Analysis of the 802.11i 4-Way Handshake

Changhua He
changhua@stanford.edu

John C Mitchell
mitchell@cs.stanford.edu

Electrical Engineering and Computer Science Departments
Stanford University, Stanford, CA 94305

ABSTRACT

802.11i is an IEEE standard designed to provide enhanced MAC security in wireless networks. The authentication process involves three entities: the supplicant (wireless device), the authenticator (access point), and the authentication server (e.g., a backend RADIUS server). A 4-Way Handshake must be executed between the supplicant and the authenticator to derive a fresh pairwise key and/or group key for subsequent data transmissions.

We analyze the 4-Way Handshake protocol using a finite-state verification tool and find a Denial-of-Service attack. The attack involves forging initial messages from the authenticator to the supplicant to produce inconsistent keys in peers. Three repairs are proposed; based on various considerations, the third one appears to be the best. The resulting improvement to the standard, adopted by the 802.11 TGi in their final deliberation, involves only a minor change in the algorithm used by the supplicant.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols

General Terms

Security

Keywords

WLAN, 802.11i, 4-Way Handshake, Denial-of-Service, Authentication, Key Management

1. INTRODUCTION

Wireless Local Area Networks (WLAN) [9, 10] provide important flexibility for college campuses, coffee shops, airports and other enterprises. Because they provide much higher transmission rates than current cellular systems, WLAN systems promise to be widely deployed in the coming years. However, security is a serious concern because the wireless medium is open for public access within a certain range.

In order to provide secure data communications over wireless links, the 802.11 Task Group proposed the Wired Equivalent Privacy (WEP) to encrypt the data stream and authenticate the wireless devices. However, significant deficiencies have been identified in both the encryption and the authentication mechanisms [3, 7]. To repair the problems in WEP without requiring additional hardware, the Wi-Fi Alliance proposed a Temporal Key Integrity Protocol (TKIP) to provide stronger security through a keyed cryptographic Message Integrity Code (MIC), an Extended IV space and a key mixing function. Furthermore, an authentication mechanism based on EAP/802.1X/RADIUS [1, 11, 17] has been developed to replace the poor Open System authentication and Shared Key authentication in WEP. As a long-term solution to securing wireless links, the latest IEEE standard 802.11i [12] was ratified on June 24, 2004. The Counter-mode/CBC-MAC Protocol (CCMP) provides data confidentiality, integrity and replay protection. The authentication process combines 802.1X authentication with key management procedures to generate a fresh pairwise key and/or group key, followed by data transmission sessions. However, 802.11i requires a WEP user to upgrade the hardware in order to use the strongest security mechanisms.

In this paper we analyze the 4-Way Handshake key management protocol in 802.11i, using a finite-state verification tool called Murφ. We find a significant and unnecessary Denial-of-Service (DoS) attack and investigate several possible repairs. We provided these basic attacks and repairs to the 802.11 TGi; the third repair in Section 5.3 was adopted. Although the 802.11i documentation was left unchanged so that the ratification would not be delayed, the repair will be added once the documentation is updated [Walker, email communication, June 30, 2004]. Note that there exist other possible DoS attacks against different layers of 802.11 networks, e.g., Physical Layer [5], MAC Layer [6] or upper layers [3]. However, in this paper, we focus on analyzing and improving the 4-Way Handshake; our repairs do not aim to prevent DoS attacks in other layers.

Through our formal verification process, we identify the functionality of each field in the handshake messages, achieve a simplified protocol that has as strong an authentication as the original one under our model, and identify some fields in the original protocol as potentially redundant. Our analysis results support and clarify passages in the 802.11i documentation, providing insights that are useful for understanding and/or implementing the protocol.

This paper is organized as follows. Section 2 describes the authentication mechanism of 802.11i and abstracts the 4-Way Handshake protocol for formal analysises. Section 3 explains the Murφ modeling of the protocol and summarizes our verification results. Section 4 analyzes the DoS attack in detail and discusses the practicality of the attack, based on the characteristics of 802.11b networks. Section 5 proposes several possible repairs and evaluates their effectiveness. Section 6 concludes the paper.

2. IEEE 802.11i

IEEE 802.11i [12] defines three data encryption algorithms: WEP, TKIP and CCMP, where WEP is included for backward compatibility, TKIP is the short-term solution to fix WEP problems, and CCMP is the long-term solution requiring additional hardware capabilities. However, in this paper we focus on the enhanced authentication protocols and do not investigate these data confidentiality protocols in any detail.

2.1 The 802.11i authentication

In order to provide better authentication and confidentiality in 802.11 networks than WEP, the standard defines a Robust Security Network Association (RSNA) based on IEEE 802.1X [11] authentication. The authentication process involves three entities, called the Supplicant, the Authenticator and the Authentication Server. Generally, a successful authentication means that the supplicant and the authenticator verify each other's identity and generate some shared secret for subsequent secure data transmissions. The authentication server can be implemented either in a single device with the authenticator, or through a separate server, assuming the link between the authentication server and the authenticator is physically secure. In infrastructure networks, the stations are supplicants, the Access Points are authenticators, and a separate RADIUS server may be used as an authentication server.

