Experiments in Wireless Internet Security

Vipul Gupta, Sumit Gupta
Sun Microsystems Labs
Sun Microsystems
901, San Antonio Road
Palo Alto, CA 94303, USA
Email: vipul.gupta@sun.com, gupta.sumit@sun.com

***Abstract*— Internet enabled wireless devices continue to proliferate and are expected to surpass traditional Internet clients in the near future. This has opened up exciting new opportunities in the mobile e-commerce market. However, data security and privacy remain major concerns in the current generation of "wireless web" offerings. All such offerings today use a security architecture that lacks *end-to-end* security. This unfortunate choice is driven by perceived inadequacies of standard Internet security protocols like SSL (Secure Sockets Layer) on less capable CPUs and low-bandwidth wireless links.**

This paper presents our experiences in implementing and using standard security mechanisms and protocols on small wireless devices. Our results show that SSL is a practical solution for ensuring end-to-end security of wireless Internet transactions even within todays technological constraints.

***Index Terms*—Wireless Internet, Wireless security, SSL, end-to-end security, J2ME™.**

I. INTRODUCTION

THE past few years have seen an explosive growth in the popularity of small, handheld devices (mobile phones, PDAs, pagers), that are wirelessly connected to the Internet. These devices, which are predicted to soon outnumber traditional Internet hosts like PCs and workstations, hold the promise of ubiquitous ("anytime, anywhere") access to a wide array of interesting services. However, limitations of these battery-driven devices like small volatile and non-volatile memory, minimal computational capability, and small screen sizes, make the task of creating secure, useful applications for these devices especially challenging.

It is easy to imagine a world in which people rely on connected handheld devices not only to store their personal data, check news and weather reports, but also for more security sensitive applications like on-line banking, stock trading and shopping - all while being mobile. Such transactions invariably require the exchange of private information like passwords, PINs and credit card numbers and ensuring the secure transport of this information through the network becomes an important concern.[1]

[1] While protecting data on the device is also important, mechanisms for doing so are already available and not discussed further in this paper.

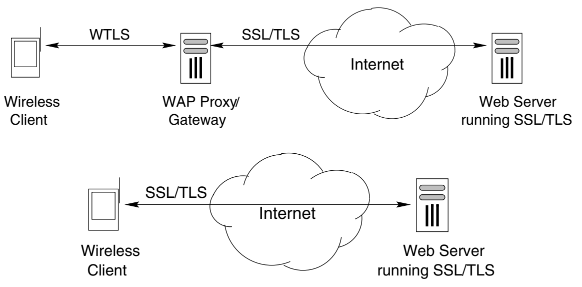


Fig. 1. (a) Proxy-based architecture (top), (b) End-to-end architecture (bottom)

On the wired Internet, Secure Sockets Layer (SSL) [1] is the most widely deployed and used security protocol[2]. It often takes years of widespread public review and multiple iterations [2], [3] to discover and correct subtle but fatal errors in the design and/or implementation of a security protocol. Over the years, the SSL protocol and its implementations have been subjected to careful scrutiny by security experts [4]. No wonder then that today SSL is trusted to secure sensitive applications ranging from web banking to online trading to all of e-commerce. The addition of SSL capabilities to mobile devices would bring the same level of security to the wireless world.

Unfortunately, none of the popular wireless data services today offer SSL on a handheld device. Driven by perceived inadequacies of SSL in a resource constrained environment, architects of both WAP and Palm.net chose to use a proxy based architecture, which is depicted in Fig. 1(a). In this approach, a different security protocol (incompatible with SSL) is used between the mobile client and the proxy/gateway(e.g., WAP uses WTLS [5] and Palm.Net uses a proprietary protocol on the wireless link). The proxy then decrypts encrypted data sent by a WAP phone using WTLS and re-encrypts it using SSL before forwarding it to the eventual destination server. The reverse process is used for traffic flowing in the opposite direction.

