802.11 Key Management Series: Part I: Key Management for WEP and TKIP

By Jesse Walker
Agenda and Series Roadmap
This is the first of two articles surveying the infrastructure used to provide cryptographic session keys to secure IEEE 802.11 wireless LANs (WLANs) [1]. This article discusses how WEP (Wired Equivalent Privacy) keys have been managed traditionally and how keys will be managed for the WEP patch called TKIP (Temporal Key Integrity Protocol). Previous articles describe WEP and TKIP. Next month’s installment describes a longer-term solution to the keying problem.

This article serves two purposes. First, it reports on the progress of the IEEE 802.11 Working Group toward resolving the security problems in the 1999 standard. More importantly, it can help organizations evaluate the risks inherent in deploying emerging products based on TKIP technology.

What is 802.11? What is WEP? What is TKIP?
802.11 is a Wireless Local Area Network (WLAN) standard created by the IEEE. The IEEE Standard 802.11-1999 specifies a suite of protocols defining an Ethernet-like communication channel using radios operating over unlicensed radio spectrum bands instead of wires. Since 802.11 uses radio instead of cables as its communication medium, anyone with a radio receiver can eavesdrop on the messages exchanged, and anyone with a radio transmitter can write to the channel. Because of this, it is essential to build security into the WLAN design. The 1999 standard accommodates this need by defining a security protocol called Wired Equivalent Privacy, or WEP. The goal of WEP was to provide the same level of security as a wired Ethernet.

It is now well known that the WEP design contains many basic flaws that render it ineffective at meeting its design goals. It fails to meet its design goals because it does not take steps to defend against packet forgery or replay, allowing an attacker to use the 802.11 infrastructure to launch attacks on the WEP encryption key. Furthermore, the WEP design does not use encryption properly, allowing both the encrypted data and the encryption key to be recovered using public domain software.

In response, the IEEE 802.11 Working Group (WG) has charted work to correct these design flaws. The approach taken is two-fold. First, the WG has defined the Temporal Key Integrity Protocol. TKIP is a WEP wrapper, designed to be deployed as a software patch on existing hardware. Given proper key management, TKIP fixes all the known problems with WEP, but still cannot provide security assurances in line with the original WEP design goals. TKIP cannot do any better than it does, because the available CPU cycles on existing hardware are too limited to allow all of the original 802.11 requirements to be met. Second, the 802.11 WG is defining a new protocol based on the Advanced Encryption Standard (AES) that can meet the original design goals, but will require new hardware.

Key Management 101
The care and feeding of cryptographic keys is called key management. Proper key management is the bedrock for any cryptographic system. Cryptographic solutions will fail to meet their objectives unless proper care is taken to keep cryptographic keys from unauthorized parties. To understand why this is so requires a little background.

All encryption schemes used in practice are “probabilistic.” This means that the encryption scheme folds some amount of “extra” public data into the underlying encrypted data to randomize the encrypted result. Encryption does not meet its design goal without this “extra” data. As an example, consider the naïve use of encryption versus a “probabilistic” scheme. Let $E_k(\cdot)$ denote encryption under the encryption key $K$. A naïve use of the encryption primitive $E_k$ would transform the plaintext values $A$ and $B$ into the ciphertext values $E_k(A)$ and $E_k(B)$, respectively. An attacker can immediately learn a great deal of information about $A$ and $B$ when they are encrypted in this way; the attacker can, for instance determine whether $A$ and $B$ are the same values, because $E_k$ always produces the same output given the same input! This property stems from the fact that the encryption function $E_k$ has to produce a result that is predictable to anyone possessing the key $K$, in order to make recovery of the plaintext from the ciphertext possible. Through frequency analysis and correlation of enough encrypted data, an attacker can often deduce large portions of the plaintext data when encryption is used in this mode.