The complete process of an 802.11i authentication consists of handshakes between the supplicant and the authenticator (security capability discovery and 802.1X [11] conversations), between the authenticator and the authentication server (RADIUS de facto [17]), and between the supplicant and the authentication server (EAP-TLS de facto, with the authenticator serving as a relay [1]). After these handshakes, the supplicant and the authentication server have authenticated each other and generate a common secret called the Master Session Key (MSK). The supplicant uses the MSK to derive a Pairwise Master Key (PMK); The AAA key material on the server side is securely transferred to the authenticator to derive the same PMK in the authenticator. Alternatively, the supplicant and the authenticator may be configured using a static Pre-Shared Key (PSK) for the PMK. Further, during a re-association, a cached PMK can be used directly in order to reduce the computational load on the authentication server during repeated authentication requests from the same user.

Regardless of whether the PMK is derived from the EAP/802.1X/RADIUS handshakes, based on a PSK, or reused from a cached PMK, a 4-Way Handshake protocol must be executed for successfully establishing a RSNA. Following the establishment of the current PMK, this key management protocol confirms the existence of the PMK, the liveness of the peers, and the selection of the cipher suite; the protocol generates a fresh Pairwise Transient Key (PTK) for each subsequent session, synchronizes the installation of PTKs into the MAC, and in the case of multicast applications transfers the Group Transient Key (GTK) from the authenticator to the supplicants. After a successful 4-Way Handshake, a secure communication channel between the authenticator and the supplicant can be constructed for subsequent data transmissions, based on the shared PTK and/or GTK. The 4-Way Handshake may be repeated using the same PMK.

In this paper we focus on analyzing the 4-Way Handshake between the authenticator and the supplicant, after a shared PMK is achieved and before the data communication begins. For the purpose of analyzing the 4-Way Handshake, a shared PMK is assumed to be known only to the authenticator and the supplicant.

2.2 The 4-Way Handshake

Once a shared PMK is agreed upon between the authenticator and the supplicant, the authenticator may begin a 4-Way Handshake by itself or upon request from the supplicant. The message exchange is shown, at an abstract level, in Figure 1. S represents the Supplicant and A represents the Authenticator; SPA and AA, SNonce and ANonce, represent the MAC address and nonces of the supplicant and authenticator, respectively; sn is the sequence number; msg1, 2, 3, 4 are indicators of different message types; $MIC_{PTK}\{\}$ represents the Message Integrity Code (MIC) calculated for the contents inside the bracket with the fresh PTK. While MAC is commonly used in cryptography to refer to a Message Authentication Code, the term MIC is used instead in connection with 802.11i because MAC has another standard meaning, Medium Access Control, in networking.

[Message 1: A $\longrightarrow$ S]
AA, ANonce, sn, msg1

[Message 2: S $\longrightarrow$ A]
SPA, SNonce, sn, msg2, MIC_{PTK}\{SNonce, sn, msg2\}

[Message 3: A $\longrightarrow$ S]
AA, ANonce, sn+1, msg3, MIC_{PTK}\{ANonce, sn+1, msg3\}

[Message 4: S $\longrightarrow$ A]
SPA, sn+1, msg4, MIC_{PTK}\{sn+1, msg4\}

Figure 1. The idealized 4-Way Handshake protocol

The fresh PTK is derived from the shared PMK through a Pseudo Random Function with output length X (PRF-X), say, PTK = PRF-X(PMK, "Pairwise key expansion" ∥ Min{AA, SPA} ∥ Max{AA, SPA} ∥ Min{ANonce, SNonce} ∥ Max{ANonce, SNonce}), and divided into KCK (Key Confirmation Key), KEK (Key Encryption Key) and TK (Temporary Key). Note that the MIC is actually calculated with KCK, which is only part of PTK. However, we do not distinguish them here because this appears to be unrelated to the authentication process.

We ignore some fields of the messages in this abstracted version of the message exchange, primarily because such fields are not essential for authentication, although they could improve the security in some sense. First, in the original protocol, a PMKID is included in *Message 1* to indicate the corresponding PMK used in the handshake. This PMKID field can improve the security until it is transmitted in the wireless link for the first time; the details will be discussed in Section 4.3. Second, the RSN IE (Information Element) fields are included in *Message 2* and *Message 3* to negotiate the cipher suite and avoid a version rollback attack. Third, an encrypted GTK is sent together in *Message 3* in the case of multicast applications.

When the 4-Way Handshake protocol runs as intended, a communicating authenticator-supplicant pair execute exactly one run of the protocol and share one valid PTK after the handshake. The authenticator can refresh the PTK either periodically or upon the request from the supplicant by running another 4-Way Handshake with the same PMK.

The authenticator and the supplicant will silently discard any received message that has an unexpected sequence number or an invalid MIC. When the supplicant does not receive *Message 1* within the expected time interval after a successful 802.1X authentication, it will disassociate, de-authenticate and try the same or another authenticator again. Note that the supplicant does not use any other timeout during the 4-Way Handshake. On the other hand, the authenticator will timeout and retry the message if it does not receive the expected reply within the configured time intervals. Furthermore, the authenticator will de-authenticate the supplicant if it does not receive a valid response after several retries.

