Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

関数型暗号を用いた秘匿配送マッチングの TEE による実現

古家直樹 [1,2]　矢内直人 [1]　藤原融 [1]

概要：物流業界では，トラック不足から荷主のオーダを余裕のある配送業者の配送ルートの途中に割り当てる配送マッチングが行われている．荷主のオーダと配送業者のルート情報には，出発地・到着地の所在地などの機微情報が含まれる．本研究ではサービス提供者に対してオーダとルート情報を秘匿する秘匿配送マッチングを提案する．秘匿計算の手法には準同型暗号等があるが，処理速度の課題がある．そこで本研究では，処理時間の課題に対して，TEE（Trusted Execution Environment）によって任意の演算が可能な関数型暗号を実現し，秘匿配送マッチングに適用することを提案する．TEE として Intel SGX を用いて処理時間とセキュリティの評価を行った．配送業者 255 社が参加するマッチングを約 1.9 秒で実施でき，提案手法の有効性を確認した．

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

Function Encryption Secret Delivery Matching System Using TEE

NAOKI FURUYA[†1, 2]　NAOTO YANAI[†1]
TORU FUJIWARA[†1]

Keywords: Function Encryption, TEE, Intel SGX, Secure Computing, Delivery Matching

1. はじめに

物流業界では，荷主のオーダを配送業者（トラック）の空きリソースに割り当て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)が様々な IT ベンダから提供されており，荷主が配送業者を直接手配できない場合などに活用されている．一般的なマッチングサービスでは，トラックのルート情報を予め収集しておき，荷主の追加オーダ（以降，オーダと称す）の配送元・配送先の情報をもとに，適切なトラックにオーダを割り当てる．荷主のオーダと配送業者のルート情報には，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オーダとルート情報を秘匿することが望まれる．

本研究では秘匿計算の技術を用いてサービス提供者に対して荷主のオーダと配送業者の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．ここで，仮に国内最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，255 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．秘匿計算には様々な手法があるが，例として準同型暗号では処理時間の制約を満たすのは困難である．そこで本研究では，処理時間の課題に対して，TEE（Trusted Execution Environment）によって任意の演算が可能な関数型暗号を実現し，秘匿配送マッチングに適用することを提案する．処理時間とセキュリティの評価を行い提案手法の有効性を示す．

以降，2 章で秘匿配送マッチングの課題と関連研究，3 章で提案手法である TEE による関数型暗号の秘匿配送マッチングの内容，4 章で評価環境の実装，5 章で評価，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の処理概要を述べ，秘匿化の対象データと課題を説明する．

2.1 配送マッチング処理の概要

一般的な配送計画問題では，トラックの燃料代やドライバの労働時間を考慮して各トラックの総走行距離が最小になるようにルートを策定する[1]．配送マッチングにおいても，トラックの燃料代とドライバ労働時間が過多とならないように，荷主のオーダをトラックのルートに割当てた際に，追加距離が最小となるトラックに対して割当てる．

図 1 を用いて割当て先の配送業者およびルートの決定方法を説明する．以降の説明では，配送業者 1 社あたりトラック 1 台と仮定して説明する．図 1 は，配送業者 2 社がそれぞれルート R1, R2 で走行予定であり，マッチングを行うプラットフォーム（以降，PF と称す）は荷主から地点 7 から 8 へ配送するオーダを受信した場合を表している．

PF は配送業者のルートを配送が始まる前に予め入手しており，荷主のオーダが PF に届くとマッチング処理が始

a) Intel Corporation またはその子会社の商標である．

1 大阪大学
Osaka University
2 (株)日立製作所
Hitachi, Ltd.

