PERFORMANCE EVALUATION OF WTLS HANDSHAKE PROTOCOL USING RSA AND ELLIPTIC CURVE CRYPTOSYSTEMS

by

BURAK BAYOGLU

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabanci University
Spring 2004

PERFORMANCE EVALUATION OF WTLS HANDSHAKE PROTOCOL USING RSA AND ELLIPTIC CURVE CRYPTOSYSTEMS

APPROVED BY:

Asst. Prof. Albert Levi ………………………….

(Thesis Supervisor)

Asst. Prof. Erkay Savas ………………………….

Asst. Prof. Özgür Gürbüz ………………………….

DATE OF APPROVAL: ………………………….

ABSTRACT

WTLS (Wireless Transport Layer Security) is the security protocol designed for WAP (Wireless Application Protocol) protocol stack. Negotiation of the security parameters and authentication of the peers require using public key cryptosystems. Public key operations are generally slow. Thus, use of these cryptosystems in resource constrained handheld devices becomes a significant problem. Server (WAP Gateway) waiting time and handshake data transmission time may also be bottlenecks that occur during the WTLS handshake.

In this study, WTLS Handshake Protocol is implemented using C++ and performance measurements are done using Nokia 7650 as client and open source Kannel gateway as the WAP Gateway. GSM CSD (Global System for Mobile Communication - Circuit Switched Data) data bearer with 9600 bps data rate has been used during the tests. Networking time has also been measured using GPRS bearer. Mutual authenticated and Server Authenticated WTLS full handshake performance with RSA (Rivest-Shamir-Adleman) and ECDH_ECDSA (Elliptic Curve Diffie-Hellman Elliptic Curve Digital Signature Algorithm) key exchange suites has been compared for three different categories. Each category contains four groups: three of these groups use certificates with ECC (Elliptic Curve Cryptography) curve parameters and the fourth group uses RSA certificates. All of the groups in each category are assumed to provide the same level of security. Three groups of ECC certificates are composed of prime, Koblitz and random curve parameters.

Client and server processing times have been measured for each handshake message of the test cases. These values have been used to analyze the processing load of the corresponding key exchange suite, overall handshake time and server queue delay.

Server has been modeled as an M/G/1 queue and the average waiting time in the server queue has been modeled based on the well-known Pollaczek-Khincin (P-K) formula. Queue delay model has been implemented in Matlab 6.0 and queue delay characteristics of the considered test cases have been analyzed using the measured server processing times.

Data transmission time model includes two components. The first component is the amount of time necessary to transmit the measured size of data with specified channel transmission rate. The second component is the traversal delay of the network that is added to the data transmission time regardless of how much data is sent.

Simulation results show that ECC has better processing time performance than RSA. Server queue delay does not seem to be bottleneck for mutual authenticated WTLS handshake using ECC certificates with prime curve parameters. Server authenticated WTLS handshake using any of the three ECC certificate types also has a good queue delay characteristic. However, there exists a practical upper limit of handshake requests per second for other key exchange suites. Traversal delay of the network is much more effective on the overall handshake time when using GSM CSD or GPRS bearer.

ÖZET

WTLS (Kablosuz Tasima Katmani Güvenligi – Wireless Transport Layer Security), WAP(Kablosuz Uygulama Protokolü – Wireless Application Protocol) protokol yigini için tasarlanmis güvenlik protokolüdür. Güvenlik parametreleri üzerinde anlasilabilmesi ve kimlik dogrulamasinin yapilabilmesi için açik anahtar kripto sistemlerinin kullanilmasi gerekmektedir. Açik anahtar islemleri genel olarak yavastir ve bu islemlerin kisitli kaynaklari olan mobil el cihazlarinda yürütülmesi daha büyük bir problem olarak karsimiza çikmaktadir. Sunucu (WAP Aggeçidi) bekleme süresi ve el sikisma verisinin gönderilmesi için gerekli süre de WTLS el sikisma protokolü için bir darbogaz olusturabilir.

Bu çalismada WTLS El Sikisma Protokolü, C++ programlama dili ile gerçeklenmistir. Istemci olarak Nokia 7650 cep telefonu, WAP Aggeçidi olarak da açik kaynak kodlu Kannel kullanilmistir. Testler sirasinda, 9600 bps iletim hizina sahip GSM CSD tasiyicisi kullanilmistir. Veri iletim süresi ayrica GPRS tasiyici için de ölçülmüstür. Karsilikli-Dogrulanmis ve Sunucu-Dogrulanmis WTLS tam el sikisma performansi, RSA ve ECDH_ECDSA anahtar degisim takimlari için üç kategori altinda karsilastirilmistir. Her kategoride dört grup bulunmaktadir. Gruplardan üç tanesi, ECC egri parametrelerine sahip sertifikalar, dördüncü grup ise RSA sertifikalarindan olusmaktadir. Bir kategori içindeki tüm gruplarin esit seviyede güvenlik sagladigi kabul edilmektedir. Ele alinan üç ECC grubu, asal, Koblitz ve rasgele egri parametrelerine sahiptir.

Tüm test durumlari için istemci ve sunucu islem süreleri, her bir el sikisma mesaji için ölçülmüstür. Ölçülen degerler, ele alinan anahtar degistirme takimi için islem süresini, el sikisma süresini ve sunucu kuyruk bekleme süresini degerlendirmek amaciyla kullanilmistir.

Sunucu, M/G/1 kuyruk yapisinda varsayilmistir ve sunucu kuyrugunda ortalama bekleme süresi Pollaczek-Khincin (P-K) formülüne dayali olarak modellenmistir. Kuyruk bekleme süresi için ortaya koyulan model, Matlab 6.0 ortaminda gerçeklenmistir ve ölçülen sunucu islem süreleri kullanilarak test durumlarinin kuyrukta bekleme karakteristikleri analiz edilmistir.

Veri iletim süresi modeli, testler sirasinda tespit edilen veri boyutunun mevcut veri iletim hiziyla gönderilmesi için gerekli sürenin yaninda veri boyutundan bagimsiz olarak iletim süresine eklenen agdan geçis süresini de göz önünde bulundurmaktadir.

Simülasyon sonuçlari, ECC islem süresi performansinin RSA'dan daha iyi oldugunu göstermistir. Sunucu kuyrugunda bekleme süresinin asal egriler için ihmal edilebilecek mertebede olmasina karsin diger alternatiflerde saniyedeki el sikisma istegi sayisinin pratik bir üst sinirinin oldugu görülmüstür. GSM CSD veya GPRS tasiyicisi kullanilmasi durumunda veri iletim süresi, iletim hizindan daha çok agdan geçis süresi tarafindan belirlenmektedir.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Asst. Prof. Albert Levi for the suggestions, ideas, and advices during the development of this thesis. Especially, I thank him for his consistent guidance and encouragement throughout my graduate experience. I would also like to thank Asst. Prof. Erkay Savas for his advices, focus, and discussions to help implementing crypto primitives for both Symbian and Win32 environments. Special thanks to Asst. Prof. Özgür Gürbüz, and Asst. Prof. Özgür Erçetin for their consultation on the queue delay performance model.

I am indebted to Sabanci University for tuition waiver and Faculty of Engineering and Natural Sciences members for valuable courses during my graduate education. Special thanks to TÜBITAK UEKAE for letting me spend working hours on my graduate study.

Finally, I would like to thank my family and all of my friends who still remember me although not seeing each other for more than a year during the hard work of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

L Total data length (in bits) including all the handshake messages and their corresponding ACK packets

$p_{c,a}$ Ratio of the key exchange suite with the smallest public key size in a group.

$p_{c,b}$ Ratio of the key exchange suite with the intermediate public key size in a group.

$p_{c,c}$ Ratio of the key exchange suite with the highest public key size in a group.

R Channel transmission rate (bit/s)

T_H Overall handshake duration

$T_{M\_C\_CH}$ Client processing time for generating the Client.Hello message

$T_{M\_C\_SH}$ Client processing time for processing Server.Hello message

$T_{M\_C\_SCERT}$ Client processing time for processing the server certificate

$T_{M\_C\_CERTREQ}$ Client processing time for processing the CertificateRequest message

$T_{M\_C\_SHD}$ Client processing time for processing the ServerHelloDone message

$T_{M\_C\_CCERT}$ Client processing time for generating the Client.Certificate message

$T_{M\_C\_CKX}$ Client processing time for generating the ClientKeyExchange message (message is sent iff RSA key exchange suite is used)

$T_{M\_C\_CERTVRFY}$ Client processing time for generating the CertificateVerify message (applicable if RSA key exchange suite is used)

$T_{M\_C\_CCCS}$ Client processing time for generating the Client ChangeCipherSpec message

$T_{M\_C\_CFIN}$ Client processing time for generating the Client.Finished message

$T_{M\_C\_SCCS}$ Client processing time for processing the Server ChangeCipherSpec message

$T_{M\_C\_SFIN}$ Client processing time for processing the Server.Finished message

$T_{M\_C\_ECDH}$ Client processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)

$T_{M\_C\_MS}$ Client processing time for computing the master secret from the premaster secret

$T_{M\_C\_RSAENC}$ Client processing time for encryption operation using the Server’s public key (applicable if RSA key exchange suite is used)

$T_{M\_S\_CH}$ Server processing time for processing the Client.Hello message

$T_{M\_S\_SH}$ Server processing time for generating the Server.Hello message

$T_{M\_S\_SCERT}$ Server processing time for generating the Server.Certificate message

$T_{M\_S\_CERTREQ}$ Server processing time for generating the CertificateRequest message

$T_{M\_S\_SHD}$ Server processing time for generating the ServerHelloDone message

$T_{M\_S\_CCERT}$ Server processing time for processing the client certificate

$T_{M\_S\_CKX}$ Server processing time for processing the ClientKeyExchange message

$T_{M\_S\_CERTVRFY}$ Server processing time for processing the CertificateVerify message (applicable if RSA key exchange suite is used)

$T_{M\_S\_CCCS}$ Server processing time for processing the Client ChangeCipherSpec message

$T_{M\_S\_CFIN}$ Server processing time for processing the Client.Finished message

$T_{M\_S\_SCCS}$ Server processing time for generating the Server ChangeCipherSpec message

$T_{M\_S\_SFIN}$ Server processing time for generating the Server.Finished message

$T_{M\_S\_ECDH}$ Server processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)

$T_{M\_S\_MS}$ Server processing time for computing the master secret from the premaster secret

$T_{M\_S\_RSADEC}$ Server processing time for decryption operation using its own private key (applicable if RSA key exchange suite is used)

T_{PD} Total processing delay of handshake messages

T_{PD_C} Overall processing time for the client side

T_{PD_S} Overall processing time for the server side

T_{QD} Server queue delay

$T_{S\_C\_CH}$ Average client processing time for generating the Client.Hello message

$T_{S\_C\_SH}$ Average client processing time for processing Server.Hello message

$T_{S\_C\_SCERT}$ Average client processing time for processing the server certificate

$T_{S\_C\_CERTREQ}$ Average client processing time for processing the CertificateRequest message

$T_{S\_C\_SHD}$ Average client processing time for processing the ServerHelloDone message

$T_{S\_C\_CCERT}$ Average client processing time for generating the Client.Certificate message

$T_{S\_C\_CKX}$ Average client processing time for generating the ClientKeyExchange message

$T_{S\_C\_CCCS}$ Average client processing time for generating the Client ChangeCipherSpec message

$T_{S\_C\_CFIN}$ Average client processing time for generating the Client.Finished message

$T_{S\_C\_SCCS}$ Average client processing time for processing the Server ChangeCipherSpec message

$T_{S\_C\_SFIN}$ Average client processing time for processing the Server.Finished message

$T_{S\_C\_ECDH}$ Average client processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)

$T_{S\_C\_MS}$ Average client processing time for computing the master secret from the premaster secret

$T_{S\_C\_RSAENC}$ Average client processing time for encryption operation using the Server’s public key (applicable if RSA key exchange suite is used)

$T_{S\_S\_CH}$ Average server processing time for processing the Client.Hello message

$T_{S\_S\_SH}$ Average server processing time for generating the Server.Hello message

$T_{S\_S\_SCERT}$ Average server processing time for generating the Server.Certificate message

$T_{S\_S\_CERTREQ}$ Average server processing time for generating the CertificateRequest message

$T_{S\_S\_SHD}$ Average server processing time for generating the ServerHelloDone message

$T_{S\_S\_CCERT}$ Average server processing time for processing the client certificate

$T_{S\_S\_CKX}$ Average server processing time for processing the ClientKeyExchange message

$T_{S\_S\_CCCS}$ Average server processing time for processing the Client ChangeCipherSpec message

$T_{S\_S\_CFIN}$ Average server processing time for processing the Client.Finished message

$T_{S\_S\_SCCS}$ Average server processing time for generating the Server ChangeCipherSpec message

$T_{S\_S\_SFIN}$ Average server processing time for generating the Server.Finished message

$T_{S\_S\_ECDH}$ Average server processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)

$T_{S\_S\_MS}$ Average server processing time for computing the master secret from the premaster secret

$T_{S\_S\_RSADEC}$ Average server processing time for decryption operation using its own private key (applicable if RSA key exchange suite is used)

T_{TD} Data transmission time (s)

$T_{traversal}$ One way traversal delay of the GSM network specific to the data bearer and GSM service provider

T_X Average service time for any of the applicable handshake messages for a given group

$T_{X,c,a}$ Average service time for any of the applicable handshake messages for the key exchange suite with the smallest public key size in a group.

$T_{X,c,b}$ Average service time for any of the applicable handshake messages for the key exchange suite with the intermediate public key size in a group.

$T_{X,c,c}$ Average service time for any of the applicable handshake messages for the key exchange suite with the highest public key size in a group.

LIST OF ABBREVIATIONS

BSC	Base Station Controller
BTS	Base Transceiver Station
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDH_ECDSA	Elliptic Curve Diffie-Hellman Elliptic Curve Digital Signature Algorithm
AMPS	Advanced Mobile Phone Service
CA	Certification Authority
CDMA	Code Division Multiple Access
CDPD	Cellular Digital Packet Data
CSD	Circuit Switched Data
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDEN	Integrated Digital Enhanced Network
IPv4	Internet Protocol version 4
ISO/OSI	International Organization for Standardization/Open System Interconnection
ITSI	Individual TETRA Subscriber Identity
LAN	Local Area Network
MAN	Metropolitan Area Network
MSC	Mobile Switching Center
MSISDN	Mobile Station International ISDN Number
OMA	Open Mobile Alliance
OS	Operating System
PC	Personal Computer
PDC	Personal Digital Communications
PRF	Pseudo Random Function
QoS	Quality of Service
RAS	Remote Access Server

RSA	Rivest Shamir Adleman
RTT	Round Trip Time
SHA	Secure Hash Algorithm
SMS	Short Message Service
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
USSD	Unstructured Supplementary Services Data
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WDP	Wireless Datagram Protocol
WML	Wireless Markup Language
WSP	Wireless Session Protocol
WTA	Wireless Telephony Application
WTLS	Wireless Transport Layer Security
WTP	Wireless Transaction Protocol
WWW	World Wide Web
XML	Extensible Markup Language

PERFORMANCE EVALUATION OF WTLS HANDSHAKE PROTOCOL USING RSA AND ELLIPTIC CURVE CRYPTOSYSTEMS

by

BURAK BAYOGLU

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabanci University
Spring 2004

PERFORMANCE EVALUATION OF WTLS HANDSHAKE PROTOCOL USING RSA AND ELLIPTIC CURVE CRYPTOSYSTEMS

APPROVED BY:

Asst. Prof. Albert Levi …………………………

(Thesis Supervisor)

Asst. Prof. Erkay Savas …………………………

Asst. Prof. Özgür Gürbüz …………………………

DATE OF APPROVAL: …………………………

ABSTRACT

WTLS (Wireless Transport Layer Security) is the security protocol designed for WAP (Wireless Application Protocol) protocol stack. Negotiation of the security parameters and authentication of the peers require using public key cryptosystems. Public key operations are generally slow. Thus, use of these cryptosystems in resource constrained handheld devices becomes a significant problem. Server (WAP Gateway) waiting time and handshake data transmission time may also be bottlenecks that occur during the WTLS handshake.

In this study, WTLS Handshake Protocol is implemented using C++ and performance measurements are done using Nokia 7650 as client and open source Kannel gateway as the WAP Gateway. GSM CSD (Global System for Mobile Communication - Circuit Switched Data) data bearer with 9600 bps data rate has been used during the tests. Networking time has also been measured using GPRS bearer. Mutual authenticated and Server Authenticated WTLS full handshake performance with RSA (Rivest-Shamir-Adleman) and ECDH_ECDSA (Elliptic Curve Diffie-Hellman Elliptic Curve Digital Signature Algorithm) key exchange suites has been compared for three different categories. Each category contains four groups: three of these groups use certificates with ECC (Elliptic Curve Cryptography) curve parameters and the fourth group uses RSA certificates. All of the groups in each category are assumed to provide the same level of security. Three groups of ECC certificates are composed of prime, Koblitz and random curve parameters.

Client and server processing times have been measured for each handshake message of the test cases. These values have been used to analyze the processing load of the corresponding key exchange suite, overall handshake time and server queue delay.

Server has been modeled as an M/G/1 queue and the average waiting time in the server queue has been modeled based on the well-known Pollaczek-Khincin (P-K) formula. Queue delay model has been implemented in Matlab 6.0 and queue delay characteristics of the considered test cases have been analyzed using the measured server processing times.

Data transmission time model includes two components. The first component is the amount of time necessary to transmit the measured size of data with specified channel transmission rate. The second component is the traversal delay of the network that is added to the data transmission time regardless of how much data is sent.

Simulation results show that ECC has better processing time performance than RSA. Server queue delay does not seem to be bottleneck for mutual authenticated WTLS handshake using ECC certificates with prime curve parameters. Server authenticated WTLS handshake using any of the three ECC certificate types also has a good queue delay characteristic. However, there exists a practical upper limit of handshake requests per second for other key exchange suites. Traversal delay of the network is much more effective on the overall handshake time when using GSM CSD or GPRS bearer.

ÖZET

WTLS (Kablosuz Tasima Katmani Güvenligi – Wireless Transport Layer Security), WAP(Kablosuz Uygulama Protokolü – Wireless Application Protocol) protokol yigini için tasarlanmis güvenlik protokolüdür. Güvenlik parametreleri üzerinde anlasilabilmesi ve kimlik dogrulamasinin yapilabilmesi için açik anahtar kripto sistemlerinin kullanilmasi gerekmektedir. Açik anahtar islemleri genel olarak yavastir ve bu islemlerin kisitli kaynaklari olan mobil el cihazlarinda yürütülmesi daha büyük bir problem olarak karsimiza çikmaktadir. Sunucu (WAP Aggeçidi) bekleme süresi ve el sikisma verisinin gönderilmesi için gerekli süre de WTLS el sikisma protokolü için bir darbogaz olusturabilir.

Bu çalismada WTLS El Sikisma Protokolü, C++ programlama dili ile gerçeklenmistir. Istemci olarak Nokia 7650 cep telefonu, WAP Aggeçidi olarak da açik kaynak kodlu Kannel kullanilmistir. Testler sirasinda, 9600 bps iletim hizina sahip GSM CSD tasiyicisi kullanilmistir. Veri iletim süresi ayrica GPRS tasiyici için de ölçülmüstür. Karsilikli-Dogrulanmis ve Sunucu-Dogrulanmis WTLS tam el sikisma performansi, RSA ve ECDH_ECDSA anahtar degisim takimlari için üç kategori altinda karsilastirilmistir. Her kategoride dört grup bulunmaktadir. Gruplardan üç tanesi, ECC egri parametrelerine sahip sertifikalar, dördüncü grup ise RSA sertifikalarindan olusmaktadir. Bir kategori içindeki tüm gruplarin esit seviyede güvenlik sagladigi kabul edilmektedir. Ele alinan üç ECC grubu, asal, Koblitz ve rasgele egri parametrelerine sahiptir.

Tüm test durumlari için istemci ve sunucu islem süreleri, her bir el sikisma mesaji için ölçülmüstür. Ölçülen degerler, ele alinan anahtar degistirme takimi için islem süresini, el sikisma süresini ve sunucu kuyruk bekleme süresini degerlendirmek amaciyla kullanilmistir.

Sunucu, M/G/1 kuyruk yapisinda varsayilmistir ve sunucu kuyrugunda ortalama bekleme süresi Pollaczek-Khincin (P-K) formülüne dayali olarak modellenmistir. Kuyruk bekleme süresi için ortaya koyulan model, Matlab 6.0 ortaminda gerçeklenmistir ve ölçülen sunucu islem süreleri kullanilarak test durumlarinin kuyrukta bekleme karakteristikleri analiz edilmistir.

Veri iletim süresi modeli, testler sirasinda tespit edilen veri boyutunun mevcut veri iletim hiziyla gönderilmesi için gerekli sürenin yaninda veri boyutundan bagimsiz olarak iletim süresine eklenen agdan geçis süresini de göz önünde bulundurmaktadir.

Simülasyon sonuçlari, ECC islem süresi performansinin RSA'dan daha iyi oldugunu göstermistir. Sunucu kuyrugunda bekleme süresinin asal egriler için ihmal edilebilecek mertebede olmasina karsin diger alternatiflerde saniyedeki el sikisma istegi sayisinin pratik bir üst sinirinin oldugu görülmüstür. GSM CSD veya GPRS tasiyicisi kullanilmasi durumunda veri iletim süresi, iletim hizindan daha çok agdan geçis süresi tarafindan belirlenmektedir.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Asst. Prof. Albert Levi for the suggestions, ideas, and advices during the development of this thesis. Especially, I thank him for his consistent guidance and encouragement throughout my graduate experience. I would also like to thank Asst. Prof. Erkay Savas for his advices, focus, and discussions to help implementing crypto primitives for both Symbian and Win32 environments. Special thanks to Asst. Prof. Özgür Gürbüz, and Asst. Prof. Özgür Erçetin for their consultation on the queue delay performance model.

I am indebted to Sabanci University for tuition waiver and Faculty of Engineering and Natural Sciences members for valuable courses during my graduate education. Special thanks to TÜBITAK UEKAE for letting me spend working hours on my graduate study.