Many strategies exist to eliminate this necessary “defect” in the naïve use of any encryption algorithm. One method is to XOR random data to the plaintext prior to encryption:

$$\text{Ciphertext} \leftarrow E_k(\text{Plaintext} \oplus \text{Random})$$

where “$\oplus$” denotes the exclusive OR operation. Another strategy maintains a counter whose value is encrypted to represent a one-time pad, then XORed with the plaintext to produce ciphertext, and then the counter is incremented:

$$\text{Ciphertext} \leftarrow E_k(\text{Counter}) \oplus \text{Plaintext}, \quad \text{Counter} \leftarrow \text{Counter} + 1$$

In both cases, the “extra” data—the random data in the first case and the counter value in the second—is communicated as public data from the encryptor to the decryptor in order for the latter to recover the plaintext from the ciphertext. Both of these schemes and many like them are known to solve the data leakage problem that occurs with the naïve use of encryption. If the same plaintext is presented multiple times for encryption, the resulting ciphertext differs each time with these schemes. All these schemes, however, lose their efficacy if the “extra” public data is repeated with the same key. Reuse of the “extra” public data that randomizes the encryption scheme reduces the scheme to the naïve one, so an attacker can resume frequency analysis and correlation.

Thus, the first requirement for an encryption scheme is that the pair <key $K$, “extra public data”> never is reused. And the word “never” really means never. History affords many examples where catastrophe resulted from violating this cardinal rule even once. For instance, in 1942 the KGB reused codebooks of one-time pad; this allowed the U.S. VERONA project to decode intercepted messages and expose the extent of Soviet espionage in the United States. Any system using encryption must therefore be assumed to fail to meet its confidentiality objectives if its design does not demonstrably prevent key pair reuse.

In practice, the implication of this is that a system using encryption needs a scheme to create fresh keys periodically, with the update period bounded by the expected time the system requires to first reuse the “extra” public data. Here “fresh” means a previously unused key, a key unrelated to any other key used previously. The new fresh key must be unrelated to a prior key, because if the prior key were compromised, an attacker could immediately compromise the new key if its relationship to the compromised key were known. Key management systems...
WEP Keying

For its “extra” public data, the WEP encryption scheme uses an initialization vector (IV). In the WEP scheme, each packet combines an IV with a base key to construct a per-packet key.

The original 1999 802.11 standard leaves key management undefined. Because of this, its design does not demonstrably prevent reuse of <key, initialization vector> pairs, so this implies a priori that WEP will fail to meet its data confidentiality goals, without any further analysis. This may seem like an invalid and harsh judgment, as vendors could define their own key management schemes to fill the void left by the standard. However, it is by now well understood that proprietary protocols in general do not work in distributed environments, because it is ordinarily impractical to guarantee that all the equipment is manufactured by the same vendor. In the case of WEP, the lack of a standard key management protocol dooms its goal of providing data confidentiality.

It is worth digging a little deeper into WEP, to work through the implications of its lack of a standardized key management model.

Manual configuration of keys can be costly. Precisely the same key bit pattern must be inserted into both the encryptor and decryptor, or decryption will fail. Entering a key manually tends to be error prone, because of the human tendency to commit typing errors, leading to a time-consuming process that has to be manually verified on a station-by-station basis as well. Because of this, human administrators tend to use keys with recognizable patterns instead of random keys, making attack still easier. And because 802.11 stations are mobile, administrators tend to deploy the same key for a station at every access point! The real situation is much worse than this suggests, in that WEP mandates only the use of group keys, and most vendors’ equipment does not support per-station keys, meaning all stations and access points within a deployment use the same key. This has disastrous consequences for WEP’s confidentiality claims.

The WEP IV space is 24-bits wide. This means there are only $2^{24}$ WEP IVs available, so to meet the requirement of no <key, IV> pair reuse, an administrator must change the WEP key after $2^{24}$ packets in the best circumstances. Reality is much less kind, however, because 802.11 does not define how to select WEP IVs, either. Implementers are instead left to fend for themselves, and cost constraints have prevented vendors from applying very much ingenuity to the problem.

The most common WEP IV selection strategy is for each system to use its WEP IV space as a counter, initialized to zero at association time and incremented by one after each packet. However, this strategy guarantees information leakage after the first exchange of packets between a station and an access point, because both are protected by the same <key, IV> pair!

A less common strategy is to select the WEP IV at random. This is the optimal strategy for maximizing the lifetimes of WEP keys. However, it suffers from the problem that the chance that two randomly selected 24-bit values will be the same is already 50% after only $O(2^{12})$ samples. But $2^{24}$ packets represents only a few seconds of WLAN traffic in the best of circumstances, meaning WEP’s design restricts WLANs to very small deployments with a staff of full-time administrators constantly updating the keys on every machine, at least when maintaining confidentiality is a goal. This simply will not do.

TKIP Keying

Background on TKIP

For this discussion, we don’t really need to understand what TKIP is or how it works; we only need to know that it addresses all the known deficiencies in WEP, when suitably coupled with key management. Readers need to grasp only two basic facts about TKIP.

