AnB2Murphi: A Translator for Converting Alice&Bob Specifications to Murphi

Yongxin Zhao[1], Hongjian Jiang[1], Jin Lv[1], Sijun Tan[2], Yongjian Li[2*]
[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

***Abstract*—As an important part of Internet of Things and 5G network technology, security protocols play a critical role in ensuring communication security. Formal analysis of security protocol has been successfully applied to find design flaws in recent years. Many formal verification tools have been used to verify the security protocols, including Murphi model checker. However, security protocols are often expressed in so-called Alice&Bob notation to describe the messages exchanged between honest principals. And security protocols defined by the A&B specifications can not be applied to the formal verification tool directly. Therefore, there is a gap between Alice&Bob specifications and the modeling languages of the formal tools. In this paper, we propose AnB2Murphi, a novel and general translator which compiles the Alice&Bob specifications of security protocols into the input language of Murphi to bridge the gap. First, we specify the Alice&Bob specifications of the security protocol. Then we take the strand space as the intermediate form between A&B specifications and Murphi formal model. Finally, we use the Murphi model checker to verify the generated model of security protocol. The case studies of security protocols like Needham-Schroeder public key protocol and 5G EAP-TLS authentication protocol demonstrate the efficiency of our translator.**

***Index Terms*—Murphi, Alice&Bob specifications, Strand Space, Security Protocol, Dolev-Yao**

I. Introduction

As an important part of Internet of Things and 5G network technology, security protocols play a critical role in ensuring communication security. It simulates the communication process of multiple entities in the complex network environment and ensures the security of communication. When designing security protocols, such security properties should be ensured, including authentication, secrecy, et al., even with the presence of an intruder who can perform malicious actions. However, the design of these protocols is usually error-prone. This has led to the development of many verification theories and automatic verification tools such as ProVerif [1], Maude-NPA [2] and Murphi [3]. Formal methods have been used to verify security protocols for many years. However, it's difficult for people who have no profound insights into verification theories to model the security protocols by these protocol analysis tools.

Alice&Bob specifications for security protocols ranges from informal narrations of message flows to formal assertions of protocol properties. These message flows provide secure network communication by using the public key encryption. In text-books, the Alice&Bob (abbr. A&B) notation [4] has been often used to describe the message exchanged between honest agents for the successful runs of security protocols. The following example expresses that Alice intends a message to Bob:

$$Alice \longrightarrow Bob : message$$

A&B notation is the most intuitive way to express the correspondence between principals of security protocols. However, the meaning of a protocol specification considered in a context only represents the ideal cryptosystem without active saboteurs. Moreover, A&B notation is too literary to be applied directly in formal verification or code implementation of security protocols. The modelling language of a formal verification tool is often low-level and detailed. Therefore, it is a cumbersome and time-consuming process to formally model security protocols.

The strand space model is a promising framework developed by Guttman et al. to prove the correctness of security protocols. The strand in strand space represents a sequence of events which denote the execution of legitimate party in a security protocol or else a sequence of actions by a penetrator. A strand space is a collection of such strands. The graph structure is generated by causal interaction between strands. With the help of mathematically straightforward methods, strand space model justifies the correctness of security protocols.

In this work, we are motivated to implement a translator AnB2Murphi, which can convert A&B specifications of security protocols to Murphi automatically. We take strand space model as the intermediate form to ensure semantic consistency during the conversion processes. The existence of the intruders in the network and the attack ability of the intruders may pose a challenge to the security of the protocol. Therefore, we construct the deductive rules for intruders and model intruders' behavioral capabilities based on Dolev-Yao model [5].

The main contributions of our work lies in the following aspects:

- **Automatic Translator.** We implement a translator AnB2Murphi to bridge the gap between high-level Alice&Bob specifications and low-level detailed Murphi model checker, which can convert the Alice&Bob specifications of security protocols to the input language of Murphi. Finally, AnB2Murphi has been successfully applied to several security protocols including typically 5G EAP-TLS authentication protocol. The verification result finds and reports the counterexample of errors in the design

*Corresponding author: lyj238@ios.ac.cn
DOI reference number: 10.18293/SEKE2021-028

of the 5G authentication protocol, which demonstrates the efficiency of our translator.

