Vol.2021-DPS-186 No.28
Vol.2021-CSEC-92 No.28
2021/3/15

Intel SGXを用いたサーバーにおけるチート対策のクライアント委任手法

高橋 孝輔[1]　橋本 正樹[1]

概要：昨今のゲームでは, チート行為によるゲームバランスの崩壊が問題となっており, ゲーム会社はチート対策を欠かすことはできない. ここで, チート対策はゲームユーザーが信頼できない可能性があるためにクライアントだけで完結させることはできず, ゲームサーバーにおいても行う必要がある. さらに, ゲームサーバーにおいては, 以前に比べてチート検知処理を行うプレイヤーの数やDDoS攻撃の増大によって, 負荷が増加していると推測される. この負荷の状態が続けば, ゲームユーザーに適切なゲーム体験を提供できなくなる. そのため, 本研究ではIntel SGXに着目し, ゲームサーバーにおけるチート対策をクライアントに委任することでサーバーリソース負荷の軽減を目的とした委任手法を提案する. 本提案手法を自作したゲームに用いたところ, ゲームサーバーにおけるcpu使用率の軽減を実証できたことを示す.

Client delegation method for cheat countermeasures on server using Intel SGX

Takahashi Kosuke[1]　Hashimoto Masaki[1]

1. はじめに

昨今のIT社会において，ゲーム市場は日に日に拡大し続けている．2020年の世界のゲーム市場は，1,593億ドル（約16兆円）と推定されている[1]．これは，2019年〜2020年における日本の建設業界の市場規模（約16.9兆円）に匹敵する[2]．ゲーム市場が拡大するにつれて，ゲームユーザーやゲーム会社にとってチート行為が非常に問題のある行為になってきている．

チート行為とは，ゲーム上での不正行為の総称であり，コンピュータゲームにおいては，ゲームを有利に進めるために，ゲームのデータやコードを改ざんする行為のことである．チート行為が行われると，ゲームバランスが崩壊するために，ゲームユーザー間に不公平が生じ，さらには，ゲームバランスが崩壊したことによるゲームユーザーの減少によって，ゲーム寿命の早期化や本来得られるはずであった利益をゲーム会社が受けることができない等の問題が発生する．

実際，Irdeto global gaming survey[3]によると，60%のユーザーがチート行為による悪影響を経験し，まったく影響を受けていないユーザーは12%のみであった．そして，76%のユーザーは，ゲームでチート行為が可能でないようにセキュアであることが重要であると答えている．これは，正規ユーザーがフェアな環境でゲームをプレイし，最高のゲーム体験を得たいと考えていることが反映されている．チート行為は，単にゲームユーザーだけが関連しているのではなく，ゲーム会社にとっても非常に問題である．同様の調査によると，チート行為が確認された時に，48%ものユーザーが課金コンテンツの購入を控え，77%のユーザーがゲームのプレイ自体を辞めると答えている．これは，基本プレイが無料で課金コンテンツによって収益を得ているFree-to-play型のゲームにとっては非常に問題となる．これらから，チート対策は，フェアなゲーム体験をクライアントに提供し，ゲーム会社が収益を正当に得るために必要不可欠である．

そのため，ゲーム会社はチート対策を行う必要があるが，その実装に目を向けると，クライアントが信頼できない

[1] 情報セキュリティ大学院大学

ゲームユーザーの可能性があるため，クライアント側ですべてのチート対策を行うことはできず，ゲームサーバー側である程度チート対策を行う必要がある．一方で，昨今のゲーム仕様を鑑みると，以前に比べて 1 つのゲームへの参加プレイヤー数が増大しているため，ゲームサーバー側でのチート対策にかかるリソース負荷は増加していることが推測できるし，また同時に，ゲームサーバーへの DDoS 攻撃が増加していることから，DDoS 対策のためのサーバー負荷も増加している [4]．これらの要因によって，ゲームサーバーが日常的に過負荷になっていることが推測され，ゲームサーバーの過負荷状態が続けば，ゲームユーザーに適切なゲーム体験を提供できなくなる恐れがある．従って，ゲームサーバーにおける負荷軽減は重要な課題であるが，ここで，DDoS 対策は，原則としてゲームサーバー側でしか行うことはできないためゲームサーバーの負荷の軽減には活用できない．そのため，本研究では，プログラムの完全性と機密性を保護できる Intel SGX に着目し，ゲームサーバーにおけるチート対策をクライアントに委任することでゲームサーバーの負荷軽減を目指す手法を提案する．

