九州大学学術情報リポジトリ
Kyushu University Institutional Repository

ネットワークの基盤とサービスに関する正当性と安全性の形式検証による評価

櫻田, 英樹

https://hdl.handle.net/2324/7329491

出版情報：Kyushu University, 2024, 博士（情報科学）, 課程博士
バージョン：
権利関係：

KYUSHU UNIVERSITY

ネットワークの基盤とサービスに関する
正当性と安全性の形式検証による評価

櫻田英樹

九州大学 大学院システム情報科学府
情報理工学専攻

2024 年 6 月

論文要旨

半導体技術やネットワーク技術の進歩に従い、計算機およびそれらをネットワークで接続して構成される情報システムは広範な社会活動において必要不可欠となっている。これに伴い、情報システムが要求通りに動作すること、つまりシステムの正当性への必要性もますます高まっている。システムの正当性を厳密に保障する方法として、正当性を数理的に厳密な形で記述し、システムがこれを満たすことを検証する形式検証という方法がある。

本論文では、具体的なシステムとして暗号プロトコルおよびネットワーク機器の設定を取り上げる。正当性として、暗号プロトコルについては安全性を、ネットワーク機器設定については設定後のネットワークにおけるパケットの到達可能性などの性質を扱う。安全性は、攻撃者の下で情報漏洩やなりすましなどの危険な状況が起こらないという性質である。暗号プロトコルについては特定のプロトコルを取り上げ、その安全性を形式検証により評価する。ネットワーク機器の設定については、形式検証により評価する手法を提案する。

本論文ではまず、第 2 章で暗号プロトコルとその安全性について述べる。これは、安全性には攻撃者の概念が不可欠であり、ネットワーク機器設定の正当性に比べ複雑な概念を含むためである。次に、第 3 章で、本研究で用いるものも含め、形式検証手法の概要について述べる。その後、本論文における次の 3 つの具体的貢献について述べる。

第 4 章では、クレジットカードを用いてインターネット上で決済を行うための暗号プロトコルである SET（Secure Electronic Transaction）の支払いを行う部分を取り上げ、その安全性のうち決済時に送受信される顧客のクレジットカード番号の秘匿性を帰納的手法と呼ばれる方法で検証した。クレジットカード番号の秘匿性は、その他の安全性の基礎となる最も重要な性質である。本研究を開始した 2000 年当時、SET の形式検証を行った先行研究ではプロトコルを大きく簡略化しており、簡略化の手順も明示されていなかった。本研究においても仕様書全体で 4 万行を超える SET を現実的な労力と時間内で検証を行うためにプロトコルの簡略化を行ったが、安全性に本質的に寄与する部分を残しそれ以外を簡略化するという手順を明確にしてこれを行った。さらに、プロトコルを記述する言語として、プロセス代数の一種にパターンマッチングの機構を付加したものを考案し、これを用いて簡略化後のプロトコルを簡潔に記述し、さらに検証のための推論規則に変換する手法をとった。

第 5 章では、web のブラウザとサーバの間の通信において認証および暗号化を行うため

の暗号プロトコルである QUIC（Quick UDP Internet Connection）を取り上げ、その安全性を検証した。先行研究において、暗号理論の枠組みを用いて QUIC の安全性を定式化し、いわゆる紙と鉛筆を用いた手作業による証明が行われている。本研究では暗号プロトコルの安全性検証ツールである ProVerif を用いて改めて検証を行った。先行研究における安全性の定式化を、形式検証が可能な形に再度定義し直して検証を行った。本研究による検証により、先行研究により定義された安全性がいずれも成り立たないことを明らかにした。本研究により、暗号学理論的な枠組みにより安全性が証明されていた場合でも、その定式化や証明に不備がある場合があるため、改めて形式検証を行うことの有効性を示すことができた。

第 6 章では、ネットワークの中でもローカルエリアネットワーク（LAN）の機器の設定に着目し、そのうち LAN を仮想的に分割する VLAN 技術の設定の正しさを取り上げ、これを形式検証により検証する方法を提案した。従来および現在においてもネットワークの構成や変更を行った場合は、実際にパケットを送信してテストを行い、誤りがないことを確認することが一般的だが、送受信する端末やパケットの種類を網羅してテストを行うことは現実的ではない。さらに、パケットが本来の相手以外に到達しないことをテストするためには、送信したパケットを、これを受信すべき端末以外の端末で監視し、到達しないことを確認する必要があり、これも網羅的に調べることは困難である。本研究では、LAN の配線データと機器の設定データからパケットの状態遷移モデルを構成する方法を提案した。さらに、ネットワークが満たすべき性質を、状態遷移モデルの性質として、様相論理を用いて記述する方法を提案した。そして、状態遷移モデルが性質を満たすことを、検証ツール NuSMV を用いて行う方法を提案し、検証にかかる時間を実験的に評価して有用性を示した。

本論文では最後に第 7 章において、上記の研究を踏まえて、検証対象のネットワークの階層構造（レイヤ）における位置付け、プロトコルのような抽象的対象とその実装のような具体的対象、検証技術の発展、安全性と正当性、対象技術分野の発展の観点から形式検証の適用と課題について俯瞰的に論じて結論とする。

Evaluation of the Correctness and Security of Network Infrastructure and Network Services through Formal Verification

Abstract

With advances in semiconductor and network technologies, information systems consisting of computers connected via networks have become indispensable for various social activities. Accordingly, the need for information systems to operate as required, in other words, the need for system correctness, is also increasing. One method to rigorously guarantee the correctness of a system is formal verification, in which the correctness is described in a mathematically rigorous form, and the system is verified to meet the requirements.

This paper focuses on cryptographic protocols and network device configurations as concrete systems. For the cryptographic protocols, we focus on security, and for the network device configuration, we focus on properties such as the reachability of packets in the network after the configuration. Security is the property that dangerous situations such as information leakage or spoofing never occur under any attacker. For cryptographic protocols, we describe the results of evaluating the security of specific protocols by formal verification. For network device settings, we propose a method to evaluate them by formal verification.

In this paper, we first discuss cryptographic protocols and their security in Chapter 2. This is because the concept of an attacker is essential for security and involves more complex concepts than the correctness of network device settings. Next, in chapter 3, we give an overview of formal verification methods, including those used in this study. We then discuss the following three specific contributions to this paper.

In Chapter 4, we focus on the Secure Electronic Transaction (SET) payment protocol, an industry standard for online payments using credit cards. Among its security features, we verified the confidentiality of the customer's credit card number, which is sent and received during payment, using a method called the inductive method. The confidentiality of the credit card number is the most important property that is the basis of other security features. When this study was initiated in 2000, previous

studies that formally verified SET had greatly simplified the protocol and did not specify a procedure for simplification. Since the entire SET specification is more than 40,000 lines long, protocol simplification is inevitable to perform verification within a realistic amount of effort and time. In this study, we clarified the procedure to simplify the protocol, leaving the parts contributing to security and simplifying the rest. In addition, in previous studies that used inductive methods for formal verification, protocols were written as logical formulas, which were not necessarily highly readable, and errors could occur when writing the protocols. To solve this problem, we devised a language based on process algebra, equipped with a pattern-matching mechanism, and used this language to describe protocols concisely after simplification. Furthermore, we devised a method for converting protocols written in this language into logical formulas for the inductive method. Using these methods, we succeeded in verifying the confidentiality of credit card numbers,

In Chapter 5, we have verified the security of the Quick UDP Internet Connection (QUIC), a cryptographic protocol for authentication and encryption of communication between a web browser and a server. In a previous study, the security of QUIC was formulated using the framework of cryptography theory and proved by hand. In this study, we verified the security again using ProVerif, a tool for verifying the security of cryptographic protocols. The security formulation in the previous study was redefined and verified in a form that allows formal verification. The results of this study show that none of the security properties defined in the previous study are valid. A detailed analysis of the validation results showed that this was not due to a flaw in the protocol itself but rather that the definition of security in the previous study was too strong compared to the security that is really needed. It means that the definition of security in the prior study was incorrectly specified and inadequately proven. In this study, we further modified the problematic part of the definition of security. We succeeded in verifying the modified security. This study shows the effectiveness of formal verification because even when security has been proven by an expert in cryptographic protocols by hand, there may be flaws in the formulation and proof.

In Chapter 6, we focused on the configuration of local area network (LAN) devices and addressed the correctness of the configuration of VLAN technology, which virtually divides LANs. We proposed a method to verify this through formal verification.

In the past and even now, when a network is configured or modified, it is common practice to test by sending packets to confirm that there are no errors. Furthermore, to test that packets do not reach unintended recipients, monitoring the transmitted packets at the terminals of such recipients is necessary, which is also difficult to test exhaustively. In this study, assuming that data on LAN wiring and device settings are available, we devised a method to construct a state transition model from the data. Furthermore, we devised a method to describe various properties that a network must satisfy, including packet reachability, leakage, and the absence of loops, in a unified manner within the framework of modal logic. We devised a fast and automatic verification method for the state transition model and the modal logical formulas using NuSMV, a tool for verifying state transition models. Before this study, some studies verified the settings of firewalls and IPsec devices, but no studies verified VLAN settings.

Based on the above studies, in Chapter 7 this paper concludes with a discussion of the application and challenges of formal verification from the perspectives of network layers, abstract objects such as protocols and concrete objects such as their implementations, the development of verification techniques, security and correctness, and the development of the target technical field.

目次

図目次

表目次

第 1 章

序論

1.1　背景

半導体技術やネットワーク技術の進歩に従い、計算機およびそれらをネットワークで接続して構成される情報システムは広範な社会活動において必要不可欠となっている。これに伴い、情報システムが要求通りに動作すること、つまりシステムの正当性への必要性もますます高まっている。一般に、システムの動作を保障する方法として、システムに要求される動作を要求仕様として記述し、それに基づいてシステムを構成し、構成したシステムの動作を要求仕様と照合するという方法がある。

要求仕様はその目的に応じて、内容や詳細度、記述法などに多様性がある。内容としては、通常時に提供されるべき機能を記述した機能要件もあれば、悪意を持った利用者からの操作によっても情報の漏洩が起こらないなどの安全性要件など、様々である。詳細度についても、動作を人間が理解するための定性的な要件もあれば、実装の際の不整合を避けるための定量的な要件もありうる。記述法としても、直観的な理解のために自然言語によって記述する場合もあれば、厳密に動作を規定するために数式や論理式などを用いる場合もある。

システムの動作を要求仕様と照合、すなわち要求仕様の検証を行う方法についても、典型的な入力についてテストを行う方法もあれば、網羅的にテストケースを生成して自動的にテストを行う方法、あるいは、システムの実装に基づいて数理的・論理的に要求仕様を満足することを証明する方法もある。

要求仕様を数理的に厳密に記述してシステムの動作を検証する方法として、形式手法と呼ばれる用法がある。形式手法という名前の由来は、自然言語などの意味が明確に定義されていない非形式的な言語を用いず、主に数理論理学に基づく意味が厳密に定義された形

式的な言語を用いて要求仕様を記述することにある。形式的な言語による要求仕様記述により、曖昧な記述を排除することができ、実装されるシステムの正当性向上に資すると言われている。また、形式的な仕様記述に基づいてシステムの動作を網羅的に検証する方法を形式検証と呼ぶ。

形式的な仕様記述を行うためにはその記述言語にある程度精通していることが必要である。形式検証を行うためにはさらに、検証手法について精通している専門家が必要となる。このため、一般的なソフトウェア開発などで行われる自然言語による仕様記述やテストケースの生成による検証に比べ、大きな人的コストがかかる。

システムの正当性の観点からは、システムに求められるあらゆる動作を形式手法により厳密かつ定量的に記述し、形式検証により網羅的に検証することが望ましい。しかし現実的には、そのような記述・検証はごく小規模のシステムを除いて不可能である。また経済的な観点からも、仕様の記述と検証にかかる費用とシステムが期待どおりに動作しなかった場合の損失を考慮して記述・検証の方法が決められる。

システムが期待どおりに動作しなかった場合の損失として、サービスが提供できないことによる機会損失もあれば、改修や復旧のための費用も考えられる。特にシステムの規模や利用者数などにより、損失は大きくなる。さらに情報漏洩などを引き起こす場合には、財産的情報の流出や、社会的信頼の毀損などによる損失もありうる。また不具合の発見が遅れたり復旧に時間を要したりするために、損失が長期間継続する場合も考えられる。このため経済的な観点から、形式手法・形式検証を用いる費用が許容される場合として、改修や復旧が困難あるいは時間を要するシステムを対象とする場合と、低い費用で形式手法・形式検証を利用できる場合が考えられる。本論文では、前者の場合としてネットワークサービスを提供する際に用いられる暗号プロトコルの安全性を、後者の場合として、ネットワークの基盤をなすネットワーク機器の設定の正しさを取り上げ、具体的な対象に形式検証を適用し、適用にあたって生じる課題を解決する。

1.2 研究対象

本論文では、具体的なシステムとして暗号プロトコルおよびネットワーク機器の設定を取り上げる。

1.2.1 暗号プロトコル

暗号プロトコルは、通信において秘匿性や相互認証などの安全性を提供するための通信の手順である。暗号プロトコルでは一般に、暗号技術を用いてデータを構成し、参加者の間で互いにこれを送り合う。暗号技術としては、公開鍵暗号、共通鍵暗号、ディジタル署名、メッセージ認証コード、ハッシュ関数などが用いられる。暗号プロトコルとして、第 4 章で扱う電子決済などに用いられるプロトコルや、第 5 章で扱う web ブラウザと web サーバの間の通信を暗号化するためのプロトコルなど、さまざまな暗号プロトコルが実社会で利用されている。

暗号プロトコルは暗号技術の専門家によって設計され、場合によっては国際標準として公開される。そして暗号プロトコルはソフトウェアとして実装され、利用者に配布され使用される。また、一つの暗号プロトコルが複数の異なるソフトウェアによって実装されることも珍しくない。広く利用される暗号プロトコルに欠陥があり安全性が提供できない場合、悪意を持った攻撃者による攻撃により情報の漏洩やなりすましなどの被害を受ける可能性が、欠陥の程度によるものの、利用者全員について生じうる。その場合、欠陥を修正して対応することになるが、暗号プロトコルの修正には時間を要し、さらにそれをソフトウェアとして実装して配布する時間も必要であり、それまでの間は利用者が攻撃の可能性に晒され続けることになる。このため、暗号プロトコルが利用され始める前に、その安全性を厳密に確かめることが望ましい。

その一方、暗号プロトコルの安全性を確かめることは容易ではない。なぜなら、送受信されるデータを攻撃者がどのように解析するのか、さらに、攻撃者自身がデータを送信する場合も含め、現実的にありうるあらゆる攻撃を考慮する必要があるためである。安全性を暗号理論の専門家が数学的証明により確かめることが行われているが、高度な暗号理論の知識が必要であるため容易ではない。暗号プロトコルの規模や複雑さが大きくなるとなおさらである。このため、安全性が証明によって確かめられていない暗号プロトコルが広く用いられることもある。また、暗号技術の専門家によって設計された単純な暗号プロトコルにおいてさえも、安全だと思われていたものが実は安全でないということが起きうる [4]。暗号技術の専門家による安全性の証明に誤りが含まれる可能性もある。このため、暗号プロトコルの安全性を形式検証により厳密に、そして可能であれば自動的に検証できることが期待される。

また、本論文では扱わないが、暗号プロトコルを実装したソフトウェアの性質を検証することも重要である。なぜなら、暗号プロトコルの実装時に脆弱性が生じうるためであ

る。たとえば暗号化されたデータを処理する際に、データの内容によって処理時間が異なるなどの、実装に起因するサイドチャネル攻撃などが知られている。単純な暗号プロトコルであっても、その実装はデータの処理の方法や順序などを詳細に記述するため複雑になりうる。このため、暗号プロトコルの安全性の検証に比べ、その実装も含めた安全性の検証は、検証にかかるコストが大きい。

1.2.2 ネットワーク機器の設定

本論文の研究におけるもう一つの研究対象はネットワーク機器の設定である。ネットワークはネットワーク機器や端末の間にケーブルを配線して構築され、さらにネットワーク機器に適切な設定を行うことで利用可能になる。ネットワーク機器は設定に基づいて通信の速度や経路などを制御する。また、近年では単に設定と呼ばれる範囲を超えて、ソフトウェア的に制御されるネットワーク機器もあり、データセンタなどで広く利用されている。

ネットワーク機器の設定に誤りがあった場合、通信ができなくなる、あるいは、特定の通信経路が混雑するなどの障害が起こりうる。さらに、送信されたデータが本来それを受信すべきでない端末によっても受信できてしまう障害も考えられ、障害に気づかれず、また、障害が悪用されて情報漏洩に至る可能性もある。

ネットワークを構築したり構成や設定を変更したりする際には、正しく通信できることを確かめるために、実際にデータを送信してテストする。しかし、ネットワークの規模が大きい場合は、すべてのネットワーク機器や端末の間でデータを送信するテストを行うことが現実的でなくなってくる。また、送受信されるデータの種類や内容に応じて通信を制御する場合に、あらゆる種類や内容のデータを実際に送信することは不可能に近い。さらに、データが本来受信すべき端末にのみ届くことを確認するためには、全ての端末において到達するデータを監視する必要があり、これも現実的ではない。テストによりネットワークの配線や設定のどこかに誤りがあることがわかった場合は、その原因を特定して修正を行う必要がある。しかし、ネットワークは一般には多くの機器から構成されており、原因の特定が難航する場合もある。特に、複数のネットワーク機器の設定の不整合からなる障害の場合は、単独の機器の設定を確認しても原因が特定できない。このため、ネットワークの配線や設定が正しく行われていることを厳密かつ自動的に検証し、さらに誤りがあればその原因を特定できることが期待される。ネットワークは構築が完了した後も配線や設定が変更されることがあるため、その都度検証を行うことを考えれば、検証を高速に行える必要がある。本論文の研究においては、配線のデータとして正しいものが与えられ

ていると仮定し、これとネットワーク機器の設定の情報を用いて検証を行う。

1.3 本論文の構成

本論文の構成について述べる。具体的な貢献について述べる前に、第 2 章で暗号プロトコルとその安全性について述べる。これは、安全性には攻撃者の概念が不可欠であり、ネットワーク機器設定の正当性に比べ複雑な概念を含むためである。次に、第 3 章で、本研究で用いるものも含め、形式検証手法の概要について述べる。その後、本論文における次の 3 つの具体的貢献について述べる。

■SET 支払いプロトコルにおけるカード番号の秘匿性検証　第 4 章では、SET 支払いプロトコルにおけるカード番号の秘匿性検証の研究について述べる。この研究では、クレジットカードを用いてインターネット上で決済を行うための暗号プロトコルである SET（Secure Electronic Transaction）の支払いを行う部分を取り上げ、その安全性のうち決済時に送受信される顧客のクレジットカード番号の秘匿性を帰納的手法と呼ばれる方法で検証した。クレジットカード番号の秘匿性は、その他の安全性の基礎となる最も重要な性質である。本研究を開始した 2000 年当時、SET の形式検証については、プロトコルを大きく簡略化して行ったものしか存在せず、どのような手順を経て簡略化を行ったかも明示されていなかった。本研究においても仕様書全体で 4 万行を超える SET を現実的な労力と時間内で検証を行うためにプロトコルの簡略化を行ったが、安全性に本質的に寄与する部分を残しそれ以外を簡略化するという手順を明確にしてこれを行った。さらに、帰納的手法により形式検証を行う先行研究においてはプロトコルを論理式として記述するが、必ずしも可読性が良いとは言えず、プロトコルの記述の際に誤りが生じる可能性があった。このため、プロトコルを記述する言語として、プロセス代数の一種にパターンマッチングの機構を付加したものを考案し、これを用いて簡略化後のプロトコルを簡潔に記述できるようにした。さらに、この言語で記述したプロトコルを帰納的手法で用いる論理式に変換する方法を考案した。これらの手法により SET の安全性のうちクレジットカード番号の秘匿性の検証に成功した。

■QUIC プロトコルの安全性検証　第 5 章では、QUIC プロトコルの安全性検証の研究について述べる。この研究では、web のブラウザとサーバの間の通信において認証および暗号化を行うための暗号プロトコルである QUIC（Quick UDP Internet Connection）を取り上げ、その安全性を検証した。先行研究において、暗号理論の枠組みを用いて QUIC の安全性を定式化し、いわゆる紙と鉛筆を用いた手作業による証明を行った研究がある。

本研究では暗号プロトコルの安全性検証ツールである ProVerif を用いて改めて安全性の形式検証を行った。先行研究における安全性の定式化はそのまま形式検証に用いることができないため、これを形式検証に適した形で再度定義し直して検証を行った。先行研究の安全性のモデルは TLS 1.3 などの重要なプロトコルにも適用できるものであり、これを形式検証の枠組みで検証できるようにすることで、TLS 1.3 の安全性検証に応用し、その結果を比較することを可能にした。本研究による検証により、先行研究により定義された安全性がいずれも成り立たないことを明らかにした。検証結果について詳細に解析したところ、プロトコルそのものの欠陥ではなく、先行研究の安全性の定義は本来必要な安全性に比べて強すぎることが原因であることがわかった。これは、先行研究における安全性の定義に誤りがあり、証明にも不備があることを意味する。本研究ではさらに安全性の定義として問題がある箇所を修正して検証を行い、これに成功した。本研究により、暗号学理論的な枠組みにより安全性が証明されていた場合でも、その定式化や証明に不備がある場合があるため、改めて形式検証を行うことの有効性を示すことができた。

■ネットワーク機器の VLAN 設定の検証　第 6 章では、ネットワーク機器の VLAN 設定の検証の研究について述べる。この研究では、ネットワークの中でもローカルエリアネットワーク（LAN）の機器の設定に着目し、そのうち LAN 上で仮想的に LAN を構成する VLAN 技術の設定の正しさを取り上げ、これを形式検証により検証する方法を提案した。従来および現在においてもネットワークの構成や変更を行った場合は、実際にパケットを送信してテストを行い、誤りがないことを確認することが一般的だが、送受信する端末やパケットの種類を網羅してテストを行うことは現実的ではない。さらに、パケットが本来の相手以外に到達しないことをテストするためには、送信したパケットを、これを受信すべき端末以外の端末で監視し、到達しないことを確認する必要があり、これも網羅的に調べることは困難である。本研究ではまず、LAN の配線のデータとして正しいものが得られていると仮定し、この仮定のもとで設定のデータから状態遷移モデルを構成する方法を考案した。さらに、ネットワークが満たすべき性質として、パケットの到達、漏洩、ループの有無を含む様々な性質を様相論理の枠組みにより統一的に記述する方法を考案した。このようにして得られた状態遷移モデルおよび様相論理式を、状態遷移モデルの検証ツール NuSMV を用いて高速かつ自動的に検証する方法を考案した。本研究以前には、このような方法でネットワーク機器の設定を検証した研究としては、ファイアウォールの設定や IPsec 機器の認証・暗号化設定の正しさを検証する研究があったものの、VLAN 設定を検証する研究はなかった。なお、本研究を行って以降、特にデータセンタ等においてネットワークをソフトウェア技術により制御する Software-defined networking（SDN）技術

の進展に伴い、SDN により運用されるネットワークの設定に形式検証を適用する研究が行われるようになった。

本論文では最後に第 7 章で、上記の研究を踏まえて、検証対象のネットワークの階層構造（レイヤ）における位置付け、プロトコルのような抽象的対象とその実装のような具体的対象、検証技術の発展、安全性と正当性、対象技術分野の発展の観点から形式検証の適用と課題について俯瞰的に論じて結論とする。

第 2 章

暗号プロトコルとその安全性

本論文で述べる 3 つの研究のうちの 2 つは、暗号プロトコルの安全性を扱ったものである。このため、本章では暗号プロトコルとその安全性について、具体例を交えつつその概要について述べる。詳細については、文献 [5][6] などを参照されたい。

2.1 暗号プロトコル

暗号プロトコルは公開鍵暗号や電子署名などの暗号技術を用いて構成される。暗号プロトコルについての初期の重要な研究として、Needham と Schroeder による研究 [7] がある。彼らはネットワーク上で互いを認証するために暗号を用いる方法を提案した。より具体的には、共通鍵暗号を用いた認証プロトコルと公開鍵暗号を用いた認証プロトコルを提案している。暗号プロトコルがどのようなものかを見るために、ここでは後者のプロトコルを図 2.1 に示す。なお、この記述は [4] らによる記述をもとに、さらに、暗号によって保護されない部分を省略しているが、本質的には元のプロトコルと同じものである。このプロトコルは、アリス（A）とボブ（B）が通信により互いを認証する。すなわち、このプロトコルの実行により、アリスは、自分が通信している相手がボブであることを確信でき、ボブは、自分が通信している相手がアリスであることを確信できることを目的とする。逆に言えば、第三者（攻撃者）がボブになりすましてアリスを相手にプロトコルの実行を完了できる場合や、逆に攻撃者がボブになりすましてボブを相手にプロトコルの実行を完了できる場合は、目的が満たされないことになる。

プロトコルの動作について述べる。前提として、あらかじめアリスとボブはそれぞれ公開鍵暗号の鍵を生成し、このうちそれぞれの公開鍵 $pubk_A$ および $pubk_B$ を安全な方法で互いに共有してあるとする。アリスはまず、他者からは推測困難な乱数 N_A を

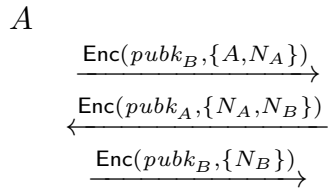

図 2.1　Needham と Schroeder による公開鍵認証プロトコル

生成し、これを自身の名前 A とともにボブの公開鍵 $pubk_B$ を用いて暗号化して暗号文 $\mathsf{Enc}(pubk_B, \{A, N_A\})$ を作り、ボブに送信する。ボブはこれを自らの秘密鍵を用いて復号し、アリスの名前 A および乱数 N_A を得る。ボブもまた推測困難な乱数 N_B を生成して、乱数 N_A とともにアリスの公開鍵 $pubk_A$ で暗号化し、暗号文 $\mathsf{Enc}(pubk_A, \{N_A, N_B\})$ をアリスに送る。アリスはこれを復号し、自身が送信した乱数 N_A が含まれていることを確認し、さらに乱数 N_B を得る。最初の暗号文では、アリスが生成した乱数 N_A をボブの公開鍵で暗号化したため、乱数 N_A を知ることができるのはアリス自身を除けばボブのみである。このため、N_A を含む暗号文を受信したアリスは、確かにボブが乱数 N_A を受信し、これに対する応答を行なったことを確信できる。アリスは次に、乱数 N_B をボブの公開鍵 $pubk_B$ で暗号化し、暗号文 $\mathsf{Enc}(pubk_B, \{N_B\})$ をボブに送信する。ボブこれを復号し、自身が送信した乱数 N_B が含まれていることを確認する。前の暗号文では、ボブが生成した乱数 N_B がアリスの公開鍵で暗号化したため、N_B を知るのはボブ自身を除けばアリスのみである。このため、N_B を含む暗号文を受信したボブは、確かにアリスが N_B を受信し、これに対する応答を行なったことを確信できる。

2.2　暗号プロトコルへの攻撃

暗号プロトコルを用いた通信に対する攻撃には様々なものがある。代表的なものとしては、盗聴、リプレイ攻撃、中間者攻撃、サービス不能攻撃（DoS）などである。前節のプロトコルを例に、盗聴、リプレイ攻撃、中間者攻撃について検討してみる。

まず、盗聴について、攻撃者がアリスとボブの間の通信を盗聴できた場合に、アリスやボブになりすますことができるかを考える。単に盗聴するだけの受動的な攻撃ではなりすましなどは行うことができない。プロトコルによっては秘密の情報を共有することが目的のものもあり、そのようなプロトコルでは、盗聴のみでも現実的な攻撃の方法になりうる。

次に、リプレイ攻撃について考える。アリスとボブがプロトコルを繰り返し実行しており、攻撃者がそれを盗聴している状況で、攻撃者が過去のプロトコル実行で盗聴したメッ

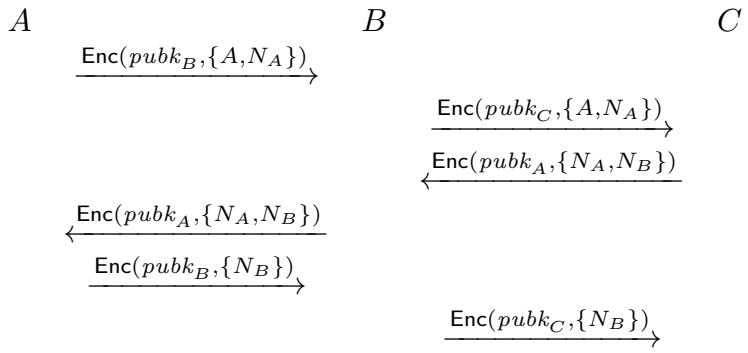

図 2.2　Lowe による攻撃

セージを再度メッセージを再度誰かに送りつけることで、アリスまたはボブになりすますことができるかどうか考える。なお、プロトコルの 1 回の実行をセッションと呼ぶ。ボブになりすますためには、ボブからアリスへ送られるべき暗号文を攻撃者がアリスに送る必要があるが、前のセッションで盗聴した暗号文には、アリスが現在の前のセッションで生成した乱数 N_A が含まれており、現在のセッションで生成した乱数が含まれていない。このため、リプレイによってなりすましを行うことはできないと考えられる。

次に、中間者攻撃について考える。ここで、多くの参加者が様々な相手とプロトコルを実行する状況を考えると、中間者攻撃が存在することが知られている [4]。この攻撃を図 2.2 に示す。この攻撃では、アリスがボブを相手にプロトコルを実行している状況を考える。この時、実はボブが攻撃者である場合は、アリスとの通信で用いるメッセージを利用して、チャーリー（C）に対して、アリスになりすますことが可能である。具体的な手順としては、ボブは最初のメッセージを復号し、それをチャーリーの公開鍵で暗号化し直して送る。次に、チャーリーからの応答をそのままアリスに転送する。そして、アリスからの応答を復号し、チャーリーの公開鍵で暗号化し直して送る。チャーリーの視点からは、チャーリーが送受信したメッセージはまさにアリスを相手としてプロトコルを実行した場合と全く同じであるが、アリスはチャーリーを相手としてはプロトコルを実行していない。

以上の考察から、プロトコルが 1 度だけ実行される場合、同じ参加者同士でプロトコルが繰り返し実行される場合、そして、多くの参加者がプロトコルを毎回異なる相手、繰り返し実行する場合、など多くの実行の状況が考えられることがわかる。また、Lowe の攻撃を見ると、「アリスは、自分が通信している相手がボブであることを確信できる」というような主張がやや曖昧であり、ボブもアリスと通信を行ったと確信できることまで要求

するかどうかが明確でないこともわかる。さらに、安全性を考えるにあたり、上記のような個々の攻撃ができないことを示すだけでなく、個々の攻撃によらないより一般的な安全性の条件を考え、この条件が成り立たない場合に相当する攻撃が存在しないことを示したい。

2.3 暗号プロトコルの実行モデル

前節の考察から、暗号プロトコルの実行される状況や攻撃の方法として様々なものが考えられ、プロトコルを設計する際にその全てをあらかじめ想定することは容易ではない。このため、網羅的に実行の状況や攻撃の方法を含むような、安全性の解析および検証の方法が必要となる。プロトコルが実行される状況および攻撃者に許される動作を、暗号プロトコルの実行モデルと呼ぶ。暗号プロトコルの実行モデルは、大別すると記号モデルと計算論的モデル（あるいは暗号学的モデル）に分けられる。以下、それぞれについて述べる。

2.3.1 記号モデル

暗号プロトコルの安全性検証についての初期の研究として、Dolev と Yao による研究 [8] がある。彼らは公開鍵暗号を用いるプロトコルの安全性を証明する際に、次の性質を持つ完璧な公開鍵暗号を仮定した。

- 暗号文は決して解読できない。
- 公開鍵は安全に公開されており、公開鍵を不正に置き換えられることはない。
- 任意の参加者が暗号化を行うことができる。
- 参加者は自分の公開鍵で暗号化された暗号文のみを復号できる。

この仮定のもとで、プロトコルで送受信されるメッセージを、暗号化や復号などの操作を表す記号の列とみなした。たとえば参加者 x の公開鍵および秘密鍵による暗号化および復号をそれぞれ記号 E_x および D_x で表した。記号の間の関係として D_xE_x などは空な記号列に等しいという関係を導入した。この関係を用いると $D_xE_xM = M$ という関係が得られる。これは、メッセージ M を参加者 x の鍵で暗号化し、さらに復号すると元のメッセージ M が得られることに対応する。この他、メッセージの連結やメッセージの一部の取り出しなどの操作も定義している。彼らはさらに、単なる盗聴のみではない能動的な攻撃者として、次のようなものを考えた。

- 攻撃者はネットワーク上を流れるメッセージを全て得ることがきる。

- 攻撃者もまたネットワークの利用者であり、任意の相手と通信を開始できる。
- 攻撃者は任意のユーザから接続を受ける可能性がある。

この時、攻撃者は通信を行うにあたり、ネットワークから得たメッセージに対して、暗号化、復号、メッセージの連結、メッセージの一部の取り出しなどの、操作を行って得られるメッセージを、ネットワーク上に送信できる。この攻撃者を想定することにより、様々な攻撃の方法を網羅的に扱うことが可能になる。彼らはこのようなモデルのもと、プロトコルの安全を定義して安全性を証明し、さらに、安全性を証明するためのアルゴリズムを考案した。彼らのモデルは現在では記号モデル、あるいは Dolev-Yao モデルと呼ばれる。記号モデルを用いた検証は自動化に適しており、さらに、安全性が満たされない場合に具体的な攻撃例を自動的に発見できる場合もある。このようなモデルはプロトコルの安全性を自動的に検証する研究において基礎となっており、第 3 章で述べる検証手法でも用いられている。

2.3.2 計算論的（暗号学的）モデル

記号モデルでは理想的に安全な暗号を仮定したが、現実的に広く使われている暗号アルゴリズムでは、暗号文はわずかな確率で解読が可能である。暗号理論的には暗号アルゴリズムの安全性は、確率的多項式時間の計算能力を持つ攻撃者を想定し、そのような攻撃者によって解読される確率が十分低いことと定義される。このような定義のもとで公開鍵暗号や電子署名などのアルゴリズムの安全性が証明されており、暗号技術の証明可能安全性と呼ばれる。この考え方を暗号プロトコルにも適用し、暗号プロトコルに対しても確率的多項式時間の攻撃者を想定するようなモデルを、暗号プロトコルの計算論的モデルあるいは暗号学的モデルと呼ぶ。計算論的モデルにおいては、プロトコル中で用いられる暗号アリゴリズムについて計算論的に定義される安全性を仮定する。暗号学的モデルにおいても、記号モデルと同様に、攻撃者はネットワークを流れるメッセージを傍受し、その計算能力の範囲内で解析して新たなメッセージを作り出し、任意の攻撃者に送信することができると仮定することが多い。

2.3.3 記号モデルの計算論的健全性

記号モデルに基づく検証では、暗号技術は理想的な安全性を持ち、暗号の解読や署名の偽造は全くできないことを仮定する。しかし、実際には暗号技術ではわずかな確率で解読や偽造などが成功するため、記号モデルに基づく検証によって暗号プロトコルが安全性で

あると示されたとしても、暗号学的モデルにおいても安全であるとは言えない。記号モデルにおいて安全ならば暗号学的モデルにおいても安全であるという性質をある意味において定義し、これを記号モデルの計算論的健全性と呼び、その証明を行う研究も行われている。その最初の研究として Abadi と Rogaway[9, 10] により受動的な攻撃者についての計算論的健全性が示されて以来、多くの研究があり、Cortier らのサーベイ [11] に詳しい。計算論的健全性はいつでも証明できるというわけではなく、暗号技術に対して計算論的に強い仮定を置く必要があったり、検証対象とする暗号プロトコルを制限する必要があったりする場合もある。そのような強い仮定や制限を避けるため、記号モデルにおける暗号技術の安全性をより弱めることで計算論的健全性を証明する研究がある [12, 13]。そのような記号モデルは従来の記号モデルよりも複雑なため、これに基づき自動的に検証を行うことができなかったが、最近になり自動検証を行う研究 [14] も現れている。

2.4 暗号プロトコルの安全性

暗号プロトコルの安全性は、プロトコルの利用目的によって様々である。一般的に要求されることが多い安全性として秘匿性、認証（真正性）、完全性が挙げられる。

秘匿性は、送受信される機密情報が攻撃者に知られないという性質である。秘匿性について、機密情報の全体を攻撃者が知ることができないという弱い秘匿性と、機密情報について情報が全く得られないという強い秘匿性が考えられる。記号モデルにおいても計算論的モデルにおいても、前者は攻撃者が機密情報を得る（出力する）ことができないという形で定式化され、後者は攻撃者が機密情報 S と S' を区別できない、すなわちどちらの場合も同様に振る舞うという形で定式化される。

認証（真正性）は、認証プロトコルの安全性として用いられ、互いに相手を認証する参加者が送受信するメッセージが一致することと定義される。たとえば、Needham と Schroeder による公開鍵認証プロトコルにおいては、アリスがボブを相手だと考えて通信を行い、それが終了した時に、アリスが送受信したメッセージと同じメッセージを送受信したボブが存在すれば、アリスはボブを認証できたことになる。Lowe による攻撃では、チャーリーはアリスが相手だと考えて通信を終了したにもかかわらず、アリスが送受信したメッセージとチャーリが送受信したメッセージが一致しない。このため、ボブ（より一般に、プロトコルにおいて最初にメッセージを受け取る側）にとっては認証が成り立たない。この考え方を一般化して、対応表明（第 5.4.4 節）によって安全性を定義することも行われている。

完全性は、一般にはデータが最新かつ正しい状態であることを意味するが、暗号プロト

コルにおいては、攻撃者がメッセージを改竄できないという性質とされることが多い。このため、正規参加者があるメッセージを受信した場合にそのメッセージが正規参加者によって送信された、という形で定式化されることが多く、前述の認証に近い性質として記述される。

プロトコルに特有の安全性を定義する場合に、攻撃者に情報を与えたり正規参加者を制御する能力を与えたりするなど、モデルを拡張することで、前述の安全性を拡張することも行われる。たとえば、第 5.4.2 節で定義されるモデルでは参加者の一部の鍵が攻撃者に漏洩されたり、参加者の一部が攻撃者に買収されたりする場合についての安全性を考えるため、攻撃者に鍵の漏洩や買収などの操作が許されている。

第 3 章

形式検証手法の概要

本章では形式検証手法の概要について、本論文で言及するものを中心に述べる。まず、形式検証を含む形式手法の概要について述べ、次に形式検証手法について汎用的な手法について述べ、最後に暗号プロトコルの検証に特化した手法について述べる。

3.1 形式手法

計算機のプログラムやハードウェア、ネットワーク、およびこれらが連携して動作するシステムなど、様々なシステムの動作を厳密に検証する手法として、形式検証（formal verification）がある。そもそも、形式検証は形式手法（formal methods）の一種である。形式手法はシステムに期待される性質を、厳密に意味が定義された言語により記述し、これをもとにシステムの開発を行う手法である。そのような仕様の方法を形式仕様記述（formal specification）と呼ぶ。

形式仕様記述には、命題論理や述語論理などの標準的な論理を用いる方法や、これをソフトウェア開発などの目的に応じて拡張した言語である言語（VDM[15, 16]、Z[17]、B[18] など)、等式に基づき記述する言語（OBJ[19]、CafeOBJ[20]、Maude[21]、CASL[22] など)、並行動作するプロセスを記述するための言語（CSP[23] など）およびこれを拡張したもの [24] など、多岐に渡る。形式仕様記述に特化して考案された言語については、その構文解析やシミュレーションなどの仕様の分析を行うツールとともに開発されているものもある。また、形式検証のための特定のツールに専用の仕様記述言語などもある。

形式検証では、対象のシステムの仕様と、システムの動作を記述したプログラム等を検証ツール（ソフトウェア）に入力し、システムの動作が仕様を満たすことを検証する。完全に自動的に検証を行える場合もあれば、部分的にしか自動化できず、人手での作業が発

大分類	小分類	ツール
汎用	定理証明システム	Coq (1984-), Isabelle (1985-)
	状態遷移システム検証	SMV (1993-), NuSMV (2000-)
	分散システム検証	Spin (1991-), Murϕ (1992-)
	代数仕様記述・解析	CafeOBJ (1998-), Maude (1999-)
暗号・プロトコル用	記号モデル	ProVerif (2001-), Tamarin (2012-), Verifpal (2019-)
	計算論的モデル	CryptoVerif (2005-), F* (2011-), EasyCrypt (2011-)

表 3.1　本論文で言及する形式検証ツールとその分類

生する場合もある。一般に、複雑なシステムや性質を記述できる、表現力の高い言語を用いるツールほど、自動化が部分的で人手での作業が必要になる。

形式検証のために使えるツールとして様々なツールが提案および開発されている。形式検証のためのツールとしては、汎用性が高いツールと、特定の応用に特化したツールがある。後者としては、本論文の研究でも行った暗号プロトコルの安全性の検証や暗号アルゴリズムの安全性の検証のためのツールのほか、特定のプログラミング言語で記述されたプログラムの正しさの検証に使われるツールなどがある。本章では、検証ツールの全体像について、本論文における研究で使用したもの、および、本論文で言及するツールを主に参照しつつ述べる。まず汎用性が高いツールについて述べ、次に暗号プロトコルおよび暗号アルゴリズムの安全性の検証のためのツールについて述べる。本論文で取り上げるツールおよびその分類を表 3.1 に示す。ここに示すのは、非常に多くのツールが提案され、現在も継続的に改良が続けられている中のごく一部である。ツール名には括弧書きによりツールが公開された年を付した。ただし、ソフトウェアとして公開された年は必ずしも明確でない場合もあり、開発の開始や論文の公開はこれに前後する。また、各ツールに関連する文献は本文中で参照する。

3.2　汎用的な形式検証ツール

形式検証の目的はシステムの動作を厳密に検証することであるため、論理に基づきシステムを定式化してその性質を証明することは自然な流れである。このために、定理証明システムと呼ばれるソフトウェアを用いることができる。特に、Coq[25, 26] や Isabelle[27] などの、高階論理に基づく定理証明が行えるツールは、その記述力の高さにより様々なシステムの記述や証明を行うことができる。一般に記述力が高い論理では、証明を自動的に行うことは難しいとされている。このため、これらのツールを用いる検証（証明）で

は、証明を行う者が証明の方針を決めて記述し、ツールの自動証明の機能を補助的に利用する。

記述力は低いものの自動化が可能な論理としては命題論理があり、命題論理の充足可能性問題を解くツールとして SAT ソルバやその拡張である SMT ソルバなどがある。また、SAT ソルバは他の検証ツールのバックエンドとしても用いられている。つまり、検証ツールの中には、システムの記述や性質を命題論理に変換し、システムの検証を充足可能性問題に帰着させるものがある。SAT ソルバや SMT ソルバについては、これらについてのコンペティションの web サイト [28, 29] に最新のツールなどの情報が掲載されている。

検証の対象という観点からは、基本的なシステムの形態として入力を与えると出力を返すものがあり、これをソフトウェアで実行する場合は入力を用いて出力を計算するプログラムを書く。そのようなプログラムの正しさを検証することは、形式検証の主なターゲットの一つである。プログラムの形式検証は長い歴史を持ち、Hoare[30] や Dijkstra[31] による論理的な定式化をはじめ非常に多くの研究が行われてきた。これらの手法は定理証明システム上で実現することも可能だが、より自動的な検証のためにプログラム検証に特化したツール（例えば Why3[32] など）の研究が行われている。

我々が日常目にするシステムには、入力を繰り返し受け付け、さらにそれに応じて内部状態を変更するシステムも多い。このようなシステムを状態遷移システムと呼び、形式検証の主要な対象である。状態遷移システムの形式検証を行うためのツールとして、SMV[33] やその派生である NuSMV[34] がある。これらのツールでは、システムの仕様を線形時相論理（LTL）[35] や計算木論理（CTL）[36, 37, 38] で記述し、状態遷移規則により与えられたシステムがこれを満たすことを自動的に検証することができる。システムの状態数が有限の場合は、状態数がかなり多い場合でも現実的な時間内で検証が完了することも多く、そこでは記号モデル検査や状態数削減などの手法が用いられている。また、無限状態のシステムであっても、状態集合を有限個の同値類に分けて同値類を状態とみなして検証を行う抽象化と呼ばれる手法により、有限状態のシステムに帰着して検証することも行われている。記号モデル検査や抽象化については、田辺ら [39] の解説に詳しい。