- **Intruder Generation.** Based on Dolev-Yao model, we construct the deduction rules for active intruders, which can help simulate the possible attacks in an insecure network, such as replay attack, man-in-the-middle attack, *etc*. Besides, we have implemented the Diffie-Hellman exchange in our work which supports the algebraic operations and digital signatures to verify the TLS protocol.

The remainder of this paper is organized as follows. In Section II, we review the most related works. In Section III, we give a brief introduction of A&B specifications, strand space and Murphi model checker. In Section IV, we present the architecture of AnB2Murphi and the corresponding relationship between Alice&Bob specifications and Murphi model, then we elaborate the implementation details. In Section V, we report the verification results of the generated Murphi model. In Section VI, we conclude this paper and discuss the future work.

II. Related Work

There has been a lot of discussion on formal verification of security protocols [6]. Besides, the research of A&B notation has received considerable attention. In [7], the authors generalized the formal protocol specification languages and gave the formal semantics for a language based on Alice and Bob including algebraic reasoning. But it was still not expressive enough. In [8], the authors proposed a formal protocol specification language based on the popular Alice&Bob notation, that was AnBx. This specification language extended the formal semantics of A&B notation with a novel notation of forwarding channels. This inspires me to use specific term tmp to represent the forward specific messages in the implementation process. In [9], Omar and Sebastian et al. formalized the language SPS and an automatic translation to robust real-world implementations and corresponding formal models, whose translation was effective.

Besides, there is also a lot of research work on converting A&B specifications to detailed implementation. David Basin et al. [10], they translated A&B protocol specifications to the input language of Tamarin. However, Tarmin [11] is a prover based on theorem proving, which verifies the correctness of the protocol by using multiset rewriting. It requires the user to understand the protocol and supply auxiliary lemmas by heuristics, which is hard even for experts. In contrast, the authors presented a methodology for using Murphi to analyze security protocol in [12]. Murphi is a model checker that supports verification with parameters and has been successfully applied to several industrial protocols. Based on the previous research work, we can see that it is significant to automatically generate the Murphi formal model from the A&B protocol specifications to verify these security protocols.

III. Preliminaries

In this section, we present A&B specifications of security protocols and introduce strand space and Murphi.

A. Formal A&B Specifications Syntax

In cryptography, Alice&Bob specifications are commonly used to describe security protocols. The A&B specifications are intuitive, succinct and yet expressive. The security protocol is specified as a list of message exchange steps of the following form:

$$A \longrightarrow B : message$$

where the initiator A sends the $message$ to the responder B.

As shown in Fig.1, we take Needham-Schroeder public key protocol [13] as the example. The A&B specifications consist of the following parts:

- **Types**. This part declares all identifiers of the protocol specification. In the example, we specify agents A and B, PK is a function that we yield the public key for agents.
- **Knowledge**. This part specifies the initial knowledge attached to each regular agent, consisting of a set of messages. We define the term $initKnwRole(Role)$ to be the initial knowledge of the agent $Role$. For instance, $initKnwRole(A) = \{A, B, Na\}$ in the example.
- **Agents**. Agents is the core of A& specifications which describes the ideal, safe run of the protocol. Every Agent contains two parts: the agent's knowledge and the list of actions to exchange messages. $Actions$ defines the process of protocol execution.
- **Environments**. This part defines a protocol instance, in which we need to give the actual parameters to instantiate the formal atomic parameters in the Knowledge part. See that $Init[1]$ is an instance of agent A and the initial knowledge of $A[1]$ is $< Alice, Intruder, Na >$.
- **Goals**. We specify the security properties that the protocol should achieve. In this work, we mainly focus on goals such as secrecy and non-injective agreement.

```
Protocol : Needham-Schroeder public key
Types:(* Global Types*)
 Agent: A,B;
 Function: PK;
Knowledge:(* Intitial Knowledge*)
 A : A, B, Na
 B : B, Nb
Agents:
 Init(A,B,Na)
 [1]+, B, (Na, A, B) : {|Na, A|}_PK(B)
 [2] − {|Na, Nb|}_PK(A)
 [3]+, B, () : {|Nb|}_PK(B)
 Resp(B,Nb)
 [1]− : {|Na, A|}_PK(B)
 [2]+, A, (Nb) : {|Na, Nb|}_PK(A)
 [3]− : {|Nb|}_PK(B)
Environments: (*Protocol instance*)
 [agent_1]Init[1] :< Alice, Intruder, Na >
 [agent_2]Resp[1] :< Bob, Nb >
Goals: (*Security Goals*)
 [secrecy] Nb secret of < A.B >
 [weakB] B non − injectively agrees with A on Na
end
```

Fig. 1: The Needham-Schroeder Public Key Protocol in A&B Specifications

B. Extended Strand Space

The A&B specifications is a relatively high-level language without implementation details, which cannot be directly used for verification with model checking tools. The strand space [14], which is one of the most successful and widely used formalizations, serves as the intermediate representation format of A&B specifications.

