Automating Compositional Analysis of Authentication Protocols

Zichao Zhang
Carnegie Mellon University

Arthur Azevedo de Amorim
Carnegie Mellon University

Limin Jia
Carnegie Mellon University

Corina S. Păsăreanu
Carnegie Mellon University and NASA Ames

***Abstract*—Modern verifiers for cryptographic protocols can analyze sophisticated designs automatically, but require the entire code of the protocol to operate. Compositional techniques, by contrast, allow us to verify each system component separately, against its own guarantees and assumptions about other components and the environment. Compositionality helps protocol design because it explains how the design can evolve and when it can run safely along other protocols and programs. For example, it might say that it is safe to add some functionality to a server without having to patch the client. Unfortunately, while compositional frameworks for protocol verification do exist, they require non-trivial human effort to identify specifications for the components of the system, thus hindering their adoption.**

To address these shortcomings, we investigate techniques for automated, compositional analysis of authentication protocols, using automata-learning techniques to synthesize assumptions for protocol components. We report preliminary results on the Needham-Schroeder-Lowe protocol, where our synthesized assumption was capable of lowering verification time while also allowing us to verify protocol variants compositionally.

I. Introduction

Cryptographic protocols are notoriously difficult to design, yet their correctness is crucial to ensure the security of software systems. Formal methods are thus valuable, as they can reveal critical bugs before these systems are deployed. Automated tools (ProVerif [8], CryptoVerif [9], Tamarin [33], etc.) are particularly interesting, as they allow us to focus on modeling the protocol rather than proving its correctness. Although these tools have been applied to ambitious case studies [7], [10], [34], [6], [20], they suffer from one important drawback: they offer little support for compositional reasoning. To verify a property, we must supply the entire protocol model at once, rather than verifying each component of the protocol against self-contained partial specifications. This is unsatisfactory, since a non-compositional analysis works under a *closed-world* assumption that provides few guarantees for when the protocol is itself a component of a larger system—for example, using a private key to sign and encrypt data simultaneously can expose vulnerabilities that are absent if only one of the functionalities is used. Furthermore, decomposition can help speed up verification and guide protocol design when components are modified, or even perhaps *removed*, in case we want to de-bloat an existing protocol without breaking its security.

We envision a future where we can combine the power of compositional reasoning with the convenience of automation. As a first step in this direction, we consider how protocol analysis can benefit from off-the-shelf, automated compositional verification tools. To illustrate, suppose that we have a complex system $M_1 \mid\mid M_2$, obtained by composing simpler pieces M_1 and M_2. We would like to show that $M_1 \mid\mid M_2$ satisfies a specification P: $M_1 \mid\mid M_2 \models P$. Rather than proving P directly, we can resort to the following *assume-guarantee rule*:

$$\frac{\langle Q\rangle M_1\langle P\rangle \qquad \langle \mathit{true}\rangle M_2\langle Q\rangle}{M_1 \mid\mid M_2 \models P} \tag{1}$$

This rule says that we can prove P by finding an *assumption* Q such that (1) P holds on M_1, assuming that Q holds on the rest of the system; and (2) the component M_2 guarantees that Q holds. Though it can be challenging to craft a suitable Q by hand, prior work [30], [16] shows that it can be inferred with L^* [3], an automaton learning algorithm, even for systems with multiple components.

We report preliminary results on the analysis of the Needham-Schroeder protocol [29] and its subsequent correction by Lowe [21] (dubbed NS and NSL, for short). We developed models of the protocols for a version of the LTSA model checker [24] extended with automaton learning [30], and used this infrastructure to synthesize assumptions to verify the protocol. Our focus is on *agreement properties* [35], [22], which say that when authentication is complete the participants are indeed talking to whom they think they are talking to.

