

A Study of the Performance of SSL on PDAs

Youngsang Shin
Computer Science Department
Indiana University
Bloomington, Indiana
Email: shiny@cs.indiana.edu

Minaxi Gupta
Computer Science Department
Indiana University
Bloomington, Indiana
Email: minaxi@cs.indiana.edu

Steven Myers
Department of Informatics
Indiana University
Bloomington, Indiana
Email: samyers@indiana.edu

Abstract—PDAs and smartphones are increasingly being used as handheld computers. Today, their network connectivity and their usages for various tasks over the Internet require privacy and authenticity. In this paper, we conduct a comprehensive and comparative study of the performance of the SSL protocol for PDA and laptop clients, both in WEP secured and open Wi-Fi environments. Unlike previous studies [1], [2], the measurements are at sub-protocol granularity allowing for researchers to consider appropriate optimizations for these resource-constrained devices. Unsurprisingly, we find that SSL handshake costs 3 times more at a PDA client than it does for a laptop client, but surprisingly most of the delay comes from network latency and other PDA architecture issues, not cryptographic computation. This suggests that more effort should be spent in minimizing communication rounds in future cryptographic protocols that will be used by PDAs, even at the cost of more cryptographic operations.

I. INTRODUCTION

Small devices such as Personal Data Assistants (PDAs) and smartphones pervade our society today. They typically come equipped with a Wi-Fi or cellular network connection, allowing them to connect to the Internet. Increasingly, they are being used as clients in scenarios that require security, such as e-commerce and e-mailing.

Wired Equivalent Privacy (WEP) [3] is still commonly used to secure many Wi-Fi communications at the data link layer, despite its known vulnerabilities [4]. Further, the Secure Sockets Layer (SSL) protocol [5] forms the basis for the vast majority of secure Web transactions. It provides security at the transport layer. While the performance impact of these protocols, especially SSL, is well studied for commodity client machines, only recently [1], [2], [6] has the community begun studying the performance of these protocols for small handheld devices such as PDAs and smartphones which have constrained computational, energy and software resources.

It is important to study how security protocols perform in constrained environments, so that better, customized options can be designed. We undertake such a study for SSL and WEP on PDAs, comparing their performance to that of laptops. Overall, our paper makes the following contributions:

a) Granularity: We perform a comprehensive and granular study of the performance of SSL protocol on PDAs. We used a high-end PDA, whose processor and running speed compares favorably to modern average smartphones such as the iPhone 3G and BlackBerry Storm, we therefore believe our results represent a majority of such devices.

Studies in [1], [2] analyzed the performance of SSL on PDAs. However, they did not observe SSL's performance in fine granular detail: They focused on measuring the running time of only each public-key cryptographic operation and the whole protocol. Specifically, the SSL protocol has two phases. The *handshake phase* allows the client and the server to securely exchange a key using public-key cryptography. This phase is known to be computationally expensive and contain many network flows. Subsequently, the *data transfer phase* uses the resulting key to exchange data securely and more efficiently. *We separate each phase into detailed steps to specifically distinguish cryptographic and network operations from the other general code.* Our granular analysis enables us to clearly observe how factors other than CPU affect the performance of SSL. CPU clock speed is not the only factor affecting the practical performance of security protocols. Other constrained resources, such as, network access, slower memory and buses in PDAs, are also contributing elements. Furthermore, it is important to understand how CPU bound cryptographic operations run inside a security protocol especially in such a constrained environment. We analyze the proportion of CPU-bound cryptographic operations in the performance of SSL as compared to the network and other general sections of code.

b) Server-Side Measurements: We study SSL performance at both the client and the server. To our knowledge, this paper is the first study to investigate how the performance of SSL on PDA clients can affect that of SSL on a server. An SSL server handles clients without the knowledge of their capabilities. The low performance of SSL on PDAs causes the server to under-serve its potential. Although the server's CPU's time can be preempted for other clients, *the specific number of permitted outstanding handshakes is fixed in modern Web or other server implementations.* Thus, the performance of SSL on PDA clients also impacts that of the server.