The security protocol defines a sequence of message-exchanged events for each agent. In strand space theory, an action that agents can take during the execution of security protocols includes sending a message and receiving a message. We denote the sending and receiving action by a set of two signs $Sign = \{+, -\}$, respectively. An event is a pair (σ, m), where $\sigma \in Sign$ and m is a message. For example, a node in a strand is like $(+, m)$ which means that the owner of the strand sends a message m.

A strand represents a sequence of events of an agent in a particular protocol run. A strand element is called a node. If s is a strand, (s, i) denotes i^{th} node in strand s. In NSPK protocol, the strand of agent A specifies a sequence of events which can be seen on the left of Fig.2. In this strand, the agent A sends a message $\{\!|Na, A|\!\}_{PK(B)}$ to agent B, and expects to receive back a message of the form $\{\!|Na, Nb|\!\}_{PK(A)}$, after which it will send $\{\!|Nb|\!\}_{PK(B)}$.

A strand space Σ is a set of strands of the principals participating in the running protocol. We have drawn the strand space model of NSPK protocol as shown in Fig. 2, from which we can see the following relationships:

- The relation $n \Longrightarrow n'$ represents the inner-strand communication. It holds between nodes n and n' if $n = (s, i)$ and $n' = (s, i + 1)$.
- The relation $n \longrightarrow n'$ represents the inter-strand communication. The inter-strand communication $(s_A, 1) \longrightarrow (s_B, 1)$ represents that agent A is sending a message $\{\!|Na, A|\!\}_{PK(B)}$ out, and agent B will finally receive a message like this.

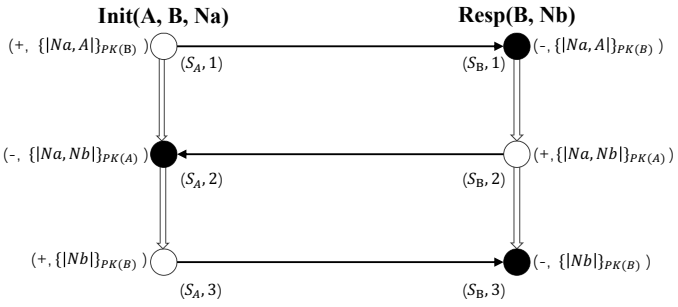

Fig. 2: Regular Strands of NSPK Protocol

Different from the aforementioned A&B specifications, the strand form can be regarded as a state machine, which is more close to the implementation level. It includes states and state transitions with communication actions of messages. It also demonstrates the asynchronous communication style of the agents in the network.

C. Murphi

Murphi is an enumerative (explicit state) model checker with its own input language, which supports the $guard \longrightarrow action$ notation. It can abstract the behavior of a system and simulate the running rules of this system. Murphi model checker has a formal verifier that is based on explicit state enumeration, which has been successfully applied to several industrial protocols. States encountered in this mode are saved in a global hash table. States generated that exist in the hash table are not expanded. The description of Murphi input language consists of declaration part, transition rule part, initialization part and property part.

```
Declaration Part
    --Constant declarations
    --Type declarations
    --Global variable declarations
    --Procedure and function declarations
TransitionRulePart
    rule "ruleName"
      guard part -- conjunction of predicates
    ==>
      action part -- a set of statements
    endrule
Initializationpart
    startstates
      --initial the value of variables
    end
Propertypart
    invariant "inv"
      -- define the security property
    end
```

Fig. 3: The Basic Structure of Murphi Model

The basic structure of a Murphi model can be seen in Fig.3, among which the most important is the transition rule part to describe the transition from one state to another state. A transition rule mainly consists of two parts: guard and action. Only if the predicates in the guard are satisfiable can the statements in the action part execute. Given a Murphi model, the model checker Murphi will enumerate the entire state space explicitly until no new reachable state can be explored or the properties fail to hold on to the protocol. Relatively, a set of all possible reachable states is regarded as the reachable state set (abbr. $RS(P)$). Murphi starts from the initial state, which is prescribed in the initialization part. Then, it will randomly choose a transition rule whose guard is satisfied and execute the corresponding action.