The first fact is that TKIP does not use cryptographic keys directly. Instead, it manufactures different keys for each direction of communications over each link. TKIP accomplishes this by mixing the address of the transmitter into each encryption key, so that every station ends up with its own encryption key, even if each starts with the same key. The receiver repeats the mixing of the alleged translator into the key prior to using it to decrypt a packet received over the link. This means that every station could in principle use the full WEP IV space prior to rekeying, since no <key, IV> pair collisions result from normal use. This by itself represents a vast increase in confidentiality over WEP—on the basis of <key, IV> pair collisions alone, TKIP keys can be effective for minutes or hours instead of seconds before their use compromises the data being sent.

The situation is actually more complex than this suggests. The WEP per-packet key construction is weak, so the per-packet WEP key can be attacked directly. In order to enjoy the benefits of collision free operation among <key, IV> pairs, TKIP replaces the WEP per-packet key construction with a stronger one.
The second fact is that TKIP attempts to restrict access to the wireless link to authorized users. It accomplishes this by inserting a message integrity code (MIC) into each packet. The MIC can be thought of as a species of cryptographic Cyclic Redundancy Check (CRC), and it is constructed to detect attempts to forge packets. The MIC also increases the efficacy of the WEP encryption underlying TKIP, by making it computationally infeasible for an attacker to use the WEP infrastructure itself as a decryption oracle. However, this concern with authenticity opens the door to new and pressing security questions that were undecidable within the context of WEP. Being able to distinguish authorized packets from forged packets is equivalent to knowing who possesses the cryptographic keys. This means the TKIP cryptographic keys should be tied somehow to authentication, so that the parties in the communication know that received packets are genuinely authorized. That is, the use of a MIC transforms a cryptographic key into an authorization token. The use of the key to protect a message should prove that the sender was the party authorized to do so and not someone else. This linking of authorization to keying did not happen by accident; it was one of the TKIP design goals.

**Background on 802.1X**

The 802.11 security enhancements have made two attempts at TKIP keying, both based on another IEEE protocol called 802.1X [2]. The first attempt failed, while the second is a partial success, allowing early TKIP deployment in some environments. This record of repeated failure is not unusual for security protocols, which are difficult to get right the first, second, or even third time. It is a pattern unusual for a standards process, however, where the norm is compromise on features proposed by various vendors to reach consensus on a 75% solution. However, 75% security solutions do not meet the requirements because they are still insecure, just as one cannot make a submarine watertight by closing three-quarters of the hatches.

Both of the attempts discussed in this article also demonstrate how supremely difficult it is to move a concrete security mechanism from its native environment to a new one. If the new environment does not satisfy all the implementation assumptions made about the native environment, the mechanism will be compromised in the new environment.

IEEE Standard 802.1X-2001 aspires to define port-based access control in 802 LANs, including WLANs, and it defines a key distribution service implemented by its EAPOL Key message. EAPOL stands for EAP on LAN, where EAP itself is an acronym for the Extensible Authentication Protocol, defined in RFC 2284 [3]. EAP is a misnomer, since it is not an authentication protocol per se, but rather a transport protocol tailored to the needs of authentication mechanisms. EAP was also designed to be easily encapsulated within any data link protocol. It provides a “plug-in” architecture for concrete authentication mechanisms, to allow their use over an arbitrary data link.

IEEE 802.1X/EAP architecture is very simple, which is one of its technical strengths. It uses three classes of devices, called Supplicant, Authenticator, and Authentication Server. In order to access a network, a Supplicant first attaches to an Authenticator. In this newly attached state, the Supplicant has no network access per se, but can send messages over its attachment to the Authenticator. The Authenticator will relay messages from the Supplicant only to the Authentication Server. This allows the Supplicant and Authenticator to execute the EAP protocol between them, which is used to transport some concrete authentication protocol. Example concrete authentication protocols include the MD5 Challenge [3] and the TLS [4] handshake. The transport for EAP itself is the 802.1X protocol over the 802 link between the Supplicant and the Authenticator. Between the Authenticator and Authentication Server, the EAP transport can be any suitable protocol, but is typically RADIUS over UDP/IP.