Finally, I would like to thank my family and all of my friends who still remember me although not seeing each other for more than a year during the hard work of this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

L	Total data length (in bits) including all the handshake messages and their corresponding ACK packets
$p_{c,a}$	Ratio of the key exchange suite with the smallest public key size in a group.
$p_{c,b}$	Ratio of the key exchange suite with the intermediate public key size in a group.
$p_{c,c}$	Ratio of the key exchange suite with the highest public key size in a group.
R	Channel transmission rate (bit/s)
T_H	Overall handshake duration
$T_{M\_C\_CH}$	Client processing time for generating the Client.Hello message
$T_{M\_C\_SH}$	Client processing time for processing Server.Hello message
$T_{M\_C\_SCERT}$	Client processing time for processing the server certificate
$T_{M\_C\_CERTREQ}$	Client processing time for processing the CertificateRequest message
$T_{M\_C\_SHD}$	Client processing time for processing the ServerHelloDone message
$T_{M\_C\_CCERT}$	Client processing time for generating the Client.Certificate message
$T_{M\_C\_CKX}$	Client processing time for generating the ClientKeyExchange message (message is sent iff RSA key exchange suite is used)
$T_{M\_C\_CERTVRFY}$	Client processing time for generating the CertificateVerify message (applicable if RSA key exchange suite is used)
$T_{M\_C\_CCCS}$	Client processing time for generating the Client ChangeCipherSpec message
$T_{M\_C\_CFIN}$	Client processing time for generating the Client.Finished message
$T_{M\_C\_SCCS}$	Client processing time for processing the Server ChangeCipherSpec message
$T_{M\_C\_SFIN}$	Client processing time for processing the Server.Finished message
$T_{M\_C\_ECDH}$	Client processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)
$T_{M\_C\_MS}$	Client processing time for computing the master secret from the

premaster secret

$T_{M\_C\_RSAENC}$ Client processing time for encryption operation using the Server's public key (applicable if RSA key exchange suite is used)

$T_{M\_S\_CH}$ Server processing time for processing the Client.Hello message

$T_{M\_S\_SH}$ Server processing time for generating the Server.Hello message

$T_{M\_S\_SCERT}$ Server processing time for generating the Server.Certificate message

$T_{M\_S\_CERTREQ}$ Server processing time for generating the CertificateRequest message

$T_{M\_S\_SHD}$ Server processing time for generating the ServerHelloDone message

$T_{M\_S\_CCERT}$ Server processing time for processing the client certificate

$T_{M\_S\_CKX}$ Server processing time for processing the ClientKeyExchange message

$T_{M\_S\_CERTVRFY}$ Server processing time for processing the CertificateVerify message (applicable if RSA key exchange suite is used)

$T_{M\_S\_CCCS}$ Server processing time for processing the Client ChangeCipherSpec message

$T_{M\_S\_CFIN}$ Server processing time for processing the Client.Finished message

$T_{M\_S\_SCCS}$ Server processing time for generating the Server ChangeCipherSpec message

$T_{M\_S\_SFIN}$ Server processing time for generating the Server.Finished message

$T_{M\_S\_ECDH}$ Server processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)

$T_{M\_S\_MS}$ Server processing time for computing the master secret from the premaster secret

$T_{M\_S\_RSADEC}$ Server processing time for decryption operation using its own private key (applicable if RSA key exchange suite is used)

T_{PD} Total processing delay of handshake messages

T_{PD_C} Overall processing time for the client side

T_{PD_S} Overall processing time for the server side

T_{QD} Server queue delay

$T_{S\_C\_CH}$ Average client processing time for generating the Client.Hello message

$T_{S\_C\_SH}$ Average client processing time for processing Server.Hello message

$T_{S\_C\_SCERT}$ Average client processing time for processing the server certificate

$T_{S\_C\_CERTREQ}$ Average client processing time for processing the CertificateRequest message

$T_{S\_C\_SHD}$ Average client processing time for processing the ServerHelloDone message

$T_{S\_C\_CCERT}$ Average client processing time for generating the Client.Certificate message

$T_{S\_C\_CKX}$ Average client processing time for generating the ClientKeyExchange message

$T_{S\_C\_CCCS}$ Average client processing time for generating the Client ChangeCipherSpec message

$T_{S\_C\_CFIN}$ Average client processing time for generating the Client.Finished message

$T_{S\_C\_SCCS}$ Average client processing time for processing the Server ChangeCipherSpec message

$T_{S\_C\_SFIN}$ Average client processing time for processing the Server.Finished message

$T_{S\_C\_ECDH}$ Average client processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)

$T_{S\_C\_MS}$ Average client processing time for computing the master secret from the premaster secret

$T_{S\_C\_RSAENC}$ Average client processing time for encryption operation using the Server’s public key (applicable if RSA key exchange suite is used)

$T_{S\_S\_CH}$ Average server processing time for processing the Client.Hello message

$T_{S\_S\_SH}$ Average server processing time for generating the Server.Hello message

$T_{S\_S\_SCERT}$ Average server processing time for generating the Server.Certificate message

$T_{S\_S\_CERTREQ}$ Average server processing time for generating the CertificateRequest message

$T_{S\_S\_SHD}$	Average server processing time for generating the ServerHelloDone message
$T_{S\_S\_CCERT}$	Average server processing time for processing the client certificate
$T_{S\_S\_CKX}$	Average server processing time for processing the ClientKeyExchange message
$T_{S\_S\_CCCS}$	Average server processing time for processing the Client ChangeCipherSpec message
$T_{S\_S\_CFIN}$	Average server processing time for processing the Client.Finished message
$T_{S\_S\_SCCS}$	Average server processing time for generating the Server ChangeCipherSpec message
$T_{S\_S\_SFIN}$	Average server processing time for generating the Server.Finished message
$T_{S\_S\_ECDH}$	Average server processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)
$T_{S\_S\_MS}$	Average server processing time for computing the master secret from the premaster secret
$T_{S\_S\_RSADEC}$	Average server processing time for decryption operation using its own private key (applicable if RSA key exchange suite is used)
T_{TD}	Data transmission time (s)
$T_{traversal}$	One way traversal delay of the GSM network specific to the data bearer and GSM service provider
T_{X}	Average service time for any of the applicable handshake messages for a given group
$T_{X,c,a}$	Average service time for any of the applicable handshake messages for the key exchange suite with the smallest public key size in a group.
$T_{X,c,b}$	Average service time for any of the applicable handshake messages for the key exchange suite with the intermediate public key size in a group.
$T_{X,c,c}$	Average service time for any of the applicable handshake messages for the key exchange suite with the highest public key size in a group.

LIST OF ABBREVIATIONS

BSC	Base Station Controller
BTS	Base Transceiver Station
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDH_ECDSA	Elliptic Curve Diffie-Hellman Elliptic Curve Digital Signature Algorithm
AMPS	Advanced Mobile Phone Service
CA	Certification Authority
CDMA	Code Division Multiple Access
CDPD	Cellular Digital Packet Data
CSD	Circuit Switched Data
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDEN	Integrated Digital Enhanced Network
IPv4	Internet Protocol version 4
ISO/OSI	International Organization for Standardization/Open System Interconnection
ITSI	Individual TETRA Subscriber Identity
LAN	Local Area Network
MAN	Metropolitan Area Network
MSC	Mobile Switching Center
MSISDN	Mobile Station International ISDN Number
OMA	Open Mobile Alliance
OS	Operating System
PC	Personal Computer
PDC	Personal Digital Communications
PRF	Pseudo Random Function
QoS	Quality of Service
RAS	Remote Access Server

RSA	Rivest Shamir Adleman
RTT	Round Trip Time
SHA	Secure Hash Algorithm
SMS	Short Message Service
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol / Internet Protocol
TLS	Transport Layer Security
USSD	Unstructured Supplementary Services Data
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WDP	Wireless Datagram Protocol
WML	Wireless Markup Language
WSP	Wireless Session Protocol
WTA	Wireless Telephony Application
WTLS	Wireless Transport Layer Security
WTP	Wireless Transaction Protocol
WWW	World Wide Web
XML	Extensible Markup Language

1. INTRODUCTION

Emerging growth of the mobility requirements for today's daily life has brought up an increasing demand on the use of WAP (Wireless Application Protocol) [1] applications. WAP is an enabling technology for mobile Internet access using resource constrained handheld devices. WAP standards have been developed by an international industry-wide organization WAP Forum. However, WAP Forum no longer exists as an independent organization since June 2002. The WAP Forum has consolidated into the Open Mobile Alliance (OMA) and the specification work continues within OMA since 2002. Latest version announced is WAP 2.0 but most handheld devices still support the preceding version WAP 1.2.

The number of mobile handheld devices accessing Internet increased rapidly for the last four years. Percentage of wireless Internet users was 16% of the overall Internet users by the year 2001, where it is 41.5% for year 2004 and expected to be 60% by the year 2007 according to the recent researches in [2] and [3]. Global m-commerce revenue was approximately 3 billion USD for year 2001, where it is predicted to be 19 billion USD by the year 2005 as stated in [4]. Therefore, security of WAP transactions becomes one of the biggest security concerns for the future Internet use. Considering today's available value-added WAP services like Mobile Internet Banking, M-commerce, etc., we can say that security requirements of WAP applications are not that different from the traditional wired Internet.

WTLS (Wireless Transport Layer Security) [5] is the security protocol designed for WAP protocol stack. WTLS is built on the Internet standard TLS v1.0 [6] which is based on the SSL v3.0 [7] protocol developed by Netscape Corp. WTLS operates between the mobile client and the server, which is also called as WAP Gateway. It addresses confidentiality, integrity and authentication of the information flow between the mobile client and the server. WAP gateway retrieves the requested WAP content from the external web servers using the regular TCP/IP protocol suite and sends the content back to the client. It is obvious that there is no end-to-end security means between the client and the content server for WAP applications. Although WTLS is used to secure the communication between WAP Gateway and client, there is no guarantee that the WAP Gateway communicates over SSL with the content server. Network participants of a typical WAP access are given in Figure 1.1.

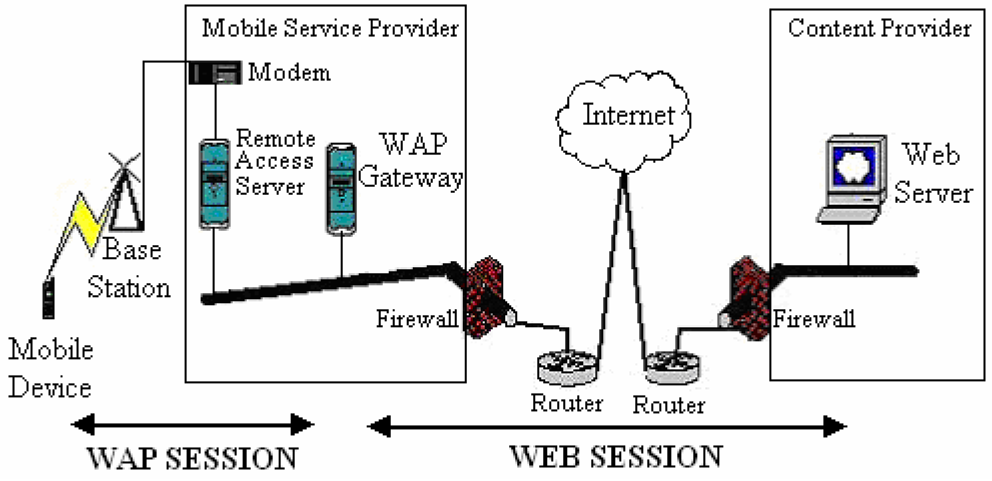


Figure 1.1 Network participants of a typical WAP access

WTLS provides all the security related parameters to the upper layer protocols. WTLS Handshake Protocol is used to negotiate on the cryptographic algorithms to be used, exchange secret keys and digital certificates.

Although there are anonymous key exchange suites that are offered by the WTLS standard [5], they are not considered secure. Neither client nor the server is authenticated at the anonymous key exchange suites. WTLS uses digital certificates for authentication of the peers and authenticated key exchange between them. Digital certificates are issued by trusted Certification Authorities (CA). They contain identity information of the peer together with the public key to be used during the handshake. Authentication of the peer requires verification of its certificate using the public key retrieved from the CA certificate. Handshake mechanism varies depending on the cryptosystem used.

Public key cryptosystems are used to verify the certificates and exchange secret keys between the peers. Public key cryptosystem operations are generally slow and the processing time significantly increases as the larger key sizes are used. Public key cryptosystems key sizes offered by the WTLS standard are not strong enough to meet today's WAP applications' security requirements. Considering the low processing power of the handheld devices, it may be reasonable to restrict the key sizes. However, WAP Forum does not seem to stand on a serious cryptographic research while recommending the key sizes at the WTLS standard [5]. Therefore, a performance evaluation of the WTLS Handshake Protocol for different key exchange suites is valuable. Especially a feasibility study on the use of stronger key sizes that has not been offered by the standard yet will put a light on the future progress of the WTLS standard.

In this thesis work, performance evaluation of the WTLS Handshake Protocol has been performed. Performance model considers the client and server processing times, server queue delay and the data transmission time over the channel as bottleneck candidates of the handshake operation. Processing times and data transmission times have been measured by performing the tests over a real GSM service provider. Test client is a Nokia 7650 phone with Symbian operating system. Symbian OS is widely used at today's smart phones. It has a flexible programming interface that supports many programming languages like C/C++, Java, etc. Crypto primitives and the WTLS Handshake Protocol have been implemented in C++. Queue delay performance was modeled analytically. However, measured server processing times have been used in that analytical model to analyze the effects of server waiting time. WTLS handshake protocol simulations have been performed over the GSM CSD data bearer, and another performance test methodology has been followed to predict the data transmission time characteristics for GPRS bearer. Timing measurements have been used together with the performance model to analyze the effects of the bottleneck candidates on the WTLS handshake protocol.

In Section 2, background information on WAP protocol and public key cryptography is given. Previous works on WTLS performance evaluation are also summarized in this section. Section 3 dwells upon the proposed performance model. Performance data gathered from implementation results are evaluated in Section 4. Section 5 gives the conclusions and the future work.

2. WAP BACKGROUND AND LITERATURE SURVEY

Before going into details of the WTLS handshake protocol, it is worth mentioning how clients access to WAP content and what are the WAP protocol stack components. Section 2.1 explains the basic operations when accessing WAP content. WAP protocol stack components are briefly defined in Section 2.2, where WTLS handshake protocol is considered in more detail in Section 2.4. Section 2.3 gives background information on public key cryptosystems that are used in the WTLS Handshake Protocol. Cryptographic functions which are used to compute the master secret are defined in Section 2.5. Discussion of previous works on WTLS/SSL performance is also given at the end of this chapter.

2.1. WAP Access Model

The most widely used WAP content access model is the one that uses mobile service provider's WAP gateway. WAP content format is WML (Wireless Markup Language) [8] which is based on the familiar WWW (World Wide Web) content format HTML (HyperText Markup Language) [9]. WML is a structured content type designed using XML (Extensible Markup Language) [10]. WAP gateway is responsible for handling client requests. Clients send the WAP request to the gateway, gateway communicates with the content provider and gets the HTML reply then applying the necessary binary WML format conversion the gateway sends the WAP content to the client. Generally the WAP gateway has two main functionalities as stated in [1]. These are:

- Protocol Gateway: Translates the WAP protocol stack to the WWW protocol stack (HTTP [11]and TCP/IP)

- Content Encoders and Decoders: Translate WAP content to the channel optimized encoded format.

Typical WAP access message flow between client, gateway and content provider is visualized in Figure 2.1.

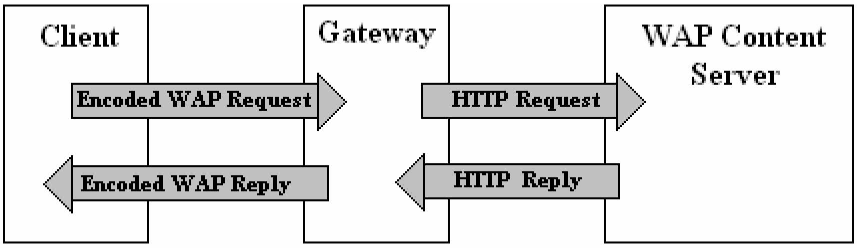


Figure 2.1 WAP access model

2.2. WAP Protocol Stack Components

WAP protocol stack is designed taking the ISO/OSI Layered Structure [12] as a reference. It is composed of 6 protocol layers. A brief description of each protocol layer is given in the following parts.

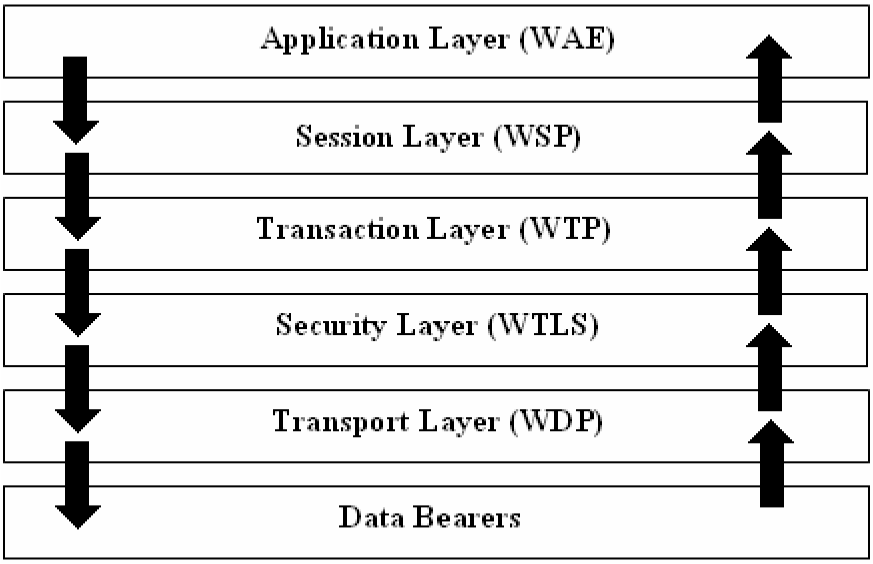


Figure 2.2 WAP protocol stack components

2.2.1. Wireless Application Environment (WAE)

WAE [13] layer includes all the specification about WAP application specification and execution especially the client side. WAE is based on both WWW and Mobile Telephony technologies. Its main purpose is to establish an interoperable environment that will allow operators, service providers and developers to build applications and services for mobile WAP clients. WAE has following functionalities:

- WML (Wireless Markup Language): Special markup language designed for resource constrained hand-held WAP clients. WAP pages are presented in WML and microbrowsers in the WAP clients know how to render WML pages. WML is similar to HTML which is used to present web pages of WWW technology.
- WMLScript [14]: A lightweight scripting language designed for WAP clients. WMLScript is especially useful to add client side logic to WAP browsing.
- WTA (Wireless Telephony Application) [15]: Telephony services and programming interfaces.

2.2.2. Wireless Session Protocol (WSP)

WSP [16] provides a means for exchange of data between client and servers by establishing a reliable session and releasing the session in an orderly manner. General features of WSP are:

- Establishing/releasing session between client and server
- Capability negotiation to agree on a common level of protocol functionality
- Exchange content between client and server
- Suspend and resume the session

WSP offers two protocols one of which is connection oriented transaction service and the second one is connectionless services over a datagram transport service.

2.2.3. Wireless Transaction Protocol (WTP)

WTP [17] is the transaction protocol defined for WAP access. WTP provides the necessary services for browsing WAP pages; actually it serves for WAP transactions. A transaction for WAP is defined as the duo of request and response. WTP lays on top of WAP datagram service WDP (Wireless Datagram Protocol) [18] and optionally the security layer WTLS (Wireless Transport Layer Security) [5] if used.

2.2.4. Wireless Transport Layer Security (WTLS)

WTLS [5] is the security protocol defined for the WAP protocol stack. Use of WTLS is not mandatory, it is optional to enable or disable WTLS protocol. WTLS is based on well known Internet standard TLS v1.0 [6] (formerly known as SSL v3 [7]) and it is optimized for use over narrow-band communication channels. WTLS provides the following basic security services for WAP applications:

- Authentication
- Integrity
- Confidentiality

Section 2.4 deals with WTLS and its sub-protocols in more detail.

2.2.5. Wireless Datagram Protocol (WDP)

WDP [18] is the transport layer protocol in the WAP architecture and it operates over data bearer services as GSM, CDMA etc. WDP offers a consistent service to upper layer protocols of WAP by communicating with different bearer services transparently.

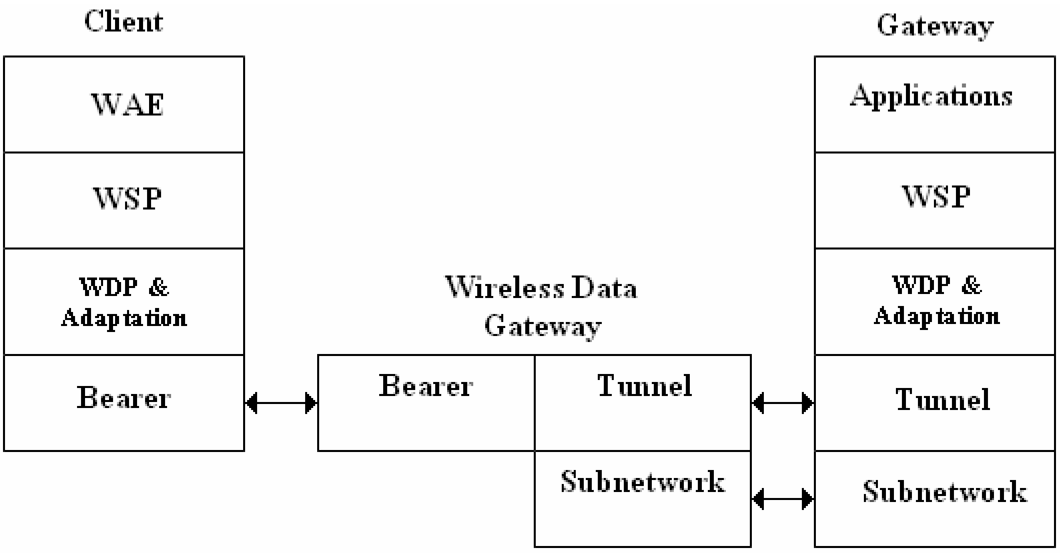


Figure 2.3 WDP structure

Figure 2.3 shows the underlying structure of the WDP layer. The adaptation layer is the part of WDP that maps the WDP functions to the underlying bearer service. The wireless data gateway forwards the WDP packets to a WAP proxy via a tunneling protocol. The sub-network may be one of the networking technologies that provide communication between peers, like LANs (Local Area Networks) operating TCP/IP over Ethernet etc. [18] The WAP proxy may directly provide the content or it may retrieve the content from the wired Internet to send it back to the client.

