暗号プロトコルの形式記述に向けた LLMチャットボットの活用

櫻田 英樹[1,2,a)]　櫻井 幸一[2]

概要：暗号プロトコルの安全性を検証する方法として形式検証がある．形式検証ではプロトコルの仕様と安全性を検証ツールと呼ばれるソフトウェアの入力言語によって記述することで自動的な検証が可能である．しかし一般にプロトコルの仕様と安全性は自然言語で記述・説明されており，ツールの言語で記述し直すことは専門的な知識と時間を要する．本研究では LLM チャットボットを活用して暗号プロトコルの形式記述を効率的に作成することを試みた．具体的には LLM チャットボットが自然言語で記述されたプロトコル仕様を理解し，これを形式的な記述に変換する過程を説明する．この手法を用いることで，形式検証ツールへの入力作成の最初のステップを支援し，形式検証ツールを使い始める際の労力を削減することを目指す．

キーワード：形式検証，暗号プロトコル，大規模言語モデル

Utilizing LLM Chatbots for Formal Descriptions of Cryptographic Protocols

Hideki Sakurada[1,2,a)]　Kouichi Sakurai[2]

Abstract: Formal verification is a method used to verify the security of cryptographic protocols. In formal verification, the specifications of a protocol and its security properties are described using the input language of a software tool called a verification tool, enabling automated verification. However, protocol specifications and security properties are generally described and explained in natural language, and rewriting them in the tool's language requires specialized knowledge and time. This study attempts to efficiently create formal descriptions of cryptographic protocols by utilizing an LLM chatbot. Specifically, it explains the process by which the LLM chatbot understands protocol specifications described in natural language and converts them into formal descriptions. By using this approach, we aim to support the initial step of creating inputs for formal verification tools and reduce the effort required when starting to use such tools.

Keywords: formal verification, security protocol, large language model

1. はじめに

1.1 背景

通信において秘匿性や完全性を保証する際に，暗号技術を組み合わせて認証や鍵交換を行う通信手順である暗号プロトコルが用いられる．暗号プロトコルは一般に複数のソフトウェアに実装される．このため暗号プロトコルに欠陥がある場合はこれを実装したソフトウェア全てに欠陥が生じる可能性がある．暗号プロトコルは一般に暗号プロトコルを含むセキュリティの専門家によって設計されるが，安全なプロトコルを設計することは容易ではないため，欠陥を含む可能性がある．広く使われている web ブラウザおよびサーバに実装されている TLS プロトコルにおいても，

[1] 日本電信電話株式会社 NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories, NTT Corporation
[2] 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
[a)] hideki.sakurada@ntt.com

これまでに欠陥が発見されて実装での対応やプロトコルの修正を余儀なくされてきた [1].

暗号プロトコルの安全性を保証する方法として，暗号理論的な枠組みによって安全性を証明することが行われている．しかしその証明は計算量的・確率的な議論を含み複雑なものとなる場合があるため，プロトコル規模によっては細部にわたり証明を与えることができず，できたとしても証明の正しさを確認するために労力が必要となる．プロトコルの安全性を保証するもう一つの方法として形式検証がある．形式検証ではまず，プロトコルの仕様とその安全性を意味が数理的・論理的に定義された言語で記述し，これを元に安全性を検証する．プロトコルの安全性を自動的に検証するためのツール（ソフトウェア）として，ProVerif[2] や Tamarin Prover[3] がある．形式ツールに与えたプロトコルの仕様と安全性およびツールの出力を確認することで検証が正しく行われたかどうかを確認することができる．その一方，形式検証に用いるツールのうち完全に自動的な検証を行うものは暗号理論的な証明のように計算量的・確率的な議論を行うことができず，プロトコルで用いられる暗号技術（公開鍵暗号，共通鍵暗号，電子署名など）は理想的な安全性を持つと仮定する．

