Performance of Host Identity Protocol on Symbian OS

Andrey Khurri, Dmitriy Kuptsov, and Andrei Gurtov
Helsinki Institute for Information Technology
Helsinki University of Technology and University of Helsinki

***Abstract*—The Host Identity Protocol (HIP) has been specified by the IETF as a new solution for secure host mobility and multihoming in the Internet. HIP uses self-certifying public-private key pairs in combination with IPsec to authenticate hosts and protect user data. While there are three open-source HIP implementations, little experience is available with running HIP on lightweight hardware such as a mobile phone. Limited computational power and battery lifetime of lightweight devices raise concerns if HIP can be used there at all. This paper describes the porting process of HIP on Linux (HIPL) and OpenHIP implementations to Symbian OS, as well as performance measurements of HIP over WLAN using Nokia E51 and N80 smartphones. We found that with 1024-bit keys, the HIP base exchange with a server varies from 1.68 to 3.31 seconds depending on whether the mobile phone is in standby or active state respectively. After analyzing HIP performance in different scenarios we make conclusions and recommendations on using IP security on lightweight hardware clients.**

I. Introduction

The current trend of moving mobile telecommunications systems to IP technology is well-recognized. However, the security aspect of using IP protocol stack on lightweight devices, such as PDAs or mobile phones, is not sufficiently explored. In particular, encryption and public key signatures implemented in software are computationally expensive operations that could stress CPU, memory and battery resources of mobile handsets.

The Host Identity Protocol (HIP) is a new security and mobility protocol standardized by the IETF [15], [16], [9], [13], [12], [18], [17], [19]. HIP uses IPsec ESP encryption of data and a version of Diffie-Hellman protocol to exchange public keys of two hosts. In this article, we describe performance measurements of two different HIP implementations ported to Symbian OS. In particular, we compare OpenHIP and HIPL protocol implementations running on two Symbian S60 smartphones with varying hardware resources. This study continues our previous work on the performance of HIP on Linux-based Nokia Internet Tablet [11]. Besides our studies HIP, has been evaluated exclusively on stationary Internet hosts with conventional PC-like resources [8], [7], [20], [10].

To check whether running IP-based security on lightweight cellular phones is feasible, we performed HIP measurements over WLAN with Nokia E51 and Nokia N80 in different scenarios. Particularly, we measured duration of the HIP base exchange and its parts, as well as CPU load, RAM utilization and power consumption during several phases of HIP daemon work. Finally, we analyzed the results and provided a set of recommendations on using unmodified HIP on lightweight clients.

Symbian OS is one of the leading operating systems for smartphones. 19.6 million Symbian mobile phones have been shipped globally in Q2 2008. The amount increased by 5% from the same period of 2007 [1]. Smartphones (in addition to traditional call and messaging functionality) comprise a set of rich media applications similar to computers. However, performance and usability of mobile applications is still a big concern. This is especially true with technologies initially designed to run on conventional PCs. The contribution of this work is evaluation of applicability of existing IP security solutions to smartphones. By porting two HIP implementations to Symbian we also contribute to HIP deployment. Our experience with the porting process might be useful for those planning to bring open source software to Symbian OS.

The rest of the paper is structured as follows. In Section II, we give a short background on HIP. Section III briefly describes Symbian OS networking architecture and our ports of HIPL and OpenHIP implementations. Section IV contains selected measurement results of the base HIP protocol and their in-depth analysis. Section V concludes the paper with a summary of key findings and future research directions.

II. Host Identity Protocol

A. HIP Architecture

The Host Identity Protocol (HIP) [15], [16], [6] was proposed to overcome the problem of using IP addresses simultaneously for host identification and routing. The idea behind HIP is based on decoupling the network layer from the higher layers in the protocol stack architecture (see Figure 1). HIP defines a new global name space, the Host Identity name space, thereby splitting the double meaning of IP addresses. When HIP is used, upper layers do not any more rely on IP addresses as host names. Instead, Host Identities (HI) are used in the transport protocol headers for establishing connections. IP addresses at the same time act purely as locators for routing packets towards the destination. For compatibility with IPv6 legacy applications, Host Identity is represented by a 128-bit long hash, the Host Identity Tag (HIT).

