木構造によるメモリ難読化にメモリ暗号化を併用したリアルタイムゲームにおけるチート対策手法の検討

中山 結莉[1,*]　永末 寛人[1]　井上 博之[2]

概要：PC ゲームにおいてメモリスキャン型チートツールによるプレイヤー座標の不正取得を防止するため，チート行為の妨害を目的とした実装可能な防御技術として既存のセキュリティ対策を補完するものとして，座標情報の格納方式として既存のメモリ難読化に加えてメモリ暗号化を組み合わせる手法を提案する．Unreal Engine を用いた TPS 型ゲームにおいて，プレイヤー座標の保存形式を木構造化し，複数のダミーノードを介したポインタチェーンによって経路を隠蔽することで，特定のメモリアドレスへの直接アクセスを困難にする．加えて，FVector 型で格納される実座標データに対して，XOR，Ascon-128，ChaCha20 による暗号化処理を導入し，ゲームのセッションを想定した短時間ではデータを読解できない構成とした．各手法を用いたプロトタイプを実装し，処理速度や暗号強度，実装容易性を定量的および定性的の両面から評価した結果，用途に応じた手法選定が有効であることが分かった．

キーワード：チート対策，メモリスキャン，構造難読化，Unreal Engine，FVector，ポインタチェーン

An Anti-Cheat Method for Real-Time Games Using the Tree-Based Memory Obfuscation and an Encryption

Yuiri Nakayama[1,*]　Hiroto Nagasue[1]　Hiroyuki Inoue[2]

Abstract: In order to prevent the unauthorized acquisition of player coordinates by memory scan-type cheat tools in PC games, we propose a method that combines the existing memory obfuscation and a memory encryption as a means of storing coordinate information. This is an implementable defensive technology aimed at hindering cheating behavior, complementing existing security measures. In a TPS-type game using the Unreal Engine, the storage format for player coordinates is structured as a tree, and the path is concealed through a pointer chain via multiple dummy nodes, making direct access to specific memory addresses difficult. Additionally, encryption processing using XOR, Ascon-128, and ChaCha20 is introduced for the actual coordinate data stored in FVector format, making it impossible to decipher the data within the short timeframe of a game session. We implemented prototypes using each method and evaluated them quantitatively and qualitatively in terms of processing speed, encryption strength, and ease of implementation. The results showed that selecting the appropriate method based on the application is effective.

Keywords: anti-cheat, memory scanning, structural obfuscation, Unreal Engine, FVector, pointer chain

1. はじめに

近年，オンラインゲームの普及とともにゲームプレイを不正に有利に進めるチート行為が深刻な問題となっている．中でもゲーム内の視界を不正に拡張する「ウォールハック（Wallhack, WH）」は，プレイヤーの位置座標や敵キャラクターの位置情報を外部から直接読み取ることで，通常では見えない敵を視認可能とする代表的な手法である．このようなチートは，競技性を損なうだけでなくゲーム運営者にとっても信頼性の低下や収益への悪影響といった深刻な課題をもたらす[1].

WH に対する防御手法として，座標情報などの重要データの格納先を動的に変化させるアドレスランダム化[2]や，データ構造の難読化，暗号化による保護などがこれまでに提案されてきた．例えば，Lars Van Caute による研究では，座標情報をメモリ上の木構造に分割し，各ノードの位置を動的に入れ替えることで，アドレスの予測を困難にするメモリ難読化技術が提案されている[3]．この手法は一定の性能を保ちながら WH などの成功率を低下させる有効なアプローチであるが，暗号化処理には簡易的な XOR 方式を採用しており，比較的短時間で攻撃者に復号されてしまうリスクを残している．本研究では，この Van Cauter の方式を基に，より高強度な軽量暗号である ChaCha20[4]や Ascon-128[5]を暗号方式に採用することで座標情報の復号耐性の向上を目指し、既存の木構造による構造難読化と組み合わせた統合的な防御手法を提案する．また，実装における設計指針や暗号方式ごとの性能比較を示し，プロトタイプによる簡易な評価を通じて，実装時の負荷や防御効果について検討する．

本研究の成果は，オンラインゲームにおける WH 対策の

1 京都産業大学大学院先端情報学研究科
Division of Frontier Informatics,
Kyoto Sangyo University Graduate School
2 京都産業大学情報理工学部
Faculty of Information Science and Engineering,
Kyoto Sangyo University
* i2586164@cc.kyoto-su.ac.jp

