Securing the Wireless Internet

Vipul Gupta and Sumit Gupta, Sun Microsystems

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 on less capable CPUs and low-bandwidth wireless links. This article presents our experiences in implementing and using standard security mechanisms and protocols on small wireless devices. We have created new classes for the Java 2 Micro-Edition platform that offer fundamental cryptographic operations such as message digests and ciphers as well as higher level security protocols like SSL. Our results show that SSL is a practical solution for ensuring end-to-end security of wireless Internet transactions even within today's technological constraints.

INTRODUCTION

In the past few years, there has been explosive growth in the popularity and availability of small handheld devices (mobile phones, PDAs, pagers) that can wirelessly connect to the Internet. These devices are predicted to soon outnumber traditional Internet hosts like PCs and workstations. With their convenient form factor and falling prices, these devices hold the promise of ubiquitous (anytime, anywhere) access to a wide array of interesting services. However, these battery-driven devices are characterized by limited storage (volatile and nonvolatile memory), minimal computational capability, and screen sizes that vary from small to very small. These limitations 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, and check news and weather reports, but also for more security-sensitive applications like online 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 their secure transport through the network becomes an important concern.[1]

On the wired Internet, Secure Sockets Layer (SSL) [1] is the most widely used security protocol.[2] Between its conception at Netscape in the mid-'90s and standardization within the Internet Engineering Task Force (IETF) in the late '90s, the protocol and its implementations have been subjected to careful scrutiny by some of the world's foremost security experts [2]. No wonder, then, that SSL (in the form of HTTPS which is simply HTTP over SSL) is trusted to secure transactions for sensitive applications ranging from Web banking to securities trading to e-commerce. One could easily argue that without SSL, there would be no e-commerce on the Web today. Almost all Web servers on the Internet support some version of SSL.

Unfortunately, none of the popular wide-area wireless data services today offer this protocol on a handheld device. Driven by perceived inadequacies of SSL in a resource-constrained environment, architects of both WAP and Palm.net chose a different (and incompatible) security protocol (e.g., WTLS [3] for WAP) for their mobile clients and inserted a proxy/gateway in their architecture to perform protocol conversions. A WAP gateway, for instance, decrypts encrypted data sent by a WAP phone using WTLS and reencrypts 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" privy to all "secure" communications. 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.

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

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

0163-6804/01/$10.00 © 2001 IEEE

We found it interesting that the architects of WAP and Palm.net made tacit assumptions about the unsuitability of standard Internet protocols (especially SSL) for mobile devices without citing any studies that would warrant such a conclusion [4]. This prompted our experiments in evaluating standard security algorithms and protocols (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 article describes our experiments in greater detail. We begin by reviewing the security architecture of current wireless Internet offerings and analyzing its shortcomings. Next, we provide an overview of SSL, highlighting aspects that make it easier to implement on weak CPUs than might appear at first. We also discuss our implementation of an SSL client, called KSSL, on a Palm PDA and evaluate its performance. Finally, we talk about mobile technology trends relevant to application and protocol developers, and offer our conclusions.

Background

As wireless data services evolve, their architects are faced with two choices that can profoundly impact the future of the wireless Internet. They can adopt (if necessary, adapt) standard Internet protocols, or create an entirely different set of standards applicable only in the wireless world. The former choice would seamlessly extend the Internet to future mobile devices, the latter could severely stunt its expansion.

The Wireless Application Protocol (WAP) Forum subscribed to the "wireless is different" philosophy for its WAP 1.0 specification which defines an entire suite of protocols that parallel standard TCP/IP and web protocols, but are incompatible with them [5]. In contrast, others like the IETF's PILC working group have put forth proposals to reuse existing protocols and standards in ways that accommodate the special characteristics of wireless networks without destroying compatibility.

Proxy-Based Architecture

Due to protocol incompatibilities, a WAP device cannot communicate directly with the large installed base of Internet hosts. Instead, all communication must go through a gateway or proxy that performs protocol (and possibly content) translation. In typical deployments of WAP, this proxy is owned and maintained by a wireless service provider, who preprograms the proxy in all of its customers' phones. This allows the service provider to control what parts of the Internet are accessible to its customers, thereby creating a "walled garden" [6].

