LINE Letter Sealingに対する Tamarin-Proverを用いた網羅的安全性検証

松本 岳大[1,a)]　田中 篤[1]　山下 恭佑[1,2]　伊藤 竜馬[1,3]　五十部 孝典[1]

概要：LINE は, 日本国内で月間アクティブユーザー数 9,800 万人 (2025 年 3 月時点) を誇る, 最も広く利用されているメッセージングアプリケーションである. その高い普及率から, 通信プロトコルの安全性は社会的に極めて重要である. しかし, LINE が採用するエンドツーエンド暗号化プロトコル Letter Sealing に対して, これまで網羅的な形式的安全性検証は行われていない. 本論文では，暗号プロトコルの形式的検証ツールである Tamarin Prover を用い，1 対 1 通信プロトコルの Version 1 および Version 2 を対象として，メッセージングアプリケーションに求められるセキュリティ性質を網羅的に評価する．Version 1 については，既知の脆弱性を再現するとともに，これまで未検証であった複数のセキュリティ性質に関する初の包括的検証を行う．Version 2 については，暗号プロトコルの修正により一部の脆弱性が解消されていることを確認した一方，依然として複数のセキュリティ性質が満たされていないことを示す．本研究は，現在運用されている Version 2 に関する初の包括的形式検証であり，その安全性の達成状況と残存する課題を明確に示すものである.

キーワード：エンドツーエンド暗号化，LINE，暗号プロトコル, 安全性解析, 形式検証

Comprehensive Security Analysis of LINE Letter Sealing Using Tamarin-Prover

Takehiro Matsumoto[1,a)]　Atsushi Tanaka[1]　Kyosuke Yamashita[1,2]　Ryoma Ito[1,3]
Takanori Isobe[1]

Abstract: LINE is the most widely used messaging application in Japan, boasting 98 million monthly active users as of March 2025. Given its widespread adoption, the security of its communication protocol holds significant societal importance. However, despite this relevance, the end-to-end encryption protocol employed by LINE—Letter Sealing—has not yet undergone comprehensive formal verification. In this study, we formally model both Version 1 and Version 2 of LINE's one-to-one communication protocol using the symbolic verification tool Tamarin-Prover. We conduct a thorough and systematic verification of critical security properties, including confidentiality, authentication, and integrity. Notably, this study constitutes the first comprehensive formal analysis of Version 2, which has previously lacked detailed examination. For Version 1, we not only reproduce known vulnerabilities such as replay attacks and spoofing attacks, but also perform exhaustive verification of various packet-level security properties. In the case of Version 2, we confirm that certain vulnerabilities have been mitigated due to changes in cryptographic algorithms. Nonetheless, our analysis uncovers several previously unidentified weaknesses that remain present in the protocol. This work provides a foundational step toward rigorous evaluation of the security guarantees provided by LINE's encryption mechanisms and underscores the ongoing need for formal verification in widely deployed communication platforms.

Keywords: End-to-End Encryption, LINE, Cryptographic Protocol, Security Analysis, Formal Verification

1. はじめに

近年，メッセージングアプリケーションは私生活からビジネスまで幅広く利用されており，通信の秘匿性と安全性は社会基盤に直結する重要な課題となっている．そのため主要なメッセージングサービスは，エンドツーエンド暗号化（End-to-End Encryption, E2EE）を採用し，強固なプライバシー保護を実現している．エンドツーエンド暗号化とは，通信する 2 者間でのみメッセージの送受信が可能であり，その他の，例えば通信システムの提供者であってもメッセージの盗聴，改ざんができない暗号化通信の方式を指す．E2EE は機密性，完全性，認証性に加え，前方秘匿性や否認可能性といった高度なセキュリティ特性を実現することが求められるが，複雑な暗号プロトコル設計に伴い脆弱性が残存する事例も多く報告されている [1][2][3].

LINE は，日本国内で月間アクティブユーザー数 9,800 万人（2025 年 3 月時点）を誇る，最も広く利用されているメッセージングアプリケーションである．その安全な通信を支える E2EE プロトコルとして Letter Sealing が導入されている．これらの仕様は．2016 年発行のホワイトペーパー [4] に記載されている．

2018 年に，Letter Sealing の Version1 に対する安全性評価が五十部と峯松により行われ，なりすまし攻撃や改ざんが可能のことが示された [5]．この結果を受け LINE 社は 2019 年に修正を行い，Letter Sealing を Version2 にアップデートし，これらの仕様をまとめたホワイトペーパーを 2019 年に発行した．Version2 に対しては喜多と五十部が安全性解析を行い，Version1 に対する攻撃の一部が対策により実行不可能であることを示した [6].