Figure 1 below depicts the 802.1X architecture:

![802.1X architecture](image)

**Figure 1—802.1X architecture overview**

The Authentication Server uses the EAP protocol to authenticate the Supplicant, as Figure 2 depicts. The exchange begins with an EAP-Identity message, asking the Supplicant to identify itself. The Authenticator passes the Supplicant’s response to the Authentication Server, which picks the correct concrete authentication protocol for authenticating this Supplicant. The concrete authentication protocol will require one or more request/response exchanges, resulting in an authentication decision. If the authentication fails, the Authentication Server notifies the Authenticator, who drops the Supplicant’s attachment. If this authentication succeeds, the Authentication server notifies the Authenticator to grant access to the Supplicant. In response, the Authentication server changes the state of the Supplicant’s attachment point to permit general  

---

2 MD5 stands for Message Digest 5  
3 TLS stands for Transport Layer Security
message traffic onto the network. It then forwards an EAP-Success message to the Supplicant to signal success. The Authenticator may also send an EAPOL key message to the Supplicant, relaying information about a cryptographic key to use to protect the subsequent session. To apply the 802.1X architecture to an 802.11 network, the station plays the role of the Supplicant, and the access point the role of the Authenticator.

![Typical 802.1X message exchange](image)

**Figure 2—Typical 802.1X message exchange**

**Attempt 1: Keying based on 802.1X-2001**

The first attempt to define 802.11 key management was to use the 802.1X model as is, without any modification. This failed due to several problems.

Problem number one is that the 802.1X model is asymmetric, in that the Authentication Server authenticates the station but not vice versa. This is a network-centric model stemming from service providers' use of RADIUS as a mechanism to fulfill their legal obligation to identify the parties they bill. It presupposes that the Supplicant can reliably determine when it is talking with the “real” network. This is simply infeasible in a wireless LAN, where it is easy to launch a “man-in-the-middle” attack. A man-in-the-middle attack is launched by a rogue device $R$ sitting between two legitimate devices $A$ and $B$, with $R$ masquerading as $A$ to $B$ and as $B$ to $A$—in 802.11 circles, this is often called the “access point in the parking lot” problem. The old RADIUS-based scheme can provide security for classical remote access because the telephone provider invests many resources to guarantee that the phone that picks up is the number you dial—and they assiduously monitor for intrusions and prosecute anyone who attempts to redirect phone calls. These presuppositions do not apply in a wireless LAN.

Two methods have been proposed to adapt 802.1X in wireless environments to protect against man-in-the-middle attacks. Method one sets up an Authentication Server internal to the station and a Supplicant in the box labeled Authentication Server. Authentications occur in both directions. This method does not work for two reasons. The first is practical, in that real access point and authentication server implementations have never been constructed to respond to messages initiated by stations, so the solution could not be fielded without first replacing all the deployed infrastructure. The second problem is that this type of solution is cryptographically naïve and cannot defeat man-in-the-middle attacks, even in principle. The only way known to accomplish this goal is by cryptographically linking the authentication handshakes. However, definition there is no such linkage between two independent authentications.

The other proposal for adapting the 802.1X authentication model to 802.11 is to restrict the concrete authentication protocols to those designed to accomplish mutual authentication and detect man-in-the-middle attack. The TLS handshake is an example of such a protocol, while the MD5 Challenge is not. This approach can be made to work.

A second aspect of the model related to keying is that 802.1X never specifies how the key distributed by the EAPOL key message is tied to authentication, so the relationship between the 802.11 key and authentication is clear neither theoretically—important for understanding the security of the overall system—nor practically. One implication of this is that key management as specified by the 2001 802.1X standard does not satisfy the needs of the TKIP authorization model. In particular, since the TKIP key is not tied to the authentication, it cannot be reliably asserted that use of the TKIP key demonstrates authorization to use the WLAN channel. A second implication is that when combined with the next concern, a race condition between the EAPOL key message and the EAP Success message, the 802.1X authorization model crumbles altogether.

802.1X has no means to cryptographically protect the EAP Success message, so it can be forged in any environment where the attacker has transmit access to the physical medium. Of course, this is always true in a wireless environment, and this defect in the 802.1X design enables a special man-in-the-middle attack. The attacker can wait for the station’s final message and then transmit an EAP Success message. This can fool the station into thinking it is safe to transmit packets prior to receiving the EAPOL key message. In particular, this premature forged Success message may convince the station to send and receive unprotected packets, providing the attacker with an opportunity to break into the station or read what should be confidential data. The bottom line is that 802.1X has no reliable means of signaling when it is safe to open the data link in a WLAN.
A final problem is that the EAPOL Key message defined in the 802.1X standard uses the same encryption scheme as does WEP to protect the distributed key. This leaves the key vulnerable to the same tools that attack WEP. Since exposure of the key compromises the data link consuming it, the EAPOL Key message format makes all of the other TKIP efforts a farce.

