POSTER: Towards Precise and Automated Verification of Security Protocols in Coq

Hernan M. Palombo
University of South Florida
hpalombo@usf.edu

Hao Zheng
University of South Florida
haozheng@usf.edu

Jay Ligatti
University of South Florida
ligatti@usf.edu

ABSTRACT

Security protocol verification using commonly-used model-checkers or symbolic protocol verifiers has several intrinsic limitations. Spin suffers the state explosion problem; Proverif may report false attacks. An alternative approach is to use Coq. However, the effort required to verify protocols in Coq is high for two main reasons: correct protocol and property specification is a non-trivial task, and security proofs lack automation. This work claims that (1) using Coq for verification of cryptographic protocols can sometimes yield better results than Spin and Proverif, and (2) the verification process in Coq can be greatly alleviated if specification and proof engineering techniques are applied. Our approach is evaluated by verifying several representative case studies. Preliminary results are encouraging, we were able to verify two protocols that give imprecise results in Spin and Proverif, respectively. Further, we have automated proofs of secrecy and authentication for an important class of protocols.

CCS CONCEPTS

• **Security and privacy → Logic and verification**;

KEYWORDS

Security Protocols; Verification; Coq

ACM Reference format:
Hernan M. Palombo, Hao Zheng, and Jay Ligatti. 2017. POSTER: Towards Precise and Automated Verification of Security Protocols in Coq. In *Proceedings of CCS '17, Dallas, TX, USA, October 30-November 3, 2017,* 3 pages.
https://doi.org/10.1145/3133956.3138846

1 INTRODUCTION

Security protocols are an ubiquitous mechanism to establish secure communications over insecure networks. As new technologies emerge, new protocols with specific requirements are being actively developed, e.g. private authentication for IoT [9]. The design of such protocols, however, tends to be error-prone. Attacks have been found on protocols that were thought to be correct for many years [3]. In this context, verification is a useful approach to provide stronger assurances that a protocol design satisfies some desired security policies.

CCS '17, October 30-November 3, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-4946-8/17/10.
https://doi.org/10.1145/3133956.3138846

There are several approaches to formally verify that protocol designs are correct. Model checking uses state transition systems to model the behavior of a system and then uses state exploration to find if an undesired state is reachable (i.e. a counter-example). Symbolic protocol verifiers are tailored tools that usually allow specification in some form of process algebra, and use some –usually automated– proof technique to find counter-examples in a similar way as model-checkers. On the other hand, theorem proving involves defining logical inference rules that describe the semantics of a system, and then using mathematical tools (e.g. induction) to prove different security-related theorems.

Unfortunately, commonly-used model checkers and protocol verifiers may yield imprecise results. For example, since an unbounded message space cannot be verified in Spin, verification may have to be reduced to single session configurations or restricted attacker models, which may lead to unsound results [2]. Further, Proverif sometimes reports false attacks due to its internal approximations [4].

On the other hand, theorem-proving does not have any of the aforementioned problems but involves greater verification effort in general. For example, the proof assistant Coq has a powerful specification language that is highly extensible but lacks built-in automation [1]. To the best of our knowledge, no previous work has used Coq to address Spin and Proverif's impreciseness problems, or explained how the process of verifying security protocols in Coq could be improved. This work fills this gap making two claims about verification of security protocols in Coq.

First, Coq can yield precise results for an important class of protocols that neither Spin nor Proverif can provide. Second, the verification process can be greatly alleviated if specification and proof engineering techniques are applied. To support our claims, we show two examples that give imprecise results in Spin and Proverif, respectively, and report the results of applying an iterative and incremental approach to achieve precise and automated verification of secrecy and authentication properties of several case studies.

Outline. Section 2 presents two examples of imprecise protocol verification in Spin and Proverif, respectively, that motivate our work. Section 3 gives a brief overview of our verification approach in Coq. Section 4 summarizes the results of several case-studies. Section 5 concludes the paper and outlines future work.

2 MOTIVATING EXAMPLES

We present two motivating examples of simple protocols yield imprecise results in Spin and Proverif, respectively.

NSPK in Spin. The NSPK protocol aims to provide mutual authentication between two parties communicating over an insecure

network [11]. The protocol involves three parties, the two authenticating clients and a server which distributes public keys to the clients. Spin cannot verify the full version of the protocol, and finds an attack on a reduced version of the protocol but the attacker model is unsound. [2]. We specify the protocol in Coq and confirm that an impersonation attack is possible. Further, we fix the protocol using a timestamp and prove that the attack is no longer possible.

NSSK in Proverif. The NSSK protocol aims to generate a session key between two parties communicating over an insecure network [11]. When the NSSK protocol is composed with a symmetric encryption scheme to generate and transmit a secret, Proverif reports a false attack [5]. We specify the protocol in Coq and verify that the attack reported by Proverif is indeed false, i.e. the secret is not leaked to an attacker.