状態遷移システムには、単一のシステムのみでなく、複数の状態遷移システムが互いに通信を行って状態を遷移させるシステムも多い。このようなシステムは入出力の同期により通信を行う非決定性オートマトンやプロセス代数（CSP[23, 40]、CCS[41] など）を用いて記述・解析されることが多い。このようなシステムの性質を検証するツールとして Spin[42] や Murϕ[43] などがある。Murϕ はハードウェアデザインのために開発されたプロトコル検査ツールであり、キャッシュコヒーレンスプロトコルの正しさの検証などに用いられた。

これらのシステムは一般に何らかのデータを扱うため、形式検証ではデータやその性質について推論することが必要となる。データ型やその性質を等式により代数的に記述し、さらにこれをもとにシステムの仕様を記述してその性質を等式推論により検証するための言語およびツールとして、代数的仕様記述言語および解析のツールがある。OBJ[19] 言語およびこれから派生した言語やそれを採用したツールとして、CafeOBJ[20] などがあり、また書き換え論理に基づくツールとして Maude[21] などがある。

3.3 暗号および暗号プロトコルの検証のためのツール

暗号および暗号プロトコルの検証のためのツールには、第 2 章で述べた記号モデルに基づくものと、計算論的モデルに基づくものがある。

計算論的モデルに基づく検証は、暗号プロトコルの安全性検証の初期の頃から行われており、多くの研究がある。初期の研究として、汎用の定理証明システム（ソフトウェア）を用いる方法や、状態遷移システム一般の性質を検証できるソフトウェアを用いる方法がある。

定理証明システムを用いる方法として、Paulson による帰納的手法 [44] がある。この手法では、攻撃者の知りうる情報の集合や、参加者の動作によって発生するイベントの列の集合を帰納的に定義し、安全性をこの集合の性質として定義する。安全性の証明は部分的には定理証明システムの機能を用いて自動化することができ、プロトコルによらない一般的な性質をあらかじめ証明して利用することができる。しかし、基本的には検証者が手作業で行う必要があるため、定理証明システムに熟練している場合でもかなりの労力が必要となる。

汎用の状態遷移システム検証ツールは、与えられた状態遷移システムにおいて、与えられた初期状態から状態遷移を繰り返して特定の状態に到達するか否かを検査することができる。これを用いて暗号プロトコルの安全性を記述するためには、正規参加者と攻撃者の動作を状態遷移システムとして記述し、攻撃が成功した状態を定義してこれに到達しないことを検証する。このような方法として、Spin[42] を用いる方法 [45, 46] や、Murϕ[43] を用いてプロトコルのセキュリティを検査する手法 [47]、CafeOBJ[20] を用いる方法 [48] などがある。状態遷移システム検証ツールは自動的な検証が可能ではあるものの、暗号プロトコルの検証に特化していないため、記述に工夫が必要であり、また、攻撃者の動作時間（動作の回数）に上限を設けて検証を行う必要があるためこの上限を超えた動作については安全性が保証できないという問題がある。

その後、より暗号プロトコルに特化した検証手法が開発され、現在はこれを実装した

ツールを用いることが主流である。そのようなツールとして、ProVerif[49]、Tamarin[50]、Verifpal[51] がある。ProVerif では、暗号プロトコル、つまり正規参加者の動作をプロセス代数の一種を用いて記述するとともに、暗号プロトコルで用いられる暗号技術の性質を等式として与えることで、攻撃者のもとでのプロトコルの動作を自動的に検証することができ、安全でない場合は場合により具体的な攻撃例を出力できる。安全性も秘匿性や認証などを容易に記述することができる。検証を自動的かつ高速に行える一方、そのために参加者の状態を明示的に扱うことができないため、プロトコルによっては検証に失敗することがある。Tamarin は ProVerif と同様に、容易に暗号プロトコルやその安全性を記述でき、さらに参加者の状態をより正確に扱うことができる。その一方、検証が停止しない場合があるため、検証に必要な性質を補題として証明して与えられるようになっている。Verifpal はこれらのツールを含む先行研究を踏まえて、学生やエンジニアが直観的にプロトコルを記述し、検証結果を解釈できることを目指したツールである。

計算論的モデルに基づく検証ツールとして、CryptoVerif[52] がある。CryptoVerif は与えられたプロトコルを暗号理論的に等価な、異なるプロトコルに少しずつ書き換えていき、最終的に自明に安全なプロトコルに書き換えることができるか否かを検査することで、与えられたプロトコルの安全性を検証する。あらゆる書き換えを網羅的に調べることはできないため、書き換えの方向を検証者がある程度制御する必要がある。CryptoVerif はプロトコルを抽象的なレベルで記述して検証を行う一方、実行可能なプログラムを記述してこの安全性を直接検証可能なプログラミング言語およびツールとして F*[53] がある[*1]。また、暗号アルゴリズムの安全性の検証の枠組みとして、Hoare 論理を拡張して確率的な推論を行えるようにした確率 Hoare 論理があり、これを定理証明システム Coq 上に実装した CertiCrypt[54] や、さらに検証を自動的に行えるようにプログラム検証ツール Why3 上で実装した EasyCrypt[55, 56] もある。計算論的モデルに基づく暗号系の検証については、川本による解説がわかりやすい [57]。記号モデルに基づく検証も含めたサーベイとして、Barbosa らによるサーベイ [58] が詳しい。

また、暗号プロトコルを実装したソフトウェアの性質を検証する研究も行われている。上記で述べた F*はそのためのツールの一つである。この方向での研究についてまとめたサーベイとして Arquint ら [59] によるものがある。また、トランスポートレイヤプロトコルの実装の検証について第 5.6.2 節でも触れる。

[*1] 正確には、ライブラリを追加して可能になる

第 4 章

SET 支払いプロトコルの秘匿性検証

4.1 研究の背景と動機

インターネットを通じた商品やサービスを購入は、我々の日常生活に欠かせないものとなっている。その際の支払いの手段としてクレジットカードを用いることも一般的である。クレジットカードを用いた支払いでは、顧客、つまりクレジットカードの利用者は、自身のクレジットカードの番号（Primary Account Number、PAN）を商店（加盟店）に送信し、加盟店はこれを支払い金額などの情報ともにクレジットカード決済網に送信して支払いの承認を求め、承認されれば決済、つまり顧客の銀行口座から加盟店の銀行口座への送金がクレジットカード決済網を通じて行われる。この手続きから容易に分かるとおり、加盟店が悪意を持っていた場合は支払い金額の改ざんや、カード番号を再利用しての不正な支払いを行うことができてしまう。また、第三者が何らかの方法でカード番号を入手できた場合も同様である。実際にそのような被害は増加しており、クレジットカードを発行する各社はその対策を迫られてきた。

被害を減少させる対策として、クレジットカード裏面にセキュリティコード（CVC、CVV）と呼ばれる数字を記載し、インターネットを通じた購入の際にはカード番号と合わせてこれを入力させることが行われている。セキュリティコードはクレジットカードの磁気ストライプ（磁気テープ）に記録されないため、店舗などにおいて磁気ストライプを不正に読み出す、スキミングと呼ばれる不正においてはセキュリティコードは読み出されない。その意味では一定の対策にはなっているものの、インターネット上の加盟店による金額の改ざんや、セキュリティコードを保存して行う不正への対策にはなっていない。

不正利用に対する抜本的な対策を図るため、クレジットカードの国際ブランドである Visa と MasterCard を中心として、SET（Secure Electronic Transaction）[60, 61, 62]

と呼ばれる暗号プロトコルが策定された。SET では、顧客と加盟店がそれぞれ支払い金額などの情報にディジタル署名を行い、支払いが決済ネットワークで承認される前に、署名と金額の一致が確認される。また、カード番号は加盟店にも読めないよう暗号化される。SET を用いる場合、顧客は専用のソフトウェアを自身の計算機にインストールし、そのソフトウェアが加盟店のサーバーと SET のプロトコルに従って通信し支払い処理を行う。

4.2 先行研究と課題

Bolignano[63] は SET を大幅に簡略化して得られる決済プロトコルを定式化し、その安全性を検証した。PAN などの秘匿性の他、支払いゲートウェイがトランザクションの承認要求を受け付けたとき、必ず対応する加盟店からの承認要求が存在する、などの性質を検証した。プロトコルおよび安全性の定式化と検証は、汎用の定理証明ツール Coq[25] 上で行った。

Lu と Smolka[64] は SET の支払いプロトコルをプロセス代数 CSP[23] を用いて記述し、加盟店による二重計上や金額の改ざん、顧客が注文金額よりも低い金額で決済を試みるなどの特定の攻撃について、それが成功しないことを検証した。検証には分散システムの検証ツール FDR[65] を用いて、並列に動作する顧客の数を 2 人に限るなどの上限を設けて検証を行なった。

Meadows[66] らは SET の支払いプロトコルの安全性を、彼らが開発しているプロトコル検証ツールである NRA Protocol Analyzer のための安全性の記述言語 NPATRL により記述した。しかし、安全性の検証には至っていない。

なお、本節の研究と同時期に行われた研究として Bella らの研究がある。Bella ら [67] ははまず、SET の支払いプロトコルを実行するための準備にあたる、顧客登録のプロトコルの安全性を検証した。さらに、支払いプロトコルの安全性の検証も行い [68, 69]、支払い情報の秘匿性だけでなく、加盟店が決済承認のメッセージを受け取ったとき、確かにその承認を行った加盟店が存在することなど、決済の承認に関するいくつかの性質も検証した。彼らの検証は、Paulson による帰納的方法に基づいており、並列に動作する参加者の数などを限定しない検証を行なっている。

本研究を開始した 2000 年の時点では、次のような課題があった。

- 先行研究における SET の安全性検証の研究は、いずれも SET を大きく簡略化して得られたプロトコルを対象としていた。このため、過度な簡略化を行わない形で

の検証が必要であった。

- SET は仕様が 400 ページ以上にわたる大きなプロトコルであるため、これにそのまま形式検証を適用することは、形式検証ツールへ入力できる形で記述し直す手間、および、検証ツールによる検証に要する時間の観点から現実的ではなく、簡略化が必要である。検証の妥当性について判断するためにはこの簡略化のプロセスが明確である必要があるが、先行研究においてはこの点が明確でなかった。
- 簡略化のプロセスを経て得られたプロトコルをわかりやすく記述でき、さらに形式検証に利用できる言語は、少なくとも Paulson の帰納的手法による検証に用いることができるものは存在しなかった。なお、Bella らの検証ではプロトコルを論理式として直接記述しており、必ずしもわかりやすいとは言えない。

4.3 本研究の貢献

本研究では、Paulson らの帰納的方法を用いて、SET の支払いプロトコルにおけるクレジットカード番号の秘匿性の検証を行う。クレジットカード番号の秘匿性は、クレジットカード取引に必要なそのほかの安全性の前提となる重要な性質である。特に本研究においては、

- Bolignano、および、Lu と Smolka らによる SET の支払いプロトコルの検証のような過度な簡略化を避けるとともに、簡略化のプロセスを明確にして検証を行なった。
- このために、プロトコルを直観的にわかりやすい形で記述する言語を提案した。この言語は、プロセス代数の一つである値渡し CCS[40] にパターンマッチングの機構を追加したものである。そして、簡略化によって得られたプロトコルをこの言語で記述した。Paulson らが行ったような論理式を直接記述する方法に比べ、プロトコルの記述を理解しやため、記述の誤りなどにより検証の妥当性が損なわれることを避けることができる。
- さらに、この言語で記述したプロトコルを、Paulson らの帰納的方法のための論理式に変換する方法を提案し、これを用いて SET の支払いプロトコルの検証を行った。これにより、Paulson らの方法およびそのための証明スクリプトを利用することで効率的に検証を行うことができる。

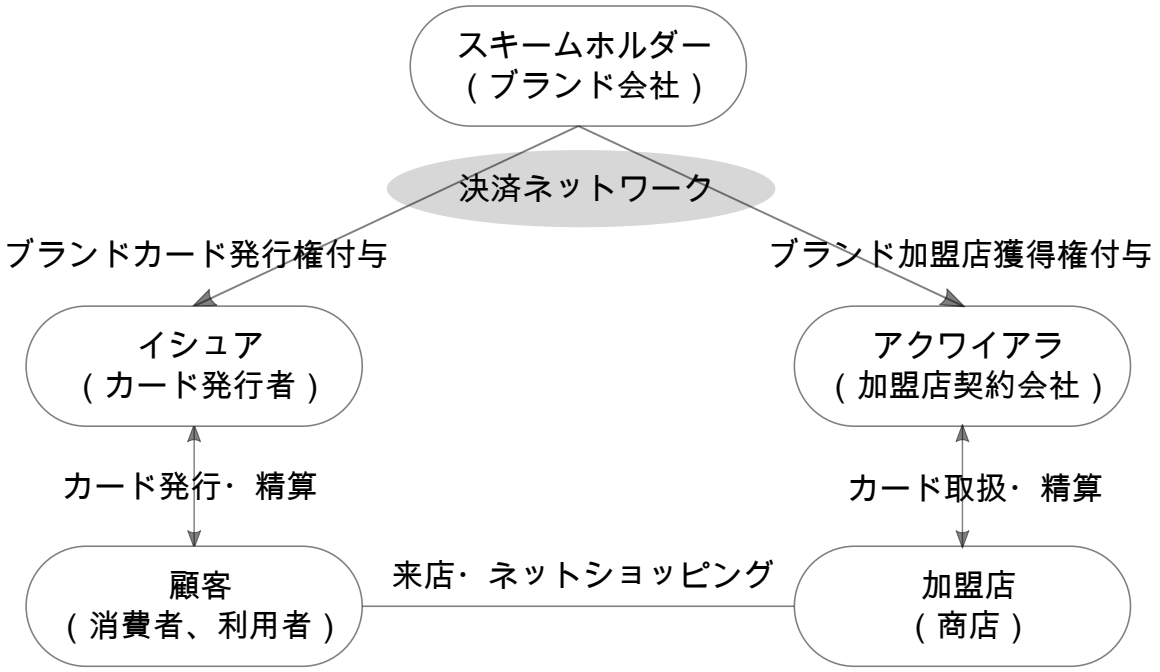

図 4.1　クレジットカード決済網（[1] をもとに作成）

4.4　準備

4.4.1　クレジットカードを用いた決済の概要

クレジットカードによる決済の概要を図 4.1 に示す。クレジットカードを利用する消費者（顧客、あるいは、利用者）は、クレジットカード発行会社（イシュア）と契約し、カードの発行を受ける。カードを用いて決済ができる小売店（加盟店）は、加盟店契約会社（アクワイアラ）と契約し、店舗において決済を行うための端末などを設置する。カードによる決済は、大きく分けると、店舗においてカードを提示して行う決済（Card Present, CP）と、インターネットなどの通信手段においてカードの情報を提示して行う決済がある。

店舗における支払いでは、顧客がカードを加盟店に提示する。加盟店は利用者のカードを端末に挿入し、端末を通じて決済に必要な情報を得る。加盟店はその情報を元にアクワイアラに決済の承認と代金の清算を依頼する。決済が承認されれば、クレジットカード決済網を通じてイシュアとの間で行われる。イシュアは清算した代金の支払いを顧客から受ける。

電話やファクシミリ、そしてインターネットを用いた通信販売においてはカードを提示せず、必要な情報を通信により情報を送受信する。これはカードを提示しない（Card Not Present、CNP）決済と呼ばれる。顧客が加盟店にカード番号（PAN）を送信する方法が最も基本的だが、不正に入手した PAN を用いて商品を購入する不正利用の可能性がある。PAN の不正な入手の方法として、店舗などでスキミングにより読み取る方法、通信販売業者が顧客から受信した PAN を不正に入手するなどが考えられる。このような攻撃に対する、暗号プロトコルを用いた最初の対策が SET プロトコルである。

なお、プリペイドカードによる決済（前払決済）およびデビッドカードによる決済（即時決済）についても、国際クレジットカードブランド（Visa や MasterCard など）が仕組みを作っているものはクレジットカードと同様の仕組みで決済が行われる[*1]クレジットカード決済との違いは、基本的には支払いが前払い、即時、後払いのいずれかであるかの違いである。

4.4.2 SET の支払いプロトコル

SET（Secure Electronic Transaction）はクレジットカード (payment card) を用いた電子決済のためのプロトコルで、Visa と MasterCard などが中心になって策定したものである。SET の目的は、

- 情報の秘匿性
- 支払いの信頼性
- 加盟店 (merchant) と顧客 (cardholder) の認証

を提供し、インターネット・ショッピングなどの新しい市場でのクレジットカードの使用を促し、クレジットカードがより広い範囲で受け入れられるようにすることである [60]。

SET は大きく分ければ、顧客と加盟店の鍵を認証局に登録する部分と、顧客・加盟店・カード会社の間で売買の情報を交換する支払いプロトコルの部分からなる。本研究ではこのうち実際の支払いを行う後者の部分を扱う。

SET の支払いプロトコルの特徴は、図 4.2 のように顧客、加盟店、支払いゲートウェイの 3 者が参加する点である。支払いゲートウェイはアクワイアラに相当する参加者で、SET のネットワークと既存のクレジットカードのネットワークを結ぶ。顧客のクレジットカード番号 (PAN) は注文要求のメッセージに含まれて加盟店に送信されるが、暗号化

[*1] 我が国で利用されている、銀行のキャッシュカードを使ったデビッドカード決済（J-Debit）は、これと異なる仕組みが採用されている。

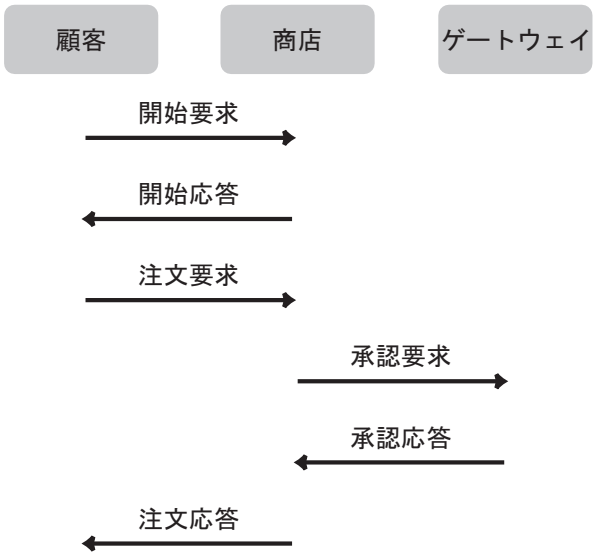

図 4.2　SET の支払いプロトコルにおけるメッセージの流れ

により加盟店には読まれない。このため加盟店によるカード番号を利用した詐欺を防止できると期待される。

しかし、SET の支払いプロトコルが安全性要求を実現しているかは明らかではないため、検証を行う必要がある。SET の支払いプロトコルには図 4.2 に示した以外にも代金引き落としを取り消すメッセージなどがあるが本研究では売買の成立を確認するために必要な図 4.2 の 6 つのメッセージに絞って秘匿性の検証を行う。

4.4.3　定理証明支援システム Isabelle/HOL とその論理体系

本研究で用いる定理証明支援システム Isabelle/HOL の概要について述べる。まずシステムの概要について述べ、続いて、本研究の内容が論理としてどのように基礎付けられるかを理解するため、Isabelle/HOL で用いる論理体系 HOL およびその拡張方法について概要を述べる。さらに、次節で述べる方法の説明を一部先取りして、プロトコル検証のための拡張についても述べる。Isabelle/HOL の開発者らによれば、HOL は古典集合論に型をつけたものとして理解できる、とのことである。論理体系 HOL およびその拡張の方法の説明においては λ 計算および論理学の基礎的な知識（[70][71] など）を仮定するものの、直観的には、論理体系 HOL で用いられる論理記号や集合の記号は通常の集合論の記号とみなせる。また、Isabelle/HOL にはこの論理体系を基礎として、帰納的に定義されるデータ型（リストや木構造など）による拡張が可能である。詳細については、

Isabelle/HOL の配布元 web サイト [27] にある文献を参照されたい。また、文献 [72] でも簡潔に解説されている。

Isabelle システム

Isabelle システムは、一階述語論理や集合論などの様々な論理体系を定義し、その論理体系において証明を行うためのソフトウェアである。Isabelle/HOL は、Isabelle 上に定義された高階論理の論理体系 HOL を用いて、HOL 上で証明を記述・検証するためのソフトウェアを指す。Isabelle/HOL の利用者は、必要に応じて定義を追加することで論理体系 HOL を拡張し、その論理体系のもとで定理を記述し、定理の証明を対話的に行うことができる。対話的な証明において、利用者はタクティクと呼ばれるコマンドを実行して、証明すべき論理式を推論規則に従い書き換えていくことで証明を行う。タクティクの中には推論規則を次々に自動的に適用し、証明を自動的に行うことができるものも用意されている。しかし、簡単な論理式を除いて、一般には完全に自動的に証明を行うことができないため、基本的には利用者が証明の方針を考え、その方針に従ってタクティクを実行することになる。

Isabelle システムには高階論理の部分体系があらかじめ組み込まれており、メタロジックと呼ばれる。メタロジックの上で様々な論理体系を記述できることから、ロジカルフレームワークとも呼ばれる。メタロジックを用いて、一階述語論理の命題や証明などはメタロジックの項として定義される。また、高階論理 HOL もメタロジックを用いて記述されるため、高階論理の部分体系を用いて高階論理を記述することになる。ただし本論文では論理体系の記述の方法については立ち入らない。Isabelle のメタロジックについては、文献 [73][74] に詳しく述べられている。

高階論理の論理体系 HOL

Isabelle/HOL の論理体系 HOL の概要について述べる。Isabelle/HOL の論理体系 HOL は、文献 [75] に詳しく述べられている通り、HOL システム [76] の論理体系 HOL[77] に基づいている。ここでは、HOL システムにおける論理体系 HOL について述べた文献 [77] に沿って概要を述べる。

HOL の項は、λ 計算の項であり、次の文法で記述される。

$$t \quad ::= \quad x \quad | \quad (t_1\ t_2) \quad | \quad \lambda x.t$$

含意 $\supset$、連言 $\wedge$、選言 $\vee$、否定 $\neg$、全称 $\forall$、存在 $\exists$ などの通常の論理記号は定数記号（定数を表す変数記号）として与えられる。このため、通常の一階述語論理における

$A \supset B$ 及び $\forall x.A$ などの論理式はそれぞれ $((\supset\ A)\, B)$ 及び $(\forall\, \lambda x.A)$ と記述される。また、Hilbert の ϵ 演算子も定数記号として与えられている。しかしこれでは読みづらいため、Isabelle/HOL 上では通常の記法 $A \supset B$ 及び $\forall x.A$ も用いることができる。関数適用についても $(f\, t)$ の代わりに $f(t)$ と記述できる。また、HOL においては、論理式を表す文法クラスは存在せず、論理式も項として表現される。上記の文法で許される任意の項を可能であるとすると矛盾した体系となるため、型付け可能な項のみが許される。型変数を許す型体系 [78] を採用している。

HOL の演繹システムは自然演繹を採用している。公理としては、反射律 $t = t$ や β 変換 $(\lambda x.t_1)t_2 = t_1[t_2/x]$ などが与えられている他、推論規則として変数への代入に関するもの、含意 $\supset$ の導入と除去（自然演繹における discharge と modus ponens）、および、ϵ 演算子についての推論規則が与えられている。HOL では論理体系を拡張する際に、定義の公理（$c = t$ の形の公理）を追加し、他の形の公理や推論規則をできるだけ追加せずに拡張が行えるようになっている。実は前述の論理記号は全て、含意 $\supset$ および Hilbert の ϵ 演算子を用いた項として定義可能で、他に公理や推論規則を追加することなくその性質を導出できる。ϵ 演算子を用いた項 $\epsilon\, \lambda x.P$ は $\epsilon x.P$ とも書かれ、「命題 P を満たすある x」を意味する。ϵ 演算子を用いて、存在記号を含む論理式 $\exists x.P$ を $P(\epsilon x.P)$ と書くことができる。HOL は純粋に論理的な核となる体系に、後述の拡張方法による拡張を行い、集合論、算術、原始帰納的関数、タプル及びリストについての理論を追加した体系となっている。Isabelle/HOL ではさらに、実数や文字列など、様々な理論がライブラリとして用意されている。なお、Isabelle においては、たとえば前提 $P(x)$ および $Q(x)$ から結論 $R(x)$ を導く HOL の推論規則は、メタロジックの論理式

$$\bigwedge x.\, \mathrm{Trueprop}(P(x)) \implies (\mathrm{Trueprop}(Q(x)) \implies \mathrm{Trueprop}(R(x)))$$

と記述される。$\bigwedge$ および $\implies$ はメタロジックの全称記号および含意記号である。Trueprop は HOL の論理式（bool 型）をメタロジックの論理式（prop 型）に強制変換（coercion）する関数記号である。定義対象の論理（HOL）とメタロジックの論理式の型を分離することで、HOL への拡張がメタロジックに影響を及ぼすことを避けられる。Isabelle システム上ではたとえば上記の推論規則は、通常は Trueprop を省略し、さらに、

$$\bigwedge x.\, [\![P(x); Q(x)]\!] \implies R(x)$$

と略記される。

HOL の意味論では、型は集合として解釈され、その型を持つ項はその集合の要素として解釈される。詳細な説明については文献 [77] およびその他の文献に譲るが、直観的に

は、項 $(t_1\, t_2)$ においては、t_1 は関数として解釈され、$(t_1\, t_2)$ はその関数に引数として t_2 の解釈を与えた結果の値と解釈される。また、項 $\lambda x.t$ は関数として解釈される。この関数は、引数が与えられたとき、t の解釈において変数 x を関数の引数の値で置き換えたものを返す。項の型に型変数が出現する場合は、型変数に対応する集合が与えられた時に、その項の意味が定まる。前述の論理記号や、集合論で一般的に用いられる記号は、通常の集合論と同様に意味が与えられる。

高階論理 HOL の拡張方法

定義の公理を用いた拡張の他に、命題や集合などを帰納的に定義することができる [79]。新たな集合を帰納的に定義する際、利用者はこれを推論規則の形で与える。ただし、Isabelle/HOL のシステムはその推論規則を直接体系に追加するのではなく、推論規則を満たす最小の命題や集合などを、最小不動点として定義する。これは定義の公理を追加することで行われる。さらに、Isabelle/HOL はこの定義から、利用者が与えた推論規則が導出可能であることを（メタロジックの上で）証明し、導出された推論規則（derived inference rule）として利用可能にする。さらに、場合分けや帰納法の公理も同時に自動的に証明する。また、最小不動点の存在を示すためには利用者が与えた推論規則の単調性が必要であるが、多くの場合これもシステムが自動的に証明する。

さらに、上記のように定義した（空でない）集合に対応する型を次のように定義することができる。その集合の要素の型を τ、新たに定義したい型を τ' とすると、まず、τ' から τ への関数の型を持つ定数記号 f を考え、f がその集合への全単射であること表す公理を追加することで、集合と型 τ' を同一視できる。

この仕組みを用いて、様々なデータ構造の型（データ型）を定義できる。たとえば自然数を要素とするリストのデータ型を定義する場合、まず、空のリストを表す定数 Nil および、与えられたリストの先頭に与えられた要素を追加したリストを作る関数 Cons を定義し、これによって作られるすべてのリストを含む最小の集合を定義する。次に、この集合を型 τ' に対応させる。型 τ' における Nil および Cons の対応物は全単射を通じて自然に定義できる。Isabelle/HOL では、データ型の定義にあたりその定義の仕組みを隠蔽しており、利用者は通常は Nil と Cons の型 τ' についての対応物のみを用いて項を構成して用いる。

暗号プロトコルの検証のための拡張

次節の議論を一部先取りする形で、暗号プロトコルの検証のための拡張について述べる。暗号プロトコルではメッセージを送受信するため、メッセージのデータ型を前述の方

法により定義する。詳細は文献 [80] に譲るが、Isabelle/HOL で提供される理論 Auth では、メッセージのデータ型 msg は、Isabelle の構文では、次の文法を用いて帰納的に定義される。

```
datatype
   msg = Agent  agent
       | Number nat
       | Nonce  nat
       | Key    key
       | Hash   msg
       | MPair  msg msg
       | Crypt  key msg
```

ここで、各行の等号および縦線 '|' のすぐ右の Agent などは msg 型への関数を表す定数記号であり、その右に列挙されているのがその入力の型である。Agent、Number、Key、Hash、MPair、Crypt は、これらによって構成されるメッセージがそれぞれ参加者の名前、数量、暗号や署名のための鍵、ハッシュ関数の出力、メッセージのペア、暗号文であることを表す。agent、nat、key はそれぞれ、参加者、自然数、鍵の型である。また、Auth には公開鍵暗号または共通鍵暗号の理論が含まれており、暗号化（または、復号）のための鍵 K から、復号（または、暗号化）のための鍵を得る関数 invKey が定義されており、等式 invKey K (invKey K) = K などが公理として与えられている。

本章で述べる SET の検証のために、数量を表すデータ型を単に自然数から構成されるのではなく価格と注文数の 2 種類に分けるような拡張を行った。さらに、乱数の型 nonce を SET プロトコル中で用いられる 2 種類の乱数の型に分けて定義する拡張を行った。

メッセージのデータ型の解釈は第 2 章で述べた記号モデルのメッセージとして解釈される。記号モデルでは、メッセージは乱数などの基本的なデータをもとに暗号化などの操作を行う記号を用いて構成した記号列とされる。このため、一階述語論理の項モデル（term model）と同様に、メッセージの項それ自身と解釈される。ただし、invKey の等式などの公理によって要請される同値関係により、解釈によって得られたメッセージに同値関係を導入していると考える。次節に述べる手法は、この解釈を通じて記号モデルにおける検証を形式的に記述しているものと考える。

上記の他にも、メッセージの送信を表す項 Says A B M および参加者によるメッセージの記録を表す項 Notes A M からなるイベントの型を定義するが、これは可読性のために定義されており、実質的には引数のタプルと考えてよい。

次節以降では可読性のため、Agent、Number、および Key を省略するとともに、Hash(m)、MPair(m_1, m_2)、Crypt(k, m)、をそれぞれ H(m)、$\{m_1, m_2\}$、$\{m\}_k$ と表記する。また、$\{m_1, \{m_2, m_3\}\}$ などは $\{m_1, m_2, m_3\}$ と略記し、$\{\{m, m'\}\}_K$ などは括弧を省略して $\{m, m'\}_K$ と略記する。また、イベントを構成するための関数記号を Says および Notes と書き、参加者やメッセージなどを表す具体的な変数は A, M などとタイプライタ書体で書く。

4.4.4 Paulson の帰納的方法

プロトコルを定理証明の技術を用いて検証する Paulson の帰納的方法について述べる。この方法では、プロトコルはトレースの集合を帰納的に定義する推論規則の集まりとして記述される。トレースは Says $A\ B\ M$ という形のイベントのリストである。Says $A\ B\ M$ は参加者 A が参加者 B にメッセージ M を送信するイベントである。トレースの集合 traceSet を定義する推論規則は一般に次の形をしている。

$$
\begin{array}{l}
[\![\texttt{trace} \in \texttt{traceSet}\,; \\
\quad \text{Says } A\ B_1\ M_1 \in \text{set}\,\texttt{trace}\,; \\
\qquad \vdots \\
\quad \text{Says } A\ B_n\ M_n \in \text{set}\,\texttt{trace}\,; \\
\quad \text{Says } C_1\ A\ L_1 \in \text{set}\,\texttt{trace}\,; \\
\qquad \vdots \\
\quad \text{Says } C_m\ A\ L_m \in \text{set}\,\texttt{trace}\,; \\
\quad \mathit{cond} \\
\Longrightarrow\ ([\text{Says } A\ B\ M] + \texttt{trace}) \in \texttt{traceSet}
\end{array}
\tag{4.1}
$$

直観的には、この推論規則は「参加者 A が参加者 $B_1, \ldots, B_n$ にそれぞれメッセージ $M_1, \ldots, M_n$ を送信し、参加者 $C_1, \ldots, C_m$ が参加者 A にそれぞれメッセージ $L_1, \ldots, L_m$ を送信し、さらに条件 *cond* を満たすならば、参加者 A は参加者 B にメッセージ M を送信する.」という意味である。この推論規則により定義されるトレースの集合 traceSet は、参加者がプロトコルに従って行動したときに発生させるイベントのリストの集合である。*cond* にはメッセージの性質などが記述される。'+' はリストの連結、'∈' は集合の要素であること、'set' はリストの要素を得る関数に対応する記号である。

前節で述べた Isabell/HOL の論理との対応関係で言えば、$[\![\ P_1; \cdots ; P_n]\!] \Longrightarrow P$ は $P_1, \ldots, P_n$ を前提として P を導く推論規則である。帰納的定義のために与える推論規則は、これをそのまま論理体系に追加するのではなく、この推論規則を導出できるような公理が Isabelle/HOL システムによって追加される。trace および traceSet は

$$
\begin{array}{l}
[\![\texttt{trace} \in \texttt{traceSet}\,;\ \texttt{Na} \notin \text{used}\ \texttt{trace}\ ]\!] \\
\quad \Longrightarrow\ ([\text{Says}\ \texttt{A}\ \texttt{B}\ \{\texttt{A}, \texttt{Na}\}] + \texttt{trace}) \in \texttt{traceSet}
\end{array}
$$

$$
\begin{array}{l}
[\![\texttt{trace} \in \texttt{traceSet}\,; \\
\text{Says}\ \texttt{A}\ \texttt{B}\ \{\texttt{A}, \texttt{Na}\} \in \text{set}\,\texttt{trace}\,; \\
\text{Says}\ \texttt{B'}\ \texttt{A}\ \{\{\texttt{B}, \texttt{K}, \texttt{Na}, \texttt{Nb}\}_{\text{shrK(A)}}, \texttt{X}\} \in \text{set}\,\texttt{trace}\ ]\!] \\
\quad \Longrightarrow\ ([\text{Says}\ \texttt{A}\ \texttt{B}\ \{\texttt{X}, \{\texttt{Nb}\}_{\texttt{K}}\}] + \texttt{trace}) \in \texttt{traceSet}
\end{array}
$$

図 4.3　Yahalom のプロトコルのイニシエータを定義する推論規則

Isabelle/HOL 上の変数である。$A, B_1, \ldots, B_n, M_1, \ldots, M_n$ などは Isabelle/HOL の項を表す。この推論規則は、変数 traceSet に対応する集合の性質を表している。このため、変数 trace および $A, B_1, \ldots, B_n, M_1, \ldots, M_n$ などに現れる変数、つまり traceSet 以外の変数は全てメタロジックの全称記号により束縛されると理解されるが、Isabelle/HOL の帰納的定義の記法では、これを省略して記述する。

例 1　Yahalom のプロトコル [81] におけるイニシエータを定義する推論規則 [82] を図 4.3 に示す。ここで shrK(A) は参加者 A と鍵サーバとの間であらかじめ共有されている鍵である。この 2 つの推論規則のうち最初の推論規則は、イニシエータである参加者 A が新しいノンス Na を生成し、参加者 B に {A, Na} を送信するという意味である。2 番目の推論規則は、過去に参加者 A が参加者 B にメッセージ {A, Na} を送信し、さらに誰か (B') からメッセージ $\{\{\texttt{B}, \texttt{K}, \texttt{Na}, \texttt{Nb}\}_{\text{shrK(A)}}, \texttt{X}\}$ を受けとったとき、B に $\{\texttt{X}, \{\texttt{Nb}\}_{\texttt{K}}\}$ を送信するという意味である。used trace は trace の中で過去に送信されたメッセージに出現するメッセージの集合である。Na $\notin$ used trace は式 (4.1) の *cond* に相当し、Na が新しく生成したノンスであることを意味する。

いま、

$$[] \in \texttt{traceSet}$$

を仮定し、図 4.3 の最初の推論規則を適用すれば、

$$[\text{Says}\ \texttt{A}\ \texttt{B}\ \{\texttt{A}, \texttt{Na}\}] \in \texttt{traceSet}$$

が得られる。さらにここでは挙げなかったレスポンダの推論規則を適用すれば、

$$[\text{Says}\ \texttt{B'}\ \texttt{A}\ \{\{\texttt{B}, \texttt{K}, \texttt{Na}, \texttt{Nb}\}_{\text{shrK(A)}}, \texttt{X}\}, \text{Says}\ \texttt{A}\ \texttt{B}\ \{\texttt{A}, \texttt{Na}\}] \in \texttt{traceSet}$$

が得られる。このように推論規則を繰り返し適用して得られる集合が traceSet である。これは Yahalom のプロトコルを実行したときに参加者が発生させるイベントの列の集合である。

帰納的方法では、攻撃者の行動も推論規則として記述することにより、攻撃者のいる環境でのプロトコル実行を考えることができる。攻撃者の能力に対する仮定は推論規則の定義の中で与えることができる。

プロトコルを帰納的方法の推論規則として与えておけば、これによって定義されるトレース集合 traceSet を用いて安全性要求を記述できる。具体的には、「traceSet に含まれる全てのトレース trace について安全性 P(trace) が成り立つ」というトレースに関する論理式として記述される。帰納的方法ではこれを証明することにより安全性を検証する。安全性要求の記述例は第 4.5.4 節 (図 4.10) で与える。

帰納的方法では、トレースの集合の定義とトレースの集合の性質がそれぞれプロトコル仕様と安全性要求の記述に相当しており両者の関係がわかりやすい。しかしながら、この方法はトレースというプロトコル仕様そのものとは直接関係が無い概念を利用するため、プロトコル仕様の記述が直観的でなくなり誤りを犯す可能性がある。特に SET に関していえば、プロトコル仕様書では参加者の動作はメッセージの受信、処理、作成および送信といった単位で記述されており、トレースと論理式を用いた記述との間にはギャップがある。さらに、式 (4.1) の論理式では、メッセージ M が過去の多くのイベントに依存している場合に前提部分の Says A B_i $M_i \in$ set trace や Says C_i A $L_i \in$ set trace という論理式の数が多くなり、記述が繁雑になるという問題もある。本研究では、プロトコル仕様を簡潔に記述できるプロトコル仕様記述言語を定義し、この言語で表現されたプロトコルを論理式へ変換することでこれらの問題を解決する。

4.5 提案手法の詳細と結果

4.5.1 プロトコル仕様記述言語

本節では、プロトコルを各参加者の役割に分けて記述でき、パターンマッチングを用いて簡潔な表現ができるプロトコル仕様記述言語を定義する。この言語は、値渡し CCS[40] に、変数によるパターンマッチングの機構を追加した言語である。この言語ではプロトコルを参加者の動作を表わすプロセスの集まりとして記述する。

参加者の動作を表すプロセス P の言語を定義する。

$$
\begin{array}{rcl}
P & ::= & \mathsf{End} \\
& | & \mathsf{Send}\ A\ T;\ P \\
& | & \mathsf{Recv}\ T;\ P \\
& | & \mathsf{New}\ X;\ P \\
& | & \mathsf{Let}\ X = T;\ P
\end{array}
$$

ここで、A、X、T はメッセージの項を表すメタ変数である。つまり、後述する例のように、具体的なプロセスの記述においては Isabelle/HOL の論理における項を記述する。

定義の各行の直観的意味は以下の通りである。End は停止しているプロセスを表す、Send $A\ T;\ P$ は参加者 A にメッセージ T を送信した後で P を実行するプロセスを表す。Recv $T;\ P$ はメッセージ T を受信した後で P を実行するプロセスを表す。New $X;\ P$ は新しいノンスを生成してそれを変数 X に束縛した後で P を実行するプロセスを表す。Let $X = T;\ P$ は変数 X に T を束縛した後で P を実行するプロセスを表す。なお、セミコロン ';' と End は誤解の無いときは省略する。Recv $T;\ P$ ではパターンマッチングが行なわれる。たとえばプロセス Let $X_1 = M;$ Recv $\{X_1, \{X_2\}_K\};\ P$ は変数 X_1 に M を束縛した後に $\{M, \{M'\}_K\}$ という形のメッセージのみを受けとり、変数 X_2 に M' を束縛し、プロセス P を実行する。また、この場合 $\{M'\}_K$ から M' を取り出すためには対応する復号化のための鍵 K^{-1} を知っている必要がある。この言語では Recv $\{X\}_K;\ P$ のように記述した場合にはこのプロセスは暗黙に K^{-1} を知っていると仮定する。

この言語は非常に単純であるが、プロトコルを各参加者の動作を表わすプロセスに分けて記述し、プロセスの内部で変数とパターンマッチングを用いることで記述を簡潔に行えるようになっている。たとえば、メッセージ $\{M, \{M'\}_K\}$ を受信するプロセスを Recv $\{X_1, \{X_2\}_K\};\ P$ と記述するか Recv $\{X_1, X_2\};\ P$ と記述するかによって、このプロセスが $\{M'\}_K$ を復号化して M' を読み出すか復号化せず変数 X_2 に束縛して暗号文のまま扱うかをそれぞれ書きわけることができる。

例 2　Yahalom のプロトコルにおけるイニシエータ（最初にメッセージを送信する参加者）のプロセス *Init* を図 4.4 に示す。これを図 4.3 と比較すると、トレースやトレースの集合などのプロトコルそのものとは直接関係のない変数が無く、簡潔である。

4.5.2　プロトコル仕様の論理式への変換方法

第 4.5.1 節で定義した言語で記述したプロトコル仕様を、帰納的方法の推論規則へ変換する関数 convert を定義する。convert はプロトコル仕様を構成するプロセスを引

$$
\begin{array}{l}
\mathsf{New}\ \mathtt{Na};\\
\mathsf{Send}\ \mathtt{B}\ \{\mathtt{A}, \mathtt{Na}\};\\
\mathsf{Recv}\ \{\{\mathtt{B}, \mathtt{K}, \mathtt{Na}, \mathtt{Nb}\}_{\mathsf{shrK}(\mathtt{A})}, \mathtt{X}\};\\
\mathsf{Send}\ \mathtt{B}\ \{\mathtt{X}, \{\mathtt{Nb}\}_{\mathtt{K}}\};\\
\mathsf{End}
\end{array}
$$

図 4.4　Yahalom のプロトコルのイニシエータを定義するプロセス

数の 1 つとしてとり、推論規則の集合を返す。convert は実際には初期状態に関する推論規則のみを生成し、その他の変換は convert1 を用いて行う。なお、これらの関数は Isabelle/HOL の論理における関数記号ではなく、Isabelle/HOL の推論規則を作るためのアルゴリズムである。convert および convert1 の定義を図 4.5 に示す。なお、定義の右辺に現れる A は、このプロセスを実行する参加者を表す変数であり、B' および env' として、参加者およびメッセージを表す項の列を表す新しい Isabelle/HOL の変数を選んで用いる。また、論理式の列を $[\![P_1, \ldots, P_n]\!]$ で表し、その連結を記号 '+' を用いて表す。

プロセスの定義により、プロセスは Send T、Recv T、New X、Let $X = T$ を要素とし、End によって終端される列とみなせる。特に Send T に注意すれば、e_{ij} を Recv T、New X、Let $X = T$ のいずれかとして、プロセスは一般に次の形をしていることがわかる。

$$
\begin{array}{l}
e_{11}; \cdots ; e_{1n_1}; \mathsf{Send}\ B_1\ T_1;\\
\qquad\qquad \vdots\\
e_{m1}; \cdots ; e_{mn_m}; \mathsf{Send}\ B_m\ T_m;\\
e_{(m+1)1}; \cdots ; e_{(m+1)n_{m+1}}; \mathsf{End}
\end{array}
\tag{4.2}
$$

convert1 はこの $e_{i1}; \cdots ; e_{in_i}; \mathsf{Send}\ B_i\ T_i$ という部分を順に次の形の推論規則に変換する。

$$
\begin{array}{l}
[\![\mathtt{trace} \in \mathtt{traceSet}\ ; \mathtt{last} \in \mathrm{set}\,\mathtt{trace}\ ; prem_i]\!]\\
\quad \Longrightarrow\ ([(\mathrm{Says}\ A\ B_i\ T_i, env)] + \mathtt{trace}) \in \mathtt{traceSet}
\end{array}
$$

