State Based Key Hop Protocol: A Lightweight Security Protocol for Wireless Networks

Mitchell, S.; Srinivasan, K.

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC · CNRC

State Based Key Hop Protocol: A Lightweight Security Protocol for Wireless Networks *

Mitchell, S., and Srinivasan, K.
October 2004

* published in the Proceedings of the 1st ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN 2004). Venice, Italy, pp. 112-118. October 4, 2004. NRC 47461.

Canada

State Based Key Hop Protocol: A Lightweight Security Protocol For Wireless Networks

Stephen Michell and Kannan Srinivasan
National Research Council
Sydney, Nova Scotia, Canada
Email: {stephen.michell, kannan.srinivasan}@nrc-cnrc.gc.ca

ABSTRACT
Sate Based Key Hop (SBKH) protocol provides a strong, lightweight encryption scheme for battery operated devices, such as the sensors in a wireless sensor network, as well as small office home office (SOHO) users. Although SBKH can be applied to many underlying protocols, in this paper, we focus on integrating SBKH with 802.11. Hence we compare SBKH with other 802.11 security protocols and show that SBKH eliminates all the issues with wired equivalent privacy (WEP) protocol, using the existing hardware and software as much as possible at a power and processing cost that is much lower than WiFi Protected Access (WPA) 1.0 or 2.0, and is cheaper than WEP.

Categories and Subject Descriptors
C.2.1[Network Architecture and Design] Wireless Communications
C.2.0[Computer Communication Networks] Security.

General Terms
Security, Performance.

Keywords
Computer Network Security, Wireless Security, State Based Encryption, Low Power Security, Wireless Sensor Network Security.

1. INTRODUCTION
In this paper, we present State Based Key Hop (SBKH) protocol, a lightweight security protocol suitable for wireless sensor and other low power devices. SBKH is also easy to maintain and so is suitable for small office home office (SOHO) users.
Although SBKH can be applied to other wireless protocols, we focus our discussions in this paper on integration of SBKH with IEEE 802.11 as it is the widely used wireless standard. Hence our comparison of SBKH is carried out against 802.11 security protocols namely: WEP and WPA. First we present a brief review of WEP and WPA followed by protocol overview of SBKH.

1.1 Background
1.1.1 Wired Equivalent Privacy (WEP)
[IEEE802.11 1999] defined an encryption scheme called wired equivalent privacy (WEP), to provide security to the 802.11 users.

PE-WASUN'04, October 7, 2004, Venezia, Italy.

WEP is a symmetric encryption scheme in which a WEP key is known or shared between two communicating nodes. WEP uses RC4 algorithm to do per packet encryption. RC4 algorithm is a stream cipher scheme [FM 2001, Mantin 2001] in which the data is encrypted by XORing data with the cipher stream generated by RC4 from an RC4 seed. WEP concatenates the WEP key (40 or 104 bits) and the initialization vector (IV) (24 bits), as the RC4 seed. For every new RC4 seed, RC4 reinitializes its states using key-scheduling algorithm (RC4-KSA). After RC4-KSA, RC4 generates the cipher stream using pseudo random generation algorithm (RC4-PRGA). Since the IV is sent in every packet, WEP carries out RC4-KSA and RC4-PRGA for every packet.
WEP is identified as being weak in a number of areas which make its continued use as a security mechanism for wireless untenable. These are discussed in more length in [SIR 2000, SM1 2004] and summarized here. [Walker 2002] identify weak key issues and shows that the forgery attacks, replay attacks and bit flipping attacks let active attackers spoof networks, make invalid packets seem valid and can derive the shared key from such attacks.

1.1.2 Wi-fi Protected Access (WPA)
IEEE 802.11i [Draft802.11 2003]'s first proposal for 802.11 legacy devices (WPA 1.0) encapsulates WEP functionalities by temporal key integrity protocol (TKIP). It also has an algorithm (Michael) to provide message integrity to protect data from any modifications. TKIP and Michael algorithms add significant processing on every packet. They also add additional overhead of 12 octets in every packet (without fragmentation) which can contribute to additional power consumption during transmission and reception.
IEEE 802.11i's second part (WPA 2.0) uses Advanced Encryption Standard (AES) and requires change in hardware. WPA 2.0 carries out AES twice under two different modes for every packet to encrypt the packet and to provide message integrity. WPA 2.0 also adds an overhead of 8 octets to every packet. Thus WPA 2.0 can also be very expensive.
Hence there is a need for a simple, robust, lightweight security protocol that carries out power-efficient encryption and decryption. SBKH is one such protocol.
[SM1 2004] introduced SBKH but restricted discussion to the basic protocol. [SM2 2004] evaluated SBKH against other encryption schemes to measure processing cost and complexity. This paper extends those to show how SBKH authenticates and resynchronizes.

