情報処理学会論文誌　デジタルプラクティス　Vol.5　No.1　11–22 (Jan. 2024)

TEE を用いた関数型暗号による秘匿配送マッチングの実現

古家 直樹[1,2,a)]　矢内 直人[1]　藤原 融[1,3]

受付日 2023年2月13日, 採録日 2023年10月20日

概要：物流業界の配送マッチングサービスでは，荷主の追加オーダをトラックのルートの途中に割り当てる．追加オーダとルートには，出発地・到着地の所在地などの機微情報が含まれることから，マッチングサービスの提供者に対して追加オーダとルートを秘匿する秘匿配送マッチングの実現が望まれる．秘密計算の手法には準同型暗号等があるが各手法とも処理速度が課題である．そこで本研究では，処理時間の課題に対して，鍵生成者が秘密鍵に演算を埋め込める関数型暗号を TEE（Trusted Execution Environment）によって実現し，秘匿配送マッチングに適用する．TEE として Intel SGX を用いて処理時間と秘匿性の評価したところ，トラックのルートが 289 件含まれるマッチングを約 2.5 秒で実施でき，TEE による関数型暗号の有効性を確認した．

キーワード：関数型暗号，TEE，Intel SGX，秘密計算，配送マッチング

Realization of Confidential Delivery Matching by Functional Encryption Using TEE

Naoki Furuya[1,2,a)]　Naoto Yanai[1]　Toru Fujiwara[1,3]

Received: February 13, 2023, Accepted: October 20, 2023

Abstract: A delivery matching service in logistics industry allocates a shipper's additional order to a truck route. Since additional orders and routes contain sensitive information such as the locations of departure and arrival points, it is desirable to realize confidential delivery matching that conceals additional orders and routes from matching service providers. There are various methods of secure computing such as homomorphic encryption, but each method has a problem of processing speed. To solve the problem of processing time, we implement functional encryption in which the key generator can embed operations in the secret key by TEE (Trusted Execution Environment), and apply it to confidential delivery matching. We evaluate the processing time and confidentiality using Intel SGX as TEE. A Matching including 289 truck routes is performed in about 2.5 seconds, confirming its effectiveness.

Keywords: functional encryption, TEE, Intel SGX, secure computing, delivery matching

1. はじめに

物流業界では，荷主がトラックを直接手配できない場合などに，サービス提供者がトラックを手配する配送マッチングサービス（以下，マッチングサービスと称す）が行われている．一般的なマッチングサービスでは，マッチング候補のトラックのルートの情報をあらかじめ収集しておき，荷主から届いた追加オーダの配送元・配送先の情報とルートの情報をもとに，適切なトラックにその追加オーダを割り当てる．荷主の追加オーダとトラックのルートには，出発地・到着地の所在地などの機微情報が含まれるので，サービス提供者に対して追加オーダとルートを秘匿することが望まれる．

本研究では秘密計算を用いてサービス提供者に対して追加オーダとルートの中身を秘匿した状態でマッチングを行う秘匿配送マッチングに取り組む．国内大手のマッチングサービス業者と同等のマッチング件数を実現しようとした場合，2.5 節で述べるように 289 件のトラックのルートが含まれる 1 回のマッチングを 3.6 秒で処理する必要がある．秘密計算の手法として一般的な準同型暗号ではこの制約を満たすのは困難である．

一方で秘密計算を高速化する方法として，TEE（Trust-

[1] 大阪大学
Osaka University, Suita, Osaka 565–0871, Japan
[2] （株）日立製作所
Hitachi Ltd., Yokohama, Kanagawa 244–0817, Japan
[3] 島根大学
Shimane University, Matsue, Shimane 690–8504, Japan
[a)] n-furuya@ist.osaka-u.ac.jp

ed Execution Environment）によって秘密計算の一手法である関数型暗号（Functional Encryption）を実現する方法が知られている [1]．ここで TEE とは，ユーザが安全にプログラムを実行できる保護領域のことを意味し，例として ARM の Trust Zone[*1] や Intel[*2] の SGX が知られている．また関数型暗号は鍵生成者（Key Manager）が秘密鍵に演算を埋め込める暗号化方式であり，公開鍵で暗号化した暗号文に秘密鍵を適用すると元の平文の代わりに演算結果が取得できる．

しかし TEE による関数型暗号においても，TEE 内でルートと追加オーダを平文に復号する処理などのオーバヘッドが懸念される．そこで本研究では，秘匿配送マッチングに TEE による関数型暗号を適用した場合の処理時間を明らかにするとともに，サービス提供者への秘匿性が満たせるか検証することを目的とする．

以降，2 章で秘匿配送マッチングの課題と関連研究，3 章で TEE による関数型暗号の秘匿配送マッチングの処理内容，4 章で評価環境の実装と評価，5 章で得られた知見，6 章でまとめを述べる．

2. 課題と関連研究

本章では，本研究で対象とする配送マッチング問題および秘匿配送マッチングの処理概要を述べ，課題と関連研究を説明する．

2.1 配送マッチング問題

マッチングサービスにおいて，複数のトラックのルートと 1 つの追加オーダに対して，その追加オーダをどのルートに割り当てるかを求める問題を配送マッチング問題と定義する．マッチングサービスでは，サービス提供者がルートと追加オーダをもとに配送マッチング問題をマッチング PF（Platform）上で解いて，追加オーダの割り当て先のトラックを決める．

ここでルートはマッチングを行う前にドライバが属する運送会社等が事前に決定しているが，そこに荷主からの追加の配送依頼（追加オーダ）を割り当てる．トラックは 1 回の配送業務で最初の出発地（起点と称す）から最終到着地（終点と称す）まで，途中の各地点で荷物の荷積み・荷降ろしを行いながら向かう．ルートはトラックの配送業務ごとに作成され，トラックが立ち寄る複数の地点が順番に記載される．たとえば「起点→地点 1→地点 2→終点」と記載する．また各地点には緯度・経度が紐づいており，ルートは地点を節点，地点間を結ぶ直線を辺とするグラフとみなすことができる．追加オーダには荷主がトラックに運んで欲しい荷物の出発地（発送元）と到着地（配送先）が記載されている．

2.2 割り当て先のトラックの決定方法

マッチングサービスのサービス提供者は以下の条件を加味して，追加オーダの割り当て先のトラックを説明する．なお，荷物の積載量や各地点への到着時間の制約も加味することも可能だが，本稿では対象外とする．

条件 1. トラックは追加オーダ中の出発地から到着地に直接向かい，途中でルートに含まれる他の地点には立ち寄らない

条件 2. 追加オーダを割り当てた場合の追加距離が最小となるルートの辺に割り当てる

条件 1 について**図 1** を用いて説明する．図 1 は 2 つのトラックが存在する場合に，地点 7 から 8 へ配送する追加オーダを割り当てる配送マッチング問題の例を示している．条件 1 はトラックが地点 7 から 8 に荷物を運ぶ途中に，そのトラックが立ち寄る予定のルートに含まれる他の地点（例．トラック 1 であれば地点 1，2，3）に立ち寄らないことを意味する．これは，途中で他の地点に寄って別の荷物の荷積み・荷降ろしがあると，追加オーダの出発地（地点 7）で積んだ荷物がトラックの荷台の奥に行ってしまい到着地（地点 8）における荷降ろしがしづらくなる，また別の地点での荷降ろし時に出発地（地点 7）で積んだ荷物が荷降ろし作業を阻害するといった懸念があるためである．

