Intel SGXを利用するコンテナのマイグレーション機構

中島 健児[1]　光来 健一[1]

概要：近年，情報漏洩対策の重要性が高まっているが，OSによる対策は攻撃者によって無効化される恐れがある．そこで，アプリケーション内の機密情報を保護するために，Intel SGXと呼ばれるCPU機構が用いられるようになっている．SGXを用いることで，CPU固有の鍵でメモリが暗号化された保護領域であるEnclaveを作成することが可能になる．しかし，コンテナ・マイグレーションによりEnclaveを含むSGXアプリケーションを別のホストに移動した場合には，アプリケーションの実行を継続することができなくなる．マイグレーションによってCPUが変わることにより，Enclaveのメモリを正常に復号できなくなるためである．本稿では，コンテナ・マイグレーション後にもSGXアプリケーション内のEnclaveを継続的に実行可能にするMigSGXを提案する．MigSGXでは，Enclaveが自身のメモリ上のデータをCPU非依存の鍵を用いて暗号化し，それを外部メモリに書き出すことで状態の保存を実現する．マイグレーション先で状態を復元する際には，Enclave自身が保存したデータを復号してメモリを上書きする．我々はMigSGXをIntel SGX SDKおよびCRIUに実装し，SGXアプリケーションの状態の保存・復元について性能を測定した．

キーワード：Intel SGX, マイグレーション, コンテナ, 情報漏洩, 機密情報

1. はじめに

近年，コンピュータ上で様々な情報を扱うようになり，情報漏洩に対する対策が喫緊の課題となっている．これまで，システムの根幹をなすOSにおいて様々な対策が行なわれてきた．例えば，LinuxではSELinux[1]やAppArmor[2]などを用いたアクセス制御によりアプリケーションの権限を制限する機能を提供している．アクセス制限により，マルウェアがアプリケーションの機密情報を盗んだり，機密情報を外部に送信したりすることを防ぐことができる．しかし，OSによる対策はOSが攻撃を受けて乗っ取られてしまうと無効化される恐れがある．その結果，マルウェアによる機密情報へのアクセスを制限できなくなったり，OSからアプリケーション内の機密情報を盗まれたりする可能性がある．

そこで，最近用いられるようになっているのがIntel SGX[3][4][5]による機密情報の保護である．SGXを用いて作成されたアプリケーション内では，CPU固有の鍵でメモリが暗号化された保護領域であるEnclaveを作成することが可能になる．Enclaveの中の情報にはOSを含むいかなるプログラムもアクセスできないため，Enclaveの内部で機密情報を扱うことで，マルウェアやOSから機密情報を保護することができる．しかし，SGXを用いるSGXアプリケーションが動作しているコンテナを別のホストにマイグレーションすると，実行を継続できなくなる．Enclaveのメモリはそれを作成したCPUの鍵で暗号化されており，マイグレーションによりEnclaveを実行するCPUが変わると鍵も変わってしまうためである．その結果，Enclaveメモリの復号に失敗してEnclaveの実行を継続できなくなってしまう．

本稿では，コンテナ・マイグレーション後にもSGXアプリケーション内のEnclaveを継続的に実行可能にするMigSGXを提案する．Enclaveの状態はSGXによる保護により外部から保存・復元することができないため，MigSGXではEnclave自身に状態の保存・復元を行わせる．状態の保存時にはEnclaveが自身のメモリ上のデータを外部メモリに書き出し，状態の復元時には保存したデータでEnclaveのメモリを上書きする．その際に，保存した状態データからの情報漏洩を防ぐために，CPU非依存の鍵を用いてEnclave内で暗号化・復号化を行う．これらの機能はEnclave内で動作するMigSGXライブラリと，Enclave外部で動作するMigSGXランタイムを通してSGXアプリケーションに提供される．

我々はMigSGXをIntel SGX SDK 1.9[6]とCRIU 3.9[7]に実装した．MigSGXランタイムはEnclave内のMigSGX

[1] 九州工業大学
Kyushu Institute of Technology

© 2018 Information Processing Society of Japan

ライブラリを SGX の ECALL 機構を用いて安全に呼び出し，Enclave のヒープ領域とデータ領域の保存・復元を行う．Enclave 内でデータ領域に関する情報を取得できるようにするために，MigSGX はコンパイル後に Enclave プログラムのバイナリに情報を埋め込む．マイグレーションを行う MigSGX マネージャと MigSGX ランタイムの間の通信をできるだけ安全に行えるようにするために，CRIU のパラサイト機構 [10] を用いる．MigSGX を用いて SGX アプリケーションの保存・復元を行い，復元後の Enclave の正常な動作を確認した．また，Enclave の保存・復元の性能および Enclave 内での暗号化オーバヘッドを測定した．

