End-to-end 秘密計算ストリーム処理の構想

中谷　翔†

† トヨタ自動車株式会社　〒 100-0004 東京都千代田区大手町 1-6-1 大手町ビル 6F
E-mail: †sho_nakatani@mail.toyota.co.jp

あらまし　パーソナルデータはデータ分析により価値をもたらし得る一方で、データ保護の対象になる。通信路や永続化されたデータの保護には暗号化が、計算中のデータには秘密計算が有効である。バッチ処理やデータベースのデータ保護では秘密計算の活用の取組みが見受けられるが、ストリーム処理での秘密計算活用は検討の余地が未だ大きい。本稿では、ストリーム処理の内部的なコンポーネントのうちウォーターマーク・ウィンドウ・ステート・チェックポイントが機密データを含み得ることを指摘し、イベントデータと併せてそれらを秘密計算、特に TEE により保護することを論ずる。

キーワード　ストリームデータ処理，セキュリティ・プライバシ，並列・分散処理，先進ハードウェア活用

1　序　　論

1.1　背景・研究の目的

個人が自らの個人情報や自らに関連付けられるデータ（以下、パーソナルデータ）をデータ管理主体に提供する機会は多々ある。Web サービスやアプリを通じたパーソナルデータ提供のほか、家電や自動車のような IoT 機器を通じた提供機会も増えてきた。

パーソナルデータ活用の面ではデータ分析や AI 分野の発展がある。パーソナルデータに主権を持つ個人に価値をもたらすサービスが進化している（例：EC サイトの推薦アルゴリズム、利用履歴からレシピ提案する調理家電）。

IoT データまでを加味すると、データの容量は莫大になり得る。経済産業省の調査報告 [1] を例にとると、2025 年頃にはすべての自動車（コネクティッドカー）が 1 日に生成するデータ量は約 10EB になると予想されている。この規模のデータを扱うためには分散処理が欠かせない。中でも、データから準リアルタイムに価値を生み出す用途ではストリーム処理が有望な技術である。

パーソナルデータを扱うに当たり、**データ管理主体**は個人情報保護法や GDPR などの法規を遵守することが求められる。永続化データや通信路のデータの暗号化によるパーソナルデータ漏洩等のリスク低減は重要であり、多くのデータ管理主体が永続化データの暗号化や通信路の暗号化といった対策を実施している。一方で、データ管理主体の預かるパーソナルデータを取り扱う主体は他にも存在し得る（図 1；3 章で詳説）。パブリッククラウドサービスの活用が背景となり、データ管理主体と**サーバーインフラ提供主体**が異なるケースは増えている。また、データ管理主体が**第三者**にデータを提供し、第三者がデータ分析やデータを元にしたサービス提供を行うケースも増えている。このことからデータ管理主体以外の主体が悪意を持った場合に、計算中のパーソナルデータを不当に観測・漏洩するリスクが高まっていると言えよう。そこで秘密計算を活用すれば、計算中のメモリや CPU バス上からデータの平文を観測することを防ぐことができる。

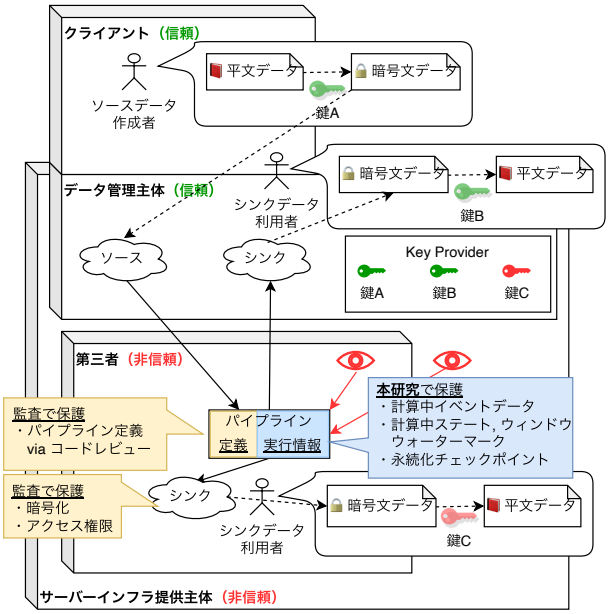

図 1　データを扱う主体・信頼境界・コンポーネント配置・データ保護方針（一例）

以上を背景とし本研究では、パーソナルデータなどの機密データを対象とするストリーム処理における、秘密計算を活用したデータ保護の手法を提案する。

1.2　ストリーム処理

ストリーム処理では、無限長になり得る流れるデータ（**イベントデータ**）を扱う。

バッチ処理やデータベースのクエリ処理ではスループットが重要な性能指標である一方で、ストリーム処理ではイベントデータが最初の入力から最後の出力までに要する時間であるレイテンシも同等以上に重要な指標である。

関係データベースにおける SQL とは事情が異なり、ストリー

ム処理においては中心的なプログラミングモデルが確立されていないのが現状である。そのような中でも Apache Beam は、Apache Flink, Apache Spark Structured Streaming, Google Cloud Dataflow を含む 10 種ほどのストリーム処理系に統一的なプログラミングモデルを提供することを目指すソフトウェアである。その中核には **Beam Model** [2], [3] があり、これがストリーム処理の標準的なプログラミングモデルに最も近い位置にあると著者は考える。Beam Model は FlumeJava [4] に端を発すると言われる [2] 高水準なデータフロー記述である。本研究においては Beam Model を対象に論を展開することとし、図 2 に、Beam Model の構成要素を本稿に必要な範囲で簡潔に示す。