2.2.6. Bearers

WAP protocols can operate over various bearer services which can be grouped as Short Message Service (SMS), circuit-switched data, and packet data. WDP protocol provides the means to operate transparently over different bearer services for the upper layer WAP protocols. Different bearer services offer different QoS (Quality of Service), throughput, error rate, and delays. New bearers may be adapted to the WAP protocol family as the mobile market evolves and better bearer services are designed for WAP use. Table 2.1 gives the available bearers from WAP 1.2.1 (June 200) standard [18], with network and address type specification.

Network	Bearer type	Address type
Any	Any	IPv4
Any	Any	IPv6
GSM	USSD	Any
GSM	SMS	GSM_MSISDN
ANSI-136	GUTS/R-Data	ANSI_136_MSISDN
IS-95 CDMA	SMS	IS_637_MSISDN
IS-95 CDMA	CSD	IPv4
IS-95 CDMA	Packet Data	IPv4
ANSI-136	CSD	IPv4
ANSI-136	Packet Data	IPv4
GSM CSD IPv4	CSD	IPv4
GSM GPRS IPv4	GPRS	IPv4
GSM USSD	USSD	IPv4
AMPS CDPD	CDPD	IPv4
PDC CSD	CSD	IPv4
PDC	Packet Data	IPv4
IDEN	SMS	iDEN_MSISDN
IDEN	CSD	IPv4
IDEN	Packet Data	IPv4
Paging network	FLEXTM	FLEX_MSISDN
PHS	SMS	PHS_MSISDN
PHS	CSD	IPv4
GSM	USSD	GSM_Service_Code
TETRA	SDS	TETRA_ITSI
TETRA	SDS	TETRA_MSISDN
TETRA	Packet Data	IPv4
Paging Network	ReFLEXTM	ReFLEX_MSIDDN
GSM	USSD	GSM_MSISDN
Mobitex	MPAK	MAN
ANSI-136	GHOST/R_DATA	GSM_MSISDN

Table 2.1 WAP network bearer types

2.3. Public Key Cryptosystems

Public key cryptosystems have been the most appropriate solution to some major security problems like integrity, authentication and non-repudiation. Main security goals are defined below.

- *Integrity*: Making sure that it will be notified if the message has been altered since the last checkpoint.
- *Authentication*: Making sure of a communicating party's identity
- *Confidentiality*: Only the intended parties can see the content of a message
- *Non-repudiation*: The sender cannot claim that he/she did not send the message

In the public key cryptosystems, each user holds a key pair. One of the keys is the private key, must be kept secret and only the owner can access it. The second key is the public key that can be accessed by any party needing it for encryption and signature validation.

Public key cryptosystems can be used for all of the security goals mentioned above. Integrity, authentication and non-repudiation can be ensured by digitally signing the message with the private key. The recipients can verify the signature by using the publicly available public key of the sender. The public key cannot decrypt the message that it encrypted, also the private cannot easily be derived from the public key. Figure 2.4 shows the signature issuance and signature verification models for public key cryptosystems in general.

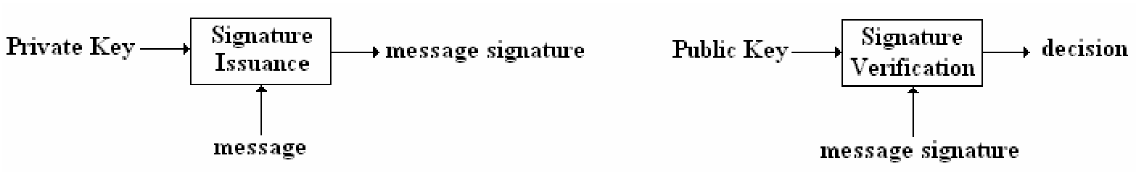


Figure 2.4 Public key cryptosystems signature issuance and verification models

Public key cryptosystems have encryption/decryption features to ensure confidentiality but it is not feasible to use public key cryptosystems for confidentiality issues directly. Symmetric algorithms are faster than public key algorithms and they are used for encryption/decryption purposes. The biggest problem of communicating securely via symmetric encryption is that the peers must agree on a secret key that only they know and no one else can capture/generate. Fortunately, public key cryptosystems contribute to the confidentiality by providing means of securely exchanging such secrets between the peers. Figure 2.5 shows the public key cryptosystems encryption and decryption models in general.

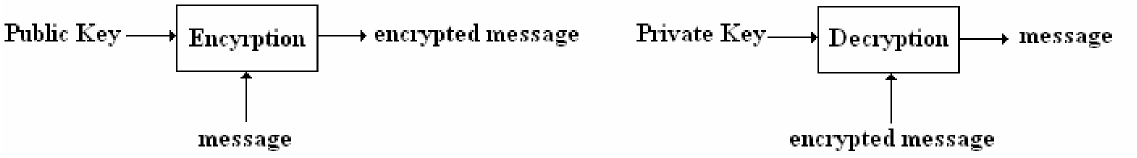


Figure 2.5 Public key cryptosystems encryption and decryption models

RSA (Rivest-Shamir-Adleman) [19] and ECC (Elliptic Curve Cryptography) [20] are the most widely used public key cryptosystems in WTLS Handshake Protocol. These cryptosystems and basic cryptographic operations are briefly defined in the following parts of this section.

2.3.1. RSA Cryptosystems

RSA was proposed by Rivest, Shamir, and Adleman in 1977. It is based on the idea that factorization of large integers into their prime factors is a hard problem. Thus, the difficulty of obtaining the private key using the public key is the one-way function that has the equivalent difficulty of finding the prime factors of a large integer.

In RSA, public and private keys are generated as follows:

- Choose two large prime numbers, p and q, compute the public modulus $n = p \times q$
- Choose a random public key, e, where e and $(p-1)\times(q-1)$ are relatively prime
- Compute the private key d as, $d = e^{-1} \bmod\left[(p-1)(q-1)\right]$

Encryption mechanism

The plaintext P, is thus encrypted to generate ciphertext C as follows:

$$C = P^{e} \bmod n$$

Decryption mechanism

and C is decrypted to recover the plaintext ,P, as:

$$P = C^{d} \bmod n$$

RSA key exchange mechanism

RSA key exchange is performed by using the encryption and decryption properties of the RSA algorithm. A premaster secret is encrypted by using the public key of the other communicating party and the encrypted premaster secret is sent to the owner of the public key. Encrypted premaster secret can only be decrypted by the owner of the private key that is related to the public key which was used to encrypt the premaster secret. After the decryption operation, both parties know the same premaster secret. Therefore, they can compute the master secret from this shared secret.

2.3.2. Elliptic Curve Cryptosystems

ECC was proposed by Koblitz [20] in 1987 and by V.S. Miller [21] in 1985 separately and it is now the strongest rival against the RSA cryptosystems because of several advantages. Each elliptic curve is a different cryptosystem. The security of ECC stems from the hardness of the ECDLP (Elliptic Curve Discrete Logarithm Problem). ECDLP is defined as below:

ECDLP Definition: Given an elliptic curve E defined over a finite field F_q, a point $P \in E(F_q)$ of order n, and a point $Q = lP$ where $0 \le l \le n-1$, determine l.

Smaller modulus values can be used in ECC to achieve the same level of security as compared to larger RSA modulus values. This brings the advantages of easier and cheaper implementations, transmitting less data. Moreover, ECC is faster as well.

ECC cryptosystems use ECDSA (Elliptic Curve Digital Signature Algorithm) [24] for signature issuance and signature verification. ECDSA is the elliptic curve analogue of the DSA [25] (Digital Signature Algorithm).

ECDSA signature generation mechanism

To generate the ECDSA signature of a message m, an entity *Alice* with domain parameters $D = (q, FR, a, b, G, n, h)$ and key pair (d,Q) performs the following operations:

1. Select a random integer k, $1 \le k \le n-1$.
2. Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$. If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $e = SHA-1(m)$.
5. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
6. Signature of the message m is, (r,s).

ECDSA signature verification mechanism

To verify *Alice*'s ECDSA signature (*r,s*) of the message *m*, an entity *Bob* should first obtain *Alice*'s domain parameters $D = (q, FR, a, b, G, n, h)$ and public key Q, and perform the following operations:

1. Verify that *r* and *s* satisfy the condition, $1 \leq r, s \leq n-1$.
2. Compute $e = SHA-1(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $X = u_1 G + u_2 Q$. If $X = 0$, signature is rejected. Otherwise, compute $v = x_1 \bmod n$ where $X = (x_1, y_1)$.
6. Signature is accepted iff $v = r$.

ECDH key exchange mechanism

ECDH (Elliptic Curve Diffie-Hellman) [22] key exchange is performed to securely share a secret between two communicating parties in ECC cryptosystems. ECDH is the elliptic curve version of the DH (Diffie-Hellman) [23] key exchange.

Suppose that *Alice* and *Bob* want to securely exchange a secret value (premaster secret key), and they are using ECC cryptosystems. Therefore, they will perform ECDH key exchange as defined below. ECDH key exchange mechanism is shown in Figure 2.6.

1. *Alice* and *Bob* agree on an elliptic curve *E*, and a large prime number *P*.
2. *Alice* and *Bob* agree on a point (*x,y*) on *E* over *GF*(*P*).
3. Both peers separately performs:
   a. *Alice* secretly chooses a positive integer *m*, and computes $(u, v) = m * (x, y)$.
   b. *Bob* secretly chooses a positive integer *n*, and computes $(r, s) = n * (x, y)$.
4. Both peers separately performs:

a. *Alice* sends (*u*,*v*) to *Bob*.

b. *Bob* sends (*r*,*s*) to *Alice*.

5. *Alice* and *Bob* computes the secret point (*g*,*h*):

a. *Alice* secretly computes $(g,h) = m*(r,s)$.

b. *Bob* secretly computes $(g,h) = n*(u,v)$.

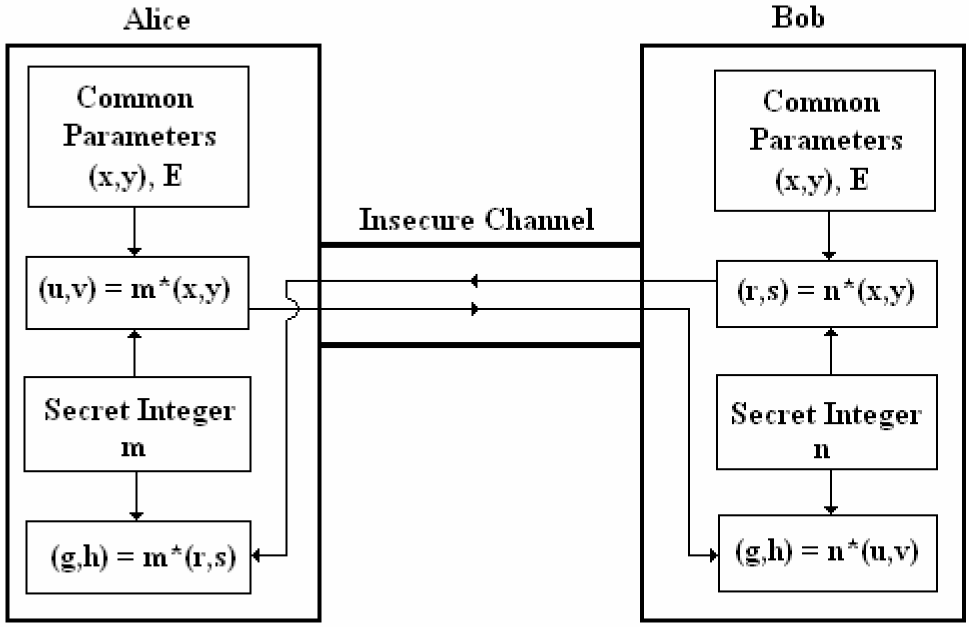


Figure 2.6 ECDH key exchange mechanism

2.3.3. Cryptographic Strength Level Comparison of RSA and Elliptic Curve Cryptosystems

Lenstra and Verheul have compared the cryptographic key sizes of RSA and ECC cryptosystems in [26]. Table 2.2 gives the equivalent ECC and RSA key sizes taken from [26], for three cryptographic strength levels considered in this study. Three different types of ECC curves have been considered, these are prime, Koblitz, and random curves. ECC curves that are used in this study, are the recommended curves by the US NIST (United States National Institute of Standards and Technologies) in [27].

Cryptographic Strength Level		ECC	RSA
Stronger	1	160 bit Prime, 163 bit Koblitz, 163 bit Random	1024 bit
	2	224 bit Prime, 233 bit Koblitz, 233 bit Random	2048 bit
	3	256 bit Prime, 283 bit Koblitz, 283 bit Random	3072 bit

Table 2.2 RSA and Elliptic Curve Cryptosystems cryptographic strength level comparison

2.4. WTLS Handshake Protocol

WTLS Handshake Protocol [5] is one of the four clients of the *WTLS Record protocol* [5]. The *WTLS Record Protocol* is a layered protocol that optionally compresses data, applies MAC, encryption and transmits the data. The other clients of the *WTLS Record Layer* are *Change Cipher Spec Protocol*, and *Alert Protocol*.

WTLS Handshake Protocol [5] is used to allow WAP client and gateways to agree upon security parameters for the record layer, authenticate themselves and report error conditions to each other. *Change Cipher Spec Protocol*, *Alert Protocol* and *Handshake Protocol* are the sub-protocols of the WTLS Handshake protocol. Figure 2.7 gives the relation between the WTLS Record Protocol, WTLS Handshake Protocol and its sub-protocols.

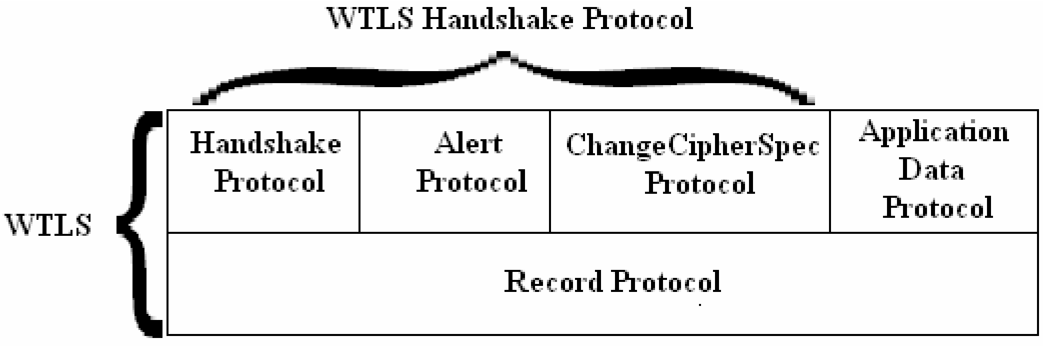


Figure 2.7 WTLS protocol components

Change Cipher Spec Protocol may be used either by the client or the gateway. It notifies the other party that the security negotiation has been completed. The following messages are protected by the agreed security parameters. The first messages that will be protected after the Change Cipher Spec message are client finished and server finished messages.

Alert Protocol is used to inform the peers about handshake errors. Alert messages convey the severity of the message and a description of the alert. Alert levels are specified as warning, critical and fatal in the standard [5]. Critical alert messages result in the immediate termination of the current connection, where the connection may continue in the case of other levels of alert messages.

Use of WTLS for WAP sessions is optional and the peers must negotiate on the security parameters before starting the secure session. WTLS Handshake Protocol is the sub-protocol that provides the necessary security parameters to the upper layer Record Protocol. The WTLS Handshake Protocol's main features are listed below:

- Exchanging the client hello messages
- Exchanging the random values
- Exchanging the authentication information (certificates, cryptographic information etc.)
- Provide means to generate a master secret from the pre-master secret and the previously exchanged random values
- Providing the security parameters to the Record Protocol

WTLS supports RSA [19] and ECC [20] cryptosystems. If the key exchange is performed using RSA cryptosystems, encryption and decryption features of RSA is used. ECDH (Elliptic Curve Diffie-Hellman) [22] key exchange method is performed if ECC is used. The standard [5] also offers anonymous key exchange suites, DH (Diffie-Hellman) [23], RSA_anon, and ECDH_anon. Anonymous key exchange suites do not authenticate any of the peers, so they are not considered as secure and not in the scope of this thesis work.

WTLS provides authentication by means of the digital certificates. Verification of the digital certificates requires public-key operations. RSA has its own verification feature. ECDSA (Elliptic Curve Digital Signature Algorithm) [24] is used for signature verification purposes if ECC is to be used.

WTLS Handshake Protocol may be performed in three basic type, these are Full Handshake, Abbreviated Handshake and Optimized Handshake.

2.4.1. Full Handshake

There exist three different types of WTLF Full Handshake. These are *Mutual Authenticated, Server Authenticated* and *Anonymous* WTLS Full Handshake. Anonymous key exchange suites are not secure because neither the client nor the server is authenticated. Also, it is possible to perform MIM (Man In the Middle) attacks if anonymous key exchange suites are used. Mutual Authenticated WTLS Full Handshake and Server Authenticated WTLS Full Handshake will be considered in this section.

2.4.1.1. Mutual Authenticated WTLS Full Handshake

Mutual Authenticated WTLS Full Handshake requires both the client and the server have a valid certificate appropriate to the selected key exchange suite. These digital certificates are used through the key exchange process to compute the premaster secret. RSA or ECDH_ECDSA key exchange suites can be used in mutual authenticated WTLS full handshake.

The first handshake message to be sent by the client is the client hello message. Critical information that the client hello message includes are the client's random value, key exchange suites supported by the client with the client's first preference first, list of trusted certificates known by the client, and list of the cryptographic options supported by the client. Version of the WTLS protocol, session id, compression methods, and key refresh period are also presented in the client hello message. After sending the client hello message, the client waits for the server hello message.

If the server can not find an acceptable set of algorithms after receiving the client hello message, it sends a handshake failure alert. Otherwise, the server responds with the server hello message. Server hello message includes the server random value, session id, selected key exchange suite, selected cipher suite, compression method and key refresh period information.

Server sends its certificate after the server hello message. WTLS supports use of X509v3, X9.68 or WTLS certificates. It is most suitable to use the WTLS certificates because they are optimized for size. Server certificate must be verified by the client upon receiving. Verification of a certificate requires the verification of the digital signature using the CA (Certification Authority) public key retrieved from the CA certificate. The client must also verify the subject, issuer, and the validity time interval of the certificate. Authenticated server certificate will be used to compute the premaster secret.

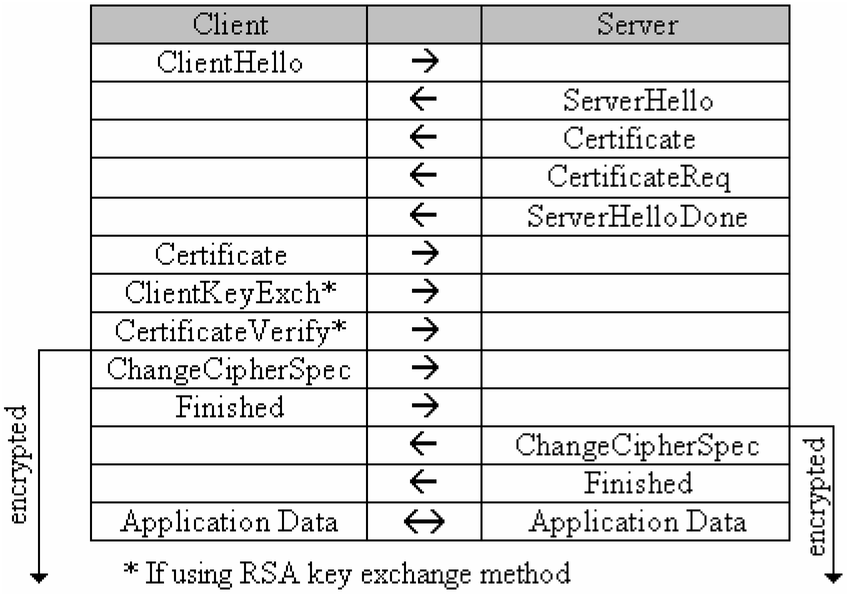
The server requests a certificate from the client by sending the certificate request message. The next message that will be sent by the server is server hello done message which indicates that all necessary server messages have been sent by the server and it is waiting for the client response.

The client must send a valid certificate to the server after receiving the server hello done message. Even if the client does not have a valid certificate, it must send an empty certificate message to the server. Then the server may continue to the handshake or it may terminate the handshake and send a critical error message to the client.

The client key exchange message is sent by the client if RSA key exchange suite is used. The certificate sent by the client has the enough information to compute the premaster secret if ECDH_ECDSA key exchange suite is used, so the client key exchange message is omitted if ECDH_ECDSA key exchange suite is used. Client key exchange message contains the encrypted premaster secret when RSA is used. Premaster secret decided by the client is encrypted using the server public key retrieved from the server certificate. The server decrypts the encrypted premaster secret using its private key.

The client must also send the certificate verify message to explicitly verify its certificate if RSA key exchange method is used. Certificate verify message includes signature of the all previous handshake messages' hash value. Server must be able to verify the signature using the public key retrieved from the client certificate.

After this point the client sends the change cipher spec message and immediately sends the client finished message. All the messages sent after the change cipher spec message is protected by the security parameters that have been agreed upon. When the server receives the change cipher spec message sent by the client, it will send server change cipher spec message to the client and all the messages that will be sent by the server will also be protected by the same security parameters. The first server message under agreed parameters will be the server finished message and the peers can start to exchange application data after sending the finished messages. Mutual Authenticated WTLS Full Handshake message flow is given at Figure 2.8.


encrypted
encrypted

Client		Server
ClientHello	→	
	←	ServerHello
	←	Certificate
	←	CertificateReq
	←	ServerHelloDone
Certificate	→	
ClientKeyExch*	→	
CertificateVerify*	→	
ChangeCipherSpec	→	
Finished	→	
	←	ChangeCipherSpec
	←	Finished
Application Data	↔	Application Data

* If using RSA key exchange method

Figure 2.8 Mutual authenticated WTLS full handshake message flow

2.4.1.2. Server Authenticated WTLS Full Handshake

Server Authenticated WTLS Full Handshake requires only the server to have a valid certificate appropriate to the selected key exchange suite. RSA or ECDH_ECDSA key exchange suites can be used in server authenticated WTLS full handshake.