以下，2 章では情報漏洩対策として用いられる SGX アプリケーションをマイグレーションする際の問題点について述べる．3 章では SGX アプリケーションのマイグレーションを可能にする MigSGX を提案する．4 章では MigSGX の実装について述べ，5 章で MigSGX の有用性を調べるために行った実験について述べる．6 章で関連研究について述べ，7 章で本稿をまとめる．

2. SGX アプリケーション

コンピュータで様々な情報を扱うようになるにつれて，機密情報が外部に漏洩する事件が年々増加している．情報漏洩を防ぐために，システムの根幹をなす OS において様々なセキュリティ対策が施されてきた．例えば，Linux では SELinux[1] や AppArmor [2] などを用いたアクセス制御が提供されており，アプリケーションの権限を制限することができる．アクセス制限により，マルウェアの実行やファイルの不正な読み書き，外部との不正な通信を禁止することができる．これにより，マルウェアが他のアプリケーションの持つ機密情報を盗み出すことを防ぐことができる．

しかし，OS が攻撃を受けて乗っ取られてしまった場合，OS のアクセス制御を無効化される恐れがある．OS には様々な脆弱性が報告され続けており，それらの脆弱性を用いた攻撃が可能である．OS が攻撃を受けた結果，マルウェアにアプリケーションの情報を読み取られ，機密情報を窃取される可能性がある．さらに，OS からアプリケーションの情報を読み取ることは容易であるため，OS 内のマルウェアによって機密情報が盗まれる可能性もある．OS の下で動作するハイパーバイザを用いて OS の挙動を監視することもできる [8] が，ハイパーバイザも同様に攻撃を受ける可能性がある．

そこで，OS やハイパーバイザに依存せずにアプリケーション内の機密情報を保護するために，Intel SGX と呼ばれる CPU の機構が用いられるようになってきた．SGX を用いるアプリケーションでは，図 1 のように CPU 固有の鍵でメモリが暗号化された保護領域である Enclave を作成することが可能になる．Enclave 内の情報には OS を含む

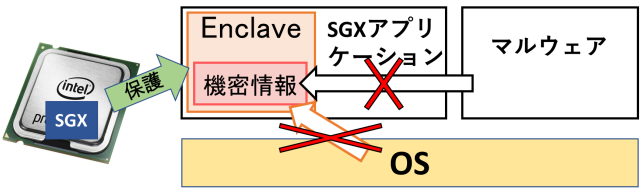

図 1　Intel SGX を用いた機密情報の保護

いかなるプログラムもアクセスできないため，Enclave 内部で機密情報を扱うことでマルウェアや OS 等から機密情報を保護できる．Enclave は CPU のみを信頼すればよく，OS やハイパーバイザ等を信頼する必要はない．Enclave では正しく署名されたプログラムのみが実行可能であり，改ざんされたプログラムは実行できない．また，実行中の Enclave 内のプログラムやデータを改ざんすることもできない．

しかし，SGX アプリケーションにはマイグレーションを行うことができないという問題がある [12]．マイグレーションは仮想マシンやコンテナ，プロセスを別のホストに移動させる技術である．SGX アプリケーションを実行しているホストにメンテナンス等が必要になりホストを停止しなくてはならない場合でも，マイグレーションを行うことによりアプリケーションを継続して実行することができる．マイグレーションはアプリケーションの内部状態を保存し，移送先ホストで復元することにより実現される．近年，Docker[9] 等のコンテナ型仮想化が普及したことにより，コンテナをマイグレーションする重要性が増している．コンテナは OS によって提供されるアプリケーションのための仮想実行環境であり，仮想マシンと比較して軽量であるという特徴がある．コンテナ・マイグレーションではコンテナ内の各アプリケーション（プロセス）がそれぞれマイグレーションされる．

SGX アプリケーションが動作しているコンテナを通常のコンテナと同様にマイグレーションした場合，移送先で SGX アプリケーションの実行を継続することができなくなる．マイグレーションの際には，SGX アプリケーション内の Enclave のメモリは移送元の CPU の鍵によって暗号化された状態のまま移送先に転送される．図 2 のように，移送先の CPU が持つ鍵は移送元の CPU の鍵とは異なるため，移送先では Enclave メモリを正しく復号することができない．そのため，Enclave の実行を継続できなくなってしまう．移送先で SGX アプリケーション自身が Enclave を一から再作成することで対処することも可能であるが，移送元での Enclave 内の状態が失われ，アプリケーション開発者の負担が大きくなる．