暗号プロトコルの仕様は一般に自然言語で記述される．形式検証を適用する場合は，これをツールの言語に合わせて述し直す必要がある．その際，プロトコルをビット列のレベルまで記述するのではなくツールの抽象度に合わせた記述が必要となる．このためツールの言語を理解していることに加えプロトコルの規模により労力が必要になる．直観的にプロトコルを記述することを目指した検証ツールとして Verifpal[4] があるが，自然言語での記述を理解して記述することは避けられない．

1.2 先行研究

近年，大規模言語モデル（LLM）を用いた言語処理技術が急速に発展してきており，プログラムの生成も可能なモデルが現れている．まず LLM によってプログラムを生成させ，それを修正・追記することでプログラムを書くという方法も一般的になりつつある．また LLM を用いて自然言語で記述されたプロトコルからその動作のモデルを抽出する研究も行われている．

Sharma と Yegneswaran[5] は，暗号プロトコルに限定しない一般的な通信プロトコルの仕様からプロトコルの参加者の状態機械を抽出する方法を提案した．具体的には LLM として ChatGPT（GPT-3.5-turbo）[6] を用いて IETF の技術文書である RFC から状態遷移図を抽出する方法を提案した．彼らの方法を暗号プロトコルの検証に用いるためには，送受信されるメッセージの中身のうち，暗号に関連する部分がどのように処理されるかという情報も抽出できる必要がある．具体的には，送受信される乱数をいつ誰が生成したか，共通鍵暗号に用いられる鍵はあらかじめどの参加者に共有されているのかなどが必要になる．

Liu ら [7] は，ブロックチェーン上で実行されるスマートコントラクトについてそのプログラムの性質を検証するために LLM を用いた．一般にプログラムの性質を検証するためにはプログラムの実行中・終了時に成り立つべき性質を与え，検証ツールによりこれを検証させる．しかし自動的な検証のためには成り立つべき性質より少し強い性質を終了時の性質あるいは実行中の性質として与える必要がある．彼らはこれを LLM を用いて与える方法を提案した．具体的にはプログラムとそれに成り立つ性質のデータベースを準備しておき，LLM にこのデータベースからの例を与えて性質が未知のプログラムについて性質を記述させた．この方法は暗号プロトコルの安全性検証においても安全性の記述のために利用できることが期待できる．しかしスマートコントラクトの場合は検証対象のプログラムが与えられているのに対し，暗号プロトコルの場合はプロトコルの仕様をツールに合わせた言語で記述する必要があり，この部分には彼らの方法は直接的には適用できない．

この他，ソフトウェアの脆弱性を自動的に修復するために LLM を用いる研究 [8] などが行われている．

1.3 課題と本研究の貢献

大きな目標として，自然言語で記述されたプロトコルの仕様と成り立つべき安全性から自動検証ツールの言語で記述された仕様と安全性を生成し検証作業を自動化したい．このための課題としてプロトコル仕様の記述を与えることと安全性の記述を与えることの 2 点がある．

本研究ではこのうち前者の課題の解決に向けた予備的な検討として，LLM に基づくチャットボットである ChatGPT を利用し比較的単純なプロトコルについて自然言語による仕様記述を元にツールの言語での記述を生成する実験を行った．その際，他のプロトコルの自然言語での記述とツールでの記述を与えてこれに倣ってツールでの記述を生成させた．その結果，単純なプロトコルについては文法エラーおよび wellformedness エラーがない記述を得ることができた．しかしプロトコルの参加者が持つ内部状態については適切な記述ができないため，修正が必要であるという課題が残されている．

より具体的には Yahalom のプロトコル [9] を対象にして記述の生成を行い，その際に例として与えるプロトコルとして Needham-Schroder の共通鍵認証プロトコル [10] を用いた．記述する言語として，Tamarin Prover の記述言語を用いた．さらに，より複雑なプロトコルとして Kerberov V プロトコル [11] についても記述の生成を行った．