c) Session Resumption Measurements: We examine SSL *session resumption*, which permits—in cases where the client re-connects to the server—an abridged cryptographic handshake.

d) PDA vs. Laptop: We compare the performance of the SSL protocol on PDA and laptop clients with commodity hardware and software configurations.

e) WEP & SSL: We study the effect of using WEP concurrently with SSL. Prior to being shown insecure, WEP

was widely accepted as a wireless security protocol. While WEP has shown to be insecure, measurements of its effect on the SSL protocol gives some insight as to what might be expected from WPA2 [7], WEP's accepted replacement. It is important to understand how this combination of security protocols affects, if at all, the performance of SSL over wireless networks. Works in [1], [6] examined the performance of SSL over an open Wi-Fi connection, but did not explore the effect of WEP.

The rest of this paper is organized as follows. Related work is discussed in Section II. Section III provides a background on SSL and WEP protocols. Section IV introduces our experimental environment and experimental methodology. In Section V, we present the performance evaluation. Section VI concludes the paper.

II. RELATED WORK

Works in [8], [9] measured the performance of an SSL Web server by instrumenting and analyzing SSL protocol stacks. They measured the timing of cryptographic operations and the entire protocol on client and server machines over Ethernet. They did not consider PDAs or wireless networks. The authors of [10], [11] measured the performance and battery usage of "naked" cryptographic operations in PDA. They did not investigate how such cryptographic algorithms performed within a security protocol such as SSL. [6] studied battery drainage due to individual cryptographic operations when SSL was run on a PDA, but they did not measure other SSL costs. [1], [2] studied the performance of SSL on a PDA, but did not analyze the detailed costs of SSL. They measured only timing of the handshake, data transfer, and cryptographic operations, while allowing different key sizes and cryptographic algorithms. [2] considered the costs of session resumption as well as full handshake. However, they did not investigate the differences between PDA and laptop clients. Furthermore, none of relevant works measured or compared the cost of WEP against open wireless environments for PDAs.

III. BACKGROUND

A. Secure Sockets Layer (SSL)

The Secure Sockets Layer (SSL) protocol [5] provides security for network traffic at the transport layer and forms the basis of most secure Web transactions on the Internet today. Originally developed by Netscape corporation as a security enhancement for their Web server and browser, it has since been adopted by almost all vendors and developers for their security sensitive client-server applications. SSL has become the de facto standard for transport layer security. The Internet Engineering Task Force (IETF) has standardized SSL as an IETF standard under the name of Transport Layer Security (TLS) protocol [12]. The two protocols are slightly different in the aspects of key expansion and authentication, but for our purposes are essentially the same. We focus on SSL in this paper.

In SSL, to start the handshake, the client sends the server *ClientHello* message (Figure 1) proposing protocol options,

such as supported ciphersuites. The server responds with a *ServerHello* message selecting a set of protocol options. It then sends its public-key certificate in a *ServerKeyExchange* message. The server finalizes its part of the negotiation with a *ServerHelloDone* message. Toward that goal of generating symmetric keys, the client sends a premaster-secret encrypted with the server's public-key in a *ClientKeyExchange* message. Both the client and the server generate a the premaster-secret and exchanged random numbers as seeds. The client then sends a *ChangeCipherSpec* message to activate the negotiated ciphersuite protocols for all further sent messages. Finally, the client sends the *Finished* message encrypted with the master-key and authenticated according to the negotiated option. The server finishes by sending a *ChangeCipherSpec* message similar to that of the client and then sending the encrypted *Finished* message.

The full SSL handshake is expensive, for it requires expensive asymmetric cryptography operations and a large number of messages. This can severely degrade performance in cases where multiple clients try to connect to an SSL server simultaneously. To minimize the overhead due to the asymmetric handshake, the SSL protocol defines a mechanism by which a previous session can be resumed without repeating the asymmetric operations. Termed as *session resumption*([5] and [12]) it starts with a *ClientHello* message just like the full handshake, but a previously established *SessionID* is used instead of new one. This allows for a reuse of the previously negotiated security options and master-key. Upon agreeing to the sessionID, the server responds with a *ServerHello* message. The next three messages of the full handshake are skipped (and thus the asymmetric cryptography). The server directly sends the *ChangeCipherSpec* and *Finished* messages to reactivate the previous security options. The client concludes the handshake by also sending the *ChangeCipherSpec* and *Finished* messages. Session resumption provides a significant performance gain especially when a client opens multiple connections to an SSL server.