3. MigSGX

本稿では，コンテナ・マイグレーション後にも SGX アプリケーション内の Enclave を継続的に実行可能にする

© 2018 Information Processing Society of Japan

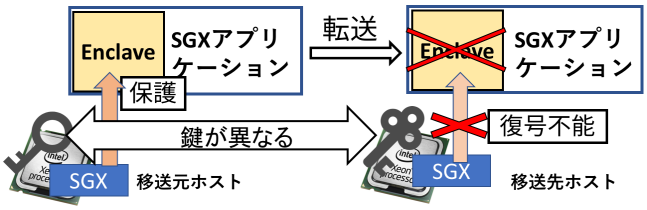

図 2　SGX アプリケーションのマイグレーション

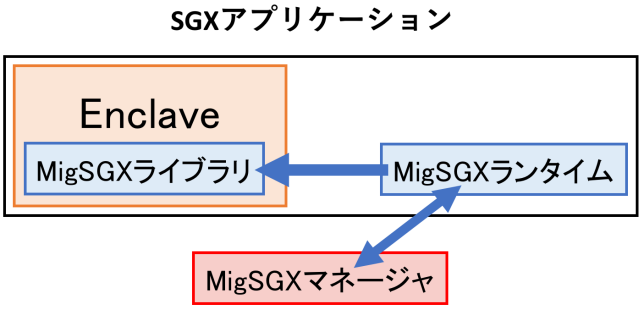

図 3　MigSGX のシステム構成

MigSGX を提案する．Enclave の状態は SGX による保護により外部から保存・復元することができないため，MigSGX では Enclave 自身に状態の保存・復元を行わせる．移送元ホストでは，Enclave が自身のメモリ上のデータを CPU 非依存の鍵を用いて暗号化し，それを外部メモリに書き出すことで状態の保存を実現する．移送先ホストでは，Enclave 自身が保存したデータを復号して自身のメモリを上書きすることにより状態の復元を実現する．暗号化に用いる CPU 非依存の鍵は，Enclave のリモートアテステーションを利用して移送元と移送先の同一の Enclave 間でのみ共有する．

MigSGX のシステム構成を図 3 に示す．MigSGX は Enclave 内部のプログラムに MigSGX ライブラリを提供する．マイグレーション時に MigSGX ライブラリが Enclave の状態の保存・復元を行うことで，アプリケーション開発者にマイグレーションを意識させない．SGX アプリケーションの Enclave 外部では MigSGX ランタイムが動作し，マイグレーション時に Enclave の状態を保存・復元するために Enclave 内の MigSGX ライブラリを呼び出す．コンテナの外部では MigSGX マネージャが動作し，MigSGX ランタイムと通信しながら SGX アプリケーションのマイグレーションを行う．

SGX アプリケーションのマイグレーションの手順は図 4 ようになる．まず，MigSGX マネージャが SGX アプリケーション内の MigSGX ランタイムと通信を行い，Enclave の状態の保存を指示する．MigSGX ランタイムが Enclave 内の MigSGX ライブラリを呼び出すと，MigSGX ライブラリは Enclave の状態を取得しながら暗号化し，Enclave 外部のメモリに書き出す．書き出しが終了すると，MigSGX マネージャは SGX アプリケーション全体の状態の保存を

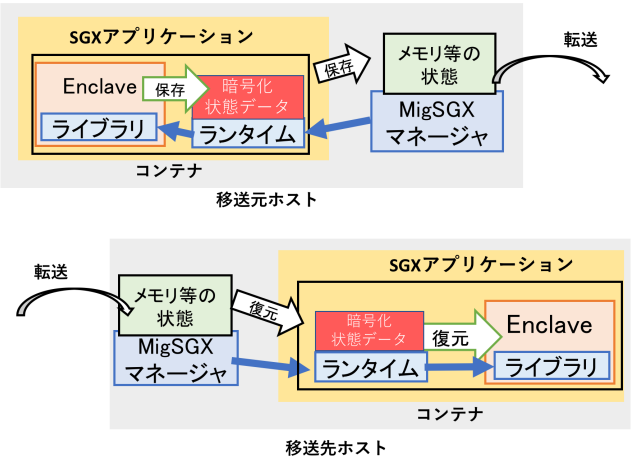

図 4　MigSGX のマイグレーション手順

行う．その際に，書き出された Enclave の状態も一緒に保存される．最後に，保存した状態データを移送先ホストに転送する．

移送先ホストの MigSGX マネージャは，状態データを受け取ると SGX アプリケーションの Enclave 以外の部分を復元する．SGX アプリケーションの復元後，MigSGX マネージャは MigSGX ランタイムと通信を行い，Enclave の状態の復元を指示する．MigSGX ランタイムは新たに Enclave を作成し，Enclave 内部の MigSGX ライブラリを呼び出す．MigSGX ライブラリは外部メモリに保存されている Enclave の状態データを復号しながら Enclave のメモリを上書きすることにより状態を復元する．

4.　実装

我々は MigSGX ライブラリと MigSGX ランタイムを Intel SGX SDK 1.9[6] に実装し，CRIU 3.9[7] をベースに MigSGX マネージャを実装した．

