Analysis of EAP-GPSK Authentication Protocol

John C. Mitchell^1, Arnab Roy^1, Paul Rowe^2, and Andre Scedrov^2

^{1} Stanford University*
^{2} University of Pennsylvania**

Abstract. The EAP-GPSK protocol is a lightweight, flexible authentication protocol relying on symmetric key cryptography. It is part of an ongoing IETF process to develop authentication methods for the EAP framework. We analyze the protocol and find three weaknesses: a repairable Denial-of-Service attack, an anomaly with the key derivation function used to create a short-term master session key, and a ciphersuite downgrading attack. We propose fixes to these anomalies, and use a finite-state verification tool to search for remaining problems after making these repairs. We then prove the fixed version correct using a protocol verification logic. We discussed the attacks and our suggested fixes with the authors of the specification document which has subsequently been modified to include our proposed changes.

1 Introduction

The Extensible Authentication Protocol (EAP) [1] is an authentication framework developed by the Internet Engineering Task Force (IETF) which runs on the data link layer and supports a variety of mechanisms for two entities to authenticate themselves to each other. EAP is not itself an authentication protocol; rather it provides a context in which the entities can negotiate an authentication method such as Generalized Pre-Shared Key (GPSK) [2]. EAP is currently deployed on Point-to-Point connections, IEEE 802 wired networks, wireless LAN networks and over the Internet. The GPSK authentication method is a lightweight protocol being developed by the IETF EAP Method Update (EMU) working group. It uses symmetric cryptography and relies on a pre-shared key, as suggested by its name, between a server and a peer. The protocol seeks to minimize the number of messages exchanged, and hence is particularly well-suited for use in handheld devices where memory and computational resources are a limitation. The protocol is designed to provide mutual authentication and key agreement between the server and the peer. In this paper, we report improvements in the protocol and report on a formal analysis of the GPSK method. The main goal of our analysis is to have a positive impact on the protocol during a critical stage of the development and standardization process. As such, our suggested improvements attempt to keep the protocol largely in tact.

* Mitchell and Roy are partially supported by the NSF grant to Stanford and Penn, "Collaborative Research: High Fidelity Methods for Security Protocols" and by the TRUST NSF Science and Technology Center.

** Partially supported by OSD/ONR CIP/SW URI projects through ONR Grants N00014-01-1-0795 and N00014-04-1-0725. Additional support from NSF Grants CNS-0429689 and CNS-0524059 and from ONR Grant N00014-07-1-1039.

In this paper, we use finite-state model checking to find errors, and Protocol Composition Logic [3,4] to prove correctness after errors have been found and repaired. The model checker we use is called Murϕ [5]. Murφ has been successfully used in the past on a variety of protocols including Kerberos [6], SSL [7], and the 802.11i 4-Way Handshake [8]. As a model checker, Murφ is well suited for finding flaws but is insufficient to prove the correctness of a protocol. So to compliment Murφ we use Protocol Composition Logic (PCL) [9] as a proof tool. Murφ was useful in detecting some of the problems with the protocol specification as we first encountered it, while PCL was useful for proving that the fixes we suggested, and which were subsequently adopted, are correct.

Our analysis uncovered three weaknesses with GPSK. The first is a repairable Denial-of-Service attack against the peer, in which the attacker forces the peer to exhaust its memory thereby blocking the protocol. This attack is virtually identical to one which was found on the 802.11i 4-Way Handshake [8]. The second weakness is due to a non-standard use of the key derivation function which is used to create session keys. Although the non-standard use does not provide an obvious attack we cannot exclude the existence of an attack. In addition, we indicate the difficulties which such non-standard usage creates when trying to prove the protocol's correctness. Finally, we identify a ciphersuite downgrading attack in which the attacker can force the peer to choose a weak hash function or encryption algorithm. If the ciphersuite is susceptible to a key-recovery attack then the attacker can learn the session keys and then eavesdrop on all subsequent communications.

In addition to discovering these weaknesses we also suggest ways to fix them and prove that our proposed fixes make the protocol secure. As indicated above, we discussed the weaknesses and our suggested fixes with the authors of the GPSK specification. In turn, the authors presented the issues to the EMU working group for open discussion. They recognized the problems and they have since incorporated our suggested fixes to the most recent protocol specification. Our interaction with the authors came at a time in which GPSK was mature enough to undergo a thorough analysis, and yet early enough in the standardization process that it was not widely implemented.

There have been many efforts to develop and use methods for proving security properties of network protocols. In recent years, most efforts have used the so-called *symbolic model,* also referred to as the *Dolev-Yao* model [10,11,12]. In the symbolic model, protocol execution and the possible actions of an attacker are characterized using a symbolic model of computation that allows nondeterminism but does not incorporate probability or computational complexity bounds. In addition to many model checking and bug-finding efforts, there have been some significant correctness proofs carried using the symbolic model, including mechanically checked formal proofs [13,14], unformalized but mathematical proofs about a multiset rewriting model [15,16,17], and work using compositional formal logic approaches [18,19,20,21,22]. Several groups of researchers have taken steps to connect the symbolic model to the probabilistic polynomial-time *computational model* used in cryptographic studies, *e.g.,* [23,24,25,26,27,28,29,3,4,30]. Protocol Composition Logic has been used to prove correctness of versions of Kerberos in the symbolic model [31], and in the computational model [32], with errors in the Diffie-Hellman variant of Kerberos and proofs of security presented in [33]. Connections between symbolic trace properties and computational soundness properties are developed in [34]. All these efforts have been aimed at proving security properties for well-established protocols. Unfortunately, protocols such as EAP-GPSK are still being designed with flaws and weaknesses that are identical to those found and fixed in previous protocols. The major contribution of this work is to integrate the methods of protocol analysis into the standardization process before the protocol is deployed in a variety of implementations.

The rest of the paper is structured as follows. Section 2 describes in more detail the EAP framework and the GPSK method. In Section 3 we present the weaknesses we found and our suggested fixes. Section 4 describes the role that Murφ played in our analysis. In Section 5 we present a proof of correctness of the fixed protocol. We conclude in Section 6.

2 EAP

The Extensible Authentication Protocol (EAP) [1] is an authentication framework which is meant to support a variety of authentication methods. No single authentication protocol is defined. Instead, it defines message types that allow an authenticator and a peer to choose and perform an authentication mechanism. EAP is designed to run on the data link layer where IP connectivity may not be available. It provides support for duplicate elimination and retransmission but relies on lower layers to properly order packets. The authenticator may act as a pass-through and use a backend authentication server. This allows an implementation in which only the end server must be configured for a particular method. The authenticator does not have to be updated with the introduction of every new authentication method. The distinction between the authenticator and the authentication server does not arise in our analysis of the GPSK authentication method. Therefore we will treat the authenticator and the server as one entity referred to simply as the server.

EAP was designed to work with Point-to-Point connections, and was subsequently adapted for IEEE 802 wired networks as well as wireless LAN networks and over the Internet. In each of these settings an attacker may be able to control the network. EAP assumes an attacker can perform such actions as eavesdropping network traffic, modifying or spoofing packets, and offline dictionary attacks among others. This allows an attacker to attempt such attacks as person-in-the-middle attacks, ciphersuite downgrading attacks, and key recovery attacks due to weak key derivation.

