This work was written as part of one of the author's official duties as an Employee of the United States Government and is therefore a work of the United States Government. In accordance with 17 U.S.C. 105, no copyright protection is available for such works under U.S. Law.

Public Domain Mark 1.0

<https://creativecommons.org/publicdomain/mark/1.0/>

Access to this work was provided by the University of Maryland, Baltimore County (UMBC) ScholarWorks@UMBC digital repository on the Maryland Shared Open Access (MD-SOAR) platform.

Please provide feedback

Please support the ScholarWorks@UMBC repository by emailing scholarworks-group@umbc.edu and telling us what having access to this work means to you and why it's important to you. Thank you.

Searching for Selfie in TLS 1.3 with the Cryptographic Protocol Shapes Analyzer

Prajna Bhandary¹, Edward Ziegler^{2(✉)}, and Charles Nicholas¹

¹ University of Maryland, Baltimore County (UMBC), Baltimore, MD 21250, USA
{prajna1,nicholas}@umbc.edu

² National Security Agency, Fort George G. Meade, MD 20755, USA
evziegl@uwe.nsa.gov

Abstract. TLS 1.3 was developed in conjunction with several formal analyses and proofs of its security properties. However, in 2019, researchers Drucker and Gueron discovered a reflection attack, they named Selfie, against the pre-shared key (PSK) mode of authentication used by TLS 1.3 by identifying a gap in the proofs. They realized that the proofs ignored the case of external PSKs. They demonstrated that if the PSK was not associated with a particular client and server pairing, such as a single PSK between a pair of hosts which could use the key as either a client or server, implicit authentication implied by the use of the PSK would fail in a reflection attack. The proofs and tools used did not account for this, so we set out to determine if it was possible to identify this attack with the Cryptographic Protocol Shapes Analyzer (CPSA). Using CPSA, which attempts to enumerate all equivalence classes of a protocol's executions, we were able to uncover the attack and verify two proposed mitigations. We were also able to identify a previously discovered impersonation attack against the use of post handshake authentication in scenarios where a PSK is used as a network key.

Keywords: Cryptographic protocols · Cryptography · Cryptographic Protocol Shapes Analyzer (CPSA) · Cybersecurity · Formal methods · Transport Layer Security · Selfie attack · Protocol analysis

1 Introduction

Version 1.3 of the Transport Layer Security (TLS 1.3) protocol is an important protocol for the security of the Internet. The majority of the web based traffic is now encrypted using some version of TLS. In January of 2021, Google, in their Transparency Report, noted that 89% of pages loaded by Chrome were served using HTTPS. Applications such as email, instant messaging, voice over IP, VPN, and others also use TLS to provide authentication, confidentiality, and integrity. As a significant amount of traffic on the Internet relies on TLS for its security, it is important that TLS is vulnerable to few, if any forms of attack.

Earlier versions of TLS have had various security weaknesses, such as Logjam [1], Triple Handshake [6], SMACK [4], Lucky13 [3] and others. Therefore, it was

important to the Internet Engineering Task Force (IETF) that TLS 1.3 not be vulnerable to those and other attacks that may exploit the protocol to violate the security goals. To achieve assurance that TLS 1.3 satisfied its security goals, the IETF undertook an unprecedented effort to provide formal analysis and security proofs of the design in studies such as [5, 7, 8, 10, 13, 14, 17, 18, 20]. In spite of this effort, Drucker and Gueron [11] discovered a reflection attack, which they named the Selfie attack, against TLS 1.3’s Pre Shared Key (PSK) based mutual authentication.

The existence of the Selfie attack does not take anything away from the formal analysis that was performed. The formal verification effort was successful in demonstrating that many of the weaknesses required by the known attacks were eliminated. As an example, the formal verification effort was able to demonstrate that attacks such as Logjam and Triple Handshake were not possible, even if TLS 1.3 was run in parallel with TLS 1.2. The tools that were used in the analyses, such as Tamarin, ProVerif and CryptoVerif, involve developing models of the protocols and proving theorems within the model. The reason that the Selfie attack was not discovered by the tools in the analysis was because the analyses weren’t comprehensive enough to cover a case where the Client uses TLS 1.3 with an external PSK and then proceeds to talk to itself. These tools assumed that a PSK can be shared only between a Client and a Server. They also assumed that if a mutual authentication could be established between a Client and a Server or between a Client and a Client, then both are valid [5].

The Cryptographic Protocol Shapes Analyze (CPSA) [19] operates differently than other protocol verification tools. Instead of specifying the properties that one wishes to verify in the model, CPSA takes a protocol definition and a partial description of an execution, built within a particular formal model, and attempts to produce descriptions of all possible executions of the protocol that complete the partial description in the presence of a powerful network adversary. From the descriptions of the executions, referred to as shapes, it is possible to identify the properties that the protocol satisfies.