実用性を向上させるとともに，今後のゲームセキュリティ技術の設計指針の一助となることを期待している．

2. 関連研究

2.1 WH とその原理

WH とは，ゲーム内で描画対象外となっている敵プレイヤーや物体の座標を直接取得し，不正に視覚化するチート手法である．これは多くの場合，ゲームプロセスのメモリ空間にアクセスし，プレイヤー座標やカメラ情報などを抽出することで実現される[6]. Unreal Engine を使った TPS 型ゲーム（三人称視点のシューティングゲーム）では，プレイヤー座標データが FVector 型と呼ばれる構造体で管理されている．図 1 に示すように，FVector は，X，Y，Z の 3 つの 32 ビット浮動小数点の値で構成され，三次元空間上の位置を表現するために用いられる．各軸の意味を示す空間的な関係は図 2 に示す通りである．このように，座標情報が明確かつ固定された形式で格納されていることから，その数値の増減傾向や変化パターンをもとにメモリスキャンを行うことで，位置情報が比較的容易に特定される危険性があり，実際にスキャンを行うチートツールも容易に入手できる．

2.2 一般的な WH 対策

WH に対しては以下のような対策が講じられてきた．

- 署名検出型アンチチート
  改造 DLL やチートツールのプロセス注入を検知する．（例：Valve Anti-Cheat(VAC)[7]，BattlEye[8]）
- 整合性チェック
  ゲームが動作するクライアントのコード改変やリソース変更を検出する．
- 描画ベース検出
  描画されていない対象に視線を向けている行動を検出する．
- サーバサイド検証
  プレイヤーが本来視認できない敵に対してエイム補正などを行っていないかをゲームサーバ側で統計的に検証する．

```
struct {
    float X;   // X 座標
    float Y;   // Y 座標
    float Z;   // Z 座標
};
```

図 1 FVector 型構造定義(C++)

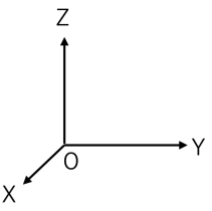

図 2 FVector 型座標軸の概念

しかし，これらの手法は回避される可能性が高く，特にクライアント側メモリの保護が十分でない場合には WH を防ぎきることが難しい．

2.3 メモリ難読化による防御手法

前述の Van Cauter の研究[3]では，プレイヤー座標を木構造として分割し，各ノードをメモリ上に離散配置することで座標全体の読み出しを困難にする手法が提案されている．加えて，座標データに対して簡易的な XOR 暗号を導入することで，メモリスキャンによる直接的な取得を妨げる実装も行われた．このアプローチの特徴としては，第一に，データ構造を複雑化することでアドレスの予測を困難にし，メモリ攻撃に対する耐性を高めるという点がある．これは Moving Target Defense（MTD）の一形態といえる．第二に，XOR 暗号を用いてデータを暗号化することで，仮にスキャンによってメモリ上の値を取得されたとしても，それ自体が意味を持たないように設計されている点が挙げられる．ただし，XOR 暗号は鍵の導出が比較的容易であり，強度の面では限定的である．このため，この研究においても，より強固な暗号アルゴリズムの導入が今後の課題であると指摘されている．

2.4 ゲームにおける暗号化技術の応用

ゲームエンジン上で暗号化を用いる試みはまだ限定的であるが，セーブデータやネットワーク通信などでは広く採用されている．ChaCha20 は，軽量かつ高速な暗号化手法としてゲーム業界でも注目されている．一方で，これらの暗号をリアルタイム処理が要求されるゲーム座標処理に適用する場合，パフォーマンスへの影響や実装の複雑さが課題となる．

2.5 本研究の位置づけ

本研究は，Van Cauter の手法をベースにしつつ，同手法で用いられていた簡易的な XOR マスク処理を，より高強度な暗号方式へと置き換えることで復号耐性の向上を図ることを出発点としている．

Van Cauter の手法は，位置情報などのゲームデータを小ブロックに分割し，階層的に管理することで不正アクセス耐性と更新効率を両立するデータ構造である．本研究ではこれを拡張し，ゲームデータを構造体単位で管理する方式を採用した．構造体単位で管理するために設計した木構造の概要を図 3 に示す．ここで offset は，各内部ノードの子配列における実スロット位置を表す．木構造の経路切替を行う間隔を定めるフレーム数である TTL_frames ごとに疑似乱数（std::mt19937_64）を用いて，std::shuffle で再割当てし，さらにゲームデータ移動先のノードも乱数によって決定される．よって，同じ論理構造でも実際に走査される経路と到達アドレスが周期的に変化する．図中の青矢印は切替前の経路，オレンジ矢印は切替後の経路を示し，最下層

