Formal Methods Analysis of the Secure Remote Password Protocol

Alan T. Sherman[1(✉)], Erin Lanus[2], Moses Liskov[3], Edward Zieglar[4], Richard Chang[1], Enis Golaszewski[1], Ryan Wnuk-Fink[1], Cyrus J. Bonyadi[1], Mario Yaksetig[1], and Ian Blumenfeld[5]

[1] Cyber Defense Lab, University of Maryland, Baltimore County (UMBC), Baltimore, MD 21250, USA
sherman@umbc.edu

[2] Virginia Tech, Arlington, VA 22309, USA
lanus@vt.edu

[3] The MITRE Corporation, Burlington, MA 01720, USA
mliskov@mitre.org

[4] National Security Agency, Fort George G. Meade, MD 20755, USA
evziegl@nsa.gov

[5] Two Six Labs, Arlington, VA 22203, USA
ian.blumenfeld@twosixlabs.com

Abstract. We analyze the *Secure Remote Password (SRP)* protocol for structural weaknesses using the *Cryptographic Protocol Shapes Analyzer (CPSA)* in the first formal analysis of SRP (specifically, Version 3).

SRP is a widely deployed *Password Authenticated Key Exchange (PAKE)* protocol used in 1Password, iCloud Keychain, and other products. As with many PAKE protocols, two participants use knowledge of a pre-shared password to authenticate each other and establish a session key. SRP aims to resist dictionary attacks, not store plaintext-equivalent passwords on the server, avoid patent infringement, and avoid export controls by not using encryption. Formal analysis of SRP is challenging in part because existing tools provide no simple way to reason about its use of the mathematical expression $v + g^b \mod q$.

Modeling $v + g^b$ as encryption, we complete an exhaustive study of all possible execution sequences of SRP. Ignoring possible algebraic attacks, this analysis detects no major structural weakness, and in particular no leakage of any secrets. We do uncover one notable weakness of SRP, which follows from its design constraints. It is possible for a malicious server to fake an authentication session with a client, without the client's participation. This action might facilitate an escalation of privilege attack, if the client has higher privileges than does the server. We conceived of this attack before we used CPSA and confirmed it by generating corresponding execution shapes using CPSA.

Keywords: Cryptographic protocols · Cryptography · Cryptographic Protocol Shapes Analyzer (CPSA) · Cybersecurity · Formal methods · Password Authenticated Key Exchange (PAKE) protocols ·

V. Nigam et al. (Eds.): Scedrov Festschrift, LNCS 12300, pp. 103–126, 2020.
https://doi.org/10.1007/978-3-030-62077-6_9

Protocol analysis · Secure Remote Protocol (SRP) · UMBC Protocol Analysis Lab (PAL)

1 Introduction

Cryptographic protocols underlie most everything that entities do in a networked computing environment, yet, unfortunately, most protocols have never undergone any formal analysis. Until our work, this situation was true for the widely deployed *Secure Remote Password (SRP)* protocol [28,51–53]. Given the complexity of protocols and limitations of the human mind, it is not feasible for experts to find all possible structural flaws in a protocol; therefore, formal methods tools can play an important role in protocol analysis.

Protocols can fail for many reasons, including structural flaws, weak cryptography, unsatisfied hypotheses, improper configuration, inappropriate application, and implementation errors. We focus on structural weaknesses: fundamental logic errors, which enable an adversary to defeat a protocol's security objective or learn secret information.

We analyze SRP for structural weaknesses in the first formal analysis of SRP (specifically, Version 3, known as *SRP-3*). Using the *Cryptographic Protocol Shapes Analyzer (CPSA)* [36] tool in the *Dolev-Yao network intruder model* [21], we model SRP-3 and examine all possible execution sequences of our model. CPSA summarizes these executions with graphical "shapes," which we interpret.

SRP is a *Password Authenticated Key Exchange (PAKE)* protocol used in 1Password, iCloud Keychain, and other products. As with many PAKE protocols, two participants use knowledge of a pre-shared password to authenticate each other and establish a session key. SRP aims to resist dictionary attacks, not store plaintext-equivalent passwords on the server, avoid patent infringement, and avoid export controls by not using encryption.

Formal analysis of any protocol is challenging, and analysis of SRP is particularly difficult because of its use of the mathematical expression $v + g^b \mod q$. This expression involves both modular exponentiation and modular addition, exceeding the ability of automated protocol analysis tools to reason about modular arithmetic. Although SRP claims to have no encryption, ironically, we overcome this difficulty by modeling the expression as encryption, which, effectively it is.

We created a new virtual protocol analysis lab at UMBC. Embodied as a virtual machine running on the Docker utility,[1] this lab includes documentation, educational modules for learning about protocol analysis, and three protocol analysis tools: CPSA, Maude-NPA [24,25], and Tamarin Prover [22].

Contributions of our work include: (1) The first formal analysis of the SRP-3 protocol for structural weaknesses, which we carried out using the CPSA tool. Ignoring possible algebraic attacks, this analysis detects no major structural weakness, and in particular no leakage of any secrets. (2) The discovery of the first attack on SRP, in which it is possible for a malicious server to fake an

[1] www.docker.com.

authentication session with the client, without the client's participation. This action might facilitate an escalation of privilege attack, if the client has higher privileges than does the server.

2 Background and Previous Work

We briefly review formal methods for analyzing cryptographic protocols, CPSA, PAKE protocols, and previous work on SRP.

2.1 Formal Methods for Analyzing Cryptographic Protocols

Several tools exist for formal analysis of cryptographic protocols, including CPSA [19,20,29], *Maude-NPA* [24,25], the Tamarin Prover [45], and ProVerif [8]. Created in 2005, CPSA outputs a set of "shapes" that describe all possible protocol executions, which can reveal undesirable execution states including ones caused by adversarial interference. Developed by Meadows [40] in 1992 as the NRL Protocol Analyzer, and rewritten into Maude language by Escobar et al. [23] in 2005, Maude-NPA works backwards from explicitly-defined attack states. The Tamarin Prover uses a multiset-rewriting model particularly well suited for analyzing stateful protocols. ProVerif is an automated cryptographic protocol verifier that operates on protocol specifications expressed in applied pi calculus, which specifications it translates into Horn clauses. We choose to use CPSA because we are more familiar with that tool, have easy access to experts, and like its intuitive graphical output.

A variety of additional tools exist to support formal reasoning, including for cryptography. For example, created in 2009, EasyCrypt[2] supports "reasoning about relational properties of probabilistic computations with adversarial code ... for the construction and verification of game-based cryptographic proofs." Cryptol [11] is a domain-specific language for cryptographic primitives. Cryptol allows for the symbolic simulation of algorithms, and thus the ability to prove properties of such by hooking into various constraint (SAT/SMT) solvers. Additionally, interactive theorem provers, such as Isabelle or Coq, have been used to analyze cryptographic functions and protocols [3,42]. These tools offer the potential to verify any property expressible in their underlying logics (higher-order logic or dependent type theory, respectively) but sacrifice automation.

