Packet Leashes: A Defense against Wormhole Attacks in Wireless Networks

Yih-Chun Hu
Carnegie Mellon University
yihchun@cs.cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

David B. Johnson
Rice University
dbj@cs.rice.edu

***Abstract*— As mobile ad hoc network applications are deployed, security emerges as a central requirement. In this paper, we introduce the *wormhole attack*, a severe attack in ad hoc networks that is particularly challenging to defend against. The wormhole attack is possible even if the attacker has not compromised any hosts, and even if all communication provides authenticity and confidentiality. In the wormhole attack, an attacker records packets (or bits) at one location in the network, tunnels them (possibly selectively) to another location, and retransmits them there into the network. The wormhole attack can form a serious threat in wireless networks, especially against many ad hoc network routing protocols and location-based wireless security systems. For example, most existing ad hoc network routing protocols, without some mechanism to defend against the wormhole attack, would be unable to find routes longer than one or two hops, severely disrupting communication. We present a new, general mechanism, called *packet leashes*, for detecting and thus defending against wormhole attacks, and we present a specific protocol, called TIK, that implements leashes.**

I. Introduction

The promise of mobile ad hoc networks to solve challenging real-world problems continues to attract attention from industrial and academic research projects. Applications are emerging and widespread adoption is on the horizon. Most previous ad hoc networking research has focused on problems such as routing and communication, assuming a trusted environment. However, many applications run in untrusted environments and require secure communication and routing. Applications that may require secure communications include emergency response operations, military or police networks, and safety-critical business operations such as oil drilling platforms or mining operations. For example, in emergency response operations such as after a natural disaster like a flood, tornado, hurricane, or earthquake, ad hoc networks could be used for real-time safety feedback; regular communication networks may be damaged, so emergency rescue teams might rely upon ad hoc networks for communication.

Ad hoc networks generally use a wireless radio communication channel. The main advantages of such networks are rapid deployment and low cost of operation, since the nodes and wireless hardware are inexpensive and readily available, and since the network is automatically self-configuring and self-maintaining. However, wireless networks are vulnerable to several attacks. In most wireless networks, an attacker can easily *inject* bogus packets, impersonating another sender. We refer to this attack as a *spoofing* attack. An attacker can also easily *eavesdrop* on communication, record packets, and *replay* the (potentially altered) packets.

In this paper, we define a particularly challenging attack to defend against, which we call a *wormhole* attack, and we present a new, general mechanism for detecting and thus defending against wormhole attacks. In this attack, an attacker records a packet, or individual bits from a packet, at one location in the network, tunnels the packet (possibly selectively) to another location, and replays it there. The wormhole attack can form a serious threat in wireless networks, especially against many ad hoc network routing protocols and location-based wireless security systems. The wormhole places the attacker in a very powerful position, able for example to further exploit any of the attacks mentioned above, allowing the attacker to gain unauthorized access, disrupt routing, or perform a Denial-of-Service (DoS) attack. We introduce the general mechanism of *packet leashes* to detect wormhole attacks, and we present two types of leashes: *geographic leashes* and *temporal leashes*. Finally, we design an efficient authentication protocol, called TIK, for use with temporal leashes. We focus our discussion in this paper on wireless ad hoc networks, but our results are applicable more broadly to other types of networks, such as wireless LANs and cellular networks.

Section II of this paper presents the wormhole attack and discusses how the wormhole attack can be used against ad hoc network routing protocols. In Section III, we present our assumptions. Section IV presents leashes and discusses a general approach for detecting wormholes. Section V discusses temporal leashes in detail and presents the TIK protocol for instant wireless broadcast authentication, and Section VI provides an evaluation of TIK and packet leashes. Section VII discusses related work, and Section VIII presents our conclusions.

II. Problem Statement

In a *wormhole attack*, an attacker receives packets at one point in the network, "tunnels" them to another point in the network, and then replays them into the network from that

This work was supported in part by NSF under grant CCR-0209204, by NASA under grant NAG3-2534, and by gifts from Schlumberger and Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, NASA, Schlumberger, Bosch, Rice University, Carnegie Mellon University, or the U.S. Government or any of its agencies.

point. For tunneled distances longer than the normal wireless transmission range of a single hop, it is simple for the attacker to make the tunneled packet arrive sooner than other packets transmitted over a normal multihop route, for example through use of a single long-range directional wireless link or through a direct wired link to a colluding attacker. It is also possible for the attacker to forward each bit over the wormhole directly, without waiting for an entire packet to be received before beginning to tunnel the bits of the packet, in order to minimize delay introduced by the wormhole. Due to the nature of wireless transmission, the attacker can create a wormhole even for packets not addressed to itself, since it can overhear them in wireless transmission and tunnel them to the colluding attacker at the opposite end of the wormhole.

If the attacker performs this tunneling honestly and reliably, no harm is done; the attacker actually provides a useful service in connecting the network more efficiently. However, the wormhole puts the attacker in a very powerful position relative to other nodes in the network, and the attacker could exploit this position in a variety of ways. The attack can also still be performed even if the network communication provides confidentiality and authenticity, and even if the attacker has no cryptographic keys. Furthermore, the attacker is invisible at higher layers; unlike a malicious node in a routing protocol, which can often easily be named, the presence of the wormhole and the two colluding attackers at either endpoint of the wormhole are not visible in the route. As such, the effect of the wormhole on legitimate nodes may even change as nodes move; two legitimate nodes previously connected only by routes through the wormhole and thus possibly unable to communicate, will be able to communicate normally if they come within direct wireless transmission range of each other.

The wormhole attack is particularly dangerous against many ad hoc network routing protocols in which the nodes that hear a packet transmission directly from some node consider themselves to be in range of (and thus a neighbor of) that node. For example, when used against an on-demand routing protocol such as DSR [21] or AODV [33], a powerful application of the wormhole attack can be mounted by tunneling each ROUTE REQUEST packet directly to the destination target node of the REQUEST. When the destination node's neighbors hear this REQUEST packet, they will follow normal routing protocol processing to rebroadcast that copy of the REQUEST and then discard without processing all other received ROUTE REQUEST packets originating from this same Route Discovery. This attack thus prevents any routes other than through the wormhole from being discovered, and if the attacker is near the initiator of the Route Discovery, this attack can even prevent routes more than two hops long from being discovered. Possible ways for the attacker to then exploit the wormhole include discarding rather than forwarding all data packets, thereby creating a permanent Denial-of-Service attack (no other route to the destination can be discovered as long as the attacker maintains the wormhole for ROUTE REQUEST packets), or selectively discarding or modifying certain data packets.

The neighbor discovery mechanisms of periodic (proactive) routing protocols such as DSDV [32], OLSR [38], and TBRPF [5] rely heavily on the reception of broadcast packets as a means for neighbor detection, and are also extremely vulnerable to this attack. For example, OLSR and TBRPF use HELLO packets for neighbor detection, so if an attacker tunnels through a wormhole to a colluding attacker near node B all HELLO packets transmitted by node A, and likewise tunnels back to the first attacker all HELLO packets transmitted by B, then A and B will believe that they are neighbors, which would cause the routing protocol to fail to find routes when they are not actually neighbors.

