LLM-PF: A LLM-Based Framework for Cryptographic Protocol Flow Extraction via a Novel IR towards Formal Verification

Anonymous ACL submission

Abstract

Formal verification is a robust and proven method for ensuring cryptographic protocol security, yet constructing formal models from natural language specifications remains a labor-intensive task demanding specialized expertise. While Large Language Models excel at text understanding, they struggle to directly generate accurate formal models due to the intricate logic and ambiguities in protocol descriptions. To bridge this gap, we propose a novel intermediate representation based on finite state machines. This representation deconstructs protocol logic into modular components, enabling LLMs to extract information step-by-step while ensuring a seamless and complete conversion to formal models. Building on this design, we developed a comprehensive extraction framework to automate the process. Extensive evaluations on a diverse set of protocols, ranging from basic to large-scale industrial standards, demonstrate the effectiveness and potential of our approach in automating formal verification.

1 Intruduction

Formal analysis techniques enable rigorous security verification of cryptographic protocols that underpin secure communication systems by constructing precise models and applying mathematical reasoning (Goldreich, 2003; Meadows, 2003). Their reliability critically depends on accurately translating natural-language protocol specifications into unambiguous formal representations. However, such translation is inherently challenging, as protocol specifications are often written in informal prose, describe complex protocol flows and state transitions, and contain implicit assumptions that are difficult to formalize.

Given the difficulty of this translation task, recent advances in Large Language Models (LLMs) have attracted increasing attention as a potential means of assisting protocol understanding and formal modeling. LLMs have demonstrated strong capabilities in text comprehension, and recent studies show that they can help interpret cryptographic protocol specifications and generate candidate formal models for subsequent analysis (Mao et al., 2025; Curaba et al., 2024; Duclos et al., 2024). Despite this promise, existing LLM- based approaches remain limited in practice.

One key limitation arises from the structural characteristics of large-scale cryptographic protocol standards. Such documents are lengthy and informationally dispersed, with non-sequential dependencies that scatter protocol logic across multiple sections. Existing work attempts to cope with this complexity either by relying on simplified descriptions, such as Alice&Bob narratives (Basin et al., 2015; Keller and Basin, 2014; Curaba et al., 2024; Li et al., 2025), or by designing specialized pipelines for direct document analysis (Mao et al., 2025; Duclos et al., 2024). While effective for small protocols, both strategies struggle to scale to large and complex specifications.

A second limitation concerns factual hallucination stemming from LLMs' lack of domain-specific protocol knowledge (Huang et al., 2025). When processing large-scale protocol specifications with rich state spaces and intricate execution paths, LLMs may introduce incorrect assumptions, omit critical conditions, or generate inconsistent transitions. Although mitigation techniques such as fine-tuning or repair modules have been explored, hallucination remains a persistent issue, particularly in settings that require precise reasoning about protocol states and message dependencies (Mao et al., 2025).

To address these challenges, we propose LLM-PF, an LLM-driven framework for extracting protocol flows from natural-language specifications using an Intermediate Representation (IR). Central to our approach is the Protocol Finite State Machine (PFSM), a finite-state-machine-based representation that explicitly captures protocol roles,

states, transitions, and message dependencies. By structuring protocol knowledge into the PFSM, our framework decouples the extraction process from downstream formal verification tasks. This design significantly mitigates hallucination and enables the scalable analysis of large, complex protocols.

Our approach draws inspiration from prior work on extracting state machines from network protocols (Sharma and Yegneswaran, 2023; Miao et al., 2025; Pacheco et al., 2022). However, adapting these methods to cryptographic protocols requires addressing fundamental distinctions (Hazzazi et al., 2023). Unlike network protocol state machines, which primarily focus on state transitions, our PFSM must explicitly describe cryptographic operations and their interaction relationships. This additional semantic complexity is essential for rigorous formal analysis and distinguishes our representation from conventional network models.

The main contributions of this paper are summarized as follows:

- **We introduce PFSM,** a novel intermediate representation based on Finite State Machine (FSM) that is LLM-friendly and supports lossless translation into formal verification languages such as ProVerif (Blanchet, 2012) and Tamarin(Meier et al., 2013). We further release a curated dataset of 10 real-world cryptographic protocols annotated in PFSM.

- **We develop an automated extraction framework** that leverages PFSM to process large-scale protocol specifications. The framework integrates workflow decomposition and enhanced retrieval to improve extraction accuracy, with experimental results demonstrating its effectiveness on large and complex protocols.

- **We design a toolchain** that automatically converts PFSM into formal verification code and protocol sequence diagrams (e.g., PlantUML (Roques and Contributors, 2025) and Mermaid (Sveidqvist and Contributors to Mermaid, 2014)).

We open-source our dataset and code for future research and reproducibility [1].

[1] https://anonymous.4open.science/r/Anonymous-LLMPF-975D

2 Related Works

We review related work from the perspective of how protocol specifications are translated toward formal analysis. In particular, prior research differs in both the form of input it assumes and the techniques it aims to achieve. Based on these dimensions, existing approaches can be broadly grouped into three categories, which we discuss in the following subsections.

Direct Conversion from Simplified Inputs. This line of work focuses on translating high-level, structured descriptions into formal models. Recent LLM-based approaches like CryptoFormalEval and P2FGPT convert Alice&Bob style inputs into models for Tamarin and ProVerif (Curaba et al., 2024; Li et al., 2025). Earlier work, such as Keller's A&B language, follows a similar structured input paradigm (Basin et al., 2015; Keller and Basin, 2014), while approaches using simple natural language descriptions have also been explored (Duclos et al., 2024). However, these methods suffer from poor scalability and generalization when facing the complexity and ambiguity of real-world protocol RFCs.

IRs for Protocol Analysis and Verification. This line of work addresses protocol complexity by extracting intermediate representations from RFCs. A common approach is to derive structured models, such as finite state machines (FSMs), for tasks including vulnerability analysis and attack synthesis (Sharma and Yegneswaran, 2023; Goo et al., 2019; Al Ishtiaq et al., 2024).

These IRs are typically designed to capture high-level state transitions while abstracting away cryptographic operations and message-level semantics (Hazzazi et al., 2023). As such, they are well-suited for exploratory analysis, but are not intended to directly support automated formal verification. Nevertheless, the role of IRs as a bridge toward formal reasoning is widely recognized. For example, MetaCP introduces an XML-based IR that can be accurately translated into multiple formal tools, although it relies on user-provided designs rather than parsing RFCs (Arnaboldi and Metere, 2019).