An EAP conversation typically consists of three phases: discovery, authentication and secure association. In the discovery phase the two agents must identify each other and negotiate an authentication method. Then they carry out the chosen method in the authentication phase. The secure association phase occurs when the authenticator and the authentication server are distinct entities and so does not affect the present analysis. Our focus in the current analysis is on the authentication phase.

2.1 EAP-GPSK

The Generalized Pre-Shared Key (GPSK) protocol [2] is an EAP authentication method which is meant to be lightweight and flexible. To this end, it uses symmetric cryptography relying on a long-term, pre-shared key (denoted by PSK) between a server and a peer. The use of symmetric cryptogrpahy minimizes the computational resources required at either end of the communication, making it suitable for smartcards, handheld devices, or any device in which computational resources and memory are a significant constraint. To increase efficiency, the protocol also attempts to minimize the number of round trips. For flexibility the protocol allows for the negotiation of cryptographic ciphersuites which detail the encryption algorithm (if any), the message integrity mechanism and the key derivation algorithm the protocol participants will use. In this way, a server may authenticate several peers with a variety of local preferences for ciphersuites which may depend on the peers' computational constraints.

We show a successful message exchange in Fig. 1 at an abstract level. In the figure, S represents the server's ID and P represents the peer's ID. SNonce and PNonce are the server and peer nonces, respectively. CSuiteList represents the list of ciphersuites supported by the server, while CSuiteSel represents the ciphersuite selected by the peer. The field $\{\text{Payload}\}_{PK}$ represents an optional encrypted payload block that is a generic mechanism for exchanging confidential data. Higher level protocols may piggy-back on GPSK using the encrypted payload block to guarantee confidentiality. This means that sensitive confidential data may be sent as early as *Message 2*. *Messages 2, 3* and *4* each have a keyed message authentication code, MAC_{SK}, appended to the end. This is essentially a keyed hash of the rest of the message, although implementations depend upon the ciphersuite which is chosen.

[***Message 1*: S $\rightarrow$ P**]: SNonce, S, CSuiteList
[***Message 2*: P $\rightarrow$ S**]: P, S, PNonce, SNonce, CSuiteList, CSuiteSel, $\{\text{Payload}\}_{PK}$, MAC_{SK}
[***Message 3*: S $\rightarrow$ P**]: PNonce, SNonce, CSuiteSel, $\{\text{Payload}\}_{PK}$, MAC_{SK}
[***Message 4*: P $\rightarrow$ S**]: $\{\text{Payload}\}_{PK}$, MAC_{SK}

Fig. 1. Successful GPSK message exchange

The keys SK and PK are both derived from a key derivation function KDF-X by way of an intermediate master key MK. KDF-X takes two arguments, a key and a seed, and outputs a bit string of length X. The notation KDF-X(Y,Z)$[i..j]$ represents the i'th through j'th octets (8 bits) of the output of the KDF-X. The PSK has length PL, while the SK and PK have length KS which is a value specified by the ciphersuite. Key derivation is defined as follows:

inputString = PNonce || P || SNonce || S.
MK = KDF-KS(0x00, PL || PSK || CSuiteSel || inputString)[0..KS-1].
SK = KDF-{128+2*KS}(MK, inputString)[128..127+KS].
PK = KDF-{128+2*KS}(MK, inputString)[128+KS..127+2*KS].

The first 128 octets of KDF-{128+2*KS}(MK, inputString)[128+KS..127+2*KS] are divided into two keys which are exported as part of the protocol. They may be used for key derivation in higher level protocols. Every EAP method which supports key derivation is required to export such keys, but we omit them because they are not relevant to the current analysis.

GPSK is intended to provide mutual authentication between the peer and the server. After a successful message exchange the server should believe the peer is authentic due to the use of the key SK (derived from the long term key PSK) for the MAC in *Message 2*. Likewise, the peer should believe the server to be authentic due to the use of SK for the MAC in *Message 3*. GPSK is also intended to provide session independence. Even if the master key MK is compromised, this should not help an attacker to compromise past or future sessions. As with any symmetric key authentication protocol, the secrecy of the long-term key PSK is crucial for all of the above properties to hold. Unlike some protocols, GPSK does not support fast re-keying because the number of round trips is already at a minimum.

The peer and the server must silently discard any message which is unexpected (*e.g.* receiving *Message 4* instead of *Message 2*), doesn't parse (*e.g.* the wrong nonce is returned), or whose MAC is invalid. The only exception is for *Message 2*. If the server receives an invalid MAC then it must respond with an EAP failure message. The peer must always be willing to accept *Message 1* from a server since there is no integrity protection.

3 Anomalies

In our analysis of EAP-GPSK we found a number of anomalies. The first is a potential Denial-of-Service attack against the peer that is reminiscent of a similar attack found on the 802.11i 4-Way Handshake. We also identified a possible problem with the way in which the master key MK is derived. Lastly, we found a potential ciphersuite downgrading attack. Let us consider these issues one by one.

3.1 Denial-of-Service Attack

There is a simple Denial-of-Service attack that is made possible by the fact that *Message 1* provides no integrity protection. The result of the attack is a discrepancy between the SKs held by the peer and the server. This causes the peer to be unable to validate the MAC in *Message 3*. The protocol is blocked and the server will timeout and de-authenticate the peer. Obvious inspection shows that *Messages 2, 3* and *4* all have integrity protection. This means that if an attacker tries to forge these messages the peer can simply discard them when the MAC does not validate properly. We do not consider it an attack to cause the peer to attempt to validate a large number of MACs.

For simplicity the attack is explained in a situation where the peer can only maintain one open conversation with a given server. A message exchange for a successful attack is shown in Fig. 2. Only the relevant portions of each message are shown. When the peer receives a legitimate *Message 1* from a server it computes the MAC key SK, and responds with *Message 2*. At this point the attacker can send a fake *Message 1* with a new nonce. Since the peer can only have one open conversation with the server, and since the peer must always accept *Message 1*, the peer will recalculate the MAC key to a different value SK' when it receives the fake *Message 1* with a new nonce. When receiving a valid *Message 3*, the peer can no longer verify the MAC because it was calculated with the key SK, and the peer is trying to validate it with the new key SK'. At this point, the protocol can no longer proceed.

[*Message 1*:S $\rightarrow$ P]: SNonce, S
[*Message 2*:P $\rightarrow$ S]: P, S, PNonce, SNonce, MAC$_{SK}$
[*Message 1*':Attacker $\rightarrow$ P]: SNonce', S
(The peer chooses a new PNonce' and calculates SK'.)
[*Message 3*:S $\rightarrow$ P]: PNonce, SNonce, MAC$_{SK}$
(The peer uses SK' to check MAC$_{SK}$ and verification fails.)

Fig. 2. A Successful DoS attack on GPSK

If the peer can only hold one open conversation with a server the attack requires a single, well-timed message from the attacker. This may be prevented by allowing the peer to maintain several open conversations with a server. However, in this case an attacker can flood the network with *Message 1*' to exhaust the peer's allocated memory. The fact that GPSK is designed to work well in devices with limited resources makes this type of attack more feasible than it otherwise might be. The peer's memory resources are likely to be very restricted, allowing an attacker to fill the peer's memory with relatively little effort. The ability to flood the network with fake messages also reduces the attacker's reliance on good timing.