1.1 研究の特徴と成果

本研究の特徴は，ゲームサーバーにおけるチート対策のクライアント実行を可能にする委任手法である．これを実現するために，送受信に関連するコンポーネントと実際にチート検知・対策を行うコンポーネントを考案した．それら 3 つのコンポーネントを用いることで，ゲームサーバーのチート対策をゲームクライアントに委任することが可能となり，サーバーリソース負荷を軽減することができる．しかしながら，ゲームクライアントに委任をした場合においてもゲームの空き時間等におけるチート検知処理の実行が可能なために，ゲームのメインロジックに悪影響を与えない．

本研究の成果は以下の通りである．

まず，既存にはない，チート対策の新しい委任手法の基本構成案を提案した．今までゲームに Intel SGX を用いる先行研究は，ゲームクライアントで行われてきた現状のチート対策を拡張・強化するための研究が多い．しかしながら，昨今のゲームサーバーに発生している問題を対処するために Intel SGX を利用した先行研究は非常に少ない．特に，昨今のゲームサーバーにおいては過負荷状態が推測され，その負荷を軽減できなければゲームユーザーに適切なゲーム体験を提供できなくなる恐れがある．そのため，本研究ではゲームサーバーの負荷軽減を目的とするクライアント委任のための基本構成を考案する．

次に，提案手法がサーバーリソース負荷を軽減できることを評価実験によって実証した．本研究では，提案手法の有効性を評価するために，Unity と呼ばれるゲームエンジンを用いて実験用にシングルプレイゲームを作成した．作成したゲームに対して本提案手法を適用し，評価実験を行った結果，ゲームサーバーの cpu 使用率が軽減されることが実証され，目的であったサーバーリソース負荷の軽減を達成できたことがわかった．提案手法を用いることで，サーバーリソースの省力化によるラグの解消や同時接続台数の増加などのメリットが得られる．また，ゲームサーバーの負荷軽減は，直接的な DDoS 攻撃への対策とはならないが，ゲームサーバーの DDoS 攻撃耐性を高めることができるものと期待できる．

1.2 本稿の構成

以降本稿では，第 2 節に本研究で用いる Intel SGX の概要と利用する機能，第 3 節では Intel SGX を実際にゲームに用いた関連研究について説明する．そして，第 4 節では本研究の提案手法，第 5 節では提案手法の実装と評価手法，第 6 節では提案手法の評価を行う．最後に，本稿をまとめる．

2. Intel SGX

Intel SGX[5] は，2015 年 8 月に製品化された第 6 世代 Intel Core プロセッサ（スカイレーク）の拡張機能であり，メモリ上に Enclave と呼ばれる暗号で保護された領域を生成し，センシティブデータの完全性，機密性を保ちながらのプログラム実行を可能にする．Intel SGX は強い脅威モデル [6] を考慮しており，intel cpu 以外のユーザーアプリケーション，OS，ハイパーバイザー等を信頼しない．また，エンクレーブプログラムの実行中はリング 3 のユーザーモードで動いているため，一般的な OS がサービスとして提供する機能を使用できない．そのため，エンクレーブで処理したデータをリモートパーティに送信する場合や HDD に保存する際は，信頼できない OS が介在することになる．Intel SGX は，これらの懸念に対する解決策としてシーリングやリモートアテステーションの機能を提供している．

- シーリング
  エンクレーブは，メインメモリ上に存在するため，PC の電源が切れると，エンクレーブ上に存在するセンシティブデータは消えてしまう．センシティブデータを電源プロセスによらず使用するために，Intel SGX は，センシティブデータを暗号化するシーリングと呼ばれる機能を提供する．シーリングでは，cpu パッケージ内にあるプラットフォーム特有の秘密情報とエンクレーブ自身の環境情報等を用いてシーリング鍵が生成されるため，異なるエンクレーブや別のプラットフォーム上でアンシーリング（復号化）することは不可能である．
- リモートアテステーション
  リモートアテステーションとは，遠隔地の Intel SGX

上で動いているエンクレーブが検証者にとって意図したものであるかを検証する認証プロトコルである．正しく検証ができた際は，エンクレーブとサーバー間で暗号化通信を行うための秘密鍵を生成する．

3. 関連研究

現時点の Intel SGX 研究は，特定の機能を提供するアプリケーションに特化した研究と汎用的な分離環境を構築する研究 [7] が主流である．ゲームのチート対策の研究は主に前者に相当しており，様々な研究が行われている．その中でも，クライアントでのチート対策に Intel SGX を用いた研究が多い [8][9][10][11]．また，ゲームソフトウェアには，難読化と呼ばれるコードの解析を妨害する技術が用いられるが，その難読化手法の一つである packing の研究も存在する [12]．本研究に最も近い関連研究としては TrustJS[13] が相当している．以下に概要を記す．