End-to-End RFC to Formal Model via IR. This line of work focuses on constructing end-to-end pipelines that translate RFCs into formal models via an intermediate representation. For example, Mao et al. (2025) proposed an LLM-assisted approach based on a lambda-calculus IR or translation into SAPIC+.

3 PFSM: An FSM-Based Protocol IR

To bridge the gap between the linguistic capabilities of LLMs and the precision required by formal models, we introduce PFSM, a FSM-based IR that balances comprehensibility for LLM-assisted extraction with convertibility to formal models.

3.1 Design Principles

The architecture of PFSM is guided by three core tenets, which collectively address the limitations of prior work:

Expressive Power through State-Driven Modeling: PFSM harnesses the intrinsic state-transition mechanism of FSMs to natively represent the intricate logic of cryptographic protocols, including conditional branching and complex flows, thus scaling to sophisticated systems.

Scalable Extraction via Modular Decomposition: We partition protocols into atomic state units, transforming monolithic descriptions into a modular architecture. This enables a divide-and-conquer strategy for LLMs to process states incrementally, mitigating context-window limits and enhancing fidelity for large-scale protocols.

Seamless Convertibility through Structural Congruence: PFSM semantics align with formal verifiers like ProVerif and Tamarin. States encapsulate knowledge and actions, while transitions dictate evolution. This ensures a direct, structurally isomorphic mapping that guarantees conversion completeness and soundness.

3.2 Core Syntax and Design Rationale

At its core, a protocol PFSM $\mathcal{P}$ is defined as:

$$\mathcal{P} = (\mathcal{E}, \mathsf{Ks}, \mathsf{M}) \tag{1}$$

where $\mathcal{E}$ is the finite set of participating entities, Ks specifies the initial knowledge distribution, and M assigns a finite state machine to each entity.

The initial knowledge distribution function:

$$\mathsf{Ks} : \mathcal{K}_{init} \mapsto 2^{\mathcal{E}} \tag{2}$$

maps each piece of initial knowledge to the subset of entities that process it. This explicit representation provides a clear blueprint for protocol initialization and supports consistent translation into formal verification models. In practice, such knowledge assignments can be instantiated as parameters, fresh values, or persistent facts in mainstream formal tools, reducing ambiguity introduced by informal specifications.

Each entity $e \in \mathcal{E}$ is associated with a unique finite state machine FSM_e, defined by the assignment function $\mathsf{M} : \mathcal{E} \mapsto \mathsf{FSM}$. The behavior of each entity e is captured by:

$$\mathsf{FSM}_e = \{\mathsf{FSM}_{e,s} \mid s \in S_e\} \tag{3}$$

where S_e denotes the finite set of state identifiers for e.

Each state unit $\mathsf{FSM}_{e,s}$ is a tuple:

$$\mathsf{FSM}_{e,s} = (K_{e,s}, O_{e,s}, \theta_{e,s}) \tag{4}$$

where $K_{e,s}$ is the knowledge set held by entity e while in state s. $O_{e,s}$ denotes the set of actions executable upon entering s, and $\theta_{e,s}$ defines the transition rules to subsequent states. State transitions are triggered by message actions, or by decision-based branch conditions.

This action-driven granularity is a deliberate design choice. By organizing protocol behavior around explicit actions rather than informal narratives, PFSM enables modular decomposition of complex protocols into atomic, analyzable units. Such decomposition supports a divide-and-conquer extraction strategy, allowing LLMs to process protocol behavior incrementally and mitigating context-window limitations when handling large specifications.

Moreover, anchoring state transitions to explicit actions aligns PFSM with the operational abstractions used by formal verification tools. Sequences of actions within a state can be mapped to process interactions in the π-calculus for ProVerif (Abadi et al., 2017), while the encapsulated structure of state units aligns with the rule-based modeling style of Tamarin (Meier et al., 2013). This structural congruence supports systematic translation to formal code and reduces reliance on ad hoc interpretation during model construction.

Overall, the core syntax of PFSM balances expressiveness, modularity, and convertibility. By making protocol control flow, knowledge evolution, and actions explicitly at the state level, PFSM provides a representation that is well suited for LLM-Driven extraction and downstream formal verification.

Appendix A details the translation semantics from PFSM syntax to ProVerif, and provides a simple illustrative example.

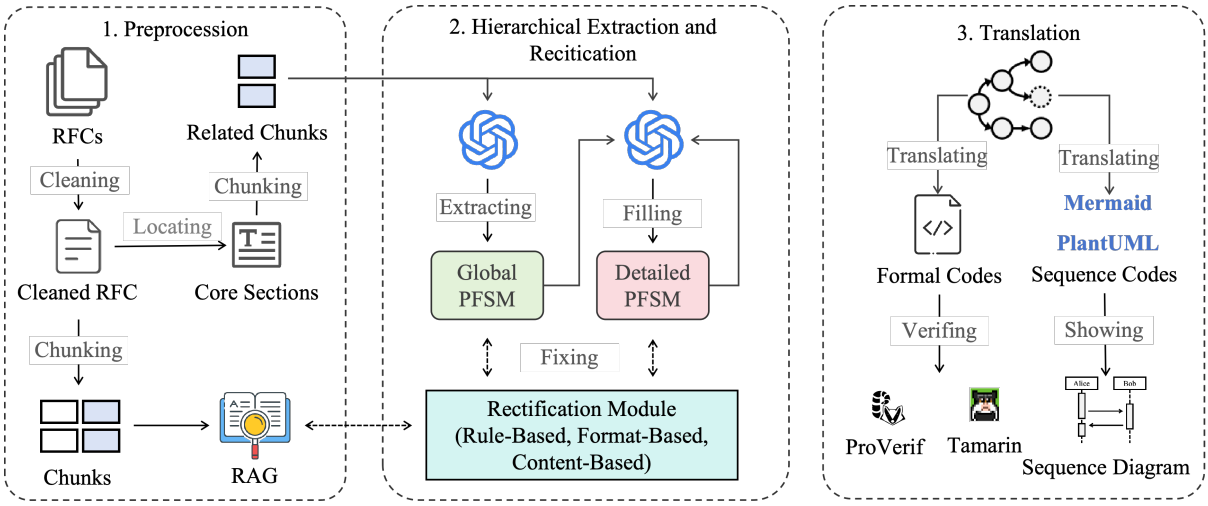

Figure 1: Overall framework of the LLM-PF

4 PFSM Extraction Framework

This chapter presents a systematic hierarchical extraction framework designed to automatically construct PFSMs from natural language specifications. The overall pipeline is illustrated in Figure 1:

4.1 Preprocessing

To address the challenges posed by lengthy and fragmented protocol specifications, which exceed the effective context window and comprehension capabilities of LLMs, a preprocessing phase is first employed to distill and structure the raw documents.

