令和5年度

修士論文

モバイル端末向け秘匿共通集合計算の提案と実装

情報科学プログラム 計算機基礎学研究室
M226181 村越允

主指導教員・主査　准教授　北須賀 輝明
副指導教員・副査　教授　中西 透
副指導教員・副査　教授　森本 康彦
副査　准教授　今井 勝喜 (福山大学)

令和 6年 2月6日

広島大学大学院先進理工系科学研究科

概要

入学や就職など自分を取り巻く環境が変化したとき，他人と新たな交流が生まれる．しかし，顔だけ知っている程度の他人と話すということは心理的負担が大きい．そこで本研究では自分と他人の接触履歴からこれまでに接触した日時を判定して会話を促すアプリケーションを提案する．接触履歴は BLE を用いて取得する．その際，BLE で自分の名前をブロードキャストするとプライバシが保護されないので，時間経過で変化するランダムな識別子をブロードキャストする．また接触履歴から接触回数を取得するためには二者間での接触履歴を見せ合う必要があるが，接触履歴は個人情報である．そのため互いの接触履歴を暗号化した状態で共通要素を見つけることができる秘匿共通集合計算を用いて接触日時と回数を導出する．本論文では，Android 端末で秘匿共通集合計算をする機能を実装し，扱うデータ量を変化させたときの実行時間を計測した．実行時間は指数関数的に増大していくことが分かった．また，各処理の実装方法を工夫することで実行時間が改善されると考えている．

目次

図目次

表目次

第 1 章

はじめに

SNS (Social Networking Service) では使用者の SNS 上の友人関係やプロフィール情報などをもとに,「知り合いかも」などといった新たな人との交流を促す機能が実装されている．似たような機能を人の接触に対して提供することができれば，新たな友人を発見することができる．特に，入学や就職など，自分を取り巻く環境が変化するようなタイミングでの利用を想定する．そこで，よく会っていたり，長時間同じ場所にいる人を見つけるために人の行動の規則性に注目する．人は平日の朝には勤務先や学校に行くなどの，規則的に決まった時間に決まった行動を起こすことが多い．そして，その際には多くの他人と接触する．この他人の中には，接触頻度が高い他人が存在する．この接触回数が多い他人は自分にとって顔はよく見かける顔見知りとなる．そのような人は，自分と同じ時間に同じ場所にいることが多い人であり，同じ勤務先や学校に通っている人である場合が多い．そして，そのような人と新たな交流が生まれるきっかけを本研究で提案するアプリケーションで提供する．ここで，自分が特定の他人に対してよく見かけると思ったとしても，自分の勘違いやよく似た他人と見間違えていることなどがある．このようなことが起きないように，自分が顔見知りを認識した時に接触を記憶することは手間がかかる．そこでスマートフォン上で動作する提案アプリケーションは接触履歴を取得し，分析することで特定の人との接触頻度を表示する．これによってこの人が顔見知りであることを容易に判断することができるようになる．

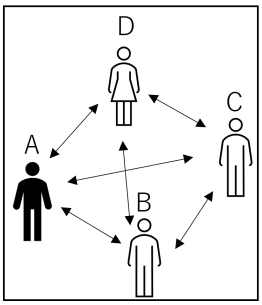
しかし，ある人 A の接触履歴を他人 B が見ることができるようになると，A と接触した人 C を追跡できるようになる可能性がある．例えば，A の接触履歴を B が入手しその履歴に C と頻繁に会っている時間や場所が含まれていた時，B は C と関係ない状態で C が A と接触しているという情報を入手できるようになる．この情報が蓄積すると B は C の生活サイクルの一部を推測できるようになってしまうため，A の交友関係のプライバ

シが保護できていなことのみならず，C のプライバシが保護出来ていないといえる．このような問題に対処するために，秘匿共通集合計算という互いの生データを見せることなくデータの共通集合を発見する手法がある．

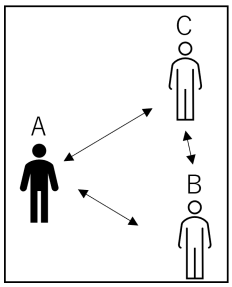
また，A,B,C などの名前は個人情報であるため，接触履歴を取得するために名前を他人に教えるとプライバシを保護できない．そのため，時間経過で変化するランダムな識別子を仮名として用いることで本人の特定を難しくする．本研究では BLE を用いて接触履歴を取得し，プライバシを保護しつつ共通集合を計算するスマートフォン向けアプリケーションを提案する．

仮名を用いない接触履歴の共通集合を求めると以下に述べるようなことができる．A と B の接触日時を調べる場合を考える．図 1.1a のように時刻 t_1 においてある場所に A,B,C,D の 4 人がいるとき，BLE の通信可能域にいる 4 人は自分の接触履歴にそれぞれの名前と接触の時刻を追加する．同様に，図 1.1b, 図 1.1c, 図 1.1d のように時刻 t_2, t_3, t_4 において接触が起きると図 1.2 のような接触状態になる．ただし $t_1 < t_2 < t_3 < t_4 < t_5$ とする．そして，図 1.1e のように時刻 t_5 において A と B が接触している状況で，互いが暗号化した接触履歴をやり取りすると，A と B は互いの接触履歴を見ることなく共通集合を計算することができる．この計算は秘匿共通集合計算 [1][2] と呼ばれる．この時，A と B は「時刻 t_1, t_2 で 2 回接触した．この時 C,D も同時に接触している」ことが分かる．例えば，図 1.1a が学校にいる状況で，1.1b が登下校時に乗る電車を表しているとき，A,B,C は同じ学校に所属し，同じ時間に同じ経路で下校しているということが推測できる．

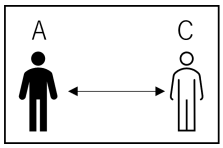
提案アプリケーションでは接触履歴に記録される名前は個人を特定できない仮名となっているため，A が B と接触した時に得られる情報は B の仮名となる．図 1.1 のような接触が起きた時，A と B の接触履歴はそれぞれ図 1.3a，図 1.3b のようになる．A,B は他の人と接触したときに，自分が使用している仮名を自分の接触履歴に記録する．これによって共通集合を計算したときに自分と相手の接触回数が分かるようになる．ここで時刻 t_2 と t_3 の間で仮名が更新されているとする．例えば，C の時刻 t_2 までの仮名を C とし，時刻 t_3 以降の仮名を C' とする．このとき、時刻 t_1 の C と時刻 t_3 の C を，A の接触履歴を使って同一人物として認識することはできない．同様に B は時刻 t_1, t_2 に接触した C や D と時刻 t_4 に接触した C,D を同一人物だと認識することはできない．そして時刻 $t_5 (t_5 > t_4)$ において A と B の間で接触履歴の共通集合を計算したときに A が得ることができる情報は，「B とは時刻 t_1, t_2 で 2 回接触している．そして時刻 t_1 では 2 人 (C と D) と同時に接触し，時刻 t_2 では 1 人 (C) と同時に接触している」となる．このように仮名を使うことで名前が分からなくなるため，プライバシを保護して接触履歴を扱うこ



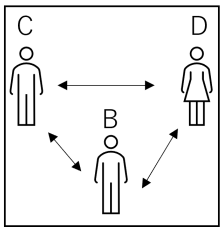
(a) A,B,C,D が接触している状態 (時刻 t_1)



(b) A,B,C が接触している状態 (時刻 t_2)



(c) A,C が接触している状態 (時刻 t_3)



(d) B,C,D が接触している状態 (時刻 t_4)

(e) A と B が接触履歴から共通集合を計算している状態 (時刻 t_5)

図 1.1: 接触履歴を計算するまでの行程

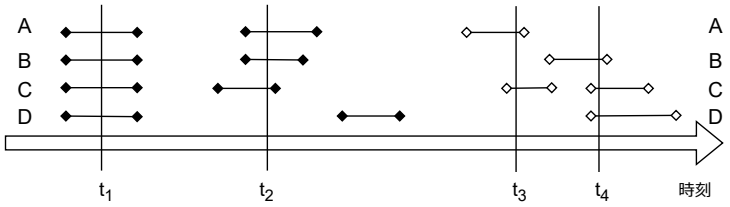
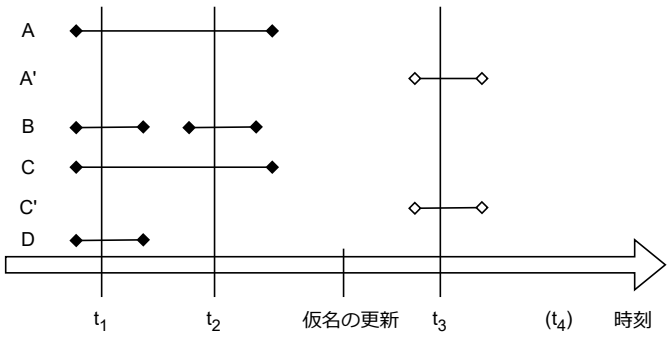


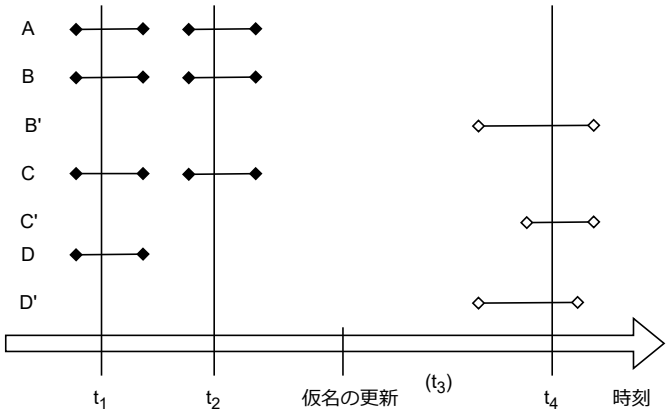
図 1.2: A,B,C,D の接触状態

とができるようになる．

しかし，秘匿共通集合計算をするため単純に互いの接触履歴を見せ合う時より，計算に長い時間がかかる．そこで本論文では様々なデータ量の接触履歴に対して秘匿共通集合計算を行い，それにかかる時間を計測し，許容できる時間とデータ量の組み合わせを発見



(a) A の接触履歴



(b) B の接触履歴

図 1.3: A,B の接触履歴

する．

本論文では，まず 2 章で関連研究を紹介する．3 章で本論文で用いる技術に対して説明する．4 章で提案アプリケーションの概要について説明する．5 章で提案アプリケーションの実装について説明する．6 章で提案アプリケーションを評価する．最後に 7 章でまとめと今後の展望について述べる．

第 2 章

関連研究

2.1 プライバシを保護した分散医療データ統合セキュリティシステム

医療の分野で患者のプライバシを保護しつつ，カルテ情報を利活用するために秘匿共通集合計算が使われている [3].

この研究では，患者のデータに対して秘匿共通集合計算を用いて突合する属性と，それに付随する属性のデータ統合をする方式 (Privacy-reserving Distributed Data Integration:PDDI) を提案している．この PDDI は以下のような特徴がある．

1. 関わったすべての機関は共通に含まれるユーザ以外の情報を知ることはできない.
2. 突合に利用する情報はどこの機関にも移動しない
3. 各機関の処理時間は機関数に影響しない
4. 第三者機関によるデータ管理が不要

この研究では，PDDI の実装の設計として，以下の点が重要視されている.