The 1978 Needham-Schroeder [41] public-key authentication protocol dramatically illustrates the value of formal methods analysis and limitations of expert review. In 1995, using a protocol analysis tool, Lowe [38] identified a subtle structural flaw in Needham-Schroeder. This flaw had gone unnoticed for 17 years in part because Needham and Schroeder, and other security experts, had failed to consider the possibility that the intended recipient might be the adversary. Thus, for example, if Alice authenticates to Bob, then Bob could impersonate Alice to Charlie. CPSA easily finds this unexpected possible execution sequence, outputting a suspicious execution shape.

[2] https://www.easycrypt.info/trac/#no1.

Cryptographers sometimes present a *Universal Composability (UC)* proof of security [12], but such proofs as typically written are long and complex and can be difficult to verify. For example, Jarecki, Krawczyk, and Xu's [33] UC proof of the OPAQUE protocol is in a 61-page complex paper. There is, however, recent work on mechanically checking UC proofs (e.g., see Canetti, Stoughton, and Varia [13]), including Dolev-Yao versions of UC (e.g., see Böhl and Unruh [9] and Delaune, Kremer, and Pereria [16].) By contrast, to analyze SRP-3, CPSA requires only a relatively short and easy-to-verify input that formally defines the protocol in terms of its variables, the participant roles, and the messages sent and received.

2.2 Cryptographic Protocol Shapes Analyzer

The *Cryptographic Protocol Shapes Analyzer (CPSA)* [29,36,43] is an open-source tool for automated formal analysis of cryptographic protocols. The tool takes as input a model of a cryptographic protocol and a set of initial assumptions called the *point of view*, and attempts to calculate a set of minimal, essentially different executions of the protocol consistent with the assumptions. Such executions, called *shapes*, are relatively simple to view and understand. Executions in which something "bad" happens amount to illustrations of possible attacks against the protocol. Conversely, when some property holds in all shapes, it is a property guaranteed by the protocol.

CPSA is a tool based on *strand space theory* [20,26], which organizes events in a partially-ordered graph. In strand space theory, *events* are transmissions or receptions of messages, and sequences of events called *strands* capture the notion of the local viewpoint of a participant in a network. CPSA also has *state events*, which comprise initializing, observing, and transitioning between states. Protocols are defined as a set of legitimate participant roles, which serve as templates for strands consistent with the protocol requirements.

Bundles are the underlying execution model, in which every reception is explained directly by a previous transmission of that exact message. A bundle of a particular protocol is a bundle in which all the strands are either (1) generic adversary behavior such as parsing or constructing complex messages, or encrypting or decrypting with the proper keys, or (2) behavior of participants in the protocol consistent with the protocol roles.

CPSA reasons about bundles indirectly by analyzing *skeletons*, which are partially-ordered sets of strands that represent only regular behavior, along with origination assumptions that stand for assumptions about secrecy and/or freshness of particular values. For example, such assumptions might include that a key is never revealed or a nonce is freshly chosen and therefore assumed unique. Some skeletons represent, more or less, the exact set of regular behavior present in some bundle consistent with the secrecy and freshness assumptions; such skeletons are called *realized* skeletons. Realized skeletons are a simplified representation of actual protocol executions. Non-realized skeletons may represent partial descriptions of actual executions, or may represent a set of conditions inconsistent with any actual execution [36].

The CPSA tool creates visualizations of skeletons as graphs in which events are shown as circles in columns, where each column represents a strand. Within each strand, time progresses downward. *Arrows* between strands indicate necessary orderings (other than orderings within strands, or those that can be inferred transitively). That is, an arrow from event P to event Q denotes that, for Q to take place, it is necessary for P to take place first. A *solid arrow* represents a transmission of some message to a reception of exactly that message. A *dashed arrow* indicates that the adversary altered the message. The color of a circle indicates the type of event: *black circles* are transmissions; *blue circles* are receptions; and *grey circles* deal with state that is assumed to be not directly observable by the attacker. A *blue arrow* from state event P to state event Q denotes that Q's strand observes, or transitions from, the state associated with P; it can appear only between two state events (e.g., grey circles) of different strands. For example, Fig. 3 in Sect. 5.1 shows such a visualization.

2.3 PAKE Protocols

PAKE protocols evolved over time in response to new requirements and newly discovered vulnerabilities in authentication protocols [10]. Initially, authentication over a network was carried out simply with a username and password sent in the clear. Unlike terminals hardwired to a computer, networks provided new and easier ways for intruders to acquire authentication credentials. Passively monitoring a network often harvested credentials sufficient to gain remote access to systems. In the 1980's, *Kerberos* [47] attempted to mitigate this vulnerability by no longer transmitting passwords. Unfortunately, the structure of Kerberos messages and the use of passwords as keys created opportunities for password guessing and dictionary attacks against the passwords, without requiring the intruder to acquire the password file directly from the server. Weak, user-chosen passwords simplified such attacks.

In 1992, with their *Encrypted Key Exchange (EKE)* protocols, Bellovin and Merrit [6] evolved PAKE protocols to address the weaknesses in user-generated passwords as keys. In 1996, that work led Jablon [32] to develop the *Simple Password Exponential Key Exchange (SPEKE)*, which is deployed in the ISO/IEC 11770-4 and IEEE 1363.2 standards. As did Kerberos, to complicate dictionary attacks, SPEKE incorporated random *salt* values into its password computations. Attacks against the protocol in 2004 [54], 2005 [48], and 2014 [31], prompted modifications to the protocol. Although these and similar protocols aimed to protect against the use of weak passwords for authentication, none protected the passwords from attack on the server's password file. Access to the server's password file provided keys to authenticate as any user on the system.

Protection of the server's authentication file became a primary new requirement that Wu [51,52] aimed to address with the *Secure Remote Password (SRP)* protocol in 1998. Wu addressed this requirement by not storing the password, but instead a *verifier* consisting of a modular exponentiation of a generator raised to the power of a one-way hash function of the password. Improving on earlier

PAKE protocols, the way SRP incorporates a random salt into the key computation prevents the direct use of server-stored verifiers as keys. In 2002, weaknesses discovered against SRP-3 led Wu to propose a new version, *SRP-6* [53].

Unfortunately, for each password, SRP publicly reveals the corresponding salt, which facilitates pre-computation dictionary attacks on targeted passwords. Aware of this vulnerability, Wu nevertheless considered SRP a significant improvement over what had come before. Avoiding pre-computation attacks led to new approaches including the *OPAQUE* protocol [27,33,34].

2.4 Previous Work

SRP [49,50] is a widely used password-authenticated key-establishment protocol, which enables two communicants to establish a secret session key, provided the communicants already know a common password. SRP is faster than the authenticated Diffie-Hellman key exchange protocol, and it aims to avoid patent infringement and export control. In this protocol, an initiator Alice (typically a client) authenticates to a responder Bob (typically a server).

In this paper, we analyze the basic version of SRP called *SRP-3*. *SRP-6* mitigates a two-for-one attack and decreases communication times by allowing more flexible message orderings.

Against a passive adversary, SRP-3 seems to be as secure as the Diffie-Hellman problem [17,28,39]. It remains possible, however, that a passive adversary can acquire information from eavesdropping without solving the Diffie-Hellman problem. Against an active adversary, the security of SRP-3 remains unproven.

Wu [52] claims to prove a reduction from the Diffie-Hellman problem to breaking SRP-3 against a passive adversary, but his proof is incorrect: his reduction assumes the adversary knows the password, which a passive adversary would not know.[3] We are not aware of any other previous effort to analyze the SRP protocol.