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

まる．PF はオーダを R1, R2 の中のどの辺に割り当てたら追加距離最小となるかを算出し，オーダの割当先を決める．また，配送業者や荷主がマッチング結果を承諾しなかった場合は，次に追加距離が短い配送業者を割当先とする．

図 2 を用いて，PF の処理概要を説明する．R1, R2 にはそれぞれ 3 つの辺（例：地点 1→2）が存在するため，△で示した合計 6 つのオーダ挿入候補が存在する．このそれぞれに対しオーダを割り当てた際の追加距離を求め，追加距離が最小となる辺（図 1 では 6→4 の途中）にオーダを割り当てる．

ここでトラックは地点 7 で荷物を積んだら，途中で他の地点には立ち寄らず地点 8 に向かい荷物を降ろすものとする．これは，途中で他の地点に寄って更に別の荷物の荷積み・荷降ろしがあると，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って地点 8 における荷降ろしがしづらくなったり，別の地点で荷降ろし時に地点 7 で積んだ荷物が荷降ろし作業を阻害することを避けるためである．

また，将来的に積載量の制約や各地点への到着時間の制約等を加味することも考えられるが，本稿では対象外とする．

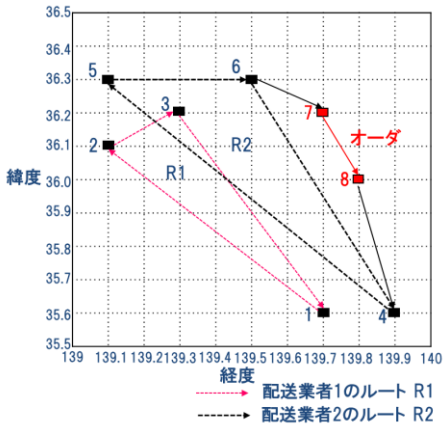

図 1　配送業者の決定方法

Figure 1　Way of Determination of Delivery Company.

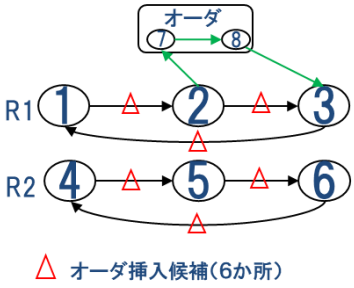

図 2　PF の処理概要

Figure 2　Processing outline of PF.

2.2 処理性能の目標

配送マッチングの国内最大手の T 社では日々6,000 件（2022 年）のマッチングを成立させている[b]．本研究でも同等のマッチング件数をめざすものとする．ここで，配送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，前日の午後（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配送業者数について説明する．以降の説明においても，1 配送業者が本サービスに参加するトラックは 1 台と仮定して説明するが，複数台が参加してもその配送業者の端末で複数件のオーダを暗号すれば良く，PF における性能要件の目標値は変わらない．

T 社では日々12,000 台のトラックの情報を扱っている．本研究でも日々12,000 件のルートを扱うものとし，仮にトラックの台数の分布が 47 都道府県で均一として都道府県ごとに分けてマッチングを行うとすると，単純計算で 1 回のマッチングで 255 台分のルートを扱うことになる．

そこで，T 社と同規模のサービスを実現するには，配送業者 255 社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が目標となる．

2.3 配送マッチングで用いるデータ

平文の配送業者のルートの内容を表 1，荷主のオーダの内容を表 2 に示す．ルート，オーダとも出発地・到着地を予め各配送業者，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．また配送業者のルートには複数の辺（1 回の配送に該当）が存在する．

次に，各配送業者・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，配送業者・荷主ごとに自身の端末で保持し，PF や他者には共有されない想定である．

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　オーダの内容

Table 2　Contents of Order

オーダ ID	出発地	到着地
1	7	8

b) https://www.trancom.co.jp/power/number/

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

表 3　住所情報
Table 3　Latitude and Longitude 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
…	…	…	…

2.4 秘匿化対象

平文による配送マッチングでは，ルートとオーダ中の出発地・到着地の地点 ID を緯度・経度に変換して，PF に送信し PF 上で割当先の配送業者を決定する．

荷主と配送業者にとって，PF（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．また PF が悪意ある第三者からの攻撃を受けると緯度・経度情報の流出の懸念もある．そこで本研究では，出発地・到着地の緯度・経度を PF に秘匿した状態でマッチングを行うことに取り組む．図 3 に秘匿配送マッチングの概要を示す．なお配送業者と荷主のクライアント端末，および 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も既存の方法でセキュリティを担保すれば良いものと考え，本研究の対象外とする．