We were interested in determining whether or not the approach taken by CPSA would have identified the Selfie attack. Using CPSA, we modeled the TLS 1.3 PSK authentication protocol and analyzed the shapes that CPSA produced and were able to identify the attack. We were also able to verify that the proposed mitigations of a unique key between a client and server or the use of identifiers would prevent the Selfie attack. We describe our approach in the following sections.

2 Background

We briefly review CPSA, the TLS 1.3 Pre Shared Key authentication protocol, and the Selfie attack discovered by Drucker and Gueron.

2.1 Cryptographic Protocol Shapes Analyser

The Cryptographic Protocol Shapes Analyzer (CPSA) [15, 19] is an open-source tool for automated formal analysis of cryptographic protocols. CPSA takes as input a model of a cryptographic protocol and a description of a partial execution with assumptions, and generates a set of minimal, essentially different descriptions of executions of the protocol that complete the partial execution, consistent with the assumptions. The descriptions of completed executions are referred to as shapes. When some property holds in all shapes generated, it is a property guaranteed by the protocol.

CPSA is based on strand space theory [9, 12] in which events are organized into partially-ordered graphs. In strand space theory, events are transmissions or receptions of messages. Strands are sequences of events that capture a local view of a participant in the network. CPSA also has state events that consist of initializing, observing and transitioning between states. Protocols are defined as a set of legitimate participant roles that serve as a template for strands consistent with the protocol.

The underlying execution model of CPSA is the Bundle, where every reception is explained directly by a previous transmission of that exact message. A bundle of a particular protocol is a set of strands where 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, i.e. partially-ordered sets of strands that represent only regular behavior, along with origination assumptions about the secrecy and/or freshness of particular values, such as keys unknown to the adversary or nonces, freshly chosen and therefore assumed unique. Skeletons in which all messages can be explained by some combination of legitimate or adversary behavior consistent with the secrecy and freshness assumptions are referred to as realized. A shape is the most general form of a set of realized skeletons. Non-realized skeletons may represent partial descriptions of actual executions, or may represent a set of conditions inconsistent with any actual execution [19].

The cpsagraph tool creates visualizations of skeletons as graphs in which events are shown as circles, black for transmissions and blue for receptions, in connected columns where each column represents a strand. Events within a strand are ordered from top to bottom, with the earliest event at the top. Arrows between strands indicate necessary orderings (other than orderings within strands, or those that can be inferred transitively). An arrow from event P to Q denotes that for Q to take place, it is necessary that event P take place first. A solid black arrow indicates that the message that was transmitted was exactly the same as the one received. Black dashed arrows indicate that the message was altered by some form of adversary behavior. CPSA represents states as gray circles. State is assumed to not be directly observable by the adversary. Observation of state is indicated by a blue arrow originating from a state event,

represented as a gray circle. See Fig. 3 in Sect. 3.1 for an example of such a visualization.

The cpsagraph tool also outputs the skeleton information associated with the graph that has been drawn. This information is in the form of a defskeleton. A defskeleton contains all the information that is necessary to describe an execution, whether it is partial, having unrealized message receptions, or realized, a complete execution of the protocol. As such, it contains a complete list of the variables used in the skeleton, the strands which identify the protocol role, the number of messages sent and received as well as state interactions, and the variable assignments for the strand in a defstrand, the assumptions made on the variables, the action taken by CPSA to create the current skeleton, the ordering of the nodes in the skeleton, and information concerning whether or not the analysis is complete. For brevity, we have only shown the partial CPSA output of a defskeleton that includes the strands in this paper. This allows one to see whether or not the strands agree on values associated with the variables. For example, in Fig. 3, both the client and the server agree on the values of the variables, but in Fig. 8, the client and server disagree on variables *a* and *b* which represent their views of the client and server.

2.2 TLS 1.3 Pre Shared Key Authentication

TLS 1.3 [21] offers a variety of options for establishing a secure connection. Several of the options support authentication of one or more of the parties through public key certificates. Another option uses a Pre Shared Key (PSK), established externally or derived from the secret value from a previous connection. Authentication when using a PSK is predicated on the assumption that the party receiving a message authenticated with the PSK knows that the message was sent by a party that also knows the PSK. TLS 1.3 allows this use of implicit authentication to save bandwidth and latency over certificate verification and to support 0-RTT mode.

As TLS 1.3 can be used to support networks of communicating peers, where every node acts as both a client and a server, it is possible to use a single PSK for authentication as both the client and the server. It is under this scenario that Drucker and Gueron identified their attack.

2.3 Selfie Attack

The Selfie attack discovered by Drucker and Gueron [11] is a reflection attack against the use of PSK authentication in TLS. The attack relies on the assumption that if two parties share a symmetric key, the receiving party knows that the message, if it passes verification with the key, was sent by a party that knows that key. As TLS permits this implicit authentication to save bandwidth and

latency in support of 0-RTT mode, it opens up the possibility of this reflection attack against a party when parties within the network can act as both client and server using the same key. In this case, the sender, when under attack, could also be the receiver of the authentic message itself.