Bauman ら [8] は，Intel SGX がどのようにゲームを保護できるかを検討し，ゲームに組み込むための完全性，機密性のフレームワークの提案をしている．完全性のフレームワークは，ゲームステータスをエンクレーブ上に置くことで原理的にメインメモリ内にあるデータの改ざんを防ぐ手法である．また，機密性のフレームワークは，ゲームの配布前に音楽などのゲームコンテンツを暗号化しておき，ゲームの開始時に正しいライセンスキーの所有者であれば Intel SGX を介してゲームコンテンツを復号化する手法の提案である．Bauman らは，機密性フレームワークのみを実際の Biniax2 と呼ばれるオープンソースパズルゲームに適用し，コピー防止機能についての評価をした．その結果，ゲームの初期化時間にのみオーバーヘッドの増加が発生したが，人間が感知するにはあまりにも早く問題にはならないと主張している．

Brandão ら [9] は，クライアントで実行中であるゲームプログラムコードやデータの改ざんを検出する Intel SGX を用いたメカニズムを提案している．その提案手法は，ゲームプログラムを変更する必要がなくデプロイ可能であり，CS:GO に適用した実験によるとゲームのパフォーマンスに大きな影響を与えないと結論付けられている．

畑ら [10] は，Bauman らの完全性フレームワークに似た P2P ゲームのメインロジックを Intel SGX 上で実装し評価を行っている．プロトタイプと ZDoom のゲームにおいて実験を行っており，マルチプレイゲームのパフォーマンスに影響を与えない結果が得られたと報告している．

Lazard ら [11] は，TEE のメリットを簡単に享受できるようにバイナリプログラム内の指定した関数を Intel SGX を含む TEE 上に移行するための TEEshift と呼ばれるツールを開発した．TEEshift では，TEE プラットフォームに依存しないことを目的としている Google asylo フレームワークを用いて開発され，利便性のためにソースコードや環境情報等を必要とせずに TEE の活用を可能にする．Lazard らは，TEEshift のパフォーマンスを検証するために，実際の 2 D シューティングゲームである Teeworlds の一部の関数を Intel SGX 上に移行させた．その結果，目立ったパフォーマンスへの影響やゲーム機能を阻害することはなかったと報告されている．

Tychalas ら [12] は，一般的な packing が用いられた実行ファイルをリバースエンジニアリングする際に，unpacking コードが狙われやすい点に着目し，Intel SGX 上で unpacking コードを実行することにより静的解析と動的解析を困難にする SGXCrypter を開発した．SGXCrypter のオーバーヘッドについては，実行ファイルのサイズが 1MB 未満であれば，ユーザーが認識できないほどである．

Intel SGX 上では単純にシステムコールを使用することはできないため，レガシーアプリケーションで Intel SGX を利用するためにはソースコードの修正が必要となる．Intel SGX の利便性の向上のために，Tsai ら [7] は，レガシーアプリケーションの修正なしに Intel SGX を利用可能な環境である Graphene-SGX を開発した．Graphene-SGX は，Graphene と呼ばれるライブラリ OS や Untrusted 領域との仲介層をレガシーアプリケーションとの間に挟むことで実装されている．そのため，一般的な OS の機能であるマルチプロセス，マルチスレッド，例外処理の利用が可能である．Tsai らは，Graphene-SGX の評価としてサーバー (lighttpd, apache, nginx) やコマンドライン (gcc, r, curl) アプリケーションのオーバーヘッドと Graphene-SGX の導入による TCB 量を調査している．その結果，実用的なオーバーヘッドと TCB サイズに抑えられると主張されている．

Goltzsche ら [13] は，クライアントにおける JavaScript コードの実行結果が信頼できないことに着目し，Intel SGX を用いることでクライアントでの JavaScript コードの機密性や完全性を保ちながら実行することを可能にする TrustJS を開発した．TrustJS は Javascript インタプリタを実装したアドオンとして提供され，クライアントは，アドオンの形でブラウザに導入する．Goltzsche らは，TrustJS を評価するためにユーザーに出力を返す前に入力検証を行う必要のあるマルチステップフォーム WEB アプリケーションを作成し，クライアントとサーバ間のレイテンシと WEB サーバのスケーラビリティを調査した．その結果，TrustJS を用いることで WEB サーバーの cpu 使用率や総処理時間の軽減が得られたと報告している．

4. 提案手法

本研究は，Intel SGX の導入によるチート検知のクライアント委任手法構成の提案である．具体的には，図 1 のクライアントサーバー構成をベースとして，ゲームサーバーにおけるチート検知処理をクライアントで行うことでゲー