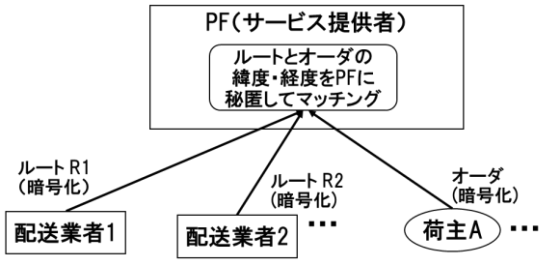

図 3　秘匿配送マッチングの概要
Figure 3　Outline of the Delivery Matching.

2.5 秘匿配送マッチング実現の課題

秘密計算の手法の例として，暗号文のままで平文の線形演算と乗算が可能な完全準同型暗号がある．しかし処理時間の課題が有ることが知られている[2].

例えばライブラリとして Microsoft の SEAL[c]を用いてクラウド環境（CPU：Intel Core Processor(Skylake, IBRS)@2GHz, RAM：16GB）上で配送業者 100 件の配送業者のマッチングを行った場合の処理時間を計測したところ，1 回あたり約 42 秒掛かる．そのため，2.3 節で説明した 255 社の配送業者のマッチングを 1 回あたり 3.6 秒以内に成立することは実現困難と考えられる．

2.6 関連研究

面は，準同型暗号をベースにした PSI（Private Set Intersection）プロトコルを用いた秘匿多肢選択マッチングプロトコルを提案している[3]．参加者が選択した回答をサーバに秘匿しつつ，参加者間でどの回答が一致しているかを判定している．マッチングの誤判定を防ぐために暗号化時に乱数を選んで暗号化するが，サーバで乱数をチェックする際の処理時間の課題がある．

これに対して，近年の CPU では OS とは独立してクリティカルな処理が実行できる TEE（Trusted Execution Environment）が提供されている[4]．Fischer らは TEE として Intel SGX（以降，SGX）を用いて，関数型暗号（Functional Encryption）を実現する Iron を提案している[5]．ここで関数型暗号では暗号文を復号する際に，元の平文は秘匿されつつ任意の演算結果を得ることができる[6]．Fischer らは TEE を用いることで，従来の関数型暗号に比べて桁違いに高速な処理が期待できると述べている．

岩田らは，SGX を用いて個人のゲノム情報をサーバに秘匿して解析を行う手法を提案しており，処理時間は秘匿化を行わない場合と比べて 2 倍以下と述べている[7]. 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（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，同じくゲノム情報の χ 二乗分布の解析を SGX の中で行っており，ORAM を用いることでメモリアクセスパタンの秘匿を図っている[8].

次に，加納らは Iron[5]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おり，従来タイムリリース暗号で用いられていた時報局や tome-lock puzzles を SGX に置き換えている[9]．また Bhatotia らは，Iron [5]を拡張し stateful（システムが現在の状態を表すデータなどを保持しており，その内容を処理に反映させる方式）かつ randomized された関数型暗号を実現した Steel を提案している[10].

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 を用いることで実現している[11]．また，処理時間を評価し，処理が複雑になっても Yao's GC（Garbled Circuit）[12]と GMW（Goldreich, Micali, and Wigderson）[13]のプロトコルに比べて高速であることを述べている．

c) Microsoft 社の商標である．https://github.com/microsoft/SEAL

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

3. 関数型暗号による秘匿配送マッチング

本章では，提案手法である TEE による関数型暗号を用いた秘匿配送マッチングについて説明する．

3.1 TEE による関数型暗号の適用

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[d]や Intel の SGX が知られている．SGX の場合は，ユーザが DRAM 上に Enclave と呼ばれる保護領域を形成し，その内部で秘密の情報を保持しながらプログラムを実行できる．