HIP offers several benefits including end-to-end security, resistance to CPU and memory exhausting denial-of-service (DoS) attacks, NAT traversal, mobility and multihoming support.

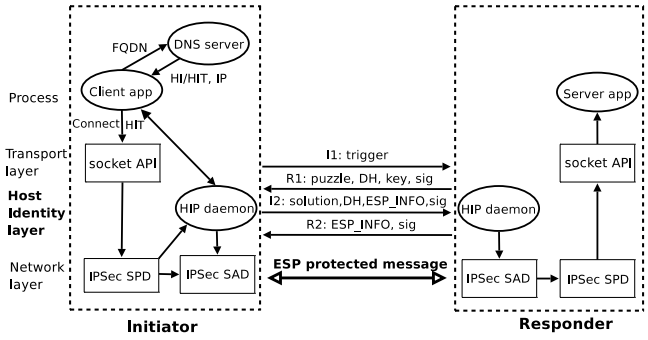

Fig. 1. HIP Architecture.

B. Base Exchange

To start communicating through HIP, two hosts must establish a HIP association. This process is known as the HIP Base Exchange (BEX) [16] and it consists of four messages transferred between the initiator and the responder. After BEX is successfully completed, both hosts are confident that private keys corresponding to Host Identifiers (public keys) are indeed possessed by their peers. Another purpose of the HIP base exchange is to create a pair of IPsec Encapsulated Security Payload (ESP) Security Associations (SAs), one for each direction. All subsequent traffic between communicating parts is protected by IPsec. A new IPsec ESP mode, Bound End-to-end Tunnel (BEET) [19] is used in HIP. The main advantage of BEET mode is low overhead in contrast to the regular tunnel mode.

Figure 1 illustrates the overall HIP architecture including the BEX. The initiator may retrieve the HI/HIT of the responder from a DNS directory [18] by sending a FQDN in a DNS query. Instead of resolving the FQDN to an IP address, the DNS server replies with a HI (FQDN→HI). Transport layer creates a packet with the HI as the destination identifier. During the next step HI is mapped to an IP address by the HIP daemon on the Host Identity layer. Finally, the packet is processed in the network layer and routed to the responder. As a result, the conventional 5-tuple socket becomes {protocol, source HI, source port, destination HI, destination port}.

C. Mobility and Multihoming

Since neither transport layer connections nor security associations (SAs) created after the HIP base exchange are bound to IP addresses, a mobile client can change its IP address (i.e., upon moving, due to a DHCP lease or IPv6 router advertisement) and continue transmitting ESP-protected packets to its peer. HIP supports such mobility events by implementing an end-to-end three-way signaling mechanism [17] between communicating nodes. HIP multihoming uses the same mechanisms as mobility for updating the peer with a current set of IP addresses of the host.

III. HIP ON SYMBIAN OS

In this section we describe the key parts of the HIPL and OpenHIP porting process and challenges that we faced while migrating to Symbian platform.

A. Networking Architecture on Symbian OS

Symbian networking architecture is dominantly based on a client-server communication model. Applications written as clients usually connect to and exchange data with particular servers (socket, telephony, serial communications, etc.). A server then communicates with low-level entities such as logical and physical device drivers (LDD and PDD) via an interface of server plug-in modules. Different module types include CSY (serial communications server), TSY (telephony server), PRT (socket server), and MTM (message type) modules. Clients do not directly access the plug-in modules. The latter are instead loaded by a server on demand [2].

The socket server (ESOCK) is responsible for socket APIs on Symbian and provides two types of interfaces: a BSD-like C socket API (based on Open C plug-in [5]) and an alternative Symbian-native C++ socket API. The socket server works with PRT protocol modules that are supplied in a form of dynamic link libraries (DLLs) with a .PRT extension. The TCPIP.PRT module comprises support of IPv4/v6, ICMP, TCP and UDP protocols, as well as DNS infrastructure [2].