$A \to B : A, Na$
$B \to S : \mathsf{senc}(\langle A, Na, Nb, B\rangle, Kbs)$
$S \to A : \mathsf{senc}(\langle B, Kab, Na, Nb\rangle, Kas), \mathsf{senc}(\langle A, Kab\rangle, Kbs)$
$A \to B : \mathsf{senc}(\langle A, Kab\rangle, Kbs), \mathsf{senc}(Nb, Kab)$

図 1 Yahalom のプロトコル
Fig. 1 The Yahalom Protocol

$A \to S : A, B, Na$
$S \to A : \mathsf{senc}(\langle Na, Kab, B, \mathsf{senc}(\langle Kab, A\rangle, Kbs)\rangle, Kas)$
$A \to B : \mathsf{senc}(\langle Kab, A\rangle, Kbs)$
$B \to A : \mathsf{senc}(Nb, Kab)$
$A \to B : \mathsf{senc}(Nb - 1, Kab)$

図 2 Needham-Schroeder の共通鍵認証プロトコル
Fig. 2 The Needham-Schroeder Symmetric-Key Protocol

2. 準備

2.1 暗号プロトコル

暗号プロトコルおよびその形式検証については文献 [12], [13] を参照されたい．本研究で扱った Yahalom のプロトコルおよび Needham-Schroeder の共通鍵認証プロトコルをそれぞれ**図 1** および**図 2** に示す．

ここで Na および Nb はノンス（予測不能な乱数）であり，これらのプロトコルの実行ごとに生成される．Kas および Kbs はそれぞれ参加者 A および B が認証サーバ S と共有している鍵である．senc は共通鍵暗号による暗号化，$\langle \_, \_ \rangle$ はメッセージの連結を表す．$A \to B$ などはその右側のメッセージを送信することを表す．

自然言語によるプロトコルの仕様記述は，これらの図のようなフローの図（アリスボブ記法）の他に，上記のような自然言語による記述も含まれる．本研究では自然言語によるプロトコル仕様としてこれらのプロトコルを説明した Wikipedia のエントリーを利用した．

2.2 プロトコル仕様の形式的記述

本研究では検証ツールとして Tamarin Prover を取り上げる．代表的なツールとして ProVerif があるが ProVerif ツールの言語では暗号文の復号や署名の検証の操作を復号や検証を行う関数を用いて明示的に記述する必要がある．その一方で Tamarin Prover の言語では単に暗号文 $\mathsf{senc}(M, K)$ の受信を記述すればよく，復号の操作が暗黙に行われる．このため LLM による抽出がより容易であると考えられる．

Needham-Schroeder の共通鍵認証プロトコルを Tamarin Prover を用いて検証するための記述の一部を**図 3** に示す．この記述は Tamarin Prover に記述例として同梱されているものを元に安全性記述に関する部分を削除して作成した．プロトコルはルールの集まりとして記述される．ルールは，[前提] --[ラベル]-> [結論] という形で記述される．ラベルはルールの適用を制御したり，安全性の記述のために用いられる．本研究はプロトコル記述にフォーカスするため，ラベルは基本的に空であるとする．

```
rule Init:
  [ Fr(~kxs)]
  --[ KeyGen($X) ]->
  [ !LongtermKey(~kxs,$X) ]

rule A_to_S:
  [ Fr(~na), !LongtermKey(~kas,$A) ]
  --[ ]->
  [ Out(<'1', $A, $B, ~na>),
      StateA1($A, $B, ~kas, ~na) ]

rule S_to_A:
  let msg =
    senc(<'2', na, y, ~kab,
          senc(<'3',~kab,x>,~kbs)>,~kas) in
  [ In(<'1', x, y, na>), !LongtermKey(~kas,x),
    !LongtermKey(~kbs,y) , Fr(~kab) ]
  --[ ]->
  [ Out(msg), !ShorttermKey(~kab,x,y) ]

rule A_to_B_1:
  let msg = senc(<'2', ~na, $B, kab, mb>,~kas) in
  [ StateA1($A, $B, ~kas, ~na),
    !LongtermKey(~kas,$A), In(msg) ]
  --[ ]->
  [ Out(mb), StateA2($A, $B, ~kas, ~na, kab, mb) ]
```