Wilson et al. [7] survey authenticated Diffie-Helman key agreement protocols. Adrian et al. [1] analyze how such protocols can fail in practice. Schmidt et al. [45] present automated analysis of Diffie-Helman protocols.

As an example of formal analysis of a protocol using CPSA, we note: In 2009, Ramsdell et al. [43] analyzed the CAVES attestation protocol using CPSA, producing shapes that prove desirable authentication and confidentiality properties. The tool successfully analyzed the protocol despite the presence of hash functions and auxiliary long-term keys. As another example, which illustrates the utility of service roles, see Lanus and Zieglar [35]. Corin, Doumen, and Etalle [15] symbolically analyze offline guessing attacks.

[3] Wu incorrectly states the direction of his reduction, but his reduction actually proceeds in the correct direction.

3 The Secure Remote Password Protocol

Figure 1 summarizes how SRP-3 works, during which Alice and Bob establish a secret *session key* K, leveraging a *password* P known to Alice and Bob.

In SRP-3, all math is performed in some prime-order group $\mathbb{Z}_q$, where q is a large prime integer. Let g be a generator for this group. The protocol uses a hash function h. For brevity, for any $x \in \mathbb{Z}_q$, we shall write g^x to mean $g^x \bmod q$.

SRP-3 works in three phases: I. Registration. II. Key Establishment and III. Key Verification. The protocol establishes a new session key K known to Alice and Bob, which they can use, for example, as a symmetric encryption key.

Phase I works as follows: Before executing the protocol, Alice must register her password P with Bob. Bob stores the values (s, v) indexed by "Alice", where s is a random *salt*, $x = h(s, P)$ is the salted hash value of Alice's password, and $v = g^x$ is a non-sensitive *verifier* derived from P, which does not reveal x or P.

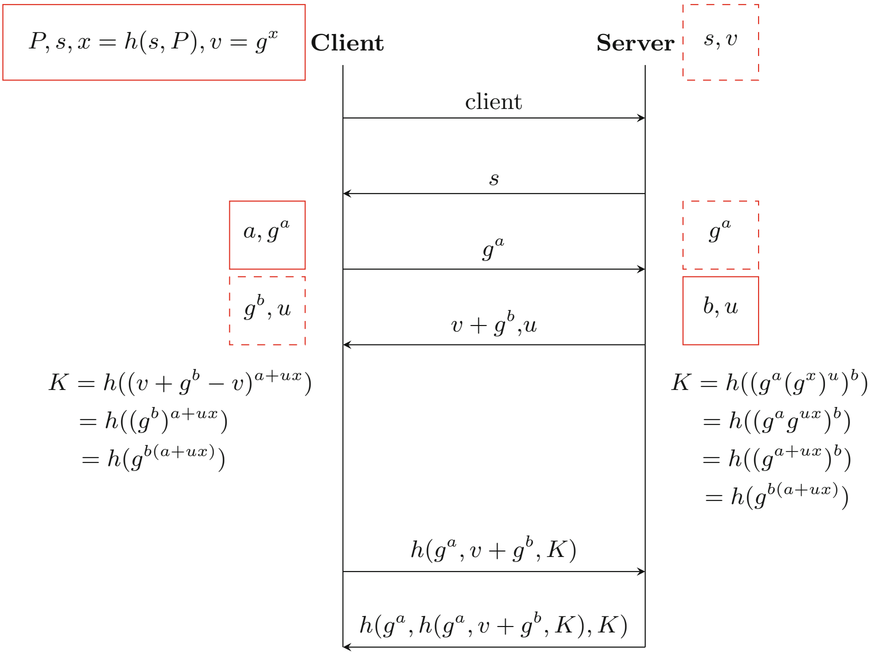

Fig. 1. Protocol diagram for SRP-3, which comprises three phases: Registration, Key Exchange, and Key Verification. During key exchange, the server transmits to the client the expression $v + g^b \mod q$, which we cannot directly model in CPSA. Variables on arrows inside the lattice diagram indicate message transmissions. Variables to the left or right of the lattice indicate terms known to the participants and the relative time within the protocol that they know them. Variables in solid boxes denote values chosen. Variables in dashed boxes denote values received.

Phase II works as follows:

1. Alice sends her identity "Alice" to Bob.
2. Bob receives Alice's identity and looks up Alice's salt s and stored verifier $v = g^x$, where $x = h(s, P)$. Bob sends Alice her salt s.
3. Alice receives s, calculates $x = h(s, P)$, and generates a random secret nonce a. Alice calculates and sends g^a to Bob.
4. Bob receives g^a and generates a random secret nonce b and a random scrambling parameter u. Bob calculates and sends $v + g^b$ to Alice, together with u.
5. Each party calculates the session key K as the hash of a common value, which each party computes differently. Alice calculates $K = h((v + g^b) - g^x)^{a+ux}$ and Bob calculates $K = h(g^a g^{ux})^b$.

Thus, in Phase II, Alice and Bob establish a common session key K. In Phase III, Alice and Bob verify that they have the same session key. Phase III works as follows:

1. Alice computes $M_1 = h(g^a, v + g^b, K)$ and sends M_1 to Bob. Bob verifies the received value by recomputing $M_1 = h(g^a, v + g^b, K)$.
2. Bob computes $M_2 = h(g^a, M_1, K)$ and sends it to Alice. Alice verifies the received value by recomputing $M_2 = h(g^a, M_1, K)$.
3. If and only if these two verifications succeed, the session key K is verified.

4 Modeling SRP-3 in CPSA

Using CPSA, we analyze SRP-3 in the Dolev-Yao network intruder model in two steps: in this section, we model SRP-3 in CPSA; in the next section, we interpret shapes produced by our model. Appendix A lists important snippets of our CPSA sourcecode.

4.1 Challenges to Modeling SRP-3 in CPSA

CPSA provides two algebras to express protocols: basic and Diffie-Hellman. The basic crypto algebra includes functions that support modeling of pairings, decomposing a pair into components, hashing, encrypting by symmetric and asymmetric keys, decrypting by keys, returning the "inverse of a key" (a key that can be used to decrypt), and returning a key associated with a name or pair of names. CPSA does not support arithmetic operations. The Diffie-Hellman algebra extends the basic crypto algebra by providing *sorts* (variable types) that represent exponents and bases, as well as functions for a standard generator g, a multiplicative identity for the group, exponentiation, and multiplication of exponents.

SRP-3 is challenging to model in CPSA because CPSA does not support any of the following computations: addition of bases when the server sends $v + g^b$, subtraction of bases when the client computes $(v + g^b) - v$, and addition of exponents (i.e., multiplication of bases) when the client computes the key. CPSA handles only multiplication of exponents, and cannot be easily modified to handle these additional algebraic operations, because CPSA makes use of general unifications in its class of messages, and a full decision procedure in the theory of rings is undecidable [14].

4.2 Our Model of SRP-3

We model SRP-3 by defining variables, messages, and associated roles. Critical modeling decisions are how to represent the problematic expression $v + g^b$, how to deal with multiplication of bases, and how to handle the initialization phase. Figure 2 shows the SRP-3 protocol diagram as we modeled SRP-3 in CPSA.