B. Wired Equivalent Privacy (WEP)

Wireless networks broadcast messages using radio waves, leaving them susceptible to eavesdropping. The Wired Equivalent Privacy (WEP) is a protocol to secure 802.11 wireless networks. WEP adopts a stream cipher, RC4, for privacy and a CRC-32 checksum for integrity. In a common configuration, 128-bit WEP key is used. WEP's cryptographic protocol is easily broken [4], but its use continues due to the sheer number of deployed devices that support it, compared to subsequent secure standards of Wi-Fi Protected Access (WPA) and WPA2 [7], which are frequently unsupported on older devices. We conducted experiments on WEP due to its pervasive use, despite its flaws.

IV. METHODOLOGY

Experimental Environment: We used a desktop, laptop, PDA, and wireless router to evaluate the performances of SSL and WEP. The desktop was used as a server. The laptop and the

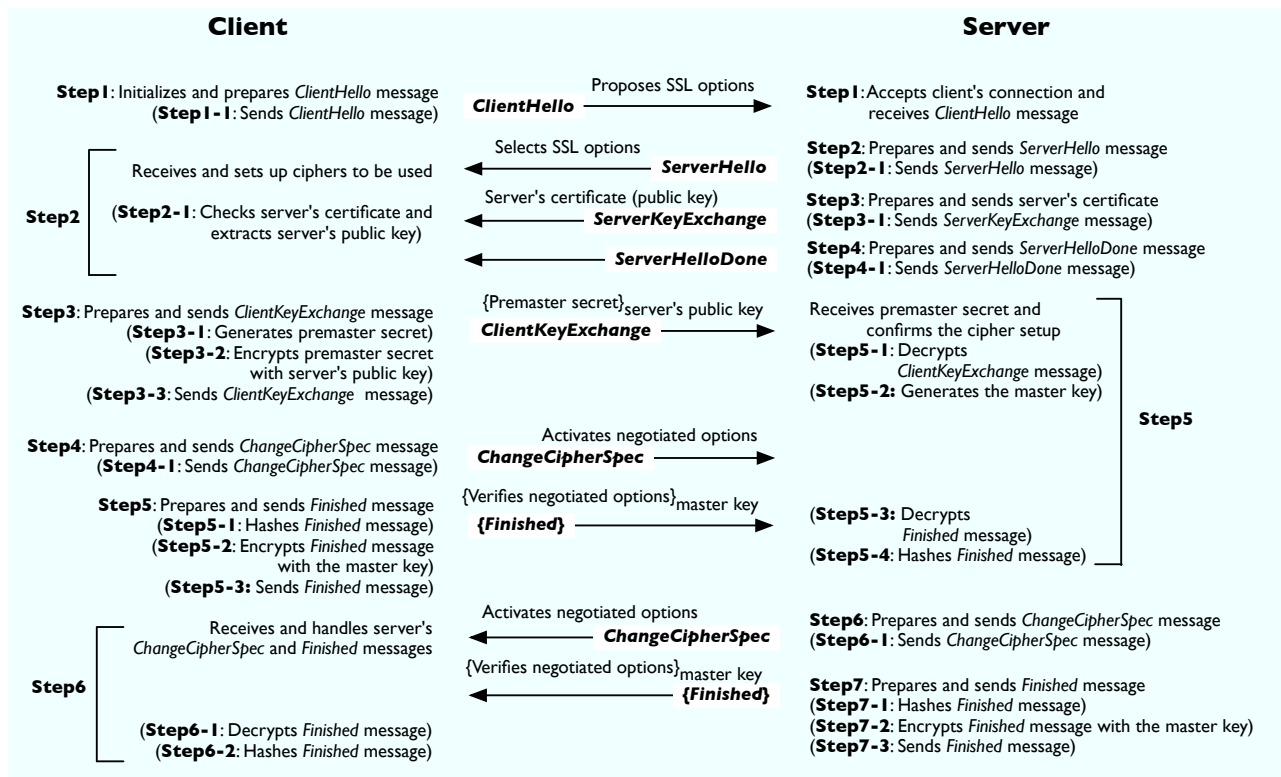


Fig. 1. Cryptography and network messages in SSL handshake

PDA were used as clients. The device configurations are shown in Table I. The PDA used in the experiment contains a high-end commodity CPU for standalone PDAs [13]. However, its calculation speed is representative or faster than most modern average smartphones, such as the iPhone 3G or BlackBerry Storm. We conducted all the experiments over wireless as the PDA only supported wireless communication. To ensure that radio interference did not affect packet-loss, and thus latency, the router was placed at 3 meters from the PDA or laptop.

Device	Model	CPU/RAM/OS	Network Connection
server	Dell GX620	Pentium IV 3.2G/1G Linux kernel 2.6.9	1Gbps Ethernet
laptop	Dell Latitude D610	Pentium M 1.86G/512M Linux kernel 2.6.9	802.11b
PDA	Dell Axim X51v	Intel PXA270 624M/64M Windows Mobile 5	802.11b
router	Netgear WGR614	MSP2007 170M/4M	802.11b/g, 10Mbps

TABLE I
CONFIGURATION OF DEVICES USED