1. 核となるロジックのみ実装すること
2. 設定はすべて設定ファイルで行えること
3. デプロイメントは極力自動で行えること
4. Web 通信のために Restful API を使用すること
5. 通信をできる限りまとめてすること

まず，核となるロジックのみを実装することで，PDDI の利用者が使用環境に合わせてカスタマイズすることができるようになる．また，従来の PDDI ソフトウェアは複数のシス

テムを用いていたため，設定方法が複雑であったが，設定をするファイルを一つにまとめることで，容易に設定できるようになった．さらに Docker を利用することで，環境構築が簡単になるため利用がしやすくなる．通信のために Restful API を使用することで，他のアプリケーションとの連携が容易になる．そして，通信をまとめてすることで，ネットワーク利用時のパフォーマンスも改善されている．このような設計で PDDI システムを実装することで，利用者は導入が容易になり，直感的で簡単な操作でシステムを利用することができるようになる．PDDI システムのパフォーマンス評価として 100,1000,10000 のデータ量で実行時間を計測しており，それぞれの最大値が 5.14 秒, 36.64 秒, 365.72 秒となった．

PDDI の適用事例としてがん検診データとがんの医療情報のデータへの適用について検討されている．ここではがんセンターが持つ癌の登録情報と，市が持つ癌検診のデータに PDDI を適用することで，個人情報を機密にしたまま癌検診率と癌重症化率を出力することができることが紹介されている．

2.2　照合タグを用いた秘匿共通集合計算プロトコルとその応用

照合タグを用いて秘匿共通集合計算をする方式が紹介されている [4]．

ここで，照合タグとは集合の各要素を推定困難な一意のデータに変換したものを表す．この照合タグを利用することで，同一または類似の集合に対して繰り返し共通集合を求めることが容易になる．ここでは，この性質が有効に働く例としてキャンセラブルバイオメトリクスへの適用について述べられている．キャンセラブルバイオメトリクスは，生体認証に用いる登録情報から生体情報を復元させない技術である．また，登録情報を更新することもできる．しかし，登録情報の更新時には，大量の暗号化された生体情報と各々の生体情報を照合する処理などが必要であり処理効率が悪い．そこで，ここで紹介された照合タグを用いた秘匿共通集合計算を用いると処理効率の向上を見込めるようになる．

2.3　効率的な他機関の Private Set Intersection

Bloom Filter を用いて効率的に秘匿共通集合計算をする方法が紹介されている [5]．

ここでは提案方式として，クライアントが Bloom Filter を生成し，サーバがフィルターを統合し，クライアントが共通要素を出力することで秘匿共通集合計算を行っている．サーバ，クライアントが持つ集合のサイズを w とし，フィルターの数を n としたとき比

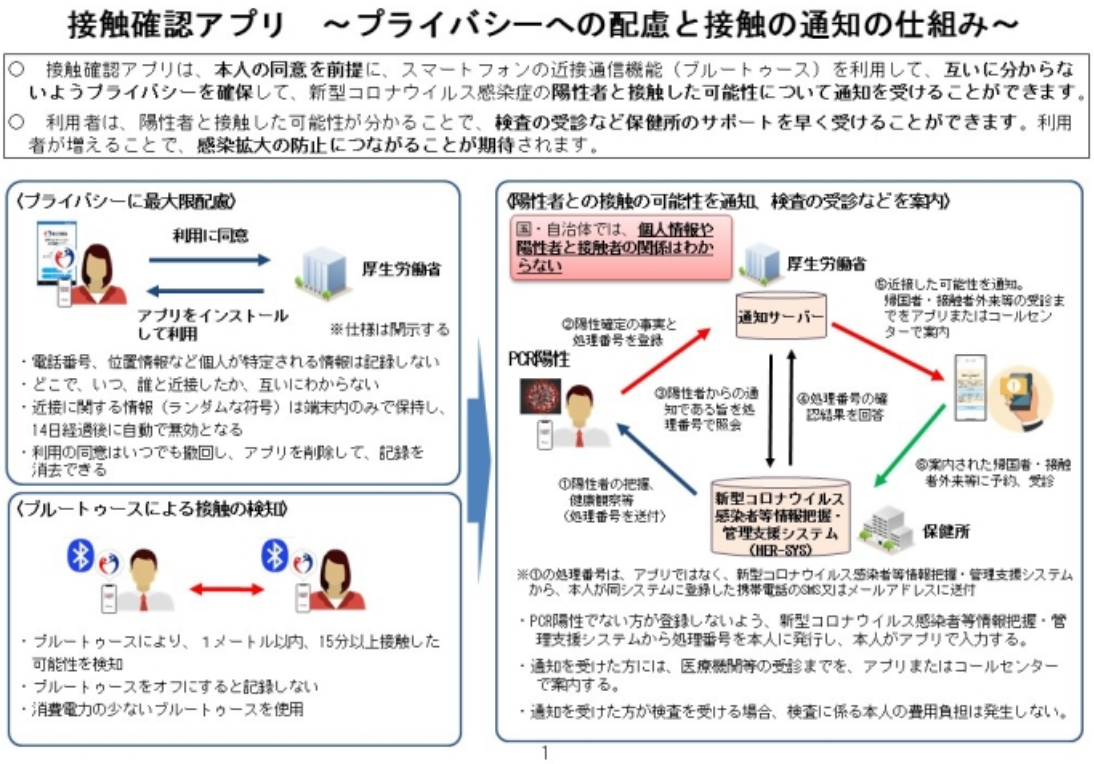
較対象の既存研究で挙げられている Paillier 暗号を利用したものの計算量が $O(n^2 + nw^2)$ となるのに対して，ここで提案されたものの計算量は $O(w^2n^2)$ となっている．

2.4　ランダム化された MAC アドレスの同定手法

パケットの受信時刻と RSSI (Received Signal Strength Indicator) からランダム化された MAC アドレスを同定する手法が存在する [6][7]．RSSI とは受信信号強度のことを表す．この研究で想定されている環境は，物理的に移動しない BLE (Bluetooth Low Energy) 端末を用いる環境である．RSSI から BLE パケットを受信する端末と送信する端末との距離が分かる．また，ある MAC アドレスを持つ端末から発せられたパケットを受信した初回の時刻と最終時刻を記録しておくことで，端末が MAC アドレスを変更する間隔が分かるようになる．これらの情報を蓄積し，比較することでランダム化された MAC アドレスから端末を特定することができる．

2.5　新型コロナウイルス接触確認アプリ COCOA

新型コロナウイルス接触確認アプリ COCOA[8] は，厚生労働省が 2020 年 6 月にリリースした接触確認アプリである．このアプリは図 2.1 のように，Bluetooth を用いて接触履歴を取得し，接触者の誰かがコロナウイルスの陽性登録をしたときにサーバを介して陽性者との接触を知らせるシステムである．Apple と Google の Exposure notification API[9][10] を使用しているため，消費電力を抑えてアプリを利用することができ，利用者本人が履歴の削除をすることができるようになっているなどプライバシに配慮されている．



(新型コロナウイルス接触確認アプリ COCOA アプリの概要 [8] から引用)

図 2.1: COCOA の仕組み

第 3 章

準備

3.1 BLE

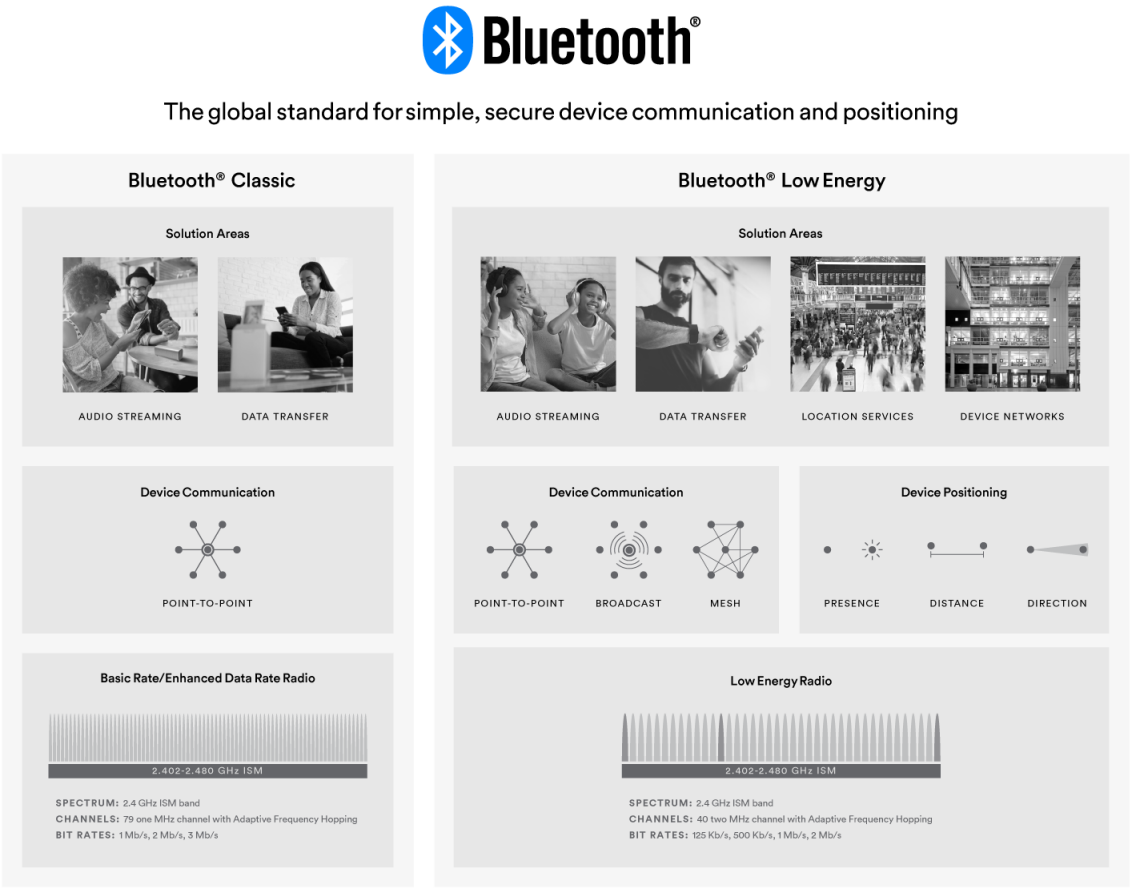
BLE とは Bluetooth Low Energy の略であり，2010 年に Bluetooth Special Interest Group(SIG)[11] によって，Bluetooth 4.0 の仕様の一部として導入されたものである。図 3.1 に Bluetooth Classic (Classic) と BLE との違いが記された画像を示す．BLE は Crassic に比べ多くの場面で利用されている．また，BLE はポイントツーポイント，ブロードキャスト，メッシュ 3 つのの通信トポロジーをサポートしており，一つの端末が複数の端末との接続が可能である．さらに BLE は位置情報を利用することもできる．

BLE を用いて通信する際には一つの端末がセントラル (Central)，もう一方の端末がペリフェラル (Peripheral) の役割を果たす．セントラルの役割を持つ端末は，通信を制御する役割を持ち，ペリフェラルの端末に対して要求を出す．一方，ペリフェラルの役割を持つ端末は，セントラルからの要求に応えるように通信をする．また，セントラル端末は，機器の性能にもよるが，複数のペリフェラル端末と同時に接続が可能である．

BLE では，ペリフェラルの端末が広告 (Advertising) パケットというものをブロードキャストし，それをセントラルの端末が受信 (スキャン) することで，通信が行われる．この広告パケットにはペリフェラル機器の名前などのデータを含めて発信できる．