これらはプロトコルに対する手動解析であり，脆弱性の発見には有効であるものの，プロトコル全体の安全性を保証するものではない．暗号プロトコルの安全性を検証する手法としては，形式検証が広く用いられている．形式検証では，安全性要件を数学的にモデル化し，プロトコルが所望の安全性を満たすかどうかを自動的に検証することが可能である．LINE の Letter Sealing プロトコルに対する形式検証としては，Proverif による評価が師と米山により行われている [7]．この研究では，Version1 において，機密性，リプレイ攻撃耐性，スプーフィング攻撃耐性といった限定的な安全性特性が検証された．しかし，この評価は Version1 に対する部分的な検証にとどまり，E2EE に要求される安全性全般を網羅するものではない．さらに，現在運用されている Version2 に関しては，網羅的な形式的安全性検証は未だ行われていない．

表 1: Letter Sealing Version1 と Version2 のセキュリティ性質の比較

	既存研究 [7]	本研究	
性質 [8]	**v1**	**v1**	**v2**
秘匿性	✓	✓	✓
認証性	×	×	✓
完全性	×	×	✓
前方秘匿性	×	×	×
後方秘匿性	-	×	×
リプレイ攻撃耐性	×	×	✓
メタデータの改ざん検知	-	×	✓
宛先検証	-	×	✓
全能攻撃者への耐性	×	×	×
暗号文に対する改ざん検出	-	✓	✓
送受信者の匿名性	-	×	×

✓ は安全であることを表し，× は脆弱，-は未検証であることを表す．v1 と v2 は Version1 と Version2 を示す．

本研究では, 暗号プロトコルの形式的検証ツール Tamarin Prover を用いて，Letter Sealing の 1 対 1 通信プロトコル Version1 および Version2 を対象に，メッセージングアプリケーションに要求される多様なセキュリティ性質（秘匿性，認証性，前方秘匿性，リプレイ攻撃耐性，否認可能性など）[8] を包括的に検証した．Tamarin Prover を採用した理由は，検証したいセキュリティ性質を lemma として詳細に記述できる柔軟性と，強力な攻撃者モデルを備えている点にある．このため，秘匿性や認証性といった標準的な性質に加え，前方秘匿性や否認可能性など，メッセージングに特有の性質を厳密に検証することができる．プロトコルの動作は rule として逐次的にモデリングし，送信者と受信者の間での鍵共有やメッセージ送信の過程を形式的に記述した．一方で，検証対象となるセキュリティ性質は lemma として定義し，例えば「送信者がメッセージを送信した場合，正当な受信者のみがその内容を復号できる」ことや「攻撃者が過去の鍵情報を取得しても，将来の通信内容を復号できない」といった形式で表現した．

Version1 と 2 に対するの評価結果を表 1 に示す．本論文の貢献は以下の通りである．

- Version1 に関しては，五十部と峯松によって指摘された既知の脆弱性 [5] を再現するとともに，Proverif を用いた師と米山の手法 [7] では未検証であった複数のセキュリティ性質について初の体系的検証を実現した．
- 現在実際に運用されている Version 2 を初めて網羅的に形式検証した．その結果，暗号アルゴリズムの改良により一部の脆弱性は解消されていることを確認した一方で，なお複数のセキュリティ性質が十分には満た

[1] 大阪大学
The University of Osaka, Japan
[2] AIST
AIST, Japan
[3] NICT
NICT, Japan
[a)] take-matsumoto@ist.osaka-u.ac.jp

されていないことが明らかとした．

本研究は，現在運用されている Version2 に関する初の包括的形式検証であり，その安全性の達成状況と残存する課題を明確に示すものである．

2. 前提知識

2.1 エンドツーエンド暗号化

エンドツーエンド暗号化（End-to-End Encryption: E2EE）とは，通信の両端に位置する正当なユーザのみがメッセージの内容を復号可能となるよう設計された暗号化方式である．通信経路上に存在するインターネットプロバイダや通信事業者，さらにはその通信サービスの提供者でさえも，暗号鍵にアクセスすることはできない．このような設計により，E2EE は第三者による傍受や改ざんに対して高い耐性を持つ．

2.2 攻撃者モデルと安全性要件

本研究における攻撃者は，アクティブな E2E adversary として定義する．この攻撃者は，通信経路上において任意のメッセージを傍受・改ざん・生成する能力を有し，特にサーバーに送信されるすべての情報にアクセス可能であると仮定する．このモデルは，E2EE 環境における現実的かつ強力な脅威を想定している．