The server and client programs were simple HTTPS servers and clients written in C. We deployed OpenSSL (version 0.9.8a) [14] on the desktop, laptop, and the PDA. To measure timings, we instrumented the OpenSSL library, and server and client programs by putting `rdtsc`¹ and

¹On Intel Pentium processors, `rdtsc` (read time stamp counter) instruction returns the number of clock cycles since the CPU was powered up or reset.

`QueryPerformanceCounter`² into the Linux and Windows systems.

Measurements Performed: We measured the cost of the SSL protocol at both the client and server. While the server stayed fixed, the clients alternated between laptop and PDA in our tests. There are two approaches to compare the performance of an SSL client on a PDA to that on the laptop. The first is to make the environment as similar as possible for both laptop and PDA. One could install identical operating system and identical version of SSL compiled using identical compilers. Indeed, such similar configurations is possible and a previous study [6] that focused on measuring battery drainage in PDAs took that approach. The drawback of this approach is that it cannot effectively represent the tweaked performance of commodity PDAs running commercial operating systems, which are optimized by hardware vendors. Vendors spend a lot of effort optimizing the OS and hardware to compliment each other. While more modern ports of Linux to smartphone hardware (such as Android to the G1) would be so optimized, it is unlikely that the port in [6] was. We left the optimized Windows Mobile on our PDA, but installed the same version of OpenSSL both laptop and PDA. This allowed us to measure SSL performance in a widely used configuration. The PDAs have an automatic CPU speed adjustment mode. This option allows an efficient use of battery, by slowing the CPU down when it is idle, and speeding it up for computationally inten-

²The API function, `QueryPerformanceCounter` retrieves the current value of the high-resolution performance counter in the Windows system.

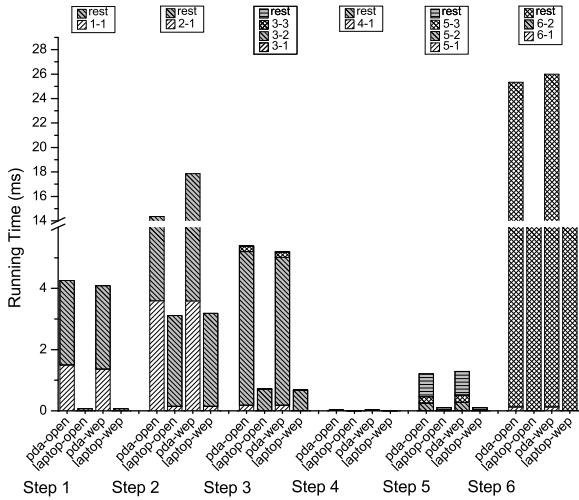


Fig. 2. Client timings for SSL handshake (ms)

