Data Dependent Keying for Wireless Networks

Manish Karir John S. Baras
Center for Satellite and Hybrid Communication Networks
Department of Electrical and Computer Engineering & Institute for Systems Engineering
University of Maryland, College Park, MD 20742, USA
{karir, baras}@isr.umd.edu

***Abstract*— The failure of the 802.11 WEP security specification to provide any reasonable level of security has come under sharp criticism recently. In this paper we propose a novel scheme for providing security in both a basestation based, as well as an ad hoc network environment. Our proposed scheme uses data exchanged between communicating peers to evolve per packet keys. In addition to the concept of Data Dependent Keying(DDK) we include other well known security primitives such as SHA-1 based HMAC and RC4 encryption to provide a complete security solution for wireless networks. We argue that our scheme provides an adequate security/overhead tradeoff, and can be easily implemented in current hardware platforms. In addition, the low overhead characteristic as well as the use of symmetric cryptographic functions makes the scheme an attractive option for sensor networks, where energy efficiency is a primary objective.**

I. Introduction

Security concerns have been an afterthought in the design of wireless networks. A survey of wireless networking research quickly reveals that this is true not only for networks that include a basestation, but is also true for ad hoc or infrastructureless networks. Therefore, we are faced with a situation where we have to backfit security schemes into wireless network scenarios. The 802.11 standard for wireless networks does include some simple security features, but the level of security they provide is simply inadequate. The situation is only made worse by oversimplistic implementations of the standard.

The failings of the security features of 802.11 standard (WEP) have come under significant criticism recently. It has been shown that the weakness of the 802.11 security mechanisms is primarily due to the poor practice of cryptographic principles. Some of the primary weaknesses that have been pointed out are: short initialization vector, which is transmitted in the clear; the use of simple CRC32; the use of a single shared key; and the lack of a mechanism to change the secret key [1].

The standards documents for various aspects of ad hoc networks are still under development by the IETF. However, even in their current forms, most proposals tend to ignore the problem of security. Only recently has focus turned to this important aspect, as attacks on routing protocols, data forwarding, node authentication and data encryption are starting to be examined in more detail.

In this paper we propose a novel scheme for providing security in both an infrastructured as well as an ad hoc wireless network environment. Our proposed scheme uses data exchanged between communicating peers to evolve per packet keys. For each pair of communicating peers, the key evolves differently based upon the data that is exchanged between them. In addition to the concept of Data Dependent Keying(DDK) we include other well known security primitives such as SHA-1 based HMAC and RC4 encryption to provide a complete security solution for wireless networks. We make the assumption that our security scheme will work in conjunction with a 802.11-like MAC layer, where each data packet that is successfully received is acknowledged. The system is bootstrapped by initializing all nodes to be deployed, with the same initial key. After bootstrapping, depending on the communication pattern and the data exchanged between pairs of nodes, a copy of the original key is used to derive subsequent keys for communication between that pair.

The rest of this paper is organized as follows. Section II outlines some related work in the area of security, both for 802.11 as well as ad hoc networks. Section III describes in detail our proposed Data Dependent Keying scheme. Section IV provides a brief description of how DDK can be used to implement security in both infrastructured as well as ad hoc networks. Section V concludes with a brief summary of our contributions, and sketches out future work.

II. Related Work

The security provided by WEP has been strongly criticized. [1] was among the first to describe in detail the flaws in 802.11 security. They clearly demonstrated the problems of 802.11 in relation to the three fundamental goals of security; Confidentiality, Access Control and Data Integrity. They argue that weakness in initialization vector(IV) use, as well as the use of simple CRC32 by 802.11, makes it vulnerable to a wide variety of attacks. In addition, they also describe flaws in some common implementations of WEP which worsen the problems. While [1] focused primarily on the weakness of IV use in WEP, [2] pointed out further problems in the WEP access control mechanisms. They argue that the only viable long term solution to the collective problems of WEP is a major overhaul of the current standard.