Such a proxy-based architecture has some serious drawbacks. The proxy is not only a potential performance bottleneck, but also represents a "man-in-the-middle" which is privy to all "secure" communications. Even though the data is en-

[2] Throughout this paper, we use SSL to refer to all versions of this protocol including version 3.1 also known as Transport Layer Security (TLSv1.0).

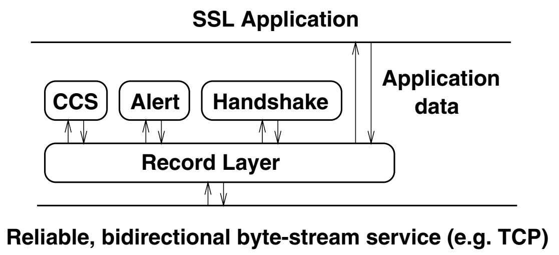


Fig. 2. SSL Architecture

crypted on both the wireless and wired hops, anybody with access to the the proxy itself could see the data in cleartext. Sometimes the situation is even worse and no encryption is used on the wireless side. This lack of end-to-end security is a serious deterrent for any organization thinking of extending a security-sensitive Internet-based service to wireless users. Banks and brokerage houses are uncomfortable with the notion that the security of their customers' wireless transactions depends on the integrity of the proxy under the control of an untrusted third party. Moreover, the proxy is often pre-programmed into the clients' devices, which may raise legal issues [6]. For instance, Palm.Net users have to go through the proxy owned and operated by Palm.

In contrast, the use of SSL between desktop PCs/workstations and Internet servers offers true end-to-end security (Fig. 1(b)). This holds true even when an HTTPS proxy is used to traverse firewalls. Unlike the WAP or Palm.net proxy, an HTTPS proxy only acts as a simple TCP relay shuttling encrypted bytes from one side to the other without decryption/re-encryption.

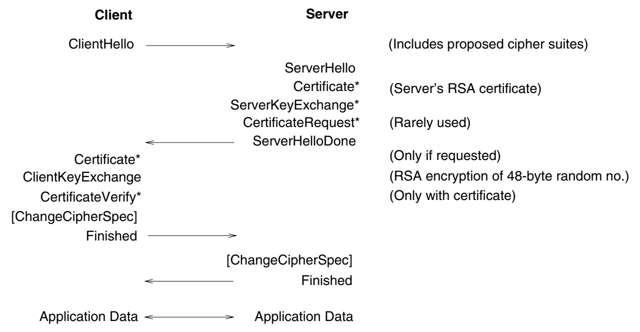
We felt that the claims about the unsuitability of SSL for mobile devices had not been adequately substantiated [7]. This prompted our experiments in evaluating SSL (considered too "big" by some) for small devices. We sought answers to some key questions: Is it possible to develop a usable implementation of SSL for a mobile device and thereby provide end-to-end security? How would near-term technology trends impact the conclusions of our investigation?

The rest of this paper describes our experiments in greater detail. Section II provides a brief overview of the SSL protocol. Section III discusses our implementation of an SSL client, called KSSL, on a Palm PDA and evaluates its performance. Section IV describes an application we've developed for secure, mobile access to enterprise resources based on KSSL. Section V talks about mobile technology trends relevant to application and protocol developers. Finally, we offer our conclusions in Section VI.

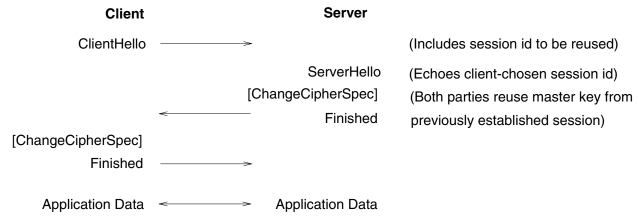
II. Secure Sockets Layer (SSL)

A. Overview

SSL offers encryption, source authentication and integrity protection of application data over insecure, public networks. Fig. 2 shows the layered nature of this protocol. The Record layer, which sits above a reliable transport service like TCP, provides bulk encryption and authentication using symmetric-key algorithms. The keys for these algorithms are derived from a *master secret* established by the Handshake protocol between the SSL client and server using public-key algorithms.