図 3 Tamarin Prover のための Needham-Schroeder の共通鍵認証プロトコルの記述の一部
Fig. 3 Parts of a Tamarin script for the Needham-Schroeder Symmetric-Key Protocol

図に示したルールのうち 1 つ目のものは共通鍵暗号の事前共有についてのルールである．このルールは，任意の鍵 ~kxs が新たに生成されたものであるとき，それを参加者 X とサーバの事前共有鍵であるとする，というルールである．ここで，Fr および LongtermKey はそれぞれデータの生成と事前共有鍵であることを表す述語である．

図に示したツールのもう一つのものは，参加者 A がサーバに最初のメッセージを送るというルール表す．Out はメッセージがネットワーク上に送信されたことを表す述語である．StateA1 は参加者 A の状態を表す述語であり，その引数にはそのプロトコルで用いている情報が与えられる．

プロトコルの実行は，述語によって記述される Fr(~kxs) や Out(<'1', $a, $B, ~na>) などのファクト（原始論理式）の多重集合の書き換えによってモデル化される．初期状態では空な多重集合があり，ルールに適用によってその要素が書き換えられていく．ただし，Fr については，Fr(~N) の形のファクトを生成するルールが Tamarin Prover にビルトインされている．!がついたファクトはファクトの無限個のコピーに相当し，書き換えによってなくなることはない．

プロトコル記述に求められる条件として，次の４点が必要となる．

- 文法エラーが無い
- wellformedness エラーが無い
- 実行可能性
- 意味的な正しさ（プロトコル仕様と合致している）

wellformedness とはルールの結論部に現れる変数（`~kxs`など）がルールの前提部に現れるなどの条件である．ただし，参加者の名前`$X$`などの公開情報はこの限りではない．実行可能であるとは，前述の通りルールを適用していくことで，プロトコルの実行を完了することを意味する．このためには，状態に関するファクトを適切に設定する必要がある．例えば，図のルール `A_to_S` の結論部に現れる `StateA1($A, $B, ~kas, ~na)` はルール `A_to_B_1` の前提部にも同じものが現れる．これによって，ルール `A_to_S`，すなわちプロトコルの最初のメッセージを送信した後の状態において，ルール `A_to_B_1` を適用して，2 番目のメッセージを受信して 3 番目のメッセージを送信することができる．文法エラーと wellformedness エラーについては生成されたスクリプトを Tamarin Prover に入力することでチェックできる．実行可能性についても，スクリプトに手を加えることで自動的なチェックが可能である．詳細は Tamarin Prover とともに配布されているスクリプト例における executability の証明を参照されたい．また，意味的な正しさは，人手により確認するほかないと考えられる．

2.3 ChatGPT

本研究では LLM に基づくチャットボットとして ChatGPT（GPT-4o）を用いた．ChatGPT はテキストによる指示（プロンプト）を与えることでテキストを生成可能なチャットボットであり，テキストの他にファイルを与えて解析させることができる．また，出力された情報について追加の指示を与えて出力を修正させたり，追加の情報を出力させることができる．本研究では ChatGPT で利用可能な LLM として現時点で最も高性能である GPT-4o を用いた．一般に LLM の出力には誤りが含まれる可能性があるため，出力の適切性を別途確認する必要がある．ChatGPT および指示の与え方については文献 [14] などを参照されたい．

3. LLM を用いたプロトコル記述生成

Tamarin Prover で暗号プロトコルを検証するためのスクリプトを，ChatGPT を用いて生成することを試みる．まず，生成が容易であると考えられる Yahalom プロトコルの生成を試み，次により複雑で仕様の規模も大きい Kerberos V[11], [15] のプロトコルの生成を試みる．