条件 2 について説明する．一般的な配送計画問題では，ドライバの労働時間やトラックの燃料代が過多とならないように，走行距離が最小となるようにルートを策定する [2]．配送マッチング問題においても，マッチングで割り当てた追加配送がドライバの負担とならないよう，追加距離最小となるルートの辺に追加オーダを割り当てる．なお追加距離は地図上の経路に沿った距離で評価することも考えられるが，経路に沿った距離がすぐに得られない場合には 2 地点（(x_1, y_1), (x_2, y_2)）間の直線（ユークリッド）

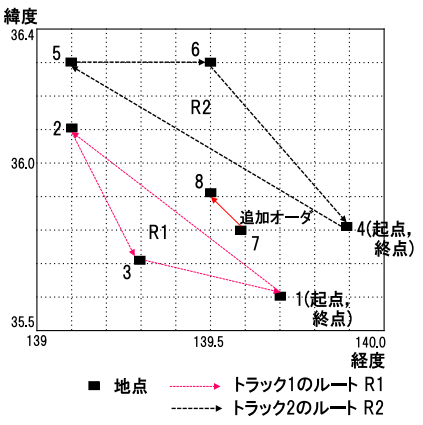

図 1　配送マッチング問題の例
Fig. 1　Example of delivery matching problem.

[*1] ARM 社の商標である．
[*2] Intel Corporation またはその子会社の商標である．

距離（sqrt{(x2-x1)^2+(y2-y1)^2}），または碁盤目状に道路が有る場合にはマンハッタン距離（|x2-x1|+|y2-y1|）で評価することが一般的であり，文献 [2] でも直線距離を用いている．そこで本研究でも追加距離は直線距離で評価するが，マンハッタン距離に変更することも容易である．

ルートの地点 i から地点 j の辺の途中に出発地 k から到着地 l の追加オーダを割り当てた場合の追加距離 L は，

L = 地点 i と出発地 k の直線（ユークリッド）距離 +
　出発地 k と到着地 l の直線距離 +
　到着地 l と地点 j の直線距離 −
　地点 i と地点 j の直線距離

と定義される．

2.3　配送マッチングで用いるデータと処理内容

配送マッチング問題で用いるトラックのルートの内容を**表 1**，荷主の追加オーダの内容を**表 2** にそれぞれ示す．ルート，追加オーダとも 1 回の配送（辺）の出発地・到着地をあらかじめ各トラックおよび荷主の端末で保持している住所情報のテーブルに登録されている場所 ID を用いて指定する．またトラックのルートには複数の辺が存在する．次に，各トラック・荷主が用いる住所情報を**表 3** に示す．住所情報は緯度・経度と住所（地番）が登録されており，トラック・荷主ごとに自身の端末で保持し，マッチング PF や他者には共有されない．配送マッチングでは，ルートと追加オーダ中の出発地・到着地の地点 ID を緯度・経度に変換してマッチング PF に送信し，同 PF 上で割り当て先のトラックを決定する．

表 1　ルートの内容
Table 1　Contents of route.

ルート ID	辺 ID	出発地	到着地
1	1	1	2
1	2	2	3
1	3	3	1

表 2　追加オーダの内容
Table 2　Contents of order.

オーダ ID	出発地	到着地
1	7	8

表 3　住所情報
Table 3　Address information.

地点 ID	緯度(°)	経度(°)	住所
1	35.6	139.7	横浜市 xx 区 xx 町 292
2	36.1	139.1	相模原市 xx 区 xx 町 12
…	…	…	…

マッチング PF は各トラックのルートを 1 日の配送が始まる前にあらかじめ入手しており，トラックはマッチング PF にルートを毎日決められた時刻までに送ることを想定している．トラックが最初の出発地を出発する前にマッチングが行われ，追加オーダがマッチング PF に届くとマッチング PF に揃っているルートを用いてマッチング処理が始まる．マッチング PF は各トラックのルートのどの辺に割り当てたら追加距離最小となるかを探索し，追加オーダの割り当て先を決める．たとえば図 1 の場合，2 つのルート R1，R2 にはそれぞれ 3 つの辺（例．R1 であれば地点 1→2，2→3, 3→1）が存在する．このそれぞれの辺に対し追加オーダを割り当てた際の追加距離が最小となる辺に追加オーダを割り当てる．ここで，トラックや荷主がマッチング結果を承諾しなかった場合は，そのトラック以外のトラックのルートで最も追加距離が短いルートの辺を割り当て先とする．

2.4　秘匿配送マッチング

荷主とトラックにとって，マッチング PF を運営するサービス提供者に出発地・到着地の緯度・経度が分かると，web 等で検索すれば取引先企業名が分かり望ましくない．そこで本研究では，出発地・到着地の緯度・経度をマッチング PF に秘匿した状態でマッチングを行う秘匿配送マッチングに取り組む．**図 2** に秘匿配送マッチングの概要を示す．以降，秘匿配送マッチングを実施するマッチング PF を秘匿配送マッチング PF と称す．トラックと荷主のクライアント端末，および秘匿配送マッチング PF とクライアント端末間の通信路のセキュリティについては既存の方法で担保するものと考え，本研究の対象外とする．

2.5　処理時間の目標

配送マッチングの国内大手の 1 社である T 社では日々 6,000 件（2021 年度）のマッチングを成立させており，日々 13,000 社の運送会社のルートを 45 拠点で扱っている [3]．なお T 社では，秘匿配送マッチングのように追加オーダとルートを秘匿した状態でのマッチングは行っていない．本研究では追加オーダとルートを秘匿した状態で T 社と同等のマッチング件数をめざすものとする．ただし，

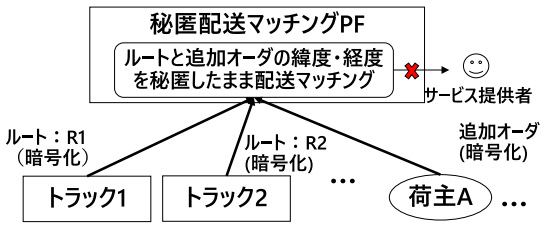

図 2　秘匿配送マッチングの概要
Fig. 2　Outline of the confidential delivery matching.

処理時間の評価対象は秘匿配送マッチング PF がルートと追加オーダを受信して割り当て先のトラックを導出するまでの時間とする．ここで，マッチングサービスは荷主が翌日のトラックを手配できない場合に利用されることが多く，午後（13–17 時）に利用が集中すると考えられる．仮に 2/3 の 4,000 件がこの 4 時間に集中した場合，1 時間あたり 1,000 件（1 回あたり 3.6 秒以内）のマッチングを成立させる必要がある．

次に 1 回のマッチングで取り扱うルート数について説明する．T 社では一日 13,000 件の空車情報を扱っており，1 件の空車情報にトラック 1 台のルート情報が含まれる．T 社と同様に 45 拠点で分けてマッチングを行うものとすると，1 回のマッチングでは，13,000 / 45 = 289 件のルートを扱う想定となる．以上から T 社と同規模のサービスを 1 台のサーバで実現するものとし，289 件のルートが含まれるマッチングを 1 回あたり 3.6 秒以内に成立することを目標とする．

2.6　秘匿配送マッチング実現の課題

秘匿配送マッチングにおいては秘匿配送マッチング PF 上で暗号化されたルートと追加オーダから，追加オーダをルートの各辺に割り当てた際の追加距離を計算し，その最小値を取得する．データを暗号化した状態で計算する技術は，一般に秘密計算と呼ばれる．秘密計算の手法の例として，暗号文のままで線形演算と乗算が可能な完全準同型暗号（Homomorphic Encryption：HE）がある．しかし処理時間の課題があることが知られている [4].