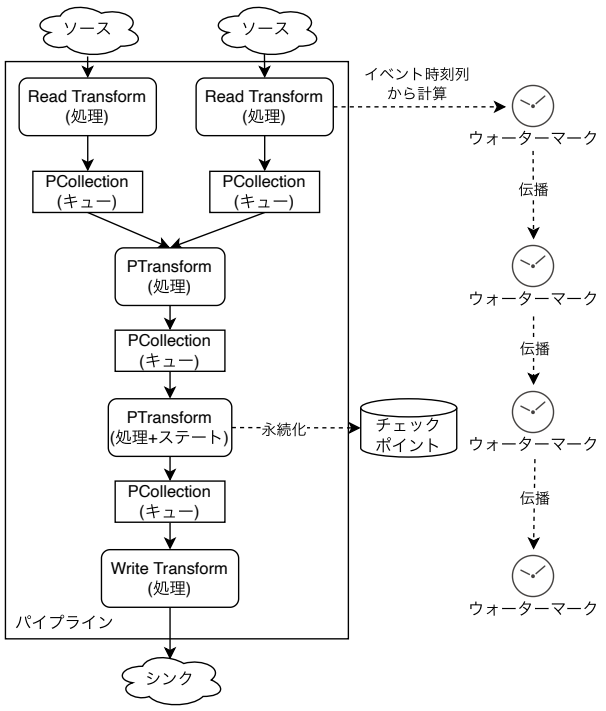

図 2　Beam Model の構成要素

パイプラインがストリーム処理全体を表すもので、DAG (Directed Acyclic Graph) 形式で記載される。DAG のノードは**トランスフォーム (PTransform, Read Transform, Write Transform)** または **PCollection** で、エッジはイベントデータの入出力を表す。

ソースから Read Transform がデータ加工することでパイプライン中にイベントデータが生成され、Write Transform ではイベントデータを**シンク**に書き出す。

トランスフォームは**ステート**を持つことができる。ストリーム処理におけるステートとは、個々のイベントデータのライフサイクルを超えて保持される状態であり、例えばある時間幅のイベントデータの列や、イベントデータの特定の数値フィールドの合計値などである。ストリーム処理系の故障に備え、ステートは**チェックポイント**として永続化される [2], [5]。

イベントデータは**イベント時刻**のフィールドを持つ。イベント時刻はパイプライン開発者が Read Transform において指定できる。例えばソースがログデータの場合、ログが記録された日時がイベント時刻として適当である。

Read Transform で観測されたイベント時刻の列を元に、**ウォーターマーク** [2], [6] が計算される。ウォーターマークとは（実時刻ではなく）イベント時刻を軸としたタイムスタンプのしきい値である。ウォーターマーク（またはそれに許容遅延を足したタイムスタンプ）は、それよりも小さなイベント時刻を持つイベントデータを遅延データと扱うために使われたり、イベント時刻ベースの**ウィンドウ**が閉じる時刻としても使用される。

1.3　TEE

サーバーインフラ提供主体・第三者までを潜在的な攻撃者とみなした場合、永続化データや通信路上のデータ保護のみならず、計算過程のデータ保護も重要である。さもなければ、メモリダンプなどの手段により機密データが攻撃者に漏洩することなどが危惧される。

このような脅威からデータの機密性を守りながら行う計算は秘密計算と呼ばれる。秘密計算の代表的な手法には、**TEE (Trusted Execution Environments)**・準同型暗号・MPC がある。準同型暗号は機密性の観点で申し分ないが、TEE と比べた場合に性能が大きく見劣りする [7]。MPC はデータ処理に複数の参加者が関与することが前提である一方で、本研究が対象にする分散データ処理は単一の参加者が行うことを中心と想定する。これらのことから、秘密計算を分散データ処理に応用することを目的とした本研究では TEE を利用することとする。

TEE は HIEE (Hardware-assisted Isolated Execution Environment) の一種であり、中でもプロセッサが提供する隔離空間で実行する処理をユーザーが定義することのできるものである。TEE における隔離空間は **enclave** と呼ばれ、この中で実行される命令や読み書きするメモリ上のデータについては、TEE 外 (**REE; Rich Execution Environment**) からは観測不可能であることがトラストアンカーとなっている。

TEE を実行するマシンと実行を要求するマシンが別々である場合には、実行マシンが期待通り TEE の機構を有しており所望の処理を実行していることを、要求マシンが確かめる必要がある。実行マシンの外部からプロセッサや処理の認証を行う手段として **RA (Remote Attestation)** がある。

攻撃者が enclave 内を直接観測することはできないにしても、TEE はサイドチャネル攻撃に対して脆弱であることが広く知られている。サイドチャネル攻撃に対する対策は様々研究されているが、原則としては enclave 内のコードを小さく保つことで、分岐の数を抑えたり、特定の関数の既知の脆弱性を突くような攻撃を回避することが有効である。

本研究では、TEE を中心に据えた暗号化により機密データを保護することを目指す。構想段階の本稿では具体的な TEE 技術に依った議論はしないが、コモディティサーバーのプロセッサで利用可能であり、かつ小さな enclave を指向する技術として、Intel SGX [8], [9] または ARM TrustZone [10] を有望視する。

1.4 ストリーム処理における機密データ

ストリーム処理におけるイベントデータを秘密計算で保護する研究は少数ながら見受けられ、2.1, 2.3 節に詳細に検討している。しかしそれらはいずれもイベントデータの保護だけを対象にしており、現代的なストリーム処理で実践的に用いられるユーザー定義のステートや、ストリーム処理系が暗黙的に計算・構成するウォーターマーク、ウィンドウ、ステート、チェックポイントの保護について言及していない。本節では、それらの機構への保護の必要性を、攻撃例を示すことで論じる。

1.4.1 ステート・チェックポイントの保護の必要性

Beam Model のトランスフォームはステートを扱うことができる。ステートには暗黙的なステートと明示的なステートがあり、このいずれにも保護が必要であると主張する。

ウィンドウを使った結合演算や、集約演算においては、暗黙的なステートが処理系内に構成される。例えば結合演算の場合、ウィンドウ中に 2 つの入力 PCollection からのイベントデータ列が蓄積され、このイベントデータ列がステートとなる。集約演算の場合も原則的にはイベントデータ列がステートとなるが、集約演算が交換法則と結合法則を満たす場合（例えば総和や平均）、最適化のために部分演算結果がステートとなることがある。

例えば結合対象のイベントデータが個人の氏名や住所であるような処理において、暗黙のステートが REE で平文計算されるならば、攻撃者はメモリダンプなどの手段により個人情報を入手することが可能である。

Beam Model ではユーザーの定義するトランスフォームにおいてステートを明示的に定義することができる。例えば、入力 PCollection が個人の氏名・居住地・収入であるトランスフォームにおいて、居住地ごとの収入の Top-k のイベントデータをステートとして保持するようなユースケースがあり得る。これも暗黙のステートと同様に、REE で平文計算される場合には驚異に晒される。