また SGX では 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 Attestation の機能を有する．Attestation には Local Attestation と Remote Attestation がある．Local Attestation は同一 CPU 上の Enclave 間の認証で，個々の CPU が持つ固有鍵で作成された Report を検証することで Enclave が同一 CPU 上で動いていることを保証する．一方 Remote Attestation は外部の Attestation サーバにて，正規の CPU 上でプログラムが動作していることの認証を行う機能である．また Local Attestation においては，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（ユーザ秘密鍵）の利用者がユーザ秘密鍵を用いて暗号文を復号する際に，任意の処理結果を得ることができるもので，利用者に元の平文の内容の秘匿も可能である．具体的には，マスター秘密鍵を保持する機関（Authority）が，暗号化されたデータに対してある関数 f の計算が可能なユーザ秘密鍵を生成する．現在は主に入力文に対するアクセス制御に用いられているが，より一般的な関数に対する関数型暗号の実現は主要な課題と言われている[6].

その一方で，Fischer らに拠れば TEE を用いてハードウェア的に関数型暗号を実現すれば，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，その演算結果を取得できる[5]．そこで本研究では TEE を用いて関数型暗号を実現した秘匿配送マッチングを提案する．

3.2 TEE による関数型暗号秘匿配送マッチングの全体像

図 4 を用いて，TEE による関数型暗号秘匿配送マッチングの全体像を説明する．各略号の意味は以下の通りである．

- DE：Decryption Enclave の略
- FE：Function Enclave の略
- KME：Key Manager Enclave の略
- PF App.：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・オーダの暗号用公開鍵，復号用秘密鍵
- 関数 f : FE で行う一連の処理（追加距離計算，配送業者決定，ログ生成）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 の mrenclave 値
- $mrenclave_{DE}$：DE の mrenclave 値
- 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は，1.鍵生成と DE の認証，2.FE の認証，3.マッチングの三つの処理からなる．これらの処理を構成する要素として PF（秘匿配送マッチング PF）と Key Manager が定義される．ここで Key Manager は 3.1 節で述べた関数型暗号の Authority を兼ねる．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．Key Manager は信用のおける第三者認証機関が担う想定である．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 Remote Attestation を受けて，DE を検証後に署名検証鍵 vk_{sign} とルート・オーダの復号用秘密鍵 sk_{pke} を提供する．

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 mrenclavef を受けて KME に送り，KME にて mrenclavef に sksign で署名を行った後，その結果である sigf を PF App.に送る．その後 PF App.は，sigf を取得して FE に送信する．

次に，FE は DE に対して Local Attestation を要求し，FE の処理内容の署名である sig_f を DE に提供する．DE は署名検証鍵 vk_{sign} で sig_f を検証し，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．

3. マッチング処理

配送業者と荷主は PF App.から公開鍵 pk_{pke} を取得し，それぞれルート・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・オーダを送り，FE は sk_{pke} にてルートとオーダを平文に戻し，マッチング処理を行う．FE はマッチング処理の結果を PF App.に通知する．

d) ARM 社の商標である．

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

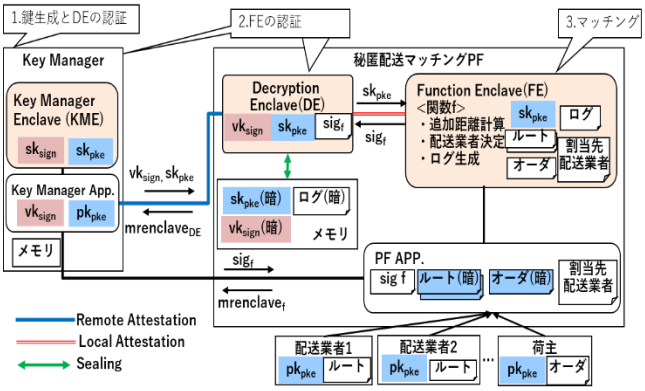

図 4　秘匿配送マッチングの全体像

Figure 4　Whole Picture of Secret Delivery Matching.