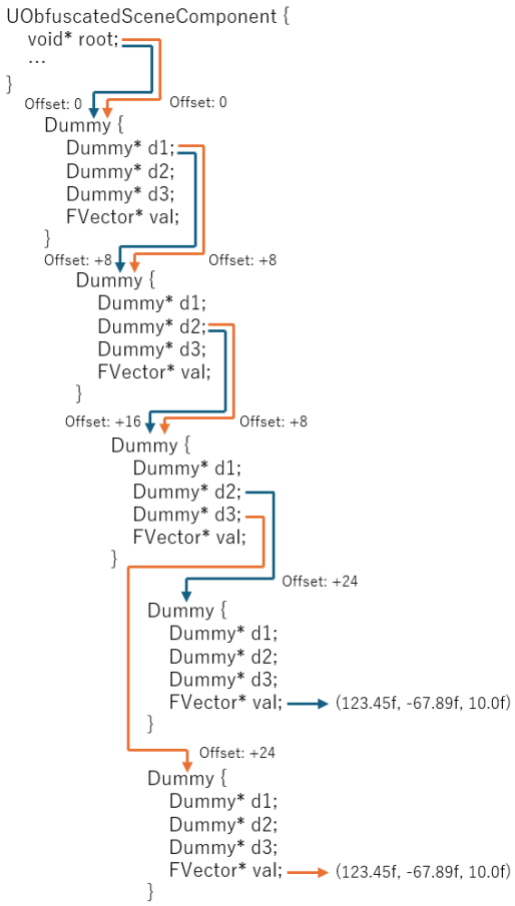

図 3 Van Cauter 提案の木構造を適用したもの

の実データを含む葉ノードが青からオレンジへと切り替わることを表している．

暗号方式には，近年 NIST によって軽量暗号標準として選定された Ascon や，IETF により標準化され多くの商用サービスに採用されている ChaCha20 といった，暗号強度が高く現実的な実装が可能な方式を用いて比較検証する．また，本研究では C++サンドボックス環境において暗号処理を導入し，将来的には Unreal Engine 5（UE5）ベースのゲームに実装を拡張し，実運用環境における選定基準の提示を目指す．なお，本研究が対象とする攻撃はクライアントのゲームプロセスのメモリに対する読み出しと値固定に限定し，値の改ざん検知は行わない．また，ネットワーク経路やサーバ側の漏えいは評価対象外とする．

3. 提案手法

本章では，ゲーム内で扱うプレイヤー座標データ（3 次元ベクトル）について，平文の露出時間を最小化し，かつ安定したポインタチェーン（再現可能な参照経路）に載せないことを目的とする秘匿化手法を示す．本手法は，(1)座標データを原則暗号化したまま保持し，描画や物理計算など使用の直前にのみ復号，使用の直後に直ちに再暗号化する方式，(2)暗号文の保存先と参照経路を時間的に変化させる MTD 木構造の組合せからなる．

3.1 概要

プレイヤーの三次元座標データ（FVector）を原則として暗号文のまま保持し，描画・物理計算などで使用直前にのみ復号し，使用直後に直ちに再暗号化する方式を採用する．さらに，暗号文の保存先と参照経路を時間的に変更する木構造を併用し，単一の参照経路に依存した追跡を困難にする．

暗号アルゴリズムとしては，従来研究で使用している XOR，実装が容易かつ高速なストリーム暗号の代表として ChaCha20（認証なし），完全性を備える軽量認証付き暗号の Ascon-128（AEAD）の 3 種類を選定した．これらを処理速度，高速な暗号化処理と機密性，完全性の三軸で，性能と防御効果のトレードオフの検証を行う．なお，ノンスは毎回一意になるよう単調カウンタで生成し，再利用しない．それぞれの暗号アルゴリズムの特徴と留意点を以下にまとめる．

- XOR : 極めて高速かつ実装が容易だが，鍵と平文の関係が単純で安全性は低い．また，機密性と完全性は提供しておらず，比較用の評価基準とする．
- ChaCha20（認証なし）: ソフトウェア実装が容易なストリーム暗号で，高いスループットと安定した遅延特性を示す．ただし提供できるのは機密性に限られ，改ざん検知や値の固定化に対する保護は含まれないため，運用上はノンス，カウンタの一意性維持に加えて別機構の併用が必要となる．
- Ascon-128（AEAD）: 暗号化と同時に認証タグで改ざん検出が可能であり，機密性と完全性を提供する．ノンスは一意であることが要件であり，本研究では単調カウンタで生成する．また，リプレイ検知は含まれないため，単調カウンタや時刻を用いたリプレイ検知の検証が別途必要である．計算コストは，ChaCha20 に比べて相対的に増加する．

以上より，毎フレームの短命な座標データに対し，利用時に即時復号→処理→即時再暗号化の運用に適した方式を選定し，ノンス再利用リスクと計算負荷のトレードオフを含めて評価する．