More generally, the two examples presented here expose some intrinsic limitations Spin and Proverif that affect verification of a wide class of protocols, properties, and configurations. Table 1 summarizes how the tools are compared in terms of preciseness and automation.

Table 1: Verification of Unbounded Number of Sessions

	SPIN	Proverif	Coq
Sound		✓	✓
No false attacks	✓		✓
Automated	✓	✓	

An unbounded number of sessions cannot be verified in Spin, and only a restricted class of attackers can be verified, i.e. those with very limited storage capabilities. These limitations may lead to results that are unsound, i.e. the verification process succeeds despite attacks are possible.

In Proverif, unbounded number of sessions can be verified but its internal abstractions may generate false attacks. Although an encoding approach using phases was proposed to reduce false attacks, it is guaranteed to be sound for a limited class of protocols, e.g. it does not work for multiparty protocols, and it has not been proven to work for correspondence assertions [5]. Further, it is difficult to know a priori if Proverif's output correspond to a false attack and phases need to be inserted into the encoding, and interpreting Proverif's outputs to rule out false attacks is sometimes nontrivial [10].

3 VERIFICATION IN COQ

A high level view of the verification process in Coq is shown in Figure 1. We follow a modular and incremental approach for developing specifications and proofs. We start with a base specification describing the protocol, attacker, and security properties that we want to prove. If the proofs succeed, we refine the specification and prove the security properties of the refinement. The process continues incrementally until either the specification is complete and all properties have been proved, or some property fails to hold and we conclude that the property is insecure. Note that the properties being proved are decidable, so in every case we can reach a conclusion.

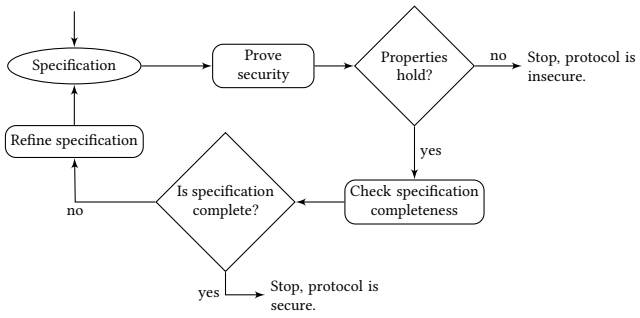

Figure 1: Incremental Verification in Coq.

Next, we discuss some general principles that guide our specification and proof development process.

3.1 Specifications

The specification process entails formalizing three main components: (1) the protocol, (2) an attacker model, and (3) the security properties.

3.1.1 Protocols. Informal protocol specifications describe a sequence of steps of the expected interactions between participants. There are several implicit assumptions in informal specifications that a formalization must disambiguate, e.g. the initial knowledge of the protocol's participants.

Our specification in Coq relies heavily on inductive definitions. First, we define the agents participating in the protocol. Second, we define messages, which can be primitives, e.g. names, nonces, and keys, or compositions of other messages, e.g. pairs or encrypted messages. Third, we define the expected protocol events, i.e. transitions. Because we are modeling an insecure network, send and receive operations are defined non-atomically as mutually inductive propositions.

3.1.2 Attacker Models. An attacker model must specify the attacker's initial knowledge and possible actions. Dolev-Yao attackers [8] have been widely used in the symbolic model. A Dolev-Yao attacker is one who may be able to spy, intercept, or inject messages at will. Weaker attacker models, e.g. a passive eavesdropper, are also sometimes desired.

In Coq, the attacker can be modeled by adding her capabilities to the mutually inductive definition of send and receive operations. Our definition contains rules for attacker's possible actions, e.g. constructors for names, nonces, cryptographic operations, and (de)composition of messages.

3.1.3 Security Properties. Two main properties that are usually requirements of security protocols are secrecy and authentication. Both can be formulated as safety properties.

Secrecy ensures that secret data can only be seen by authorized users. More precisely, secrecy is an invariant asserting that a token marked as secret cannot be learned by an attacker. In Coq, secrecy is formalized as a theorem stating that the attacker can never learn the secret.

The goal of authentication is to assert that an agent is really who he claim to be. An authentication protocol satisfies the property iff

every time the authentication of a requester succeeds, the requester is whomever the authenticator expects to have authenticated. In Coq, it is an implication stating that if the protocol ends (and authentication succeeds), the authenticator has not authenticated the attacker.

3.2 Proofs

The proofs for secrecy and authentication properties are completed in several steps.

First, we specify a separate inductive definition of the attacker's knowledge that is used in the proofs of secrecy and authentication. This definition is different from the attacker's model described in Section 2 in that it indicates what the attacker may eventually learn as opposed to her capabilities.

