Vol.2019-OS-145 No.11
2019/3/1

sgx-ca: Intel SGXを用いたクラウドベンダ非信頼のアプリケーション実行基盤の提案と実装

秋山 晃丞[†1,a)]　福田 浩章[†1,b)]　菅谷 みどり[†1,c)]

概要：近年，物理マシンのリソースを仮想マシンとしてユーザに提供するクラウドサービスの普及が進んでいる．このようなクラウドサービスではユーザはクラウドベンダが所持している物理マシンの OS やハイパーバイザなどの特権ソフトウェアにユーザのアプケーションを展開する必要があるが，クラウドサービスの構築に使用される特権ソフトウェアは脆弱性を含む可能性があり，その脆弱性に対して様々な攻撃方法が報告されている．また，クラウドベンダ外部からのネットワークを介した攻撃に加え，クラウドベンダ内部から物理マシンに直接攻撃を行い，情報を盗み取る方法も提案されている．
そこで，Intel SGX を用いてユーザは信頼しないクラウドベンダで実行されているアプリケーションに対して完全性と機密性を検証し，信頼性を保証する方法が用いられている．
しかし，複数のアプリケーションの実行した際にユーザはそれぞれのアプリケーションに検証の必要な情報を管理する必要がある，また，ユーザはクラウドベンダで実行されているアプリケーションごとに安全な通信経路を確保しなければならない．
そこで本稿では，Intel SGX を用いてユーザがクラウドベンダを信頼することなく認証サーバを信頼することでクラウドベンダで実行されているアプリケーションを信頼することができユーザが検証のために必要な情報を管理する必要がなく，安全な通信経路の確保はクラウドベンダの物理マシンごとに行えばよいシステム，sgx-ca を提案する．

キーワード：Intel SGX，クラウドサービス

sgx-ca: Application execution platform on untrusted cloud vendor using Intel SGX

Keywords: Intel SGX, Cloud service

1. はじめに

近年，物理マシンのリソースを仮想マシンとしてユーザに提供するクラウドサービスの普及が進んでいる．このようなクラウドサービスではユーザはクラウドベンダが所持している物理マシンの OS やハイパーバイザなどの特権ソフトウェアにユーザのアプケーションを展開する必要がある．しかし，クラウドサービスの構築に使用される特権ソフトウェアは脆弱性を含む可能性があり，その脆弱性に対して様々な攻撃方法が報告されている [1][2]．また，クラウドベンダ外部からのネットワークを介した攻撃に加え，クラウドベンダ内部から物理マシンに直接攻撃を行い，情報を盗み取る方法も提案されている [3].

これらを受けて，耐タンパー性をもつ CPU である Intel SGX[4] を用いてユーザが展開したアプリケーションを保護するシステムが提案されている [5]．SGX は Intel の CPU に搭載されている物理メモリのデータ保護機構であり，CPU 内部に保持している秘密鍵によって物理メモリの暗号化を行う．この暗号化された領域を ENCLAVE と呼び，特権ソフトウェアは ENCLAVE にあるデータを復号化することはできず，メモリにアクセスして読み取るこ

[†1] 現在，芝浦工業大学
Presently with Shibaura Institute of Technology
[a)] ma17003@shibaura-it.ac.jp
[b)] hiroaki@shibaura-it.ac.jp
[c)] doly@shibaura-it.ac.jp

ともできない．また，SGX は ENCLAVE に対するデータを改ざんを検出することができる．これにより SGX は ENCLAVE に格納されているデータの完全性と機密性を保証する．ENCLAVE で実行されているプログラムは他の物理マシンから検証を行うことができる．ユーザがアプリケーションを信頼しないクラウドベンダで実行したい場合には，ユーザがアプリケーションを直接検証する必要がある．しかし，ユーザが複数のアプリケーションを実行した場合は，それぞれのアプリケーションを検証するためにユーザはクラウドベンダと安全な通信経路を確立しなければならない．また，ユーザはアプリケーションの検証のために必要な情報を管理する必要ある．

そこで本稿では，ユーザがクラウドベンダを信頼することなく認証サーバを信頼することでクラウドベンダで実行されているアプリケーションを信頼することができるシステム，sgx-ca を提案する．これにより，ユーザはクラウドベンダと安全な通信経路の確立ならびに検証に必要な情報の管理をする必要がなくなる．また，安全な通信経路の確保はクラウドベンダの物理マシンごとに 1 度だけでよい．