3.3 鍵生成と DE の認証の処理詳細

まず，Key Manger 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 5 を用いて説明する．

(1) KME は鍵生成を行う

(2) DE は Remote Attestation（R.A.）を Key Manager App に要求

(3) sk_{pke} と vk_{sign} を DE に配布

ここで，(2) では通常の R.A.の処理に加えて，KME にて $mrenclave_{DE}$ の一致検証を行って DE を認証する．KME は予め正解値の $mrenclave_{DE}$ の値を保持しておき，その正解値と Remote Attestation で送られる $mrenclave_{DE}$ との一致検証を行う．これにより，第三者による 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．なお，DE の認証は定期的（例. 1 日に一回）に行う想定だが，sk_{pke} と vk_{sign} は DE が一度受け取ったら，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，以後 PF のメモリから取得する．

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 で保持している正解値の DE の mrenclave を置き替え，sk_{pke} と vk_{sign} を再発行する想定である．

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 5 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 Local Attestation（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 PF 上で行われるようにする．

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，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 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．

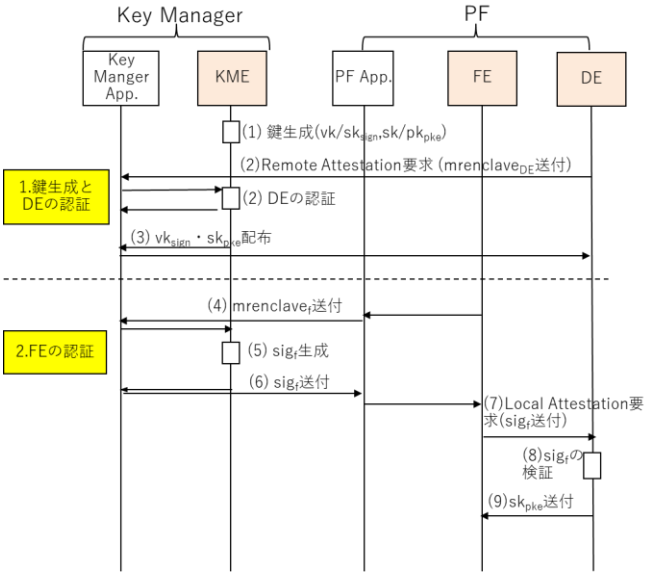

図 5　FE の認証を行うまでの処理フロー

Figure 5　Procedure Flow till FE Attestation

3.5 マッチング処理の詳細

図 6 を用いてマッチングの処理の詳細を説明する．

(10) PF App.は予め 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) 配送業者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) 荷主はオーダを暗号化し PF App.に送付

(14) FE は PF App.からオーダを取得し復号

(15) FE はマッチング処理を実施．合わせてログ記録．

(16) FE はオーダの割当先の配送業者を PF App.に通知．

(17) PF App.は割当先の配送業者に通知

(18) 配送業者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・配送業者を通知

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

(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) FE は DE にログを送付（１日の処理終了後）

(21) DE は Sealing でログ，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 のメモリ制約の上限を超えない範囲で複数件まとめて送る想定である．また（17）で配送業者に通知する際に，予め FE でオーダを sk_{pke} で暗号化して配送業者に送付し，配送業者は pk_{pke} でオーダを復号して，受諾するか否かを選択する想定である．