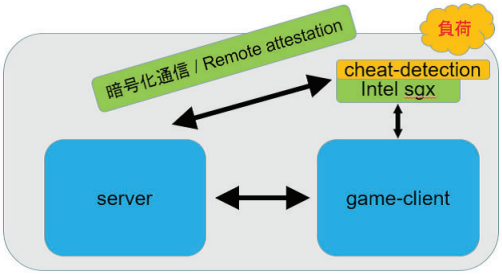

図 1 提案手法の概要
Fig. 1 Outline of the proposed method

ムサーバーの負荷を改善することを目的としている．ここで，クライアントで実行するタイミングが非常に重要になるが，本来サーバーで行われていたためにゲームのメインロジックとは無関係にチート対策の実行が可能である．そのため，チート検知処理をゲームの隙間時間等に実行することが可能になり，メインロジックに悪影響を与えずにサーバーリソース負荷の軽減が可能になる．

関連研究で紹介したように，Intel SGX を用いるゲームのチート対策は，クライアントに着目した研究が主流である．それらのすべてが，ゲームクライアントにおけるメインロジックの改ざんを防止するための研究やゲームクライアントに DRM(Digital Rights Management) を導入する研究に分類でき，ゲームクライアントに着目した研究が非常に多い．そのため，本研究では，ゲームクライアントに着目するのではなく，ゲームサーバーにおける昨今の問題を取り扱う点で新規性がある．著者らの知る限りでは，ゲームサーバーと Intel SGX の両方に着眼点を置いている研究は存在しない．

本提案手法の前提条件として，PC ゲームであること，クライアント PC に Intel SGX が搭載されていること，ゲームサーバーとエンクレーブプログラム間でリモートアテステーションが既に行われており秘密鍵を共有しているものとする．本提案手法を実現するためには，クライアントに Intel SGX が搭載されていることが非常に重要であるが，ゲーム業界に対してこの問題は注目に値しないと考えている．実際，Steam の調査 [14] によると 81%のゲームユーザーが Intel cpu を使用している．さらに，一般的な PC のゲームユーザーは非常に高いスペックの PC を使用しているために，Intel SGX が機能する高性能 cpu が搭載されている可能性が高い．

Intel SGX が想定している脅威モデルは，2 節で記載したように強い脅威モデルであり，Intel SGX 独特の攻撃手法が多数発表されている [15][16][17]．しかし，攻撃手法に対する改善案も多数発表されているため，これらの攻撃手法は本提案手法の対象外とする．つまり，Intel SGX 自体の完全性が保証されており，実装するエンクレーブプログラムに脆弱性が含まれていないとする．

4.1 設計方針

本研究では，提案手法の対象を一般的なアプリケーションではなくクライアントサーバー型のゲームに絞っている．適用対象を限定した理由として，ゲームにおける冗長性と本提案手法の導入の容易さを考慮したためである．

まず，一般的なアプリケーションと比べると，ゲームシステムはサービスを提供する以上の冗長なリソースが必要になる．具体的に，オンラインゲームの場合，ユーザーのゲームプレイにおいてラグが発生しないように高速な通信回線を使用し，急激なトラフィックの増加が発生した場合においてもゲームサービスの品質を保つために複数のゲームサーバーを用いて冗長性を持たせる必要がある．しかし，一般的なアプリケーションの場合，ある程度のラグは許容されるためにゲームシステム程の冗長性は必要なく，サーバーリソース負荷の軽減を目的とした本提案手法を用いるメリットは少ないと考えた．

次に，本研究では任意の時間にクライアントで実行することによって委任手法を達成するが，一般的なアプリケーションと比較してゲームシステムの方が提案手法の導入が容易である点を挙げる．なぜならば，ゲームシステムに本提案手法を適用する場合は，ゲームイベントの終了後に発生するロード時間に導入すれば良いが，一般的なアプリケーションの場合，ゲームシステムのように実行場所が明確ではないため導入が難しいからである．

以上から，本研究では，提案手法を用いる際に導入が容易であり，サーバーリソース軽減に関してメリットが大きいと思われるゲームシステムを対象としている．

3 節で紹介した関連研究から，ゲームのチート対策に Intel SGX を用いる場合は，アプリケーション専用のエンクレーブを開発することが一般的である．なぜならば，汎用的な分離環境を用いると TCB 量が増大し脆弱性等が入り込む可能性が高くなり，一定の処理速度が要求されるゲームにとっては余分な機能があまりにも多くオーバーヘッドの観点からも相応しくない．また，現状では対応プラットフォームが Linux のみであるために一部のプラットフォームにしか対応できない恐れもある．そのため，本研究では，アプリケーション専用のエンクレーブを開発し，提案手法を実証した．

4.2 提案手法の構成

本提案手法は 3 つのコンポーネントからなる．それらをそれぞれ ECALL_SEND, ECALL_RECE, ECALL_DETE と呼ぶ．3 つのコンポーネントを設計する際にサーバークライアント方式を用いているゲーム全体で使用可能なように，汎用性を持たせている．特に，サーバークライアント方式では，データの送受信が必須になるためにそれぞれ専用のコンポーネントを設けた．また，ECALL_DETE コンポーネントは最後に実行する必要があり，それ以前に実行