たとえばライブラリとして Microsoft の SEAL を用いて**表 4** に示すクラウド環境上で秘匿配送マッチングの処理を実装した場合，4 つの辺が含まれる 100 件のルートに対して行った場合の処理時間を計測したところ，1 回のマッチング（追加距離の算出と最小追加距離の算出）に約 42 秒掛かることが分かった．ここで SEAL では準同型暗号のベースとなる格子暗号の方式として，実数を整数に見立てて計算する CKKS（Cheon, Kim, Kim, Song）方式を用いた．

表 4　準同型暗号による実装環境
Table 4　Implementation environment using HE.

項目		仕様
準同型暗号ライブラリ		Microsoft SEAL v3.5.6 CKKS 方式
言語		C++
クラウド環境	vOS	Ubuntu16.04LTS
	CPU	Intel Core Processor (Skylake, IBRS) 2GHz
	RAM	16GB
	Disk	42GB

そのため，289 件のルートが含まれるマッチングを 3.6 秒以内に成立することは実現困難である．また他の準同型暗号のライブラリでも処理速度は同等である [4]．準同型暗号よりも高速な秘密計算の手法として秘密分散があるが，ユークリッド距離計算における乗算の処理などにおいてサーバ間の通信のオーバヘッドが発生し，やはり処理速度が課題になると考えられる [4].

そこで，ハードウェア的に高速な秘密計算が実施可能である TEE を用いて秘匿配送マッチングシステムを構築し，処理時間およびデータの秘匿性を検証する．ここで処理時間については，TEE を用いた場合には TEE の内部でルートと追加オーダを平文に復号してからマッチング処理を行うが，その復号に掛かる処理時間などのオーバヘッドが懸念されるため，2.5 節で述べた処理時間の制約を満たせるかは不明という課題がある．

2.7　関連研究

本研究と同様に，暗号化された状態でユークリッド距離計算を行う研究として，中西らは準同型暗号を用いた顔認証基盤を開発している [5]．認証処理で行われる画像間の類似度算出の際に，準同型暗号でユークリッド距離計算を行うが，この処理時間の高速化が必要と述べている．

このようなソフトウェアによる秘密計算に対して，近年の CPU で提供されている TEE を用いたハードウェアによる秘密計算の実現も行われている [6]．Fisch らは TEE として SGX を用いて，関数型暗号を実現する IRON を提案している [1]．TEE を用いない一般的な関数型暗号では処理速度の課題があるが，TEE を用いることで従来の関数型暗号よりも数桁高速な処理が期待できると述べている．

ここで関数型暗号は鍵生成者が秘密鍵に任意の演算や処理を埋め込める暗号化方式であり，データ（平文）を公開鍵で暗号化した後，暗号文に秘密鍵を適用すると，元の平文は秘密鍵の利用者に秘匿しつつ任意の演算・処理結果を得ることができる [7]．また複数の暗号文を入力として秘密鍵を施すと演算結果が得られる多入力関数型暗号も存在する [8]．IRON では鍵生成者が生成した秘密鍵をサーバ管理者が取得できない方法で TEE に送付し，TEE の中で暗号文を秘密鍵で平文に復号して演算を行い，演算結果を TEE の外に出力する．また鍵生成者が TEE 内の処理で一意に決まるハッシュ値を認証することで，鍵生成者が許容した処理以外は TEE 内で実施できない．これにより関数型暗号に該当する処理を TEE で実現している．

次に，加納らは IRON を拡張し，一定時間が経過するまで復号できないことを保証した関数型暗号である，関数型タイムリリース暗号を提案している [9]．また Bhatotia らは，IRON を拡張しシステムの内部状態に依存し，かつ

ランダム化された関数型暗号を実現した Steel を提案している [10]．[9], [10] とも IRON を拡張して関数型暗号で取り扱い可能な関数の種別を拡大することを目的としているが，本研究の配送マッチングのように具体的な適用先の問題（アプリケーション）に対して適用評価を行ったものではなく，[9], [10] で開発した暗号が実用的であるかの評価が課題である．

岩田らは，SGX を用いて個人のゲノムデータをサーバに秘匿して解析を行う手法を提案している [11]．ここで，SGX ではメモリサイズの制限があることから，分割したゲノム情報の暗号文を高速かつ安全に TEE 内に展開するためにデータの牽引化とパス型 ORAM [12]（Oblivious RAM：サーバからアクセスパタンを秘匿しつつランダムアクセスを行うための暗号学的技術）を用いている．また Mandal らは，同じくゲノムデータの χ 二乗分布の解析を SGX で行っており，サーバにゲノムデータを秘匿するとともに ORAM を用いることでメモリアクセスパタンの秘匿も図っている [13]．最後に，Felsen らは 2 つのパーティがお互いの入力値は明かさずに公開された関数の出力値を得る Secure Function Evaluation，および片方のパーティが関数を明らかにせずにもう片方のパーティからの秘匿化された入力値に対する結果を得る Private Function Evaluation を，SGX で実現している [14]．また処理時間を評価し，処理が複雑になっても Yao のガーブル回路 [15] と GMW（Goldreich, Micali, and Wigderson）[16] のプロトコルに比べて高速であることを述べている．

[11], [13], [14] とも入力データを暗号化して SGX に送り，平文で処理を行う点は本研究と共通であるが，関数型暗号で考慮される鍵生成者が不在なため，仮に TEE 内の処理をサービス提供者（サーバ管理者）が書き換えた場合 TEE 内の復号鍵を入手できることを考慮していない課題がある．

3. 関数型暗号による秘匿配送マッチング

本章では，TEE による関数型暗号を用いた秘匿配送マッチングの処理内容について説明する．

3.1 TEE による関数型暗号

Fisch らは TEE として SGX を用いてハードウェア的に関数型暗号を実現することで，TEE の外に入力値を秘匿した状態で任意の関数を高速に処理し，その結果を取得できる IRON と呼ばれるシステムを開発している [1]．そこで本研究では IRON と同様に TEE として SGX を用いて関数型暗号を実現し，秘匿配送マッチングに適用する．SGX では主記憶装置上に Enclave と呼ばれる保護領域を形成し，その内部で秘密情報を保持しながらプログラムを実行できる．

IRON では鍵生成者が SGX に復号用秘密鍵を渡す際に，Remote Attestation と呼ばれる Enclave で実行するバイナリが意図したものであるか外部から確認することができる機能を用いる．このように Key Manager は単に鍵を生成するのみならず，Enclave 内の処理の妥当性を確認する役割を担う．