These architectural choices raise a number of concerns:

- *Scalability*: Since a proxy must process data packets to and from a large number of mobile devices, it represents a potential performance bottleneck (besides being a single point of failure). The insertion of a proxy also precludes end-to-end flow control. Since the wired side of a proxy has greater bandwidth than its wireless side, the proxy needs to maintain large data buffers for each active data flow.
- *Legal*: A recent French court ruling, prohibiting France Telecom from dictating which gateways its customers could use, is an indication that such an approach may have legal and anti-trust implications [6].
- *Security*: The most glaring drawback of a proxied architecture is the lack of end-to-end security. In the process of decrypting and reencrypting traffic, the proxy gets to see all communication in the clear.[3] This "WAP gap" problem is depicted in Fig. 1.; even though the wired and wireless "hops" are encrypted, the proxy is privy to all information exchanged. Sometimes the situation is even worse, since weak encryption (or none at all) is used on the wireless side, giving mobile users a false sense of security.

This problem is not unique to WAP. The Palm.net security architecture uses a proprietary protocol between the wireless device and the Palm.net proxy (owned and operated by Palm). SSL is used only between the proxy and the eventual destination. Thus, when a Palm VII user accesses an HTTPS URL to establish a secure connection with a Webserver, the connection that is set up isn't truly secure. This is unacceptable to security savvy organizations; for example, Sun's corporate firewall explicitly disallows connection attempts from the Palm.net proxy.

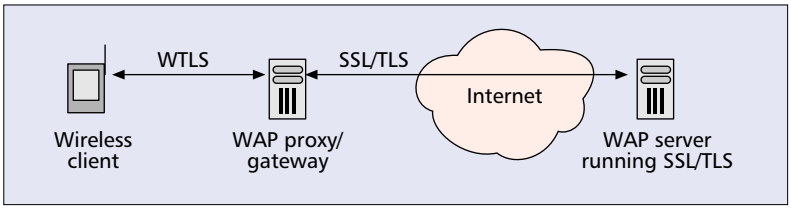

Figure 1. *Proxy-based architecture.*

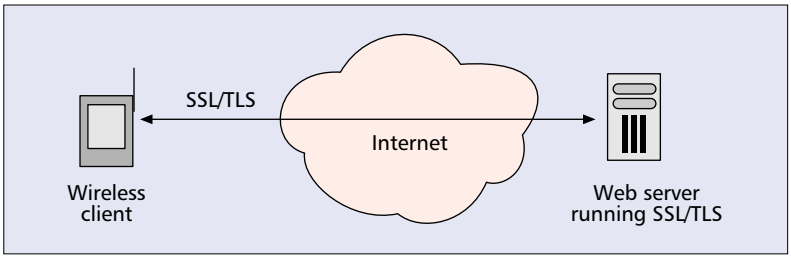

Figure 2. *End-to-end architecture.*

End-to-End Architecture

In contrast, the use of SSL between desktop PCs/workstations and Internet servers offers end-to-end security (Fig. 2). This holds true even when an HTTPS proxy is used to traverse firewalls. Unlike the WAP or Palm.net proxy, an HTTPS proxy does not perform decryption/reencryption of data. Rather, it acts as a simple TCP relay shuttling encrypted bytes from one side to the other without modification.

The expression "old is gold" is especially apt

[3] *This also makes the proxy an extremely attractive target for hackers.*

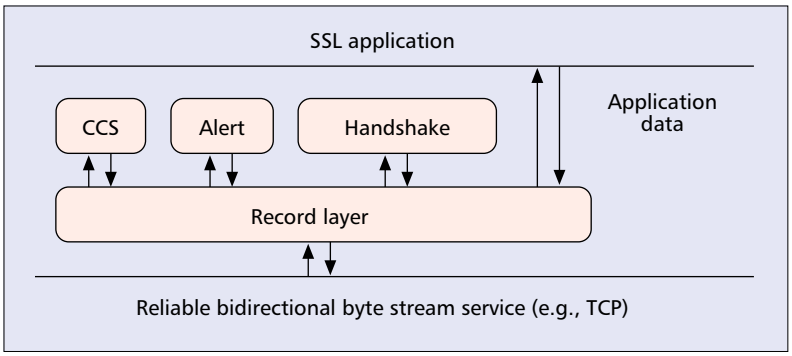

Figure 3. *SSL architecture.*

when considering security protocols. Very often, it takes years of widespread public review and multiple iterations [7] to discover and correct subtle but fatal errors in the design and/or implementation of a security protocol. After more than five years of public scrutiny and deployment experience,[4] SSL is the most widely trusted security protocol for all sorts of web-based transactions. The addition of SSL capabilities to mobile devices would bring the same level of security to the wireless world.