Our HIP implementations for Symbian are entirely based on Open C plug-in that provides support of many standard C socket APIs. The Open C plug-in serves as an interface between the HIP daemon application and the PRT protocol modules in the Symbian networking stack.

B. Main Porting Stages

The porting process comprised several stages such as installation of the development environment, examination of the existing HIPL and OpenHIP source code, preparation of Symbian project structure and makefiles, compilation, debugging and testing.

1) Development Environment: We started with no prior knowledge and experience of Symbian OS. To begin porting process we needed to install an S60 3rd Edition Platform SDK for Symbian OS, a Carbide.c++ IDE and an Open C SDK plug-in for S60 3rd Edition SDK. The Open C plug-in brings support of nine standard POSIX and middleware C libraries to Symbian OS and allows easier porting of the existing C applications to S60 3rd Edition devices [5]. The availability of Open C plug-in played an essential role in our project as it provided access to many standard C functions and allowed to reuse the existing HIP implementations avoiding extensive modifications.

2) Project Preparation: Before actual porting we recommend to study existing software, its features and dependencies, and identify potential limitations of the target platform. To import the HIPL and OpenHIP code to the Carbide IDE and start working on the project we created a set of Symbian project files, *bld.inf* and **.mmp*, which are platform and compiler independent files in Symbian OS. To create those we studied existing Linux makefiles in the HIP projects. Having prepared the project files, one can build the project for different Symbian platforms and compilers.

For OpenHIP we chose a set of source files needed to run HIP in userspace mode, since we believed that this mode should be compatible with any platform that supports standard POSIX C libraries. We also included implementation of security association database (SADB) and PFKEY [14] protocol (with BEET mode support) for communication with SADB.

3) Compilation: The most common cause for compilation errors in the code was implicit data type conversions. Symbian compiler needs an explicit type casting to be performed. Furthermore, the Symbian compiler does not allow declaration of data types in the middle of a function. To avoid the compilation errors we had to add a few extra definitions to the Open C header file *netinet6/in6.h* for HIPL project.

OpenHIP architecture, in turn, was better suited for porting. In fact, we did not change any system headers. Similarly with HIPL, we have been using preprocessor logical statements to separate system-specific code parts, and in case of missing functionality reimplemented it.

4) Debugging: When debugging the HIPL code we found a number of porting issues that arose only during execution of the HIPL daemon. The errors were caused by a difference in Linux and Symbian emulator compilers. The most interesting issues were detected in data structures that contain an array of zero elements. The first error type concerned the size of such structures. In Linux, a structure member declared as an array of zero elements (e.g., *uint8_t data[0]*) does not increase the size of whole structure. On the contrary, the size of the same structure in the Symbian emulator was bigger due to the size of the ”null” array treated by the Symbian emulator compiler as one byte.

The second error type was related to memory alignment. Upon referencing arrays of zero elements in a structure, the program running on Linux and on Symbian emulator tried to access different memory blocks within that structure. Interestingly, we found that Symbian compiler always rises the total size of the structure elements preceding a ”null” array to an even value by adding an extra memory byte. As a result, to access a correct value recorded in the ”null” array we had to shift the pointer appropriately. It is worth mentioning that this specific feature has been detected only with the *mwcsym2* compiler that is used with the Symbian emulator. When the HIPL code was built for the target hardware with the *GCCE compiler*, the program behaved similarly with Linux and all changes we have made for the emulator needed to be restored.

C. Limitations of the Prototypes

Both HIP implementations are entirely written in C and consist of a HIP userspace daemon and several HIP libraries. As the HIPL project was originally developed for Linux, the implementation contained few platform-dependent features such as the *NETLINK* socket for kernel and userspace communication. To protect payload data, HIPL uses the IPsec protocol that resides in Linux kernel. Due to limited public Symbian SDK and restricted access to Symbian network stack, our HIP prototypes for Symbian support only the base protocol part without ESP encapsulation of data packets in the system kernel. However, with OpenHIP we ported a userspace alternative – PFKEY protocol and SADB. As a result, we were able to successfully encrypt/decrypt UDP encapsulated incoming ESP packets.