ここで traceSet は、推論規則によって定義しようとしているトレースの集合を表わす変数である。ただし、ここでは第 4.4.4 節のトレースの定義を拡張して、トレースは $(\mathrm{Says}\ C\ D\ T, env)$ あるいは $(\mathrm{Notes}\ C\ T, env)$ という 2 つ組のリストであるとする。$\mathrm{Notes}\ C\ T$ は参加者 C の状態をトレースに記録するためのもので、参加者 C のプロセスが初期状態でどのようなパラメータを与えられているかを記録するために主に用いられ

$$
\begin{aligned}
&\mathrm{convert}(P, [X_1, \ldots, X_n]) = \\
&\quad \{\mathtt{trace} \in \mathtt{traceSet} \implies ([(\mathrm{Notes}\ \{X_1, \ldots, X_n\}, [X_1, \cdots, X_n])] + \mathtt{trace}) \in \mathtt{traceSet}\} \\
&\quad \cup\ \mathrm{convert1}(P, (\mathrm{Notes}\ \{X_1, \ldots, X_n\}, [X_1, \cdots, X_n]), [X_1, \cdots, X_n], \{\})
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{convert1}(\mathsf{End}, last, env, prems) = \{\} \\
&\mathrm{convert1}(\mathsf{New}\ X; P, last, env, prems) = \mathrm{convert1}(P, last, env + [X], prems + [\![X \notin \mathrm{used}\ \mathtt{trace}]\!]) \\
&\mathrm{convert1}(\mathsf{Let}\ X = T; P, last, env, prems) = \mathrm{convert1}(P, last, env + [X], prems + [\![X = T]\!]) \\
&\mathrm{convert1}(\mathsf{Recv}\ M; P, last, env, prems) = \\
&\quad \mathrm{convert1}(P, last, env + [X_1, ..., X_n], prems + [\![(\mathrm{Says}\ B'\ A\ M, env')\ \in \mathrm{set}\,\mathtt{trace}]\!]) \\
&\mathrm{convert1}(\mathsf{Send}\ B\ M; P, last, env, prems) = \\
&\quad \{[\![\mathtt{trace} \in \mathtt{traceSet};\ last \in \mathrm{set}\,\mathtt{trace}]\!] + prems \implies ([(\mathrm{Says}\ A\ B\ M, env)] + \mathtt{trace}) \in \mathtt{traceSet}\} \\
&\quad \cup\ \mathrm{convert1}(P, (\mathrm{Says}\ A\ B\ M, env), env, [\![]\!])
\end{aligned}
$$

図 4.5　プロトコル仕様の論理式への変換

$$
\begin{aligned}
&\mathrm{convert}(\mathsf{New}\ \mathtt{Na}; \cdots, [\mathtt{A}, \mathtt{B}]) = \\
&\quad \{\mathtt{trace} \in \mathtt{traceSet} \implies ([(\mathrm{Notes}\ \mathtt{A}\ \{\mathtt{A}, \mathtt{B}\}, [\mathtt{A}, \mathtt{B}])] + \mathtt{trace}) \in \mathtt{traceSet}\} \cup R_1
\end{aligned}
$$

$$
\begin{aligned}
R_1 &= \mathrm{convert1}(\mathsf{New}\ \mathtt{Na}; \cdots, (\mathrm{Notes}\ \{\mathtt{A}, \mathtt{B}\}, [\mathtt{A}, \mathtt{B}]), [\mathtt{A}, \mathtt{B}], [\![]\!]) \\
&= \mathrm{convert1}(\mathsf{Send}\ \mathtt{B}\ \{\mathtt{A}, \mathtt{Na}\}; \cdots, (\mathrm{Notes}\ \mathtt{A}\ \{\mathtt{A}, \mathtt{B}\}, [\mathtt{A}, \mathtt{B}]), [\mathtt{A}, \mathtt{B}, \mathtt{Na}], [\![\mathtt{Na} \notin \mathrm{used}\ \mathtt{trace}]\!]) \\
&= \{[\![\mathtt{trace} \in \mathtt{traceSet};\ (\mathrm{Notes}\ \mathtt{A}\ \{\mathtt{A}, \mathtt{B}\}, [\mathtt{A}, \mathtt{B}]) \in \mathrm{set}\,\mathtt{trace};\ \mathtt{Na} \notin \mathrm{used}\ \mathtt{trace}]\!] \\
&\qquad \implies ([(\mathrm{Says}\ \mathtt{A}\ \mathtt{B}\ \{\mathtt{A}, \mathtt{Na}\}, [\mathtt{A}, \mathtt{B}, \mathtt{Na}])] + \mathtt{trace}) \in \mathtt{traceSet}\} \cup R_2
\end{aligned}
$$

$$
\begin{aligned}
R_2 &= \mathrm{convert1}(\mathsf{Recv}\ \{\{\mathtt{B}, \mathtt{K}, \mathtt{Na}, \mathtt{Nb}\}_{\mathsf{shrK(A)}}, \mathtt{X}\}; \cdots, \\
&\qquad\qquad (\mathrm{Says}\ \mathtt{A}\ \mathtt{B}\ \{\mathtt{A}, \mathtt{Na}\}, [\mathtt{A}, \mathtt{B}, \mathtt{Na}]), [\mathtt{A}, \mathtt{B}, \mathtt{Na}], [\![]\!]) \\
&= \mathrm{convert1}(\mathsf{Send}\ \mathtt{B}\ \{X, \{N_B\}_K\}; \mathsf{End}, \\
&\qquad\qquad (\mathrm{Says}\ \mathtt{A}\ \mathtt{B}\ \{\mathtt{A}, \mathtt{Na}\}, [\mathtt{A}, \mathtt{B}, \mathtt{Na}]), [\mathtt{A}, \mathtt{B}, \mathtt{Na}, \mathtt{Nb}, \mathtt{K}, \mathtt{X}], \\
&\qquad\qquad [\![(\mathrm{Says}\ \mathtt{S}\ \mathtt{A}\ \{\{\mathtt{B}, \mathtt{K}, \mathtt{Na}, \mathtt{Nb}\}_{\mathsf{shrK(A)}}, \mathtt{X}\}, env') \in \mathtt{trace})]\!] \\
&= \{[\![\mathtt{trace} \in \mathtt{traceSet};\ (\mathrm{Says}\ \mathtt{A}\ \mathtt{B}\ \{\mathtt{A}, \mathtt{Na}\}, [\mathtt{A}, \mathtt{B}, \mathtt{Na}]) \in \mathtt{trace}; \\
&\qquad (\mathrm{Says}\ \mathtt{S}\ \mathtt{A}\ \{\{\mathtt{B}, \mathtt{K}, \mathtt{Na}, \mathtt{Nb}\}_{\mathsf{shrK(A)}}, \mathtt{X}\}) \in \mathtt{trace}]\!] \\
&\qquad \implies ([(\mathrm{Says}\ \mathtt{A}\ \mathtt{B}\ \{\mathtt{X}, \{\mathtt{Nb}\}_{\mathtt{K}}, [\mathtt{A}, \mathtt{B}, \mathtt{Na}, \mathtt{Nb}, \mathtt{K}, \mathtt{X}])] + \mathtt{trace}) \in \mathtt{traceSet}\} \cup R_3
\end{aligned}
$$

$$
\begin{aligned}
R_3 &= \mathrm{convert1}(\mathsf{End}, (\mathrm{Says}\ \mathtt{A}\ \mathtt{B}\ \{\mathtt{X}, \{\mathtt{Nb}\}_{\mathtt{K}}, [\mathtt{A}, \mathtt{B}, \mathtt{Na}, \mathtt{Nb}, \mathtt{K}, \mathtt{X}]), [\mathtt{A}, \mathtt{B}, \mathtt{Na}, \mathtt{Nb}, \mathtt{K}, \mathtt{X}], [\![]\!]) \\
&= \{\}
\end{aligned}
$$

図 4.6　Yahalom のプロトコルにおけるイニシエータの論理式への変換

る。env は項のリストで、参加者 C がイベント Says $C\ D\ T$ を発生させたときに C のプロセスが保持している変数の値が格納されていると考えてよい。env に含まれているパラメータを T_i から参照できるため、T_i が過去の複数のイベントで生成あるいは受信されたパラメータを参照している場合でも推論規則の前提部分に複数の Says イベントを加えておく必要がない。これにより第 4.4.4 節の式 (4.1) に表われる Says $A\ B_i\ M_i \in \mathrm{set}\,\mathtt{trace}$ や Says $A\ B_i\ M_i \in \mathrm{set}\,\mathtt{trace}$ の一部が省略できる。$last$ は (Notes $A\ T, env'$) あるいは (Says $A\ B'\ T', env'$) の形の 2 つ組であり、参加者 A が直前に行った動作に相当する。$prem_i$ は（';' によって結合された）論理式の列であり、式 (4.1) における $cond$ に相当する。$prem_i$ の中身は $e_{i1}, \ldots, e_{in_i}$ によって決まる。e_{ij} が Recv T、Let $X = T$、New X のときそれぞれ Says $B\ A\ T \in \mathtt{trace}$、$X = T$、$X \notin \mathrm{used}\ \mathtt{trace}$ が $prem_i$ に追加される。

例 3　Yahalom のプロトコルにおけるイニシエータのプロセス (図 4.4) を変換した例を図 4.6 に示す。プロセス *Init* の Send までの部分をそれぞれ式 (4.2) の形の推論規則に変換していることがわかる。また、推論規則の前提部分にノンスの生成 New とメッセージの受信 Recv に対応する論理式がそれぞれ追加されていることがわかる。この推論規則の集合は、Notes イベントと env の部分が冗長であることを無視すれば、図 4.3 の推論規則の集合と等価であることも容易にわかる。

4.5.3　SET の支払いプロトコルの簡略化

SET は仕様書が 400 ページ以上もある巨大なプロトコルであるため、これをそのまま検証することは膨大な時間と労力を要する。本研究では、検証すべき安全性要求に着目してプロトコルの簡略化を行うことにより、この問題に対処する。具体的には、プロトコル仕様から検証すべき安全性要求とは無関係な部分を省略し、より単純なプロトコルを作る。

本研究では Meadows らの与えた安全性要求の検証をめざしているため、プロトコル仕様に含まれるパラメータのうち、この安全性要求に出現しないものを省略する。Meadows らの安全性要求で参照されるパラメータは以下のとおりである。

1. 顧客が生成するトランザクション ID
2. 加盟店が生成するトランザクション ID
3. 顧客の名前
4. 注文内容のハッシュ値

5. 顧客の PAN
6. 顧客の $PANSecret$
7. 購入金額
8. 加盟店の名前
9. 引落とし金額
10. 承認された購入金額
11. $PReq$(購入要求) における電子署名の有無

このうち 9 および 10 は本研究で扱うメッセージに含まれないため省略する。また、本研究では署名つきの $PReq$ メッセージのみを取り扱うため、11 は省略する。SET では購入要求のメッセージに電子署名を添付するか否かは利用者が選択することができるが、本研究では顧客は購入要求のメッセージに必ず電子署名を添付し、これを受信する加盟店およびゲートウェイは必ず署名を検証すると仮定する。この仮定により電子署名を添付しない顧客についての安全性は検証できないが、11 の省略が検証結果に影響を与えることはない。

たとえば $AuthReq$(承認要求) のメッセージの簡略化は図 4.7 のように行われる。この図はメッセージの構造を木状に表したものである。省略されないパラメータには灰色の印をつけてある。

ここではハッシュ H、暗号化 E などのメッセージ操作が用いられている。SET の仕様では、ハッシュアルゴリズムやビット列への符号化まで定義されているが、本章の検証は記号モデルに基づくためメッセージはノンスや参加者名などを表す記号に暗号化などを表す記号を並べて構成される記号列である。メッセージ操作は図 4.8 の等式により定義される。

このような簡略化を行い、プロトコルの参加者である顧客、加盟店、ゲートウェイの動作を前節で述べた言語で記述した。顧客の動作を図 4.9 に示す。加盟店およびゲートウェイの動作は本章の末尾に示す。

4.5.4 秘匿性の記述と検証

前節で得られた簡略化された SET の支払いプロトコル仕様を第 4.5.2 節で提案した変換方法を用いて論理式に変換し、これを推論規則として秘匿性の論理式を証明する。

ここで検証する秘匿性は顧客のカード番号（PAN）についてのものである。PAN の秘匿性の論理式を図 4.10 に示す。この論理式は、攻撃者がネットワークを監視して得られ

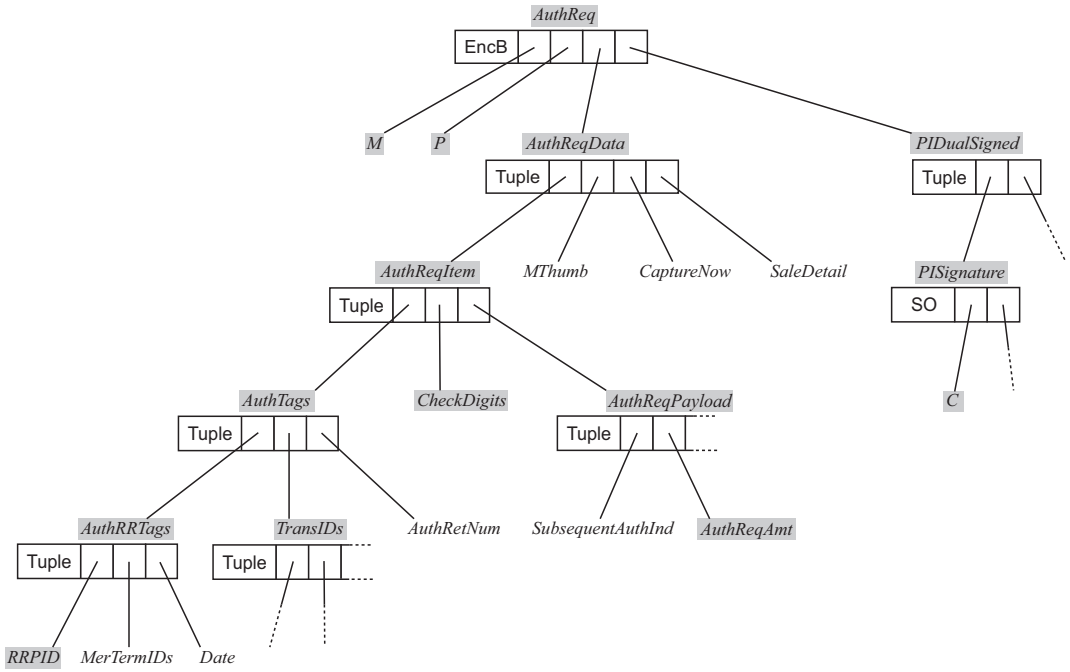

図 4.7　AuthReq(承認要求) メッセージの簡略化

リンク	$\mathsf{L}(M,N) = \{M, \mathsf{H}(N)\}$
署名	$\mathsf{SO}(A,M) = \{\mathsf{H}(M)\}_{\mathsf{priK}(\mathsf{signer}(A))}$
暗号化	$\mathsf{E}(A,M,K) = \{\{M\}_K, \{K\}_{\mathsf{pubK}(\mathsf{cryptor}A)}\}$
暗号化	$\mathsf{EX}(A,M,Mp,K) = \{\{\mathsf{L}(M,Mp)\}_K, \{K,Mp\}_{\mathsf{pubK}(\mathsf{cryptor}A)}\}$
署名つきメッセージ	$\mathsf{S}(A,M) = \{M, \mathsf{SO}(A,M)\}$
署名後暗号化	$\mathsf{Enc}(S,R,T,K) = \mathsf{E}(R, \mathsf{S}(S,T), K)$
署名後暗号化	$\mathsf{EncB}(S,R,T,B,K) = \{\mathsf{Enc}(S,R,\mathsf{L}(T,B),K), B\}$

図 4.8　メッセージ演算

るデータの集合 analz (spies `trace`) にカード番号 `PAN` が含まれるのは次の 3 つの場合のいずれかであるという意味である。

- 顧客自身が悪意を持っており ($\mathtt{C} \in \mathrm{bad}$)、攻撃者に直接 `PAN` を漏らしてしまう場合
- ゲートウェイの秘密鍵があらかじめ攻撃者に漏れており、顧客がそのゲートウェイを用いて注文をしようとする場合。なお、暗号化と署名の鍵を区別して用いるために、参加者 `P` の鍵が漏れている場合を $\mathrm{cryptor}(\mathtt{P}) \in \mathrm{bad}$ により表す。
- ゲートウェイ自身が悪意を持っており ($\mathtt{P} \in \mathrm{bad}$)、攻撃者に直接秘密を漏らしてしまう場合。

$Customer$(C, PAN, PANSecret, P, M, OD, ODSalt, PurchAmt) =

New RRPID1, LID_C, Chall_C

;; *PInitReq*

Send M {RRPID1, LID_C, Chall_C}

;; *PInitRes*

Recv {{LID_C, LID_M, XID}, RRPID1, Chall_C, Chall_M}

Let TransIDs = {LID_C, LID_M, XID}

PANData = {PAN, PANSecret}

PIHead = {TransIDs, H(OD), PurchAmt}

OIData = {TransIDs, RRPID2, Chall_C, H(OD), ODSalt}

PIData = {PIHead, PANData}

New K_PReq

;; *PReq*

Send M { { SO(C, {H(PIData), H(OIData)}),
EX(P, L(PIHead, OIData), PANData, K_PReq)},
{OIData, H(PIData)}}

;; *PRes*

Recv S(M, {TransIDs, RRPID2, Chall_C})

図 4.9　顧客の動作

これは、顧客かゲートウェイが悪意を持っていなければ加盟店が悪意を持っていても秘匿性が守られるということもできる。

証明は定理証明システム Isabelle 上で行う。前節で記述した参加者の動作を第 4.5.2 節の論理式に変換したものをルールとして与え、秘匿性の論理式を証明する。たとえば図 4.9 の顧客の動作は図 4.11 の 3 つの論理式に変換される。

秘匿性の証明自体は次のような標準的な順で行った。

1. 参加者の秘密鍵の秘匿性の証明
2. E, EX, Enc, EncB で用いられる対称鍵の秘匿性の証明
3. PAN の秘匿性の証明

以上の証明のために、Isabelle と共に配布されている帰納的方法によるプロトコル検証の Isabelle スクリプト (合計約 1600 行) を第 4.5.2 節の *env* に相当する変更を加えて

$$
\begin{aligned}
&\texttt{trace} \in \texttt{traceSet} \wedge \texttt{PAN} \in \mathrm{analz}(\mathrm{spies}\ \texttt{trace}) \rightarrow \\
&\quad \exists \texttt{C}\ \texttt{PANSecret}\ \texttt{env}. \\
&\qquad \texttt{C} \in \mathrm{bad}\ \wedge\ (\mathrm{Notes}\ \texttt{C}\ \{\texttt{PAN}, \texttt{PANSecret}\}, \texttt{env}) \in \texttt{trace} \\
&\vee \exists \texttt{P}\ \texttt{C}\ \texttt{M}\ \texttt{RRPID1}\ \texttt{LID\_C}\ \texttt{Chall\_C}\ \cdots. \\
&\qquad \mathrm{cryptor}(P) \in \mathrm{bad}\ \wedge\ (\mathrm{Says}\ \texttt{C}\ \texttt{M}\ \{\texttt{RRPID1}, \texttt{LID\_C}, \texttt{Chall\_C}\}, [\cdots]) \in \texttt{trace} \\
&\vee \exists \texttt{P}\ \texttt{C'}\ \texttt{PANSecret}\ \texttt{env}. \\
&\qquad \texttt{P} \in \mathrm{bad}\ \wedge\ (\mathrm{Notes}\ \texttt{P}\ \{\texttt{C'}, \texttt{PAN}, \texttt{PANSecret}\}, \texttt{env}) \in \texttt{trace}
\end{aligned}
$$

図 4.10　PAN の秘匿性の論理式

再利用することができた。これに加え、SET の支払いプロトコル固有のスクリプト (合計約 800 行) を新たに追加し、検証を行なった。これらの変更・追加したスクリプトは文献 [83] として公開されている。このスクリプトに記述されている証明が正しいことを Isabelle99-1 でチェックするために、CPU が PentiumIII-S 1.26GHz、メモリを 4G バイト搭載した計算機を用いて約 3 分 23 秒を要した。なお、OS は RedHat Linux 7.2 で、使用した ML の処理系は Poly/ML 3.3X である。

4.6　関連研究

関連研究として、クレジットカードを用いた決済のための暗号プロトコルについて形式検証を適用した研究について述べる。なお、本研究を開始して以降の研究状況についても述べ、さらに、インターネット上の決済だけでなく、店舗における決済についても述べる。まず店舗においてカードを提示する場合の決済について、そして、インターネットでの取引などでカードを直接的には提示しない決済について述べ、これらに形式検証を適用した研究について述べる。

4.6.1　カードを提示しての決済

店舗における支払いでは、顧客がカードを加盟店に提示する。加盟店は利用者のカードを端末に挿入し、端末を通じて決済に必要な情報を得る。加盟店はその情報を元にアクワイアラに決済の承認と代金の清算を依頼する。決済が承認されれば、クレジットカード決済網を通じてイシュアとの間で清算が行われる。イシュアは清算した代金を顧客から受

ける。

従来はカードの磁気ストライプに記録されたカード番号（Primary Account Number、PAN）を読み取る方式が主流であったが、PAN を不正に読み出すスキミングと呼ばれる不正への対策のため、IC チップを用いた方式に移行している。加盟店端末はクレジットカード上の IC チップとの通信により決済に必要な情報を読み出し、必要に応じて利用者の検証のために暗証番号（PIN コード）の入力を要求し、それが正しいものかどうか IC チップに問い合わせる。情報の読み出し方法として、IC チップ上の電極を用いる接触型のものと、NFC を用いる非接触のものがある。さらに、加盟店の端末がイシュアと通信を行いながら処理を行う方法（オンライン・トランザクション）と、事後に必要な情報をイシュアに送る方法（オフライン・トランザクション）がある。これらのプロトコルはクレジットカード決済の業界標準である EMV の仕様 [84] により規定されている。

4.6.2 カードを提示しない決済

電話やファクシミリ、そしてインターネットを用いた通信販売においてはカードを提示せず、必要な情報を通信により送受信する。これはカードを提示しない（Card Not Present、CNP）決済と呼ばれる。顧客が加盟店にカード番号（PAN）を送信する方法が最も基本的だが、不正に入手した PAN を用いて商品を購入する不正利用の可能性がある。PAN の不正な入手の方法として、店舗などでスキミングにより読み取る方法、通信販売業者が顧客から受信した PAN を不正に入手するなどが考えられる。このため、不正利用に対する対策が講じられている。ここでは SET、Visa 3-D Secure、および、EMV 3-D Secure を取り上げる。なお、PAN の不正入手への対策としての PAN のトークン化や、不正に入手された PAN の利用への対策としてのカード券面に記載されたセキュリティコードもある。

セキュリティコード

最も単純で広く利用されている対策として、カード裏面に記載されている 3 桁または 4 桁の数字であるセキュリティコードが用いられている。セキュリティコードは 1997 年ごろから大手クレジットカードブランドにより導入された。決済の際に、顧客はセキュリティコードを加盟店に送信し、加盟店はトランザクションの承認の際にこれをアクワイアラに送信し、アクワイアラはカードに紐付けられている正しいセキュリティコードであることをクレジットカード決済網を通じて確認する。セキュリティコードは磁気ストライプに記録されないため、磁気ストライプの内容がスキミングにより読み取られてもセキュリ

ティコードを知られることはない。もちろん、取引の際に加盟店がセキュリティコードを不正に保存した場合は PAN とセキュリティコードを用いて不正利用することが可能である。

SET (Secure Electronic Transaction)

抜本的な対策として、カード決済のために設計された通信プロトコルを用いる方法がある。初期の取り組みとして、VISA と MasterCard などが中心となって策定された、SET プロトコル [60, 61, 62, 85] [*2]がある。SET プロトコルでは、顧客と加盟店がそれぞれ、トランザクションにディジタル署名を行い、これらは加盟店を通じてアクワイアラが運営する支払いゲートウェイに送信される。その際、顧客は PAN を公開鍵暗号により暗号化し、アクワイアラのみが PAN を知ることができる。アクワイアラはまず、両者からの署名の検証を検証し、さらに署名されたトランザクションの内容が一致していることが確認し、その後、クレジットカード決済網を通じてトランザクションを処理する。この方法ではディジタル署名を用いることでトランザクションの認証を行える一方、顧客はディジタル署名のための鍵の発行を受け、トランザクションへの署名を行うために専用のソフトウェアを利用する必要があることが普及の妨げになったと考えられる。

Visa 3-D Secure (1.0)

より多くの利用者に受け入れられる方法として、専用のソフトウェアを必要としない方法である 3-D Secure[86] が VISA により提案された。3-D Secure では、顧客はディジタル署名を行うかわりに、イシュアが運用するアクセスコントロールサーバにアクセスして認証を受ける。web ブラウザを用いた商品の購入手続きの途中で、ブラウザは加盟店の web サイトからアクセスコントロールサーバの web サイトに転送（リダイレクト）され、トランザクションの承認のためにアクセスコントロールサーバからパスワードの入力を求められる。リダイレクトされた先が正規のアクセスコントロールサーバであることを、利用者はあらかじめアクセスコントロールサーバに登録しておいたメッセージによって確認する。また、ブラウザの機能によってサーバの証明証を検証することでも確認できる。3-D Secure では、顧客はすべての取引についてパスワードの入力が求められるため、顧客がパスワードを思い出せない場合などに、顧客が取引を諦める場合がある。これを、俗に「かご落ち」「カート放棄（cart abandonment）」などと呼ぶことがある。

[*2] SET プロトコルを策定したコンソーシアムの web ページはすでに閉鎖されている。

EMV 3-D Secure (2.0)

不正の防止と支払いの手間（checkout friction）を両立させて「かご落ち」による機会損失を減らすために、クレジットカード各社は協力して 2016 年に EMV 3-D Secure[87] を策定した。EMV 3-D Secure では、取引のリスクに応じて、顧客にパスワードなどの入力を求めない手間が少ない（frictionless）方法を選択する仕組みが規定されている。そして、顧客が使用するデバイスとして、スマートフォン、ラップトップ PC、タブレット PC などの利用も想定し、顧客の認証方法としてパスワード、SMS やスマートフォンのアプリを利用したワンタイムパスワード、バイオメトリクス認証の使用も想定している。さらに、2023 年にはコア機能仕様 [88] に、W3C が策定した、支払い時の本人認証を行うプロトコルである SPC（Secure Payment Confirmation）の利用も追加されている。VISA による 3-D Secure をバージョン 1.0、EMV 3-D Secure をバージョン 2.0 と呼ぶこと多い。クレジットカードの国際ブランド各社（VISA、MasterCard、JCB、AMEX）は 2022 年 10 月をもってバージョン 1.0 の使用を終了している。

4.6.3 カードを提示しての決済の形式検証

カードを提示しての決済の安全性の研究のうち、形式検証を用いたものを紹介する。

De Ruiter と Poll[89] は、プロトコル検査ツールの 1 つである ProVerif[49] を用いて、接触型のトランザクションの検証を行った。特に、加盟店によるカードの認証の方法として 3 つの方式、そしてオンライン及びオフラインのトランザクションについて ProVerif のスクリプトとして定式化して検証を行った。その結果、カードが PIN コードの正しさを確認できていないにも関わらず端末は PIN コードが正しく確認できた状態になる、などの中間者攻撃が可能であることを示した。なお、この攻撃自体は Murdoch[90] らによってすでに発見されていたものである。

Basin ら [91] は、2000 ページにもわたる EMV 仕様から、決済の形式的モデルを抽出し、プロトコル検証ツールの 1 つである Tamarin[50] を用いてその安全性の検証を試みた。特に、オフラインまたはオンラインでのトランザクション、および 5 種類の利用者確認方法の組み合わせを考慮して検証を行った。その結果、Visa ブランドのカードを用いた非接触の決済において、利用者を確認するための暗証番号（PIN コード）の入力をバイパスすることが可能であることなどの欠陥（中間者攻撃による攻撃）を発見した。さらにこれに続く研究で、Visa 以外のブランドのカードを Visa ブランドのカードと誤認させる攻撃を発見し、他のブランドのカードでも同様な攻撃が可能なことを示した [92]。さら

に、加盟店端末がオフラインでのトランザクションの認証に失敗した際の処理の不備をついて PIN コードによる顧客の意思確認をバイパスする方法も発見し、これを修復したプロトコルの安全性を Tamarin 上で検証している [93]。

4.6.4 カードを提示しない決済の形式検証

SET の安全性検証

Bolignano[63] は SET を大幅に簡略化して得られる決済プロトコルを定式化し、その安全性を検証した。PAN などの秘匿性の他、支払いゲートウェイがトランザクションの承認要求を受け付けたときは必ず対応する加盟店からの承認要求が存在する、などの性質を検証した。プロトコルおよび安全性の定式化と検証は、汎用の定理証明ツール Coq[25] 上で行った。

Lu と Smolka[64] は SET の支払いプロトコルをプロセス代数 CSP[23] を用いて記述し、加盟店による二重計上や金額の改ざん、顧客が注文金額よりも低い金額で決済を試みるなどの特定の攻撃について、それが成功しないことを検証した。検証には分散システムの検証ツール FDR[65] を用いて、並列に動作する顧客の数を 2 人に限るなどの上限を設けて検証を行なった。

本章の研究では SET の支払いプロトコルを、本質的に支払いの承認に関連する部分を残して簡略化し、そのうち PAN の秘匿性を検証した。プロトコルの定式化と検証には Paulson[80] の手法を用い、汎用の定理証明ツール Isabelle/HOL[27] 上で行った。

本章の研究と平行して、Bella らははまず、SET の支払いプロトコルを実行するための準備にあたる、顧客登録のプロトコルの安全性を検証した [67]。さらに、支払いプロトコルの安全性の検証も行い [68, 69]、支払い情報の秘匿性だけでなく、加盟店が決済承認のメッセージを受け取ったとき、確かにその承認を行った加盟店が存在することなど、決済の承認に関するいくつかの性質も検証した。

Visa 3-D Secure (1.0) の安全性検証

Pasupathinathan らは、Visa 3-D セキュアおよび MasterCard による類似のプロトコルの安全性を検証した。彼らは顧客のパスワードなどの秘匿性、および、正常終了したトランザクションにおいて送受信された支払い情報について利用者とアクセスコントロールサーバの間で合意が成立しているという性質などを検証した。その結果、悪意を持った加盟店が、利用者を正規のアクセスコントロールサーバにリダイレクトする代わりに自らが制御するサーバにリダイレクトし、そこで入力されたパスワードを盗む攻撃が可能である

ことを示した。悪意を持った加盟店は、利用者が正規のアクセスコントロールサーバに登録したメッセージをあらかじめ読み出すことができ、これを自らのサーバで表示する。利用者がサーバの証明書を確認できない場合は、正規のアクセスコントロールサーバと区別できない。この攻撃は、利用者のモバイル端末などを利用しており、ブラウザがサーバ証明書を正しく確認できない場合にこの攻撃が成功する。彼らは分散システム検証のためのツールである FDR2[65] および、これを用いて作成した暗号プロトコルの解析ツール Casper[94] を用いて検証を行った。

EMV 3-D Secure (2.0) の安全性検証

渡辺と米山 [95] は、EMV 3-D Secure においてパスワードの入力を要求するチャレンジフローの安全性の検証を ProVerif を用いて行なった。彼らはパスワードなどのチャレンジデータの秘匿性や、顧客の認証性、つまりアクセスコントロールサーバがチャレンジデータを受け取って顧客を認証したときはそのチャレンジデータは確かにその顧客が送信したという性質を検証した。彼らはチャレンジフローのうち、web ブラウザを用いるものとスマートフォンなどのアプリを用いるものを対象に検証を行なった。彼らの検証では、顧客は常に正規のアクセスコントロールサーバに接続すると仮定しており、Visa 3-D Secure に対して Pasupathinathan らの指摘した攻撃が可能か否かは検証していない。

4.7 まとめと今後の課題

本研究では、SET の安全性を、過度な簡略化を避けて検証を行った。このために、プロトコル仕様を簡潔に記述できる言語を定義し、この言語で記述されたプロトコル仕様を Paulson の帰納的方法で用いられる論理式に変換する方法を定義した。これにより大きなプロトコルを簡潔に表現でき、また、安全性の検証には帰納的方法の証明スクリプトの多くを再利用できるため検証を効率よく進めることができる。

さらに、この言語と変換方式を SET の支払いプロトコルに適用し、カード番号（PAN）の秘匿性を検証することができた。その際、現実的な時間内での検証を可能にするため、検証すべき性質とは関係ないと考えられる部分をプロトコル仕様から省略し、適度に簡略化されたプロトコルを得ることができた。

本研究には次のような課題が残されている。

- 検証、すなわち安全性の論理式の証明は、プロトコル仕様を変換した論理式を用いて行うため、検証を行う者は変換後の論理式を理解する必要が生じてしまう。検証

をより見通しよく行うために、プロトコル記述言語のレベルでも安全性に関する推論を可能とし、帰納的方法のレベルでの推論との対応関係をつけるという方法が考えられる。

- SET の支払いプロトコルの簡略化を、検証対象の安全性要求からは参照されない部分を省略することにより行なった。しかし、この簡略化が本当に安全性に影響を与えないことは明らかではない。オリジナルのプロトコルの安全性を厳密に示すためには、安全性を保存する簡略化の理論が必要となる。

また、クレジットカード決済の安全性を保護する技術は本研究以降、大きく発展してきている。また、そこで暗号プロトコルへの形式検証の適用についても、関連研究の節で概観した。これを踏まえた今後の課題として次が考えられる。

- 店舗における決済において、クレジットカードと加盟店端末の通信に非接触型の通信も用いられるようになり、クレジットカードの代わりにスマートフォンが利用可能になっている。また、カードを提示しないインターネットなどを通じた支払いにおいても、web ブラウザに加えスマートフォンアプリが用いられるようになり、プロトコルの仕組みに差異が生じている。さらに、同じ EMV 仕様であっても、国際ブランドによって利用の仕方に差異がある。つまり、利用の形態によって少しずつ異なるプロトコルが使われることになるため、これらの組み合わせのそれぞれについて安全性を検証する必要が生じてきた。このような多様性を形式検証の枠組みでうまく吸収する工夫が求められている。
- web ブラウザが主要な購入経路になっており、Visa および EMV の 3-D Secure のようにブラウザの転送（リダイレクト）の機能や、サーバ証明書の検証の機能が用いられている。さらに、利用者があらかじめアクセスコントロールサーバに登録したメッセージを確認するなど、利用者もプロトコルの一部を構成している。このような、ブラウザの機能を前提としたプロトコルを検証するためには、ブラウザの機能のモデル化が必要になる。このような方向性の研究として [96] らの研究がある。また、ブラウザとサーバとの通信の一部はブラウザが表示している HTML ファイルに埋め込まれたスクリプトによって実行されるため、サーバが悪意を持っている場合にその実行に影響を与えることで、顧客に誤認を与えることもできるため、顧客への情報提示の適切さも形式検証のスコープに入ってくる可能性がある。
- 本章では触れなかったが、クレジットカードを利用する決済では、カード番号の情報を隠すためにカード番号をトークンと呼ばれるデータで置き換える仕組みがある。また、クレジットカードと類似に仕組みを用いる決済方法としてバーコード

や二次元コードを用いる決済もある。これらに形式検証を適用することも課題となる。

- 本章では安全性を中心に述べたが、コンプライアンスも重要な観点である。クレジットカードを利用した決済にはその仕組みに法的規制や業界団体におけるガイドラインが存在し、これらを遵守していることを確認するために、形式検証が利用できる可能性がある。具体的には、欧州の決済サービス司令（PSD2）[97] では遠隔からの決済時に強力な利用者認証を行うことが義務付けられており、この基準を満たすべく EMV 3-D Secure が採用されている。日本国内においても、クレジットカードのセキュリティ対策の強化に向けて、経済産業省が方向性を打ち出しており [98]、クレジットカード事業の業界団体である日本クレジット協会も「クレジットカード・セキュリティガイドライン」[99] を公表し、EMV 3-D Secure の導入などを求めている。

$$
\begin{aligned}
&\mathtt{trace} \in \mathtt{traceSet} \\
&\ \Longrightarrow\ (\mathrm{Notes}\ C\ \{C, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}\}, \\
&\qquad [\mathtt{C}, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}]) \in \mathrm{set}\ \mathtt{trace}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathtt{trace} \in \mathtt{traceSet}; \\
&\ (\mathrm{Notes}\ \mathtt{C}\ \{\mathtt{C}, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}\}, \\
&\quad [\mathtt{C}, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}]) \in \mathrm{set}\ \mathtt{trace}; \\
&\ \mathtt{RRPID1} \notin \mathrm{used}\ \mathtt{trace};\ \mathtt{LID\_C} \notin \mathrm{used}\ \mathtt{trace};\ \mathtt{Chall\_C} \notin \mathrm{used}\ \mathtt{trace}[\!] \\
&\ \Longrightarrow\ [(\mathrm{Says}\ C\ M\ \{\mathtt{RRPID1}, \mathtt{LID\_C}, \mathtt{Chall\_C}\}, \\
&\qquad [\mathtt{C}, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}, \\
&\qquad\ \mathtt{RRPID1}, \mathtt{LID\_C}, \mathtt{Chall\_C}])] + \mathtt{trace} \in \mathtt{traceSet}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\mathtt{trace} \in \mathtt{traceSet}; \\
&\ (\mathrm{Says}\ \mathtt{C}\ \mathtt{M}\ \{\mathtt{RRPID1}, \mathtt{LID\_C}, \mathtt{Chall\_C}\}, \\
&\ [\mathtt{C}, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}, \\
&\ \ \mathtt{RRPID1}, \mathtt{LID\_C}, \mathtt{Chall\_C}]) \in \mathrm{set}\ \mathtt{trace}; \\
&\ (\mathrm{Says}\ \mathtt{M'}\ \mathtt{C}\ \{\{\mathtt{LID\_C}, \mathtt{LID\_M}, \mathtt{XID}\}, \mathtt{RRPID1}, \mathtt{Chall\_C}, \mathtt{Chall\_M}\}, \mathtt{env'}) \in \mathrm{set}\ \mathtt{trace}; \\
&\ \mathtt{TransIDs} = \{\mathtt{LID\_C}, \mathtt{LID\_M}, \mathtt{XID}\}; \\
&\ \mathtt{PANData} = \{\mathtt{PAN}, \mathtt{PANSecret}\}; \\
&\ \mathtt{PIHead} = \{\mathtt{TransIDs}, \mathsf{H}(OD), \mathtt{PurchAmt}\}; \\
&\ \mathtt{OIData} = \{\mathtt{TransIDs}, \mathtt{RRPID2}, \mathtt{Chall\_C}, \mathsf{H}(OD), \mathtt{ODSalt}\}; \\
&\ \mathtt{PIData} = \{\mathtt{PIHead}, \mathtt{PANData}\}; \\
&\ \mathtt{K\_PReq} \notin \mathrm{used}\ \mathtt{trace}[\!] \\
&\ \Longrightarrow\ [(\mathrm{Says}\ C\ M\ \{\{\mathsf{SO}(C, \{\mathsf{H}(\mathtt{PIData}), \mathsf{H}(\mathtt{OIData})\}), \\
&\qquad\qquad \mathsf{EX}(\mathtt{P}, \mathsf{L}(\mathtt{PIHead}, \mathtt{OIData}), \mathtt{PANData}, \mathtt{K\_PReq})\}, \\
&\qquad\qquad \{\mathtt{OIData}, \mathsf{H}(\mathtt{PIData})\}\}, \\
&\qquad [\mathtt{C}, \mathtt{PAN}, \mathtt{PANSecret}, \mathtt{P}, \mathtt{M}, \mathtt{OD}, \mathtt{ODSalt}, \mathtt{PurchAmt}, \\
&\qquad\ \mathtt{RRPID1}, \mathtt{LID\_C}, \mathtt{Chall\_C}, \mathtt{LID\_M}, \mathtt{XID}, \mathtt{Chall\_M}, \\
&\qquad\ \mathtt{TransIDs}, \mathtt{PANData}, \mathtt{PIHead}, \mathtt{OIData}, \mathtt{PIData}, \mathtt{K\_PReq}])] + \mathtt{trace} \in \mathtt{traceSet}
\end{aligned}
$$

図 4.11　顧客の動作を定義する推論規則

$Merchant$(M, OD, ODSalt, AuthReqAmt, C, P) =

;; *PInitReq*

Recv {RRPID1, LID_C, Chall_C}

New LID_M, XID, Chall_M

Let TransIDs = {LID_C, LID_M, XID}

;; *PInitRes*

Send C S(M, {TransIDs, RRPID1, Chall_C, Chall_M}

Let OIData = {TransIDs, RRPID2, Chall_C, H(OD), ODSalt}

;; *PReq*

Recv { {SO(C, {HPIData, H(OIData)}), PIBody},
{OIData, HPIData}}

New RRPID3, K1

;; *AuthReq*

Send P EncB(M, P, {RRPID3, TransIDs, AuthReqAmt},
{SO(C, {HPIData, H(OIData)}), PIBody}, K1)

;; *AuthRes*

Recv Enc(P, M, {RRPID3, TransIDs, AuthReqAmt}, K2)

;; *PRes*

Send C S(M, {TransIDs, RRPID2, Chall_C})

図 4.12　加盟店の動作

```
Gateway(P, C, M, PAN, PANSecret) =
    ;; AuthReq
    Recv EncB(M, P, {RRPID3, TransIDs, AuthAmt},
                { SO(C, { H({{TransIDs, HOD, AuthAmt}, PANData}),
                          HOIData}),
                  EX(P, {{TransIDs, HOD, AuthAmt}, HOIData},
                     PANData, K_PReq)},
                K_PReq)
    Let TransIDs = {LID_C, LID_M, XID},
        PANData = {PAN, PANSecret}
    New K2
    ;; AuthRes
    Send M Enc(P, M, {RRPID3, TransIDs, AuthAmt}, K2)
```

図 4.13　ゲートウェイの動作

第 5 章

QUIC プロトコルの QACCE 安全性の解析と修正

5.1 研究の背景と動機

今日、通信の当事者が用いる端末は、パーソナルコンピュータやスマートフォン、web サーバなど多岐にわたり、また、通信経路も有線 LAN や無線 LAN、ワイヤレス通信事業者が提供するモバイルネットワーク、インターネットなど様々である。このような通信において、送信されたデータが受信されたことを確認し、経路上でデータが消失や遅延した際の再送やデータの並べ替えなどの機能を持つトランスポート・プロトコルが用いられ、web ページのデータの転送などはトランスポート・プロトコルの上のプロトコルを用いて行われる。特に、なりすましや情報漏えいを避けるための認証や暗号化などの機能を備えたトランスポート・プロトコルとして TLS[100] が広く用いられている。

TLS を用いて通信を開始する際に、使用可能な暗号方式の情報を交換し、双方向あるいは片方向の認証を行い、暗号通信に用いる鍵の情報を交換する、セキュリティ・ハンドシェイクが実行される。このとき、セキュリティ・ハンドシェイクが完了するまで実際にデータを送受信できない。また、TLS は同じくトランスポートプロトコルである TCP プロトコルの上で動作するため、TLS のセキュリティ・ハンドシェイクの前に、TCP プロトコルの通信を可能にするためのハンドシェイクも必要となる。このため、TLS を用いて実際にデータを送受信する前にハンドシェイクのための遅延（latency）があることが課題となっており、低遅延性とセキュリティを両立可能な新たなトランスポート・プロトコルが研究されている [101, 102, 103, 104] 。そのようなプロトコルの 1 つとして QUIC (Quick UDP Internet Connection)[104] が提案された。

QUIC は Google 社によって開発され、同社の web ブラウザである Chrome に 2013 年に実装された。TLS が TCP 上で動作するのに対し、QUIC はパケットの再送などの機能が無い UDP 上で動作し、遅延を小さくするために独自の転送制御方式を採用している。さらに、QUIC ではセキュリティ・ハンドシェイクにかかる通信回数が TLS よりも少ない。QUIC は Google 社の提供するサービスのためのサーバで広く実装されており、さらに、Chrome は世界で最も広く利用されている web ブラウザの一つである。このため、QUIC が提供するセキュリティを明確にすることは実用上重要である。しかしながら、Google 社の文書では、この点について形式的に厳密な議論は、本研究を開始した 2016 年の時点では少なくとも公開されている範囲では見つかっていない。なお、5.6.1 に述べる通り、QUIC は IETF において標準化されている。本研究では、Google によって提案されたバージョンの QUIC を対象とする。