更に、ステートはチェックポイントとして永続化されるため、チェックポイントの暗号化もデータの機密性を保つために必要な措置である。

1.4.2 ウォーターマーク・ウィンドウの保護の必要性

イベント時刻の列は機密性の高いデータになり得る。イベント時刻列が行動ログの役割を果たし、データ列の内容の推測に繋がるためである。

例えば、個別の車両の行動を分析するパイプラインを考える。このパイプラインは *(*車両 *ID,* 開始日時*,* 終了日時*)* の組をパラメーターとして暗号化された経路で受け取り、車両の運転状況やセンサデータから総合的に運転行動における過失などを判断するものとする。ストリーム処理系実行サーバーにアクセスのある者が、そのパイプラインで何らかの事故の調査をしているという知識と報道などから事故の情報を得ている場合、イベント時刻の列から調査対象の事故を推定できる可能性がある。

ウォーターマークとイベント時刻ベースのウィンドウは、いずれもイベント時刻の列から計算される。ウォーターマークをイベント時刻列に基づき進めるアルゴリズムはヒューリスティック含め様々あるが、現在のウォーターマークを観測すれば直近で処理したイベント時刻を推測できる。同様に、イベント時刻ベースのウィンドウの始点と終点にはイベント時間軸でのタイムスタンプが付与されており、現在有効なウィンドウを観測すれば直近で処理したイベント時刻列の時間帯を推測できる。

以上のことから、ウォーターマークやイベント時刻ベースのウィンドウは enclave 内で扱うことが機密データの漏洩を防ぐために望ましいと考えられる。

1.5 貢　　献

TEE を用いたストリーム処理の既存研究（2.1 節）と比較した本研究の新規の貢献について述べる（ただし本稿は構想段階であるため貢献見込み）。

（1） ストリーム処理のプログラミングモデルとして一定の市民権を得ている Beam Model を対象にした、**UDF (User-Defined Function)** までを含んだ秘密計算を確立

（2） 現代的なストリーム処理の構成要素として一般的なステート・チェックポイント・ウォーターマーク・ウィンドウを機密性の高い情報を含み得ると指摘し、その保護方法を議論

（3） 実用を目指し、Apache Beam をフロントエンドとし Apache Flink または Apache Spark Structured Streaming などの OSS の著名なストリーム処理系をバックエンドとし手法を実装

（4） 分散ストリーム処理系に秘密計算を適用し、スループットだけでなくレイテンシについても元の処理系に匹敵するパフォーマンスを実現

1.6 本稿の構成

本稿の構成を示す。2 章で関連研究と本研究との比較を論じる。3 章で本研究の主題である end-to-end 秘密計算ストリーム処理のユースケースや実行モデルの概要を記す。4 章ではセキュリティ要件として、脅威モデルやそれに対応するためのデータの機密性を示す。5 章では主にセキュリティ要件に呼応する形でシステム設計を示す。本稿時点ではシステムは設計段階であり、続く 6 章で本稿の結論と今後の課題を述べる。

2 関 連 研 究

2.1 TEE による秘密計算ストリーム処理

TEE を用いたセキュアなストリーム処理を実現する試みは、SecureStreams [11] と StreamBox-TZ [12] に見られる。

SecureStreams は本研究と同じく、サーバー上での分散ストリーム処理を処理過程まで含めてセキュアに実行することを目的としている。TEE としては SGX を用いており、enclave の中に Lua の VM を格納し、Lua で記述された MapReduce [13] 処理を実行する。トランスフォームを Lua で汎用的に記述できる点は本研究の目的とも合致するが、処理系・プログラミングモデルとも独自のものである点は実用を遠ざけてしまっている。また、SecureStream はイベント時刻・ウィンドウ・ウォーターマーク・ステートといった概念に触れておらず、現代的なスト

リーム処理系とは乖離がある。また、性能指標はスループットに偏重している。更に、脅威モデルに対する考察がほとんど見受けられず、セキュリティに関する反証可能性を有していない。

StreamBox-TZ [12] は、IoT のエッジ（デバイスとサーバーの中継点）におけるストリーム処理を対象に、IoT データの機密性・整合性の担保と、出力の検証可能性と、高スループット・低レイテンシを目指す研究である。エッジという制約から単一サーバーのストリーム処理を検討している点は本研究と異なるが、それ以外の目的は本研究と近しい。TEE としては TrustZone を用いており、集計・ソート・ジョイン・フィルタなどの 23 個のプリミティブを小さな enclave 内コードで実現し、その組み合わせで Spark Streaming のトランスフォームを大部分実現している。トランスフォームのプリミティブを TEE で実現するアイディアは、攻撃対象を小さくするためにも性能を予見可能にするためにも優れていると考えるが、他方で UDF の enclave 内実行は範疇外とされている。また、ウォーターマークは暗号化されておらず、かつウィンドウ以外で構成される明示的なステートについても考慮されていない。処理系は独自の StreamBox [14] をベースにしており、実用からは乖離がある。

2.2 TEE による秘密計算データベース

TEE を用いたデータベースに関する研究は [15], [16], [17], [18], [19], [20], [21], [22], [23] など様々あるが、データベース（特に関係データベース）と本研究が対象にする Beam Model によるストリーム処理では、秘密計算やデータの暗号化を行う上で異なる考慮点がある（表 1）。

表 1　関係データベースとストリーム処理（Beam Model）の秘密計算・暗号化における考慮点の違い

	データベース	ストリーム処理
データ生存期間	data-at-rest	data-in-motion
処理生存期間	adhoc	continuous
秘密計算対象処理	関係代数演算	UDF 含むトランスフォーム
インデックス	あり	なし
ウォーターマーク	なし	あり
ステート	なし	あり

表 1 は主たる傾向でしかなく、当然例外はある。例えばデータベースであってもマテリアライズドビューは continuous な生存期間を持つと言えるし、UDF までを秘密計算対象とする取組みも現在のところ見受けられないが理論上はあり得る。

ストリーム処理はデータ生存期間が短いため、暗号化されたデータそのものへの観測から攻撃者に与える情報は少ないと考えられる。他方、同一のトランスフォームが持続的なプロセスとして動作し続けるので、攻撃者が所望の入力をパイプラインに投入できる状況下においては、入出力の関係から処理の解析が比較的容易とも考えられる。

関係データベースにおいては、関係代数演算を主たる秘密計算の対象としている。実際、関係データベースに対するクエリは関係代数で完結することが多く、ストリーム処理と比較して UDF の活用は少ない [1]。ストリーム処理では外部ライブラリ呼び出しを含むような広範な処理を含むことが実用上よくあるため、これも秘密計算の対象とすべきであると考える。