Open C plug-in itself has a set of limitations that required us to modify the existing source code and disable a part of its functionality. Examples of unsupported or restricted features in the Open C are *signals*, *fork()* and *exec()*, *wait()* and *waitpid()* functions, multiple *I/O consoles* [4]. Because of Open C constraints our HIP ports for Symbian use only UDP sockets to send HIP control packets excluding a raw-socket alternative as in the original HIP software. In OpenHIP, we used UDP encapsulation also for ESP packets, so that the raw socket limitation can be bypassed for ESP as well. The architecture of OpenHIP allowed us to support almost full-featured HIP implementation on Symbian OS. However, to run legacy applications over HIP an equivalent of Linux TUN/TAP driver needs to be implemented.

The HIP code ported to Symbian OS requires *NetworkService* system capability, which identifies a functionality for remote access to services that can produce cost to the phone user, such as network usage. Both HIP implementations compiled for the target hardware were signed with enabled *NetworkService* capability against a specific phone International Mobile Equipment Identity (IMEI) number and, without recompiling, cannot be used on any other S60 3rd edition mobile phone. To install the HIP daemon on another phone one has to sign the package with its own IMEI number at *www.symbiansigned.com*.

TABLE I
TECHNICAL SPECIFICATIONS OF TESTED PHONE MODELS

Smartphone Models →	**E60**	**N80**	**E51**
CPU Clock Rate, MHz	220	220	369
SDRAM / Free Exec RAM, MB	64/21	64/18	96/50
Battery Capacity, mAh	1020	860	1050

IV. PERFORMANCE EVALUATION

This section presents results of our measurements with the Host Identity Protocol on Symbian OS. First we introduce the hardware platforms and the network setup used to carry out experimentation. Then we report measurement results obtained with HIPL and OpenHIP prototypes and analyze them.

A. Our Testbed

We have tested HIPL and OpenHIP code running on several Symbian phones: Nokia E60, Nokia N80, and Nokia E51. The first two devices are based on S60 3rd Edition platform and Symbian OS v9.1, whereas Nokia E51 is a slightly newer smartphone that runs Symbian OS v9.2 and uses S60 3rd Edition Feature Pack 1 developer platform.

The general specifications of the tested phone models are summarized in Table I. Nokia E60 has equivalent to N80

TABLE II
BASE EXCHANGE DURATION WITH HIPL AND OPENHIP

Nokia E51	**Mean/Standard Deviation (s)**	
↓ **Scenario/Implementation** →	HIPL	OpenHIP
Phone→Server (Active)	3.17/0.11	3.09/0.17
Phone→Server (Standby)	1.68/0.06	1.90/0.12
Server→Phone (Active)	3.31/0.10	2.76/0.11
Server→Phone (Standby)	1.76/0.14	1.85/0.07
Phone→Phone (Active)	6.42/0.71	4.30/0.07
Phone→Phone (Standby)	3.78/0.13	3.50/0.12

TABLE III
KEY PAIR CREATION OF DIFFERENT SIZE ON NOKIA E51

Nokia E51	**Mean/Standard Deviation (s)**		
Key Length (bits) →	512	1024	2048
DSA	4.90/1.46	31.48/16.54	389.99/308.61
RSA	0.51/0.13	3.56/1.28	40.73/31.20

hardware resources and shows similar performance. To obtain competitive results we measured HIP performance on a more powerful Nokia E51 phone. All phone models support IEEE 802.11 b/g connectivity standards with WPA2 encryption.

We measured the performance of HIP over WLAN. Our experimental network consisted of a D-Link DGL-4300 access router, three mobile phones, and an Intel(R) Xeon(TM) server with 3.2 GHz CPU and 2 GB of RAM. The server was placed into the same network as cellular phones.