2. PROTOCOL OVERVIEW
SBKH is a state-based encryption protocol in which two communicating nodes share a common knowledge of the RC4 state. Whereas WEP and WPA 1.0 reinitialize RC4 state for every packet and generate cipher stream from the initialized RC4 state, SBKH does not reinitialize RC4 states, rather it maintains the

same RC4 seed for a duration known to a pair of communicating nodes. This will require the initialization of the RC4 state (running RC4-KSA) to be done only when the base key changes. After this, communicating nodes keep using the same cipher stream, following the stream together, byte-by-byte to encrypt and decrypt packets exchanged between them. This is referred to as the nodes being *State Synchronized*.

To avoid weak key issue with RC4, SBKH does not begin decryptions at the start of an encryption stream, rather SBKH communicating nodes run down the cipher stream by a known, shared offset after every RC4-KSA, i.e. whenever a base key is changed and before communication begins. Only after running down the stream can a node encrypt or decrypt packets successfully. A second offset is used to determine where the resynchronization mechanism is initiated to recover from any loss of synchronization due to active attacks and possible implementation errors.

SBKH has been designed to operate with existing hardware and with existing 802.11 protocols as much as possible with minimal changes to the firmware. This is important to the users of millions of 802.11 cards shipped, where a change in the hardware will not solve the security issues with these existing 802.11 cards.

3. PROTOCOL DETAILS

3.1 Notation and Shared Parameters

In SBKH, RC4-KSA is executed only once for any key for any pair of communicating nodes. We define the term *communicating nodes* to indicate two nodes which are exchanging data and management packets, i.e. a node and an access point in a managed network, and two nodes in an ad-hoc network that are directly exchanging packets. We also define the notion of *Uplink (U)* and *Downlink (D)*. In a managed network, *D* is the direction from the *Access Point,* and *U* is the reverse. In an *IBSS, U* is the direction from node *A* which requested authentication with node *B*. *U* and *D* are appended as subscripts to other parameters to reflect the direction to which the parameter applies.

Since SBKH is state-based, communicating nodes require the following shared parameters to successfully maintain state: *Base Key Pair, Key Duration*, *RC4 states*, *Offsets*, an explicit SBKH sequence counter (*SSC*) and a *Nonce*. Note that nodes never exchange information to indicate these parameters, to indicate when key change occurs or to identify the next key pair selected. Such information was distributed before authentication by some means outside the scope of the SBKH protocol.

Base Key Pair

A *Base Key Pair* consists of two full 64 bit or 128 bit keys ($BaseKey_U$ and $BaseKey_D$) which are used as RC4 seeds for the communication between two nodes. These keys may be selected from a *Base Key List*, which may be common to all nodes within the same BSS or common only to a communicating node pair. The key may also be a per session key that is agreed between the two communicating nodes. The generation and distribution of the Base Key Pair and the Base Key List is out of scope of this paper, although we assume a strong key generation and distribution algorithm and note that the way that keys are selected or generated must preserve uniqueness of each base key across the BSS/IBSS.

Key Duration

Key Duration indicates when a base key pair is changed. SBKH uses the beacon time stamp and *Key Duration* to have common knowledge of key change between a pair of communicating nodes. Change of base key pair may be just as easy as selecting the next key pair from the *Base Key List,* as long the key uniqueness condition is preserved.

Offsets

Offsets are used to indicate how far down the cipher stream after running RC4-KSA a node starts encrypting and decrypting messages for a given *Base Key Pair*. This is referred to as running down the cipher stream. Running down the cipher stream for Offset number of octets happens only when a key rollover takes place. The purpose is to discard Offset number of encryption octets from the start of a stream, strengthening RC4. There are two types of Offsets: *Initial Offset (I-offset)* and *Sync Offset (S-offset)*. *S-offset* is used during resynchronization mechanism to encrypt and decrypt resynchronization frames. *I-offset* indicates the position where encryption and decryption of all other frames exchanged between A and B begin. Both *I-offset* and *S-offset* are non-zero values, which are distinct from each other. It is strongly recommended that *S-offset* < *I-offset,* to avoid encryption key reuse during resynchronization. Recommended values for the offsets are in the range of 300 to a few thousand, since the purpose is to avoid the RC4 weak key syndrome [FMS 2001] that occurs for some keys on the initial cipher stream octets.

RC4 States

RC4 state is a state array with 256 state elements and two indices. Each state element and each index is of 8 bits in length, making the overall RC4 state to be of 258 octets in length. For a successful communication between two nodes A and B, the RC4 state corresponding to A and B must be the same for a message to be encrypted and decrypted successfully, i.e. A and B are *encryption state synchronized.* A and B stay *state synchronized* by always encrypting and decrypting exactly same number of bytes (since each message successfully decrypted was also successfully encrypted). Since A and B operate asynchronously, SBKH maintains two sets of RC4 states, one for each communication direction: uplink and downlink. SBKH defines five pairs of RC4 states: Initial RC4 States ($IRC4_U$, $IRC4_D$), Previous RC4 States ($PRC4_U$, $PRC4_D$), Current RC4 States ($CRC4_U$, $CRC4_D$), Next RC4 States ($NRC4_U$, $NRC4_D$) and Sync RC4 States ($SRC4_U$, $SRC4_D$). The notation may be extended, so that $CRC4_{U,j,B}$ corresponds to the state in the receiver for $CRC4_U$ for packet j as maintained by node B and $SSC_U = j \bmod (2^{24}-1)$.