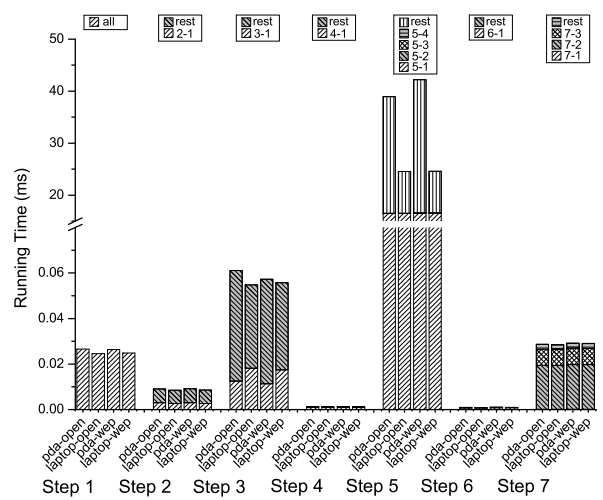


Fig. 3. Server timings for full SSL handshake (ms)

sive tasks. We enabled it on the PDA to further mimic the settings under which PDAs are generally used.

We evaluated the performance of both the handshake and data transfer phases. For the handshake, we investigated the cost of cryptographic operations *as well as the processing of full messages. The latter was largely lacking in other studies dealing with PDAs.* We evaluated both handshakes and the session resumptions. For the bulk data transfer phase measurements, our clients downloaded a pre-specified file from the server. To observe the effect of using the SSL protocol over WEP, we performed all timing measurements both with and without WEP. The latter is often termed *open mode*. The WEP key size in all cases was 128 bits. All data-points represent the mean result of 100 identical runs.

V. EXPERIMENTAL RESULTS

A. The Granular Performance of the SSL Handshake

Figure 1 enumerates and describes the various steps of the SSL handshake. We separate out cryptographic and network operations at each entity into detailed sub-steps. It enables us to clearly distinguish cryptographic and network operations from the other general operations performed by the SSL code.

Figure 2 shows the handshake costs at the client when RSA (1024 bit key) was used for the exchange of premaster-secret, AES (256 bit key) was used for symmetric key encryption, and SHA-1 for hashing. These are the default values in many SSL implementations. First we focus on the performance of the PDA client in open (insecure) mode (PDA-open bars depicted in Figure 2). The PDA client took a total of 50.6ms to finish the handshake. This time includes network latency, which was small since all of the messages are small and the wireless router setup was optimal. Most of this time was spent on steps 2, 3, and 6, each of which contains cryptographic operations. During step 2 the client verifies the server’s SSL certificate and extracts its public-key. Certificate verification requires digital signature verification, an expensive asymmetric cryptography operation. In total, this operation cost almost 7% of the

client’s total handshake cost. In step 3, the PDA client sends a premaster-secret encrypted with the server’s public-key. This also involves asymmetric public-key encryption (Note that with RSA it is common to choose a public-key that allows for faster encryptions, at the expense of longer decryptions). This operation accounted for almost 10% of the total handshake cost. The most expensive client operation in the full handshake occurs in step 6. This step took 46% of the total handshake cost. Only a small component of this cost was due to the symmetric key decryption and hashing, which took a total of 0.129ms. The rest of it, 25.21ms, was caused mainly by a blocking socket read function, `BIO_read`³ in OpenSSL. The function itself took 93% of 25.21ms. Thus, its cost is not directly related to the overhead for SSL protocol itself, but implementation dependent.