IV. Translate to Murphi

In this section, we discuss how to translate A&B specifications to Murphi. Meanwhile, we take NSPK protocol as a running example.

A. The Architecture of AnB2Murphi

First of all, The overall architecture of AnB2Murphi shown in Fig.4 consists of two main phases: Converting and Verification.

- **Converting**. This phase aims at converting high-level A&B specifications of security protocols into corresponding Murphi model. It includes two important processes: parsing and generating. The parsing process uses Ocaml/menhir tools to analyze and transform the structure of A&B specifications into the extended strand space. The generating process is the focus of our framework. After parsing the A&B specifications, we generate the transition rule part and the built-in environment into the Murphi model based on the extended strand space.
- **Verification**. This phase aims at checking the Murphi model generated in the converting phase. Murphi compiler first compiles the model into a C++ file, then uses the compiling and executing mode of the C++ program to perform the verification process, which accelerates the speed of model detection.

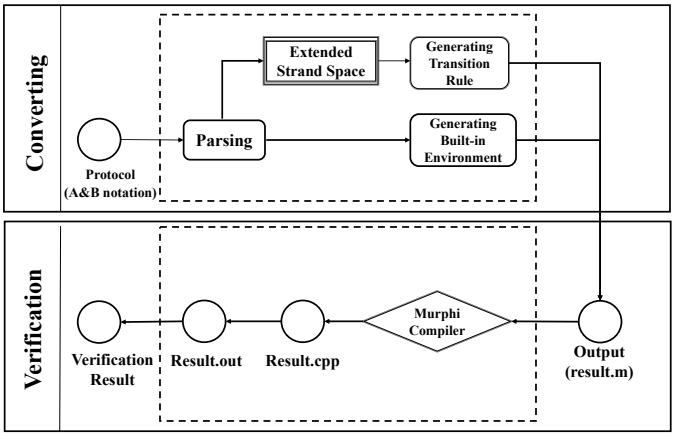

Fig. 4: The Architecture of AnB2Murphi

We have given the description of A&B specifications and the structure of Murphi model in Section.III. The corresponding relationship between the A&B specifications and Murphi is shown in Fig.5. The rest of this section focuses on the implementation phases, mainly including Converting and Verification.

B. Converting

Converting is mainly constructed by Parsing and Generating processes. The Parsing process transforms the A&B specifications into the extended strand space to ensure the consistency of semantic during the conversion processes. And the Generating process is used to generate transition rule part and built-in environment in Murphi model.

1) Parsing: The main task of this process is to analyze the structure of A&B specifications by tool Ocaml/Menhir. This process is relatively simple and we will not elaborate on it too much. The result of parsing A&B specifications is given directly below.

$$\begin{aligned}&\mathsf{type\ ProtocolContext} = [\\ &|\mathsf{Protocol\ of\ type} * \mathsf{knowledge} * \mathsf{agent} * \mathsf{environment} * \mathsf{goal}\\ &|\mathsf{Null} \qquad\qquad\qquad\qquad]\end{aligned}$$

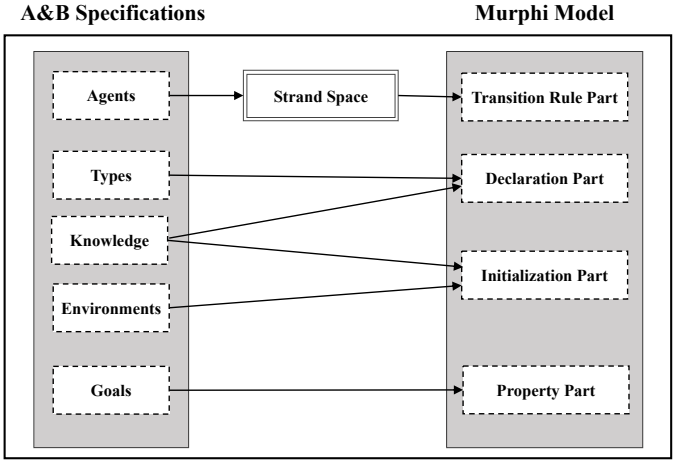

Fig. 5: Correspondence of A&B Specifications and Murphi

We translate the security protocol as a type of ProtocolContext, which includes the following five parts: type, knowledge, agent, environment and goal. The Protocol is the tag of a protocol. Therefore, the A&B specifications of a protocol will be parsed into $\mathsf{Protocol}(\mathsf{t}, \mathsf{k}, \mathsf{a}, \mathsf{e}, \mathsf{g})$, in which the element in the quintuple are corresponding to the five parts of the A&B specifications, respectively.