- *IRC4* are (collectively) the RC4 states after performing RC4-KSA and RC4-PRGA for *I-Offset* number of bytes for every Base Key. A node may start encrypting data packets with a new Base Key only after calculating the RC4 states $IRC4_U$ and $IRC4_D$ corresponding to $BaseKey_U$ and $BaseKey_D$ respectively.*PRC4* are the RC4 states corresponding to previously successfully *transmitted* or *received and acknowledged* SBKH encrypted packet. $PRC4_U$ and $PRC4_D$*CRC4* are the RC4 states with which encryption and decryption of the subsequent packet takes place for a given Base Key. $CRC4_U$ and $CRC4_D$ are updated independently.
- *NRC4* are the RC4 states corresponding to I-*Offset* for the next Base Key Pair. $NRC4_U$ and $NRC4_D$ are updated independently.
- $SRC4_U$ and $SRC4_D$ are the RC4 states corresponding to *S-offset* for a given Base Key Pair and are used in the resynchronization protocol. These can be calculated the same way IRC4 states are calculated.

PRC4 and *CRC4* states are continuously maintained for a pair of nodes: *IRC4; NRC4* and *SRC4* states are logical states and may be

generated as needed or maintained continuously, an option of interest to resource-limited devices.

SBKH Sequence Counter (SSC)
SBKH uses the 24-bit IV field with MAC frames of 802.11 as SBKH sequence counters (SSC) that are maintained for each direction (SSC_U and SSC_D) for a pair of directly communicating nodes. SSC is different from 802.11 MAC's sequence number, which is maintained for the whole network and not for a pair of nodes. The ability to maintain a pair-wise sequence counter helps decision making while trying to decrypt an incoming SBKH-encrypted packets.

Nonce
Each pair of communicating nodes maintain a shared nonce to permit verification of authentication, deauthentication, association and disassociation messages. This nonce is created during authentication and changed during resynchronization. The nonce is 128 bits (16 bytes) in length.

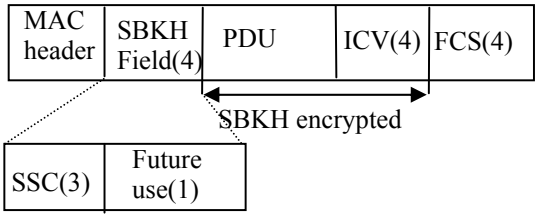


Figure 1: SBKH Data Packet

3.2 Basic Protocol Operation

Communication begins after authentication (see 4.4) for a pair of communicating nodes with SSC_U and SSC_D initialized to zero,

$PRC4_U = CRC4_U = IRC4_U$ and

$PRC4_D = CRC4_D = IRC4_D$.

Note that there is no connection between $IRC4_U$ and $IRC4_D$ since they reside on different cipher streams.

In Fig. 2, two nodes A and B are exchanging packets encrypted based on a Base Key shared between A and B. A sends packet j in its uplink encrypted at $CRC4_{U,j,A}$ to B. After receiving packet j, B compares $SSC_{U,j}$ with its $SSC_{U,j-1}$ which according to B was the last successfully acknowledged packet's SSC_U. If $SSC_{U,j}$-$SSC_{U,j-1}$=1, then B decrypts packet j at $CRC4_{U,j,B}$. After successful decryption ($CRC4_{U,j,B} = CRC4_{U,j,A}$) B acknowledges the packet to A and also updates its SSC_U to $SSC_{U,j}$. B then updates its $PRC4_{U,j+1,B} = CRC4_{U,j,B}$ and its $CRC4_U = CRC4_{U,j+1,B}$, the state where decryption of the packet corresponding to $SSC_{U,j+1}$ will begin. After the receipt of B's acknowledgment, A updates its $PRC4_{U,j+1,A} = CRC4_{U,j,A}$ and its $CRC4_U = CRC4_{U,j+1,A}$, the state where encryption of the packet with $SSC_U = j+1$ will begin . A also updates $SSC_U = SSC_{U,j+1}$ which will be used in packet with $SSC_U = j+1$. The same discussion applies for packets sent by B to A on its downlink.

Retransmissions and Packet Drops
For retries, there is no update of PRC4, CRC4 and SSC at A or B. If a packet or fragment times out and is dropped, the subsequent packet or fragment has the same SSC as the dropped packet or fragment and the transmitter does not update PRC4 and CRC4. Hence, encryption of the subsequent packet begins from the same place as that of the dropped packet.