3.2 座標格納処理の暗号化

ゲーム内で更新されるプレイヤー座標である 3 成分の浮動小数点ベクトルをメモリへ格納する直前に暗号化し，暗号文として保持する．処理フローは図 4 に示す．

TPS 型ゲームの例として，UE5/Lyra Starter Game への適用を想定する．具体的には，USceneComponent::SetWorldTransform()または UPrimitiveComponent::MoveComponentImpl()に入る直前の関数呼び出し箇所をラッパー関数で包み，平文の FVector を暗号文（バイト列）と復号に必要な付加情報（ノンス，必要に応じて認証タグ）を一体化したデータへ変換して内部に渡す．例えば，SetEncryptedTransform()のようなラッパー関数を用意する．

3.3 座標取得時の復号

座標を利用する直前にだけ復号し，使用直後に同一メモリ領域を再暗号化して上書きする．具体的には，座標読み出しの呼び出し箇所（例 : AActor::GetActorLocation()または USceneComponent::GetComponentLocation()）をラッパー関数で包み，関数入口で復号して一時変数として受け取り，描画や物理計算に渡したのち，関数終了時に再暗号化して保存する．その際，復号値は関数スコープ外に残さない．

3.4 手法の特徴と前提

本手法は，座標の格納・取得の呼び出し箇所をラッパー関数で置き換え，その内部に暗号化，復号の最小限の前後処理を閉じ込めることで，既存のゲームロジックや描画処理への改変を抑えつつ導入できる．暗号方式は差し替え可能な設計とし，具体的な選択可能な方式（XOR, ChaCha20, Ascon-128）やその性質は3.1 節に示した．さらに，暗号文の保存先と参照経路を時間とともに切り替える木構造（MTD）を併用し，固定的な参照経路（ポインタチェーン）への依存を避ける．

ただし，メモリアロケータの挙動やコンパイラ最適化は環境に依存するため，本稿で示す結果は実験条件下での観測に限られる．また，鍵の配布，ローテーション等の鍵管理や，復号境界に対するフック対策は評価対象外とし，ユーザ空間でのスナップショット型攻撃（メモリスキャン/値固定）に対する抵抗性と性能の両立を主として検証する．

4. 実装

本章では，提案手法を C++サンドボックス（Google Benchmark を用いた単体実行バイナリ）として実装し，フレーム処理を模した環境で性能および挙動を評価できるようにした．サンドボックスは，UE5/Lyra Starter Game の「座標更新→使用（描画/物理）→保存」という最小限のデータフローのみを抽出して再現している．

4.1 共通の設計方針

プレイヤー座標は常に暗号文（バイト列）の形でメモリに保持し，平文は原則としてヒープや静的領域に残さない．

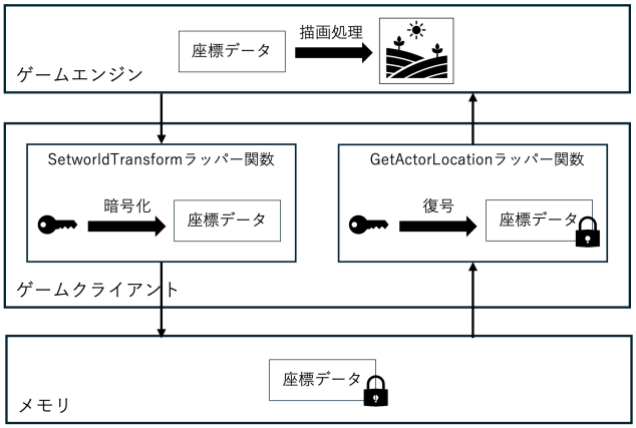

図 4 ゲームクライアントにおける座標データの暗号化と復号の処理フロー

描画や物理計算で使用する直前にのみ復号し，使用が終わり次第，同一領域を直ちに再暗号化して上書きする．さらに，保存先と参照経路を時間とともに切り替える木構造を併用し，特定の安定した参照経路への依存を低減する．ここでは鍵管理（配布・ローテーション等）は評価範囲外とし，ノンスは単調カウンタで毎回一意に生成して再利用を避ける．実装は Windows11，MSVC，C++20 を前提に構築し，計測には Google Benchmark v1.8.3 を用いた．また，ChaCha20 用のライブラリ libsodium と Ascon-128 用のライブラリ ascon-c は，静的にリンクした．