(a) Full Handshake


Client
Server
ClientHello
(Includes session id to be reused)
ServerHello
[ChangeCipherSpec]
Finished
(Echoes client-chosen session id)
(Both parties reuse master key from previously established session)
[ChangeCipherSpec]
Finished
Application Data
Application Data

(b) Abbreviated Handshake

Fig. 3. The SSL Handshake protocol

SSL is very flexible and can accommodate a variety of algorithms for key agreement, encryption and hashing. To guard against adverse interactions (from a security perspective) between arbitrary combinations of these algorithms, the standard specification explicitly lists combinations of these algorithms, called *cipher-suites*, with well-understood security properties.

The Handshake protocol is the most complex part of SSL with many possible variations (Fig. 3). In the following subsection, we focus on its most popular form, which uses RSA key exchange and does not involve client-side authentication. While SSL allows both client- and server-side authentication, only the server is typically authenticated due to the difficulty of managing client-side certificates. Client authentication, in such cases, happens at the application layer, *e.g.* through the use of passwords sent over an SSL-protected channel.

1) Full SSL Handshake: When an SSL client encounters a new server for the first time, it engages in the full handshake shown in Fig. 3(a). The client and server exchange random nonces (used for replay protection) and negotiate a mutually acceptable cipher suite in the first two messages. The server then sends its RSA public-key inside an X.509 certificate. The client verifies the public-key, generates a 48-byte random number (the *pre-master secret*) and sends it encrypted with the server's public-key. The server uses its RSA private-key to decrypt the pre-master secret. Both end-points use the pre-master secret to create a master secret which, along with previously exchanged nonces, is used to derive the cipher keys, initialization vectors and MAC (Message Authentication Code) keys for the Record Layer.

2) Abbreviated SSL HandShake: A client can propose to reuse the master key derived in a previous session by including that session's (non-zero) ID in the first message.[3] The server indicates its acceptance by echoing that ID in the second message. This results in an abbreviated handshake without certificates or public-key cryptographic operations so fewer (and shorter) messages are exchanged (see Fig. 3(b)). An abbreviated handshake is significantly faster than a full handshake.

B. SSL on Small Devices: Common Perception v/s Informed Analysis

SSL is commonly perceived as being too heavyweight for the comparatively weak CPUs in mobile devices and their low-bandwidth, high latency wireless networks. The need for RSA operations in the handshake, the verbosity of X.509 encoding, the chattiness (multiple round trips) of the handshake and the large size of existing SSL implementations are all sources of concern.

However, we are not aware of any empirical studies evaluating SSL for small devices and careful analysis of the protocol's most common usage reveals some interesting insights:

- Some constraints ease others. If the network is slow, the CPU doesn't need to be very fast to perform bulk encryption and authentication at network speeds.
- A typical SSL client only needs to perform RSA public-key, rather than private-key, operations for signature verification and encryption. Their small exponents (typically no more than 65537, a 17-bit value) make public-key operations much faster than private-key operations. It is worth pointing out that the performance of RSA public-key operations is comparable to that of equivalent Elliptic Curve Cryptography (ECC [8]) operations [9].
- There are several opportunities to amortize the cost of expensive operations across multiple user transactions. Most often, a client communicates with the same server multiple times over a short period of time, *e.g.* such interaction is typical of a portal environment. In this scenario, SSL's session reuse feature greatly reduces the need for a full handshake.
  Although SSL can be used to secure any connection-oriented application protocol (SMTP, NNTP, IMAP), it is used most often for securing HTTP. The HTTP 1.1 specification encourages multiple HTTP transactions to reuse the same TCP connection. Since an SSL handshake is only needed immediately after TCP connection set up, this "persistent HTTP" feature further decreases the frequency of SSL handshakes.