If the transmission wasn't a retry and if the previously acknowledged SSC ($SSC_{U,j}$) and the received SSC from A ($SSC_{U,k}$) differ by more than 1 (i.e. k>j+1), then B may drop the packet without acknowledgment or may initiate resynchronization (see section 4.4). If the transmission of packet j is a retry (retry field in MAC frame set to 1), and if B previously acknowledged packet j and updated $CRC4_U$ to $CRC4_{U,j+1,B}$, and $PRC4_U$ to $PRC4_{U,j+1,B}$ (= $CRC4_{U,j,B}$), then B decrypts the packet from $PRC4_{U,j+1,B}$ or could optimize by sending an acknowledgment without re-decrypting.

If the transmission was not a retry and if the previously acknowledged SSC (SSC_U) and the transmitted SSC ($SSC_{U,j}$) are the same, then B identifies that acknowledgment of the previously transmitted packet was not received by A and the packet was dropped after retries. B decrypts the new packet using RC4 state $PRC4_{U,j+1,B}$ since A encrypted the packet at $CRC4_{U,j,A} = PRC4_{U,j+1,B}$. After decrypting the packet and acknowledging it, B updates $CRC4_U$ to $CRC4_{U,j+1,B}$, the state immediately following the last byte decrypted and leaves $PRC4_{U,j+1,B}$ unchanged.

3.3 Key Hopping

Two nodes communicating with each other remain State Synchronized as mentioned in section [4.2] if the Base Key pair has not changed. If the Key Duration parameter indicates time to change the Base Key pair, the transmitter starts decrypting packets and fragments using the new Base Key following the key change.

For the following discussion refer to Fig. 2 and assume that A identified a need for key change before encrypting packet j. This discussion only considers A and B updating $BaseKey_U$; the same protocol is used to update $BaseKey_D$ for B sending to A on Downlink, with B and A interchanged.

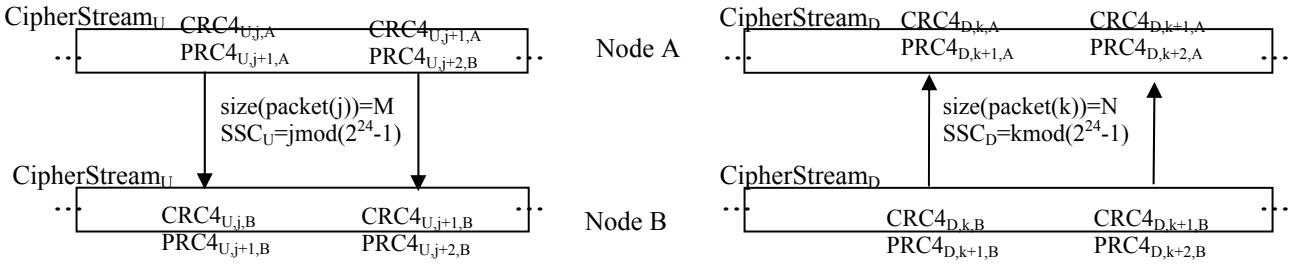
A calculates $IRC4_U$ based on the new $BaseKey_U$, and updates $PRC4_{U,j,A}$ and $CRC4_{U,j,A}$ to $IRC4_U$ after receipt of the acknowledgment of packet j-1. A then continues encryption of packet j based on the new $BaseKey_U$. When B identifies time to change Base Key, it calculates $NRC4_D$ and $NRC4_U$, based on the new $BaseKey_U$, but does not immediately update $PRC4_{U,j,B}$ and $CRC4_{U,j,B}$. B keeps decrypting subsequent packets using the cipher stream based on the $OldBaseKey_U$ until the decryption fails once, and then tries the decryption with $NRC4_U$ (note that for some circumstances B can optimize and try $NRC4_U$ first or decrypt both in parallel). The decryption succeeds, and B updates its $PRC4_U$ as $PRC4_{U,j+1,B} = NRC4_U$ and its $CRC4_{U,j+1,B}$ as the state where the decryption of the subsequent packet (j+1) will begin based on the new Base Key. B then clears $NRC4_U$. Following this, encryption and decryption of subsequent packets exchanged follow the discussion in section [4.2] until the next key change.

3.4 Initial Synchronization and Resynchronization

Initial state synchronization set up, termination and resynchronization take place through the use of *authentication*, *deauthentication*, *association, disassociation and reassociation* messages. Each of these management packets contains an encrypted payload portion for verification.