One obstacle for the formal analysis of security protocols is dealing with rich attacker behavior. A popular threat model is the *symbolic* (or *Dolev-Yao* [19]) paradigm, which says that the attacker has complete control over the network, but is constrained by standard cryptographic assumptions. Thus, the attacker might be able to shuffle, drop or replay messages, but cannot decrypt a message without the corresponding key. To ease the modeling of such threats, we developed *Taglierino*, a domain-specific language for describing protocols and attacker behavior as LTSA automata.

Taglierino requires users to bound the possible attacker behaviors to ensure that its output is finite and it can be analyzed by LTSA. (Any attack can in principle be found with Taglierino if we make this bound large enough.) Though finite, we observed that Dolev-Yao attackers produced in this way require a large number of states (>700k) to cover interesting behaviors. Synthesizing component assumptions directly using such attackers leads to bloated assumptions that are expensive to check and hard to interpret. To facilitate a compositional analysis of NSL, we carry a first decomposition step where we generate assumptions about the behavior of the attacker using *alphabet refinement* [30]. This decomposition shows that we can replace the attacker by a much simpler one (3 rather than 700k states). We use this refined attacker to generate assumptions for the initiator of the protocol. The assumptions are small (10–20 states), so they can be examined by decomposition and used for checking replaced components.

The rest of the document proceeds as follows. After a quick overview of the NS protocol and how it is modeled in Taglierino (Section II), we present our analysis of the protocol in Section III, explaining how we generated assumptions for the protocol initiator and used them to verify protocol variants and detect bugs. We discuss related work in Section IV and conclude in Section V.

II. An Overview of NS

The Needham-Schroder public key protocol [29] is intended to provide mutual authentication of two agents, Alice (A) and Bob (B). The protocol can be summarized as follows:

$$\begin{array}{rl}
(1) & A \rightarrow S : A, B \\
(2) & S \rightarrow A : \{B, pk_B\}sk_S \\
(3) & A \rightarrow B : \{n_A, A\}pk_B \\
(4) & B \rightarrow S : B, A \\
(5) & S \rightarrow B : \{A, pk_A\}sk_S \\
(6^*) & B \rightarrow A : \{n_A, n_B\}pk_A \\
(7) & A \rightarrow B : \{n_B\}pk_B
\end{array}$$

Alice starts by contacting the key server S asking for Bob's public key pk_B. The server returns this information to Alice signed with its own secret key sk_S, to prove that pk_B is authentic. Then, Alice encrypts a fresh cryptographic nonce n_A and sends it to Bob, along her own identity. Bob asks the key server for Alice's public key pk_A, and then sends n_A back to Alice along another fresh nonce n_B, all of this encrypted with Alice's key. Finally, Alice acknowledges the end of the handshake to Bob by sending him n_B back. (The protocol turns out to contain a vulnerability in message (6^*); we'll come back to this shortly.)

The intended specification for the protocol can be informally stated as follows:

- When Alice receives Message 6, she knows that Bob accepted her connection.
- When Bob receives Message 7, he knows that Alice has tried to contact him.

To formalize this property, we model the behavior of the system as a series of finite automata running in parallel. Each automaton defines a language of traces over the following alphabet:

- $send_i(m)$: The agent i has sent the message m over the network.
- $recv_i(m)$: The agent i has received the message m from the network.
- $begin_i(e, m)$: The agent i claims that the event e has begun, using the data item m as an identifier.
- $end_i(e, m)$: The agent i claims that the event e has ended, using the data item m as an identifier.

```
agent "Alice" $ do
  hostX <- receive
  begin "authAB" hostX
  send [alice, hostX]
  sig <- receive
  [pkX, host] <- checkSign spkS sig
  when (host == hostX) $ do
    send $ aenc pkX [na, alice]
    m <- receive
    [nx, ny] <- adec skA m
    if (nx == na) then
       send $ aenc pkX ny
    else fail "nonce␣mismatch"
```

Fig. 1: Implementation of Alice in NS.