The security aspects of ad hoc networks have also come under criticism. Various forms of attacks have been described on the routing mechanisms for these networks. Most of the protocols that have been proposed do not incorporate security in their design. This leaves them vulnerable to several attacks based on spoofed, altered, or replayed routing information[11].

0-7803-7954-3/03/$17.00 ©2003 IEEE.

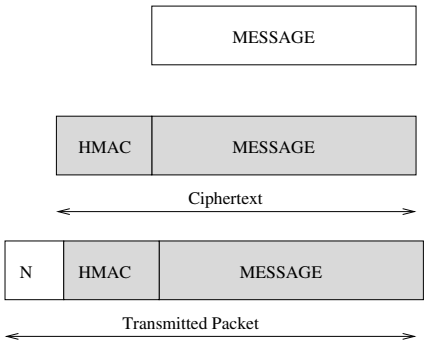

Fig. 1. Packet structure while using DDK

Recent work has focused on adding security to the various routing schemes. [3] proposed the use of certificates with routing protocols such as AODV and DSR to provide protection from various attacks where routing packet headers might be modified by malicious nodes, as well as scenarios where false routing packets can be inserted into the network by malicious nodes. [4] on the other hand has developed protocol extensions to AODV which provide security via the use of asymmetric cryptography. They do not describe how public keys for all the nodes are distributed. In addition [5] [6] have proposed a delayed disclosure based protocol for broadcast and multicast authentication. They do not however, address how those protocols can be used for unicast data security, and do not provide a method for encryption of the data being transmitted. [7] is a secure ad hoc routing protocol that uses symmetric cryptography and is based on the broadcast authentication described in [5] [6]. It modifies the original DSR routing protocol to make it more secure. [11] discusses attacks and countermeasures for some attacks on sensor networks, and focuses primarily on routing and data forwarding attacks.

To the best of our knowledge, a *data dependent* keying mechanism has not been studied for its applicability in wireless networks.

III. Data Dependent Keying

The Data Dependent Keying(DDK) scheme, aims at providing a viable alternative to WEP. In addition, DDK can also be used to provide security in wireless ad hoc and sensor networks. The subsequent sections describe the different facets of this scheme. While designing DDK we attempted to keep the following design principle in mind.

- Data confidentiality: Only two communicating parties should know the content of their communications.
- Data integrity: It should not be possible for a malicious third party to modify data in transit from one party to its communicating peer.
- Security from replay attacks: The scheme should be protected against simple replay attacks, where a malicious third party records communications between two peers and at a later date plays back some or all of the packets.
- Protection from passive attacks: It should not be possible to build a key dictionary or otherwise compromise security via passively monitoring the wireless channel.

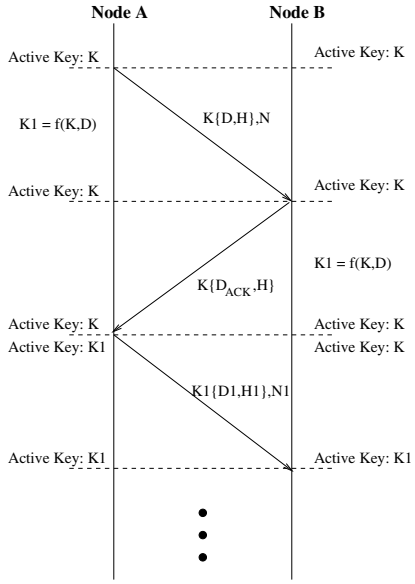

Fig. 2. Data Dependent Key Evolution

- Protection from random errors in the wireless link: The scheme should be robust against random and unpredictable behavior of the wireless channel including operation in a hostile environment.

The following sections describe key features of DDK and how they collectively form a system that meets the goals outlined in the design principles.

