いずれも充足していることを確認できた．今後は，TEE として，Intel SGX よりも汎用的な処理を想定する Intel TDX や AMD SEV などのメモリ暗号化ベースの TEE を前提として，本稿同様の秘匿プログラム実行を可能とするプロトコル設計を行う予定である．

### 参考文献

[1] Alhassan, M. Y., Günther, D., Kiss, Á. and Schneider, T.: Efficient and Scalable Universal Circuits, *J. Cryptol.*, Vol. 33, No. 3, pp. 1216–1271 (2020).
[2] Bahmani, R., Barbosa, M., Brasser, F., Portela, B., Sadeghi, A., Scerri, G. and Warinschi, B.: Secure Multiparty Computation from SGX, *Financial Cryptography*, Lecture Notes in Computer Science, Vol. 10322, Springer, pp. 477–497 (2017).
[3] Barbosa, M., Portela, B., Scerri, G. and Warinschi, B.: Foundations of Hardware-Based Attested Computation and Application to SGX, *EuroS&P*, IEEE, pp. 245–260 (2016).
[4] Ben-Or, M., Goldwasser, S. and Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract), *STOC 1988*, pp. 1–10 (1988).
[5] Blanchet, B.: Automatic verification of security protocols in the symbolic model: The verifier proverif, *Foundations of security analysis and design VII*, Springer, pp. 54–87 (2013).
[6] Chandran, N., Goyal, V. and Sahai, A.: New Constructions for UC Secure Computation Using Tamper-Proof Hardware, *EUROCRYPT*, Lecture Notes in Computer Science, Vol. 4965, Springer, pp. 545–562 (2008).
[7] Choi, J. I., Tian, D. J., Hernandez, G., Patton, C., Mood, B., Shrimpton, T., Butler, K. R. B. and Traynor, P.: A Hybrid Approach to Secure Function Evaluation using SGX, *AsiaCCS*, ACM, pp. 100–113 (2019).
[8] Consortium, C. C.: Confidential Computing: Hardware-Based Trusted Execution for Applications and Data (V1.2) (2021).
[9] Dolev, D. and Yao, A.: On the security of public key protocols, *IEEE Transactions on information theory*, Vol. 29, No. 2, pp. 198–208 (1983).
[10] ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Transactions on Information Theory*, Vol. 31, No. 4, pp. 469–472 (1985).
[11] Felsen, S., Kiss, Á., Schneider, T. and Weinert, C.: Secure and Private Function Evaluation with Intel SGX, *CCSW@CCS*, ACM, pp. 165–181 (2019).
[12] Gentry, C.: Fully homomorphic encryption using ideal lattices, *STOC*, pp. 169–178 (2009).
[13] Goldreich, O.: Cryptography and cryptographic protocols, *Distributed Comput.*, Vol. 16, No. 2-3, pp. 177–199 (2003).
[14] Gupta, D., Mood, B., Feigenbaum, J., Butler, K. R. B. and Traynor, P.: Using Intel Software Guard Extensions for Efficient Two-Party Secure Function Evaluation, *Financial Cryptography Workshops*, Lecture Notes in Computer Science, Vol. 9604, Springer, pp. 302–318 (2016).
[15] Intel: 3rd Gen Intel Xeon Scalable Processors (2021).
[16] Katz, J.: Universally Composable Multi-party Computation Using Tamper-Proof Hardware, *EUROCRYPT*, Lecture Notes in Computer Science, Vol. 4515, Springer, pp. 115–128 (2007).
[17] Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K. and Costa, M.: Oblivious Multi-Party Machine Learning on Trusted Processors, *USENIX Security Symposium*, USENIX Association, pp. 619–636 (2016).
[18] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, *EUROCRYPT*, pp. 223–238 (1999).
[19] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G. and Russinovich, M.: VC3: Trustworthy Data Analytics in the Cloud Using SGX, *IEEE Symposium on Security and Privacy*, IEEE Computer Society, pp. 38–54 (2015).
[20] Shamir, A.: How to Share a Secret, *Communications of the ACM*, Vol. 22, No. 11, pp. 612–613 (1979).
[21] Valiant, L. G.: Universal Circuits (Preliminary Report), *STOC*, ACM, pp. 196–203 (1976).
[22] Yao, A. C.: How to Generate and Exchange Secrets (Extended Abstract), *FOCS*, pp. 162–167 (1986).
[23] 桐淵直人，五十嵐大，濱田浩気，菊池　亮：プログラマブルな秘密計算ライブラリ MEVAL3，*SCIS* (2018).