Figure 1 taken from [11] illustrates the attack. In this case Alice, acting as the client, wishes to communicate with Bob, acting as the server. The adversary, Eve, is able to intercept and reflect the messages back to Alice. The communication takes place as follows:

- Alice sends the ClientHello message with a pre shared key extension intended for Bob.
- Eve intercepts the message and reflects it back to Alice, pretending (implicitly) to be Bob.
- Alice receives the message containing the PSK extension she uses with Bob.
- Alice, acting as server, replies (presumably to Bob) with ServerHello and ServerFinished messages which authenticate that Alice knows the PSK.
- Eve captures these messages and echoes them back to Alice.
- Alice, acting as client, authenticates the ServerHello and ServerFinish messages as being created with the PSK she shares with Bob, believing that she has authenticated Bob when she has actually opened a Selfie session with herself.
- Alice, as client, then completes the connection by authenticating herself through the ClientFinished Message.
- Eve intercepts and reflects the ClientFinished message back to Alice to complete Alice’s server run of the protocol.

At the end of the run, Alice has established two sessions with herself acting as both client and server which she believes exist with Bob, in violation of the TLS claimed property of [21] [Appendix E]: “Peer authentication: The client’s view of the peer identity should reflect the server’s identity. If the client is authenticated, the server’s view of the peer identity should match the client’s identity.”

Fig. 1. The Selfie Attack [11]. Eve tricks Alice into believing she is talking to Bob while she is actually talking with herself.

3 Modeling TLS 1.3 PSK Authentication

Using CPSA, we analyze the TLS 1.3 PSK authentication when the pre-shared keys are distributed out of band in the Dolev-Yao intruder model. We develop several models of the PSK authentication and analyze them to explore the selfie attack and to verify proposed fixes in [11].

3.1 Models of the TLS 1.3 PSK Authentication

We built two models of the TLS PSK authentication. In the first model, we modeled the authentication directly as specified. In this initial model, there are no indications of who is acting as client or server. The authentication is implicit. In the second model, we made the direction of the key used explicit so that we could identify which party was acting as client and which was acting as server. We describe the models and results below.

Initial Model. The TLS 1.3 handshake, when using PSK authentication, consists of only three messages, shown in Fig. 2 taken from Sect. 2.2 of [21].

The initial message from the client is sent unencrypted. Additionally, many of the fields in the ClientHello are unlikely to change in subsequent handshakes, with the exception of the random. Our model simplifies the ClientHello to the random, `n1` in our model, as that is the only component of the message that will distinguish two ClientHello messages from the same client to the same server.

Fig. 2. Message flow for PSK handshake

The `pre_shared_key` is an identifier of the PSK to use for the connection. We model this simply as a unique index that is tied to the key through the use of state, observable to both the client and the server. As the `key_share` is an optional set of parameters that are used as a fallback to the full handshake should the PSK not be accepted, we chose not to include those additional messages.

We made similar choices for the **ServerHello**, `pre_shared_key` and `key_share` components of the server’s response. We made a simplifying assumption that the Client did not send an extension request in the **ClientHello**, so we can safely ignore the **EncryptedExtensions** as they should only be sent if the client made a request. Finally, the **Finished** message is the first attempt at authenticating the server to the client. It consists of a hash of all the previous message components with the `finished_key`. The `finished_key` is created from a hash of “finished” and the base key. For this modeling, we simplified the TLS keying scheme to a single key that was produced as a hash of the PSK and the generated random nonces from each party. The key generated will be unique as long as at least one party generates a fresh nonce and the extra complication of generating additional keys is unnecessary to model the authentication.

The final message is the client’s attempt to authenticate to the server by creating a **finished** message of its own consisting of the all the messages that it has received. The distinction between the client’s **finished** message and the server’s is that the client’s message includes the server’s **finished** message in the hash.

In addition to modeling the client and server roles in the protocol, we model an additional role that acts to distribute the key to the parties. This role, **key-Placement**, initializes a state with the index and the key that both the client and server can observe, representing the pre-placement of keys at both the client and the server. See Fig. 15 in the Appendix for the model.

Fig. 3. Shape showing an execution of the protocol from the client’s perspective. As identification of the parties is implicit (neither strand indicates the parties, as can be seen in the `defstrand` statements), the Selfie attack is included in this equivalence class represented by the shape. (Color figure online)

The Selfie attack is described as a reflection against the client. For our analysis, we defined a partial execution of the protocol consisting of a single client with the assumption that the PSK was known only to the client and server. CPSA produced a single shape for the execution, shown in Fig. 3. The shape indicates that all executions of the protocol involve only parties that share the PSK. Both the client and the server observe the same PSK, indicated by the solid blue lines. Additionally, we can see that the messages were received from the expected parties in the correct order and unaltered, as indicated by the solid black lines showing the ordering. If an adversary were able to alter the messages, a dashed line would have indicated that the message had been altered from the point of origination to the destination. As there are no indications of the parties identities in the model, the Selfie attack is also present in the shape as a single party could be acting as both client and server.