Messages and data items are drawn from a set *Term* that contains an infinite supply of nonces, cryptographic keys, encrypted messages, etc. To keep the models finite, we restrict this set to a finite subset $A \subseteq Term$ of *allowed terms*. Our goal is to prove *agreement* [35], [22]: if an event of the form $end_i(e, m)$ occurs in an execution trace, than the trace has an earlier occurrence of the event $begin_j(e, m)$. For instance, Alice might emit $begin_A(auth_{AB}, B)$ at the beginning of the protocol to signal that she wishes to communicate with Bob, and Bob would emit $end_B(auth_{AB}, B)$ after receiving $\{n_B\}pk_B$ to indicate that the connection was successful.

Each protocol participant corresponds to a finite automaton. These automata are specified in Taglierino using a domain-specific language similar to process calculi used in protocol verification [8], [1]. Figure 1 shows the model of Alice in Taglierino. A preamble, not shown in the figure, declares constants such as the nonce `na`, Alice's identity `alice`, Alice's private key `skA`, and Server's public signature key `spkS`. Alice communicates with the network using `send` and `receive`. The first received message (`hostX <- receive`) means that Alice is willing to run the protocol with any other agent chosen by the network. Upon sending or receiving from the network, Alice can manipulate messages using cryptographic primitives; for example, `aenc` and `adec` stand for asymmetric encryption and decryption and `checkSign` is for checking the signature.

The protocol implementation in Taglierino is compiled down to models for the LTSA model checker [24]. In addition to the honest agents, our compiler generates another automaton that describes how messages are transmitted in the network. This transmission follows the symbolic model of cryptography [19]: an agent i can receive a message m if and only if the predicate $\mathit{knows}(M, m)$ holds, where M is the set of messages that have been sent to the network up to that point. Intuitively, this amounts to assuming that an attacker can intercept all messages sent in the network and gets to decide what is delivered in the end, potentially tampering with the result. The definition of knows is standard; for instance it includes the following clauses

$$\frac{m \in M}{\mathit{knows}(M, m)}$$

$$\frac{\mathit{knows}(M, \mathit{sk}(k)) \qquad \mathit{knows}(M, \{m\}\mathit{pk}(k))}{\mathit{knows}(M, m)},$$

which say that the attacker can always reproduce messages it has previous seen, and also decrypt a message m if it can extract the corresponding decryption key $\mathit{sk}(k)$ from its knowledge. The network automaton does not have begin or end events in its alphabet, since those are controlled by the honest agents of the system.

III. Analyzing the Protocol

When Bob receives $\{n_B\}pk_B$, he thinks that Alice has decided to contact him because there is no other way he could have received this message: the nonce n_B was freshly generated, and only Alice has the power to open the encrypted message $\{n_A, n_B\}pk_B$. Unfortunately, this reasoning is flawed: an attack found by Lowe [21] shows that Alice could have really meant to contact a malicious third party Mallory (M), who uses Alice's messages to trick Bob into believing he is communicating with Alice directly. If Bob implements a banking service, for example, this might allow Mallory to gain access to Alice's account without her permission. The fix found by Lowe is to include Bob's identity in one of the messages:

$$(6)\ \ B \rightarrow A : \{n_A, n_B, B\}pk_A$$

Lowe's analysis shows that the original sixth message does not have enough information for Alice to know who she is really talking to. This corrected message allows her to stop sending message (7) when she realizes who her contact is.

In this section, we show how we can decompose the resulting NSL protocol in a way that allows us to detect the original flaw and also check the correctness of variants of the protocol, at least in a bounded sense. More precisely, we start by generating an assumption A for Alice in NSL; as a by product of this process, we establish the correctness of NSL through the application of (1). Then, we use A to analyze two variants of the protocol where Alice behaves slightly different. Since Alice is the only component that changes, we can verify that the variants are correct simply by checking that Alice satisfies the assumption A.