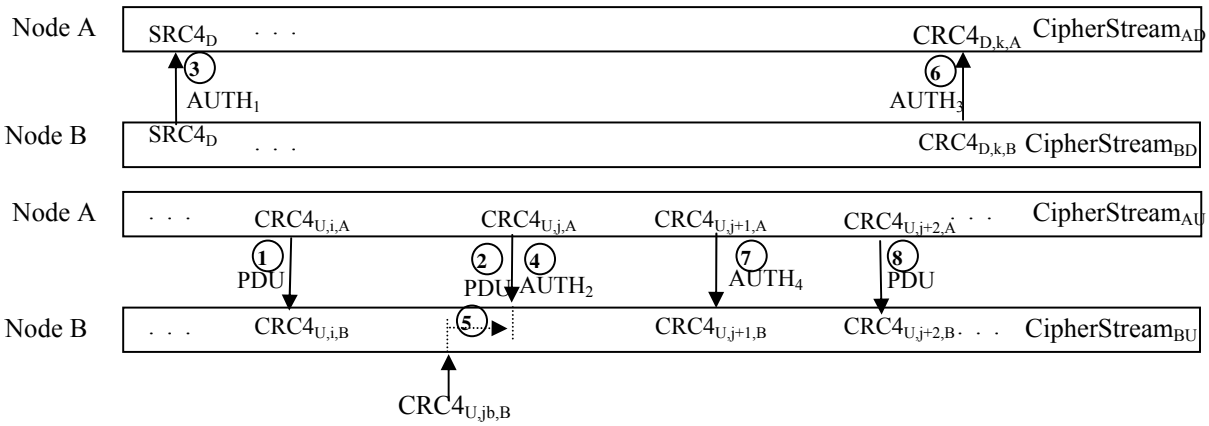
Each pair of communicating nodes establish and maintain a shared nonce to permit verification of authentication, deauthentication, association and disassociation messages. This nonce is created at Authentication and changed at Resynchronization.

Initial state synchronization is established during authentication process using *authentication* messages with reason field within *authentication* messages set to INIT.



Note: CipherStreamU/D are generated by unique keys

Figure 2: SBKH State Synchronization



Resynchronization Message Details

1. PDU before sync lost
2. PDU out of sync
3. AUTH pt1 B -> A
4. AUTH pt2 A -> B
5. B syncs forward
6. AUTH pt3 B -> A
7. AUTH pt4 A -> B
8. PDU after resync

Figure 3: SBKH Resynchronization

Authentication

Since authentication messages are directed management messages, acknowledgment is assumed unless otherwise stated. Also, some contain nonces encrypted at CRC4 and SSC to help synchronize SSC.

When node A wishes to communicate with node B, where B is the access point in a managed network or another node in an ad hoc network, A finds BaseKey$_U$ and BaseKey$_D$ and I-offset either from the key list or by some means not specified here, generates the state IRC4$_U$ and IRC4$_D$ and generates and sends to B the following:

`Auth(SBKH,`*`Reason`*`=INIT,1,(nonce,ICV)@IRC4`$_U$`)`

where "@IRC4$_U$" means encrypted using the state designated by IRC4 on BaseKey$_U$,, nonce is any nonce generated by node A.

ds no response (because deauthentication relies upon a common shared nonce, which A and B failed to establish).

Using this 3-way authentication, B guarantees that A has a new nonce and can successfully communicate from IRC4$_U$ and IRC4$_D$. B prevents replay attacks by selecting the new nonce, while A avoids replay attacks by the generation and inclusion of a unique initial nonce which B must return.

On receipt of the message, B uses a similar process to select the same key pair, decrypts nonce and ICV at IRC4$_U$ and validates nonce using ICV. If Auth part 1 fails to decrypt or validate, B ignores the message and sends no response; otherwise, B generates a new nonce, encrypts nonce, new_nonce and ICV at IRC4$_D$ and sends

`Auth(SBKH,`*`Reason`*`=INIT,2,(nonce,new_nonce, ICV)@IRC4`$_D$`).`

A decrypts (nonce, new_nonce ICV)@IRC4$_D$, re-encrypts (new_nonce, ICV)@IRC4$_{U+20}$ and sends

`Auth(SBKH,INIT,3,(new_nonce,ICV)@IRC4`$_{U+20}$`)`

B decrypts and CRC's this message, and either acknowledges it or sen

Deauthentication

B may send deauthenticate messages to A if B has authentication for node A and needs to terminate direct communication with A. These messages could be used to release resources associated with the communication between B and A. B terminates communication with A by sending

Deauth(SBKH,Reason,1,SSC_U,(nonce,ICV)@$CRC4_U$))
A decrypts the packet and responds by sending

Deauth(SBKH,Reason,2,SSC_D,(nonce,ICV)@$CRC4_D$))

Neither B nor A release resources until the deauthenticate messages are exchanged. Deauthentication can also occur implicitly when B has been silent for an implementation dependent amount of time (possibly infinite) and B did not notify A that it would be asleep, or when a resynchronization is attempted and fails after an implementation dependent number of attempts.

Association
Association happens as defined in [IEEE802.11 1999], except that *associate*, *disassociate*, and *reassociate* packets contain $SSC_{U/D}$ and an encrypted field (nonce, ICV)@$CRC4_{(U\ or\ D)}$ as the final field. *Associate, disassociate* and *reassociate* messages that fail decryption are ignored, protecting the network against attacks using rogue associate or disassociate messages.