**Attempt 2: Revised keying based on 802.1X**

Since the first attempt at key management undercut all of the other efforts to shore up 802.11 security, the design has been revised extensively [5]. This revision attempts to address the problems discovered in the original design. The first problem is how to defeat man-in-the-middle attacks against 802.1X-based authentication. The industry has responded to this challenge by inventing two hybrid mechanisms called Tunneled TLS (TTLS) [6] and Protected EAP (PEAP) [7]. PEAP and TTLS are very similar and based on the same idea. Both use two serial authentications—i.e., one authentication, coupled temporally to a second—with the first being the TLS handshake. The TLS handshake is performed first because it can protect itself from attack, and it magically produces a cryptographic key that can protect the second authentication and subsequently as a session key. The second authentication is a legacy method, to identify the station to the Authentication Server. Because legacy authentication is insecure in a WLAN, it requires the cryptographic protection afforded by the prior TLS handshake.

The TTLS/PEAP schemes are very similar to those used by many e-commerce servers to grant access to restricted information like electronic shopping carts for particular users. In the e-commerce setting the web browser first establishes an SSL [5] connection to a file server, and a legacy authentication method such as a password prompt executes, protected by a cryptographic key magically created by SSL. In the e-commerce setting, the merchant identifies itself using a certificate signed by a public certificate authority. The web browser can use the certificate to verify the identity of the merchant in two ways. First, the web browser extracts the name of the merchant from the certificate, performs a reverse DNS lookup on the address of the merchant’s web site, and compares the two names. If the two differ, the web browser posts a warning to the user. This is an important security step, because a failure to match usually indicates a man-in-the-middle attack by a hacker trying to steal the user’s credit card number [2]. In this way, SSL can protect the user from man-in-the-middle attack. Second, the web browser can retrieve the latest certificate revocation list—i.e., the list of compromised certificates—from the certificate authority that issued the certificate. If the merchant’s certificate is in this list, then the browser knows it can no longer trust the identity and alerts the user. The WLAN situation is a radically different environment. PEAP and TTLS are both still vulnerable to man-in-the-middle attacks, because the authentication of the Authentication Server to the station in the TLS step can be only one-way. The station can in principle detect this kind of attack (1) if the Authentication Server’s certificate has not been compromised and (2) if there were some analog of the reverse DNS lookup. In the WLAN case there is nothing that can be done about compromise of the Authentication Server’s certificate. If the attacker who compromised the certificate also controls the access point to which the station has attached, then it is possible to block access to the certificate revocation list. This is a risk well known from the remote access case, however, and many organizations choose to live with the threat, as the cost savings, convenience, and productivity generally outweigh the cost of an occasional compromise.

Of greater concern is that in the WLAN case no equivalent to the reverse DNS lookup has been standardized. An alternative solution to the latter problem exists, however, which is to provision each station with the Authentication Server’s certificate. If the station already knows the certificate of each Authentication Server it trusts, then it can rely on this configuration instead. However, this strategy requires that an organization securely install the Authentication Server certificate on every station—usually an expensive proposition—and that each station maintain its list of trusted certificates in some secure storage devices, lest an attacker corrupt the cache of trusted certificates on some station. Presently secure storage devices are not common in 802.11 stations.

The remark about provisioning stations with certificates illustrates that the TTLS/PEAP solution requires deployment of a Public Key Infrastructure (PKI) in order to succeed, and avoiding the cost of a PKI has been one of the over-riding goals of relying on the existing RADIUS infrastructure. It is not sufficient for the authentication server to send a certificate publishing its public key to the station as in the e-commerce case, because a hacker can legally obtain a perfectly valid certificate from the same certificate authority and publish it via a rogue access point. The problem is that the station needs to know which networks it should be willing to trust. If we restrict ourselves only to known techniques, this requires the station to know the identities of all networks it trusts prior to the TLS authentication step. The bottom line is that requiring TTLS or PEAP can greatly enhance the quality of the authentication, but it closes all the avenues for attack only with a costly investment in a PKI.
The second problem addressed by the revised keying proposal is closing the race condition between the EAPOL key and Success messages. The solution to this problem is much more satisfactory than that for defeating man-in-the-middle attacks. In the new architecture, the EAP Success message has no meaning. Instead, a new species of EAPOL Key message is introduced to perform a four-message *key confirmation handshake* between the station and the access point. 