また，IRON では関数型暗号の処理を行う PF である Decryption Node PF において，関数型暗号の演算を行う Function Enclave（以降，FE と称す）と，鍵生成者から秘密鍵を受け取り，FE を認証した後に秘密鍵を提供する Decryption Enclave（以降，DE と称す）の 2 つの Enclave から構成される．DE が FE を認証する際は，同一 CPU 上の Enclave 間の認証である Local Attestation の機能を用いる．Local Attestation，Remote Attestation ともに，Enclave を認証する際には SGX が Enclave プログラムの生成（ビルド）時に Enclave プログラムから計算するハッシュ値 mrenclave が用いられる．mrenclave は Enclave プログラムに依存する．

次に関数型暗号では，秘密鍵の利用者が秘密鍵で暗号文を復号する際に，任意の演算結果を得ることができるものであり，複数の暗号文を入力として秘密鍵を施すと演算結果が得られる多入力関数型暗号も存在する [8]．本研究では，複数のルートと 1 つの追加オーダの暗号文から追加距離最小となる割り当て先のルートの辺を求める多入力関数型暗号に該当し，サービス提供者は演算結果である割り当て先のルートの辺のみを取得する．

3.2 秘匿配送マッチングシステムの全体像

図 3 を用いて，TEE による関数型暗号秘匿配送マッチングシステムの全体像を説明する．IRON では任意の関数型暗号の処理（図中の関数 f）を扱えるように設計されているが，本研究では関数 f は秘匿配送マッチングに特化し，追加距離計算と割り当て先トラック決定を行う．図 3

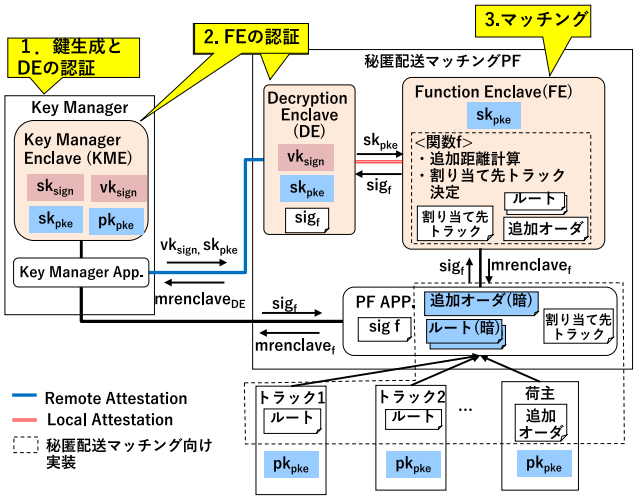

図 3　秘匿配送マッチングシステムの全体像

Fig. 3　Whole picture of secret delivery matching system.

で，秘匿配送マッチング向けに実装した箇所を点線で示す．図 3 における各略号の意味を以下に示す．

- DE：Decryption Enclave の略
- FE：Function Enclave の略
- KME：Key Manager Enclave の略
- PF App.：秘匿配送マッチング PF の Enclave 外のアプリ
- Key Manager App.：Key Manager の Enclave 外のアプリ
- (pk_{pke}, sk_{pke})：ルート・追加オーダの暗号用公開鍵，復号用秘密鍵
- 関数 f：FE で行う処理の関数
- (vk_{sign}, sk_{sign})：関数 f の署名検証鍵，署名用秘密鍵
- $mrenclave_f$：FE から生成されるハッシュ値 mrenclave
- $mrenclave_{DE}$：DE から生成されるハッシュ値 mrenclave
- sig_f：$mrenclave_f$ に対する電子署名

関数型暗号による秘匿配送マッチングシステムは，1. 鍵生成と DE の認証，2.FE の認証，3. マッチングの 3 つの処理からなる．これらの処理を構成する要素として秘匿配送マッチング PF と Key Manager（鍵生成者）が定義される．秘匿配送マッチング PF には DE と FE の 2 つの Enclave，および PF App. が内部モジュールとして存在する．また，Key Manager には内部モジュールとして KME と Key Manager App. が存在する．ここで Key Manager は IRON では Trusted Authority の役割を兼ねるが，本研究でも同様に Trusted Authority を兼ねるものとし，秘匿配送マッチング PF を運営するサービス提供者の不正を監視する．Key Manager はサービス提供者とは独立した立場の第三者認証機関が担当する．

ここで，DE の認証は Key Manager が Remote Attestation（以降，R.A. と称す）を用いて行い，通常の R.A. に加えて，Key Manager が DE の mrenclave 値である $mrenclave_{DE}$ の正解値をあらかじめ保持しておき，R.A. で DE から送付される $mrenclave_{DE}$ が毎回同じであるかを確認することで，DE 内のソースコードの改変が無いかを確認する．また，FE の認証は，DE が FE の中の処理に改ざんがないことを認証する処理である．以降，各処理とその構成要素を説明する．

1.　鍵生成と DE の認証

Key Manager は (pk_{pke}, sk_{pke})，(vk_{sign}, sk_{sign}) を KME 内で生成する．また，Key Manager App. は DE からの R.A. を受けて，DE を認証後に署名検証鍵 vk_{sign} とルート・追加オーダの復号用秘密鍵 sk_{pke} を提供する．その際，KME にて DE の mrenclave 値（$mrenclave_{DE}$）が，KME はあらかじめ保持している正解値の $mrenclave_{DE}$ の値と比較して一致検証を行う．これにより，第三者による秘匿配送マッチング PF の成りすまし，サービス提供者や第三者の DE の改ざんを抑止する．ここで R.A. には Enclave の認証の際に，Intel 社の IAS（Intel Attestation Service）と呼ばれるサーバに問い合わせる方式である EPID（Enhanced Privacy ID）方式 [17] を用いる．

2.　FE の認証

Key Manager App. は PF App. から FE の処理内容（関数 f）のハッシュ値である $mrenclave_f$ を受けて KME に送り，KME にて $mrenclave_f$ に sk_{sign} で署名を行い，その結果である sig_f を PF App. に送る．PF App. は，sig_f を取得後 FE に送信する．

次に，FE は DE に対して Local Attestation を要求し，sig_f を DE に提供する．DE は署名検証鍵 vk_{sign} で sig_f を復号して得られる $mlenclave_f$ の値が Local Attestation（以降，L.A. と称す）で得られる $mlenclave_f$ の値と一致するか検証し，検証結果が True であれば FE に復号用秘密鍵 sk_{pke} を提供する．これにより FE の処理内容が Key Manager が署名した処理から改ざんがなされていないことを担保する．

ここで，FE と DE を分離した理由を説明する．KME では $mrenclave_{DE}$ の値（正解値）を保持しているが，KME の正解値の書き換えが頻繁に起きるのは管理上望ましくない．ここで，DE は FE に比べて処理内容の変更が少ないと考えられる．FE と DE を分離すれば，KME で保持するのは書き換え頻度の少ない DE の mrenclave で良く，分離しない場合と比べて正解値の書き換え頻度を減らすことが可能となる．

3.　マッチング処理

トラックと荷主は PF App. から公開鍵 pk_{pke} を取得し，それぞれルート・追加オーダを暗号化して PF App. に送る．PF App. は FE に暗号化されたルート・追加オーダを送り，FE は sk_{pke} にてルートと追加オーダを平文に戻し，マッチング処理を行う．FE はマッチング処理結果を PF App. に通知する．