4.2 各手法の実装

各手法の UE5 への適用に先立ち，Google Benchmark を用いたサンドボックス環境で座標処理の最小構成を実装し，暗号化方式の差による挙動と性能を比較する．扱う座標は 3 成分の浮動小数点ベクトル（float x, y, z）で，合計 12 バイトの簡易構造体であり，UE5 の FVector に相当するデータ形状を想定する．1 イテレーションを 1 フレーム相当とみなし，「更新→暗号化して保存→（使用直前だけ）復号→使用後ただちに再暗号化」の流れを統一的に再現する．

なお，本サンドボックス実装における鍵，ノンス生成と暗号化/復号の接続関係を図 5 に示す．暗号，復号の直前に参照される鍵生成コンポーネントを用意し，ベンチマーク開始時にセッション鍵（方式に応じて 256bit/128bit）を確定する．各フレームではノンスを単調カウンタで一意に生成し再利用しない．なお ChaCha20 のカウンタはライブラリ内部で自動管理され，外部では未使用である．

以下では，(1)暗号化も木構造も使用せず描画のみを行う場合，(2)暗号化のみ行い木構造は使用しない場合，(3)暗号化も木構造も使用する場合，それぞれの方式について実装内容を説明する．

4.2.1 描画のみ

座標更新と使用相当の軽微な計算のみを行い，暗号化は

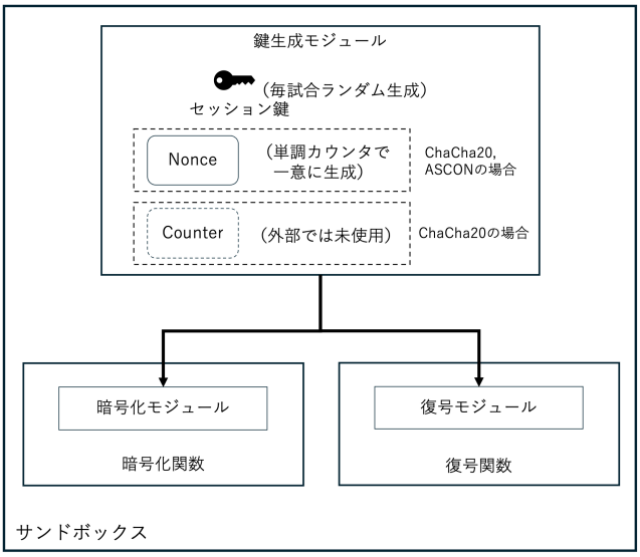

図 5 各暗号化方式における鍵管理方式

一切行わない．これを以後の方式に対する処理性能の比較の基準とする．

4.2.2 暗号化（木構造なし）

暗号文は，単一の保存先に保持し，フレーム毎に使用直前にだけ復号して用い，直後に同一領域へ再暗号化して上書きする．以下の 3 方式で性能を比較する．

- XOR：1 バイト鍵で各バイトを XOR する最小実装とする．高速だが安全性は低くなるが，演算コストが極めて小さく，ゲームプレイへの影響が最小限で済む．一方で，鍵と平文の関係が単純な規則で決まるため，鍵推測やビット反転に対して脆弱であることから，安全性は限定的となる．
- ChaCha20（認証なし）：256 ビット鍵のストリーム暗号として用い，認証は付与しない（機密性のみ）．実装上のノンスは 64 ビットとし，単調カウンタで毎回一意に生成する．
- Ascon-128（AEAD）: 128 ビット鍵，128 ビットノンス，128 ビットタグを用いる．復号時にタグ検証を行い，改ざんや値固定攻撃を検知可能となる．ノンスは単調カウンタで毎回一意に生成する．

4.2.3 暗号化+木構造

暗号文の保存先と参照経路を時間的に変化させる木構造を用いる．管理モジュールの tick()を用いて，フレーム毎に経路のローテーションを行い，暗号化した座標データを動的に移動する．これにより，単一のポインタチェーンに依存した追跡の再利用性を下げることを狙う．また，暗号アルゴリズムは4.2.2 節と同一のものを使用して比較する．

4.3 定量評価

本節では，描画のみ（基準），暗号化，暗号化+木構造の各方式を比較し，性能と平文露出抑制の観点から評価する．対象データは 3 成分の浮動小数点ベクトル（float x, y, z，計 12 バイト）であり，UE5 の FVector と同等のデータ形状を想定する．また，以下の評価手法はゲームクライアントのメモリ上の座標データに対する読み出しおよび値固定攻撃を想定する．暗号方式は XOR，ChaCha20（認証なし），Ascon-128（AEAD）を用いる．各指標と定義を以下に示す．

- 性能：フレーム時間tの p_{50}，p_{95}［ns］（中央値，95 パーセンタイル），オーバーヘッド

$$O = \frac{t_{p50} - u_{p50}}{u_{p50}}$$

（t：各実装の実時間，u：描画のみの場合の実時間）
本評価ではフレームレートを固定しない．代わりに，描画のみの方式で得られたu_{p50}を基準に，各方式の比をオーバーヘッドOとして算出する．