We compare the effort to verify the protocols compositionally and monolithically. Our results (Section III-E) show that compositional verification considerably outperformed monolithic verification when it can reuse the assumption A; if A needs to be regenerated, compositional verification is more expensive. All experiments were performed with a 1.6 GHz Intel Core i5 CPU and 8.0 GB RAM, running 64-bit Ubuntu 18.04 LTS.

Let M_1 and M_2 be two component in the system and P be the property we want to check. We use αM to denote the alphabet of an component M and Σ_I to denote the interface alphabet, that is, $\Sigma_I = \alpha M_1 \cap \alpha M_2$.
Let σ be an arbitrary trace where σ_n denotes the nth action on trace σ and Σ be a arbitrary set of alphabet, we define

$$find(\Sigma, \sigma) = \begin{cases} \sigma_i, & \text{if } \sigma_i \subseteq \Sigma_I \wedge \sigma_i \nsubseteq \Sigma \\ \emptyset, & \text{otherwise} \end{cases}$$

where i is the first index scanning from the end of trace σ to the beginning such that the conditions hold.

1) Obtain trace σ from checking $\langle \mathit{true} \rangle M_1 \langle P \rangle$.
2) Initialize $\Sigma = find(\emptyset, \sigma)$.
3) Use the classic learning framework for Σ. If the framework returns true with assumption Q, we report the Q and STOP. When the framework returns false with counterexample trace σ'. This, however, does not necessarily means that $M_1 \mid\mid M_2$ violates P. Real violations are discovered by the learning framework only if the alphabet is Σ_I and thus we go to the next step.
4) If $find(\Sigma, \sigma')$ returns $\emptyset$, we report false and STOP. If $find(\Sigma, \sigma')$ returns an action a, we update $\Sigma = \Sigma \cup a$ and go to step 3.

Fig. 2: Alphabet refinement process.

Component	#States	#Trans.	Assumption	
			#States	#Trans.
Attacker	775030	4343487	3	178
Alice	14	163	6	69

Fig. 3: Comparison of the original component with its generated assumption, in terms of states and transitions.

A. Generating Assumptions with NSL

Our model of NSL allows all the original messages of the protocol to be exchanged in the network, but includes other terms that enable Lowe's attack in the original NS: $\{n_B\}pk_M$, $\{pk_M, M\}sk_S$, etc. We chose these terms heuristically, by taking the legitimate messages exchanged by Alice and Bob and scrambling some of the parameters. In total, our model allows 31 messages to be exchanged in the network. When setting up the model, we make sk_M, Mallory's secret key,

Attacker	Alice
$send_i(\{n_A, n_B, M\}pk_A)$	$send_A(\{n_A, n_B, M\}pk_A)$
$send_i(\{n_A, n_B, B\}pk_M)$	$send_A(\{n_A, n_B, B\}pk_M)$
$send_i(\{n_A, n_B, M\}pk_M)$	$send_A(\{n_A, n_B, M\}pk_M)$
$send_i(\{n_B, n_B, B\}pk_M)$	$send_A(\{n_B, n_B, B\}pk_M)$
$send_i(\{n_B, n_B, M\}pk_M)$	$send_A(\{n_B, n_B, M\}pk_M)$
$send_i(\{n_M, n_B, B\}pk_M)$	$send_A(\{n_M, n_B, B\}pk_M)$
$send_i(\{n_M, n_B, M\}pk_M)$	$send_A(\{n_M, n_B, M\}pk_M)$
$send_i(\{n_B\}pk_B)$	$send_A(\{n_B\}pk_B)$
$send_i(\{n_B\}pk_M)$	$send_A(\{n_B\}pk_M)$
$send_i(\{B, pk_B\}sk_S)$	$send_A(\{B, pk_B\}sk_S)$
$recv_i(\{n_A, n_B, M\}pk_A)$	$recv_A(\{n_A, n_B, M\}pk_A)$
$recv_i(\{n_A, n_B, B\}pk_M)$	$recv_A(\{n_A, n_B, B\}pk_M)$
$recv_i(\{n_A, n_B, M\}pk_M)$	$recv_A(\{n_A, n_B, M\}pk_M)$
$recv_i(\{n_B, n_B, B\}pk_M)$	$recv_A(\{n_B, n_B, B\}pk_M)$
$recv_i(\{n_B, n_B, M\}pk_M)$	$recv_A(\{n_B, n_B, M\}pk_M)$
$recv_i(\{n_M, n_B, B\}pk_M)$	$recv_A(\{n_M, n_B, B\}pk_M)$
$recv_i(\{n_M, n_B, M\}pk_M)$	$recv_A(\{n_M, n_B, M\}pk_M)$
$recv_i(\{n_B\}pk_B)$	$recv_A(\{n_B\}pk_B)$
$recv_i(\{n_B\}pk_M)$	$recv_A(\{n_B\}pk_M)$
$recv_i(\{B, pk_B\}sk_S)$	$recv_A(\{B, pk_B\}sk_S)$
	$begin_A(auth_{AB}, B)$
	$begin_A(auth_{AB}, M)$