5.2 先行研究と課題

QUIC に対するセキュリティ要求を形式的に定義し、これを適当な仮定の下で証明した研究はいくつか存在する。Fischlin と Günther [105] は Bellare と Rogaway による鍵交換プロトコルのモデル化 [106] をもとに、複数のステージからなる鍵交換プロトコルの安全性を定義し、QUIC のうち鍵交換を行う部分がこれを満たすことを証明した。彼らは鍵交換を行う部分を対象とし、QUIC 全体の安全性は対象としなかった。Lychev ら [107] は、QUIC や TLS1.3 のドラフトを含む、鍵交換の機能を持つトランスポートプロトコルを抽象化したプロトコルとして QC（Quick Connections）プロトコルを考え、その安全性のモデルとして、QACCE（Quick Authentication and Confidential Channel Establishment）モデルを提案した。本章では、QACCE モデルで定義される安全性を QACCE 安全性と呼ぶ。QACCE モデルは QUIC や TLS1.3 などのプロトコルの、鍵交換の部分を含む全体をモデル化しており、そこで定式化されたセキュリティ要件として、サーバへのなりすまし、通信路の毀損（corruption）、IP スプーフィングという 3 種類の攻撃に対する耐性を定義している。 彼らはさらに、QUIC がこのモデルで定義される安全性を満たすことを証明した。

先行研究において未解決の課題として次が挙げられる。

- Lychev らの定義する安全性は、後述するようにオラクルへのアクセスについての条件などを用いたやや複雑なモデルとなっており、このモデルにおける安全性の証明が正しいことを確認することは容易ではない。このため、形式検証によっても安

全性の検証ができることが望ましいが、そのような研究は行われていなかった。

- Lychev らの QC モデルおよび QACCE 安全性は、QUIC のみならず TLS1.3 を含むプロトコルのモデルおよび安全性であるため、このモデルおよび安全性を形式検証の枠組みで記述し直すことができれば、TLS1.3 の検証および QUIC の安全性との比較を形式検証の枠組みで行うことができる。しかし、そのような研究は行われていなかった。

5.3 本研究の貢献

本研究では、QUIC プロトコルの安全性（QACCE 安全性）を形式検証により自動的に検証した。本研究の貢献は次のとおりである。

- QACCE 安全性について、暗号プロトコル検証ツール ProVerif の安全性記述言語を用いて、形式検証が可能な形での定義を与えた。QACCE 安全性は TLS1.3 プロトコルにも定義できる安全性であるため、本研究で与えた定義を用いて今後さらに、TLS1.3 の安全性の検証にも用いることができる。また、QUIC と TLS1.3 の安全性を形式検証を用いて詳細に比較することも可能になる。
- Lychev らが QC モデルの枠組みで定式化した QUIC プロトコルについて、ProVerif のプロトコル記述言語での記述を与えた。これと前述の安全性定義をもとに QUIC プロトコルの検証を行った。
- 本研究の検証の結果、QUIC プロトコルが QACCE 安全性を満たさないという結果が得られた。これは、Lychev らの定式化および証明に不備があるということを意味する。検証結果について詳細に検討したところ、QACCE 安全性には不必要に強い条件が含まれている、つまり、現実的に必要と考えられる安全性要件よりも強い安全性を要求してしまっていることが明らかになった。また、検証の結果発見された攻撃は、QUIC の現実的な安全性には影響がないこともわかった。
- この結果を受けて、QACCE 安全性の条件を弱めて、より適切な定義に修正を行った。この定義に基づき再度検証を行ったところ、QUIC が安全であるとの結果を得た。

検証には ProVerif ツール [49, 108, 109, 110] を利用した。なお、QUIC の仕様には状態の更新と時刻の同期が含まれるが、ProVerif ではこれらを扱うことができないため、これらを省略する形で QUIC プロトコルを記述した。これに合わせて安全性の定義を行っ

たため、QACCE 安全性よりも弱い安全性定義となっている。この点を解決することは今後の課題である。

5.4 準備

5.4.1 QC プロトコルの定義

本節では、QC（Quick Connections）プロトコルを [107] を引用して定義する。QC プロトコルは特定のプロトコルではなく、以下に述べるような要素から構成されるプロトコルの総称であり、QUIC などのプロトコルを抽象的した雛形のようなものである。QC プロトコルは、これを実現する具体的なプロトコルの安全性を一般的な形で定義するために用いられる。

QC プロトコルはクライアントとサーバの間のプロトコルである。サーバは電子署名のための公開鍵と秘密鍵のペアを保持する。クライアントとサーバはまず、初期セッション鍵を共有し、次に最終セッション鍵を共有する。これらはそれぞれ、後者を共有する前および後のデータの暗号化に用いられる。

QC プロトコルにはセキュリティパラメータ λ が関連づけられ、次の要素を含む。

- ディジタル署名に用いるサーバ鍵の生成プロトコル Kg: 入力 λ に対し、公開鍵と秘密鍵を返す。
- 認証付き暗号（AEAD）スキーム $(\mathcal{E}, \mathcal{D})$: 鍵空間 $\{0,1\}^\lambda$、ヘッダ空間 $\{0,1\}^*$、メッセージ空間 $\{0,1\}^*$ を持つ。
- 初期ベクトル（IV）抽出関数 get_iv: 鍵とヘッダを入力すると、IV を返す。IV は AEAD により各メッセージを暗号化および復号する際に用いられる。
- サーバ状態生成関数 scfg_gen: サーバは大域的な状態を持ち、その一部 `scfg` を更新するために用いられる。大域的な状態は最初は空文字列 ε で初期化されている。

プロトコルの実行には、全ての参加者に共通な時間の概念が関連づけられており、時間は離散的な時間 $\tau_1, \tau_2, \ldots$ に区切られている。サーバの公開鍵と署名鍵は鍵生成関数により次のように生成される $(pk, sk) \leftarrow_R \mathsf{Kg}(\lambda)$。各参加者には、サーバの公開鍵 pk とメッセージのリスト $M^{send} = M_1, \ldots, M_m$ が入力として与えられる。ここで、 $m \in \mathbb{N}$ であり、各 i について $M_i \in \{0,1\}^*$ である。サーバにはそれに加えて秘密鍵が入力として与えられる。一般にはサーバが送信するメッセージとクライアントが送信するメッセージは相互に依存するが、ここでは議論を簡単にするためにそのようなことは考えず、単にメッ

セージのリストが与えられているとする。また、全ての参加者は大域的な状態を持つ。

QC プロトコルは 4 つのフェーズから構成される。参加者が送信するメッセージはこれらのフェーズのいずれかに属するが、第 2 と第 3 のステージには重なりがあってもよい。

第 1 ステージ：初期鍵の共有 このステージの最後に、参加者は初期鍵 $ik = (ik_c, ik_s, iaux)$ を決める。ここで $iaux \in \{0,1\}^*$ は暗号化および復号に使用できる任意の付加情報であり、初期状態では ε となっている。

第 2 ステージ：初期データの送受信 このステージでは入力されたメッセージが、AEAD および初期鍵により暗号化されて送受信される。サーバは ik_s により暗号化を行い、ik_c により復号する。その一方、クライアントは ik_c により暗号化を行い、ik_s により復号する。このステージの最後に、参加者はそれぞれこのステージで受信したメッセージのリスト $M^{iget} = M_1, \ldots, M_{m'}$ を出力する。ここで、$m' \in \mathbb{N}$ であり、各 $M_i \in \{0,1\}^*$ である。M^{iget} は空 ε であってもよい。

第 3 ステージ：（最終）鍵の共有 このステージの最後に、参加者は（最終）セッション鍵 $k = (k_c, k_s, aux)$ を決める。ここで $aux \in \{0,1\}^*$ は暗号化および復号に使用できる任意の付加情報であり、初期状態では ε となっている。

第 4 ステージ：データの送受信 このステージでは入力されたメッセージが、AEAD および最終鍵により暗号化されて送受信される。サーバは k_s により暗号化を行い、k_c により復号する。その一方、クライアントは k_c により暗号化を行い、k_s により復号する。このステージの最後に、参加者はそれぞれこのステージで受信したメッセージのリスト $M^{get} = M_1, \ldots, M_{m''}$ を出力する。ここで、$m'' \in \mathbb{N}$ であり、各 $M_i \in \{0,1\}^*$ である。M^{iget} は空 ε であってもよい。

パケットを受信した参加者が $\perp$ を出力したとき、その参加者はパケットを拒否したといい、そうでない場合は受理したという。ある QC プロトコルにおいてクライアントが第 1 ステージで鍵 ik を決めるとき、第 1 ステージで送受信する全てのパケットがあるサーバとの接続に帰属するならば、そのクライアントはそのサーバとの鍵 ik を決めたという。逆にサーバが鍵を決める場合も同様に定義する。また、第 3 ステージで決める鍵 k についても同様に定義する。ある参加者が第 1 ステージにおいて、他のある参加者との鍵を決めたとき、これらの参加者を互いの通信相手と呼ぶ。このプロトコルが正しいというためには、ある参加者の入力 M^{send} が他方の M^{iget}, M^{get} と一致する必要がある。言い換えれば、このプロトコルが正しいとは、参加者が送受信しようと思っているデータを想定した相手と送受信でき、その順序も保存されるということである。

5.4.2 QACCE モデル

QC プロトコルのセキュリティモデルとして、QACCE（Quick ACCE）モデルを [107] を引用して定義する。QACCE モデルは、TLS のセキュリティモデルである ACCE（Authenticated and Confidential Channel Establishment）モデルを、鍵共有とデータ転送のそれぞれが 2 つのステージを持つ QC プロトコルにあわせて拡張したモデルである。

セキュリティパラメータ λ と QC プロトコルを 1 つ決める。攻撃者 A に関連づけられた試行 $\mathbf{Exp}_{\Pi}^{\mathsf{QACCE}}(A)$ を以下のように定義する。

まず、クライアントの集合 $C = \{C_1, \ldots, C_\ell\}$ およびサーバの集合 $S = \{S_1, \ldots, S_\ell\}$ を考える。ここで $\ell \in \mathbb{N}$ はサーバおよびクライアントの最大の人数である。試行では、最初にサーバの鍵ペアと初期状態を $(pk_i, sk_i) \leftarrow \mathsf{Kg}(\lambda)$ および $\mathtt{scfg}_i^t \leftarrow \mathrm{scfg\_gen}(sk_i, \tau_i, \lambda)$ により、各時間 t および添字 i について生成する。同じ参加者が複数回プロトコルを直列あるは並列に実行する場合を考慮するため、参加者 $P_i \in C \cup S$ に対して状態を持つオラクル $\pi_{p,i}^1, \ldots \pi_{p,i}^d$ を関連付ける。ここで、$d \in \mathbb{N}$ であり、$p \in \{c, s\}$ はクライアントおよびサーバのどちらを実行しているかを示す。サーバのオラクルは各時間の最初に $\mathtt{scfg}_i^t$ を取得する。参加者は、攻撃者も含め、現在の時間を知っているものとする。さらに、サーバのオラクルは自身がどのステージを実行しているか知っているものとする。初期鍵 ik を共有したサーバおよびクライアントのオラクルは互いの通信相手であるという。

攻撃者 A は全てのサーバの公開鍵 $pk_1, \ldots, pk_\ell$ を与えられており、全ての参加者の全てのオラクルに以下で定義するクエリを送って働きかけることができる。以下の定義では、括弧の中は攻撃者が与えた値であるとする。ただし、太字の箇所は固定されており変更できない。また、括弧内にオラクル $\pi_{p,i}^q$ が現れる場合は、攻撃者は p, i, q を決められるとする。

- $\mathsf{connect}(\pi_{\mathbf{c},i}^q, \pi_{\mathbf{s},j}^r) \quad (i, j \in [\ell], q, r \in [d])$
  このクエリは、攻撃者がクライアントに対し、特定のサーバに対し、この時間における最初の通信を開始させるために用いられる。このクエリにより、$\pi_{c,i}^q$ は、プロトコルに従って $\pi_{s,j}^r$ に通信開始を要求するための最初のパケットを出力する。ただし、受信者である $\pi_{s,j}^r$ に配送されず、A にのみ与えられる。
- $\mathsf{resume}(\pi_{\mathbf{c},i}^q, \pi_{\mathbf{s},j}^r) \quad (i, j \in [\ell], q, r \in [d])$
  QC プロトコルのうち、0-RTT のセッション再開が可能なプロトコルのためのク

エリである。0-RTT のセッション再開は、一度セッションを確立して初期鍵 ik を共有した後で実行することができる。このため、オラクル $\pi^q_{c,i}$ において ik が未定の場合はこのクエリは $\perp$ を返す。そうでない場合は、$\pi^q_{c,i}$ は 0-RTT の接続のリクエストを開始するために $\pi^r_{s,j}$ に送信するパケットを出力する。出力は攻撃者 A に与えられ、攻撃者はそれをオラクルに送信、改変、および廃棄することができる。

- $\mathsf{send}(\pi^r_{p,j}, m)$ $(p \in \{c,s\},\, j \in [\ell],\, r \in [d],\, m \in \{0,1\}^*)$
  このクエリは攻撃者がパケットをオラクルに送信するために用いられる。このクエリにより、パケット m がオラクル $\pi^r_{p,j}$ に送信される。パケットに含まれるヘッダの情報は攻撃者が作る必要があり、$\pi^r_{p,j}$ はこれをチェックすることができる。$\pi^r_{p,j}$ がデータ送信のステージにある場合はこのクエリは $\perp$ を出力し、そうでない場合は $\pi^r_{p,j}$ からのプロトコルに従った返信を出力する。攻撃者は出力を改変、廃棄、あるいは他のオラクルに送信することができる。
- $\mathsf{reveal}(\pi^q_{p,i})$ $(p \in \{c,s\},\, i \in [\ell],\, q \in [d])$
  このクエリは攻撃者がオラクルの初期鍵を知るために用いられる。このクエリはオラクル $\pi^q_{p,i}$ の初期鍵 ik を出力する。
- $\mathsf{reveal}(\pi^q_{p,i})$ $(p \in \{c,s\},\, i \in [\ell],\, q \in [d])$
  このクエリは攻撃者がオラクルの最終鍵を知るために用いられる。このクエリはオラクル $\pi^q_{p,i}$ の最終鍵 k を出力する。
- $\mathsf{corrupt}(S_i)$ $(i \in [\ell])$
  このクエリは攻撃者がサーバを買収し、その時点における長期にわたり保存される秘密を得るために用いられる。この秘密にはサーバの状態 $\mathtt{scfg}^t_i$ も含む。このクエリにより、サーバは sk_i、その時点での $\mathtt{scfg}^t_i$、およびその他の S_i の状態の情報を得る。
- $\mathsf{encrypt}(\pi^r_{p,j}, m_0, m_1, H, \mathsf{init})$ $(p \in \{c,s\},\, j \in [\ell],\, r \in [d],\, m_0, m_1, H \in \{0,1\}^*,\, \mathsf{init} \in \{0,1\})$
  このクエリはこれまでのクエリと異なり、初期あるいは最後のデータ送受信のステージを扱う。init によりどちらのステージに対応するかが決まる。暗号スキームの選択平文攻撃への安全性と同様に、攻撃者が与えた 2 つのメッセージからランダムに選び、それを暗号化したメッセージを出力する。また、AEAD の安全性と同様にヘッダー H は攻撃者が決めることができる。これは、QUIC の場合は H には攻撃者が IP アドレスとポート番号、そしてシーケンス番号を与える必要があることを意味する。しかし、AEAD の安全性モデルと異なり、攻撃者は IV（初期ベクトル）を選べないとする。これは、QUIC では IV は参加者の持つ秘密に依存し

て決まり、攻撃者はこれを自由に選べないためである。このため、IV は get_iv 関数により決まる。さらに、IV は攻撃者には与えられない。攻撃者は H として、送信先の IP アドレスとポート番号が $\pi^r_{p,j}$ に対応し、送信元の IP アドレスとポート番号がこの試行中のあるオラクル $\pi^q_{p',i}$ に対応するものを与えなければならない。ここで、$p' \in \{c,s\}\backslash\{p\}$ である。このクエリは次のように動作する。まず、m_0 と m_1 の長さが異なるときは $\perp$ を返す。さらに、$init = 1$ のときは、$\pi^r_{p,j}$ が初期データの送受信のステージでない場合かまたは $\mathrm{IV} \leftarrow \mathrm{get\_iv}(ik, H)$ がすでに使用されている場合は $\perp$ を返し、そうでない場合は $(H, \mathcal{E}(ik_{p'}, IV, H, m_{b^r_{p,j}}))$ を返す。また、$init = 0$ のときは、$\pi^r_{p,j}$ が最終のデータの送受信のステージでない場合かまたは $\mathrm{IV} \leftarrow \mathrm{get\_iv}(k, H)$ がすでに使用されている場合は $\perp$ を返し、そうでない場合は $(H, \mathcal{E}(k_{p'}, IV, H, m_{b^r_{p,j}}))$ を返す。ただし、k、ik、$ik_{p'}$ は $\pi^r_{p,j}$ が持っている鍵である。

- $\mathsf{decrypt}(\pi^r_{p,j}, \mathcal{C}, H, \mathsf{init})$ $(p \in \{c,s\},\ j \in [\ell],\ r \in [d],\ \mathcal{C}, H \in \{0,1\}^*,\ \mathsf{init} \in \{0,1\})$ このクエリも初期あるいは最後のデータ送受信のステージを扱う。また、AEAD の認証についての性質を反映させられるよう定義されている。攻撃者が、これまでのクエリにより得た暗号文とは異なる新しい暗号文を得ることができれば、このクエリによりチャレンジビットを得ることができ、この試行で勝つことができる。このクエリは次のように動作する。まず、$(H, \mathcal{C})$ がそれ以前の $\mathsf{encrypt}(\pi^r_{p,j}, *, *, *, \mathsf{init})$ の形のクエリの出力である場合は $\perp$ を返す。$init = 1$ のときは、$\pi^r_{p,j}$ が初期データの送受信のステージでないかまたは $\mathcal{D}(ik_p, \mathrm{IV}, H, \mathcal{C}) = \perp$ のときは $\perp$ を返し、そうでない場合は $b^r_{p,j}$ を返す。ただし、$\mathrm{IV} \leftarrow \mathrm{get\_iv}(ik, \mathrm{H})$ である。$init = 1$ のときは、$\pi^r_{p,j}$ が最終のデータの送受信のステージでないかまたは $\mathcal{D}(k_p, \mathrm{IV}, H, \mathcal{C}) = \perp$ のときは $\perp$ を返し、そうでない場合は $b^r_{p,j}$ を返す。ただし、$\mathrm{IV} \leftarrow \mathrm{get\_iv}(k, \mathrm{H})$ である。
- $\mathsf{connprivate}(\pi^q_{c,i}, \pi^r_{s,j})$ $(i,j \in [\ell],\ q,r \in [d])$ このクエリにより、最初の接続要求のパケットが $\pi^r_{s,j}$ に送られる。$\pi^r_{s,j}$ からのプロトコルに従った応答は $\pi^q_{c,i}$ に送られ、攻撃者には見えない。さらに、その応答に対する $\pi^q_{c,i}$ からの応答も攻撃者には見えない。このクエリは IP スプーフィング（IP アドレスの偽装）に対応する。

攻撃者はクエリを行った後、4 つ組 (p,i,q,b) を出力する。ここで、 $p \in \{c,s\}$ である。

次に、一致する通信（matching conversation）を定義する。$p \in \{c,s\}$、$p' \in \{c,s\}\backslash\{p\}$、$i,j \in [\ell]$、および $q,r \in [d]$ について、オラクル $\pi^q_{p,i}$ がステージ 1 およびステージ 3 において鍵を共有するために送受信したメッセージを時系列で並べたものを $R^q_{p,i}$ と書き、こ

れを $pi^q_{p,i}$ のメッセージ記録と呼ぶ。2 つのメッセージ記録 $R^q_{p,i}$ および $R^r_{p',j}$ について、$R^q_{p,i}$ が $R^r_{p',j}$ のある先頭部分と一致するとき、$R^q_{p,i}$ が $R^r_{p',j}$ のプレフィックスであるという。このとき、$R^r_{p',j}$ が $R^q_{p,i}$ のプレフィックスであり、$\pi^q_{p,i}$ が最後のメッセージを送信したか、あるいは、$R^q_{p,i}$ が $R^r_{p',j}$ のプレフィックスであり、$\pi^r_{p',j}$ が最後のメッセージを送信したとき、$\pi^q_{p,i}$ が $\pi^r_{p,j}$ と一致する通信を持つという。

次に、攻撃者 A による試行における、攻撃の成功を測る尺度を次のように定義する。

- サーバへのなりすましの尺度（server impersonation advantage） $\mathbf{Adv}^{\mathsf{s-imp}}_{\Pi}(A)$ は、あるオラクル $\pi^q_{c,i}$ が存在して最終鍵 k を決めており、次の条件を満たすサーバーのオラクル $\pi^r_{s,j}$ が存在しない確率である。その条件は、$\pi^q_{c,i}$ が $\pi^r_{s,j}$ と一致する通信を持たず、$\pi^q_{c,i}$ が $\pi^r_{s,j}$ と共有した初期鍵がもしあればそれが reveal クエリにより取得されておらず、S_j が $\mathsf{corrupt}$ クエリにより買収されていないことである。この定義は、攻撃者がある正規のサーバになりすまし、クライアントがそのサーバと鍵を共有したと信じており、その鍵は攻撃者と共有されたかもしれないという攻撃を扱う。
- 通信路の毀損の尺度（channel-corruption advantage） $\mathbf{Adv}^{\mathsf{ch-corr}}_{\Pi}(A)$ を $2\Pr[b = b^q_{p,i}] - 1$ で定義する。ここで、試行において、$p = s$ ならば、$\pi^q_{s,i}$ はあるクライアントオラクル $\pi^r_{p',j} = \pi^r_{c,j}$ が存在して次の条件を満たし、$p = c$ ならば、$\pi^q_{c,i}$ の通信相手 $\pi^r_{p',j}$ が同じ条件を満たすとする。その条件とは、このセッションにおけるサーバ S について次を満たすことである。
  - S が買収された時間 τ_t およびその後の時間において $\mathsf{encrypt}(\pi^q_{p,i}, *, *, *, 1)$ および $\mathsf{encrypt}(\pi^r_{p',j}, *, *, *, 1)$ の形のクエリが送られていない。
  - S が買収された時間 τ_t より後の時間において $\mathsf{encrypt}(\pi^q_{p,i}, *, *, *, *)$ および $\mathsf{encrypt}(\pi^r_{p',j}, *, *, *, *)$ の形のクエリが送られていない。
  - $\mathsf{encrypt}(\pi^q_{p,i}, *, *, *, *)$ の形のクエリで生成した暗号文を復号できる鍵を出力する $\mathsf{reveal}(\pi^q_{p,i})$ クエリおよび $\mathsf{reveal}(\pi^q_{p,i})$ クエリを送っておらず、$\mathsf{encrypt}(\pi^r_{p',j}, *, *, *, *)$ の形のクエリで生成した暗号文を復号できる鍵を出力する $\mathsf{reveal}(\pi^r_{p',j})$ クエリおよび $\mathsf{reveal}(\pi^r_{p',j})$ クエリを送っていない。

  この定義は、クライアントとサーバの間で送受信されたメッセージについての情報を攻撃者が、それ以前（上の条件の 1 つ目に対応)、あるいはその時間に（上の条件の 2 つ目に対応）サーバを買収することなく、また、初期あるいは最終のセッション鍵を出力させることなく得ることができる場合を扱っている。
- IP アドレスの偽装の尺度（IP spoofing advantage） $\mathbf{Adv}^{\mathsf{ips}}_{\Pi}(A)$ を次の条件を満た

すオラクル $\pi^q_{c,i}$ および $\pi^r_{s,j}$ およびメッセージ m' が存在する確率と定義する。
- 攻撃者が $\mathsf{send}(\pi^r_{s,j}, m')$ というクエリを送り、$\pi^r_{s,j}$ がこれを拒否しない。
- S_j は買収されていない。
- m' はそれ以前の接続要求のクエリ（connect または resume）の出力でない。
- 時間 τ_t における $\pi^q_{c,i}$ に関する他のクエリは $\mathsf{connprivate}(\pi^q_{c,i}, \pi^r_{s,j})$ のみである。

この定義は、攻撃者がサーバに、そのサーバに接続の要求をしたことがないか、あるいは、最初の接続のみ要求したが同じ時間にそれ以上の接続要求をしなかったクライアントについての接続要求を受け入れさせる攻撃を扱っている。つまり、攻撃者はサーバが受け付けることを期待して接続要求を送り、それより前にはconnprivate、つまり攻撃者にはその出力が見えない接続要求のクエリしか送ることができない。

以上の定義を用いて、QC プロトコル Π が QACCE 安全であることを、その尺度 $\mathbf{Adv}^{\mathsf{QACCE}}_{\Pi} = \mathbf{Adv}^{\mathsf{s-imp}}_{\Pi} + \mathbf{Adv}^{\mathsf{ch-corr}}_{\Pi} + \mathbf{Adv}^{\mathsf{ips}}_{\Pi}$ がセキュリティパラメータ η について無視できることと定義する。

5.4.3 QUIC プロトコル

本節では Lychev らによる QUIC プロトコルの定式化について [107] を引用して説明する。この定式化では QC プロトコルの説明で述べた暗号プリミティブの定義を用いる。また、この定式化で用いる関数の定義を表 5.1 および表 5.2 に示す。

QUIC では、参加者は各接続の送信者と受信者の IP アドレスおよびポート番号の組に、これを識別する ID である cid を関連付ける。参加者は、受信した全てのパケットについて送信者と受信者の IP アドレスおよびポート番号をチェックし、その接続の以前の IP アドレスおよびポート番号と異なっている場合は接続を終了する。以下の説明では記述しないが、このチェックは必ず行われるものとする。

$\mathrm{AEAD} = (\mathcal{E}, \mathcal{D})$ を認証つき暗号、$\mathcal{SS} = (\mathsf{Kg}_s, \mathsf{Sign}, \mathsf{Ver})$ をディジタル署名スキーム、λ をセキュリティパラメータとする。QUIC でサポートされる署名アルゴリズムは ECDSA-SHA256 および RSA-PSS-SHA256 である。AES の Galois-Counter モード（GCM）[111] が AEAD として用いられる。QUIC の鍵生成アルゴリズムは $(pk, sk) \leftarrow \mathsf{Kg}(\lambda)$ および $k_{\mathtt{stk}} \xleftarrow{\$} \{0,1\}^{128}$ により鍵を生成し、pk をサーバの公開鍵、$(sk, k_{\mathtt{stk}})$ をサーバの秘密鍵とする。ここで、$k_{\mathtt{stk}} \xleftarrow{\$} \{0,1\}^{128}$ は集合 $\{0,1\}^{128}$ からの一様なランダムサンプリングを表す。

サーバの scfg は scfg_gen 関数により、時間ごとに更新される。ここで、$\mathcal{H}$ はハッシュ関数 SHA-256 である。scfg 生成およびそれに含まれる公開パラメータへの署名はクライアントからの接続要求に依存しない。

QUIC には 1-RTT および 0-RTT という 2 つの接続確立スキームがある。1-RTT はクライアントがある時間において最初にサーバへ接続を試みる再に用いられる。0-RTT はクライアントがその時間に少なくとも 1 回は接続を確立したサーバに再度接続する場合に用いられる。

1-RTT 接続確立

時間 τ_t において最初にクライアントがサーバに接続を試みる場合について述べる。プロトコルは図 5.1 のように、QC モデルの 4 つのステージを実行する。クライアント C およびサーバ S はともに現在の時間 τ_t を知っているものとする。C および S の入力メッセージはそれぞれ $M_c = (M_c^1, M_c^2, \ldots, M_c^u)$ および $M_s = (M_s^1, M_s^2, \ldots, M_s^w)$ である。S は前述のとおり生成した鍵 (pk, sk) および $k_{\mathtt{stk}}$ を持っている。

■初期鍵共有ステージ　このステージは 3 つのパケット m_1、m_2、および m_3 からなる。C は初期接続要求パケット m_1 を送信して接続を開始する。m_1 にはランダムに選んだ接続 ID である cid が含まれ、互いがセッションを識別するために用いられる。具体的には、C は c_i_hello(pk) を実行し、シーケンス番号 1 を持つパケットを出力する。

S は s_reject(m_1) を実行して応答する。m_2 に含まれるソースアドレストークン stk は、このセッションおよびそれ以降の 0-RTT 接続確立において自らを S に識別させるために用いる。これは TLS のセッションチケット [112] のようなものであり、クライアントの IP アドレスとタイムスタンプを認証付き暗号 AEAD の暗号化アルゴリズム $\mathcal{E}$ および鍵 k_{stk} により暗号化したものである。初期ベクトル $\mathtt{iv}_{\mathrm{stk}}$ はランダムに選ばれる。クライアントは以降の接続要求において、有効期限が切れるか IP アドレスが変更になるまで stk を用いることができる。単純化のため、ここでは stk の有効期限はそれが生成されて設定された時間の中までとする。m_2 は S の現在の状態 $\mathtt{scfg}_{\mathrm{pub}}^t$ を含む。$\mathtt{scfg}_{\mathrm{pub}}^t$ は S' の Diffie-Hellman（DH）公開値と、全ての公開値についてのサーバの秘密鍵 sk を用いた署名 prof を含み、有効期限を持つ。

C は m_2 を受信した後、c_hello(m) を実行して応答する。ここで、C は $\mathtt{scfg}_{\mathrm{pub}}^t$ の署名と有効期限を確認し、ノンスと DH の公開値を生成してサーバに送る。このとき、PKI の存在を仮定し、C は S の公開鍵を持っており署名を検証できるとする。

この時点で、C と S はそれぞれ get_i_key_c(m_3) および get_i_key_s(m_3) を実行して初

クライアント（M_c, pk_j）		サーバ（M_s）
$M_c = (M_c^1, M_c^2, \ldots, M_c^u)$		$M_s = (M_s^1, M_s^2, \ldots, M_s^w)$
（1）初期鍵の共有		
$m_1 \leftarrow \mathrm{c\_i\_hello}(pk)_j$		
	$\xrightarrow{m_1}$	
		$m_2 \leftarrow \mathrm{s\_reject}(m_1)$
	$\xleftarrow{m_2}$	
$m_3 \leftarrow \mathrm{c\_hello}(m_2)$		
	$\xrightarrow{m_3}$	
$ik \leftarrow \mathrm{get\_i\_key\_c}(m_3)$		$ik \leftarrow \mathrm{get\_i\_key\_s}(m_3)$
（2）初期データの送受信		
for $\alpha \in [\imath]$		for $\beta \in [\jmath]$
$\mathtt{sqn}_c \leftarrow \alpha + 2$		$\mathtt{sqn}_s \leftarrow \beta + 1$
$m_4^\alpha \leftarrow \mathrm{pak}(ik, \mathtt{sqn}_c, M_c^\alpha)$		$m_5^\beta \leftarrow \mathrm{pak}(ik, \mathtt{sqn}_c, M_s^\beta)$
$m_4 \leftarrow (m_4^1, \ldots, m_4^\imath)$	$\xrightarrow{m_4}$	$m_5 \leftarrow (m_5^1, \ldots, m_5^\jmath)$
	$\xleftarrow{m_5}$	
$\mathrm{process\_packets}(ik, m_5)$		$\mathrm{process\_packets}(ik, m_4)$
（3）（最終）鍵の共有		
		$\mathtt{sqn}_s \leftarrow 2 + \jmath$
		$m_6 \leftarrow \mathrm{s\_hello}(m_3, ik, \mathtt{sqn}_s)$
	$\xleftarrow{m_6}$	
$k \leftarrow \mathrm{get\_key\_c}(m_6, \mathtt{sqn}_s)$		$k \leftarrow \mathrm{get\_key\_s}(m_6)$
（4）（最終）データの送受信		
for $\alpha \in \{\imath + 1, \ldots, u\}$		for $\beta \in \{\jmath + 1, \ldots, w\}$
$\mathtt{sqn}_c \leftarrow \alpha + 2$		$\mathtt{sqn}_s \leftarrow \beta + 2$
$m_7^\alpha \leftarrow \mathrm{pak}(k, \mathtt{sqn}_c, M_c^\alpha)$		$m_8^\beta \leftarrow \mathrm{pak}(k, \mathtt{sqn}_c, M_s^\beta)$
$m_7 \leftarrow (m_7^1, \ldots, m_7^\imath)$	$\xrightarrow{m_7}$	$m_8 \leftarrow (m_8^1, \ldots, m_8^\jmath)$
	$\xleftarrow{m_8}$	
$\mathrm{process\_packets}(k, m_8)$		$\mathrm{process\_packets}(k, m_7)$

図 5.1　1-RTT の接続確立

期鍵 ik を得る。このとき、サーバは同じ接続を複数回処理しないようにする必要があり、使用済みのノンスをストライクレジスタ（srike-register）`strike` と呼ばれるしくみで管理する。`strike` にはすでに処理したノンスの乱数が保存されており、乱数が `strike` にすでに含まれていれば、サーバは接続要求を拒否する。さらに、ストライクレジスタには有効な時間域 $\mathtt{strike}_{\mathtt{rng}}$ が決まっており、ノンスに含まれるタイムスタンプがこの時間域

に含まれない場合も、接続を拒否する。これにより、サーバはこの時間域の間だけストライクレジスタを保持しておけばよい。このために、クライアントとサーバの間の時刻の同期が必要である。ここでは、$\mathtt{strike}_{\mathtt{rng}}$ はストライクレジスタが生成された時間に限るとする。

生成された初期鍵 $ik = (ik_c, ik_s, \mathtt{iv})$ は、128 ビットのアプリケーション鍵 ik_c および ik_s、そして 2 つの 4 バイトの初期ベクトルプレフィクスからなる $\mathtt{iv} = (\mathtt{iv}_c, \mathtt{iv}_s)$ として用いられる。C は S に送信するデータを暗号化するために ik_s と $\mathtt{iv}_s$ を用い、S から受信するデータを復号するために ik_c と $\mathtt{iv}_c$ を用いる。S はこの逆を行う。このステージは $\mathtt{scfg}^t_{\mathrm{pub}}$ および stk が有効である時間 τ_t において一度だけ実行される。

■初期データの送受信ステージ　このステージは 2 つのパケット列 m_4 および m_5 からなる。C および S はそれぞれ初期データ $M^1_c, \ldots, M^\imath_c$ および $M^1_s, \ldots, M^\jmath_s$ を、それぞれ各 $\alpha \in [\imath]$ について $\mathrm{pak}(ik, \mathtt{sqn}_c, M^i_\alpha)$ および各 $\beta \in [\jmath]$ について $\mathrm{pak}(ik, M^j_\beta)$ を実行して暗号化し送信する。ここで、ik を用いた AEAD による暗号化が用いられる。また、$\mathtt{sqn}_c$ および $\mathtt{sqn}_s$ はそれぞれ C および S が送信するパケットのシーケンス番号である。

get_iv は AEAD のための初期ベクトル（IV）を出力する。初期ベクトルは、S が送信するパケットにおいては $\mathtt{iv}_c$ と $\mathtt{sqn}_s$ を連結したもの、C が送信するパケットにおいては $\mathtt{iv}_s$ と $\mathtt{sqn}_c$ を連結したものである。$\mathtt{iv}_c$ および $\mathtt{iv}_s$ の長さは 4 バイト、$\mathtt{sqn}_c$ および $\mathtt{sqn}_s$ の長さは 8 バイトであるため、初期ベクトルの長さは 12 バイトである。

QUIC ではシーケンス番号は各パケットにその内容によらず順に与えられ、最初のパケットのシーケンス番号は 1 である。図 5.1、表 5.1、および表 5.1 における $\imath$ および $\jmath$ はそれぞれ C および S が（最終）鍵共有ステージの前に送るメッセージの個数に相当する。C および S それぞれ相手からパケットを受信すると、process_packets によりこれを復号してシーケンス番号の順に出力する。

■（最終）鍵共有ステージ　このステージはメッセージ m_6 のみからなり、S は s_hello を実行して新たな Diffie-Hellman 鍵共有のための値を生成し、このうち公開値を ik により AEAD を用いて暗号化してクライアントに送信する。

クライアントはサーバから受信した新たな公開値の真正性を ik を用いて検証する。この時点で、サーバとクライアントはそれぞれ $\mathrm{get\_key\_s}(m_6)$ および $\mathrm{get\_key\_c}(m_6)$ を実行して（最終の）セッション鍵 k を得ることができる。

■（最終）データの送受信ステージ　このステージパケットの列 m_7 および m_8 からなる。このセッションでは以降は、C および S はそれぞれ残りのデータ列 $M^{\imath+1}_c, \ldots, M^u_c$ およ

scfg_gen(sk, τ_t, λ) :
$q \xleftarrow{\$}$ { 大きさ λ の素数 }, $g \xleftarrow{\$}$ {$\mathbb{Z}_q$ の生成元 }
$x_s \xleftarrow{\$} \mathbb{Z}_{q-1}, y_s \leftarrow g^{x_s}, \mathtt{pub}_s \leftarrow (g, q, y_s), \mathtt{sec}_s \leftarrow x_s$
$\mathtt{expy} \leftarrow \tau_{t+1}, \mathtt{scid} \leftarrow \mathcal{H}(\mathtt{pub}_s, \mathtt{expy})$
$\mathtt{str} \leftarrow$ "QUIC server config signatuer"
$\mathtt{prof} \leftarrow \mathsf{Sign}(sk, (\mathtt{str}, 0\text{x}00, \mathtt{scid}, \mathtt{pub}_s, \mathtt{expy}))$
$\mathtt{scfg}^t_{\mathrm{pub}} \leftarrow (\mathtt{scid}, \mathtt{pub}_s, \mathtt{expy}, \mathtt{prof})$
$\mathtt{scfg} \leftarrow (\mathtt{scfg}^t_{\mathrm{pub}}, \mathtt{sec}_s)$

c_i_hello(pk) :
$\mathtt{cid} \xleftarrow{\$} \{0,1\}^{64}$
return $(\mathtt{IP}_c, \mathtt{IP}_s, \mathtt{port}_c, \mathtt{port}_s, \mathtt{cid}, 1)$

s_reject(m) :
$\mathtt{iv}_{\mathrm{stk}} \xleftarrow{\$} \{0,1\}^{96}$
$\mathtt{stk} \leftarrow (\mathtt{iv}_{\mathrm{stk}}, \mathcal{E}(k_{\mathrm{stk}}, \mathtt{iv}_{\mathrm{stk}}, \varepsilon, (\mathtt{IP}_c, \mathtt{current\_time}_s)))$
return $(\mathtt{IP}_s, \mathtt{IP}_c, \mathtt{port}_s, \mathtt{port}_c, \mathtt{cid}, 1, \mathtt{scfg}^t_{\mathrm{pub}}, \mathtt{stk})$

c_hello(m) :
abort if either
  $\mathtt{expy} \leq \tau_t$, or
  $\mathsf{Ver}(pk, (\mathtt{str}, 0\text{x}00, \mathtt{scid}, \mathtt{pub}_s, \mathtt{expy}), \mathtt{prof}) \neq 1$
  where $\mathtt{str} \leftarrow$ "QUIC server config signature"
$r \xleftarrow{\$} \{0,1\}^{160}, \mathtt{nonc} \leftarrow (\mathtt{current\_time}_c, r)$
$x_c \xleftarrow{\$} \mathbb{Z}_{q-1}, y_c \leftarrow g^{x_c}, \mathtt{pub}_c \leftarrow (g, q, y_c)$
$\mathtt{pkt\_info} \leftarrow (\mathtt{IP}_c, \mathtt{IP}_s, \mathtt{port}_c, \mathtt{port}_s)$
return $(\mathtt{pkt\_info}, \mathtt{cid}, 2, \mathtt{stk}, \mathtt{scid}, \mathtt{nonc}, \mathtt{pub}_c)$

xtrct_xpnd($\mathtt{pms}, \mathtt{nonc}, \mathtt{cid}, m, \ell, \mathtt{init}$) :
$\mathtt{ms} \leftarrow \mathtt{HMAC}(\mathtt{nonc}, \mathtt{pms})$
if $\mathtt{init} = 1$ then $\mathtt{str} \leftarrow$ "QUIC key expansion"
else $\mathtt{str} \leftarrow$ "QUIC forward secure key expansion"
$\mathtt{info} \leftarrow (\mathtt{str}, 0\text{x}00, \mathtt{cid}, m, \mathtt{scfg}^t_{\mathrm{pub}})$
$key \leftarrow (\mathtt{T}(1), \mathtt{T}(2), \ldots)$ の最初の ℓ バイト
where $\mathtt{T}(i) \leftarrow \mathrm{HMAC}(\mathtt{ms}, (\mathtt{T}(i-1), \mathtt{info}, 0\text{x}0i))$,
  $\mathtt{T}(0) \leftarrow \varepsilon$,
return key

get_i_key_c(m) :
$\mathtt{ipms} \leftarrow y_s^{x_c}$
return xtrct_xpnd($\mathtt{ipms}, \mathtt{nonc}, \mathtt{cid}, m, 40, 1$)

get_i_key_s(m) :
$(\mathtt{iv}_{\mathrm{stk}}, \mathrm{tk}) \leftarrow \mathtt{stk}$
$\mathrm{dec} \leftarrow \mathcal{D}(k_{\mathrm{stk}}, \mathtt{iv}_{\mathrm{stk}}, \varepsilon, \mathrm{tk})$
abort if either
  $\mathrm{dec} = \perp$,
  dec の最初の 4 バイトが $\mathtt{IP}_c$ と異なる,
  dec の最後の 4 バイトに対応する
  タイムスタンプが許される時間の外,
  $r \in \mathtt{strike}, \quad \tau_t \notin \mathtt{strike}_{\mathtt{rng}}$,
  $\mathtt{scid}$ に対応する $\mathtt{scfg}^{t'}_{\mathrm{pub}}$が
  未知あるいは有効期限が切れている, or
  $\mathtt{pub}_c$の g および q が $\mathtt{pub}_s$のものと異なる
$\mathtt{ipms} \leftarrow y_c^{x_s}$
return xtrct_xpnd($\mathtt{ipms}, \mathtt{nonc}, \mathtt{cid}, m, 40, 1$)

get_iv(H, κ) :
$(k_c, k_s, \mathtt{iv}_c, \mathtt{iv}_s) \leftarrow \kappa$
クライアントの場合 $src \leftarrow c, dst \leftarrow s$
サーバの場合 $src \leftarrow s, dst \leftarrow c$
$(\mathtt{cid}, \mathtt{sqn}) \leftarrow H$
return $(\mathtt{iv}_{dst}, \mathtt{sqn})$

pak($\kappa, \mathtt{sqn}, m$) :
$(k_c, k_s, \mathtt{iv}_c, \mathtt{iv}_s) \leftarrow \kappa$
クライアントの場合 $src \leftarrow c, dst \leftarrow s$
サーバの場合 $src \leftarrow s, dst \leftarrow c$
$\mathtt{pkt\_info} \leftarrow (\mathtt{IP}_{src}, \mathtt{IP}_{dst}, \mathtt{port}_{src}, \mathtt{port}_{dst})$
$H \leftarrow (\mathtt{cid}, \mathtt{sqn})$
$\mathtt{iv} \leftarrow \mathrm{get\_iv}(H, \kappa)$
return $(\mathtt{pkt\_info}, \mathcal{E}(k_{dst}, \mathtt{iv}, H, m))$