4.1　Enclave メモリの保存

Enclave の状態を保存する際に，MigSGX ランタイムはまず，状態を保存するためのバッファを確保する．次に，Enclave 内の MigSGX ライブラリを ECALL 機構を用いて呼び出す．ECALL はアプリケーション開発者が指定した Enclave 内の関数だけを安全に呼び出せるようにするために SGX が提供する機構である．MigSGX ライブラリは Enclave の状態として Enclave メモリ上のデータを取得し，それを暗号化して確保したバッファに書き込む．

我々は Enclave の状態を保存するための ECALL 関数を MigSGX ライブラリに定義し，MigSGX ランタイムから呼び出せるようにした．Intel SGX SDK では，アプリケーション開発者は Enclave 定義言語（EDL）を用いて，ECALL で呼び出すことのできる Enclave 内の関数を定義する．しかし，この方法ではアプリケーションごとに状態保存のための ECALL 関数を定義する必要がある．そこで，

© 2018 Information Processing Society of Japan

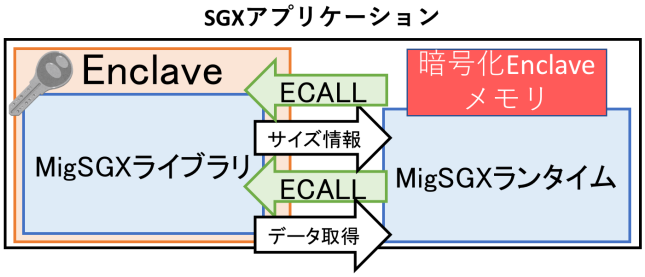

図 5　Enclave の状態データの取得

MigSGX では EDL を用いずに MigSGX ライブラリ内部でアプリケーションに依存しない ECALL 関数を定義した．

MigSGX ランタイムはまず，図 5 のように ECALL を用いて Enclave のヒープ領域とデータ領域のサイズを取得し，それらの領域のデータを格納するために必要なバッファを確保する．次に，そのバッファを指定して ECALL を実行し，Enclave のヒープ領域とデータ領域の暗号化されたデータを取得する．ヒープ領域は malloc 等で動的に確保される領域であり，データ領域は，大域変数または静的変数が含まれる領域である．0 で初期化された変数はデータ領域ではなく BSS 領域に格納されるが，本稿では区別する必要がない場合はまとめてデータ領域と呼ぶ．Enclave メモリ上にはスタック領域も存在するが，MigSGX ではマイグレーション時に Enclave 内の ECALL 関数が実行中でないことを想定しているため，スタック領域のデータの保存は行っていない．また，移送先で Enclave を再作成する際にプログラムがロードされるため，コード領域についても保存しない．

ECALL で呼び出された Enclave 内の MigSGX ライブラリにおいて，ヒープ領域の先頭アドレスとサイズは SDK の内部 API を用いて取得可能である．しかし，データ領域についてはプログラムをロードした後の Enclave 内に情報が存在しない．そこで，MigSGX では Enclave プログラムのコンパイル後に生成されたバイナリに対して署名を行う際に，図 6 のようにバイナリからデータ領域と BSS 領域の先頭アドレスとサイズをそれぞれ取得し，その情報をバイナリに埋め込む．この際に，データ領域と BSS 領域は連続していることを利用して，データ領域の先頭アドレスと両方の領域のサイズの合計のみを埋め込む．そして，MigSGX ライブラリからその情報を取得できるように，データ領域の情報を取得する API を SDK に追加した．

MigSGX ライブラリはヒープ領域とデータ領域のデータを MigSGX ランタイムのバッファに書き出す際に，CPU 非依存の鍵を使って暗号化を行う．MigSGX ランタイムは Enclave の外部で動作しており，攻撃を受けて情報が漏洩する可能性があるためである．暗号処理を高速化するために，CPU の暗号化支援機構である AES-NI を利用する wolfSSL[11] の関数を MigSGX ライブラリに移植した．

MigSGX ランタイムのバッファへのデータの書き込み

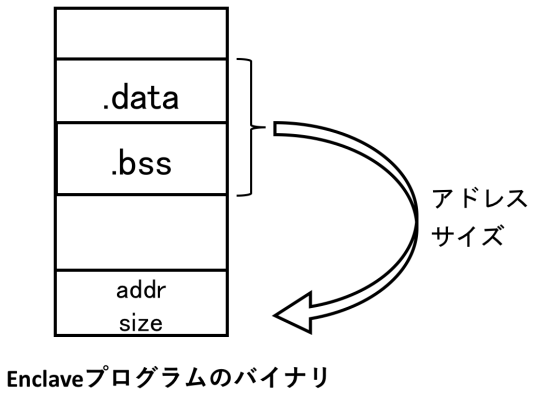

図 6　データ領域・BSS 領域の情報の埋め込み

が完了すると，MigSGX ランタイムは Enclave を終了させる．Enclave は移送先では実行を継続することができないためである．Enclave を終了させておかないと，そのメモリが移送先に転送されることになり，マイグレーション時間が増加する．その後，Enclave を操作するために使われる SGX デバイスをクローズする．CRIU は SGX デバイスの状態を保存できず，マイグレーションに失敗してしまうためである．SGX デバイスは Enclave を終了させた後には使われないため，クローズしても動作に影響はない．