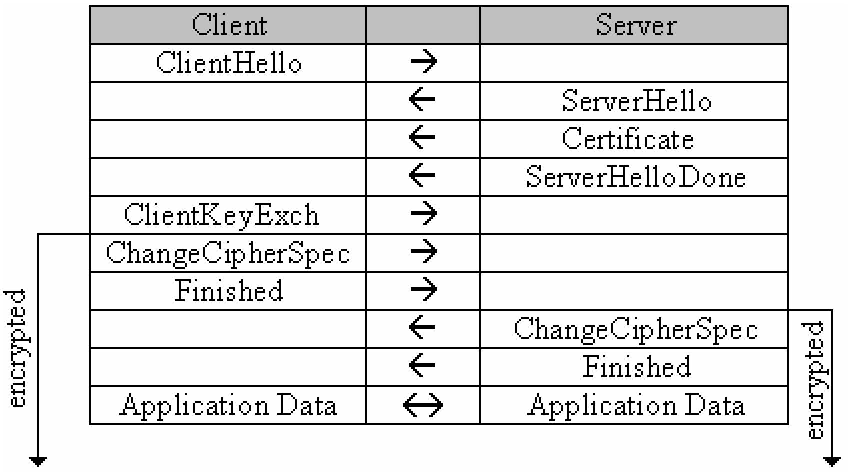
The first handshake message to be sent by the client is the client hello message. Client hello message includes client's random value, key exchange suites supported by the client with the client's first preference first, list of trusted certificates known by the client, list of the cryptographic options supported by the client, version of the WTLS protocol, session id, compression methods, and key refresh period. After sending the client hello message, the client waits for the server hello message.

The server sends the server hello message to the client after receiving the client hello message. Server hello message includes the server random value, session id, selected key exchange suite, selected cipher suite, compression method and key refresh period information.

Server sends its certificate after the server hello message. Server certificate must be verified by the client upon receiving. After the certificate message, server sends the server hello done message which indicates that all necessary server messages have been sent by the server and it is waiting for the client response.

The client key exchange message is sent by the client after receiving the server hello done message. Premaster secret is set with the client key exchange message. The client encrypts the premaster secret using the server's public key and sends the encrypted premaster secret in the client key exchange message. The server then decrypts the RSA encrypted premaster secret using its private key. The client must send its EC Diffie-Hellman public key in the client key exchange message to the server. Server uses its private key and client's public key to compute the premaster secret. Similarly, client uses its private key and server's public key to compute the premaster secret.

Client sends the change cipher spec message and the client finished message after the client key exchange message. All the messages sent after the change cipher spec message is protected by the security parameters that have been agreed upon. When the server receives the change cipher spec message sent by the client, it will send server change cipher spec message to the client and all the messages that will be sent by the server will also be protected by the same security parameters. Server Authenticated WTLS Full Handshake message flow is given at Figure 2.9.

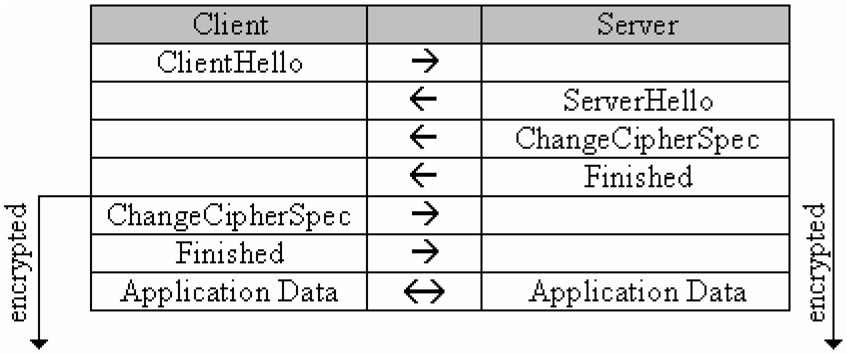


Client		Server
ClientHello	→	
	←	ServerHello
	←	Certificate
	←	ServerHelloDone
ClientKeyExch	→	
ChangeCipherSpec	→	
Finished	→	
	←	ChangeCipherSpec
	←	Finished
Application Data	↔	Application Data

Figure 2.9 Server authenticated WTLS full handshake message flow

2.4.2. Abbreviated Handshake

The client may resume a previously established secure session with the server instead of performing a full handshake. In the case of resuming an old secure session, the client sends a Client Hello message with the Session ID of the session to be resumed. The server checks its secure session cache. If a match is found, the server sends Server Change Cipher Spec message and Server Finished message. The client must respond with Client Change Cipher Spec message and Client Finished message and peers start to exchange application data in a secure manner. If the server can not find the secure Session ID in the Client Hello message, it can not resume the previous session and a new full handshake is initiated. WTLS Abbreviated Handshake Protocol message flow is given at Figure 2.10.



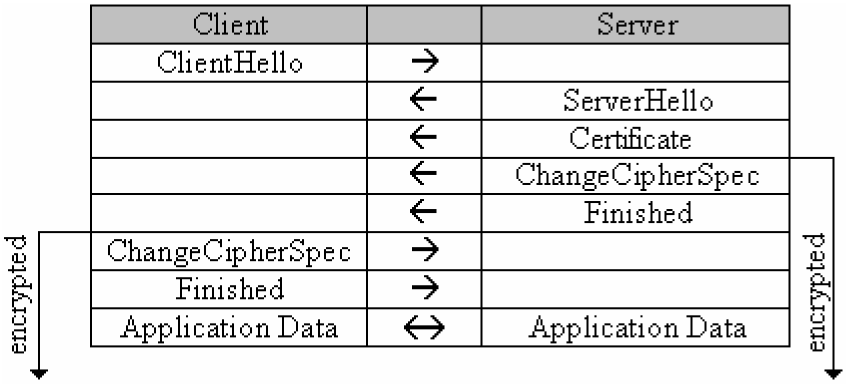
Client		Server
ClientHello	→	
	←	ServerHello
	←	ChangeCipherSpec
	←	Finished
ChangeCipherSpec	→	
Finished	→	
Application Data	↔	Application Data

encrypted
encrypted

Figure 2.10 WTLS abbreviated handshake message flow

2.4.3. Optimized Handshake

When the server receives the Client Hello message, it can retrieve the client's certificate from a distribution center or its own certificate store. For example when the client certificate contains the ECDH [22] (Elliptic Curve Diffie-Hellman) parameters, the server can directly compute the premaster secret and master secret by using the certificate retrieved from the store. In this case the server sends its certificate to the client, then Server Change Cipher Spec message and Server Finished message. The client sends Client Change Cipher Spec message and Client Finished message. After all these messages, application data can be exchanged between the peers. WTLS Optimized Handshake Protocol message flow is given at Figure 2.11.



Client		Server
ClientHello	→	
	←	ServerHello
	←	Certificate
	←	ChangeCipherSpec
	←	Finished
ChangeCipherSpec	→	
Finished	→	
Application Data	↔	Application Data

encrypted
encrypted

Figure 2.11 WTLS optimized handshake message flow

2.5. Master Secret Computation in WTLS Handshake Protocol

Regardless of the key exchange suite used, peers must compute the master secret for using in the bulk encryption process after agreeing on the premaster secret. The same algorithm is used to convert the premaster secret into the master secret for all types of key exchange suites. It is recommended that the premaster secret is deleted from memory after computing the master secret.

$$mastersecret = PRF\ (\,premastersecret,\ \text{“}master\ secret\text{”}, Client.random+Server.random\,) \qquad (\,2.1\,)$$

The “+” operation stands for the concatenation of two random values generating a new random value as an input to the function. The premaster secret key length may vary depending on the key exchange suite selected but the master secret will always be 20 bytes in length. PRF is the pseudo-random function defined in [5].

If the RSA key exchange suite is used, premaster secret is determined by the client. Client encrypts the premaster secret using the server’s RSA public key and sends it to the server in Client Key Exchange Message. The server decrypts the premaster secret using its own private key and continues to compute the master secret.

If the ECDH_ECDSA key exchange suite is used, the ECDH computation is performed and the negotiated key (Z) is used as the premaster secret to compute the master secret.

$PRF(.)$ is defined as:

$$PRF(\, secret, label, seed) = P\_hash(secret, label + seed) \qquad (2.2)$$

$P\_hash(\sec ret, data)$ is defined as the data expansion function using a single hash function. $P\_hash(.)$ function can be iterated as many times as needed, at each iteration it generates additional 20 bytes of output.

$$\begin{aligned} P\_hash(secret, data) \quad = \quad & HMAC\_hash(secret, A(0)+data) + \\ & HMAC\_hash(secret, A(1)+data) + \qquad (2.3) \\ & HMAC\_hash(secret, A(2)+data) + \ldots \end{aligned}$$

Where;

$$A(0) = data$$

$$A\,(i) = HMAC\_hash(secret, A(i\text{-}1))$$

$HMAC\_hash(.)$ is defined as below:

$$HMAC\_hash(K, data) = H(K \oplus opad \quad + \quad H(K \oplus ipad \quad + \quad data)) \qquad (2.4)$$

Where;

$$ipad = the \quad byte \quad 0x36 \quad repeated \quad 64 \quad times$$

$$opad = the \quad byte \quad 0x5C \quad repeated \quad 64 \quad times$$

$H(.)$ is one of the allowed cryptographic hash functions in the standard [5]. Available hash functions defined in the standard are SHA-1 [28] and MD5 [29].

It is enough to iterate the $P\_hash(.)$ function once to generate the 20 byte master secret if we use SHA-1 as the cryptographic hash function.

2.6. Previous Work on WTLS and SSL/TLS Performance Analysis

Levi and Savas have proposed an analytical performance model for WTLS handshake protocol in [30]. WTLS handshake performance was considered as three sub-problems which are client and server processing time, server queue delay and transmission time over the network. Three groups of cryptosystems with equal level of security using RSA and ECC were compared in the paper. The first group contains RSA 1024 bit, ECC 160p (prime), ECC 163k (Koblitz), ECC 163r (random) curves. The second group contains RSA 2048 bit, ECC 224p, ECC 233k, and ECC 233r curves. The last group contains RSA 3072 bit, ECC 256p, ECC 283k, and ECC 283r curves. The first two groups of the same security level contain the ECC curves and RSA key lengths that are offered by the WTLS standard [5]. The third group is composed of the curves that are not in the WTLS standard but offers the level of security that might be needed for today's WAP applications. Public key operations, namely encryption and signature verification with public key, decryption and signature generation with private key were implemented using state of the art techniques and timing measurements, were done to use in the performance model. Overall processing time is significantly lower when using ECC public key cryptosystems for both server authenticated and mutual authenticated WTLS Full Handshake. They model the server as an M/M/1 queue, so the queue delay is computed by using the well known formula $T_s r/(1-r)$, where T_s is the average server processing time and r is the utilization factor. The data transmission time is computed as $T_{data} = 8L/R$, where R is the data transmission rate in bits and L is the overall data size transmitted in bytes. The performance model proposed is successfully considering the necessary terms that may be a bottleneck of the WTLS handshake. Practically M/M/1 model is not well suited for computing the queue delay because the average processing time of each intermediate handshake messages must be considered separately (not the overall handshake processing time) and also the model does not include the extra networking time that may come out when using different types of data bearers.

Apostolopoulos et. al. take a close look at the SSL protocol, the ancestor of WTLS, with an eye on performance in [31] and [32]. They benchmark the performance of industry wide web servers Netscape and Apache, and quantify the overheads of different components of SSL protocol using SPECWeb96 [33] benchmark tool. They vary the degree of session reuse from 0 to 100 percent. SSL performance improves as the ratio of session reuse increases. Protocol overhead is analyzed associated with the SSL handshake protocol, encryption and authentication during data transfer. Performance analysis of SSL handshake protocol is affected by an increase in data volume due to additional data items and computational overhead for crypto functions. Server authenticated WTLS full handshake measurements were done using 512 bit, 768 bit and 1024 bit RSA public keys, ECDH key agreement is not in the analysis scope. They conclude the performance analysis with the finding that the crypto operation bottleneck comes from private key operation performed at the server for server authenticated WTLS handshake using RSA key exchange scheme.

Herwono and Liebhardt have done simulation-based performance measurements of the WTLS protocol in [34]. They simulate the relevant protocols WTP [17], WTLS [5] and WDP [18]. The benchmark software was coded using C/C++. Handshake durations for four types of full handshake and one optimized handshake were compared. The key exchange schemes considered are RSA, ECDH_ECDSA, ECDH_ECDSA (optimized), RSA_anon and ECDH_anon. The key length of RSA types was chosen 1024 bit and ECC key length was chosen 160 bit respectively. They vary the effective mean channel throughput from 1 kbit/s to 20 kbit/s. Handshake duration decreases up to a point as the network throughput increases. Increase in the network throughput does not affect the handshake duration after approximately 8-10 kbit/s. Systems with higher network throughputs have the crypto operations processing time as an upper bound for the handshake. On the other hand, the crypto operations processing time become negligible for network throughputs lower than 5 kbit/s. The transmission time is seriously a bottleneck for those types of systems.

Herwono and Liebhardt has a similar work that compares RSA_anon (2048 bit), ECDH_anon (224 bit), RSA_anon (1024 bit), and ECDH_anon (160 bit) in [35]. As an expected result, ECDH key agreement overall handshake durations are lower then RSA equivalent ones.

Krishna et. al. [36] analyze the performance and architectural impact of SSL in terms of throughput, number of processors, handshake frequency etc. Server public key used is RSA 512 bit and private key encryption used is 128 bit RC4. They consider three different cases for the SSL handshake. The first case is the SSL handshake followed by a very small data transfer. The second case is the SSL handshake followed by encrypted transfer of a huge size web page. The last case is the SSL handshake followed by 36 KB(determined average web page size) web page transfer. Results of the simulations show that the overall handshake duration is significantly higher for the first case which shows that the number of handshake operations is a key factor for SSL handshake performance.

Gupta et. al. present an estimation of the performance improvements that can be expected in SSL protocol by adding ECC support in [37]. They modify the OpenSSL [38] cryptographic library OpenSSL0.9.6.b to support ECDH, ECDSA and X.509 certificates with ECC parameters. The analytical model especially considers the handshake crypto latency and server crypto throughput however they are also aware of the extra delays due to message parsing, hashing and network latency. RSA (1024-2048 bit) and ECDH_ECDSA (163-193 bit) handshakes are compared in three cases. One of these three cases is a client talking to an Ultra 80 server simulating a wireless web scenario. They measure the performance of public key algorithms for RSA encrypt, verify, decrypt and sign, ECDSA verify and sign operations and use these values at their analytical model. The results show that 1024 bit RSA performs better than 163 bit ECC curve while 193 bit ECC curve is faster then 2048 bit RSA for server authenticated SSL handshake. ECDH key agreement is faster for both key sizes when mutual authenticated SSL handshake is performed.

3. WTLS PERFORMANCE EVALUATION

The performance evaluation model includes three main factors that may be bottleneck for the WTLS handshake protocol:

i. Processing time for all cryptographic operations and message parsing operations performed during the handshake
ii. Queue delay due to the load on the WAP gateway
iii. Transmission time of the handshake messages

There are also some other sources of delay like parallel processes at the server or client etc. that may effect the handshake duration but these are thought to be negligible when considering the above three factors.

Mutual Authenticated WTLS Full Handshake and Server Authenticated WTLS Full Handshake performance was considered separately and two different performance models were proposed although the models conceptually have the same properties.

The overall handshake duration is modeled as:

$$T_H = T_{PD} + T_{TD} + T_{QD} \qquad (3.1)$$

The notation used in Eq. (3.1) is defined in Table 3.1.

Symbol	Meaning
T_H	Overall handshake duration
T_{PD}	Total processing delay of handshake messages
T_{TD}	Transmission delay of handshake messages
T_{QD}	Server queue delay

Table 3.1 Overall handshake duration notations

Performance model includes 48 cases, 24 of which is Mutual Authenticated and the remaining 24 cases are Server Authenticated WTLS handshake. Three categories of different security level have been considered. Each category includes four groups of the same level of security. Cryptosystems that have the same level of security have been given in Table 2.2. Three of the four groups in a category are ECDH_ECDSA key exchange suites that use prime, Koblitz and random ECC curve parameters. The last group contains the RSA key exchange suites that offer the same level of security compared to the ECC curves in the same category. These categories and groups will be referred wherever necessary in the performance model and analysis. CA certificate public key parameter specifier and the corresponding client/server certificate public key parameter specifier uniquely specify a key exchange suite. Available parameter specifiers that will be used in the model are given in Table 3.2. Three categories and corresponding four groups of each category are given in Table 3.3.

Public Key Type	Parameter Specifier
ECC	1 (wtls7_160)
ECC	2 (nist163k)
ECC	3 (nist163r)
RSA	20 (RSA1024)
ECC	5 (nist224p)
ECC	6 (nist233k)
ECC	7 (nist233r)
RSA	21 (RSA2048)
ECC	8 (nist256p)
ECC	9 (nist283k)
ECC	10 (nist283r)
RSA	22 (RSA3072)

Table 3.2 Public key parameter specifiers

Category #	Group #	CA Certificate parameter specifier	Client/Server Certificate parameter specifier
1	1	1 (wtls7_160)	1 (wtls7_160)
	2	2 (nist163k)	2 (nist163k)
	3	3 (nist163r)	3 (nist163r)
	4	20 (RSA1024)	20 (RSA1024)
2	1	5 (nist224p)	1 (wtls7_160)
		5 (nist224p)	5 (nist224p)
	2	6 (nist233k)	2 (nist163k)
		6 (nist233k)	6 (nist233k)
	3	7 (nist233r)	3 (nist163r)
		7 (nist233r)	7 (nist233r)
	4	21 (RSA2048)	20 (RSA1024)
		21 (RSA2048)	21 (RSA2048)
3	1	8 (nist256p)	1 (wtls7_160)
		8 (nist256p)	5 (nist224p)
		8 (nist256p)	8 (nist256p)
	2	9 (nist283k)	2 (nist163k)
		9 (nist283k)	6 (nist233k)
		9 (nist283k)	9 (nist283k)
	3	10 (nist283r)	3 (nist163r)
		10 (nist283r)	7 (nist233r)
		10 (nist283r)	10 (nist283r)
	4	22 (RSA3072)	20 (RSA1024)
		22 (RSA3072)	21 (RSA2048)
		22 (RSA3072)	22 (RSA3072)

Table 3.3 Performance model categories and groups

The following three sections give the performance model for Mutual Authenticated and Server Authenticated WTLS Full Handshake.

3.1. Processing Time Model

The performance model does not exclude the handshake messages that do not contain any public key operation. Processing delays for both generating the message at the client or server and processing of the message at the opposite side are considered. Some of the handshake messages' processing delays are expected to be almost independent of the key exchange suite selected because of using the same algorithm for any of the key exchange suite or messages do not contain any cryptography related operations at all.

Depending on the key exchange suite selected, applicable handshake messages will be different and processing time for those not applicable messages will be accepted as 0 in the model to be able to give a common formula for RSA and ECDH_ECDSA key exchange suites. Regardless of which one of the performance model categories or groups are used, the processing time model is only based on whether it is mutual or server authenticated WTLS handshake.

3.1.1. Mutual Authenticated WTLS Full Handshake

Mutual Authenticated WTLS Full Handshake time interval notations for each handshake message are given in Table 3.4.

Symbol	Meaning
$T_{M\_C\_CH}$	Client processing time for generating the Client.Hello message
$T_{M\_C\_SH}$	Client processing time for processing Server.Hello message
$T_{M\_C\_SCERT}$	Client processing time for processing the server certificate
$T_{M\_C\_CERTREQ}$	Client processing time for processing the CertificateRequest message
$T_{M\_C\_SHD}$	Client processing time for processing the ServerHelloDone message
$T_{M\_C\_CCERT}$	Client processing time for generating the Client.Certificate message
$T_{M\_C\_CKX}$	Client processing time for generating the ClientKeyExchange message (message is sent iff RSA key exchange suite is used)
$T_{M\_C\_CERTVRFY}$	Client processing time for generating the CertificateVerify message (applicable if RSA key exchange suite is used)
$T_{M\_C\_CCCS}$	Client processing time for generating the Client ChangeCipherSpec message
$T_{M\_C\_CFIN}$	Client processing time for generating the Client.Finished message
$T_{M\_C\_SCCS}$	Client processing time for processing the Server ChangeCipherSpec message
$T_{M\_C\_SFIN}$	Client processing time for processing the Server.Finished message
$T_{M\_C\_ECDH}$	Client processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)
$T_{M\_C\_MS}$	Client processing time for computing the master secret from the premaster secret
$T_{M\_C\_RSAENC}$	Client processing time for encryption operation using the Server's public key (applicable if RSA key exchange suite is used)
T_{PD_C}	Overall processing time for the client side
$T_{M\_S\_CH}$	Server processing time for processing the Client.Hello message

Symbol	Meaning
$T_{M\_S\_SH}$	Server processing time for generating the Server.Hello message
$T_{M\_S\_SCERT}$	Server processing time for generating the Server.Certificate message
$T_{M\_S\_CERTREQ}$	Server processing time for generating the CertificateRequest message
$T_{M\_S\_SHD}$	Server processing time for generating the ServerHelloDone message
$T_{M\_S\_CCERT}$	Server processing time for processing the client certificate
$T_{M\_S\_CKX}$	Server processing time for processing the ClientKeyExchange message
$T_{M\_S\_CERTVRFY}$	Server processing time for processing the CertificateVerify message (applicable if RSA key exchange suite is used)
$T_{M\_S\_CCCS}$	Server processing time for processing the Client ChangeCipherSpec message
$T_{M\_S\_CFIN}$	Server processing time for processing the Client.Finished message
$T_{M\_S\_SCCS}$	Server processing time for generating the Server ChangeCipherSpec message
$T_{M\_S\_SFIN}$	Server processing time for generating the Server.Finished message
$T_{M\_S\_ECDH}$	Server processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)
$T_{M\_S\_MS}$	Server processing time for computing the master secret from the premaster secret
$T_{M\_S\_RSADEC}$	Server processing time for decryption operation using its own private key (applicable if RSA key exchange suite is used)
T_{PD_S}	Overall processing time for the server side

Table 3.4 Mutual authenticated WTLS full handshake performance model notations

Mutual Authenticated WTLS Full Handshake message flow is given in Table 3.5.