3.3　鍵生成と DE の認証の処理詳細

まず，Key Manager が DE を認証して sk_{pke} と vk_{sign} を送付するまでの手順について，**図 4** を用いて説明する．

(1)KME は鍵生成を行う

(2)DE は R.A. を Key Manager App に要求

(3)sk_{pke} と vk_{sign} を DE に配布

sk_{pke} と vk_{sign} は DE が一度受け取ったら，秘匿配送マッチング PF のメモリに Sealing（SGX 内のデータを暗号化してメモリに書き込む処理）で保持しておき，秘匿配送マッチング PF のメモリから取得するが，DE の認証は定期的（例．1 日に 1 回）に行い DE の不正を抑止する．

また DE は原則的に処理内容を変更しないものとするが，もし処理の変更が必要な場合は，Key Manager は DE から新たな $mrenclave_{DE}$ を取得し KME で保持している正

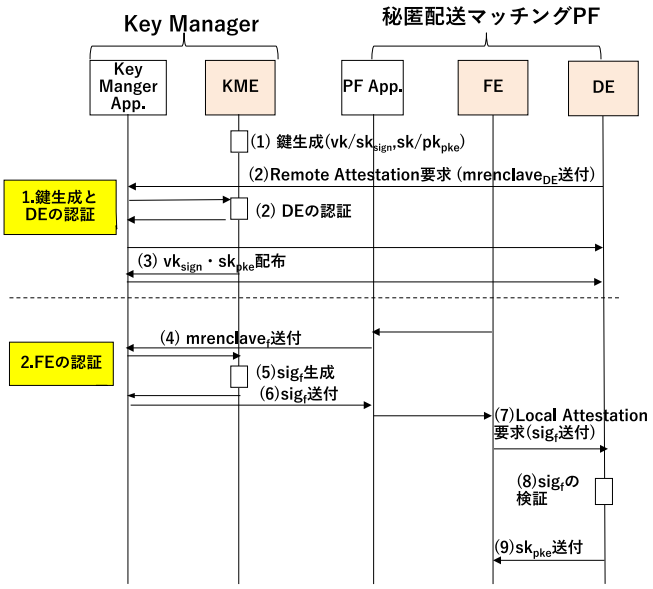

図 4　鍵生成と DE，FE の認証の処理フロー

Fig. 4　Procedure flow of key generation and DE, FE attestation.

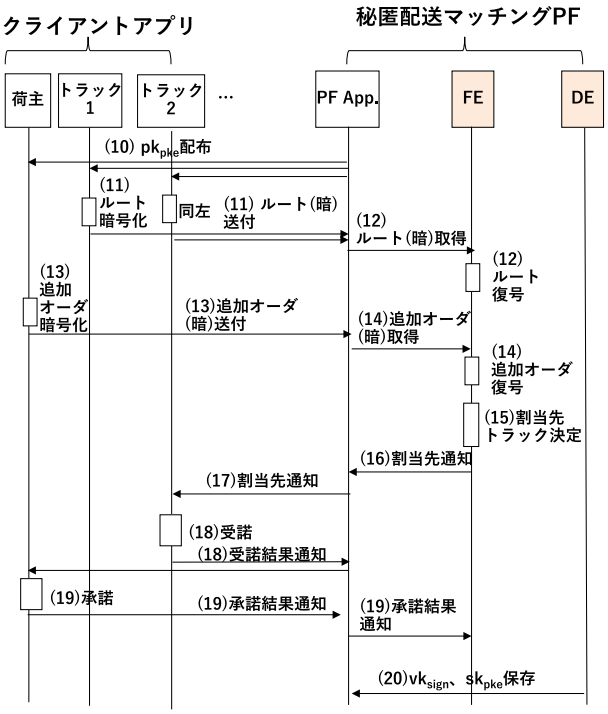

図 5　マッチングの処理フロー

Fig. 5　Procedure flow of matching.

解値の DE の mrenclave を置き替え，sk_{pke} と vk_{sign} を再発行する．

3.4　FE の認証

次に，DE が FE を認証して sk_{pke} を送付するまでの処理を同じく図 4 を用いて説明する．

(4)PF App. は $mrenclave_f$ を取得し，$mrenclave_f$ を Key Manager App. に送付

(5)KME は $mrenclave_f$ を sk_{sign} で署名して sigf を生成

(6)Key Manager App. は sig_f を PF App. に送付し，PF App は sig_f を FE に送付

(7)FE は DE に L.A. を要求し，sig_f を送付

(8)DE は sig_f を vk_{sign} で復号し，L.A. で取得した FE の mrenclave 値と一致するか検証

(9)DE は（8）の検証結果が True であれば，L.A. 成立．DE は FE に sk_{pke} に送付

（4）～（6）は初回および FE の処理内容の変更や FE の追加があった場合のみ行うものとし，sig_f は PF App. が一度 Key Manager から取得したら PF App. が保持する．一方（7）～（9）は定期的（例．1 日に 1 回）に行うことで，Key Manager が署名を行った処理のみが秘匿配送マッチング PF 上で行われるようにする．

3.5　マッチング処理の詳細

図 5 を用いてマッチングの処理の詳細を説明する．なお（17）以降の処理は将来的に秘匿配送マッチングサービス事業を実現する際には必要となるが，本研究における処理時間の評価対象には含めていない．

(10)PF App. はあらかじめ Key Manager から取得した暗号用公開鍵 pk_{pke} をクライアントに配布（初回のみ）

(11) トラックはルートを暗号化し，PF App. に送付

(12)FE は PF App. からルートを取得し復号

(13) 荷主は追加オーダを暗号化し PF App. に送付

(14)FE は PF App. から追加オーダを取得し復号

(15)FE はマッチング処理にて割り当て先トラック・辺を決定.

(16)FE は割り当て先のトラックと辺を PF App. に通知.

(17)PF App. は割り当て先のトラックに通知

(18) トラックは受諾するかを選択し，受諾結果を PF App. に通知，および PF App. が荷主に受諾結果・トラックを通知

(19) 荷主が承諾結果を PF App. に通知し，PF App. は FE に承諾結果を通知

(20)DE は Sealing で sk_{pke} と vk_{sign} を保存

（1 日の処理終了後）

ここで（12）にて PF App. はルートを SGX のメモリ制約

（第一世代の SGX では EPC（Enclave Page Cash）96 MB）の上限を超えない範囲で複数件まとめて送る．また（17）でトラックに通知する際に，あらかじめ FE で追加オーダを sk_{pke} で暗号化してトラックに送付し，トラックは pk_{pke} で追加オーダを復号して，受諾するか否かを選択する．また，（18）（19）でトラックや荷主が承諾しなかった場合は，そのトラックのルート以外で追加距離が最短の辺を割り当て先として，（17）以降の処理を繰り返す．

なお（19）で FE に承諾結果を通知するのは，一度マッチングが成立したルートは次のマッチングの割り当て候補から外すためである．

3.6　サービス提供者の不正抑止

ここで，FE を実行するサービス提供者が，FE の処理に不正を加えると秘密鍵や復号したルート，追加オーダを閲覧できる懸念がある．そこで，3.4 節で説明したように Key Manager が FE の処理内容の認証を行って sig_f を生成し，DE が FE を認証することで，秘匿配送マッチング PF では Key Manager が認証したプログラム以外は実行できないようにしている．