3.2 秘匿共通集合計算

秘匿共通集合計算 (Private Set Intersection: PSI) [1][2][12][13][14][15] とは二者間で，それぞれが保持している集合を秘密にしたまま，それらの共通集合を求める計算法である．この手法は，集合にプライベートな情報が入っているときに使われることが多い．あ



(Bluetooth Special Interest Group(SIG)[11] から引用)

図 3.1: Bluetooth Classic と Bluetooth Low Energy の違い

る異なる二つの機関が持つ集合の共通要素を求めたいとき，簡単な方法はどちらか一方に集合を渡す，または第三者にお互いの情報を渡して集合演算をしてもらうことである．しかし，このような方法ではデータのプライバシの保護をすることができない．秘匿共通集合計算を用いると，異なる二つの機関が所有する集合データに対して，お互いの集合の内容を明かすことなく共通の集合を求めることができる．

秘匿共通集合計算の仕組みを図 3.2 に示す．機関 A，B がそれぞれ集合 $X = \{x_i : 1 \leq i \leq |X|\}, Y = \{y_j : 1 \leq j \leq |Y|\}$，秘密鍵 s, c を持っている状態を考える．A がサーバ側，B がクライアント側である．まず，A は X の各要素をハッシュ関数 H を用いてハッシュ化する，すなわち $H(X) = \{H(x_i) : 1 \leq i \leq |X|\}$ を計算する．そして秘密鍵 s で暗号化した $H(X)^s$ を B に送る．ここで $H(X)^s$ は集合 $H(X)$ の各要素 n を s 乗し

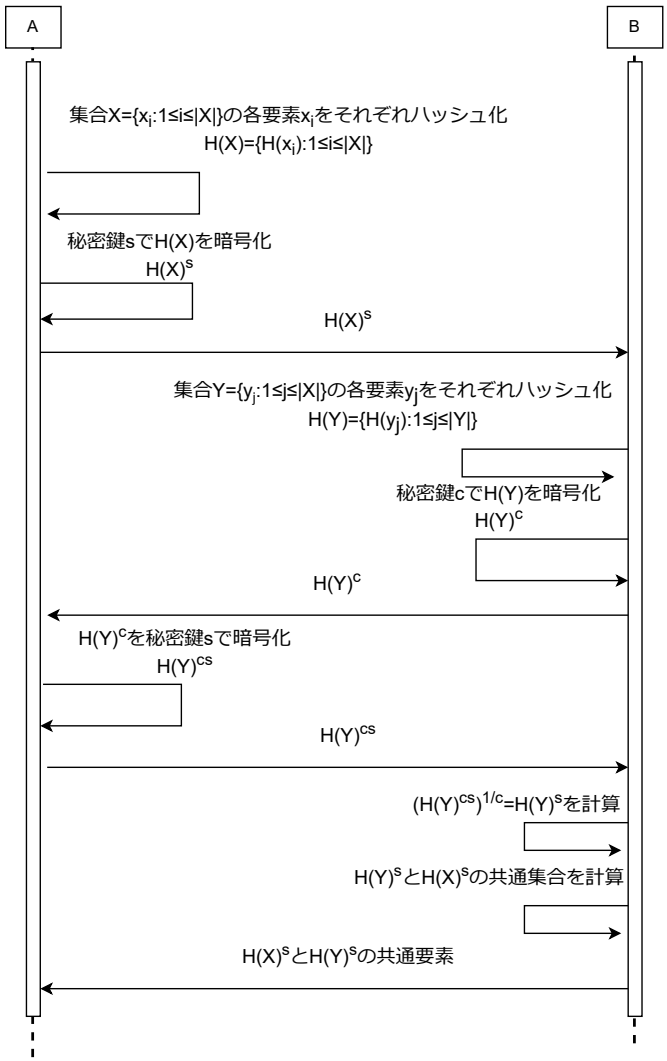


図 3.2: 秘匿共通集合計算の動き

た n^s を要素とする集合を表すこととする．以降も集合の累乗は同様に各要素の累乗を表す．同様に，B は Y の各要素をハッシュ関数 H を用いてハッシュ化する，すなわち $H(Y) = \{H(y_j) : 1 \leq j \leq |Y|\}$ を計算する．そして秘密鍵 c で暗号化した $H(Y)^c$ を A に送る．次に A は送られてきた $H(Y)^c$ を自身の秘密鍵で暗号化した $H(Y)^{cs}$ を B に送る．そして，B は送られた $H(Y)^{cs}$ を以下のように自身の秘密鍵で復号する．

$$(H(Y)^{cs})^{\frac{1}{c}} = H(Y)^s$$

B は $H(X)^s$ と $H(Y)^s$ の共通部分を A に知らせることによって，A，B は集合 X と集合 Y の共通部分を知ることができる．これらの処理が終わったとき，A と B はそれぞれ

$X \cap Y$ と相手の集合の要素数 $|X|$ または $|Y|$ を知ることができる．本研究では集合 X, Y を接触履歴として両者で共通する接触を求めるために秘匿共通集合計算をする．

3.3　BoringSSL

BoringSSL[16] は，Google が管理している TLS プロトコルのオープンソース実装である．OpenSSL から分岐したものであり，OpenSSL[17] と互換性がある．また，Android OS もバージョン 6.0(API レベル 23) で OpenSSL から BoringSSL ライブラリに移行しているため，本研究では，BoringSSL ライブラリを利用して秘匿共通集合計算をする．

第 4 章

提案アプリケーション

4.1　使用想定環境

ユーザは日常生活の中で提案アプリケーション (以降アプリと呼ぶ) をバックグラウンドで動作させ，自分の仮名を周辺に BLE でブロードキャストし，自分が受信した端末の仮名と時刻を接触履歴として記録する．ユーザがよくあっている人かもしれないという人を見つけた時，その人と接触履歴の秘匿共通集合計算を実行する．接触履歴の秘匿共通集合計算は過去 3 ヶ月，1 ヶ月，全てのデータから選択して行われる．この時，毎週同じ曜日の同じ時間に接触しているなど，規則性が見られたならば，次の接触が起きるであろう時刻に使用している仮名を交換する．このようにあらかじめ将来の仮名を渡すことによって，これ以降は定期的な接触が続く限り秘匿共通集合計算を行わなくても，接触を検出することができるようになる．これらの処理は図 4.1 のようにすべて使用者同士のスマートフォンのみで完結し，外部にサーバなどを必要としない．

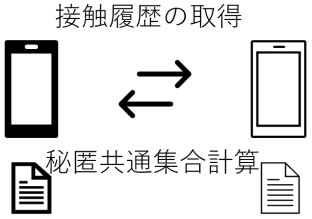


図 4.1: 提案システムのイメージ図

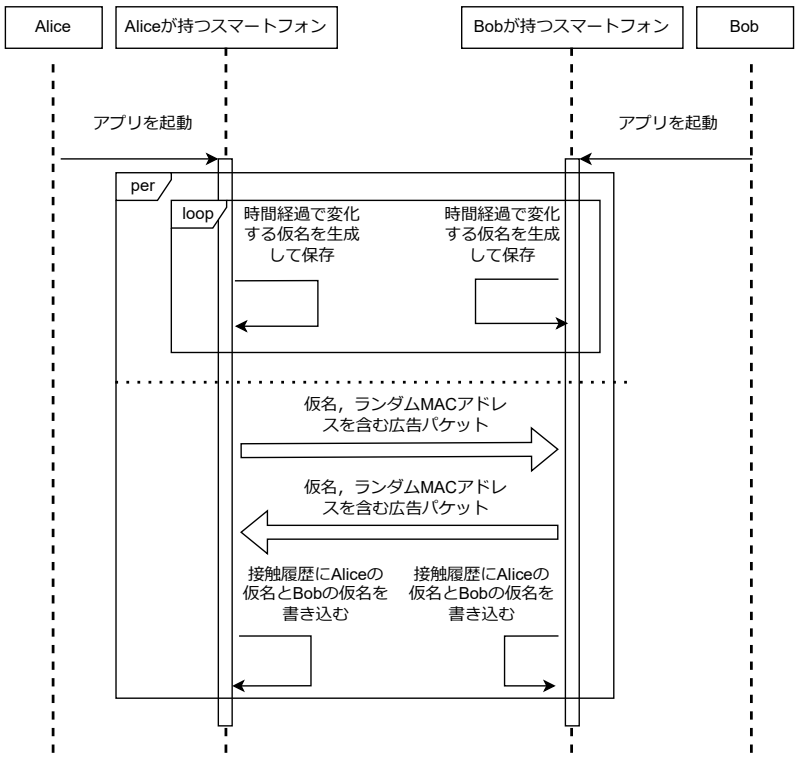


図 4.2: アプリ実行中の動作

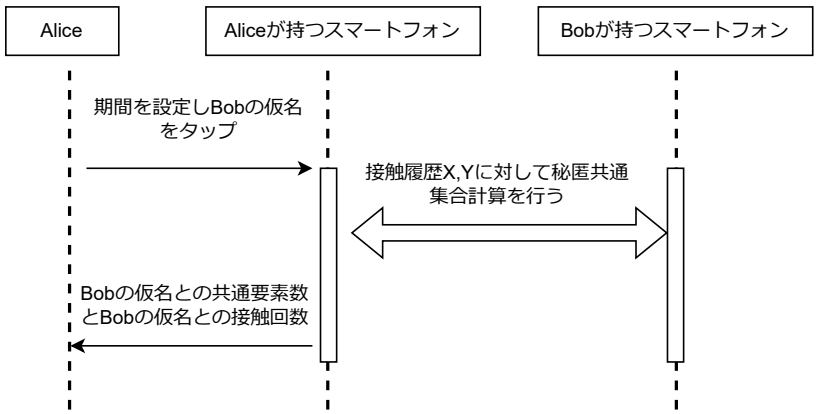


図 4.3: Alice と Bob の接触履歴の共通集合を求めるときの動作

4.1.1　システムの全体像

図 4.2，4.3 に提案システムのシーケンス図を示す． 図 4.2 にアプリ実行中の動作を示す．このように，ユーザー Alice がアプリを起動すると，アプリは仮名の生成をする．仮

名の生成と並行して後述する広告パケットのブロードキャストをするため図 4.2 において per で囲んでいる．この仮名は，デバイスの名前やユーザーの名前の代わりとなるものであり，時間経過で変化する．提案システムでは仮名を 15 分ごとに変化させる．そのため，15 分間隔で loop 内の処理が実行される．この処理と並行してアプリは広告パケットをブロードキャストする．広告パケットには生成される仮名とランダム化された BLE の MAC アドレスが含まれる．Alice と Bob が BLE の通信域内に入り互いが広告パケットをスキャンすると接触したとみなされ，互いの接触履歴に Alice と Bob の仮名が記録される．このとき接触相手の仮名のみではなく，自分の仮名を接触履歴に書き込むことで後にする秘匿共通集合計算の時に相手との接触回数が分かるようになる．そして，図 4.3 にはユーザー Alice が Bob に対して接触履歴の共通集合を求めるときの動作を示す．ユーザーが任意のタイミングで現在接触している人の仮名をタップしたとき，設定された期間の接触履歴を用いて選択したデバイスとの秘匿共通集合計算を実行する．