4.2　Enclave メモリの復元

Enclave の状態を復元する際に，MigSGX ランタイムはまず，移送元と同じ Enclave プログラムを用いて Enclave を再作成する．その後，Enclave の状態を復元するための ECALL を用いて MigSGX ライブラリを呼び出す．MigSGX ライブラリは，移送元で MigSGX ランタイムのバッファに保存された Enclave の状態データを読み込み，Enclave の内部で復号して Enclave メモリの復元を行う．

移送先で Enclave を再作成する際に，通常通りに Enclave を作成すると Enclave メモリを復元した後で Enclave 内のプログラムを正常に実行することができない場合がある．この問題は移送元の Enclave 内部でポインタ等のアドレスに依存する変数が使われていた場合に生じる．Enclave は OS によってプロセスのメモリアドレス空間にランダムに配置され，図 7 のように移送元の Enclave と移送先で再作成した Enclave との間でベースアドレスが変わるためである．移送先の Enclave が移送元のアドレスにアクセスしても，正しいデータにアクセスすることはできない．

そこで，移送先の MigSGX ランタイムは Enclave を移送元と同じアドレスに再作成する．そのために，移送元で Enclave を作成する際に Enclave のベースアドレスを MigSGX ランタイム内に保存しておく．そして，移送先で Enclave を再作成する際には保存しておいたアドレスを指定して Enclave を作成する．Enclave のメモリは mmap システムコールを用いて確保されるため，その第 1 引数にア

© 2018 Information Processing Society of Japan

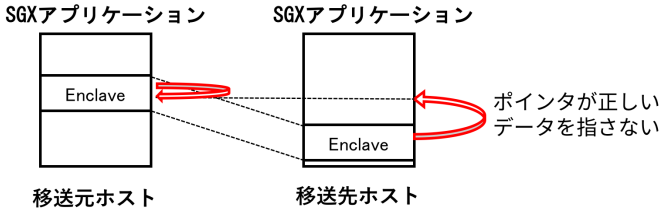

図 7　Enclave の再作成に伴う問題

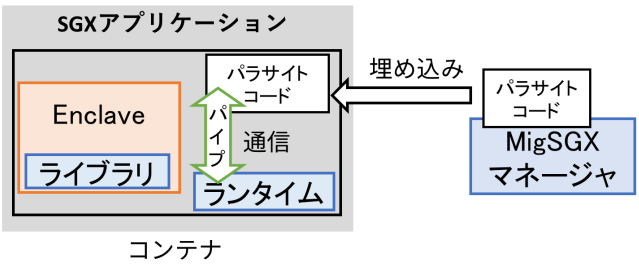

図 8　パラサイト機能を利用した通信

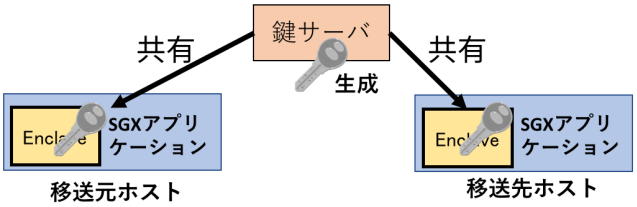

図 9　Enclave 間での鍵の共有

ドレスを指定することで Enclave の配置を固定する．このアドレスは移送元で Enclave に使われていたメモリ領域であり，その Enclave は終了させられているため，移送先でも確実に同じアドレスにメモリを確保することができる．

4.3　MigSGX ランタイムとの通信

MigSGX マネージャは SGX アプリケーションに対して Enclave の状態の保存・復元を指示するために，MigSGX ランタイムと通信を行う必要がある．この通信には様々な方法が考えられるが，アクセス制限を行える方法を選択する必要がある．アクセス制限がなければ攻撃者は容易に Enclave の状態を外部メモリに保存させることができ，暗号化された状態データを取得して解析に利用することができる．また，状態保存後には Enclave が終了させられるため，DoS 攻撃を行うこともできる．

ネットワーク通信を用いると IP アドレスやポート番号より細かいアクセス制限は難しい．公開鍵暗号を用いた認証を行う方法も考えられるが，MigSGX マネージャのデジタル証明書の管理が煩雑になる．Unix ドメインソケットや名前付きパイプを用いると，作成されるファイルに対してパーミッションを設定することでアクセス制限を行うことができる．しかし，SGX アプリケーションを実行したユーザにはアクセスを許可する必要があるため，そのユーザの権限を奪われるとアクセス制限は無力化する．共有メモリやメッセージキューを用いると，セグメント識別子やメッセージキュー識別子さえ分かればどのプロセスでもアクセスすることができる．シグナルを用いると，送信元プロセスのユーザ ID に基づいてアクセス制限を行うことができる．しかし，シグナルではシグナル番号以外の情報を送ることができない．