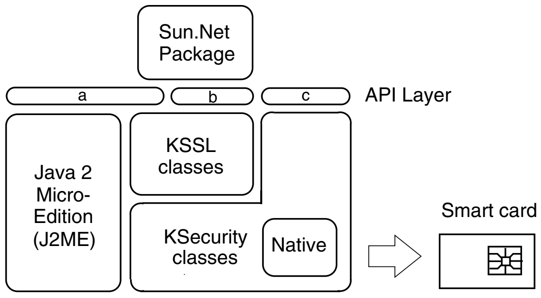


Fig. 4. KSSL Implementation Architecture

III. KSSL AND KSECURITY

"KiloByte" SSL (KSSL), is a small footprint, SSL client for the Mobile Information Device Profile (MIDP) of Java 2 Micro-Edition (J2ME™) [10]. Its overall architecture and relationship to the base J2ME platform is depicted in Fig. 4.

The KSecurity package provides fundamental cryptographic functions such as random number generation, encryption, and hashing that are missing from base J2ME. It reuses the Java Card™ API which opens up the possibility of using a Java Card as a hardware crypto accelerator with minimal changes to the KSSL code. Some of the compute intensive operations (such as modular exponentiation) are implemented as native methods in C.

An SSL client also needs to process X.509 certificates and maintain a list of trusted certificate authorities. Since the Java Card APIs do not deal with Certificates or KeyStores, we modeled these classes as subsets of their Java 2 Standard Edition (J2SE™) counterparts.

The SSL protocol (box labeled KSSL) is written purely in Java™ and implements the client-side of SSLv3.0, the most popular version of SSL. It offers only two cipher suites *RSA_RC4_128_MD5* and *RSA_RC4_40_MD5* since they are fast and almost universally implemented by SSL servers. Client-side authentication is not implemented. KSSL interoperates successfully with SSL servers from iPlanet™, Microsoft, Sun™ and Apache (using OpenSSL).

A. Performance

- *Static Memory Requirements:* For PalmOS, the addition of KSSL and KSecurity classes increases the size of the base J2ME implementation by about 90 KBytes. This additional memory is reasonable compared to the size of base J2ME which is typically a few hundred KBytes. It is possible to reduce the combined size of KSSL and KSecurity packages to as little as 70 KByte if one is willing to sacrifice the clean interface between them (applications will only be able to access security services through SSL).
- *Bulk encryption and authentication* Table I shows the timing results of some cryptographic operations on the Palm.

[3]The server can also force a full-handshake by returning a new session ID.

TABLE I
PERFORMANCE OF KSSL CRYPTOGRAPHIC PRIMITIVES ON PDAS

	PalmVx (20MHz)	Visor (33MHz)
RSA (1024-bit)		
Verify†	1433 ms	806 ms
Sign	80.91 sec	45.11 sec
RSA (768-bit)		
Verify†	886 ms	496 ms
Sign	36.22 sec	20.19 sec
MD5		
1024 bytes	292 Kbits/s	512 Kbits/s
4096 bytes	364 Kbits/s	655 Kbits/s
SHA-1		
1024 bytes	124 Kbits/s	227 Kbits/s
4096 bytes	140 Kbits/s	256 Kbits/s
RC4		
1024 bytes	117 Kbits/s	215 Kbits/s
4096 bytes	190 Kbits/s	351 Kbits/s

†With a public-key exponent of 65537

We found that the bulk encryption and authentication algorithms are adequately fast even on the Palm's CPU. On a 20Mhz chip (found in Palm Vx, Palm IIIc etc), RC4, MD5, and SHA all run at over 100Kbits/s, Since each SSL record requires both a MAC computation and encryption, the effective speed of bulk transfer protected by RC4/MD5 is well over 50Kbits/s, far more than the 9.6Kbits/s bandwidth offered by our Omnisky CDPD network service.

