Wireless Netw (2008) 14:895–913
DOI 10.1007/s11276-007-0031-z

Mobile multi-layered IPsec

Heesook Choi · Hui Song · Guohong Cao · Thomas F. La Porta

Published online: 8 March 2007

Abstract To achieve high throughput in wireless networks, smart forwarding and processing of packets in access routers is critical for overcoming the effects of the wireless links. However, these services cannot be provided if data sessions are protected using end-to-end encryption as with IPsec, because the information needed by these algorithms resides inside the portion of the packet that is encrypted, and can therefore not be used by the access routers. A previously proposed protocol, called Multi-layered IPsec (ML-IPsec) modifies IPsec in a way so that certain portions of the datagram may be exposed to intermediate network elements, enabling these elements to provide performance enhancements. In this paper we extend ML-IPsec to deal with mobility and make it suitable for wireless networks. We define and implement an efficient key distribution protocol to enable fast ML-IPsec session initialization, and two mobility protocols that are compatible with Mobile IP and maintain ML-IPsec sessions. Our measurements show that, depending on the mobility protocol chosen, integrated Mobile IP/ML-IPsec handoffs result in a pause of 53–100 milliseconds, of which only 28–75 milliseconds may be attributed to ML-IPsec. Further, we provide detailed discussion and performance measurements of our MML-IPsec implementation. We find the resulting protocol, when coupled with SNOOP, greatly increases throughput over scenarios using standard TCP over IPsec (165% on average). By profiling the MML-IPsec implementation, we determine the bottleneck to be sending packets over the wireless link. In addition, we propose and implement an extension to MML-IPsec, called dynamic MML-IPsec, in which a flow may switch between plaintext, IPsec and MML-IPsec. Using dynamic MML-IPsec, we can balance the tradeoff between performance and security.

Keywords IPsec · Mobile IP · Wireless TCP · Multi-Layered IPsec · SNOOP · Security

H. Choi (✉) · H. Song · G. Cao · T. F. La Porta
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802
e-mail: hchoi@cse.psu.edu

H. Song
e-mail: hsong@cse.psu.edu

G. Cao
e-mail: gcao@cse.psu.edu

T. F. La Porta
e-mail: tlp@cse.psu.edu

1 Introduction

Data confidentiality and integrity are two critical issues for wireless, mobile networks. These issues are of growing importance as wireless service providers attempt to increase wireless data traffic by providing mobile VPN services. The most widely accepted method for ensuring data confidentiality and integrity is to pass encrypted data end-to-end using a mechanism such as IPsec [14].

For wireless networks, smart forwarding and processing of packets are also critical for overcoming the effects of the wireless links, especially highly variable delay and error rates. Several studies have shown that techniques such as smart scheduling with respect to the type of data being sent and regulation of TCP acknowledgment information, can greatly improve end-to-end performance in a wireless network [2, 6]. However, these services cannot be provided if end-to-end encryption is used, such as in IPsec, because the information needed by these algorithms resides inside the portion of the packet that is encrypted, and can therefore not be used by mobile routers.

Previous work, called Multi-layered IPsec (ML-IPsec) [22] applies a modified version of IPsec so that certain portions of the user information may be exposed to particular intermediate network elements in a route. In this way, portions of a datagram may be encrypted end-to-end, while portions may be read and operated upon by network elements providing performance enhancements. However, the ML-IPsec as defined in [22] is designed for static environments and does not examine mobility.

In this paper we extend ML-IPsec to deal with mobility and make it suitable for wireless networks. We call our resulting protocol *Mobile ML-IPsec (MML-IPsec)*. We make the following four contributions: (1) we define and present performance measurements of an efficient key distribution protocol to enable fast MML-IPsec session initialization; (2) we define and present performance measurements of two mobility protocols that maintain MML-IPsec sessions; (3) we provide detailed discussion and performance measurements of our MML-IPsec implementation to quantify its performance impact compared to non-secure communication and communication using IPsec; and (4) we provide a detailed discussion and performance measurements of our implementation of SNOOP, and SNOOP executing over MML-IPsec to quantify the benefits of using MML-IPsec to enable performance enhancing algorithms in a wireless environment.

Our measurements in a wireless environment show that, depending on the mobility protocol chosen, integrated Mobile IP/ML-IPsec handoffs result in a pause of 53–100 milliseconds, of which only 28–75 milliseconds may be attributed to MML-IPsec. We found MML-IPsec only marginally reduced throughput compared to scenarios in which no encryption is used (9%), or those in which IPsec is used (4%), and when coupled with SNOOP, greatly increased throughput over scenarios using standard TCP end-to-end (50% on average), or using TCP over IPsec (165% on average). Our conclusion, based on these results, is that MML-IPsec is a worthwhile protocol to pursue because it enables large performance improvements while providing end-to-end secure transfer of user data. By profiling the source code, we determine the bottleneck in MML-IPsec to be sending packets over the wireless link.

For many applications, security may be provided without encrypting all the packets. For example, in video streaming service, certain frames contain more information than others. In this case, applying MML-IPsec to only the high quality frames will provide more efficient transport without sacrificing security. Using MPEG-2 [1] video streaming as an example, we can choose to only encrypt the I-Frames and transmit the remaining frames, i.e., P-Frames and B-Frames, without encryption. As another example, if a MH determines that it has a high signal strength, and therefore likely has a reliable wireless link, it may not require a performance enhancing algorithm in the access network. In this case, it may choose to run IPsec end-to-end, and disable MML-IPsec.

Based on this observation, we propose and implement an extension to MML-IPsec, called dynamic MML-IPsec, in which a flow may switch between plaintext, IPsec and MML-IPsec. Using dynamic MML-IPsec, we can balance the tradeoff between performance and security.

The rest of this paper is organized as follows. In Section 2 we present an overview of related and previous work, including a description of ML-IPsec. In Section 3 we discuss our design of MML-IPsec, our model for integrating Mobile IP with IPsec and MML-IPsec, the software platform on which we base our implementation, and the test bed used to evaluate the performance of the protocols. In Section 4 we present our key distribution protocol, our two mobility protocols, and characterize their performance. In Section 5 we present our implementations of MML-IPsec and SNOOP and their performance. The source code of MML-IPsec is profiled to identify the bottleneck of the infrastructure. We also present the design and evaluation results of dynamic MML-IPsec. Section 6 concludes the paper.

2 Background

Several studies have shown that the performance of classic data communication protocols can be quite poor when used over wireless links. In particular, the performance of TCP, the reliable Internet transport protocol, can be degraded by the loss and delay characteristics of a wireless link. Consequently, there have been several efforts aimed at improving the performance of TCP on wireless links. Two of the more promising works do not require any modifications to TCP, but instead perform smart processing (forwarding, filtering, scheduling) on TCP/IP packets based on information gleaned from observing packet flows. In [2], the authors show that by snooping on TCP/IP packets at the wireless edge, determining when packet loss has occurred by detecting duplicate acknowledgments, and performing fast local retransmissions, TCP performance can be greatly improved.

More recently in [6], it was uncovered that TCP performance is adversely affected by the highly variable delay experienced on the 3G wireless links. The effect is that TCP acknowledgments sent on the uplink tend to be compressed causing them to arrive at transmitters back-to-back. The compression of acknowledgments results in transmitters sending bursts of data. These bursts of data can overflow buffers at the wireless edge resulting in high packet loss. The solution proposed is to regulate the flow of acknowledgments to ensure that buffers do not overflow.

In both of these proposals, the node at the wireless edge must observe information in the TCP header to execute their algorithms. The need to have nodes inside the network

examine packet payloads to perform smart packet processing or perform packet classification is in direct conflict with the current Internet model of security implemented by the IPsec protocol suite [11–14]. IPsec supports a variety of operational modes including packet authentication, packet encryption, or both. In the most secure mode, tunnel mode, the entire IP packet is encrypted and encapsulated with a new IP packet header. Therefore, intermediate nodes in the network do not have access to the original IP header information, nor the information contained in any of the transport layer or application layer protocols. This precludes the network from performing smart packet processing and packet classification to improve end-to-end performance.

There are several possible solutions to this problem. Protocols such as TLS [7] and SSL [8] provide security above the transport layer. With their use, user payloads are encrypted, but the TCP and IP headers are in the clear. Therefore, intermediate nodes may access required information to perform many of the performance enhancements discussed above. The main drawback is that the TCP and IP header information is in the clear throughout the entire network allowing for possible eavesdropping to determine communication patterns and traffic characterizations. Also, these protocols do not enable application layer packet classification for protocols such as RTP.

A more flexible solution is defined in the ML-IPsec protocol [22]. This protocol allows a user to define zones within an IP packet. Each zone is encrypted and authenticated with its own security association (SA). Each zone may be accessed (decrypted) by different network elements. This requires SAs to be established between a client and several nodes in a network, each of which can decrypt a certain portion of the IP packet while being unable to view the entire packet.

For example, consider the wireless network of Fig. 1. In this example, the corporate firewall acts as a Mobile IP Home Agent (HA). Foreign Agent (FA) 1 requires access to TCP/IP header information to perform smart packet processing. Using IPsec, secure communication would entail running an IPsec tunnel between the HA and Mobile Host (MH), in which case FA1 would not have access to the TCP/IP header information. Using ML-IPsec, this header information would be included in Zone A which is accessible to FA1. However, the user payload would be placed in Zone B which is not accessible to FA1. In this way, the user information is protected end-to-end and the TCP/IP header information is protected from all nodes except FA1 which may perform smart packet processing.

While ML-IPsec is a promising start, it has limitations and several unknowns. First, it requires that SAs (secret keys, algorithms, parameters, etc.) be established between multiple elements for a single data session. This requires an efficient key distribution algorithm which has yet to be defined. Second, mobility is not supported. The mobility requires that new SAs be established as a mobile host moves during a data session. For example, in Fig. 1, if the mobile host moves from base station 1 (BS1) to BS2, SA1 must move from FA1 to FA2. These modifications must be performed quickly so that sessions are not disrupted during a handoff. This is more complex than mobility in basic Mobile IP [18] because in this MML-IPsec system, multiple SAs exist that operate on the user data.