Client Processing Time	Message(Client)		Message(Server)	Server Processing Time
$T_{M\_C\_CH}$	ClientHello	→		$T_{M\_S\_CH}$
$T_{M\_C\_SH}$		←	ServerHello	$T_{M\_S\_SH}$
$T_{M\_C\_SCERT}$		←	Certificate	$T_{M\_S\_SCERT}$
$T_{M\_C\_CERTREQ}$		←	CertificateReq	$T_{M\_S\_CERTREQ}$
$T_{M\_C\_SHD}$		←	ServerHelloDone	$T_{M\_S\_SHD}$
$T_{M\_C\_CCERT}$	Certificate	→		$T_{M\_S\_CCERT}$
$T_{M\_C\_CKX}$	ClientKeyExchange (if RSA)	→		$T_{M\_S\_CKX}$
$T_{M\_C\_CERTVRFY}$	CertificateVerify (if RSA)	→		$T_{M\_S\_CERTVRFY}$
$T_{M\_C\_CCCS}$	ChangeCipherSpec	→		$T_{M\_S\_CCCS}$
$T_{M\_C\_CFIN}$	Finished	→		$T_{M\_S\_CFIN}$
$T_{M\_C\_SCCS}$		←	ChangeCipherSpec	$T_{M\_S\_SCCS}$
$T_{M\_C\_SFIN}$		←	Finished	$T_{M\_S\_SFIN}$

Table 3.5 Mutual authenticated WTLS full handshake message flow

Using the notation given up to here, the total processing delay of handshake messages for Mutual Authenticated WTLS Handshake Protocol T_{PD} can be given as the sum of client side processing time and the server side processing time.

Client side processing time T_{PD_C};

$$T_{PD\_C} = T_{M\_C\_CH} + T_{M\_C\_SH} + T_{M\_C\_SCERT} + T_{M\_C\_CERTREQ} + T_{M\_C\_SHD} + T_{M\_C\_CCERT} + T_{M\_C\_CKX} + T_{M\_C\_CERTVRFY} + T_{M\_C\_CCCS} + T_{M\_C\_CFIN} + T_{M\_C\_SCCS} + T_{M\_C\_SFIN} + T_{M\_S\_CH} \quad (3.2)$$

Server side processing time T_{PD_S};

$$T_{PD\_S} = T_{M\_S\_SH} + T_{M\_S\_SCERT} + T_{M\_S\_CERTREQ} + T_{M\_S\_SHD} + T_{M\_S\_CCERT} + T_{M\_S\_CKX} + T_{M\_S\_CERTVRFY} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN} + T_{M\_S\_SCCS} + T_{M\_S\_SFIN} \quad (3.3)$$

Overall processing time T_{PD};

$$T_{PD} = T_{PD\_S} + T_{PD\_C} \quad (3.4)$$

Although the model includes 24 additive components of the handshake processing delays, some of these processing delays are expected to be more affective on the total processing time. Those messages and operations that need special care are defined below.

$T_{M\_C\_SCERT}$: This is the client processing time for verifying the received server certificate. Certificate verification operation is not bounded with the certificate signature verification using the CA public key. Correctness and validity of the information in the certificate must also be checked for. Subject of the certificate must match with the previously known gateway identity, the issuer of the certificate must be the same with the CA certificate's subject, the certificate must not be used for any time interval that it is not valid etc.

$T_{M\_S\_CCERT}$: This is the server processing time for verifying the received client certificate. Certificate verification process is the same as described for $T_{M\_C\_SCERT}$.

$T_{M\_C\_CKX}$: This message is sent only when the RSA key exchange scheme is used. The premaster secret is set during preparation of this message so the master secret is computed following this message regardless of the key exchange suite selected. If RSA key exchange suite is used, premaster secret is encrypted using the gateway's public key derived from the gateway certificate($T_{M\_C\_RSAENC}$).ECDH operation is performed to compute the premaster secret ($T_{M\_C\_ECDH}$) if ECDH_ECDSA key exchange is used. After computing the premaster secret, the client computes the master secret ($T_{M\_C\_MS}$) as described in Section 2.5.

$T_{M\_S\_CKX}$: If ECDH_ECDSA key exchange suite is used, all the computation is identical with described for the client side $T_{M\_C\_CKX}$. If RSA key exchange suite is used, the server must perform a private key operation to decrypt the premaster secret which was encrypted by the client using gateway's RSA public key.

$T_{M\_C\_CERTVRFY}$: Only performed when RSA key exchange suite is used. CertificateVerify message's objective is to explicitly verify the certificate of the client. The client generates a hash value of all of the handshake messages prior to this one and signs the hash value using its RSA private key with signing capability.

$T_{M\_S\_CERTVRFY}$: This is the server processing time to verify the signature sent by the client to the gateway in the CertificateVerify message. This operation includes a public key signature verification operation.

$T_{M\_C\_CFIN}$, $T_{M\_S\_CFIN}$, $T_{M\_C\_SFIN}$, $T_{M\_S\_SFIN}$: These are the processing times related to the generation and verification of the finished messages. Similar to the CertificateVerify message, finished messages are also performed on the hash value of all handshake messages prior to the corresponding message together with a finished label and master secret. The $PRF(.)$ described in Section 2.5 is used to generate and verify the finish messages.

Typically due to the low computational power of the mobile clients, the cryptographic operations performed at the client side are also expected to be much more point of interest when considering the overall processing delay for Mutual Authenticated WTLS Handshake. Especially $T_{M\_C\_CERTVRFY}$ is willing to be on the top of the list because of the high cost private key operation performed on the resource constrained client environment.

3.1.2. Server Authenticated WTLS Full Handshake

Server Authenticated WTLS Full Handshake time interval notations for each handshake message are given in Table 3.6.

Symbol	Meaning
$T_{S\_C\_CH}$	Average client processing time for generating the Client.Hello message
$T_{S\_C\_SH}$	Average client processing time for processing Server.Hello message
$T_{S\_C\_SCERT}$	Average client processing time for processing the server certificate
$T_{S\_C\_CERTREQ}$	Average client processing time for processing the CertificateRequest message
$T_{S\_C\_SHD}$	Average client processing time for processing the ServerHelloDone message
$T_{S\_C\_CCERT}$	Average client processing time for generating the Client.Certificate message
$T_{S\_C\_CKX}$	Average client processing time for generating the ClientKeyExchange message
$T_{S\_C\_CCCS}$	Average client processing time for generating the Client ChangeCipherSpec message
$T_{S\_C\_CFIN}$	Average client processing time for generating the Client.Finished message
$T_{S\_C\_SCCS}$	Average client processing time for processing the Server ChangeCipherSpec message
$T_{S\_C\_SFIN}$	Average client processing time for processing the Server.Finished message
$T_{S\_C\_ECDH}$	Average client processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)
$T_{S\_C\_MS}$	Average client processing time for computing the master secret from the premaster secret
$T_{S\_C\_RSAENC}$	Average client processing time for encryption operation using the Server's public key (applicable if RSA key exchange suite is used)
$T_{S\_S\_CH}$	Average server processing time for processing the Client.Hello message
$T_{S\_S\_SH}$	Average server processing time for generating the Server.Hello

Symbol	Meaning
	message
$T_{S\_S\_SCERT}$	Average server processing time for generating the Server.Certificate message
$T_{S\_S\_CERTREQ}$	Average server processing time for generating the CertificateRequest message
$T_{S\_S\_SHD}$	Average server processing time for generating the ServerHelloDone message
$T_{S\_S\_CCERT}$	Average server processing time for processing the client certificate
$T_{S\_S\_CKX}$	Average server processing time for processing the ClientKeyExchange message
$T_{S\_S\_CCCS}$	Average server processing time for processing the Client ChangeCipherSpec message
$T_{S\_S\_CFIN}$	Average server processing time for processing the Client.Finished message
$T_{S\_S\_SCCS}$	Average server processing time for generating the Server ChangeCipherSpec message
$T_{S\_S\_SFIN}$	Average server processing time for generating the Server.Finished message
$T_{S\_S\_ECDH}$	Average server processing time for ECDH operation (applicable if ECDH_ECDSA key exchange suite is used)
$T_{S\_S\_MS}$	Average server processing time for computing the master secret from the premaster secret
$T_{S\_S\_RSADEC}$	Average server processing time for decryption operation using its own private key (applicable if RSA key exchange suite is used)

Table 3.6 Server authenticated WTLS full handshake performance model notations

Server Authenticated WTLS Full Handshake message flow is given in Table 3.7.

Client Processing Time	Message(Client)		Message(Server)	Server Processing Time
$T_{S\_C\_CH}$	ClientHello	→		$T_{S\_S\_CH}$
$T_{S\_C\_SH}$		←	ServerHello	$T_{S\_S\_SH}$
$T_{S\_C\_SCERT}$		←	Certificate	$T_{S\_S\_SCERT}$
$T_{S\_C\_CERTREQ}$		←	CertificateReq	$T_{S\_S\_CERTREQ}$
$T_{S\_C\_SHD}$		←	ServerHelloDone	$T_{S\_S\_SHD}$
$T_{S\_C\_CCERT}$	Certificate (includes no certificate)	→		$T_{S\_S\_CCERT}$
$T_{S\_C\_CKX}$	ClientKeyExchange	→		$T_{S\_S\_CKX}$
$T_{S\_C\_CCCS}$	ChangeCipherSpec	→		$T_{S\_S\_CCCS}$
$T_{S\_C\_CFIN}$	Finished	→		$T_{S\_S\_CFIN}$
$T_{S\_C\_SCCS}$		←	ChangeCipherSpec	$T_{S\_S\_SCCS}$
$T_{S\_C\_SFIN}$		←	Finished	$T_{S\_S\_SFIN}$

Table 3.7 Server authenticated WTLS full handshake message flow

The total processing delay of handshake messages for Server Authenticated WTLS Handshake Protocol T_{PD} can be given as the sum of client side processing time and the server side processing time.

Client side processing time T_{PD_C};

$$T_{PD\_C} = T_{S\_C\_CH} + T_{S\_C\_SH} + T_{S\_C\_SCERT} + T_{S\_C\_CERTREQ} + T_{S\_C\_SHD} + T_{S\_C\_CCERT} + T_{S\_C\_CKX} + T_{S\_C\_CCCS} + T_{S\_C\_CFIN} + T_{S\_C\_SCCS} + T_{S\_C\_SFIN} \qquad (3.5)$$

Server side processing time T_{PD_S};

$$T_{PD\_S} = T_{S\_S\_CH} + T_{S\_S\_SH} + T_{S\_S\_SCERT} + T_{S\_S\_CERTREQ} + T_{S\_S\_SHD} + T_{S\_S\_CCERT} + T_{S\_S\_CKX} + T_{S\_S\_CCCS} + T_{S\_S\_CFIN} + T_{S\_S\_SCCS} + T_{S\_S\_SFIN} \tag{3.6}$$

Overall processing time T_{PD};

$$T_{PD} = T_{PD\_S} + T_{PD\_C} \tag{3.7}$$

The model includes 22 additive components of the handshake processing delays. Some of these processing delays are expected to be more affective on the total processing time.

$T_{S\_C\_SCERT}$: This is the client processing time for verifying the received server certificate. Certificate verification operation is not bounded with the certificate signature verification using the CA public key. Correctness and validity of the information in the certificate must also be checked for. Subject of the certificate must match with the previously known gateway identity, the issuer of the certificate must be the same with the CA certificate's subject, the certificate must not be used for any time interval that it is not valid etc.

$T_{S\_C\_CKX}$: This message is sent for both RSA and ECDH_ECDSA key exchange suites. The premaster secret is set during preparation of this message so the master secret is computed following this message regardless of the key exchange suite selected. If RSA key exchange suite is used, premaster secret is encrypted using the gateway's public key derived from the gateway certificate ($T_{S\_C\_RSAENC}$). ECDH operation is performed to compute the premaster secret ($T_{S\_C\_ECDH}$) If ECDH_ECDSA key exchange suite is used. After computing the premaster secret, the client computes the master secret ($T_{S\_C\_MS}$) as described in Section 2.5.

$T_{S\_S\_CKX}$: If ECDH_ECDSA key exchange suite is used, all the computation is identical with described for the client side $T_{S\_C\_CKX}$. If RSA key exchange suite is used, the server must perform a private key operation to decrypt the premaster secret which was encrypted by the client using gateway’s RSA public key.

$T_{S\_C\_CFIN}$, $T_{S\_S\_CFIN}$, $T_{S\_C\_SFIN}$, $T_{S\_S\_SFIN}$: These are the processing times related to the generation and verification of the finished messages. Finished messages are generated by using the hash value of all handshake messages prior to the corresponding message together with a finished label and master secret. The $PRF(.)$ described in Section 2.5 is used to generate and verify the finish messages.

Typically due to the low computational power of the mobile clients, the cryptographic operations performed at the client side are also expected to be much more point of interest when considering the overall processing delay for Server Authenticated WTLS Handshake.

3.2. Queue Delay Model

Queue delay is the time that a WTLS handshake message has to wait for the service. Generally the queue delay depends on the number of servers, average service time, and arrival rate. The server is modeled as an M/G/1 queue with the assumptions given below:

$$\left\{\begin{array}{lll} M & (memoryless) & : Poisson \quad arrival \quad process, \quad arrival \quad rate \quad l \\ G & (general) & : General \quad service \quad time \quad distribution, \quad mean \quad E[x] = \overline{X} = 1/m \\ 1 & & : Single \quad server, \quad load \quad r = l\overline{X} \quad (in \quad a \quad stable \quad queue \quad one \quad has \quad r < 1) \end{array}\right.$$

A simple representation of the queue delay components is given in Figure 3.1.

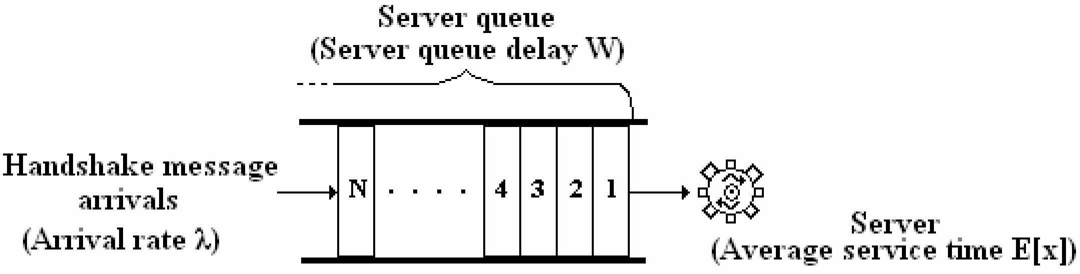


Figure 3.1 Server queue delay components

Server waiting time for a handshake message is the sum of mean time needed to serve the customers ahead in the queue and unfinished work in the server. Eq. (3.8) gives a general representation of the server waiting time. [39]

$$E[W] = \underbrace{\underbrace{E[Nq]}_{\text{number of waiting customers}} \times \underbrace{E[X]}_{\text{mean service time}}}_{\text{mean time needed to serve the customers ahead in the queue}} + \underbrace{E[R]}_{\text{unfinished work in the server}} \qquad (3.8)$$

We can express the mean queue length $E[Nq]$ in terms of the waiting time $E[W]$ as given in Eq. (3.9).

$$E[Nq] = \lambda E[W] \qquad (3.9)$$

Substituting Eq. (3.9)in Eq. (3.8);

$$E[W] = \lambda E[W] \times E[X] + E[R]$$

$$E[W](1 - \lambda E[x]) = E[R]$$

We find:

$$E[W] = \frac{E[R]}{(1 - \lambda E[x])} = \frac{E[R]}{(1 - \rho)} \qquad (3.10)$$

Therefore, it remains to determine the unfinished work in the server $E[R]$, to represent the server waiting time $E[W]$.

The residual service time $E[R]$ can be deduced by using Figure 3.2 which represents the evolution of the unfinished work in the server, as a function of time.

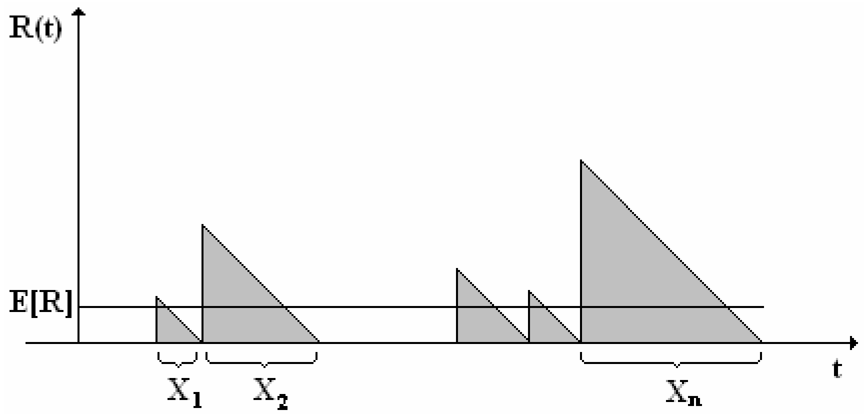


Figure 3.2 Evolution of the unfinished work in the server

Consider a long interval of time t. The average value of the sawtooth curve in Figure 3.2 can be calculated by dividing the sum of the areas of the triangles by the length of the interval.

$$E[R]=\frac{1}{t}\int_{o}^{t}R(t')dt'=\frac{1}{t}\sum_{i=1}^{n}\frac{1}{2}X_i^2=\underbrace{\frac{n}{t}}_{\lambda}\times\underbrace{\frac{1}{n}\times\sum_{i=1}^{n}\frac{1}{2}X_i^2}_{\frac{1}{2}E[X^2]}=\frac{\lambda E[X^2]}{2} \qquad (3.11)$$

Substituting (3.11) in (3.10), we can derive the final function for the server waiting time as given below.

$$T_{QD}=E[W]\;=\;\frac{\lambda E[X^2]}{2(1-\rho)} \qquad (3.12)$$

Where;

l	:	Arrival rate
$E[X^2] = \overline{X^2}$	:	Second moment of service time
$r = lE[X]$	:	Utilization (load) of the server
$E[X] = \overline{X}$	:	Average service time

Eq. (3.12) which is in fact the P-K (Pollaczek-Khinchin) formula [39], will be used to model the queue delay. Regarding which type of client/server certificate and CA certificate has been used, average service time $\overline{X}$ and second moment of service time $\overline{X^2}$ changes. I will consider these values for different cases and interpret the queuing delay as a function of arrival rate l .

Queue delay is affected by only the messages which the server (WAP Gateway) accepts. This is because the messages waiting to be served in the queue are the reason for queue delay and those messages are only the incoming handshake messages coming form the clients.

Processing time intervals that need to be considered for a mutual authenticated WTLS handshake are given in Table 3.8.

Message(Client)		Message(Server)	Server Processing Time
ClientHello	→		$T_{M\_S\_CH}$
	←	ServerHello	
	←	Certificate	
	←	CertificateReq	
	←	ServerHelloDone	
Certificate	→		$T_{M\_S\_CCERT}$
ClientKeyExchange (if RSA)	→		$T_{M\_S\_CKX}$
CertificateVerify (if RSA)	→		$T_{M\_S\_CERTVRFY}$
ChangeCipherSpec	→		$T_{M\_S\_CCCS}$
Finished	→		$T_{M\_S\_CFIN}$
	←	ChangeCipherSpec	
	←	Finished	

Table 3.8 Mutual authenticated handshake message flow for queue delay analysis

Processing time intervals that need to be considered for a server authenticated WTLS handshake are given in Table 3.9.

Message(Client)		Message(Server)	Server Processing Time
ClientHello	→		$T_{S\_S\_CH}$
	←	ServerHello	
	←	Certificate	
	←	CertificateReq	
	←	ServerHelloDone	
Certificate (includes no certificate)	→		$T_{S\_S\_CCERT}$
ClientKeyExchange	→		$T_{S\_S\_CKX}$
ChangeCipherSpec	→		$T_{S\_S\_CCCS}$
Finished	→		$T_{S\_S\_CFIN}$
	←	ChangeCipherSpec	
	←	Finished	

Table 3.9 Server authenticated handshake message flow for queue delay analysis

Average service time $\overline{X}$ and the second moment of the service time $\overline{X^2}$ will be presented as a function of the available handshake messages processing times for both mutual authenticated and server authenticated WTLS full handshake.

For the mutual authenticated case,

$$\overline{X} = \frac{\left(T_{M\_S\_CH} + T_{M\_S\_CCERT} + T_{M\_S\_CKX} + T_{M\_S\_CERTVRFY} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{(number \quad of \quad applicable \quad messages)} \tag{3.13}$$

$$\overline{X^2} = \frac{\left(T^2{}_{M\_S\_CH} + T^2{}_{M\_S\_CCERT} + T^2{}_{M\_S\_CKX} + T^2{}_{M\_S\_CERTVRFY} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{(number \quad of \quad applicable \quad messages)} \tag{3.14}$$

$$(number \quad of \quad applicable \quad messages) = \begin{cases} 6 & if \qquad RSA \\ 4 & if \qquad ECDH\_ECDSA \end{cases} \tag{3.15}$$

For the server authenticated case,

$$\overline{X} = \frac{\left(T_{S\_S\_CH} + T_{S\_S\_CCERT} + T_{S\_S\_CKX} + T_{S\_S\_CCCS} + T_{S\_S\_CFIN}\right)}{(number \quad of \quad applicable \quad messages)} \qquad (3.16)$$

$$\overline{X^2} = \frac{\left(T^2{}_{S\_S\_CH} + T^2{}_{S\_S\_CCERT} + T^2{}_{S\_S\_CKX} + T^2{}_{S\_S\_CCCS} + T^2{}_{S\_S\_CFIN}\right)}{(number \quad of \quad applicable \quad messages)} \qquad (3.17)$$

$$(number \quad of \quad applicable \quad messages) = \begin{cases} 5 & if \quad RSA \\ 5 & if \quad ECDH\_ECDSA \end{cases} \qquad (3.18)$$

The average service times will be presented in the terms of three possible service times that may be used for a given group. Handshake messages processing times for each one of the three possible key exchange suites specified by public key parameters specifiers given in Table 3.3 will affect the average service time for the corresponding handshake message depending on its ratio p .

Queue delay model will be proposed for category#3. The same logic applies for the other categories with a slight change in the notation. Further analysis needs to introduce new formulations:

$$T_X = p_{c,a}T_{X,c,a} + p_{c,b}T_{X,c,b} + p_{c,c}T_{X,c,c} \qquad (3.19)$$

$$p_{c,a} + p_{c,b} + p_{c,c} = 1 \qquad (3.20)$$

Meanings of the symbols used in the above formulas are given in Table 3.10.

Symbol	Meaning
T_X	Average service time for any of the applicable handshake messages for a given group
$T_{X,c,a}$	Average service time for any of the applicable handshake messages for the key exchange suite with the smallest public key size in a group.
$T_{X,c,b}$	Average service time for any of the applicable handshake messages for the key exchange suite with the intermediate public key size in a group.
$T_{X,c,c}$	Average service time for any of the applicable handshake messages for the key exchange suite with the highest public key size in a group.
$p_{c,a}$	Ratio of the key exchange suite with the smallest public key size in a group where 'c' represents the CA certificate public key parameter specifier, and 'a' represents the client/server certificate public key parameter specifier.
$p_{c,b}$	Ratio of the key exchange suite with the intermediate public key size in a group where 'c' represents the CA certificate public key parameter specifier, and 'b' represents the client/server certificate public key parameter specifier.
$p_{c,c}$	Ratio of the key exchange suite with the highest public key size in a group where 'c' represents the CA certificate public key parameter specifier and, client/server certificate public key parameter specifier.