B. Measurements and Analysis

In basic scenarios, we established a HIP association between each of the Nokia phones and the server. We evaluated each stage of the base protocol separately including HIP daemon initialization, asymmetric key pair creation, daemon idle time, and protocol handshake (HIP base exchange). In this article, we mostly report results obtained on Nokia E51 that showed better performance than two other models.

With Nokia E51, we utilized Nokia Energy Profiler, a convenient tool that runs on the phone in background and allows monitoring hardware usage in real time, as well as exporting data to a PC for future analysis. Profiling data includes information about such parameters as power and memory consumption, and CPU load.

1) HIP Base Exchange Duration: In this section, we analyze HIP handshake duration in different scenarios. Surprisingly, we found a significant difference in HIP base exchange performance measured in *active* and *standby* phone states. We use terminology from [3] and slightly modify it. We call a phone state *active* when its display is switched on and refreshing (with backlight either on or off). In turn, we call a phone state *standby* when its display is in partial refresh (backlight is off; either date and time, text or animation is shown).

As Table II indicates, the total average time for HIPL base exchange initiated from the E51 phone to the server equals 3.17 sec in the active phone state. Switching the phone to the standby mode reduces HIP base exchange duration almost twice (1.68 sec). We believe the reason for such a great difference in performance is that in the standby mode no graphics are drawn and display is not refreshing, which releases extra CPU cycles that are utilized by the HIP daemon. On the other hand, in the active phone state, the processor load is close to 100% due to constant display refreshing, which prolongs processing time by the HIP daemon; we observed such behavior by activating and deactivating the phone screen with running Nokia Energy Profiler.

The scenario with E51 as a HIP initiator is more natural for the Internet where lots of connections are initiated from mobile clients to servers. In the opposite direction (server → phone) duration of the HIPL handshake slightly rises and becomes 3.31 sec in the active and 1.76 sec in the standby state (see Table II). Thus, time variation of the HIP base exchange performed in two opposite directions is insignificant. In our previous study [11] we obtained similar performance results with merely equal HIP handshake duration measured in two directions between a Nokia 770 Internet Tablet and a server. Further comparison to our previous work [11] indicates that the HIP base exchange performance on Linux-based Nokia 770 Internet Tablet is better than on Symbian-based Nokia E51 smartphone (1.40 vs. 1.68 sec), although the latter has more CPU and RAM resources. We explain this phenomenon as an impact of the Open C plug-in that is used with our Symbian HIP ports to wrap C function calls to native Symbian APIs.

Further analysis of the results in Table II indicates that the OpenHIP implementation shows slightly better performance than HIPL in the active phone state establishing a HIP association between the E51 and the server on the average during 3.09 sec and in the opposite direction in 2.76 sec. Though the effect of standby state in OpenHIP case is less significant than with HIPL (OpenHIP 35% against HIPL 47% time decrease when switching to standby mode).

A large part of the Internet traffic nowadays is generated by P2P applications. Keeping that in mind we measured HIP implementation performance running on two mobile phones. Our preliminary tests show (see Table II) that two Nokia E51 smartphones in standby state are able to establish a HIPL association in 3.78 sec and an OpenHIP association in 3.50 sec. Switching to the active mode significantly increases the HIP handshake time for HIPL (6.42 sec) while not seriously affecting OpenHIP (4.30 sec).

Although it is interesting to know the HIP base protocol performance level in the standby phone state (i.e., when the HIP daemon is implemented as an engine and runs in background) we have to rely on the results obtained in the active state. This is because we expect user to interact with mobile phone (thus, activating the display) while using applications that might benefit from HIP.