また、データベースではインデックスを暗号化した上で高速な検索を実施することが重要であるが、ストリーム処理は流れてくるイベントデータを全て計算対象とするため、インデックスへの考慮は不要である。

その一方でウォーターマークやステートはストリーム処理固有であり、かつ攻撃者に対して情報を与え得るものであるため、特別に対処する必要があることを本研究では主張する。

2.3 準同型暗号による秘密計算データベース・ストリーム処理

ストリーム処理の秘密計算を準同型暗号により実現する研究も存在する [24], [25]。しかしいずれも、現実的な性能で実現できる演算が限定的であり、ストリーム処理系のようなデータ処理基盤に準同型暗号を応用するには性能面が現段階では不足していると言われる [12]。

2.4 TEE で VM を秘匿化する技術

AMD SEV [26] は、VM (Virtual Machine) のメモリを TEE により秘匿化する技術である。OS やハイパーバイザーからの脅威から保護されるので、例えばパイプラインをストリーム処理系と一括で VM 化し、AMD SEV を利用するような使い方で本研究の目的を達成できる可能性もあると考える。

ただし、VM 内で実行される OS やライブラリのコードまでを信頼することが前提となってしまうため、攻撃ベクトルが広がることは避けられない [27]。また、RA (Remote Attestation) の安全性が証明されていないとの報告 [28] もあり、鍵交換を RA に依拠する本研究では現時点では採用が難しい。

また、SEV がデータベースやストリーム処理の要求性能に耐えられるという報告は知る限り存在せず、この点も慎重に検討する必要がある。

SEV 以外にも SGX でもアプリケーションやライブラリを一括で秘匿化する技術 [29], [30], [31] も存在するが、これにも SEV と同様の考慮が必要となる。

3 End-to-end 秘密計算ストリーム処理の概要

3.1 ユースケース

End-to-end 秘密計算ストリーム処理は、ソースとシンクを両端点 (end) とし、攻撃者がソース〜シンク間で機密データを（計算中のも含め）読み取ることを防ぐ技術である。図 1 に、本研究が対象とするデータを扱う主体、その中のアクターとコンポーネント、コンポーネントにおけるデータ保護方針を示す。

主体としては、データに主権を持つ**クライアント**、クライアントのデータを第一義的に取扱う**データ管理主体**、データ管理主体からのデータ提供を受けデータ分析やデータを元にし

1：ストリーム処理で UDF が多くなる理由を以下のように考察する。(1) データベースは前処理済みのデータが格納されるが、ストリーム処理では未加工データの前処理を記述することが多い。(2) 従ってストリーム処理はデータモデルが関係モデルと比べ緩やかであり、処理も定型化しづらい。

たサービス提供を行う**第三者**、そして**サーバーインフラ提供主体**を考える。サーバーインフラ提供主体はデータ管理主体が兼ねることもあり得る。また、第三者は存在しないこともあり得る。クライアントとデータ管理主体はデータ保護の目的において信頼でき、それ以外の主体は信頼できないこと（非信頼）を想定する。

パイプラインにデータを入力するのは**ソースデータ作成者**（人間の他、プロセスやセンサーなど）である。ソースデータ作成者はクライアントに所属する。ソースデータ作成者は自らの管理する暗号鍵（5.1 節に詳細）により、イベントデータを暗号化して入力することができる。ソースは複数取ることができ、ソースデータ作成者も複数になり得る。各々のソースデータ作成者は別個の暗号鍵を使用できる。

パイプラインからシンクにデータが出力され、**シンクデータ利用者**（人間の他、プロセスなど）がそのデータを読み取る。シンクデータ利用者はクライアント・データ管理主体・第三者のいずれかに所属する。出力データはシンクデータ利用者の管理する暗号鍵により暗号化することができる。シンクデータはデータベースや JSON ファイルなどの形式で構造化されることがあり、その場合はフィールドごとに暗号鍵を切り替えて、各シンクデータ利用者のアクセス制御を行うことが可能である。

パイプラインはデータ管理主体または第三者のサーバー（サーバーインフラ提供主体によるものであり得る）にデプロイされ、実行される。第三者またはサーバーインフラ提供主体がパイプラインを実行するサーバーにアクセスを有する場合、end-to-end 秘密計算ストリーム処理ではこの主体を潜在的な脅威と捉え（4.1 節に後述）、パイプラインの実行情報（計算過程・通信路上・永続化されたデータ）の観測による機密データの漏洩を防ぐ。意図的かを問わず、パイプラインの静的な定義・シンクの設定に不備がある場合には、パイプライン実行以外の箇所からの機密データの漏洩が懸念される。これに対するデータ保護にはデータ管理主体による監査が有効であると捉え、本研究の対象外とする。

3.2 実行モデル

サーバー上にデプロイされたパイプラインは、指定されたストリーム処理系（Apache Flink, Apache Spark Structured Streaming などが一例）により実行される。

トランスフォームは分散並列実行される。前段の PCollection やソースが異なるトランスフォームが並列実行される他、同一の PCollection 中の異なるイベントデータが別々のサーバー・CPU コアでトランスフォーム処理され得る。

トランスフォーム処理を行うサーバー間の通信や、そのサーバーとソース・シンクとの通信は、IP ネットワークにより行われる。サーバー同士は同一ネットワークセグメントに配置される場合が典型的だが、ソースやシンクはインターネットをまたぐ場合もある。

4 セキュリティ要件

4.1 脅威モデル

本研究では、信頼できない主体における **strong adversary** と **weak adversary** を脅威モデルとして考慮する [2]。

Strong adversary は Cipherbase [15] が対象にしている攻撃者 [3]であり、サーバー上のストリーム処理の管理・実行プロセス、OS、ハイパーバイザーに関して最大限の観測能力を持つ。また、通信路・ディスク・メモリ・CPU バスといったハードウェアに関しても最大限の観測能力を持つが、enclave 内の状態や計算は観測できない。

Cipherbase は weak adversary も考慮しており、これはディスクまたはメモリ上の *data-at-rest* に対するスナップショットを取得する能力を持つ。Data-at-rest とは、概念的に演算よりも長いライフタイムを持つデータのことであり、ストリーム処理においてはウォーターマークや、ステートとその永続化データであるチェックポイントが例となる。