For DSDV, if each routing advertisement sent by node A or node B were tunneled through a wormhole between colluding attackers near these nodes, as described above, then A and B would believe that they were neighbors. If A and B, however, were not within wireless transmission range of each other, they would be unable to communicate. Furthermore, if the best existing route from A to B were at least $2n + 2$ hops long, then any node within n hops of A would be unable to communicate with B, and any node within n hops of B would be unable to communicate with A. Otherwise, suppose C were within n hops of A, but had a valid route to B. Since A advertises a metric of 1 route to B, C would hear a metric $n+1$ route to B. C will use that route if it is not within $n+1$ hops of B, in which case there would be an n-hop route from A to C, and a route of length $n+1$ from C to B, contradicting the premise that the best real route from A to B is at least $2n + 2$ hops long.

The wormhole attack is also dangerous in other types of wireless networks and applications. One example is any wireless access control system that is based on physical proximity, such as wireless car keys, or proximity and token based access control systems for PCs [11, 24]. In such systems, an attacker could relay the authentication exchanges to gain unauthorized access.

One partial approach for preventing wormhole attacks might be to use a secret method for modulating bits over wireless transmissions; once a node is compromised, however, this approach is likely to fail unless the radio is kept inside tamper-resistant hardware. Another approach, known as RF watermarking, authenticates a wireless transmission by modulating the RF waveform in a way known only to authorized nodes [12]. RF watermarking relies on keeping secret the knowledge of which RF waveform parameters are being modulated; furthermore, if that waveform is exactly captured at the receiving end of the wormhole and exactly replicated at the transmitting end of the wormhole, the signal level of the resulting watermark is independent of the distance it was tunneled. As a result, the watermark may still be intact, even though the packet was made to travel beyond the normal wireless transmission range. Although intrusion detection could be used in some cases to detect a wormhole attacker, it is generally difficult to isolate the attacker in a software-only approach, since the packets sent by the wormhole are identical to the packets sent by legitimate nodes. In contrast to these

approaches, the approach we present in this paper, called *packet leashes*, and the specific protocol we present, called TIK, provide a general solution that does not suffer from these problems.

III. Assumptions and Notation

The acronym "MAC" may in general stand for "Medium Access Control" protocol or "Message Authentication Code." To avoid confusion, we use "MAC" in this paper to refer to the network Medium Access Control protocol at the link layer, and we use "HMAC" to refer to a message authentication code used for authentication (HMAC is a particular instance of a message authentication code [4]).

For reasons such as differences in wireless interference, transmit power, or antenna operation, links between nodes in a wireless network may at times successfully work in only one direction; such a *unidirectional* wireless link between between two nodes A and B might allow A to send packets to B but not for B to send packets to A. In many cases, however, wireless links are able to operate as *bidirectional* links. A MAC protocol generally is designed to support operation over unidirectional links or is designed only for bidirectional links; the introduction of our TIK protocol does not affect the capability of the MAC protocol to operate over unidirectional links.

Security attacks on the wireless network's physical layer are beyond the scope of this paper. Spread spectrum has been studied as a mechanism for securing the physical layer against jamming [36]. Denial-of-Service (DoS) attacks against MAC layer protocols are also beyond the scope of the paper; MAC layer protocols that do not employ some form of carrier sense, such as pure ALOHA and Slotted ALOHA [1], are less vulnerable to DoS attacks, although they tend to use the channel less efficiently.

We assume that the wireless network may drop, corrupt, duplicate, or reorder packets. We also assume that the MAC layer contains some level of redundancy to detect randomly corrupted packets; however, this mechanism is not designed to replace cryptographic authentication mechanisms.

We assume that nodes in the network may be resource constrained. Thus, in providing for wormhole detection, we use efficient *symmetric* cryptography, rather than relying on expensive *asymmetric* cryptographic operations. Especially on CPU-limited devices, symmetric cryptographic operations (such as block ciphers and hash functions) are three to four orders of magnitude faster than asymmetric cryptographic operations (such as digital signatures).

We assume that a node can obtain an authenticated key for any other node. Like public keys in systems using asymmetric cryptography, these keys in our protocol TIK (Section V) are public values (once disclosed), although TIK uses only symmetric (not asymmetric) cryptography. A traditional approach to this authenticated key distribution problem is to build on a public key system for key distribution; a trusted entity can sign public-key certificates for each node, and the nodes can then use their public-key to sign a new (symmetric) key being distributed for use in TIK. Zhou and Haas [46] propose such a public key infrastructure; Hubaux, Buttyán, and Čapkun bootstrap trust relationships from PGP-like certificates without relying on a trusted public key infrastructure [19]; Kong et al [25] propose asymmetric mechanisms for threshold signatures for certificates. Alternatively, a trusted node can securely distribute an authenticated TIK key using only symmetric-key cryptography [35] or non-cryptographic approaches [42].

IV. Detecting Wormhole Attacks

In this section, we introduce the notion of a *packet leash* as a general mechanism for detecting and thus defending against wormhole attacks. A leash is any information that is added to a packet designed to restrict the packet's maximum allowed transmission distance. We distinguish between *geographical leashes* and *temporal leashes*. A geographical leash ensures that the recipient of the packet is within a certain distance from the sender. A temporal leash ensures that the packet has an upper bound on its lifetime, which restricts the maximum travel distance, since the packet can travel at most at the speed of light. Either type of leash can prevent the wormhole attack, because it allows the receiver of a packet to detect if the packet traveled further than the leash allows.

A. Geographical Leashes

To construct a geographical leash, in general, each node must know its own location, and all nodes must have loosely synchronized clocks. When sending a packet, the sending node includes in the packet its own location, p_s, and the time at which it sent the packet, t_s; when receiving a packet, the receiving node compares these values to its own location, p_r, and the time at which it received the packet, t_r. If the clocks of the sender and receiver are synchronized to within $\pm\Delta$, and ν is an upper bound on the velocity of any node, then the receiver can compute an upper bound on the distance between the sender and itself, d_{sr}. Specifically, based on the timestamp t_s in the packet, the local receive time t_r, the maximum relative error in location information δ, and the locations of the receiver p_r and the sender p_s, then d_{sr} can be bounded by $d_{sr} \leq ||p_s - p_r|| + 2\nu \cdot (t_r - t_s + \Delta) + \delta$. A standard digital signature scheme or other authentication technique can be used to enable a receiver to authenticate the location and timestamp in the received packet. This approach is similar to [13].

In certain circumstances, bounding the distance between the sender and receiver, d_{sr}, cannot prevent wormhole attacks; for example, when obstacles prevent communication between two nodes that would otherwise be in transmission range, a distance-based scheme would still allow wormholes between the sender and receiver. A network that uses location information to create a geographical leash could control even these kinds of wormholes. To accomplish this, each node would have a radio propagation model. A receiver could verify that every possible location of the sender (a $\delta + \nu(t_r - t_s + 2\Delta)$ radius around p_s) can reach every possible location of the receiver (a $\delta + \nu(t_r - t_s + 2\Delta)$ radius around p_r).

B. Temporal Leashes

To construct a temporal leash, in general, all nodes must have tightly synchronized clocks, such that maximum difference between any two nodes' clocks is Δ. The value of the parameter Δ must be known by all nodes in the network, and for temporal leashes, generally must be on the order of a few microseconds or even hundreds of nanoseconds. This level of time synchronization can be achieved now with off-the-shelf hardware based on LORAN-C [30], WWVB [31], or GPS [10, 44]; although such hardware is not currently a common part of wireless network nodes, it can be deployed in networks today and is expected to become more widely utilized in future systems at reduced expense, size, weight, and power consumption. In addition, the time synchronization signal itself in such systems may be subject to certain attacks [6, 14]. Esoteric hardware such as cesium-beam clocks, rubidium clocks, and hydrogen maser clocks, could also be used in special applications today to provide sufficiently accurate time synchronization for months. Although our general requirement for time synchronization is indeed a restriction on the applicability of temporal leashes, for applications that require defense against the wormhole attack, this requirement is justified due to the seriousness of the attack and its potential disruption of the intended functioning of the network.