本研究では，以下に示すセキュリティ性質について検討を行う．これらの性質は，安全なメッセージングプロトコルに関する体系的整理を行った研究 [8] において定義されているものであり，プロトコルの設計および評価において広く参照されている．ここで留意すべきは，通信プロトコルが必ずしも全てのセキュリティ性質を同時に満たすわけではないという点である．各プロトコルは，利用環境や脅威モデルに基づき，設計者が優先する性質を重視し，他の性質をある程度犠牲にして設計されることが多い．

定義 2.1 (秘匿性)**.**
通信相手以外の第三者がメッセージ内容を復元できない性質を秘匿性と呼ぶ．この性質が保証されることで，通信内容の機密性が保たれる．

定義 2.2 (認証性)**.**
通信相手が主張する身元を正しく確認できる性質を認証性と呼ぶ．この性質により，通信に関与する当事者は互いが意図した相手であることを確認できる．

定義 2.3 (完全性)**.**
通信中のメッセージや宛先が送信者によって生成されたものであり，改ざんされていないことを受信者が確認できる性質を完全性と呼ぶ．この性質により，攻撃者によるメッセージの改ざんや挿入が検出可能となる．

定義 2.4 (前方秘匿性)**.**
将来，長期鍵やセッション鍵が漏洩したとしても，過去の通信内容の秘匿性が保たれる性質を前方秘匿性と呼ぶ．この性質が成立していれば，過去に暗号化されたメッセージは将来的にも安全である．

定義 2.5 (後方秘匿性)**.**
過去の通信内容が第三者に漏洩した場合でも，将来の通信内容の秘匿性が保持される性質を後方秘匿性と呼ぶ．

定義 2.6 (リプレイ攻撃耐性)**.**
過去に送信されたメッセージを第三者が再送信しても，受信者が正当な通信として処理しない性質をリプレイ攻撃耐性と呼ぶ．

定義 2.7 (メタデータの改ざん検知)**.**
送信者や受信者の識別子，鍵 ID，通信の種別やバージョン情報などのメタデータが第三者によって改ざんされた場合に，その不正を検知できる性質をメタデータの改ざん検知と呼ぶ．この性質により，通信内容そのものだけでなく，関連するメタ情報の完全性も保証される．

定義 2.8 (宛先検証)**.**
受信者が，自身に送られたメッセージであることを確認できる性質を宛先検証と呼ぶ．この性質により，第三者によるメッセージの誤配信やすり替えが防止される．

定義 2.9 (全能攻撃者耐性)**.**
ネットワーク上の全ての通信を監視・改ざん・削除・挿入できる能力を持つ全能攻撃者（Global Adversary）の存在を仮定した場合においても，機密性・完全性・認証性など，プロトコルが設計上重視するセキュリティ性質を維持できる能力を全能攻撃者耐性と呼ぶ．この性質は，安全なメッセージングを提供する上で最も強力な脅威モデルに対する防御能力を意味する．

定義 2.10 (送受信者の匿名性)**.**
第三者が通信の送信者または受信者を特定できない性質を送受信者の匿名性と呼ぶ．この性質により，誰が通信を行ったかが秘匿される．

2.3 Tamarin Prover の基本概念

Tamarin Prover [9] は，セキュリティプロトコルをモデリングし，自動的に安全性検証を行うためのツールである．プロトコルはマルチセット書き換え規則（multiset rewriting rules）として記述され，攻撃者の行動やネットワーク上のメッセージも含めた状態遷移を形式的に表現できる．これにより，無数の並列実行や攻撃者の介入を考慮したうえで，指定されたセキュリティ性質が保持されるかどうかを検証可能である．

Tamarin においては，プロトコルの動作やメッセージの送受信は **rule** で記述される．一方，暗号プロトコルが満たすべきセキュリティ性質は **lemma** として定義され，トレース性質や等価性の形で検証される．自動探索の結果，性質が保持される場合はその証明が得られ，成立しない場合には攻撃シナリオが反例として提示される．

さらに，**restriction** はプロトコル実行における制約条件を表現するものであり，攻撃者の行動やプロトコルの動作に関する不自然なシナリオを除外する役割を持つ．これにより，より現実的な通信環境を前提とした検証が可能となる．

このように，**rule** はプロトコルの動作を，**lemma** は検証対象となる安全性の要件を，**restriction** は検証の前提となる制約条件を記述するものであり，本研究におけるモデル化と検証の基盤を成す．

3. LINE の E2EE プロトコルの説明

本章では, LINE における 1 対 1 通信のプロトコルフローについて説明する．本論文では，Letter Sealing Version1 および Version2 について，以降はそれぞれ v1，v2 と記す．