Fig. 4. First shape produced by the initial model from the server’s perspective. Identification is implicit. The Selfie attack is represented in the equivalence class represented by the shape.

For completeness, we also analyzed the model from the server’s perspective. This analysis produced two shapes, Figs. 4 and 5, both indicating that it would only complete with a client that shared the PSK. The second shape in Fig. 5 shows an additional start of the client with the same PSK, but a different choice of nonce. It indicates that the adversary could mix messages initially from the clients, but only one can complete. As with the view from the client, the server only knows that the party acting as the client is in possession of the PSK and may in fact be the same party acting as both client and server.

Model with Identities. To better visualize which identities are communicating with each other, we modified the keyPlacement role to include the parties sharing the key, as shown in Fig. 18. Instead of a single state being initialized with the key and the index, we initialize two states, each with the same key and index, but with the identities of the parties using the key listed in the order of the client and the server. Two states were necessary, as a party could be acting as either the client or the server with the same key. With this approach, shapes representing exchanges that involve the Selfie attack would be visualized as accessing both

states from the keyPlacement role, with the client accessing one state and the server accessing the other, yet communicating with each other. CPSA generated four shapes, Figs. 6, 7, 8, and 9, from the perspective of the client.

Fig. 5. Second shape produced by the initial model from the server’s perspective. This represents the possibility of additional clients that share the PSK initiating a run of the protocol. The shape indicates that the client on the left may have initiated the contact with the server before the client on the right, but the adversary changed the message from the client on the left to replace the nonce with the nonce of the client on the right. The server completes the run with the client on the right, as indicated in the variable assignments in the defstrand statements. Again, the Selfie attack is also represented in the possible executions.

Fig. 6. Shape illustrating a correct authentication, no Selfie Attack, where the server is assigned the name “a” and the client is assigned the name “b”.

Figures 6 and 7 represent the classes of executions that are free of the Selfie attack. In each case, the client and the server accessed the same state, indicating that they agree on the assignment of values to the client, represented as variable *a*, and the server, represented as variable *b*. In Fig. 6, both parties agree that the client is “b” and the server is “a”. This can be seen in the assignment of the variables, “a” representing the client and “b” representing the server, in each of the `defstrand` statements mapping variables of the displayed strands. Figure 7 shows the case where the key for the opposite direction is used and maps the client to “a” and the server to “b” in both the client and server strands.

Fig. 7. Complement to the shape in Fig. 6 illustrating a correct authentication, no Selfie Attack, where the server is assigned the name “b” and the client is assigned the name “a”.

Fig. 8. Shape illustrating the Selfie attack where the client strand is “b” believing the server it is communicating with is “a”, while the server strand it is actually communicating with is itself, “b”, believing that it is communicating with client “a”.

Fig. 9. Complement to the shape in Fig. 8 illustrating the Selfie attack where the client strand is “a” believing the server it is communicating with is “b”, while the server strand it is actually communicating with is itself, “a”, believing that it is communicating with client “b”.

Figures 8 and 9 represent the classes of executions where the Selfie attack takes place. This is visible in the graph of the shape where the client observes one state and the server observes another. As the only difference between the states is the assignment of server and client, if two interacting strands are observing different states, it indicates that the same party is acting as both the client and the server in the communication. By making the direction a key is being used explicit, we were able to split the set of equivalent executions represented in the original model, shown in Fig. 3, into the various subsets of executions, those that show expected behavior and those that show the Selfie attack.

The models from the server’s perspective also illustrate both expected executions and executions including the Selfie attack. For brevity, those shapes are not included here.

3.2 Modeling the Proposed Fixes to the Selfie Attack

The authors of [11] proposed two solutions to the Selfie attack. The first is the use of server certificates, although the authors point out that the use of certificates with the PSK defeats the purpose of using the PSK. The second approach is to require the use of a unique key between the client and the server. We validate each of the solutions in the following sections by modifying our model that supports directional use of the keys with identities. This approach allows us to visually inspect the graphs to determine if the Selfie attack has been eliminated by noting whether or not the same state is observed by both client and server, as was described in Sect. 3.

PSK with Server Identity. There are two approaches to verifying the identity of a party with a certificate. The first is to include the certificates in the handshake. The second is to request the certificate after establishing a secure connection with the PSK. To model this, we extended the protocol by adding a request for the certificate and the response.