Standardization and Data Cleaning. We first employ document parsing tools to convert multi-source heterogeneous documents into a unified Markdown format. Subsequently, we utilize regular expressions combined with heuristic rules to eliminate explicit noise, such as page numbers, hyperlinks, and images, ensuring that the input data is clean and consistent.

Core Section Localization. Cryptographic protocol specifications typically exhibit a rigid structure where descriptions of protocol logic are concentrated in specific sections—for instance, the *Protocol Details* in UAF (FIDO Alliance, 2017) or the *Authentication Framework* in 5G (3GPP, 2022). Conversely, auxiliary sections such as the introduction, definitions, security proofs, and appendices, while providing context, are largely irrelevant to the control flow. Due to the limitations of LLMs in precisely locating information within massive unstructured text (Hengle et al., 2025; Liu et al., 2025b), our framework employs semantic recognition to identify these core sections, designating them as primary input for PFSM generation.

Semantic Chunking. To address LLM context limits, we chunk localized protocol sections. Simultaneously, we construct a vector knowledge base via fine-grained chunking of the entire specification to mitigate cross-chapter information loss. This allows the subsequent rectification algorithm to retrieve missing contextual definitions via Retrieval-Augmented Generation (RAG) (Lewis et al., 2020), ensuring complete state descriptions.

4.2 Hierarchical Extraction and Rectification

Extracting PFSM from unstructured protocol specifications presents three key challenges for LLMs.

C1: Noise interference in entity identification. Protocols often contain non-functional components (e.g., PKI or databases) that models may misclassify as participants. Furthermore, physically isolated components described in specifications should be abstracted into a single logical entity, increasing model comprehension complexity. For instance, this abstraction is required in the 5G AKA protocol, where the internal USIM and ME components of a User Equipment (UE) should be merged into one entity (3GPP, 2022).

C2: Complexity of state transitions. Protocol operations involve multi-branch logic and fine-grained cryptographic primitives, making it difficult for models to accurately parse all atomic operations and state flows in a single pass.

C3: Difficulty in capturing inter-entity dependencies. The core of a PFSM lies in the inter-entity dependencies established via message exchange; in multi-branch scenarios, correctly aligning states

across different entities often exceeds the foundational capabilities of current models.

To address these challenges, we propose a hierarchical extraction framework that decomposes the task to maximize the model's reasoning efficacy. The algorithm consists of four stages, as outlined in Algorithm 1.

Entity Identification (Addressing C1). This stage aims to filter non-functional noise and extract only participants $\mathcal{E}$ directly involved in protocol interactions. For ambiguously defined entities, we additionally introduce a human-in-the-loop mechanism to calibrate the entity list via expert feedback, ensuring the accuracy of the basic modeling units.

Global PFSM Extraction (Addressing C2). Upon confirming the entity set, we construct a global PFSM $\mathcal{P}_g$. $\mathcal{P}_g$ defines the macro-level state machine, specifying the name and description of each state for every entity $e \in \mathcal{E}$. It captures the state transition logic (i.e., the next state for a given entity) and explicitly encodes the inter-entity dependencies established via message exchange, specifically mapping each sending state to its corresponding receiving state. $\mathcal{P}_g$ serves as a blueprint for subsequent refinement, clearly delineating the protocol's topological structure.

Global PFSM Rectification (Addressing C3). To mitigate logical discontinuities in $\mathcal{P}_g$, we formulate a set of rules targeting specific integrity failures: *Content Integrity*, *Transition Integrity*, and *Interaction Consistency*. These rules explicitly identify violations—such as incomplete state fields, missing intermediate states, or unmatched interactions—and provide feedback to the LLM for iterative regeneration.

Detailed PFSM Filling and Rectification (Addressing C2). Based on the global skeleton $\mathcal{P}_g$, the algorithm proceeds to generate the detailed PFSM $\mathcal{P}$. To handle protocols that may exceed output token limits, we employ an iterative batch extraction strategy initialized with a batch size threshold t. As shown in Algorithm 1, the loop iteratively selects the next batch of states B via `KGetBatch`. Within each iteration, the `KFill` function constructs the detailed state information for B. This is immediately followed by the `KVerify` function, which initiates a verification and repair mechanism: rule-based checks validate the legality of atomic operations, and for any missing definitions or ambiguous parameters, RAG technology retrieves relevant information from the full document C to automatically complete the descriptions. The verified batch is then aggregated into $\mathcal{P}_d$. This cycle repeats until all states are processed, culminating in the final, accurate PFSM.

Algorithm 1: Hierarchical Extraction and Rectification Algorithm

Input: Chunks C, Threshold t
Output: Verified PFSM $\mathcal{P}$

$\mathcal{E} \leftarrow$ `ExtractEntities` (C)
$\mathcal{P}_g \leftarrow$ `GenGlobalPFSM`$(C, \mathcal{E})$
$\mathcal{P}_g \leftarrow$ `RectifyStruct`$(\mathcal{P}_g, C, \mathcal{E})$
$\mathcal{P}_d \leftarrow \emptyset$
while $(B \leftarrow$ `NextBatch`$(\mathcal{P}_g)) \neq \emptyset$ **do**
  $\mathcal{P}_t \leftarrow$ `FillDetails`$(\mathcal{P}_g, B, C)$
  $\mathcal{P}_t \leftarrow$ `VerifyAndFix`$(\mathcal{P}_t, C)$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_d$
end
$\mathcal{P} \cup \mathcal{P}_d$
return $\mathcal{P}$

4.3 Translation Module

To maximize the utility of the extracted PFSM, we developed a translation ecosystem supporting both formal verification and visualization. For rigorous security analysis, a high-performance translator converts the PFSM into specification languages (e.g., ProVerif) to verify properties like reachability (see Appendix A). Additionally, visualization translators (PlantUML, Mermaid) generate intuitive diagrams for inspection.

5 Experiments and Evaluation

This chapter presents a comprehensive evaluation of our proposed LLM-PF framework, guided by the following research questions:

- **RQ1**: How effectively does the LLM-PF framework extract PFSMs across protocols of varying scales?
- **RQ2**: To what extent does the LLM-PF framework outperform direct LLM baselines?
- **RQ3**: How do the hierarchical extraction and rectification mechanisms contribute to the validity and completeness of the generated models?