A. Data Dependent Key Evolution

One of the key ideas behind DDK is the ability to derive a key chain that depends only on previous data exchanged between two communicating pairs. The nodes on deployment are initialized with the same secret initial key. Subsequent keys are derived from the initial key and the data exchanged between pairs of nodes. As the keys are evolved based on data *transmitted* from a node, for any pair of communicating nodes, the keys used to send data from node A to B are different than the keys used to send data from node B to A.

The key evolution for a single pair of nodes A and B can be summarized as follows. Initially both nodes A and B have the same key *K*. Node A uses a hash generation algorithm, an encryption algorithm and the key *K* to encrypt data that it sends to Node B. It then computes the next key *K1* via some function performed on the key *K* and the data *D*

$$K1 = f(K, D)$$

Node B on receiving encrypted data from Node A proceeds to decrypt it using key *K* to obtain data *D*. Node B then verifies that the packet has been correctly received via the HMAC, as described in the next section. If the packet was received correctly, it then computes the next key *K1* via the same operation that Node A used as it has the same key *K* and data *D* . At this point Node B does not install this new key *K1* as the *active* key. Both A and B have to agree on when key *K1* should be used. Node A can only install *K1* as the *active*

key when it is sure that node B has correctly computed *K1* as well and vice versa.

After node B has correctly received the data, and used it to compute *K1*, it generates a link layer ACK for that data. This ACK implicitly informs node A that B has correctly computed the next key. The ACK packet includes a keyed hash value to prevent an ACK spoofing attack. When node A receives and verifies this ACK, it installs key *K1* as the *active* key and will use it to send the next packet. Node B will not install the key *K1* as the *active* key unless it receives the next packet from Node A which is not a retransmission of the previous packet. Node B can distinguish a new packet from a retransmission, via the nonce field in the packet which is different for each *new* packet. The key evolution process is also illustrated in Figure 2. After each successful exchange of data between nodes A and B, a new key is derived to be used for the transmission of the subsequent packet.

If the data packet from node A is lost, it will be retransmitted after a timeout using the *active* key, which is the key that was used for the last packet. If the ACK for the data packet from node B were to get lost, node A will retransmit the packet using the previous key *K* and node B will use also use the previous key *K* to decrypt the packet.

One important aspect of the key evolution process that we omitted so far, for purposes of clarity, is the actual details of the function *f*. This function can be chosen from a class of functions as long as it preserves the essential feature that, it should take the current key *K* and the last data item encrypted using *K*, as input and produce a new key *K1* in a manner such that *K1* depends on the value of both the previous key *K* as well as the previous Data *D*. For example, a simple implementation of f might be to compute a hash of the data, and then subsequently use this hash in a simple XOR operation. This can be achieved by using a keyed hash function that takes the random nonce, which is transmitted with the packet as the key:

$$K1 = K \oplus H(D, N)$$

Both the sending and the receiving nodes are correctly able to perform this operation as they both have the same data and the nonce. While any data corruptions to the data will be identified by a failure of the HMAC check, it is possible for the nonce to be modified silently by the channel without detection. This however, does not pose a serious problem as the resynchronization procedure described in section III.E will detect a loss of synchronization and will subsequently force a key resynchronization.

B. HMAC

In addition to the data dependent key evolution, we also include SHA-1 based HMAC, on the original data packet [8][9]. The MD5 message digest could also be used in place of SHA-1, what is important is that a keyed hash function be used. The choice of the particular hash function would probably depend on the platform for which DDK was being implemented. For an environment that incorporates basestations and laptops, SHA-1 might be a good option, while for implementation on platforms that have limited energy or computation capabilities, MD5 might be a better option. We describe the use of SHA-1 algorithm here as an example.