以下，2 節でクラウドベンダに対する攻撃と課題について述べる．3 節でユーザがクラウドベンダと安全な通信経路の確保ならびに検証に必要な情報の管理をする必要がなくなる sgx-ca を提案する．4 節で sgx-ca の実装について述べる．5 節で評価の結果について述べる，6 で関連研究について述べ，7 節で本稿のまとめと今後の課題を述べる．

2. Intel SGX

物理マシンのリソースを仮想マシンとしてユーザに提供するクラウドサービスの普及に伴い，クラウドベンダへの攻撃の事例も多くなっている．さらに，ネットワークを介したクラウドベンダ外部からの攻撃に加え，クラウドベンダ内部からの攻撃も提案されている [3]．これらの対策として，Intel TXT[9] などの耐タンパー性を持つ CPU を用いたシステムの保護や完全準同型暗号 [7] を用いた機密情報の漏洩対策などが提案されている．しかし，耐タンパー性を持つ CPU を用いたシステムの保護は特権ソフトウェアの完全性を保証することはできるが，脆弱性が存在していた場合はその脆弱性を用いた攻撃を受けることになる．また，特権ソフトウェアのアップデートした際の更新箇所が大きくなる．また，完全準同型暗号を用いた方法は処理のオーバヘッドが非常に大きくなる [8]．

そこで，Intel はこれらの問題点の解決のため新しい耐タンパー性を持つチップである Intel SGX[4] を発表した．SGX は Intel の CPU に搭載されている物理メモリのデータ保護機構である．SGX は CPU 内部に保持している秘密鍵によって物理メモリの暗号化を行う．この暗号化された領域を ENCLAVE と呼び，特権ソフトウェアは ENCLAVE にあるデータを復号化することはできない．また，SGX は ENCLAVE にあるデータの改ざんを検出することができる．これにより SGX は ENCLAVE にあるデータの完全性と機密性を保証する．SGX で動作するアプリケーションは ENCLAVE で動作するプログラムと通常の物理メモリで動作するプログラムで構成される．本稿では前者を ENCLAVE プログラム，後者を単にプログラムと呼ぶ．

実行したアプリケーションの ENCLAVE プログラムを検証する際は SGX がアテステーションの機構を提供している．アテステーションにはローカルアテステーションとリモートアテステーションの 2 種類がある．ローカルアテステーションは同一物理マシンで実行されている ENCLAVE プログラムが鍵交換を行い，安全な通信経路を確保する．リモートアテステーションは異なる物理マシンの ENCLAVE プログラムの完全性を Intel が提供するサービスを用いて検証する．これらの検証の際には，実行されている ENCLAVE プログラムに対して測定を行い生成した測定値と検証者が実行する前に測定した測定値を比較する．

単に SGX を用いる場合，ユーザが信頼しないクラウドベンダでアプリケーションを実行したい場合には，ユーザがクラウドベンダで実行されているアプリケーションを検証する必要がある．したがって，ユーザが複数のアプリケーションを実行し，それらに対して検証を行いたい場合は，ユーザはアプリケーションごとに安全な通信経路を確立しなければならない．また，ユーザはアプリケーションの検証のために必要な情報を事前に測定し管理する必要ある．

3. 提案

本稿では，ユーザがクラウドベンダを直接信頼することなく認証サーバを信頼することでクラウドベンダで実行されているアプリケーションの ENCLAVE プログラムを信頼することができるシステム，sgx-ca を提案する．sgx-ca を用いることでユーザは自身の物理マシンでのアプリケーションのコンパイルと ENCLAVE プログラムの検証のために必要な情報の管理の必要がなくなる．また，クラウドベンダが複数のアプリケーションを実行しても，認証サーバが行うリモートアテステーションな回数は一度だけでよい．sgx-ca の管理者は認証サーバを提供する．この認証サーバが ENCLAVE プログラムの測定や検証をユーザの代わりに行う．また，クラウドベンダはこの認証サーバから直接検証を受ける sgx-ca が提供する特別なアプリケーションを実行する．この特別なアプリケーションはユーザのアプリケーションの検証を認証サーバの代わりに行う．

sgx-ca のアーキテクチャを**図 1** に示す．sgx-ca は認証サーバ，Intel 提供 サービス，クラウドベンダならびにアプリケーションデータベースの 4 つのコンポーネントによ