3.1 Letter Sealing v1 の 1 対 1 通信暗号化プロトコル

鍵生成と共有

各クライアントは Curve25519 に基づく鍵ペア (sk, pk) を生成し，公開鍵をサーバに登録する．メッセージ送信時には，相手の公開鍵と自分の秘密鍵を用いて共通鍵 *SharedSecret* を計算し，暗号処理の基盤とする：

$$\begin{aligned} SharedSecret &= ECDH_{\text{Curve25519}}(sk_A, pk_B) \\ &= ECDH_{\text{Curve25519}}(sk_B, pk_A). \end{aligned}$$

暗号化処理

送信者は，共有鍵 *SharedSecret* と乱数 *salt* を入力として鍵 K_{enc} と初期化ベクトル IV_{enc} を導出する．これらを AES-256-CBC に入力し，メッセージ M を暗号化する：

$$C = AES\text{-}CBC(K_{enc}, IV_{enc}, M).$$

生成された暗号文 C を SHA-256 に入力し，その出力値を AES-ECB に入力して MAC 値を成する：

$$\begin{aligned} MAC_{plain} &= SHA256(C), \\ MAC_{enc} &= AES\text{-}ECB(K_{enc},\ MAC_{plain}[0{:}15] \\ &\quad \oplus MAC_{plain}[16{:}31]). \end{aligned}$$

送信データ構造と復号

送信されるデータは，暗号文と付随するメタ情報をまとめた以下の構造を持つ：

$$\{version,\ content\_type,\ salt,\ C,\ MAC_{enc},\ sender\_key\_ID,\ recipient\_key\_ID\}$$

受信者は同様の鍵導出を行い，C を復号した後，送信された MAC_{enc} と自ら計算した MAC を比較する．両者が一致した場合にのみ，メッセージは正当と認められユーザに表示される．

3.2 Letter Sealing v2 の 1 対 1 通信暗号化プロトコル

Letter Sealing v2 は，v1 で指摘された完全性検証や改ざん検出の脆弱性を克服するために導入された改良版プロトコルである．最大の変更点は，暗号化方式を AES-CBC ＋ MAC から認証付き暗号 AES-GCM へ移行した点である．これによりメッセージの暗号化と完全性検証を統合的に実現した．さらに，パケット構造におけるヘッダ情報の明示化，および AAD（Additional Authenticated Data）の導入により，メッセージの一意性と送受信者の正当性検証が強化されている．

鍵共有には依然として Curve25519 に基づく ECDH が用いられるが，その後の鍵導出処理では 16 バイトの乱数 *salt* を用い，メッセージごとに異なる暗号鍵が生成される．AES-GCM で必要となるナンス（nonce）は，各メッセージ送信時に生成され，暗号化および認証処理において一意性を保証するために利用される：

$$(C,\ tag) = AES\text{-}GCM(K_{enc},\ nonce,\ M,\ AAD).$$

さらに，AAD には送受信者 ID や鍵 ID，バージョン情報などが含まれ，これらが暗号文とともに認証対象となる．この仕組みにより，メタデータ改ざんに対する耐性が確保されている．

送信時には，平文メッセージ M は次のように暗号化され，暗号文 C と認証タグ *tag* が生成される：

$$(C,\ tag) = AES\text{-}GCM(K_{enc},\ nonce,\ M,\ AAD).$$

受信者は同様の鍵導出とナンス生成を行い，タグを検証することで改ざんの有無を確認する．タグが一致しない場合，メッセージは破棄される．この設計により，v1 に比べて完全性・認証性が大幅に改善されている．

4. Tamarin を用いた自動検証

本章では，Tamarin を用いた自動検証について述べる．はじめに，v1 と v2 共通である鍵共有のモデリングについて述べる．次に，v1 と v2 それぞれの 1 対 1 通信におけるメッセージ暗号化プロトコルのモデリングおよびそれらの検証について述べる．紙面の制約上，全ての性質の検証について言及することができないため，v1 から v2 への変更に伴う重要な性質に焦点を当て，その他の検証については割愛する．なお，本研究で使用したソースコードはすべて GitHub[10] にアップロードしている．ソースコードの詳細やその他の性質の検証については [10] を参照されたい．

4.1 鍵共有のモデリング

本節では，1 対 1 通信のための鍵共有を Tamarin の **rule** として記述してモデリングする．本論文のソースコードでは，A を送信者，B を受信者と定義する．

長期秘密鍵の生成・登録

この **rule** は，A と B それぞれにランダムな ID (aID, bID) と長期秘密鍵 (skA, skB) を生成している．

```
rule Init:
  [ Fr(~aID), Fr(~skA), Fr(~bID), Fr(~skB) ]
  --[ InitPhase(~aID, ~skA, ~bID, ~skB) ]->
  [ !InitA(~aID, ~skA), !InitB(~bID, ~skB) ]
```