そこで，MigSGX では CRIU のパラサイト機構 [10] を用いて通信を行う．パラサイト機構はアプリケーションに動的にコードを埋め込んで実行することを可能にする．埋め込まれるコードはパラサイトコードと呼ばれる．まず，MigSGX マネージャは図 8 のように SGX アプリケーションのプロセスを一時停止してパラサイトコードを埋め込み，プロセスを再開してパラサイトコードを実行する．パラサイト機構を利用するには管理者権限が必要となるため，管理者権限が奪われない限りは安全に通信を行うことができる．

しかし，パラサイトコードが直接 ECALL を用いて Enclave の保存・復元を行うのは難しい．パラサイトコードから SGX アプリケーションの中の MigSGX ランタイムの関数を呼び出すのが難しいためである．そこで，MigSGX ランタイムは保存・復元処理を行うスレッドをそれぞれ立ち上げ，メインスレッドとそれぞれのスレッド間で匿名パイプを作成する．匿名パイプはプロセス外部からアクセスできないため安全である．この匿名パイプをパラサイトコードから特定できるようにするために，ファイル記述子に専用の番号を割り当てる．パラサイトコードは匿名パイプを通じて保存・復元処理を行うスレッドを呼び出し，そのスレッドから ECALL を用いて MigSGX ライブラリを呼び出す．

4.4　鍵管理

MigSGX では信頼できる鍵サーバを用意して，Enclave メモリの暗号化・復号化の際に用いる CPU 非依存の鍵を管理する．SGX アプリケーションをマイグレーションする際にはまず，鍵サーバで鍵を生成し，移送元 Enclave とその鍵を共有する．その際に，Enclave のリモートアテステーションを行い，Enclave と鍵を一対一に対応づける．移送先で Enclave が再作成されると，鍵サーバとの間で同じ鍵を共有する．その際にも，Enclave のリモートアテステーションを行い，移送元の Enclave と同一である場合にのみ鍵を送信する．これにより，鍵を不正に取得されて，Enclave メモリを復号されるのを防ぐ．

5.　実験

SGX アプリケーションを保存・復元した後の動作を確認し，保存・復元の性能を調べる実験を行った．マイグレーションを行う際には保存した状態を移送先ホストに転送する必要があるが，今回の実験では同じホスト上で保存・復元

© 2018 Information Processing Society of Japan

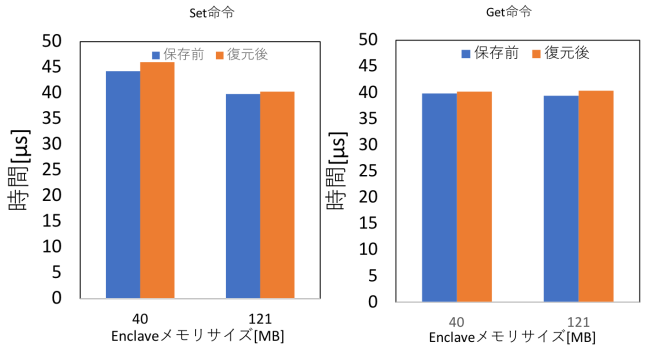

図 10 set 命令の性能

図 11 get 命令の性能

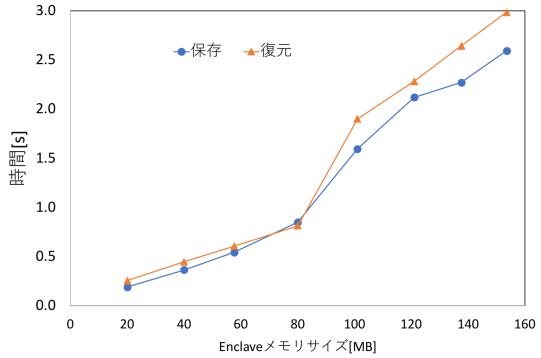

図 12 保存・復元の性能

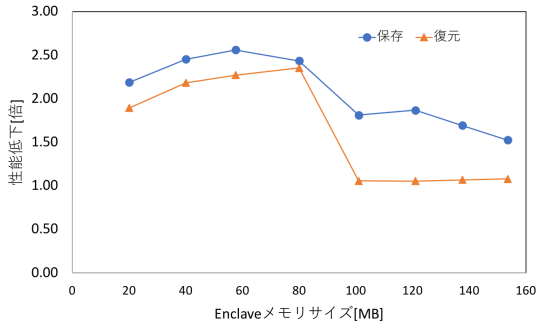

図 13 暗号化オーバーヘッド

のみを行った．SGX アプリケーションとして，Enclave 内でデータを管理するキーバリューストアを作成した．実験には，Intel Xeon E3-1225 v5 の CPU，4GB のメモリを搭載したマシンを用いた．OS には Intel SGX ドライバをインストールした Linux 4.4.0 を用いた．また，MigSGX を実装した Intel SGX SDK 1.9 および CRIU 3.9 を用いた．