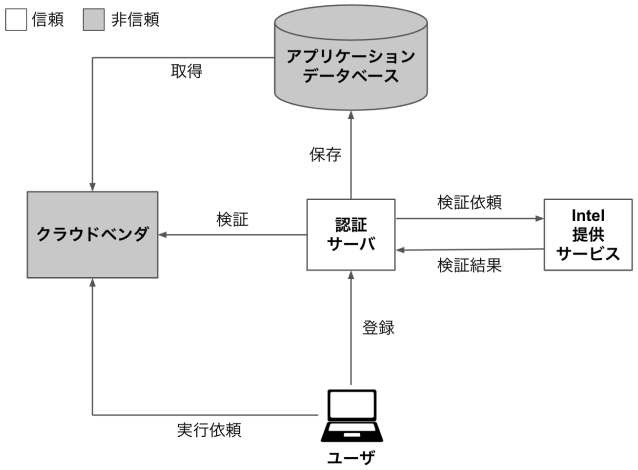

図 1 sgx-ca のアーキテクチャ

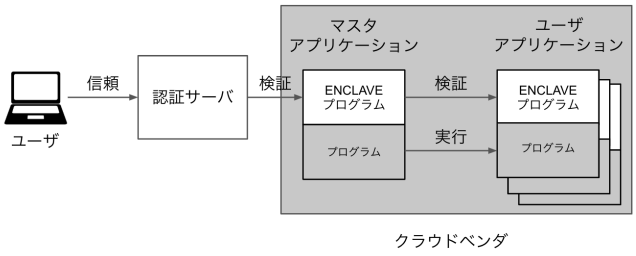

図 2 アプリケーションの検証方法

り構成されている．認証サーバはユーザアプリケーションを受け取る．次に，ユーザアプリケーションの ENCLAVE プログラムに対して測定を行い，これにより得られた情報と共にアプリケーションデータベースにアプリケーションを保存する．また，クラウドベンダの検証も行う．Intel 提供サービスは認証サーバと通信を行い検証依頼を受けて，ENCLAVE プログラムやクラウドベンダの SGX の完全性と機密性を検証したのち検証結果を認証サーバに返す．アプリケーションデータベースは認証サーバがユーザから受け取ったアプリケーションの保存を行う．クラウドベンダは認証サーバからの検証を受けたのち，ユーザからのアプリケーション実行依頼を受けてアプリケーションデータベースよりアプリケーションを取得し実行を行う．

sgx-ca の脅威モデルはクラウドベンダ外部からのネットワークを介した攻撃に加え，クラウドベンダ内部からの物理マシンと特権ソフトウェアへの攻撃も想定している．ユーザが信頼するのは認証サーバと Intel 提供サービスである．

これより，認証サーバによるアプリケーションの検証方法，認証サーバ秘密鍵，ユーザの利用方法，ユーザ実行したいアプリケーションの実行と検証手順ならびに想定される構成要素への改ざんについて述べる．

3.1 認証サーバによるアプリケーションの検証方法

認証サーバによるアプリケーションの検証方法の概要を**図 2** に示す．sgx-ca ではユーザはクラウドベンダを直接信頼せず，代わりに認証サーバを信頼する．この認証サーバは，ユーザが実行したいアプリケーションの ENCLAVE プログラムに対して測定を行う．また，そのときの測定値を管理する．sgx-ca ではクラウドベンダは認証サーバから検証を受ける特別なアプリケーションを実行する．このアプリケーションは認証サーバから完全性と機密性を検証され，結果が正しければ認証サーバはこの特別なアプリケーションを信頼する．次に，この検証を受けた特別なアプリケーションはユーザから実行依頼のあったアプリケーションを実行し検証を行い，結果が正しければ信頼する．このユーザは認証サーバを信頼し，認証サーバはクラウドベンダが実行した特別なアプリケーションを信頼し，この特別なアプリケーションはユーザから実行依頼のあったアプリケーションを実行し信頼する信頼性のチェーンを用いることでユーザが直接認証サーバが信頼しなくてもクラウドベンダで実行されるユーザのアプリケーションの信頼性を実現している．

sgx-ca では，この認証サーバから検証を受ける特別なアプリケーションをマスタアプリケーションと呼び，認証サーバの管理者はマスタアプリケーションの事前に測定値を測定して保存する．その後，マスタアプリケーションのソースコードを各クラウドベンダに配布する．ユーザからの実行依頼を受けて実行したアプリケーションをユーザアプリケーションと呼ぶ．