Fig. 4: Alphabets of generated assumptions. The identifier i ranges over A and B.

available to the attacker, while keeping all other private keys secret. We also bounded the attacker to learn at most 4 messages in addition to its initial knowledge.

When compiled, our model had a large attacker of more than 700k states. To obtain a more tractable model, we decomposed the system to generate an assumption for the attacker (i.e. letting $M_1 = Alice \,||\, Bob \,||\, Server$ and $M_2 = Attacker$ in rule (1)). To facilitate learning, we used *alphabet refinement* [30], a technique that generates more compact assumptions by limiting the possible interactions between components. Roughly speaking, alphabet refinement consists in gradually adding actions to the interface of M_1 and M_2 until we successfully generate a sound assumption for the attacker or manage to prove that the property did not hold. (Figure 2 describes this process in more detail.)

After refinement, we further decomposed the system using the assumption on the attacker to generate an assumption for Alice. Figure 3 shows the size of the original components with their generated assumption; Figure 4 shows the alphabets. The fact that we were able to generate an assumption for Alice means that the NSL protocol satisfies agreement. We will now see how this generated assumption facilitates the analysis of protocol variants.

B. Finding Lowe's Flaw in NS

We modified Alice in NSL such that the agent identity in message (6) is not checked. The behavior of the modified protocol is equivalent to the original NS and allows Alice, while thinking she is contacting Mallory, to accept the message:

$$(6)\;\; B \to A : \{n_A, n_B, B\}pk_A$$

and continue with:

$$(7)\;\; A \to M : \{n_B\}pk_M$$

This behavior enables Lowe's attack on NS, which we rediscovered by checking the modified Alice against the assumption generated in the previous section.

In principle, it is possible this method yields a spurious counterexample. The automaton learning technique generates the weakest assumption for Alice to validate agreement, but the assumption was computed using an *abstraction* that has more behaviors than the original attacker, and thus imposes more restrictions on Alice than would be necessary. To rule out the possibility that our counterexample is spurious, we double-check that it can be produced by this variant of NSL. Even when combined with the time to recheck the counterexample, the time spent to find this bug compositionally was much smaller than the time spent on monolithic bug finding, thus strengthening the case for compositional verification.

C. Serverless NSL

A common simplification of NSL is to assume that Alice knows the keys of the agents she wants to contact from the start. This amounts to removing the communication between Alice and Server (messages (1) and (2)). We were capable of verifying this version of Alice against our previously generated assumption, thus confirming that this serverless variant of NSL is correct.

D. Interpreting the Assumptions

Figure 3 shows that assumption learning with alphabet refinement was capable of significantly abstracting the behavior of the attacker and of Alice, yielding automata that are much smaller in terms of number of states and number of transitions. The alphabets of the assumptions (Figure 4) list the actions that must be controlled for the property to hold; removing them from the alphabet has the effect of allowing the attacker to freely perform those actions, regardless of whether a send action was triggered by an honest agent or of whether the attacker had enough knowledge to deliver a message.