4.1.2　COCOA との違い

提案したアプリケーションは 2.5 節で紹介した COCOA とよく似ている．しかし，COCOA は新型コロナウイルス陽性者との接触を検知したときの通知などのためにサーバが利用されている．しかし，提案システムにはサーバは利用されておらず，使用者のスマートフォンのみでシステムが完結する．そのためサーバへの攻撃による情報の流出のリスクが提案システムにはない．しかし，使用者のスマートフォンが攻撃されると，その使用者の接触履歴が漏れてしまう．

4.2　想定される脅威

4.2.1　仮名を使用しないとき

使用者が仮名を使用しない状況を考える．図 4.4 のように提案アプリを使用していない攻撃者が近くにいた場合，攻撃者に実名が知られていしまうためアプリ使用者のプライバシが保護されない．BLE の広告パケットをスキャンするアプリは簡単に利用できるため，攻撃者は容易に提案アプリの使用者の名前を知ることができる．また，ブロードキャストする名前をニックネームなどの実名ではないものにした場合でも，名前が変化しないため，端末の追跡をすることができてしまう．これらの脅威を回避するために提案システムでは後述する時間経過で変更される仮名を採用する．なお 2.4 で述べた方法などで仮名から端末が特定される可能性があることに注意が必要であるが，本研究ではこの特定は行わ

図 4.4: 仮名を使用しないときの問題のイメージ

図 4.5: 成りすましを行うことができる問題のイメージ

ない．

4.2.2　仮名を使用するが，接触履歴を暗号化しないとき

4.2.1 節に挙げた問題を回避するために時間経過で変更される仮名を使用するときを考える．仮名により，使用者につながる情報をブロードキャストすることがなくなるためプライバシの保護につながる．また，仮名は時間の経過で変化するため，端末の追跡をすることもできなくなる．しかし，仮名を利用しているが，接触履歴を暗号化していないと起こりうる問題について考える．図 4.5 のようにユーザ A，B，攻撃者 M が存在するときを考える．A と M が接触履歴を交換したときに M は A の接触履歴を入手することができる．その後 B と M が接触履歴を交換し，接触回数を表示する．そのとき，M は入手した A の接触履歴を B と交換することができる．このように暗号化されていない接触履歴を交換すると，他人に成りすますことができるようになる．また，図 4.6 のように A が B と接触し，別の時刻で C が B と接触した後，A が C と接触履歴を交換する状況を考える．この時，A，B，C の仮名は変更されていないとする．A と C が接触履歴を交換した際に，A，C はそれぞれ互いが B と接触していたことが分かってしまう．この時，B の知らないうちに B の情報 A や C に流れてしまう．このように接触履歴は個人の情報を含むため生のデータのまま交換することは危険である．そのため接触履歴の交換の際には暗号

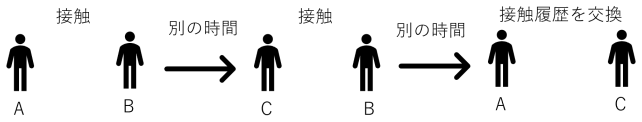


図 4.6: 接触履歴を暗号化しない場合の問題のイメージ

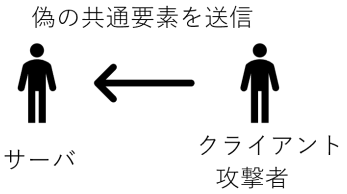


図 4.7: 暗号化した接触履歴を用いるときの問題のイメージ

化する必要がある．

4.2.3　仮名を利用し接触履歴を暗号化したとき

4.2.2 節に挙げた問題を回避するために暗号化した接触履歴を交換するときを考える．暗号化した接触履歴を交換することで，交換相手が誰と接触していたかという情報が漏れなくなる．しかし，暗号化した接触履歴から接触回数を求めるための計算が複雑になり，その計算過程で問題が発生する．秘匿共通集合計算の実行過程の仕様上，暗号化した接触履歴同士の共通要素を求め送信する役割はクライアントである．図 4.7 のように攻撃者がクライアントの役割となった場合，偽の共通要素を送ることができる．この時サーバは偽の情報を送られたことを判別できない．提案システムにおいてこのような攻撃が行われると，顔見知りでない人を顔見知りとして認識してしまう．そのため，共通要素を求める処理を互いにするなどして，偽の共通要素を送られたとしてもそれを検知できるシステムになっていると良い．

4.3　仮名

アプリが生成する仮名は 15 分ごとに変更される．仮名の作成方法は 2.5 節で紹介した Exposure Notification[10] と同様である．これは，ユーザーのプライバシを保護するための仕様である．仮名の代わりに，デバイス名やユーザー名などの変化することが少なく，個人を特定しやすい情報をブロードキャストするとデバイスの追跡をすることが容易にで

きてしまうため，仮名を用いる．

以下のように仮名 RN の作成をする [10].

1. 以下の式を用いて一時公開鍵 tek_i を作成する
   $i = \left\lfloor \frac{in(t)}{rp} \right\rfloor \times rp$
   $tek_i = CRNG(16)$
   ただし
   t: 現在時刻 (タイムスタンプ (秒))
   $in(t) = \frac{t}{60 \times 10} : IntervalNumber$
   rp: 一時公開鍵の有効期間 (RollingPeriod) 24 時間
   $CRNG(n)$:n バイトの暗号用乱数を生成する関数 (Cryptographic Random Number Generator)
2. 以下の式を用いて一時公開鍵 tek_i から 16 バイトの一時識別鍵 tik_i を作成する
   $tik_i = HKDF(tek_i)$
   $HKDF$ は RFC5869[18] で定義されている HMAC に基づく鍵導出関数である (HMAC-based Key Derivation Function).
3. 以下の式を用いて一時識別鍵 tik_i から仮名 $RN_{i,j}$ を作成する
   $RN_{i,j} = AES_{128}(tik_i, Data_j)$
   $AES_{128}(key, Data)$:AES128 で暗号化する関数
   j: 仮名を生成する時刻
   $Data$:$in(j)$ が含まれるデータ

16 バイトの一時公開鍵や一時識別鍵を使用することで衝突の可能性が低くなり，偽陽性のリスクが低減される．

4.4　接触履歴

接触履歴は，端末が発する広告パケットを 5 分間隔でスキャンすることで取得する．各端末は，仮名の情報を含めた広告パケットをブロードキャストする．そして，端末は 5 分間隔でスキャンを繰り返し，得られた広告パケットに含まれた仮名と，時刻を接触履歴として保存する．この時，Alice の接触履歴に Bob の仮名しか登録されていないと Alice の接触履歴には Alice の仮名がなく，Bob の接触履歴には Bob の仮名がない状態となるため，接触履歴の共通集合計算をする際に互いの共通集合として Alice と Bob の接触が検出されない．そのため，接触の際にはユーザー自身の仮名も登録するようにする．

4.5　アプリがする秘匿共通集合計算

3.2 節で説明した秘匿共通集合計算をする．秘匿共通集合計算は 2 者が持つスマートフォンのみで行われるものであり，外部にサーバなどを実装する必要はない．図 3.2 において A を Alice，B を Bob とすると，集合 X は Alice が持つ 1 ヶ月間の接触履歴の集合となる．同様に集合 Y は Bob が持つ 1 ヶ月間の接触履歴の集合となる．暗号化には楕円曲線暗号 [19] を用いる．楕円曲線暗号の鍵生成について説明する．Alice は公開されている EC ドメインパラメータ $D = (q, FR, a, b, G, n, h)$ を持っているとして，以下のように鍵が生成される．

1. Alice は $[1, n-1]$ の区間の整数 d をランダムに選択する．
2. $Q = dG$ を計算する．
3. Alice の公開鍵を Q，秘密鍵を d とする．

楕円曲線暗号を用いることで RSA 暗号を用いる方法と比べて少ないビット数で同じ安全性を保つことができる [20]．相手に漏れる情報は，接触履歴に含まれる仮名の総数，自分と接触していた時の自分の仮名，自分と相手の両方がほぼ同時に接触した人が使っていた仮名となる．

アプリケーションとして実装する動作は図 4.8，4.9 のようになる．図 4.8 ではサーバの秘匿共通集合計算の動作を表している．サーバは，PSI 開始のボタンを押したときデータベースから自分の接触履歴を読み込む．次に接触履歴を暗号化してクライアントに送信する．その後，クライアントが暗号化した接触履歴を受け取る．この時，受け取る集合はクライアントの接触履歴である．そして受け取った接触履歴を暗号化してクライアントに送信する．最後にクライアントが計算した共通要素を受け取ってサーバの秘匿共通集合計算が終了する．次に，図 4.9 にクライアントの秘匿共通集合計算の動作を示す．クライアントもサーバ同様 PSI 開始ボタンを押して秘匿共通集合計算を開始する．後述するが，PSI 開始ボタンはサーバよりも後に押さなければならない．クライアントはサーバの接触履歴の受け取りから始まる．次にデータベースを読み込み接触履歴の暗号化をする．そして暗号化した接触履歴をサーバに送信する．その後，サーバから送られてくる接触履歴を受け取る．この時受け取る接触履歴はクライアントのものだが，サーバ，クライアント両方の鍵で暗号化されている．この受け取った接触履歴を復号する．この状態でクライアントは互いの接触履歴を所持しており，その二つの接触履歴はともにサーバの鍵で暗号化されている．これらの接触履歴の共通集合を求めサーバに送信する．以上の動作で秘匿共通集合

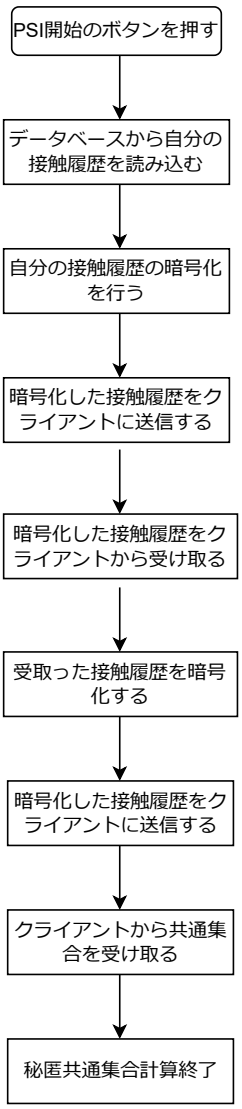


図 4.8: サーバのフローチャート

計算が終了する.