Table 3.10 Average service time formula notations

All the notation and formulas have been defined to present the queue delay modeling. The next step will be to define the individual queue delay models for all groups regarding mutual authenticated and server authenticated WTLS full handshake.

3.2.1. Category#3-Group#1 – Mutual Authentication

Queue delay model for category#3-group#1 mutual authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Client/Server certificate public key parameter specifier	Expected Ratio
8 (nist256p)	1 (wtls7_160)	$p_{8,1}$
8 (nist256p)	5 (nist224p)	$p_{8,5}$
8 (nist256p)	8 (nist256p)	$p_{8,8}$

Table 3.11 Category#3-Group#1

Using Eq. (3.13), Eq. (3.14) and Eq. (3.15), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{M\_S\_CH} + T_{M\_S\_CCERT} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{4} \qquad (3.21)$$

$$\overline{X^2} = \frac{\left(T^2{}_{M\_S\_CH} + T^2{}_{M\_S\_CCERT} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{4} \qquad (3.22)$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{M\_S\_CH} = p_{8,1}T_{M\_S\_CH,8,1} + p_{8,5}T_{M\_S\_CH,8,5} + p_{8,8}T_{M\_S\_CH,8,8} \qquad (3.23)$$

$$T_{M\_S\_CCERT} = p_{8,1}T_{M\_S\_CCERT,8,1} + p_{8,5}T_{M\_S\_CCERT,8,5} + p_{8,8}T_{M\_S\_CCERT,8,8} \qquad (3.24)$$

$$T_{M\_S\_CCCS} = p_{8,1}T_{M\_S\_CCCS,8,1} + p_{8,5}T_{M\_S\_CCCS,8,5} + p_{8,8}T_{M\_S\_CCCS,8,8} \qquad (3.25)$$

$$T_{M\_S\_CFIN} = p_{8,1}T_{M\_S\_CFIN,8,1} + p_{8,5}T_{M\_S\_CFIN,8,5} + p_{8,8}T_{M\_S\_CFIN,8,8} \qquad (3.26)$$

$$p_{8,1} + p_{8,5} + p_{8,8} = 1 \qquad (3.27)$$

T_{QD} is computed using Eq. (3.12).

3.2.2. Category#3-Group#1 – Server Authentication

Queue delay model for category#3-group#1 server authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Server certificate public key parameter specifier	Expected Ratio
8 (nist256p)	1 (wtls7_160)	$p_{8,1}$
8 (nist256p)	5 (nist224p)	$p_{8,5}$
8 (nist256p)	8 (nist256p)	$p_{8,8}$

Table 3.12 Category#3-Group#1

Using Eq. (3.16), Eq. (3.17) and Eq. (3.18), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{S\_S\_CH} + T_{S\_S\_CCERT} + T_{S\_S\_CKX} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{5} \qquad (\,3.28\,)$$

$$\overline{X^2} = \frac{\left(T^2{}_{S\_S\_CH} + T^2{}_{S\_S\_CCERT} + T^2{}_{S\_S\_CKX} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{5} \qquad (\,3.29\,)$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{S\_S\_CH} = p_{8,1}T_{S\_S\_CH,8,1} + p_{8,5}T_{S\_S\_CH,8,5} + p_{8,8}T_{S\_S\_CH,8,8} \qquad (\,3.30\,)$$

$$T_{S\_S\_CCERT} = p_{8,1}T_{S\_S\_CCERT,8,1} + p_{8,5}T_{S\_S\_CCERT,8,5} + p_{8,8}T_{S\_S\_CCERT,8,8} \qquad (\,3.31\,)$$

$$T_{S\_S\_CKX} = p_{8,1}T_{S\_S\_CKX,8,1} + p_{8,5}T_{S\_S\_CKX,8,5} + p_{8,8}T_{S\_S\_CKX,8,8} \qquad (\,3.32\,)$$

$$T_{S\_S\_CCCS} = p_{8,1}T_{S\_S\_CCCS,8,1} + p_{8,5}T_{S\_S\_CCCS,8,5} + p_{8,8}T_{S\_S\_CCCS,8,8} \qquad (\,3.33\,)$$

$$T_{S\_S\_CFIN} = p_{8,1}T_{S\_S\_CFIN,8,1} + p_{8,5}T_{S\_S\_CFIN,8,5} + p_{8,8}T_{S\_S\_CFIN,8,8} \tag{3.34}$$

$$p_{8,1} + p_{8,5} + p_{8,8} = 1 \tag{3.35}$$

T_{QD} is computed using Eq. (3.12).

3.2.3. Category#3-Group#2 – Mutual Authentication

Queue delay model for category#3-group#2 mutual authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Client/Server certificate public key parameter specifier	Expected Ratio
9 (nist283k)	2 (nist163k)	$p_{9,2}$
9 (nist283k)	6 (nist233k)	$p_{9,6}$
9 (nist283k)	9 (nist283k)	$p_{9,9}$

Table 3.13 Category#3-Group#2

Using Eq. (3.13), Eq. (3.14) and Eq. (3.15), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{M\_S\_CH} + T_{M\_S\_CCERT} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{4} \tag{3.36}$$

$$\overline{X^2} = \frac{\left(T^2{}_{M\_S\_CH} + T^2{}_{M\_S\_CCERT} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{4} \tag{3.37}$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{M\_S\_CH} = p_{9,2}T_{M\_S\_CH,9,2} + p_{9,6}T_{M\_S\_CH,9,6} + p_{9,9}T_{M\_S\_CH,9,9} \tag{3.38}$$

$$T_{M\_S\_CCERT} = p_{9,2}T_{M\_S\_CCERT,9,2} + p_{9,6}T_{M\_S\_CCERT,9,6} + p_{9,9}T_{M\_S\_CCERT,9,9} \quad (3.39)$$

$$T_{M\_S\_CCCS} = p_{9,2}T_{M\_S\_CCCS,9,2} + p_{9,6}T_{M\_S\_CCCS,9,6} + p_{9,9}T_{M\_S\_CCCS,9,9} \quad (3.40)$$

$$T_{M\_S\_CFIN} = p_{9,2}T_{M\_S\_CFIN,9,2} + p_{9,6}T_{M\_S\_CFIN,9,6} + p_{9,9}T_{M\_S\_CFIN,9,9} \quad (3.41)$$

$$p_{9,2} + p_{9,6} + p_{9,9} = 1 \quad (3.42)$$

T_{QD} is computed using Eq. (3.12).

3.2.4. Category#3-Group#2 – Server Authentication

Queue delay model for category#3-group#2 server authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Server certificate public key parameter specifier	Expected Ratio
9 (nist283k)	2 (nist163k)	$p_{9,2}$
9 (nist283k)	6 (nist233k)	$p_{9,6}$
9 (nist283k)	9 (nist283k)	$p_{9,9}$

Table 3.14 Category#3-Group#2

Using Eq. (3.16), Eq. (3.17) and Eq. (3.18), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{S\_S\_CH} + T_{S\_S\_CCERT} + T_{S\_S\_CKX} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{5} \quad (3.43)$$

$$\overline{X^2} = \frac{\left(T^2{}_{S\_S\_CH} + T^2{}_{S\_S\_CCERT} + T^2{}_{S\_S\_CKX} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{5} \tag{3.44}$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{S\_S\_CH} = p_{9,2}T_{S\_S\_CH,9,2} + p_{9,6}T_{S\_S\_CH,9,6} + p_{9,9}T_{S\_S\_CH,9,9} \tag{3.45}$$

$$T_{S\_S\_CCERT} = p_{9,2}T_{S\_S\_CCERT,9,2} + p_{9,6}T_{S\_S\_CCERT,9,6} + p_{9,9}T_{S\_S\_CCERT,9,9} \tag{3.46}$$

$$T_{S\_S\_CKX} = p_{9,2}T_{S\_S\_CKX,9,2} + p_{9,6}T_{S\_S\_CKX,9,6} + p_{9,9}T_{S\_S\_CKX,9,9} \tag{3.47}$$

$$T_{S\_S\_CCCS} = p_{9,2}T_{S\_S\_CCCS,9,2} + p_{9,6}T_{S\_S\_CCCS,9,6} + p_{9,9}T_{S\_S\_CCCS,9,9} \tag{3.48}$$

$$T_{S\_S\_CFIN} = p_{9,2}T_{S\_S\_CFIN,9,2} + p_{9,6}T_{S\_S\_CFIN,9,6} + p_{9,9}T_{S\_S\_CFIN,9,9} \tag{3.49}$$

$$p_{9,2} + p_{9,6} + p_{9,9} = 1 \tag{3.50}$$

T_{QD} is computed using Eq. (3.12).

3.2.5. Category#3-Group#3 – Mutual Authentication

Queue delay model for category#3-group#3 mutual authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Client/Server certificate public key parameter specifier	Expected Ratio
10 (nist283r)	3 (nist163r)	$p_{10,3}$
10 (nist283r)	7 (nist233r)	$p_{10,7}$
10 (nist283r)	10 (nist283r)	$p_{10,10}$

Table 3.15 Category#3-Group#3

Using Eq. (3.13), Eq. (3.14) and Eq. (3.15), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{M\_S\_CH} + T_{M\_S\_CCERT} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{4} \qquad (3.51)$$

$$\overline{X^2} = \frac{\left(T^2{}_{M\_S\_CH} + T^2{}_{M\_S\_CCERT} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{4} \qquad (3.52)$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{M\_S\_CH} = p_{10,3}T_{M\_S\_CH,10,3} + p_{10,7}T_{M\_S\_CH,10,7} + p_{10,10}T_{M\_S\_CH,10,10} \qquad (3.53)$$

$$T_{M\_S\_CCERT} = p_{10,3}T_{M\_S\_CCERT,10,3} + p_{10,7}T_{M\_S\_CCERT,10,7} + p_{10,10}T_{M\_S\_CCERT,10,10} \qquad (3.54)$$

$$T_{M\_S\_CCCS} = p_{10,3}T_{M\_S\_CCCS,10,3} + p_{10,7}T_{M\_S\_CCCS,10,7} + p_{10,10}T_{M\_S\_CCCS,10,10} \qquad (3.55)$$

$$T_{M\_S\_CFIN} = p_{10,3}T_{M\_S\_CFIN,10,3} + p_{10,7}T_{M\_S\_CFIN,10,7} + p_{10,10}T_{M\_S\_CFIN,10,10} \qquad (3.56)$$

$$p_{10,3} + p_{10,7} + p_{10,10} = 1 \qquad (3.57)$$

T_{QD} is computed using Eq. (3.12).

3.2.6. Category#3-Group#3 – Server Authentication

Queue delay model for category#3-group#3 server authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Server certificate public key parameter specifier	Expected Ratio
10 (nist283r)	3 (nist163r)	$p_{10,3}$
10 (nist283r)	7 (nist233r)	$p_{10,7}$
10 (nist283r)	10 (nist283r)	$p_{10,10}$

Table 3.16 Category#3-Group#3

Using Eq. (3.16), Eq. (3.17) and Eq. (3.18), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{S\_S\_CH} + T_{S\_S\_CCERT} + T_{S\_S\_CKX} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{5} \qquad (3.58)$$

$$\overline{X^2} = \frac{\left(T^2{}_{S\_S\_CH} + T^2{}_{S\_S\_CCERT} + T^2{}_{S\_S\_CKX} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{5} \qquad (3.59)$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{S\_S\_CH} = p_{10,3}T_{S\_S\_CH,10,3} + p_{10,7}T_{S\_S\_CH,10,7} + p_{10,10}T_{S\_S\_CH,10,10} \qquad (3.60)$$

$$T_{S\_S\_CCERT} = p_{10,3}T_{S\_S\_CCERT,10,3} + p_{10,7}T_{S\_S\_CCERT,10,7} + p_{10,10}T_{S\_S\_CCERT,10,10} \qquad (3.61)$$

$$T_{S\_S\_CKX} = p_{10,3}T_{S\_S\_CKX,10,3} + p_{10,7}T_{S\_S\_CKX,10,7} + p_{10,10}T_{S\_S\_CKX,10,10} \qquad (3.62)$$

$$T_{S\_S\_CCCS} = p_{10,3}T_{S\_S\_CCCS,10,3} + p_{10,7}T_{S\_S\_CCCS,10,7} + p_{10,10}T_{S\_S\_CCCS,10,10} \qquad (3.63)$$

$$T_{S\_S\_CFIN} = p_{10,3}T_{S\_S\_CFIN,10,3} + p_{10,7}T_{S\_S\_CFIN,10,7} + p_{10,10}T_{S\_S\_CFIN,10,10} \qquad (3.64)$$

$$p_{10,3} + p_{10,7} + p_{10,10} = 1 \qquad (3.65)$$

T_{QD} is computed using Eq. (3.12).

3.2.7. Category#3-Group#4 – Mutual Authentication

Queue delay model for category#3-group#4 mutual authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Client/Server certificate public key parameter specifier	Expected Ratio
22 (RSA3072)	20 (RSA1024)	$p_{22,20}$
22 (RSA3072)	21 (RSA2048)	$p_{22,21}$
22 (RSA3072)	22 (RSA3072)	$p_{22,22}$

Table 3.17 Category#3-Group#4

Using Eq. (3.13), Eq. (3.14) and Eq. (3.15), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{M\_S\_CH} + T_{M\_S\_CCERT} + T_{M\_S\_CKX} + T_{M\_S\_CERTVRFY} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{6} \qquad (3.66)$$

$$\overline{X^2} = \frac{\left(T^2_{M\_S\_CH} + T^2_{M\_S\_CCERT} + T^2_{M\_S\_CKX} + T^2_{M\_S\_CERTVRFY} + T^2_{M\_S\_CCCS} + T^2_{M\_S\_CFIN}\right)}{6} \qquad (3.67)$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{M\_S\_CH} = p_{22,20}T_{M\_S\_CH,22,20} + p_{22,21}T_{M\_S\_CH,22,21} + p_{22,22}T_{M\_S\_CH,22,22} \qquad (3.68)$$

$$T_{M\_S\_CCERT} = p_{22,20}T_{M\_S\_CCERT,22,20} + p_{22,21}T_{M\_S\_CCERT,22,21} + p_{22,22}T_{M\_S\_CCERT,22} \quad (3.69)$$

$$T_{M\_S\_CKX} = p_{22,20}T_{M\_S\_CKX,22,20} + p_{22,21}T_{M\_S\_CKX,22,21} + p_{22,22}T_{M\_S\_CKX,22,22} \quad (3.70)$$

$$T_{M\_S\_CERTVRFY} = p_{22,20}T_{M\_S\_CERTVRFY,22,20} + p_{22,21}T_{M\_S\_CERTVRFY,22,21} + p_{22,22}T_{M\_S\_CERTVRFY,} \quad (3.71)$$

$$T_{M\_S\_CCCS} = p_{22,20}T_{M\_S\_CCCS,22,20} + p_{22,21}T_{M\_S\_CCCS,22,21} + p_{22,22}T_{M\_S\_CCCS,22,22} \quad (3.72)$$

$$T_{M\_S\_CFIN} = p_{22,20}T_{M\_S\_CFIN,22,20} + p_{22,21}T_{M\_S\_CFIN,22,21} + p_{22,22}T_{M\_S\_CFIN,22,22} \quad (3.73)$$

$$p_{22,20} + p_{22,21} + p_{22,22} = 1 \quad (3.74)$$

T_{QD} is computed using Eq. (3.12).

3.2.8. Category#3-Group#4 – Server Authentication

Queue delay model for category#3-group#4 server authenticated WTLS full handshake will be given in this section.

CA certificate public key parameter specifier	Server certificate public key parameter specifier	Expected Ratio
22 (RSA3072)	20 (RSA1024)	$p_{22,20}$
22 (RSA3072)	21 (RSA2048)	$p_{22,21}$
22 (RSA3072)	22 (RSA3072)	$p_{22,22}$

Table 3.18 Category#3-Group#4

Using Eq. (3.16), Eq. (3.17) and Eq. (3.18), average services time and the second moment of the service time can be interpreted as:

$$\overline{X} = \frac{\left(T_{S\_S\_CH} + T_{S\_S\_CCERT} + T_{S\_S\_CKX} + T_{M\_S\_CCCS} + T_{M\_S\_CFIN}\right)}{5} \qquad (3.75)$$

$$\overline{X^2} = \frac{\left(T^2{}_{S\_S\_CH} + T^2{}_{S\_S\_CCERT} + T^2{}_{S\_S\_CKX} + T^2{}_{M\_S\_CCCS} + T^2{}_{M\_S\_CFIN}\right)}{5} \qquad (3.76)$$

Eq. (3.19) and Eq. (3.20) states that:

$$T_{S\_S\_CH} = p_{22,20}T_{S\_S\_CH,22,20} + p_{22,21}T_{S\_S\_CH,22,21} + p_{22,22}T_{S\_S\_CH,22,22} \qquad (3.77)$$

$$T_{S\_S\_CCERT} = p_{22,20}T_{S\_S\_CCERT,22,20} + p_{22,21}T_{S\_S\_CCERT,22,21} + p_{22,22}T_{S\_S\_CCERT,22,22} \qquad (3.78)$$

$$T_{S\_S\_CKX} = p_{22,20}T_{S\_S\_CKX,22,20} + p_{22,21}T_{S\_S\_CKX,22,21} + p_{22,22}T_{S\_S\_CKX,22,22} \qquad (3.79)$$

$$T_{S\_S\_CCCS} = p_{22,20}T_{S\_S\_CCCS,22,20} + p_{22,21}T_{S\_S\_CCCS,22,21} + p_{22,22}T_{S\_S\_CCCS,22,22} \qquad (3.80)$$

$$T_{S\_S\_CFIN} = p_{22,20}T_{S\_S\_CFIN,22,20} + p_{22,21}T_{S\_S\_CFIN,22,21} + p_{22,22}T_{S\_S\_CFIN,22,22} \qquad (3.81)$$

$$p_{22,20} + p_{22,21} + p_{22,22} = 1 \qquad (3.82)$$

T_{QD} is computed using Eq. (3.12).

3.3. Transmission Time Model

Data transmission time is a really complex part of the WTLS handshake protocol performance modeling due to the different characteristics of available data bearer types and existing of intermediate systems like BTS (Base Transceiver Station), BSC (Base

Station Controller), MSC (Mobile Services Switching Center), SGSN (Serving GPRS Support Node), GGSN (Gateway GPRS Support Node) etc. servicing between the WAP gateway and client.

Typically the data transmission time T_{TD} can be computed by a simple formula given below:

$$T_{TD} = \frac{L}{R} \tag{3.83}$$

Where;

L : Data length (in bits)
R : Channel transmission rate (bit/s)
T_{TD} : Data transmission time

Eq. (3.83) is not sufficient for the WAP access transmission time computation because it does not consider the extra time cost reasoning from the data bearer type. Several works on WTP performance over the wireless channel ([40], [41], [42], [43]) has shown that the traversal delay of the GSM network is very affective on data transmission time.

The WAP architecture requires sending an ACK (acknowledgement) packet to the sending peer after receiving a WTP data packet. The time duration between the transmission of the data packet and receipt of the ACK packet is defined as RTT (Round Trip Time). The RTT value is the most important parameter that must be the main consideration of the transmission time modeling.

The RTT value contains the data transmission time in addition to the traversal delay of the GSM network. The data transmission time can easily be computed using Eq. (3.83). The traversal delay varies for different data bearer types. It may also vary for different GSM network providers, traffic load in the current GSM cell, etc. even using the same data bearer as stated in [40].

Therefore, overall transmission delay for all of the groups of three categories can be modeled as below:

$$T_{TD} = \frac{L}{R} + 2 \times \left(number \quad of \quad applicable \quad messages \right) \times T_{traversal} \qquad (3.84)$$

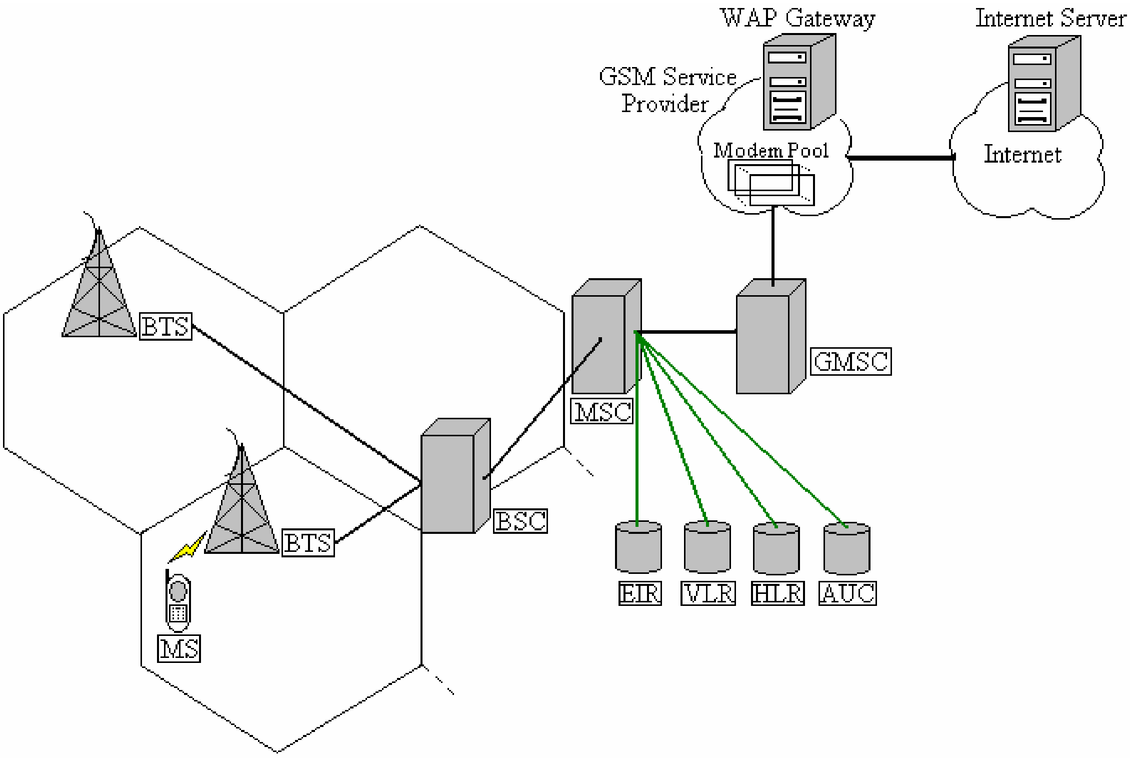
The meaning of the symbols used in Eq. (3.84) is given in Table 3.19.