される ECALL_SEND と ECALL_RECE コンポーネントの使用回数や順番を柔軟に構成できるようにコンポーネントの依存関係を最小限に設計している．以下に 3 つのコンポーネントを説明する．

- ECALL_SEND
  クライアントからゲームサーバーへのデータ送信時にチート検知準備処理を行うコンポーネントである．このコンポーネントでは，まずゲームサーバーへの送信内容が適切な値かどうかについての確認を行う．この検証は，本来ゲームサーバー側で行われる処理であるが，Intel SGX を用いることでエンクレーブプログラムの完全性と機密性が保証されるため，クライアント上での処理が可能となる．その後，ゲームサーバーと共有している秘密鍵を用いて送信内容を暗号化し，ECALL_DETE コンポーネントで後に送信データを用いるためのシーリングを行う．
- ECALL_RECE
  ゲームサーバーからクライアントへのデータ受信時にチート検知準備処理を行うコンポーネントである．このコンポーネントでは，ゲームサーバーと共有している秘密鍵を用いてゲームサーバーからのデータを復号化し，後でチート検知に用いるためのゲームサーバーからの情報をシーリングする．このゲームサーバーからの情報は，チート検知処理に依存したデータを想定しているが，本提案手法を用いる開発者の実装依存である．
- ECALL_DETE
  上記のコンポーネントで生成したシーリングを用いて実際のチート検知処理を行うコンポーネントである．最初に，アンシーリングを行い，チート検知用のデータを取得する．その後，そのデータをもとに実際のチート検知処理を行い，その結果をゲームサーバーと共有している秘密鍵を用いて暗号化する．本提案手法では，チート検知処理を任意時間に実行可能ではあるが，チート行為を野放しにしておくことは 1 節で説明したようにゲーム会社とゲームユーザー共に非常にリスクがある．そのため，ECALL_DETE コンポーネントは可能な限り早くに実行される必要がある．このコンポーネントで行われるチート検知処理は，本提案手法を用いる開発者の実装依存である．

本提案手法ではシーリングを用いているが，エンクレーブ上に割り当てられたヒープ領域を使用することで HDD ではなくメインメモリ上に保存することも可能である．しかし，電源プロセスやゲームクライアントの終了によってエンクレーブ上のデータが消えてしまう可能性がある．これは，特に ECALL_DETE コンポーネントが使用される前に起きるとチート検知処理が行えなくなり非常に問題である．そのため，HDD にシーリングデータを保存しておくことで，予期せぬゲームクライアントの終了等による保護の欠如が発生しないようにしている．

（1）ゲームプログラムの実行
（2）ゲームサーバーへゲーム状態の送信 (ECALL_SEND)
（3）ゲームプログラムの実行
（4）ゲームサーバーから任意情報の取得 (ECALL_RECE)
...
（n）任意の時間にチート検知処理 (ECALL_DETE)

図 2 提案手法コンポーネント使用具体例
Fig. 2 Example of using components of the proposed method

次に，本提案手法を構成するコンポーネントの具体的な使用例を挙げる．上記で説明したコンポーネントすべてを使用するかどうかは実装依存であるが，具体例のためにすべてを使用する．図 2 は，クライアントからゲームサーバーへの送信データとゲームサーバーからの受信データを使用して，任意の時間にチート検知処理を行う具体例である．

5. 実装と評価手法

本研究における提案手法の評価のために，ボット検知処理に関する実験を行った．具体的には，ボット検知処理をゲームサーバーで行う従来手法と Intel SGX を利用してクライアントで行う提案手法で比較実験を行っている．以降の節では，実験のために作成した提案手法と従来手法の実装と評価手法について説明する．

5.1 エンクレーブプログラムの実装

本提案手法の実装のために，Intel が提供しているリモートアテステーション [18] とシーリング [19] のサンプルコードを使用している．4 節で記載した前提条件より，リモートアテステーションはゲーム起動後の初期化時に行うことを想定しているため，3 つのコンポーネントを呼び出す前に実行される．リモートアテステーションを行うことによりゲームクライアントで動作しているエンクレーブの整合性を保証し，同時にゲームサーバーとエンクレーブ間での秘密鍵を共有することでゲームクライアントに処理結果等の情報漏洩を防止することができる．さらに，本提案手法ではシーリングを使用しているが，シーリングは完全性と機密性のみを保証するためにリプレイアタックが可能である．そのため，Intel SGX SDK Developer Reference[20] で推奨されている Trusted Time 手法をリプレイアタック対策として用いた．Trusted Time 手法は，オーバーヘッドが高く通信時間に影響を与える可能性があるが，Intel SGX 開発者にとって一般的な問題であるため本研究の対象外とする．