In addition, the possible messages which can be exchanged between agents in security protocols are defined below:

```
type message = [
  | Nonce of identifier (*Nonce*)
  | Agent of roleName (*Agent Identifier*)
  | Const of identisirt (*Const Number*)
  | Pk of roleName (*Public Key*)
  | Sk of roleName (*Secret Key*)
  | K of roleName * roleName (*Symmetry Key*)
  | Mod of message * message (*Mod*)
  | Exp of message * message (*Exp*)
  | Tmp of messageName (*Temporary*)
  | Sign of message * message (*Signature*)
  | Aenc of message * message (*Aencrypt*)
  | Senc of message * message (*Sencrypt*)
  | Concat of message list (*Concatenation*)
]
```

The $message$ consists of atomic messages and compound messages. Atomic messages are non-divisible messages, such as $\mathsf{Nonce}, \mathsf{Agent}, \mathsf{Pk}$ and so on. Compound messages are composed of multiple atomic messages, which are encrypted symmetrically or asymmetrically, or combined by connections.

According to the Agents part of in A&B specifications, we construct the extend strand space model of the protocol by using the following methods.

(1) $\mathsf{genStrand}(\mathsf{agent}, \mathsf{role_i})$: This method converts the Agents in A&B specifications into a node of strand $\mathsf{role_i}$.

$$\mathsf{genStrand}(\mathsf{Agent}(\mathsf{roleName}, \mathsf{message\ list}, \mathsf{action\ list}), \mathsf{role_i}) = \mathsf{action\ list\ if\ roleName} = \mathsf{role_i}.$$

In the A&B specifications, Agents is expressed into the form of $\mathsf{Agent}(\mathsf{roleName}, \mathsf{message\ list}, \mathsf{action\ list})$. If $\mathsf{role_i}$ is the name of the agent, then the result of method genStrand is the corresponding action list.

(2) genAction(action, sign): This method converts the action of some node in strand space to the send action and the receive action.

$$\mathsf{genAction}(\mathsf{action},\mathsf{sign}) = \begin{cases} (+,M) & \text{if } sign = + \\ (-,M) & \text{if } sign = - \\ \epsilon & otherwise \end{cases}$$

The $action$ has been shown in below, and consists of the Send and Receive parts. If the $sign$ is $+$, the result returns the sending action; if $sign$ is $-$, then the result returns the receiving action; otherwise the result is ϵ.

```
type action = [
    |Send of int * sign * roleName * message list * message
    |Receive of int * sign * message
    |Null          ]
```

In Needham-Schroeder public key protocol, there are two agents A and B. The strand space and actions between two agents can be constructed by the above two methods.

2) Generating: This process generates the input language of Murphi model based on the extended strand space constructed at last process. Generating process consists of two sub-processes: Generating Transition Rule and Generating Built-In Environment.

As for Generating Transition Rule process, we first generate the transition rules for regular principals. The strand of agents can be regarded as a state machine. Each node in the strand can be treated as a state of the state machine. When an agent receives a message or sends a message, its state will be changed, i.e., a state transition has occurred. For each regular principal, we use a variable $commit$ to record whether it has completed the execution of the protocol. In Murphi, $rule$ expresses the state transition. After generating the strand of each agent from $agents$, we use the method trans() to convert the i^{th} node of strand $role_i$ into Murphi rule. The behavior of regular principal is relatively simple. It only consists two parts: construct the sent message and deconstruct the received message. The function genSendAct() and genRecvAct() generate the action of corresponding rule for sending or receiving message.

```
trans(act, M, i, role_i) =
    let atoms = getAtoms(M) in
    match act with
    | (+, M) ⟶ genRuleName(role_i, i);
                genSendGuard(role_i, i);
                genSendAct(role_i, i, atoms)
    | (−, M) ⟶ genRuleName(role_i, i);
                genRecvGuard(role_i, i);
                genRecvAct(role_i, i, atoms)
```

Then we generate deduction rules for active intruders. Based on the Dolev-Yao model, we describe the behavior of intruders:

- Eavesdrop and intercept any message in the network;
- Participate in the operation of the agreement as a legal principal or counterfeit legal principal;
- Deduce new knowledge from existing knowledge set;
- Forge messages according to the knowledge he obtained and send them to the regular principal who may accept.