3.2 認証サーバ秘密鍵

認証サーバはマスタアプリケーションの検証が完了した後，安全な通信路を用いて認証サーバ秘密鍵をマスタアプリケーションの ENCLAVE プログラムに送信する．この秘密鍵は認証サーバやユーザが対応する公開鍵を用いて暗号化したデータを復号する．クラウドベンダは認証サーバから送られる秘密鍵を得ることはできない．クラウドベンダが秘密鍵を得る方法にはリモートアテステーションを偽って認証サーバから秘密鍵を受け取る方法と，秘密鍵を復号するように改ざんされたマスタアプリケーションを実行する方法が考えられる．前者は，リモートアテステーションにより秘密鍵が渡されるためクラウドベンダの特権ソフトウェアはこの通信内容をできない．また，リモートアテステーションにより SGX の完全性も保証されるため，リモートアテステーションを偽った通信を行い認証サーバから秘密鍵を受け取ることはできない．後者は，リモートアテステーションの際に認証サーバが事前に測定した測定値とクラウドベンダのマスタアプリケーションの ENCLAVE プログラム測定値を比較する．マスタアプリケーションに改ざんを行っていた場合この値が異なるため，改ざんを検出できる．

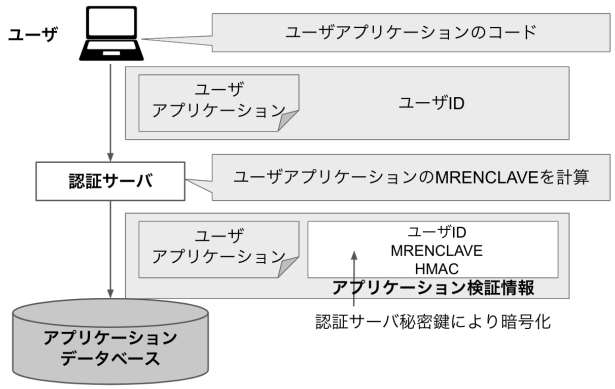

図 3 認証サーバへの登録

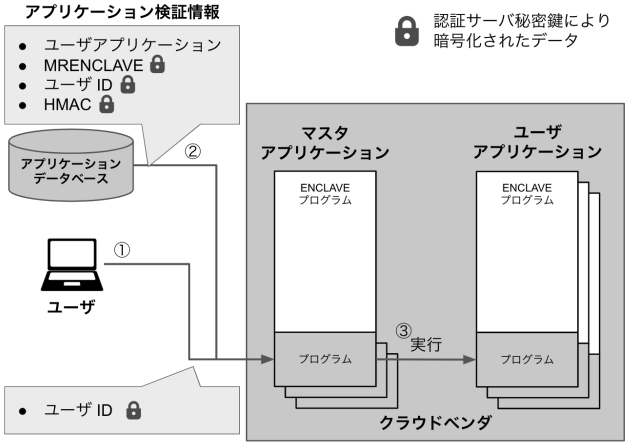

図 4 検証に使用する情報の取得

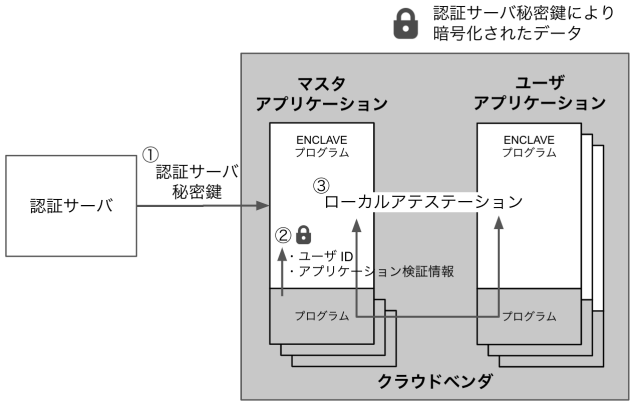

図 5 ユーザアプリケーション実行手順

3.3 ユーザの利用方法

ユーザは認証サーバに事前に申請を行い，認証サーバがユーザを一意に識別するユーザ ID を受け取る．また，ユーザがクラウドベンダで実行されるアプリケーションを開発する際は sgx-ca が提供するライブラリを用いる．このライブラリにはマスタアプリケーションとリモートアテステーションを行い検証を受ける処理が書かれている．ユーザはこのライブラリをリンクし，このライブラリが提供するエンドポイントからアプリケーションが開始するようにアプリケーションを開発する．

3.4 ユーザアプリケーションの実行の手順

ユーザがアプリケーションをクラウドベンダで実行する際は，まずユーザアプリケーションの認証サーバの登録を行う．そしてマスタアプリケーションのプログラムがユーザアプリケーションの実行し，クラウドベンダの ENCLAVE プログラムが検証を行う．ユーザアプリケーションの ENCLAVE プログラムの検証を行うために，ユーザ ID と ENCLAVE プログラムのハッシュから計算される値である MRENCLAVE を使用する．この値を用いて ENCLAVE プログラムの改ざんを検出する．sgx-ca ではこれらのデータに HMAC を付加したものをアプリケーション検証情報として検証に使用する．MRENCLAVE は CPU に非依存の値であり，異なる CPU で MRENCLAVE を計算しても対象の ENCLAVE プログラムが同一の場合，MRENCLAVE も同一となる．これより，ユーザアプリケーションの認証サーバへの登録，ユーザアプリケーションの実行ならびにユーザアプリケーションの検証の手順について述べる．