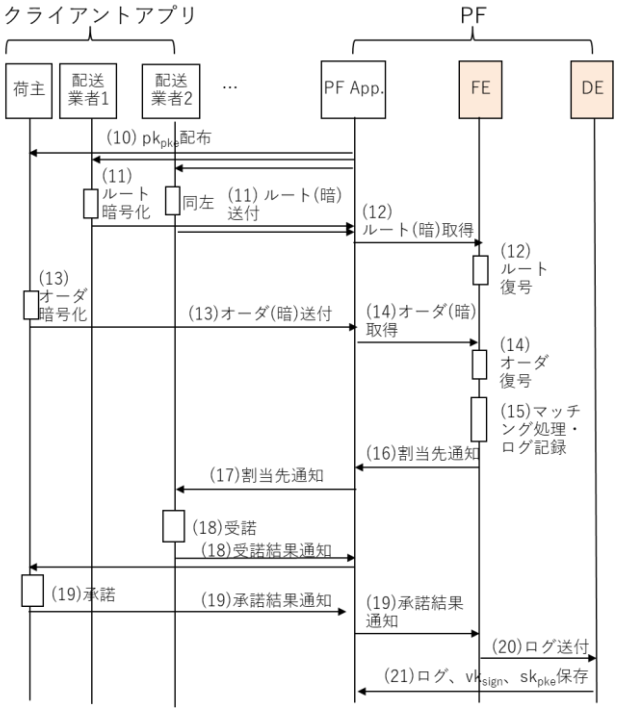

図 6　マッチングの処理フロー

Figure 6　Procedure Flow of Matching.

また，（18）（19）で配送業者や荷主が承諾しなかった場合は，次に追加距離が短い配送業者を割当先として，（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荷主が承諾した配送業者は次のマッチングの割当候補から外すことを想定しているためである．

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，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 Key Manager が予め 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，PF では Key Manager が認証したプログラム以外は実行できない仕組みとしている．

また，Key Manager は，3.4 節の（4）～（6）を行う際に，PF に $mrenclave_f$ と合わせて FE のソースコードを一般に公開，もしくは提出させ，FE の処理に不正（例：平文にしたルートとオーダを ocall で PF App.に出力する等）がないかを人手等で確認する想定である．また DE の不正についても，初回および処理内容を変更した際に同様に Key Manager にソースコードを公開もしくは提出させて不正がないか確認する想定である．

次に，サービス提供者が配送業者と結託をすると FE から取得したマッチング結果を書き換えて，特定の配送業者にオーダを割り当ててしまう懸念がある．そこで，FE 内のマッチング結果はログに記録しておき，サービス提供者が改ざんできない形で PF のメモリに保持しておく．サービス提供者のマッチング結果の書き換えの不正が疑われる際は，ログとの付け合わせを行うことで不正の有無の検証を行う．

4. 実装

3 章で説明してき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 や開発環境等については，以下に記載の通りである．評価環境では，Key Manager および配送業者，荷主は PF と同じ PC で実装している．SGX は EPC（Enclave Page Cash）の上限が 96MB の第一世代のものを用いている．

・開発に用いた PC：
  ・マシン名：HP 250G7 Notebook PC
  ・CPU：Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz
  ・RAM：8GB
  ・OS：Windows 10 Home
・開発環境：
  ・Visual Studio Community 2017
  ・Intel SGX SDK ver 2.15.100.4
・言語：C++
・用いた暗号ライブラリ：OpenSSL1.1.1m[e]
・用いた暗号：
  ・(pk_{pke}, sk_{pke})：RSA-2048bit with OAEP SHA-256
  ・(vk_{sign}, sk_{sign})：RSA-3072bit

なお，評価環境では，図 6 の（15）のログ記録および（17）以降の処理は未実装であり，今後実装を進めることを検討している．また（2）の Remote Attestation については PF の処理とは切り分けて実装し，処理時間を評価している．

5. 評価

本章では，評価環境を用いて行った処理性能の評価と，セキュリティの評価について述べる．

e) https://www.openssl.org/

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

5.1 処理性能の評価

2.2 節で述べた，1 回あたり 3.6 秒以内にマッチングを成立させる目標を満たせるか確認する．荷主は 1 人（1 オーダ）とし，各配送業者のルートには 4 つの辺が含まれるものと仮定して，処理時間の計測を行った．計測には，Windows の Query Performance Counter を使用した．処理時間は複数回処理を行った平均値を用いている．

表 4 にマッチング処理以外の時間を示す．Remote Attestation に約 4 秒掛かるが，1 日に一回利用の少ない夜間などに行えば良い処理のため，問題ないと考える．ルートとオーダの暗号化については将来的にクライアント端末で行うが，いずれも所要時間は僅かである．