There are two legitimate protocol participants, which we model by the client and server roles (see Fig. 8). We organize each of these roles into two phases: initialization and main. The initialization phase establishes and shares the password, and it establishes the salt and verifier in the long-term memory of the server.

We model the problematic expression $v+g^b$ as $\{|g^b|\}_v$, which is the encryption of g^b using v as a symmetric key. Indeed, this modular addition resembles a Vernam Cipher. Thus, knowing g^b requires knowledge of v. Previous researchers have similarly modeled modular addition or exclusive-or as encryption (e.g., see Arapinis et al. [2] and Ryan and Schneider [44].)

The other problematic expressions occur in the calculation of the key. The key K is supposed to be equal to $(g^b)^{a+ux}$. Here, each party calculates this value by calculating g^{ab} and g^{bux} and multiplying them together. The client can calculate these values from g^b by raising g^b to the a power and to the ux power. The server calculates these values by raising g^a to the b power, and by raising $g^x = v$ to the bu power.

We emulate the multiplication of these base values by hashing them; since both parties can calculate the two factors, each can calculate the hash of the two factors. Thus, we represent the key K as $K = h(g^{ab}, g^{bux})$, where h stands for cryptographic hashing.

Finally, we explain how we model the initialization phase, and in particular, how the client communicates their salt and verifier to the server. In the beginning of the client and server roles, one could exchange the salt and verifier as a message. This strategy, however, would prevent CPSA from exploring scenarios in which the same client or server conducts multiple executions of the protocol using the same password information exchanged during initialization. Instead, we model the initialization phase using *service roles*, which provide a function or service to one or more participant roles. Our service roles generate values, store them in state, and exchange the values across a secure channel. These values persist in state that can be accessed only by instances of the appropriate main-phase roles.

Specifically, the client-init service role initializes a state record with the value {"client state", s, x, client, server} (see Fig. 8). The "client state" string literal serves the function of a label, enabling us to write client roles to observe state that begins with that string. We store the salt and password hash because each client role directly uses these values. The names of the client and server help to link the state to the correct client-server pair.

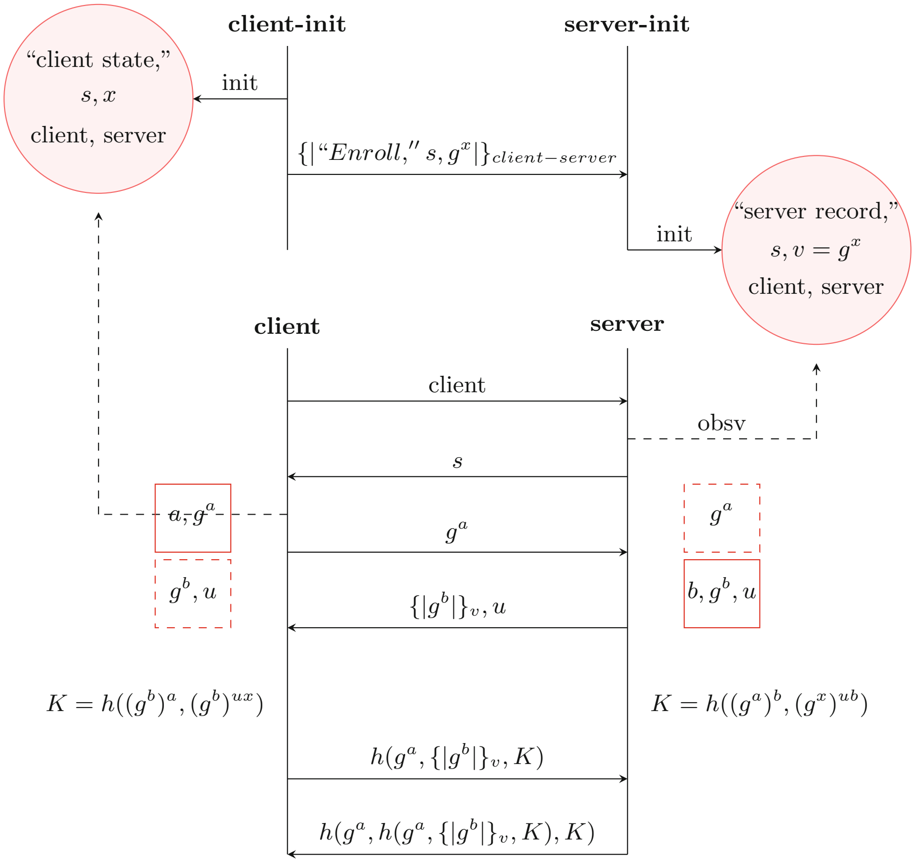

Fig. 2. Protocol diagram for SRP-3, as we modeled it in CPSA. We introduce two service roles, client-init and server-init, that handle the setup phase by instantiating values for s, x, and $v = g^x$, and by making these values available to the legitimate client and server. We model the computation $v + g^b$ as an encryption of g^b under key v. The red circle indicates the variables stored inside the state. Solid lines pointing to the circles denote initializing state values, and dashed lines indicate observing state. (Color figure online)

After initializing its state, the client-init role sends a string literal "Enroll", together with the salt and verifier. The client-init role encrypts this message using a long-term key known by the particular client and server. The server-init role receives this message and initializes the server's state by storing a string literal "server record", the salt and verifier, and the names of the client and server.

To prevent CPSA from instantiating an unlimited number of server-init and client-init roles, we add a rule that disregards any executions in which there is more than one instance of the server-init role for a specific client-server pair (see Fig. 9).

The model above is sufficient to verify most of the security properties of SRP, but cannot verify the property that compromise of the server's authentication database cannot be used directly to gain immediate access to the server. The reason is that if SRP meets its security goals, the verifier v is not leaked to the adversary by the protocol. Therefore, to test whether or not access to v allows the adversary to impersonate a client to the server, we need to use a model in which the server-init role is modified to transmit the verifier it receives for a client. This model provides the adversary with access to v that they cannot obtain from SRP. For this property, it is sufficient to test only the server's point of view. Compromise of a server's authentication database would allow anyone to impersonate a server to the client and is not a property that SRP was designed to prevent.

5 Interpreting Shapes from the SRP-3 Model

We generate and interpret shapes showing executions of our model of SRP-3 under various assumptions from the perspectives of various roles. Specifically, we define skeletons that provide the perspectives of an honest client and an honest server, respectively (see Figs. 10 and 11). We also define *listeners* to detect possible leaked values of the password hash x or verifier v (see Figs. 12 and 13). Finally, we investigate if an adversary directly using a compromised verifier could authenticate as a client (see Fig. 6). CPSA completed its search, generating all possible shapes for each point of view (see [37] for an explanation).

Figures 3, 4, 5 and 6 display selected shapes that highlight our main findings. These shapes show that, when the client and server are honest, there is no attack against our model of SRP-3: the only way the protocol completes is between a client and a server. Similarly, CPSA found no leakage of x or v. CPSA also found that an adversary directly using a compromised verifier cannot authenticate as a client without access to internal values of the server.

Our public GitHub repository [46] includes interactive web-based visualizations of our CPSA shapes and skeletons, which provide more detailed information than do the static images in this paper.