3.4.1 ユーザアプリケーションの認証サーバへの登録

認証サーバへの登録の概要を**図 3** に示す．まず，ユーザはユーザアプリケーションとユーザ ID を認証サーバに送信し，アプリケーションの登録を申請する．これらを受け取った認証サーバは測定を行う．ユーザアプリケーションの ENCLAVE プログラムの MRENCLAVE を測定する．次に，ユーザ ID と MRENCLAVE に改ざん防止のための HMAC を付加したアプリケーション検証情報を認証サーバの秘密鍵により暗号化する．そして，ユーザアプリケーションと暗号化したアプリケーション検証情報をアプリケーションデータベースに保存する．

3.4.2 ユーザアプリケーションの実行

マスタアプリケーションがユーザアプリケーションを実行する際は検証に必要な情報を ENCLAVE プログラムに渡す．このときに渡す情報はユーザ ID とアプリケーション検証情報である．これらの情報をマスタアプリケーションが取得し，ユーザアプリケーションを実行するまでの概要を**図 4** に示す．①ユーザがアプリケーションの実行する際はクラウドベンダにアプリケーションの実行を依頼する．このときにユーザはユーザ ID を認証サーバに渡す．②実行依頼を受け取ったマスタアプリケーションはアプリケーションデータベースから該当するユーザアプリケーションとそれに付随する情報を取得する．③マスタアプリケーションはユーザアプリケーションを実行する．

3.4.3 ユーザアプリケーションの検証

マスタアプリケーションの ENCLAVE プログラムがユーザアプリケーションの検証を行う際は，マスタアプリ

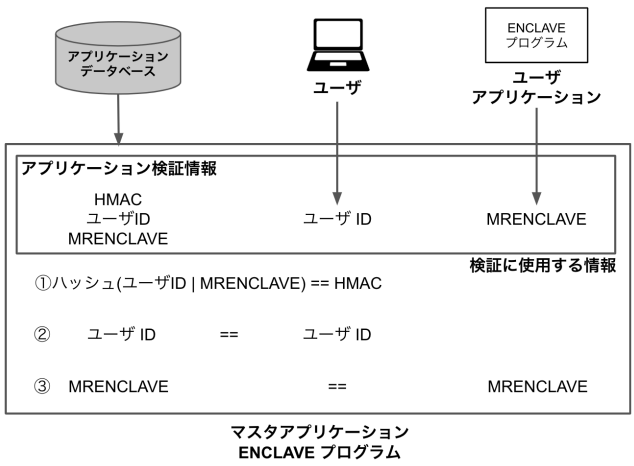

図 6 マスタアプリケーションによる検証手順

ケーションのプログラムから ENCLAVE プログラムへ検証に用いるデータを渡す．検証のために必要な情報はアプリケーションデータベースに保存されていたアプリケーション検証情報，実行を依頼したユーザのユーザ ID ならびにクラウドベンダが実行しようとしているユーザアプリケーションの MRENCLAVE である．検証に用いるこれらデータをマスタアプリケーションの ENCLAVE プログラムに渡す手順の概要を**図 5** に示す．①マスタアプリケーションは事前に認証サーバとリモートアテステーションを行い，認証サーバから認証サーバ秘密鍵を受け取る．この通信はクラウドベンダがマスタアプリケーションを起動した一度のみ行えば良い．②マスタアプリケーションのプログラムはユーザから受け取ったユーザ ID とアプリケーションデータベースから取得したアプリケーション認証情報を ENCLAVE プログラムに引き渡す．③これらを受け取ったマスタアプリケーションの ENCLAVE プログラムはユーザアプリケーションとローカルアテステーションを行い，ユーザアプリケーションの ENCLAVE プログラムの MRENCLAVE を取得する．次に，マスタアプリケーションの ENCLAVE プログラムが受け取った情報を基にユーザアプリケーションの ENCLAVE プログラムを検証する．