Strong adversary と weak adversary はともに **passive adversary**（または *semi-honest*, *honest-but-curious* と呼ばれる）に属する攻撃者であり、計算自体には介入せずにもっぱら観測のみを行う。これに対し **active adversary**（または *malicious adversary*）は、計算ロジックやデータの改ざん・要求されていない計算の実行・計算要求の無視など、計算への介入も行う攻撃者である [32]。本研究での考慮は strong adversary までに留め、active adversary への防御は今後の課題とする。

4.2 データの機密性

本研究において機密性を担保するデータは、イベントデータ・ウォーターマーク・ウィンドウ・ステート・チェックポイントである。これらのデータについて **operational** の機密性 [15] を提供する。すなわち、strong adversary や weak adversary はデータのみを観測しても何らの情報も得られないが、データに関する演算の入出力を通じた観測により何らかの情報を得ることができる。例えば、暗号化された入力に一致フィルタリングや equi-join に含まれる一致比較演算を適用している場合、2 値出力結果から入力のパラメータとの一致性、あるいは入力同士の一致性に関する情報を得ることができる [4]。

TEE は一般にサイドチャネル攻撃に対し脆弱であり、演算の入出力の観測が可能との報告 [33], [34] があるため、operational な機密性までをターゲットとする。サイドチャネル攻撃に対する防御はプロセッサベンダーのアップデートや SGX-LAPD [35] や T-SGX [36] などの研究も活発な分野であり、今後の発展が期待されるところである。

2：図 1 の目のアイコンが該当。

3：後発の研究 [16], [17] は同じ strong adversary の用語で、通信路・ディスク・メモリへの改ざんまでをも含む一方で、それらの改ざんに対する安全性の議論を欠いている。本研究では Cipherbase の定義に倣う。

4：このケースでは 2 値出力をソルトを使って暗号化すれば機密性を高めることができる。

5 設　　計

Strong adversary, weak adversary に対する operational な機密性を担保するため、暗号化と秘密計算を応用する。本章では、暗号鍵、鍵交換、暗号化対象やその指定方法の設計を述べる。

5.1 暗　号　鍵

イベントデータのフィールド用の暗号鍵は、Always Encrypted [16] を参考にし、**CMK (Column Master Key)** と **CEK (Column Encryption Key)** の二種類を使用する。CEK は対象鍵であり、フィールドを暗号化・復号する際に使用する。CMK は非対称鍵であり、CEK の暗号化と、CMK メタデータ（CMK の保管場所等）への署名に使用する。

図 3 に、CMK と CEK の配置を示す。

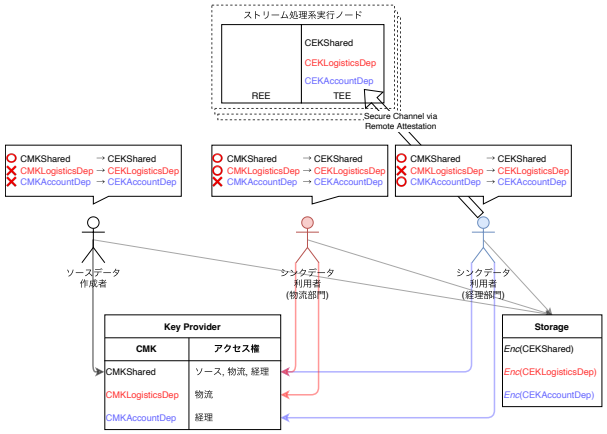

図 3　CMK と CEK（暗号文・平文）の配置

CMK は Key Provider（例: Key Vault サービスや HSM）に保管されている。Key Provider は認証機能を備えており、アクセス権が設定された人間やアプリケーションにのみ CMK は開示される。CMK にアクセス権を持つのはソースデータ作成者やシンクデータ利用者などのクライアントだけであり、ストリーム処理系はアクセス権を持たないのがセキュリティ上の前提である。

CEK は CMK で暗号化・署名された状態で、クライアントからアクセス可能なストレージ（例：ローカルストレージやオブジェクトストレージ）に配置されている。ストリーム処理系やその実行サーバーも暗号化された CEK にアクセス可能であっても良いが、CMK へのアクセスがないため平文の CEK を取得することはできない。CEK に付与された署名は CMK の公開鍵で検証可能とし、CEK の所有者を CMK の所有者と紐付ける。

大部分のトランスフォームは、イベントデータを平文にしなければ計算を実行できない。従って、トランスフォームを実行する enclave に、平文の CEK を安全に伝送する手段が必要となる。この伝送路には、RA で確立したセッション鍵を使った Secure Channel を用いる。詳しくは 5.2 節を参照。

CEK は、イベントデータのスキーマのフィールドごとに切り替えることができる。例えば図 4 では、EC サイトでの購入情報をストリーム処理し、最終的に購入情報 (Purchase) がシンクに出力される。Purchase スキーマの各フィールドは異なる CEK で暗号化されており、物流部門と経理部門が復号できるものとできないものに分かれる。ここでこれらの CEK はそれぞれ異なる CMK から生成されたものであり、CMK は Key Provider により部門ごとにアクセス権限が設定されている。つまり CMK に対する認証・アクセス権限により、イベントデータのフィールドごとのアクセス権限が設定できる。

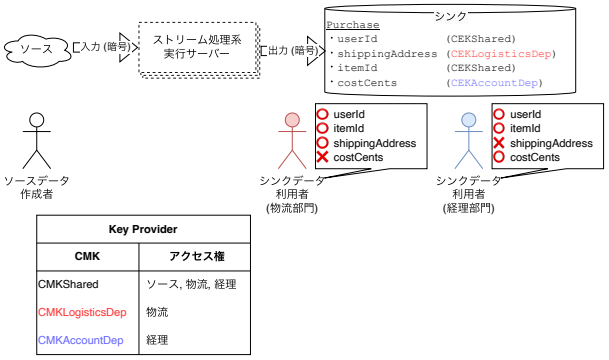

図 4　CMK のアクセス権分離によるイベントデータの読み書き権限の分離

5.1.1 CEK の enclave への配布

Always Encrypted [16] などのデータベース製品では、enclave がクエリ計算に CEK を必要とする場合には、クライアントがクエリ時点で enclave を CEK に配布すれば良い（鍵交換問題については 5.2 節参照）。