また Key Manager は，3.4 節の（4）〜（6）を行う際に，秘匿配送マッチング PF に $mrenclave_f$ と合わせて FE のソースコードを提出させ，FE の処理に不正（例：SGX 内で平文にしたルートと追加オーダを OCALL で PF App. に出力するなど）がないかを確認する想定である．また DE の処理についても，初回および処理内容を変更した際に Key Manager にソースコードを提出させて不正がないか確認する想定である．

4.　評価環境の実装と評価

本章では，評価環境の実装および処理時間の評価と，サービス提供者に対する秘匿性の評価について述べる．

4.1　評価環境の実装

3 章で説明した秘匿配送マッチングの評価環境の実装を行った．開発に用いた PC や開発環境等を**表 5** に示す．

評価環境では，Key Manager およびトラック，荷主は秘匿配送マッチング PF と同じ PC で実装している．また（17）以降の処理は秘匿配送マッチング PF の処理時間に与える影響はわずかと判断し評価対象外とする．（2）の R.A. については秘匿配送マッチング PF の処理とは切り分けて実装し，処理時間を評価する．

4.2　処理時間の評価方法

2.5 節で述べた，289 件のトラックのルートが含まれる配送マッチング問題を 1 回あたり 3.6 秒以内に解く目標を満たせるか確認する．また，ルート件数を SGX のメモリサイズの上限を超えない最大 5 万件まで増やした場合の処理時間を評価する．各トラックのルートには 4 つの辺が含まれるものと仮定する．これはたとえば倉庫や集配所などの出発地から 3 地点（例．朝出発して午前中 1 地点，午後 2 地点）に立ち寄って各地点で荷降ろしと荷積みを行い，出発地に戻る場合のルートに該当する．処理時間の計測には Windows の Query Performance Counter を使用した．処理時間は 10 回処理を行った平均値を用いる．

ここで SGX では Stack と Heap のサイズについて，双方のサイズの和が EPC の上限を超えない範囲で変更ができる．Intel SGX SDK で設定されている Stack と Heap の初期値はそれぞれ 0.26 MB と 1.0 MB であるが，同じルート件数の場合に Stack と Heap のサイズを変えて処理を試行しても処理時間は殆ど変わらないことが分かった．また評価環境では Heap に平文にしたルートと追加オーダを保持しているため，ルート件数を増やすと Heap の枯渇が起き，マッチング処理が行われなくなる．そこでルート件数を増やすに連れて Heap サイズを増大させて処理時間を評価する．なお，Stack は Enclave 内外のデータの授受に用いられるメモリで，FE の中にルート，オーダを送る際に使われているが，Stack サイズを変えても処理時間への影響はない．

表 5　用いた PC および開発環境
Table 5　Used PC and development environment.

項目		仕様
PC	マシン名	HP 250G7 Notebook PC
	CPU	Intel Core i3-7020U CPU, 2.30GHz
	RAM	16GB
	OS	Windows10 Home
	SGX	第一世代（EPC：96MB）
NW 環境	プロトコル	Wi-Fi 802.11n
	NW 帯域	2.4GHz
	リンク速度（送受信）	300 / 300 (Mbps)
開発環境	統合開発環境	Visual Studio Community 2017
	言語	C++
	SGX の SDK	Intel SGX SDK ver 2.15.100.4
	R.A.の種別	EPID 方式
暗号	ライブラリ	OpenSSL1.1.1m
	(pk_{pke}, sk_{pke})	RSA-2048bit with OAEP SHA-256
	(vk_{sign}, sk_{sign})	RSA-3072bit

4.3　処理時間の評価結果

評価結果を説明する．評価結果は秘匿配送マッチング PF におけるマッチング処理（暗号化されたルートと追加オーダを FE に送り，割り当て先のトラックを導出する処理）とマッチング処理よりも前に行う処理（鍵生成と FE，DE の認証およびルートと追加オーダの暗号化）に分けて説明する．

表 6 にマッチング処理よりも前に行う処理とその実施頻度を示す．R.A. に約 4 秒掛かるが，利用の少ない夜間などに行えば良い処理のため，問題ない．なお本研究と同じ EPID 方式の R.A. の処理時間は，文献 [17] では数 10 msec，文献 [18] では約 1.8 秒と述べており，[18] にて大

表 6　マッチング処理よりも前に行う処理の処理時間
Table 6　Time of processing performed before matching.

項目	処理時間 (msec)	実施頻度
鍵の生成	700	DE の処理変更時
DE の認証（R. A.）	4,065	1 日 1 回程度
sig_rの生成	4	FE の処理変更時
DE による FE の認証(L.A.)	4	1 日 1 回程度
追加オーダの暗号化	0.2	毎回のマッチング時
ルートの暗号化（1 件あたり）	4	毎回のマッチング時
合計	4,777.2	-

部分が IAS への問い合わせに要する時間と報告されている．[17], [18] の R.A. の処理時間が本研究の計測結果より短いのは，NW 環境（[17] は 1 Gb イーサネット，[18] は詳細未公表であるが大学の学内 LAN を介して IAS に接続）や CPU の性能差（[17] は Intel Core i5-10400@4.3 GHz，[18] は Intel Core i7-9700K@3.60 GHz）によるものと考える．またルートと追加オーダの暗号化については本来クライアント端末で行うが，いずれも所要時間はわずかである．

次に，**表 7** にルート件数と秘匿配送マッチング PF におけるマッチング処理の所要時間の関係を示す．ルート件数が 1,000 件以下の場合は Stack と Heap とも 8.4 MB に設定し，1 万件，5 万件のときは Heap の枯渇が起きないように Heap サイズを設定した．ルート件数がいずれの場合でも，FE 内で行うルートの復号に掛かる時間が支配的であり，1〜3 の処理全体の 99% 以上を占めるが，ルートが 289 件の場合でも 1〜3 の処理を合わせて 2.5 秒でマッチングができ，2.5 節で示した 3.6 秒以内という制約を満たすことができる．またルート件数とルートの復号に掛かる処理時間の関係を**図 6** に，ルート件数と割り当て先トラック決定に掛かる処理時間の関係を**図 7** にそれぞれ示す．双方の処理時間ともルート件数に対して線形で増えることが分かる．

ここで Stack と Heap のサイズはルート件数 5 万件のときに設定した合計 86.65 MB が上限であり，それ以上のサイズを割り当てると Enclave の起動ができなくなった．但しこれは Windows 環境下で第一世代の SGX を用いた場合の事象であり，Linux 環境下では処理速度が低下する代わりに，より大きなメモリサイズを設定できることが知られている．なお第二世代の SGX では上限の EPC サイズを動的に変更でき，ルート件数や各トラックの辺の数をさらに増やすことが可能と考えられる．

ここで，評価環境ではルートを全件まとめて SGX の中に送り復号して，それが完了したら荷主の追加オーダを

表 7　PF におけるマッチング処理の処理時間
Table 7　Processing time of matching on PF.