表 5.1　QUIC プロトコルで用いられる関数（1）

び $M_s^{\jmath+1}, \ldots, M_s^w$ を k により暗号化および認証する。

ik と同様に、$k = (k_c, k_s, \mathtt{iv})$ は、128 ビットのアプリケーション鍵 k_c および k_s、そして 2 つの 4 バイトの初期ベクトルプレフィクスからなる $\mathtt{iv} = (\mathtt{iv}_c, \mathtt{iv}_s)$ として用いられる。C は S に送信するデータを暗号化するために k_s と $\mathtt{iv}_s$ を用い、S から受信するデータを復号するために k_c と $\mathtt{iv}_c$ を用いる。S はこの逆を行う。

process_packets$(\kappa, p_1, \ldots, p_v)$:
$(k_c, k_s, \mathtt{iv}_c, \mathtt{iv}_s) \leftarrow \kappa$
クライアントの場合 $src \leftarrow c$, $dst \leftarrow s$
サーバの場合 $src \leftarrow s$, $dst \leftarrow c$
for each $\gamma \in [v]$:
  $(H_\gamma, c_\gamma) \leftarrow p_\gamma$
  $\mathtt{iv}_\gamma \leftarrow \mathrm{get\_iv}(H_\gamma, \kappa)$
  $m_\gamma \leftarrow \mathcal{D}(k_{src}, iv_\gamma, H_\gamma, c_\gamma)$
return $(m_1, ldots, m_v)$

s_hello$(m_3, ik, \mathtt{sqn})$:
$(ik_c, ik_s, \mathtt{iv}_c, \mathtt{iv}_s) \leftarrow ik$
$\tilde{x}_s \xleftarrow{\$} \mathbb{Z}_{q-1}, \tilde{y}_s \leftarrow g^{\tilde{x}_s}, \tilde{\mathtt{pub}}_s \leftarrow (g, q, \tilde{y}_s)$
$H \leftarrow (\mathtt{cid}, \mathtt{sqn})$
$e \leftarrow \mathcal{E}(ik_c, (\mathtt{iv}_c, \mathtt{sqn}), H, (\mathtt{scfg}^t_{\mathrm{pub}}, \tilde{\mathtt{pub}}_s, \mathtt{stk}))$
return $(\mathtt{IP}_s, \mathtt{IP}_c, \mathtt{port}_s, \mathtt{port}_c, H, e)$

get_key_c(m) :
$(\mathtt{IP}_s, \mathtt{IP}_c, \mathtt{port}_s, \mathtt{port}_c, \mathtt{cid}, \mathtt{sqn}, e) \leftarrow m$
abort if $\mathcal{D}(ik_c, (\mathtt{iv}_c, \mathtt{sqn}), (\mathtt{cid}, \mathtt{sqn}), e)) = \perp$
$\mathtt{pms} \leftarrow \tilde{y}_s^{x_c}$
return xtrct_xpnd$(\mathtt{pms}, \mathtt{nonc}, \mathtt{cid}, m, 40, 0)$

get_key_s(m) :
$\mathtt{pms} \leftarrow y_c^{\tilde{x}_s}$
return xtrct_xpnd$(\mathtt{pms}, \mathtt{nonc}, \mathtt{cid}, m, 40, 0)$

c_hello_0_RTT$(\mathtt{stk}, \mathtt{scfg}^t_{\mathrm{pub}})$:
$\mathtt{cid} \xleftarrow{\$} \{0,1\}^{64}$
$r \xleftarrow{\$} \{0,1\}^{160}, \mathtt{nonc} \leftarrow (\mathtt{current\_time}_c, r)$
$x_c \xleftarrow{\$} \mathbb{Z}_{q-1}, y_c \leftarrow g^{x_c}, \mathtt{pub}_c \leftarrow (g, q, y_c)$
$\mathtt{pkt\_info} \leftarrow (\mathtt{IP}_c, \mathtt{IP}_s, \mathtt{port}_c, \mathtt{port}_s)$
return $(\mathtt{pkt\_info}, \mathtt{cid}, 2, \mathtt{stk}, \mathtt{scid}, \mathtt{nonc}, \mathtt{pub}_c)$

表 5.2　QUIC プロトコルで用いられる関数（2）

0-RTT 接続確立

クライアント C がサーバ S とすでに時間 τ_t において接続を確立したことがあれば、C は初期鍵共有のステージにおいて m_1 および m_2 の送受信を省略し、c_hello_0_RTT を実行してすでに持っている stk と scid、および新たに生成した cid、nonc、$\mathtt{pub}_c$ を含むメッセージを生成し、S に送ることで接続を要求する。ここで、$\mathtt{pub}_c$ は新しく生成した Diffie-Hellman の一時的な公開鍵である。

S は m_3 を受信すると、nonc が新しいものであることをストライクレジスタのチェックにより確かめ、stk が有効であることと scid が既知であり失効していないことを確認する。この確認に失敗した場合は、S は 1-RTT に移行し、s_reject によりメッセージを作成して送信し、以降に実行は 1-RTT の場合と同じである。この確認が成功した場合は、最初の 2 つのパケットの送受信が省略されたことを考慮したシーケンス番号を使うことを除いて、1-RTT と同じプロトコルを実行する。

5.4.4　形式検証におけるプロトコル記述言語

本節では ProVerif においてプロトコルおよびその安全性を記述する言語のうち、本研究の検証で必要な部分について述べる。また、プロトコル中で用いるデータの型についての説明は省略する。言語の全体およびその意味などの詳細は文献 [49] を参照されたい。

ProVerif では応用 π 計算と呼ばれる言語を利用し、これに関数記号や等式規則を追加してプロトコルを記述する。

■言語、項、等式　無限個の変数 $x, y, z, n, \ldots$ が与えられていると仮定する。変数は通信路やその上で送受信されるデータなどを表す。ProVerif の利用者が宣言する関数記号を変数に適用して得られる記号列を項と呼ぶ。関数記号はデータに対する操作を表し、引数をもたない関数記号を定数と呼ぶ。関数記号の宣言とともに、これを含む等式を定義できる。項は、等式を通じて等価な他の項に書き換えられる。関数記号のうちいくつかはデストラクタとして定義される。デストラクタは、それがプロトコルの実行中に現れるとすぐに等式によって書き換えられるか、それができない場合はそのプロトコルの実行は終了する。

■プロセス　プロトコルはプロセスとして記述される。プロセスは次の文法により再帰的に定義される。

$$
\begin{aligned}
P, Q ::= \;& \texttt{0} \mid \texttt{!}P \mid P \texttt{||} Q \mid \texttt{in(}c\texttt{, }x\texttt{);}\, P \mid \texttt{out(}c\texttt{, }t\texttt{);}\, P \mid \texttt{new } x\texttt{;}\, P \mid \\
& \texttt{if } cond \texttt{ then } P \mid \texttt{let } x\texttt{=}t \texttt{ in } P \mid \texttt{event } ev\texttt{(}t_1\texttt{,...,}t_n\texttt{);}\, P \mid \\
& \texttt{insert } tab\texttt{(}t\texttt{);}\, P \mid \texttt{get } tab\texttt{(=}t\texttt{);}\, P \mid
\end{aligned}
$$

プロセス 0 は何もしないプロセスであり、他のプロセスの一部として現れる場合は記述を省略する。$!P$ はプロセス P の無限個の並列実行を表す。$P||Q$ はプロセス P および Q の並列実行を表す。並列に実行されるプロセスは、互いに通信を行うことができ、コマンド in(c, x) および out(c, t) により通信路 c からの項の受信と変数 x への格納、および、通信路 c への項 t の送信を表す。また、コマンド new x により鍵やノンスなどのランダムなデータを新たに生成して変数 x に格納する。コマンド if $cond$ then により条件 $cond$ をチェックし、条件が偽なら実行を停止する。ここで $cond$ は項の等式 $t = t'$ または不等式 t<>t' である。コマンド let x=t in により変数 x に項 t が代入される。コマンド event ev(t_1,...,t_n) によりイベントが発行される。ここで ev はイベントを構成するための関数記号である。イベントの発行はプロトコルの動作に影響を与えず、セキュリティの定義から参照される。項の集合を表すテーブルと呼ばれるデータ構造があり、コマンド insert tab(t) によりテーブル tab への項 t の追加、そして、コマンド get tab(=t) によりテーブル tab に項 t が含まれることをチェックする。含まれない場合はプロセスは停止する。

さらに、プロセスを記述する言語にパターンマッチングの機能を追加して拡張した言語を用いる。たとえば、let (x, =s)=t in P は、項 t を項のペア (t_1, t_2) に分解し、t_1 を x に格納し、さらに if s=t_2 in P と同様に振る舞う。

■セキュリティ要求　ProVerif により解析可能なセキュリティのうち、本研究の検証では対応表明（correspondence assertion）と観測等価性（observational equivalence）を用いる。対応表明は次のように記述される。

query x_1,...,x_n;
  event ev(x_1,...,x_n) ==> event $ev_1(\tilde{x}^{(1)})$ || ⋯ ||$ev_k(\tilde{x}^{(k)})$

ここで、各 $\tilde{x}^{(j)}$ は $\{x_1, \ldots, x_n\}$ に含まれる変数からなる変数列である。この言明は、変数 $x_1, \ldots, x_n$ の任意の値について、もしイベント event ev(x_1,...,x_n) が発行されたなら、$ev_1(\tilde{x}^{(1)}), \ldots, ev_k(\tilde{x}^{(k)})$ のいずれかがその実行において発行されたという性質である。

観測等価性は特別な関数記号 choice を用いたプロセスとして記述される。プロセス中に現れる choice[t_0, t_1] の形の項を全て t_0 で置き換えたプロセスと t_1 で置き換えたプロセスを攻撃者が識別できないとき、このプロセスが観測等価性を満たすという。

5.5　提案手法の詳細と結果

5.5.1　形式検証のための暗号プリミティブの記述

QUIC で用いられる暗号プリミティブ、すなわち暗号プロトコル中で用いられる暗号アルゴリズムは、認証付き暗号（AEAD）、Diffie-Hellman 鍵共有、ディジタル署名、ハッシュ関数（$\mathcal{H}$）、および鍵生成関数（xtrct_xpnd）である。これらの ProVerif における定式化を図 5.2 に示す。ディジタル署名および Diffie-Hellman 鍵共有の定式化は、ProVerif を用いた他の検証例と同様である。なお、ProVerif を用いた検証例のいくつかが ProVerif のソースコードとともに配布されている。

キーワード fun に引き続き関数をその型とともに宣言している。bitstring、nonce_t、bool、G、および exponent、はそれぞれビット列、ノンス（乱数）、ブール値（true または false）、Diffie-Hellman 鍵共有で用いる群 G の元、同じくその指数、および自然数に対応する型である。

キーワード reduc とともに関数の間に成り立つ等式を定義している。たとえば次の等式

```
Ver(pk(rg), m, Sign(sk(rg), m, rs)) = true
```

は、テキスト m に対する秘密鍵 sk(rg) および乱数 rs を用いた署名 Sign(sk(rg), m, rs) について、検証アルゴリズム Ver を適用すると、Ver の入力のテキストが署名のテキスト

```
(* Signature *)
fun pk(bitstring): bitstring.
fun sk(bitstring): bitstring.
fun Sign(bitstring, bitstring, nonce_t): bitstring.
fun Ver(bitstring, bitstring, bitstring): bool
reduc
  forall rg: bitstring, m: bitstring, rs: nonce_t;
  Ver(pk(rg), m, Sign(sk(rg), m, rs)) = true.

(* Diffie-Hellman *)
const q: natural. (* used just as a spaceholder *)
const g: G.
fun exp(G, exponent): G.
nounif x: exponent; attacker(exp(g, x)). (* for speedup *)
equation
  forall x: exponent, y: exponent;
  exp(exp(g, x), y) = exp(exp(g, y), x).
(* Generaiton of server's secret (for every time period) *)
fun x_s(bitstring, bitstring): exponent [private].

(* Hash *)
fun Hash(bitstring): bitstring.

(* xtrct_xpnd  *)
fun xtrct_xpnd(G, bitstring, bitstring, bitstring, bitstring): bitstring.
fun k_c_of(bitstring): bitstring.
fun k_s_of(bitstring): bitstring.
fun iv_c_of(bitstring): bitstring.
fun iv_s_of(bitstring): bitstring.

(* AEAD *)
fun E(bitstring, bitstring, bitstring, bitstring): bitstring.
fun Ef(bitstring, bitstring, bitstring, bitstring, bitstring): bitstring.
reduc
  forall k: bitstring, iv: bitstring, h: bitstring, m: bitstring;
  D(k, iv, h, E(k, iv, m, h)) = m;
  forall k: bitstring, iv: bitstring, h: bitstring, m: bitstring, r: bitstring;
  D(k, iv, h, Ef(k, iv, m, h, r)) = m.
reduc
  forall m: bitstring, iv: bitstring, k: bitstring, h: bitstring;
  extract_IV(E(k, iv, m, h)) = iv;
  forall m: bitstring, iv: bitstring, k: bitstring, h: bitstring, r: bitstring;
  extract_IV(Ef(k, iv, m, h, r)) = iv.
reduc
  forall m: bitstring, iv: bitstring, k: bitstring, h: bitstring;
  extract_AD(E(k, iv, m, h)) = h;
  forall m: bitstring, iv: bitstring, k: bitstring, h: bitstring, r: bitstring;
  extract_AD(Ef(k, iv, m, h, r)) = h.
```

図 5.2　暗号プリミティブの記述

と一致しており、さらに検証鍵 pk(rg) が署名鍵 sk(rg) に対応するならばその結果が true になることを意味する。ここで、rg は鍵ペア (sk(rg), pk(rg)) を生成するときに用いた乱数である。

Diffie-Hellman 鍵共有については、equation により、$(g^x)^y = (g^y)^x$ に相当する等式を与えている。reduc で与えられた等式は等号の左から右への書き換えにしか用いられないが、equation で与えられた等式では双方向の書き換えが行われる。ProVerif は整数の性質を全て正確に扱うことができないため、整数の性質を利用した攻撃は十分には検出できない。QUIC プロトコルの定式化のため、指数の生成に用いる関数 x_s を宣言している。これは、セッションおよび時間ごと必ず新たな指数が生成されるようにするために用いている。

ハッシュ関数と鍵生成関数はそれぞれ関数 Hash と xtrct_xpnd により表され、これらの関数については等式が定義されていない。このため、これらの関数からその引数の情報を得ることは正規参加者にも攻撃者にもできない。また、鍵生成関数の出力から、送受信するメッセージを暗号化するための鍵および IV（初期ベクトル）を得るための関数 k_c_of、k_s_of、iv_c_of、および iv_s_of も同様に宣言する。

AEAD の定式化ではまず暗号化を表す関数 E を宣言する。項 E(k, iv, m, h) は平文 m を鍵 k、IV iv、認証データ（additional authenticated data）h を用いて暗号化した暗号文を表す。そして復号アルゴリズムを表す関数記号 D およびこれに関する等式を定義する。この他、暗号文から IV や認証データを抽出する関数 extract_IV および extract_AD を抽出する関数も定義した。これはプロトコルの正規参加者は使用せず、攻撃者が攻撃に使用することができる。

5.5.2 形式検証のための QUIC プロトコルの記述

QACCE モデルではプロトコルは状態を保持するオラクルとして定式され、QUIC プロトコルおいても参加者は状態を保持している。しかし ProVerif は状態の更新と時刻を限定的な形でしか扱えないため、本研究では QUIC プロトコルを状態の更新を明示的には扱わない形で定式化しなおし、サーバの設定とソースアドレストークン（stk）の有効期限が切れることはないとする。有効期限について条件を緩めた影響で、1 つのサーバにおいて同じ時間に 2 つの設定（$\mathtt{scfg}^t_{\mathrm{pub}}$）が同時に有効になることがないという前提が崩れる。このため Lychev らによって [107] 発見された攻撃よりも多くの、現実的には実行不可能な攻撃が本研究の検証では検出される。したがって検出された攻撃については QUIC プロトコルの定義に照らして実行可能性を検討する必要がある。

前節で導入した言語を用いて QUIC プロトコルのクライアントおよびサーバを記述したプロセスをそれぞれ図 5.4 および図 5.3 に示す。これらのプロセスは、図 5.5 に示すメイン（main）プロセスから、IP アドレスやディジタル署名の鍵などのパラメータを与えられて繰り返し起動され、公開通信路からメッセージを受信し、プロトコルの定義に従って公開通信路から応答を送信する。攻撃者は公開通信路で送受信されるメッセージを傍受することができ、さらに公開通信路からこれらのプロセスにメッセージを送信することもできる。これは QACCE モデルにおける send クエリに対応する。なお、攻撃者の動作は検証ツール ProVerif に組み込まれているため、プロトコルの記述には現れない。

クライアントプロセスは、main プロセスから自身及び通信相手のサーバの IP アドレスとポート番号を与えられて起動される。これらの IP アドレスとポート番号は、メインプロセスが攻撃者より受信したものである。クライアントプロセスはメッセージ m_2 を受信して処理した後は、プロトコルの残りの部分を回数に制限なく繰り返し実行する。この繰り返し動作は記号 “!” で記述されている。この繰り返しのうち最初の 1 回が 1-RTT のセッションに、2 回目以降が 0-RTT のセッションに対応する。このように、攻撃者がクライアント自身及び通信相手の IP アドレス及びポート番号を決められるようにし、さらに 0-RTT セッションを繰り返し起動しておくことで QACCE モデルにおける connect 及び resume クエリに相当する動作が ProVerif のスクリプトとして記述される。

サーバプロセスも同様に、自らの IP アドレスとポート番号は攻撃者から受け取ったものがサーバプロセスを通じて与えられて起動される。また、サーバがソースアドレストークンを生成するための鍵 $k_{\mathtt{stk}}$ 及びディジタル署名の鍵（pk(rg) 及び sk(rg)）、さらに時間ごとに生成されるサーバの状態 scfg はメインプロセスが生成したものが与えられる。

クライアントおよびサーバのプロセスは、プロトコルの実行中に図 5.6 に示すプロセス Oenc、Odec、Ocorrupt、および、Oreveal を起動する。これらのプロセスは、QACCE モデルにおけるクエリ encrypt、decrypt、corrupt、及び、reveal に対応する。また、安全性の定義においてこれらのクエリの呼び出しについての条件があるため、これらのプロセスは応答する際に対応するイベントを event 文により発行し、形式検証における安全性の記述から参照できるようにする。なお、Odec および Ocorrupt ではサーバが持つ署名鍵に対応する検証鍵をサーバを識別する ID として使用している。また、この他のイベントの発行がクライアントおよびサーバのプロセスに含まれており、安全性の定義から参照される。

QACCE モデルの通信路の毀損の尺度（channel-corruption advantage）においては、encrypt クエリに与えた 2 つの平文のどちらが暗号化されたかを示す秘密のビット $b_{p,i}^{j}$ を攻撃者が知る確率を考慮している一方、その他の尺度ではこのビットは用いられず、

encrypt クエリは単に暗号文を作るために用いられる。このため、Oenc は後者の用途のためのプロセスとして、単一の平文を入力するよう定義した。通信路の毀損の尺度における encrypt クエリに対応するプロセスとして Oenc2 を定義した。この尺度について形式検証を行う際は Oenc の代わりに Oenc2 を用いる。QACCE モデルでは encrypt クエリにおいて同じ IV（初期ベクトル）を繰り返し使うことができないが、ProVerif でこれを直接表現することが難しいため、Oenc2 では同じ入力でも毎回異なる暗号文が生成されるよう、暗号化関数として E の代わりに Ef を用いた。これにより、攻撃者が Oenc2 に (msg0, msg1) を送り、次に (msg0, msg1') を送って暗号文を比較することでどちらが暗号化されたか比較するという QACCE モデルでは考慮する必要のない攻撃が成功しないようにしている。

Odec はクエリで与えられた暗号文の復号に成功した場合に、秘密のビット b_p^j を返す代わりに decrypt イベントを発行し、通信路の毀損の尺度の検証において安全性の記述の際にこれを参照する。クライアントおよびサーバのプロセスは、鍵の共有に成功した場合に自らの役割（サーバあるいはクライアント）ともに共有した鍵をテーブル exchanged_keys に保存し、Odec はクエリに応答する際にこのテーブルを参照して通信相手が鍵を共有したかどうか確認し、共有していない場合は応答を中止する。これは、QACCE モデルにおける通信路の毀損の尺度（channel-corruption advantage）の定義において $b_{p,i}^j$ を持っている $\pi_{p,i}^j$ の通信相手が正規の参加者であること必要があるためである。この条件がない場合、通信相手が攻撃者である場合には攻撃者が鍵を知っているため、復号可能な暗号文を送信できてしまう。

さらに、後述する IP アドレスの偽装に対する安全性の形式検証のために攻撃者には内容が見えない秘匿通信路 cp を用いる。この通信路に攻撃者がメッセージを送ることはできるようにするために、公開通信路 c からメッセージを受け取り秘匿通信路 cp に転送するプロセス c_to_cp を用いる。

5.5.3 形式検証のための QACCE 安全性の記述

QACCE 安全性は次のように、サーバへのなりすまし、通信路の毀損、および、IP アドレスの偽装に対する安全性として記述される。記述は ProVerif へのクエリによって定義される。ここではクエリとは、第 5.4.4 節で述べた対応表明と観測等価性の記述のことであり、攻撃者からオラクルへのアクセスとは異なる。

```
let server(pk_s: bitstring, sk_s: bitstring, k_stk: bitstring,
    current_time_s: bitstring, scfg_t_pub: bitstring, sec_s: exponent,
    IP_s: bitstring, port_s: bitstring) =
  let (scid: bitstring, pub_s: bitstring, expy: bitstring, prof: bitstring)
      = scfg_t_pub in
  Ocorrupt(pk_s, (sk_s, k_stk, sec_s)) |
  (* Initial Key Agreement *)
  in(c, m1: bitstring);
  let (IP_c: bitstring, =IP_s, port_c: bitstring, =port_s, cid: bitstring,
       =seqno(phase_initial_key_agreement)) = m1 in
  new iv_stk: bitstring;
  let stk = (iv_stk, E(k_stk, iv_stk, (IP_c, current_time_s), null)) in
  let m2 = (IP_s, IP_c, port_s, port_c, cid, seqno(phase_initial_key_agreement),
            scfg_t_pub, stk) in
  out(cp, m2);
  Ocorrupt(pk_s, (sk_s, k_stk, sec_s, iv_stk)) |
  ! in(c, m3: bitstring);
    insert conversations((role_server, m1, m2, m3));
    let (pkt_info: bitstring, =cid, =seqno(phase_initial_key_agreement),
  stk': bitstring, =scid, nonc: bitstring, pub_c: bitstring) = m3 in
    let (=IP_c, =IP_s, =port_c, =port_s) = pkt_info in
    let (iv_stk': bitstring, tk: bitstring) = stk' in
    let (=IP_c, current_time_s': bitstring) = D(k_stk, iv_stk', null, tk') in
    event server_accept_m3(pk_s, cid, m3);
    let (=g, =q, y_c: G) = pub_c in
    let (ipms: G) = expS(y_c, sec_s) in
    let ik = xtrct_xpnd(ipms, nonc, cid, m3, n1) in
    insert exchanged_keys((role_server, ik));
    (Oreveal(cid, phase_initial_data_exchange, role_client, k_s_of(ik)) |
     Oreveal(cid, phase_initial_data_exchange, role_server, k_c_of(ik)) |
     (* Initial Data Exchange *)
     (! Oenc((role_client, ik), k_c_of(ik), iv_c_of(ik), cid,
      phase_initial_data_exchange, role_server)) |
     (! Odec((role_client, ik), pk_s, k_s_of(ik), iv_s_of(ik), cid,
      phase_initial_data_exchange, role_client)) |
     (* Key Agreement *)
     new x_s': exponent;
     Ocorrupt(pk_s, (sk_s, k_stk, sec_s, iv_stk, x_s')) |
     let y_s' = exp(g, x_s') in
     let pub_s' = (g, q, y_s') in
     let sqn = seqno(phase_key_agreement) in
     let e = E(k_c_of(ik), (iv_c_of(ik), sqn), (scfg_t_pub, pub_s', stk'),
        (cid, sqn)) in
     let m6 = (IP_s, IP_c, port_s, port_c, ((cid, sqn), e)) in
     let pms = expS(y_c, x_s') in
     event server_k_set((role_server, m1, m2, m3, m6), cid);
     out(c, m6);
     let k = xtrct_xpnd(pms, nonc, cid, m6, n0) in
     insert exchanged_keys((role_server, k));
     (Oreveal(cid, phase_data_exchange, role_client, k_s_of(k)) |
      Oreveal(cid, phase_data_exchange, role_server, k_c_of(k)) |
      (* Data Exchange *)
      (! Oenc((role_client, k),
       k_c_of(k), iv_c_of(k), cid, phase_data_exchange, role_server)) |
      (! Odec((role_client, k), pk_s,
       k_s_of(k), iv_s_of(k), cid, phase_data_exchange, role_client)))).
```

図 5.3　サーバプロセスの記述

```
let client(pk_s: bitstring, IP_c: bitstring, IP_s: bitstring,
           port_c: bitstring, port_s: bitstring) =
  (* Initial Key Agreement *)
  new cid: bitstring;
  let m1 = (IP_c, IP_s, port_c, port_s, cid, seqno(phase_initial_key_agreement)) in
  event client_initiate(cid);
  out(c, m1);
  in(cp, m2: bitstring);
  let (=IP_s, =IP_c, =port_s, =port_c, =cid, =seqno(phase_initial_key_agreement),
       scfg_t_pub: bitstring, stk: bitstring) = m2 in
  ! in(c, (current_time_c: bitstring, cid': bitstring));   (*Input from adversary*)
    let (scid: bitstring, pub_s: bitstring, expy: bitstring, prof: bitstring)
        = scfg_t_pub in
    if Ver(pk_s, (str_qscfg, n0, scid, pub_s, expy), prof) = true then
    new r: bitstring;
    let nonc = (current_time_c, r) in
    new x_c: exponent;
    let y_c = exp(g, x_c) in
    let pub_c = (g, q, y_c) in
    let pkt_info = (IP_c, IP_s, port_c, port_s) in
    let (iv_stk: bitstring, tk: bitstring) = stk in
    let m3 = (pkt_info, cid', seqno(phase_initial_key_agreement), stk, scid, nonc,
              pub_c) in
    event client_send_m3(m3);
    out(c, m3);
    let (=g, =q, y_s: G) = pub_s in
    let ipms = exp(y_s, x_c) in
    let ik = xtrct_xpnd(ipms, nonc, cid', m3, n1) in
    insert exchanged_keys((role_client, ik));
    (Oreveal(cid', phase_initial_data_exchange, role_client, k_s_of(ik)) |
     Oreveal(cid', phase_initial_data_exchange, role_server, k_c_of(ik)) |
     (* Initial Data Exchange *)
     (! Oenc((role_server, k), k_s_of(ik), iv_s_of(ik), cid',
             phase_initial_data_exchange, role_client)) |
     (! Odec((role_server, k), pk_s, k_c_of(ik), iv_c_of(ik), cid',
             phase_initial_data_exchange, role_server)) |
     (* Key Agreement *)
     in(c, m6: bitstring);
       (* k := get_key_c(m6, sqn_s) *)
     let sqn = seqno(phase_key_agreement) in
     let (=IP_s, =IP_c, =port_s, =port_c, ((=cid', =sqn), e: bitstring)) = m6 in
     let (=scfg_t_pub, pub_s': bitstring, =stk)
         = D(k_c_of(ik), (iv_c_of(ik), sqn), (cid', sqn), e) in
     let (=g, =q, y_s': G) = pub_s' in
     let pms = exp(y_s', x_c) in
     let k = xtrct_xpnd(pms, nonc, cid', m6, n0) in
     insert exchanged_keys((role_client, k));
     event client_k_set((role_server, m1, m2, m3, m6), cid', pk_s);
     (Oreveal(cid', phase_data_exchange, role_client, k_s_of(k)) |
      Oreveal(cid', phase_data_exchange, role_server, k_c_of(k)) |
       (* Data Exchange *)
      (! Oenc((role_server, k),
              k_s_of(k), iv_s_of(k), cid', phase_data_exchange, role_client)) |
      (! Odec((role_server, k), pk_s,
              k_c_of(k), iv_c_of(k), cid', phase_data_exchange, role_server)))).
```

図 5.4　クライアントプロセスの記述

```
process
  (! new rg: bitstring;
     (! new k_stk: bitstring;
        out(c, pk(rg));
        ! in(c, IP_s: bitstring);
          in(c, port_s: bitstring);
        ! in(c, current_time_s: bitstring);
          let y_s = exp(g, x_s(rg, current_time_s)) in
          let sec_s = x_s(rg, current_time_s) in
          let pub_s = (g, q, y_s) in
          let str = str_qscfg in
          let expy: bitstring = s(current_time_s) in
          let scid = Hash((pub_s, expy)) in
          new r: nonce_t;
          let prof = Sign(sk(rg), (str_qscfg, n0, scid, pub_s, expy), r) in
          let scfg_t_pub = (scid, pub_s, expy, prof) in
          server(pk(rg), sk(rg), k_stk, current_time_s, scfg_t_pub, sec_s,
                 IP_s, port_s))
     | (! in(c, IP_c: bitstring);
          in(c, IP_s: bitstring);
          in(c, port_c: bitstring);
          in(c, port_s: bitstring);
          client(pk(rg), IP_c, IP_s, port_c, port_s))
     | tap_m2)
```

図 5.5　メインプロセスの記述

■サーバへのなりすまし　サーバへのなりすましに対する安全性は次のクエリにより記述する。

```
query conv: bitstring, cid: bitstring, pk_s: bitstring;
  event(client_k_set(conv, cid, pk_s)) ==>
  event(server_k_set(conv, cid))
  || event(revealed(cid, phase_initial_data_exchange, role_server))
  || event(corrupted(pk_s)).
```

このクエリは、クライアントが接続 ID cid を持つセッションにおいて一連のメッセージ列 conv を送受信することでサーバ pk_s と鍵を共有したことを表すイベントが発生したなら次のいずれかが起こっている、という性質を表す。なお、ここではサーバが持つ署名鍵に対応する検証鍵 pk_s をサーバの識別子として用いている。

- 同じ接続 ID および一致するメッセージ列により鍵を共有したことを表すイベントを発行したサーバが存在する。
- 同じ接続 ID でプロトコルを実行したサーバで、初期データ送受信のステージで初期鍵を漏洩したものが存在する。
- サーバ pk_s が買収されている。

```
(* Encryption *)
let Oenc(matching_conversation: bitstring,
         key: bitstring, iv: bitstring,
         sess: bitstring, ph: bitstring, sender_role: bitstring) =
  in(c, (msg: bitstring, H: bitstring));
  let (cid: bitstring, sqn: bitstring) = H in
  let C = E(key, (iv, sqn), msg, H) in
  event encrypt(sess, ph, sender_role, C, H);
  out(c, (H, C)).

(* Encryption, for analyzing secrecy *)
let Oenc2(check_exchanged: bitstring,
          key: bitstring, iv: bitstring,
          sess: bitstring, ph: bitstring, sender_role: bitstring) =
  in(c, (msg0: bitstring, msg1: bitstring, H: bitstring));
  let msg = choice[msg0, msg1] in
  let (cid: bitstring, sqn: bitstring) = H in
  new r: bitstring;
  let C = Ef(key, (iv, sqn), msg, H, r) in
  get exchanged_keys(=check_exchanged) in
  out(c, (H, C)).

(* Decryption *)
let Odec(check_exchanged: bitstring, pk_s: bitstring,
         key: bitstring, iv: bitstring,
         sess: bitstring, ph: bitstring, sender_role: bitstring) =
  in(c, (C: bitstring, H: bitstring));
  let (cid': bitstring, sqn: bitstring) = H in
  let nonce = (iv, sqn) in
  let msg = D(key, nonce, H, C) in
  get exchanged_keys(=check_exchanged) in
  event decrypt(pk_s, sess, ph, sender_role, C, H).

(* Reveal *)
let Oreveal(sess: bitstring, ph: bitstring, sender_role: bitstring, k: bitstring) =
  (in(c, =q_reveal); event revealed(sess, ph, sender_role); out(c, k)).

(* Corruption *)
let Ocorrupt(pk_s: bitstring, s: bitstring) =
  in(c, =q_corrupt);
  event corrupted(pk_s);
  out(c, s).

(* Allows attacker to send message through private channel "cp" *)
let c_to_cp = ! in(c, m: bitstring); out(cp, m).
```

図 5.6　攻撃者からのクエリに応えるプロセスの記述

言い換えれば、買収や鍵の漏洩などを攻撃者が行わないなら、クライアントが鍵を共有したならば、対応する正規のサーバーが存在して同じメッセージ列により鍵の共有を完了したという性質を表す。

■通信路の毀損　QACCE モデルにおいては、通信路の毀損は攻撃者による decrypt クエリにおいて復号が成功する場合と、これによらず攻撃者が $b^r_{p,j}$ の推測に成功する場合がある。形式検証においてはこれらをそれぞれ対応表明と観測等価性として別々に定義する。これらの安全性をそれぞれメッセージの真正性（authenticity）と秘匿性（secrecy）と呼

ぶことにする。

メッセージの真正性をサーバへのなりすましの場合と同様に次のクエリにより記述する。

```
query pk_s: bitstring, cid: bitstring, ph: bitstring,
      sender_role: bitstring, C: bitstring, H: bitstring;
  event(decrypt(pk_s, cid, ph, sender_role, C, H)) ==>
  event(encrypt(cid, ph, sender_role, C, H))
  || event(revealed(cid, ph, sender_role))
  || event(corrupted(pk_s)).
```

このクエリは、ある接続において、接続 ID が cid、セッションに参加しているサーバが pk_s、ステージ（フェーズ）が ph であるときに、送信者の役割（サーバまたはクライアント）が sender_role であるような decrypt クエリが暗号文 C およびヘッダ H について成功したイベントが発生したなら次のいずれかが起こっているという性質を表す。

- その暗号文は同一の接続、ステージ、送信者の役割、ヘッダについて、Oenc により作られた暗号文である。
- 同一の接続、ステージ、送信者の役割について、送信用の鍵が漏洩した。
- サーバが買収された。

言い換えれば、買収や鍵の漏洩を攻撃者が行なっていないなら、復号に成功する暗号文は Oenc へ要求する以外の方法では得られない、という性質を表す。なお、QACCE の定義では、サーバが買収される前に行われた encrypt クエリを用いて攻撃を行うことが許されているが、ProVerif における検証では時間を扱うことが難しいためこれを許さない定義になっている。

秘匿性の定式化ではこれまでに説明した QUIC プロトコルの記述をわずかに変更した 2 つのプロトコルの変種を考える。これらの変種では Oenc の代わりに Oenc2 を用いる。Oenc2 は 2 つの平文 msg0 および msg1 を受け取り choice[msg0,msg1] を暗号化する。choice[msg0,msg1] は 2 つの変種のうち最初のものにおいては msg0、もう 1 つにおいては msg1 を表す。ProVerif は choice を含むプロトコル記述については、自動的にこのような 2 つの変種を考え、攻撃者から見てどちらの変種が実行されているか識別できるか否かを検証する。上記の記述を完全に行なったものは [113] より取得できる。

■IP アドレスの偽装　攻撃者がこの攻撃に成功するためには、攻撃者はメッセージ m_3 を作成してサーバに送り、これを受理させなければならない。このとき、そのセッションにおける最初のメッセージ m_1 および、m_1 へのサーバの応答 m_2、そして m_2 に対するクライアントの応答は攻撃者には見えてはいけない。これは、QACCE モデルの定義において、このセッションを行なっている時間におけるクライアントへのクエリは connprivate のみであることに対応する。このため、この攻撃についての安全性の検証においては、サーバのプロセスを変更し、メッセージ m_1、m_2、m_3 は秘匿通信路 cp を通じて送信されるようにしする。ただし、攻撃者は他のセッションにおいてはこれらのメッセージを知ることができるため、その場合はそのセッションの ID `cid` を含むイベント `capture(cid)` を発行する。また、クライアントが m_3 を送信する際に、イベント `client_send_m3(m3)` を発行するものとする。

このように変更したクライアントおよびサーバのプロセスについて、IP についての安全性は次の ProVerif の安全性クエリとして定義できる。

```
query S: bitstring, cid: bitstring, stk: bitstring m3: bitstring;
event(server_accept_m3(S, cid, stk, m3)) ==>
event(client_send_m3(m3))
|| event(session_captured(cid))
|| event(corrupted(S)).
```

これは、サーバ S が最初の接続要求に対し m_2 を送信し、その後 m_3 を受理するのは、そのセッションを攻撃者が傍受したか、正規のクライアントが m_3 を送信したか、あるいは、S が買収された場合のみであるという性質である。

5.5.4　形式検証の結果

前節で定義した安全性の全てについて、ProVerif は攻撃を発見した。ただし、後述するように、これらの攻撃はプロトコルの欠陥というよりもむしろ、QACCE 安全性の定義が強すぎることに起因するものである。以下では発見された攻撃は QACCE モデルおよび QUIC プロトコルの定義に即して説明する。

■サーバへのなりすまし　ProVerif により、次のような中間者攻撃が発見された。攻撃者はまず、サーバから有効なソースアドレストークン stk' を入手する。次に、stk' が失効する前に、クライアントとサーバの間の通信に介入し、サーバが送信するメッセージ

m_2 に含まれる新たなソースアドレストークン stk を、stk' で置き換える。クライアントはそれ以降、stk' を用いてプロトコルを実行するが、これもまた有効なソースアドレストークンであるため、クライアントとサーバは鍵共有に成功する。このとき、メッセージ m_2 についてはサーバとクライアントの通信が一致しないため、一致する通信（matching conversation）を持たない。これにより、QACCE モデルおよび前節で定義したサーバへのなりすましの定義によれば、攻撃が成功したことになる。もちろんこの攻撃により、通常の意味でのサーバへのなりすましが成功したとは言えない。

しかし、このように定義を改めた場合でも次のような別の攻撃が発見される。クライアントとサーバが初期データの送受信のステージに進んだ際に、encrypt クエリを用いて次の最終鍵の共有のステージのメッセージ m_6 に相当するメッセージ m_6' を構成しておき、最終鍵の共有のステージでサーバが送信した m_6 を m_6' に置き換える。ここで、m_6' に含まれる $\tilde{\mathtt{pub}}'_s$ は攻撃者が任意に生成したものであるため、m_6 に含まれる $\tilde{\mathtt{pub}}_s$ と（高い確率で）異なる。このため、サーバとクライアントの間でメッセージ m_6 が一致しない。QACCE モデルでは、encrypt クエリにおいて攻撃者がシーケンス番号を含むヘッダを任意に選ぶことができるが、実際の QUIC プロトコルの実行では、初期データの送受信のステージでサーバやクライアントが行う暗号化ではシーケンス番号は自動的に生成されるため、攻撃者が関与して特定のシーケンス番号を含む暗号文を作らせることはできない。したがって、この攻撃によってもまた、通常の意味でサーバへのなりすましが成功したとは言えない。

■通信路の毀損　通信路の毀損のうちメッセージの真正性の検証において、前述のサーバへのなりすましについての 2 つ目の攻撃に類似の攻撃が検出された。中間攻撃者はメッセージ m_6 を傍受し、これを decrypt クエリに渡すことで復号に成功することができる。具体的には、m_6 からヘッダ H を取り出し、この通信の当事者であるクライアント $\pi^r_{c,j}$ へクエリ $\mathsf{decrypt}(\pi^r_{c,j}, m_6, H, \mathsf{init})$ を発行すればよい。メッセージ m_6 は初期鍵によって正しく暗号化されているため、復号に成功し、攻撃者は秘密のビットを得ることができる。

通信路の毀損のうち秘匿性の検証においては、次のような中間者攻撃が発見された。攻撃者は初期鍵共有ステージの通信を傍受し、接続 ID$\mathtt{cid}$、サーバがメッセージ m_2 で送信する公開情報 $\mathtt{scfg}^t_{\mathrm{pub}}$ およびソースアドレストークン $\mathtt{stk}$ を得る。次に、任意に選んだ値 $\tilde{\mathtt{pub}}'_s$ について 3 つ組 $m = (\mathtt{scfg}^t_{\mathrm{pub}}, \tilde{\mathtt{pub}}'_s, \mathtt{stk})$ および空なビット列 $m' = \varepsilon$ を構成し、初期データの送受信のフェーズで、このセッションのサーバに対しクエリ $\mathsf{encrypt}(\pi^r_{s,j}, m, m', H, \mathsf{init})$ を発行する。この結果得た暗号文 m_6 をクライアントに送信し、クライアントが（最終の）データ送受信のステージに移行するか否かを観察するこ

とで、encrypt クエリによって得た暗号文に対応する平文が m であるか m' であるかがわかる。

■IP アドレスの偽装　IP アドレスの偽装については、次のような攻撃が発見された。攻撃者は、あるセッションにおける正しいソースアドレストークン `stk` を傍受し、これが失効する前に同じクライアントとサーバに対して以下を行う。攻撃者は同じサーバに対し、クライアントの IP アドレスとポート番号の情報を含む最初の接続要求のメッセージ m_1 を send クエリにより送り、サーバがその（応答 m_2 を送信した後、`stk` を含む応答 m_3 を作成してサーバに送る。このとき、攻撃者に m_2 が見えなかったとしても、前のセッションで傍受した `stk` を用いて m_3 を構成し、サーバに受理させることができる。この結果は特に驚くべきことではない。ソースアドレストークンはクライアントが最近その IP アドレスを使ったことを確認するためのものである。このため、トークンをその有効期間内に保持していた者が、その IP アドレスを所有しているとしてセッションを開始することを防ぐことはできない。これはソースアドレストークンによる保護の限界を示した当然の結果である。Lychev らの QUIC の定式化では、ソースアドレストークンの有効期限をそれが発行された時間に限り、同時に複数のソースアドレストークンが有効になることがない定式化になっているため、彼らの定式化ではこの攻撃は不可能である。本研究の定式化では、有効期限の条件を緩めているためこのような攻撃が発見された。実際の QUIC プロトコルにおいても同時に複数のソースアドレストークンが有効になることは考えられるため、このような場合も考慮されて然るべきと考えられる。

5.5.5　QACCE 安全性の修正

前節において発見された攻撃はいずれも、プロトコルが期待する安全性を満たさないという意味での攻撃ではない。これは QACCE 安全性の定義が強すぎることを意味する。本節では、QACCE 安全性の定義を修正する方法について検討する。

サーバへのなりすましについて発見された攻撃のうち最初のものは、一致する通信の定義が強すぎることに起因する。特にメッセージ m_1 および m_2 は攻撃者により改変が可能なため、一致する通信の定義から除き、m_3 および m_6 の一致のみを考えるようにすべきである。

さらに、サーバへのなりすましについての 2 つ目の攻撃は、攻撃者が encrypt クエリにおいて、攻撃者が任意にシーケンス番号を選べることに起因するため、encrypt クエリにおいてシーケンス番号を選べないようにすれば良い。これにより、通信路の毀損につい

ての 2 つ目の攻撃も除外することができる。さらに、decrypt クエリについても、他のステージで用いられたシーケンス番号を持つ暗号文を許さないようにすることで、通信の毀損についての 1 つ目の攻撃を除外することができる。

IP アドレスの偽装については、事前に有効なソースアドレストークンを取得することができれば IP アドレスの偽装に成功することは明らかであるため、そのような状況は安全性の定義から除外すべきと考えることもできる。この場合、これを除外する条件を追加すればよい。

安全性の定義を以上のように修正し、これを ProVerif のスクリプトに反映させて改めて検証を行ったところ、新たな攻撃は発見されなかった。ただし、IP アドレスの偽装については ProVerif により検証が終了しなかったため検証を打ち切っている。

5.6 関連研究

関連研究として、トランスポートレイヤの暗号プロトコルである QUIC と TLS、そして TLS の前身である SSL を取り上げ、これらに形式検証を適用した研究について述べる。なお、本研究を開始して以降の研究状況についても述べる。まずトランスポートレイヤの暗号プロトコルである QUIC と TLS の発展について時系列に基づいて述べ、次にこれらに形式検証を適用した研究について述べる。

5.6.1 トランスポートレイヤの暗号プロトコルの概要

web サーバとクライアントの間の通信では、これを安全に行うために、web サーバのみ、あるいはクライアントも含めた双方を認証して鍵を共有し、その鍵に基づいて通信内容を暗号化することが一般的である。さらに、メッセージが正しい順序で受信され、消失などがおこっていないことも保証する。このようなプロトコルは web 以外の通信でも用いられ、TCP/IP のプロトコルスタック上でトランスポートレイヤに相当するため、本章ではトランスポートレイヤの暗号プロトコルと呼ぶ。

トランスポートレイヤの暗号プロトコルとして、SSL およびその後継である TLS、Google によって提案された QUIC、および、後に IETF によって標準化されたバージョンの QUIC を取り上げる。本章では、QUIC プロトコルのうち、Google により提案されたものを Google QUIC、IETF により標準化されたものを IETF QUIC と呼んで区別する。まず、これらのプロトコルの概要について述べる。詳細は [114] などを参照されたい。

TLS/SSL および QUIC は、前述の通り、主に web サーバとクライアント間の通信に

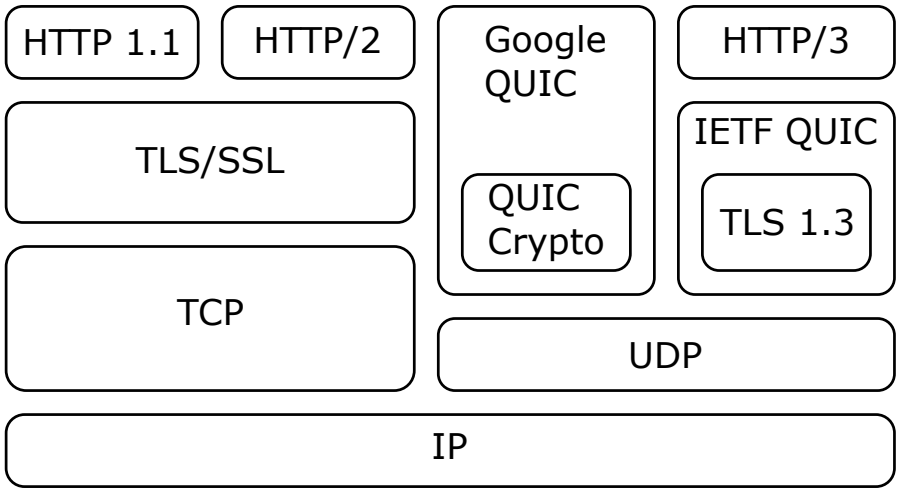

図 5.7　HTTP と TLS および QUIC の関係 ([2] をもとに作成)