One might argue that the attack can be prevented by not allowing the peer to respond to a new *Message 1* when it is waiting for *Message 3*. However, this is not a viable option. This solution introduces the possibility that the protocol will be blocked, even in the absence of an attacker. It also enables another DoS attack which is even easier to execute as we explain below.

The principal concern, as was explained in [8], is that *Message 2* will not reach the server. Since EAP is frequently run on wireless LANs, packet loss is a legitimate concern. If the server does not receive *Message 2*, it will re-transmit *Message 1* after an appropriate timeout. The peer will discard re-transmissions because it is waiting for *Message 3*, and the server will never get a response. Alternatively, an attacker can cause the same problem by sending one fake *Message 1* to a peer before the server initiates a conversation. This will force the peer into a state in which it is waiting for *Message 3*, and any attempts by a server to authenticate the peer will be ignored.

This anomaly is virtually identical to a DoS attack found in the 802.11i 4-Way Handshake and presented in [8]. The solution we propose is therefore analogous to the solution which was proposed by the authors of [8] and ultimately adopted by the 802.11 working group. The memory exhaustion attack is possible because the peer is forced to maintain state for each *Message 1* it receives. The proposed solution allows the peer to maintain state for each server regardless of the number of times it receives *Message 1*. Since the number of servers associated with a given peer is likely to be small, this should drastically reduce the memory used by a peer.

The first time a peer receives *Message 1* from a server it will choose a fresh PNonce and remember it. If it receives another *Message 1* from the same server before completing the protocol then the peer will re-use PNonce. *Message 2* will remain as it is. The fact that *Message 3* has a MAC whose key depends on the PSK allows the peer to trust its contents. Instead of storing the SK used in *Message 2*, the peer will recalculate SK when it receives *Message 3*. This solution requires *Message 3* to contain enough information for the peer to re-compute SK. Currently, *Message 3* does not contain the server's ID which is necessary to compute SK. One way to fix this is to leave the message format as it is, and change the key derivation. If SK no longer depends on the server's ID, then the peer can compute it at this stage. Another option is to add the server's ID to *Message 3* and leave the key derivation alone. This would also provide the peer with all the information it needs to re-compute SK. Ultimately, the latter solution was adopted by the EMU working group.

Let us see how this solution defends against the attack in Fig. 2. When the attacker sends a fake *Message 1*, the peer no longer updates its own nonce. The peer will compute a new SK' for *Message 2*' based on the fake SNonce'. It will store neither SNonce' nor the new SK'. Now when the peer receives a legitimate *Message 3*, it can verify that PNonce was correctly returned, and it can use all the information provided to re-compute SK and verify the dependence on PSK. This will convince the peer that the legitimate server sent *Message 3*.

While this solution does not introduce integrity to *Message 1*, it does prevent the peer from changing state in response to an unauthenticated message. By re-using PNonce an attacker can now cause the peer to produce many copies of *Message 2* with the same key by sending many forged copies of *Message 1* using the same nonce. If the encrypted payload changes every time then this gives the attacker access to many different encryptions under the same key. The peer should be aware of this and choose a ciphersuite which is strong enough to resist cryptanalysis under these conditions.

3.2 Non-standard Key Derivation

The second anomaly we found involved the derivation of the master key MK. Recall from above that MK is derived by:

MK = KDF-KS(0x00, PL || PSK || CSuiteSel || inputString)

The use of "0x00" as the key to the KDF is not standard. In TLS [35] for example the master_secret, which corresponds to MK, is derived from a KDF which uses the long-term shared key (pre_master_secret) as its key input:

master_secret = PRF(pre_master_secret, "master secret", ClientHello.Random + ServerHello.Random)

The derivation of MK in GPSK does not provide an obviously reliable way for an attacker to learn MK or the keys SK and PK. It is unwise, however, to deviate too much from accepted standard usage which has undergone thorough investigation in other protocols.

The current derivation poses problems on theoretical grounds as well. Cryptographic implementations are often accepted because they provide strong guarantees when one assumes that the implementations satisfy standard assumptions. For example, the key derivation function for TLS is transparently assumed to act as a pseudo-random function (PRF). Let us now examine what assumptions about KDF are necessary for the current implementation to provide strong keys.

The KDF is being used in two different capacities, one for the MK derivation and another for the SK and PK derivations. In the MK derivation it acts like a hash function or a randomness extractor whereas in the SK and PK derivations, it acts like a PRF.

The cryptographic security of SK and PK are defined assuming uniformly random execution of a key generation algorithm. Assuming, for the particular schemes in use for this protocol, that the keys are sampled uniformly from the key space of a given length, we would require SK and PK to be computationally indistinguishable from purely random numbers of the same length.

Working backwards from the SK and PK derivations, if we model the KDF as a PRF, we can ensure that SK and PK are computationally indistinguishable from random, as long as MK itself is also computationally indistinguishable from random. So the requirement reduces to ensuring that MK is pseudo-random. What should the KDF with key 0x00 behave like in order to ensure this? Let us denote by $\mathrm{kdf}_{00}(y)$ the function KDF(0x00, y). We have the following alternatives:

1. kdf_{00} is a Random Oracle: Then MK would indeed be perfectly random by definition. However, this is a very strong assumption and although a theoretically useful model, it is unrealizable in practice and its usage is debatable [36].

2. KDF is a PRF: This is too weak. It is possible to construct perfectly valid PRFs which output a constant if the key is 0x00. This does not violate pseudo-randomness because a PRF's output 'seems' to be random with a randomly chosen key. The probability of a key being all 0's is exponentially small in the security parameter and hence this is a very low probability event.

3. kdf_{00} is a pseudo-random generator (PRG): The trouble with this model is that a PRG's output is claimed to be pseudo-random only if the input seed is uniformly random and unknown. However, for this protocol, parts of the seed are known (the nonces, IDs, ...) and neither is it uniformly random (IDs and CSuiteSel have structure).

Thus the alternatives that are weaker than the random oracle model do not guarantee strong keys. Perhaps the simplest solution to the problem is to use PSK as the key when deriving MK. In that case, if we assume that KDF is a PRF, MK will be indistinguishable from a random number of the same length because PSK is random and unknown. This implies that both SK and PK will be indistinguishable from random.

In talking with the authors of the specification it seems that the reason this approach was not taken from the start was because different ciphersuites have different key lengths and PSK might not be the right length for some of them. The working group finally decided to require PSK to be long enough for all current (and many future) ciphersuites. Then PSK will be truncated to be the right length if it is too long for the chosen ciphersuite.

3.3 Ciphersuite Downgrading Attack

The last anomaly we found was a potential ciphersuite downgrading attack. Just as with the DoS attack, this also arises from the fact that the first message has no integrity protection. An attacker who controls the network can modify a legitimate message from a server. In this case the attacker can change CSuiteList to include only weak ciphersuites.

In particular, the attacker might force the peer to choose a ciphersuite which does not provide an encryption mechanism, or which provides an encryption algorithm that the attacker can break in real time. In this case, any data which is passed in the encrypted payload block in *Message 2* can be read by the attacker.