3.1 Yahalom プロトコルの記述の生成

Yahalom プロトコルを検証するためのスクリプトの生成を試みる．ChatGPT に与える自然言語によるプロトコル記述として Wikipedia に記載のプロトコル記述 [16] を用いた．Wikipedia の記述は自然言語により簡潔に記述されており図 1 に相当するプロトコルのフローも記載されていることから，Tamarin Prover のスクリプトに近い形の情報が含まれており生成が容易であると考えられる．生成の方法として単に Yahalom プロトコルの自然言語による記述のみを与えた場合と自然言語からスクリプトを生成する例を与えてそれに倣って生成させる場合を試行した．

3.1.1 プロトコルのみを与える場合

指示として「以下は Yahalom プロトコルの説明である．このプロトコルを Tamarin Prover のスクリプトとして記述せよ」を与え，さらに Yahalom プロトコルの説明を与える．このとき Tamarin Prover のスクリプトの文法は与えないため ChatGPT が持つ情報を利用することになる．この場合 Tamarin Prover 風のスクリプトを出力するが，Tamarin Prover の文法には存在しない宣言や構文を含むスクリプトが出力される．出力の内容としてはプロトコルのフローに概ね沿った内容になっているもののメッセージの送信や受信を表す述語が用いられていないなどの誤りを含む．

3.1.2 記述例を与える場合

文法の誤りを避けるため類似のプロトコルを Tamarin Prover で検証するためのスクリプトを与え，それに倣って記述させる．具体的には「以下に【Yahalom プロトコルの説明】,【Needham Schroeder 秘密鍵認証プロトコルの説明】,【Needham Schroeder 秘密鍵認証プロトコルの Tamarin Prover のためのスクリプト】を示す．Needham Schroeder 秘密鍵認証プロトコルの説明からの Tamarin Prover のためのスクリプトの記述を参考に Yahalom プロトコルの Tamarin Prover のためのスクリプトを記述せよ.」という指示と共に Yahalom プロトコルの説明，Needham Schroeder 秘密鍵認証プロトコルの説明およびその Tamarin Prover のためのスクリプトを与えた．Needhham-Schroeder の共通鍵認証プロトコルを用いた理由は，それが Yahalom プロトコルと同様，共通鍵暗号を用いた認証鍵交換プロトコルであるためである．Needhham-Schroeder の共通鍵認証プロトコルの自然言語により記述は Wikipedia に記載の内容 [17] を用いた．

得られた出力について文法エラーと wellformedness エラーについてまず確認した．この場合文法エラーは顕著に少なくなり，Tamarin Prover に入力しても文法エラーは報告されなくなった．しかし wellformedness エラーが報告された．具体的には事前共有鍵を用いた暗号文が出現するルールにおいてルールの前提部にルールの前提部に現れない変数がルールの結論部に使用されている場合が検出され

た．原因は，そのルールが実行される際の状態を表すファクト（図 3 における `StateA1($A, $B, ~kas, ~na)` など）が欠けていることにあった．このため，「参加者の動作を表すルールの結論部に”State”で始まるファクトがあった場合に，その参加者の次の動作を表すルールの前提部に同じファクトが現れるようにせよ．ただし，それぞれの参加者の動作を表す最初のルールの前提部には”State”で始まるファクトは現れない.」という指示によって状態を表すファクトの追加を試みた．その結果 wellformedness エラーが部分的に解消された．

次に意味的な正しさについて確認した．次の 3 点の問題点があった．

- メッセージの 2 つ組を送信する箇所で一つずつメッセージを送信している
- 状態を表すファクトの使用が不適切
- 復号できない暗号文を復号できるものとして扱っている