Resynchronization
Situations may occur in SBKH where the encryption state may be lost between two nodes in one communication direction (Uplink or Downlink). This may occur if an active attack has fooled the transmitter in that direction with fake acknowledgments, if a node fails to update its state in nonvolatile storage before power-down, or if implementation errors permit state to be lost between a pair of nodes. The resynchronization portion of the protocol corrects such synchronization errors.
Resynchronization is implemented by a 4-way handshake using a four-part *authentication* message set with reason field within *authentication* message set to SYNC. Either party in the communicating pair can initiate a resynchronization if it discovers a loss of synchronization in the direction where it is the receiver. A resynchronization is capable of resynchronizing a single direction or both directions as needed, as explained below. Figure 3 shows two communicating nodes, A and B, resynchronizing, where A->B is uplink. B is expecting to receive PDU_i, shown as action 2 in figure 3 but instead receives PDU_j, j>i and B cannot successfully decrypt this message. B declares $CRC4_{U,j,B}$ invalid and sends message at 3 at the dedicated resynchronization offset $SRC4_D$,

Auth(SBKH,Reason=SYNC,1,(nonce,ICV)@$SRC4_D$).

A decrypts the Auth message at $SRC4_D$ and uses nonce and ICV to validate the authenticate message (invalid messages are ignored). A then selects a new encryption state $CRC4_U$, which is either $CRC4_{U,j,A}$ or a state which can be reached from $CRC4_{U,j,A}$ by executing RC4-PRGA for an implementation-dependent number of octets and sends the message at 4,

Auth(SBKH,Reason=SYNC,2,SSC_U,(nonce,ICV)@CRC4_U)

B acknowledges this message, though it may not yet decrypt the payload. This is acceptable because there are two more message parts to complete and confirm the resynchronization. B begins running forward on $CipherStream_U$ (action 5), decrypting successive bytes of the payload until a successful decryption of (nonce, ICV) occurs or an implementation dependent limit has been reached. If the limit is reached without a successful decryption, B retries the resynchronization and may eventually declare itself *implicitly deauthenticated*. Upon successful decryption and verification of (nonce, ICV), B updates $CRC4_{U,j,B}$, selects a new nonce and sends the message at 6

Auth(SBKH,SYNC,3,SSC_D,(nonce,new_nonce,ICV)@$CRC4_{D,k,B}$).

A decrypts nonce, new_nonce and verifies using ICV, generates a new ICV based only on new_nonce, encrypts (new_nonce,ICV) at $CRC4_{U,j+1,A}$ and sends the message at 7,

Auth(SBKH,SYNC,4,SSC_U,(new_nonce,ICV)@$CRC4_{U,j+1,A}$).

B receives and decrypts this message. Both nodes update nonce to new nonce and start communicating, shown as message 8.
Note that the process of decrypting forward (running down the cipher stream) is a secure process. The likelihood that two distinct places of length 20 octets in the encryption stream have the same encryption sequence is approximately 10^{-48}. Also, since synchronization failures always result in the transmitter being ahead of the receiver, B runs forward to resynchronize. If B does not need to move $CRC4_U$, B can detect active attacks such as false data messages and take countermeasures.

Two-way Resynchronization:
If A had also lost synchronization on $CRC4_D$ and discovers this as the resynchronization of $CRC4_U$ is in progress, the protocol executes as described above except that, upon receipt of Authenticate part 3, A cannot immediately decrypt the message. A sends ACK anyway and begins running down the encryption stream from $CRC4_D$, decrypting successive bytes of (nonce, new_nonce, ICV) payload until successful decryption occurs.
Once A successfully decrypts nonce, new_nonce and ICV, A updates $CRC4_D$ to that new position and sends

Auth(SBKH,SYNC,4,SSC_U,(new_nonce,ICV)@$CRC4_U$)

A protocol failure causes B to become implicitly deauthenticated and B must attempt to reauthenticate with A in the BSS or IBSS.

Support for Broadcasting and Multicasting
The lack of acknowledgment for broadcast and multicast messages makes maintenance of state synchronization between nodes hard. Our future papers will investigate this issue in detail.

4. ANALYSIS OF SBKH PROTOCOL

We evaluate SBKH according to the following criteria, which are problematic in other protocols such as WEP or WPA: use of RC4, denial of service attacks, replay/modified packet attacks, key change knowledge, verification, implementation complexity, and power/processing costs or time costs.

4.1 Use of RC4

SBKH uses RC4 in a way which makes effective use of RC4's strengths and avoids most of its weaknesses. Instead of the problematic stateless approach, a single RC4 encryption stream is followed for multiple packets for each communication direction of each pair of communicating nodes, starting at $IRC4_U$ and $IRC4_D$, and without exchanging key-specific or state-specific knowledge. SBKH communicating nodes maintain state and

follow specific protocols to ensure the states remain synchronized.