5.1 Dataset and Ground Truth

To ensure a robust analysis, we curated a dataset comprising 10 protocols ranging from

simple to complex benchmarks. The set centers on Needham-Schroeder Symmetric Key (NSSK) (1978) and Needham-Schroeder Public Key (NSPK) (1978), alongside their variants: Neuman-Stubblebine (Neuman) (1993), Yahalom (1990), Otway-Rees (OR) (1987), and Wide-Mouth-Frog (WMF) (1990). Beyond this group, we included the classic Kao-Chow (KC) (1995) protocol and three complex protocols: 5G AKA (2022), FIDO UAF (2017), and EDHOC (2021). Notably, for protocols lacking official RFC standards, we adapted specifications from the LLM-aided dataset (2025) and transformed them into RFC-style documents.

The ground-truth PFSM were manually constructed by domain experts based on original RFC documentation. For each protocol, we generated the corresponding formal verification code. To ensure consistency, these PFSMs were validated by automatically translating them into the written formal verification code. This rigorous methodology supports comprehensive end-to-end verification.

Table 1 provides an overview of the dataset, detailing the scale of the RFC documents, PFSMs, and the generated formal verification code.

Table 1: Overview of the Dataset Complexity

	Description		Rows		
	Roles	Rounds	RFCs	PFSM	ProVerif
NSSK (1978)	3	5	112	441	90
NSPK (1978)	2	3	88	296	64
NS (1993)	3	4	83	425	114
Yahalom (1990)	3	4	123	402	136
OR (1987)	3	4	99	433	118
WMF (1990)	3	2	66	252	89
KC (1995)	3	4	20	78	97
5G (2022)	4	11	470	1286	400
UAF (2017)	3	8	4010	1370	149
EDHOC (2021)	2	4	1428	3426	553

5.2 Evaluation Metrics

Given the extraction algorithm, we evaluate the LLM-PF in two dimensions.

Global State Machine Metrics. To evaluate the large model's capability in recognizing cryptographic protocol state transitions and to validate the effectiveness of our global state extraction algorithm, we employ standard metrics for the extraction task. We count the correctly predicted states, the false positives predicted in error, and the ground truth states that were missed. Based on these counts, we report Precision (P_g), Recall (R_g), and F1-score (F_g) as the primary evaluation metrics.

State Detail Metrics. To conduct a fine-grained assessment of the correctly identified states, we evaluate the fidelity of their internal attributes: the knowledge set K_s, the action set O_s, and the transition θ_s. For each attribute, we individually count the true matches, false positives, and false negatives. Specifically, knowledge correctness is assessed via the cosine similarity of textual descriptions followed by global substitution alignment; actions are evaluated by strict matching against defined function names and parameters; and transitions are determined by strict matching against defined transition rules. We calculate the Precision (P_d), Recall (R_d), and F1-score (F_d) for each attribute based on these counts and derive the overall metrics for the state details by taking the weighted average of these scores.

5.3 Results

We evaluated our method on four state-of-the-art LLMs: GPT-5.2 (OpenAI, 2025), Gemini-2.5 Pro (Comanici et al., 2025), DeepSeek-V3 (Liu et al., 2025a), and Grok-4 (xAI, 2025).

5.3.1 Overall Performance (RQ1)

Figure 2 illustrates the performance of the LLM-PF framework across a spectrum of cryptographic protocols, categorized by their complexity. For each protocol, the left subplot reports the P_g, R_g, and F_g for global state extraction, while the right subplot details the metrics P_d, R_d and F_d for detailed state extraction.

Validation and Generalization. The evaluation includes both lightweight protocols and industrial standards. For lightweight protocols, the extraction pipeline achieves precision consistently above 80%. While these results validate the fundamental soundness of the LLM-PF architecture on self-contained specifications, the primary objective of this work is to address the challenges posed by real-world industrial protocols. In this domain, the framework maintains moderate precision. This performance variation highlights the complexity of industrial-grade formalization, a task that remains challenging even for state-of-the-art models.

Findings on Industrial Protocol Complexity. Qualitative analysis reveals that performance on large protocols is constrained by both the inherent instability of LLMs and the structural ambiguity of technical specifications.

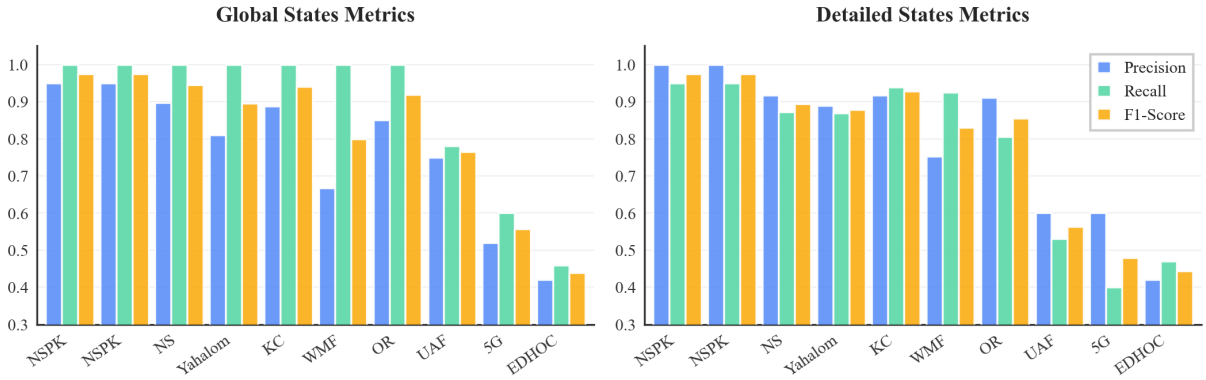

Figure 2: Overall Performance of LLM-PF across different protocols using the ChatGPT model.

Finding 1: Non-deterministic Generation. Although LLM-PF incorporates mechanisms to mitigate hallucinations, we observe notable variance in output quality across multiple runs, even with a temperature setting of 0.

Finding 2: Specification Structural Ambiguity. First, *cross-references* fragment logic; for instance, 5G AKA (p.52) defers AUTN verification to an external document ("as described in TS 33.102"), hindering the model's ability to infer the complete mechanism (3GPP, 2022). Second, *noisy examples* mislead extraction; in UAF (Section 3.4.1), concrete registration request examples cause the model to erroneously treat instance-specific data as part of the generic flow (FIDO Alliance, 2017). Third, *long-range dependencies* make error handling difficult to capture. In 5G AKA, exception handling is isolated in a "Void" section (6.1.3.2.1, p.54) (3GPP, 2022). However, our results confirm that the Content Inspection and Repair Mechanisms are effective in bridging these dependencies.