2) Asymmetric Key Pair Creation: Table III includes duration of creating public-private key pairs of different size

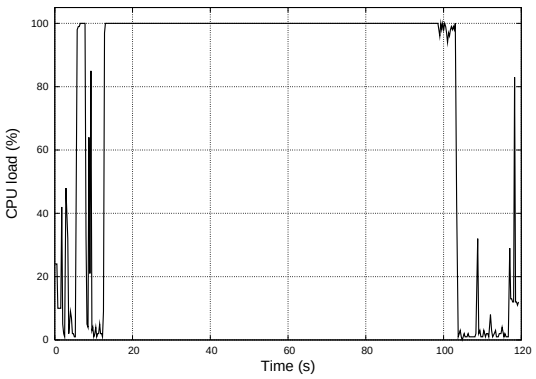

Fig. 2. HIPL Daemon Initialization. CPU Load on E51

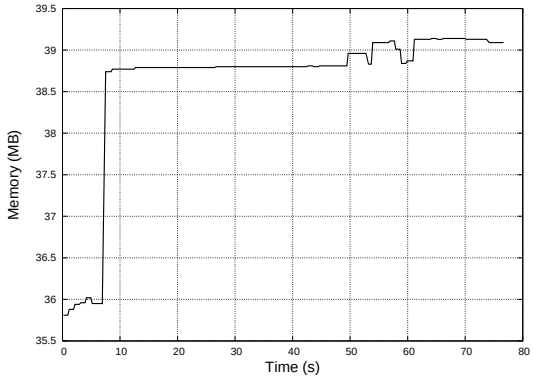

Fig. 3. OpenHIP Daemon Initialization with BEX. RAM Usage on E51

on Nokia E51. The results indicate an exponential growth of the key pair generation time with increasing the key length. With conventional 1024-bits keys, an asymmetric DSA key pair is created on E51 in 31.48 sec. The generation of an equivalent RSA key pair is much faster and equals 3.56 sec on the average. It is worth noting that the use of stronger keys (e.g., with length of 2048 bits) would produce a delay of several minutes with DSA and tens of seconds with RSA.

One might argue that keys are needed to be created only once, e.g., upon installing HIP and this would not affect the overall phone performance on the long run. Nevertheless, we believe that stressing the mobile phone even for a short time period is inconvenient and might be considered dangerous in special cases when the phone functionality is crucial (as with emergency calls). According to our practical experience, generation and usage of lengthy keys on mobile phones with lower amount of RAM and CPU power, such as E60 or N80, seriously affects handset performance and can potentially make the phone completely unresponsive for a long time.

3) Hardware Utilization: In this section, we report the indicators of hardware utilization that were collected with Nokia Energy Profiler on Nokia E51.

a) CPU Load: The CPU Load during HIPL daemon initialization and asymmetric key pair creation on E51 is presented in Figure 2. Most of the 2-minute time slot the usage stays at 100% and this corresponds to generation of four different public-private key pairs. The rest of the graph has few random peaks that account for precreation of the HIP R1 packets. In idle time the HIPL daemon does not consume a lot of processor power. We also noticed that switching the phone to the active state rises the CPU load notably. CPU utilization with OpenHIP implementation is similar to the HIPL case.

b) RAM Usage: Although each mobile phone has certain amount of RAM memory, only a part of it is available to applications. For example, on Nokia E60 only 21 out of 64 MB are available to the executables. The rest of the memory is reserved for exclusive use by the system. This approach reduces the number of mobile S60 applications that can be simultaneously ran on the device. According to our profiler data, memory usage on N80 phone stays on the level of 20 MB during HIP daemon initialization, key creation and HIP handshake. Note that this value depicts the overall memory usage by all running application. Assuming that other applications are in the idle state, we can figure out the memory use by HIP daemon. On Nokia E51, dynamics of memory use is almost the same. The only difference is the amount of available RAM, which is bigger on E51. According to technical specifications, E51 allocates to the applications approximately 50 MB out of total 96 MB of RAM.