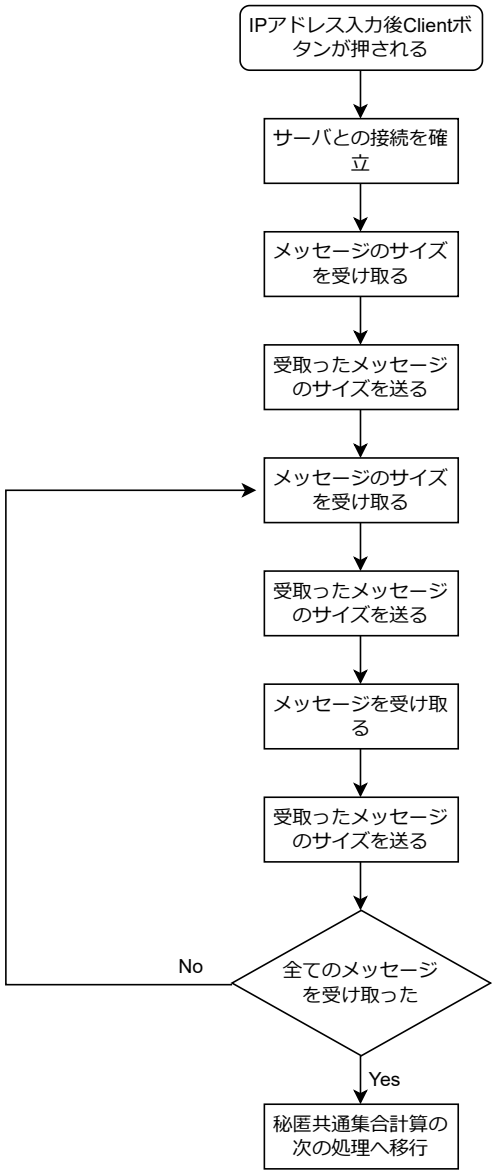


図 4.9: クライアントのフローチャート

第 5 章

実装

5.1 開発環境

開発環境を表 5.1 に示す．最小 SDK とは Android アプリ開発時に設定するものであり，これに対応したバージョン未満の OS を持つ端末にアプリをインストールすることはできない．本研究では，アプリ作成のために Kotlin を使い，アプリ内で BoringSSL のライブラリを使用するために，C++ を用いる．現在の実装では，仮名を生成する機能の実装はできていない．

5.2 BLE

BLE の実装について述べる．BLE の機能を使用するために AndroidManifest.xml ファイルに以下の権限を追加した．

- android.permission.BLUETOOTH_ADVERTISE
- android.permission.BLUETOOTH_CONNECT

表 5.1: 実装環境

開発環境	Android Studio
最小 SDK	31 (Android12)
言語	Kotlin, C++
テスト端末	Google Pixel 7

- android.permission.BLUETOOTH_SCAN
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION

AndroidManifest.xml とは，アプリケーションの設定に関する内容が記述されるファイルであり，特定の Android の機能を利用するときにはこのファイルに権限を追加する必要がある．

提案アプリではアプリをスタートしたときに BLE の広告パケットをブロードキャストする．広告パケットのブロードキャストに必要なことは大きく分けて二つある．一つ目はガットサーバの設定である．この設定で広告パケットの UUID やコールバックを設定する．ここで UUID とは Universally Unique IDentifier の略称であり，一意の ID のことである．スキャンの際に特定の UUID を持つ広告パケットをフィルタリングできる．またコールバックの設定とは，BLE 接続された時の処理を設定することである．今後このコールバックの処理として IP アドレスの送信をする予定である．二つ目の処理は広告パケットの設定である．本研究ではこの設定の際にデバイス名を仮名に変更する処理をする．

次に広告パケットのスキャンについて述べる．本研究ではアプリのスタート時に広告パケットをスキャンし，発見したデバイスの名前とスキャン時刻をデータベースに登録する．今後の実装として定期的にスキャンするためのタイマー処理を追加する必要がある．

5.3　Android Studio と BoringSSL

Android Studio で BoringSSL ライブラリを使用するためには，BoringSSL をビルドし，バイナリファイルを生成する必要がある．ここで必要なツールは，バージョン 3.12 以上の CMake，Ninja，Android NDK である．必要なツールがインストールされている WSL2 を用いてプログラム 5.1 に記されているコマンドを実行した．また本研究ではデバッグのやりやすさを考慮して静的ライブラリを生成した．

プログラム 5.1: BoringSSL のビルドコマンド

```
1    $ cmake -DANDROID_ABI=arm64-v8a -DANDROID_PLATFORM=android-31 -
         DCMAKE_TOOLCHAIN_FILE=/home/fcs2023/Android/Sdk/ndk
         /25.2.9519653/build/cmake/android.toolchain.cmake -GNinja -B
         build -DBUILD_SHARED_LIBS=OFF
2    $ ninja -C build
```

ここで，Android の cpu アーキテクチャを arm64-v8a，Android SDK のバージョンを 31 と設定した．そのため，条件を満たさない Android 搭載のスマートフォンでは提案アプリケーションを実行することはできない．

プログラム 5.1 の実行が成功すると，libcrypt.a，libdecrepit.a，libssl.a という 3 つのファイルが生成される．これらのファイルを Android プロジェクトの任意のディレクトリに保存し，BoringSSL のヘッダファイルをプロジェクト内に移動する．そして，プログラム 5.2 のようにプロジェクト内の CMakeLists.txt にライブラリを追加することを記述する．また，アプリケーションのバージョンなどを設定するファイルである build.gradle 内に，abiFilters 'arm64-v8a' と記述することで，arm64-v8a の cpu を持つスマートフォンにのみアプリをインストールさせることができる．

プログラム 5.2: BoringSSL の CMakeLists.txt へのライブラリファイルの追加

```
1   add_library(crypto STATIC IMPORTED)
2   set_target_properties(crypto
3           PROPERTIES IMPORTED_LOCATION
4           ${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libcrypto.a
                )
5
6   add_library(decrepit STATIC IMPORTED)
7   set_target_properties(decrepit
8           PROPERTIES IMPORTED_LOCATION
9           ${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libdecrepit
                .a)
10
11  add_library(ssl STATIC IMPORTED)
12  set_target_properties(ssl
13          PROPERTIES IMPORTED_LOCATION
14          ${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libssl.a)
15
16  target_include_directories(kotlinpsi PRIVATE ${PROJECT_SOURCE_DIR}/
        include/)
17
18  target_link_libraries(kotlinpsi
19          crypto
20          ssl
21          decrepit
```

表 5.2: 接触履歴を保存するデータベースの構成

日付 (LocalDateTime)	仮名 (ByteArray)
2024/1/19 13:00	728

```
22              ${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libcrypto.a
23              ${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libssl.a
24              ${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libdecrepit
                    .a)
```

5.4　接触履歴の取得

提案システムでは取得した接触履歴は個人のスマートフォンに保存される．本研究では Room[21] という Android Jetpack に含まれるライブラリを使用して，表 5.2 のように接触履歴をデータベースとして保存する．

5.4.1　Room の導入

Room[21] を Android アプリケーションに導入するには依存関係を追加する必要がある．app ディレクトリ内の build.gradle にプログラム 5.3 のような文を追加しなければならない．

プログラム 5.3: Room のための依存関係 (build.gradle ファイル)

```
1
2     dependencies {
3
4       implementation 'androidx.room:room-ktx:2.6.0'
5       implementation 'androidx.room:room-migration:2.6.0'
6       implementation "androidx.room:room-rxjava2:2.6.0"
7       implementation "androidx.room:room-rxjava3:2.6.0"
8       implementation "androidx.room:room-guava:2.6.0"
9       testImplementation "androidx.room:room-testing:2.6.0"
10      implementation "androidx.room:room-paging:2.6.0"
11
12      implementation "androidx.room:room-runtime:2.6.0"
```

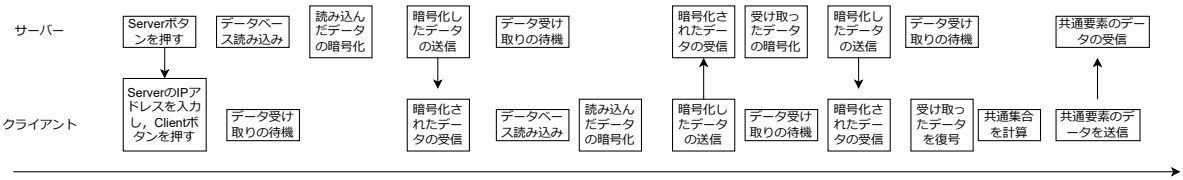


図 5.1: 秘匿共通集合計算の処理の流れ

```
13        annotationProcessor "androidx.room:room-compiler:2.6.0"
14        ksp "androidx.room:room-compiler:2.6.0"
15        implementation "androidx.room:room-ktx:2.6.0"
16
17
18        androidTestImplementation 'androidx.test.espresso:espresso-core
              :3.5.1'
19        androidTestImplementation "androidx.room:room-testing:2.6.0"
20    }
```

Room のデータベースを使用するために以下のようなファイル構成にする．

- Contact.kt　データベースの構成要素を決めるファイル．
- ContactDao.kt　データベースを操作するクエリ文などを記述するファイル．
- ContactDatabase.kt　データベースのバージョンや名前を決めるファイル．
- ContactConverter.kt　Room で扱うことができない型の変数を変換するファイル．
- ContactViewModel.kt　Dao で記述したクエリ文を実行するためのファイル．

本研究ではデータベースに格納する日時を LocalDateTime 型と定義したが，Room ではこの型が存在しないため，ContactConverter.kt ファイルを用いて String に変換して格納している．

5.5　秘匿共通集合計算

Android 端末上での秘匿共通集合計算について述べる．秘匿共通集合計算はユーザが顔見知りの人と接触しているときにする．図 5.1 に秘匿共通集合計算の処理の流れを示す．サーバは上段，クライアントは下段の処理を左側から順番に実行する．図中の「Server ボタンを押す」と「Server の IP アドレスを入力し，Client ボタンを押す」のように矢印でつながれた処理は矢印の始点につながれた処理を先に実行する必要がある処理である．ま

アルゴリズム 1 PSI のために自分の接触履歴を暗号化する疑似コード

1: データベースから自分の接触履歴 X を取得
2: **for** X の一つ一つを x としてループ **do**
3:　x の仮名を取り出す
4:　x の仮名と秘密鍵 (key) をパラメータとして C++ を呼び出す
5:　パラメータを C++ で扱えるように変換
6:　暗号化に使用する楕円曲線を P-256 で定義
7:　暗号化のために x の名前要素を BIGNUM 型変数 mes に変換
8:　EC_POINT 型の変数 ps を宣言
9:　暗号化のために鍵 key を EC_POINT 型に変換して ps に代入
10:　$ps \times mes$ を計算し，px に代入
11:　kotlin にパラメータを渡すため px の型を ByteArray に変換
12:　暗号化された接触履歴が格納されるリスト enc_mes_list に px を追加
13: **end for**

ずサーバの役割となるスマートフォンを持つ人がボタンを押すことで秘匿共通集合計算が実行される．サーバの役割を持つ人がボタンを押した後にクライアントの役割となるスマートフォンを持つ人がサーバの IP アドレスを入力しボタンを押す．ユーザはこれらの操作をするのみで接触履歴の秘匿共通集合計算をすることができる．Android が提供するライブラリ ViewModel を利用して，各処理ごとに変化する変数を利用することで図 5.1 のような流れに沿って処理するようにしている．

5.5.1　接触履歴の暗号化

図 5.1 のデータの暗号化について述べる．秘密鍵 key はあらかじめ BoringSSL ライブラリを利用して生成されている．この秘密鍵 key を利用して接触履歴の暗号化が行われる．接触履歴の暗号化は**アルゴリズム 1** のように行われる．まず，前述した Room ライブラリを使用し，ローカルデータベースを読み込む．そして暗号化をするために BoringSSL のライブラリを使用するため，暗号化する仮名と秘密鍵を引数として C++ を呼び出す．BoringSSL のライブラリを使用して Kotlin から受け取ったデータの型を C++ で使えるように変換する．本研究では暗号化に楕円曲線暗号を採用しているため，メッセージと鍵の積である $mes \times ps$ をすることで暗号化される．

5.5.2　暗号化した接触履歴の送受信

図 5.1 の「暗号化したデータの送信」,「暗号化されたデータの受信」の動作について述べる．本研究では Socket 通信を利用した．インターネットを利用するために AndroidManifest.xml ファイルに以下の権限を追加した．