5.1 動作確認

まず，作成したキーバリューストアにいくつかのデータを格納し，その後で MigSGX を用いて保存・復元を行った．その結果，状態の復元後でも SGX アプリケーションが正常に動作し，状態の保存前に格納していたデータを取り出すことができた．

次に，状態の保存・復元による性能への影響を調べた．キーバリューストアに set 命令でデータを格納する性能と get 命令でデータを取得する性能を保存と復元の前後で比較した．測定結果を図 10 と図 11 に示す．実験結果より，復元後の性能が 1〜4%低下していることがわかる．この原因は現在調査中である．

5.2 保存・復元にかかる時間

MigSGX マネージャが SGX アプリケーションの状態を保存・復元するのにかかる時間を測定した．この実験では，Enclave に割り当てるメモリサイズを 20〜160MB まで変化させた．測定結果を図 12 に示す．状態の復元のほうが少し時間がかかっているが，保存にかかる時間も復元にかかる時間も Enclave のメモリサイズにほぼ比例して増加した．しかし，100MB 付近で時間が大きく増加した．これは，Enclave のメモリサイズが Enclave ページキャッシュ（EPC）のサイズを超えたためである．実験に用いた CPU における EPC のサイズは約 98MB であった．Enclave が EPC 上にないメモリページにアクセスすると，メインメモリ上に暗号化されて格納されているデータが EPC に読み込まれて復号される．同時に，EPC 上のいずれかのページが暗号化されてメインメモリ上に書き出される．このオーバヘッドのために，Enclave メモリの保存・復元により長い時間がかかるようになった．

5.3 暗号化のオーバヘッド

Enclave の内部で CPU 非依存の鍵を用いて AES 暗号化・復号化を行うオーバーヘッドを調べるために，暗号化を行なわない場合と性能を比較した．暗号化を行った場合の保存・復元の性能低下は図 13 のようになった．Enclave のメモリサイズが 80MB 以下の時の性能低下は保存時で 2.5 倍程度，復元時で 2.2 倍程度であった．一方，100MB 以上の時にはオーバーヘッドが大きく減少し，保存時の性能低下は 1.7 倍程度，復元時で 1.1 倍程度であった．100MB 付近で暗号化のオーバヘッドが減少するのは，EPC のサイズを超えたことにより保存・復元にかかる時間が大幅に増加したためである．復元時に暗号化オーバヘッドが非常に小さくなる理由は現在調査中である．

6. 関連研究

本研究では SGX アプリケーションが動作しているコンテナのマイグレーションを対象としているが，SGX アプリケーションを含む VM のマイグレーションについては先行研究がある．SGX ハードウェアを拡張する手法 [12] では，CPU に新しい命令セットを導入することで Enclave に対して透過的な VM マイグレーションを可能にしている．そのために，移送元ホストと移送先ホストにマイグレーション用 Enclave を作成してリモートアテステーションを行い，マスターシークレットを交換する．それぞれのホスト上でマスターシークレットを CPU に格納すると，SGX はマイグレーション用の鍵を生成する．移送元ホストが Enclave

© 2018 Information Processing Society of Japan

メモリを転送する際には SGX がこの鍵を用いて Enclave メモリを暗号化し直して書き出し，移送先ホストでも同じ鍵を用いて復号したデータを再作成した Enclave のメモリに格納する．この手法は SGX アプリケーション側でのサポートを必要としないが，Intel SGX が対応しない限り利用することができない．

ソフトウェアのみで実現する手法 [13] では，MigSGX と同様に Enclave 自身に状態の保存や復元を行わせる．さらに，Enclave 内で動いているスレッドが停止するのを待ってから状態を保存できるように Two-phase Checkpointing を行う．また，Enclave 内のスレッドによる Enclave 外部の関数呼び出しも追跡して，移送先でその状態を再現する．これらの機構は MigSGX でも用いることができる．マイグレーション時には，ハイパーバイザが VM 内のゲスト OS に割り込みを送り，OS が SGX アプリケーションにシグナルを送る．攻撃者がシグナルを送ることは考慮されておらず，ゲスト OS への修正も必要となる．