一方、ストリーム処理ではトランスフォームは常時プロセスであるため、パイプラインのデプロイ時点で enclave に CEK を配布することが必要となる。かつ、データベースではクエリを入力する主体と処理結果を受け取る主体が同一であるのと対照的に、ストリーム処理ではパイプライン開発者・ソースデータ作成者・シンクデータ利用者が分かれている場合もある。CEK の平文へのアクセスには CMK へのアクセスが必要であり、CMK へのアクセスには認証が必要であることから、デプロイ時にはパイプライン開発者のみならず、CMK へのアクセス権限を保有する主体が認証に関与することが求められる [5]。

しかしデプロイ時の CEK 配布だけでは、サーバーの自動的なスケールアウトや障害復旧に耐えることができない。動的に追加されるサーバーの enclave にも CEK が必要となるためである。サーバー追加時には enclave 同士が安全な伝送路を確立して CEK を配布する仕組みが必要になると現時点では考えるが、詳細な検討は今後の課題とする。

5：例えばデプロイジョブにおいて、認証済みの CMK 権限保有者による確認を促すプロンプトを表示するなどの手段があり得る。デプロイ時に CMK 権限保有者の同席が望めないのであれば、デプロイ時に一時的に CMK へのアクセス権限をパイプライン開発者に供与する運用もあり得る。

5.1.2 システム鍵

本研究では、イベントデータの他にもステート・ウォーターマーク・ウィンドウを処理系内部で暗号化する。これらの暗号化には CMK, CEK と同構造の **SMK (System Master Key)** と **SEK (System Encryption Key)** を用いる。SMK にはデータ管理主体のアクターがアクセス権限を持つものとする。

5.2 Remote Attestation (RA)

RA には大きく分けて 2 つの役割がある。

（1） TEE を搭載する CPU、並びに enclave 内で実行されるプログラムの完全性を検証

（2） RA を要求するクライアントと enclave の間で安全にセッション鍵を交換

本研究では後者の役割を活用し、暗号化されたセッションを通じてクライアントから平文の CEK を enclave に配布する（図 3）。

StreamBox-TZ [12] や Always Encrypted [16] では前者の役割を活用し、トランスフォームやクエリの実行のたびに RA を行うことを原則としている。これは両者ともに active adversary を脅威モデルとしており、データ処理系が誠実に処理を実行することを信頼していないためと考えられる。例えば、要求された処理とは違う処理を実行したり、要求された処理を無視するような懸念への対処である。図 1 に示した本研究の保護範囲においては前者の役割の活用は不要だが、同図の監査の実現に前者の活用は有益であるため、システム全体としては活用することが望ましい。

5.3 データの暗号化・秘密計算

4.2 節に記載したセキュリティ要件を達成するため、各種のデータについて行う暗号化や秘密計算について論ずる。

5.3.1 イベントデータ

イベントデータに対しては、パイプライン開発者の要求に応じてフィールドごとの粒度の暗号化を施すことでデータの機密性を担保する。パイプライン開発者による暗号化指定は、Beam Model のスキーマ（PCollection に入るイベントデータの型）定義におけるアノテーションで実現する（図 5）。

アノテーションの中では、暗号鍵・暗号タイプ・暗号アルゴリズムを選択できる。

暗号鍵には 5.1 で説明した CEK を用いる。CEK は、必要に応じてフィールドごとに切り替えることができ、図 5 では図 4 に対応するように各フィールドの CEK が選ばれている。

暗号タイプには *RND*（ランダム）または *DET*（決定的）の二種類がある。RND では暗号の IV（初期ベクター）に疑似乱数を用いて暗号化し、同一の平文から異なる暗号文が生成される（暗号文に IV を付随させて復号可能にする）。DET では固定値の IV を用いて暗号化し、同一の平文からは同一の暗号文が生成される。セキュリティ強度が高いのは RND であるが、DET なフィールド同士の比較は REE でも行うことが可能である。従って、一致選択・内部結合・集計などのキーに使うフィールドは、DET としておくことで TEE の利用を回避し、処理性能が向上する余地がある。

```
@DefaultSchema(JavaFieldSchema.class)
public class Purchase {
  @Encrypted(key="CEKShared", encType=RND,
             encAlgorithm=AES_256_CBC)
  public String userId;

  @Encrypted(key="CEKLogisticsDep", encType=RND,
             encAlgorithm=AES_256_CBC)
  public ShippingAddress shippingAddress;

  @Encrypted(key="CEKShared", encType=DET,
             encAlgorithm=AES_256_CBC)
  public long itemId;

  @Encrypted(key="CEKAccountDep", encType=RND,
             encAlgorithm=AES_256_CBC)
  public long costCents;
}
```

図 5　イベントデータの暗号化指定

暗号アルゴリズムは、暗号処理の性能と暗号の堅牢さのトレードオフを鑑みて、AES の CBC モードをサポートする。ただし、今後の暗号アルゴリズムの危殆化を考慮して暗号アルゴリズムは選択可能にする。

トランスフォームは分散実行されるため、イベントデータはサーバーをまたぐ場合がある。ネットワークでイベントデータが転送される場合も *@Encrypted* 指定されたフィールドの CEK による暗号化は維持され、転送先サーバーの enclave において必要に応じ平文化される。

ここからは、全てのフィールドが RND 暗号化されていると仮定した場合の機密性を論じる。

ストリーム処理の実行中にイベントデータが平文になるのは enclave 内だけである。それを除く REE・通信路・永続化データにおいては常に暗号化がされている。従って、strong adversary であっても暗号化されたイベントデータについて観測できるのは、原則 [6]暗号文のみである。すなわち、イベントデータには operational の機密性が担保されている。

Strong adversary, weak adversary がいかなる情報を知るかはイベントデータに対する操作次第である。例えば Cipherbase の研究 [15] において、インデックスを用いない関係代数演算では weak adversary はいかなる情報も知ることができず、strong adversary も集計演算におけるフィールドの一致性程度の情報しか知ることができないことが論じられている。本研究も同様の操作については同レベルの機密性を有する。しかし Beam Model は関係データベースのクエリと比して UDF を多用するものである。従って、攻撃者が知ることのできる情報はパイプライン開発者がトランスフォームで実現する操作に応じて様々であると考えられ、この詳細な分類は今後の課題とする。

5.3.2 ステート・チェックポイント

明示的なステートに対しては、パイプライン開発者がステー

6：ただし、enclave 内で実行されるトランスフォームが平文をステートなど通じ REE に書き出すような実装をしていた場合はこの限りではない。本研究では、平文を TEE から REE に恣意的に書き出す場合を考慮外とする。