While these operations sound reasonable, packet loss must be taken into account, as well as the possibility of malicious messages from an attacker. While the authenticator can initialize only one handshake instance and accept only the expected response, the supplicant must accept all messages in order to allow the handshake to proceed. Hence, an attacker can easily interfere with the handshake protocol by inserting a forged *Message 1*. This leads to more severe vulnerabilities than might be expected. We find this vulnerability by Murφ modeling, with details discussed in the following sections.

3. Murφ MODELING

Murφ [8] is a verification tool that will exhaustively search all execution sequences of a nondeterministic finite-state system. Mitchell et. al. [14] successfully applied this tool to the verification of small security protocols, such as the Needham-Schroeder public key protocol, the Kerberos protocol and the TMN cellular telephone protocol. Subsequently, Mitchell et. al. [15] adopted a "rational reconstruction" methodology using Murφ to analyze the SSL 3.0 handshake protocol. In this paper we will use a similar methodology to analyze the 4-Way Handshake.

3.1 The Protocol Model

In order to use Murφ for verification of security protocols, we need to formulate a protocol model, add an attacker to the system, state the desired security properties, and run the protocol for some specific choice of system size parameters. The Murφ system uses explicit state enumeration to automatically check whether all reachable states of the model satisfy the given properties. A trace of messages will be output if any specified properties are violated, thus identifying the steps involved in any successful attack.

In our Murφ model, we consider the idealized 4-Way Handshake protocol in Figure 1. Because the PMK is assumed to be secure, the most important properties here are the PTK consistency and freshness. For simplicity, we assume that the cryptographic functions cannot be broken unless the key is disclosed.

The authenticator and the supplicant are programmed to follow the protocol. Each pair of authenticator and supplicant shares a PMK and tries to execute a given number of 4-Way Handshake sessions sequentially. The attacker is able to masquerade as any participant in the system by forging the MAC address. However, the attacker is assumed not to know the shared PMK of any pair of honest participants. The attacker can also eavesdrop on every message, remember nonces and MICs of each message, insert forged messages, and replay stored messages. Furthermore, the attacker can compose *Message 1* from stored nonces, and respond to every message with an arbitrary combination of known nonces and MICs. It is not obviously easy for the attacker to intercept and block delivery of a message transmitted over a wireless link; however, we assume the attacker is capable of doing so because any message might be lost in a wireless environment, and this has the same effect on the protocol.

The system size parameter indicates the number of authenticators, supplicants and attackers in the system and the number of sequential sessions each pair of participants can execute. We executed the Murφ model with different fields enabled in the message format, identified the functionality of each field, and achieved a simplified message format, keeping the same properties as the original one. The details will be discussed in Section 3.2. Furthermore, we found a DoS attack using *Message 1* that will block the protocol very easily. Note that although our verification process often reveals attacks, failure to find attacks does not imply that the protocol is completely secure, because the Murφ model may simplify certain details and is inherently limited to the configurations of the small number of entities and the capabilities of the attackers.

3.2 The Protocol Clarifications

Starting from the simplest message format (only nonces included), we increase the complexity of the messages until the complete protocol in Figure 1 is reached. For each different message format, the Murφ model checks all possible executions and finds the attacks that arise due to the absence of certain fields. Through this approach, we identify the functionality of each field in the message. We will not describe this process in detail. Instead, we summarize the outcome for specific fields, in some cases verifying the claims in the documentation of the standard, and in other cases revealing ineffective or redundant mechanisms.

First, the message flag, which is a combination of the *Key ACK*, *Key MIC*, and *Secure* bits in the *Key Information* field, is necessary and should be protected by the MIC field in the message. This flag makes *Message 1*, *2*, *3*, *4* distinguishable; otherwise, the attacker can easily use MICs in *Message 2* and *Message 3* to forge a valid *Message 4*, using *Message 2* to forge a valid *Message 3* or vice versa. Furthermore, the authenticator should only generate messages of type *1* and *3*; the supplicant should only generate messages of type *2* and *4*.

Second, nonces are used to make every message fresh and derive the fresh PTK. These should be generated in an unpredictable and globally unique way. Otherwise, the protocol might be vulnerable to replay attacks or pre-computation attacks. Fortunately, the proposed nonce generation algorithm in the standard appears to satisfy these requirements with high probability.

Third, the sequence number does not appear to be necessary for any security objectives in the 4-Way Handshake. Replay attacks are prevented by the freshness of nonces and PTKs. Furthermore, the sequence number does not provide any performance improvement because eventually the MIC field must be checked if the attacker modifies the sequence number to a valid value. Including the sequence number will provide minor performance improvement only when the attacker blindly replays messages without modifying the sequence number, or when messages arrive out of order. Hence, we consider this field to be largely redundant.