おいて認証を行い、秘匿性および完全性を保護するためのプロトコルである。電子証明書を用いてサーバを認証することを可能にするとともに、Diffie-Hellman 鍵交換を用いて、その後の通信のための暗号鍵を共有する。クライアントの認証は必須ではないが可能である。暗号鍵の共有までの通信をセキュリティハンドシェイクと呼び、使用する暗号スイートについてのネゴシエーションも行われる。

これらのプロトコルは web のブラウザとサーバの間の通信のためのプロトコルである HTTP の下位レイヤのプロトコルとして最もよく用いられる。これらの関係を図 5.7 に示す。

これらのプロトコルは、主に安全性の向上のための発展を遂げており、新しいバージョンを策定することで前のバージョンで発見された脆弱性を取り除いてきている。これまでに発見された脆弱性としては、再ネゴシエーションに関するもの、ブロック暗号の CBC モードにおける初期ベクトル（IV）の予測可能性およびパディングに関するもの、RSA 暗号のパディングに関するもの、オプションの機能であるデータの圧縮を用いたサイドチャネル攻撃、強度が低いブロック暗号やハッシュ関数に関するものなどがある。さらに、プロトコルの脆弱性を実装の脆弱性を組み合わせる攻撃も発見されている。実装の脆弱性としては、プロトコルの後方互換性のために旧バージョンのプロトコルも利用可能にしているサーバにおいて旧バージョンのプロトコルの使用を中間者攻撃により強制させることができる脆弱性、および、中間車攻撃により暗号スイートとして暗号強度の低い輸出グレードの暗号を中間者攻撃により使用させることができる脆弱性などがある。

さらに、特に TLS1.3 および QUIC においては、HTTP とともに、データの送受信に

かかる遅延を小さくする取り組みと、複数のデータを並行して送信する多重化の取り組みとともに発展してきた。TLS/SSL は TCP 上で動作し、パケットの消失や受信順序の入れ替わりが起きた際の検出と再送や並べかえは TCP の機能を用いる。TCP は接続開始時に 3 ウェイハンドシェイクが必要であり、TLS/SSL プロトコルのデータを送信できるまでに遅延が生じる。さらに、パケットの消失が起きた際にパケットが再送されるまでは後のデータが処理されない（いわゆる head of line ブロッキング）ため、ここでも遅延が生じる。さらに、TCP の 3 ウェイハンドシェイクの後に、TLS/SSL のセキュリティハンドシェイクによってサーバの認証および鍵交換を行うため、HTTP のデータを送受信できるまでにはさらに遅延がある。送受信の多重化については、多重化されない場合、たとえば複数のテキストや画像などのコンテンツかならなる web ページについて、どれか 1 つのコンテンツの処理やダウンロードに時間がかかった場合、それが完了するまで他のコンテンツをダウンロードすることができない。もちろん複数の TLS/SSL のセッションを利用すれば可能だが、前述の遅延の問題があることに加えて、認証・鍵共有のための負荷が大きい。このため、HTTP/2 では、単一の TLS のセッション上で複数のストリームを用いてコンテンツを並行してダウンロードできるようにして多重化を行った。しかし、HTTP/2 においても、TCP のパケット消失が起こった場合は、すべてのストリームに遅延が発生することにはかわりがない。

遅延の改善と多重化を抜本的に実現するため、Google QUIC は TCP ではなく UDP 上で動作するプロトコルとなっており、パケットの再送などは Google QUIC の機能に含まれている。このようにして TCP の head of line ブロッキングの問題を回避しつつ、多重化も実現した。さらに、セキュリティハンドシェイクの途中でデータの送受信を開始できるようするとともに、終了したセッションを再開して即座にデータの送受信も可能にすることで、セキュリティハンドシェイクに係る遅延を改善した。

Google QUIC におけるセキュリティハンドシェイクにおける改善は、TLS 1.3 においても取り込まれている。また、IETF での QUIC の標準化にあたり、IETF QUIC のセキュリティハンドシェイクは TLS 1.3 のものを用いる形となっている。HTTP/3 は IETF QUIC 上のプロトコルとして開発された。

SSL

SSL（The Secure Soket Layer Protocol）は米国のネットスケープコミュニケーションズ社によって開発されたプロトコルである。SSL には 1.0, 2.0, 3.0 の 3 つのバージョンがあるが、バージョン 1.0 には安全上の欠陥があり、公開されなかった。SSL 2.0 は 1994 年に公開され [115, 116]、同社の web ブラウザに実装された。SSL 2.0 には、メッセージ

認証コードとして MD5 しか用いることができない、ハンドシェイクメッセージが保護されないため弱い暗号スイートを使用させる中間者攻撃が可能である、などの脆弱性が知られており、利用が停止されている [117]。脆弱性に対応するため、SSL 3.0 が 1996 年に公開された [118]。

TLS

SSL の開発は IETF に引き継がれ、TLS（Transport Layer Security）プロトコルとして標準化されている。1999 年に TLS 1.0[119]、2006 年に TLS 1.1[120]、2008 年に TLS 1.2[121]、2018 年に TLS 1.3[122] として標準化されている。TLS 1.0 から TLS 1.1 および TLS 1.2 への変更は基本的には前のバージョンに見つかっていた安全上の問題の改善である一方、TLS 1.3 への変更は、セキュリティハンドシェイクによる遅延を小さくする 0-RTT の導入や、レガシーな暗号スイートの削除、静的 RSA 暗号の削除、ハンドシェイクメッセージの暗号化、など多岐にわたる大きな変更となった。

Google QUIC

TCP 上で動作する TLS の遅延の問題を解決するため、Google 社は UDP 上で動作する新しいトランスポートレイヤ暗号プロトコルとして QUIC を提案した。TCP の遅延を回避するために UDP を用いるだけでなく、TLS1.3 にも引き継がれる 0-RTT も導入している。QUIC は同社のサービスおよび web ブラウザに QUIC を実装して利用している。QUIC のセキュリティハンドシェイクは QUIC Crypto と呼ばれている。

IETF QUIC

QUIC はその後、IETF において標準化が進められた。IETF での標準化ではセキュリティハンドシェイクとしては TLS1.3 のものが使用され、QUIC Crypto は採用されていない。このため、UDP 上で動作し、セキュリティハンドシェイクは TLS1.3 のものを用い、ハンドシェイク後のデータ（レコードレイヤのプロトコル）については独自のパケットの暗号化、並べ替え、再送などが行われる。HTTP プロトコルの最新版である HTTP/3 は IETF QUIC 上で動作し、1 つの QUIC のセッション上で複数の HTTP のリクエストを並行して処理することが可能で、遅延の低減をさらに向上させている。

5.6.2 トランスポートレイヤ暗号プロトコルへの形式検証の適用

本節では、暗号プロトコルの形式検証の枠組みおよびツールについて述べ、その後で SSL、TLS、QUIC の形式検証の取り組みについて述べる。

SSL

Michell ら [123] は SSL3.0 のドラフト [124] をもとにそのハンドシェイクプロトコルの検証を行った。汎用の状態遷移システム検査ツール Murϕ[43] を用い、検査が有限時間で完了するよう、攻撃者も含めたシステムの取りうる状態の数に上限を設けて検査を行った。彼らは SSL 3.0 の検証のために検証の正確さを損なわずに状態空間を削減する工夫も行った。この研究では、SSL 3.0 を極度に単純化した安全ではないプロトコルから開始し、攻撃を発見できるかどうかを確認し、プロトコルに少しずつ署名や暗号を追加していくことで攻撃を防ぐことができていることを確認した。安全性としては以下のものを検証した：

- サーバとクライアントが共有した鍵などの機密情報が確かに一致すること。
- 共有した機密情報が攻撃者に知られないこと。
- 互いに相手を認証できていること。なお、ここではクライアント認証の利用を仮定している。
- 合意した暗号スイートは両者がサポートしている暗号スイートの中で最も強力なものであること
- 双方が SSL 3.0 をサポートする場合は SSL 3.0 でハンドシェイクが行われること

その結果、これらの性質が成り立つことを検証できた。ただし、最後の性質については、セッションの再開が行われる場合は成り立たないことも示した。なお、この事実は Wagner と Schneier によっても指摘されている [125]。

TLS

Paulson[126] は TLS 1.0 の安全性を、汎用の定理証明システムである Isabelle/HOL[27] 上で帰納的手法 [44] により検証した。具体的には次の安全性を証明した:

- セッション鍵（正確には premaster secret および master secret）の秘匿性。
- ハンドシェイクで送受信されたすべてのメッセージのハッシュ値を含む Finished

メッセージの認証。すなわち、このメッセージが攻撃者によって捏造されたものではなく、意図した通信相手が作成したものであること。このハッシュ値はハンドシェイクで生成された鍵も用いて生成されるため、認証によりセッション全体の完全性が保証される。ただし、この研究では、クライアント証明書を用いない場合を考えているため、サーバ側の認証については、セッション鍵（premaster secret）が特定の（攻撃者でない）参加者から送信されたと仮定している。また、送受信されたすべてのメッセージのかわりに、送受信されたすべてのノンス（乱数）およびサーバおよびクライアントの ID を用いている。

この研究では、安全性を証明することができ、攻撃は発見されなかった。

Ogata と Futatsugi[127] は等式推論を用いて TLS 1.0 の安全性の検証を行った。特に、フルハンドシェイクとセッションの再開を組み合わせた場合についても考慮し、安全性としてセッション鍵（premaster secret）の秘匿性、クライアントが受信した Finished メッセージの認証、他のメッセージの認証も含めたやや強い形のメッセージの認証、そして、セッションを再開したときに受け取った Finished メッセージによる ServerHello メッセージの認証について検証を行った。プロトコルの記述は状態遷移システムとして記述し、状態遷移はシステムを観測して得られる情報を表す関数についての等式によって記述される。検証は、等式推論と機能法によって証明スクリプトを記述する形で行われ、CafeOBJ システムによってチェックされる。

Bhargavan ら [128] は、TLS1.0 の実装を関数型プログラミング言語 F#を用いて行い、この実装から形式検証のためのモデルを抽出し、プロトコル検査ツール ProVerif および CryptoVerif での検証を行った。再ネゴシエーションを含まない形式化ではあるが、セッションの再開を含む形で検証を行い、片方向および双方向の認証と、セッション鍵およびセッション鍵で暗号化されたデータの秘匿性の検証を行った。さらに、ProVerif による検証では、クライアント証明書を含まない片方向の認証とユーザのパスワード認証を組み合わせた双方向認証についても検証を行った。ProVerif での検証では Diffie-Hellman 鍵共有などの暗号プリミティブが抽象化され、暗号プリミティブは理想的な安全性を持つと仮定して検証を行う一方、CryptoVerif での検証では暗号プリミティブの安全性は暗号学的（計算量的）な仮定に基づいて検証を行った。

Tran と Futatsugi[129] は、前述の [127] の手法に加えて、自動証明の手法を用いて TLS 1.2 の安全性の証明を行った。部分的に人手により補助であるものの、証明が自動的に生成されるところに特徴がある。検証した安全性は、セッション鍵の秘匿性と、クライアントが受け取った Finished メッセージの秘匿性である。

TLS 1.3 については、そのドラフト仕様の段階から形式検証の取り組みが行われており、その結果が標準化の議論や仕様に反映されている。ドラフト仕様も TLS 1.3 の仕様の web ページ [122] よりダウンロードできる。

荒井ら [130] はプロトコル検査ツール ProVerif を用いて、TLS 1.3 のドラフト 6 のセキュリティハンドシェイクの検証を試みた。フルハンドシェイクのみを対象としており、アプリケーションデータの秘匿性と片方向の認証について検証した結果、欠陥は見つからなかった。

Cremers ら [131] は、プロトコル検査ツール Tamarin を用いて、TLS 1.3 のドラフト 10 のセキュリティハンドシェイクの検証を行った。フルハンドシェイク、事前共有鍵（PSK）を用いたハンドシェイク、事前共有鍵と（EC）DHE を用いたハンドシェイクを扱っている。特に、事前共有鍵を用いる場合における 0-RTT データの送信、および、前のセッションで共有した事前共有鍵を使ったセッションの再開も扱っている。安全性としては、片方向認証（サーバ認証）、クライアント認証を行う場合の双方向認証、セッション鍵の秘匿性および前方秘匿性（perfect forward security）、ハンドシェイクで交換されるメッセージの完全性を検証し、これらの安全性が成り立つことを検証した。さらに、ドラフト 10 を拡張してハンドシェイク後にクライアント認証を行う場合についても検証し、事前共有鍵を用いたハンドシェイクについてハンドシェイク後クライアント認証を行う場合は、中間者攻撃によるなりすましが可能であることを発見した。

Bhargavan ら [132] は、前述の [128] と同様のアプローチでプロトコル検査ツール ProVerif および CryptoVerif を用いて TLS1.3 のドラフト 18 の検証を行った。ProVerif での検証では、このドラフトのバージョンの TLS1.3 と TLS1.2 の両方が実行されうる場合において、TLS1.2 の脆弱性（RSA のパディングに関する脆弱性）を利用した TLS1.3 における署名の偽造、および、TLS1.3 を TLS1.2 にダウングレードさせる攻撃が可能であることを示した。ProVerif での検証では Diffie-Hellman 鍵共有などの暗号プリミティブが抽象化され、暗号プリミティブは理想的な安全性を持つと仮定して検証を行う一方、CryptoVerif での検証では暗号プリミティブの安全性は暗号学的（計算量的）な仮定に基づいて検証を行った。ただし、その反面、検証を完全に自動的に行うことはできず、検証に必要な補題を用意するなどの作業が必要になる。さらに、彼らは TLS1.3 の参照実装を作り、その実装のコアの部分の安全性の根拠として上記の検証を位置づけた。

Cremers ら [133] は、前述の [131] の検証を発展させ、TLS 1.3 のドラフト 21 のセキュリティハンドシェイクの検証を行った。ドラフト 21 はドラフト 10 に対して、0-RTT や事前共有鍵などが変更されており、これを反映した検証となっていおり、形式検証におけるプロトコルの記述は [131] に比べてより詳細になっており、検証対象の安全性も、ドラ

フト仕様に記述されている安全性を反映した網羅的なものになっている。

Stettler[134] は、プロトコル検査ツール Tamarin を用いて、TLS 1.3 のドラフト 13 のセキュリティハンドシェイクの検証を行った。フルハンドシェイク、事前共有鍵（PSK）を用いたハンドシェイク、セッションの再開、0-RTT を扱い、検証対象の安全性としては、鍵の秘匿性と、片方向認証（サーバ認証）、および、クライアント認証を行う場合の双方向認証、0-RTT データ以外の前方秘匿性を対象としている。

Blanchet[135] は、プロトコル検証ツール CryptoVerif を用いて、TLS1.3 の検証を行った。CryptoVerif を用いた通常の検証においては同一の参加者が複数のセッションを並行して実行する場合を一般には扱えないが、この研究ではそのような場合での安全性も含めた検証を行った。このために、あらかじめ複数のセッションについての安全性を満たす条件を考えてこれを定理として証明しておき、その条件がなり立つことを CryptoVerif で検証した。

Bhargavan ら [136] は、TLS1.3 においてハンドシェイクの最初のメッセージである Client Hello メッセージを暗号化する拡張（ECH）[137] も考慮した検証を、プロトコル検査ツール ProVerif を用いて行った。Client Hello の暗号化の目的は、このメッセージにオプションとして含むことができる、サーバのホスト名（SNI）を隠すことである。これにより、同一の IP アドレスで複数のサーバをホストしている場合にサーバへのアクセスなのか隠すことができ、クライアントのプライバシを攻撃者から保護することができる。彼らは、ECH の初期のバージョンの検証も行い、初期のバージョンではプライバシが保護されないことと、新しいバージョンでは保護されることを検証した。また、プライバシ以外の、通常の認証付き鍵交換プロトコルとしての安全性も網羅的に検証を行った。

ここまでに紹介した形式検証は、基本的にはセキュリティハンドシェイクの検証を行ったものであるが、セキュリティハンドシェイクによって共有される鍵を用いて暗号化を行うレコードレイヤの検証を行った研究として Delignat-Lavaud らの研究 [138] がある。この研究では、レコードレイヤを関数型プログラミング言語である F*を用いて実装し、そのプログラムを暗号理論的に等価な他のプログラムに順に書き換えていき、自明に安全性が成り立つプログラムに帰着させることで検証を行った。検証方法としては CryptoVerif が内部で行っている方法と同様である。

Google GUIC

本論文の研究において、櫻田ら [139] は Google QUIC をプロトコル検査ツール ProVerif を用いて検証した。Lychev らによる安全性モデル [107] を ProVerif で検証可能な形に定式化しなおすことでこれを行った。検証の結果、Lychev らが定義した安全性を満たさな

いことが明らかになった。しかし、発見された攻撃は現実的には脅威にならないものであり、Lychev らの定義した安全性が強すぎることがわかった。また、これは Lychev らによる Google QUIC の安全性証明には誤りがあることを意味する。

Zhang ら [140, 141, 142] は、プロトコル検査ツール ProVerif および Verifpal、そして汎用のモデル検査ツール Spin を用いて、Google QUIC の安全性を検証した。ただし、彼らの論文では IETF で標準化中の QUIC を検証したと主張しているが、実際には Google QUIC を対象としており、混乱が見られる。彼らは、最初の鍵交換においてクライアントの Diffie-Hellman 公開鍵が保護されず、攻撃者がクライアントになりすますことができることを問題視し、公開鍵にクライアントが署名する方式を提案している。しかし、そもそも Google QUIC ではクライアントの認証をプロトコルのレベルで行うことを目的としていないため、Google QUIC の目的とする安全性からは外れた安全性を考えたときの指摘であるといえる。

IETF QUIC

IETF QUIC は TLS1.3 のセキュリティハンドシェイクを用いているため、その安全性も TLS1.3 のそれに帰着される。しかし、セキュリティハンドシェイクによって共有される鍵を用いて暗号化を行うレコードレイヤは TLS1.3 と異なるため、この部分の安全性は検証に値する。Delignat-Lavaud ら [143] は、前述の TLS1.3 のレコードレイヤの実装と検証 [138] と同様の方法で、IETF QUIC のレコードレイヤの実装と検証を行っている。彼らは IETF QUIC の複数のドラフトを対象に検証を行っており、その過程でドラフトの誤りを発見して後のドラフトに反映されている。

5.7 まとめと今後の課題

本研究では、暗号プロトコルの安全性を自動的に検証するツールである ProVerif を用いて、QUIC プロトコルの安全性の検証を行った。具体的には、QUIC プロトコルおよび、その安全性のモデルである QACCE モデルを ProVerif のスクリプトとして記述し、検証を行った。その結果、QACCE モデルのいずれの安全性についても攻撃が検出された。これは QUIC プロトコルが安全でないということを意味せず、QACCE モデルにおける安全性の定義が強すぎることに起因する。このため本章ではさらに、QACCE 安全性を適切に修正するする方法についても検討を行った。また、修正した安全性をもとに検証を行ったところ、新たな攻撃は発見されなかった。

本章の結果は、プロトコルの安全性を手作業で解析する暗号学的な営みが、形式検証に

より補完されるケーススタディと見ることができる。手作業による解析には誤りが含まれる可能性があり、仮に安全でないプロトコルについて安全であるとの解析結果が出たとき、これを信じることは大きな危険を伴う可能性がある。本章の結果は、解析の誤りを発見してこのような危険性を回避するための方法の一つとして形式検証を用いることができることを示している。なお、本研究により形式検証によりプロトコルの安全性が完全に証明されるわけではない。本研究で用いた手法は暗号アルゴリズムについて理想的な安全性と仮定しており、暗号アルゴリズムに特有の性質を利用した攻撃は本研究で用いた手法では検出できないためである。この意味で、暗号学的な解析もまた形式検証を補完する位置付けであると言える。

本研究に残された直接的な課題として、次が挙げられる。

- もともと QC モデルおよび QACCE 安全性は TLS1.3 も含む形で定式化されていることから、本研究で形式検証のために改めて定義を与えた QACCE 安全性を TLS1.3 に適用する。これにより、QUIC と TLS1.3 に対して同じ枠組みで形式検証を適用することで、両者を比較することも可能になる。
- 本研究で用いた検証ツールである ProVerif には、時刻と状態変化を扱う機能が限定されており、明示的に扱うことができなかった。Tamarin などのツールではこれらを扱うことが可能であると考えられるため、より詳細な検証が可能になる。

SSL および TLS への形式検証の適用も含めた関連研究から、暗号プロトコルのための形式検証手法も含めた課題として次が挙げられる。

- 暗号プロトコルの形式検証では、現実的な労力と時間内で検証が完了するよう簡略化を行う。その際、例外処理やオプショナルな機能は簡略化によって無視されることが多い。TLS では、プロトコルの本体に暗号スイートのネゴシエーションや、TLS のバージョンのネゴシエーションなどがあるが、これを含む形で検証した研究は少ない。また、TLS には拡張が様々な定義されていおり、これを含む形の検証も少ない。このような付加的な機能を含む検証を行うことは、単に規模が大きなプロトコルとして扱う他ないのか、あるいは効率的に行う必要があるのかは重要な課題である。
- 暗号プロトコルのみならず、一般にプロトコルの参加者は内部状態を持ち、状態により動作が異なる。状態を持つ暗号プロトコルを扱うことができ、自動的な検証が行えるツールとして Tamarin があるが、場合により完全に自動的とは言えず、人手での介入が必要である。状態を持つ暗号プロトコルの検証をより自動的に行うこ

とも必要である。

- 暗号プロトコルへの攻撃には、処理にかかる時間やエラー処理の有無などを利用したサイドチャネル攻撃、そして DoS（Denial of Service）攻撃も考えられる。このような攻撃に対する安全性を検証するためには、これをプロトコルのモデルに組み込む必要がある。また、そのようなモデルに対して検証を行う方法も必要である。さらに、このような攻撃は暗号化などに用いる暗号アルゴリズムの仕組み（ブロック暗号や RSA 暗号のパディング、圧縮など）を利用することもあり、記号的モデルに基づく自動検証ツールでこれを扱うことは難しい。その一方、計算論的モデルに基づく検証ツールでは、安全でない場合に攻撃を自動的に発見することができず、単に検証に失敗することになり、検証に必要な補題が不足しているのか、あるいは、安全でないのか判別がつかないという問題もある。さらに、そのような攻撃はセキュリティハンドシェイクではなく、ハンドシェイク後のレコードレイヤのプロトコルに対するものであることもあり、レコードレイヤの検証を実装を含めて記述して検証する必要がある。Bhargavan らが行ったような、レコードレイヤの検証を含め、より実装に近い形での検証の研究を発展させる必要がある。
- 安全性としては、セキュリティハンドシェイクの検証では、共有した鍵の秘匿性および、片方向あるいは双方向の認証は検証の対象となる。その上で、鍵の秘匿性として前方秘匿性まで検証を行うか、認証として鍵や通信相手の情報だけでなくハンドシェイクのすべてのメッセージがサーバとクライアントで一致することを検証するか、セッションの再開について再開前と再開後の参加者に変更はないことも検証するか、などどこまで検証するかは様々である。また、レコードレイヤまで検証を含める場合は、メッセージのリプレイ攻撃についても検証を行う場合がある。TLS 1.3 では RFC の Appendix.E に満たすべき安全性について非形式的な説明と議論があるが、これを含め、どこまで形式的に検証できるかは課題である。また、同じ安全性についても、検証に仕様するツールやモデル化の方法などによりわずかな違いが生じる可能性もあり、同じ安全性でも定式化の仕方やツールによって差異がある可能性もあるため、検証結果については場合により詳細な解析が必要である。

しかし、Mitchell らによる SSL 3.0 のドラフトの検証から現在までの発展を見ると、検証ツールの発達にも支えられて、モデル化や検証される安全性も含めより詳細な検証が可能になってきている。同時に、標準化の段階で形式検証による解析も行われており、形式検証の重要性は増している。また、多大な労力は必要であるものの、プロトコルそのものだけでなく、実装にも踏み込んだ検証が可能になってきており、これらの研究は今後もさ

らに発展が期待できる。

第 6 章

タグ VLAN を用いた LAN の設定検証

6.1　研究の背景と動機

ネットワーク機器、特に Ethernet 機器の低価格化により、安価に LAN を構築できるようになり、様々な場所で多くの人がネットワークを利用できるようになった。また同時に機器の高機能化も進み、設定次第で様々な機能を用いて便利なネットワークを構築できるようになった。しかし、このような機能は設定を誤るとネットワークに思わぬ障害をもたらすことがある。

ネットワークを新たに構築したり、ネットワーク機器の設定を変更したりする際には、パケットを実際に送ってテストを行うことが多い。特に、通信ができなくなるような障害はテストによって比較的容易に発見できる。しかし、ネットワーク機器の全てのポートと、利用される可能性がある全てのパケットの組み合わせを網羅することは現実的には不可能であるため、テストに漏れが生じる恐れがある。特にパケットの漏洩を調べる場合には、ネットワーク機器の全てのポートについて、パケットをダンプする装置または PC を接続し、パケットが到達するか否かを確認しなければならない。また、IP（インターネット・プロトコル）を用いたネットワークでは、ping や traceroute といった検査用のパケットを送信することにより、2 点間の到達性や中継経路を調べることが容易であるのに対し、特に Ethernet の場合にはそのような仕組みがないため、テストがより困難である。このように、設定の正しさをテストによって確認することは網羅性において限界がある。

テストなどによって誤りが存在することがわかった場合でも、その原因がどこにあるかを特定することは、誤りの発見と同様に困難であることがある。たとえばパケットが本来

到達すべきでない端末に到達することがわかった場合、それがどのように起こったかを特定するためには、各中継機器の間にパケットをダンプする機器を狭んで通信をモニタする必要があった。このため、設定の人手での検査やテストと同様の問題があった。

6.2 先行研究と課題

ネットワークの機器の設定の正しさを検証する先行研究として、Guttman[144] は、ファイアウォールのパケットフィルタの設定の正しさを検査する方法を提案した。ファイアウォールが複数あるネットワークにおいて、端末と外部のネットワークの間に複数のファイアウォールが存在する場合に、遮断すべきパケットはこのどちらかによって遮断される必要があるが、本来到達すべきパケットを遮断してしまうと通信に障害を来す。さらに、外部のネットワークとの接続点が複数ある場合には、ある経路からは遮断されるべきパケットが遮断されない、ということが起こってはならない。つまり、複数のファイアウォールについて、整合性を持って設定する必要があり、ネットワークの規模や安全性のポリシーによってはこれは容易ではない。彼らはこの課題を解決すべく、安全性のポリシーを記述する言語を提案し、ポリシーを満たす設定を導出するアルゴリズムを提案した。

Guttman ら [145] は上記のアプローチを発展させ、IPsec によりパケットの認証と暗号化を行うネットワークにおいて、安全性のポリシーを記述する方法と、それを実現するネットワーク機器の設定を導出するアルゴリズムを提案した。IPsec ではパケットに認証用のヘッダを付加することで、パケットの発信元などの改竄を防ぐことができる。さらに、パケットの暗号化を行うことで、所望の相手にのみパケットを読ませることができる。認証ヘッダの付加とその検証、および、暗号化と復号は同一のネットワーク上の複数の機器で行われる場合があるため、これを整合性を持って設定することは容易ではない。このため、安全性のポリシーを記述する言語を提案し、ポリシーを満たす設定を導出するアルゴリズムを提案した。

先行研究では、IP パケットのレイヤで、ネットワークの安全性に関するポリシーを満たすべく、機器の設定を自動的に行う方法を提案した。先行研究には次のような課題があった。

- ネットワーク機器の設定によるパケットの制御は Ethernet のレベルで行われるものがある。特に、LAN を複数の仮想的な LAN（VLAN）に分割する技術の中で、タグ VLAN と呼ばれるものがあり、そこではパケットにタグと呼ばれる情報を付

加し、タグの内容に応じてパケットの転送先を決め、タグの削除あるいは書き換えを行う。この場合も、複数のネットワーク機器を整合性を持って設定し、VLAN のポリシーを満たす必要がある。先行研究では VLAN の設定の正しさを厳密に確認する研究は行われていなかった。

- 先行研究では、ファイアウォールや IPsec など、目的に応じてポリシーを記述する言語を定義し、設定を導出するアルゴリズムを定義していた。その一方、LAN においては、その利用目的によってより幅広い性質を求められる可能性があるため、そのような性質を統一的に記述できることが望ましいが、そのような方法は提案されていなかった。

6.3 本研究の貢献

本研究では、LAN 機器の設定のうち、VLAN に関する設定の正しさを形式検証により自動的に検証する方法を提案した。具体的な貢献は次のとおりである。

- ネットワークについて期待される性質、たとえば、VLAN 内でパケットが到達することや、他の VLAN にパケットが漏洩しないこと、パケットがループしないことなどの、ネットワークに期待される性質を様相論理を用いて統一的に記述する方法を提案した。
- ネットワークの機器の間の物理的な配線の情報が与えられていることを前提に、これと VLAN 設定の情報を元に、パケットの状態遷移モデルを構成する方法を提案した。
- これらの方法により、多くの研究の蓄積とそれに基づくツール実装がある汎用のモデル検査ツールを利用し、高速化かつ自動的な検証を可能にした。さらに、そのようなツールの中には性質が満たされない場合に反例を出力する機能を持つものがあり、これを用いることで、設定に誤りがあることを特定できることを示した。

ただし、先行研究では設定を自動的に導出するアルゴリズムを提案したが、本研究は設定が与えられた際にその正しさを検証するにとどまっており、自動的な導出は今後の課題である。

本研究ではまず、ネットワーク機器の配線の情報と設定の情報から計算機上にネットワークのモデルを構成する (第 6.5.1 節)。次に、要求仕様、すなわちネットワークに期待される検査すべき性質を CTL (計算木論理) の論理式として記述する (第 6.5.2 節)。CTL

を用いることにより、2 点間の到達性以外にも様々な性質を記述・検証することが可能になり、ネットワークの利用目的に応じた個々のネットワークに特有の性質を調べることができる。そしてネットワークのモデルと要求仕様の論理式をモデル検査ソフトウェアに入力し、検査を行う (第 6.5.3 節)。モデル検査では性質が満たされない場合に反例を求めることができる場合がある。これを用いて設定誤りによって生じる障害の具体例を求めることができ、設定誤りの箇所の特定などに利用することができる。さらに、検査に必要な時間の目安を得るために、人工的に生成したネットワークのデータに対して検査を行い、時間を計測し、現実的な時間内に検査を行なうことができることを示した (第 6.5.4 節)。本方法は広い意味ではシミュレーションによる検査であり、一般にそのようなシミュレーションを網羅的に行おうとすると多くの計算が必要になるが、効率的なモデル検査アルゴリズムおよびその実装である NuSMV を利用することにより、網羅的な検査を高速に行うことができる。

提案方法は、ネットワークの配線と設定の情報をもとに機器の設定の誤りを発見するものであるため、ネットワーク機器の故障や、ケーブルの断線・配線ミスのような物理的原因による障害を発見することはできない。この意味で、テストによる障害発見と補完的関係にあるといえる。また、障害の原因が物理的なものである場合でも、本方法を用いて原因が機器の設定にあるのか否かという切り分けに利用できる。

6.4 準備

6.4.1 タグ VLAN による仮想 LAN の構成

Ethernet 機器の便利な機能の 1 つとして、タグ VLAN[146] と呼ばれる機能がある。これは物理的な LAN を複数の仮想的な LAN セグメント (VLAN) に分割し、VLAN 内で閉じた通信を可能にする機能であり、多くの Ethernet 機器に実装されている。各 VLAN に属するパケットは、パケットに付加されたタグと呼ばれる情報によって識別される。図 6.1 にタグ VLAN を用いた簡単なネットワークを示す。このネットワークは VLAN A および VLAN B の 2 つの VLAN から構成される。VLAN A に属する端末 `Term1` からパケットが送信されると、LAN スイッチである `Switch1` がそれを受信し、`Switch2` に転送する。このとき `Switch1` はポート 2 に VLAN A の端末である `Term1` が接続されていることから、パケットに VLAN A を表すタグ `100` を付加する。逆に `Switch2` は VLAN A を表すタグが付加されたパケットを、VLAN A の端末である `Term3` が接続されているポート 2 から送信する。なお、この例ではパケットの転送先は 1 箇所のみであるが、実際

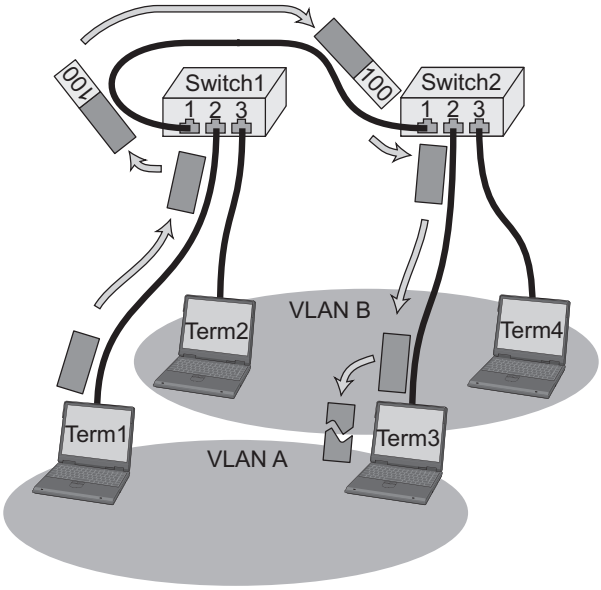

図 6.1　2 つのタグ VLAN から構成される LAN

には転送先が複数であるのが通常である。このようにして VLAN A に属する端末 Term1 および Term3 は VLAN の中で閉じた通信を行うことができる。VLAN B についても同様である。タグ VLAN は企業などでネットワークを部署ごとに論理的に分割するためにしばしば用いられる。

タグ VLAN は便利な反面、設定を誤ると障害の原因となることが多い。たとえば、Switch2 の設定が誤っており VLAN A を表すタグのついたパケットがポート 2 ではなくポート 3 に転送されるとしよう (図 6.2)。この場合、Term1 と Term3 の通信ができなくなるという障害が起こる。さらに、VLAN A のパケットが VLAN B に属する Term4 に漏洩することになり、機密情報を通信している場合などでは、セキュリティ上の大きな問題ともなりうる。特に、通信ができなくなる障害はユーザがすぐに気付くことが多いが、パケットの漏洩はパケットが端末で廃棄されるだけであるため発見が遅れる恐れがある。

タグ VLAN では機器の間でタグの利用の方法を統一する必要がある。たとえば Switch1 と Switch2 では、VLAN A を表すタグとして同じものを用いなければならない。このため、タグ VLAN の設定の正しさを調べるためには、各ネットワーク機器の設定を独立に検査するのではなく、各機器の設定か。タグの利用方法が全体として整合することを確認しなければならない。いいかえれば、機器の設定という局所的情報から、パケットの到達性や漏洩などネットワークの大域的性質を確認する必要がある。小規模なネットワークでは人手で確認することも可能であるが、ネットワークの規模が大きくなる

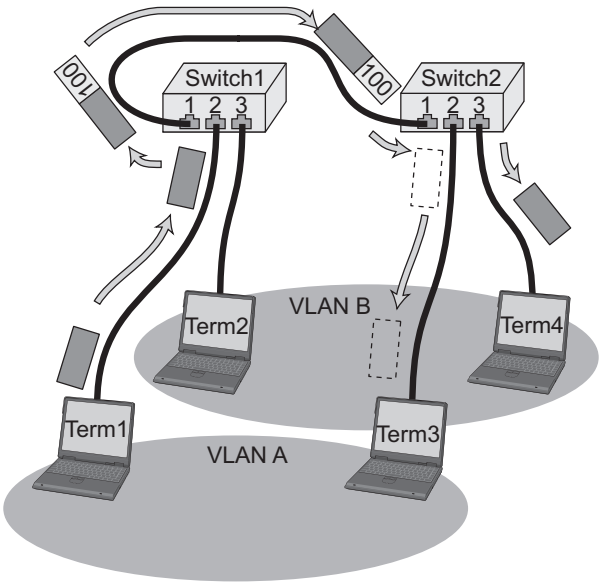

図 6.2　タグ VLAN の設定ミス

につれ困難になると同時に、確認誤りが生じやすくなる。特に最近では、階層的な VLAN を構成するために 1 つのパケットに複数のタグを同時に付加できる機能が利用されることもあり、その場合にはさらに困難さが増すと考えられる。このように、人手による設定の確認は、規模や正確さにおいて限界がある。

6.4.2　CTL の論理式

CTL（計算木論理）は状態遷移の分岐構造の性質を記述できる言語である。また、モデルの大きさと検査すべき論理式について線型時間でモデル検査を行う効率的なアルゴリズムが知られている。CTL の定義とその意味について述べる。

定義 1（CTL 論理式）　CTL の論理式は次の文法により定義される。

$$
\begin{aligned}
P ::= & \ ap \\
| & \ \texttt{!}\ P \mid P\ \texttt{\&}\ P \mid P \texttt{|} P \mid P\ \texttt{->}\ P \\
| & \ \texttt{AX}\ P \mid \texttt{AF}\ P \mid \texttt{AG}\ P \\
| & \ \texttt{EX}\ P \mid \texttt{EF}\ P \mid \texttt{EG}\ P \\
| & \ \texttt{A[}P\ \texttt{U}\ P\texttt{]} \\
| & \ \texttt{E[}P\ \texttt{U}\ P\texttt{]}
\end{aligned}
$$

論理式の形式的意味の定義は文献 [147] に譲り、本章では以降の議論に必要となる CTL

の論理式の直感的意味について述べる。CTL の論理式の真偽はモデルの状態に対し評価される。ap は原子論理式であり、変数 node、port、tag および phase を左辺とし、これらの変数の取り得る値を右辺とする等式である。! P、P & P、P | P および P -> P はそれぞれ否定、論理積、論理和、および含意である。AX P、AF P、および AG P の意味はそれぞれ、「任意の次状態において P が成り立つ」、「与えられた状態からの任意の状態遷移列において、将来 P が成り立つ」、および「与えられた状態からの任意の状態遷移列において、常に P が成り立つ」である。EX P、EF P、および EG P の意味はそれぞれ、「P が成り立つ次状態が存在する」、「与えられた状態からの状態遷移列で、将来 P が成り立つものが存在する」、および「与えられた状態からの状態遷移列で、常に P が成り立つものが存在する」である。A[P U Q] の意味は、「与えられた状態からの任意の状態遷移列において、将来 Q が成り立ち、Q が成り立つまでは P が成り立つ」である。E[P U Q] の意味は、「与えられた状態からの状態遷移列で、将来 Q が成り立ち、Q が成り立つまでは P が成り立つものが存在する」である。

6.5 提案手法の詳細

6.5.1 ネットワークのモデル

タグ VLAN を用いた Ethernet のモデルを構成する方法について述べる。まず構成しようとするモデルについて述べ、次にこれを自動的に構成する方法について述べる。モデルを構成するには、状態の表現方法、初期状態、および状態遷移規則を与える必要がある。

タグ VLAN の設定がパケットの転送経路にどう反映されるかを調べるのが目的であるため、パケットが状態を遷移するモデルを考える。パケットの状態を次の 4 つの変数を用いて表す。

- パケットが存在するノード (スイッチや端末) を表す変数 node
- パケットが存在するノード上のポートを表す変数 port
- パケットに付加されているタグを表す変数 tag
- パケットの移動状態を表す変数 phase

パケットにタグが付加されていない場合は変数 tag の値は null であるとする。変数 phase は incoming、outgoing および discarded のいずれかの値を持ち、それぞれパケットがノードに受信された状態、ノードから送信された状態、および破棄された状態を表す。

モデルの初期状態は、パケットが最初に送信されたときの状態を表し、どの端末から送信されたパケットの性質を検査するかによって決まる。初期状態はたとえば次のように、

```
node = Term1 &
port = 1 &
tag = null &
phase = outgoing
```

変数の値を指定することにより与える。これは、ノード Term1 のポート 1 からタグの無いパケットが送信された状態を表す。また、単に

```
node = Term1 &
phase = outgoing
```

のように一部の変数のみの値を指定してもよい。この場合は、初期状態の集合は Term1 から送信された任意のパケットの状態からなる。さらに、初期状態を全く指定しない場合は、任意の状態が初期状態になりうる。初期状態の指定は、どのような性質を検査するかにも依存するため、次節に述べる要求仕様記述と合せて考える必要がある。

パケットはネットワークの物理的トポロジ、すなわちケーブルの配線と、ノードの VLAN 設定に従って状態を遷移する。図 6.1 のネットワークにおける遷移の一例を図 6.3 に示す。左下のパケットはトポロジに従って端末 Term1 のポート 1 からスイッチ Switch1 のポート 2 に移動するため、変数 node および port がそれぞれ Term1 および 1 である状態からそれぞれ Switch1 および 2 である状態へ遷移する。このときタグは変化しないので、変数 tag の値は null のままである。変数 phase の値は、パケットが送信されようとしている状態を表す outgoing から受信された状態を表す incoming へ変化する。次に、パケットはスイッチ Switch1 の VLAN 設定に従って、ポート 2 からポート 1 へスイッチされ、VLAN A に対応するタグ 100 を付加される。このとき変数 node は変化せず、変数 phase は incoming から outgoing へ変化する。以上のように、パケットは変数 phase の値が outgoing の状態にあるときトポロジに従って遷移し、incoming の状態にあるときノードの VLAN 設定に従って遷移するものとする。また、変数 phase の値が discarded のとき、パケットは状態を遷移しないものとする。たとえば、端末 Term3 によって受信されたパケットは他の端末に転送されないため破棄されて discarded 状態に遷移し、その後は状態を遷移しない。この例ではパケットは端末 Term3 にしか到達しないが、一般に Ethernet のようなブロードキャスト型のネットワークではパケットは途中で複製されて複数の端末に転送される。たとえば、Switch2 がパケットを Term4 にも転

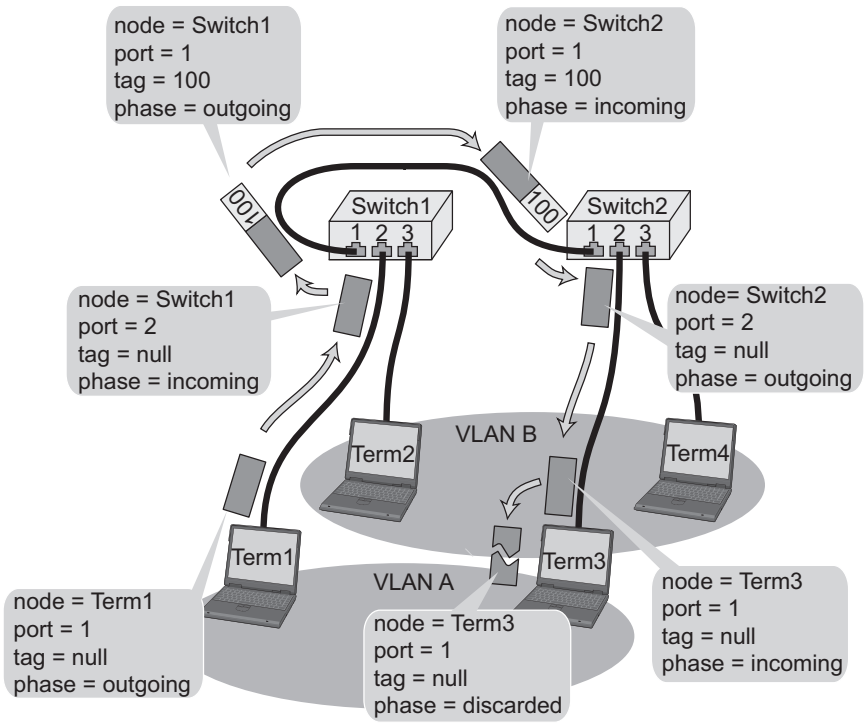

図 6.3　状態遷移列の例

送する場合が考えられる。この場合は、Switch2 が受信したパケットが、node = port2 である状態にも node = port3 である状態にも遷移可能であるとする。すなわち、パケットの複製をモデルの非決定性によって表現する。