To use temporal leashes, when sending a packet, the sending node includes in the packet the time at which it sent the packet, t_s; when receiving a packet, the receiving node compares this value to the time at which it received the packet, t_r. The receiver is thus able to detect if the packet traveled too far, based on the claimed transmission time and the speed of light. Alternatively, a temporal leash can be constructed by instead including in the packet an expiration time, after which the receiver should not accept the packet; based on the allowed maximum transmission distance and the speed of light, the sender sets this expiration time in the packet as an offset from the time at which it sends the packet. As with a geographical leash, a regular digital signature scheme or other authentication technique can be used to allow a receiver to authenticate a timestamp or expiration time in the received packet.

C. Discussion

An advantage of geographical leashes over temporal leashes is that the time synchronization can be much looser. Another advantage of using geographical leashes in conjunction with a signature scheme (i.e., a signature providing non-repudiation), is that an attacker can be caught if it pretends to reside at multiple locations. This use of non-repudiation was also proposed by Sirois and Kent [41]. When a legitimate node overhears the attacker claiming to be in different locations that would only be possible if the attacker could travel at a velocity above the maximum node velocity ν, the legitimate node can use the signed locations to convince other legitimate nodes that the attacker is malicious.

We define $\delta'(t)$ to be a bound on the maximum relative position error when any node determines its own location twice within a period of time t. By definition, $\delta'(t) \leq 2\delta$. In addition, when t is small, $\delta'(t)$ should be small, since the algorithm a node uses to determine its location should be aware of physical speed limits of that node. If some node claims to be at locations p_1 and p_2 at times t_1 and t_2, respectively, that node is an attacker if $\frac{||p_2-p_1||-\delta'(|t_2-t_1|)}{|t_2-t_1|} > \nu$. A legitimate node detecting this from these two packets can also broadcast the two packets to convince other nodes that the first node is indeed an attacker. Each node hearing these messages can check the two signatures, verify the discrepancy in the information, and rebroadcast the information if it has not previously done so. To easily perform duplicate suppression in rebroadcasting this information, each node can maintain a *blacklist*, with each entry in the blacklist containing a node address and the time at which that blacklist entry expires. When a node receives a message showing an attacker's behavior, it checks if that attacker is already listed in its blacklist. If so, it updates the expiration time on its current blacklist entry and discards the new message; otherwise, it adds a new blacklist entry and propagates the message.

A potential problem with leashes using a timestamp in the packet is that in a contention-based MAC protocol, the sender may not know the precise time at which it will transmit a packet it is sending. For example, a sender using the IEEE 802.11 MAC protocol may not know the time a packet will be transmitted until approximately one slot time (20 μs) prior to transmission. Generating an inefficient digital signature, such as RSA with a 1024-bit key, could take three orders of magnitude more time than this slot time (on the order of 10 ms). The sender, however, can use two approaches to hide this signature generation latency: either increase the *minimum* transmission unit to allow computation to overlap with transmission, or use a more efficient signature scheme, such as Schnorr's signature [40], which enables efficient signature generation after pre-processing.

V. Temporal Leashes and the TIK Protocol

In this section, we discuss temporal leashes in more detail and present the design and operation of our TIK protocol that implements temporal leashes.

A. Temporal Leash Construction Details

We now discuss temporal leashes that are implemented with a packet expiration time. We consider a sender who wants to send a packet with a temporal leash, preventing the packet from traveling further than distance L. (All nodes are time synchronized up to a maximum time synchronization error Δ.) Thus, $L > L_{min} = \Delta \cdot c$, where c is the propagation speed of our wireless signal (i.e., the speed of light in air, which is very close to the speed of light in a vacuum). When the sender sends the packet at local time t_s, it needs to set the packet expiration time to $t_e = t_s + L/c - \Delta$. When the receiver receives the packet at local time t_r, it further processes the packet if the temporal leash has not expired (i.e., $t_r < t_e$); otherwise it drops the packet. This assumes that the packet sending and receiving delay are negligible, such that the sender can predict the precise sending time t_s and the receiver can

immediately record t_r when the first bit arrives (or derive t_r during reception since the bitrate of transmission is known).

The receiver needs a way to authenticate the expiration time t_e, as otherwise an attacker could easily change that time and wormhole the packet as far as it desires.

In unicast communication, (point-to-point) nodes can use *message authentication codes* for authentication: the sender S and receiver R must share a secret key K, which they use in conjunction with a message authentication code function (for example HMAC [4]) to authenticate messages they exchange. To send a message M to a receiver R, the sender S sends

$$S \rightarrow R: \quad \langle M, \text{HMAC}_K(M) \rangle \ ,$$

where the notation $\text{HMAC}_K(M)$ represents the message authentication code computed over message M with key K. The packet sent from S to R contains both the intended message M and $\text{HMAC}_K(M)$. When R receives this message, it can verify the authenticity of the message by comparing the received HMAC value to the HMAC value that it computes for itself over the received message with the secret key K it shares with the sender S.

However, using message authentication codes in the standard way has two major drawbacks. First, in a network with n nodes, we would need to set up $\frac{n(n-1)}{2}$ keys, one for each pair of nodes. Key setup is an expensive operation, which makes this approach impractical in large networks. Second, this approach cannot efficiently authenticate broadcast packets. To secure a broadcast packet, the sender would need to add to the packet a separate message authentication code for each receiver, making the packet extremely large (and likely exceeding the network's maximum packet size). The need to include separate message authentication codes in the packet could be avoided by having multiple receivers share the same key, but this might allow a subset of colluding receivers to impersonate the sender [9].

Instead, attaching a *digital signature* to each packet could be used to solve the two problems discussed above: each node needs to have only one public-private key pair, and each node needs to know only the public key of every other node. Thus, only n public keys need to be distributed in a network with n nodes. Furthermore, a digital signature provides nonrepudiation and authentication for broadcast packets in the same way as for unicast packets.

However, digital signatures have several drawbacks. First, digital signatures are usually based on computationally expensive *asymmetric* cryptography. For example, the popular 1024-bit RSA digital signature algorithm [39], roughly equivalent to use of a 72-bit key in a symmetric encryption algorithm [27], requires about 10 ms on an 800 MHz Pentium III processor for signature generation. Signature verification is more efficient, but still requires about 0.5 ms on a fast workstation. Adding a digital signature to each packet is computationally expensive for the verifier (receiver), but overwhelmingly expensive for the signer (sender). On less powerful CPUs, each digital signature generation and verification takes on the order of seconds [8].

Since many wireless applications rely heavily on broadcast communication, and since setting up $O(n^2)$ keys is expensive, we design the TIK protocol in Section V-C, based on a new protocol for efficient broadcast authentication that simultaneously provides the functionality of a temporal leash.

B. Tree-Authenticated Values

The TIK protocol we present in Section V-C requires an efficient mechanism for authenticating keys. In this section, we discuss the efficient hash tree authentication mechanism.

Authenticating a sequence of values efficiently is an important primitive used in many security protocols. One-way hash chains are predominantly used for this purpose. One of the first uses of one-way hash chains was for one-time passwords by Lamport [26], which Haller later used to design the S/KEY one-time password system [17]. To motivate why we use a tree structure instead of a one-way hash chain to authenticate values, we briefly describe the drawbacks of a one-way chain.