There is an additional concern if the weak ciphersuite is susceptible to a key-recovery attack. If the attacker is able in real time to recover the session keys based only on the knowledge of the nonces and the IDs of the peer and the server, then he will break authentication. The attacker could block *Message 2* and replace the peer's nonce with a freshly chosen nonce, and compute the corresponding key to create a valid MAC. In this way the attacker could impersonate both the peer and the server giving him full control over their communication.

To counteract such an attack the protocol designers have required that CSuiteList in *Message 1* must contain two specified ciphersuites. While this should ensure that some of the ciphersuites offered meet some minimum strength, it does not ensure that the peer chooses the strongest ciphersuite which it supports. Although the peer will likely support both of these ciphersuites it has the freedom to choose a weaker ciphersuite if it is offered. In addition, one of the required ciphersuites does not support encryption. Although this ciphersuite is among those required to be supported because it does not suffer from a key-recovery attack, an unwitting peer may choose this ciphersuite and still attempt to send confidential data in *Message 2*.

This attack seems unavoidable as long as *Message 1* continues to lack integrity protection. As long as the peer is allowed to send encrypted data in *Message 2* before the CSuiteList is authenticated, this data might be sent even after choosing a ciphersuite without support for encryption. However, the damage of this attack can be fully avoided as long as the peer is aware of the potential problem and chooses a strong ciphersuite. Also, if the peer wishes to send confidential data, then it must choose a ciphersuite which supports encryption, and it must wait until *Message 4* to send the data. After discussing the problem with the protocol authors, the specification was amended to contain warnings for the peer.

4 Model Checking the Protocol

Finite-state verification tools such as Murφ have proven to be very useful in the analysis of security protocols. Murφ was successfully used in [6] to verify small protocols such as the Needham-Schroeder public key protocol, the Kerberos protocol, and the TMN cellular telephone protocol. In [7] Murφ was also successfully applied to the analysis of the SSL 3.0 handshake protocol using a "rational reconstruction" methodology which was adopted in [8] to analyze the 802.11i 4-Way Handshake.

Tools such as Murφ, commonly called model checkers, verify specified properties of a nondeterministic system by explicitly enumerating all possible execution sequences and checking for states which violate the specified properties. Although security protocols are infinite systems, many flaws have been found in finite (and very small) approximations to them. While finite verification has been successful in finding bugs, failure to find a bug does not mean the protocol is secure. Certain simplifications must be made when creating a model and these may eliminate crucial details. The restriction to a finite state space with only a small number of participants is also a crucial limitation.

The use of Murφ for the current analysis served two main goals. First, it was used to help detect the flaws found in the original specification of GPSK [2]. Second, we used it to perform a preliminary search for new flaws which may have been introduced by the fixes we proposed. Since a run of Murφ in which no flaws are found does not imply the security of the protocol the next section is dedicated to the describing the process of giving a formal proof of the security properties.

To use Murφ to verify the protocol we must create a model of the protocol following the specification, add a model of an attacker, state the security properties we would like to check and then run Murφ on the model. In this last step Murφ searches through all of the possible traces of the protocol checking at each step if any of the security properties fail. If so then it will return the trace which ends in the violation. This allows the user to see what caused the problem.

In formulating a model of the protocol the honest participants do not stray from the specification. They act deterministically. We assume that every pair of servers and clients shares a long term PSK. Since PSK is never used as a key we will assume that the attacker cannot recover any PSK. The MAC key SK is simply modeled as a list containing PSK and the inputString. Again, since Murφ is not well-suited for discovering low level cryptographic attacks we assume that SK will remain secret. Thus our model contains no mechanism for the attacker to learn SK. Since the encrypted payload block is not explicitly used as part of the authentication mechanism, we exclude this component of the messages. For simplicity we model a good ciphersuite list and a bad one by a 1 or a 0 respectively. We similarly model a good (strong) ciphersuite selection and a bad one. Since we cannot model the lower level details of the ciphersuites (*i.e.* the way they function on bitstrings) this good/bad distinction should be enough to detect a ciphersuite downgrading attack.

Despite the above restrictions on the attacker, he still can act nondeterministically in a variety of ways. The attacker can eavesdrop and remember any message sent on the network. Since there is no encryption used, the attacker can read the plaintext of any message and decompose it into its various parts. He can block any message, and he can generate and send any message which is made up of components from other messages. In particular, this means that an attacker can replay complete messages. Since SK is assumed to be secure, the attacker will remember and use the MACs he sees as indecomposable units.

Since PSK is assumed to remain secret the properties we are concerned with in our analysis are mutual authentication and consistency of the key SK. Namely, at the end of a session the peer and the server must each believe they are talking to each other, and they must share the same key SK. We also check to see if the protocol is blocked by the attacker as it is in the DoS attack. Finally, we have Murφ print a trace if a ciphersuite downgrading attack occurs. Once these attacks were detected we added a feature which allows us to turn these attacks off. This allows us to more efficiently search for other attacks.

Although Murφ only creates finite models, it does allow us to specify parameters which will determine the size of the model. In this way, we can write one model that can be scaled up or down simply by switching the values of these parameters. In modeling GPSK we chose to vary the number of peers and the number of servers. We can control the number of nonces available, the number of sessions to be completed and the number of distinct actions the attacker may execute.

Murφ proved useful in successfully detecting and verifying the existence of the DoS attack as well as the ciphersuite downgrading attack. We then ran Murφ on the model with the suggested fixes. Since our proposed fix for the ciphersuite downgrading attack does not change the message exchange, we simply turn the attack off in order to assume that the peer chooses a strong ciphersuite. We successfully verified that no errors exist when there is just one peer and one server engaging in up to 10 sessions. In addition we verified correctness with one peer and two servers engaging in 3 sessions total as well as the situation with two peers and one server engaging in up to 3 sessions. In attempting to verify the case with 2 peers and 2 servers engaging in 2 sessions total we ran out of memory. While the verification didn't run to completion Murφ also failed to find any flaws before exhausting memory. This may be taken as tentative evidence of a lack of an attack since Murφ prints a violating trace as soon as a problem is found. The model checking of this protocol was very familiar the previous experience of model checking the 802.11i 4-Way Handshake [8], so we did not investigate it further.

5 Proof of Correctness

In this section, we present a formal correctness proof of EAP-GPSK using *Protocol Composition Logic (PCL)* [37,38,39,40,41,19,18]. In previous work, PCL has been proved sound for protocol runs that use any number of principals and sessions, over both symbolic models and (for a subset of the logic at present) over more traditional cryptographic assumptions [3].

5.1 Overview of Proof Method

We begin with a brief discussion of PCL relevant to the analysis of EAP-GPSK.

Modeling protocols. A protocol is defined by a set of roles, each specifying a sequence of actions to be executed by an honest agent. In PCL, protocol roles are represented using a simple "protocol programming language" based on *cords* [37]. The possible protocol actions include nonce generation, signatures and encryption, communication steps, and decryption and signature verification via pattern matching. Programs can also depend on input parameters (typically determined by context or the result of set-up operations) and provide output parameters to subsequent operations.

Protocol Logic and the Proof System. For a summary of the proof system and the proof of soundness of the axioms and the rules, we refer the reader to [9,38,19]. Most protocol proofs use formulas of the form $\theta[P]_X\phi$, which means that starting from a state where formula θ is true, after actions P are executed by the thread X, the formula ϕ is true in the resulting state. Formulas ϕ and ψ typically make assertions about temporal order of actions (useful for stating authentication) and/or the data accessible to various principals (useful for stating secrecy).