Third, there is no data available on the performance tradeoffs between the overhead of supporting multiple zones versus the benefits of packet classification or smart packet processing. Specifically, mobile access routers, e.g., FAs, will have hundreds of flows passing through them, so the overhead of the key distribution and initialization, handoffs, and

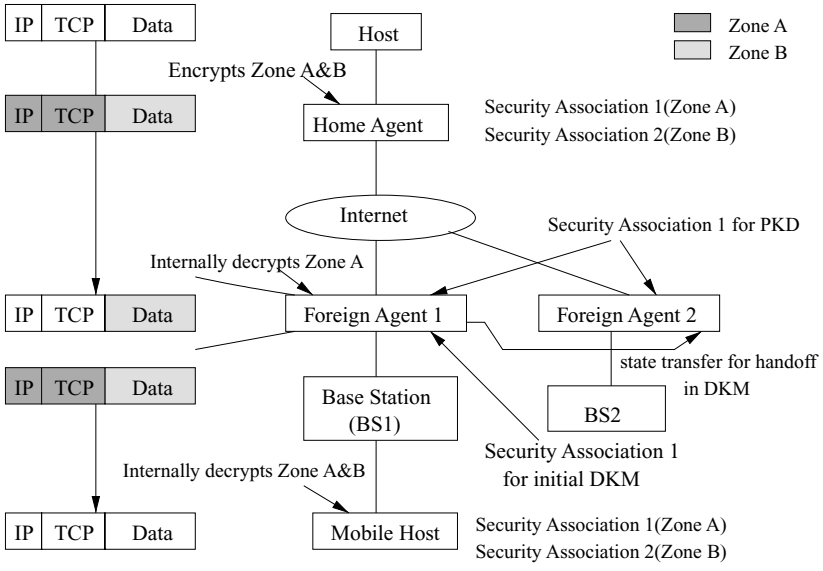


Fig. 1 Mobile multi-layered ipsec example

per packet processing, must be kept low to achieve high performance.

IKE [9,11] supports key distribution and mutual authentication between two nodes but requires extensions to support the multiple SAs used in ML-IPsec and is not suitable for mobility. Support for key distribution in mobile networks is the focus of [20]. An efficient method of key distribution and authentication between a home network, security server in a foreign network, and a mobile host is presented. However, this work does not address the distribution of multiple keys required for a ML-IPsec, does not account for mobility, and does not provide any implementation or performance insights.

3 Mobile multi-layered IPsec - services and software platform

In the following subsections we present background on ML-IPsec and how to make it suitable for wireless networks; then we propose solutions to integrate IPsec and mobile IP; the last two subsections describe the software platform on which we base our implementation, and our test bed.

3.1 Protocol services

The original ML-IPsec [22] is defined to allow network layer packets to be segmented into zones, each of which is protected, i.e., encrypted, authenticated, or both, independently. Corresponding hosts have access to all zones and can therefore authenticate and decrypt the entire packets. Selected intermediate nodes are given access to one or more selected zones, and may therefore decrypt and authenticate only these portions of the packet. Before communication can commence, a set of SAs, called a composite security association (CSA), must be established, one for each zone in each node for which access to the zone is permitted. As defined in [22], the number of zones and the number of intermediate nodes with access to at least one zone are not limited. Also, there is no limit on the number of zones in a packet, and zones are not required to cover contiguous bits in a packet.

We have somewhat restricted the definition of ML-IPsec to meet the needs of the known methods of enhancing wireless system performance, while keeping the processing complexity low. First, we limit the number of allowed intermediate nodes to a single node, specifically the Mobile IP FA. We choose this node because the vast majority of wireless enhancements operate on a node close to, or supporting a wireless link, and do not require changes to any other portion of the network. Second, we limit the number of zones to two, one for the packet header and one for the payload. The rationale is that most algorithms require access to TCP/IP header information, and not packet payload. This restriction can be easily relaxed.

Finally, we define zones as contiguous portions of the packet to ease processing. Certainly we can let a zone consist of a collection of sub-zones (i.e., continuous blocks) distributed over the whole datagram. We define each zone as one continuous block for two reasons. First, in our settings, it is nature to have only two continuous blocks: one is the IP head and the other is the payload. Second, it eliminates the need for maintaining sub-zone information for each zone. With reduced processing complexity, our protocol is more practical and efficient at core devices such as routers.

In addition to these changes, we have also defined a key distribution protocol for MML-IPsec and two mobility protocols (Section 4).

3.2 Integrating mobile IP, IPsec, and MML-IPsec

We assume that if the basic IPsec is used, the IPsec tunnel extends between the HA and the MH. If MML-IPsec is used, the MML-IPsec tunnel includes the HA and the MH, while the FA has access to the header part of the TCP/IP packet. In addition, we assume reverse tunneling [17] is used for data transmitted from the MH so that packets in both directions are consistently encrypted.

Several research projects [3,4] have proposed solutions to integrate Mobile IP and IPsec. SecMIP [4] is based on the configuration in which the MH uses the DNS/DHCP service to get the COA (Care-of-address). Without using the FA, the MH gets a new collocated COA using DHCP. SecMIP uses the IPsec tunnel to protect the Mobile IP tunnel. While this is a simple scheme to provide security in Mobile IP, the handoff delay is high because the MH must re-establish the IPsec tunnel on every handoff.

Secure Mobile networking (SMN) project [3], by Portland State University, establishes an IPsec tunnel between the HA and the MH. When the MH moves to the foreign network, it has to register its COA to the HA. However, the established IPsec tunnel prevents the Mobile IP registration message from reaching the FA because every packet, including the Mobile IP registration packet, is encrypted with IPsec. In order to solve this problem, the client software is changed so that the Mobile IP packet is not encrypted. The re-establishment of IPsec tunnel on every handoff will degrade the end-to-end performance because it incurs high handoff latencies.

We propose a different integration model for Mobile IP and IPsec which is then largely re-used for integrating Mobile IP and MML-IPsec. This model is similar to SMN, but does not require changes to the client software, and does not require IPsec tunnels to be re-established after each handoff. To enable Mobile IP registration messages to be received by the FA when an IPsec tunnel is in place between the MH and

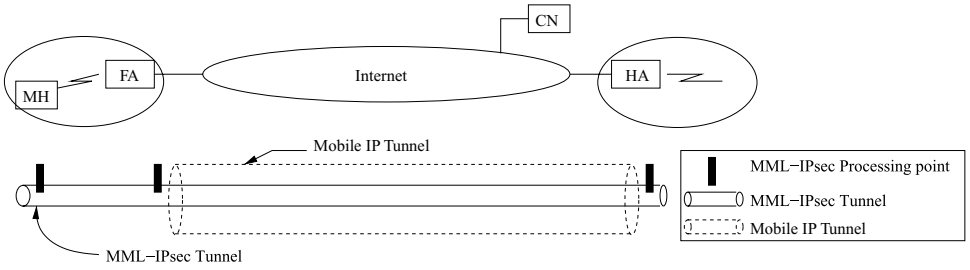


Fig. 2 Integrating Mobile IP and MML-IPsec/IPsec

HA, we add an additional routing entry in the MH and leverage the fact that route selection chooses the route having the longest prefix match among multiple matched entries. When an agent advertisement is received by a MH, it adds a route entry for the FA. The new route entry specifies the FA address as the gateway for all packets destined to the FA. After adding this route, the Mobile IP registration message addressed to the FA will match the new route, and therefore be sent directly to the FA, instead of using the old entry through which packets are encrypted.

To eliminate the need to re-establish IPsec tunnels after each handoff, we leverage the fact that when using Mobile IP, while the COA of the MH changes after each IP layer handoff, the IP addresses of the MH and HA remain constant. Therefore, the IPsec tunnel may remain intact. In our model, during Mobile IP registration, the HA simply updates the routing entry for IPsec packets destined to the MH to be forwarded through the new Mobile IP tunnel. This does not require any message beyond the standard Mobile IP registration.

Figure 2 depicts this integration model which can be applied to both IPsec and MML-IPsec. The MML-IPsec tunnel is established between the HA, FA and MH, within the Mobile IP tunnel. An advantage of our model is that it does not restrict the scheme of obtaining a COA as that in SecMIP, and provides a seamless integration between Mobile IP and ML-IPsec. Table 1 shows the overall handoff flow between the HA, FA, and MH. This illustrates that our proposed model seamlessly integrates Mobile IP with ML-IPsec.

Next, let us consider the the data traffic flow from the MH to the correspondent node (CN). The FA receives the encrypted traffic through the MML-IPsec tunnel or IPsec tunnel. The outer IP header has the source address as the MH IP address, and the destination address as the HA IP address. The FA routes this data traffic into the Mobile IP tunnel based on the reverse tunnel of the Mobile IP. Next, let's consider the traffic from the CN to the MH. This traffic is intercepted by the HA. The HA encrypts the traffic and encapsulates it within the Mobile IP tunnel, which has the outer header source address as the HA IP address, and the destination address as the FA IP address. If IPsec is used, when receiving the encapsulated traffic, the FA decapsulates the Mobile IP outer header and transmits the encrypted traffic to the MH. If MML-IPsec is used, the intermediate node decrypts the first zone (packet header), performs some processing, re-encrypts this zone, and forwards the data.

Table 1 Overall handoff flow of Integrated Mobile IP and MML-IPsec

MH
– Receive an Agent Advertisement (from a new FA)
  1. change routing table to set the new FA as a default router
  2. send Mobile IP Registration message to the new FA
– Receive a Mobile IP Registration Reply message
  1. Finished

FA
– Receive a Mobile IP Registration message
  1. *initiate key distribution**
  2. send Mobile IP Registration message to the HA
– Receive a Mobile IP Registration Reply message
  1. add a new Mobile IP tunnel into routing table for the MH
  2. activate MML-IPsec for the MH
  3. forward the Mobile IP Registration Reply to the MH

HA
– Receive a Mobile IP Registration message
  1. reset the previous configuration (MML-IPsec and Mobile IP tunnel)
  2. change local configurations to map MML-IPsec into a new Mobile IP tunnel
  3. send a Mobile IP Registration reply message to the FA

* Optional based on key distribution protocol.

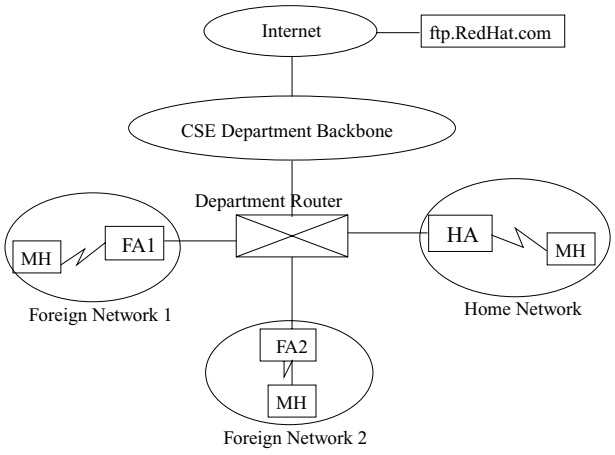


Fig. 3 Mobile Multi-Layered IPsec test bed

3.3 Software platform

Our implementation is based on Linux FreeS/WAN version 1.99, an open source IPsec implementation available free on the web (under GNU license term), on Linux kernel version 2.4.21. The FreeS/WAN system has two major components: the Pluto Daemon and the Kernel IPsec Support (KLIPS). The Pluto Daemon implements the IKE protocol [9,11]. We modified portions of the Pluto Daemon to implement our key distribution protocols and to support the interface between the key distribution protocols and the MML-IPsec transport module. We made major modifications to KLIPS to integrate IPsec and SNOOP.