- android.permission.INTERNET
- android.permission.ACCESS_NETWORK_STATE

暗号化した接触履歴の送信には**アルゴリズム 2** や図 5.2，受信には**アルゴリズム 3** や図 5.3 のように行われる．送信者は送信データとして List<ByteArray>で表されるデータを送信する．送受信時には，送信者がまず送る接触履歴の数を受信者に送り，受信者がそれを送信者に送り返す．この操作によって送受信者がメッセージの受信の準備が整ったことを確認する．その後送信するメッセージのサイズを送り合う．これらの処理によって送信者と受信者がデータを受け取る準備ができているかどうかを確認する．また，データの送受信はメインスレッドではないスレッドでする必要がある．そのため，Android のライブラリである lifecyclescope を利用して別のスレッドで処理している．

今回の実装ではサーバは通信相手を選択することはできない．複数のクライアントが同じサーバの IP アドレスを入力して通信を開始したとき，一番早くサーバと接続した端末と秘匿共通集合計算を行ってしまう．そのため，後述するが，選択した相手同士で秘匿共通集合計算をすることができるように改善していく必要がある．

5.5.3　接触履歴の復号と共通集合の計算

アルゴリズム 4 のようにして復号と共通集合の計算が行われる．暗号化と同様に復号には BoringSSL のライブラリを使用するため，必要なパラメータを引数として C++ を呼び出す．楕円曲線暗号で暗号化しているため復号には暗号化されたメッセージを鍵で割ればよい．本研究では鍵の逆元 $key\_inverse$ を求め，暗号化されたメッセージと鍵の逆元の積 $Y\_enc * key\_inverse$ を計算して復号する．

復号したクライアントの接触履歴とサーバの接触履歴はともにサーバの鍵で暗号化された状態である．これらを BoringSSL で定義されている関数 EC_POINT_cmp() を利用することで共通要素を見つけることができる．

アルゴリズム 2 暗号化した接触履歴を送信する疑似コード

Require: 受信者との接続が確立している，送信するデータのリスト *mes* を持っている
Ensure: 次の秘匿共通集合計算の処理に移る

```
1: mes のサイズを受信者に送る
2: 受信者からメッセージ mes_size を受け取る
3: if mes_size が mes のサイズと一致するとき then
4:     for mes の一つ一つを send_mes としてループ do
5:         send_mes のサイズを受信者に送る
6:         受信者からメッセージ mes_size を受け取る
7:         if mes_size が send_mes のサイズと一致するとき then
8:             send_mes を送信
9:             受信者からメッセージ mes_size を受け取る
10:            if mes_size が send_mes のサイズと一致しないとき then
11:                送信失敗とする
12:            end if
13:        end if
14:    end for
15: end if
```

5.5.4　アプリの UI と操作

図 5.4, 5.5, 5.6 に提案アプリの画面を示す．まず，アプリを起動した際に図 5.4a のような画面が表示される．表示されている START ボタンを押したとき，BLE の広告パケットのブロードキャストとスキャンが行われ，図 5.4b のような画面に遷移する．スキャンを行った時に発見したデバイスが図 5.4b のようにスマートフォンの画面中央に表示される．またそれと同時にスキャンの時刻とデバイスの名前 (仮名) がデータベースに追加される．スキャンの動作は今後定期的にするようにする必要がある．左上の DATABASE ボタンはこの後の評価のために用意したデータの csv を読み込むためのボタンである．真ん中のテキストである「あなたの仮名は 4482 です」というものは自分の仮名を伝えるものである．提案システムの完成形としてこの画面で現在の接触者を表示し，その中から一人を選択することで接触履歴の秘匿共通集合計算をすることを想定している．そのためアプリ使用者は自分の仮名を他者に教える必要があるためこのテキストが表示される．テキ

アルゴリズム 3 暗号化された接触履歴を受信する疑似コード

Require: 送信者との接続が確立されている
Ensure: 次の秘匿共通集合計算の処理に移る
1: 送信者からメッセージのサイズ mes_size を受け取る
2: 送信者に受け取ったメッセージのサイズを送る
3: **while** 受け取ったメッセージのサイズ mes_size の分だけループ **do**
4:　送信者からメッセージのサイズ $send\_mes\_size$ を受け取る
5:　送信者に受け取ったメッセージのサイズを送る
6:　**if** メッセージのサイズを正常に受け取ったとき **then**
7:　　送信者からメッセージを受け取る
8:　　受け取ったメッセージをリストに追加
9:　　送信者に受取ったメッセージのサイズを送る
10:　**end if**
11: **end while**

ストの下のラジオボタンは秘匿共通集合計算に用いる接触履歴の期間を設定するボタンである．期間を選択しなかった場合，全てのデータで秘匿共通集合計算が実行される．ラジオボタンの下の SERVER，CLIENT ボタンによって秘匿共通集合計算が実行される．秘匿共通集合計算においてサーバの役割となるスマートフォンを持つ人は SERVER ボタンをタップすると接触履歴の暗号化が開始される．クライアントの役割となるスマートフォンを持つ人は CLIENT ボタンの左側にあるテキストボックスにサーバの IP アドレスを入力してから CLIENT ボタンを押す．そうすると暗号化された接触履歴の受け取りが開始される．実装の都合上サーバ側がボタンを押してからクライアント側がボタンを押さなくてはならない．サーバ，クライアントはそれぞれ対応するボタンを押した時点で秘匿共通集合計算のための操作は終了である．秘匿共通集合計算が開始されると，サーバ，クライアントのスマートフォンははそれぞれ図 5.5a，5.6a のような画面に遷移する．ここで画面上部に現在どのような処理を行っているかのテキストが表示される．このテキストがない場合，大きいデータ数で秘匿共通集合計算を実行すると，計算中は画面が動かない．それではアプリが正常に動いているのかわからなくなってしまうため，現在の状態をテキストで表示するようにした．そして，秘匿共通集合計算が終了すると，図 5.5b，5.6b のような画面になる．ここでは相手と接触した日付と時刻が表示される．現状では，2024/1/19 15:00,2024/1/19 15:05 に接触していた際にその二つの時刻が表示されるが，

アルゴリズム 4 二重に暗号化されたクライアントの接触履歴に対して復号し共通要素を求める疑似コード

Require: サーバの鍵で暗号化されたサーバの接触履歴 X，サーバとクライアントの鍵で二重に暗号化されたクライアントの接触履歴 Y_enc

Ensure: 共通要素の送信処理に移行

1: **for** X の一つ一つを X_mes としてループ **do**
2: **for** Y_enc の一つ一つを Y_encmes としてループ **do**
3: X_mes，Y_encmes，鍵 key をパラメータとして C++ を呼び出す
4: パラメータを C++ で扱えるように変換
5: key の逆元を計算し $key\_inverse$ に代入
6: $Y\_enc \times key\_inverse$ を計算し，Y_dec に代入
7: Y_dec と X_mes を比較する
8: 比較して等しければ e を true，そうでなければ e を false として kotlin に返す
9: サーバ，クライアントそれぞれが対応する部分に e を格納
10: **end for**
11: **end for**

今後の課題として，このような場合のとき，2 回の接触ではなく 1 回の接触としたいと考えている．

5.6　今後の実装

今後は仮名の機能を実装する必要がある．現状は，アドバタイズする名前を乱数にしているため規則性がない．しかし，時間を元に生成される仮名を用いることができるようになれば顔見知りとして信頼できる人に将来の仮名リストを渡すことができる．信頼できる人を違うデータベースに登録することができるようになれば扱う接触履歴のサイズが小さくなるため，秘匿共通集合計算にかかる時間を短縮することができるようになる．

また，現在の実装では図 5.4b のように広告パケットをブロードキャストしている端末を表示している．今後の実装として，表示される仮名を選択した際に図 5.7 のような確認ダイアログを見せたのちに秘匿共通集合計算を行えるようにしていきたいと考えている．これによってサーバとクライアントが明確に通信相手を指定することができるようになる．BLE でサーバの IP アドレスを送信することでクライアントはサーバの IP アドレスを入力する手間を省くことができる．