5.2 本提案手法と従来手法における共通事項の実装

図 3 が Unity[21] と呼ばれるゲームエンジンを用いて作成したゲームクライアントである．これは，プレイヤー（四角）が動き，敵（ひし形）を攻撃することによって倒していくシングルプレイアクションゲームである．プレイヤー操作にはすべてキーボードキーを対応させ，矢印キーで移動，スペースキーで攻撃動作を行う．敵に近づくことで自動で敵の方向を向き，スペースキーで攻撃を行うことができる．ゲーム実行中は，1 バイトに対応付けたプレイヤーの座標を一定間隔ごとに取得し，そのデータを用いてボット検知処理を行う．提案手法と従来手法における実験中での差異をなくすために，クライアントとゲームサーバーともにリモートアテステーションのネットワーク通信関数を利用して実装を行った．

ボット検知手法の実装は，Mitterhofer ら [22] のボット対策の考えに基づいて実装を行った．Mitterhofer らは，ゲームボットがプログラムであり，同じ挙動を繰り返す点に注目してボット検知技術を提案している．本研究は，チート対策手法の提案が目的ではないため，ある程度の精度で検知のできる簡易ボット検知処理を実装した．具体的には，座標ログが与えられた際に繰り返される部分バイト列の数をカウントし，閾値より大きければゲームボットと判断する．実装したボット検知処理の計算量は，ログを入力として $\mathcal{O}(n^3)$ 程度で抑えられる．

5.3 本提案手法と従来手法における実装の違い

作成したゲームは C#で記述する Unity を用いている．ゲームプログラムと本提案手法の統合時に，記述言語が C/C++であるエンクレーブプログラムは，単純にマネージドコード上でエンクレーブプログラムを動かすことはできない．そのため，エンクレーブプログラムを C/C++で作成した中間層 DLL から呼び出すように実装を行い，ゲームプログラムではその中間層 DLL を使用することで統合を行った．

本提案手法と従来手法におけるゲームサーバーの処理は全く異なる．本提案手法の場合は，ボット検知処理をクライアントのエンクレーブ上で行うため，ゲームサーバーでは処理結果を復号化することのみを行い，クライアントに処理結果に応じたレスポンスを返す．従来手法の場合は，ゲームサーバー側でボット検知処理を行い，クライアントに処理結果に応じたレスポンスを返す（図 4）．

5.4 評価手法

本提案手法の評価には，ゲームサーバーとクライアント間の通信時間とゲームサーバーの cpu 使用率を用いる．本研究では，通信時間をクライアントでボット検知処理を開始してから，ゲームサーバーからのレスポンスが戻るまでの時間と定義する．具体的に，本提案手法においてはエンクレーブ上でチート検知処理行った後にゲームサーバーにリクエストを送信し，ゲームサーバーからのレスポンスが戻るまでの時間が相当する．そして，従来手法においてはゲームサーバーにログを送信した後にボット検知処理を行い，ゲームサーバーからのレスポンスが戻るまでの時間が相当する．

図 3 ゲームクライアント
Fig. 3 Game client

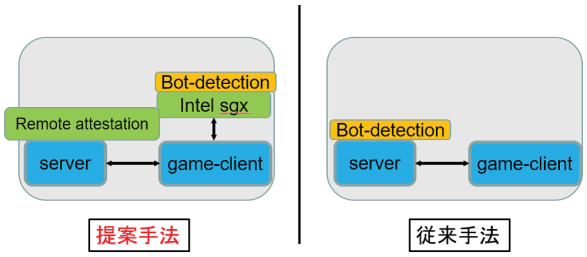

図 4 本提案手法と従来手法の違い
Fig. 4 Difference between the proposed method and the conventional method

作成したゲームクライアントとゲームサーバーを 1 台の PC 環境内（ローカル）で実行し，実験を行った．使用した PC のスペックは，Intel core i5-8250u(1.60GHz / 3.40GHz)，メモリ 8GB，Windows 10，使用した Intel SGX SDK はバージョン 2.5.101.3 である．

実際にゲームボットを動かして本提案手法と従来手法で計測を行うことが望ましいが，ゲームサーバーにおけるボット検知処理に関する cpu 使用率のモニタリングに余分な情報が多く含まれるため，あらかじめログを準備して実験を行っている．つまり，提案手法に関する計測では，ECALL_DETE コンポーネントのみがサーバー cpu 使用率と通信時間に関連する．

ボット検知処理に渡す入力として，ログを 100 バイトから 1900 バイトまで 100 バイトずつ増加させながら実験を行った．ボット検知処理にかかる通信時間については，それぞれのログに対して 10 回計測を行い，平均を導出している．