Although the use of a certificate was proposed, we instead chose to model the addition of the `server_name` encrypted extension in the handshake as proposed in [16]. The analysis in the model is similar, but it has the added advantage of not requiring the additional processing necessary with the use of a certificate in an actual implementation. CPSA produced only shapes where the connected client and server were using the same state as shown in Fig. 10 and Fig. 11. As both shapes indicate, the client and server both agree on who is the client and who is the server. The mitigation of including the server’s name in the response does prevent the Selfie attack.

To model the authentication request after establishing a connection, we included a certificate request message and a certificate verify message. We chose to have the server authenticate the client to continue the flow of send and receive, although we could have chosen to have the client authenticate the server. This approach creates a number of shapes that could be problematic if there were multiple parties sharing the same key. An example of one of the problematic shapes is Fig. 12. In this case, the client on the left is accessing the state for the

Fig. 10. Shape produced when a modified handshake is used to convey the server’s name to the client. Both sides are using the same state, indicating that they agree on the values of the client and server. In this case, the client is “b” and the server is “a”.

Fig. 11. The complement to the shape in Fig. 10. In this shape, both the client and server are using the same state, therefore they agree on the values of the client and server. In this case, the client is “a” and the server is “b”.

reverse key of the server, indicative of the Selfie attack. The client on the right completes the authentication as expected, but all parties share the key. There is nothing to prevent the client on the left from continuing after the authentication completes. This is an example of the Selfie attack.

Authentication outside of the handshake does not preclude a Selfie attack, but the results could be much worse. If the same keys were used for several participants, then it would be possible for any participant to impersonate another as outlined in [2]. We added an additional set of keys for a third participant to see if CPSA would identify the attack. CPSA generated a considerable number of shapes, many of which indicated that one party could impersonate another. Figure 13 is an example. In this case, the server strand is “b” and it believes that it is communicating with “b-0”. The client on the left is “a” and it is communicating with the server “b” while the client on the right is “b-0” and is communicating with “a”, the client on the left. The client “a” is impersonating as “b-0” to the server “b”. Because all parties have access to the same key, it is possible to impersonate any party to any other with this approach. If certificates are to be used, they must be used in the handshake to prevent this attack.

Unique PSK per Client/Server Pair. The second solution proposed was the use of a unique key for any client/server pair. We were able to model this by simply creating a fresh key and index for each state initialization in the `keyPlacement` role, representing a unique key for each direction between a pair of parties. From the clients point of view, only one shape is possible. This is shown in Fig. 14. Both the client and server agree on all variables. It is not possible for the client to chose one key and the server to chose the other in this situation, eliminating the Selfie attack.

4 Discussion

We set out to determine if the approach used by CPSA would be effective in identifying the Selfie attack that was missed in the previous formal analysis of TLS 1.3. As the previous analysis built models of the protocol and proved theorems based upon those models, it was necessary to know what properties one hoped existed in the protocol to develop the theorems that one wished to prove. Unfortunately, there were gaps in the earlier analysis as the models either failed to consider the case of external PSKs or left the analysis of the implicit authentication properties of external PSKs for future work. The approach used by CPSA doesn't require knowledge of the properties one wishes to prove. Instead of proving particular theorems about the protocol model, CPSA attempts to enumerate all possible equivalence classes of executions of the protocol that satisfy the assumptions. From the equivalence classes, referred to as shapes, it is then possible to determine what properties hold across all executions. We were looking to see if we could identify the Selfie attack with this approach.

The shapes that CPSA generated with the initial model, Figs. 3, 4 and 5, demonstrated that only parties that share the key were able to interact. That includes the case where a party is interacting with itself. With implicit authentication, it is therefore impossible to know which parties are interacting. If a party that can interact with itself is attempting to interact with another party, then the Selfie attack is possible. The shapes produced by CPSA only show that an adversary, that does not know the PSK, cannot generate or modify the messages. The adversary can reflect the messages back to a party that can act as both client and server, confusing that party as to whom it is connecting. This is the Selfie attack.

However, since one could identify the Selfie attack in the shapes produced by CPSA in the initial model, we modified the model to include the direction a key was being used by adding the client's and the server's identities to the state. We used two states to represent a bidirectional key. The states had identical index and key, but differed in which party was the client and which was the server, with the roles swapped between the two states. This allowed CPSA to subdivide the original equivalence class into four equivalence classes from the client's perspective, shown in Figs. 6, 7, 8, and 9. By adding identities and associating direction with the keys, we were able to make the Selfie attack visually evident, as there were now shapes, Figs. 8 and 9, representing executions of the Selfie attack. This

Fig. 13. Shape with three parties sharing a key and using post handshake authentication to illustrate that any party could authenticate as any other party. In this case, the server is “b” and completes the handshake with the client on the left who is “a”, but authenticates the client on the right who is “b-0”. The client on the right believes they are authenticating to “a”, the client on the left. Client “a” can now impersonate client “b-0” to the server.

authentication taking place with a different strand, as illustrated in Fig. 12. In this case, it would be possible for the server to be communicating with itself while authenticating a different client. Such a situation lends itself to anyone being able to impersonate anyone else in network keyed environments as identi-

Fig. 14. Shape produced when directional keys are used. Only one shape is produced, as only one key can be used between any pair of parties, guaranteeing agreement by the parties on who is the client and who is the server.

fied in [2]. We were able to illustrate this (see Fig. 13) by adding an additional party that was using the same key.