5.1 Client Point of View

Figures 3 and 4 show the two shapes generated from the perspective of an honest client. The first shape is what we had expected. One added client-init strand provides state needed for the client to access password information, and one added server-init strand provides password information to the server strand. The solid lines in the shape prove that the messages must come from the expected parties, and the shape closely reflects the protocol diagram for our model.

The second shape explores the possibility that the adversary could replay the client's initial message to the server resulting in the server beginning two protocol runs with the client. We are able to verify that it is the same server by observing that the server variables in both strands are instantiated with the same value.

Only one of the server strands is able to complete, because the messages between the two runs of the protocol cannot be confused. The shape indicates that there is not any way for the adversary to take advantage of initiating multiple runs of the protocol with the server.

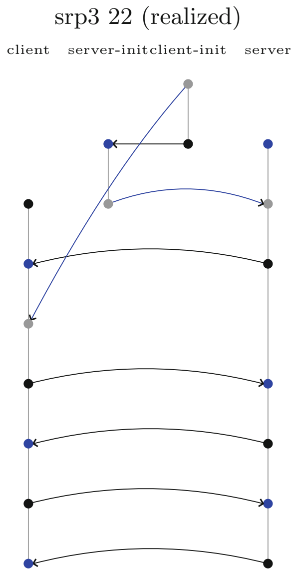

Fig. 3. Shape showing an execution of SRP-3 from the client's perspective. The client-init service role begins the execution. The blue arrow from the client-init strand to the client strand denotes that the client observes the initial state from client-init. Similarly, the blue arrow from the server-init strand to the server strand denotes that the server observes state from server-init. Horizontal black arrows between the client and server represent successful message transmissions and receptions between these two protocol participants. This graphical output from CPSA reveals expected behavior. (Color figure online)

5.2 Server Point of View

Figure 5 shows the first of two shapes generated from the perspective of an honest server. As happens for the client, two shapes result. The first shape is similar to the protocol diagram for our model and is what we had expected. A client is needed to complete the protocol, as are the service roles server-init and client-init. The second shape indicates a replay of the client's initial message resulting in two server strands with the same server as indicated in the strands' instantiated variables. As with the additional shape in the client's view, only one of the server's strands is able to complete, indicating that there is no attack against the protocol from the server point of view.

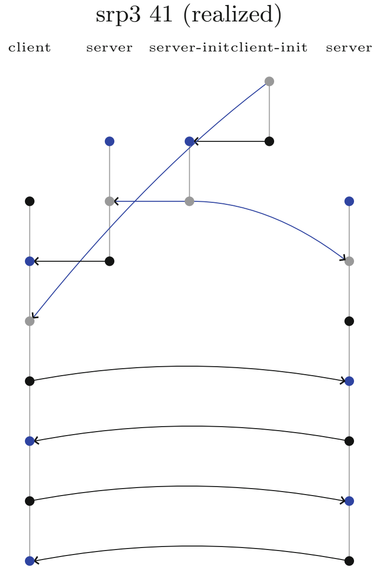

Fig. 4. Shape showing an execution of SRP-3 from the client's perspective, with an additional run of the server. This graphical output from CPSA reveals two server roles accessing the same state, causing them to behave like two instances of the same server. The client can begin the protocol with one instance of the server, then complete it with the other. This intriguing shape does not suggest any harmful attack but is an unavoidable consequence of CPSA exploring two server strands.

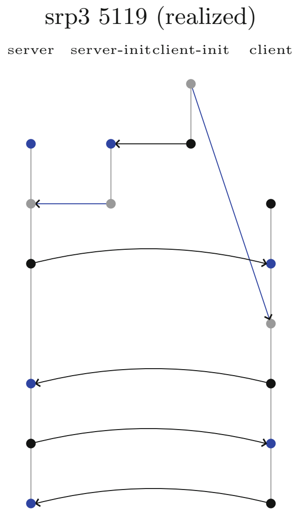

Fig. 5. Shape showing an execution of SRP-3 from the server's perspective. This figure is similar to Fig. 3, except CPSA is now trying to explain the server events. CPSA is able to explain the server events only by involving client-init, server-init, and client roles, thus revealing expected behavior.

5.3 Privacy Properties

It is important that the password hash $x = h(s, P)$ and the verifier $v = g^x$ remain secret. To determine whether a network adversary can observe either of these values in our model of SRP-3, we define two input skeletons to test these privacy properties, one for x and one for v (see Figs. 12 and 13). Because the client knows x, we add the listener for x to the client point of view. Similarly, because the server knows v, we add the listener for v to the server point of view. Listeners in CPSA represent a test that a value can be found by the adversary.

For each of these skeletons, we ran CPSA. In each case, CPSA returned an empty tree, meaning that there is no way to realize the skeleton as a shape, which means that no such attack is possible in our model. In each case, CPSA ran to completion, indicating that it explored all possible shapes for the model.

5.4 Leaked Verifiers

CPSA analysis of listeners for v confirms that the SRP protocol does not leak the verifier v. Therefore, to analyze the protocol when the adversary has access to v, we modified server-init to leak the verifier to the adversary. In the presence of this variant of the server-init role, CPSA discovered two main shapes: one is the ordinary server point of view (Fig. 5); the other shows that the adversary is able to impersonate a client if the verifier has indeed leaked (Fig. 6).

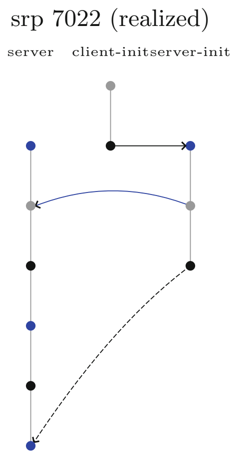

Fig. 6. Shape showing an execution of SRP-3 from the server's perspective, when the verifier is leaked to the adversary and $u = b$. It is suspicious that CPSA can explain all server events without invoking the client. In the last event of the server-init strand, the server-init leaks the verifier to the adversary. The dashed arrow indicates that the adversary is able to use the leaked verifier, together with their knowledge of b (since u is publicly known), to satisfy the server strand's final event, and complete the protocol. This shape indicates an attack where the adversary impersonates the client to the server, when the adversary learns the verifier and b.

The situation is more subtle. The adversary is able to impersonate the client only if they know both v and b, as an adversary might learn if the adversary comprised the server. Initially, in our model of SRP-3, we did not require that b and u be distinct, only that they be uniquely generated. CPSA found the impersonation attack in part because CPSA deduced that the adversary could learn b if $b = u$, since SRP-3 reveals u. Subsequently, when we added an additional assumption that $b \neq u$, CPSA discovered only the expected shapes. This fact validates the assertion that SRP is secure from an adversary directly using the verifier to authenticate as a client without access to internal values of the server.

6 A Malicious Server Attack Against SRP

Our analysis in Sect. 5 assumes that legitimate participants of SRP-3 are honest, meaning they will execute the protocol faithfully. In this section, we explore an attack on SRP-3 in which the server is compromised. For example, an adversary might corrupt the server to run a malicious process. In this attack, the malicious server authenticates to itself, pretending to be a particular client, without the client's involvement. A possible goal of this attack might be for the malicious server to escalate its privileges to those of the client, which might be higher than those of the server. For example, a company might have a high-power, low-trust offline computing server used by individuals with sensitive access elsewhere in the network.