1) One-way Hash Chain: Consider the chain of length w with the values $C_0, \ldots, C_{w-1}$. We can generate this chain by randomly selecting the last value C_{w-1}, and repeatedly applying a one-way hash function H to derive the previous values: $C_{w-2} = H(C_{w-1})$, ..., $C_i = H(C_{i+1})$. The beginning of the chain, C_0, serves as a commitment to the entire chain and allows anybody to authenticate the following values of the chain. Because the function H is one-way and provides second pre-image collision resistance (also called weak collision resistance), it is computationally intractable for an attacker to invert H or to find another value $C_i' \neq C_i$, given C_i and C_{i-1}, that satisfies $C_{i-1} = H(C_i')$. (Menezes, van Oorschot, and Vanstone, have a more detailed discussion on one-way hash functions or the second pre-image collision resistance property [28].)

Therefore, if we know that one-way chain value C_i is authentic, and we later learn C_{i+1} with the property that $C_i = H(C_{i+1})$, we know that value C_{i+1} is authentic and is the value that follows C_i on the chain. More generally, we can verify C_j given the authentic value C_i by checking that $C_i = H^{j-i}(C_j)$, with $j > i$. Values from a one-way hash chain are very efficient to verify if we verify the values in sequence. However, for the TIK protocol we present in Section V-C, we would use the values very sparsely. Even though the one-way hash function is very efficient to compute, this computation would still require a substantial verification overhead — we thus use a tree structure for more efficient authentication of values.

2) Hash Tree: To authenticate the sequence of values $v_0, v_1, \ldots, v_{w-1}$, we place these values at the leaf nodes of a binary tree. (For simplicity, we assume a balanced binary tree, so w is a power of 2.) We first "blind" all the values with a one-way hash function H to prevent disclosing additional values (as we will describe below), so $v_i' = H(v_i)$ for each i. We then use the Merkle hash tree construction [29] to commit to the values $v_0', \ldots, v_{w-1}'$. Each internal node of the binary tree is derived from its two child nodes. Consider

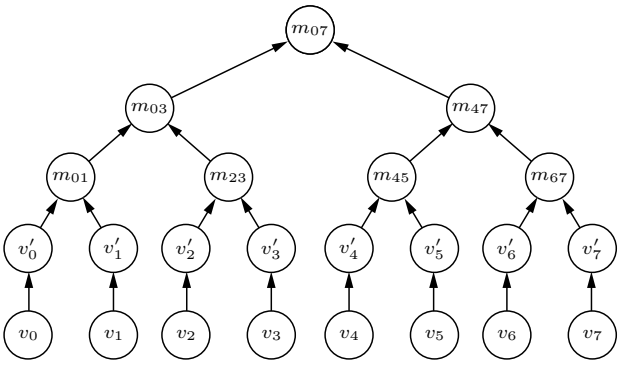

Fig. 1. Merkle hash tree

the derivation of the parent node m_p from the left and right child nodes, m_l and m_r, respectively: $m_p = H(m_l \mid\mid m_r)$. We compute each level of the tree recursively, from the leaf nodes to the root node. Figure 1 shows this construction over the eight values $v_0, v_1, \ldots, v_7$, with $m_{01} = H(v'_0 \mid\mid v'_1)$, $m_{03} = H(m_{01} \mid\mid m_{23})$, and so on.

The root value of the tree is used to authenticate all leaf values. To authenticate a value v_i, the sender discloses i, v_i, and all values necessary to verify the path up to the root of the tree. For example, if a sender wants to authenticate key v_2 in Figure 1, it includes the values v'_3, m_{01}, m_{47} in the packet. A receiver with an authentic root value m_{07} can then verify that

$$H\left[\, H\left[\, m_{01} \mid\mid H\left[\, H[\, v_2 \,] \mid\mid v'_3 \,\right]\right] \mid\mid m_{47}\right]$$

equals the stored m_{07}. If the verification is successful, the receiver knows that v_2 is authentic.

The extra $v'_0, v'_1, \ldots, v'_7$ in Figure 1 are added to the tree to avoid disclosing (in this example) the value v_3 in order to authenticate v_2.

3) Hash Tree Optimization: In TIK, the depth of the hash tree can be quite large: given a fixed time interval I, the tree is of depth $\lceil \log_2(t/I) \rceil$, where t is the amount of time between rekeying. For example, if the time interval is 11.5 μs and nodes can be rekeyed once per day, then the tree is of depth 34. As a result, storing the entire tree is impractical.

It is possible, however, to store only the upper layers of the tree and to recompute the lower layers on demand. To reconstruct a tree of depth d requires 2^{d-1} applications of the pseudo-random function (PRF) and $2^d - 1$ applications of the hash function, but this technique saves a factor of 2^{d-1} in storage. This technique can also be further improved by amortizing this calculation. Specifically, a node keeps two trees of depth d: one that is fully computed and currently being used, and one that is being filled in. Since a total of $2^{d-1} + 2^d - 1$ operations are required to fill in the tree, and the full tree will be used for 2^{d-1} time intervals, the node needs to perform only 3 operations per time interval, independent of the size of the tree.

We can now compute the true calculation and storage cost for the hash tree that we use in TIK. Let D be the depth of the entire tree, and let d be the depth of the part of the tree that is recomputed on demand. The initial computation of the tree requires 2^{D-1} evaluations of the PRF, and $2^D - 1$ evaluations of the hash function. This initial computation can be done offline and is not time-critical. To choose d, we consider the value of d that minimizes the total storage needed for the tree. Since total storage is given by $2^{D-d+1} - 1 + 2 \cdot (2^d - 1)$, storage for the tree is minimized when

$$\begin{aligned} \frac{\partial}{\partial d}\left(2^{D-d+1} - 1 + 2^{d+1} - 2\right) &= 0 \\ (-\ln 2)2^{D-d+1} + (\ln 2)2^{d+1} &= 0 \\ 2^{d+1} &= 2^{D-d+1} \\ d+1 &= D-d+1 \ . \end{aligned}$$

The optimal choice for d is $\frac{D}{2}$, and the total storage requirement for the tree is $2^{\lceil \frac{D}{2} \rceil + 1} + 2^{\lfloor \frac{D}{2} \rfloor + 1} - 3$. This represents a storage requirement of just $O(\sqrt{t/I})$. For example, a tree of depth 34 requires only 2.5 megabytes to store, much smaller than the full tree size of 170 gigabytes; once the tree is generated, it can be used at a cost of 3 operations per time interval.

A similar approach can be taken for the generation of future hash trees. That is, once a single hash tree has been generated, each future hash tree can be generated while the current one is used, for a cost of 3 hash functions per time interval plus total storage space for the tree of $2^{\lceil \frac{D}{2} \rceil + 1} + 2^{\lfloor \frac{D}{2} \rfloor + 1} - 2$. Only the root of each new tree needs to be distributed, and as mentioned in Section III, these values can be distributed using only symmetric-key cryptography [35], non-cryptographic approaches [42], or by sending them using the current hash tree for authentication.

C. TIK Protocol Description

Our TIK protocol implements temporal leashes and provides efficient instant authentication for broadcast communication in wireless networks. TIK stands for *TESLA with Instant Key disclosure*, and is an extension of the TESLA broadcast authentication protocol [34]. We contribute the novel observation that a receiver can verify the TESLA *security condition* (that the corresponding key has not yet been disclosed) as it receives the packet (explained below); this fact allows the sender to disclose the key in the same packet, thus motivating the protocol name "TESLA with Instant Key disclosure."