Finding 3: Combinatorial Branching. A distinct challenge arises from branching logic, as seen in EDHOC (Selander et al., 2021), which defines multiple mutually exclusive authentication modes. This complexity leads to a precision of approximately 40%, as the model struggles to maintain a coherent global state across disjointed branches. This suggests that future work should explore protocol pre-processing strategies, such as extracting different modes separately.

Model Capability Analysis. We further evaluated the framework's behavior across different LLM backbones in terms of F1-score, as shown in Figure 3. The results indicate that recent state-of-the-art models uniformly excel at understanding and identifying PFSMs for lightweight protocols. However, for large-scale industrial protocols, models such as Grok and ChatGPT demonstrate superior capability in extracting structured state machines compared to others. Despite the strong performance of these backbones, the structural challenges described above (e.g., the branching in EDHOC) (Selander et al., 2021) still pose significant hurdles, necessitating the specialized mechanisms within the LLM-PF framework to achieve viable results. Additionally, incomplete operational descriptions, such as the omitted SQN handling mechanism in 5G AKA (p.35) (3GPP, 2022), remain a bottleneck.

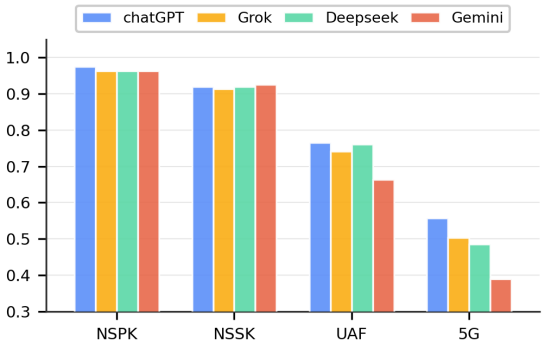

Figure 3: Comparison of F1-scores for global state machine extraction across different large language models (LLMs) and protocols.

5.3.2 Baseline Comparison (RQ2)

To evaluate the effectiveness of our framework, we established a strong baseline using direct prompt engineering: we instructed the LLM to generate the complete PFSM in a single pass using the raw protocol documents as input. Both the baseline and our LLM-PF framework were evaluated using the same automated metrics.

As shown in Figure 4, the comparative analysis

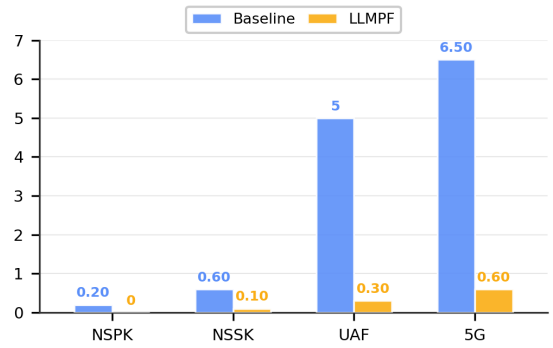

Figure 4: Comparison of average structural errors during global state machine extraction between LLM-PF and Baseline across different protocols.

reveals a clear dependency on protocol scale. For small-scale protocols (e.g., NSSK), the baseline demonstrates acceptable competence, and LLM-PF achieves only marginal improvements. This is primarily because the rectification module in LLM-PF corrects minor edge inconsistencies that are relatively sparse in simple protocols. However, in the context of large-scale protocols (e.g., 5G AKA), the baseline suffers a catastrophic drop in performance. This is due to the complexity of the PFSM structure and the difficulty of capturing inter-state dependencies and internal operations within the context window of a single forward pass. In this scenario, the advantage of LLM-PF's hierarchical design becomes significant. Notably, for protocols with extensive documentation, the baseline often exceeds the context window limits of the LLM, resulting in incomplete or null generations. In contrast, LLM-PF's modular design effectively manages context length and structural complexity, ensuring robust generation across all scales.

5.3.3 System Analysis (RQ3)

We conducted an ablation study to investigate the specific contributions of the global PFSM extraction and the rectification mechanism.

The Necessity of the Global PFSM. While the inability of LLMs to accurately identify all states in a single pass necessitates a decomposed strategy, a naive 'Chain Extraction' approach—extracting states sequentially based on predecessors—remains problematic. Although this method bypasses immediate context window limits, our analysis suggests that LLMs struggle to maintain coherence across multiple turns of dialogue. Specifically, such a sequential method fails to capture the migratory relationships between states effectively and suffers from the progressive accumulation of hallucinations along the extraction chain. Consequently, we prioritized the Global PFSM extraction, which enables the model to rapidly capture the overarching state transition logic and message interdependencies at once, providing a robust structural guide for subsequent detailed extraction.

The Impact of the Rectification Mechanism. To quantify the impact of the rectification module, we conducted 10 independent trials of global PFSM extraction and measured the average number of structural errors, as shown in Figure 4. For small-scale protocols, direct extraction proves relatively reliable (averaging 0.2 errors), and our module successfully eliminates these residual inaccuracies. In contrast, large-scale protocols exhibit instability in direct extraction, yielding an average of 5-7 structural errors per run. The introduction of the rectification module significantly mitigates these issues, reducing the average unresolved errors to just 0.3 - 0.6. This demonstrates that the feedback loop is essential for ensuring the structural validity of complex protocol models.

6 Conclusion

This paper investigates the application of LLMs to cryptographic protocol flow extraction. We propose a standardized protocol description that can be comprehensively transformed into formal models and sequence diagrams. Leveraging LLMs, we introduce a hierarchical extraction and rectification algorithm framework that decomposes the task, thereby enhancing the capability to extract large-scale protocols. Experimental results on 10 protocols of varying complexity and four mainstream LLMs indicate that although LLMs are feasible, their performance on complex flows remains suboptimal. Specifically, our framework significantly improves the capability of extracting cryptographic protocols compared to the baseline generated by directly using LLMs; however, we acknowledge that current limitations, such as the inability to recognize document cross-references, still exist. Future work will focus on addressing these issues, extending the evaluation to a broader range of protocols, and conducting security analysis based on the standardized descriptions.

Limitations

While our proposed framework demonstrates promising capabilities, its effectiveness is currently influenced by several factors. The reliance on large models presents practical challenges, including significant time and computational costs, and the approach has thus far been validated on a limited set of protocol types and scales, with extensive testing across diverse models and protocol categories remaining for future work.

Furthermore, while our framework mitigates hallucinations to a certain extent, it cannot entirely eliminate the LLM from drawing erroneous conclusions. We also observe that ensuring high stability of model outputs can be difficult; even with deterministic settings and our content completion mechanism—which mitigates many common errors—the system may not fully address rare, extreme failures. Additionally, the quality of the generated PFSM is heavily dependent on the clarity of the standard documentation. Omissions in protocol descriptions or dependencies that span multiple documents can significantly impair the accuracy of the generation results.