3.4 Test bed

To evaluate the performance of our protocols, we set up a test bed as shown in Fig. 3. The base stations are DELL Pentium desktops (P4 2.4 GHz), and the MHs are DELL Pentium laptops (Mobile P4 2.4 GHz). These are equipped Orinoco Prism 802.11b wireless cards configured in ad hoc mode so that the desktop machines act as base stations. All these machines are running RedHat 9.0 with Linux kernel version 2.4.21.

4 Key distribution and mobility management

In this section we present efficient automatic key management protocols for MML-IPsec integrated with Mobile IP. We include procedures for session initialization and mobility management. Our goal is to enable fast handoffs while maintaining MML-IPsec sessions. In our model, the nodes involved in the MML-IPsec session are the MH, HA, and FA. The MH and HA have access to both zones in the MML-IPsec packets; the FA serves as the intermediate node and has access to the first zone of the packet containing the TCP/IP header. The key management protocols are responsible for establishing the required SAs between these nodes, and for enabling mobility.

The key management protocols have two phases. In the first phase, a MML-IPsec session is established using the initialization procedure. This includes determining if a FA will be involved in the secure session, and hence requires the use of MML-IPsec. The second phase of the protocols supports mobility. We propose two protocols for this purpose. The first, called *Proactive Key Distribution (PKD)*, pre-establishes SAs with not only the current FA, but its neighbors as well. Therefore, when a MH moves to a new FA, the SA already exists. The second, called *Dynamic Key Migration (DKM)*, requires SAs to migrate between FAs as a user moves.

In the following subsections we first discuss MLIKE Initialization, PKD, and DKM. Our description is based on IKE version 2.0 (IKEv2) [11]. We implemented key distribution protocols in both IKE version 1.0 (IKEv1) and IKEv2. These implementations are separate packages since IKEv1 and IKEv2 are not compatible. We present our implementation and performance results.

4.1 Initialization

When a MH leaves its home network, it executes Mobile IP registration procedures. In addition, Initialization is invoked. Figure 4 shows the Initialization flows between the HA, FA and MH, with the Mobile IP registration.

The Initialization phase begins after the HA has sent the Mobile IP registration reply to the FA. First, the HA establishes an IKE SA [9, 11, 15] with the FA and MH so that session key information may be exchanged securely. Note that the establishment of the IKE SA between the FA and HA occurs in parallel with sending the Mobile IP registration reply to the MH.

The second step of the Initialization is to establish the CSAs in the MH, HA and FA. The CSA has two elements, a zone map and a zone list. The zone map states the start and stop positions of the zones in the IP datagram. The zone list contains the SAs for all the zones. The HA, FA, and MH create and keep an instance of the CSA. The source and destination (HA and MH) store a complete list of SAs. The FA has a non-null SA in the zone list for the zone that it supports, and a null SA for the zone that it does not support.

The HA and MH setup a MML-IPsec CSA using the MLIKE AUTH SA Exchanges (flow (7) in Fig. 4). During these exchanges, the MH and HA exchange the complete zone map and SAs to compose a CSA. We define a new payload type called “Zone Map” which delivers the zone map information. In addition, we modify the key exchange

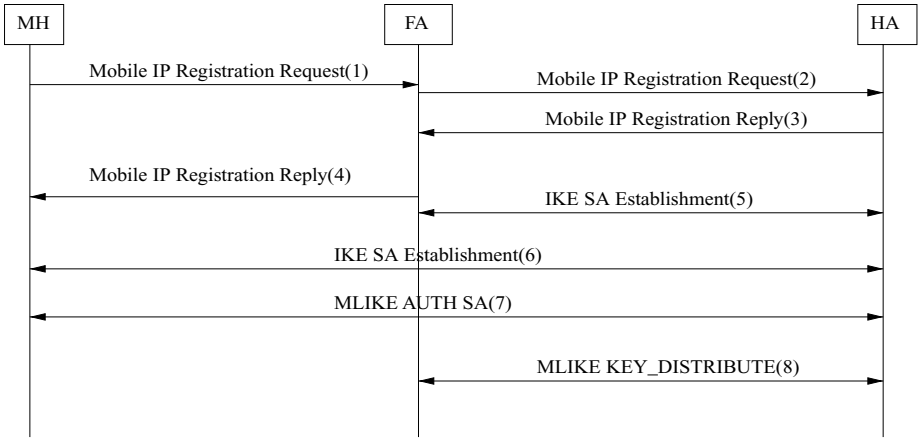


Fig. 4 Mobile IP registration and MML-IPsec key initialization

protocol to allow for multiple SAs to be included. The secret key values for all zones are decided in the MML-IPsec AUTH SA Exchanges.

Once the CSA is established between the MH and HA, the HA delivers the CSA to the FA, using the IKE SA with the FA. For the zone to which the FA has access, i.e., the zone covering the TCP/IP header, the HA sends the corresponding non-null SAs to the FA (flow (8) in Fig. 4) with the corresponding symmetric key values. A new payload, called "SECRET", delivers the symmetric key values.

Upon completion of the Initialization procedure, data transmission using MML-IPsec may take place.

4.2 Proactive key distribution (PKD)

The goal of PKD is to enable a fast handoff by pre-distributing keys in FAs that are neighbors of the current FA, so that very little overhead is incurred during the real-time handoff. For example, in Fig. 1, SA1 is placed in both FA1 and FA2 when the session is established. The distribution of the CSA information to these neighboring FAs is performed after the Initialization exchange is complete, so the Initialization overhead will not be increased by PKD. The disadvantage of this approach is that the active key information must be stored in more nodes than are actively being used, thus creating a higher chance of the session key being compromised.

Figure 5 shows the PKD protocol flow. The FA, when finished Initialization, notifies the HA of its neighbor FAs (flow (1) in Fig. 5). The HA establishes an IKE SA to each neighboring FA to transmit ML-IPsec CSA securely (flows (2) and (4) in Fig. 5). The HA distributes the MML-IPsec CSA information established via IKE SA to the neighbor FAs (flows (3) and (5) in Fig. 5).

PKD can be performed in two ways: (a) point-to-point sequential key distribution; (b) multicast key distribution. In this paper, we use the former to simplify the implementation.

When the MH moves to a new FA, the handoff latency is low because the MML-IPsec CSA information is already loaded in the new FA. When the new FA receives a Mobile IP registration reply from the HA, the FA internally activates the MML-IPsec CSA. The HA only changes the internal binding of the MML-IPsec tunnel to the new Mobile IP tunnel with the neighbor FA.

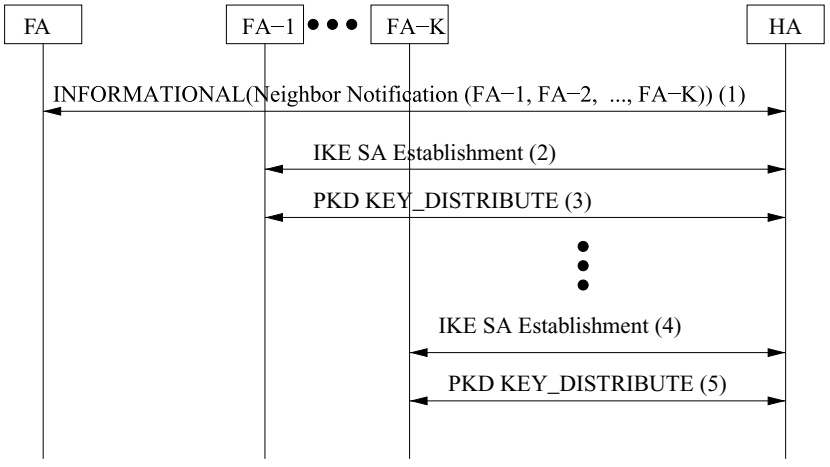


Fig. 5 Proactive key distribution protocol flow

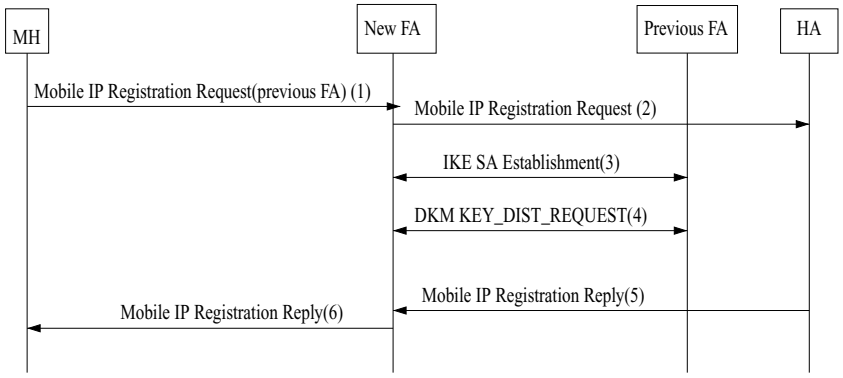


Fig. 6 Directed key migration protocol flow

4.3 Directed key migration (DKM)

Unlike PKD, in DKM the CSA information is only stored in the FA that is actively serving the MH. Therefore, when a MH changes FAs, the CSA must be migrated from the old FA to the new FA in a secure manner. For example, in Fig. 1, SA1 must be moved from FA1 to FA2. This method only requires that the CSA be stored in a single intermediate node, but incurs a higher latency than PKD because more signaling is required during the handoff. After a handoff, rekeying may take place as described in Section 4.4, so that only one FA has the current CSA.

Figure 6 shows the DKM protocol flow. When a MH moves to a new FA, it detects the movement using standard Mobile IP techniques. We modify the Mobile IP Registration [18] by adding an extension to include the previous FA address, as is done in Mobile IP with Route Optimization [19].

First, the MH transmits a Mobile IP registration message with the previous FA information to the new FA (flow (1) in Fig. 6). The new FA uses the previous FA information to decide where to retrieve the MML-IPsec CSA information. The new FA initiates the DKM protocol, and relays the Mobile IP registration message to the HA simultaneously.

In DKM, if there is no IKE SA established between the previous FA and the new FA, the new FA establishes an IKE SA with the previous FA so the key information is transferred securely. Once this IKE SA is established, the new FA transmits the MML-IPsec CSA information request to the previous FA (flow (4) in Fig. 6). The previous FA authenticates the new FA and sends the response to the new FA. The response message includes the MML-IPsec CSA including the secret key values.