The SHA-1 algorithm produces a 160-bit message digest for every data packet. The HMAC algorithm uses the SHA-1 algorithm to obtain the MAC using the key K and the original data packet M as input. If H denotes the SHA-1 hash function, *ipad* denotes n repetitions of the byte 0x36, and *opad* denotes n repetitions of the byte 0x5C and the secret key is denoted by K, then the generation of the HMAC can be denoted as shown below:

$$HMAC(D, K) = H(K \oplus opad, H(K \oplus ipad, D))$$

The use of keyed HMAC is necessary as, it was well pointed out in [10], encryption without authentication is simply not suffient to offer any level of security. In comparison WEP uses an unkeyed CRC32 algorithm, and thereby offers little security. The HMAC is even more for important for DDK because we evolve the keys between two peers based on the assumption that they both have the same copy of the original data. Using HMAC helps us ensure not only that a received data packet came from who we think it came from, but also protects us against random bit corruption in the wireless link.

C. Encryption

Using the building blocks described in the previous two subsections on data dependent keying and HMAC, we now describe how secure communications can be build on top of them. While we describe the use of RC4 for encryption and decryption, it should be noted that other encryption algorithms such as RC5 might be chosen as well. The decision of which algorithm is chosen would depend on the capabilities of the implementation platform.

We start with message or data packet *D* that has be transmitted from A to B. We first compute the HMAC over *D*, using the current active key *K* as described in the subsection on HMAC. We then append the HMAC to the original data packet to generate the plaintext *P*.

$$HMAC = HMAC(D, K)$$

$$P =< D, HMAC >$$

The secret active key K is used to generate a RC4 key-stream of the same length as *P* and the two are XOR'ed together to generate the ciphertext *C*.

$$C = RC4(K) \oplus P$$

$$C = RC4(K) \oplus (< D, HMAC(D, K) >)$$

Finally, we append a random nonce *N* to the ciphertext to help protect from a replay attack as well as to help the receiver distinguish a new packet from the retransmission of a previous packet. This result is then transmitted to B. The nonce field is also used in the synchronization recovery procedure as described later.

$$A \rightarrow B :< N, C >$$

D. Decryption

When a node receives an encrypted packet from a peer, it compares the nonce from the received packet with the stored value of the nonce from the previous packet, to determine whether this is a new packet or a retransmission of the previous packet. If this is a retransmission of the previous packet, then the current value of the active key *K* is used. If this is a new packet, then the previously computed new key *K'* is used. To decrypt the packet, we first extract the ciphertext *C* from the received packet. Next we generate a RC4 key stream using the key *K* or *K'*. The plain text is extracted from the ciphertext via:

$$P = RC4(K) \oplus C =< D, HMAC >$$

At this point we generate a HMAC *H'= HMAC(D,K)* of the data portion of the plaintext, and then verify that *H'* is the same as the HMAC *H* in the plaintext. If *H'* is the same as *H* then the received packet is valid and accepted for delivery to the upper layers. In addition, a link layer ACK is sent back to the sender that acknowledges the receipt of the packet.

E. Synchronization Recovery

Even though by using HMAC we obtain a reasonable level of assurance that the packet received by node B is the same as was transmitted by node A, there is still a finite possibility that both the message portion of the packet and the HMAC will be modified in such a way that it still produces a valid but different message than was transmitted. Since DDK relies on maintaining data synchronized keys between two communicating parties, this situation would desynchronize the key information between the two nodes. To protect against this we have developed a checkpoint based re-synchronization method. As the keys evolve over time, at periodic intervals a special message is exchanged between the two that establishes the current valid key as the *fallback* key. This special message is simply the ID of a node that is encrypted using the current key and transmitted just like any other data packet. On receiving this packet a node installs the current active key as the *fallback* key. If the ID field in the received packet is set to NULL, the key evolution process is placed in a disabled state. This feature is used to perform simple key revocation.

De-synchronization is detected when two communicating nodes can no longer receive messages from each other as the HMAC check on received packets from another node repeatedly fails. Or when a node fails to get valid acknowledgements from the node it is communicating with. In this scenario, both nodes revert to the *fallback* key and attempt to reestablish communication. Only if they both have the same fallback key can they reestablish communication.