The purpose of key confirmation handshake is to verify that each peer using the link possesses the session key, and to guarantee that the session key is fresh.  

![Key Confirmation Handshake](image)

**Figure 3—Key confirmation handshake events**

The access point begins the exchange by sending the first key confirmation message to the station. This message contains a random value. In response, the station chooses its own random value and sends this to the access point in the second message. At this point the station and access point possess both random values. They use a cryptographic primitive called a *pseudo-random function* to mix this pair of random values with the session key to obtain a new operational key, which is consumed by TKIP. The operational key is fresh because of the randomness contributed by each party. The protocol requires both the station and the access point to contribute their own random value, in case the peer cheats and reuses a prior value.

After computing the operational key, the access point configures its TKIP implementation to send and receive data traffic to and from the station using this key. The access point then sends the third key confirmation message and unlocks the station’s 802.1X port to receive but not to send data traffic. When the station receives the third message, it configures its TKIP implementation to send and receive data under the operational key. The station then transmits the fourth key confirmation message to the access point. When the access point receives this message, it enables the 802.1X port for the station to finally send data.

This scheme guarantees that no data passes over the link until after the TKIP keys are in place to protect the data. The session key is used to protect each of the key confirmation messages, defending from attack. The entire exchange could be improved somewhat by eliminating one message, but the security properties of a properly optimized protocol would be the same as the current one.

We said that the key confirmation handshake introduces a new species of 802.1X EAPOL Key message. This corrects one more problem from the first attempt at key management, in that it replaces the WEP-like key wrap algorithm with a more secure encryption scheme.

**Attempt 2's Discontents**

Although not perfect, 802.11’s second attempt at key management looks pretty good as far as it goes. However, analysis indicates that the second attempt does not go far enough, so it is unsuitable for anything but an interim solution. Attempt 2 leaves several major security problems unaddressed: lack of binding between the authentication and the session key, reliance on RADIUS to distribute the session key, a hard-to-adapt architecture, and overall complexity. Most of these problems are outside the scope of 802.11 or even the LAN standards group 802, so must be addressed in a different standards body.

To comprehend the potential risks it poses, we will need to understand Attempt 2’s keying paradigm. TKIP keys arise in one of two ways. The first source is a static key. A static key is configured on the 802.1X Authentication Server and the 802.11 station. The second source is the TLS protocol, which magically negotiates a fresh key on every invocation. This mechanism uses public/private key techniques to establish a randomly generated key shared between the Authentication Server and the station. Regardless of the key source, real implementations use RADIUS to push the key from the Authentication Server to the access point. There are several problems with this architecture.

The first is that this architecture is insecure when used with static keys. Recall that encryption schemes require auxiliary data to prevent information about the encrypted data and the key to leak to an attacker, and that the <key, auxiliary-data> pair can never be reused. Since the scheme does not demonstrably protect against key reuse, direct distribution of a static key cannot be considered secure. A static key cannot be safely pushed; it can only be used for authentication or for wrapping a fresh key. This may seem like a minor issue, but the practical

---

8 Stations and APs still transfer the EAP Success message, but this is only for backward compatibility with legacy EAP implementations. An EAP Success received prior to the key confirmation handshake is insecure, and one received after is irrelevant.
implication is that 802.11 key management, as currently defined, requires a dynamically generated session key. This in turn means that TLS must be used for authentication.

A more fundamental issue relates to a gap between authentication and the distributed session key. The concrete authentication method on top of EAP uses application layer identifiers such as user identifiers or machine or service instance names. Any session key EAP provides is of necessity bound to the identifiers used at this level. TKIP, however, uses MAC addresses to identify the right key to decapsulate a received packet. When a TKIP packet arrives, TKIP selects the encryption and authentication keys associated with the alleged packet transmitter for its hop. Thus, the session key that TKIP consumes is bound to a pair of MAC addresses, not to the application layer identifiers. This gap negates TKIP’s authorization model.