Note that the DKM protocol is processed in parallel with the Mobile IP registration between the new FA, HA and MH.

4.4 Rekeying and revocation

There are several reasons why rekeying or key revocation may take place when using MML-IPsec. For example, if a CSA lifetime expires, or a CSA is determined to be insecure, it may be revoked, or if secure communication is still desired, rekeying may take place during which a new CSA is established. Further, key revocation may take place when a Mobile IP tunnel is deleted, for example when a MH returns to its home network or powers off. Finally, rekeying may take place after a handoff, if the number of FAs that share the CSA exceeds a threshold value. Note that in DKM the threshold value is one since only the current FA has the CSA.

IKEv2 [11] defines rekeying procedures so that the peers can initiate the establishment of a new IPsec SA, while the old IPsec SA is active. This minimizes the interruption of data transmission. In the same way, a new MML-IPsec CSA is established while the old one is being used. Once the new CSA is established, the MH, FA and HA change to use the new CSA for data transfer, and the old CSA is deleted.

Figure 7 shows the CSA rekeying which is triggered when MML-IPsec CSA lifetime expires. Either a HA or a MH initiates the rekeying (flow (1) in Fig. 7). The HA distributes the new CSA to the current FA in which the MH resides (flow (2) in Fig. 7).

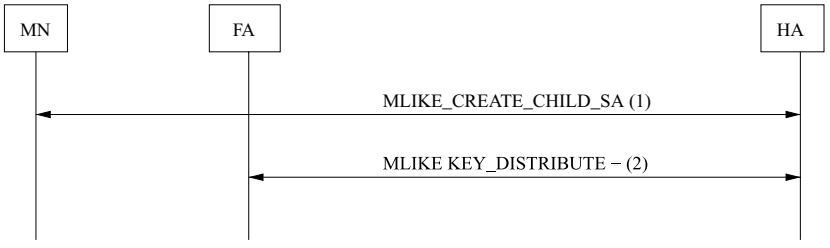


Fig. 7 MML-IPsec rekeying by lifetime expiration

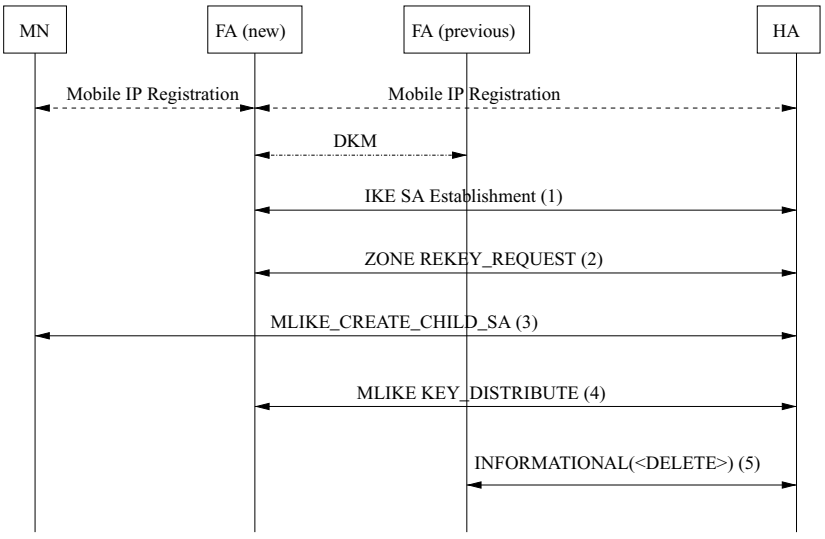


Fig. 8 MML-IPsec rekeying/revocation in DKM

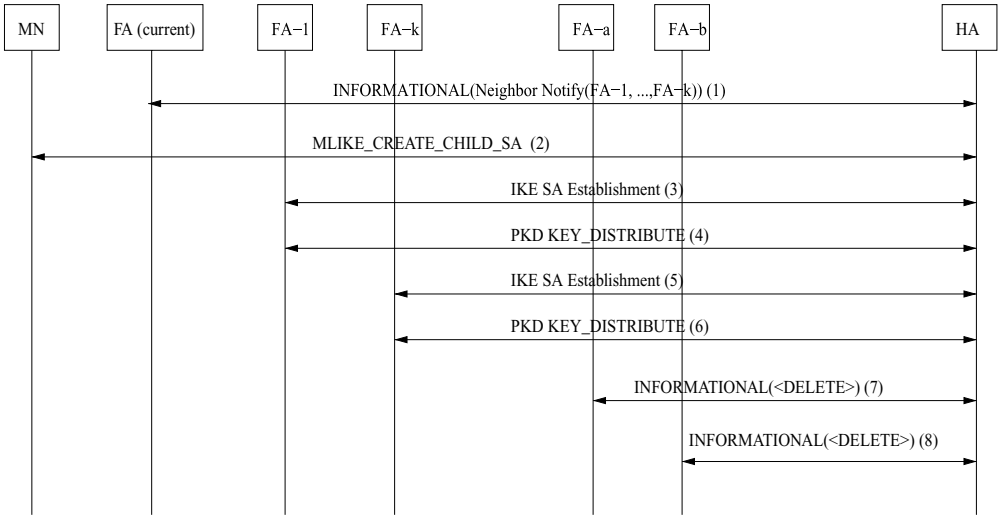


Fig. 9 MML-IPsec rekeying/revocation in PKD

In DKM, a new FA retrieves a MML-IPsec CSA from the previous FA. If the secret information remains with the previous FA, it increases the vulnerability of an attack. To address this vulnerability, the HA and the MN reestablish a new MML-IPsec CSA, where only the key values for the header zone are changed. The HA distributes this new CSA to the new FA. The new FA uses the old CSA until the new CSA is established. After establishing the new CSA, the HA revokes the old CSA from the previous FA.

Figure 8 shows the rekeying and revocation procedures in DKM. After retrieving the MML-IPsec CSA from the previous FA, a new FA requests rekeying to the HA (flow (2)). The HA initiates rekeying (flow (3)) and, in the same way as Fig. 7, distributes the new CSA to the new FA (flow (4)). The old CSA is deleted from the previous FA (flow (5)).

In PKD, as a MH moves, the MML-IPsec CSA is distributed to more FAs. As the number of FAs with the CSA increases, the less secure MML-IPsec is. To address this problem, the HA initiates rekeying when the number of FAs having the MML-IPsec CSA exceeds a threshold (a system parameter). First, the HA rekeys the MML-IPsec CSA with the MH, where only the keys for the header zone are changed (flow (2) in Fig. 9). Second, the HA distributes the MML-IPsec CSA to a new set of neighboring FAs (flows (3)–(6) in Fig. 9). Finally, the HA exchanges INFORMATIONAL (flows (7) and (8) in Fig. 9) to delete the previously distributed CSA from $S = \{\text{a set of FAs } | A - B\}$, where $A =$

{a set of existing neighboring FAs} and B = {a new set of current FA's neighboring FAs}.

4.5 Implementation

We implemented the key distribution protocols (MLIKE Initialization, PKD, and DKM) on the test bed shown in Fig. 3. We implemented MML-IPsec key distribution protocols based on IKEv1 and IKEv2. We describe the IKEv2 implementation. The Dynamic Mobile IP Linux implementation developed by Helsinki University of Technology (HUT) [10] was used in the test bed. Each subnetwork has a different set of wireless configuration parameters such as *essid* and *channel number*. The handoff occurs when the wireless configuration parameters are changed. We manually trigger handoffs through a shell script to run controlled experiments.

The implementation consists of several blocks: ML-IKE, PKD, DKM, Key Management, and the interface to MML-IPsec transport which will be described in the next section. The communication between these blocks is via the Unix Domain Socket in Linux.

ML-IKE manages, negotiates, and establishes the MML-IPsec CSA for Initialization, while PKD and DKM support mobility. ML-IKE also manages the state machine of the protocols.

The Key Management keeps the establishment state of both IKE SA and MML-IPsec CSA based on the source and destination addresses. Using the connection status, it coordinates Mobile IP and ML-IKE. If there is already an established MML-IPsec CSA in the case of PKD, the key management activates MML-IPsec through adding the route entry and IPsec binding. Otherwise, it initiates the establishment of the IKE SA if necessary.

To implement the interface between the key exchange protocols and the MML-IPsec module, we modified the Pluto Daemon by adding a new user interface to construct the CSA. The interface constructs a newly-defined zone message and sends the message to the PF_KEYv2 [16] socket. PF_KEYv2 is a new socket protocol family used by trusted privileged key management applications (e.g., ML-IKE, PKD and DKM) to communicate with the operating system's key management internals (i.e., FreeS/WAN's Security Association Database (SADB)). We have modified the PF_KEYv2 source code to handle the newly-defined zone message. The PF_KEYv2 socket will construct the CSA when it receives a zone message.

To implement these protocols based on IKEv1, we extended the IKE of FreeS/WAN [21], which is running in the Pluto Daemon, to include the new exchange types described previously. Furthermore, we implemented IKEv2 and key distribution protocols, using only a small part of Free/SWAN since FreeS/WAN does not support IKEv2.

Table 2 Message processing time

	Processing time		
Message flow	Initiator	Responder	Total
IKE SA establish	13.3 ms	13.42 ms	26.72 ms
MLIKE CSA establish	53.2 ms	29.1 ms	82.3 ms
MLIKE CREATE(rekey)	1.3 ms	1.35 ms	2.65 ms
MLIKE key distribution	0.4 ms	0.2 ms	0.6 ms
PKD key distribution	0.3 ms	0.2 ms	0.5 ms
DKM key distribution	0.3 ms	0.2 ms	0.6 ms
Neighbor notification	2.0 ms	7.3 ms	9.3 ms

4.6 Performance

In this section, we discuss the performance of the key management protocols measured on our test-bed. We implemented and tested MML-IPsec key distribution protocols based on IKEv1 and IKEv2. In this section, we show the performance of IKEv2-based implementation. We tested the performance of our integrated IPsec/Mobile IP solution as a baseline.

In Table 2, we show the message processing time measured by the *gettimeofday* system call in Linux. The results in Table 2 measure the processing time on receiving a message or event, excluding the pre-processing, post-processing and transmission time over the media. The node initiating the message flow is the "Initiator," and the recipient is called the "Responder". For example, in DKM, the Initiator is the new FA and the Responder is the previous FA providing the CSA information. In Initialization, the MH is the Initiator and the HA is the Responder.