- 露出抑制：平文露出時間

$$E = v_{p95} - w_{p95}[\mathrm{ns}]$$

（v：再暗号化完了時刻，w：復号開始時刻）
復号開始から再暗号化完了までの時間を対象とする．

- メモリ：暗号文拡張率 =

$$\frac{|ct|_{bit} + |nonce|_{bit} + |tag|_{bit}}{|pt|_{bit}}$$

（ct：暗号文，$nonce$：ノンス，tag：認証タグ，pt：平文サイズ）
暗号化後に実際に保持し，転送する総バイト数を平文のバイト数で割った比率である．本研究の対象データは座標の float×3 で合計 12 バイトなので，pt=12 で計算を行った．XOR は 1 倍，ChaCha20（nonce=8）は約 1.67 倍，Ascon-128（nonce=16，tag=16）は約 3.67 倍となる．これらは木構造の有無に依存しない固定の増加分である．

- 改ざん耐性：値固定攻撃成功率
開始フレームを乱数で選び，その時点の暗号文をスナップショットし，その後 200 フレームの間，固定値で上書きする．今回は，読み出し座標の開始値と各成分の絶対値の差が 16 フレーム連続して 10^{-5} 以内であれば成功とした．
- ノンス運用：ノンス衝突件数，カウンタの巻戻りの有無

なお，本評価で用いる木構造の深さは 3（固定）とする．また，表 1 の「+木構造」行と表 2 は，$TTL_{frames} = 100$（TTL_{frames}：木構造の経路切替を行う間隔を定めるフレーム数）とする．TTL_{frames}は，値が大きいほど参照経路の再利用性が下がり，ポインタチェーン追跡や取得済み値の連続上書きの成功率を抑える．また，$TTL_{frames} = 0$は，木構造の経路切替を行う間隔が最大頻度であることを意味する．一方，更新処理の負荷とキャッシュ効率の悪化を招く．値が小さい設定だと性能に有利だが，経路が安定し予測されやすくなる．したがって，本指標は大きいほど防御寄り，小さいほど性能寄りとなることを意味する．

表 1 に示すとおり，XOR は最小のオーバーヘッドであり，ChaCha20，Ascon-128 は平文露出時間Eをいずれもサブマイクロ秒オーダーに抑えられている一方で，Oは約 3〜6 倍に増加している．木構造の追加は全方式でOをさらに増加させるが，Eは 0.1〜0.6μs に収まった．Google Benchmark は CSV 出力の少数表示桁に制約があり，ns〜us 帯の下位桁が 0 に丸められて出力されない場合がある．

表 2 に示すとおり，暗号化方式に依存せず値固定攻撃は成立しており，暗号化だけでは抑止できていない．木構造を有効化すると経路切替の効果で成功率は下がるものの，完全には無効化できていない．なお，ChaCha20 と Ascon-128 では，ノンスの衝突やカウンタの巻戻りは観測されず，ノンス運用は適切に機能していた．XOR と木構造のみの場合はノンス運用の対象外となる．

表 3 に示すとおり，1 フレームの間に暗号化された座標データが入っている葉ノードが何回切替わるかを表す，葉ノード切替頻度はTTL_{frames}にほぼ反比例し，TTL_{frames}が

表 1 処理負荷と平文露出時間への影響

方式	t_{p95}[ns]	オーバーヘッドO	平文露出時間E[ns]
描画のみ	1.56	1	対象外
XOR	49.4	1.34	100
ChaCha20	463.1	3.93	300
Ascon-128	574.6	4.66	400
XOR +木構造	459.3	3.90	300
ChaCha20 +木構造	822.9	6.20	500
Ascon-128 +木構造	908.8	6.83	600

表 2 値固定攻撃成功率とノンス運用の測定結果

方式	値固定攻撃成功率[%]	ノンス衝突件数[件]	カウンタの巻戻り
木構造のみ	90.5	対象外	対象外
XOR	100	対象外	対象外
ChaCha20	100	0	無
Ascon-128	100	0	対象外
XOR+木構造	90.0	対象外	対象外
ChaCha20+ 木構造	90.5	0	無
Ascon-128 +木構造	90.5	0	対象外