Secure Sockets Layer

Overview

SSL provides encryption, source authentication, and integrity protection of application data over insecure public networks. The protocol requires a reliable bidirectional byte stream service. Typically, this service is provided by TCP, which guarantees that there is no duplication, loss, or reordering of bytes.

As shown in Fig. 3, SSL is a layered protocol. The Record layer sits above the underlying transport and provides bulk encryption and authentication services using symmetric key algorithms. The keys for these algorithms are established by the Handshake protocol, which uses public-key algorithms to create a *master secret* between the SSL client and server. This master secret is further used to derive cipher keys, initialization vectors, and message authentication code (MAC) keys for use by the Record layer. Until these keys are installed, the Record layer acts as a simple bidirectional passthrough for all data. Conceptually, the Alert and Change Cipher Spec (CCS) protocols sit within the same layer as the Handshake protocol. The former is used for notification of any protocol failures. The latter is used to signal successful completion of the handshake, and the start of bulk encryption and authentication in an SSL stream.

SSL is very flexible and can accommodate a variety of algorithms for key agreement (RSA, DH, etc.), encryption (RC4, 3DES etc.), and hashing (MD5, SHA, etc.). 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. 4). In the following subsection, we focus on its most popular form, which uses RSA key exchange and does not involve client-side authentication. The SSL protocol allows both client- and server-side authentication. However, due to the unwieldy problem of maintaining client-side certificates, only the server is typically authenticated. Client authentication, in such cases, happens at the application layer above SSL, for example, through the use of passwords (one-time or otherwise) sent over an SSL-protected channel. The server's *CertificateRequest* message as well as the client's certificate and *CertificateVerify* messages, shown in Fig. 4, are only needed for client-side authentication and rarely encountered in practice.

Full SSL Handshake — The client initiates a new SSL session by sending a *ClientHello* message that contains a random number (used for replay protection), a session ID (set to zero), and a set of supported cipher-suites.

If the server is unwilling to support any of

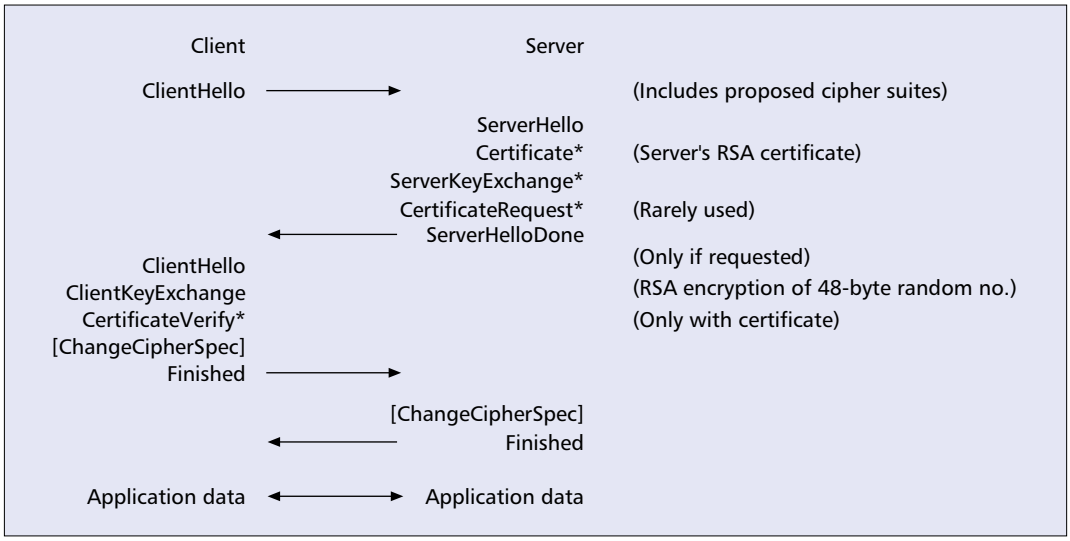

Figure 4. *Full SSL handshake (comments in the third column are specific to RSA key exchange).*