次に，ENCLAVE プログラムが渡されたデータを用いて検証を行う手順を**図 6** に示す．① HMAC はアプリケーション検証情報のユーザ ID と MRENCLAVE から生成したハッシュ値と比較する．②アプリケーション検証情報のユーザ ID とユーザから渡されたユーザ ID を比較する．③アプリケーション検証情報の MRENCLAVE とユーザアプリケーションの ENCLAVE プログラムからローカルアテステーションで受け取った MRENCLAVE を比較する．これらの検証結果が正しければマスタアプリケーションの ENCLAVE プログラムはユーザアプリケーションの ENCLAVE プログラムの完全性と機密性を認める．その後，マスタアプリケーションの ENCLAVE プログラムは認証サーバに実行の完了を通知する．通知を受け取った認証サーバはユーザに実行が完了したことを報告する．

3.5 想定される構成要素への改ざん

想定される sgx-ca の構成要素への改ざんには以下が考えられる．

① マスタアプリケーションの ENCLAVE プログラムへの改ざん
② ユーザアプリケーションの ENCLAVE プログラムへの改ざん
③ アプリケーションデータベースのアプリケーション認証情報への改ざん
④ クラウドベンダの物理マシンが搭載している SGX のなりすまし
⑤ 登録したユーザとは異なるユーザによる実行依頼である

①は，アプリケーション検証情報に付加した HMAC で検証可能である．②は，認証サーバがリモートアテステーションの際に事前に測定した MRENCLAVE と比較を行い検証可能である．攻撃者がクラウドベンダのマスタアプリケーションを改ざんしても，MRENCLAVE が異なる．③は，マスタアプリケーションがリモートアテステーションした際に ユーザアプリケーションから取得した MRENCLAVE と，アプリケーション認証情報の MRENCLAVE が異なるため検知できる．また，sgx-ca が提供しているライブラリに対する改ざんも MRENCLAVE を変化させるため検証できる．④は，認証サーバからのリモートアテステーションにより，SGX の信頼性を検証できる．⑤は，アプリケーション認証情報に付加したユーザ ID により検証できる．

4. 実装

本稿では，マスタアプリケーション，ユーザアプリケーションが使用するライブラリならびに認証サーバを Intel SGX SDK 1.6[6] を用いて実装した．

4.1 sgx-ca 提供ライブラリ

ユーザが開発する際に使用する sgx-ca 提供ライブラリの処理の概要を**図 7** に示す．まず，sgx-ca 提供ライブラリはユーザアプリケーションの ENCLAVE プログラムを実行し初期化を行う．次に，マスタアプリケーションから渡されたマスタアプリケーションへの接続情報を用いて，マスタアプリケーションへ接続する．今回の実装では，マスタアプリケーションとユーザアプリケーションは unix ドメインソケットを用いてプロセス通信を行う．そのため，マスタアプリケーションから渡される接続情報は unix ドメインソケットへのフォルダパスである．①sgx-ca 提供ライブラリはユーザコードのエントリポイントとなる関数

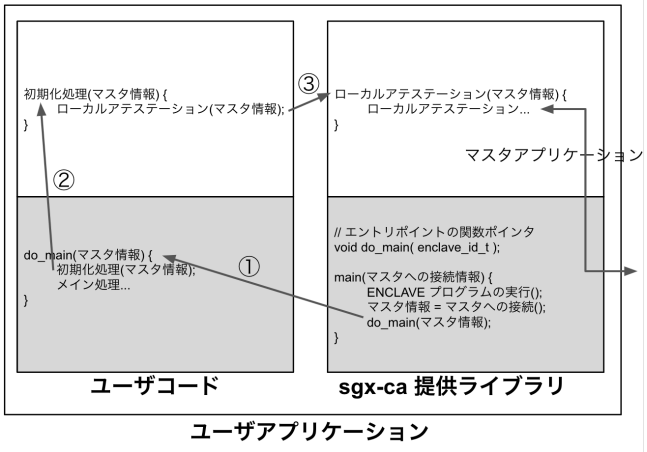

図 7 sgx-ca 提供ライブラリの処理

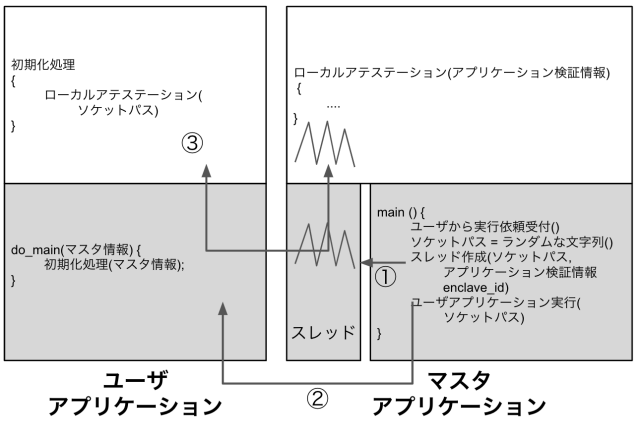

図 8 ユーザアプリケーションの実行手順

を実行する．②この関数にはローカルアテステーションを行う初期化処理が記載されている．この初期化処理が完了した後，ユーザコードの処理が開始される．③ユーザアプリケーションの測定のためにプログラムに MRENCLAVE を出力する処理も行う．