TIK implements a temporal leash and thus enables the receiver to detect a wormhole attack. TIK is based on efficient *symmetric* cryptographic primitives (a message authentication code is a symmetric cryptographic primitive). TIK requires accurate time synchronization between all communicating parties, and requires each communicating node to know just one public value for each sender node, thus enabling scalable key distribution.

We now describe the different stages of the TIK protocol in detail: sender setup, receiver bootstrapping, and sending and verifying authenticated packets.

1) Sender Setup: The sender uses a pseudo-random function (PRF [15]) $\mathcal{F}$ and a secret master key $\mathcal{X}$ to derive a series of keys $K_0, K_1, \ldots, K_w$, where $K_i = \mathcal{F}_{\mathcal{X}}(i)$. The main advantage of this method of key generation is that the sender can efficiently access the keys in any order. Assuming the PRF is secure, it is computationally intractable for an attacker to find the master secret key $\mathcal{X}$, even if all keys $K_0, K_1, \ldots, K_{w-1}$ are known. Without the secret master key $\mathcal{X}$, it is computationally intractable for an attacker to derive a key K_i that the sender has not yet disclosed. To construct the PRF function $\mathcal{F}$, we can use a pseudo-random permutation, i.e., a block cipher [16], or a message authentication code, such as HMAC [4].

The sender selects a key expiration interval I, and thus determines a schedule with which each of its keys will expire. Specifically, key K_0 expires at time T_0, key K_1 expires at time $T_1 = T_0 + I$, ..., key K_i expires at time $T_i = T_{i-1} + I = T_0 + i \cdot I$.

The sender constructs the Merkle hash tree we describe in Section V-B to commit to the keys $K_0, K_1, \ldots, K_{w-1}$. The root of the resulting hash tree is $m_{0,w-1}$, or simply m. The value m commits to all keys and is used to authenticate any leaf key efficiently. As we describe in Section V-B, in a hash tree with $\log_2(w)$ levels, verification requires only $\log_2 w$ hash function computations (in the worst case, not considering buffering), and the authentication information consists of $\log_2 w$ values.

2) Receiver Bootstrapping: We assume that all nodes have synchronized clocks with a maximum clock synchronization error of Δ. We further assume that each receiver knows every sender's hash tree root m, and the associated parameters T_0 and I. This information is sufficient for the receiver to authenticate any packets from the sender.

3) Sending and Verifying Authenticated Packets: To achieve secure broadcast authentication, it must not be possible for a receiver to forge authentication information for a packet. When the sender sends a packet P, it estimates an upper bound t_r on the arrival time of the HMAC at the receiver. Based on this arrival time, the sender picks a key K_i that will not have expired when the receiver receives the packet's HMAC ($T_i > t_r + \Delta$). The sender attaches the HMAC to the packet, computed using key K_i, and later discloses the key K_i itself, along with the corresponding tree authentication values (as discussed in Section V-B), after the key has expired.

Because of the time synchronization, the receiver can verify after receiving the packet that the key K_i used to compute the authentication has not yet been disclosed, since the receiver knows the expiration time for each key and the sender only discloses the key after it expires; thus, no attacker can know K_i, and therefore if the packet authentication verifies correctly once the receiver later receives the authentic key K_i, the packet must have originated from the claimed sender. Even another receiver could not have forged a new message with a correct message authentication code, since only the sender knew the key K_i at the time t_r that the receiver received the packet. After the key K_i expires at time T_i, the sender then discloses key K_i (and the corresponding tree authentication values); once the receiver gets the authentic key K_i, it can authenticate all packets that carry a message authentication code computed with K_i. This use of delayed key disclosure and time synchronization for secure broadcast authentication was also used by the TESLA protocol [34].

The above protocol has the drawback that message authentication is delayed; the receiver must wait for the key before it can authenticate the packet. We observe that we can remove the authentication delay in an environment in which the nodes are tightly time synchronized. In fact, the sender can even disclose the key in the same packet that carries the corresponding message authentication code.

Figure 2 shows the sending and receiving of a TIK packet. The figure shows the sender's and receiver's timelines, which may differ by a value of up to the maximum time synchronization error Δ. The time t_s here is the time at which the sender S begins transmission of the packet, and time T_i is the disclosure time for key K_i. The packet contains four parts: a message authentication code (shown as HMAC in Figure 2), a message payload (shown as M), the tree authentication values necessary to authenticate K_i (shown as T), and the key used to generate the message authentication code (shown as K_i). The TIK packet is transmitted by S as

$$S \rightarrow R: \quad \langle \text{HMAC}_{K_i}(M), M, T, K_i \rangle \ ,$$

where the destination R may be unicast or broadcast. After the receiver R receives the HMAC value, it verifies that the sender did not yet start sending the corresponding key K_i, based on the time T_i and the synchronized clocks. If the sender did not yet start sending K_i, the receiver verifies that the key K_i at the end of the packet is authentic (using the hash tree root m and the hash tree values T), and then uses K_i to verify the HMAC value in the packet. If all these verifications are successful, the receiver accepts the packet as authentic.

The TIK protocol already provides protection against the wormhole attack, since an attacker who retransmits the packet will most likely delay it long enough that the receiver will reject the packet because the corresponding key has already expired and the sender may have disclosed it. However, we can also add an explicit expiration timestamp to each packet for the temporal leash, and use TIK as the authentication protocol. For example, each packet could include a 64-bit timestamp with nanosecond resolution, allowing over 580 years of use starting from the epoch. Since the entire packet is authenticated, the timestamp is authenticated.

A policy could be set allowing the reception of packets for which the perceived transmission delay, i.e., the arrival time minus the sending timestamp, is less than some threshold. That threshold could be chosen anywhere between $\tau - \Delta$ and $\tau + \Delta$, where the more conservative approach of $\tau - \Delta$ never allows tunnels but rejects some valid packets, and the more liberal approach of $\tau + \Delta$ never rejects valid packets, but may allow tunneling of up to $2c\Delta$ past the actual normal transmission range.

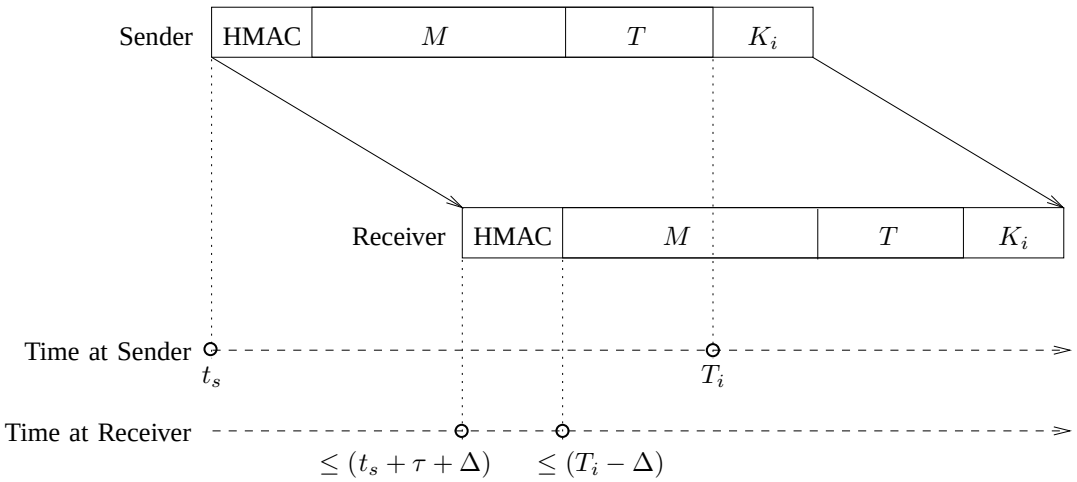

Fig. 2. Timing of a packet in transmission using TIK

With a GPS-disciplined clock [44], time synchronization to within $\Delta = 183$ ns with probability $1-10^{-10}$ is possible. If a transmitter has a 250 m range, the $\tau - \Delta$ threshold accepts all packets sent less than 140 m and some packets sent between 140 and 250 m; the $\tau + \Delta$ threshold accepts all packets sent less than 250 m but allows tunneling of packets up to 110 m beyond that distance.