In order to measure the handoff latency, we evaluate the time delay from the point that the MH sends a Mobile IP registration message until it finishes establishing the MML-IPsec CSA or IPsec SA. Table 3 shows the handoff delay for the pure Mobile IP, Mobile IP integrated with IPsec, and Mobile IP integrated with MML-IPsec.

The Mobile IP registration is necessary for all cases. The pure Mobile IP handoff comprises only the Mobile IP registration overhead and shows almost the same delay for the initialization and handoff phases. In the case of Mobile IP with IPsec, the initialization time consists of the Mobile IP registration and IPsec secure connection establishment between the HA and MH.

Similarly, the initialization of MML-IPsec takes between 539–542 milliseconds depending on if PKD or DKM is used. This includes the Mobile IP registration and MML-IPsec establishment with the key distribution to the FA or FAs. The MML-IPsec initialization time is made up of three components. First, the message processing time, including IKE SA establishments, MLIKE AUTH SA Exchanges, MLIKE Key Distribution, and Mobile IP registration procedures, is

Table 3 Handoff delay

Phase	Pure MIP	MIP with IPsec	MIP with MML-IPsec	
			PKD	DKM
Initialization	26 ms	197 ms	539(219) ms	542(222) ms
Handoff	25 ms	54 ms	52.7 ms	100.0 ms

Note. MIP is an abbreviated form of Mobile IP.

measured at approximately 220 milliseconds. Second, the FA runs four shell scripts to create local connections and bind the local connections into MML-IPsec interfaces when receiving the Mobile IP registration reply. The four shell scripts take 320 milliseconds to execute. Finally, there are transmission and internal communication latencies which are responsible for the remaining of the delay. These shell commands, and their additional overhead, may be eliminated if the source code for the various initialization procedures are modified to interact directly, a change planned for our next version.

In Table 3, we show the total latency for the Initialization (latency in parenthesis if the shell script overhead is eliminated), and handoff delay.

The MIP with IPsec column in Table 3 shows the initialization and handoff delay without performance enhancement functions in the FAs. The initialization of Mobile IP with IPsec takes 197 milliseconds, which includes Mobile IP registration (26 milliseconds), processing time of IKE SA and session SA establishment (109 milliseconds in the total shown in Table 2), IPsec message transmission delay, and internal configuration in the HA and MH. Here, the handoff delay takes only 54 ms.

However, if IPsec is integrated into FAs to deliver the performance enhancement and security functions together, the handoff may take at least as long as the initialization (197 ms). The performance enhancement functions require FAs to use the partial information (e.g. TCP/IP header) of the encrypted packet. In order to use the partial information, the FA must access the information in plaintext. Therefore, whenever a MH moves to a new foreign network, IPsec should be re-established between the HA and new FA, and between the new FA and MH.

On the other hand, the handoff in the MML-IPsec only takes 52.7 ms and 100 ms in PKD and DKM respectively. Thus, the MML-IPsec improves the overall handoff delay in an integrated setting of performance and security functions. The MML-IPsec Initialization in IKEv1 takes about 110 milliseconds more than in IKEv2. The handoff delay measurements show that a handoff using PKD incurs an additional 28 milliseconds of delay, while a handoff using DKM incurs an extra 75 milliseconds of delay. The handoff delays with IKEv1 are similar to IKEv2. These results are encouraging when we consider that these are well within the range of a TCP time-out value.

5 MML-IPsec transport implementation and evaluation

In this section, we first present our detailed implementations of the MML-IPsec protocol and SNOOP [2], and their integration. Then, we present the experimental methodology and the performance evaluation results. To improve performance, we propose an additional update to MML-IPsec called *dynamic MML-IPsec* in which a flow may switch between plaintext, IPsec and MML-IPsec. Some preliminary results in terms of throughput and overhead are discussed.

5.1 Implementation details

5.1.1 The implementation of MML-IPSEC

We follow the high-level design outlined in [22] to implement MML-IPsec. As discussed in Section 3, we modified this design to make it suitable for wireless networks, and to account for the fact we used Linux FreeS/WAN version 1.99 while in [22], version 1.1 was used.

In KLIPS, we introduce two new concepts as discussed in Section 3 and [22]: a zone and a CSA. A CSA has two elements: a zone map and a list of SAs for all the zones.

One SA in the SA list is chosen to be the designated SA. The designated SA is responsible for maintaining parameters at the IP datagram level and representing the CSA in security processing [22]. The designated SA must be consistent across all nodes involved in a CSA, and all nodes must have access to the corresponding zone. In our case, all the authorized nodes, including end hosts and authorized FAs, have access to the first zone; i.e., the TCP/IP header portion of the packet. Thus, we always choose the first zone's SA as the designated SA.

We create two new data structures, called zone and subzone, to accommodate the zone concept. The data structure for SAs is modified to accommodate the concept of CSA.

Figure 10 shows the modifications to the FreeS/WAN source code for realizing MML-IPsec. We modify three procedures: *ipsec_rcv()*, *ipsec_rcv_1()*, and *ipsec_tunnel_start_xmit()*.

ipsec_rcv() is used to authenticate (and/or decrypt) incoming packets before forwarding them to the transport layer. We extended it to handle multiple zones. For outgoing processing when sending a packet, *ipsec_tunnel_start_xmit()* is modified to perform encryption and authentication on two zones instead of the entire packet. *ipsec_rcv_1()* is a procedure added by us, which is used for forward packet processing. More specifically, when an authorized FA receives a MML-IPsec packet for which it has a valid SA, it uses *ipsec_rcv_1()* to decrypt the first zone of the packet. At this point, the FA may perform any smart processing, which may need the information contained in the packet header. Once this is done,

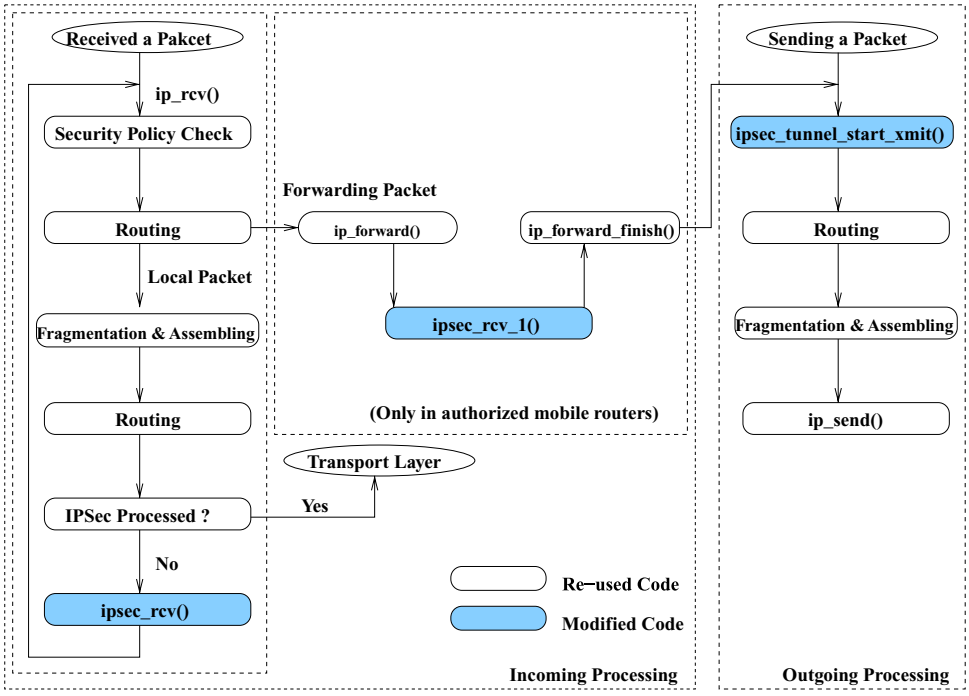


Fig. 10 Implementation of MML-IPsec

the first zone of the packet is re-encrypted and the packet is forwarded to the next hop.

5.1.2 The implementation of SNOOP

The SNOOP protocol was first defined in [2]. SNOOP executes in a base station and monitors the TCP header within the packet it is forwarding. The idea behind SNOOP is to detect errors incurred by the wireless link at the base station, and perform local retransmissions to recover from the errors locally. In this way, the TCP sender will not see the transmission error, resulting in a larger average window size and hence higher throughput. SNOOP detects errors by observing duplicate acknowledgments. If a duplicate acknowledgment is observed, SNOOP retransmits the subsequent TCP segment from its local buffer, and deletes the duplicate acknowledgment from the traffic stream. In this way, the lost segment is recovered without an end-to-end TCP retransmission. For details, please refer to [2].

The SNOOP protocol was first implemented in BSD/OS 2.0. In this work, we ported the code from user space and implemented it in the Linux kernel. To achieve this goal, we carried out three major modificiations. First, we designed a complete set of timer management functions to replace those in BSD (which are not supported in Linux). Second, we re-implemented the protocol using *sk_buff*, the data structure for memeory buffer management in Linux kernel, replacing the implementation based on *mbuf* in BSD. We also carefully re-designed SNOOP using the features of the *sk_buff* data structure. For example, in order to avoid copying the whole data packet when caching the packet at the base station, we use *pskb_copy()* to copy and save the data packet, which is similar to the reference counting mechanism in *mbuf*. In this way, we also get a private *sk_buff* header in which we can modify the TOS (Type of Service) field of the IP header to send a locally retransmitted packet with a higher priority. Lastly, we implemented a Poisson-distributed bit-error model. This model is used to generate errors (similar to that in [2]) and to evaluate the performance of SNOOP in a controlled settings.

5.1.3 Integration of MML-IPsec and SNOOP

The integration of MML-IPsec and SNOOP occurs at the authorized intermediate routers. In our test bed, these are the routers acting as base stations and FAs.

Figure 11 shows how the Linux kernel is modified to integrate MML-IPsec and SNOOP. From the figure, we can see that before the packet is forwarded, *ipsec_rcv_1()* is called to decrypt the first zone of the packet. The *ipsec_snoop_ctrl()* function is then called, which has access to the TCP/IP headers in plain text, and the SNOOP protocol is executed. If the packet should be forwarded, the first zone will be re-encrypted before it is transmitted to the next hop.