4.2 ユーザアプリケーションの測定

認証サーバが ENCLAVE プログラムの MRENCLAVE を取得する際は，ENCLAVE プログラムを実行して取得する．この測定には SGX で追加された ENCLAVE プログラムのみが実行できる CPU 命令である EREPORT 命令を使用する．この EREPORT 命令で得られる構造体の中に MRENCLAVE が含まれている．認証サーバのプログラムは ENCLAVE プログラムにリンクされた sgx-ca 提供ライブラリにある MRENCLAVE を出力する処理を呼び出して MRENCLAVE を取得する．

4.3 ユーザアプリケーションの実行

マスタアプリケーションによるユーザアプリケーションの実行手順の概要を**図 8** に示す．①マスタアプリケーションはユーザから実行依頼を受け付ける．実行依頼が来たらランダムな文字列を生成する．②スレッドを作成し生成した文字列を渡す．このスレッドは，受け取ったソケットパスを用いてユーザアプリケーションとのローカルアテステーションを行う．③マスタアプリケーションはユーザアプリケーション実行してソケットパスを渡す．ソケットパスを受け取ったユーザアプリケーションは，マスタアプリケーションとローカルアテステーションを行う．このときに，マスタアプリケーションは ENCLAVE プログラムを 1 度のみ実行する．SGX は ENCLAVE プログラムのスレッド管理機構を持っているため，プログラムのスレッドを作成した際にもう一度 ENCLAVE プログラムを実行する必要はない．

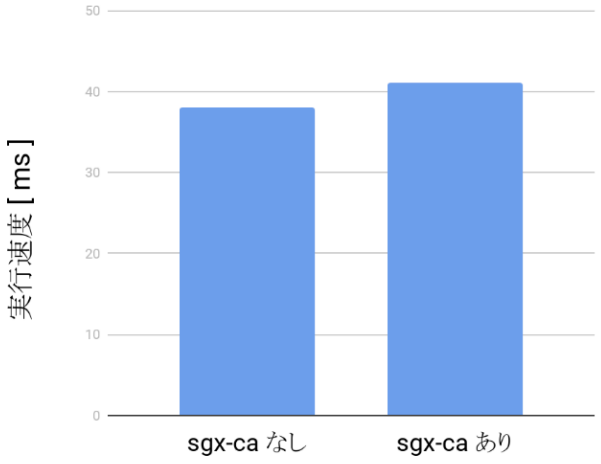

図 9 ユーザアプリケーション実行時のオーバヘッド

5. 評価

アテステーションによるアプリケーション実行時のオーバヘッドと，sgx-ca が提供するライブラリをリンクすることによるユーザアプリケーションのサイズの増大に対し評価を行った．評価に使用した物理マシンの CPU は i7-7700K，マザーボードは BIOS のバージョンは 2.70 の ASRock Z270 Extreme4，メモリは 16GB のメモリを搭載している．また，OS は Ubuntu 18.04 TLS を使用した．

5.1 ユーザアプリケーション実行時のオーバヘッド

sgx-ca を用いることによるユーザアプリケーション起動時のオーバヘッドについて評価した．評価に用いたコードは何もせずに main 関数を呼び出して終了するコードである．その理由は，sgx-ca では性能低下は起動時のみ起こるからである．評価の結果を**図 9** に示す．sgx-ca によるアプリケーションのオーバヘッドは 3ms でありこれは実用に耐えうると考えられる．