To address these inherent uncertainties and uphold the highest standards of reliability, we have incorporated a human-in-the-loop strategy. Although LLM-PF strives to minimize human intervention, the ultimate goal of formal verification requires bridging the gap between symbolic models and real-world scenarios. Consequently, certain details still necessitate expert determination to ensure the formal model accurately reflects protocol semantics, such as the identification of principal confirmations discussed in Section 5.3. This expert validation and correction are crucial to guarantee the integrity of the final formal specifications.

References

3GPP. 2022. 3GPP TS 33.501: Security architecture and procedures for 5g system. Technical Specification 3GPP TS 33.501. Release 17.

Martín Abadi, Bruno Blanchet, and Cédric Fournet. 2017. The applied pi calculus: Mobile values, new names, and secure communication. *Journal of the ACM (JACM)*, 65(1):1–41.

Abdullah Al Ishtiaq, Sarkar Snigdha Sarathi Das, Syed Md Mukit Rashid, Ali Ranjbar, Kai Tu, Tianwei Wu, Zhezheng Song, Weixuan Wang, Mujtahid Akon, Rui Zhang, and 1 others. 2024. Hermes: Unlocking security analysis of cellular network protocols by synthesizing finite state machines from natural language specifications. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4445–4462.

Luca Arnaboldi and Roberto Metere. 2019. Towards a data centric approach for the design and verification of cryptographic protocols. *arXiv preprint arXiv:1910.02656*.

David Basin, Michel Keller, Saša Radomirović, and Ralf Sasse. 2015. Alice and bob meet equational theories. In *Logic, Rewriting, and Concurrency: Essays Dedicated to José Meseguer on the Occasion of His 65th Birthday*, pages 160–180. Springer.

Bruno Blanchet. 2012. Security protocol verification: Symbolic and computational models. In *International conference on principles of security and trust*, pages 3–29. Springer.

Michael Burrows, Martin Abadi, and Roger Needham. 1990. A logic of authentication. *ACM Transactions on Computer Systems (TOCS)*, 8(1):18–36.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Cristian Curaba, D'Ambrosi Denis, and Alessandro Minisini. 2024. Cryptoformaleval: Integrating large language models and formal verification for automated cryptographic protocol vulnerability detection. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*.

Martin Duclos, Ivan A Fernandez, Kaneesha Moore, Sudip Mittal, and Edward Zieglar. 2024. Utilizing large language models to translate rfc protocol specifications to cpsa definitions. *arXiv preprint arXiv:2402.00890*.

FIDO Alliance. 2017. Fido uaf protocol specification. Proposed standard, FIDO Alliance. Accessed: 2026-01-04.

Oded Goldreich. 2003. Cryptography and cryptographic protocols. *Distributed Computing*, 16(2):177–199.

Young-Hoon Goo, Kyu-Seok Shim, Min-Seob Lee, and Myung-Sup Kim. 2019. Protocol specification extraction based on contiguous sequential pattern algorithm. *IEEE Access*, 7:36057–36074.

Mohammad Mazyad Hazzazi, Raja Rao Budaraju, Zaid Bassfar, Ashwag Albakri, and Sanjay Mishra. 2023. A finite state machine-based improved cryptographic technique. *Mathematics*, 11(10):2225.

Amey Hengle, Prasoon Bajpai, Soham Dan, and Tanmoy Chakraborty. 2025. Multilingual needle in a

haystack: Investigating long-context behavior of multilingual large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5165–5180.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and 1 others. 2025. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55.

I. Long Kao and Randy Chow. 1995. An efficient and secure authentication protocol using certified keys. *ACM Operating Systems Review*, 29(3):14–21.

Michel Keller and Prof Dr David Basin. 2014. Converting alice&bob protocol specifications to tamarin. *ETH Zurich*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474.

Qiang Li, Jihong Han, Lin Yuan, Xiangcheng Li, and Xiaoyu Wang. 2025. Constructing formal models of cryptographic protocols from alice&bob style specifications via llm. *Scientific Reports*, 15(1):11866.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Jiaheng Liu, Dawei Zhu, Zhiqi Bai, Yancheng He, Huanxuan Liao, Haoran Que, Zekun Wang, Chenchen Zhang, Ge Zhang, Jiebin Zhang, and 1 others. 2025b. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*.

Ziyu Mao, Jingyi Wang, Jun Sun, Shengchao Qin, and Jiawen Xiong. 2025. Llm-aided automatic modelling for security protocol verification. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 734–734. IEEE Computer Society.

Catherine Meadows. 2003. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE journal on selected areas in communications*, 21(1):44–54.

Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The tamarin prover for the symbolic analysis of security protocols. In *International conference on computer aided verification*, pages 696–701. Springer.

Zhang Miao, Feng Runhan, Tang Hongbo, Zhao Yu, Yang Jie, Qiu Hang, and Liu Qi. 2025. Automated extraction of protocol state machines from natural language specifications. *arXiv preprint arXiv:2510.14348*.

Roger M Needham and Michael D Schroeder. 1978. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999.

B Clifford Neuman and Stuart G Stubblebine. 1993. A note on the use of timestamps as nonces. *ACM SIGOPS Operating Systems Review*, 27(2):10–14.

OpenAI. 2025. Introducing gpt-5.2: A new generation of language models. https://openai.com/blog/gpt-5-2/. Accessed: 2026-01-05.

Dave Otway and Owen Rees. 1987. Efficient and timely mutual authentication. *ACM SIGOPS Operating Systems Review*, 21(1):8–10.

Maria Leonor Pacheco, Max von Hippel, Ben Weintraub, Dan Goldwasser, and Cristina Nita-Rotaru. 2022. Automated attack synthesis by extracting finite state machines from protocol specification documents. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 51–68. IEEE.

Arnaud Roques and PlantUML Contributors. 2025. PlantUML Software.

Göran Selander, John Mattsson, and Francesca Palombini. 2021. Ephemeral diffie-hellman over cose (edhoc). *Internet Engineering Task Force, Internet-Draft draft-ietf-lake-edhoc-09*.

Prakhar Sharma and Vinod Yegneswaran. 2023. Prosper: Extracting protocol specifications using large language models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*, pages 41–47.

Knut Sveidqvist and Contributors to Mermaid. 2014. Mermaid: Generate diagrams from markdown-like text.

xAI. 2025. Grok-4-0709: An overview. https://x.ai/news/grok-4. Accessed: 2026-01-05.