表 3 TTL frames と葉ノード切替頻度の関係

TTL_frames [frames]	葉ノード切替頻度 [per frame]
0	2.99
25	0.039
50	0.019
100	0.009

表 4 定性評価の結果

手法	メモリスキャン耐性	実装容易性
XOR	×	◎
ChaCha20	◎	◎
Ascon-128	◎	△〜○

増える毎に段階的に低下した．また，$TTL_{frames} = 0$では，木構造の経路切替の最大頻度が 1 フレーム毎に 2.99 回であった．

4.4 定性評価

暗号化方式の選定において重要となるメモリスキャン耐性（チートツールでの検出が困難であること）と，実装容易性（UE 環境への統合容易性，ライブラリ成熟度，デバッグのしやすさ）を，既知の実装前提に基づき定性的に比較する．比較結果を表 4 に示す．表中の記号評価は四段階評価（◎, ○, △, ×）で，◎は良好，○は概ね良好，△は注意点が多く要配慮，×は本研究の要件を満たすには不十分であることを意味する．以下に，各手法における評価の理由を示す．

- XOR : 実装が最も容易である．しかし，暗号の安全要件を満たさず，メモリスキャンや構造解析に対して脆弱である．また，同一鍵を使い回す運用では，暗号文間の統計的相関が露見し，差分から平文が推測可能となる．AEAD による改ざん検知も行えないため，安全要件を満たす運用は困難である．以上より，XOR は研究用のベースラインとしては有用だが，実運用向けの選択肢にはならない．
- ChaCha20 : ライブラリ libsodium による API が簡潔で実装容易性が高い．暗号文は統計的にランダムに近い性質を示し，座標の規則性を手掛かりにするメモリスキャンの成立を抑制する．実装では 8 バイトのノンスに単調増加カウンタを用いることで再利用リスクを抑制し，木構造の経路を TTL で定期的に切り替えつつ，各フレームでノンスを更新する構成と組み合わせることで，追跡への耐性を向上させる．一方，AEAD を伴わないため改ざん検知は暗号層だけでは提供されず，レンダーループ側での範囲チェックや，時系列一貫性チェック等の軽量検査を併用する設計が望ましい．総じて，平文露出の抑制と実装容易性のバランスに優れ，本研究目的に対する基本選択肢として妥当である．
- Ascon-128 : 軽量暗号でありながら AEAD による改ざん検知を備え，メモリ耐性は非常に高い．一方で，プロジェクトやビルド環境によっては，参照実装の API 形態や周辺ドキュメント量，組込み方などの差分が実装者の負担となりやすく，ChaCha20 と比べて実装，デバッグの難度が相対的に高くなる場合がある．したがって，チート耐性の最優先や軽量性を重視する設計方針では有力候補となるが，UE への適用や既存コードベースへの統合容易性を重視する場合には，初期導入のコストや保守性を吟味する必要がある．

4.5 評価のまとめ

採用した方式の中で，XOR は最小オーバーヘッドであった．ChaCha20 と Ascon-128 はオーバーヘッドが増加するが，平文露出時間は十分に小さい値であった．木構造の導入でオーバーヘッドはさらに増加するが，平文露出時間はほぼ同程度に抑えられた．TTL_{frames}を大きくすると葉の切替頻度が段階的に低下した．ただし，ノンスの衝突やカウンタの巻戻りは観測されず，ノンス運用は正常であった．値固定攻撃は，方式に依存せず成立した．木構造の経路切替で成功率は下がるが単独では抑止しきれないため，再生検知や時系列チェックなど上位レイヤでの対策が必要となる．また， ChaCha20 は Ascon-128 と比較して暗号拡張率が小さく，メモリ負荷をより抑制できる可能性がある．

総合的に，基本構成は実装容易な ChaCha20，改ざん検知を重視する場合は Ascon-128 を選び，XOR は実用的でない

ことが分かった．

5. 考察

本研究は，座標の常時暗号化と木構造の併用で，WH に対する平文露出時間の短縮を実現している．評価の結果，暗号化は 1 フレームあたりの処理時間を増やす一方で，平文露出はサブマイクロ秒オーダーに抑えられた．しかし，値固定攻撃は暗号方式に依らず成立し，木構造によって成功率は下がるが無効化には至らなかった．

5.1 チート対策としての実用性

暗号化は，数値パターンに依存する静的メモリスキャンは困難化すると考えられるが，暗号文の再利用による値固定攻撃は防げない．また，木構造は，一度特定した参照経路やアドレスなどの再利用性を下げる補助的効果を持つが，単独では十分な効果は得られなかった．よって，改ざん検知，リプレイ検知（単調カウンタや時刻），範囲と時系列の整合性チェックなどの上位レイヤでの検証を併用する構成が実用上必須である．

5.2 実装上の課題と解決策

Ascon-128 は AEAD を提供する一方，実装やデバッグの負担と計算コストが相対的に高い．ChaCha20 は，実装が容易で性能も良好だが，認証には別機構が必要となる．鍵配布やローテーションは本研究の対象外であり，実運用では，鍵管理，鍵バッファの即時消去，一定時間での鍵更新を設計に組み込むことで安全性を高めることが期待できる．