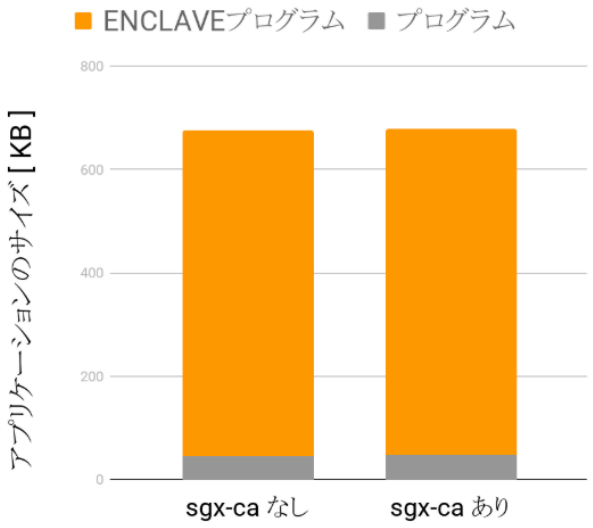

図 10 ライブラリによるユーザアプリケーションの増加

5.2 ライブラリによるユーザアプリケーションの増加

sgx-ca 提案ライブラリのリンクによるアプリケーションの増大を評価した．また，この評価項目についても同様にアプリケーションは main 関数を呼ぶのみあるが，具体的な main 関数の内容には依存しない．評価の結果を**図 10** に示す．これらの差は約 3.4KB であった．これは実用に耐えうると考えられる．

6. 関連研究

Intel TXT[9] を用いて特権ソフトウェアの完全性を保証したうえでアプリケーションを検証を行うことは可能であるが，特権ソフトウェアが含む脆弱性を用いて攻撃される可能性がある．また，特権ソフトウェアのアップデートのさいの変更箇所が大きくなる．

完全準同型暗号を用いたクラウドベンダ内部からの情報漏洩への対策が提案されている [7]．しかし，完全準同型暗号を用いた計算のオーバヘッドは非常に大きくなる [8]．SGX を用いたアプリケーションもオーバヘッドは増大するが，その差は数倍程度である [13]．

SGX を用いた他のアプリケーションの実行基盤の提案には SGX-Aware[5] がある．この提案は Kubernetes[11] を用いてアプリケーションを実行し，ENCLAVE の使用量をもとに物理マシンをスケジューリングする方法である．しかし，この提案ではユーザがクラウドベンダの各物理マシンに安全な通信経路を確保して ENCLAVE プログラムの検証を行う必要がある．

ライブラリ OS により未改変のアプリケーションを ENCLAVE プログラムで実行する提案がある [12][13]．これらの提案を sgx-ca で使用することによりユーザは未改変のアプリケーションをクラウドベンダを信頼せずに実行できるようになる．

7. まとめ

本稿では，ユーザがクラウドベンダを信頼することなく認証サーバを信頼することでクラウドベンダで実行されているアプリケーションの ENCLAVE プログラムを信頼することができるシステム，sgx-ca を提案した．sgx-ca は Intel SGX を用いることでユーザがクラウドベンダを直接信頼することなく認証サーバを信頼することでクラウドベンダで実行されているアプリケーションを信頼することができる．さらにユーザは SGX の環境がなくても使用でき，測定情報を管理する必要がない．実装は Intel SGX SDK 1.6 を用いて sgx-ca を用いて行った．また，アプリケーション実行時のオーバヘッドとサイズの増大を評価し，それぞれ sgx-ca 未使用の場合と大きな差がないことを示した．

今後の課題として，ライブラリ OS に対応することでユーザが未改変のアプリケーションを sgx-ca で実行できるようにすることが挙げられる．また，測定情報を CPU 命令のエミュレーションにより取得することなどを挙げられる．

参考文献

[1] S. Checkoway and H. Shacham. Iago attacks: why the system call API is a bad untrusted RPC interface. In 18th International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.
[2] VENOM: https://venom.crowdstrike.com/
[3] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino , A. J. Feldman, J. Appelbaum, and E. W. Felten Lest We Remember: Cold Boot Attacks on Encryption Keys, Proceedings of the 17th Usenix Security Symposium, 2008.
[4] Intel SGX: https://software.intel.com/en-us/sgx
[5] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni and C. Fetzer SGX-Aware Container Orchestration for Heterogeneous Clusters, IEEE International Conference on Distributed Computing Systems, 2018.
[6] Intel Corporation. Intel software guard extensions for Linux OS. https://github.com/01org/linux-sgx.
[7] C. Gentry. A fully homomorphic encryption scheme. PhD thesis, Stanford University, 2009.
[8] M. Naehrig, K. Lauter, and V. Vaikuntanathan. Can homomorphic encryption be practical? Proceedings of the 3rd ACM workshop on Cloud computing security workshop, pages 113124. ACM, 2011.
[9] D. Grawrock. Dynamics of a Trusted Platform: A building block approach. Intel Press, 2009.
[10] T. Alves and D. Felton. Trustzone: Integrated hardware and software security. Information Quarterly, 2004.
[11] kubernetes: https://kubernetes.io/
[12] C. Tsai, D. E. Porter, M. Vij ”Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX” Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, 2017.
[13] A. Baumann, M. Peinado, and G. Hunt. ”Shielding applications from an untrusted cloud with haven” Proceed-

ings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 267283, 2014