The proof system extends first-order logic with axioms and proof rules for protocol actions, temporal reasoning, knowledge, and a specialized form of invariance rule called the *honesty rule*. The honesty rule is essential for combining facts about one role with inferred actions of other roles, in the presence of attackers. Intuitively, if Alice receives a response from a message sent to Bob, the honesty rule captures Alice's ability to use properties of Bob's role to reason about how Bob generated his reply. In short, if Alice assumes that Bob is honest, she may use Bob's role to reason from this assumption.

5.2 Formal Description of EAP-GPSK in the PCL Programming Language

```
$\mathbf{GPSK : Server} \equiv [$
new $SNonce$;
send $SNonce.\hat{S}.CSL$;

receive $\hat{P}.\hat{S}.PNonce.SNonce.$
  $CSL.CSS.enc1.mac1$;
$MK :=$ prg $PSK$;
$InputString := PNonce.\hat{P}.SNonce.\hat{S}$;
$SK :=$ kdf1 $InputString, MK$;
$PK :=$ kdf2 $InputString, MK$;
$pl1 :=$ symdec $enc1, PK$;
verifymac $mac1, \hat{P}.\hat{S}.PNonce.SNonce.$
  $CSL.CSS.enc1, SK$;
$enc2 :=$ symenc $pl2, PK$;
$mac2 :=$ mac $PNonce.SNonce.$
  $CSL.enc2, SK$;
send $PNonce.SNonce.\hat{S}.CSL.enc2.mac2$;

receive $enc3.mac3$;
verifymac $mac3, enc3, SK$;
$pl3 :=$ symdec $enc3, PK$;
$]_S$
```

```
$\mathbf{GPSK : Peer} \equiv [$
receive $SNonce.\hat{S}.CSL$;
new $PNonce$;
$MK :=$ prg $PSK$;
$InputString := PNonce.\hat{P}.SNonce.\hat{S}$;
$SK :=$ kdf1 $InputString, MK$;
$PK :=$ kdf2 $InputString, MK$;
$enc1 :=$ symenc $pl1, PK$;
$mac1 :=$ mac $\hat{P}.\hat{S}.PNonce.SNonce.$
  $CSL.CSS.enc1$;
send $\hat{P}.\hat{S}.PNonce.SNonce.$
  $CSL.CSS.enc1.mac1$;

receive $PNonce.SNonce.\hat{S}.CSL.$
  $enc2.mac2$;
verifymac $mac2, PNonce.SNonce.$
  $CSL.enc2, SK$;
$pl2 :=$ symdec $enc2$;
$enc3 :=$ symenc $pl3$;
$mac3 :=$ mac $enc3, SK$;
send $enc3.mac3$;
$]_P$
```

5.3 EAP-GPSK Security Properties

Setup Assumption. To establish security properties of the *EAP-GPSK* protocol, we assume that the Server $\hat{S}$ and the Peer $\hat{P}$ in consideration are both honest and the only parties which know the corresponding shared PSK. However, we allow all other principals in the network to be potentially malicious and capable of reading, blocking and changing messages being transmitted according to the symbolic model of a network attacker.

$$\phi_{setup} \equiv \mathsf{Honest}(\hat{P}) \wedge \mathsf{Honest}(\hat{S}) \wedge (\mathsf{Has}(X, PSK) \supset \hat{X} = \hat{S} \vee \hat{X} = \hat{P})$$

Security Theorems. The secrecy theorem for *EAP-GPSK* establishes that the signing and encryption keys SK and PK should not be known to any principal other than the peer and the server. For server $\hat{S}$ and peer $\hat{P}$, this property is formulated as $SEC_{gpsk}(S, P)$ defined as:

$$SEC_{gpsk}(S, P) \equiv (\mathsf{Has}(X, PK) \vee \mathsf{Has}(X, SK)) \supset (\hat{X} = \hat{S} \vee \hat{X} = \hat{P})$$

Theorem 1 (Secrecy). *On execution of the server role, key secrecy holds. Similarly for the peer role. Formally, EAP-GPSK* $\vdash SEC^{server}_{pk,sk}, SEC^{peer}_{pk,sk}$*, where*

$$\begin{aligned} SEC^{server}_{pk,sk} &\equiv [\mathbf{GPSK} : \mathbf{Server}]_S \; SEC_{gpsk}(S, P) \\ SEC^{peer}_{pk,sk} &\equiv [\mathbf{GPSK} : \mathbf{Peer}]_P \; SEC_{gpsk}(S, P) \end{aligned}$$

Proof Sketch. We skip the rigorous formal proof here, but the proof intuition is as follows: PSK is assumed to be known to $\hat{P}$ and $\hat{S}$ only. The keys SK, PK are derived by using PSK in a key derivation function (MK could be a truncation of PSK or generated by application of a PRG to PSK, according to the length needed). The honest parties use SK, PK as only encryption or signature keys - none of the payloads are derived by a kdf application. This is the intuition why SK, PK remain secrets. A rigorous proof would employ a stronger induction hypothesis and induction over all honest party actions. □

The authentication theorem for *EAP-GPSK* establishes that on completion of the protocol, the principals agree on each other's identity, protocol completion status, the cryptographic suite list and selection, and each other's nonces. The authentication property for *EAP-GPSK* is formulated in terms of matching conversations [42]. The basic idea of matching conversations is that on execution of a server role, we prove that there exists a role of the intended peer with a corresponding view of the interaction. For server $\hat{S}$, communicating with client $\hat{P}$, matching conversations is formulated as $AUTH_{gpsk}(S, P)$ defined below:

$$\begin{aligned} AUTH_{gpsk}(S, P) \equiv \quad & (\mathsf{Send}(S, msg1) < \mathsf{Receive}(P, msg1)) \wedge \\ & (\mathsf{Receive}(P, msg1) < \mathsf{Send}(P, msg2)) \wedge \\ & (\mathsf{Send}(P, msg2) < \mathsf{Receive}(S, msg2)) \wedge \\ & (\mathsf{Receive}(S, msg2) < \mathsf{Send}(S, msg3)) \end{aligned}$$

Theorem 2 (Authentication). *On execution of the server role, authentication holds. Similarly for the peer role. Formally, EAP-GPSK* $\vdash AUTH^{server}_{peer}, AUTH^{peer}_{server}$*, where*

$$\begin{aligned} AUTH^{server}_{peer} &\equiv [\mathbf{GPSK} : \mathbf{Server}]_S \; \exists \eta.\ P = (\hat{P}, \eta) \wedge AUTH_{gpsk}(S, P) \\ AUTH^{peer}_{server} &\equiv [\mathbf{GPSK} : \mathbf{Peer}]_P \; \exists \eta.\ S = (\hat{S}, \eta) \wedge AUTH_{gpsk}(S, P) \end{aligned}$$

Proof Sketch. The formal proof in PCL is in Appendix A. We describe the proof intuition here. We needed to add two new axioms **MAC0** and **VMAC** (also written in Appendix A) to the extant PCL proof system in order to reason about macs. Axiom

$\mathbf{MAC0}$ says that anybody computing a mac on a message m with key k must possess both m and k. Axiom $\mathbf{VMAC}$ says that if a mac is verified to be correct, it must have been generated by a `mac` action.