The corresponding results for the laptop client in open mode (denoted laptop-open) are alongside those of the PDA-open mode in Figure 2. First, note every step of the laptop took less time than for the PDA. *Overall, the laptop took only 31% of the time the PDA took to finish the handshake.* This is not surprising because the laptop is not resource constrained like the PDA. Unsurprisingly, the steps that cost the most time for PDA are also the ones that took most time for the laptop client. *Further, the cryptographic operations in the laptop client took a total of 5.9% of the total handshake cost while those of the PDA clients took 18.5%.* This is very surprising, as the power management software on the PDA’s CPU reduces its frequency when computational tasks are light, and increases the frequency when they are more burdensome, so one would expect the percentage time to be less on the PDA. When removing the network latency and network packet processing times and comparing the cryptographic operations as a percentage of time, we do see the expected relation, with the PDA taking 72% of time on cryptographic operations vs.

³`BIO_read` function in OpenSSL is responsible for reading from a blocking socket. The blocking problem is identified in OpenSSL version 0.9.8a. We did not pursue the problem in other versions of OpenSSL.

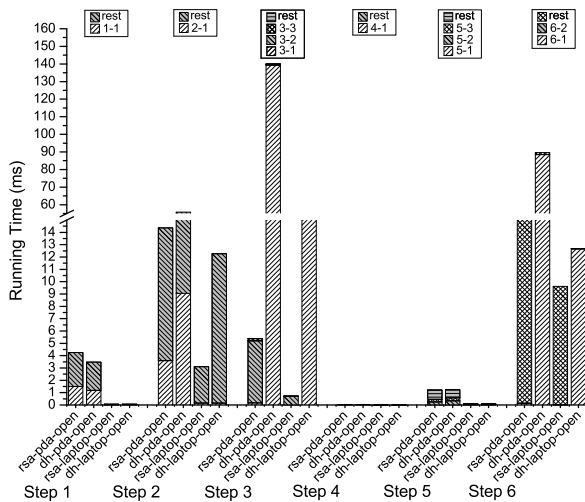


Fig. 4. A comparison of RSA and Diffie-Hellman Key Exchange for PDA and laptop clients (open wireless connection)

90% on the laptop. However, this shows that factors other than the processor speed differentials must be seriously considered when designing protocols for PDAs. We do not have data to motivate why the network and stack latencies are so much higher on the PDA.

Effect of WEP on Handshake: Next, we repeated the above experiments for PDA and laptop clients when WEP was used (respectively pda-wep and laptop-wep), and results are also shown in Figure 2. Overall, using WEP increased the cost of each SSL handshake step by a small amount. Specifically, the PDA client took 8% more time under WEP in comparison to the open mode, whereas the laptops performance remained consistent in both modes. This is unexpected because, while WEP adds another layer of security on top of SSL and slightly reduces bandwidth it should not greatly affect latency. Since the amount of data being transmitted is relatively small there should be little noticeable effect on either device. Finally, one would not expect it to affect the PDA relatively more than it did the laptop. Therefore, we further measured the effect of WEP on transmission on PDAs and laptops, with the results presented later in Section D.

Handshake at the Server: Figure 3 shows the handshake cost at the server when clients connect to it one at a time in both open and WEP modes. As expected, the server handles most of the steps faster than the laptop client. The exception is the decryption of the *ClientKeyExchange* message (step 5), which requires the use of its private key. This is the expected outcome since we used a small and efficiently computable encryption exponents (65,537) at the expense of a large decryption exponent (as is typical usage with RSA public-keys). Further, the server took longer to finish the handshake when the client was a PDA than when it was a laptop. This is because the handshake proceeds in lockstep and the server must wait for the client. Therefore, by providing knowledge of a PDA client to a server it can optimize and enter in to more