- *SSL Handshake Latency* A typical handshake requires two RSA public-key operations: one for certificate verification and another for pre-master secret encryption. As shown in Table I, our implementation of RSA takes 0.8-1.5 seconds on a 20MHz Palm CPU, depending on the key size (768 bits or 1024 bits). Other factors such as network delays and the time to parse X.509 certificates also impact the handshake latency. In our experiments with Palm MIDP (Mobile Information Device Profile of J2ME [10]), we found that a full handshake can take approximately 10 seconds. In most scenarios, a client communicates with the same SSL server repeatedly over a small duration. In such cases, the handshake overhead can be reduced dramatically. For example, simply caching the server certificate (indexed by an MD5 hash) eliminates the overhead of certificate parsing and verification. This brings down the latency of a subsequent full handshake to around 7-8 seconds. An abbreviated handshake only takes around 2 seconds. Finally, by using persistent HTTP, one make the amortized cost of an SSL handshake arbitrarily close to zero.

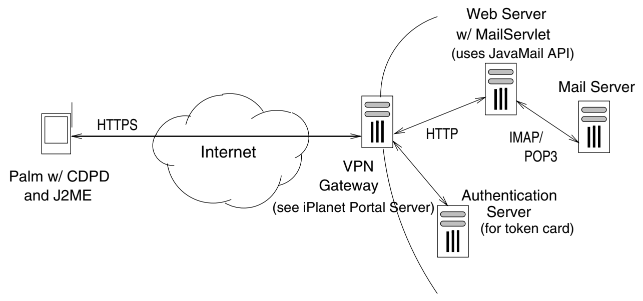


Fig. 5. Secure Email Client Architecture

IV. SECURE MOBILE ENTERPRISE ACCESS

To evaluate usability of the KSecurity and KSSL packages for "real-world" applications, we have developed a J2ME midlet suite (a MIDP application is called a midlet) that enables Sun employees to securely access enterprise services like corporate email, calendar and directory from a PalmVx connected to the Internet via Omnisky's wireless CDPD modem. The application size is about 55 KB and it runs in 64 KB of application heap.

The overall architecture (shown in Fig. 5) reuses Sun's existing SSL-based virtual private network (VPN) for remote employees, which is based on the iPlanet Portal Server. All communication between the VPN gateway and the mobile device is protected end-to-end by SSL. Remote users must authenticate themselves through a challenge-response mechanism (using token cards) over an SSL-secured channel before they can access the Intranet. Authenticated users are allowed to communicate with specially designed servlets running on a web server behind the firewall. These main purpose of these servlets is to format data obtained from mail, calendar or LDAP servers for consumption by the remote application. The servlets are capable of maintaining session-specific state. This allows for features such as "chunking" where a long email message is sent to the user in smaller parts for bandwidth efficiency (this avoids sending any long messages in which the user has no interest).

By making effective use of certificate caching and SSL session reuse, we are able to reduce the response time of each user transaction to about 8 seconds which is comparable to the response time of accessing public web pages in the clear. For now, the persistent HTTP feature has not been turned on at the VPN gateway. Exploiting this feature would reduce the response time to around 5 seconds.

V. TECHNOLOGY TRENDS

Since starting this project about an year ago, we've seen several examples of technology's relentless march towards smaller, faster and more capable devices. Newer Palm PDAs like the PalmVx and PalmIIIc use 20MHz processors and the Handspring Visor Platinum (another PalmOS device) features a 33MHz processor, both considerable improvements over the earlier 16MHz CPUs. All of them offer 8MB of memory. The

Compaq iPaQ pocket PC, in comparison, carries a 200MHz StrongARM processor and 16-32MB of memory. The CPU enhancements have a direct impact on the speed of SSL's cryptographic operations. Significant performance gains are also obtainable by using hardware accelerators in the form of tiny smart cards (and related devices like the iButton). The Schlumberger Cyberflex smart card, for instance, can perform 1024-bit RSA operations (both public- and private-key) in under one second!

Similarly, improvements can also be seen in the speed of wireless networks. Metricom's Ricochet service now offers wireless data speeds of 128Kbits/s in several U.S. cities and 3G networks hold the promise of even faster communication in the next year or two. These improvements help reduce the network-related latency of an SSL handshake.

Even smart compilation techniques, which had so far been available only on more capable PCs and workstations, are now available on small devices and can boost the performance of J2ME applications by as much as a factor of five.