The only difference between the alphabet for Alice and for the Attacker is that the Attacker alphabet includes actions for Bob, whereas Alice's includes her *begin* events. Most of the controlled actions are variants of (6) encrypted with pk_M. If the attacker is free to forge such messages indiscriminately, he is capable of learning the nonce n_B even before Bob is contacted by Alice or Mallory. When this is true, the attacker

Protocol	Attack	Compile time(ms)	#States Attacker	Monolithic verification			Compositional verification			
				#States	#Transitions	Time(ms)	#States	#Transitions	Time(ms)	
NSL public key [21]	No	2851	775030	388	2738	8	18	163	1	*
NS public key [29]	Yes [21]	2674	775030	10880	102449	97	19 (3104)	164 (22979)	1 (22)	**
NSL public key (variant)	No	2182	775030	9792	86094	115	13	99	1	

Fig. 5: Experimental results (cf. Section III-E)

has all the information needed to impersonate Alice and break agreement. (Note that we didn't include n_B in the allowed set of messages, so it is not possible for the attacker to learn this value directly.) Interestingly, the expected message (6) in a normal run of the protocol, $\{n_A, n_B, B\}pk_A$, is not in the alphabet. Intuitively, since the attacker does not control pk_A, the only thing he can do with this message is relaying it to Alice. If Alice meant to talk to Bob anyway, she will eventually trigger $begin$ and send her response (7) to Bob, which does not pose any harm for agreement. Otherwise, if she meant to talk to Mallory, receiving this message will trigger a mismatch between Bob's identity and Mallory's; thus, she'll stop running and never send (7) to Bob.

E. Results

Figure 5 summarizes the results of verifying the three variants of NSL above. Each row describes:

- whether the variant is vulnerable to an attack;
- how long it took to compile the various automata produced by Taglierino;
- the number of states in the attacker component;
- results for monolithic verification: the number of states and transitions of the compiled automata, as well as the time spent to verify them;
- results for compositional verification: the number of states and transitions of the compiled automata used to check that Alice satisfies the generated assumption, as well as the time to perform this check.

Note that the results of compositional verification for the first row (*) are somewhat redundant, since the system is automatically verified as a byproduct of generating the assumptions. We included those numbers for completeness. In each column under the results of compositional verification for the second row (**), the first number refers to the process of generating the counterexample, whereas the second number refers to the process of rechecking it, as explained in Section III-B. In all cases, we observe that compositional verification requires substantially fewer resources than monolithic verification. However, these numbers do not include the time spent to generate Alice's assumption, which amounts to approximately 5 minutes, implying that the benefits of compositional verification mostly apply when we expect to reuse the generated assumptions for several protocol variants.

IV. Related Work

Compositional verification and assume-guarantee reasoning [27], [32], [25], [26], [28] have been studied extensively, as a way to address the state-space explosion problem in model checking [15]. Progress has been made in automating compositional reasoning using learning and abstraction-refinement techniques for iterative building of the necessary assumptions [17], [31], [11]. Other learning-based approaches for automating assumption generation have been proposed as well, e.g. [12], [2], [13], [14], with many other research works to follow.

All this work was done in the context of applying automated compositional verification to general-purpose software. While there have been many model checkers that target security protocols, for example [4] surveys a number of them and [23], [5] have been applied to Needham-Schroeder protocol, they all verify the entire protocol at once. In fact, there is relatively little research on compositional analysis of security protocols, which pose special challenges due to the complexity introduced by the attacker model. Among the most prominent works in this direction is Protocol Compositional Logic (PCL) [18], a logic and system for proving security properties of network protocols. PCL supports compositional reasoning about complex security protocols and has been applied to a number of industry standards including SSL/TLS, IEEE 802.11 i and Kerberos V5. Despite its success, PCL is limited by the large amount of manual effort that is involved in performing the proofs. Other tools can use the help of humans to guide the proving effort with intermediate lemmas; examples include the Tamarin [33] and the CryptoVerif provers [9]; however, this functionality still requires the entire protocol code. It would be interesting to investigate how to integrate the properties discovered by our framework in such tools. Tamarin is a natural first candidate for experiments in this area, since it works under the symbolic model, just like Taglierion. CryptoVerif, by contrast, is used for proofs in the computational model of cryptography, which would represent a significant depart from our setting.