D. MAC Layer Considerations

A TDMA MAC protocol may be able to choose the time at which a frame begins transmission, so that the message authentication code is sent by time $T_i - \frac{r}{c} - 2\Delta$. In this case, the minimum payload length is $\frac{r}{c} + 2\Delta$ times the bit rate of transmission. For additional efficiency, different nodes should have different key disclosure times, and the MAC layer should provide each node with the MAC layer time slot it needs for authenticated delivery.

As mentioned in Section V-C, a CSMA MAC protocol may not be able to control that time at which a frame is sent relative to the key disclosure times. In this case, the minimum payload length needs to be chosen so that a key disclosure time is guaranteed to occur somewhere during the packet's transmission. For example, if the network physical layer is capable of a peak data rate of 100 Mbps and a range of 150 m, and if the key disclosure interval is chosen to be 25 μs and time synchronization is achieved to within 250 ns, then the minimum packet size must be at least 325 bytes. However, if each value in the hash tree is 80 bits long, and the depth of the tree is 31, then the minimum payload size is just 15 bytes.

If a MAC protocol uses a Request-to-Send/Clear-to-Send (RTS/CTS) frame handshake, the minimum packet size can be reduced by carrying the message authentication code inside the RTS frame. In this case, the frame exchange for transmitting a data packet would be

$$\begin{aligned} A \rightarrow B: &\quad \langle \text{RTS}, \text{HMAC}_{K_i}(M) \rangle \\ B \rightarrow A: &\quad \langle \text{CTS} \rangle \\ A \rightarrow B: &\quad \langle \text{DATA}, M, \textit{tree values}, K_i \rangle \ . \end{aligned}$$

In particular, instead of having a minimum message size of $\frac{r}{c} + 2\Delta + I$ times the transmission data rate, where I is the duration of a time interval, the minimum message size is just $2\Delta + I - 2t_{\textit{turn}}$ times the data rate, where $t_{\textit{turn}}$ is the minimum allowed time between receiving a control frame (i.e., the RTS or CTS) and returning a corresponding frame (the CTS or DATA frame, respectively). This minimum message length includes the length of the CTS, DATA header, payload, and hash tree values.

VI. Evaluation

A. TIK Performance

To evaluate the suitability of our work for use in ad hoc networks, we measured computational power and memory currently available in mobile devices. To measure the number of repeated hashes that can be computed per second, we optimized the MD5 hash code from ISI [43] to achieve maximum performance for repeated hashing.

Our optimized version performs 10 million hash function evaluations in 7.544 s on a Pentium III running at 1 GHz, representing a rate of 1.3 million hashes per second; the same number of hashes using this implementation on a Compaq iPaq 3870 PocketPC running Linux took 45 s, representing a rate of 222,000 hashes per second. Repetitive, simple functions like hashes can also be efficiently implemented in hardware; Helion Technology [18] claims a 20k gate ASIC core design (a third the complexity of Bluetooth [3] and less than a third the complexity of IEEE 802.11 [23]) capable of more than 1.9 million hashes per second and a Xilinx FPGA design using 1650 LUTs capable of 1 million hashes per second. In terms of memory consumption, existing handheld devices, such as the iPaq 3870, come equipped with 32 MB of Flash and 64 MB of RAM. Modern notebooks can generally be equipped with hundreds of megabytes of RAM.

A high-end wireless LAN card such as the Proxim Harmony 802.11a [37] has a transmission range potentially as far as

250 m and data rate as high as 108 Mbps. With time synchronization provided by a Trimble Thunderbolt GPS-Disciplined Clock [44], the synchronization error can be as low as 183 ns with probability $1-10^{-10}$. If authentic keys are re-established every day, with a 20-byte minimum packet size and an 80-bit message authentication code length, the tree has depth 33, giving a minimum payload length of 350 bytes (a transmisison time of 25.9 μs) and a time interval of 24.7 μs. Assuming that the node generates each new tree while it is using its current tree, it requires 8 megabytes of storage and needs to perform fewer than 243,000 operations per second to maintain and generate trees. To authenticate a received packet, a node needs to perform only 33 hash functions. To keep up with link-speed, a node needs to verify a packet at most every 25.9 μs, thus requiring 1,273,000 hashes per second, for a total computational requirement of 1,516,000 hashes per second. This can be achieved today in hardware, either by placing two MD5 units on a single FPGA, or with an ASIC. Many laptops today are equiped with at least 1.2 GHz Pentium III CPUs, which should also be able to perform 1.5 million hash operations per second.

Current commodity wireless LAN products such as commonly used IEEE 802.11b cards [2] provide a transmission data rate of 11 Mbps and a range of 250 m. Given the same time synchronization, rekeying interval, minimum packet size, and message authentication code length, the tree has depth 30, giving a minimum payload length of 320 bytes (a transmission time of 232 μs) and a time interval of 231.5 μs. Assuming that the node generates each new tree while it is using its current tree, it requires just 2.6 megabytes of storage and needs to perform just 26,500 operations per second. To authenticate a received packet, a node needs to perform only 30 hash functions. Since any IP packet authenticated using TIK would take at least 232 μs to transmit in this example, TIK can authenticate packets at link-speed using just 13,000 hashes per second, for a total of 39,500 hash functions per second, which is well within the capability of an iPaq, with 82.2% of its CPU time to spare.

In a sensor network such as Hollar et al's weC mote [22, 45], nodes may only be able to achieve time synchronization accurate to 1 s, have a 19.6 kbps link speed, and 20 m range. In this case, the smallest packet that can be authenticated is 4900 bytes; since the weC mote does not have sufficient memory to store this packet, TIK is unusable in such a resource-scarce system. Furthermore, the level of time synchronization in this system is such that TIK could not provide a usable wormhole detection system.

B. Security Analysis

Packet leashes provide a way for a sender and a receiver to ensure that a wormhole attacker is not causing the signal to propagate farther than the specified normal transmission distance. When geographic leashes are used, nodes also detect tunneling across obstacles such as mountains that are otherwise impenetrable by radio. As with other cryptographic primitives, a malicious receiver can refuse to check the leash, just like a malicious receiver can refuse to check the authentication on a packet. This may allow an attacker to tunnel a packet to another attacker without detection; however, that second attacker cannot then retransmit the packet as if it were the original sender without then being detected.

A malicious sender can claim a false timestamp or location, causing a legitimate receiver to have mistaken beliefs about whether or not the packet was tunneled. When geographic leashes are used in conjunction with digital signatures, nodes may be able to detect a malicious node and spread that information to other nodes, as discussed in Section IV-C. However, this attack is equivalent to the malicious sender sharing its keys with the wormhole attacker, allowing the sending side of the wormhole to place appropriate timestamps or location information on any packets sent by the malicious sender that are then tunneled by the wormhole attacker.