The intruder in the Murphi model can deduce the message according to the deduction rule listed in Table.I. Besides, it keeps the $Knws$ which stores the initial and deductive knowledge the intruder holds. As for the example, the public keys of agents A and B are in the intruder's knowledge set initially. After intercepting a message from the network, the intruder makes the deduction of the message based on the knowledge, and adds the knowledge into his knowledge set $Knws$.

TABLE I: Deduction Rules

Decryption	(decrypt) $\dfrac{\{\!\|m\|\!\}_K \in Knws \wedge inv(K) \in Knws}{\{\!\|m\|\!\} \in Knws}$
Encryption	(encrypt) $\dfrac{\{\!\|m\|\!\} \in Knws \wedge K \in Knws}{\{\!\|m\|\!\}_K \in Knws}$
Separation	(deconcat) $\dfrac{\{\!\|m1; m2\|\!\} \in Knws}{\{\!\|m1\|\!\} \in Knws \wedge \{\!\|m2\|\!\} \in Knws}$
Concatenation	(enconcat) $\dfrac{\{\!\|m1\|\!\} \in Knws \wedge \{\!\|m2\|\!\} \in Knws}{\{\!\|m1; m2\|\!\} \in Knws}$
Signature	(signature) $\dfrac{m \in Knws \wedge SK \in Knws}{\{\!\|m\|\!\}_{SK}^{sign} \in Knws}$
Verify	(verify) $\dfrac{\{\!\|m\|\!\}_{SK}^{sign} \in Knws}{m \in Knws}$
Hash	(hash) $\dfrac{m \in Knws}{hash(m) \in Knws}$

After generating transition rules which include the transition rules for regular principals and the potential intruders, the built-in environment is then generated to define the procedures and functions in Murphi. It mainly includes the construction procedure and destruction procedure of the exchanged messages.

(1) Extracting the exchanged messages from $actions$ and generating the $patlist$ consisting of the message patterns and its sub-patterns:

$$\mathsf{patList} = \mathsf{rmEquiv}(\mathsf{delDup}(\mathsf{getPatList}(\mathsf{actions}))$$

There are three methods involved in the derivation of this process in Ocaml. Method getPatList(actions) extracts the messages and their sub-messages from $actions$ and returns a pattern list. Then method delDup(list) is used to delete the duplicate items in the $list$ and return without replicas. Finally, method rmEquiv(list) helps remove the equivalence class in the list. For example, $Agent(A)$ and $Agent(B)$ belong to the same class and just keep one.

(2) Generating the construction procedures of messages. Based on the generated pattern list, we generate the corresponding built-in construction and destruction procedure for each pattern. In Murphi, we use a global array $msgs$ to store message generated during the protocol execution. During construction process, we first find out whether the message exists in the $msgs$. If it exists, we directly returns its index; if not, we construct the message and add the index of message into $msgs$.

C. Verification

In Converting phase, we generate the Murphi model of the security protocol illustrated in A&B specifications, and we

output the results to the file $result.m$.

Murphi model checker will first check whether there exists some errors in the Murphi model. If not, Murphi compiler convert the $result.m$ into a C++ program $result.cpp$. Then, we compile the C++ program using GNU compiler g++, which gives us the resulting executable, a verifier that computes reachability for the specific problem. Finally, execute the program $result.out$ to verify the security protocols, and the result of the verification will be output to the terminal.

V. Evaluation

We have implemented several security protocols in A&B specifications which come from the security protocols repository [15]. Significantly, we model the 5G EAP-TLS Authentication protocol which plays a critical role of the first safeguard in ensuring the communication security. There are up to 17 messages-exchanged in the authentication protocol which involves sophisticated cryptography terms. When verifying this complicated model, Murphi needs to explore the whole state space. Basing on cmurphi5.4.9.1 as the verification engine, we find that the weak-agreement and secrecy $prekey$ are violated. Murphi generates counterexamples for these two properties.