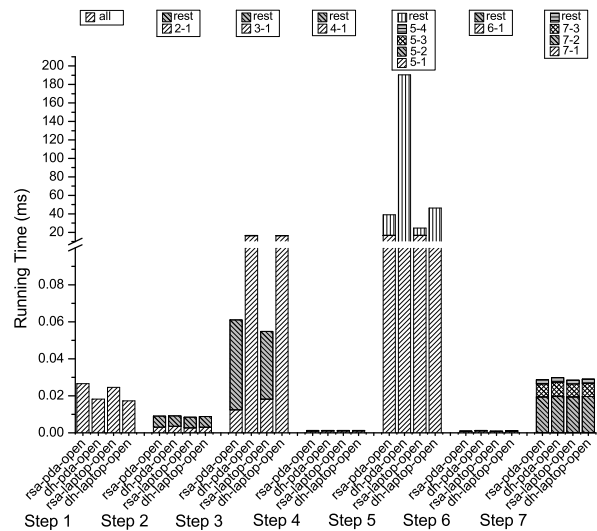


Fig. 5. A comparison of RSA and Diffie-Hellman Key Exchange at the SSL server (open wireless connection)

concurrent handshakes than it might otherwise (most servers restrict the number of active concurrent handshakes they are willing to perform to limit computational loads). A server that services many PDAs should be able to handle a larger number of handshakes due to the implicit slack precipitated by waiting for the PDAs. We note that this could be implemented in SSL through the TLS extension mechanism [15].

B. The Effect of Diffie-Hellman vs. RSA in Handshake

SSL supports multiple key exchange options and multiple cryptographic primitives for encryption and hashing. Since symmetric-key related operations account for a rather small proportion of cost, we focus on handshake key-exchange options. Figures 4 and 5 show a comparison of timings for client and server respectively when Diffie-Hellman (DH) and RSA with 1024-bit moduli are used. With DH, once again, steps 2, 3, and 6 take the longest time at the client, as was the case with RSA. However, each of them take longer than they took for RSA. This is expected because the expensive operation at the core of RSA and DH are exponentiation. As mentioned earlier, in the case of RSA the exponent for encryption can be chosen “intelligently” to speed up calculation of the exponentiation, whereas in DH the exponents must be chosen randomly. The total running time for the full handshake at the PDA and laptop clients is respectively 6 and 3 times longer using DH. The running times at the server follow similar trends because the client and the server have to finish the handshake in tandem. With RSA, we have an opportunity for optimization by adjusting the public exponent, causing the public-key operations in PDAs to be faster than the private key operations in a server. Thus, there is a preference for implementing RSA on servers that will handshake with many PDAs.

C. Handshake under Session Resumption

Session resumption allows a client to reconnect to the server using previously negotiated options. We conducted tests similar to those for full handshake and measured timings for each step at each client as well as the server. We focused on RSA key exchange for this case. Session resumption requires fewer steps and excludes any asymmetric cryptographic operations. As a result, it cost only a fraction of the time in comparison to the full handshake. *Specifically, the session resumption costs were 28% and 14% of the full handshake costs for the PDA and laptop clients respectively.* Given that we have shown that a large part of the latency on the PDA is due to the networking, it is unclear how much of the saving on the PDA result from a reduced number of network flows. PDA and laptop session resumption times were 4.89ms and 4.18ms. Thus, the lack of asymmetric cryptographic operations allows for little performance difference between the PDA and laptop clients.

D. Data Transfer & WEP

The next stage after the SSL handshake is that of data transfer. We measured data transfer times for an SSL client when the client downloads multiple sizes of HTML files from the SSL server. The sizes are 130K, 256K, 512K, 1024K, and 2048K bytes. Per work in [16], the size of 130K bytes is an average transfer size for an HTML session. We examined data transfer times with different sizes of files to investigate the effect of WEP, if any.

In Figure 6 we show the differences in time to transfer different files over WEP vs. an open channel, for both the PDA and laptop clients. While WEP requires some encoding and therefore a reduction in bandwidth (and thus slower transfer times for larger files), we expected the slopes of the plotted lines to be consistent. However, this is clearly not the case and the lines diverge. We conclude that either the PDA is not able to compute the symmetric WEP encryption fast enough to keep the wireless pipe full or the same network bottlenecks that affected the PDA handshake are again apparent here.