Fourth, the MAC addresses of the authenticator and the supplicant do not appear to be necessary for the authentication process. In particular, it may not be necessary to include these addresses in the PTK derivation. From the documentation, the MAC addresses are used to bind the PTK to the peers. However, by establishing a PMK successfully, the shared PMK has already bound all the following keys with the peers. If the PMK is based on a PSK shared by a group of users, the fresh nonces will bind the PTK to the peers. Including MAC addresses in the PTK derivation does not add any more security to the keys cryptographically.

Based on these clarifications, we achieve a simplified protocol with a simpler message format than the one in Figure 1. Under our Murφ model, the protocol shown in Figure 2 has the same authentication properties. Here the PTK is derived without AA and SPA, say, PTK = PRF-X(PMK, "Pairwise key expansion" || Min{ANonce, SNonce} || Max{ANonce, SNonce}).

[Message 1: A $\longrightarrow$ S]
ANonce, msg1

[Message 2: S $\longrightarrow$ A]
SNonce, msg2, MIC_{PTK}\{SNonce, msg2\}

[Message 3: A $\longrightarrow$ S]
ANonce, msg3, MIC_{PTK}\{ANonce, msg3\}

[Message 4: S $\longrightarrow$ A]
msg4, MIC_{PTK}\{msg4\}

Figure 2. The simplified 4-Way Handshake protocol

4. DoS ATTACK

In addition to verifying the functionality of each field in the message format, our Murφ model finds an attack on *Message 1*, easily causing PTK inconsistency between the authenticator and the supplicant. In this situation, protocol execution will be blocked, eventually leading to an authenticator timeout and a subsequent de-authentication of the supplicant. To avoid such an attack, the supplicant needs to keep all the received nonces and the corresponding PTKs, which ultimately leads to a DoS attack. Section 4.1 describes the attack in detail; Section 4.2 explains the inherent cause of the attack; Section 4.3 analyzes some limitations of the attack due to detailed implementations, and Section 4.4 illustrates the practicality of the attack.

4.1 The DoS attack

Because the attacker is capable of impersonating the authenticator, composing a *Message 1*, and sending to the supplicant, there is a simple one-message attack that causes PTK inconsistency, as shown in Figure 3. The attacker sends a forged *Message 1* to the supplicant after *Message 2* of the 4-Way Handshake. The supplicant will calculate a new PTK corresponding to the nonces for the newly received *Message 1*, causing the subsequent handshakes to be blocked because this PTK is different from the one in the authenticator. The attacker can determine the appropriate time to send out *Message 1* by monitoring the network traffic or just flooding *Message 1* with some modest frequency.

[Message 1: A $\longrightarrow$ S]
AA, ANonce, sn, msg1

[Message 2: S $\longrightarrow$ A]
SPA, SNonce, sn, msg2, MIC_{PTK}\{SNonce, sn, msg2\}

> **[Message 1': Attacker $\longrightarrow$ S]**
> AA, ANonce', sn, msg1
>
> \{The supplicant generates SNonce' and
> derives a new PTK' from SNonce' and ANonce'\}

[Message 3: A $\longrightarrow$ S]
AA, ANonce, sn+1, msg3, MIC_{PTK}\{ANonce, sn+1, msg3\}

> \{PTK and PTK' not consistent,
> MIC not verified, Protocol blocked\}

Figure 3. The one-message attack on the 4-Way Handshake protocol

This attack arises from the vulnerability of *Message 1*. In the 802.11i documentation [12] the designer seems to be aware of possible problems in *Message 1* and proposes the following solution to defend against the attacks. The supplicant stores both a Temporary PTK (TPTK) and a PTK, and updates TPTK upon receiving *Message 1*, updating PTK only upon receiving *Message 3* with a valid MIC. In this solution the attacker cannot drive the supplicant to change its shared PTK because it is not feasible to forge *Message 3*. However, this approach works only when different handshake instances (one between the supplicant and the authenticator, others between the same supplicant and the attacker) are executed sequentially; that is, the forged *Message 1* does not intervene between the legitimate *Message 1* and *Message 3* of the 4-Way Handshake. Obviously this will not prevent the problem shown in Figure 3 because the supplicant still cannot correctly verify the MIC in *Message 3* from the authenticator.

We can modify this approach slightly to store two possible keys in TPTK and PTK and try verifying the MIC in *Message 3* with either TPTK or PTK. This modification solves the problem in

Figure 3; however, the attacker can still cause problems for the supplicant side by sending out more forged messages with different nonces rather than only one. Therefore, in order to assure the handshake is non-blocking with the legitimate authenticator, the supplicant must use sufficient memory to store all the received nonces and the derived PTKs, until it finishes a handshake and obtains a legitimate PTK. Once the supplicant receives *Message 3*, it can use the PTK corresponding to the nonce in the message to verify the MIC. The derivation of PTKs might not lead to a CPU exhaustion attack because PTK calculations are not computationally expensive. However, a memory exhaustion attack always exists because the number of *Message 1*s can theoretically be unbounded. Though this memory exhaustion attack occurs on the supplicant side, which is not as severe as if it were the server, this is still a problem because it is quite easy for the attacker to forge and flood *Message 1*s.