公開鍵生成と登録

これらの **rule** は，Diffie-Hellman のベースである g^{sk} の演算により，A と B の公開鍵 (pkA, pkB) を生成する．

```
rule KeyGeneration_A:
  let pkA = 'g' ^ ~skA in
  [ !InitA(~aID, ~skA) ]
  --[ GenKeyA(~aID, ~skA) ]->
  [ !LongTermKey(~aID, ~skA), !PublicKey(~aID, pkA
      ) ]

rule KeyGeneration_B:
  let pkB = 'g' ^ ~skB in
  [ !InitB(~bID, ~skB) ]
  --[ GenKeyB(~bID, ~skB) ]->
  [ !LongTermKey(~bID, ~skB), !PublicKey(~bID, pkB
      ) ]
```

サーバーへ公開鍵を登録

この 2 つの **rule** では，生成された公開鍵をそれぞれ Key Server に登録する．これにより，A と B がお互いの公開鍵を取得できる状態となる．

```
rule RegisterPK_A:
  [ !PublicKey(~aID, pkA) ]
  --[ PKRegisteredA(~aID, pkA) ]->
  [ !KeyServer(~aID, pkA) ]

rule RegisterPK_B:
  [ !PublicKey(~bID, pkB) ]
  --[ PKRegisteredB(~bID, pkB) ]->
  [ !KeyServer(~bID, pkB) ]
```

共通鍵の生成

これらの **rule** により，A と B はそれぞれお互いに相手の公開鍵と自身の秘密鍵から共通鍵 (SharedSecret) を生成する．この共通鍵は後続の暗号化と復号にて用いる．ソースコード上において，SharedSecret は ss と表記している．

```
rule GenerateSharedSecretA:
  let ss = pkB ^ ~skA in
  [ !LongTermKey(~aID, ~skA), !PublicKey(~bID, pkB
      ), In(pkB) ]
  --[ A_SharedSecretGen(~aID, ~bID, pkB),
      SessionInit(),SessionSecret(~aID, ~bID, ss)
      ]->
  [ !SharedSecret_A(ss, ~aID, ~bID) ]

rule GenerateSharedSecretB:
  let ss = pkA ^ ~skB in
  [ !LongTermKey(~bID, ~skB), !PublicKey(~aID, pkA
      ), In(pkA) ]
  --[ B_SharedSecretGen(~bID, ~aID, pkA),
      SessionInit(), SessionSecret(~bID, ~aID, ss)
      ]->
  [ !SharedSecret_B(ss, ~bID, ~aID) ]
```

4.2 v1 および v2 のモデリング

本節では，鍵共有完了後の v1 および v2 におけるそれぞれのメッセージ暗号化処理，受信側での復号処理，および攻撃者が操作可能な脅威シナリオについて，**rule** 記述と共に解説する．

4.2.1 v1 のモデリング

メッセージ暗号化と MAC

A は，メッセージの暗号化準備として乱数（salt）とメッセージ本体を生成し，鍵導出関数（KDF）によりセッション鍵を生成する（TriggerEncrypt）．続く EncryptAndTag では，導出された鍵と salt を用いてメッセージを暗号化した後，MAC を計算して送信データに添付する．このとき，ヘッダ情報（aID と bID）を含む構造で出力することで，後続の認証性・整合性検証が可能となる．

```
rule TriggerEncrypt:
  [ !SharedSecret_A(ss, ~aID, ~bID),Fr(~m), Fr(~
      salt)]
  --[ TriggerEncryptMessage(ss, ~aID, ~bID) ]->
  [ !TriggerEncryptData(~m, ~salt, ss, ~aID, ~bID)
      ]

rule EncryptAndTag:
  [ !TriggerEncryptData(~m, ~salt, ss, ~aID, ~bID)
      ]
  --[ Sent(~aID, ~bID, ~m) ]->
  [ Out(< <~aID, ~bID>,
        < pair(enc(~m, kdf(ss, ~salt)), ~salt),
          mac(enc(~m, kdf(ss, ~salt)), kdf(ss, ~
              salt))>>)
  ]
```

受信メッセージの検証と復号

受信側では，$ReceiveEncAndSalt_B$ にて送られてきたヘッダ付きメッセージを一時的に保存し，次に $BVerifyTagAndDecrypt$ にて MAC 検証と復号を行う．まずペイロードから暗号文と salt を分離し，鍵導出関数によりセッション鍵を再生成する．続いて，メッセージ本体の改ざん有無を MAC で検証し，正当であればメッセージを復号・出力する．

```
rule ReceiveEncAndSalt_B:
  [ !SharedSecret_B(ss, ~bID, ~aID), In(< <sender,
      ~bID>, <payload, mac_val> >) ]
  --[ ReceivedByB(sender, payload) ]->
  [ !ReceivedDataB(ss, ~bID, sender, payload,
      mac_val) ]

rule BVerifyTagAndDecrypt:
  let
    ct = fst(payload)
```

```
    salt = snd(payload)
    expected_mac = mac(payload, kdf(ss, salt))
    m = dec(ct, kdf(ss, salt))
  in
  [ !ReceivedDataB(ss, ~bID, sender, payload,
      mac_val) ]
  --[ Eq(mac_val, expected_mac), BVerifiedTag(m,
      sender, ~bID) ]->
  [ !BDecryptedMessage(m) ]
```