	ルート件数（件）				
	100	289	1,000	10,000	50,000
Stack Size (MB)	8.4	8.4	8.4	8.4	0.65
Heap Size (MB)	8.4	8.4	8.4	50.3	86.0
1.ルートの取得と復号 (ms)	706.6	2,459.0	7,398.3	76,515.9	346,858.0
2.追加オーダの取得と復号(ms)	2.1	1.8	1.6	1.7	1.6
3.割り当て先トラック決定 (ms)	0.5	2.2	3.6	20.4	172.7
1〜3 合計 (ms)	709.2	2,463.0	7,403.5	76,524.0	347,032.3
2・3 の合計 (ms)（荷主にとっての応答時間）	2.6	4.0	5.2	22.1	174.3

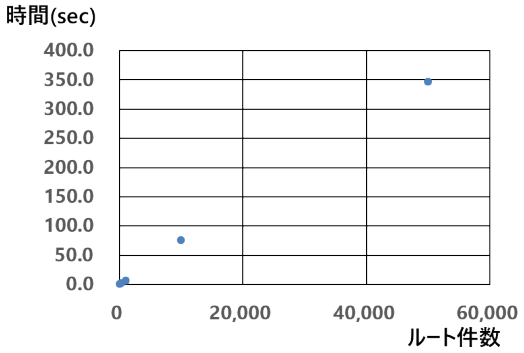

図 6　ルート件数とルートの復号に掛かる時間の関係
Fig. 6　Relationship between the num. of routes and the time taken to decode the root.

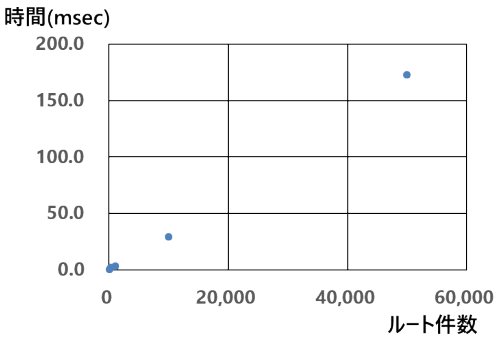

図 7　ルート件数と割り当て先トラック決定処理の時間の関係
Fig. 7　Relationship between num. of routes and processing time of deciding assigned edge.

SGX に送って復号後にマッチングをしている．一方，実際のサービスにおいては秘匿配送マッチング PF にルートが到着したら順次あらかじめ SGX の中に送って事前に復号しておけばよく，荷主にとっての応答時間は表 7 の 2 と 3 の合計と考えることができる．すると，ルート件数が 5

万件の場合でも，荷主にとっては 174.3 msec で応答が返って来ることから，十分高速なマッチングができている．

なお本来は荷主端末と秘匿配送マッチング PF との通信時間を考慮する必要がある．荷主から秘匿配送マッチング PF には表 2 に示す追加オーダを RSA-2048 bit で暗号した暗号文を送信し，また秘匿配送マッチング PF から荷主にはマッチング結果を送信するが，ともにデータサイズは数 kB であるため，通信時間はわずかと考える．

4.4　サービス提供者に対する秘匿性の評価

本節では，サービス提供者に対するルート・追加オーダの秘匿が実現できているか検証する．**表 8** に想定されるサービス提供者のルートと追加オーダの閲覧方法とそれに対して本研究で検討した対策を示す．

まず秘密鍵 sk_{pke} を Key Manager から DE に提供する際にサービス提供者が不正入手する可能性がある．これについては，SGX では R.A. を行った際に，ECDH 鍵交換が行われ AES 共通鍵が生成される [18], [19]．この共通鍵で sk_{pke} を暗号化した後に Key Manager から DE に渡すことで，サービス提供者の sk_{pke} の入手が抑止できる．なお [18] ではこの共通鍵は Enclave 外から入手不可能と述べているが，DE のソースコードをサービス提供者が改変すれば入手できる可能性が有る．しかし，そうすると $mrenclave_{DE}$ の値が変わって Key Manager が保持している正解値と一致せず DE の認証が通らなくなるため，DE のソースコードの改変は不可能である．

次に，サービス提供者が FE，DE のソースコードを改ざんして秘密鍵 sk_{pke} を入手する，ECALL や OCALL などによってルート・追加オーダを閲覧できる可能性がある．これらに対しても DE については，Key Manager は DE の認証時に R.A. に加えて $mrenclave_{DE}$ の値をあらかじめ保持している正解値と照合して DE の処理の不変性を検証しているため，DE の改ざんを防ぐことができる．FE についても DE が FE を認証する際に $mrenclave_f$ の値が sig_f を復号して得られる mrenclave の値と一致するか検証しているため，FE の改ざんを防ぐことができる．

よって，サービス提供者が FE, DE の中の処理を改ざんしてルート，追加オーダを閲覧することはできない．なお，サービス提供者が FE，DE の処理を変更する正当な理由がある場合は Key Manager がソースコードを確認することで，不正が無いことを確認する想定である．

表 8　サービス提供者のルートと追加オーダの閲覧方法と対策
Table 8　Way of browsing routes and orders of service provider and countermeasures.

閲覧方法	対策
Key Manager から DE に sk_{pke} を渡す際に入手	・R.A.による秘密鍵 sk_{pke} の提供（AES 共通鍵で暗号化）
FE，DE の処理を改ざん	・Key Manager による DE の認証 ・Key Manager による FE の署名と DE による FE の認証 ・ソースコード変更時の Key Manager の確認

4.5　今後の課題

SGX ではサイドチャネル攻撃からの脆弱性が指摘されており，良く知られているものとして，プログラムの中身や実行順序をメモリの配置される場所から推測して情報を得るキャッシュタイミング攻撃がある [9]．対策として，文献 [11], [13] のように ORAM を用いてメモリやディスク上に保管したデータへのアクセスパタンを秘匿する方法が一般的であるが，計算時間が大きく増加する懸念がある．それに対して，Costan らはメモリアクセスパタンが秘匿可能な TEE を実現した Sanctum を開発しており，処理時間のオーバヘッドは通常の TEE の 10% 程度である [20]．今後本研究でも適用を検討する．

その他，本研究ではサービス提供者への秘匿性を評価対象としたが，Key Manager やクライアント端末，通信路のセキュリティ，および秘匿配送マッチング PF における秘匿性以外のセキュリティの担保方法についても検討が必要と考える．また現在は追加距離最小という指標でのみ割り当て先トラックを決定しているが，積載量や時間制約などの他の指標を加味したマッチングについても検討する．

5.　得られた知見

本研究では準同型暗号のライブラリとして SEAL を用いた場合（2.6 節に記載）と，TEE として SGX を用いて関数型暗号を実施した場合の秘匿マッチングの処理時間を計測し，後者の場合において処理時間の制約を満たせることを確認するとともに，処理時間は SGX のメモリサイズの上限内でトラックのルート数に対して線形で増加することが知見として得られた．

また復号処理に掛かる時間が秘匿配送マッチング PF のマッチング処理時間の 99% 以上を占めることが分かった．ただし秘匿配送マッチング PF では受信したトラックのルートを随時 TEE 内に送り復号して保持しておくことで，ルート件数が増えても高速なマッチング処理が実施でき，荷主に対する応答時間を十分に短くできる．準同型暗号では暗号文を用いて演算を行うため事前にルートを復号しておくことはできないが，TEE の場合は事前に復号することで荷主に対する応答時間を担保できることが TEE の利点である．