5.3 ゲーム実装への応用可能性

提案した木構造と暗号化の手法は，座標に限らずカメラ座標，可視状態フラグ，弾道や当たり判定などの短寿命なクライアント側データにも同じ容量で組み込みが可能である．これにより，メモリからの直接読み取りや値の固定化の成功率が低下する．これにより，First Person Shooting（FPS）ゲームや TPS ゲーム以外においてもミニマップや位置座標を扱う Multiplayer Online Battle Arena（MOBA）や Massively Multiplayer Online（MMO）などに転用が可能である．一方で，フレーム時間とメモリ占有は増えるため，適用対象を重要データに絞る構成が現実的である．

5.4 今後の課題

今後の課題として，まず値固定検知に対する強化が挙げられる．具体的には，AEAD の追加データにフレーム番号などの単調増加値を含め，復号時に現在地と一致しない暗号文を無効とする必要がある．併せて，マルチスレッドおよびマルチプレイ環境下でのスケール特性の評価，UE5 での検証，さらに GPU やドライバ経路，DMA に起因するデータ転送経路での平文露出の対策を進める．加えて，対象データ型の拡張と，バッファ配置やサイズの最適化により，適用範囲と実行効率の向上を図る．総合方針としては，基本構成に実装容易な ChaCha20，改ざん検知を重視する場合には Ascon-128 を採用し，XOR は評価用の基準としての採用に限定する．

6. まとめ

本研究は，WH によるゲームクライアントのプロセスメモリからの座標の読み取りを想定し，座標の常時暗号化と木構造によるメモリ難読化を併用する防御手法提案し，C++サンドボックス環境で定量評価を行なった．暗号化は 1 フレームあたりの処理時間を増加させたが，平文露出時間はサブマイクロ秒オーダーに抑制できた．一方で，値固定攻撃は防げず，木構造が成功率を低下させる段階に留まった．この攻撃は WH とは直接関係のない手法であるが，提案手法に対してこの攻撃が成立する場合，復号された値を用いたチートツール開発の足掛かりとなる可能性があるため，対策が重要である．また，併用する暗号化方式としては，処理速度や実装容易性を優先する場合は ChaCha20，改ざん検知を優先する場合は Ascon-128 が有効であり，XOR は実用的ではないことが判明した．

今後は，AEAD の追加データを用いたリプレイ検知の強化，UE5 への適用とマルチスレッド環境でのスケール評価，GPU や DMA 経路を含む平文露出時間の抑制，対象となるデータ型拡張とバッファ最適化を進め，多層防御の設計を実運用可能な形で確立することを目指す．

参考文献

[1] Choi Minsik, Ko Gwangsu, and Cha Sang Kil: BotScreen: Trust Everybody, but Cut the Aimbots Yourself, USENIX Security Symposium, p.1–16 (2023).

[2] Shacham, H., Page, M., Pfaff, B., Goh, E.J., Modadugu, N., and Boneh, D.: On the effectiveness of address-space randomization, Proceedings of the 11th ACM Conference on Computer and Communications Security, pp.298–307, Association for Computing Machinery (2004).

[3] Lars Van Cauter: Anti-Cheat Techniques for Games, Ghent University Master's Thesis, p.1–78 (2024).

[4] Hosseinzadeh, S., Rauti, S., Laurén, S., Mäkelä, J.-M., Holvitie, J., Hyrynsalmi, S., and Leppänen, V.: Diversification and obfuscation techniques for software security: A systematic literature review, Information and Software Technology, Vol.104, pp.72–93 (2018).

[5] Chen, Z., Gao, Q., Zhang, Y., and Shan, H.: ASCON: Anatomy-aware supervised contrastive learning framework for low-dose CT denoising, International Conference on Medical Image Computing and Computer-Assisted Intervention 2023, pp.355–365 (2023).

[6] Islam, M., Dong, B., Chandra, S., Khan, L., and Thuraisingham, B.: GCI: A GPU-Based Transfer Learning Approach for Detecting Cheats of Computer Game, IEEE Transactions on Dependable and Secure Computing, Vol.19, No.2, pp.804–816 (2022).

[7] Valve Corporation: Valve Anti-Cheat (VAC) System (online)，入手先 ⟨https://help.steampowered.com/en/faqs/view/571A-97DA-70E9-FF74⟩ (2025.08.13).

[8] BattlEye Innovations e.K.: BattlEye – The Anti-Cheat Gold Standard (online)，入手先 ⟨https://www.battleye.com/⟩ (2025.08.13).