V. Conclusion and Future Work

We have carried out a first experiment towards automating the compositional verification of protocols, using the NS and NSL protocols as a case study. Our results show that synthesized assumptions can be used to verify variants of the original protocol and yield faster checks. We see several promising directions for future work. Besides trying out more case studies, we would like to improve the performance of our assumption generation, which right now takes a few minutes to complete (≈ 5). It would also be interesting to use the generated assumptions to guide the design and simplification of other protocols, or to incorporate those in manual proofs of correctness.

REFERENCES

[1] Abadi, M., Blanchet, B., Fournet, C.: The applied pi calculus: Mobile values, new names, and secure communication. CoRR **abs/1609.03003** (2016), http://arxiv.org/abs/1609.03003

[2] Alur, R., Madhusudan, P., Nam, W.: Symbolic compositional verification by learning assumptions. In: Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings. pp. 548–562 (2005)

[3] Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6, https://doi.org/10.1016/0890-5401(87)90052-6

[4] Basin, D.A., Cremers, C., Meadows, C.A.: Model checking security protocols. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 727–762. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_22, https://doi.org/10.1007/978-3-319-10575-8_22

[5] Basin, D.A., Cremers, C.J.F., Horvat, M.: Actor key compromise: Consequences and countermeasures. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014. pp. 244–258. IEEE Computer Society (2014). https://doi.org/10.1109/CSF.2014.25, https://doi.org/10.1109/CSF.2014.25

[6] Basin, D.A., Dreier, J., Hirschi, L., Radomirovic, S., Sasse, R., Stettler, V.: A formal analysis of 5g authentication. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 1383–1396. ACM (2018). https://doi.org/10.1145/3243734.3243846, https://doi.org/10.1145/3243734.3243846

[7] Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified models and reference implementations for the TLS 1.3 standard candidate. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 483–502. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.26, https://doi.org/10.1109/SP.2017.26

[8] Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: 14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada. pp. 82–96. IEEE Computer Society (2001). https://doi.org/10.1109/CSFW.2001.930138, https://doi.org/10.1109/CSFW.2001.930138

[9] Blanchet, B.: A computationally sound mechanized prover for security protocols. In: 2006 IEEE Symposium on Security and Privacy (S&P 2006), 21-24 May 2006, Berkeley, California, USA. pp. 140–154. IEEE Computer Society (2006). https://doi.org/10.1109/SP.2006.1, https://doi.org/10.1109/SP.2006.1

[10] Blanchet, B.: Symbolic and computational mechanized verification of the ARINC823 avionic protocols. In: 30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017. pp. 68–82. IEEE Computer Society (2017). https://doi.org/10.1109/CSF.2017.7, https://doi.org/10.1109/CSF.2017.7

[11] Bobaru, M.G., Pasareanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In: Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings. pp. 135–148 (2008)

[12] Chaki, S., Clarke, E.M., Sinha, N., Thati, P.: Automated assume-guarantee reasoning for simulation conformance. In: Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings. pp. 534–547 (2005)

[13] Chen, Y.F., Clarke, E.M., Farzan, A., Tsai, M.H., Tsay, Y.K., Wang, B.Y.: Automated assume-guarantee reasoning through implicit learning. In: Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings. pp. 511–526 (2010)

[14] Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFA's for compositional verification. In: Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings. pp. 31–45 (2009)

[15] Clarke, E., Grumberg, O., Peled, D.: Model Checking. MIT press (December 1999)