また準同型暗号と TEE の双方を用いて実装して得られた TEE の利点の知見として，TEE 内では平文の処理のた

め，準同型暗号（SEAL を使用）に比べてマッチング処理自体の実装は容易でマッチングアルゴリズムの拡張性も高いこと，またアルゴリズムを拡張した場合の追加の処理時間は平文と同等と考えることができ，処理時間を見積もりやすいことが挙げられる．このように秘匿配送マッチングで求められる高速性と拡張容易性の面で TEE は準同型暗号よりも優れており，現時点では TEE を用いることが適切と考える．

6. まとめ

サービス提供者に対してルートと追加オーダを秘匿した状態で配送マッチングを実現する秘匿配送マッチングにおいては処理時間の課題があり，準同型暗号では制約を満たせない．一方関数型暗号を TEE で実現すれば高速化が可能であることは知られていたが，TEE の中でルートと追加オーダを平文に戻す際のオーバヘッドがあるため，処理時間の制約を満たせるかは不明であった．

本研究では，IRON [1] のプロトコル（システム構成，処理手順）をベースに，SGX を用いた多入力関数型暗号を実現し，評価環境を構築して処理時間を評価した．その結果トラックのルートが 289 件存在するマッチングを約 2.5 秒で実施でき，3.6 秒以内という制約を満たせることを確認した．また，サービス提供者にルートと追加オーダが秘匿できることを確認し，秘匿マッチングにおける TEE を用いた関数型暗号の有効性を確認した．

謝辞　本研究の推進に際し，多数のご助言をいただいた薦田憲久 大阪大学名誉教授に感謝いたします．

参考文献

[1] Fisch. B., Vinayagamurthy. D., Boneh. D., and Gorbunov. S.: IRON: Functional Encryption using Intel SGX, *Proc. Conference on Computer and Communications Security*, pp.765–782 (2017).
[2] 棚橋　優，今堀慎治：配送計画問題に対するデータベース付きメタ戦略（最適化の基礎理論と応用），京都大学数理解析研究所講究録，Vol.1879, pp.164–179 (2014).
[3] トランコム株式会社：2023 年 3 月期第 2 四半期決算説明会（オンライン），入手先〈https://www.trancom.co.jp/files/topics/1159_ext_02_0.pdf〉（参照 2023-02-07）.
[4] 菊池　亮，五十嵐大：秘密計算の発展：データを隠しつつ計算する仕組みとその発展，電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review, Vol.12, No.1, pp.12–20 (2018).
[5] 中西　聖，成末義哲，森川博之：準同型暗号を用いた顔認証連携基盤における応答時間の評価，マルチメディア，分散，協調とモバイルシンポジウム 2022 論文集，pp.1534–1538 (2022).
[6] 須崎有康：Trusted Execution Environment の実装とそれを支える技術，電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review, Vol.14, No.2, pp.107–117 (2020).
[7] Boneh. D., Sahai. A., and Waters. B.: Function Encryption: Definitions and Challenges, *Cryptology ePrint Archive*, Paper 2010/543 (2010).
[8] Goldwasser. S., Gordon S. D., Goyal. V., Jain. A., Katz. J., Liu. F., Sahai. A., Shi. E., and Zhou. H. S.: Multi-Input Functional Encryption, *Proc. EUROCRYPT 2014*, pp.578–602 (2014).
[9] 加納英樹，Tibouchi. M.，阿部正幸：Intel SGX を用いた関数型タイムリリース暗号，暗号と情報セキュリティシンポジウム予稿集，No.2A3–5, pp.1–7 (2019).
[10] Bhatotia, P., Kohlweiss, M., Martinico, L., and Tselekounis, Y.: Steel: Composable Hardware-based Stateful and Randomized Functional Encryption, *Proc. IACR International Conference on Public-Key Cryptography*, pp.709–736 (2021).
[11] 岩田大輝，清水佳奈：Intel SGX を用いた個人ゲノム情報解析システム，暗号と情報セキュリティシンポジウム予稿集，No.1C1–1, pp.1–8 (2020).
[12] Mandal, A., Mitchell, J. C., Montgomery, H., and Roy, A.: Data Oblivious Genome Variants Search on Intel SGX, *Proc. Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pp.296–310 (2018).
[13] Stefanov, E., Dijk, M., Shi, E., Chan, T., H., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path ORAM: An Extremely Simple Oblivious RAM Protocol, *Journal of ACM*, Vol.65, No.4, pp.1–26 (2018).
[14] Felsen, S., Kiss, Á., Schneider, T., and Weinert, C.: Secure and Private Function Evaluation with Intel SGX, *Proc. 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pp.165–181 (2019).
[15] Yao. A.: Protocols for Secure Computations (Extended Abstract), *Proc. IEEE FOCS '82*, pp.160–164 (1982).
[16] Goldreich, O., Micali, S., and Wigderson, A.: How to Play any Mental Game, *Proc. 19th ACM Symposium on Theory of Computing*, pp.218–229 (1987).
[17] 矢川　嵩，照屋唯紀，須崎有康，阿部洋丈：Intel SGX における 2 つのリモートアテステーションの利点と欠点の考察，暗号と情報セキュリティシンポジウム予稿集，No.1C2–3, pp.1–8 (2022).
[18] 加藤風芽，新城　靖：Intel SGX を利用した自己破壊・出現データの実装，コンピュータシステム・シンポジウム予稿集，pp.42–51 (2020).
[19] Intel: Code Sample: Intel® Software Guard Extensions Remote Attestation End-to-End Example（オンライン），入手先〈https://www.intel.com/content/www/us/en/developer/articles/code-sample/software-guard-extensions-remote-attestation-end-to-end-example.html〉（参照 2023-08-07）.
[20] V. Costan, I. Lebedev, and S. Devadas.: Sanctum: Minimal Hardware Extensions for Strong Software Isolation, *Proc. 25th USENIX Conference on Security Symposium*, pp.857–874 (2016).

古家 直樹（正会員）

2007 年京都大学大学院工学研究科電気工学専攻博士前期課程修了．同年（株）日立製作所入社．現在，同社研究開発グループ DX エンジニアリング研究部主任研究員．物流システムおよび情報セキュリティに関する研究開発などに従事．また 2020 年大阪大学大学院情報科学研究科マルチメディア工学専攻博士後期課程入学，2023 年 9 月単位修得退学．日本経営工学会会員．2020 年日本経営工学会論文賞受賞．

矢内 直人（正会員）

2011 年筑波大学大学院システム情報工学研究科博士前期課程修了．2014 年筑波大学大学院システム情報工学研究科博士後期課程修了．同年から 2021 年まで大阪大学大学院情報科学研究科マルチメディア工学専攻・助教，2021 年 12 月から同専攻・准教授に着任，現在に至る．専門は情報セキュリティ．情報セキュリティに関する実践的教育にも従事．2013 年 SCIS 論文賞，2015 年 CSS 論文賞，2020 年辻井重雄セキュリティ論文賞など各賞受賞．

藤原 融（正会員）

1981 年大阪大学基礎工学部情報工学科卒業．1986 年同大学院基礎工学研究科博士課程修了．工学博士．同年同大学助手．1997 年より同大学教授．2002 年から同大学大学院情報科学研究科マルチメディア工学専攻所属．2023 年より島根大学材料エネルギー学部教授．同時に大阪大学情報科学研究科招へい教授．符号理論，情報セキュリティの研究に従事．電子情報通信学会フェロー，IEEE, ACM 各会員．