メッセージの組の送受信については，「1 つのルールで 2 つのメッセージを送信する（2 つ Out がある）ものについては 1 つの Out でメッセージの組を送信するようにせよ．メッセージの組を送信する場合それを受信するルールでもメッセージの組を受信するようにせよ.」という指示により修正できた．状態を表すファクトの使用が不適切であるとは，サーバ S のメッセージ送信を表すルールにおいて，参加者 A の状態を表すファクトを使用しているなどである．指示により修正を試みたが，汎用性のある指示で修正することはできなかった．復号できない暗号文の扱いについては，具体的に，3 番目のメッセージ受信した参加者 A がメッセージに含まれる参加者 B の鍵で暗号化された暗号文 $\mathsf{senc}(\langle A, Kab\rangle, Kbs)$ を復号してしまうということである．期待される動作としては 2 つ目の暗号文は未知のものであるとしてこの部分は `In(senc(<$B, ~Kab, ~Na, ~Nb>, ~Kas), M)` のように暗号文を記述す代わりに変数 `M` で表す必要がある．これについても指示を与えて修正を試みたが，汎用性のある指示で修正することはできなかった．

上記の通り，状態を表すファクトや暗号文の複合については正しく記述することができなかったものの，送受信されるメッセージについては追加の指示により適切に記述することができた．プロトコルの記述を開始する際の雛形としては用いることができると考えられる．また，修正のための適切な指示を見つけることができなかった問題点についても，そのような指示が存在する可能性はあると考えられる．

3.2 Kerberos V プロトコルの記述の生成

より複雑なプロトコルとして Kerberos V を取り上げ，Tamarin Prover のスクリプトの生成を試みる．いくつかの試行を行ったがいずれの場合も文法エラー，wellformedness エラー，実行可能性については Yahalom プロトコルと同様の状況であったため意味的にプロトコルがどの程度正しく記述されているかに着目する．

ChatGPT への指示として「以下に【Needham Schroeder 共通鍵認証プロトコルの説明】,【Needham Schroeder 共通鍵認証プロトコルの Tamarin Prover のためのスクリプト】を示す．Needham Schroeder 共通鍵認証プロトコルの説明からの Tamarin Prover のためのスクリプトの記述を参考に RFC4120 の記載を元に，RFC4120 に記載のプロトコルの Tamarin Prover のためのスクリプトを記述せよ.」との指示を与えて生成させた．

生成されたプロトコルは概ね Kerberos V のフローに従うものであったがメッセージの内容については誤りを含む．例えばクライアントが認証サーバから受け取るメッセージ（KRB_AS_REP メッセージ）に含まれるチケットと呼ばれる情報はクライアントが所持する鍵では暗号化されないが，生成されたスクリプトではクライアントが所持する鍵で暗号化されていた．RFC4120 との対応を明らかにするため,「生成されたスクリプトの各メッセージに対応する，RFC4120 の記載を抜粋せよ.」との指示を与えた．その結果 RFC における対応するセクション番号は正しいものを返したが, 抜粋された内容は RFC に存在しないものが出力された．また「KRB_AS_REP メッセージのフォーマットについて記載した箇所を抜粋せよ.」との指示を与えたところ, 該当する箇所を特定できなかった．実は RFC においては KRB_AS_REP メッセージの定義は KRB_TGS_REP メッセージとフォーマットが共通化されており, KRB_KDC_REP として定義されている．このため「KRB_AS_REP メッセージのフォーマットは KRB_KDC_REP メッセージとして定義されている．これを踏まえてスクリプトを修正せよ.」との指示を与えたところ，下記のメッセージとともに該当箇所が修正された．「AS to Client Rule: The message now reflects the KRB_KDC_REP format, which includes an encrypted part for the client (enc_part) and the Ticket_tgs (ticket for the TGS).」

このように ChatGPT を用いて文法エラーや不正確な箇所は残るもののスクリプトを生成でき，不正確な箇所についても指示を与えることである程度生成できることがわかった．

4. おわりに

本研究では，自然言語で記述された暗号プロトコルの仕様から，プロトコルの自動検証ツールのためのプロトコル記述を自動的に生成するという課題に対して予備的な検討を行った．具体的には LLM（ChatGPT，GPT-4o）を利用し比較的単純なプロトコルについて自然言語による仕様記述を元にツールの言語での記述を生成する実験を行った．