We have validated one of the recommendations for external PSK usage outlined in [16], that keys be used only between client/server pairs. We did not evaluate the other proposed solutions of using external PSK importers to protect against the attack, although the proposal is similar to the mitigations that have been investigated.

5 Conclusions

Using CPSA, we were able to model TLS 1.3 PSK authentication and identify the Selfie attack where other formal tools did not. We were also able to use CPSA to subdivide the shape representing the equivalence class from our original model into shapes representing classes of executions that included the Selfie attack and those that did not. This allowed us to verify that the proposed mitigations were effective. It also allowed us to demonstrate that a post handshake authentication would not be effective at preventing impersonation attacks if multiple hosts shared the PSK.

Although CPSA was able to identify an attack that was missed in the previous formal analysis, nothing is taken away from the formal analysis that was performed before. That formal verification effort was successful in verifying the

elimination of a number of weaknesses that existed in previous versions of TLS. Additionally, there are properties that can be proven in the other tools that cannot be proven in CPSA. The various tools have their strengths and weaknesses, with no tool offering to verify all properties that one may wish. This effort highlights the benefits of using a variety of tools to validate the security properties of a protocol. The approach taken by CPSA may best be used when all properties of a protocol may not be well understood. We would advocate for more formal analysis using a variety of tools to provide greater coverage and assurance in the security properties of cryptographic protocols.

A Appendix

Source code for the CPSA models used in the analysis (Figs. 16, 17, 19, 20, 21 and 22).

```
(defprotocol tls basic

  (defrole keyPlacement
    (vars (index text) (psk skey))
    (trace
      (init (cat index psk)))
      (uniq-orig index)
      (non-orig psk))

  (defrole client
    (vars (index n1 n2 text) (psk skey))
    (trace
      (obsv (cat index psk))
      (send (cat n1 index))
      (recv (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
      )
    )

  (defrole server
    (vars (index n1 n2 text) (psk skey))
    (trace
      (recv (cat n1 index))
      (obsv (cat index psk))
      (send (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
      )
    )
  (comment "Protocol without identities. Shows Selfie attack.")
)
```

Fig. 15. CPSA model of TLS 1.3 PSK authentication

```

(defskeleton tls
  (vars (n1 index text) (psk skey))
  (defstrand client 3 (n1 n1) (index index) (psk psk))
  (uniq-orig n1)
)

```

Fig. 16. Skeleton indicating a partial run of the protocol by the client.

```

(defskeleton tls
  (vars (a b name) (n2 index text) (psk skey))
  (defstrand server 4 (n2 n2) (index index) (psk psk))
  (uniq-orig n2)
)

```

Fig. 17. Skeleton indicating a partial run of the protocol by the server.

```

(defprotocol tls1 basic

  (defrole keyPlacement
    (vars (a b name) (index text) (psk skey))
    (trace
      (init (cat index a b psk)) ;; clientialization of key for parties a and b
      (init (cat index b a psk))) ;; added so CPSA knows the key is bidirectional
    (uniq-orig index)
    (non-orig psk))

  (defrole client
    (vars (a b name) (index n1 n2 text) (psk skey))
    (trace
      (obsv (cat index a b psk))
      (send (cat n1 index))
      (recv (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2))))
    )

  (defrole server
    (vars (a b name) (index n1 n2 text) (psk skey))
    (trace
      (recv (cat n1 index))
      (obsv (cat index a b psk))
      (send (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2))))
    )
  )
  (comment "Protocol with identities added to state. Shows Selfie attack.")
)