With OpenHIP, the time interval from 8th to 50th second in the initialization phase (see Figure 3) corresponds to the key creation and serialization. The reader should take into consideration a fact from previous paragraph and notice that Figure 3 shows overall memory consumption. In fact, HIP starts its initialization at point of 36 MB. Since in OpenHIP all precreated keys are stored in RAM (as well as serialized to the file system) the memory use increases by 3 MB. However, during the base exchange and idle time (Figure 3, the time interval from 50th to 80th second) memory usage does not grow drastically, and only increases by a slight amount during BEX packets processing (which is freed afterward) and adding SAs to SADB. In fact, we believe that using 3 to 4 MB of RAM is within the normal bounds and should not stress a phone performance. As a result, we think that from RAM utilization prospective both HIPL and OpenHIP could be run on a Symbian mobile device without major changes to its architecture.

c) Power Consumption: Figure 4 illustrates the power consumed by the E51 phone with the running HIP daemon during its initialization and idle time. The value of 0.62 Watt represents an average level of power consumption over a 2-minutes measurement period. The peaks on the graph between the 10th second and the 52nd second timestamps show the maximum power consumption over the measurement period and they account for the creation of the DSA and RSA key pairs.

To compare how HIP affects the battery life of the phone we measured the average power consumption while phone was in "normal" use (i.e., no HIP daemon was running) and with HIP daemon doing the Base Exchange. As a result, we obtained 222mW/60mA and 333mW/90mA correspondingly, or around 18 and 12 hours of a 1050-mAh battery lifetime.

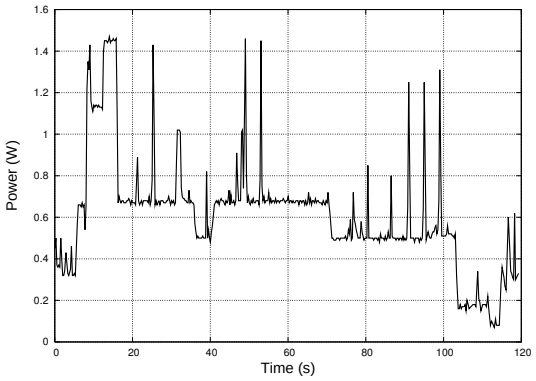

Fig. 4. HIPL Daemon Initialization. Power Consumption on E51

V. Conclusions

This paper presented measurements and performance evaluation of two separate Host Identity Protocol ports on Symbian OS. We found several interesting results. Most of them should be regarded as recommendations on using IP security on lightweight hardware such as Symbian OS mobile phones.

- The unmodified HIP protocol can be used in scenarios where a lightweight mobile phone communicates with the rest of the Internet through a single proxy server. A single HIP Base Exchange is then sufficient for the whole browsing session.
- For scenarios involving two mobile phones or multiple parallel HIP associations, unmodified HIP is too heavy for lightweight devices. Whilst the base exchange between a phone and a server takes $1.68 - 3.17$ seconds, two mobile phones require $3.50 - 6.42$ seconds to establish an association.
- The public-private key pair generation might stress the phone, especially with the key length greater than 1024 bits.
- Key creation stresses the CPU and consumes a lot of power, but otherwise, the HIP daemon in idle mode consumes few resources. However, WLAN transmission impact has to be considered separately.
- The OpenHIP protocol implementation had been a lot easier to port and showed slightly better performance over HIPL.
- Better performance results could have been achieved if HIP were implemented using native Symbian C++ APIs rather than Open C plug-in. Open C is a wrapper to native Symbian APIs and, thus, produces an additional overhead comparing to native applications.
- We believe that our measurement results are applicable to a wide range of IP security and mobility protocols in addition to HIP, and to other Symbian S60 phone models with similar hardware. Most such security protocols rely on similar public-key and IPsec ESP operations as HIP.

In future research, we plan to compare our empirical results with related work on evaluation of different security mechanisms on mobile lightweight devices. It could provide insights for reducing the overhead of public-key cryptography that we observed in this paper.

Acknowledgments

We thank Paras Tikmani and Sankalp Bose, our Indian summer interns, for their contribution to the project. We are grateful to Andrey Lukyanenko and Miika Komu who assisted a lot during our debug sessions. We also thank our partners from Nokia Research Center who contributed a lot of brain power to the discussions on migration to Symbian.

References

[1] Symbian Fast Facts Q2 2008, Mar. 2008. [Online] Available at: http://www.symbian.com/about/fast.asp.