4.2 Parallel Instances

Some may discount this DoS vulnerability as arising from insufficient modelling of the characteristics of wireless networks. Specifically, we have assumed that an attacker may intercept messages and possibly interfere with the delivery of messages. The following arguments show that the vulnerability arises instead from the need to engage in the parallel execution of multiple instances of the handshake protocol.

It is feasible for the authenticator (initiator) to have at most one active handshake in progress with each supplicant. The authenticator expects a correct response for every message it sends out. It can discard an unexpected response and retry the previous message or terminate the handshake if the expected response is not received during a given time interval and certain number of retries.

However, the supplicant (responder) cannot use a similar strategy. More specifically, if the supplicant is configured to be stateful and expects some specific message reply, packet loss or malicious messages from an attacker can cause deadlock and block the protocol. The following arguments are intended to clarify this statement. Assume that the supplicant discards unexpected messages in the intermediate stage of a handshake execution. Consider the case in which the supplicant accepts *Message 1* and sends out *Message 2*, but this message is lost. The authenticator will never get the expected response (*Message 2*); thus, it retries *Message 1* after a timeout. However, the supplicant will discard this retried *Message 1* because it is expecting a *Message 3*. On the other hand, an attacker can simply initialize a handshake by sending a forged *Message 1* to cause the supplicant to be blocked for the legitimate *Message 1* from the authenticator. Therefore, in the intermediate stage of a handshake, the supplicant must allow any *Message 1* to ensure the protocol to proceed.

The arguments above show that the supplicant must allow multiple handshake instances to run in parallel. In other words, the supplicant should allow *Message 1* at any stage. This makes the one-message attack and DoS attack unavoidable. However, some fields and mechanisms we omitted when abstracting the protocols can defend against the attack in some sense, although they cannot inherently eliminate the attack. The following section will analyze two limitations from the PMKID and the Link Layer Data Encryption.

4.3 Limitations

Our basic analysis is based on the idealized handshake protocol shown in Figure 1. In our Murφ model we have not considered the effect of including PMKID in *Message 1* or the involvement of Link Layer Data Encryption for subsequent handshakes.

A PMKID is included in *Message 1* and transmitted in clear text at the beginning of the 4-Way Handshake. It is calculated as PMKID = HMAC-SHA1-128(PMK, "PMK Name" ‖ AA ‖ SPA). With the assumption that PMK is secure, this PMKID is not disclosed to any attacker until the first time it is sent through vulnerable wireless links. This limits the attacker to a DoS attack only after the PMKID is seen in the link. When a PSK is configured for PMK, the attacker may learn the PMKID easily because the PMK, thus the PMKID, is unchanged for a substantial period of time. During a re-authentication process, the supplicant tries to use the cached PMK to communicate with the authenticator; it is possible for the attacker to know the corresponding PMKID earlier if the attacker keeps monitoring the network for some time (The same PMKID might be transmitted in the previous *Message 1*; at least the supplicant might include it in the re-association request message). In both cases, it is easy for the attacker to construct a forged *Message 1*. However, when the 802.1X authentication is used to establish a PMK dynamically, the PMKID will be different for every session; hence, the attacker cannot know the PMKID until it sees *Message 1* from the authenticator. As a summary, including PMKID in *Message 1* makes the attack more difficult, but it does not eliminate the attack. Instead of blindly flooding *Message 1*, the attacker has to read *Message 1* from the authenticator first, forge its own *Message 1* with the PMKID, and flood the messages.

When sequential 4-Way Hhandshakes occur under the same PMK, with the exception of the first handshake, all the following sessions are protected by the Link Layer Data Encryption. This mechanism can substantially improve the security of the protocol. All the handshake messages are transmitted in an EAPOL-Key format, which are encapsulated into data frames. Once the supplicant and the authenticator have some shared PTK, the following data frames will be protected by that PTK through encryption and authentication code. With the reasonable assumption that data encryption and MIC computation are cryptographically perfect, and replays can be detected, the attacker will not be able to intercept the transmissions. This mechanism ensures that the subsequent handshake sessions (with the same PMK) are secure except the first one. The attacker needs to catch up with the first 4-Way Handshake session and construct the attack. Obviously, it causes more difficulties for the attacker.

As a result, the Link Layer Data Encryption will limit the attacks to occur only before the first PTK is established, that is the first 4-Way Handshake protocol instance. Furthermore, when 802.1X is used to set up a PMK, the PMKID included in *Message 1* can limit the attacks to occur only after the first legitimate *Message 1* is seen in the wireless link. Combinations of the Data Encryption and the PMKID mitigate the vulnerability to only a limited duration; thus, the attacker has to interfere with the protocol in a more timely way. However, these mechanisms do not eliminate the inherent vulnerability of the protocol; the attack is still possible.