Symbol	Meaning
L	Total data length (in bits) including all the handshake messages and their corresponding ACK packets
R	Channel transmission rate (bit/s)
T_{TD}	Data transmission time (s)
$T_{traversal}$	One way traversal delay of the GSM network specific to the data bearer and GSM service provider

Table 3.19 Transmission delay modeling formula notations

Two different data bearers have been considered in the data transmission time model. These are GSM CSD and GPRS. It is necessary to grasp the architectural components that cause the traversal delay in order to clearly understand the reasons of this extra cost. Therefore, system architecture for WAP access using GSM CSD, and GPRS is discussed below.

A typical GSM CSD network architecture has been given in Figure 3.4. There are many GSM network components between the mobile station and the WAP Gateway. GSM CSD is a circuit switched data bearer. The mobile station initiates a data call to a specified GSM number which is connected to a dial-up modem. This is theoretically the same way that a client connects to a RAS (Remote Access Server). Modem pool is connected to the WAP Gateway. Therefore, there are many other interfaces between the mobile station and the WAP Gateway. Each component's function is briefly defined next.



MSC:	Mobile Services Switching Center	AUC:	Authentication Center
GMSC:	Gatway Mobile Services Switching Center	EIR:	Equipment Identity Register
BTS:	Base Transceiver Station	HLR:	Home Location Register
MS:	Mobile Station	VLR:	Visitor Location Register

Figure 3.3 GSM CSD network architecture

MS: Mobile station is typically a cellular phone or PDA with GSM capabilities

BTS: The Base Transceiver Station handles the radio interface to the mobile station by the means of transceiver and antennas. BTS defines a cell and communicates with the clients within its cell.

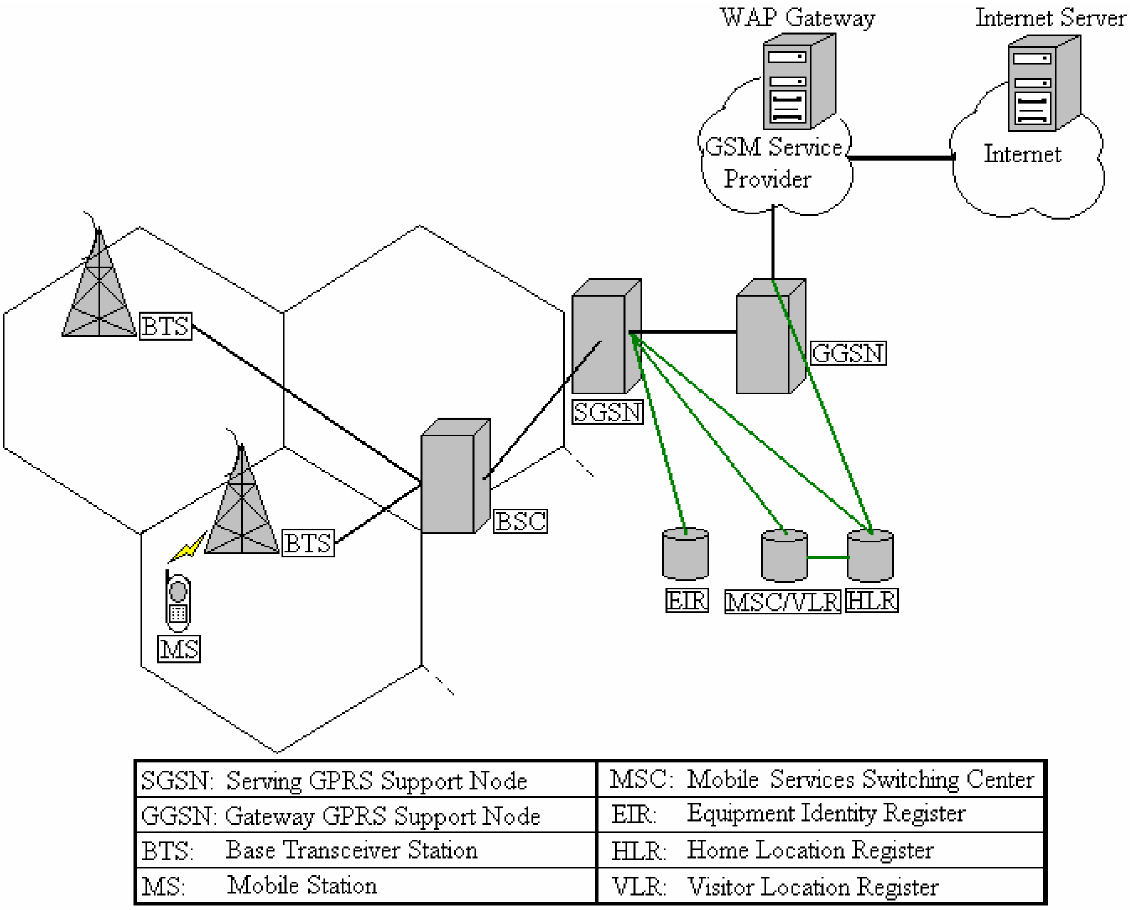
BSC: The Base Station Controller manages the radio resources for one or more BTSs. The BSC establishes the connection between the mobile stations and the Mobile Switching Center (MSC).

MSC: The Mobile Services Switching Center manages the telephony switching functions. It also handles the mobility management operations. Toll ticketing, network interfacing, common channel signaling are other tasks of MSCs.

GMSC: The Gateway Mobile Services Switching Center is used to interconnect MSCs.

Home Location Register (*HLR*), Visitor Location Register (*VLR*), Authentication Center (*AUC*), and Equipment Identity Register (*EIR*) are the databases that are used for the purpose of call control and network management.

Figure 3.4 gives a typical GPRS network system architecture. There are some common components with the GSM CSD network like BTS, BSC, MSC and the operational databases. Two new components specific to the GPRS network are Serving GPRS Support Node (SGSN), and Gateway GPRS Support Node (GGSN).



SGSN: Serving GPRS Support Node	MSC: Mobile Services Switching Center
GGSN: Gateway GPRS Support Node	EIR: Equipment Identity Register
BTS: Base Transceiver Station	HLR: Home Location Register
MS: Mobile Station	VLR: Visitor Location Register

Figure 3.4 GPRS network architecture

SGSN: The Serving GPRS Support Node routes the packet switched data to and from the mobile station. It also performs mobility management, location management, logical link management, authentication and charging for calls.

GGSN: The Gateway GPRS Support Node converts the GPRS packets coming from the SGSNs into packet data protocol appropriate to the outside data network. Data protocol used is usually IP or X.25. Similarly it converts the outside data network packets to the GPRS packet format and forwards it to the corresponding SGSN which is communicating with the Mobile Station.

System architectures for both GSM CSD and GPRS data bearer types have been discussed up to here. It is obvious that there is no direct interface between the mobile station and the WAP gateway. WAP architecture requires many format conversions to be performed between the architectural components. This is the main reason for the network traversal delay.

4. IMPLEMENTATION RESULTS

Timing measurements were performed using a Nokia Symbian 6.1 [44] phone, Nokia 7650, as the client and open source Kannel Gateway [45] running on Cygwin [46] as the WAP gateway. Crypto primitives and the WTLS Handshake Protocol [5] were implemented using C++. Microsoft Visual Studio 6.0 has been used as the development environment for both client and server sides, where Nokia's Symbian 6.1 SDK was used to build the code for the ARM [47] processor of Nokia 7650. The gateway (server) operating system was Microsoft Windows 2000 Advanced Server SP4 running on an Intel P4 2.4 GHz CPU and 1GB DDR RAM.

Three main factors on WTLS Handshake Protocol performance are considered. These are:

i. Processing time at both client and server
ii. Server queue waiting time
iii. Handshake data transmission time over the GSM network

Client and server processing time model is given at Section 3.1. Processing time for each handshake message appropriate to the selected handshake suite must be measured to compute the overall handshake processing time T_{PD}. Each handshake suite has been performed for 15 times during the tests and the arithmetic mean of these 15 different values has been recorded as the average processing time for the corresponding handshake message. 24 different cases for both mutual authenticated and server authenticated WTLS handshake have been performed and the measured timing values were used to compute the overall processing time for client (T_{PD_C}) and server (T_{PD_S}). Detailed analysis of client and server processing time for a total of 48 test cases has been given in Section 4.1.

Server queue delay model is given at Section 3.2. The server is modeled as an M/G/1 queue and the average waiting time T_{QD} for a key exchange suite can be computed using (3.12). Server queue waiting time is strictly dependent on the average processing time of each handshake message in the selected key exchange suite. The queue delay model in conjunction with the measured timing values has been used to compute T_{QD} . Computations were done using Matlab 6.0 and analyzed in Section 4.2.

The data transmission delay T_{TD} model is given in Section 3.3. T_{TD} has been computed by subtracting the total processing time T_{PD} from the measured overall handshake time. Then using the transmission delay model, the GSM service provider's traversal delay $T_{traversal}$ has been computed for the selected data bearer. Data transmission delay analysis is given in Section 4.3.

4.1. Processing Time Analysis

Processing times of the client side and the server side has been measured for 24 mutual authenticated and 24 server authenticated WTLS handshake test cases. Each test case differs by the CA certificate public key parameter specifier and the peers' certificate public key parameter specifier. These test cases are in fact the three categories defined in Section 3. Mutual authenticated WTLS handshake processing times are given in Table 4.1 and server authenticated WTLS handshake processing times are given in Table 4.2. Processing time analysis has been done using the timing values given in these two tables and corresponding figures generated using these timing values.

Category#	Group#	CA Cert.	Client/Server Cert.	Client Processing Time(ms)	Server Processing Time(ms)
1	1	160p	160p	745.65	10.57
	2	163k	163k	502.74	24.48
	3	163r	163r	489.01	24.01
	4	rsa1024	rsa1024	3780.40	45.86
2	1	224p	160p	1477.19	18.66
		224p	224p	1860.47	22.52
	2	233k	163k	733.85	36.85
		233k	233k	886.93	43.61
	3	233r	163r	758.14	38.04
		233r	233r	885.57	46.11
	4	rsa2048	rsa1024	5192.50	60.52
		rsa2048	rsa2048	23567.50	264.69
3	1	256p	160p	2071.71	27.46
		256p	224p	2441.38	29.55
		256p	256p	2730.20	33.04
	2	283k	163k	1207.24	58.48
		283k	233k	1320.61	65.16
		283k	283k	1552.83	75.92
	3	283r	163r	1293.13	60.90
		283r	233r	1372.31	67.33
		283r	283r	1632.93	78.63
	4	rsa3072	rsa1024	7678.87	78.40
		rsa3072	rsa2048	26239.86	298.91
		rsa3072	rsa3072	74109.11	891.53

Table 4.1 Mutual authenticated WTLS handshake overall processing times

Category#	Group#	CA Cert.	Server Cert.	Client Processing Time(ms)	Server Processing Time(ms)
1	1	160p	160p	738.95	3.85
	2	163k	163k	501.89	8.42
	3	163r	163r	485.63	12.51
	4	rsa1024	rsa1024	1006.78	36.99
2	1	224p	160p	1451.11	3.68
		224p	224p	1853.71	7.90
	2	233k	163k	724.86	8.77
		233k	233k	861.78	14.70
	3	233r	163r	759.96	9.31
		233r	233r	884.06	26.76
	4	rsa2048	rsa1024	2451.88	37.80
		rsa2048	rsa2048	3925.60	240.39
3	1	256p	160p	2079.21	3.73
		256p	224p	2470.75	7.97
		256p	256p	2742.43	11.48
	2	283k	163k	1206.30	8.50
		283k	233k	1308.86	15.44
		283k	283k	1552.11	25.86
	3	283r	163r	1286.09	8.41
		283r	233r	1368.78	15.61
		283r	283r	1625.03	26.67
	4	rsa3072	rsa1024	4935.73	38.37
		rsa3072	rsa2048	6396.90	244.18
		rsa3072	rsa3072	8815.48	871.41

Table 4.2 Server authenticated WTLS handshake overall processing times

Analyzing the processing times for the category#1, server processing times are always lower than the client processing times as an expected result. Category#1-group3(163 bit random curve) client side processing time is the lowest one among the others for both server authenticated and mutual authenticated WTLS handshake. 163 bit

Koblitz curve is on the second and the 160 bit prime curve is the worst case for the ECDH_ECDSA key exchange suites of Category#1 client side processing times. Group#4(RSA 1024) key exchange is significantly slower when compared to the ECDH_ECDSA key exchange suites in the same category. Server authenticated WTLS handshake client processing time for the group#4 (RSA key exchange using 1024 bit public key) is 1006.78 ms, where it is 485.63 ms for the group#3 (ECDH_ECDSA key exchange using 163 bit random curve). Mutual authenticated WTLS handshake client processing time for the group#4 is 3780.4 ms and it is 489.01 ms for the group#3. Comparing the results for category#1 processing times of server authenticated and mutual authenticated handshakes, it is clear that the ECDH_ECDSA key exchange client processing times increase a little for the mutual authenticated WTLS handshake. On the other hand, group#4 key exchange client processing time significantly increases from 1006.78 ms to 3780.4 ms for the mutual authenticated WTLS handshake.

Measurements show that the processing times of category#1 behave different for the server side. ECDH_ECDSA key exchange suites' processing times are always lower than the RSA key exchange but surprisingly the 160 bit prime curve which is the worst case for the client side performs better than the other ECC curves for the server side. Group#2 and group#3 server processing times are almost equal for the server authenticated WTLS handshake, where group#2 performs 50% better than group#3 for the mutual authenticated WTLS handshake. Another interesting concern is that the ratio of the server side processing times for the server authenticated WTLS handshake is significantly different. ECDH_ECDSA key exchange using 160 bit prime curve performs in a total processing time of 3.85 ms, where 163 bit Koblitz curve performs in 8.42 ms, 163 bit random curve performs in 12.51 ms and 1024 bit RSA key exchange suite performs in 36.99 ms. The ratios of the group#2(163 bit Koblitz curve), group#3(163 bit random curve) and group#4(1024 bit RSA) processing times over group#1(160 bit prime curve) processing time are 2.18, 3.24 and 9.6 respectively. The ratios of the processing times are not that much for the client side. The same rule also applies for the mutual authenticated WTLS handshake with the following ratios 2.31, 2.27 and 4.33 respectively.

Category#1 client processing times and server processing times for mutual authenticated and server authenticated WTLS handshake cases are given in Figure 4.1 and Figure 4.2.

Client processing times for category#1 ECDH_ECDSA key exchange suites of mutual authenticated and server authenticated WTLS handshakes are almost equal, where the RSA key exchange suite processing time significantly increases for the mutual authenticated WLS handshake as seen in Figure 4.1.

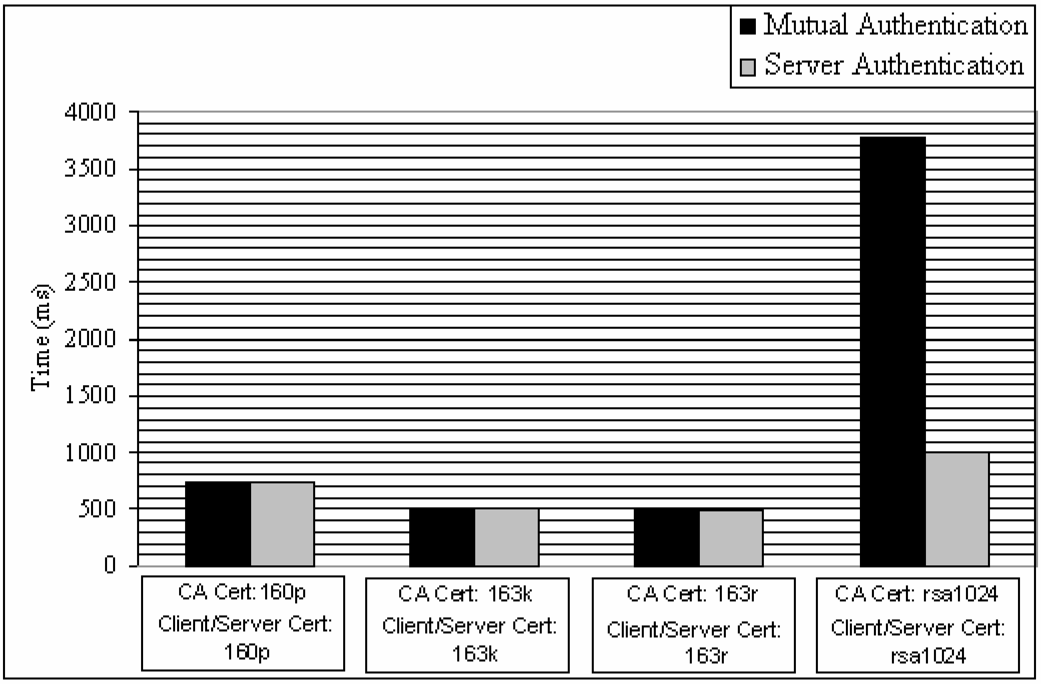


Figure 4.1 Category#1 client processing times

Server processing times for category#1 mutual authenticated WTLS handshake are significantly higher then the server authenticated WTLS handshake groups' processing times. Server authenticated and mutual authenticated handshake processing times are compared in Figure 4.2.

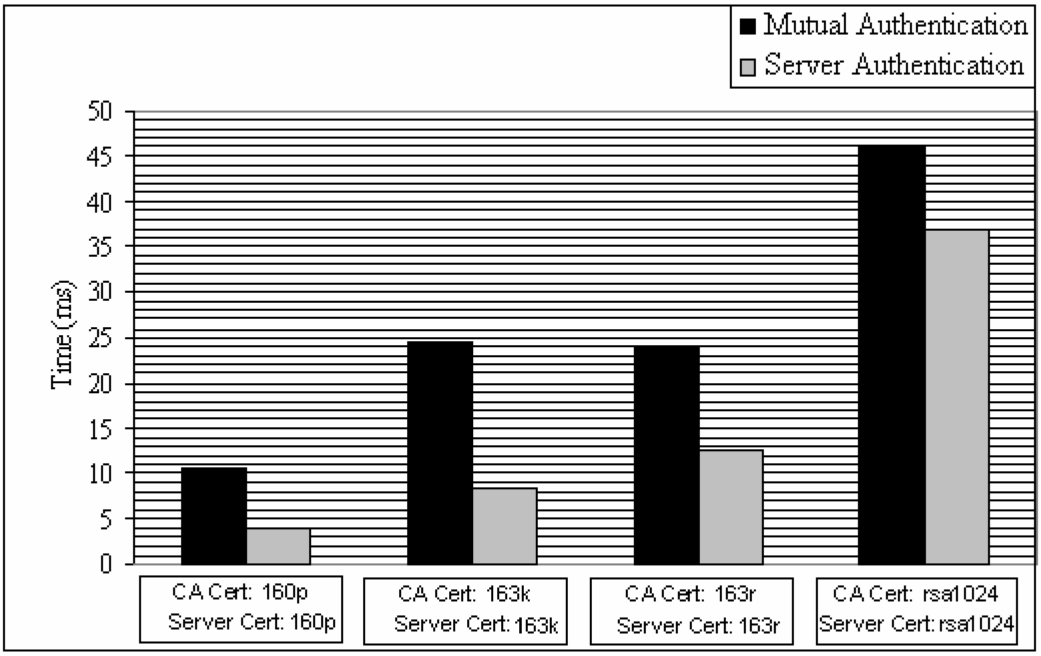


Figure 4.2 Category#1 server processing times

Category#2 mutual authenticated and server authenticated WTLS handshakes client and server processing times are given in Figure 4.3 and Figure 4.4 respectively.

Category#2 processing times have similar characteristic with category#1 but there are also some slight changes. Generally speaking, group#4 (RSA key exchange suites) handshakes always have more processing time when compared to the ECDH_ECDSA ones. Comparing the certificates that offer the same level of security, the key exchange suite that uses client certificate with 163 bit Koblitz curve parameters signed with 233 bit Koblitz curve private key has the lowest client processing time, where the client certificate with 160 bit prime curve parameters signed with 224 bit prime curve private key has the highest client processing time. Group#1 always has the highest client processing times for both server authenticated and mutual authenticated WTLS handshake. Meanwhile the client processing time when using client certificate with 233 bit random curve parameters signed with 233 bit random curve private key has only a little more client processing time when compared to the client certificate with 233 bit Koblitz parameters. Client processing time significantly increases when peers have certificates that use 2048 bit RSA public key.

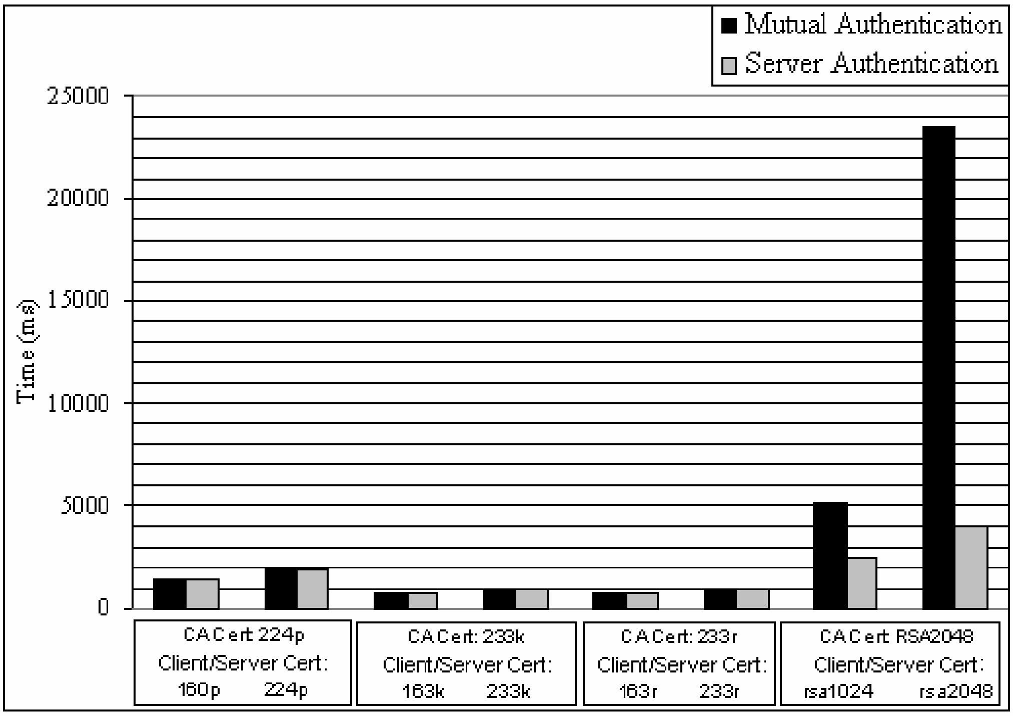


Figure 4.3 Category#2 client processing times

Similar to the category#1 handshakes, category#2 handshakes have the lowest server processing times when using ECDH_ECDSA key exchange suites with prime curve parameters.