Pair wise independence of communication encryption creates a strong encryption protocol, even if the listener is an insider, while maintaining synchronization between communicating pairs. We have shown through model checking of the protocol that senders and receivers stay synchronized, except for situations involving hard shutdown where state may become lost or an active interloper forcing loss of synchronizations. For such situations the resynchronization protocol lets communication nodes re-establish direct communication.

4.2 Denial of Service (DoS) Attacks

SBKH is much less susceptible to denial service attacks than are either WEP or WPA, since more of the security protocol is private to the parties, as follows:

Fake (dis)associations: Disassociations and associations must contain nonce encrypted at $CRC4_{U/D}$ using the same strong scheme as data messages; hence any such in-the-clear messages or improperly encrypted messages will be ignored or recorded as an active attack.

Fake authentications: Authentications contain a nonce encrypted at IRC4 or SRC4 to avoid spoofing of such messages in SBKH.

4.2.1 DoS Attack through Resynchronization

SBKH has introduced a new portion of the protocol called resynchronization, which might be triggered due to acknowledgement spoofing. One might think that by triggering the resynchronization quite often between a communicating pair there is a potential DoS attack within SBKH. It should be noted that this spoofing will successfully trigger resynchronization only if the receiver node did not receive the original packet or if the receiver received the original packet with channel errors.

To desynchronize two communicating nodes, an intruder must force the receiver to fail reception of the message, and then must generate false ACK packets for the transmitter using the receiver's MAC address and the correct SSC, all without detection by the receiving node or the transmitting node. It is not enough to do only ACK spoofing: either the intruder will only be replacing the receiver's ACK with his own and doing no damage, or the receiver will detect the attack and can notify the transmitter to take countermeasures.

Therefore, we believe that such an active attack is unlikely, and that resynchronization protocol usage will be rare. Further investigation such as countermeasures in the case of such DoS attacks will be presented in future articles.

4.3 Replay and Modified Packet Attacks

Replay attacks of encrypted packets assume that the decryption stream is still valid. Under SBKH, the encryption point within the cipher stream changes as soon as a packet is acknowledged, and any replayed packets will fail encryption validity checks (ICV) and will be noted as an active attack, making replay attacks useless. Similarly, modified packet attacks will fail and be ignored or logged as active attacks. Note that these attacks were successful and unidentifiable in WEP.

4.4 No Key-change Knowledge in SBKH

With SBKH, almost all knowledge of key indices, initializations and authorizations, and key changes are implicit in the protocol and cannot be determined by an analysis of in-the-clear messages. Specifically,

(i) The first few bytes are discarded and encryption begins at some later position in each encryption stream
(ii) Encryption and Decryption for different packets are at different points within a cipher stream
(iii) Key-hopping depends only upon a private Key Duration parameter known only to the communicating pair
(iv) Data, either from higher protocols or that form part of SBKH, is never presented in the clear and also encrypted, leaving less clues for eavesdroppers.

By using this approach, it is computationally hard for an interloper to find the decryption location, even if they had the same *BaseKey*.

4.5 Verification of SBKH

Since SBKH is a pair-wise state-based protocol, the communicating node pair must maintain an exact copy of the RC4 state so that the next set of encryption octets can be matched. If a node gets out of synchronization by even a single octet, then encryption synchronization is lost and recovery requires the resynchronization protocol.

The challenge, therefore, is to verify that SBKH nodes keep the same place in the encryption chain even when corrupted packets, timeouts, retransmits, and key rolling occur.

To verify the correct operation of SBKH, we subjected significant portions of the protocol to formal verification using the Promela formal specification language and the SPIN model checker [SPIN 2003]. This approach performs static analysis of all of the interleavings of the two nodes encrypting messages via a state-based encryption and exchanging those messages via a medium. We modeled message corruption due to channel errors and retransmission as well as behavior at key rolling and confirmed that the protocol is robust over these domains. Through this analysis we confirmed that encryption states must be maintained by each node in each direction and that a pairwise message counter (SSC) improves efficiency and eliminates retransmissions due to wrong state selection for decryption. We also determined through this process that an active attacker could send false acknowledgments for corrupted packets to one node and force it to lose encryption synchronization. We therefore developed a resynchronization protocol, discussed in subsection [3.3]. We have not yet formally verified the resynchronization.

4.6 Implementation Complexity

SBKH is both simpler and more complex than existing protocols, such as WEP or WPA. It is simpler in that the key initialization step is not required on the creation or reception of each packet. This translates directly to some simplification and likely power saving (for battery-operated systems) for systems that use SBKH, as well as improved performance over other protocols. It is more complex in that more state is required to be maintained for decryption, specifically each non-access point node must maintain 4 encryption RC4 states for each node with which it is communicating, consisting of 258 bytes plus possible ancillary information. This additional storage should amount to a very few extra kilobytes for every other actively communicating node. We do acknowledge the need for encrypted management message such as a Deauthentication message to indicate release of resources tied to a node that is leaving the network.