4.4 Practicality

When a PSK, or cached PMK, is configured to a current PMK, the attacker can interfere with the 4-Way Handshake either before *Message 1* or after *Message 1* because it knows the corresponding PMKID from previous eavesdropping. Therefore, the attacker may just send out a forged *Message 1* periodically, the attack succeeding if one of the forged *Message 1*s falls between the legitimate *Message 1* and *Message 3*. Even when 802.1X authentication is used to establish the PMK dynamically, the attacker can still construct the attack after seeing the legitimate *Message 1* from the authenticator. The attacker can succeed with a memory DoS attack by increasing the frequency of sending forged *Message 1*s. In any case, this is different from a trivial network jamming or frequency jamming, and it is hard for the administrator to eliminate the attack because the attacker sends regular messages in a regular way. The following calculations indicate why this attack is practical and how it is different from a network jamming.

Assume that a basic *Message 1* is sent (only PMKID included in the *Key Data* field of the frame) through 802.11b networks [10]. A MSDU, consisting of 30 bytes MAC header, 6 bytes EAPOL header, 117 bytes EAPOL-Key frame, and 4 bytes checksum are given to the Physical Layer to transmit. Consider the short PLCP frame and an 11Mbps data rate, the DSSS preamble and header are transmitted in 96 µs, the data are transmitted in 114 µs. Even more count in DIFS (50µs), SIFS (10µs) and ACK (96µs + 10µs), *Message 1* is sent and ACKed in 376 µs. Assume that a timeout in the authenticator is set to the default value (100 ms). If the attacker has a full control on the Network Interface Card (NIC), and no random backoff time is inserted between two consecutive messages, a total of 265 *Message 1*s can be sent. Even if the random backoff time is included, on average 310 µs, it is still possible for the attacker to send about 145 messages. This number may be much larger if the timeout is set to a larger value, if we take into account the multiple retry times, and if the network accommodates a higher data rate like 54Mbps in 802.11a/g networks. Among the possible number of *Message 1*s, the attacker only needs one of them to reach the supplicant in order to block the protocol. Moreover, the large number of messages is sufficient for launching a DoS attack on the supplicant side in general.

5. EFFECTIVE DEFENCES

This kind of DoS attack also exists in some other protocols where the responder needs to store states, i.e. IKE [2, 13] and TCP [16, 18]. We can simply repair it by transmitting both ANonce and SNonce in *Message 3*; this improves the protocol to a stateless one [2, 4, 13]. However, even with that improvement, the 4-Way Handshake might be still vulnerable to replay attacks. On the other hand, once some mechanism is implemented to defend against the replay attack, our repairs in Section 5.2 can already make the handshake secure. Furthermore, adding ANonce to *Message 3* significantly change the format of the packet. Hence, we do not suggest this approach. Instead, we propose three other approaches that do not require significant modifications of the packet format or the protocol itself.

5.1 Random-Drop Queue

The problem here is similar to the well-studied TCP SYN flooding DoS attacks [16, 18], which can be mitigated in some known ways. The supplicant can keep a queue of all the initiated, but incomplete, handshake instances. According to the calculations in Section 4.4, the queue size might be too large for the supplicant; the situation becomes even worse if a longer timeout period or a higher data rate is implemented. Therefore, a feasible improvement would be to implement the queue with a random-drop policy. The supplicant maintains a certain size of queue, say, Q entries to store the states. Once all entries in the queue are filled, one of them is randomly replaced by the new state. Denote that the number of malicious *Message 1*s is n between the legitimate *Message 1* and *Message 3*, the probability that the handshake will be blocked by the malicious messages is P, we have $P = 1 - \left(1 - \frac{1}{Q}\right)^n$, as shown in Figure 4.

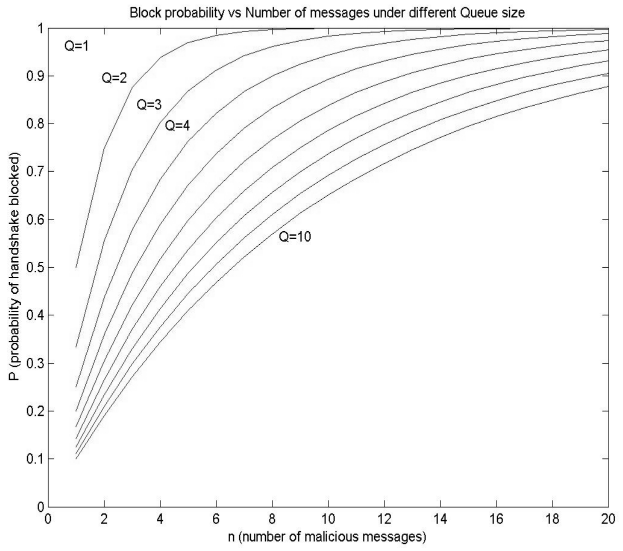

Figure 4. Effectiveness of random-drop queue

From Figure 4, when $Q = 1$, the attacker can block the handshake with probability 1 by inserting only one message. When Q increases, the attacker needs to insert more messages in order to block the handshake with a high probability. However, increasing Q could be quite expensive and performance reductive for the supplicant. Furthermore, even a queue of size 10 is not going to help very much because the attacker can block the handshake with probability above 0.8 by inserting 16 messages. That is a trivial number of messages, compared to the total possible number of *Message 1*s the attacker can insert between the legitimate *Message 1* and *Message 3*.