4.2.2 v2 のモデリング

AEAD を用いたメッセージ暗号化とタグ出力

AEAD のモデリングは Cremers ら [11] によって示された手法をもとに実装している．ここでは，AEAD の暗号化および復号関数をそれぞれ 4 入力関数 senc, sdec として定義する．鍵 k，ナンス n，関連データ ad，平文 m に対して，senc と sdec を用いて等式として定義することにより，暗号化と復号の一意性を保証している．

```
functions: senc/4, sdec/4
equations: sdec(k, n, ad, senc(k, m, n, ad)) = m

rule A_Send_Message_to_B:
    let
    Encryption = senc(ss, ~m, ~nonce, <~aID, ~bID
        >)
    Tag = get_tag(Encryption)
    in
    [ SharedSecret_A(ss, ~aID, ~bID), Fr(~m), Fr(~
        salt), Fr(~nonce) ]
    --[ Sent(~aID, ~bID, ~m), Sent_with_key(~aID,
        ~bID, ~m, ss) ]->
    [ SharedSecret_A(ss, ~aID, ~bID),
      Out(<Encryption, Tag>) ]
```

攻撃者条件の明示

本プロトコルにおける脅威モデルは以下のような攻撃者行動を含む．

- **AttackerClaimsPublicKey**: 攻撃者が他人の公開鍵を自分のものであると偽る．
- **ServerManipulatesHeader**: 攻撃者がメッセージの送信者 ID を改ざんする．
- **Reveal_LTK**: 攻撃者が長期秘密鍵（Long-Term Key）を奪取する．

これらの攻撃行為を明示的に **rule** としてモデル化し，安全性検証における前提条件を明確にしている．

```
rule AttackerClaimsPublicKey:
  [ Fr(~attackerID), In(pk) ]
  -->
  [ !PublicKey(attackerID, pk) ]

rule ServerManipulatesHeader:
  [ Fr(~attackerID),
    In(< <aID, bID>, payload >)
  ]
  -->
  [ Out(< <~attackerID, bID>, payload >) ]

rule Reveal_LTK:
  [ !LongTermKey(id, sk) ]
  --[ Corrupted(id) ]->
  [ Out(sk) ]
```

4.3 制約と安全性検証

本節では，v1 および v2 のモデリングに対して適用した制約（restriction）と，プロトコルの安全性を検証するための **lemma** について記述する．紙面の制約上，特に重要な **lemma** をピックアップして詳細を説明する．

single_session（制約：セッションの一意性）

プロトコルの簡潔な検証を行うために，セッションは 1 回のみ発生するものと制約する．これにより，A と B が同時に複数のメッセージを交わす複雑なケースを除外し，証明の対象を 1 回の通信に限定している．

```
restriction single_session:
  "All #i #j. SessionInit() @ i & SessionInit() @
      j ==> #i = #j"
```

lemma: message_secrecy（秘匿性）

送信者が送ったメッセージ本文 m は，攻撃者に漏洩しないことを保証する．すなわち，正規の送信イベントが存在しても，攻撃者は m を知識ベース（$K(m)$）に追加できない．

```
lemma message_secrecy:
  "All aID bID m #i.
     Sent(aID, bID, m) @ i
     ==> not(Ex #j. K(m) @ j)"
```

lemma: auth_A_believes_B_sent（認証性）

送信者からのメッセージが，受信者において正しく検証された場合，その送信が事前に発生していたことを保証する．この性質により，なりすましや偽造メッセージが認証されないことを確認できる．

```
lemma auth_A_believes_B_sent:
  "All m aID bID #i.
     AVerifiedTag(m, bID, aID) @ i
     ==> Ex #j. (Sent(bID, aID, m) @ j & j < i)"
```

lemma: no_replay_to_A（リプレイ攻撃耐性）

同一メッセージが，正規の送信者を装って複数回受信者に認証されることはない，というリプレイ攻撃耐性を示す．

```
lemma no_replay_to_A:
  "All m sender recipient #i #j.
     AVerifiedTag(m, sender, recipient) @ i
     & AVerifiedTag(m, sender, recipient) @ j
     ==> #i = #j"
```

lemma: metadata_tamper_detection（メタデータの改ざん検知）

この **lemma** は v2 の検証のみで使用する．AEAD の関

連データの改ざん検知が正しくプロトコルモデルに反映されていることを保証する．

```
lemma metadata_tamper_detection:
  "All ~m ~ss ~nonce ~aID ~bID.
   (Ex A_Sent_Message_to_B)@i &
   (BVerifiedTag(~m, ~aID, ~bID))@j & i < j
   ==> (Eq(aad_sender, ~aID))@j & (Eq(aad_receiver
       , ~bID))@j"
```