$AUTH^{server}_{peer}$: The Server verifies the $mac1$ on $msg2$ to be a mac with the key SK. By axiom $\mathbf{VMAC}$, it must have been generated by a `mac` action and by $\mathbf{MAC0}$, it must be by someone who has SK. Hence by secrecy, it is either P or S and hence in either case, an honest party. It is an invariant of the protocol that a mac action on a message of the form $\hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS.enc$ is performed by a thread of $\hat{X}$, captured by Γ_1 - hence it must be a thread of $\hat{P}$, say P. Also using Γ_1 we prove that P received the first message and generated nonce $PNonce$ and sent it out first in the message $msg2$. From the actions of S, we also have that S newly generated $SNonce$ and sent it out first in $msg1$. Using this information and axioms $\mathbf{FS1}, \mathbf{FS2}$, we can order the receives and sends as described in $AUTH^{server}_{peer}$.

$AUTH^{peer}_{server}$: The Peer verifies the $mac2$ on $msg3$ to be a mac with the key SK. By axiom $\mathbf{VMAC}$, it must have been generated by a `mac` action and by $\mathbf{MAC0}$, it must be by someone who has SK. Hence by secrecy, it is some thread of either $\hat{P}$ or $\hat{S}$ and hence in either case, an honest party. It is an invariant of the protocol that a mac action on a message of the form $YNonce.XNonce.\hat{Y}.CSL.enc$, is performed by a thread of $\hat{Y}$, captured by Γ_2 - hence it must be a thread of $\hat{S}$, say S.

However this mac does not bind the variables CSS and $enc1$ sent in $msg2$. So to ensure that S received the exact same message that P sent, we use Γ_2 to further reason that S verified a mac on a message of the form $msg2$ and axioms $\mathbf{VMAC}, \mathbf{MAC0}$ again to reason that this mac was generated by threads of $\hat{S}$ or $\hat{P}$. Now, we can use Γ_1 and the form of $msg2$ to reason that a thread of $\hat{P}$ did it which also generated $PNonce$ - hence by $\mathbf{AN1}$, it must be P itself. Now we use an invariant stating that a thread generating such a mac does it uniquely, captured by Γ_3, thus binding $CSS, enc1$. Now we use $\mathbf{FS1}, \mathbf{FS2}$ as in the previous proof to establish the order described in $AUTH^{peer}_{server}$. □

Discussion. The formal proof presented above applies to the case where fresh nonces are generated every time. When the peer uses the same nonce repeatedly until it succeeds in completion we have to use a different form of reasoning to ensure the intended message ordering. Specifically, the predicate $\mathsf{FirstSend}(P, PNonce, msg2)$ does not necessarily hold anymore. However, we can still appeal to the fact, that a MAC must have been generated and sent out before it could be received and verified, in order to order messages. Formalizing this requires the new axiom $VMAC'$:

$$\begin{aligned}\mathbf{VMAC'}\quad & \mathsf{Receive}(X, m2) \wedge \mathsf{Contains}(m2, m') \wedge \mathsf{VerifyMac}(X, m', m, k) \wedge \\ & \neg\mathsf{Mac}(X, m, k) \supset \quad \exists Y, m1.\ \mathsf{Mac}(Y, m, k) \wedge \mathsf{Contains}(m1, m') \wedge \\ & (\mathsf{Send}(Y, m1) < \mathsf{Receive}(X, m2))\end{aligned}$$

The proof above uses axioms previously proved sound in the symbolic model. While proofs for some properties of *EAP-GPSK* in the computational model could be carried out in computational PCL ([3,32]), we currently do not have the technical machinery to prove message ordering as a consequence of using fresh nonces in CPCL.

6 Conclusions

In this paper we analyzed the EAP-GPSK authentication protocol. We found three anomalies: a repairable DoS attack, an anomaly in the derivation of the master key MK, and a potential ciphersuite downgrading attack. While the third anomaly seems unavoidable, proper awareness of an attacker's ability to weaken CSuiteList in *Message 1* should prevent problems from arising.

We found that by flooding the network with fake *Message 1*'s, an attacker can force a peer to re-compute the MAC key SK, causing the peer to be unable to correctly process *Message 3* from a legitimate server. This attack is especially worrisome when considering that GPSK is designed to work on devices with limited memory which can easily be exhausted. We propose a fix that allows the peer to maintain state per server instead of state per message.

We identified an anomaly in the derivation of the master key MK. Specifically, MK was derived using a KDF with constant key 0x00. While this does not provide an obvious way for an attacker to reliably learn session keys, it is better to use a more standard implementation.

We used a finite state verification tool named Murφ to search for new problems which may have arisen from the fixes we proposed. We found none. Finally we proved the fixed protocol correct. The analysis was introduced during the standardization process. Throughout our analysis we discussed the weaknesses and possible solutions with the IETF EMU working group. The changes we suggested have subsequently been adopted by the protocol designers. They have been incorporated in the latest internet draft.

Acknowledgements

We would like to thank the protocol designers Charles Clancy and Hannes Tschofenig as well as the EMU working group chair Joe Salowey for fruitful and enjoyable discussions. We also want to thank Sam Hartman for putting us in contact with the EMU working group. We also thank Adam Barth and Taral Joglekar for helpful input in our initial discussions. Finally, we thank the anonymous reviewers for their useful suggestions and comments.

References

1. Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J.: Extensible Authentication Protocol. RFC 3748 (2004)
2. Clancy, T., Tschofenig, H.: EAP Generalized Pre-Shared Key (EAP-GPSK) (work in progress) (2007), `http://www.ietf.org/internet-drafts/draft-ietf-emu-eap-gpsk-05.txt`
3. Datta, A., Derek, A., Mitchell, J.C., Shmatikov, V., Turuani, M.: Probabilistic polynomial-time semantics for a protocol security logic. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 16–29. Springer, Heidelberg (2005)
4. Datta, A., Derek, A., Mitchell, J.C., Warinschi, B.: Computationally sound compositional logic for key exchange protocols. In: Proceedings of 19th IEEE Computer Security Foundations Workshop, pp. 321–334. IEEE, Los Alamitos (2006)