5.2 Dynamic MML-IPsec: an extension to MML-IPsec

To reduce the processing overhead at the FA, we propose an extension to MML-IPsec, called *dynamic MML-IPsec*, which allows MML-IPsec tunnel endpoints (either a MH or

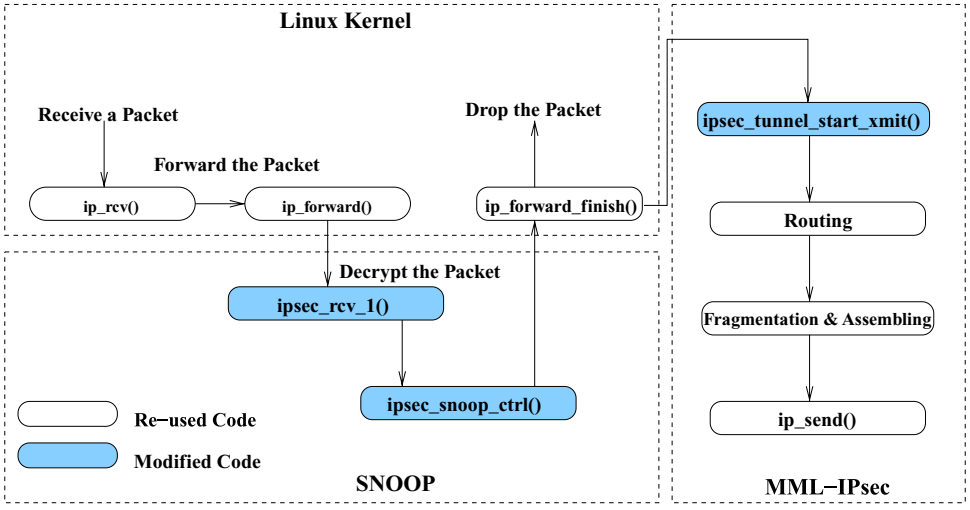


Fig. 11 Integration of MML-IPsec and SNOOP

a HA) to dynamically switch modes between MML-IPsec, IPsec, and plaintext. This strategy is desirable in several situations. For example, certain applications (such as MPEG-2 [1] video streaming) have certain frames containing more information than others. In these cases, applying MML-IPsec only to the high quality frames will provide more efficient transport without sacrificing security. More specifically, for the video streaming case, we can choose to encrypt the I-Frames and transmit the remaining frames—i.e., P-Frames and B-Frames—in clear text. As another example, if a MH determines that it has a high signal strength, and therefore likely has a reliable wireless link, it may not need to run a performance enhancing algorithm at the FA. In this case, it may choose to run IPsec end-to-end, and disable MML-IPsec.

Using dynamic MML-IPsec, a flow (or TCP/IP connection) may switch between plaintext, IPsec, and MML-IPsec. Switching from plaintext to MML-IPsec is simple: one entity just sends packets in MML-IPsec mode instead of in plaintext. When the other entity receives the first MML-IPsec packet, it knows that it should switch back to the MML-IPsec mode and starts to send packets in MML-IPsec mode accordingly. In a similar way, one can switch from plaintext to IPsec. This is also true for switching from IPsec to MML-IPsec.

Switching from MML-IPsec to plaintext, however, involves negotiating the parameters for dynamic MML-IPsec (between the two entities). The parameters may include how long they will communicate in plaintext, or in what interval the packets should be encrypted (such as for the video streaming case), etc. The two parties can use the secured MML-IPsec channel to negotiate these parameters. Then, based the negotiated parameters, both of them can switch into plaintext mode synchronously. Switching from MML-IPsec to IPsec (or from IPsec to plaintext) can be implemented in a similar way as above.

In this work, we present a simple design for dynamic MML-IPsec. That is, a tunnel endpoint can enable and disable MML-IPsec encryption periodically. In other words, instead of encrypting the packets all the time, we only encrypt the every i-th packet. The performance of dynamic MML-IPsec in such a design is evaluated in Section 5.5.

5.3 The experimental setup

We use the test bed shown in Fig. 3 to evaluate the performance of MML-IPsec and SNOOP. To validate our implementation of SNOOP, we transferred a large file from a fixed host to a MH with SNOOP running on the base station. A 5 MB file was transferred from ftp.redhat.com to the MH, with and without SNOOP running on the base station. Figure 12 shows the results for an average of five runs for different bit error rates, comparing the performance of TCP Reno and TCP Reno with SNOOP. These results match closely with that in [2].

Since we cannot implement MML-IPsec in ftp.redhat.com, and we want to run experiments in a controlled environment, we modify our test bed as shown Fig. 13. The laptop, named TLP1, acts as a MH. One desktop, named NETS, is configured as the base station for the MH. Another

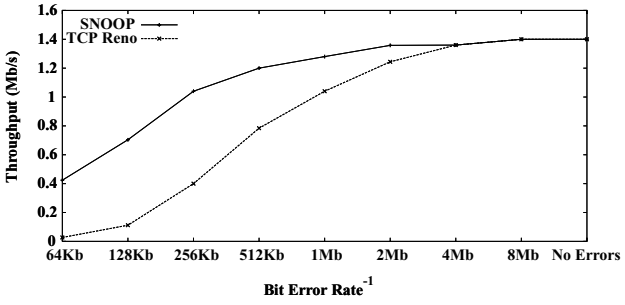


Fig. 12 Throughput Comparison of SNOOP and TCP Reno

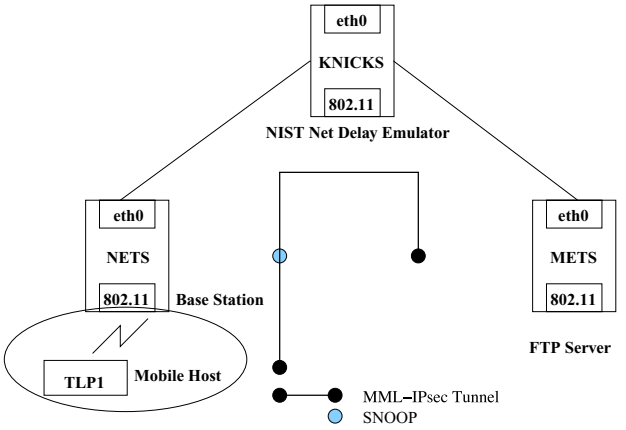


Fig. 13 Testbed

desktop, named METS, is configured as a FTP server. NETS and METS are connected using a 100M bps Ethernet. MML-IPsec is installed in TLP1, NETS and METS. We installed NIST Net [5], a network emulation package that runs on Linux, in KNICKS. NIST Net allows a Linux PC to be set up as a router to emulate a wide variety of network conditions. In our experiments, we use it to insert packet transmission delays between NETS and METS to emulate a wide area network. For example, to emulate downloading a file from a remote ftp site (e.g., ftp.redhat.com) with 50 ms delay, we insert 50 ms transmission delay using NIST Net. As shown in Fig. 13, all the traffic between TLP1 (the ftp client) and METS (the ftp server) are configured to route through KNICKS, where packet transmission delays are inserted.

SNOOP is installed in NETS and is integrated with MML-IPsec. We limit the raw bandwidth of the wireless link to 2M bps so that we can compare the results to that in [2]. The TCP data packet size in our experiments is 1,460 bytes when MML-IPsec is not running. When MML-IPsec is running, because of the additional IPsec headers (such as ESP header and IP over IP header if IPsec tunnel mode is in use), the TCP data packet size is smaller than 1,460 bytes. For example, in our experiments, where we use the tunnel-mode ESP protocol to encrypt the packet, the TCP data packet size is 1,391 bytes.

5.3.1 System parameters

We list the test bed's configuration parameters in Tables 4 and 5. Table 5 shows the parameters we use to test SNOOP, and Table 5 shows the parameters in evaluating MML-IPsec. In Table 4, the SNOOP Maximum Window, which is the data buffer size in the SNOOP module at the base station, is set to 50. This is large enough to buffer all the data packets sent from the sender so that the SNOOP module is never overloaded.

Table 4 SNOOP-related TCP module parameters

Maximum segment size (bytes)	1,460
Sender buffer size (bytes)	65,535
SNOOP maximum window	50
SNOOP maximum connections	64
Fast transmit	Enabled
Fast recovery	Enable
Selective ACK (SACK)	Disabled
Karn's algorithm	Enabled
SNOOP retransmission threshold	4
SNOOP initial RTO (ms)	100
SNOOP minimum RTO (ms)	100
SNOOP persist timeout (ms)	1,000
SNOOP Garbage timeout (ms)	10,000
NIST net transmission delay (ms)	100

Table 5 MML-IPsec parameters

IPsec tunnel mode	Enabled	
Encryption	Enabled	
Authentication	Disabled	
ML-IPsec overhead (bytes)	69	
Maximum segment size (bytes)	1,391	
Number of zones	2	
	First zone	Second zone
Zone size (bytes)	40	Variable
Encryption protocol	ESP	ESP
Encryption algorithm	3DES	3DES

In the SNOOP module, three timers are running: the local retransmission timer, the persist timer and the garbage timer. The SNOOP Initial RTO (Retransmission TimeOut) is used to set the initial timeout value for the local retransmission timer. The persist timer will expire if the SNOOP module does not receive any packets from either the receiver or the sender for a long period of time. When it expires, it will retransmit all the un-acknowledged data packets in its buffer. We set the persist timer to 1 s. The garbage timer is used to clear the buffer for a connection if there is no activity in that connection for a very long period of time. We set the garbage timer to 10 s in our implementation.

When testing MML-IPsec, we establish a MML-IPsec connection between TLP1 (the MH) and METS (the ftp server). The packets transmitted in this connection are encrypted using the tunnel-mode ESP encryption protocol. The IP packet is divided into two zones each of which is encrypted separately. As shown in Table 5, the first zone is 40 bytes, which includes the 20 byte IP header and 20 byte TCP header (not including the TCP Options and Padding fields); and the second zone covers the remainder of the packet. Both zones are encrypted with the ESP 3DES encryption algorithm using different SAs, each of which has different encryption/decryption keys. NETS has the SA for the first

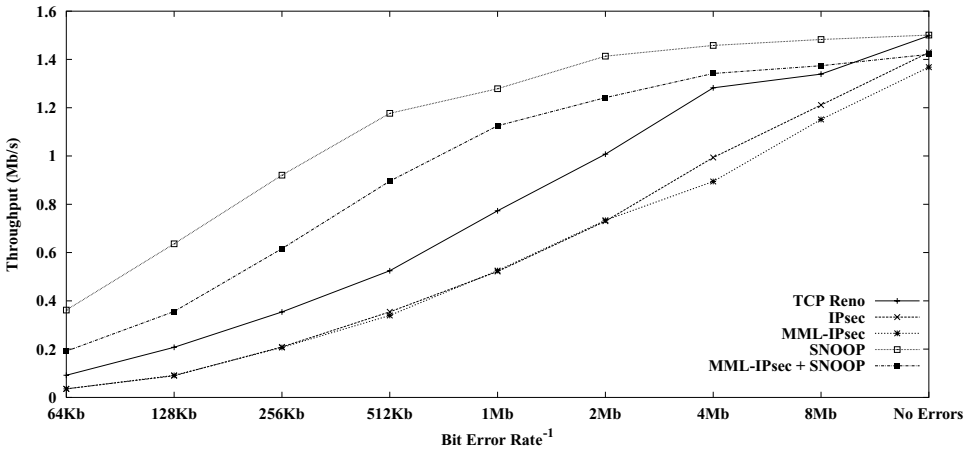


Fig. 14 Throughput comparison of different configurations

zone and decrypts the TCP/IP header of the packets so that it can execute SNOOP if enabled.