ト定義に対して *@EncryptedState* アノテーションをすることで暗号化指定できる（図 6）。

```
PCollection<...> perUser = readPerUser();
perUser.apply(ParDo.of(new DoFn<..., ...>() {
  @EncryptedState
  @StateId("myState")
  private final StateSpec<...> numElem = ...;
  ...
}));
```

図 6　明示的ステートの暗号化指定

暗黙的なステートは、ウィンドウに入力されるイベントデータのスキーマに少なくとも 1 つの *@Encrypted* フィールドが含まれている場合に、自動的に暗号化される。

いずれの場合も暗号鍵には SEK、暗号アルゴリズムは IV を疑似乱数とした AES の CBC モードが内部的に使用される。ステートの平文は enclave の内部だけで計算される。

図 7 に、enclave 内のステートをチェックポイントに永続化する様子を示す。ステートを保有するトランスフォームが処理系の生成するチェックポイントイベント（チェックポイント ID を伴う）を受け取ると、ステートをチェックポイントに永続化する処理が進行する [5]。チェックポインティング処理は既存のストリーム処理系の実装を流用するため、大部分 REE で行う。チェックポイントイベントを受領した REE は、TEE からステートのスナップショットを取得する要求をし、TEE は SEK で暗号化した上で REE にステートを返却する。REE は暗号化されたステートを平文のチェックポイント ID を付随する形でチェックポイントに永続化する。

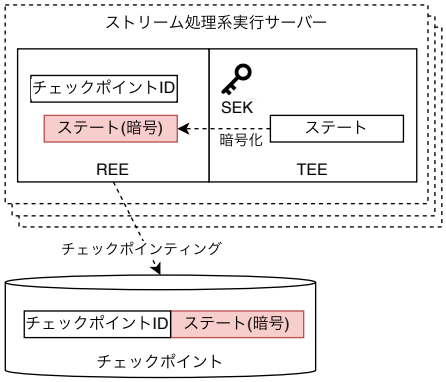

図 7　TEE でのステートの秘密計算・通信・チェックポイント

障害復旧時にチェックポイントからステートを復元する際には、REE が復元すべきチェックポイント ID を特定した後にチェックポイントから暗号文ステートを読み取り、それを enclave に送信し、enclave が SEK により平文のステートに復号する。

5.3.3　ウォーターマーク・ウィンドウ

ウォーターマークの生成・更新はストリーム処理系の内部で暗黙的に行われることが主流である。また、生成・更新されたウォーターマークは、一定の周期などを起因とし、下流のトランスフォームを実行するサーバーに伝播される [2], [37]。

1.4.2 小節で議論したように、ウォーターマークとウィンドウ端点のイベント時間軸でのタイムスタンプは機密データになり得る。

ウォーターマークの生成・更新は enclave の中で行い、SEK で暗号化（AES CBC モード; ランダム IV）した後 REE に送り、必要に応じてネットワークを介した他の REE へ伝播する。伝播された暗号ウォーターマークは enclave 内で SEK で復号される。ウォーターマークを使用する計算（例：イベントデータのイベント時刻との比較・ウィンドウのクローズ判定）は、必要に応じて引数を enclave に送信し、enclave 内で実行する。

ウィンドウの生成（イベントデータのイベント時刻に応じて、割り当てるべきウィンドウが未だ存在しなければ生成する）は enclave の中で行うで、REE がウィンドウ端点のタイムスタンプを観測できないようにする。ウィンドウ内のステートを結合・集約計算などする際は、5.3.2 小節の記載に従って enclave 内で暗号化されたステート（分散配置され得る）を特定のサーバーの enclave に転送し、その enclave で計算を実行する。

6　結　　論

本稿では、strong adversary, weak adversary の脅威に対抗し、分散ストリーム処理のソース〜シンク間のデータへ operational 機密性を提供する手法を論じた。保護対象となるデータはイベントデータのみならず、明示的または暗黙的なステート・チェックポイント・ウォーターマーク・ウィンドウを含む。暗号鍵には CMK, CEK あるいは SMK, SEK の二重構造を採用し、CMK のアクセス権限制御によるイベントデータのフィールド単位でのアクセス権限を実現する方法を示した。実用化を念頭に入れ、UDF を含む Beam Model を対象のプログラミングモデルとした。

6.1　今後の課題

本稿に記載した設計に基づく実装が大きな課題の一つであり、現時点では以下のようなチャレンジがあると考える。(1) 分散環境で TEE を用いた低レイテンシ志向のストリーム処理を実現できるか、(2) ウォーターマーク・ウィンドウ・ステートを TEE で計算するに当たり、既存のストリーム処理系の内部実装やパイプライン開発者への API への影響を局所化できるか、(3)UDF 含むトランスフォームの実装に対し、セキュリティへの考慮から Beam SDK の課す以上の制約を設ける必要があるか、(4) オートスケール時に enclave 内に CEK を安全に配布できるか。

本稿では passive adversary の一部である strong adversary と weak adversary を脅威モデルとしたが、active adversary への対応も重要な課題である。StreamBox-TZ [12] では頻繁な RA により active adversary への対応を図っていたが、分散環境に同手法を適用した場合の性能への影響は自明ではない。かつ、active adversary が改竄・破壊し得る箇所の同定とその際のシステムとしてのリカバリの議論は、先行研究 [12], [16], [17]

においても不十分であると見受けられ、より精緻な議論を目指すべきと考える。

AMD SEV に代表されるような VM 型の TEE でストリーム処理の機密性と性能を両立できるならば、本稿で論じた設計に基づく実装よりも複雑性が大きく低減し、実用性も向上すると考えられる。この点についても今後の調査・評価が必要と考える。

文　献

[1] 経済産業省. 令和元年度高度な自動走行システムの社会実装に向けた研究開発・実証事業 自動運転が活用されうるコネクテッド技術・商用モビリティサービスに関する国内外動向調査 調査報告書. `https://www.meti.go.jp/meti_lib/report/2019FY/000328.pdf`, Mar 2020. (Accessed on 12/24/2022).

[2] Tyler Akidau, Slava Chernyak, and Reuven Lax. *Streaming Systems*. O'Reilly Media, Sebastopol, CA, July 2018.