5.2 Message 1 Authentication

Since there is already some common secret (PMK) shared between the authenticator and the supplicant, another possible repair is to add a MIC to *Message 1*, which will prevent the attacker from forging that message. In order to exploit the same hardware or software as in processing other messages, we can derive a trivial PTK based on the PMK and some specific values of nonces (e.g., 0), then calculate the MIC with this derived PTK. Note that after a MIC is added, *Message 1* and *Message 3* are still distinguishable by the *Secure* bit.

If the PMK is dynamically generated through an 802.1X authentication process, this would solve the problem. However, if a PSK or a cached PMK is used for the current PMK, the authenticated *Message 1* is still vulnerable to replay attacks since the PMK is static for a relatively long time. Therefore, the authenticator should keep a monotonically increasing sequence counter to defend against the replay attacks. One global sequence counter per authenticator appears to work for all supplicants. The supplicant can detect the replayed messages by comparing the counter of a received message against the counter of the largest-numbered previous message.

Fortunately, the requirement that the counter must be monotonically increasing appears feasible since there are apparently 8 octets set aside for this sequence counter. In fact, there appears to be sufficient space in the message format so that clock time could be used as the counter value, eliminating the possible problem of counter rollover. Furthermore, this specific sequence counter is also consistent with its usage in the group key handshakes and imposes no significant influences on other parts of the standard. Note that we need not worry about the synchronization of the clock time since only the local time in the authenticator side is used.

5.3 Nonce Re-use

The third repair is to eliminate the intermediate states on the supplicant side. Specifically, the supplicant can re-use the values of SNonce until a legitimate handshake is completed and a shared PTK is achieved between the supplicant and the authenticator. In other words, the supplicant does not update its nonce responding to each received *Message 1* until *Message 3* is received and verified. Note that there are no requirements for the authenticator to re-use the values of ANonce, because the legitimate ANonce will ultimately reach the supplicant via a valid *Message 3*.

In this approach the supplicant only needs to remember one SNonce of its own, which eliminates the memory DoS attack. Although it is still possible for the attacker to send out forged *Message 1*s with different nonces, the supplicant need not store every received ANonce and the corresponding PTK. It merely derives a PTK from the stored SNonce and the received ANonce, then computes a MIC from the derived PTK and sends out the corresponding *Message 2*. Upon receiving *Message 3*, the supplicant will again derive a PTK from the stored SNonce and the received ANonce, then verify the MIC using the derived PTK. Once the MIC is verified, *Message 4* is sent out and the corresponding PTK can be used as the session key.

This approach is a robust solution to the memory exhaustion attack; however, it uses more computation on the supplicant side. Specifically, the PTK is calculated twice for each received nonce: the first time when *Message 1* is received, and the second time when *Message 3* is received. If the computation power is poor for some devices, flooding *Message 3* might cause a CPU exhaustion attack, or substantially decrease the performance because the supplicant needs to re-compute the PTK first, then verify the MIC.

In practice, the CPU load of calculating the PTK varies considerably depending on the implementation and the type of CPU [Moore, email communications, May 11, 2004]. Generally, the PTK calculation is about 1.5 times slower than the calculation of MIC. However, the calculation of PTK can be improved about 4~5 times by merging the loops and pre-calculating intermediate results; the MIC calculation can be improved about 1.5~2 times by caching intermediate results. Hence, the PTK calculation does add to the CPU load, but not as much as another MIC calculation.

Of course, the supplicant can store all the received nonces and the derived PTKs to handle the computation load, but then obviously the memory exhaustion attack recurs. There is a tradeoff here that the supplicant needs to make between the memory consumption and the CPU consumption. If the environment is such that most of the messages are expected to be legitimate, the supplicant can store one copy of the derived PTK and received ANonce, and use them to verify the MIC in received *Message 3* directly. The supplicant re-computes the PTK only if the nonce in the message does not match the stored ANonce. This combined approach seems to be the most reasonable solution to the 4-Way Handshake problems.

6. CONCLUSIONS

The 4-Way Handshake protocol in IEEE 802.11i has been analyzed using Murφ. We identify the functionality of each field in the messages, in some cases supporting assertions made in the protocol documentation, and in a few cases suggesting alternatives. A simplified protocol is presented that has the same authentication properties as the original one under our Murφ model. Most significantly, we find and analyze an effective DoS attack on *Message 1* in the protocol. As in many other case studies, this attack can be prevented by an extremely simple modification to the protocol. However, this clear improvement in the protocol was not apparent before our security analysis.