It is essential that the same key not be used more than once to encrypt messages. If an attacker obtains two messages, encrypted with the same key he can obtain critical information regarding the plaintext messages via a simple XOR operation on them.
If

$$C1 = RC4(K) \oplus P1$$

$$C2 = RC4(K) \oplus P2$$

then

$$C1 \oplus C2 = P1 \oplus P2$$

It is important that the fallback key not be used directly, as it has already been used once before. Therefore, when a node is attempting to perform synchronization recovery, it derives a key based on the fallback key using a random nonce.

$$K' = RC4(K, N)$$

This value of the nonce *N* is then transmitted to the receiver at the head of the transmitted packet. On receiving this packet a node that is in synchronization recovery mode will use the value in the nonce field of the packet to derive K', and then proceed to use that key for packet decryption as usual. This operation is similar in spirit to the operation of WEP, however, in our case a node is only expected to be in this mode for a very limited amount of time and is only expected to transmit a few packets while in this mode, therefore, we do not suffer from the same problems as WEP as key collisions are unlikely when we only use the fallback key to derive at most a few derivative keys. Moreover, a backoff algorithm is used to limit how frequently resynchronization is attempted to aleviate attacks where de-synchronization is being caused by a malicious host.

IV. Security in Wireless Networks

In this section we briefly illustrate how DDK can be used to provide security in both an ad hoc network as well as in networks which utilize basestations.

A. Securing Infrastructured Networks

In a network that contains basestations DDK is similar in operation to WEP. The primary difference is that unlike WEP in DDK the key is changed with the transmission of every data packet. This addresses some of the shortcomings of WEP. The other significant advantage of using DDK is the use of a HMAC algorithm instead of a simple CRC to verify data integrity. During deployment, the basestation is configured with an initial key, which is also distributed to all the initial clients. Using this initial key, the basestation performs the re-synchronization procedure with each client, to bootstrap the key evolution process.

Adding a new node simply consists of installing the initial key onto a new client and introducing it into the network. The basestation will perform resynchronization with it and incorporate it into the network. As the basestation contains a separate key for communication with each client; key revocation is achieved by the basestation sending a checkpoint message with the id field set to NULL. This will place the key evolution state machine on the client in the disabled state. At the same time the basestation will disable its key evolution state machine for that particular client.

B. Securing Ad Hoc Networks

DDK can also be used to provide security in an ad hoc network environment. In an ad hoc network, each node performs the same functionality that a basestation would perform

in a fixed infrastructure network, and security is provided on a hop by hop basis. Therefore, in an ad hoc network, each node would maintain a separate key evolution state machine for each of its next hop neighbors. As a data packet is forwarded through the network, encryption is performed at each hop and at each hop a separate encryption key is used for encryption. To bootstrap the key evolution, each authorized node is given the same initial key during network deployment. The nodes then perform the re-synchronization process to start the key evolution process depending on the data they exchange with their next hop neighbors.

Adding a new node simply consists of installing the same initial key that was used during the initial network deployment on the new node and inserting it into the network. Depending on its neighbors, it will evolve subsequent encryption keys. If a node suspects one of its neighbors, it can revoke its key, via a simple process of sending it a checkpoint message with the ID field set to NULL. This will disable the key evolution state machine for exchanging data packets between those two particular next hop neighbors. The node that sends the NULL checkpoint message will ignore any further messages from the suspect node. Other nodes may however, continue communication with the suspect node. This protects us from malicious, captured nodes attempting to shutdown the entire network by simply sending out NULL checkpoint messages as they travel through the network. Only communication with a particular node is disabled.