本章で利用するモデル検査ソフトウェアではモデルを論理式によって定義される関係を用いて表現する。図 6.1 のネットワークのモデルは図 6.4 のように表現される。ここで、case 文

$$
\begin{array}{l}
\texttt{case} \\
\quad cond_1 : exp_1 ; \\
\quad cond_2 : exp_2 ; \\
\quad \vdots \\
\quad cond_n : exp_n ; \\
\quad \texttt{esac;}
\end{array}
$$

の真偽は、条件 $cond_1, cond_2, \ldots, cond_n$ に真であるものが存在しない場合は真であるとする。存在する場合はその最初のもの $cond_i$ に対応する exp_i の真偽をこの case 文の真偽とする。遷移後の状態における各変数 node、port、tag および phase の値を next(node)、next(port)、next(tag) および next(phase) で表し、可能な遷移は論理式を真とするような各変数の値によって表される。たとえば、図 6.3 の左下の状態で

```
1  case
2   phase = outgoing:
3    case
4     node = Switch1 & port = 1:
5      next(node) = Switch2 & next(port) = 1 & next(tag) = tag & next(phase) = incoming;
6     node = Switch1 & port = 2:
7      next(node) = Term1 & next(port) = 1 & next(tag) = tag & next(phase) = incoming;
       .
8      .
       .
9     node = Switch2 & port = 1:
10     next(node) = Switch1 & next(port) = 1 & next(tag) = tag & next(phase) = incoming;
       .
11     .
       .
12    node = Term1  & port = 1:
13     next(node) = Switch1 & next(port) = 2 & next(tag) = tag & next(phase) = incoming;
       .
14     .
       .
15    TRUE:
16     next(node) = node & next(port) = port & next(tag) = tag & next(phase) = discarded;
17   esac;
18  phase = incoming:
19   case
20    node = Switch1:
21     case
22      port = 1 & tag = 100:
23       next(node) = node & next(port) = 2 & next(tag) = null & next(phase) = outgoing;
24      port = 2 & tag = null:
25       next(node) = node & next(port) = 1 & next(tag) = 100 & next(phase) = outgoing;
26      port = 1 & tag = 200:
27       next(node) = node & next(port) = 3 & next(tag) = null & next(phase) = outgoing;
28      port = 3 & tag = null:
29       next(node) = node & next(port) = 1 & next(tag) = 200 & next(phase) = outgoing;
30      TRUE:
31       next(node) = node & next(port) = port & next(tag) = tag & next(phase) = discarded;
32     esac;
       .
33     .
       .
34    TRUE:
35     next(node) = node & next(port) = port & next(tag) = tag & next(phase) = discarded;
36   esac;
37  phase = discarded:
38   next(node) = node & next(port) = port & next(tag) = tag & next(phase) = phase;
39 esac
```

図 6.4　状態遷移規則の論理式

は、phase = outgoing であるので、3-17 行目の case 文が評価され、node = Term1 かつ port = 1 であるので、12 行目の条件が最初に真となり 13 行目が評価され、図 6.3 における次状態については確かに真となることがわかる。パケットは通常複数の転送先、すなわち複数のポートに転送される。この場合、exp_i に相当する箇所にいずれの転送先についても真となり、それ以外で偽となる論理式を書けばよい。たとえば、Switch1 のポート 1 でタグ 100 を持つパケットが受信されたとき、これがポート 2 だけでなくポート 4 にも転送される場合には、図の 23 行目の next(port) = 2 を論理和

(next(port) = 2 | next(port) = 4) で置換えればよい。

次に、ネットワークのモデルを自動的に構成する方法について述べる。モデルを構成するために必要な情報は、ネットワークの物理的トポロジと、各機器のタグ VLAN の設定である。

ネットワークの物理的トポロジとはすなわち、どのネットワーク機器のどのポートが他のどのネットワーク機器のどのポートと接続されているか、という情報である。図 6.1 のネットワークの場合のトポロジは表 6.1 のようなデータである。たとえば最初の行は、

表 6.1　トポロジのデータ

ノード	ポート	ノード	ポート
Switch1	1	Switch2	1
Switch1	2	Term1	1
Switch1	3	Term2	1
Switch2	2	Term3	1
Switch2	3	Term4	1

ノード Switch1 のポート 1 はノード Switch2 のポート 1 とケーブルで接続されていることを表す。本章ではトポロジのデータの取得方法について特に述べないが、既存のネットワーク管理ツールのデータや、ネットワーク機器が持っている MAC アドレスとポートの対応データベース (いわゆる FDB) からある程度自動的に作成することができると考えられる。また、既に発生している障害の原因を切り分けるためなどに本方法を用いる場合は、障害などにより稼働中のシステムのトポロジを自動的に作成することが困難な場合がある。この場合は、トポロジのデータをネットワークの設計仕様書などから作成することも考えられる。この場合、本方法による検査で障害が検出されないならば、原因はネットワーク機器の設定ではなく、設計仕様書のトポロジと実際のトポロジとの不整合 (配線ミス、断線など) などであることがわかる。

トポロジのデータは、状態遷移モデルのトポロジに従う遷移規則に変換される。たとえば表 6.1 のデータは図 6.4 の 2-17 行目に変換される。パケットはケーブルを双方向に移動するため、トポロジのデータの各エントリはモデルの 2 つの遷移規則に対応する。たとえば表 6.1 の最初のエントリは、図 6.4 において Switch1 から Switch2 への移動を表す 4-5 行目の遷移と、逆方向の移動を表す 9-10 行目の遷移に変換される。この変換はテキスト処理によって実現可能である。

各機器のタグ VLAN の設定は、機器の設定ファイルの一部である。機器の設定ファイ

```
1  create vlan "VLAN-A"
2  create vlan "VLAN-B"
3
4  configure vlan "VLAN-A" tag 100
5  configure vlan "VLAN-A" add port 1 tagged
6  configure vlan "VLAN-A" add port 2 untagged
7
8  configure vlan "VLAN-B" tag 200
9  configure vlan "VLAN-B" add port 1 tagged
10 configure vlan "VLAN-B" add port 3 untagged
```

図 6.5　エクストリームネットワークス社の製品の VLAN 設定例

```
1  vlan database
2    vlan 100 name VLAN-A
3    vlan 200 name VLAN-B
4
5  interface ethernet 1/1
6  switchport allowed vlan add 100,200 tagged
7
8  interface ethernet 1/2
9  switchport allowed vlan add 100 untagged
10 switchport native vlan 100
11
12 interface ethernet 1/3
13 switchport allowed vlan add 200 untagged
14 switchport native vlan 200
```

図 6.6　エスエムシーネットワーク社の製品の VLAN 設定例

ルは通常 TFTP などのプロトコルで取出せることが多い。また、`telnet` コマンド、シリアルコンソール、web ブラウザなどから管理用のインターフェースにログインして表示することもできるため、いずれにせよ自動的に取得することができる。タグ VLAN の設定データの例を図 6.5 および図 6.5.1 に示す。これらはそれぞれエクストリームネットワークス社の製品とエスエムシーネットワークス社の製品の例で、両方とも図 6.1 のネットワークにおける Switch1 の動作をさせるための設定である。 図 6.5 ではまず VLAN A および B を宣言し (1-2 行目)、次にそのそれぞれについて (4-6 行目および 8-9 行目)

VLAN 識別用のタグ (それぞれ 100 および 200) および各 VLAN に属するポートが設定されている。各ポートの設定に tagged または untagged の指定がある。tagged の指定があるポートでは、VLAN のパケットは、タグのついた状態で送受信される。untagged の指定があるポートでは、VLAN のパケットは、タグの無い状態で送受信される。タグの無い状態は、本章のモデルではタグ null が付加された状態に相当する。図 6.5.1 の例でも同様に VLAN を宣言 (1-3 行目) し、今度はポート (interface) ごとにポートがどの VLAN に属するかを設定する。図 6.5 および図 6.5.1 のいずれの設定においても、ポートとタグの組が与えられたとき、それがどの VLAN に属するか、あるいは全く属しないかを判断することができる。ネットワーク機器は、この情報をもとにパケットの転送を行っている。たとえば、ポート 2 からタグの無いパケットを受信した場合には、このパケットが VLAN A に属することがわかり、同じ VLAN に属する全てのポートとタグの組を転送先として選ぶ。この場合はポート 1 とタグ 100 の組のみであるので、タグを 100 に書換えてポート 1 から送信する。

タグ VLAN の設定データは、状態遷移モデルの VLAN 設定に従う遷移の規則に変換される。たとえば図 6.5 あるいは図 6.5.1 のデータは、図 6.4 では VLAN 設定に従う遷移を記述した部分である 18-36 行目のうち、Switch1 の設定に関する部分である 20-32 行目に変換される。各遷移規則は、ネットワーク機器が行う動作をそのまま記述すればよい。たとえば、VLAN A に関しては 22-23 行目および 24-25 行目において 2 つの遷移規則が記述されている。前者はポート 1 からタグ 100 が付加されたパケットを同じ VLAN に属するポート 2 からタグ無しで転送する動作である。後者はその逆の転送である。このような VLAN 設定のモデルへの変換もテキスト処理によって実現可能である。

なお、ケーブルの接続されていないポートからパケットが送信された場合や、パケットが他のポートに転送されない場合は、パケットは破棄される。これは、破棄状態 (phase = discarded) への遷移によって表現される。図 6.4 の 15-16 行目および 30-31 行目などがこれに相当する。37-38 行目は破棄状態のパケットが同じ状態に留まり続けることを表している。これは、モデル検査ソフトウェアでは全ての状態について次状態が存在せねばならないという前提条件があるためである。

本章では、1 つのパケットには高々 1 つのタグが付加されるタグ VLAN を考えたが、タグをタグの配列に置き換えたモデルを構成することも容易である。この拡張により、1 つのパケットにある上限以下の複数個のタグが同時に付加されるような、拡張されたタグ VLAN にも本方法は応用可能となる。

6.5.2 ネットワークに期待される性質の記述

ネットワークの要求仕様、すなわちネットワークに期待される検査すべき性質を、CTL の論理式を用いて記述する方法について例を用いて述べる。

本研究で用いる検査ツール NuSMV では、モデルの初期状態を様相記号を用いない論理式として記述し、その初期状態について、将来どのように遷移するかを CTL の論理式で記述する。本研究でもこれに倣って記述する。なお、NuSMV ではモデルの初期状態は論理式を満たす複数の状態を同時に考えて検証を行うことができる。

例 4 端末 Term1 から送信したパケットが端末 Term3 に到達するという、通信が可能であることを意味する性質を考える。この場合、まず、パケットが最初に端末 Term1 上に存在することをモデルの初期状態として、

```
node = Term1
```

のように指定しておき、CTL の論理式として

```
EF (node = Term3)
```

を記述する。

これは、Term1 のポート 1 で送信フェーズにあるパケットを初期状態とし、この状態から始まる状態遷移列で将来ノードが Term3 になるものが存在するということになる。ここで指定した初期状態には、タグ、ポート番号、フェーズが異なる複数の状態が含まれる。このため、CTL の論理式はこれらの状態全てについて性質が成り立つという主張と解釈される。

本方法のモデル化では、パケットの複製をモデルの非決定性によって表現するため、複製された全てのパケットを表す AF でなく、複製されたパケットの 1 つを表す EF を用いる。別のモデル化として、モデルの状態が複製されたパケットの集合を表すようにモデル化する方法も考えられる。この場合は、AF を用いて、どの遷移列においても将来必ずパケットの集合に端末 Term3 のパケットが含まれる状態に遷移する、という記述のしかたが可能である。ただし、このモデル化の方法では、特に Ethernet のようにパケットの複製が頻繁なネットワークではモデルの状態数が多くなり、効率的でない場合がある。

例 5 端末 Term1 から送信したパケットが他の VLAN に属する端末 Term3 に到達しない、という性質は、前の例の CTL の論理式の否定

```
! EF (node = Term3)
```

によって記述される。これはパケットの漏洩が起こらないというセキュリティ上重要な性質を表す論理式である。

例 6 パケットがループしない、という性質は次の論理式で記述される。

```
AF (phase = discarded)
```

これは、パケットは任意の状態遷移列について、将来必ず廃棄フェーズに遷移するということを意味する。本方法ではパケットの廃棄は廃棄状態への遷移として明示的にモデルの中で表現されるため、このように記述できる。ここでは、初期状態を指定していない。すなわち、任意の状態を初期状態として、上の論理式が成り立つという性質を表している。Ethernet のようなブロードキャスト型のネットワークの場合、パケットのループが発生するとネットワークが輻輳し通信速度の低下や通信断絶を招く。このため、パケットのループの検出は運用上重要な性質である。

例 7 ネットワーク上に、機密情報を保管しているノード `credential` およびこのノードへのアクセスを監視するノード `audit` が存在するとする。このとき、`credential` に届くパケットは必ず `audit` により監視される、すなわち `audit` に届く、という性質は以下の論理式で記述される。

```
EF (node = credential)
-> EF (node = audit)
```

このように、`EF` などの時相演算子を複数組み合わせてネットワークに関する様々な性質を CTL で記述できる。

本研究では、ネットワークの要求仕様を CTL の論理式を用いて記述したが、CTL のかわりに他の言語を用いて記述することも考えられる。この場合、言語の記述能力と記述した性質のモデル検査の効率について考慮する必要がある。一般に両者はトレードオフの関係にある。すなわち、記述能力が高い言語ほどモデル検査の効率は悪くなる。また、両者を考慮してある性質に関しては CTL を用い他の性質に関しては他の言語で記述するという利用法も考えられる。

記述能力については、通信が可能であること (例 5)、パケットが漏洩しないこと (例 5) およびループが無いこと (例 6) に関しては CTL で記述できると同時に、システム検証の際に CTL と同様によく用いられる LTL でも記述できる。しかし、例 7 のようなパケッ

トの複製による分岐構造に関する性質を表現することは、提案方法で用いているモデル上ではできない。逆に、パケットの転送経路の 1 つを線として見るような性質についてはCTL のかわりに LTL を用いなければならない場合があると考えられる。

モデル検査の効率については、CTL も LTL もともに記号モデル検査により高速に検査できることが知られている。特に CTL のモデル検査の計算量はモデルと論理式の大きさについて線型時間であることが知られている [148]。

6.5.3 VLAN 設定の検査

ネットワークの性質を検査するためには、ネットワークのモデルの論理式 (第 6.5.1 節) および検査する性質を表す第 6.5.2 節) を入力としてモデル検査ソフトウェアを実行すればよい。

モデル検査ソフトウェアは、モデルの初期状態における CTL の論理式の真偽を判定し、その結果を出力する。たとえば図 6.1 のネットワークのモデル (図 6.4) に対して、第 6.5.2 節の例 5 の論理式を入力して検査を行なうと、この性質は真 (`true`) であるので、出力は次のようになる。

```
-- specification
   !EF node = Term4
   is true
```

「パケットが到達しない」というような性質を検査し、この性質が偽である場合には、モデル検査ソフトウェアは反例 (`Counterexample`) として、パケットが到達するような状態遷移の列を出力する。例 5 の性質を同じモデルについて検査すると、出力は図 6.7 のようになる。ただし、読みやすさのために出力を適宜編集している。ここでは、ノード `Term1` のポート `1` からタグの無いパケットが送信されようとしている初期状態 (`State: 1.1`) から、変数 `node`、`port` および `phase` が変化して次状態 (`State: 1.2`) に遷移する。以降遷移の繰り返しによりパケットが `Term3` に到達するまでの全ての遷移が記述されており、経路およびタグの書換えがどのように行われたかわかる。

パケットの漏洩やそれに類似の問題が発生した場合には、原因の特定のために、パケットがどのような経路を通り漏洩したのか知る必要がある。反例の出力はまさにパケットの経路を出力するものであり、障害の原因の特定に役立つ。また、モデル検査ソフトウェアには与えられた初期状態から可能な状態遷移を対話的に選択してシミュレーションを行なう機能もあり、原因の特定などに役立てることが可能である。

```
-- specification
   !EF node = Term3
   is false
-- as demonstrated by the following
   execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
->State: 1.1<- node=Term1    port=1  tag=null  phase=outgoing
->State: 1.2<- node=Switch1  port=2            phase=incoming
->State: 1.3<-               port=1  tag=100   phase=outgoing
->State: 1.4<- node=Switch2                    phase=incoming
->State: 1.5<-               port=2  tag=null  phase=outgoing
->State: 1.6<- node=Term3    port=1            phase=incoming
```

図 6.7　到達性の反例

6.5.4　性能評価

前述したように CTL のモデル検査では線型時間のアルゴリズムが知られており、高速に検査を実行することができる。本節では、検査に必要な時間の定量的な目安を得るため、人工的にランダムに構成したネットワークのデータについて検査時間を測定する。

検査を行なうネットワークはフルメッシュ型でランダムに構成したものである。すなわち、全てのノードのポートがランダムに選ばれたいずれかのノードのいずれかのポートとケーブルで接続されているとする。さらに、パケットのスイッチングとタグの書き換えもランダムに行われるとする。すなわち、任意のポートで受信した任意のパケットについて、ランダムに選ばれたポートへランダムに選ばれたタグを付けて転送する。

このネットワーク上で、あらかじめ決められた 2 つのポート間でパケットが到達しないという性質および、ループが存在しないという性質を検査する。以下では 1 ノードあたりのポート数を 24、使用するタグの数を 20 にした場合の検査時間を示す。モデル検査ソフトウェアとして NuSMV のバージョン 2.2.2 を用いた。ハードウェアとして、インテル社の Xeon プロセッサの 3.20GHz およびメモリを 4GB 搭載した計算機を用いた。

あらかじめ決められた 2 つのポート間でパケットが到達しないという性質 (到達不能性) を検査するために初期状態として

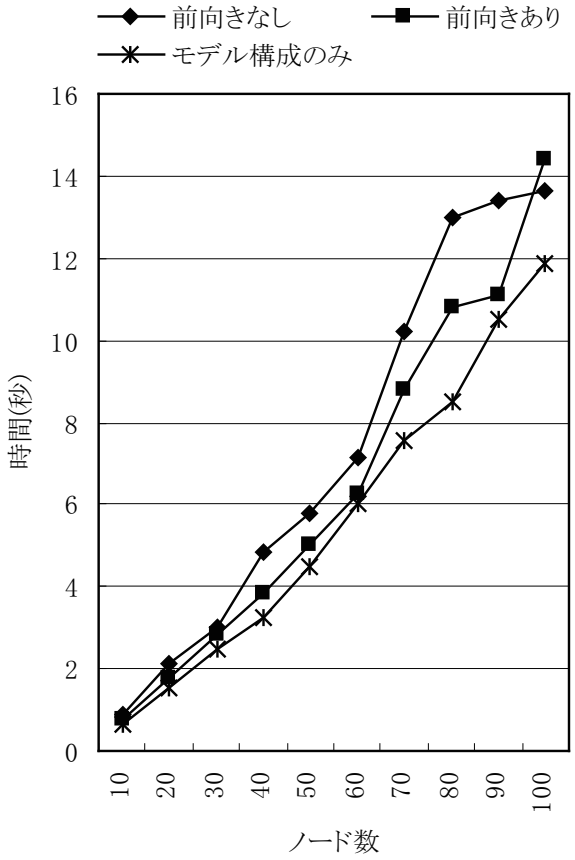

図 6.8　到達不能性の検査時間

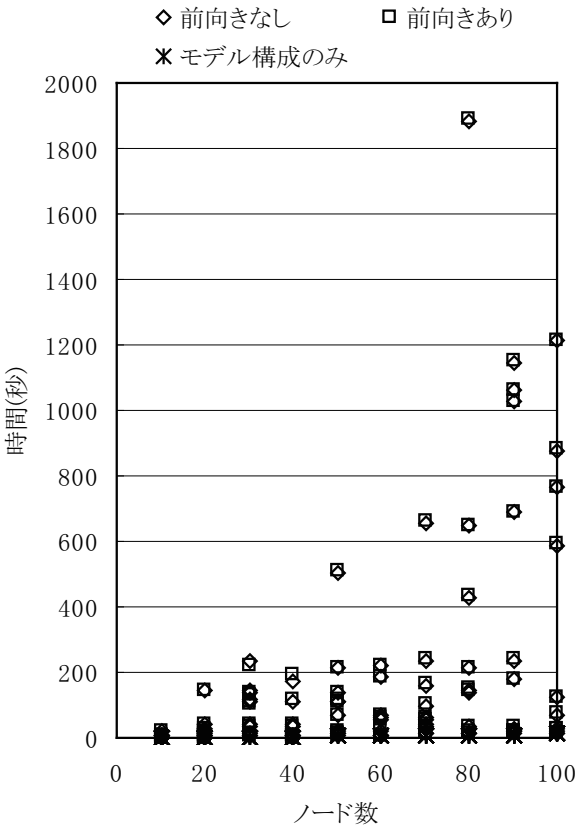

図 6.9　無ループ性の検査時間

```
node = 0 & port = 0
```

を指定し、要求仕様として

```
! EF (node = 1 & port = 0)
```

を指定する。これは例 5 と類似の性質である。検査時間を図 6.8 に示す。前向き探索を行なわずに検査した場合、前向き探索を行い検査した場合、および、検査を行わずその前処理であるモデルの内部表現の構成のみを行った場合の検査時間を示した。前向き探索とは、あらかじめモデルの状態を初期状態から到達可能なものに制限する処理で、NuSMV で-f オプションを使用する場合に相当する。モデルの状態数に対して、初期状態から到達可能な状態の数が小さい場合には、前向き探索を行なうと検査時間を短くすることができる。図に示した結果においても、その効果が表れている。いずれの場合も、検査は十数秒で完了している。

ループが存在しないという性質 (無ループ性) を検査するために、要求仕様として

```
AF (phase = discarded)
```

を指定した。また、ネットワーク上で作られる全てのパケットについてループが起らないことを示すために、全ての状態が初期状態になりうるとする。このため、初期状態を特に指定せずに検査する。これは例 6 と同じ性質である。評価に用いたネットワークはフルメッシュの構成であるため、この性質は成り立たず、反例が出力される。検査時間を図 6.9 に示す。ネットワークの構成によって検査時間にばらつきがあるため、異なる 10 個のネットワークについて時間を測定し、その値をプロットした。この場合では、前向き探索を行なう場合と行わない場合では検査時間がほとんど同じである。これは、全ての状態が初期状態になりうるため、前向き探索によって状態を減らすことができないためであると考えられる。検査には最大で約 30 分の時間を要するが、現実的に利用可能な範囲であると考えられる。また、実際に利用するネットワークを構成する際には、ツリー状を基本としてネットワークを構成することが多い。すなわち、測定に用いたネットワークは通常用いられるネットワークよりも検査にはるかに多くの時間が必要な例である。ツリー状のネットワークで行った同様の実験では、ノード数が約 2800 台の場合でも約 15 分で計算が完了する。これは約 470 台の 8 ポートスイッチを 4 段のツリー状に接続し、その下に残りのノードを端末として接続したネットワークである。

6.6 関連研究および今後の課題

本研究の手法を提案して以降、SDN（Software-Defined Networking）と呼ばれる、ネットワークの設定をプログラムにより変更し、VLAN を含むネットワークの機能を柔軟に設定、変更する技術が出現してきた。SDN に対しても形式手法の適用が提案されておきており、その主な手法の中には本研究で行なった手法と極めて近いものがある。このため、本章では SDN へ形式検証を適用する既存研究の主なものを取り上げ、本研究で提案した手法との対比および位置づけについて述べる。

6.6.1 Software-defined networking（SDN）の概要

SDN の概要について述べる。SDN には様々な形態があるが、ここでは文献 [149] を参考に、SDN のうち、形式検証について検討するために必要な共通部分に的を絞って述べる。SDN には決まった定義は無いが、ソフトウェアによって制御されるネットワークで、

図 6.10 のように制御層（またはコントロールプレーン）とインフラストラクチャ層（またはデータプレーン）に分離されており、制御層における 1 つの制御デバイス（SDN コントローラ）でインフラストラクチャ層の複数のネットワーク機器（スイッチ）を制御するもの、と説明されることが多い。また、制御層も API を通じてさらにその上のアプリケーション層から、ソフトウェアやオペレータの要求により動作する。

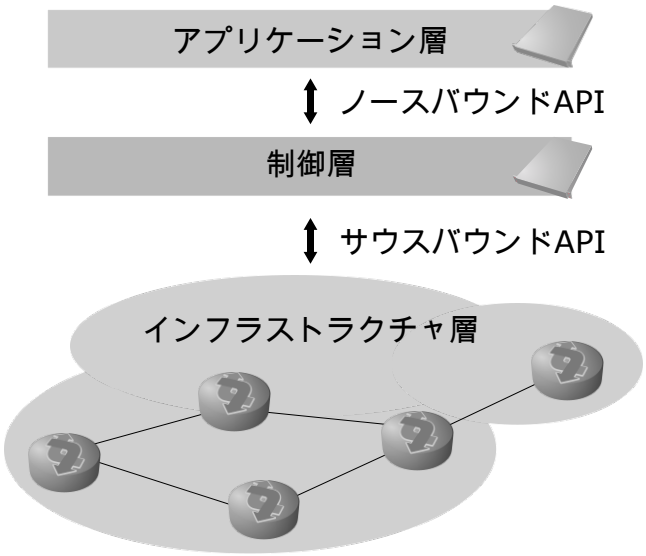

図 6.10　SDN の模式図（文献 [3] の図をもとに作成）

SDN コントローラはサウスバウンド API を通じ、スイッチにパケットの転送ルールを書き込むことでインフラストラクチャ層を制御する。サウスバウンド API として、OpenFlow[150] などのプロトコルが用いられる。OpenFlow では、SDN コントローラがスイッチに対して指定のパケットを送信させるためのメッセージや、逆にスイッチが受信したパケットを SDN コントローラに転送するためのメッセージが規定されている。これらのメッセージにより、SDN コントローラがスイッチを直接的に制御して、インフラストラクチャ層のトポロジ検出や遅延測定のためのパケットを送受信させることができる。さらにアプリケーション層のソフトウェアはノースバウンド API を通じて制御層にリクエストを送る。ノースバウンド API については OpenFlow の仕様を策定している Open Networking Foundation で議論されていたようだが、現状では標準的なプロトコルは無く、SDN コントローラの実装に依存する。SDN を用いない従来型のネットワーク構築との対比で言えば、従来型のネットワーク機器、特に IP ルーターはコマンドラインインタフェースなどを用いてオペレータにより設定が行われていた。また、ルーティングプロトコルを通じて得た周辺のネットワークからの経路情報を集積し、それを元にパケットを受

信した際のアクション（転送ルール）を計算して決定していた。SDN では、転送ルールに基づいてパケットを転送する処理のみを転送レイヤの機器に行わせ、ルーティングプロトコルを用いた経路情報の集積や、それを元にした転送ルールの計算などの負荷が大きい処理は SDN コントローラに担わせる。

SDN が利用されるケースとして、データセンタ内のネットワーク運用管理がある [151]。データセンタ内では多数のユーザやサービスを同一の物理インフラに収容するため、サーバやサーバ間のネットワークを仮想化する。仮想ネットワークの設定変更の度に、物理的なネットワーク機器の設定をコマンドラインインタフェースにより手作業で変更するのは非効率であるため、SDN による集中制御が用いられる。また、データセンタの構成によりアプリケーションレイヤから制御レイヤを通じて柔軟にサービスの提供やそれに伴うネットワーク機器の変更をまとめて行える利点がある。また、SDN コントローラや転送レイヤのスイッチを低いコストで構成することもできる [152]。SDN コントローラの実装として、オープンソースソフトウェア（OSS）として公開されているものを利用することができる。転送レイヤのスイッチとして、汎用の部品を組み合わせて設計・製造され、スイッチ専用のオペレーティングシステム（スイッチ OS）を搭載しない状態で出荷される、ホワイトボックススイッチまたはベアメタルスイッチと呼ばれる比較的安価なネットワーク機器を用いることができる。スイッチ OS として OSS として公開されているものを利用することができる。

その他のケースとして、広域のネットワーク（WAN）におけるトラフィックの効率化や測定に SDN を用いるケースや、通信事業者によって、SDN 技術を用いた仮想ネットワークサービスが SD-WAN などの名称によって提供されるケースがある。さらに、5G などの無線ネットワークをサービスに応じて仮想的に分割して提供する SD-RAN と呼ばれる技術の要素技術にもなっている。

SDN はソフトウェアによりネットワークを制御・管理できるようにする、いわばネットワークをプログラム可能にする仕組みである。ネットワークのプログラム可能性を高める方向性として、転送レイヤにおけるスイッチ内のパケット処理をより低いレベルでプログラム可能にするための、P4（Programmable Protocol-Independent Packet Processors）言語 [153] がある。高速なスイッチにおいて、パケットは多段のパイプライン処理を並列に実行して処理される。P4 はパイプラインの処理をより抽象的なレベルで記述することを可能にし、P4 のプログラムはコンパイラにより物理的なスイッチのパイプランで実行可能になる。P4 によりプログラム可能なスイッチ用チップが商用化されている。

SDN とセキュリティの関係については他の文献（[154] 等）に譲るが、SDN への攻撃の観点と、SDN を利用して侵入検知システムなどの機能を実現する観点がある。

6.6.2 SDN への形式検証の適用

本章の研究で提案した手法は SDN 技術が現れる前に提案したものであるが、SDN の主な形式検証手法と類似している。このため、SDN へ形式検証を適用する既存研究の主なものを取り上げ、本章で提案した手法との対比および位置づけについて述べる。対比および位置づけについて述べることが目的であるため、網羅的な紹介は他の文献 [155, 156, 157] に譲る。

SDN への形式検証の適用は、制御層の検証とインフラストラクチャ層の検証に大別できる。後者のほうが本章の手法に近いため、先に後者について述べる。また、P4 言語によってプログラムされたインフラストラクチャ層の検証については別途述べる。

インフラストラクチャ層の検証

SDN においても、VLAN と同様にスイッチの設定の不整合などのミスによって、パケットの消失などの障害が起こり得る。このような設定ミスが起こらないことを検証するのがインフラストラクチャ層の検証である。

SDN ではトポロジの情報を SDN コントローラが保持することが前提といえる。さらに、SDN コントローラによって設定されたスイッチの転送ルールを取得することができる。このため、本研究におけるネットワークのモデル化のための情報を容易に収集することができる。その一方で、SDN ではイーサネットだけでなくそれに含まれる IP パケットなど様々な種類のパケットを扱う可能性があるため、これに対応した検証方法が必要となる。

本研究の方法と極めて類似した検証手法として、SDN コントローラが持つ情報を元にパケットの状態遷移モデルを作り到達性などの性質を計算木論理で記述して検証する方法として、FlowChecker[158] がある。検証のためにモデル検査を用いるところも共通している。

また、それ以前に提案された手法として Anteater[159, 160] がある。Anteater ではパケットの状態遷移モデルおよび、発見したい欠陥（ループの検出、パケットの消失、設定の不整合）を命題論理の論理式で記述し、それらの充足可能性を SAT ソルバで判定することで欠陥の有無を調べる。本研究の方法を IP ネットワークに拡張することは可能だが、参照するヘッダの情報を予めモデル内の変数としてのモデルに組み込んでおく必要がある。このため、パケットのフォーマットが変更になったり、新たにヘッダが追加されたりした場合にはモデルの構成法を拡張する必要がある。これを避けてより一般的にヘッダ

の情報を扱える方法として、Header Space Analysis[161] と呼ばれる手法がある。この手法では、パケットの集合をそのヘッダのビットパターンにより表現する。これは文字列の集合を正規表現により表現することと類似しており、正規表現と同様に和、積、補集合、差の演算が可能である。トポロジと転送ルールの情報から、ビットパターンの状態遷移モデルを構成し、到達可能性やループの検出、VLAN と同様の仮想的な分割（スライス）における漏洩の検査などを行うことができる。

ここまでに紹介した手法は、当然ではあるがネットワークの規模が大きくなるにつれて検証に要する時間が大きくなるため、設定変更時に瞬時に検証を行うことが難しくなる。これに対応するために NetPlumber[162] や VeriFlow[163] では、ネットワークが満たすべき性質を不変条件として定義しておき、設定の変更の差分をもとに不変条件が保たれることを検証する。このような差分に基づく検証は、汎用のモデル検査ツールを用いる本研究の方法や FlowChecker では難しい。

制御層の検証

制御層の検証では、SDN コントローラがアプリケーション層からの入力に対してスイッチの設定を適切に行い、インフラストラクチャ層に正しい動作をさせることを検証する。動作の正しさ、すなわち検証すべき性質を定義することができれば、それにインフラストラクチャ層の検証を行えばよい。

Kuai[164] は検証者が与えた SDN コントローラのプログラムおよび検証すべき性質に基づいて、制御層とインフラストラクチャ層を 1 つのシステムとしてモデル化し、検証を行う。サウスバウンド API の通信とインフラストラクチャ層の通信はともにメッセージとして抽象化され、SDN コントローラとスイッチは非同期でメッセージを交換する有限状態機械としてモデル化される。これらの有限状態機械の集まりの動作列が与えられた性質を満たすことを検証する。SDN コントローラの動作をモデルの一部に織り込み、さらに非同期な通信を許しているため、SDN コントローラが複数のスイッチの設定を変更する際に変更が反映される順序によって起こりうる不具合も発見できる。この方法は、本章の方法において状態遷移モデルを SDN コントローラの動作も織り込んで構成することに対応するが、モデルの状態数が極めて大きくなり、そのための工夫が必要なことが想定される。Kuai では、並行システムの検証の分野で用いられる、partial order reduction と呼ばれる状態数削減手法を用いている。

ここまで紹介してきた方法は全て、トポロジが具体的に決まっている場合に適用できる方法であるが、特定のトポロジによらない形で SDN コントローラのプログラムを検証する手法もある。Guha ら [165] は、アプリケーション層からの入力を OpenFlow のコマン

ドに翻訳するような SDN コントローラを考え、その動作を検証する方法を提案した。アプリケーション層からの入力は NetCore と呼ばれる言語で記述されるプログラムであり、各スイッチがパケットを受信したとき取るべきアクション（どのポートにパケットを送るか、その際の条件など）を指示するものとした。SDN コントローラはこれを、OpenFlow のコマンド列に変換して出力する。そして、SDN コントローラの出力によりスイッチを設定したとき、インフラストラクチャ層が NetCore 言語のプログラムの指示どおりに動作することを検証する。このために必要な、NetCore 言語の形式的意味、SDN コントローラの変換規則、および OpenFlow のコマンド列の意味をある種の論理式として汎用の定理証明システム Coq[25, 26] の上で記述し、その関係を証明することで検証が行われる。このような一般的な定義に基づく検証を自動的に行うことは困難なため、検証は Coq 上で人手により証明を記述することで行われ、証明の一部の自動化と正しさの確認を Coq の機能として行うことができる。さらに、Coq で記述された証明から、OCaml[166] 言語のプログラムを抽出することができ、厳密な証明つきの SDN コントローラソフトウェアが得られる。

P4 によってプログラムされたインフラストラクチャ層の検証

P4 によってプログラムされたインフラストラクチャ層の検証は、プログラムの検証の側面が強く、手法としては SDN コントローラの検証と共通するため分けて述べる。

プログラミング言語としての P4 は C 言語に似た構文を持つ手続き型言語である。プログラミング言語としては C 言語と異なりポインタもループもないため、既存のプログラム検証手法は適用しやすいといえる。P4 で記述されるパケット処理のうち基本的なものを実行順に並べると、ヘッダのデータ型（構造体）の宣言に基づく構文解析、チェックサムの検証、テーブルを参照した条件分岐によるヘッダの書き換えと廃棄または送信ポートの指定、送信ポートに依存したヘッダの書き換え、チェックサムの再計算、書き換えたヘッダのビット列への変換がある。いずれも処理としては単純なものである。なお、テーブルは P4 のプログラムとしては記述されず、SDN コントローラから与えられるものを参照する。

P4 のプログラムのバグとして、たとえば、条件分岐の書き間違いにより、全てのパケットを対象としたフィルタのためのテーブルを一部のパケットに適用してしまう、あるいは、ヘッダの構文解析に失敗した場合に適用されない、SDN コントローラが与えるテーブルに誤記がある、あるいは、複数テーブルが与えられる際に不整合がある、など、多岐にわたり、また、不注意によって大きな影響がでる場合がある。この他、複数のスイッチを P4 でプログラムした際の全体としての不整合といったインフラストラクチャ層全体の

性質も検討すべきだが、そこまで検証を行った研究を筆者は知らない。このため、ここでは単体のスイッチを P4 でプログラムする際の検証手法について紹介する。

C 言語のプログラム検証手法を利用して P4 プログラムを検証する方法 [167] が提案されている。この方法では、P4 言語を拡張し、検証したい性質を P4 プログラム中にアサーション（assert 文）として埋め込めるようにした。そして、このプログラムを SDN コントローラから与えられたテーブルの情報も付加して C 言語のプログラムに変換し、プログラム解析手法の一つである記号実行によりアサーションとして記述された性質が成り立つことを検証する。

この方法ではテーブルの情報も付加して検証を行うため、P4 のプログラムをコンパイルする際ではなく、実行する際に検証を行う必要がある。その一方、コンパイル時の検証を可能にするため、p4v[167] ではテーブル参照による条件分岐を非決定的分岐（いずれも起こりうる分岐）として扱う方法を提案した。さらに、テーブルに対する仮定を記述できるようにして、非決定的分岐を制約して現実に即した検証を行い、SDN コントローラから送り込まれるテーブルの満たすべき条件を明示できるようにした。この方法では、アサーションつきプログラムをガード付きコマンド言語（GCL）[168] に翻訳し、GCL の解析手法を適用する。

ここまで紹介した手法は、いずれも P4 のプログラムを他の言語に翻訳する。その際に問題となるのが、P4 のプログラムの意味が形式的に厳密に与えられていない点である。P4 言語に形式的に意味を与える研究 [169] も行われている。

6.7 まとめと今後の課題

本研究では、タグ VLAN を用いる LAN について、ネットワーク機器の配線および設定の情報から状態遷移モデルを構成する方法、および、ネットワークに期待される性質を様相論理により統一的に記述する方法を提案し、さらに汎用のモデル検査ツールを用いて設定の正しさを自動的かつ高速に検証する方法を提案した。また、要求を満たさない場合に、その具体的を発見できることも示した。

第 6.2 節で述べた通り、Guttman らによるファイアウォールおよび IPsec についての研究では、設定を自動的に導出するアルゴリズムを提案していた。その一方、本研究では与えられた設定の正しさを検証するにとどまっている。タグ VLAN の場合にも自動的な導出がある程度可能であると考えられるが、実際のネットワークでは、トラフィックの集中の回避などを考慮するため、これを表現できるようモデルを拡張する必要があると考えられる。そのような拡張と自動的導出は今後の課題である。

関連研究として概観した、SDN への形式検証への適用からは、次のような課題が導かれる。

- SDN が利用される主な領域として、データセンタと、通信事業者のネットワークがある。これらのネットワークで要求される性質は本研究で扱ったパケットの到達性やループの不存在などの比較的単純な性質だけでなく、通信経路の容量がトラフィックに対して十分であるか、パケットの優先度が適切に設定されるか、遅延が一定以内に抑えられるか、など多岐にわたる。また、5G の通信網の仮想化・オープン化に SDN を利用する場合は無線通信装置の制御まで行うため、そもそもネットワークが満たすべき性質を書き下すことが容易ではない。パケットの優先度や経路の容量については、既存研究を拡張することもある程度は可能と考えられる。しかし、モデルの状態数が巨大になると考えられ、実用上十分な速度での検証は難しくなるため、使用されるスイッチや転送ルール、トポロジにある種の一様性・対称性を仮定して、それを元に状態数を削減するなどの工夫が必要になると見られる。
- 大規模で複雑な SDN や、アプリケーション層からの要求による設定変更が頻繁な SDN では、複数のスイッチの設定変更が必要な処理を行う際の変更順序を適切に行う必要性が高くなる。手法としては Kuai のような非同期通信による設定変更でモデル化できるが、特に順序が重要になるようなケースに的を絞って高速に検証を行い、適切な変更順序を提示する手法が望ましい。
- スイッチや通信路などのリソースの効率的かつ耐障害性のある利用のためには、SDN コントローラが評価関数に基づいて自動的に転送ルールを設定することが必要になる。これをネットワークの検証と合わせると、検証すべき性質を制約として与える制約付きの最適化問題となる。さらに、あらかじめトラフィックを確定的に予測することはできないため、SDN の機能を用いてトラフィックや遅延 を計測し、それをもとに確率モデルを構成した確率的な最適化となる。SDN の機能、特に P4 がもたらすプログラム可能性を用いてネットワークを計測し、それをフィードバックとしてネットワークをクローズドループ制御する、という考え方がすでに提案されている [170]。
- P4 プログラムの検証の研究はいずれもスイッチ単体を対象としていたが、もちろんインフラストラクチャ層全体の検証ができることが望ましい。インフラストラクチャ層の全体を検証する手法と組み合わせて用いることが考えられる。ただし、スイッチの挙動が転送ルールなどの単純な形でモデル化できない場合は別のモデル化手段が必要になる。

- セキュリティの観点からは、SDN によりネットワーク機能を仮想化してユーザに提供する場合、ユーザはインフラストラクチャ層の一部を利用した通信と監視、および、場合により SDN コントローラへの入力が可能になる。この場合、これらの操作が悪意を持って行われることを想定する必要がある。たとえば、あるユーザが SDN コントローラに入力を与えた際、それが他のユーザに与える影響が限定的である、あるいはその入力により生じたネットワーク変化から、他のユーザの通信についての情報が得られない、などの性質を検証できることが望ましい。ごく単純なケースでは、Kuai のような並行プロセスモデルにおける非干渉性（non-interference）として定式化できる可能性がある。

SDN によるネットワーク管理は今後ますますその利用が拡大すると考えられる。ネットワークのプログラム可能性の向上に伴い、形式検証の重要性が高まっていると考えられるが、どのような形で適用すれば最も効果的であるかは、SDN の具体的な事例を元により詳細に検討することで見えてくると考えられる。

第 7 章

結論

7.1 本論文の貢献

本論文ではまず、情報システム一般に適用可能な形式検証手法および、暗号プロトコルの検証に特化した形式検証手法ついて概観した。次に、ネットワークサービスを支える暗号プロトコルおよびネットワーク機器の設定への形式検証の適用について述べた。具体的には次の検証事例および検証方法の提案について論じた。

- ネットワークを通じ、クレジットカードを用いた電子決済のための暗号プロトコル SET における、クレジットカード番号の秘匿性の検証
- トランスポートレイヤにおいて通信相手の認証と通信内容の秘匿性を保証するための暗号プロトコル QUIC について、先行研究で提案されていた QACCE 安全性の定義に基づいた検証、および、これを通じた QACCE 安全性の定義の不備を発見と修正
- LAN を構成するネットワーク機器における、VLAN 設定の検証手法の提案

上記において、これらの検証事例を現在までの研究の動向の中に位置付け、課題についても論じた。具体的に以下の 3 つのトピックについて述べた。

- クレジットカードを用いた決済の仕組みと、そこで用いられる暗号プロトコルの安全性の形式検証
- トランスポートレイヤの暗号プロトコルの安全性の形式検証
- ネットワークをソフトウェアにより構築および管理する Software-defined networking（SDN）における設定の正しさの形式検証

7.2 考察

本論文において論じてきた 3 つの検証事例、すなわち、電子商取引プロトコル SET の安全性検証、トランスポートレイヤプロトコル QUIC の安全性検証、そして LAN における機器の VLAN 設定の正当性の検証の研究、およびこれらに関連する研究の動向と課題の調査を踏まえ、ネットワークの安全性および正当性を担保するための形式検証技術の利用について論じる。特に議論の観点として、ネットワークの階層構造（レイヤ）における検証対象の位置付け、プロトコルのような抽象的対象とその実装のような具体的対象、検証技術の発展、安全性と正当性、対象技術分野の発展を取り上げる。