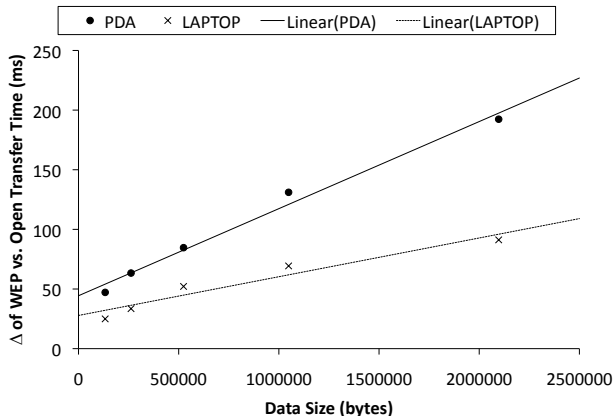


Fig. 6. Linear regression of the **difference** in transfer time between a WEP and open connection for files of different lengths on the laptop vs. PDA clients

VI. CONCLUSION

We make several key conclusions in this paper. First, there is room for optimization of SSL handshake on the server-side by taking client's capabilities into account. Second, as expected, session resumption cuts down the handshake costs substantially. However, the measured *relative* gains for a laptop are more than those experienced by a PDA. The differences between PDA and laptop resources become somewhat less apparent at the data transfer phase which requires less expensive cryptographic operations for the SSL protocol. However, when combined with WEP the computational penalty reasserts itself for larger transfers. We also find that the use of RSA is preferable than Diffie-Hellman in SSL handshakes from the perspective of optimizing costs at the client.

Our results indicate that all aspects of resource constraints on a PDA affect the performance of the SSL protocol, not just the CPU, which plays a large role in cryptographic operations. They point to the need for optimizing security protocols for handheld devices. This is necessary not just to improve experience for clients connecting through the devices but to ensure that the server performance does not degrade as a result of serving a large number of these clients. In the future, when smartphones are expected to be the dominant computational device, this will be essential.

REFERENCES

- [1] P. G. Argyroudis, R. Verma, H. Tewari, and D. O'Mahony, "Performance Analysis of Cryptographic Protocols on Handheld Devices," in *IEEE NCA 2004*, 2004.
- [2] D. Berbecaru, "On Measuring SSL-based Secure Data Transfer with Handheld Devices," in *Int'l Symposium on Wireless Communication Systems*, 2005.
- [3] IEEE Computer Society, "Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," IEEE Standard 802.11, 1999 Edition, 1999.
- [4] N. Borisov, I. Goldberg, and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11," in *ACM Mobicom*, July 2001.
- [5] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL 3.0 Protocol," November 1996.
- [6] N. R. Potlapally, S. Ravi, A. Raghunathan, and N. K. Jha, "A Study of the Energy Consumption Characteristics of Cryptographic Algorithms and Security Protocols," *IEEE Transactions On Mobile Computing*, vol. 5, no. 2, pp. 128 – 143, 2005.
- [7] IEEE Computer Society, "IEEE Standard 802.11i," 2004.
- [8] G. Apostolopoulos, V. Peris, and D. Saha, "Transport Layer Security: How much does it really cost?" in *IEEE INFOCOM*, 1999.
- [9] C. Coarfa, P. Druschel, and D. S. Wallach, "Performance Analysis of TLS Web Servers," in *NDSS*, 2002.
- [10] D. S. Wong, H. H. Fuentes, and A. H. Chan, "The Performance Measurement of Cryptographic Primitives on Palm Devices," in *IEEE ACSAC*, no. 17, 2001.
- [11] C. T. R. Hager, S. F. Midkiff, J.-M. Park, and T. L. Martin, "Performance and Energy Efficiency of Block Ciphers in Personal Digital Assistants," in *IEEE PerCom*, 2005.
- [12] T. Dierks and E. Rescorla, "The Transportation Layer Security (TLS) Protocol, version 1.1," RFC4346, April 2006.
- [13] PDADB.net, "http://www.pdadb.net."
- [14] "OpenSSL Project." [Online]. Available: <http://www.openssl.org>
- [15] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport Layer Security (TLS) Extensions," RFC3546, 2003.
- [16] A. King, "The Average Web Page," <http://www.optimizationweek.com/reviews/average-web-page>, October 2006.