```

Fig. 18. CPSA model of TLS 1.3 PSK authentication with identities associated with the key to identify direction of use.


```

(defprotocol tls2 basic

  (defrole keyPlacement
    (vars (a b name) (index text) (psk skey))
    (trace
      (init (cat index a b psk))
      (init (cat index b a psk)))
    (uniq-orig index)
    (non-orig psk)
    (neq (a b)))

  (defrole client
    (vars (a b name) (index n1 n2 text) (psk skey))
    (trace
      (obsv (cat index a b psk))
      (send (cat n1 index)) ;; client hello
      (recv (cat n2 ;; server hello
                (enc b (hash psk n1 n2)) ;; encrypted extension servername
                (hash (hash psk n1 n2) n1 index n2 (enc b (hash psk n1 n2))))) ;; server finish
      (send (hash (hash psk n1 n2) n1 index n2 (enc b (hash psk n1 n2))
                (hash (hash psk n1 n2) n1 index n2 (enc b (hash psk n1 n2))))) ;; client finish
      )
    )

  (defrole server
    (vars (a b name) (index n1 n2 text) (psk skey))
    (trace
      (recv (cat n1 index)) ;; client hello
      (obsv (cat index a b psk))
      (send (cat n2 ;; server hello
                (enc b (hash psk n1 n2)) ;; encrypted extension servername
                (hash (hash psk n1 n2) n1 index n2 (enc b (hash psk n1 n2))))) ;; server finish
      (recv (hash (hash psk n1 n2) n1 index n2 (enc b (hash psk n1 n2))
                (hash (hash psk n1 n2) n1 index n2 (enc b (hash psk n1 n2))))) ;; client finish
      )
    )
  (comment "Protocol with servername extension. No selfie attack.")
)

```

Fig. 19. CPSA model of PSK authentication with server name included in the handshake.

```

(defprotocol tls3 basic

  (defrole keyPlacement
    (vars (a b name) (index0 index1 text) (psk0 psk1 skey))
    (trace
      (init (cat index0 a b psk0))
      (init (cat index1 b a psk1)) ;; clientialization of key for parties a and b
    )
    (uniq-orig index0 index1)
    (non-orig psk0 psk1)
    (neq (index0 index1) (psk0 psk1) (a b)))

  (defrole client
    (vars (a b name) (index n1 n2 text) (psk skey))
    (trace
      (obsv (cat index a b psk))
      (send (cat n1 index))
      (recv (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
    )
  )

  (defrole server
    (vars (a b name) (index n1 n2 text) (psk skey))
    (trace
      (recv (cat n1 index))
      (obsv (cat index a b psk))
      (send (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
    )
  )

  (comment "Protocol with directional key, names added. Does not shows Selfie attack.")
)

```

Fig. 20. CPSA model of PSK authentication with directional keys.

```

(defprotocol tls4 basic

  (defrole keyPlacement
    (vars (a b c name) (index text) (psk skey))
    (trace
      (init (cat index a b psk))
      (init (cat index b a psk)))
    (uniq-orig index)
    (neq (a b));; (a c) (b c))
    (non-orig psk))

  (defrole client
    (vars (a b ca name) (index n1 n2 text) (psk skey))
    (trace
      (obsv (cat index a b psk))
      (send (cat n1 index))
      (recv (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (enc "CertificateRequest" (hash psk n1 n2)))
      (send (enc (enc a (privk ca)) (enc (hash (hash psk n1 n2) n1 index n2
      (hash (hash psk n1 n2) n1 index n2) "CertificateRequest") (privk a)) (hash psk n1 n2)))
      )
    (non-orig (privk ca))
    )

  (defrole server
    (vars (a b ca name) (index n1 n2 text) (psk skey))
    (trace
      (recv (cat n1 index))
      (obsv (cat index a b psk))
      (send (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (enc "CertificateRequest" (hash psk n1 n2)))
      (recv (enc (enc a (privk ca)) (enc (hash (hash psk n1 n2) n1 index n2
      (hash (hash psk n1 n2) n1 index n2) "CertificateRequest") (privk a)) (hash psk n1 n2)))
      )
    (non-orig (privk ca))
    )

  (comment "Protocol with post authentication assuming group key, shows impact of selfie attack")
)

```

Fig. 21. CPSA model of PSK use with post handshake authentication.

```

(defprotocol tls5 basic

  (defrole keyPlacement
    (vars (a b c name) (index text) (psk skey))
    (trace
      (init (cat index a b psk))
      (init (cat index b a psk))
      (init (cat index c b psk))
      (init (cat index b c psk))
      (init (cat index a c psk))
      (init (cat index c a psk)))
    (uniq-orig index)
    (neq (a b) (a c) (b c))
    (non-orig psk))

  (defrole client
    (vars (a b ca name) (index n1 n2 text) (psk skey))
    (trace
      (obsv (cat index a b psk))
      (send (cat n1 index))
      (recv (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (enc "CertificateRequest" (hash psk n1 n2)))
      (send (enc (enc a (privk ca)) (enc (hash (hash psk n1 n2) n1 index n2
        (hash (hash psk n1 n2) n1 index n2) "CertificateRequest") (privk a)) (hash psk n1 n2)))
      )
    (non-orig (privk ca))
    )

  (defrole server
    (vars (a b ca name) (index n1 n2 text) (psk skey))
    (trace
      (recv (cat n1 index))
      (obsv (cat index a b psk))
      (send (cat n2 (hash (hash psk n1 n2) n1 index n2)))
      (recv (hash (hash psk n1 n2) n1 index n2 (hash (hash psk n1 n2) n1 index n2)))
      (send (enc "CertificateRequest" (hash psk n1 n2)))
      (recv (enc (enc a (privk ca)) (enc (hash (hash psk n1 n2) n1 index n2
        (hash (hash psk n1 n2) n1 index n2) "CertificateRequest") (privk a)) (hash psk n1 n2)))
      )
    (non-orig (privk ca))
    )
  (comment "Protocol with post authentication assuming group key, shows impact of selfie attack")
  )

```

Fig. 22. CPSA model of PSK use with post handshake authentication and group key-ing.