To analyze this attack, we define a malicious server role, which we call *malserver* (see Fig. 14). We provide to malserver only the information that an honest server would have access to by observing the state initialized by a server-init role. Consequently, malserver must compute the key using the same method as carried out by an honest server. Malserver also acts like a client, initiating the protocol and sending messages consistent with those from the client role. Figure 14 also defines an associated skeleton, which enables CPSA to compute a strand of the malserver role.

Figure 7 shows the first of two shapes produced by CPSA from the malserver skeleton. As for honest participants, CPSA also produced a second shape that shows the protocol can be started and completed with two different honest server roles on the same machine. Figure 7 shows the malserver initiating the protocol by sending the client's name and proceeding to interact with the server as though it were the client, all the way through to the key verification messages. For executions with a legitimate client, CPSA adds client-init and server-init strands, as a result of the setup phase in which a client sends name, salt, and verifier to the server. Here, however, there is no client strand. The server sends the final black node on its strand only after the server verifies the hash provided by the malserver strand, indicating that the server believes it is communicating with the specified client.

The attack is possible because the malserver role is operating on the server it is attacking (the server and malserver variables are equal) and has access to the server's internal values, as we discuss in the analysis of the leaked verifiers.

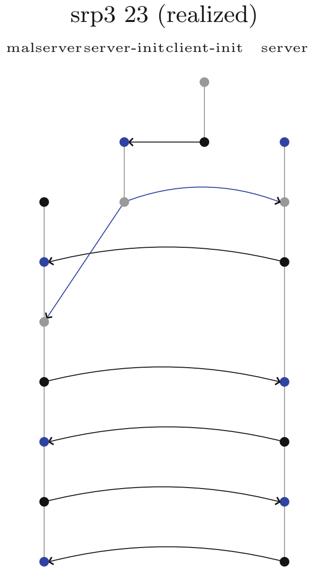

Fig. 7. Shape showing an execution of SRP-3 from the perspective of a malicious server impersonating the client. It is notable that CPSA can explain all events of the malicious server strand, simply by the malicious server knowing the state of the honest server (without the malicious server knowing the client's password). This graphical output from CPSA reveals that a malicious server can impersonate the client to itself (the server), thereby potentially inheriting the client's higher privileges.

Even though this attack is not a part of the Dolev-Yao model that CPSA uses, by creating a special malserver role outside of the normal protocol roles, we were able to coax CPSA to explore the attack. This approach is similar to work by Basin and Cremers [4,5].

7 Discussion

We briefly discuss two limitations of our work: one arising from our modeling of the problematic expression $v + g^b$ as encryption, the other arising from our choice of CPSA's point of view (see Sect. 2.2). We also comment briefly on our experiences using CPSA.

Modeling the problematic expression as encryption enabled CPSA to carry out its work. A consequence of this crucial decision, however, is that we analyzed a slight variation of SRP-3 that might be stronger than SRP-3. By abstracting these algebraic operations as strong encryption, our analysis cannot find possible "algebraic attacks" that might take advantage of detailed algebraic relationships. We are not aware of any such attacks on SRP-3 and do not suspect that they exist, but we cannot exclude their possible existence. The consequences of this crucial modeling decision are similar to those from the common practice of modeling a particular encryption function as a strong encryption function, which excludes the possibility of finding attacks that exploit possible weaknesses in the particular encryption function.

CPSA exhaustively explores possible executions of a protocol from a specified point of view and set of assumptions. Such analysis holds only when those assumptions are satisfied for that point of view. For example, initially, CPSA did not find the malicious server attack described in Sect. 6. CPSA did not find this attack because the adversary requires access to variables v and b, that are not available through the messages exchanged and the assumptions of the model. We were able to show that SRP-3 does not leak those values. Similarly, initially, CPSA could not verify SRP-3's property that access to the state variable v by the adversary would not allow the adversary to impersonate a client directly. To verify that property would require a model that made v available to the adversary.

Subsequently, we explored two models to investigate possible impersonation attacks. One model gave the adversary v; the other model gave the adversary v and b. With these models, CPSA showed that the adversary can impersonate the client if they know v and b, but not if they know only v (see Sect. 5).

Different assumptions and points of view can influence analyses. All formal methods tools explore properties only within a specified scope and do not find attacks outside that scope. Although CPSA did not initially discover the malicious server attack, we were able to enlarge CPSA's scope of search to find it. It is possible, however, that there might be additional attacks outside our scope of search.

During our analysis of SRP-3, the graphical outputs of CPSA helped us gain insights into the properties of the protocol. Nevertheless, using CPSA effectively was challenging. It required learning a new complex language, gaining experience interpreting shapes, devising techniques to model algebraic expressions that cannot be expressed directly in CPSA, and exploring ways to expand CPSA's point of view. Embodied as a virtual machine, our virtual protocol lab avoids the need for users to carry out complex installation procedures for each tool.

We found the following existing techniques useful. (1) Service roles (e.g., client-init, server-init) permitted us to share state between protocol participants (e.g., server, client) and, more generally, to model aspects of protocols that do not directly involve communications among participants. (2) We modeled certain algebraic expressions as basic cryptographic operations such as encryption or hashing. (3) Defining additional protocol participants (e.g., listeners, malicious server) enabled us to explore additional properties of SRP-3 and to expand the capabilities of the Dolev-Yao adversary. We are sharing theses and other lessons in our lab's educational materials.

8 Conclusion

Using CPSA, we formally analyzed the SRP-3 protocol in the Dolev-Yao network intruder model and found it free of major structural weakness. We did find a weakness that a malicious server can fake an authentication session with a client without the client's participation, which might lead to an escalation of privilege attack.

Limitations of our analysis stem in part from our cryptographic modeling. CPSA will not find attacks that exploit weak cryptography, and our use of CPSA will not find any algebraic attacks. As all tool users must, we trust the correctness of CPSA and its execution. Our results do not speak to a variety of other potential issues, including possible implementation and configuration errors when using SRP-3, inappropriate applications of it, and side-channel attacks.

Open problems include formal analysis of other PAKE protocols [30], including the recent OPAQUE protocol [27,33,34], which, unlike SRP, tries to resist precomputation attacks by not revealing the salt values used by the server. OPAQUE is the most promising new protocol possibly to replace SRP. Because quantum computers can compute discrete logarithms in polynomial time, it would be useful to study and develop post-quantum PAKE protocols [18] that can resist quantum attack.

We hope that our work, as facilitated by the virtual protocol analysis lab created at UMBC, will help raise the expectation of due diligence to include formal analysis when designing, standardizing, adopting, and evaluating cryptographic protocols.

Acknowledgments. We appreciate the helpful comments from Akshita Gorti and the reviewers. Thanks also to John Ramsdell (MITRE) and other participants at the Protocol eXchange for fruitful interactions. This research was supported in part by the U.S. Department of Defense under CySP Capacity grants H98230-17-1-0387, H98230-18-1-0321, and H98230-19-1-0308. Sherman, Golaszewski, Wnuk-Fink, Bonyadi, and the UMBC Cyber Defense Lab were supported also in part by the National Science Foundation under SFS grants DGE-1241576, 1753681, and 1819521.

To appear in Festschrift in Honour of Professor Andre Scedrov, Vivek Nigam, Editor, LNCS, Springer (June 11, 2020).