表 4　マッチング処理以外の処理時間

Table 4　Processing Time Except for Matching

項目	処理時間(msec)	実施頻度
鍵の生成	700	DE の処理変更時
Remote Attestation	4,065	1 日 1 回程度
sig_f の生成	4	FE の処理変更時
Local Attestation	4	1 日 1 回程度
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 配送業者あたり）	4	毎回のマッチング時
合計	4,777	-

次に，表 5 に PF におけるマッチング処理の所要時間を示す. FE 内でルートの復号に掛かる時間が支配的であるが，配送業者が 255 社の想定でも 1～3 の処理を合わせて 1.9 秒でマッチングができ，2.4 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．現在の評価環境ではログの記録は未実装であるが，仮に実装しても処理時間に大きな影響は無いものと考える．

表 5　マッチング処理の処理時間 (msec)

Table 5　Processing Time of Matching (msec)

	配送業者数（ルート数）		
	100	255	1,000
1.ルートの復号	822.0	1,873.6	7,227.0
2.オーダの復号	2.1	1.8	1.6
3.マッチング処理	0.8	1.0	3.6
1～3 合計	824.9	1,876.4	7.232.2
2・3 の合計 （荷主にとっての応答時間）	2.9	2.8	5.2

ここで，評価環境ではルートの復号は一度に配送業者の数だけまとめて SGX の中に送り復号して，それが完了したら荷主のオーダを SGX に送って復号後にマッチングをしている．一方，実際には PF にルートが到着したら予め SGX の中に送って復号しておき，荷主のオーダが PF に到着したら SGX に送って復号してマッチング処理をすれば良く，荷主にとっての応答時間は表 5 の 2 と 3 の合計の時間と考えることができる．ただし荷主端末と PF との通信時間を含める必要がある．すると，配送業者が 1,000 社の場合でも，荷主にとっては通信時間を除けば 5.2msec で応答が返ってくることから，十分高速なマッチングができている．

5.2 セキュリティの評価

本節では，PF のセキュリティについて評価する．Key Manager は信用のおける第三者認証機関が担うものと仮定し，評価の対象外とする．

PF におけるセキュリティリスクとしては，表 6 に示す 3 つが考えられる．1 については，マッチング処理の入力となるルート・オーダに対する不正で，改ざん・閲覧・持ち出しが考えられる．これに対しては，秘密鍵 sk_{pke} を FE, DE 内で保持し，TEE の中でのみルート・オーダを復号することで，TEE 外へのルート・オーダの開示を防ぐ．また，秘密鍵 sk_{pke} を Key Manager から DE に提供する際に $mlenclave_{DE}$ で DE を認証しているため，DE の成りすまし・改ざんを防ぐことができる．FE についても Key Manager が処理内容を署名し，さらに DE が Local Attestation で FE を認証する際に sig_f を検証しているため FE の成りすまし・改ざんを防ぐことができる．よって，第三者や荷主・配送業者が FE, DE に成りすまして秘密鍵を入手することやサービス提供者が FE, DE の中の処理を改ざんしてルート，オーダを閲覧することはできない．

なおルート, オーダの暗号には RSA OEAP を用いており，IND-CCA2（暗号文 m_0, m_1 が平文 c_0, c_1 のどちらを暗号化したものか識別できないという識別不可能性を持ち，適応的選択暗号文攻撃の耐性がある）の安全性が確保されている．一方で，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[7][8]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対策一般的であるが，計算時間が大きく増加する懸念がある．それに対して，Costan らはオープンアーキテクチャの RISC-V でメモリアクセスパタンが秘匿可能な TEE を実現した Sanctom を開発している[14].