[4] *During this time, SSL has seen several revisions.*

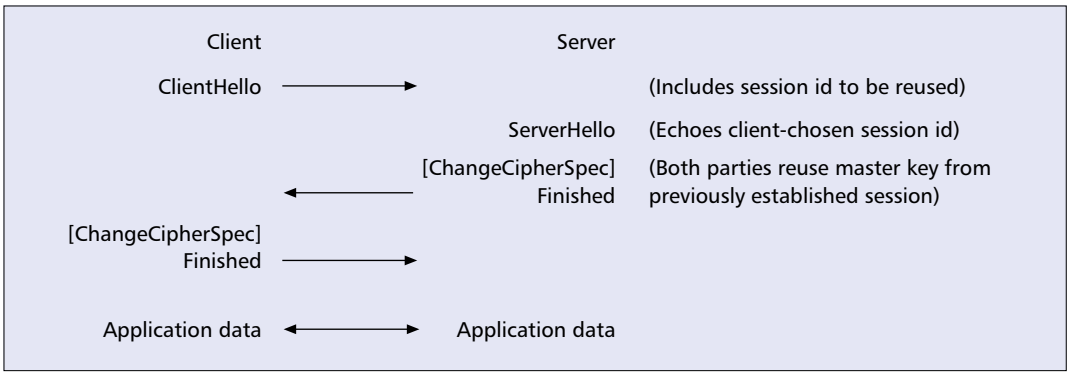

Figure 5. *Abbreviated SSL handshake.*

the proposed cipher suites, it aborts the handshake and issues a failure notification. Otherwise, it generates a random number and a session ID and sends them in the *ServerHello* message along with the selected cipher suite. The *ServerHello* is followed by a *Certificate* message containing the server's RSA public key in an X.509 certificate.[5] If this key is unsuitable for generating the *ClientKeyExchange* message (e.g., if the key is authorized for signing but not for encryption), the server includes another RSA public key in the *ServerKeyExchange* message and signs it with the private key corresponding to its certified public key.

The client verifies the server's public key. It then generates a 48-byte random number, called the *pre-master secret*, and encrypts it with the server's public key. The result is sent to the server in the *ClientKeyExchange* message. The client also computes a *master secret* based on the pre-master secret, and the client and server random numbers exchanged previously. The master key is processed further to derive the symmetric keys used for bulk encryption and authentication. These keys are installed in the record layer and a *ChangeCipherSpec* message is sent to signal the end of in-the-clear communication. The *Finished* message is the first one from the client side to be secured by the negotiated cipher suite. It ensures that any tampering of prior handshake messages, sent in the clear, can be detected.

Upon receiving the *ClientKeyExchange* message, the server decrypts the pre-master secret and follows the same steps as the client to derive the master secret and symmetric keys for the Record Layer. After installing these keys in the Record Layer, the server is able to validate the client's *Finished* message. The server also sends the *ChangeCipherSpec* and *Finished* messages to complete the handshake.

From here on, application data can be exchanged and is protected by the encryption/hashing algorithms negotiated in the handshake. Each direction of the traffic's flow uses distinct encryption and MAC keys.

Abbreviated SSL HandShake — The SSL specification also supports a feature called session reuse, which allows the client and server to reuse the master key derived in a previous session.

The abbreviated handshake protocol is shown in Fig. 5. Here, the *ClientHello* message includes the nonzero ID of a previously negotiated session. If the server still has that session information cached and is willing to reuse the corresponding master secret, it echoes the session ID in the *ServerHello* message. Otherwise, it returns a new session ID, thereby signaling the client to engage in a full handshake. The derivation of symmetric keys from the master secret and the exchange of *ChangeCipherSpec* and *Finished* messages is identical to the full handshake scenario.

The abbreviated handshake does not involve certificates or public key cryptographic operations so fewer (and shorter) messages are exchanged. Consequently, an abbreviated handshake is significantly faster than a full handshake.