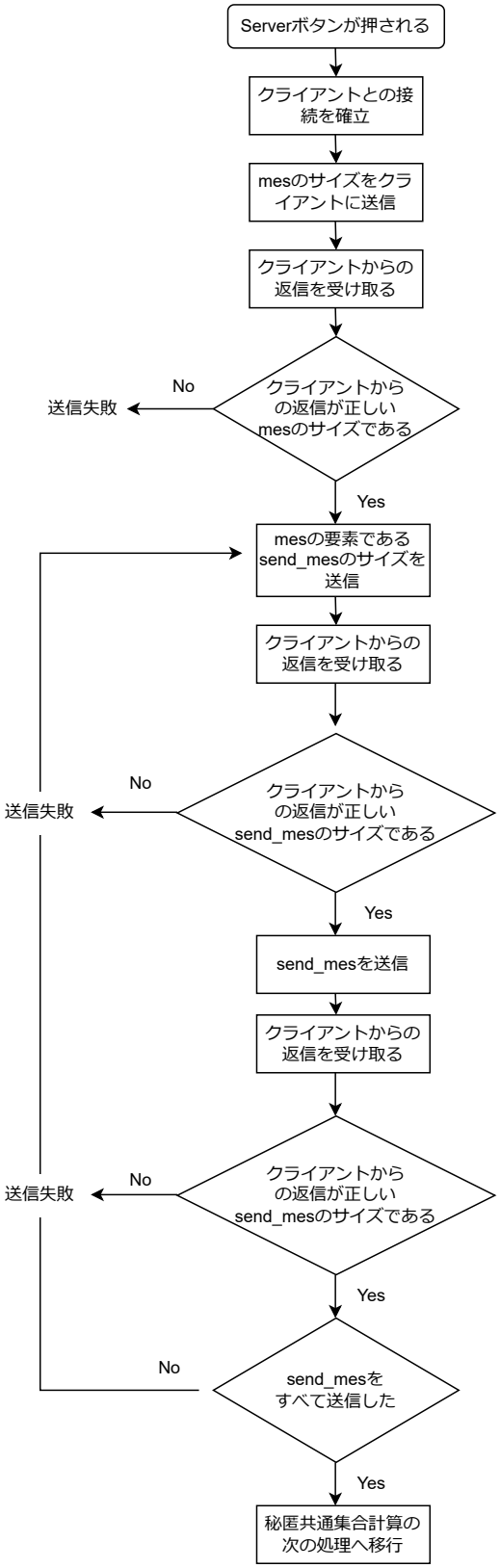


図 5.2: 送信処理のフローチャート

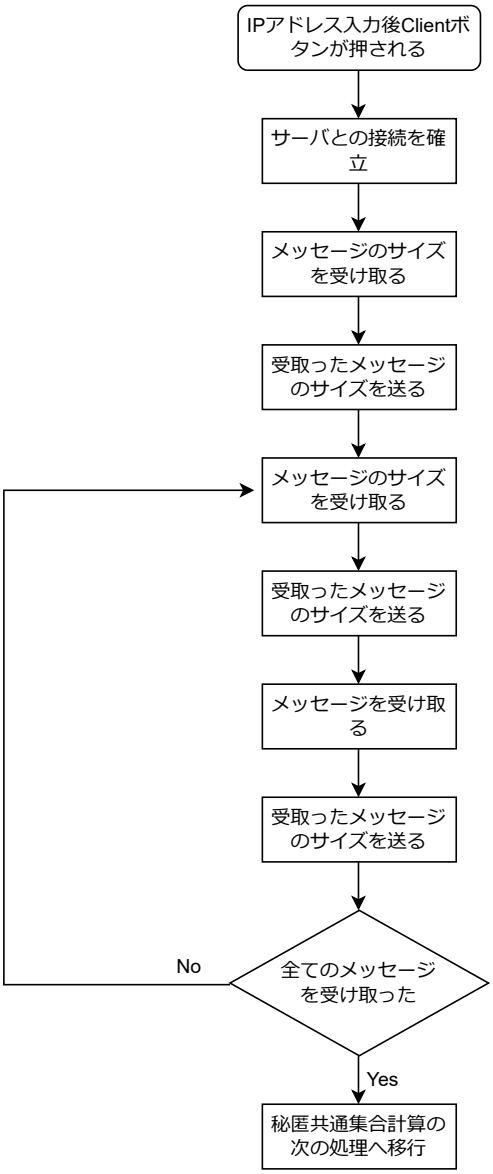
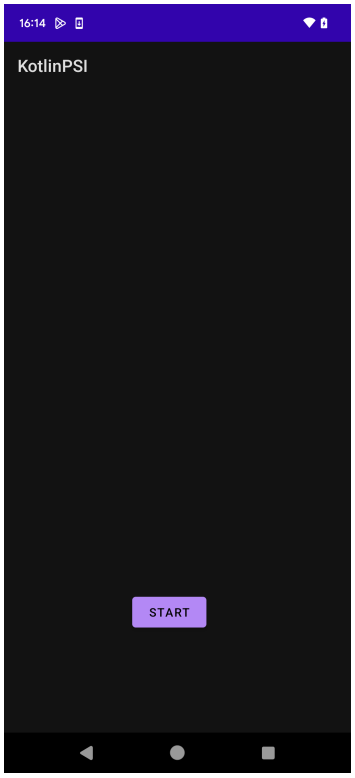
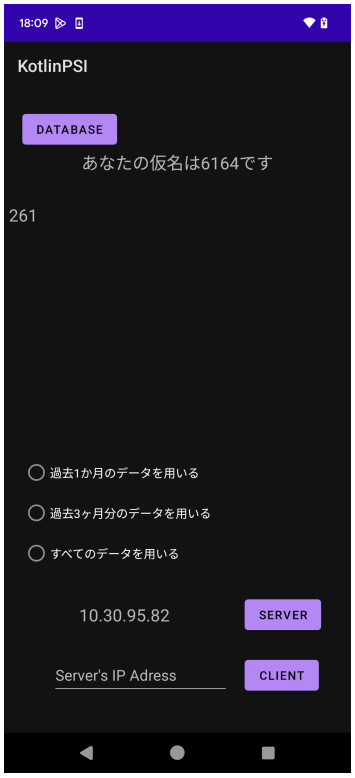


図 5.3: 受信処理のフローチャート

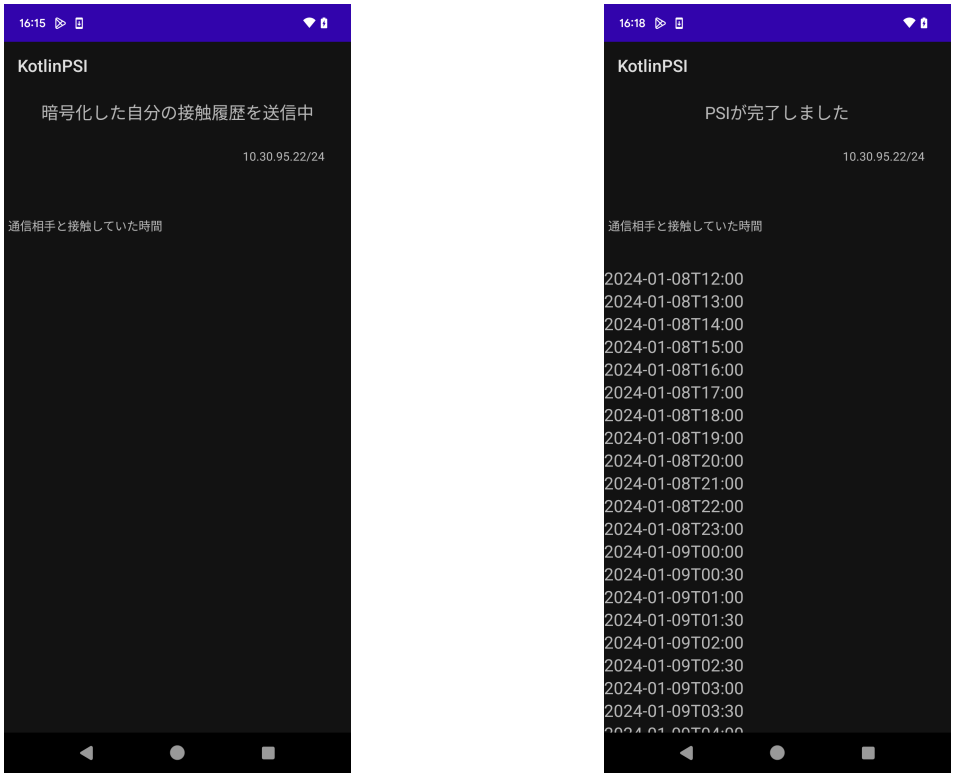


(a) アプリ起動時の画面



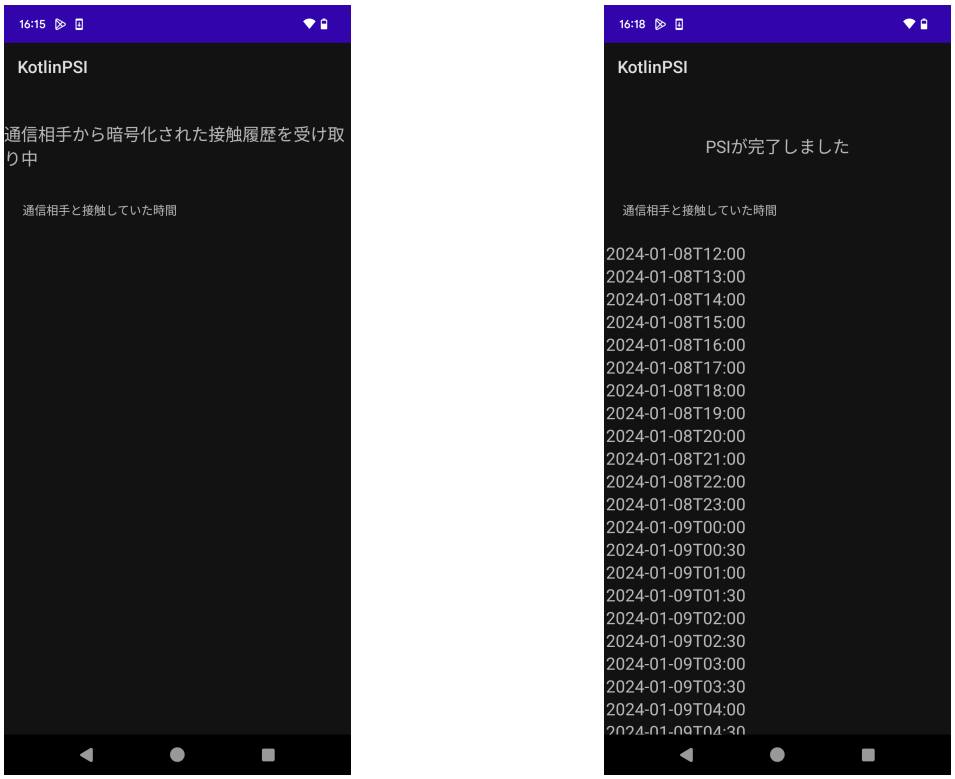
(b) 秘匿共通集合計算実行前の画面

図 5.4: 秘匿共通集合計算前のアプリの画面



(a) 秘匿共通集合計算実行中の画面 (サーバ)　(b) 秘匿共通集合計算実行終了時の画面 (サーバ)

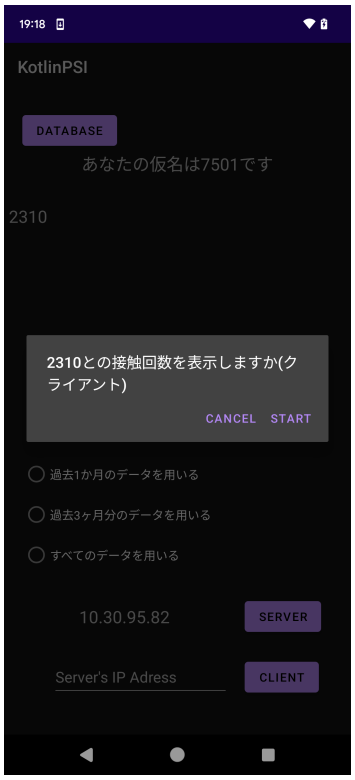
図 5.5: サーバの秘匿共通集合計算実行時のアプリの画面



(a) 秘匿共通集合計算実行中の画面 (クライアント)　(b) 秘匿共通集合計算実行終了時の画面 (クライアント)

図 5.6: クライアントの秘匿共通集合計算実行時のアプリの画面

(a) 秘匿共通集合計算実行前の確認画面 (サーバ)



(b) 秘匿共通集合計算実行前の確認画面 (クライアント)

図 5.7: サーバ，クライアントの PSI 確認画面

第 6 章

評価

この章では実装した提案アプリの評価をする．アプリをインストールした Google Pixcel 7 を 2 台用意し，異なるデータ数，共通集合の組み合わせとなるデータを用意した．

まず，共通集合が 6 割のデータにおけるサーバの PSI 実行時間のグラフを図 6.1 に示す．ここで，PSI 実行時間をサーバのものとした理由は，PSI の処理開始がサーバの暗号化の処理であるためである．図 6.1 に示すように，データ数が増えると実行時間は指数関数的に増大していく．この待ち時間が長いと感じるかどうかは人によって異なるが，速い時間で完了するほうが良い．そのため各処理を最適化し，無駄な動作をなくすことで秘匿共通集合計算にかかる時間を小さくしていく必要がある．例えば，現状では図 5.1 に示すようにサーバ，クライアントごとに一つの処理を終えてから次の処理を行っている．しかし，クライアントがサーバの暗号化した接触履歴の受け取りとクライアントの接触履歴の暗号化は並列処理が可能である．このように並列処理をすることができる部分を改善していくことで秘匿共通集合計算にかかる時間を小さくすることができる．また，Bloom Filter などを用いて効率的に共通要素を見つける手法 [22][23] を検討する必要もある．さらに，秘匿共通集合にかかる時間を予測し使用者に伝えることで長い時間がかかるようなデータ量のときでも時間を許容してくれる使用者が増えると考えている．

図 6.2 にデータ数が 100 のときの共通集合の割合に対する秘匿共通集合計算にかかる時間の関係を示す．共通集合の割合 2 割で 9.449，6 割で 9.286，10 割で 8.78 秒という結果になった．ここから分かる通り，共通集合の割合で秘匿共通集合計算にかかる時間は誤差の範囲内であるといえる．

図 6.1，図 6.2 にサーバ，クライアントの PSI にかかる各処理の時間を示す．クライアントの処理である復号と共通集合計算にかかる時間が突出して大きくなっている．サーバの集合を X，クライアントの集合を Y としたとき，復号と共通集合計算以外は $O(|X|)$