[3] Apache Software Foundation. Apache Beam Programming Guide. `https://beam.apache.org/documentation/programming-guide/`, 2022. (Accessed on 12/19/2022).

[4] Craig Chambers, Ashish Raniwala, Frances Perry, Stephen Adams, Robert R Henry, Robert Bradshaw, and Nathan Weizenbaum. FlumeJava: Easy, Efficient Data-Parallel Pipelines. *ACM Sigplan Notices*, Vol. 45, No. 6, pp. 363–375, 2010.

[5] Josh Fischer and Ning Wang. *Grokking Streaming Systems: Real-time event processing*. Simon and Schuster, April 2022.

[6] Tyler Akidau, Alex Balikov, Kaya Bekiroğlu, Slava Chernyak, Josh Haberman, Reuven Lax, Sam McVeety, Daniel Mills, Paul Nordstrom, and Sam Whittle. MillWheel: Fault-Tolerant Stream Processing at Internet Scale. *Proceedings VLDB Endowment*, Vol. 6, No. 11, pp. 1033–1044, August 2013.

[7] Arne J. Berre. *Big Data in Bioeconomy: Results from the European DataBio Project*. Springer, Germany, 2021.

[8] Victor Costan and Srinivas Devadas. Intel SGX Explained. *Cryptology ePrint Archive*, 2016.

[9] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, HASP 2016, New York, NY, USA, 2016. Association for Computing Machinery.

[10] Sandro Pinto and Nuno Santos. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.*, Vol. 51, No. 6, jan 2019.

[11] Aurélien Havet, Rafael Pires, Pascal Felber, Marcelo Pasin, Romain Rouvoy, and Valerio Schiavoni. SecureStreams: A Reactive Middleware Framework for Secure Data Stream Processing. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, DEBS '17, pp. 124–133, New York, NY, USA, June 2017. Association for Computing Machinery.

[12] Heejin Park, Shuang Zhai, Long Lu, and Felix Xiaozhu Lin. StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 537–554, 2019.

[13] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, Vol. 51, No. 1, p. 107–113, jan 2008.

[14] Hongyu Miao, Heejin Park, Myeongjae Jeon, Gennady Pekhimenko, Kathryn S. McKinley, and Felix Xiaozhu Lin. StreamBox: Modern Stream Processing on a Multicore Machine. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '17, p. 617–629, USA, 2017. USENIX Association.

[15] Arvind Arasu, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, and Ravi Ramamurthy. Transaction Processing on Confidential Data using Cipherbase. In *2015 IEEE 31st International Conference on Data Engineering*, pp. 435–446, April 2015.

[16] Panagiotis Antonopoulos, Arvind Arasu, Kunal D Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, Ravi Ramamurthy, Jakub Szymaszek, Jeffrey Trimmer, Kapil Vaswani, Ramarathnam Venkatesan, and Mike Zwilling. Azure SQL Database Always Encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, pp. 1511–1525, New York, NY, USA, June 2020. Association for Computing Machinery.

[17] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. Building Enclave-Native Storage Engines for Practical Encrypted Databases. *Proceedings VLDB Endowment*, Vol. 14, No. 6, pp. 1019–1032, April 2021.

[18] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. Speicher: Securing LSM-Based Key-Value Stores Using Shielded Execution. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies*, FAST'19, p. 173–190, USA, 2019. USENIX Association.

[19] Saba Eskandarian and Matei Zaharia. ObliDB: Oblivious Query Processing for Secure Databases. *Proc. VLDB Endow.*, Vol. 13, No. 2, p. 169–183, oct 2019.

[20] Taehoon Kim, Joongun Park, Jaewook Woo, Seungheun Jeon, and Jaehyuk Huh. ShieldStore: Shielded In-Memory Key-Value Storage with SGX. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, New York, NY, USA, 2019. Association for Computing Machinery.

[21] Christian Priebe, Kapil Vaswani, and Manuel Costa. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 264–278, May 2018.

[22] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. StealthDB: a Scalable Encrypted Database with Full SQL Query Support, 2019.

[23] Wenting Zheng, Ankur Dave, Jethro G. Beekman, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, p. 283–298, USA, 2017. USENIX Association.

[24] Lukas Burkhalter, Anwar Hithnawi, Alexander Viand, Hossein Shafagh, and Sylvia Ratnasamy. TimeCrypt: Encrypted data stream processing at scale with cryptographic access control. November 2018.

[25] Julian James Stephen, Savvas Savvides, Vinaitheerthan Sundaram, Masoud Saeida Ardekani, and Patrick Eugster. STYX: Stream Processing with Trustworthy Cloud-based Execution. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, pp. 348–360, New York, NY, USA, October 2016. Association for Computing Machinery.

[26] AMD Sev-Snp. Strengthening vm isolation with integrity protection and more. *White Paper, January*, p. 8, 2020.

[27] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. A Comparison Study of Intel SGX and AMD Memory Encryption Technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, No. Article 9 in HASP '18, pp. 1–8, New York, NY, USA, June 2018. Association for Computing Machinery.

[28] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pp. 1087–1099, New York, NY, USA, November 2019. Association for Computing Machinery.

[29] Chia-Che Tsai, Donald E. Porter, and Mona Vij. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '17, p. 645–658, USA, 2017. USENIX Association.

[30] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, p. 689–703, USA, 2016. USENIX Association.

[31] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.*, Vol. 33, No. 3, aug 2015.

[32] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 2004.

[33] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A Survey of Published Attacks on Intel SGX, 2020.

[34] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. CacheOut: Leaking Data on Intel CPUs via Cache Evictions, 2020.

[35] Yangchun Fu, Erick Bauman, Raul Quinonez, and Zhiqiang Lin. SGX-LAPD: Thwarting Controlled Side Channel Attacks via Enclave Verifiable Page Faults. In *Research in Attacks, Intrusions, and Defenses*, pp. 357–380. Springer International Publishing, 2017.

[36] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *Proceedings 2017 Network and Distributed System Security Symposium*, Reston, VA, 2017. Internet Society.

[37] Tyler Akidau, Edmon Begoli, Slava Chernyak, Fabian Hueske, Kathryn Knight, Kenneth Knowles, Daniel Mills, and Dan Sotolongo. Watermarks in Stream Processing Systems: Semantics and Comparative Analysis of Apache Flink and Google Cloud Dataflow. *Proceedings VLDB Endowment*, Vol. 14, No. 12, pp. 3135–3147, October 2021.