Five configurations have been tested: pure TCP Reno, TCP Reno over SNOOP, TCP Reno over IPsec, TCP Reno over MML-IPsec, and TCP Reno over SNOOP integrated with MML-IPsec. To evaluate these configurations under different wireless link conditions, we use the Poisson-distributed bit-error model to generate a wide range of bit-error rates. We test these configurations by transferring a 10 MB file from METs (the ftp server) to TLP1 (the MH) with different bit-error rates. For each configuration, we execute the transfer five times. Our comparison is based on the average throughput of the five runs.

5.4 Performance evaluation results

We use throughput as the metric to evaluate the performance of different configurations. To further explain the reason behind these results, we also illustrate the TCP congestion window size and the TCP sequence number of different configurations.

5.4.1 Throughput

Figure 14 compares the throughput of different configurations under different bit-error rates. From the figure, we see that SNOOP greatly improves the performance of TCP Reno when the error rate is high. For example, when the bit-error rate is 1.53×10^{-5} (1-bit error in 64 K bits), the throughput with SNOOP is three times higher than that without SNOOP. In fact, TCP Reno with SNOOP always achieves a higher throughput regardless of the bit-error rate. Even when the bit-error rate is very small, e.g., 1.19×10^{-7} (1-bit error in 8 M bits), SNOOP achieves 10% higher throughput than regular TCP Reno.

From the curves of IPsec and MML-IPsec, we can see that when no error exists, the overhead of these two protocols are about 5% and 9%, respectively. As the bit-error rate increases, the throughput of both protocols drops dramatically. It is interesting to see that when the bit-error rate is higher than 4.77×10^{-7} (1-bit error in 2 M bits), the throughput of IPsec and MML-IPsec is almost the same in most cases. For example, when the bit-error rate is 1-bit error per 2 M bits, the throughput of both approaches drop by 27%. When the bit-error rate is 1-bit error per 64 K bits, their throughput drops by 62%. This is because the overhead incurred by the error is more significant than the overhead incurred by encryption/decryption.

When SNOOP is integrated with MML-IPsec, the throughput is higher than either TCP Reno over MML-IPsec or IPsec. SNOOP integrated with MML-IPsec can deliver as much as five times higher throughput than that without SNOOP. Even when the error rate is low, SNOOP integrated with MML-IPsec can improve the performance by 19% and 14% compared to TCP over MML-IPsec and IPsec, respectively.

From the figure, we can also see that in an error-prone environment, SNOOP over MML-IPsec always achieves higher throughput than TCP Reno alone. The improvement increases as the bit-error rate increases. Figure 14 shows that SNOOP over MML-IPsec increases the throughout by 3% to 108% for different bit-error rates.

These results conclusively show that in a wireless error-prone environment, by integrating performance enhancing algorithms such as SNOOP with MML-IPsec we can achieve security and performance simultaneously.

5.4.2 TCP sequence number

Figure 15 shows the evolution of TCP sequence number versus time with different configurations when the bit-error rate is 1.9×10^{-6} (1-bit error in 512 K bits). From the figure, we can see that if SNOOP is running, the sequence number progresses much faster than that without SNOOP. For instance,

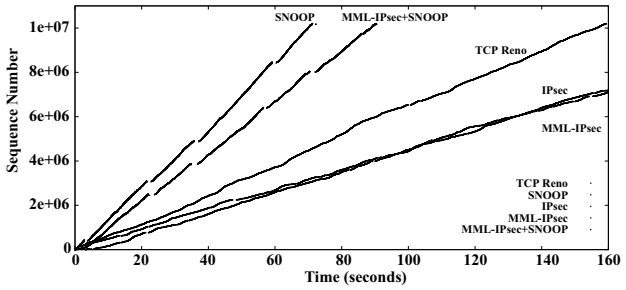


Fig. 15 Sequence number evolution of different configurations

when SNOOP is integrated with MML-IPsec, the sequence numbers increase at twice the rate of TCP with IPsec.

5.4.3 TCP congestion window size

Figure 16 shows the size of the congestion window versus time at the ftp server with a bit-error rate of 1.9×10^{-6} (1-bit error in 512 K bits). This figure compares three configurations: pure TCP Reno, TCP Reno over SNOOP, IPsec, MML-IPsec, and SNOOP integrated with MML-IPsec.

From the Fig. 16(a), we see that SNOOP maintains a much larger congestion window than pure TCP Reno. On average, the congestion window size of SNOOP is about 35 while TCP Reno's is about 6. This can be explained as follows. In SNOOP, the base station caches all the data packets before forwarding them to the MH over the wireless link. When the base station detects a packet loss, it will retransmit the packets to the MH from its local cache and suppress the duplicate ACKs. In this way, the sender's congestion control mechanisms, such as fast retransmit and fast recovery, will not be invoked. Thus, SNOOP prevents the congestion window from shrinking. In the case of pure TCP Reno, wireless loss will be treated as congestion in the network. Whenever a packet is lost or three duplicate ACKs are received, the transmitter will drop the congestion window size to half and then increase the window size gradually.

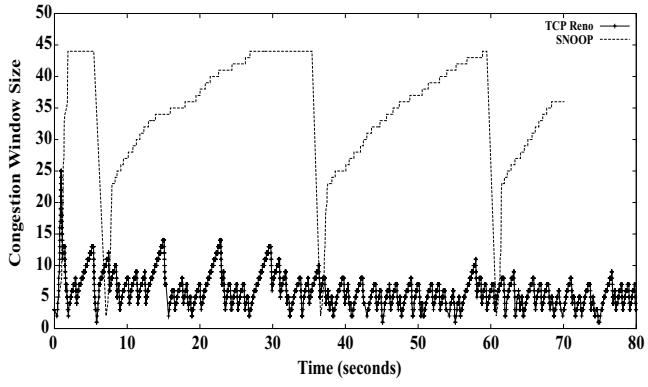
Figure 16(a) also shows that the congestion window size of SNOOP drops to two on several occasions. This is because of timeouts at the sender. Sender timeout occurs when the congestion window is very large and there are several packet losses in the same sending window. When the congestion window size is large, for example 40, a packet loss will generate as many as 39 duplicate ACKs if all the packets after the lost packet are received correctly. Our tests shows that to process all these duplicate ACKs requires up to ten times of the normal packet processing time. Therefore, packets transmitted after the lost packet will experience a long round-trip time and it is possible that the timer for these packets will expire before getting acknowledged. In such cases, the sender will fall back to the slow start phase and reduce the congestion window to two.

From Fig. 16(b), we can also see that when integrated with MML-IPsec, SNOOP has a much larger congestion window size than pure MML-IPsec, which can be also explained as above. The figure shows that the integration of SNOOP increases the average congestion window size of the MML-IPsec protocol by a factor of five, from 5 to 26. The improvement in the congestion window size shown here also explains the dramatic performance improvement achieved in Fig. 14, which shows that the throughput of MML-IPsec with SNOOP is increased by 170% compared to without SNOOP, when the bit-error rate is 1.9×10^{-6} (1-bit error in 512 K bits).

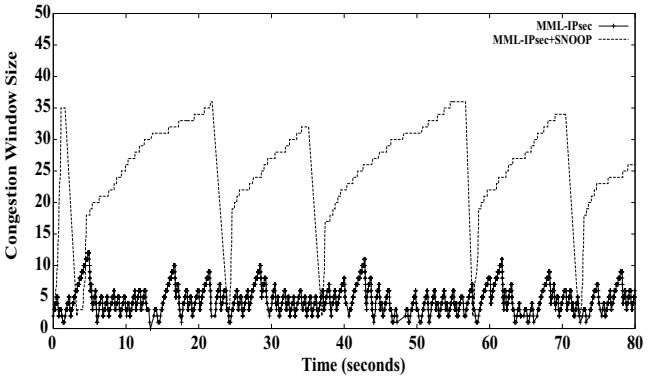
5.4.4 Code profiling measurements

To identify the processing bottleneck of our implementation of MML-IPsec, we profiled the source code. The purpose is to check whether we can further improve the performance by improving the coding.

In MML-IPsec, KLIPS performs encryption and decryption based on the security association negotiated by the Pluto Daemon. The code profiling is done in the KLIPS part by



(a) TCP Reno vs. SNOOP



(b) MML-IPsec vs. MML-IPsec with SNOOP

Fig. 16 Congestion window size

Table 6 Processing time of MML-IPsec (in microseconds)

Caller function	Callee function	Total time
ipsec_rcv		44
	decapsulation*	32
	memmove	0
	others	12
ipsec_tunnel_start_xmit		104
	ipsec_findroute	0
	encapsulation*	104
	ip_route_output	0
ip_send		3
decapsulation*		32
	policy_check	0
	lifetime_check	0
	authentication	5
	decryption	7
	discard_espheader	0
	ip_fast_csum	0
	policy_check	0
	others	20
encapsulation*		104
	cal_headroom	2
	strip_hardheader	0
	encryption*	91
	ipsec_findroute	0
	others	11

Note. the functions with '*' are operations instead.

measuring the processing time of the functions that carry out the encryption and decryption process. In KLIPS, a packet is encrypted using the function *ipsec_tunnel_start_xmit()* before sent. At the receiving side, it is decrypted by calling the function named *ipsec_rcv()*. Table 6 shows the processing times of this two functions, by calculating the processing times of all the functions it has called. We measure them at the HA. In the table, "others" shows the total processing time incurred by the other functions, which are not showed in the table. If the processing time is 0, it means that the processing time is less than 1 microsecond. The processing time for *ip_send()* is also included in this table. This value is small because the packet is transmitted through the fast Ethernet interface, which is 100 M bps.

From Table 6, we can see that, compared to IPsec, MML-IPsec introduces an extra 6 microseconds ((5+7)/2 = 6) in *ipsec_rcv* and 46 microseconds (91/2 = 46))in *ipsec_tunnel_start_xmit*. In our MML-IPsec implementation, each packet has two zones, and we need to do encryption/decryption and authentication twice for each packet. Considering the total processing time of encryption or decryption is within 1 millisecond, we claim that the overhead incurred by an extra zone is affordable.

We also measure the processing time of SNOOP in the intermediate node (i.e., FA) to quantify the snooping overhead.