These developments should alleviate any remaining concerns about SSL's suitability for wireless devices. They also highlight an interesting phenomenon - In the time it takes to develop and deploy new (incompatible) protocols, technology constraints can change enough to raise serious questions about their long-term relevance. The following quote captures this sentiment rather well:

"Don't skate to the puck; skate to where it's going."
— *Wayne Gretzsky (Ice Hockey Legend)*

In light of this view, it is reassuring to see the WAP Forum embracing standard IETF and W3C protocols for its next (WAP 2.0) specification.

VI. Conclusions and Future Work

Our experiments show that SSL is a viable technology even for today's mobile devices and wireless networks. By carefully selecting and implementing a subset of the protocol's many features, it is possible to ensure acceptable performance and compatibility with a large installed base of secure web servers while maintaining a small memory footprint.

Our implementation brings mainstream security mechanisms, trusted on the wired Internet, to wireless devices for the first time. The use of standard SSL ensures end-to-end security, an important feature missing from current wireless architectures. An early access version of Sun's MIDP reference implementation incorporating KSSL can be downloaded from [10].

In our ongoing effort to further enhance cryptographic performance on small devices, we plan to explore the use of smart cards as hardware accelerators and Elliptic Curve Cryptography in our implementations.

Acknowledgments

We based portions of our native code on Ian Goldberg's PalmOS port of the crypto algorithms from SSLeay. We would also like to thank Nagendra Modadugu from Stanford University, and Sheueling Chang from Sun Microsystems for their help with optimizing RSA.

References

[1] A. Frier, P. Karlton, and P. Kocher, "The SSL3.0 Protocol Version 3.0", see http://home.netscape.com/eng/ssl3/.

[2] R. M. Needham, and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Communications of the ACM*, vol. 21, no. 12, Dec 1978, pp. 993–999.

[3] R. M. Needham, and M. D. Schroeder, "Authentication Revisited," *Operating System Review*, vol. 21, no. 1, Apr 1990, pp. 35–38.

[4] D. Wagner, and B. Schneier, "Analysis of the SSL 3.0 protocol," *2nd USENIX Workshop on Electronic Commerce*, 1996. Available from http://www.cs.berkeley.edu/~daw/papers/.

[5] WAP Forum, "Wireless Transport Layer Security Specification", see http://www.wapforum.org/what/technical. htm

[6] S. Bradner, "The problems with closed gardens," *Network World*, Jun 12, 2000, at http://www.nwfusion. com/columnists/2000/0612bradner.html

[7] T. Miranzadeh, "Understanding Security on the Wireless Internet", see http://www.tdap.co.uk/uk/archive/billing/bill(phonecom_9912).html

[8] N. Koblitz, *A Course in Number Theory and Cryptography 2nd Edition*, Springer-Verlag.

[9] D. Boneh, and N. Daswani, "Experiments with Electronic Commerce on the PalmPilot," *Financial Cryptography '99*, Lecture Notes in Computer Science, vol. 1648, 1999, pp. 1–16.

[10] Mobile Information Device Profile (MIDP) of Java 2 Micro-Edition (J2ME), see http://java.sun.com/products/midp.

Dr. Vipul Gupta is a Senior Staff Engineer at Sun Microsystems Laboratories with expertise in Internet security protocols and mobile computing. He is an active participant in the Internet Engineering Task Force (IETF) and the WAP Security Group. He has authored/co-authored over thirty technical publications in these and related areas. Prior to joining Sun, he was an Assistant Professor at the State University of New York where he taught courses in computer networking, parallel processing, and operating systems and conducted research funded by the National Science Foundation and industry sponsors that included IBM, NEC and NYSEG.

Sumit Gupta is a Member of Technical Staff at Sun Microsystems Laboratories. He completed his B.Tech(EE) from Indian Institute of Technology, Kanpur, India and His M.S.(EE) from Texas A&M University. At Texas A&M, Sumit worked on Multimedia Networking and Quality of Service Issues. Since joining Sun in 1998, he has worked on various topics related to IP multicast, IP telephony, and wireless. His current interests include wireless IP security and Internet security protocols.