The protocol is supposed to allow only one active handshake at any time and to generate a shared PTK between a corresponding supplicant and authenticator. However, upon analysis we show that the supplicant must allow multiple handshakes to execute in parallel, in order to ensure protocol completion in the presence of packet loss. This leads to vulnerabilities, allowing an attacker to block the handshake by simply inserting one forged message. Furthermore, the attacker can construct a memory DoS attack if the supplicant is implemented to store the states of all incomplete handshakes. When 802.1X authentication is implemented, the PMKID included in *Message 1* will limit the attacker to constructing this attack only after the first *Message 1* is seen in the link; Link Layer Data Encryption can protect the subsequent sessions after the first PTK is established. When both of these mechanisms are implemented, this attack can only be performed between *Message 1* and *Message 3* of the first legitimate 4-Way Handshake instance. These implementations cause more difficulties, but the attacker can still launch the attack if it keeps monitoring and intercepting the network in a timely way. However, since the attack can be prevented simply and completely, there is no need to live with a protocol subject to this attack.

We discuss and compare three repairs. First, we could use a random-drop queue of a certain size to avoid memory exhaustion without any modifications to the protocol. However, our calculations indicate that the protocol remains quite vulnerable with reasonable queue sizes. Second, with a more significant change, a MIC calculated from the PMK can be added to *Message 1* to prevent the attacker from forging *Message 1*. This remedy also requires a monotonically increasing sequence counter to be implemented in the authenticator side to prevent replay attacks. The local clock time of the authenticator appears to be a simple increasing counter that would do the job. Third, without any modification to the protocol itself, we can simply re-use the nonces in the supplicant until one legitimate 4-Way Handshake is completed. This approach inherently eliminates the vulnerability but might consume more computation power in the supplicant. We suggest the combined approach that re-uses nonces and stores one entry of the received nonce and derived PTK.

7. ACKNOWLEDGEMENTS

We thank Jesse Walker for his interest and feedback. We are indebted to Tim Moore, who provided many valuable comments and extensive discussion of 802.11i and the 4-Way Handshake protocol.

8. REFERENCES

[1] Aboba, B., and Simon, D. PPP EAP TLS authentication protocol. *RFC 2716*, October, 1999.

[2] Aiello, W., Bellovin, S. M., Blaze, M., Ioannidis, J., Reingold, O., Canetti, R., and Keromytis, A. D. Efficient, DoS-Resistant, Secure Key Exchange for Internet Protocols. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 48-58, Washington D. C., USA, 2002.

[3] Arbaugh, W. A., Shankar, N., and Wang, J. Your 802.11 Network has no Clothes. In *Proceedings of the First IEEE International Conference on Wireless LANs and Home Networks*, pages 131-144, December, 2001.

[4] Aura, T., and Nikander, P. Stateless Connections. In *Proceedings of International Conference on Information and Communications Security (ICICS'97)*, pages 87-97, Beijing, November, 1997.

[5] AusCERT AA-2004.02. Denial of Service vulnerability in IEEE 802.11 wireless devices. May 13, 2004. Available at http://www.auscert.org.au/render.html?it=4091.

[6] Bellardo, J., and Savage, S. 802.11 Denial-of-Service attacks: real vulnerabilities and practical solutions. In *Proceedings of the USENIX Security Symposium*, pages 15-28, August, 2003.

[7] Borisov, N., Goldberg, I., and Wagner, D. Intercepting mobile communications: the insecurity of 802.11. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, Rome, Italy, July, 2001.

[8] Dill, D. L. The Murφ verification system. In *the 8th International Conference on Computer Aided Verification*, pages 390-393, July, 1996.

[9] IEEE Standard 802.11-1999. Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control and Physical Layer Specifications. 1999.

[10] IEEE Standard 802.11b-1999. Higher-Speed Physical Layer Extension in the 2.4 GHz Band, Supplement to IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. September, 1999.

[11] IEEE Standard 802.1X-2001. IEEE Standard for Local and metropolitan area networks – Port-Based Network Access Control. June, 2001.

[12] IEEE P802.11i/D10.0. Medium Access Control (MAC) Security Enhancements, Amendment 6 to IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 11: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications. April, 2004.

[13] Matsuura, K., and Imai, H. Modified Aggressive Mode of Internet Key Exchange Resistant against Denial-of-Service Attacks. *IEICE Transactions on Information and Systems*, Volume E83-D(5), pages 972-979, May, 2000.

[14] Mitchell, J. C., Mitchell, M., and Stern, U. Automated Analysis of Cryptographic Protocols Using Murφ. *IEEE Symposium on Security and Privacy*, pages 141-151, 1997.

[15] Mitchell, J. C., Shmatikov, V., and Stern, U. Finite-State Analysis of SSL 3.0. *Seventh USENIX Security Symposium*, pages 201-216, 1998.

[16] Ricciulli, L., Lincoln, P., and Kakkar, P. TCP SYN Flooding Defense. In *Communication Networks and Distributed Systems Modeling and Simulation (CNDS'99)*, 1999.

[17] Rigney, C., Willens, S., Rubens, A., and Simpson, W. Remote Authentication Dial In User Service (RADIUS). *RFC 2865*, June, 2000.

[18] Schuba, C. L., Krsul, I. V., Kuhn, M. G., Spafford, E. H., Sundram, A., and Zamboni, D. Analysis of a Denial of Service attack on TCP. *1997 IEEE Symposium on Security and Privacy*, 1997.