Next, we prove several related lemmas; one stating that anything that is received can be learned by the attacker; a second one saying that the attacker learns everything that follows from the attacker model; and a third one asserting that what cannot be learned by the attacker is outside the attacker model. With these lemmas in hand, secrecy and authentication theorems are then proved using case analysis.

4 OTHER CASE-STUDIES

Apart from the examples presented in Section 2, we verify a representative sample of authentication and key-establishment protocols, namely the DS [6], TMN [12], and DH [7]. Our case-study selection is motivated by the relevance of the protocols and the observation that many security protocols use a common set of cryptographic primitives, which are present in the cases analyzed here.

For instance, the DS protocol, proposed as a fix to the NSPK protocol, is chosen to exercise the tools with a protocol that uses signatures, timestamps, and symmetric encryption. The TMN protocol serves as an interesting example for its use of Vernam encryption techniques, i.e. XOR functions, a technique commonly used in cryptography. Finally, DH is chosen because many protocol families, e.g. TLS, rely on the use of exponentiation for an unauthenticated agreement of a shared key between two parties.

4.1 Results

In most cases, we are able to prove that secrecy and authentication hold. However, some versions of the protocols are flawed and the proofs do not go through.

For example, for the DS protocol, we start with the original version described by Dennim and Sacco [6] which is known to be insecure. The proof of secrecy fails because an attacker can perform a MITM attack and the protocol may end with Bob sending the secret nonce to the attacker. To confirm the protocol's insecurity, the negation of the secrecy lemma is proved, which states that the attacker may be able to learn the secret nonce.

Finally, we fix the protocol specification and attempt to prove secrecy again. This time both proofs succeed as the MITM attack is no longer possible.

4.2 Automation

In all cases, the proofs follow a similar pattern. Based on this observation, the next step is to automate the proofs to increase the robustness and generality of our approach. For secrecy, we add the constructors of the attacker's knowledge-vector to the `Hint` database and then use the `auto` tactic to complete the proof. For authentication, we define a new tactic `solve_auth` in *Ltac* which pattern matches on the current goal.

5 CONCLUSION AND FUTURE WORK

This work shows that Coq provides precise results for verification of security protocols that cannot be directly obtained in Spin or Proverif. A second contribution is that the verification process in Coq can be significantly improved if specifications are developed modularly and incrementally, and proofs are automated. Towards this end, we explained general specification principles and automated proofs of secrecy and authentication properties for an important class of authentication and key-establishment protocols. Our ongoing work is to make the approach more general by verifying other types of protocols, e.g. those with privacy-related properties, adding more automation, and analyzing larger case studies.

REFERENCES

[1] Bruno Barras, Samuel Boutin, Cristina Cornes, et al. 2002. *The Coq proof assistant reference manual–version 7.2.* Technical Report. Technical Report 0255, INRIA.

[2] Noomene Ben Henda. 2014. Generic and efficient attacker models in SPIN. In *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*. ACM, 77–86.

[3] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. 2017. *Verified models and reference implementations for the TLS 1.3 standard candidate.* Ph.D. Dissertation. Inria Paris.

[4] Bruno Blanchet, Ben Smyth, and Vincent Cheval. 2016. ProVerif 1.94 pl1: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. (2016).

[5] Tom Chothia, Ben Smyth, and Chris Staite. 2015. Automatically checking commitment protocols in proverif without false attacks. In *International Conference on Principles of Security and Trust*. Springer, 137–155.

[6] Dorothy E Denning and Giovanni Maria Sacco. 1981. Timestamps in key distribution protocols. *Commun. ACM* 24, 8 (1981), 533–536.

[7] Whitfield Diffie and Martin E Hellman. 1976. New directions in cryptography. *Information Theory, IEEE Transactions on* 22, 6 (1976), 644–654.

[8] Danny Dolev and Andrew C Yao. 1983. On the security of public key protocols. *Information Theory, IEEE Transactions on* 29, 2 (1983), 198–208.

[9] Mohamed Amine Ferrag, Leandros A Maglaras, Helge Janicke, and Jianmin Jiang. 2016. Authentication Protocols for Internet of Things: A Comprehensive Survey. *arXiv preprint arXiv:1612.07206* (2016).

[10] Murat Moran and Dan S Wallach. 2017. Verification of STAR-Vote and Evaluation of FDR and ProVerif. *arXiv preprint arXiv:1705.00782* (2017).

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

[12] Makoto Tatebayashi, Natsume Matsuzaki, and David B Newman. 1989. Key distribution protocol for digital mobile communication systems. In *Conference on the Theory and Application of Cryptology*. Springer, 324–334.