C. Comparison Between Geographic and Temporal Leashes

Temporal leashes have the advantage of being highly efficient, especially when used with TIK, as described in Section V. Geographic leashes, on the other hand, require a more general broadcast authentication mechanism, which may result in increased computational and network overhead. Location information also may require more bits to represent, further increasing the network overhead.

Geographic leashes have the advantage that they can be used in conjunction with a radio propagation model, thus allowing them to detect tunnels through obstacles. Furthermore, geographic leashes do not require the tight time synchronization that temporal leashes do. In particular, temporal leashes cannot be used if the maximum range is less than $c\Delta$, where c is the speed of light and Δ is maximum clock synchronization error; geographic leashes can be used until the maximum range is less than $2\nu\Delta$, where ν is the maximum movement speed of any node.

To evaluate the practicality of geographic leashes, we consider a radio of range 300 m, maximum movement speed of 50 m/s, a relative positioning error of 3 m, and time synchronization error of 1 ms. Then $t_r - t_s \leq 2$ ms, since the propagation time is at most 1 ms and the time synchronization error is at most 1 ms. Then $d_{sr} \leq ||p_s - p_r|| + 100 \text{ m/s} \cdot 2 \text{ ms} + 3 \text{ m} = ||p_s - p_r|| + 3.2$ m. Since $||p_s - p_r||$ could be as much as 3 m, the effective transmission range of the network interface is reduced by at most 6.2 m.

To compare the effectiveness of geographic leashes and temporal leashes, we compare the distance derived using each approach: $d_{sr} \leq ||p_s - p_r|| + 2\nu \cdot (t_r - t_s + \Delta) + \delta$ for geographic leashes and $d_{sr} \leq c \cdot (t_r - t_s + \Delta)$ for temporal leashes. We use $\frac{d_{\max}}{c}$ to denote the maximum propagation time. Then the maximum error is bounded by $\delta + 2\nu(\frac{d_{\max}}{c} + 2\Delta) + \delta = 2\delta + 4\nu\Delta + 2\nu\frac{d_{\max}}{c}$ for geographic leashes, and by $2c\Delta$ for temporal leashes. Geographic leashes are then more effective when $\delta < c\Delta - 2\nu\Delta - \frac{\nu}{c}d_{\max}$. In general, ν is much smaller than c. Given sufficient computing power and network bandwidth, geographic leashes should be used when $\delta < c\Delta$, and temporal leashes should be used when $\delta \geq c\Delta$.

VII. Related Work

Radio Frequency (RF) watermarking is another possible approach to providing the security described in this paper. Since we are aware of no published specific details, it is difficult to assess its security. If the radio hardware is kept secret, such as through tamper-resistant modules, some level of security can be provided against compromised nodes; however, if the radio band in which communications are taking place is known, then an attacker can attempt to tunnel the entire signal from one location to another.

It may be possible to modify existing intrusion detection approaches to detect a wormhole attacker; since the packets sent by the wormhole are identical to the packets sent by legitimate nodes, such detection would more easily be achieved jointly with hardware able to specify some sort of direction of arrival information for received packets. To the best of our knowledge, no work has been published regarding the possibility of using intrusion detection systems specifically to detect wormhole attackers.

Brands and Chaum [7] propose a three-way handshake which bounds the distance between a node and a verifier by measuring the round trip time between them. Our technique is able to detect wormholes with only a single message, and reuqires corrections for clock skew between the sender and receiver.

TESLA generally chooses longer time intervals than TIK does, in order to reduce the amount of computation needed to authenticate a new key. As a result, TESLA is capable of functioning with much looser time synchronization than is required by TIK. Given a sufficient level of time synchronization, TIK provides an advantage over hop-by-hop authentication with TESLA, with respect to latency and packet overhead, but it suffers with respect to byte overhead. In particular, since TIK key disclosure always occurs in the same packet as the data protected, packets can be verified instantly; with TESLA, on the other hand, packets must wait, on average 1.5 time intervals, which is especially significant when packets are authenticated hop-by-hop, as may be required in a multi-hop ad hoc network routing protocol.

The IEEE 802.11i Task Group is designing modifications to IEEE 802.11 [20] to improve security. These modifications generally use a single shared key, or, when multiple keys are used, the keys are used between multiple clients and a single base station. Since base stations are not present in ad hoc networks, and since a single shared key does not prevent any attacks launched from a compromised node, these proposals do not sufficiently address authentication for ad hoc network routing. Furthermore, none of the current proposals within IEEE 802.11i address the wormhole attack.

Other Medium Access Control protocols also specify privacy and authenticity mechanisms. These mechanisms typically use one or more shared keys, allowing compromised nodes to forge packets. Furthermore, to the best of our knowledge, none of these mechanisms protect against wormhole attacks.

VIII. Conclusions

In this paper, we have introduced the *wormhole attack*, a powerful attack that can have serious consequences on many proposed ad hoc network routing protocols; the wormhole attack may also be exploited in other types of networks and applications, such as wireless access control systems based on physical proximity. To detect and defend against the wormhole attack, we introduced *packet leashes*, which may be either *geographic* or *temporal* leashes, to restrict the maximum transmission distance of a packet. Finally, to implement temporal leashes, we presented the design and performance analysis of a novel, efficient protocol, called TIK, which also provides instant authentication of received packets.

TIK requires just n public keys in a network with n nodes, and has relatively modest storage, per packet size, and computation overheads. In particular, a node needs to perform only between 3 and 6 hash function evaluations per time interval to maintain up-to-date key information for itself, and roughly 30 hash functions for each received packet. With commodity hardware such as 11 Mbps wireless links, TIK has computational and memory requirements that are easily satisfiable today; 2.6 megabytes for hash tree storage represents, for example, less than 3% of the standard memory on an Compaq iPaq 3870 with no external memory cards, and since the StrongARM CPU on the iPaq is capable of performing 222,000 symmetric cryptographic operations per second, TIK imposes no more than an 18% load on CPU time, even when flooded with packets at the maximum speed of the wireless network, and normally uses less CPU load than that in normal operation.

When used in conjunction with precise timestamps and tight clock synchronization, TIK can prevent wormhole attacks that cause the signal to travel a distance longer than the nominal range of the radio, or any other range that might be specified. Sufficiently tight clock synchronization can be achieved in a wireless LAN using commercial GPS receivers [44], and wireless MAN technology could be sufficiently time-synchronized using either GPS or LORAN-C [30] radio signals.

A MAC layer protocol using TIK efficiently protects against replay, spoofing, and wormhole attacks, and ensures strong freshness. TIK is implementable with current technologies, and does not require significant additional processing overhead at the MAC layer, since the authentication of each packet can be performed on the host CPU.

Our geographic leashes are less efficient than temporal leashes, since they require broadcast authentication, but they can be used in networks where precise time synchronization is not easily achievable. The dominant factor in the usability of geographic leashes is the ability to accurately measure location; because node movement is very slow relative to the speed of light, the effects of reduced time synchronization accuracy are slight.

References

[1] Norman Abramson. The ALOHA System—Another Alternative for Computer Communications. In *Proceedings of the Fall 1970 AFIPS*

Computer Conference, pages 281–285, November 1970.

[2] Agere Systems Inc. Specification sheet for ORiNOCO World PC Card. Allentown, PA. Available at `ftp://ftp.orinocowireless.com/pub/docs/ORINOCO/BROCHURES/US/World%20PC%20Card%20US.pdf`.