5. 検証の結果

検証の結果，v1 および v2 におけるセキュリティ性質の充足状況は表 1 の通りである．

5.1 v1 に関する考察

本検証により，LINE における 1 対 1 通信プロトコル v1 は，以下に挙げる複数の基本的・高度なセキュリティ性質を満たしていないことが明らかとなった：メタデータ保護，宛先検証，匿名性（送信者，受信者，通信者全体）．これらの性質が欠如していることで，以下のような具体的なリスクおよび攻撃可能性が生じる．

5.1.1 メタデータ改ざん検知の欠如

v1 のメッセージ構造では，送信者 ID，受信者 ID，バージョン，暗号方式といったメタデータがすべて暗号化されずにヘッダーとして平文で送信される．このことにより，攻撃者はネットワーク上のトラフィックを監視することで，誰が誰にいつメッセージを送っているのかといった通信パターンを容易に把握できる．このようなメタデータの漏洩は，匿名性の喪失や，社会的関係の可視化といった深刻なプライバシー侵害をもたらす．

5.1.2 宛先検証の欠如

v1 におけるメッセージ構造では，受信者 ID が平文で含まれており，かつその正当性を検証する手段が存在しない．この設計上の欠陥により，攻撃者は暗号化されたメッセージの受信者 ID を改ざんし，任意の第三者に転送する攻撃を実行可能である．宛先検証の欠如は情報漏洩と信頼性低下の両面において重大なセキュリティリスクをもたらす可能性がある．

5.1.3 匿名性（送信者・受信者・通信者全体）の欠如

v1 では，メッセージ中に送信者および受信者 ID が明示されているため，外部の攻撃者が通信者の識別や関係性を特定可能である．また，通信に関与している全体の参加者をも識別できるため，参加者匿名性も実現されていない．これは，匿名掲示板やセキュアチャットなどのプライバシーを重視するアプリケーションにとって致命的な欠陥となる．

5.2 v2 に関する考察

v1 に関する考察（5.1 節）では，主に認証性や完全性といった重要なセキュリティ要件が十分に満たされていない点に焦点を当てた．これに対し，本節では v2 において新たに実現されたセキュリティ機能や改善点に着目し，どのように既存の脆弱性が解消されたかを整理する．特に，AAD（追加認証データ）の活用や暗号モードの変更など，設計面の改良によって達成されたセキュリティ強化の具体的な側面を評価する．

5.2.1 認証性

AAD に送信者 ID および受信者 ID を含めることで，受信者はメッセージの送信元が正当であるかを検証可能となった．また改ざん検知もついていることにより，なりすましや偽装送信への耐性が強化された．

5.2.2 完全性

AES-GCM は暗号文とともに認証タグを生成するため，受信者は復号と同時に改ざんの有無を検出することができる．v1 で存在したメッセージの完全性欠如の問題がここで解消された．

5.2.3 リプレイ攻撃耐性

各メッセージに異なる salt や version が付与されるようになったことで，同一メッセージの再送信（リプレイ）を受信側で検出できるようになり，セッションの一意性が保証されるようになった．

5.2.4 メタデータの改ざん検知

通信に含まれるメタ情報（送受信者 ID，鍵 ID など）の改ざんや追跡が困難となり，通信のプライバシーが向上した．

5.2.5 宛先検証

AAD 内に受信者 ID に改ざん検知がついたことで，意図しない第三者への誤送信や改ざんによる転送リスクが抑制されるようになった．

5.3 考察のまとめ

これらの改善により，LINE の 1 対 1 通信におけるセキュリティは総合的に大きく向上したといえる．ユーザーにとっては，メッセージの盗聴や改ざん，不正な送信・受信といったリスクが軽減され，より安心して通信を行える基盤が整備されたといえる．

v2 では認証性や完全性などいくつかの性質が改善された一方で，依然として匿名性，前方秘匿性，後方秘匿性といった重要なセキュリティ要件において脆弱性が残されている．v2 ではプロトコルの安全性に関して大きな前進が見られる一方で，ユーザーのプライバシーや長期的な通信の秘匿性に関わる根本的な設計課題が依然として残存している．今後の改善においては，匿名性の向上と鍵更新機構の導入が不可欠である．