Enclave の内部状態だけでなく，外部の永続状態をマイグレーションできるようにする手法 [14] も提案されている．Enclave 外部の永続状態としては，Enclave ごとの専用の鍵によってシール（暗号化）されたデータや単調増加しかしないカウンタなどがある．この手法ではマイグレーション用ライブラリが提供され，Enclave はライブラリの中で生成された鍵を用いてデータの暗号処理を行う．これにより，鍵を移送先に転送してデータの復号を行うことが可能になる．単調カウンタについては，マイグレーション用ライブラリが移送元での値を転送し，移送先で再設定する．これらの手法は MigSGX でも利用可能である．

GPU 上で動作しているプログラム（GPU カーネル）と協調することによって GPGPU アプリケーションをマイグレーションする手法 [15] が提案されている．GPU には CPU 側からアクセスできない状態があり，GPU カーネルを CPU 側から一時停止させることもできない．そこで，協調的マルチタスクを実現する GPGPU アプリケーションのためのフレームワークである GLoop[16] を用いることでマイグレーションを実現している．GLoop を用いて GPU カーネルを一時停止し，GPU のすべての実行コンテキストを CPU から取り出せるようにする．その後，プロセスのマイグレーションを行い，移送先で GPU の状態を復元する．GPU カーネルを一時停止させるソフトウェア機構は MigSGX でも利用できる可能性がある．

7. まとめ

本稿では，コンテナ・マイグレーション後にも SGX アプリケーション内の Enclave を継続的に実行可能にする MigSGX を提案した．Enclave の状態は外部から保存・復元することができないため，MigSGX では Enclave 自身に状態の保存・復元を行わせる．Enclave の状態データを Enclave の外部メモリに安全に保存するために，CPU 非依存の鍵を用いて暗号化を行う．我々は MigSGX を Intel SGX SDK および CRIU に実装した．実験により，MigSGX による保存・復元の後で SGX アプリケーションが正常に動作することを確認できた．また，SGX アプリケーションの保存や復元にかかる時間は Enclave のメモリサイズにほぼ比例するが，100MB 以上で大幅に増加することが分かった．

今後の課題は，Enclave のメモリサイズが大きくなった時に，Enclave メモリを効率よく移送先ホストに転送できるようにすることである．現在の実装では，一旦，MigSGX ランタイムのバッファに Enclave メモリ全体を格納しているため，一時的に Enclave メモリの 2 倍のメモリが必要となる．また，SGX アプリケーションの停止時間を減らすためにライブマイグレーションに対応する．そのためには，Enclave メモリの更新部分だけを保存・転送できるようにする仕組みが必要となる．さらに，Enclave の状態の暗号化・復号化に用いる CPU 非依存の鍵を鍵サーバと Enclave 間で安全に共有する仕組みを実装することも今後の課題である．

参考文献

[1] SELinux Project: SELinux, https://selinuxproject.org/
[2] AppArmor security project: AppArmor, https://gitlab.com/apparmor/apparmor/wikis/home/
[3] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative Technology for CPU based Attestation and Sealing," Workshop on Hardware and Architectural Support for Security and Privacy, 2013.
[4] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using Innovative Instructions to Create Trustworthy Software Solutions," Workshop on Hardware and Architectural Support for Security and Privacy, 2013.
[5] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative Instructions and Software Model for Isolated Execution," Workshop on Hardware and Architectural Support for Security and Privacy, 2013.
[6] Intel Corporation: Intel Software Guard Extensions (Intel SGX) SDK, https://software.intel.com/en-us/sgx-sdk
[7] OpenVZ team: CRIU, https://criu.org/
[8] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," Proc. Network and Distributed Systems Security Symp., pp.191-206, 2003.
[9] Docker, Inc.: Docker , https://www.docker.com/
[10] Open VZ team: CRIU Compel, https://criu.org/Compel
[11] wolfSSL: wolfSSL, https://www.wolfssl.com/
[12] J. Park, S. Park, J. Oh, J. Won, "Toward Live Migration of SGX-Enabled Virtual Machines" Proc. 2016 IEEE World Congress on Services, pp.111-112, 2016.
[13] J. Gu, Z. Hua, Y. Xia, H. Chen, B. Zang, H. Guan, J. Li, "Secure Live Migration of SGX Enclaves on Untrusted Cloud" Proc. 47th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks, pp.225-236, 2017.

© 2018 Information Processing Society of Japan

[14] F. Alder, A. Kurnikov, A. Paverd, N. Asokan, ”Migrating SGX Enclaves with Persistent State” Proc. 2018 48th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks, pp.195-206, 2018.
[15] 湯原 昌, 鈴木 勇介, 河野 健二, ”GPGPU アプリケーションのマイグレーションを可能にするフレームワーク” 2018年並列／分散／協調処理に関する『熊本』サマー・ワークショップ, 2018.
[16] Suzuki, Y., Yamada, H., Kato, S. and Kono, K.: GLoop: An Event-driven Runtime for Consolidating GPGPU Applications, Proc. 2017 Symp. Cloud Computing, pp. 80-93, 2017.

© 2018 Information Processing Society of Japan