5. Dill, D.L.: The Murϕ Verification System. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 147–158. Springer, Heidelberg (1996)
6. Mitchell, J.C., Mitchell, M., Stern, U.: Automated Analysis of Cryptographic Protocols Using Murϕ. In: IEEE Symposium on Security and Privacy, pp. 141–151 (1997)
7. Mitchell, J.C., Shmatikov, V., Stern, U.: Finite-State Analysis of SSL 3.0. In: SSYM 1998: Proceedings of the 7th conference on USENIX Security Symposium, 1998, Berkeley, CA, USA, pp. 16–16. USENIX Association (1998)
8. He, C., Mitchell, J.C.: Analysis of the 802.11i 4-Way Handshake. In: WiSe 2004: Proceedings of the 3rd ACM Workshop on Wireless Security, pp. 43–50. ACM, New York (2004)
9. Datta, A., Derek, A., Mitchell, J.C., Roy, A.: Protocol Composition Logic (PCL). Electron. Notes Theor. Comput. Sci. 172, 311–358 (2007)
10. Meadows, C.: A model of computation for the NRL protocol analyzer. In: Proceedings of 7th IEEE Computer Security Foundations Workshop, pp. 84–89. IEEE, Los Alamitos (1994)
11. Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, A.: Modelling and Analysis of Security Protocols. Addison-Wesley Publishing Co., Reading (2000)
12. Fábrega, 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, Oakland, CA, pp. 160–171. IEEE Computer Society Press, Los Alamitos (1998)
13. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6, 85–128 (1998)
14. Bella, G., Paulson, L.C.: Kerberos version IV: Inductive analysis of the secrecy goals. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 361–375. Springer, Heidelberg (1998)
15. Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A.: A Formal Analysis of Some Properties of Kerberos 5 Using MSR. In: Fifteenth Computer Security Foundations Workshop — CSFW-15, Cape Breton, NS, Canada, pp. 175–190. IEEE Computer Society Press, Los Alamitos (2002)
16. Butler, F., Cervesato, I., Jaggard, A.D., Scedrov, A.: Verifying confidentiality and authentication in kerberos 5. In: Futatsugi, K., Mizoguchi, F., Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 1–24. Springer, Heidelberg (2004)
17. Cervesato, I., Jaggard, A., Scedrov, A., Tsay, J.K., Walstad, C.: Breaking and fixing public-key kerberos (Technical report)
18. Cervasato, I., Meadows, C., Pavlovic, D.: An encapsulated authentication logic for reasoning about key distribution. In: CSFW-18, IEEE Computer Society, Los Alamitos (2005)
19. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system and compositional logic for security protocols. Journal of Computer Security 13, 423–482 (2005)
20. Hasebe, K., Okada, M.: Non-monotonic properties for proving correctness in a framework of compositional logic. In: Foundations of Computer Security Workshop, pp. 97–113 (2004)
21. Hasebe, K., Okada, M.: Inferences on honesty in compositional logic for security analysis. In: Futatsugi, K., Mizoguchi, F., Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 65–86. Springer, Heidelberg (2004)
22. He, C., Sundararajan, M., Datta, A., Derek, A., Mitchell, J.C.: A modular correctness proof of ieee 802.11i and tls. In: ACM Conference on Computer and Communications Security, pp. 2–15 (2005)
23. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). Journal of Cryptology 15, 103–127 (2002)
24. Backes, M., Pfitzmann, B., Waidner, M.: A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015 (2003)
25. Baudet, M., Cortier, V., Kremer, S.: Computationally Sound Implementations of Equational Theories against Passive Adversaries. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg (2005)

26. Canetti, R., Herzog, J.: Universally composable symbolic analysis of mutual authentication and key-exchange protocols. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 380–403. Springer, Heidelberg (2006)
27. Herzog, J.: Computational Soundness for Standard Assumptions of Formal Cryptography. PhD thesis, MIT (2004)
28. Adão, P., Bana, G., Scedrov, A.: Computational and information-theoretic soundness and completeness of formal encryption. CSFW 18, 170–184 (2005)
29. Micciancio, D., Warinschi, B.: Soundness of formal encryption in the presence of active adversaries. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 133–151. Springer, Heidelberg (2004)
30. Warinschi, B.: A computational analysis of the Needham-Schroeder(-Lowe) protocol. In: Proceedings of 16th Computer Science Foundation Workshop, pp. 248–262. ACM Press, New York (2003)
31. Roy, A., Datta, A., Derek, A., Mitchell, J.C., Seifert, J.P.: Secrecy analysis in Protocol Composition Logic. In: Proceedings of 11th Annual Asian Computing Science Conference (to appear, 2006)
32. Roy, A., Datta, A., Derek, A., Mitchell, J.C.: Inductive proofs of computational secrecy. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 219–234. Springer, Heidelberg (2007)
33. Roy, A., Datta, A., Mitchell, J.C.: Formal proofs of cryptographic security of Diffie-Hellman-based protocols. In: 3rd Symposium on Trustworthy Global Computing, TGC 2007. LNCS, vol. 4912, pp. 312–329. Springer, Heidelberg (2008)
34. Roy, A., Datta, A., Derek, A., Mitchell, J.C.: Inductive trace properties for computational security. In: WITS (2007)
35. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (2006)
36. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: STOC 1998: Proceedings of the thirtieth annual ACM symposium on Theory of computing, pp. 209–218. ACM, New York (1998)
37. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for protocol correctness. In: Proceedings of 14th IEEE Computer Security Foundations Workshop, pp. 241–255. IEEE, Los Alamitos (2001)
38. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for proving security properties of protocols. Journal of Computer Security 11, 677–721 (2003)
39. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: A derivation system for security protocols and its logical formalization. In: CSFW-16, pp. 109–125. IEEE, Los Alamitos (2003)
40. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Secure protocol composition. In: Proceedings of 19th Annual Conference on Mathematical Foundations of Programming Semantics. Electronic Notes in Theoretical Computer Science, vol. 83 (2004)
41. Datta, A., Derek, A., Mitchell, J.C., Pavlovic, D.: Abstraction and refinement in protocol derivation. In: CSFW-17, pp. 30–45. IEEE, Los Alamitos (2004)
42. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)

A Formal Proofs

New Axioms

$$\begin{array}{ll}
\mathbf{MAC0} & \mathsf{Mac}(X, m, k) \supset \mathsf{Has}(X, m) \wedge \mathsf{Has}(X, k) \\
\mathbf{VMAC} & \mathsf{VerifyMac}(X, m', m, k) \supset \exists Y.\, \mathsf{Mac}(Y, m, k) \wedge m' = MAC[k](m)
\end{array}$$

Invariants

$$\begin{aligned}
\Gamma_1 \equiv\ & \mathsf{Mac}(Z, \hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS.enc, K) \supset \\
& \hat{Z} = \hat{X} \wedge (\mathsf{Receive}(Z, YNonce.\hat{Y}.CSL) < \\
& \mathsf{Send}(Z, \hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS.enc.mac)) \wedge \\
& mac = MAC[K](\hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS.enc) \wedge \\
& \mathsf{FirstSend}(Z, XNonce, \hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS.enc.mac) \\
\Gamma_2 \equiv\ & \mathsf{Mac}(Z, YNonce.XNonce.\hat{Y}.CSL.enc, SK) \wedge SK = KDF1[K](YNonce.\hat{Y}.XNonce.\hat{X}) \supset \\
& \hat{Z} = \hat{Y} \wedge \exists CSS', enc1.\ (\mathsf{Send}(Z, YNonce.\hat{Y}.CSL) < \\
& \mathsf{Receive}(Z, \hat{Y}.\hat{X}.YNonce.XNonce.CSL.CSS'.enc1.mac1) < \\
& \mathsf{Send}(Z, YNonce.XNonce.CSL.enc.mac)) \wedge \\
& mac1 = MAC[SK](\hat{Y}.\hat{X}.YNonce.XNonce.CSL.CSS'.enc1) \wedge \\
& mac = MAC[SK](YNonce.XNonce.CSL.enc) \wedge \\
& \mathsf{VerifyMac}(Z, mac1, \hat{Y}.\hat{X}.YNonce.XNonce.CSL.CSS'.enc1, SK) \wedge \\
& \mathsf{FirstSend}(Z, YNonce, YNonce.\hat{Y}.CSL) \\
\Gamma_3 \equiv\ & \mathsf{Mac}(Z, \hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS.enc, K) \wedge \\
& \mathsf{Mac}(Z, \hat{X}.\hat{Y}.XNonce.YNonce.CSL.CSS'.enc', K) \supset CSS = CSS' \wedge enc = enc'
\end{aligned}$$