The current architecture attempts to bind the session key to MAC addresses implicitly instead of explicitly. This means that both the station and access point have to assume the other party is the correct one, and the authentication server has to assume there is no man-in-the-middle attack against the authentication. This is very risky, and should not be assumed in general.

This is not a theoretical issue. The literature of key management is littered with the remains of failed protocols that do not bind keys to identifiers at the level of the key consumers. When this binding does not occur, or does not occur correctly, the protocol leaves the door open to abuses by the protocol participants, and increases the opportunities for hacking. The current architecture treats key management as a data transport problem—how to move a key from the authentication server to the access point—and it is not. Rather, key management is primarily a problem of how to bind keys onto a particular context in a verifiable way. This gap represents a flaw in the current design.

Attempt 2 uses a RADIUS-based key distribution scheme, invented for Microsoft Point-to-Point Encryption (MPPE). This scheme treats a hash function as a pseudo-random function in order to perform a counter mode encryption. The construction used does not rest on adequate theoretical underpinnings, so its security properties are not understood. However, this is not the most serious problem with the approach. Instead, RADIUS does not have a scheme to manage the key encryption keys it uses for key distribution, so real implementations tend to rely on weak keys or even passwords for this function, just as today’s WEP does. This means the RADIUS key used to distribute keys can usually be broken by brute force attack.

There are a number of ways to ameliorate these problems. To implement Attempt 2, the channel between the access point and authentication server should be wired, and intrusion detection should monitor this channel as well, looking for active attempts to interject bogus keys into the channel. Stations should be configured to abort any authentication request showing evidence of a man-in-the-middle attack. These conditions, however, limit deployment of TKIP with Attempt 2 to enterprises and service providers with IT staffs experienced in security matters.

A second basic problem with the approach is that it is fairly inflexible, and it is not readily apparent how one uses it to support many configurations. 802.11, for instance, is defining “side channel” communication. In this mode, stations communicate directly among themselves instead of through an access point. The existing model does not and cannot address this configuration directly. The second attempt at key management only addresses how a key is established between a station and the authentication server, and how the authentication server pushes this key out to an access point. A whole new approach will be needed to establish keys between pairs of stations. Similar problems arise when one attempts to apply the existing model to broadcast/multicast key distribution, and to ad hoc networks.

A final problem with the present key management definition is its complexity. Almost every cryptographer who has examined it has declared it insecure—not because of any evident flaws in the protocol, but rather because it is so complex that it defies systematic analysis. When someone asks whether the key management protocol is secure, the proper response is that no one knows or can know, because its design has too many independent parts to thoroughly examine and analyze. A system consisting of \( n \) components has \( n^9 \) relationships among the components. A proper security analysis needs to examine each one of these relationships. The chances are overwhelming that some relationship exists among the components of the design that can be exploited by an attacker, and there are potentially thousands or millions of hackers who will have years to probe these relationships, whether by design or by accident.

A second problem with the complexity of the existing solution is that it makes interoperability more difficult. There are two ways to achieve simplicity in an engineering design: by adopting architectures based on inherently simple algorithms, or by abstracting complexity with higher layer constructs. The current design does the latter, but the cost is that all independent implementations must abstract the problem in the same way. Specification to this level of detail constitutes design work, which is outside the scope of any standard.

Summary
IEEE 802.11 has made great progress toward fielding an interim solution to its security problems, with the formulation of TKIP and an 802.1X-based key management protocol. The TKIP security assurances depend entirely on those of the key management protocol. While the key management protocol is not perfect, it can provide a viable interim solution in many managed networks. For a solution deployable in a wider market share, it will be necessary to wait for a more robust key management protocol. In the next article in this series, we will discuss the shape such a protocol must take, as well as 802.11’s progress toward realizing such a protocol.

For Further Reading

\( n! = 1 \cdot 2 \cdot 3 \ldots (n-1) \cdot n \) and is pronounced n factorial. This value grows prodigiously with the value of \( n \), as \( 3! = 6, 4! = 24, 5! = 120, 10! = 3628800 \), etc.
About the Author

Jesse Walker is the network security architect for Intel's Platform Networking Group. He is the technical editor for the 802.11 security enhancements, and was the first person to publicly identify the security flaws in the original 802.11 WLAN protocol. He joined Intel as part of the Shiva acquisition. Jesse has also been active in other industry standardization efforts, including IPSec. Jesse holds a Ph.D. in mathematics from the University of Texas at Austin. Jesse may be contacted at jesse.walker@intel.com.