Considering likely hardware support for RC4, we believe that it should be easy to selectively generate RC4 state from a Base Key or use a pre-generated RC4 state. Hence, SBKH should be compatible with existing hardware.

All things considered, we believe that SBKH is competitive with existing and proposed encryption technologies.

4.7 Implementation Experience

We have implemented the encryption scheme using standard RC4 libraries on worstation-class processors, and have modeled the encryption, transmission, reception, decryption, and key-hopping parts of the protocol using a model checker. Some issues that we uncovered led us to propose a formal association / disassociation and a resynchronization phase. The issues uncovered were not with the basic protocol, but with a consideration of implementation errors or some possible active attacks.

4.8 Power/processing Costs or Time Costs

SBKH is less expensive in terms of power and CPU resources of the transmitter/receiver than is WEP, and is significantly cheaper than WPA. Our comparisons [SM2 2004] of WEP-based RC4 encryptions, WPA-like encryptions and SBKH on standard desktop and workstation computers predicts a 45% efficiency compared to WEP, 57% efficiency compared to WPA 1.0 and 60% efficiency compared to WPA 2.0 at packet size of about 200 bytes. Since the average packet size for most networks is less than 200 bytes [PK 2003], SBKH should show significant reductions for most 802.11 networks, but especially for small-packet networks where the data exchanged is in the tens of bytes as in wireless sensor networks.

The basic protocol as described herein, we believe is robust, internally consistent, and efficient.

5. CONCLUSION

SBKH implements encryption in a novel state based way so as to provide cheap and robust security without additional overheads of encryption. SBKH saves significant processing power especially for packet sizes smaller than 200 bytes as would be seen in wireless networks by avoiding state initialization on every packet. Hence we recommend SBKH for battery-operated devices such as wireless sensor nodes where processing power savings directly translates to longer battery life which implies longevity of the nodes. SBKH also provides ease of maintenance: a simple implementation may choose to maintain the same key forever (Key Duration = ∞). Due to its ease of maintenance, we also recommend SBKH for SOHO users.

6. Future Work

There are a few areas associated with SBKH which need further investigation:

(i) The formal model using the model checker has served us well, but is near its limits in computability. An alternative approach is to use a theorem proving approach to validate the protocol.

(ii) Implement SBKH on real hardware.

(iii) A strong key generation and distribution recommendation would be very useful.

(iv) Further investigation of countermeasures to protect against active attacks is also necessary.

7. Bibliography

[Draft 2003] Draft Amendment to STANDARD FOR Telecommunications and Information Exchange Between Systems – LAN/MAN Specific Requirements – Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: MAC Security Enhancements, IEEE Std. 802.11i/D7.0, Oct. 2003.

[IEEE802.11 1999]Telecommunications and Information Exchange Between Systems – Local and Metropolitan Networks – MAC and PHY Control, IEEE Press 1999.

[IEEE802.11 2001] IEEE Standard for Local and metropolitan area networks – Port-Based Network Access Control, Oct. 2001.

[FM 2000] Fluhrer S. and McGrew D. Statistical Analysis of the Alleged RC4 Keystream Generator, FSE: Fast Software Encryption, FSE2000, Springer-Verlag, 2000.

[FMS 2001] Fluhrer S., Mantin I., Shamir I., Weaknesses in the key scheduling algorithm of RC4, SAC'2001, 2001.

[Mantin 2001] Mantin I., Analysis of the Stream Cipher RC4. Weizmann Institute of Science, Nov. 2001.

[Moskowitz 2003] Moskowitz R., Simple Secrets/Simple Security, ICSA Labs, 2003.

[PK 2003] Prasithsangaree P., Krishnamurthi P., Analysis of Energy Consumption of RC4 and AES Algorithms in Wireless LANs, Global Telecommns. Conf., Globecom '03, Dec. 2003.

[SM1 2004] Srinivasan K., Michell S., State Based Key Hop (SBKH) Protocol, Wireless 2004 Conference, Jul. 2004.

[SM2 2004] Srinivasan K., Michell S., Performance of State Based Key Hop (SBKH) Protocol for Security on Wireless Networks, IEEE Vehicular Technology Conference 2004 Fall, Sep. 2004.

[SIR 2001] Stubblefield A., Ioannidis J., and Rubin A. D., Using the Fluhrer, Mantin, and Shamir Attack to Break WEP, AT&T Labs Technical Report TD-4ZCPZZ, Aug. 2001.

[SPIN 2003] The SPIN Model Checker: Primer and Reference Manual, Addison Wesley, 2003.

[Walker 2002] Walker J., 802.11 Security Series – Part II: The Temporal Key Integrity Protocol (TKIP), Intel Corporation, 2002.