まずネットワークのレイヤについて、SET プロトコルは上位レイヤ、QUIC プロトコルは中位レイヤ、タグ VLAN は下位レイヤと大まかに位置付けることができる。一般にネットワークのレイヤ構造においては、より下位のレイヤほど共通的な機能を提供し、より上位のレイヤほど目的に特化した機能を提供する。上位レイヤにおいて下位レイヤの差異や機能を抽象化することで単純な機能を提供する場合もあるものの、一般には下位のレイヤは上位のレイヤに比べて単純な機能を提供すると言ってよい。言い換えれば、下位レイヤの機能を提供する情報システムは上位レイヤに比べて要求仕様が単純であると言える。これが形式検証の適用にも反映され、下位レイヤである VLAN 設定の検証では、状態遷移モデルの検証ツールという比較的汎用性が高く単純なモデルに適用できる技術によって検証を行うことができた。中位レイヤである QUIC プロトコルの検証においては、提供する機能、すなわちセキュリティにより特化した検証ツールを用いて、暗号プロトコルの安全性を記述・検証を行う必要があった。その一方で、安全性の記述は暗号プロトコルの安全性の記述の方法として標準的な一致表明と観測等価性を用いて記述することができた。上位レイヤである SET プロトコルの検証ではより柔軟に性質を記述できる、高階論理に基づく定理証明システム Isabelle を用いた。これは、この研究を開始した当時は暗号プロトコルに特化したツールとして使えるものがなかったためであり、安全性としてはカード番号の秘匿性という暗号プロトコルの安全性としては基本的なものである。しかし一般には、上位レイヤ、特にアプリケーションレイヤのプロトコルは、認証・鍵交換プロトコルなどの安全性を超えたより目的に特化した機能を提供する場合も十分に考えられ、その場合は暗号プロトコル専用の検証ツールでは性質の記述力が不足する可能性がある。このため一般的な傾向として、より上位のレイヤの情報システムの安全性はより記述力の高い言語で記述し、そのためのツールを用いて検証する必要があると言える。

レイヤの観点に、検証にかかるコストの観点を加味すると、次のようなことも言える。

あるレイヤの情報システムは、より下位のレイヤの構成をより上位のレイヤから隠蔽し抽象化することで、上位のレイヤの情報システムを下位のレイヤを気にせずに構築できるようにする。たとえば本論文で取り上げた VLAN は、単一の LAN としての構造を隠蔽して、より上位のレイヤに対して複数の独立した LAN の機能として見せる。このため、下位レイヤの情報システムはそれが使われる目的や状況などにより構成が大きく異なる場合がある。このような場合は、VLAN を構成する毎に検証を行うことになるため、よりコスト（時間、労力）が低い高速な検証方法が求められる。ただし、要求仕様の種類としてはある程度共通なさまざまなシステムが存在する可能性がある。VLAN の場合で言えば、さまざまなケースで VLAN が構成されるものの、求められる性質は VLAN の分離やループの不存在などある程度共通性がある。このため、本論文の VLAN 設定検証で行ったようなコストが低い検証を行える可能性がある。これに対し、中位・上位のレイヤの情報システムである QUIC プロトコルや SET プロトコルは、直下のレイヤが提供する抽象化の恩恵を受けて、より下位レイヤの構成によらない構成を行うことができるため、基本的な構成は利用の状況などによらず、同じシステムが用いられる。このため、不具合があった場合の損害を鑑みて、ある程度コストをかけて検証を行うことが正当化される。

検証にかかるコストの観点では、プロトコルのような雛形的なものと、その実装の違いも重要である。SET や QUIC などのプロトコルは、それをさまざまな形のソフトウェアとして実装することができる。プロトコルに欠陥があればそれを実装したソフトウェア全てに影響が及ぶ可能性が高いため、実装を行う前に検証により安全性を厳密に確認しておきたい。プロトコルの安全性の検証に比べて実装の検証は大きなコストを伴うこともあり、様々な実装を抽象化した共通部分であるプロトコルの安全性を検証することが経済的と考えられる。ただし、第 5.6.2 節で紹介した Bhargavan らの研究のように、特に重要だと考えられるプロトコルについては実装の検証も行われるようになりつつある。また、システムの構成がある程度頻繁に変わるようなシステムであっても、VLAN の検証のように検証作業が自動化しやすい場合はその実装も含めて検証を行うことがコストに見合う場合もある。

形式検証技術の発展の観点からは、第 3 章で述べたように、定理証明システムや、状態遷移システム検証ツールは、形式手法の最初期の段階から研究され使用されてきた。この 2 つはある意味で両極端にあると言える。つまり、前者は極めて高い記述能力を持つが、証明の自動化を完全に行うことが難しく、証明の方針や具体的な方法を人手で決める必要がある。後者は記述力が限定されるものの高速化かつ自動的に検証を行うことができる。下位レイヤの情報システムである VLAN の検証では後者を用いた。これは、ネットワークの構築や構成変更のたびに形式検証の専門家が検証を行うことはコストの観点で難しい

ことと、VLAN の検証は状態遷移システムに落とし込むことが容易だったためである。暗号プロトコルの安全性検証についても、初期の研究では前者および後者を用いて検証が行われてきたことは第 3 章で述べたとおりである。SET の検証の研究では前者を用いて検証を行った。暗号プロトコルの安全性検証に特化したツールの研究が進み、特に ProVerif の登場により自動検証による安全性の検証が行われるようになった。本論文の研究でも、QUIC の検証では ProVerif を用いた。汎用的な手法を用いることは少なくなったが、定理証明システムはその記述力の高さから、特に上位レイヤのシステムについて、ProVerif などでは扱えない性質を検証するために利用できると考える。また、暗号プロトコルで用いられる暗号アルゴリズムそのものの性質を計算論的モデルのもとで検証するツールである EasyCrypt に関連して、その前身とも言えるツールである CertiCrypt は定理証明システム Coq 上で構築された。このように、既存の専用ツールによる検証が難しい場合は、汎用的で記述力が高い手法を用いることができ、さらに、その理論的な有効性を示したのち、より効率的な検証を行うための専用ツールをあらためて開発するという流れもある。

検証すべき性質の観点からは、SET や QUIC などの暗号プロトコルの検証と VLAN 設定の検証では、それぞれ安全性と正当性を検証しており、検証すべき性質が大きく異なる。前者においては悪意を持って意図的に機能の提供を阻害しようとする攻撃者が仮定される一方、後者では基本的には通常の利用の範囲で不具合を防げば良い。前者の安全性の検証では、攻撃者によるありとあらゆる攻撃の可能性を網羅的に確認して検証を行う必要があるため、攻撃者のモデルを内蔵する安全性の特化した手法やツールが必要であった。もちろん VLAN などの下位レイヤのネットワークにおいても安全性（セキュリティ）が必要になる場合も多く、その場合は攻撃者を考慮した検証が必要になる。特に、SDN などのソフトウェア技術により構築されるネットワークにおいては、コントローラのソフトウェアと転送レイヤのハードウェアの間の通信に介入して攻撃が行われるケースなどを考える場合もある。

検証対象のドメインにおける利便性の追求やそれを提供する技術の発展も考慮するに値する観点である。クレジットカードを用いた決済プロトコルだけを見ても、カードを店舗で提示する決済においては磁気ストライプの読み出しから、IC チップを用いた通信と暗号化、非接触の通信の利用と発展している。システムの構成として複雑さが増しており、さらに、非接触の通信の利用により、中間者攻撃などの可能性もより現実的になっており、検証すべき安全性も検証対象のシステムも複雑になっている。オンラインでの決済でも同様に、SET のような比較的単純な仕組みから、3-D Secure 1.0 および 2.0 と発展する過程で、本人認証に用いるデバイスや利用者が用いる端末の多様性が増しており、ここでも検証すべき安全性および対象のシステムの双方が複雑になっている。QUIC プロトコル

においても、それまでの TLS 1.2 に比べてより低い遅延を求められ、そのために 0-RTT が導入されることで新たなリプレイ攻撃の可能性が生じている。VLAN の検証においても、データセンタなどに代表されるような、ネットワークの仮想化・ソフトウェア化により SDN が導入され、コントローラの正しさなどのより検証が困難な性質が求められてきている。したがって同じ対象ドメインであっても、その進歩に応じて適切に検証方法を選択していくことが求められる。

形式検証は一般にはコストが大きい手法ではあるが、以上に考察したように、扱った対象のネットワークの階層構造（レイヤ）における位置付け、プロトコルのような抽象的対象とその実装のような具体的対象、検証技術の発展、安全性と正当性、対象技術分野の発展などの観点から適切に手法やツールを選択することで、コストに見合った検証が可能な場合は多いと考えられる。さらに、形式検証以外の方法で安全性や正当性が解析されていたとしても、本論文における QUIC プロトコルの例のように、その解析の欠陥を発見できる場合があるため、他の手法と組み合わせて用いることも有効であると考えられる。

謝辞

本論文の研究を進めるにあたりご指導いただきました、九州大学大学院システム情報科学研究院情報学部門の櫻井幸一教授に深く感謝いたします。

本論文をまとめるにあたり貴重なご助言とご指導を賜りました、九州大学大学院システム情報科学研究院 I&E ビジョナリー特別部門の廣川真男教授、九州大学マス・フォア・インダストリ研究所応用理論研究部門の溝口佳寛教授に深く感謝いたします。

本論文の研究を日本電信電話株式会社において進めるにあたり、長年にわたりご指導いただき、本論文のアドバイザとしても貴重なご意見を賜りました関東学院大学理工学部の塚田恭章教授に心から感謝いたします。同僚として形式検証技術について多くの議論をしていただきました、日本電信電話株式会社コミュニケーション科学基礎研究所の真野健主任研究員、愛知工業大学情報科学部の河辺義信教授に感謝いたします。

本論文の研究を進めるにあたりご議論いただきました情報通信研究機構サイバーセキュリティ研究所の吉田真紀主任研究員、茨城大学学術研究院応用理工学野の米山一樹教授、株式会社東芝研究開発センターの花谷嘉一上席研究員に感謝いたします。

本論文の執筆を強く勧めて下さった、日本電信電話株式会社基礎数学研究センタの若山正人数学研究プリンシパル、日本電信電話株式会社コミュニケーション科学基礎研究所の原田登上席特別研究員に感謝いたします。

最後に、常に私を励まし、本研究の遂行に様々な面において協力してくれた妻、夏子、そして娘、珠子に感謝します。

参考文献

[1] 宮居雅宣. 決済サービスとキャッシュレス社会の本質. 一般社団法人金融財政事情研究会, 2020.

[2] Efstratios Chatzoglou, Vasileios Kouliaridis, Georgios Karopoulos, and Georgios Kambourakis. Revisiting QUIC attacks: a comprehensive review on QUIC security and a hands-on study. *International Journal of Information Security*, Vol. 22, No. 2, pp. 347–365, April 2023.

[3] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-Defined networking: A comprehensive survey. *Proc. IEEE*, Vol. 103, No. 1, pp. 14–76, January 2015.

[4] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, Vol. 56, No. 3, pp. 131–136, 1995.

[5] 國廣昇, 安田雅哉, 水木敬明, 高安敦, 高島克幸, 米山一樹, 大原一真, 江村恵太. 暗号の理論と技術 量子時代のセキュリティ理解のために. 講談社, 2024.

[6] 萩谷昌己, 塚田恭章（編）. 数理的技法による情報セキュリティ. 共立出版, 2010.

[7] Roger Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, Vol. 21, No. 12, pp. 993–999, 1978.

[8] Danny Dolev and Andrew C. Yao. On the security of public-key protocols (extended abstract). In *22nd Annual Symposium on Foundations of Computer Science*, pp. 350–357, Nashville, Tennessee, 28–30 October 1981. IEEE.

[9] Martí;n Abadi and Phillip Rogaway. Reconciling two views of cryptography. In Jan van Leeuwen, Osamu Watanabe, Masami Hagiya, Peter D. Mosses, and Takayasu Ito, editors, *Theoretical Computer Science: Exploring New Frontiers*

of Theoretical Informatics, pp. 3–22, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[10] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption)*. *Journal of Cryptology*, Vol. 15, No. 2, pp. 103–127, 2002.

[11] Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *Journal of Automated Reasoning*, Vol. 46, No. 3, pp. 225–259, 2011.

[12] Gergei Bana and Hubert Comon-Lundh. Towards unconditional soundness: Computationally complete symbolic attacker. In Pierpaolo Degano and Joshua D. Guttman, editors, *Principles of Security and Trust*, pp. 189–208, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[13] Gergei Bana and Hubert Comon-Lundh. A computationally complete symbolic attacker for equivalence properties. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, p. 609 – 620, New York, NY, USA, 2014. Association for Computing Machinery.

[14] Simon Jeanteur, Laura Kovács, Matteo Maffei, and Michael Rawson. Cryptovampire: Automated reasoning for the complete symbolic attacker cryptographic model, 2024.

[15] ISO/IEC 13817-1:1996: Information technology – programming languages, their environments and system software interfaces – Vienna Development Method – specification language. Technical report, International Organization for Standardization, March 1996.

[16] Cliff B. Jones. Systematic software development using VDM. *Prentice Hall International Series in Computer Science*, 1990.

[17] J. Michael Spivey. *Understanding Z: a specification language and its formal semantics*, Vol. 3. Cambridge University Press, 1988.

[18] Jean-Raymond Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[19] Joseph Goguen, Timothy Winkler, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. *Software engineering with OBJ: algebraic specification in action*, 11 1999.

[20] CafeOBJ: Algebraic specification and verification. `https://cafeobj.org/`. Ac-

cessed: 12.06.2024.

[21] The Maude system. https://maude.cs.illinois.edu/w/index.php/The_Maude_System. Accessed: 12.06.2024.

[22] Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: the Common Algebraic Specification Language. *Theoretical Computer Science*, Vol. 286, No. 2, pp. 153–196, 2002.

[23] Charles Antony Richard Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[24] Tommaso Bolognesi and Ed Brinksma. Introduction to the iso specification language lotos. *Computer Networks and ISDN Systems*, Vol. 14, No. 1, pp. 25–59, 1987.

[25] The Coq proof assistant. https://coq.inria.fr/. Acccessed: 12.06.2024.

[26] Gilles Dowek, Amy Felty, Hugo Herbelin, Gérald Huet, Catherine Parent, Christine Paulin-Mohring, Benjamin Werner, and Chetan Murthy. The Coq proof assistant user's guide: Version 5.8. Rapport INRIA 154, INRIA, 1993.

[27] Isabelle. https://isabelle.in.tum.de/. Accessed: 12.06.2024.

[28] The international SAT competition web page. http://www.satcompetition.org/. Accessed: 12.06.2024.

[29] The international satisfiability modulo theories (SMT) competition. https://smt-comp.github.io/. Accessed: 12.06.2024.

[30] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, Vol. 12, No. 10, pp. 576–580, 1969.

[31] Edsger W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, Vol. 18, No. 8, pp. 453–457, 1975.

[32] Why3: Where programs meet provers. https://www.why3.org/. Accessed: 12.06.2024.

[33] Kenneth L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic Press, 1993.

[34] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Computer Aided Verification, 14th International Conference, CAV 2002,Copen-*

hagen, Denmark, July 27-31, 2002, Proceedings, Vol. 2404 of *Lecture Notes in Computer Science*, pp. 359–364. Springer, 2002.

[35] Amir Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, Vol. 13, pp. 45–60, 1981.

[36] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The temporal logic of branching time. *Acta Informatica*, Vol. 20, pp. 207–226, 1983.

[37] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logic of Programs*, Vol. 131 of *Lecture Notes in Computer Science*, pp. 52–71. Springer, 1981.

[38] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In Jacobus W. de Bakker and Jan van Leeuwen, editors, *ICALP*, Vol. 85 of *Lecture Notes in Computer Science*, pp. 169–181. Springer, 1980.

[39] 田辺良則, 高井利憲, 高橋孝一. 抽象化を用いた検証ツール. コンピュータ ソフトウェア, Vol. 22, No. 1, pp. 1_2–1_44, 2005.

[40] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[41] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Information and Computation*, Vol. 100, No. 1, pp. 1–40, 1992.

[42] Verifying multi-threaded software with Spin. `https://spinroot.com/spin/whatispin.html`. Accessed: 12.06.2024.

[43] David L.. Dill, Andreas J. Drexler, Alan J. Hu, and C. Hang Yang. Protocol verification as a hardware design aid. In *Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pp. 522–525, 1992.

[44] Lawrence C Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, Vol. 6, No. 1-2, pp. 85–128, January 1998.

[45] 田中慎也, 佐藤文明, 水野忠則. Spin に基づくセキュリティプロトコル検証システム. 情報処理学会論文誌, Vol. 42, No. 2, pp. 147–154, 02 2001.

[46] Audun Jøsang. Security protocol verification using SPIN. In *Proceedings of the First SPIN Workshop, INRS-Telecommunications, Montreal, Canada, 1995*, pp. 1–9, 1995.

[47] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of

cryptographic protocols using murϕ. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No.97CB36097)*, pp. 141–151, 1997.

[48] Kazuhiro Ogata and Kokichi Futatsugi. Rewriting-based verification of authentication protocols. *Electronic Notes in Theoretical Computer Science*, Vol. 71, pp. 208–222, 2004. WRLA 2002, Rewriting Logic and Its Applications.

[49] ProVerif: Cryptographic protocol verifier in the formal model. `https://bblanche.gitlabpages.inria.fr/proverif/`. Accessed: 12.06.2024.

[50] Tamarin prover. `https://tamarin-prover.com/`. Accessed: 12.06.2024.

[51] Verifpal. `https://verifpal.com/`. Accessed: 12.06.2024.

[52] CryptoVerif: Cryptographic protocol verifier in the computational model. `https://bblanche.gitlabpages.inria.fr/CryptoVerif/`. Accessed: 12.06.2024.

[53] F*: A proof-oriented programming language. `https://www.fstar-lang.org/`. Accessed: 12.06.2024.

[54] Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, p. 90 – 101, New York, NY, USA, 2009. Association for Computing Machinery.

[55] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. EasyCrypt: A tutorial. In Alessandro Aldini, Javier Lopez, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII: FOSAD 2012/2013 Tutorial Lectures*, pp. 146–166. Springer International Publishing, Cham, 2014.

[56] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pp. 71–90, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[57] 川本裕輔. 暗号系の安全性検証 - 入門から計算機による証明まで. コンピュータ ソフトウェア, Vol. 33, No. 4, pp. 4_67–4_83, 2016.

[58] Manuel Barbosa, Gilles Barthe, Karthik Bhargavan, Bruno Blanchet, Cas Cremers, Kevin Liao, and Bryan Parno. SoK: Computer-aided cryptography. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 777–795, 2021.

[59] Matteo Avalle, Alfredo Pironti, and Riccardo Sisto. Formal verification of security protocol implementations: a survey. *Form. Asp. Comput.*, Vol. 26, No. 1, p. 99 – 123, jan 2014.

[60] SET Secure Electronic Transaction LLC. SET secure electronic transaction book 1: Business description, 1997.

[61] SET Secure Electronic Transaction LLC. SET secure electronic transaction book 2: Programmer's guide, 1997.

[62] SET Secure Electronic Transaction LLC. SET secure electronic transaction book 3: Formal protocol definition, 1997.

[63] Dominique Bolignano. Towards the formal verification of electronic commerce protocols. In *10th IEEE Computer Security Foundations Workshop*, pp. 133–146, 1997.

[64] Shiyong Lu and S. Smolka. Model checking SET secure electronic transaction protocol. In *Proceeding of 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, pp. 358–365, 1999.

[65] Formal Systems Ltd. FDR2 user manual, 1998.

[66] Catherine Meadows and Paul Syverson. A formal specification of requirements for payment transactions in the SET protocol. In *Financial Cryptography '98*, LNCS 1465. Springer-Verlag, 1998.

[67] Giampaolo Bella, Fabio Massacci, Lawrence C. Paulson, and Piero Tramontano. Formal verification of cardholder registration in SET. In *Sixth European Symposium on Research in Computer Security: ESORICS 2000*, 2000.

[68] Giampaolo Bella and Lawrence C. Paulson. Verifying set purchase protocols. Report 524, Computer Lab., Univ. of Camridge, 2001.

[69] Giampaolo Bella, Fabio Massacci, and Lawrence C. Paulson. Verifying the SET Purchase Protocols. *Journal of Automated Reasoning*, Vol. 36, No. 1, pp. 5–37, January 2006.

[70] 林晋. 数理論理学. コロナ社, 1989.

[71] 大堀淳. プログラミング言語の基礎理論. 共立出版, 1997.

[72] 南出靖彦. Isabelle/HOL. 知識ベース 知識の森 7 群 1 編 ソフトウェア基礎 2 章 ソフトウェア検証, pp. 13–20. 電子情報通信学会, 2010.

[73] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of*

Automated Reasoning, Vol. 5, pp. 363–397, 1989.

[74] Simon Roßkopf and Tobias Nipkow. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, Vol. 67, No. 1, pp. 1–21, 2023.

[75] Lawrence C. Paulson, Tobias Nipkow, and Markus Wenzel. Isabelle's logics. `https://isabelle.in.tum.de/doc/logics.pdf`, 2024. Accessed: 04.06.2024.

[76] The HOL system. `https://www.cl.cam.ac.uk/research/hvg/HOL/`. Accessed: 07.06.2024.

[77] Mike Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report UCAM-CL-TR-68, Cambridge University, 1985. Accessed: 07.06.2024.

[78] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, Vol. 17, No. 3, pp. 348–375, 1978.

[79] Lawrence C. Paulson. A fixedpoint approach to implementing (co)inductive definitions. In *Automated Deduction — CADE-12*, pp. 148–161. Springer 1994.

[80] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, Vol. 6, No. 1, pp. 85–128, 1998.

[81] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Transactions on Computer Systems*, Vol. 8, No. 1, pp. 18–36, 1990.

[82] Lawrence C. Paulson. Relations between secrets: Two formal analyses of the yahalom protocol. *Journal of Computer Security*, Vol. 9, No. 3, pp. 197–216, 2001.

[83] Hideki Sakurada. Isabelle proof scripts for the secrecy in the set payment protocol. `https://github.com/sakurada/set-secrecy/`. Accessed: 12.06.2024.

[84] EMVCo, LLC. EMV specifications and associated bulletins. `https://www.emvco.com/specifications/`. Accessed: 12.06.2024.

[85] Grandy N. Drew. *USING SET for Security Electronic Commerce*. Prentice Hall PTR, 1998.

[86] Visa International Service Association. 3-D Secure introduction, version 1.0.2, 2002.

[87] EMVCo, LLC. EMV 3-D Secure. `https://www.emvco.com/emv-technologies/3-d-secure/`. Accessed: 12.06.2024.

[88] EMVCo, LLC. EMV 3-D Secure protocol and core functions specification (v2.3.1.1). `https://www.emvco.com/specifications/?post_id=90915`, 2023.

Accessed: 12.06.2024.

[89] Joeri de Ruiter and Erik Poll. Formal analysis of the EMV protocol suite. In *Theory of Security and Applications*, Lecture notes in computer science, pp. 113–129. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[90] Steven J. Murdoch, Saar Drimer, Ross Anderson, and Mike Bond. Chip and pin is broken. In *2010 IEEE Symposium on Security and Privacy*, pp. 433–446, 2010.

[91] David Basin, Ralf Sasse, and Jorge Toro-Pozo. The EMV Standard: Break, Fix, Verify. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1766–1781, May 2021.

[92] David Basin, Ralf Sasse, and Jorge Toro-Pozo. Card brand mixup attack: Bypassing the PIN in non-Visa cards by using them for visa transactions. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 179–194. USENIX Association, August 2021.

[93] David Basin, Patrick Schaller, and Jorge Toro-Pozo. Inducing authentication failures to bypass credit card PINs. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 3065–3079, Anaheim, CA, August 2023. USENIX Association.

[94] Andrew William Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pp. 98–107, 1995.

[95] 渡部翔, 米山一樹. EMV 3-D セキュアにおける Challenge Flow の形式検証. 2024 年 暗号とセキュリティシンポジウム (SCIS2024), 2024.

[96] Quoc Huy Do, Pedram Hosseyni, Ralf Küsters, Guido Schmitz, Nils Wenzler, and Tim Würtele. A formal security analysis of the w3c web payment apis: Attacks and verification. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 215–234, 2022.

[97] The European Parliament and of the Council. Directive (EU) 2015/2366 of the European Parliament and of the Council of 25 November 2015. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02015L2366-20151223`, 2015. Accessed: 12.06.2024.

[98] 経済産業省 商務・サービスグループ商取引監督課. クレジットカードシステムのセキュリティ対策の更なる強化に向けた方向性（クレジット・セキュリティ対策ビ

ジョン2025）. https://www.meti.go.jp/policy/economy/consumer/credit/2022060221001.pdf, 2022. Accessed: 12.06.2024.

[99] クレジット取引セキュリティ対策協議会. クレジットカード・セキュリティガイドライン【4.0 版】＜公表版＞. https://www.j-credit.or.jp/security/pdf/Creditcardsecurityguidelines_4.0_published.pdf, 2023. Accessed: 12.06.2024.

[100] Tim Dierks and Christopher Allen. The TLS protocol version 1.0. In *RFC 2246 (Proposed Standard), Internet Engineering Task Force*, 1999.

[101] Bryan Ford. Structured streams: a new transport abstraction. In *SIGCOMM 2007*, pp. 361–372, 2007.

[102] Randall Stewart. Stream Control Transmission Protocol. In *RFC 4960 (Proposed Standard), Internet Engineering Task Force*, 2007.

[103] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and K. K. Ramakrishnan. Towards a SPDY'ier mobile web? In *CoNEXT 2013*, pp. 303–314, 2013.

[104] Jim Roskind. QUIC (Quick UDP Internet Connections): Multiplexed Stream Transport Over UDP. https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/, 2013. Accessed: 12.06.2024.

[105] Marc Fischlin and Felix Günther. Multi-Stage Key Exchange and the Case of Google's QUIC Protocol. In *ACM Conference on Computer and Communications Security 2014*, pp. 1193–1204, 2014.

[106] Mihir Bellare and Phillip Rogaway. Entity Authentication and Key Distribution. In *CRYPTO 1993*, pp. 232–249, 1993.

[107] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How Secure and Quick is QUIC? Provable Security and Performance Analyses. In *2015 IEEE Symposium on Security and Privacy*, pp. 214–231. IEEE, May 2015.

[108] Bruno Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, Vol. 17, No. 4, pp. 363–434, July 2009.

[109] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *The Journal of Logic and Algebraic Programming*, Vol. 75, No. 1, pp. 3 – 51, 2008.

[110] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with ProVerif. In David Basin and John Mitchell, editors, *2nd Conference on*

Principles of Security and Trust (POST 2013), Vol. 7796 of *Lecture Notes in Computer Science*, pp. 226–246, Rome, Italy, March 2013. Springer.

[111] David McGrew and John Viega. The galois/counter mode of operation (gcm). *submission to NIST Modes of Operation Process*, Vol. 20, pp. 0278–0070, 2004.

[112] Joseph Salowey, Hao Zhou, Pasi Eronen, and Hannes Tschofenig. Transport layer security (TLS) session resumption without server-side state. RFC 5077, RFC Editor, January 2008. `http://www.rfc-editor.org/rfc/rfc5077.txt`.

[113] Proverif script for formal verification of the QUIC protocol. `https://github.com/sakurada/quic-verification`. Accessed: 12.06.2024.

[114] Ivan Ristić. *Bulletproof TLS and PKI, Second Edition: Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck, 2022. （邦訳：齋藤孝道監訳「プロフェッショナル TLS & PKI 改題第 2 版」，2023 年）.

[115] Kipp E.B. Hickman. SSL 2.0 PROTOCOL SPECIFICATION, 1994.

[116] Kipp E.B. Hickman. The SSL protocol. Internet-Draft draft-hickman-netscape-ssl-00.txt, IETF Secretariat, 1995.

[117] Sean Turner and Tim Polk. Prohibiting secure sockets layer (SSL) version 2.0. RFC 6176, RFC Editor, March 2011.

[118] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The secure sockets layer (SSL) protocol version 3.0. RFC 6101, RFC Editor, August 2011.

[119] Christopher Allen and Tim Dierks. The TLS protocol version 1.0. RFC 2246, RFC Editor, January 1999.

[120] Tim Dierks and Eric Rescorla. The TLS protocol version 1.1. RFC 4346, RFC Editor, April 2006.

[121] Tim Dierks and Eric Rescorla. The TLS protocol version 1.2. RFC 5246, RFC Editor, August 2008.

[122] Eric Rescorla. The transport layer security (TLS) protocol version 1.3. RFC 8446, RFC Editor, August 2018.

[123] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium (USENIX Security 98)*, pp. 1–15, San Antonio, TX, January 1998. USENIX Association.

[124] Alan O. Freier, Philip L. Karlton, and Paul C. Kocher. The SSL protocol version 3.0. Internet-Draft draft-ietf-tls-ssl-version3-00.txt, IETF Secretariat,

November 1996.

[125] David Wagner and Bruce Schneier. Analysis of the SSL 3.0 protocol. In *2nd USENIX Workshop on Electronic Commerce (EC 96)*, pp. 1–12, Oakland, CA, November 1996. USENIX Association.

[126] Lawrence C Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Information and System Security*, Vol. 2, No. 3, pp. 332–351, August 1999.

[127] K Ogata and K Futatsugi. Equational Approach to Formal Analysis of TLS. In *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pp. 795–804. IEEE, 2005.

[128] Karthikeyan Bhargavan, Cédric Fournet, Ricardo Corin, and Eugen Zălinescu. Verified Cryptographic Implementations for TLS. *ACM Transactions on Information and System Security*, Vol. 15, No. 1, pp. 1–32, March 2012.

[129] Duong Dinh Tran and Kazuhiro Ogata. Formal verification of TLS 1.2 by automatically generating proof scores. *Computers & Security*, Vol. 123, p. 102909, December 2022.

[130] 荒井研一, 渡辺大, 櫻田英樹. ProVerif による TLS1.3 ハンドシェイクプロトコルの形式検証. コンピュータセキュリティシンポジウム 2015 論文集, pp. 1003–1010, 10 2015.

[131] Cas Cremers, Marko Horvat, Sam Scott, and Thyla van der Merwe. Automated analysis and verification of TLS 1.3: 0-RTT, resumption and delayed authentication. In *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 470–485, 2016.

[132] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 483–502. IEEE, May 2017.

[133] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, p. 1773 – 1788, New York, NY, USA, 2017. Association for Computing Machinery.

[134] Vincent Stettler. Formally analyzing the TLS 1.3 proposal. Bachelor Thesis, ETH Zurich, `https://ethz.ch/content/dam/ethz/special-`

interest/infk/inst-infsec/information-security-group-dam/research/software/TLS-1.3_thesis_vincent_stettler.pdf, 2016. Accessed: 12.06.2024.

[135] Bruno Blanchet. Composition Theorems for CryptoVerif and Application to TLS 1.3. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pp. 16–30. IEEE, July 2018.

[136] Karthikeyan Bhargavan, Vincent Cheval, and Christopher Wood. A symbolic analysis of privacy for TLS 1.3 with Encrypted Client Hello. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, p. 365 – 379, New York, NY, USA, 2022. Association for Computing Machinery.

[137] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. TLS Encrypted Client Hello. https://datatracker.ietf.org/doc/draft-ietf-tls-esni/13/, 2021.

[138] Antoine Delignat-Lavaud, Cedric Fournet, Markulf Kohlweiss, Jonathan Protzenko, Aseem Rastogi, Nikhil Swamy, Santiago Zanella-Beguelin, Karthikeyan Bhargavan, Jianyang Pan, and Jean Karim Zinzindohoue. Implementing and Proving the TLS 1.3 Record Layer. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 463–482. IEEE, May 2017.

[139] Hideki Sakurada, Kazuki Yoneyama, Yoshikazu Hanatani, and Maki Yoshida. Analyzing and fixing the QACCE security of QUIC. In Lidong Chen, David McGrew, and Chris Mitchell, editors, *Security Standardisation Research*, pp. 1–31, Cham, 2016. Springer International Publishing.

[140] Jingjing Zhang, Lin Yang, Xianming Gao, and Qiang Wang. Formal analysis of QUIC handshake protocol using ProVerif. In *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 132–138. IEEE, August 2020.

[141] Jingjing Zhang, Lin Yang, Xianming Gao, Gaigai Tang, Jiyong Zhang, and Qiang Wang. Formal Analysis of QUIC Handshake Protocol Using Symbolic Model Checking. *IEEE Access*, Vol. 9, pp. 14836–14848, 2021.

[142] Jingjing Zhang, Xianming Gao, Lin Yang, Tao Feng, Dongyang Li, and Qiang Wang. A Systematic Approach to Formal Analysis of QUIC Handshake Protocol

Using Symbolic Model Checking. *Security and Communication Networks*, Vol. 2021, pp. 1–12, August 2021.

[143] Antoine Delignat-Lavaud, Cédric Fournet, Bryan Parno, Jonathan Protzenko, Tahina Ramananandro, Jay Bosamiya, Joseph Lallemand, Itsaka Rakotonirina, and Yi Zhou. A Security Model and Fully Verified Implementation for the IETF QUIC Record Layer. In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1162–1178. IEEE, May 2021.

[144] Joshua D. Guttman. Filtering postures: Local enforcement for global policies. In *Proceedings 1997 IEEE Symposium on Security and Privacy*, pp. 120–129. IEEE Computer Society, May 1997.

[145] Joshua D. Guttman, Amy L. Herzog, and F. Javier Thayer. Authentication and confidentiality via IPSEC. In *Computer Security - ESORICS 2000, 6th European Symposium on Research in Computer Security, Toulouse, France, October 4-6, 2000, Proceedings*, Vol. 1895 of *Lecture Notes in Computer Science*, pp. 255–272. Springer, 2000.

[146] IEEE. IEEE standards for local and metropolitan area networks: Virtual bridged local area networks, 2003. IEEE Std 802.1Q-2003.

[147] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.

[148] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transaction on Programming Languages and Systems*, Vol. 8, No. 2, pp. 244–263, 1986.

[149] Larry Peterson and Carmelo Cascone and Brian O'Connor and Thomas Vachuska and Bruce Davie. *Software-Defined Networks: A Systems Approach*. Systems Approach LLC, January 2021. (邦訳: Software-Defined Networks ソフトウェア定義ネットワークの概念・設計・ユースケース, 進藤 資訓 (翻訳, 読み手), 海老澤 健太郎 (翻訳, 監修), 小林 正幸 (翻訳, 読み手), 翔泳社, 2022.).

[150] Open Networking Foundation. *OpenFlow Specifications*.

[151] 鈴木一哉, 下西英之, 千葉靖伸, 高宮安仁, 須堯一志, 金海好彦, 石井秀治. Openflow 技術とその応用. コンピュータ ソフトウェア, Vol. 30, No. 2, pp. 2_3–2_13, 2013.

[152] 藤田智成. OSS に見る IT の最新動向：4. Software-Defined Networking(SDN) を支える OSS 技術. 情報処理, Vol. 56, No. 3, pp. 248–252, February 2015.

[153] Open Networking Foundation. *P4 Open Source Programming Language.*

[154] Juan Camilo Correa Chica, Jenny Cuatindioy Imbachi, and Juan Felipe Botero Vega. Security in SDN: A comprehensive survey. *Journal of Network and Computer Applications*, Vol. 159, p. 102595, June 2020.

[155] 山崎智史, 八鍬豊, 登内敏夫. 形式手法による SDN/OpenFlow 高信頼化技術の研究動向. コンピュータ ソフトウェア, Vol. 33, No. 2, pp. 2_26–2_42, 2016.

[156] Nitin Shukla, Mayank Pandey, and Shashank Srivastava. Formal modeling and verification of software-defined networks: A survey. *International Journal of Network Management*, Vol. 29, No. 5, p. e2082, 2019. e2082 nem.2082.

[157] Alireza Souri, Monire Norouzi, Parvaneh Asghari, Amir Masoud Rahmani, and Ghazaleh Emadi. A systematic literature review on formal verification of software-defined networks. *Transactions on Emerging Telecommunications Technologies*, Vol. 31, No. 2, p. e3788, 2020. e3788 ETT-18-0271.R3.

[158] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*, SafeConfig '10, p. 37 – 44, New York, NY, USA, 2010. Association for Computing Machinery.

[159] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, p. 290 – 301, New York, NY, USA, 2011. Association for Computing Machinery.

[160] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. *SIGCOMM Comput. Commun. Rev.*, Vol. 41, No. 4, p. 290 – 301, aug 2011.

[161] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 113–126, San Jose, CA, April 2012. USENIX Association.

[162] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 99–111, Lombard, IL, April 2013. USENIX Association.

[163] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. VeriFlow: Verifying Network-Wide invariants in real time. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pp. 15–27, Lombard, IL, April 2013. USENIX Association.

[164] Rupak Majumdar, Sai Deep Tetali, and Zilong Wang. Kuai: A model checker for software-defined networks. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 163–170, October 2014.

[165] Arjun Guha, Mark Reitblatt, and Nate Foster. Machine-verified network controllers. *SIGPLAN Not.*, Vol. 48, No. 6, pp. 483–494, June 2013.

[166] OCaml. `https://ocaml.org/`. Accessed: 12.06.2024.

[167] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. p4v: practical verification for programmable data planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pp. 490–503, New York, NY, USA, August 2018. Association for Computing Machinery.

[168] Edsger W Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, Vol. 18, No. 8, pp. 453–457, August 1975.

[169] Ryan Doenges, Mina Tahmasbi Arashloo, Santiago Bautista, Alexander Chang, Newton Ni, Samwise Parkinson, Rudy Peterson, Alaia Solko-Breslin, Amanda Xu, and Nate Foster. Petr4: formal foundations for p4 data planes. *Proc. ACM Program. Lang.*, Vol. 5, No. POPL, pp. 1–32, January 2021.

[170] Nate Foster, Nick McKeown, Jennifer Rexford, Guru Parulkar, Larry Peterson, and Oguz Sunay. Using deep programmability to put network owners in control. *SIGCOMM Comput. Commun. Rev.*, Vol. 50, No. 4, pp. 82–88, October 2020.

公表論文一覧

本論文の成果に関する論文

第 4 章の内容は以下の論文で公表された。

1. 櫻田英樹. SET 支払いプロトコルの秘匿性検証. 情報処理学会論文誌 第４４巻 第５号 pp.2106-2116, 2003.
2. Hideki Sakurada, Kouichi Sakurai. SoK: Directions and Issues in Formal Verification of Payment Protocols. Advanced Information Networking and Applications, Proceedings of the 38th International Conference on Advanced Information Networking and Applications (AINA-2024), Volume 4, Lecture Notes on Data Engineering and Communications Technologies, Vol.202, pp.111-119, Springer, 2024.

第 5 章の内容は以下の論文で公表された。

1. Hideki Sakurada, Kazuki Yoneyama, Yoshikazu Hanatani, Maki Yoshida. Analyzing and Fixing the QACCE Security of QUIC. Proceedings of the Third International Conference on Security Standardisation Research, SSR 2016, Gaithersburg, MD, USA, December 5-6, Lecture Notes in Computer Science, Vol.10074, pp.1-31, 2016.

第 6 章の内容は以下の論文で公表された。

1. 櫻田英樹. モデル検査を用いたタグ VLAN の設定検査. 情報処理学会論文誌 第４７巻 第７号 pp.2247-2257, 2006.
2. Hideki Sakurada, Kouichi Sakurai. Research Directions in Formal Verification of Network Configurations toward Verification of Mobile Networks. Post-

proceedings of The 7th International Conference on Mobile Internet Security (MobiSec 2023), Communications in Computer and Information Science, Springer, 2024. （公表予定）

参考論文

1. 櫻田英樹. 文脈計算の環境計算による解釈. 情報処理学会論文誌：プログラミング Vol.41 No.SIG09(PRO8) pp.8-24. 2000.
2. Hideki Sakurada, Yasuyuki Tsukada. A Role-Based Specification of the SET Payment Transaction Protocol. Advances in Network and Distributed Systems Security. IFIP International Federation for Information Processing, vol. 78, pp.1-13. Springer, 2000.
3. Yoshinobu Kawabe, Ken Mano, Hideki Sakurada, Yasuyuki Tsukada. Theorem-proving anonymity of infinite-state systems. Information Processing Letters, Vol.101, No.1, pp.46-51. 2007.
4. Yoshinobu Kawabe, Hideki Sakurada. A Formal Approach to Designing Anonymous Software. 5th ACIS International Conference on Software Engineering Research (SERA 2007,) IEEE Conference Proceedings, pp.203-212 2007.
5. 川本 裕輔, 真野 健, 櫻田 英樹, 萩谷 昌己. 関数部分知識と匿名性検証. 日本応用数理学会論文誌 第１７巻 第４号 pp.559-576. 2007.
6. Yoshinobu Kawabe, Hideki Sakurada. An Adversary Model for Simulation-Based Anonymity Proof. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A(4), pp.1112-1120. 2008.
7. Yoshinobu Kawabe, Ken Mano, Hideki Sakurada, Yasuyuki Tsukada. On Backward-Style Anonymity Verification IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E91-A(9) pp.2597-2606. 2008.
8. Yusuke Kawamoto, Hideki Sakurada, Masami Hagiya. Computationally Sound Formalization of Rerandomizable RCCA Secure Encryption. Formal to Practical Security, Papers Issued from the 2005-2008 French-Japanese Collaboration, Springer, Lecture Notes in Computer Science, Vol.5458, pp.158-180. 2009.
9. Hubert Comon-Lundh, Kawamoto Yusuke, Hideki Sakurada. Computational

and Symbolic Anonymity in an Unbounded Network. JSIAM Letters, Vol.1, pp.28-31. 2009.

10. Ichiro Hasuo, Yoshinobu Kawabe, Hideki Sakurada. Probabilistic anonymity via coalgebraic simulations. Theoretical Computer Science, Vol.411, No.22-24, pp.2239-2259. 2010.
11. Yasuyuki Tsukada, Ken Mano, Hideki Sakurada, Yoshinobu Kawabe. Anonymity, Privacy, Onymity, and Identity: A Modal Logic Approach. Transactions on Data Privacy, Vol.3, No.3, pp.177-198. 2010.
12. Ken Mano, Yoshinobu Kawabe, Hideki Sakurada, Yasuyuki Tsukada. Role Interchange for Anonymity and Privacy of Voting. Journal of Logic and Computation, Vol.20, No.6, pp.1251-1288. 2010.
13. Sakurada Hideki. Automatic verification of anonymity of protocols. JSIAM Letters Vol.3 pp.97-100 . 2011.
14. 河辺 義信, 真野 健, 櫻田 英樹, 塚田 恭章電子投票プロトコルに対する無証拠性の定理証明. 情報処理学会論文誌 第 52 巻 第 9 号 pp.2549-2561. 2011.
15. Hubert Comon-Lundh, Masami Hagiya, Yusuke Kawamoto, Hideki Sakurada. Computational Soundness of Indistinguishability Properties without Computable Parsing. Information Security Practice and Experience, 8th International Conference, ISPEC 2012, Lecture Notes in Computer Science, Vol.7232, pp.63-79, Springer. 2012.
16. Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, Hideki Sakurada. Application of a Process Calculus to Security Proofs of Quantum Protocols. The 2012 International Conference on Foundations of Computer Science, WorldComp Proceedings. 2012.
17. Gergei Bana, Pedro Adão, Hideki Sakurada. 19. Computationally Complete Symbolic Attacker in Action. IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2012,) Leibniz International Proceedings in Informatics (LIPIcs), V.18, pp.546-560. 2012.
18. Yasuyuki Tsukada, Hideki Sakurada, Ken Mano, Yoshifumi Manabe. An Epistemic Approach to Compositional Reasoning about Anonymity and Privacy. Proceedings of the 14th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 2013,) pp.239-248. 2013.
19. Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, Hideki

Sakurada. Automated Verification of Equivalence on Quantum Cryptographic Protocols. 5th International Symposium on Symbolic Computation in Software Science (SCSS 2013,) pp.64-69. 2013.

20. Hideki Sakurada. Computational Soundness of Symbolic Blind Signatures under Active Attacker. Foundations and Practice of Security - 6th International Symposium (FPS,) Lecture Notes in Computer Science, Vol.8352, pp.247-267. 2014.
21. Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, Hideki Sakurada. Semi-automated verification of security proofs of quantum cryptographic protocols. Journal of Symbolic Computation, Vol.73, March – April, pp.192-220. 2016.
22. Yasuyuki Tsukada, Hideki Sakurada, Ken Mano, Yoshifumi Manabe. On compositional reasoning about anonymity and privacy in epistemic logic. Annals of Mathematics and Artificial Intelligence, Vol.78, No.8, pp.101-129. 2016.
23. Ken Mano, Hideki Sakurada, Yasuyuki Tsukada. Trust Trust Me (The Additivity.) Trust Management XI - 11th IFIP WG 11.11 International Conference(IFIPTM,) pp.135-151, IFIP Advances in Information and Communication Technology, Vol.505, Springer. 2017.
24. Ken Mano, Hideki Sakurada, Yasuyuki Tsukada. Quality and Quantity Pair as Trust Metric. IEICE Transactions on Information and Systems E106.D(2) pp.181-194. 2023.

索引