[16] Cobleigh, J.M., Giannakopoulou, D., Pasareanu, C.S.: Learning assumptions for compositional verification. In: Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. pp. 331–346 (2003). https://doi.org/10.1007/3-540-36577-X_24, https://doi.org/10.1007/3-540-36577-X_24

[17] Cobleigh, J.M., Giannakopoulou, D., Pasareanu, C.S.: Learning assumptions for compositional verification. In: Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference, TACAS 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings. pp. 331–346 (2003)

[18] Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol composition logic (PCL). Electron. Notes Theor. Comput. Sci. **172**, 311–358 (2007). https://doi.org/10.1016/j.entcs.2007.02.012, https://doi.org/10.1016/j.entcs.2007.02.012

[19] Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on information theory **29**(2), 198–208 (1983)

[20] Kobeissi, N., Bhargavan, K., Blanchet, B.: Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017. pp. 435–450. IEEE (2017). https://doi.org/10.1109/EuroSP.2017.38, https://doi.org/10.1109/EuroSP.2017.38

[21] Lowe, G.: Breaking and fixing the needham-schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1055, pp. 147–166. Springer (1996). https://doi.org/10.1007/3-540-61042-1_43, https://doi.org/10.1007/3-540-61042-1_43

[22] Lowe, G.: A hierarchy of authentication specifications. In: Proceedings 10th Computer Security Foundations Workshop. pp. 31–43. IEEE (1997)

[23] Luo, X., Chen, Y., Gu, M., Wu, L.: Model checking needham-schroeder security protocol based on temporal logic of knowledge. In: 2009 International Conference on Networks Security, Wireless Communications and Trusted Computing. vol. 2, pp. 551–554 (2009)

[24] Magee, J., Kramer, J.: State models and java programs. wiley Hoboken (1999)

[25] McMillan, K.L.: Verification of an implementation of Tomasulo's algorithm by compositional model checking. In: Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings. pp. 110–121 (1998)

[26] McMillan, K.L.: Circular compositional reasoning about liveness. In: Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings. pp. 342–345 (1999)

[27] Misra, J., Chandy, K.M.: Proofs of networks of processes. IEEE Trans. Software Eng. **7**(4), 417–426 (1981)

[28] Namjoshi, K.S., Trefler, R.J.: On the competeness of compositional reasoning. In: Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings. pp. 139–153 (2000)

[29] Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Commun. ACM **21**(12), 993–999 (1978). https://doi.org/10.1145/359657.359659, https://doi.org/10.1145/359657.359659

[30] Pasareanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the l* algorithm to automate assume-guarantee reasoning. Formal Methods Syst. Des. **32**(3), 175–205 (2008). https://doi.org/10.1007/s10703-008-0049-6, https://doi.org/10.1007/s10703-008-0049-6

[31] Pasareanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. Formal Methods in System Design **32**(3), 175–205 (2008)

[32] Pnueli, A.: In transition from global to modular temporal reasoning about programs. In: Logics and Models of Concurrent Systems, NATO ASI Series (1985)

[33] Schmidt, B., Meier, S., Cremers, C.J.F., Basin, D.A.: Automated analysis of diffie-hellman protocols and advanced security properties. In: Chong, S. (ed.) 25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012. pp. 78–94. IEEE Computer Society (2012). https://doi.org/10.1109/CSF.2012.25, https://doi.org/10.1109/CSF.2012.25

[34] Whitefield, J., Chen, L., Sasse, R., Schneider, S., Treharne, H., Wesemeyer, S.: A symbolic analysis of ecc-based direct anonymous attestation. In: IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019. pp. 127–141. IEEE (2019). https://doi.org/10.1109/EuroSP.2019.00019, https://doi.org/10.1109/EuroSP.2019.00019

[35] Woo, T.Y., Lam, S.S.: A semantic model for authentication protocols. In: Proceedings 1993 IEEE Computer Society Symposium on Research in Security and Privacy. pp. 178–194. IEEE (1993)