The experiments are carried out on a PC equipped with the macOS Catalina and Intel Core i7 with 2.6Ghz CPU and 16GB RAM. The verification result of these security protocols are shown in Table.II. The source code of translator AnB2Murphi can be accessed at [16].

TABLE II: VERIFICATION RESULTS

Protocols	Unsatisfied	Time (sec.)	Memory
NeedhamSchroder	secrecy(Nb)	0.10	56
	weakB	0.15	
Lowe′s NeedhamSchroder	no error	0.10	58
Diffie–Hellman	secrecy(Na)	0.10	64
Otway–Rees	no error	2.13	117
CCITT X.509(1)	secrecy(Ya)	0.21	53
	weakB	0.84	
	weakA	0.84	
CCITT X.509(1c)	no error	0.45	69
Woo and Lam Pi	secrecy(Nb)	0.10	69
Andrew Secure RPC	secrecy(Kab)	2.77	54
EAPTLS authentication	secrecy(prekey)	1.21	1700
	weakC	151.55	

VI. Conclusion and Future Work

In this paper, we have implemented an automatic translator AnB2Murphi to bridge the gap between high-level Alice&Bob specifications and low-level Murphi model checker, which can help verify the security protocol described in the A&B specifications. We design a scheme to translate the actions of regular principals into strand space, which can well describe the communication relationship between principals and the state transition of themselves. Based on Dolev-Yao model, we construct the deduction rules for intruders, which can help simulate the possible attacks in insecure networks. The translator is implemented in Ocaml/Menhir, which is a simple but powerful parser generator for the Ocaml programming language. AnB2Murphi has been successfully applied to several typical security protocols. The results of verification are consistent with those already proved.

For the weakness of the current work, we would like to point out that the A&B specifications of a protocol are intuitive. Because Murphi is a model checker, it is good at examine multiple runs of protocol, and give us the counterexample trace when the protocol does not satisfy the specification. But it bothers us to specify the actual parameters in the environment of A&B specifications. We will fix this question by integrating machine learning into the work.

Acknowledgements

This work is supported by National Key Research and Development Program (2020AAA0107800), Shanghai Science and Technology Commission Program under Grant 20511106002, Grant 61672503 from National Science Foundation in China.

References

[1] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial," *Version from*, pp. 05–16, 2018.

[2] S. Escobar, C. Meadows, and J. Meseguer, "Maude-npa: Cryptographic protocol analysis modulo equational properties," in *Foundations of Security Analysis and Design V*. Springer, 2009, pp. 1–50.

[3] D. L. Dill, "The mur ϕ verification system," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 390–393.

[4] C. Caleiro, L. Vigano, and D. Basin, "On the semantics of alice&bob specifications of security protocols," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 88–122, 2006.

[5] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[6] V. Cortier and S. Kremer, *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, 2011, vol. 5.

[7] S. Mödersheim, "Algebraic properties in alice and bob notation," in *2009 International Conference on Availability, Reliability and Security*. IEEE, 2009, pp. 433–440.

[8] M. Bugliesi, S. Calzavara, S. Mödersheim, and P. Modesti, "Security protocol specification and verification with anbx," *Journal of Information Security and Applications*, vol. 30, pp. 46–63, 2016.

[9] O. Almousa, S. Mödersheim, and L. Viganò, "Alice and bob: Reconciling formal models and implementation," in *Programming Languages with Applications to Biology and Security*. Springer, 2015, pp. 66–85.

[10] M. Keller and P. D. D. Basin, "Converting alice&bob protocol specifications to tamarin," Ph.D. dissertation, Bachelor's thesis, ETH Zurich, 2014. Available at http://www. infsec. ethz . . . , 2014.

[11] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The tamarin prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 696–701.

[12] J. C. Mitchell, M. Mitchell, and U. Stern, "Automated analysis of cryptographic protocols using mur/spl phi," in *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)*. IEEE, 1997, pp. 141–151.

[13] R. M. Needham and M. D. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[14] F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman, "Strand spaces: Proving security protocols correct," *Journal of computer security*, vol. 7, no. 2/3, pp. 191–230, 1999.

[15] security protocols open repository, http://www.lsv.ens-cachan.fr/spore.

[16] AnB2Murphi, https://github.com/AnB2Murphi/AnB2Murphi.