A Translator: From PFSM to ProVerif

A.1 Pipeline

Let the input protocol specification be a PFSM $\mathcal{P} = (\mathcal{E}, \{\mathsf{FSM}_e\}_{e\in\mathcal{E}})$, where $\mathcal{E}$ is the set of entities/roles and FSM_e is the local state machine of entity e. The translator outputs a ProVerif specification:

$$\mathsf{Proc}_e \triangleq \mathsf{CompileRole}(\mathsf{FSM}_e), \tag{5}$$

$$\mathsf{PV}(\mathcal{P}) \triangleq \mathcal{H} \cup \bigcup_{e\in\mathcal{E}} \mathsf{Proc}_e \cup \mathsf{Proc}_{\text{init}}. \tag{6}$$

where $\mathcal{H}$ is the fixed ProVerif header (types, channels, cryptographic primitives, etc.) and $\mathsf{Proc}_{\text{init}}$ is the top-level initialization process.

A.2 Abstraction

Each entity PFSM is abstracted as

$$\mathsf{FSM}_e \triangleq (S_e,\, s_e^0,\, O_e,\, \theta_e), \tag{7}$$

where S_e is the set of states, s_e^0 is the initial state, $O_e(s)$ returns the `onEntry` operation sequence at state s, and $\theta_e(s)$ returns the outgoing transitions at s (e.g., `AUTO/CONDITIONAL`).

Terms, patterns, and substitution. Terms and patterns are tuples built from variables and constants:

$$t ::= x \mid c \mid (t_1, \ldots, t_n),$$
$$p ::= x \mid c \mid (p_1, \ldots, p_n).$$

A substitution σ maps variables to terms; applying σ to t is written $t\sigma$. We write $\mathsf{match}(p, t\sigma) = \sigma'$ when p matches $t\sigma$ and yields bindings σ', otherwise $\mathsf{match}(p, t\sigma) = \bot$.

A.3 Event Language and Event Programs

Events. We abstract `onEntry` and transition actions into events:

$$\alpha \in \mathsf{Ev} ::= \mathrm{SEND}(ch, t) \mid \mathrm{RECV}(ch, p) \mid \mathrm{SPLIT}(t \Rightarrow \vec{x}) \mid \mathrm{OP}(x := f(\vec{t})).$$

Fresh-name generation is modeled as a dedicated event $\mathrm{RAND\_GEN}(x)$, which compiles to the ProVerif statement `new`. We use vector notation $\vec{x}$ and $\vec{t}$ to denote tuples (e.g., $\vec{x} = (x_1, \ldots, x_k)$ and $\vec{t} = (t_1, \ldots, t_n)$), and write $\vec{t}\sigma$ for $(t_1\sigma, \ldots, t_n\sigma)$.

Event programs. Role behavior is expressed as an event program:

$$P ::= \mathbf{0} \mid \alpha; P \mid \mathbf{if}\ \varphi\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2 \mid \mathbf{case}\ t\ \mathbf{of}\ \{p_i \Rightarrow P_i\}_{i=1}^{k}\ \ \mathbf{default} \Rightarrow P_0,$$

where φ is a boolean guard (from PFSM conditionals) and **case** uniformly captures "receive-then-discriminate" branching (e.g., by tag/shape).

A.4 Operational Semantics (Short Form)

Configuration and multiset scheduling. A configuration is

$$C \triangleq \langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle,$$

where $\mathcal{N}$ is the set of generated fresh names, $\mathcal{I}$ is a *multiset* of concurrently running instances, and σ is the current environment/substitution.

An instance is written $inst \triangleq \{\alpha; P\}$, meaning "execute event α then continue with P". We use multiset union $\uplus$, multiset difference $\setminus_m$, and multiset inclusion $\subseteq_m$.

We use the following multiset-update notation (replace one instance by its continuation):

$$\mathcal{I}[inst \mapsto P] \;\triangleq\; (\mathcal{I} \setminus_m \{inst\}) \uplus \{P\}.$$

Step relation. Each semantic step emits a *label* ℓ that is later mapped to ProVerif code:

$$\langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle \xrightarrow{\ell} \langle \mathcal{N}',\, \mathcal{I}',\, \sigma' \rangle.$$

Core rules.

$$\frac{inst = \{\mathrm{RAND\_GEN}(x); P\} \uplus \mathcal{I} \qquad n \notin \mathcal{N}}{\langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle \xrightarrow{\mathrm{new}(x)} \langle \mathcal{N} \cup \{n\},\, \mathcal{I}[inst \mapsto P],\, \sigma[x \mapsto n] \rangle} \;[\mathrm{GEN}]$$

$$\frac{inst = \{\mathrm{SEND}(ch, m); P\} \uplus \mathcal{I}}{\langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle \xrightarrow{\mathrm{out}(ch, m\sigma)} \langle \mathcal{N},\, \mathcal{I}[inst \mapsto P],\, \sigma \rangle} \;[\mathrm{SEND}]$$

$$\frac{inst = \{\mathrm{RECV}(ch, p); P\} \uplus \mathcal{I} \qquad \mathsf{match}(p, v) = \sigma'}{\langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle \xrightarrow{\mathrm{in}(ch, p)} \langle \mathcal{N},\, \mathcal{I}[inst \mapsto P],\, \sigma \cup \sigma' \rangle} \;[\mathrm{RECV}]$$

$$\frac{inst = \{\mathrm{SPLIT}(t \Rightarrow \vec{x}); P\} \uplus \mathcal{I} \qquad \mathsf{match}(\vec{x}, t\sigma) = \sigma'}{\langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle \xrightarrow{\mathrm{let}\ \vec{x} = t\sigma} \langle \mathcal{N},\, \mathcal{I}[inst \mapsto P],\, \sigma \cup \sigma' \rangle} \;[\mathrm{SPLIT}]$$

$$\frac{inst = \{\mathrm{OP}(x := f(\vec{t})); P\} \uplus \mathcal{I}}{\langle \mathcal{N},\, \mathcal{I},\, \sigma \rangle \xrightarrow{\mathrm{let}\ x = f(\vec{t}\sigma)} \langle \mathcal{N},\, \mathcal{I}[inst \mapsto P],\, \sigma[x \mapsto f(\vec{t}\sigma)] \rangle} \;[\mathrm{OP}]$$

A.5 Label-to-ProVerif Mapping (Code Generation)