次に 2 に関しては，サービス提供者が FE, DE の中の処理を改ざんして特定の配送業者が優位となるようにする，1 で述べたようにルート，オーダを閲覧可能にするなどの不正が考えられる．これについては 1 で述べたように, Key Manager による FE の処理と DE の認証を行うことでサービ

Vol.2022-CSEC-99 No.1
Vol.2022-SPT-49 No.1
Vol.2022-EIP-98 No.1
2022/12/22

表 6　PF におけるセキュリティリスク
Table 6　Security Risks on the PF

セキュリティリスク	誰による不正か	対策
1.ルート・オーダの改ざん・閲覧・持ち出し	・サービス提供者 ・第三者 ・荷主・配送業者	・秘密鍵 sk_{pke} を FE, DE 内に配置 ・Key Manager による DE の認証 ・Key Manager による FE の処理の署名と DE による FE の認証 ・ソースコードの公開，確認
2.マッチング処理の改ざん	・サービス提供者	・Key Manager による DE の認証 ・Key Manager による FE の処理の署名と DE による FE の認証 ・ソースコードの公開，確認
3.マッチング結果の改ざん	・サービス提供者 ・第三者 ・荷主・配送業者	・ログの記録

ス提供者の改ざんを抑止する．なお，サービス提供者が正当な理由で FE, DE の処理を変更する際は Key Manager にそれぞれ $mrenclave_{DE}$, $mrenclave_f$ を提出するとともに，ソースコードも公開もしくは Key Manager が確認する想定である．

最後に 3 については, FE から取得したマッチング結果を PF. App 上で書き換えてしまう不正である．これについては，FE の中でログを記録し，Sealing を用いて改ざんできない形でメモリに保持しておき, 不正が疑われる際には FE から DE を経由して Remote Attestation で KME に送付して，Key Manager がログのマッチング結果と確認することで抑止できる．

6. まとめ

本研究では，サービス提供者に対してルートと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技術の開発を行った．秘匿計算には様々な手法があるが，処理時間が課題となる．そこで提案手法では，暗号化された配送業者のルートと荷主のオーダを TEE の中で復号してマッチング処理を行い，その結果のみをサービス提供者および利用者に知らせる．これにより配送業者のルートが 255 件存在する一回のマッチングを約 1.9 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．またセキュリティの評価を行い, 提案手法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] 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, 2014, vol. 1879, p.164-179.
[2] 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, 2018, vol. 12, no. 1, p. 12-20.
[3] 面和成．秘匿多肢選択マッチングプロトコルの提案と評価. in コンピュータセキュリティシンポジウム論文集，2014，p. 659-666.
[4] 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，2020，vol. 14，no. 2，p.107-117.
[5] Fisch. B., Vinayagamurthy. D., Boneh. D., and Gorbunov. S.. IRON : Functional Encryption using Intel SGX. in Proceedings of Conference on Computer and Communications Security，2017，p.765-782.
[6] Boneh. D., Sahai. A., and Waters. B.. Function Encryption: Definitions and Challenges. Cryptology ePrint Archive, Paper 2010/543, 2010,
[7] 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. in 暗号と情報セキュリティシンポジウム予稿集，2020.
[8] Mandal, A., Mitchell, J. C., Montgomery, H., and Roy, A.. Data Oblivious Genome Variants Search on Intel SGX. In Proceedings Data Privacy Management, Cryptocurrencies and Blockchain Technology, 2018, p. 296-310.
[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，2019.
[10] Bhatotia, P., Kohlweiss, M., Martinico, L., and Tselekounis, Y.. Steel: Composable Hardware-based Stateful and Randomized Functional Encryption. in Proceedings of IACR International Conference on Public-Key Cryptography, 2021, p. 709-736.
[11] Felsen, S., Kiss, Á., Schneider, T., and Weinert, C.. Secure and Private Function Evaluation with Intel SGX. in Proceedings of 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop, 2019, p. 165-181.
[12] Yao. A.. Protocols for Secure Computations (Extended Abstract). in Proceedings of IEEE FOCS '82, 1982, p. 160-164.
[13] Goldreich, O., Micali, S., and Wigderson, A.. How to Play any Mental Game. in Proceedings of 19th ACM Symposium on Theory of Computing, 1987, p. 218-229.
[14] V. Costan, I. Lebedev, and S. Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. in Proceedings of 25th USENIX Conference on Security Symposium, 2016, p.857-874