Used in this manner, DDK can significantly enhance the security of an ad hoc network. It can prevent an entire class of simple eavesdropping and replay attacks as in order to be able to receive packets, a node must be able to obtain the initial key used during deployment. Sinkhole, spoofed/altered packets, selective forwarding, and route falsification attacks [11] are also prevented as a node can not participate in the ad hoc network, unless it has a copy of the valid initial key. DDK does however fail to provide any security in the scenario where a node is able to fraudulently able to obtain this initial key. The only option in that scenario is for the next layer of security, possibly at the routing layer to detect the malicious node, and use DDK to trigger a key revocation.

V. Conclusions and Future Work

In this paper we introduce the concept of Data Dependent Keying. Data Dependent Keying scheme provides confidentiality, as well as data integrity for a wireless network. We have described the key evolution mechanism, as well as related security primitives such as a SHA-1 based HMAC and the use of RC4 algorithm, which together form a complete security solution. We also provide illustrative examples of how DDK can be used in an infrastructure based as well as in an ad hoc network environment. For an ad hoc network environment, since we only maintain pairwise keys on a hop by hop basis, we are not concerned with scalability as the number of nodes within the range of a given node is not expected to be very large.

We are currently working on implementing DDK in a linux testbed via modifications to the device drivers for 802.11b wireless cards as well as investigating the feasibility of its use for a sensor network using sensor networking nodes developed intially at UC Berkeley. The sensor node platform based on TinyOS offers much promise as we have easy access and control over the MAC layer functions that are responsible for link layer ACK generation.

Though in this paper we focused on the use of RC4/SHA-1, the concept of DDK is much more general, and allows users to pick from a set of algorithms, as long as they meet the key criteria. Various alternatives that can also be used to implement DDK need to be examined in greater detail. Along those lines we are investigating efficient algorithms for the key evolution function. Alternate resynchronization/checkpointing methods also need to be studied. We believe data dependent keying is a promising flexible security model for wireless networks that needs to be examined and studied in greater detail.

Acknowledgements

The material presented in this paper is based upon work partially supported by the U.S. Army Research Office under Award No DAAD 190110494 and partially through collaborative participation in the Collaborative Technology Alliance for Communications and Networks sponsored by the U.S. Army Research Laboratory under Cooperative Agreement DAAD 190120011. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the U.S. Army Research Office or the U.S. Government.

References

[1] H. Borisov, I. Goldberg, and D. Wagner. Intercepting mobile communications: The insecurity of 802.11. *ACM Proceedings of the Seventh Annual International Conference on Mobile Computing And Networking*, July 2001.

[2] W. Arbaugh, N. Shanker, J. Wan, and K. Zhang. Your 802.11 wireless networks has no clothes. *IEEE Wireless Communications*, pages 44–51, Dec 2002.

[3] Kimaya Sanzgiri, Bridget Dahill, and Brian N. Levine. A Secure Routing Protocol for Ad hoc Networks. *International Conference on Network Protocols (ICNP)*, Nov 2002.

[4] Manel Guerrero Zapata and N. Asokan. Securing Ad hoc Routing Protocols. In *Proceedings of the 2002 ACM Workshop on Wireless Security (WiSe 2002)*, pages 1–10, September 2002.

[5] A. Perrig, R. Canetti, D. Song, and D. Tygar. The TESLA Broadcast Authentication Protocol. *RSA Cryptobytes*, Summer 2002.

[6] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient and Secure Source Authentication for Multicast. *Proceedings of Network and Distributed System Security Symposium*, Feb 2001.

[7] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks. *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (Mobicom 2002)*, Sep 2002.

[8] J. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication. *IETF Informational RFC 2104*, Feb 1997.

[9] Mihir Bellare, Ran Cannetti, and Hugo Krawczyk. Keying Hash Functions for Mesage Authentication. *Lecture Notes in Computer Science, vol 1109*, pages 1–15, 2002.

[10] J. Walker. Unsafe at any key size: An analysis of the wep encapsulation. *IEEE Document 802.11-00/362*, Oct 2000.

[11] Chris Karlof and David Wagner. Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures. *First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.