We map each transition label ℓ to ProVerif code via $\Phi(\ell)$. We write $(\vec{x} : \vec{\tau})$ to denote $(x_1 : \tau_1, \ldots, x_k : \tau_k)$, where the types are declared in $\mathcal{H}$. The mapping Φ from labels to ProVerif statements is:

$$\begin{aligned}
\Phi(\mathrm{new}(x)) &\triangleq \texttt{new } x\texttt{: bitstring;} \\
\Phi(\mathrm{out}(ch, m)) &\triangleq \texttt{out(}ch\texttt{, } m\texttt{);} \\
\Phi(\mathrm{in}(ch, p)) &\triangleq \texttt{in(}ch\texttt{, } p\texttt{);} \\
\Phi(\mathrm{let}\ \vec{x} = t) &\triangleq \texttt{let (}\vec{x}\texttt{:}\vec{\tau}\texttt{) = } t \\
&\qquad\qquad \texttt{in} \\
\Phi(\mathrm{let}\ x = f(\vec{t})) &\triangleq \texttt{let } x\texttt{: } \tau \texttt{ = } f(\vec{t}) \\
&\qquad\qquad \texttt{in.}
\end{aligned} \tag{8}$$

A.6 From FSM to Event Programs (Control-Flow Extraction)

We compile a state s of entity e into an event program by concatenating its `onEntry` code and its outgoing transitions:

$$\mathsf{CompileState}(e, s) \triangleq \mathsf{CompileOps}(O_e(s)); \mathsf{CompileTrans}(e, s, \theta_e(s)). \tag{9}$$

The role-level event program starts from the initial state:

$$p_e \triangleq \mathsf{CompileState}(e, s_e^0).$$

A.7 End-to-End Summary

The complete translation can be summarized as:

Step 1. Extract PFSM.

$$\mathcal{P} \mapsto (\mathcal{E}, (\mathsf{FSM}_e)_{e\in\mathcal{E}}).$$

Step 2. Compile to event programs.

$$p_e \triangleq \mathsf{CompileState}(e, s_e^0).$$

Step 3. Run semantics (labels).

$$p_e \Downarrow \ell_1 \cdots \ell_n.$$

Step 4. Emit ProVerif code.

$$\mathsf{Proc}_e \triangleq \Phi(\ell_1); \cdots ; \Phi(\ell_n).$$

Step 5. Assemble output.

$$\mathsf{PV}(\mathcal{P}) = \mathcal{H} \cup \bigcup_{e\in\mathcal{E}} \mathsf{Proc}_e \cup \mathsf{Proc}_{\text{init}}.$$

A.8 Notation (Quick Reference)

$\mathcal{P}$ Input protocol PFSM.

$\mathcal{E}$ Entity set; e denotes an entity/role.

FSM_e
: Local PFSM $(S_e, s_e^0, O_e, \theta_e)$ of entity e.

Proc_e
: ProVerif process for e; $\mathsf{Proc}_{\text{init}}$ is the initialization process.

t, p
: Term and pattern; σ is a substitution/environment; $t\sigma$ denotes substitution application.

C Configuration $C = \langle \mathcal{N}, \mathcal{I}, \sigma \rangle$, where $\mathcal{N}$ is the fresh-name set and $\mathcal{I}$ is a multiset of instances.

$inst$
: Instance $inst = \{\alpha; P\}$ (execute event α then continue with P).

ℓ, Φ
: Transition label ℓ and its label-to-ProVerif mapping Φ.

A.9 An Example

Figure 5 illustrates a concrete instantiation of PFSM on a simple cryptographic protocol in which Alice sends a digitally signed value to Bob, who verifies the signature. The left-hand side of the figure shows the PFSM representation of the protocol, while the right-hand side presents its corresponding formal realization.

At the protocol level, the participating entities Alice and Bob correspond to elements in the entity set $\mathcal{E}$, and their initial knowledge is specified by the knowledge distribution function Ks. These components define the static context under which protocol execution begins.

The behavior of each participant is modeled as a finite state machine. As shown in Figure 5, Alice is represented by a single state unit whose action set includes a composite action that computes a signature and sends the resulting message to Bob. Bob is similarly modeled by a single state unit, whose actions include receiving Alice's message and performing signature verification. These actions correspond to the action sets $\theta_{e,s}$. defined in PFSM.

Protocol control flow is captured by the transition functions $\theta_{e,s}$. In this example, Bob's transition is conditional: successful verification leads to a success state, while failure leads to an alternative outcome. This illustrates how PFSM explicitly represents conditional branching and protocol flow at the state level.

Overall, Figure 5 demonstrates how PFSM organizes protocol entities, knowledge, actions, and control flow into a structured intermediate representation. Beyond improving interpretability, this design also facilitates the systematic generation of formal code for verification tools such as ProVerif, by aligning PFSM constructs with the operational abstractions employed in formal verification frameworks.

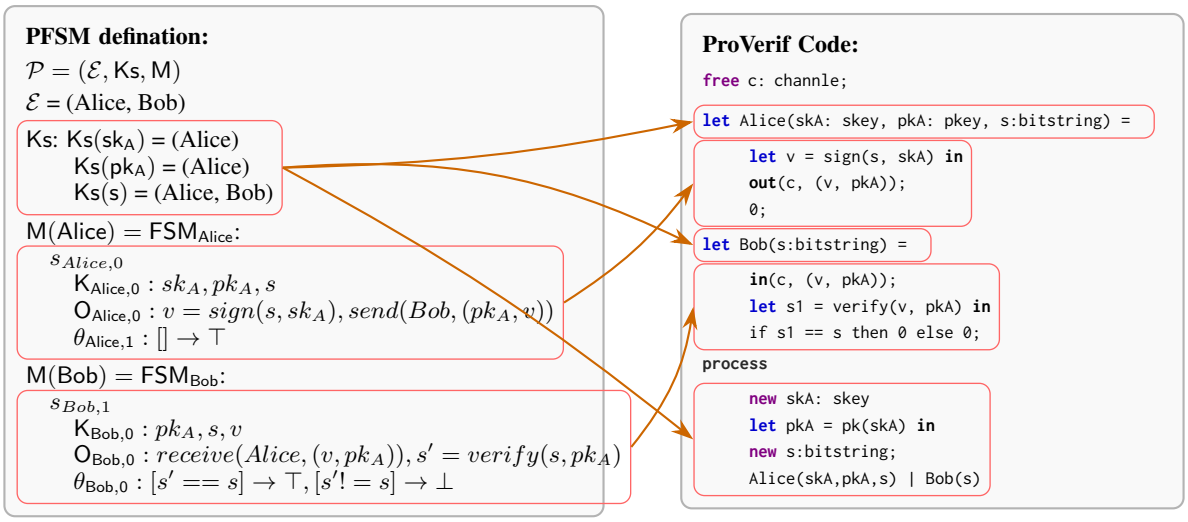

Figure 5: Mapping PFSM defination to Formal Code