[2] Forum Nokia. Symbian OS: Overview To Networking, June 2005. [Online] Available at: http://sw.nokia.com/id/c4536832-3dd0-45af-94be-1c4289cc3003/Symbian_OS_%Overview_To_Networking_v1_0_en.pdf.

[3] Forum Nokia. Nokia Energy Profiler Quick Start Guide, Feb. 2009. [Online] Available: http://www.forum.nokia.com/Resources_and_Information/Explore/Developmen%t_Process_and_User_Experience/Power_Management/Nokia_Energy_Profiler_Quick_Sta%rt.xhtml.

[4] Forum Nokia. Open C API Reference, Feb. 2009. [Online] Available: http://www.forum.nokia.com/document/CDL_Extension_S60_3rd_Ed_FP2/?conte%nt=GUID-719955DA-415B-420E-9F9B-F6DB37615EC5/html/Open_C_DocumentationIndexPag%e.html.

[5] Forum Nokia. Open C/C++ Documentation, Feb. 2009. [Online] Available at: http://www.forum.nokia.com/Resources_and_Information/Documentation/Open%_C_and_C++.xhtml.

[6] A. Gurtov. *Host Identity Protocol (HIP): Towards the Secure Mobile Internet.* Wiley and Sons, 2008.

[7] T. R. Henderson. Host mobility for IP networks: A comparison. *IEEE Network*, 17(6):18–26, Nov. 2003.

[8] T. R. Henderson, J. M. Ahrenholz, and J. H. Kim. Experience with the Host Identity Protocol for secure host mobility and multihoming. In *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC'03)*, Mar. 2003.

[9] P. Jokela, R. Moskowitz, and P. Nikander. Using the encapsulating security payload (ESP) transport format with the host identity protocol (HIP). IETF RFC 5202, Mar. 2008.

[10] P. Jokela, T. Rinta-Aho, T. Jokikyyny, J. Wall, M. Kuparinen, H. Mahkonen, J. Melen, T. Kauppinen, and J. Korhonen. Handover performance with HIP and MIPv6. In *Proc. 1st International Symposium on Wireless Communication Systems, ISWCS'04*, Sept. 2004.

[11] A. Khurri, E. Vorobyeva, and A. Gurtov. Performance of Host Identity Protocol on lightweight hardware. In *MobiArch '07: Proceedings of the 2nd ACM/IEEE International Workshop on Mobility in the Evolving Internet Architecture*, pages 1–8, New York, NY, USA, Aug. 2007. ACM.

[12] J. Laganier and L. Eggert. Host identity protocol (HIP) rendezvous extension. IETF RFC 5204, Mar. 2008.

[13] J. Laganier, T. Koponen, and L. Eggert. Host identity protocol (HIP) registration extension. IETF RFC 5203, Apr. 2008.

[14] D. L. McDonald, C. W. Metz, and B. G. Phan. PF_KEY key management api, version 2: draft-mcdonald-pf-key-v2-05, Feb. 2005. Work in progress.

[15] R. Moskowitz and P. Nikander. Host Identity Protocol architecture. RFC 4423, IETF, May 2006.

[16] R. Moskowitz, P. Nikander, P. Jokela, and T. Henderson. Experimental host identity protocol (HIP). IETF RFC 5201, Apr. 2008.

[17] P. Nikander, T. Henderson, C. Vogt, and J. Arkko. End-host mobility and multihoming with the host identity protocol (HIP). IETF RFC 5206, Apr. 2008.

[18] P. Nikander and J. Laganier. Host identity protocol (HIP) domain name system (DNS) extension. IETF RFC 5205, Mar. 2008.

[19] P. Nikander and J. Melen. A bound end-to-end tunnel (BEET) mode for ESP: draft-nikander-esp-beet-mode-09, Aug. 2008. Work in progress.

[20] P. Nikander, J. Ylitalo, and J. Wall. Integrating security, mobility, and multi-homing in a HIP way. In *Proc. of Network and Distributed Systems Security Symposium (NDSS'03)*, San Diego, CA, USA, Feb. 2003. Internet Society.