6. 結論

本稿では，LINE におけるエンドツーエンド（E2E）暗

号通信のうち，1 対 1 通信プロトコルで用いられている Letter Sealing v1 および v2 に対し，Tamarin Prover を用いて形式的かつ網羅的なセキュリティ検証を行った．これまでの研究では，セキュリティ性質に関して網羅的に検証されているものはなく、v2 に関しては全く検証されていなかった．

本研究では，論文 [8] で示された複数のセキュリティ性質を 1 つずつ明示的に定義し，検証用コードを構築した．その結果，v1 においては認証性，完全性，前方秘匿性，リプレイ攻撃耐性，メタデータ保護，宛先検証，匿名性など，複数の重要な性質が欠落していることを明らかにした．

一方，v2 ではプロトコル設計が大幅に見直され，認証付き暗号の導入や AAD の活用などにより，認証性，完全性，リプレイ攻撃耐性などが大きく改善された．しかしながら，前方秘匿性や通信者匿名性などは依然として保証されておらず，長期的な鍵漏洩やトラフィック分析といった攻撃に対する脆弱性が残っていることも確認された．

これらの結果は，LINE の E2E 暗号通信がバージョンアップにより一定の安全性向上を達成している一方で，E2E 暗号化としての理想的な要件を完全には満たしておらず，今後さらなる改善が求められることを示している．

謝辞

本研究は，JSPS 科研費 JP24H00696 と JST AIP 加速課題 JPMJCR24U1 の支援を受けたものである．

参考文献

[1] Kimura, H., Ito, R., Minematsu, K., Shiraki, S. and Isobe, T.: Not in The Prophecies: Practical Attacks on Nostr, *10th IEEE European Symposium on Security and Privacy, EuroS&P 2025, Venice,Italy, June 30 - July 4, 2025*, IEEE (2025).

[2] Albrecht, M. R., Dowling, B. and Jones, D.: Formal Analysis of Multi-device Group Messaging in WhatsApp, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques Madrid, Spain, May 4-8, 2025, Proceedings, Part VIII* (Fehr, S. and Fouque, P., eds.), Lecture Notes in Computer Science, Vol. 15608, Springer, pp. 242–271 (online), DOI: 10.1007/978-3-031-91101-9_9 (2025).

[3] Paterson, K. G., Scarlata, M. and Truong, K. T.: Three Lessons From Threema: Analysis of a Secure Messenger, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023* (Calandrino, J. A. and Troncoso, C., eds.), USENIX Association, pp. 1289–1306 (online), available from ⟨https://www.usenix.org/conference/usenixsecurity23/presentation/paterson⟩ (2023).

[4] LINE Corporation: LINE Encryption Overview, Technical report, LINE Corporation (2021). `https://scdn.line-apps.com/stf/linecorp/en/csr/line-encryption-whitepaper-ver2.1.pdf` (Accessed 2025-08-22).

[5] Isobe, T. and Minematsu, K.: Breaking Message Integrity of an End-to-End Encryption Scheme of LINE, *Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II* (López, J., Zhou, J. and Soriano, M., eds.), Lecture Notes in Computer Science, Vol. 11099, Springer, pp. 249–268 (online), DOI: 10.1007/978-3-319-98989-1_13 (2018).

[6] 喜多光太郎・五十部孝典：LINE の End-to-End Encryption Version 2 に対する安全性評価，信学技報, vol. 120, no. 384, ICSS2020-47, pp. 126-131, 2021 年 3 月 (2021).

[7] Shi, C. and Yoneyama, K.: Verification of LINE Encryption Version 1.0 using ProVerif, *IEICE Trans. on Information and Systems*, Vol. 102, No. 8, pp. 1439–1448 (2019).

[8] Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I. and Smith, M.: SoK: Secure Messaging, *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, IEEE Computer Society, pp. 232–249 (online), DOI: 10.1109/SP.2015.22 (2015).

[9] Meier, S., Schmidt, B., Cremers, C. and Basin, D. A.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings* (Sharygina, N. and Veith, H., eds.), Lecture Notes in Computer Science, Vol. 8044, Springer, pp. 696–701 (online), DOI: 10.1007/978-3-642-39799-8_48 (2013).

[10] Take-0216: tamarin_line: A library for working with Tamarin-Prover, `https://github.com/Take-0216/tamarin_line` (2024). Accessed: 2025-08-22.

[11] Cremers, C., Dax, A., Jacomme, C. and Zhao, M.: Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security, *USENIX Security Symposium*, USENIX Association, pp. 5935–5952 (2023).