A CPSA Sourcecode

We list critical snippets of CPSA sourcecode that we used to model SRP-3 and carry out our analysis. A complete electronic version is available from our public GitHub repository [46].

```
(defprotocol srp3 diffie-hellman

  (defrole client-init
    (vars (s text) (x rndx) (client server name))
    (trace
     (init (cat "Client state" s x client server))
     (send (enc "Enroll" s (exp (gen) x) client (ltk client server))))
    (uniq-gen s x))

  (defrole server-init
    (vars (s text) (v mesg) (client server name))
    (trace
     (recv (enc "Enroll" s v client (ltk client server)))
     (init (cat "Server record" s v client server))))

  (defrole client
    (vars (client server name) (a rndx) (b u x expt) (s text))
    (trace
     (send client)
     (recv s)
     (obsv (cat "Client state" s x client server))
     (send (exp (gen) a))
     (recv (cat (enc (exp (gen) b) (exp (gen) x)) u))
     (send (hash (exp (gen) a)
           (enc (exp (gen) b) (exp (gen) x)) u
           (hash (exp (gen) (mul b a)) (exp (gen) (mul b u x)))))
     (recv (hash (exp (gen) a)
           (hash (exp (gen) a)
                 (enc (exp (gen) b) (exp (gen) x)) u
                 (hash (exp (gen) (mul b a)) (exp (gen) (mul b u x))))
           (hash (exp (gen) (mul b a)) (exp (gen) (mul b u x))))))
    (uniq-gen a))

  (defrole server
    (vars (client server name) (a expt) (b u rndx) (s text) (v base))
    (trace
     (recv client) ; Server receives Client's name
     (obsv (cat "Server record" s v client server))
     (send s)
     (recv (exp (gen) a))
     (send (cat (enc (exp (gen) b) v) u))
     (recv (hash (exp (gen) a)
           (enc (exp (gen) b) v) u
           (hash (exp (gen) (mul a b)) (exp v (mul u b)))))
     (send (hash (exp (gen) a)
           (hash (exp (gen) a)
                 (enc (exp (gen) b) v) u
                 (hash (exp (gen) (mul a b)) (exp v (mul u b))))
           (hash (exp (gen) (mul a b)) (exp v (mul u b))))))
    (uniq-gen u b))
)
```

Fig. 8. Modeling of SRP-3 in CPSA. We define four roles: client-init, server-init, client, and server. The client-init and server-init roles are service roles that initialize common values between the client and server roles.

```
(defrule at-most-one-server-init-per-client
  (forall ((z0 z1 strd) (client server name))
          (implies
           (and (p "server-init" z0 1)
                (p "server-init" z1 1)
                (p "server-init" "client" z0 client)
                (p "server-init" "client" z1 client)
                (p "server-init" "server" z0 server)
                (p "server-init" "server" z1 server))
           (= z0 z1))
)
```

Fig. 9. Rule added to SRP-3 to prevent CPSA from instantiating an unlimited number of server-init roles, which would prevent CPSA from terminating.

```
(defskeleton srp3
  (vars (client server name))
  (defstrand client 7 (server server) (client client))
  (non-orig (ltk client server)))
```

Fig. 10. Client skeleton of SRP-3, which provides CPSA a starting point for analyzing SRP-3 from the client's perspective.

```
(defskeleton srp3
  (vars (client server name))
  (defstrand server 7 (server server) (client client))
  (non-orig (ltk client server)))
```

Fig. 11. Server skeleton of SRP-3, which provides CPSA a starting point for analyzing SRP-3 from the server's perspective.

```
(defskeleton srp3
  (vars (client server name))
  (defstrand client 7 (server server) (client client))
  (deflistener x)
  (non-orig (ltk client server)))
```

Fig. 12. Client skeleton of SRP-3 with listener for the value x, which provides CPSA a starting point for analyzing SRP-3 from the client's perspective. The listener role helps CPSA determine whether an execution of SRP-3 can leak the value x.

```
(defskeleton srp3
  (vars (client server name))
  (defstrand server 7 (server server) (client client))
  (deflistener v)
  (non-orig (ltk client server)))
```

Fig. 13. Server skeleton of SRP-3 with listener for the value v, which provides CPSA a starting point for analyzing SRP-3 from the server's perspective. The listener role helps CPSA determine whether an execution of SRP-3 can leak the value v.

```
(defrole malserver
    (vars (client server name) (a rndx) (b u expt) (s text) (v base))
    (trace
     (send client)
     (recv s)
     (obsv (cat "Server record" s v client server))
     (send (exp (gen) a))
     (recv (cat (enc (exp (gen) b) v) u))
     (send (hash (exp (gen) a)
                 (enc (exp (gen) b) v) u
                 (hash (exp (gen) (mul a b)) (exp v (mul u b)))))
     (recv (hash (exp (gen) a)
                 (hash (exp (gen) a)
                       (enc (exp (gen) b) v) u
                       (hash (exp (gen) (mul a b)) (exp v (mul u b))))
                (hash (exp (gen) (mul a b)) (exp v (mul u b))))))
    (uniq-gen a)
  )

  (defskeleton srp3
    (vars (client server name))
    (defstrand malserver 7 (server server) (client client))
    (non-orig (ltk client server)))
```

Fig. 14. Modeling a malicious server in CPSA. We define the malserver role to behave like a client while having access to the legitimate server's initialized variables. The associated skeleton provides CPSA a starting point for analyzing the malicious server attack from the perspective of the malicious server.

References

1. Adrian, D., et al.: Imperfect forward secrecy: how Diffie-Hellman fails in practice. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, CCS 2015, pp. 5–17. ACM, New York (2015). https://doi.org/10.1145/2810103.2813707
2. Arapinis, M., et al.: New privacy issues in mobile telephony: fix and verification. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS 2012, pp. 205–216. Association for Computing Machinery, New York (2012). https://doi.org/10.1145/2382196.2382221
3. Bartzia, E.-I., Strub, P.-Y.: A formal library for elliptic curves in the Coq proof assistant. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 77–92. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08970-6_6
4. Basin, D., Cremers, C.: Modeling and analyzing security in the presence of compromising adversaries. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 340–356. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15497-3_21
5. Basin, D., Cremers, C.: Know your enemy: compromising adversaries in protocol analysis. ACM Trans. Inf. Syst. Secur. **17**(2) (2014). https://doi.org/10.1145/2658996

6. Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: IEEE Symposium on Research in Security and Privacy, pp. 72–84, May 1992
7. Blake-Wilson, S., Menezes, A.: Authenticated Diffie-Hellman key agreement protocols. In: Proceedings of the Selected Areas in Cryptography, SAC 1998, pp. 339–361. Springer, Heidelberg (1999). http://dl.acm.org/citation.cfm?id=646554.694440
8. Blanchet, B., Smyth, B., Cheval, V.: Proverif 1.90: automatic cryptographic protocol verifier, user manual and tutorial (2015). http://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf
9. Böhl, F., Unruh, D.: Symbolic universal composability. J. Comput. Secur. **24**(1), 1–38 (2016)
10. Boneh, D., Shoup, V.: A graduate course in applied cryptography version 0.5, January 2020. https://crypto.stanford.edu/~dabo/cryptobook/BonehShoup_0_5.pdf
11. Browning, S.: Cryptol, a DSL for cryptographic algorithms. In: ACM SIGPLAN Commercial Users of Functional Programming, p. 1. ACM (2010)
12. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, FOCS 2001, p. 136. IEEE Computer Society, USA (2001)
13. Canetti, R., Stoughton, A., Varia, M.: EasyUC: using EasyCrypt to mechanize proofs of universally composable security. In: 2019 IEEE 32nd Computer Security Foundations Symposium (CSF), pp. 167–183 (2019)
14. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
15. Corin, R., Doumen, J., Etalle, S.: Analysing password protocol security against off-line dictionary attacks. Electron. Notes Theoret. Comput. Sci. **121**, 47–63 (2005)
16. Delaune, S., Kremer, S., Pereira, O.: Simulation based security in the applied pi calculus. In: Kannan, R., Kumar, K.N. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. Leibniz International Proceedings in Informatics (LIPIcs), vol. 4, pp. 169–180. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2009). http://drops.dagstuhl.de/opus/volltexte/2009/2316
17. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theor. **22**(6), 644–654 (2006). https://doi.org/10.1023/A:1008302122286
18. Ding, J., Alsayigh, S., Lancrenon, J., RV, S., Snook, M.: Provably secure password authenticated key exchange based on RLWE for the post-quantum world. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 183–204. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-52153-4_11
19. Doghmi, S., Guttman, J., Thayer, F.J.: Skeletons and the shapes of bundles. In: Proceedings of the 7th International Workshop on Issues in the Theory of Security, pp. 24–25 (2006)
20. Doghmi, S.F., Guttman, J.D., Thayer, F.J.: Searching for shapes in cryptographic protocols. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 523–537. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_41
21. Dolev, D., Yao, A.C.: On the security of public key protocols. In: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science, SFCS 1981, pp. 350–357. IEEE Computer Society, Washington, DC (1981). https://doi.org/10.1109/SFCS.1981.32

22. Dreier, J., Duménil, C., Kremer, S., Sasse, R.: Beyond subterm-convergent equational theories in automated verification of stateful protocols. In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 117–140. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_6. https://hal.inria.fr/hal-01430490/document
23. Escobar, S., Meadows, C., Meseguer, J.: A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties. Theoret. Comput. Sci. **367**(1–2), 162–202 (2006)
24. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA: cryptographic protocol analysis modulo equational properties. In: Aldini, A., Barthe, G., Gorrieri, R. (eds.) FOSAD 2007-2009. LNCS, vol. 5705, pp. 1–50. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03829-7_1
25. Escobar, S., Meadows, C., Meseguer, J.: Maude-NPA, Version 3.0, April 2017
26. Fabrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: why is a security protocol correct? In: Proceedings of the 1998 IEEE Symposium on Security and Privacy (Cat. No. 98CB36186), pp. 160–171, May 1998. https://doi.org/10.1109/SECPRI.1998.674832
27. Green, M.: Let's talk about PAKE, October 2018. https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/
28. Green, M.: Should you use SRP? October 2018. https://blog.cryptographyengineering.com/should-you-use-srp/
29. Guttman, J.D., Liskov, M.D., Ramsdell, J.D., Rowe, P.D.: The Cryptographic Protocol Shapes Analyzer (CPSA). https://github.com/mitre/cpsa
30. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**, 1–48 (2018)
31. Hao, F., Shahandashti, S.F.: The SPEKE protocol revisited. In: Chen, L., Mitchell, C. (eds.) SSR 2014. LNCS, vol. 8893, pp. 26–38. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-14054-4_2
32. Jablon, D.P.: Strong password-only authenticated key exchange. ACM Comput. Commun. Rev. **26**(5), 5–26 (1996)
33. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163 (2018). https://eprint.iacr.org/
34. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10822, pp. 456–486. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78372-7_15
35. Lanus, E., Zieglar, E.: Analysis of a forced-latency defense against man-in-the-middle attacks. J. Inf. Warfare **16**(2), 66–78 (2017). https://www.jstor.org/stable/26502758
36. Liskov, M.D., Ramsdell, J.D., Guttman, J.D., Rowe, P.D.: The Cryptographic Protocol Shapes Analyzer: A Manual. The MITRE Corporation (2016)
37. Liskov, M.D., Rowe, P.D., Thayer, F.J.: Completeness of CPSA. Technical report MTR110479, The MITRE Corporation (2011)
38. Lowe, G.: An attack on the Needham-Schroeder public-key authentication protocol. Inf. Process. Lett. **56**(3), 131–133 (1995). http://www.sciencedirect.com/science/article/pii/0020019095001442
39. Maurer, U.M., Wolf, S.: The Diffie-Hellman protocol. Des. Codes Cryptography **19**(2–3), 147–171 (2000). https://doi.org/10.1023/A:1008302122286
40. Meadows, C.: NRL protocol analyzer. J. Comput. Secur. **1**(1) (1992)

41. 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
42. Paulson, L.C.: Relations between secrets: two formal analyses of the Yahalom protocol. J. Comput. Secur. **9**(3), 197–216 (2001)
43. Ramsdell, J.D., Guttman, J.D., Millen, J.K., O'Hanlon, B.: An analysis of the CAVES attestation protocol using CPSA. arXiv preprint arXiv:1207.0418 (2012)
44. Ryan, P.Y.A., Schneider, S.A.: An attack on a recursive authentication protocol. A cautionary tale. Inf. Process. Lett. **65**(1), 7–10 (1998). https://doi.org/10.1016/S0020-0190(97)00180-4
45. Schmidt, B., Meier, S., Cremers, C., Basin, D.: Automated analysis of Diffie-Hellman protocols and advanced security properties. In: 2012 IEEE 25th Computer Security Foundations Symposium, pp. 78–94, June 2012
46. Sherman, A.T., et al.: PAL GitHub repository, June 2020. https://github.com/egolaszewski/UMBC-Protocol-Analysis-Lab
47. Steiner, J.G., Neuman, B.C., Schiller, J.I.: Kerberos: an authentication service for open network systems. In: Proceedings Winter USENIX Conference, pp. 191–202 (1988)
48. Tang, Q., Mitchell, C.J.: On the security of some password-based key agreement schemes. In: Hao, Y., et al. (eds.) CIS 2005. LNCS (LNAI), vol. 3802, pp. 149–154. Springer, Heidelberg (2005). https://doi.org/10.1007/11596981_22
49. Taylor, D., Wu, T., Mavrogiannopoulos, N., Perrin, T.: RFC 5054, Using the secure remote password (SRP) protocol for TLS authentication. Technical report, RFC Editor, November 2007. https://doi.org/10.17487/rfc5054
50. Wu, T.: RFC 2944, Telnet Authentication: SRP. Technical report, RFC Editor, September 2000. https://doi.org/10.17487/rfc2944
51. Wu, T.: The secure remote password protocol. In: Proceedings of the Internet Society on Network and Distributed System Security (1998)
52. Wu, T.: The SRP Authentication and Key Exchange System, RFC 2945, September 2000
53. Wu, T.: SRP-6: Improvements and Refinements to the Secure Remote Password Protocol, October 2002
54. Zhang, M.: Analysis of the SPEKE password-authenticated key exchange protocol. IEEE Commun. Lett. **8**(1), 63–65 (2004). https://doi.org/10.1109/LCOMM.2003.822506