Formal Proof of $AUTH^{server}_{peer}$

$$\begin{array}{rll}
\mathbf{AA1} & [\mathbf{GPSK} : \mathbf{Server}]_S\ \mathsf{VerifyMac}(S, mac1, \hat{P}.\hat{S}.PNonce.SNonce. & \\
& \quad CSL.CSS.enc1, SK) & (1) \\
SEC^{server}_{pk,sk}, \mathbf{VMAC} & [\mathbf{GPSK} : \mathbf{Server}]_S\ \exists X.\ (\hat{X} = \hat{P} \vee \hat{X} = \hat{S}) \wedge & \\
& \quad \mathsf{Mac}(X, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS.enc1, SK) & (2) \\
\Gamma_1 & [\mathbf{GPSK} : \mathbf{Server}]_S\ \exists \eta.\ P_0 = (\hat{P}, \eta) \wedge & \\
& \quad \mathsf{Mac}(P_0, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS.enc1, SK) \wedge & \\
& \quad \mathsf{Receive}(P_0, msg1) < \mathsf{Send}(P_0, msg2) \wedge & \\
& \quad \mathsf{FirstSend}(P_0, PNonce, msg2) & (3) \\
\text{Inst } P_0 \longrightarrow P & [\mathbf{GPSK} : \mathbf{Server}]_S\ \mathsf{Mac}(P, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS.enc1, SK) \wedge & \\
& \quad \mathsf{Receive}(P, msg1) < \mathsf{Send}(P, msg2) \wedge & \\
& \quad \mathsf{FirstSend}(P, PNonce, msg2) & (4) \\
\mathbf{FS1} & [\mathbf{GPSK} : \mathbf{Server}]_S\ \mathsf{FirstSend}(S, SNonce, msg1) & (5) \\
\mathbf{FS2}, (-2, -1) & [\mathbf{GPSK} : \mathbf{Server}]_S\ (\mathsf{Send}(S, msg1) < \mathsf{Receive}(P, msg1)) \wedge & \\
& \quad (\mathsf{Receive}(P, msg1) < \mathsf{Send}(P, msg2)) \wedge & \\
& \quad (\mathsf{Send}(P, msg2) < \mathsf{Receive}(S, msg2)) & (6) \\
\mathbf{AA4} & [\mathbf{GPSK} : \mathbf{Server}]_S\ (\mathsf{Receive}(S, msg2) < \mathsf{Send}(S, msg3)) & (7) \\
(-2, -1) & AUTH^{server}_{peer} & (8)
\end{array}$$

Formal Proof of $AUTH^{peer}_{server}$

$\mathbf{AA1}$	$[\mathbf{GPSK:Peer}]_P$ $\mathsf{VerifyMac}(P, mac2, PNonce.SNonce.\hat{S}.CSL.enc2, SK)$	(9)
$SEC^{server}_{pk,sk}, \mathbf{VMAC}$	$[\mathbf{GPSK:Peer}]_P\ \exists X.\ (\hat{X} = \hat{P} \vee \hat{X} = \hat{S}) \wedge$ $\mathsf{Mac}(X, PNonce.SNonce.\hat{S}.CSL.enc2, SK)$	(10)
$\Gamma_2, (-1)$	$[\mathbf{GPSK:Peer}]_P\ \exists \eta.\ S_0 = (\hat{S}, \eta) \wedge$ $\exists CSS', enc1'.\ (\mathsf{Send}(S_0, SNonce.\hat{S}.CSL) <$ $\mathsf{Receive}(S_0, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1'.mac1) <$ $\mathsf{Send}(S_0, PNonce.SNonce.CSL.enc2.mac)) \wedge$ $mac1 = MAC[SK](\hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1') \wedge$ $mac = MAC[SK](PNonce.SNonce.CSL.\hat{S}.enc2) \wedge$ $\mathsf{FirstSend}(S_0, SNonce, SNonce.\hat{S}.CSL)$	(11)
Inst $S_0 \longrightarrow S$	$[\mathbf{GPSK:Peer}]_P\ \exists CSS', enc1'.\ (\mathsf{Send}(S, SNonce.\hat{S}.CSL) <$ $\mathsf{Receive}(S, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1'.mac1) <$ $\mathsf{Send}(S, PNonce.SNonce.\hat{S}.CSL.enc2.mac)) \wedge$ $mac1 = MAC[SK](\hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1') \wedge$ $mac = MAC[SK](PNonce.SNonce.\hat{S}.CSL.enc2) \wedge$ $\mathsf{VerifyMac}(S, mac1, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1', SK) \wedge$ $\mathsf{FirstSend}(S, SNonce, SNonce.\hat{S}.CSL)$	(12)
Inst $CSS', enc1'$, $\mathbf{VMAC}, \mathbf{MAC0}$	$[\mathbf{GPSK:Peer}]_P\ \exists X.\ (\hat{X} = \hat{P} \vee \hat{X} = \hat{S}) \wedge$ $\mathsf{Mac}(X, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1', SK)$	(13)
$\Gamma_1, \mathbf{AA1}, (-1)$	$[\mathbf{GPSK:Peer}]_P\ \mathsf{New}(X, PNonce) \wedge \mathsf{New}(P, PNonce)$	(14)
$\mathbf{AN1}, (-1)$	$[\mathbf{GPSK:Peer}]_P\ X = P$	(15)
$\mathbf{AA1}, (-3, -1)$	$[\mathbf{GPSK:Peer}]_P\ \mathsf{Mac}(X, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS'.enc1', SK) \wedge$ $\mathsf{Mac}(X, \hat{P}.\hat{S}.PNonce.SNonce.CSL.CSS.enc1, SK)$	(16)
$\Gamma_3, (-1)$	$[\mathbf{GPSK:Peer}]_P\ CSS' = CSS \wedge enc1' = enc1$	(17)
$(-2, -1)$	$[\mathbf{GPSK:Peer}]_P\ (\mathsf{Send}(S, msg1) < \mathsf{Receive}(S, msg2) < \mathsf{Send}(S, msg3))$	(18)
$\mathbf{FS1}$	$[\mathbf{GPSK:Peer}]_P\ \mathsf{FirstSend}(P, PNonce, msg2)$	(19)
$\mathbf{FS2}, (12, -2, -1)$	$[\mathbf{GPSK:Peer}]_P\ (\mathsf{Send}(S, msg1) < \mathsf{Receive}(P, msg1)) \wedge$ $(\mathsf{Send}(P, msg2) < \mathsf{Receive}(S, msg2)) \wedge$ $(\mathsf{Receive}(S, msg2) < \mathsf{Send}(S, msg3))$	(20)
$\mathbf{AA4}$	$[\mathbf{GPSK:Peer}]_P\ (\mathsf{Receive}(P, msg1) < \mathsf{Send}(P, msg2))$	(21)
$(-2, -1)$	$AUTH^{peer}_{server}$	(22)