ゲームサーバーにおける cpu 使用率は，実験の開始から終わりまでを連続して計測している．計測には，Windows 標準ツールのパフォーマンスモニタを用いた．GUI であ

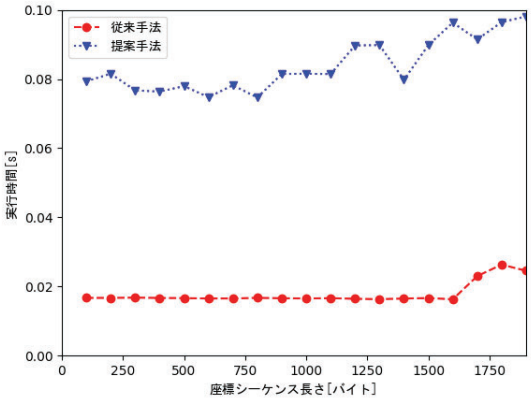

図 5 通信時間
Fig. 5 Communication time

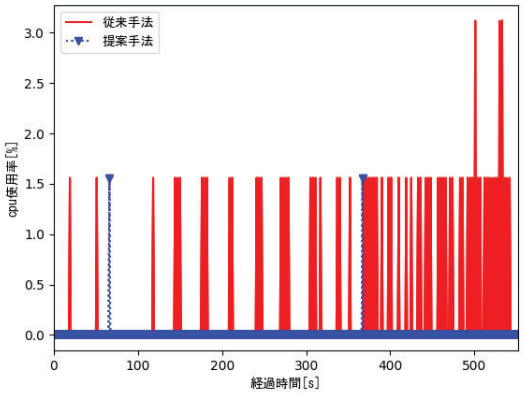

図 6 サーバー cpu 使用率
Fig. 6 Server cpu usage rate

るパフォーマンスモニタは計測において利便性に欠けるため，PowerShell の Get-Counter コマンドレットを使用して cpu 使用率を計測している．単純に Unity で作成したゲームで使用することはできないため，計測用 PowerShell スクリプトを別プロセスとして起動する DLL を作成して統合した．

6. 評価

6.1 通信時間

ボット検知処理を含む通信時間に関しての実験結果（図 5）は，提案手法が従来手法に比べて 60ms 程度のオーバーヘッドを増加させる結果になった．この主な原因は，エンクレーブ上での実行によるものではなく，5.1 節で説明した Trusted Time 利用によるオーバーヘッドと考えられる．しかし，本研究で実験したボット検知処理については，ゲームのリアルタイム性を失わないと結論付ける．なぜならば，ボット検知処理自体はゲームの主なロジックではなく，ゲーム会社側での任意の時間に実行が可能だからである．

6.2 サーバー cpu 使用率

ゲームサーバーの cpu 使用率は，本提案手法と従来手法で多くがそれぞれ 0%，1.5%の使用率であった（図 6）．また，図 6 の後半部分に近づくにつれてログの入力が大きくなるために多くの計算リソースを必要とするが，本提案手法の場合では 0%の cpu 使用率で抑えられていることがわかる．このことから，本提案手法によって，ゲームサーバーの cpu 使用率を軽減できることが実証された．この軽減理由は，従来手法ではボット検知処理を行っているのに対して，本提案手法では暗号文の復号化処理を行うだけで済むためだからである．加えて，このことは，本提案手法をマルチプレイゲームについて用いた場合も同様の結果が得られることを示している．補足として，今回の実験では，ゲームサーバーの処理内容がボット検知処理関連しか行っていないため，cpu 使用率の最大値が 3 %程度になっている．

6.3 送受信コンポーネントのオーバーヘッド

この節では，ECALL_SEND と ECALL_RECE コンポーネントを用いる際に発生するオーバーヘッドについて評価する．これらのコンポーネントは，5.1 節で説明したリプレイアタック対策のために Trusted Time を使用している．6.1 節の評価と同様に，送受信コンポーネントにおいてもリプレイアタック対策がオーバーヘッドを増加させる主要因になっている．実際，データの送受信時に本提案手法を用いた場合，著者らの環境では 1 パケットあたり処理時間が 17 倍に増加した．

本提案手法を低い RTT が望まれるゲームに用いることは適さないが，戦略ゲーム等の高い RTT が許容されるゲームであれば十分に実用可能である．また，本提案手法をすべてのゲームサーバーとの送受信時に用いるのではなく，チート検知処理に用いるデータのみに絞ることでゲーム全体への影響を減らすことができると考えている．もしくは，シーリングのリプレイアタック対策を取り扱っている既存の研究を用いることで本提案手法は幅広いゲームに適用することが可能である．（ROTE[23] 等）