Table 7 Processing time of SNOOP (in microseconds)

Function	Total
ipsec_rcv_1	42
ipsec_tunnel_start_xmit	0
ip_send	547
snoop_ctrl	13
snoop_getconn	1
snoop_untimeout	0
snoop_freebuf	1
snoop_wired_clear	0
snoop_done	14
snoop_addconn	3
snoop_conninit	7
snoop_gargage_timeout	22
snoop_data	0
snoop_wired	4

Table 7 shows the processing time of different functions in the SNOOP module.

Among the SNOOP functions, *ipsec_rcv_1* is called to decrypted the first zone of the packet. *snoop_ctrl* shows the average processing time used to snoop a packet, which is 13 microseconds. This shows that the overhead of SNOOP at the intermediate node is negligible. The function *ip_send* takes 547 microseconds because this time the packet is sending out through the slow (2 M bps) wireless link.

The code profiling shows that sending the packet through the wireless link is the bottleneck of our MML-IPsec infrastructure. Encryption, decryption and snooping operations only add tolerable delay to the system, thus they are fairly efficient. Since the bottleneck is a physical limitation, we claim that the performance of MML-IPsec cannot by improve much by optimizing the coding.

5.5 Dynamic MML-IPsec measurements

In the following, we evaluate the performance of dynamic MML-IPsec by enabling and disabling MML-IPsec

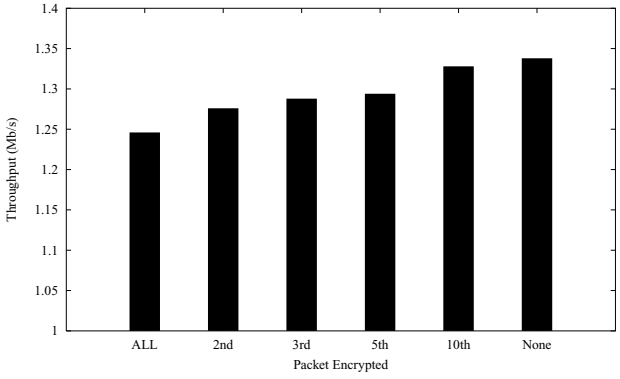


Fig. 17 Dynamic MML-IPsec throughput

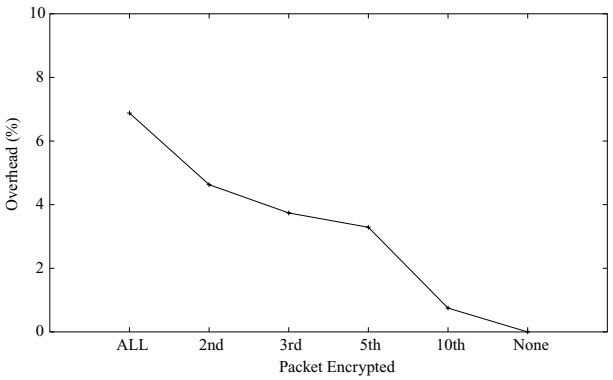


Fig. 18 Dynamic MML-IPsec overhead

encryption periodically. Figure 17 shows the throughputs of MML-IPsec when we encrypt the packets following a certain packet interval. That is, we encrypt every second, third, fifth packet and so on. The figure shows that, when we encrypt all the packets, the throughput is about 1.25 M bps. The throughput increases as we encrypt fewer packets.

Figure 18 shows the overhead of dynamic MML-IPsec. From the figure we can see that, if we encrypt all packets, the overhead is about 7%, compared to pure IPsec. The overhead decreases as we increase the encrypting interval. For example, if we only encrypt every the 10th packet, we only introduces 0.75% overhead compared to IPsec. This shows that using dynamic MML-IPsec is beneficial in applications where it is applicable.

6 Conclusions and future work

In this paper we have presented a simplified version of ML-IPsec, an efficient key distribution protocol for initializing secure wireless sessions, and two protocols for managing mobility for these secure sessions. We call this suite of protocols MML-IPsec. We showed through extensive performance testing of our implementations of these protocols that MML-IPsec successfully enables performance enhancing algorithms to be introduced into wireless networks. Specifically, we also support revoking keys in nodes that are no longer in an active route and rekeying without disrupting data transfer. In particular we showed that when SNOOP is integrated with MML-IPsec, impressive throughput gains are achieved over pure TCP, or if a secure session is desired, TCP over IPsec.

The MML-IPsec source code is profiled to identify the bottleneck of the implementation. It turns out that sending the data packet through the wireless link is the bottleneck. We also proposed an enhancement to MML-IPsec, called Dynamic MML-IPsec, which can achieve a tradeoff between performance and security. In addition, we showed that the mobility protocols add only a small amount of delay to the handoff time, significantly less than a typical TCP time-out value.

The compromise incurred when using MML-IPsec is that a single intermediate node inside a network is able to access TCP/IP header information in plain text; note that no other node in the network can view this information and that the user payload is still encrypted end-to-end. We believe these results justify the use of protocols such as MML-IPsec and motivate further work to improve the security aspects of this protocol.

Acknowledgments We would like to appreciate for anonymous referees' valuable comments which help us to improve the presentation of this paper. This work was supported by The Technology Collaborative (TTC), and NSF Grants CNS-0508114 and CNS-0519460.

References

1. Information Technology- Generic Coding of Moving Pictures and Associated Audio Information: Video, ISG/IEC International Standard 13818-2 (June 1996).
2. H. Balakrishnan, S. Seshan, E. Amir and R.H. Katz, Improving TCP/IP performance over wireless networks, in: *Proc. of ACM Mobicom* (1995).
3. J. Binkley, An integrated IPSEC and mobile-IP for freeBSD. Technical Report 01-10 (2001).
4. T. Braun and M. Danzeisen, Secure mobile IP communication. in: *Workshop on Wireless Local Networks at the 26th Annual IEEE Conference on Local Computer Networks* (2001).
5. M. Carson and D. Santay, NIST Net–A linux-based network emulation tool, *Computer Communication Review* (2004).
6. M. Chan and R. Ramjee, TCP/IP performance over 3G wireless links with rate and delay variation, in: *Proc. of ACM Mobicom* (2002).
7. T. Dierks and C. Allen, The TLS protocol, *IETF RFC 2246* (1999).
8. A. Freier, P. Karlton and P.C. Kocher, The SSL protocol, version 3.0, *Netscape Communications Corp.* (1996).
9. D. Harkins and D. Carrel, The internet key exchange protocol, *IETF RFC 2409* (1998).
10. Helsinki university of technology, Dynamics Mobile IP Software (October 2001).
11. C. Kaufman, Internet key exchange (IKEv2) protocol, *IETF Internet Draft draft-ietf-ipsec-ikev2-17.txt* (2004).
12. S. Kent and R. Atkinson. IP authentication header, *IETF RFC 2402* (1998).
13. S. Kent and R. Atkinson, IP encapsulating security payload (ESP), *IETF RFC 2406* (1998).
14. S. Kent and R. Atkinson, Security architecture for the internet protocol, *IETF RFC 2401* (1998).
15. D. Maughan, M. Schertler, M. Schneider and J. Turner, Internet security association and key management protocol (ISAKMP), *IETF RFC 2408* (1998).
16. D. McDonald, C. Metz and B. Phan, PF_KEY Key Management API, Version 2, *IETF RFC 2367* (July 1998).
17. G. Montenegro. Reverse tunneling for mobile IP, *IETF RFC 2344* (1998).
18. C. Perkins. IP mobility support for IPv4, *IETF RFC 3220* (2002).
19. C. Perkins and D. Johnson, Route optimization in mobile IP, *Mobile IP Working Group Draft draft-ietf-mobileip-optim-09.txt* (2000).

20. L. Salgarelli, M. Buddhikot, J. Garay, S. Patel and S. Miller, Efficient authentication and key distribution in wireless IP networks, *IEEE Communications Magazine* (2003).
21. www.freeswan.org. Introduction to FreeS/WAN 1.99 (October 2002).
22. Y. Zhang and B. Singh. A multi-layer IPSec protocol, in: *Proc. of 9th USENIX Security Symposium* (2000).

Heesook Choi is a Ph.D. candidate in the Department of Computer Science and Engineering at the Pennsylvania State University. She received her B.S. degree in Computer Science and Statistics and M.S. degree in Computer Science from the Chungnam National University, Korea, in 1990 and 1992 respectively. She was a senior research staff in Electronics and Telecommunications Research Institute (ETRI) in Korea before she enrolled in the Ph.D. program at the Pennsylvania State University in August 2002. Her research interests lie in security and privacy in distributed systems and wireless mobile networks, focusing on designing algorithms and conducting system research.

Hui Song is a Ph.D. candidate in the Department of Computer Science and Engineering at the Pennsylvania State University, University Park. He received the M.E. degree in Computer Science from Tsinghua University, China in 2000. His research interests are in the areas of network and system security, wireless ad-hoc and sensor networks, and mobile computing. He was a recipient of the research assistant award of the Department of Computer Science and Engineering at the Pennsylvania State University in 2005.

Guohong Cao received his BS degree from Xian Jiaotong University, Xian, China.He received the MS degree and Ph.D. degree in computer science from the Ohio State University in 1997 and 1999 respectively. Since then, he has been with the Department of Computer Science and Engineering at the Pennsylvania State University, where he is currently an Associate Professor. His research interests are wireless networks and mobile computing. He has published over one hundred papers in the areas of sensor networks, wireless network security, data dissemination, resource management, and distributed fault-tolerant computing. He is an editor of the IEEE Transactions on Mobile Computing and IEEE Transactions on Wireless Communications, a guest editor of special issue on heterogeneous wireless networks in ACM/Kluwer Mobile Networking and Applications, and has served on the program committee of many conferences. He was a recipient of the NSF CAREER award in 2001.

Thomas F. La Porta received his B.S.E.E. and M.S.E.E. degrees from The Cooper Union, New York, NY, and his Ph.D. degree in Electrical Engineering from Columbia University, New York, NY. He joined the Computer Science and Engineering Department at Penn State in 2002 as a Full Professor. He is the Director of the Networking and Security Research Center at Penn State. Prior to joining Penn State, Dr. La Porta was with Bell Laboratories since 1986. He was the Director of the Mobile Networking Research Department in Bell Laboratories, Lucent Technologies where he led various projects in wireless and mobile networking. He is a Bell Labs Fellow.

Dr. La Porta was the founding Editor-in-Chief of the IEEE Transactions on Mobile Computing and served as Editor-in-Chief of IEEE Personal Communications Magazine. He is currently the Director of Magazines for the IEEE Communications Society and is a member of the Communications Society Board of Governors. He has published over 50 technical papers and holds 28 patents. His research interests include mobility management, signaling and control for wireless networks, mobile data systems, and protocol design.