The abbreviated handshake does not involve certificates or public-key cryptographic operations so fewer (and shorter) messages are exchanged. Consequently, an abbreviated handshake is significantly faster than a full handshake.

SSL on Small Devices: Common Perception vs Informed Analysis

SSL is commonly perceived as being too heavyweight for comparatively weaker CPUs and 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 protocol, and the large size of existing SSL implementations are all sources of concern and contribute to the perception that SSL is too "big" for small devices.

However, we are not aware of any empirical studies evaluating SSL for small devices and a 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 need only 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

[5] *The Certificate message may contain a chain of two or more certificates linking the server's certificate to a trusted certification authority (CA).*

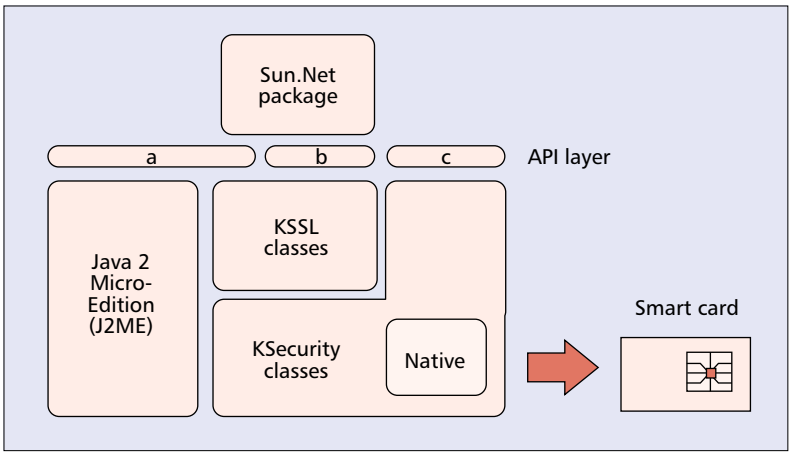

Figure 6. *KSSL implementation architecture.*

times over a short period of time; for example, such interaction is typical of a portal environment. In this scenario, SSL's session reuse feature greatly reduces the need to perform public key RSA operations.

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 setup, this "persistent http" feature further decreases the frequency of SSL handshakes.

KSSL and KSecurity

"KiloByte" SSL (KSSL), is a small footprint, SSL client for the Mobile Information Device Profile (MIDP) of J2ME [10] Its overall architecture and relationship to the base J2ME platform is depicted in Fig. 6.

The KSecurity package provides basic cryptographic functions such as random number generation, encryption, and hashing that are missing from base J2ME. It reuses the JavaCard API, which opens up the possibility of using a JavaCard as a hardware crypto accelerator with minimal changes to the KSSL code. Some of the compute-intensive operations (like modular exponentiation of large integers) 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 JavaCard APIs do not deal with certificates or KeyStores, we modeled these classes as subsets of their Java 2 Standard Edition (J2SE) counterparts. The KSecurity APIs are labeled c in Fig. 6.

The SSL protocol (represented by the box labeled KSSL in Fig. 6) is written purely in Java and its functionality can be accessed through the J2ME Connector API (labeled a in Fig. 6). The Connector API works with URLs (even serial I/O is represented this way), and the addition of KSSL allows us to support HTTPS URLs in addition to HTTP URLs. We also provide another simple, but not yet standardized, API (b) that offers applications greater control over the SSL protocol; for example, the ability to seek user input upon encountering problematic certificates.

Supported Features

The following is a list of features offered by the KSecurity and KSSL packages:

- *Keys*: Symmetric keys of different lengths and RSA public/private keys with modulus lengths up to and including 1024-bits.
- *Ciphers*: RSA (for key exchange) and RC4 (for bulk encryption). DES code is available but no longer included by default.
- *Message digests*: MD5 and SHA.
- *Signatures*: RSA with both MD5 and SHA (as described in PKCS#1).
- *Certificates*: Only X.509 certificates containing RSA keys and signed using RSA with MD5 or SHA are supported. X.509v3 extensions are handled correctly — *subjectAltName*, *basicConstraints*, *keyUsage*, and *extendedKeyUsage* are recognized. Unrecognized extensions that are marked as critical result in an error notification.
- *KeyStore*: Only supports certificate storage (currently, private keys or symmetric keys cannot be stored).
- *SSL*: KSSL is a client-side only implementation of SSLv3.0 Other versions, SSLv2.0 or SSLv3.1 (aka TLS1.0), are not currently supported since they are not used as frequently. The client only offers two cipher suites, *RSA_RC4_128_MD5* and *RSA_RC4_40_MD5*, since they are fast and almost universally implemented by SSL servers from the very early days of the protocol. Client-side authentication is not implemented because it is rarely used and requires (highly CPU-intensive) private-key RSA operations on the client. The server is authenticated via RSA signatures. There are no restrictions on the server's certificate chain length. The client maintains an extensible set of trusted Certification Authorities. The SSL client supports session reuse, works on J2ME running on PalmOS, Solaris and Windows, and inter-operates with SSL servers from iPlanet, Microsoft, Sun and Apache (using OpenSSL).

Performance

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 kilobytes. 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: We found that the bulk encryption and authentication algorithms are adequately fast even on the Palm's CPU. On a 20 MHz chip (found in Palm Vx, Palm IIIc, etc.), RC4, MD5, and SHA all run at over 100 kb/s. Since each SSL record requires both a MAC computation and encryption, the effective speed of bulk transfer protected by RC4/MD5 is around 50 kb/s, far more

than the 9.6 kb/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. Depending on the key size (768 bits or 1024 bits), our implementation of RSA takes 0.5–1.5 seconds on a 20MHz Palm CPU. 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 on a CDPD network, we found that a full handshake can take around 10-13 s. While this is unacceptable for random Web browsing, targeted mobile applications such as those for banking or secure remote access tend to communicate 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 7–8 s. An abbreviated handshake only takes around 2 s. Finally, by using persistent HTTP, one make the amortized cost of an SSL handshake arbitrarily close to zero.

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

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 s. 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 s, almost identical to the response time of accessing public Web pages in the clear.

Technology Trends

Since starting this project about a year ago, we've seen several examples of technology's relentless march toward smaller, faster, and more capable devices. Newer Palm PDAs like the PalmVx and PalmIIIc use 20 MHz processors, and the Handspring Visor Platinum (another PalmOS device) features a 33 MHz processor, both considerable improvements over the earlier 16MHz CPUs. All of them offer 8 mbytes 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 128 kb/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 with the older 32 kb/s Ricochet service, we see 15–20 percent faster handshakes compared to the CDPD network. Powerful handhelds like the iPaQ using 802.11 for wireless connectivity have no problems whatsoever running SSL.

Even smart compilation techniques, which had 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.

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. The latest version of J2ME MIDP 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.

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.

Acknowledgments

We based portions of our native cryptographic code on Ian Goldberg's PalmOS port of 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"; http://home.netscape.com/eng/ssl3

We plan to explore the use of smart cards as hardware accelerators and Elliptic Curve Cryptography in our implementations.

[2] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 Protocol," *2nd USENIX Wksp. Elect. Commerce*, 1996; http://www.cs. berkeley.edu/\verb+~+daw/papers
[3] WAP Forum, "Wireless Transport Layer Security Specification"; http://www.wapforum.org/what/technical. htm
[4] T. Miranzadeh, "Understanding Security on the Wireless Internet," see http://www.tdap.co.uk/uk/archive/billing/bill(phonecom_9912).html
[5] R. Khare, "W* Effect Considered Harmful," *IEEE Internet Computing*, vol. 3, no. 4, pp. 89-92, July/Aug 1999.
[6] S. Bradner, "The Problems with Closed Gardens," *Network World*, Jun 12, 2000, at http://www.nwfusion
[7] R. M. Needham and M. D. Schroeder, "Authentication Revisited," *Operating System Review*, vol. 21, no.1, Apr 1990, pp. 35-38.
[8] N. Koblitz, *A Course in Number Theory and Cryptography 2nd ed.*, 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); http://java.sun.com/products/midp

BIOGRAPHIES

VIPUL GUPTA (vipul.gupta@sun.com) 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 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 (gupta.sumit@sun.com) 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.