7. まとめ

本研究では，Intel SGX を使用してゲームサーバーにおけるチート対策をクライアントに委任する提案手法の有効性について検証した．本提案手法の評価として，ゲームサーバーとクライアント間における通信時間，ゲームサーバーの cpu 使用率を計測した．その結果，ゲームサーバーの cpu 使用率を軽減できることが実証されたが，送受信時や通信時間に大きなオーバーヘッドが発生することが判明した．しかし，このオーバーヘッドはリプレイアタック対

策によるもので，Intel SGX 開発者にとっては一般的な問題であり，より実用的な先行研究が提案されているため，本研究の適用を狭めるものではない．

参考文献

[1] : 2020 Video Game Industry Statistics, Trends & Data, https://www.wepc.com/news/video-game-statistics/.
[2] : 建設業界の動向や現状，ランキング等を研究 (2020年版)-業界動向サーチ，https://gyokai-search.com/3-kensetu.htm.
[3] : Irdeto Global Gaming Survey — Irdeto, https://resources.irdeto.com/irdeto-global-gaming-survey.
[4] : インターネットの現状／セキュリティレポート — ゲーム業界 —セキュリティも一人ではプレイできません（第6巻，第2号） — Akamai，https://www.akamai.com/jp/ja/multimedia/documents/state-of-the-internet/soti-security-gaming-you-cant-solo-security-report-2020.pdf.
[5] : インテル ソフトウェア・ガード・エクステンションズ (インテル SGX)，https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/software-guard-extensions.html.
[6] Costan, V. and Devadas, S.: Intel SGX Explained., *IACR Cryptol. ePrint Arch.*, Vol. 2016, No. 86, pp. 1–118 (2016).
[7] Tsai, C.-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)*, pp. 645–658 (2017).
[8] Bauman, E. and Lin, Z.: A case for protecting computer games with sgx, *Proceedings of the 1st Workshop on System Software for Trusted Execution*, pp. 1–6 (2016).
[9] Brandão, A., Resende, J. S. and Martins, R.: Employment of Secure Enclaves in Cheat Detection Hardening, *International Conference on Trust and Privacy in Digital Business*, Springer, pp. 48–62 (2020).
[10] 畑輝史，河野健二ほか：TEE によるスケーラブルかつチート耐性のあるオンラインゲームアーキテクチャ，研究報告システムソフトウェアとオペレーティング・システム (OS)， Vol. 2020, No. 7, pp. 1–8 (2020).
[11] Lazard, T., Götzfried, J., Müller, T., Santinelli, G. and Lefebvre, V.: TEEshift: protecting code confidentiality by selectively shifting functions into tees, *Proceedings of the 3rd Workshop on System Software for Trusted Execution*, pp. 14–19 (2018).
[12] Tychalas, D., Tsoutsos, N. G. and Maniatakos, M.: SGXCrypter: IP protection for portable executables using Intel's SGX technology, *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, pp. 354–359 (2017).
[13] Goltzsche, D., Wulf, C., Muthukumaran, D., Rieck, K., Pietzuch, P. and Kapitza, R.: Trustjs: Trusted client-side execution of javascript, *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6 (2017).
[14] : Steam ハードウェア & ソフトウェア調査，https://store.steampowered.com/hwsurvey/.
[15] Moghimi, A., Irazoqui, G. and Eisenbarth, T.: Cachezoom: How SGX amplifies the power of cache attacks, *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, pp. 69–90 (2017).
[16] Xu, Y., Cui, W. and Peinado, M.: Controlled-channel attacks: Deterministic side channels for untrusted operating systems, *2015 IEEE Symposium on Security and Privacy*, IEEE, pp. 640–656 (2015).
[17] Lee, S., Shih, M.-W., Gera, P., Kim, T., Kim, H. and Peinado, M.: Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing, *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 557–574 (2017).
[18] : GitHub - intel/sgx-ra-sample, https://github.com/intel/sgx-ra-sample.
[19] : Intel SGX SDK for Windows v2.5.101.3 Sample Code, https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions/sdk.html.
[20] : Intel SGX SDK Developer Reference for Windows, https://software.intel.com/content/www/us/en/develop/download/sgx-sdk-developer-reference-windows.html.
[21] : Unity Real-Time Development Platform, https://unity.com/.
[22] Mitterhofer, S., Kruegel, C., Kirda, E. and Platzer, C.: Server-side bot detection in massively multiplayer online games, *IEEE Security & Privacy*, Vol. 7, No. 3, pp. 29–36 (2009).
[23] Matetic, S., Ahmed, M., Kostiainen, K., Dhar, A., Sommer, D., Gervais, A., Juels, A. and Capkun, S.: {ROTE}: Rollback protection for trusted execution, *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1289–1306 (2017).