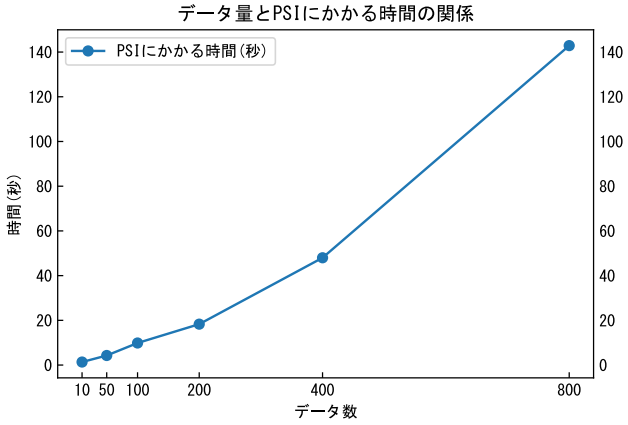


図 6.1: 接触履歴のデータ数とサーバの PSI 完了までの時間の関係 (共通集合 6 割)

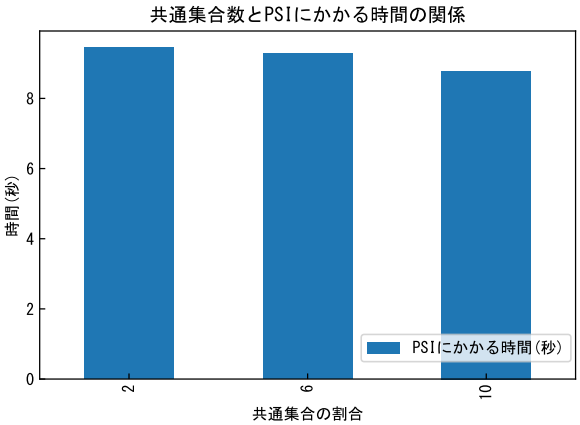


図 6.2: 共通集合の割合を変えた時のサーバの PSI 完了までの時間の変化

または $O(|Y|)$ 時間であるが，復号と共通集合計算の処理のみ $O(|X||Y|)$ となるため時間がかかっていると考えられる．また，クライアントが復号と共通集合計算を行っている間，サーバは何も行わない待ち状態になる．この時間にサーバも共通要素を計算することができると，4.2.3 で述べたクライアントが偽の情報を送った際にサーバが検知できるようになる．

図 6.3 にサーバとクライアントが持つデータ数が違う時の秘匿共通集合計算にかかる時間を示す．クライアントのデータ量が多いときのほうが少し時間がかかる傾向がある．

データ数	接触履歴の暗号化	データ送信	データ受信	クライアントの履歴の暗号化
10	0.0019	0.455	0.243	0.0023
50	0.0068	1.317	0.845	0.0297
100	0.0178	2.533	2.363	0.0482
200	0.0416	3.677	3.471	0.0696
400	0.0537	7.290	7.128	0.1046
800	0.1141	14.362	12.941	0.1527

表 6.1: サーバの PSI にかかる各処理の時間 (秒)

データ数	接触履歴の暗号化	データ送信	データ受信	復号，共通集合計算
10	0.0042	0.390	0.323	0.0695
50	0.0144	0.939	1.192	0.4773
100	0.0293	2.495	2.365	1.5430
200	0.0532	3.676	3.575	5.9760
400	0.0745	7.024	7.198	23.6640
800	0.1240	12.808	14.239	96.6430

表 6.2: クライアントの PSI にかかる各処理の時間 (秒)

サーバ ＼ クライアント	10	50	100	200	400	800
10	1.509	3.067	5.567	8.875	16.107	29.580
50	2.560	4.962	6.684	10.354	19.176	48.500
100	3.700	5.594	9.277	14.358	25.020	57.588
200	7.334	10.761	12.887	21.992	33.474	62.160
400	13.044	17.325	20.588	31.702	49.356	87.714
800	21.733	28.625	36.297	51.638	81.152	144.736

表 6.3: 異なるデータ数に対する PSI にかかる時間 (秒)

第 7 章

おわりに

本論文では，Bluetooth Low Energy を用いて接触履歴を取得し，プライバシに配慮して接触履歴の共通集合計算をするアプリケーションを提案した．そして Android 端末のみで秘匿共通集合計算をするアプリケーションを実装し，実行時間を評価した．BLE の広告パケットに含めるデータとして時間変化する仮名を用いることで，デバイス名などから端末の追跡ができないようになる．そして楕円曲線暗号を用いて暗号化した接触履歴を使って秘匿共通集合計算をすることで，他者に接触履歴を開示することなく接触人数を知ることができるようになる．また今回は，Android Studio を用いて秘匿共通集合計算をする Android アプリケーションを実装した．Android Studio の機能である native C++ を用いて BoringSSL のライブラリを使用した．そして実装したアプリケーションに対してデータ量を変化させ実行時間を計測し評価した．

今後の課題は提案アプリの完成である．BLE を用いて接触履歴を取得する機能や，仮名を生成する機能を実装する必要がある．さらに，本文中でも触れたが，別スレッドで行っている暗号化された接触履歴の送受信時に暗号化処理をするなど，並列化処理をうまく使い秘匿共通集合計算にかかる時間を短縮することが課題である．また，追加の機能として場所の情報を追加することも挙げられる．本研究では新たな環境に身を置く人を対象としたアプリケーションを提案した．環境が変わるとき，その人が住んでいた場所が元居た場所から離れる可能性は高い．そのため新たな場所でしか会っていない人と元居た場所でもよく会っていた人が混ざってしまうことがある．そこで，新たな場所で行動しているときのデータと元居た場所で活動しているときのデータを分けることができれば利便性が向上すると考えられる．

謝辞

本研究は広島大学大学院先進理工系科学研究科先進理工系科学専攻情報科学プログラム計算機基礎学研究室において行ったものです．本論文を作成するにあたり，北須賀輝明准教授から丁寧なご指導を賜りました．ここに感謝の意を表します．

また，論文について有益なご助言をいいただきました中西透教授，森本康彦教授，今井勝喜准教授に感謝申し上げます．

最後に計算機基礎学研究室の皆様には本研究においてご助言，ご協力いただきました．お礼申し上げます．

外部発表一覧

1. M. Murakoshi, T. Kitasuka and T. Nakanishi, ”A File Sharing Method Using a Delay Tolerant Network in Daily Life,”Proc. of TENCON 2022 - 2022 IEEE Region 10 Conference (TENCON), 6 pages, November 2022, doi: 10.1109/TENCON55691.2022.9977889.
2. 村越 允，北須賀 輝明，中西 透，“BLE 接触履歴とプライバシ保護を備えた新たな友人の発見方法”，情報処理学会研究報告，ユビキタスコンピューティングシステム (UBI)，Vol.2023-UBI-80 No.36，2023 年 11 月.
3. Takuto Hashibe，Makoto Murakoshi，Teruaki Kitasuka and Toru Nakanishi，“A Serverless Signaling Scheme for WebRTC Using Bluetooth LE”，Proc. of CANDAR 4pages，November 2023.

参考文献

[1] Bernard A. Huberman, Mat t Franklin and Tad Hogg, “Enhancing Privacy and Trust in Electronic Communities”, Feldman S.I., Wellman M.P. (eds.)ACM conference on Electronic commerce, EC–pp pp.78–86, November 1999.

[2] Lea Kisser, Dawn Song, “Privacy-Preserving Set Operations”, CRYPTO 2005: Advences in Cryptology, pp.241-257.

[3] 宮地 充子, 高野 祐樹, 中正 和久, “プライバシを保護した分散医療データ統合セキュリティシステム”, 第 40 回医療情報学連合大会, 2020 年 11 月.

[4] 千田 浩司, 五十嵐 大, 高橋 克巳, “照合タグを用いた秘匿共通集合計算プロトコルとその応用”, コンピュータセキュリティシンポジウム 2009 論文集 pp.1–6, 2011 年 10 月.

[5] 西田 昌平, 宮地 充子, “効率的な多機関の Private Set Intersection”, Computer Security Symposium 2014 October 2014.

[6] 秋山 周平, 森本 涼也, 谷口 善明, “MAC アドレスがランダム化された BLE 機器の同定手法”, 2021 年度情報処理学会関西支部 支部大会, 2021 年 9 月.

[7] 秋山 周平, 谷口 義明, “MAC アドレスがランダム化された BLE パケットからの同一機器推定手法の改良と評価”, 2022 年度情報処理学会関西支部 支部大会 2022 年 9 月.

[8] 厚生労働省新型コロナウイルス感染症対策推進本部デジタル庁国民向けサービスグループ, “新型コロナウイルス接触確認アプリ COCOA”, 2022 年 7 月. https://www.mhlw.go.jp/content/10900000/000959660.pdf (2023 年 9 月 参照)

[9] Google, “Exposure Notifications API”, 2022 年 3 月 . https://developers.google.com/android/exposure-notifications/exposure-notifications-api?hl=en (2023 年 9 月 参照)

[10] Apple, Google, “Exposure Notification cryptography Specification”, April

2020. https://blog.google/documents/69/Exposure_Notification_-_Cryptography_Specification_v1.2.1.pdf/ (2023 年 9 月 参照)

[11] Bluetooth SIG, “Bluetooth テクノロジーウェブサイト”, 2023 年. https://www.bluetooth.com/ja-jp/ (2023 年 9 月 参照)

[12] 長尾 佳高, 宮地充子, “プライバシを保護したデータ突合プロトコル”, Computer Security Symposium 2020 26 - 29 October 2020.

[13] 土井 アナスタシヤ, 中井 雄士, 品川 和雅, 渡邉 洋平, 岩本 貢, “カードを用いた秘匿共通集合プロトコル”, Computer Security Symposium 2021 26 - 29 October 2021.

[14] 磯崎 邦隆, 菊池 浩明, “アドレスリストを秘匿し交わりの大きさを求める方式とその定点観測への応用”, マルチメディア通信と分散処理ワークショップ 2008 年 12 月.

[15] 野島 良, “高速秘匿共通集合計算プロトコルの設計方法について”, 情報通信研究機構季報 Vol.54 Nos.2/3 2008.

[16] Maurice Lam, “Boringssl”, 2023 年 9 月. https://boringssl.googlesource.com/boringssl (2023 年 9 月 参照)

[17] OpenSSL Management Committee, “OpenSSL Cryptography and SSL/TLS Toolkit”, 2023 年 9 月. https://www.openssl.org/ (2023 年 9 月 参照)

[18] H.Krawczyk, “HMAC-based Extract-and-Expand Key Derivation Function (HKDF)”, Internet Engineering Task Force, RFC5869, May 2010.

[19] Don Johnson, Alfred Menezes, Scott Vanstone, “The Elliptic Curve Digital Signature Algorithm (ECDSA)”, International Journal of Information Security, Volume 1, Number 1, pp.36–63, August 2001.

[20] 光成滋生, “暗号と認証”, 株式会社技術評論社 137 ページ 2022 年 2 月.

[21] Android Developers, “Room”, 2023 年 2 月. https://developer.android.com/jetpack/androidx/releases/room?hl=ja (2023 年 1 月参照)

[22] 白木 徹, 寺西 裕一, 竹内 亨, 春本 要, 西尾 章治郎, “P2P ネットワークにおける Bloom Filter を用いた移動履歴に基づくユーザ探索手法の提案”, 情報処理学会論文誌 Vol.51 No.9 1905-1915 Sep.2010.

[23] 菊池 浩明, 佐久間 淳, “Bloom フィルタを用いたマッチング数の秘匿比較”, Computer Security Symposium 2011 19 - 21 October 2011.