[3] ARC International. ARC releases BlueForm, a comprehensive solution for Bluetooth systems on a chip. Press Release 6-04-01, Elstree, United Kingdom. Available at `http://www.arccores.com/newsevents/PR/6-04-01-2.htm`, June 4 2001.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology – CRYPTO '96*, edited by Neal Koblitz, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, Berlin Germany, 1996.

[5] Bhargav Bellur and Richard G. Ogier. A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, pages 178–186, March 1999.

[6] Matt Bishop. A Security Analysis of the NTP Protocol Version 2. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, November 1990.

[7] Stefan Brands and David Chaum. Distance-Bounding Protocols. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology (CRYPTO 1994)*, volume 839 of *Lecture Notes in Computer Science*, pages 344–359. Springer-Verlag, August 1994.

[8] Michael Brown, Donny Cheung, Darrel Hankerson, Julio Lopez Hernandez, Michael Kirkup, and Alfred Menezes. PGP in Constrained Wireless Devices. In *Proceedings of the 9th USENIX Security Symposium*, pages 247–262, August 2000.

[9] Ran Canetti, Juan Garay, Gene Itkis, Daniele Micciancio, Moni Naor, and Benny Pinkas. Multicast Security: A Taxonomy and Some Efficient Constructions. In *Proceedings of the Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, pages 708–716, March 1999.

[10] Tom Clark. Tom Clark's Totally Accurate Clock FTP Site. Greenbelt, Maryland. Available at `ftp://aleph.gsfc.nasa.gov/GPS/totally.accurate.clock/`.

[11] Mark Corner and Brian Noble. Zero-Interaction Authentication. In *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking (MobiCom 2002)*, pages 1–11, September 2002.

[12] Defense Advanced Research Projects Agency. Frequently Asked Questions v4 for BAA 01-01, FCS Communications Technology. Washington, DC. Available at `http://www.darpa.mil/ato/solicit/baa01_01faqv4.doc`, October 2000.

[13] Y. Desmedt. Major Security Problems with the "Unforgeable" (Feige-)Fiat-Shamir Proofs of Identity and How to Overcome Them. In *Proceedings of the 6th worldwide computer congress on computer and communications security and protection (SecuriCom 88)*, pages 147–159, March 1998.

[14] Eran Gabber and Avishai Wool. How to Prove Where You Are. In *Proceedings of the 5th ACM Conference on Computer and communications Security*, pages 142–149, November 1998.

[15] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to Construct Random Functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[16] Shafi Goldwasser and Mihir Bellare. Lecture Notes on Cryptography. Summer Course "Cryptography and Computer Security" at MIT, 1996–1999, August 1999.

[17] Neil M. Haller. The S/KEY One-time Password System. In *Proceedings of the 1994 Symposium on Network and Distributed Systems Security*, edited by Dan Nesset and Robj Shirey, pages 151–157, February 1994.

[18] Helion Technology Ltd. High Performance Solutions in Silicon — MD5 Core. Cambridge, England. Available at `http://www.heliontech.com/core5.htm`.

[19] Jean-Pierre Hubaux, Levente Buttyán, and Srdjan Čapkun. The Quest for Security in Mobile Ad Hoc Networks. In *Proceedings of the 2001 ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2001)*, pages 146–155, October 2001.

[20] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[21] David B. Johnson, David A. Maltz, and Josh Broch. The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In *Ad Hoc Networking*, edited by Charles E. Perkins, chapter 5, pages 139–172. Addison-Wesley, 2001.

[22] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next Century Challenges: Mobile Networking for Smart Dust. In *Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 271–278, August 1999.

[23] Dean Kawaguchi and Sarosh Vesuna. Symbol Technologies, Inc. Automates System-To-Gates Design Flow For Wireless LAN ASIC with COSSAP and Behavioral Compiler. Mountain View, California. Available at `http://www.synopsys.com/news/pubs/bctb/sep98/frame_art1.html`, September 1998.

[24] Tim Kindberg, Kan Zhang, and Narendar Shankar. Context Authentication Using Constrained Channels. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002)*, pages 14–21, June 2002.

[25] Jiejun Konh, Petros Zerfos, Haiyun Luo, Songwu Lu, and Lixia Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad-Hoc Networks. In *Proceedings of the Ninth International Conference on Network Protocols (ICNP 2001)*, pages 251–260, November 2001.

[26] Leslie Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, November 1981.

[27] Arjen K. Lenstra and Eric R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology: The Journal of the International Association for Cryptologic Research*, 14(4):255–293, September 2001. Available at `http://www.cryptosavvy.com/`.

[28] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press Series on Discrete Mathematics and its Applications. CRC Press, 1997.

[29] Ralph Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–136, April 1980.

[30] David L. Mills. A Computer-Controlled LORAN-C Receiver for Precision Timekeeping. Technical Report 92-3-1, Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, March 1992.

[31] David L. Mills. A Precision Radio Clock for WWV Transmissions. Technical Report 97-8-1, Department of Electrical and Computer Engineering, University of Delaware, Newark, DE, August 1997.

[32] Charles E. Perkins and Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Proceedings of the SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, August 1994.

[33] Charles E. Perkins and Elizabeth M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99)*, pages 90–100, February 1999.

[34] Adrian Perrig, Ran Canetti, Doug Tygar, and Dawn Song. Efficient Authentication and Signature of Multicast Streams over Lossy Channels. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 56–73, May 2000.

[35] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security Protocols for Sensor Networks. In *Proceedings of the Seventh Annual Annual International Conference on Mobile Computing and Networks (MobiCom 2001)*, pages 189–199, July 2001.

[36] Raymond L. Pickholtz, Donald L. Schilling, and Laurence B. Milstein. Theory of Spread Spectrum Communications — A Tutorial. *IEEE Transactions on Communications*, 30(5):855–884, May 1982.

[37] Proxim, Inc. Data sheet for Proxim Harmony 802.11a CardBus Card. Sunnyvale, CA. Available at `http://www.proxim.com/products/all/harmony/docs/ds/harmony_11a_cardbus.pdf`.

[38] Amir Qayyum, Laurent Viennot, and Anis Laouiti. Multipoint Relaying: An Efficient Technique for flooding in Mobile Wireless Networks. Technical Report Research Report RR-3898, Project HIPERCOM, INRIA, February 2000.

[39] Ron L. Rivest, Adi Shamir, and Leonard M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[40] Claus P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[41] Karen E. Sirois and Stephen T. Kent. Securing the Nimrod Routing Architecture. In *Proceedings of the 1997 Symposium on Network and Distributed Systems Security (NDSS'97)*, pages 74–84, February 1997.

[42] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *Security Protocols, 7th International Workshop*, edited by B. Christianson, B. Crispo, and M. Roe. Springer-Verlag, Berlin Germany, 1999.

[43] Joseph D. Touch. Performance Analysis of MD5. In *Proceedings of the ACM SIGCOMM '95 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 77–86, August 1995.

[44] Trimble Navigation Limited. Data Sheet and Specifications for Trimble Thunderbolt GPS Disciplined Clock. Sunnyvale, California. Available at `http://www.trimble.com/thunderbolt.html`.

[45] Alec Woo. CS294-8 Deeply Networked Systems Mote Documentation and Development Information. Berkeley, CA. Available at `http://www.cs.berkeley.edu/~awoo/smartdust/`.

[46] Lidong Zhou and Zygmunt J. Haas. Securing Ad Hoc Networks. *IEEE Network Magazine*, 13(6):24–30, November/December 1999.