References

1. Adrian, D., et al.: Imperfect forward secrecy: how Diffie-Hellman fails in practice. In: 22nd ACM Conference on Computer and Communications Security (Oct 2015)
2. Akhmetzyanova, L., Alekseev, E., Smyshlyaeva, E., Sokolov, A.: On post-handshake authentication and external psks in tls 1.3. *J. Comput. Virology Hacking Tech.* **16**(4), 269–274 (2020)
3. Al Fardan, N.J., Paterson, K.G.: Lucky thirteen: breaking the TLS and DTLS record protocols. In: 2013 IEEE Symposium on Security and Privacy, pp. 526–540 (2013). <https://doi.org/10.1109/SP.2013.42>

4. Beurdouche, B., et al.: A messy state of the union: taming the composite state machines of TLS. In: 2015 IEEE Symposium on Security and Privacy, pp. 535–552 (2015). <https://doi.org/10.1109/SP.2015.39>
5. Bhargavan, K., Blanchet, B., Kobeissi, N.: Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. Research Report RR-9040, Inria Paris, May 2017. <https://hal.inria.fr/hal-01528752>
6. Bhargavan, K., Lavaud, A.D., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: breaking and fixing authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy, pp. 98–113 (2014). <https://doi.org/10.1109/SP.2014.14>
7. Cremers, C., Horvat, M., Hoyland, J., Scott, S., van der Merwe, T.: A comprehensive symbolic analysis of TLS 1.3. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, pp. 1773–1788. Association for Computing Machinery, New York (2017). <https://doi.org/10.1145/3133956.3134063>, <https://doi.org/10.1145/3133956.3134063>
8. Cremers, C., Horvat, M., Scott, S., van der Merwe, T.: Automated analysis and verification of TLS 1.3: 0-rtt, resumption and delayed authentication. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 470–485 (2016). <https://doi.org/10.1109/SP.2016.35>
9. 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
10. Dowling, B., Fischlin, M., Günther, F., Stebila, D.: A cryptographic analysis of the TLS 1.3 draft-10 full and pre-shared key handshake protocol. Cryptology ePrint Archive, Report 2016/081 (2016). <https://eprint.iacr.org/2016/081>
11. Drucker, N., Gueron, S.: Selfie: reflections on TLS 1.3 with PSK. Cryptology ePrint Archive, Report 2019/347 (2019). <https://eprint.iacr.org/2019/347>
12. 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>
13. Fischlin, M., Günther, F.: Replay attacks on zero round-trip time: the case of the TLS 1.3 handshake candidates. Cryptology ePrint Archive, Report 2017/082 (2017). <https://eprint.iacr.org/2017/082>
14. Fischlin, M., Günther, F., Schmidt, B., Warinschi, B.: Key confirmation in key exchange: a formal treatment and implications for TLS 1.3. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 452–469 (2016). <https://doi.org/10.1109/SP.2016.34>
15. Guttman, J.D., Liskov, M.D., Ramsdell, J.D., Rowe, P.D.: The Cryptographic Protocol Shapes Analyzer (CPSA). <https://github.com/mitre/cpsa>
16. Housley, R., Hoyland, J., Sethi, M., Wood, C.: Guidance for External PSK Usage in TLS, draft-ietf-tls-external-psk-guidance-02, August 2021
17. Krawczyk, H., Wee, H.: The optls protocol and TLS 1.3. In: 2016 IEEE European Symposium on Security and Privacy (EuroS P), pp. 81–96 (2016). <https://doi.org/10.1109/EuroSP.2016.18>
18. Li, X., Xu, J., Zhang, Z., Feng, D., Hu, H.: Multiple handshakes security of TLS 1.3 candidates. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 486–505 (2016). <https://doi.org/10.1109/SP.2016.36>

19. Liskov, M.D., Ramsdell, J.D., Guttman, J.D., Rowe, P.D.: The Cryptographic Protocol Shapes Analyser: A Manual. Manual, The MITRE Corporation, May 2020. <https://github.com/mitre/cpsa>
20. Paterson, K.G., van der Merwe, T.: Reactive and proactive standardisation of TLS. In: Chen, L., McGrew, D., Mitchell, C. (eds.) SSR 2016. LNCS, vol. 10074, pp. 160–186. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49100-4_7
21. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446, August 2018