その際，他のプロトコルの自然言語での記述とツールでの記述を与えて，これに倣ってツールでの記述を生成させた．

その結果，単純なプロトコルについて，自然言語での記述においてもメッセージフローを明示している場合については，送受信されるメッセージの内容についてはある程度正確な記述が得られることがわかった．ただし，参加者の持つ状態についての記述は不正確であり，追加の指示を与えても修正できなかった．より複雑なプロトコルについて自然言語での記述として RFC をそのまま利用させる場合は，メッセージの内容についても不正確な箇所があった．しかし追加の指示により生成されたスクリプトと RFC との対応を出力させ，それが不適切であった場合に追加の指示を与えることで修正が可能であった．

今後の課題として，状態を表すファクトを適切に出力される方法を見つけることと，生成されたスクリプトを Tamarin Prover に入力した時にエラーが報告される場合に ChatGPT に与える追加の指示を自動的に作成することが挙げられる．本研究では Tamarin Prover のスクリプトを直接出力させたが，より生成しやすい中間言語で生成させそれを Tamarin Prover のスクリプトに変換する方法も検討に値する．なぜなら Tamarin Prover のスクリプトは送信するルールと受信するルールで同じ形のメッセージや状態のファクトを記述することになるため冗長性があり，ChatGPT の出力は冗長部分が整合しない場合があるためである．

より複雑なプロトコルに対してもスクリプトが正しく生成できるようにすることも課題である．逆に，そのような生成が可能であるよう，プロトコルの標準化において自然言語における記述を工夫して記述するということも考えられる．

参考文献

[1] Ristić, I.: *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）．

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

[3] Tamarin Prover, `https://tamarin-prover.com/`.

[4] Verifpal, `https://verifpal.com/`.

[5] Sharma, P. and Yegneswaran, V.: PROSPER: Extracting Protocol Specifications Using Large Language Models, *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, HotNets '23, New York, NY, USA, Association for Computing Machinery, p. 41–47 (online), DOI: 10.1145/3626111.3628205 (2023).

[6] ChatGPT, `https://chatgpt.com/`.

[7] Liu, Y., Xue, Y., Wu, D., Sun, Y., Li, Y., Shi, M. and Liu, Y.: PropertyGPT: LLM-driven Formal Verification of Smart Contracts through Retrieval-Augmented Property Generation, `https://arxiv.org/abs/2405.02580` (2024).

[8] Tihanyi, N., Jain, R., Charalambous, Y., Ferrag, M. A., Sun, Y. and Cordeiro, L. C.: A New Era in Software Security: Towards Self-Healing Software via Large Language Models and Formal Verification, `https://arxiv.org/abs/2305.14752` (2024).

[9] Burrows, M., Abadi, M. and Needham, R.: A logic of authentication, Technical Report SRC Researh Report 39, Digital Equipment Corporation Systems Research Center (1989).

[10] Needham, R. and Schroeder, M. D.: Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM*, Vol. 21, No. 12, pp. 993–999 (1978).

[11] Neuman, B. C. and Ts'o, T.: Kerberos: an authentication service for computer networks, *IEEE Communications Magazine*, Vol. 32, No. 9, pp. 33–38 (online), DOI: 10.1109/35.312841 (1994).

[12] 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).

[13] 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).

[14] 岡　瑞起，橋本康弘：プロンプトリテラシー，翔泳社 (2024).

[15] Neuman, C., Yu, T., Hartman, S. and Raeburn, K.: The Kerberos Network Authentication Service (V5), RFC 4120, RFC Editor (2005).

[16] Yahalom (protocol), `https://en.wikipedia.org/wiki/Yahalom_(protocol)`.

[17] Needham–Schroeder protocol, `https://en.wikipedia.org/wiki/Needham%E2%80%93Schroeder_protocol`.