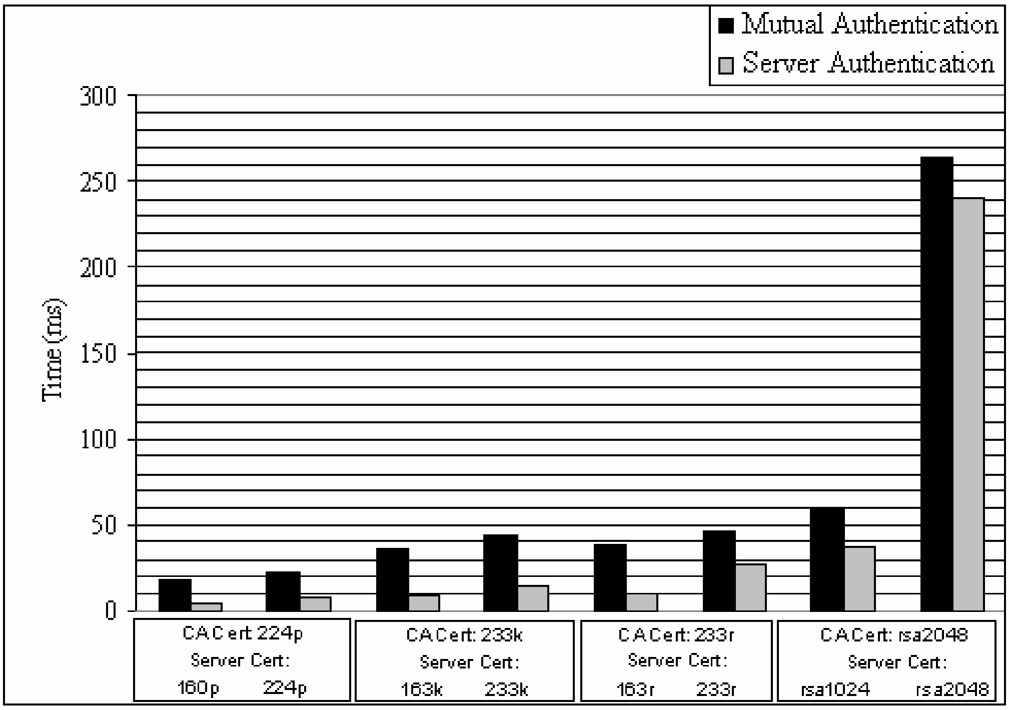


Figure 4.4 Categor y#2 server processing times

Category#3 mutual authenticated and server authenticated WTLS handshakes client and server processing times are given in Figure 4.5 and Figure 4.6.

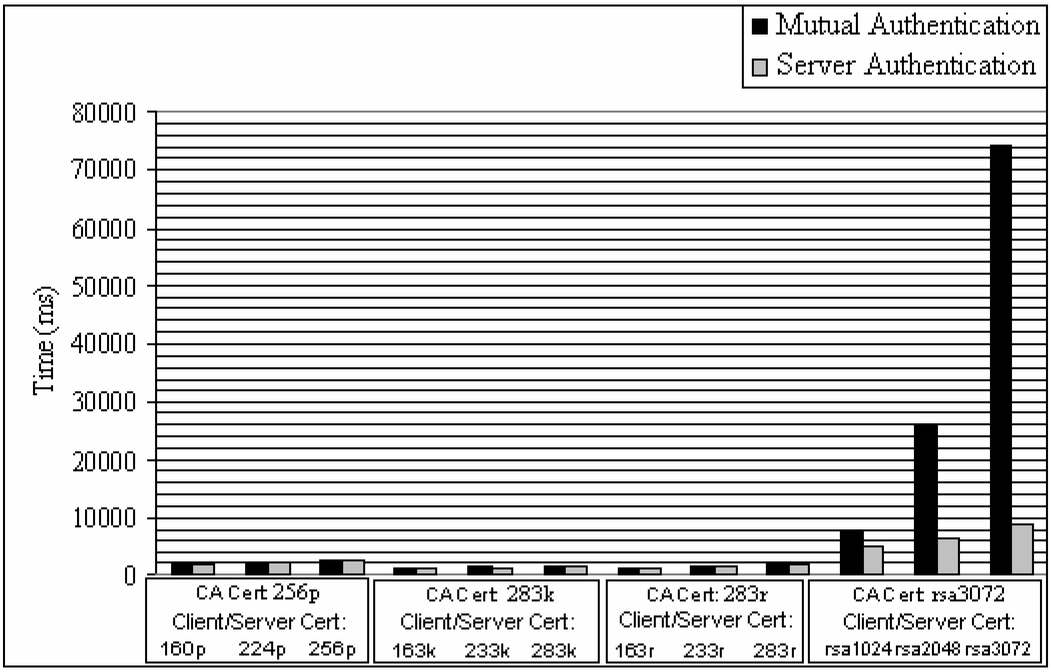


Figure 4.5 Category#3 client processing times

Client processing times of category#3-group#4(RSA) handshakes always has the highest value, where the group#2 (Koblitz curves) handshakes have the lowest client processing times. Group#1 (prime curves) have the highest client processing time among the other ECDH_ECDSA key exchange suites.

Server processing times of category#3-group#1(prime curves) has the lowest value as it is the same case for category#1 and category#2 server processing times. Server processing time with RSA 3072 bit public key certificates has the highest value. It is also interesting that the server processing time has significantly high value for both mutual authenticate and server authenticated WTLS handshake.

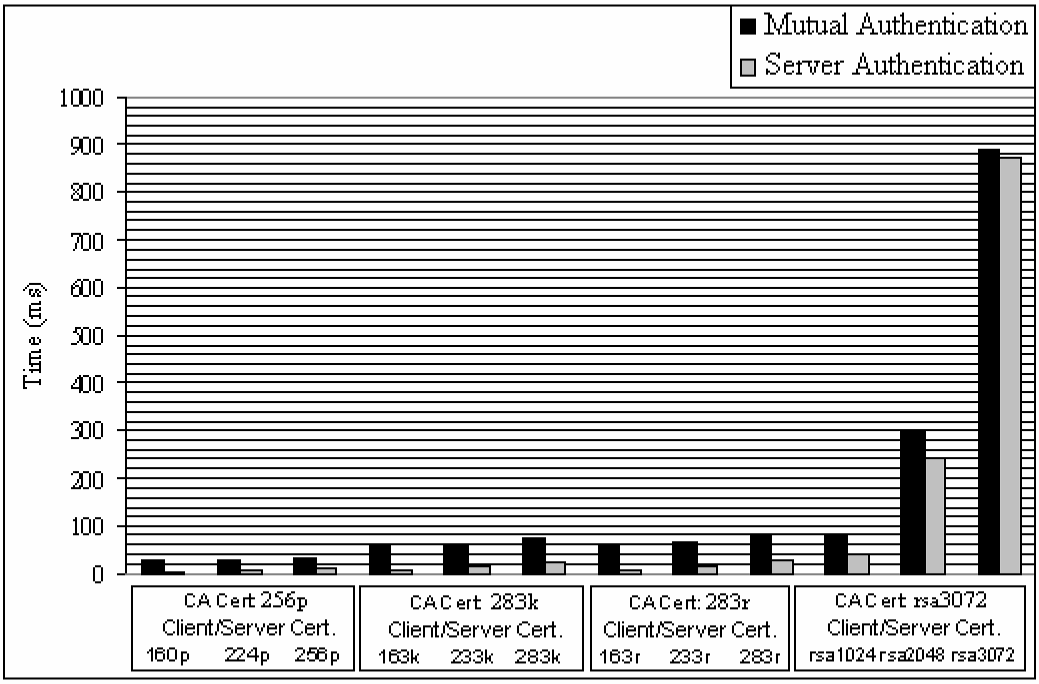


Figure 4.6 Category#3 server processing times

Comparison of the client and sever processing times for both mutual authenticated and server authenticated WTLS handshakes has been given up to here. A more detailed analysis that gives the processing time percentage for each handshake message for a given category and group is given next.

Mutual authenticated WTLS handshake ECDH_ECDSA and RSA key exchange suites client processing time percentages are given in Table 4.3 and Table 4.4. It is obviously seen that there are two main operations for ECDH_ECDSA key exchange suites and three main operations for RSA key exchange suites. $T_{M\_C\_SCERT}$ is 65%-87% of overall client processing time and $T_{M\_C\_CKX}$ is 11%-34% of overall client processing time for ECDH_ECDSA key exchange suites. This shows that two main handshake messages for ECDH_ECDSA key exchange suites are ClientKeyExchange and Certificate verification. ECDH operation is performed to compute the premaster secret. Generation and processing of ClientKeyExchange message requires two operations one which is ECDH operation in $T_{M\_C\_ECDH}$ ms and the other is computing the master secret from the premaster secret in $T_{M\_C\_MS}$ ms. Measurements show that the ECDH operation gets more than 99% of ClientKeyExchange message processing time

$T_{M\_C\_CKX}$. Verifying the server certificate using the CA certificate and the ECDH operation to compute the premaster secret are the two main operations that constitute more than 98% of overall client processing time. Processing times for other handshake messages seem to be negligible for the client side.

Considering the mutual authenticated RSA key exchange suites, we see three handshake messages ClientKeyExchange, Server.Certificate and CertificateVerify that have client processing time percentage of 13%-6%, 6%-57% and 35%-88% respectively. These three messages' client processing time constitutes more than 99% of the overall client processing time and the other processing times can be neglected. CertificateVerify message includes a private key operation, $T_{M\_C\_CERTVRFY}$ has the biggest portion of the client processing time. ClientKeyExchange message processing includes two main operations, the first one is the encryption of the premaster secret using the server's public key and the second operation is computing the master secret from the premaster secret. The public key encryption time $T_{M\_C\_RSAENC}$ gets more than 99% of ClientKeyExchange message processing time $T_{M\_C\_CKX}$. Therefore, we can say that three main operations that form 99% of client processing time when using RSA key exchange suites are verifying the server certificate using the CA certificate in $T_{M\_C\_SCERT}$, encrypting the premaster secret using the server certificate in $T_{M\_C\_RSAENC}$ ms and finally signing the hash value of the previous handshake messages using the client private key in $T_{M\_C\_CERTVRFY}$ ms.

CA Cert.	Client Cert.	Others	$T_{M\_C\_CKX}$	$\frac{T_{M\_C\_ECDH}}{T_{M\_C\_CKX}}$	$\frac{T_{M\_C\_MS}}{T_{M\_C\_CKX}}$	$T_{M\_C\_SCERT}$
160p	160p	1.24	33.38	99.66	0.34	65.38
224p	160p	0.20	16.24	99.67	0.33	83.56
224p	224p	1.02	33.48	99.85	0.15	65.51
256p	160p	0.40	11.86	99.67	0.33	87.74
256p	224p	0.36	24.99	99.86	0.14	74.65
256p	256p	0.38	32.57	99.91	0.09	67.05
163k	163k	1.81	32.92	99.47	0.53	65.27
233k	163k	1.57	21.17	99.42	0.58	77.26
233k	233k	0.91	33.69	99.71	0.29	65.40
283k	163k	0.51	12.66	99.44	0.56	86.83
283k	233k	0.64	20.80	99.68	0.32	78.56
283k	283k	0.92	32.67	99.83	0.17	66.41
163r	163r	1.50	31.62	99.45	0.55	66.89
233r	163r	1.20	21.11	99.49	0.51	77.68
233r	233r	0.68	31.79	99.70	0.30	67.53
283r	163r	0.84	12.50	99.48	0.52	86.65
283r	233r	0.48	20.88	99.72	0.28	78.64
283r	283r	0.76	32.79	99.83	0.17	66.45

Table 4.3 Mutual authenticated WTLS handshake ECDH_ECDSA key exchange suites client processing time percentages

CA Cert.	Server Cert.	Others	$T_{M\_C\_CKX}$	$\frac{T_{M\_C\_MS}}{T_{M\_C\_CKX}}$	$\frac{T_{M\_C\_RSAENC}}{T_{M\_C\_CKX}}$	$T_{M\_C\_SCERT}$	$T_{M\_C\_CERTVRFY}$
rsa1024	rsa1024	0.26	13.20	0.17	99.83	13.20	73.34
rsa2048	rsa1024	0.38	9.65	0.19	99.81	37.61	52.36
rsa2048	rsa2048	0.02	8.23	0.05	99.95	8.29	83.47
rsa3072	rsa1024	0.17	6.51	0.19	99.81	57.33	35.99
rsa3072	rsa2048	0.05	7.50	0.04	99.96	16.75	75.71
Rsa3072	rsa3072	0.02	5.97	0.02	99.98	5.94	88.08

Table 4.4 Mutual authenticated WTLS handshake RSA key exchange suites client processing time percentages

Figure 4.7 gives the change in percentages of ECDH operation processing times, and verification of server certificate processing times of the selected ECDH key exchange suites for the client side. It is obviously seen that the processing time percentage of ECDH operation increases as larger keys are used. CA certificate public key type and the CA public key size are common for a given category. Therefore, verification of the server certificate always requires the same key public key size to be used. Thus the certificate verification operation processing time changes slightly as larger keys are used. On the other hand, ECDH operation client processing time percentage increases as larger keys are used for the mutual authenticated WTLS full handshake.

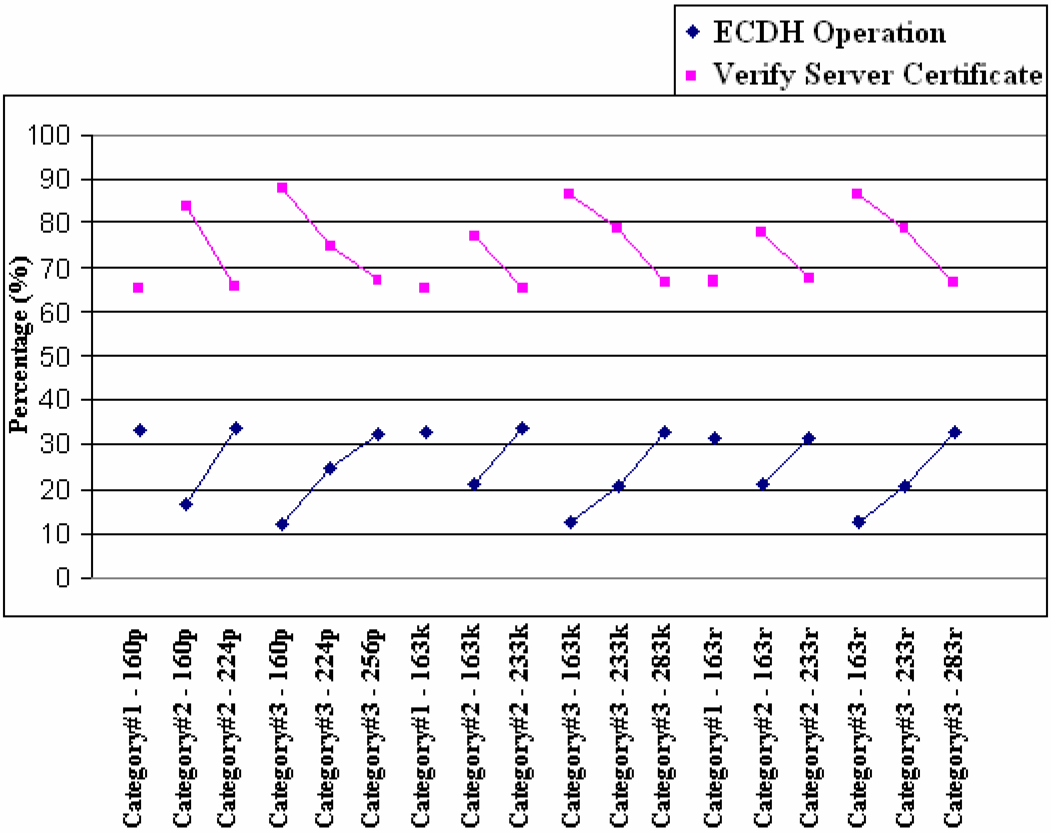


Figure 4.7 Client processing time percentages of mutual authenticated ECDH key exchange

Mutual authenticated WTLS full handshake client processing time percentage changes of RSA encryption, generation of certificate verify message and verification of server certificate are given in Figure 4.8. RSA encryption client processing time percentage changes slightly while the percentage of the private key operation to generate the certificate verify message significantly increases as larger keys are used.

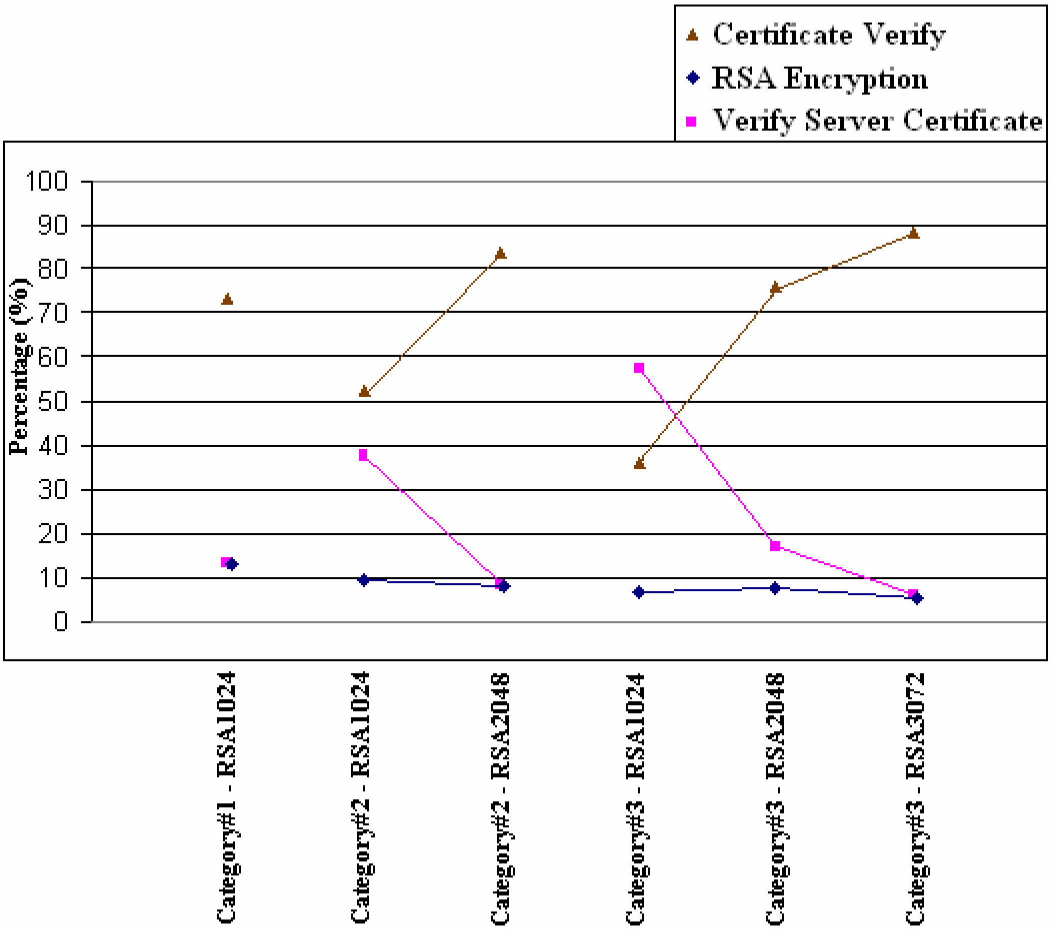


Figure 4.8 Client processing time percentages of mutual authenticated RSA key exchange

Mutual authenticated WTLS handshake ECDH_ECDSA and RSA key exchange suites server processing time percentages are given in Table 4.5 and Table 4.6. Similar to the client processing times for ECDH_ECDSA key exchange suites, $T_{M\_S\_ECDH}$ and $T_{M\_S\_CCERT}$ constitute more than 95% of the server processing times for mutual authenticated WTLS handshake. Computing the premaster secret by performing the ECDH operation, and verification of the client certificate are two main components of the server processing time for the mutual authenticated ECDH key exchange.

CA Cert.	Client Cert.	Others	$T_{M\_S\_CKX}$	$\frac{T_{M\_S\_ECDH}}{T_{M\_S\_CKX}}$	$\frac{T_{M\_S\_MS}}{T_{M\_S\_CKX}}$	$T_{M\_S\_CCERT}$
160p	160p	4.40	31.93	98.58	1.28	63.67
224p	160p	2.53	16.59	98.58	1.30	80.88
224p	224p	2.07	32.43	99.35	0.60	65.51
256p	160p	1.73	11.71	98.46	1.41	86.56
256p	224p	1.64	25.13	99.35	0.60	73.23
256p	256p	1.51	33.00	99.52	0.44	65.49
163k	163k	1.91	32.30	99.34	0.60	65.79
233k	163k	1.31	22.39	99.32	0.63	76.30
233k	233k	1.11	32.74	99.63	0.34	66.15
283k	163k	0.84	13.52	99.33	0.62	85.63
283k	233k	0.74	22.99	99.63	0.34	76.27
283k	283k	0.63	33.34	99.79	0.20	66.02
163r	163r	1.91	32.22	99.39	0.56	65.87
233r	163r	1.23	20.89	99.35	0.59	77.88
233r	233r	1.04	33.60	99.67	0.30	65.36
283r	163r	0.77	12.90	99.29	0.65	86.33
283r	233r	0.76	22.50	99.65	0.32	76.74
283r	283r	0.62	33.13	99.76	0.22	66.26

Table 4.5 Mutual authenticated WTLS handshake ECDH_ECDSA key exchange suites server processing time percentages

Processing time percentages of the ECDH operation increase when larger keys are used for a given category as shown in Figure 4.9.

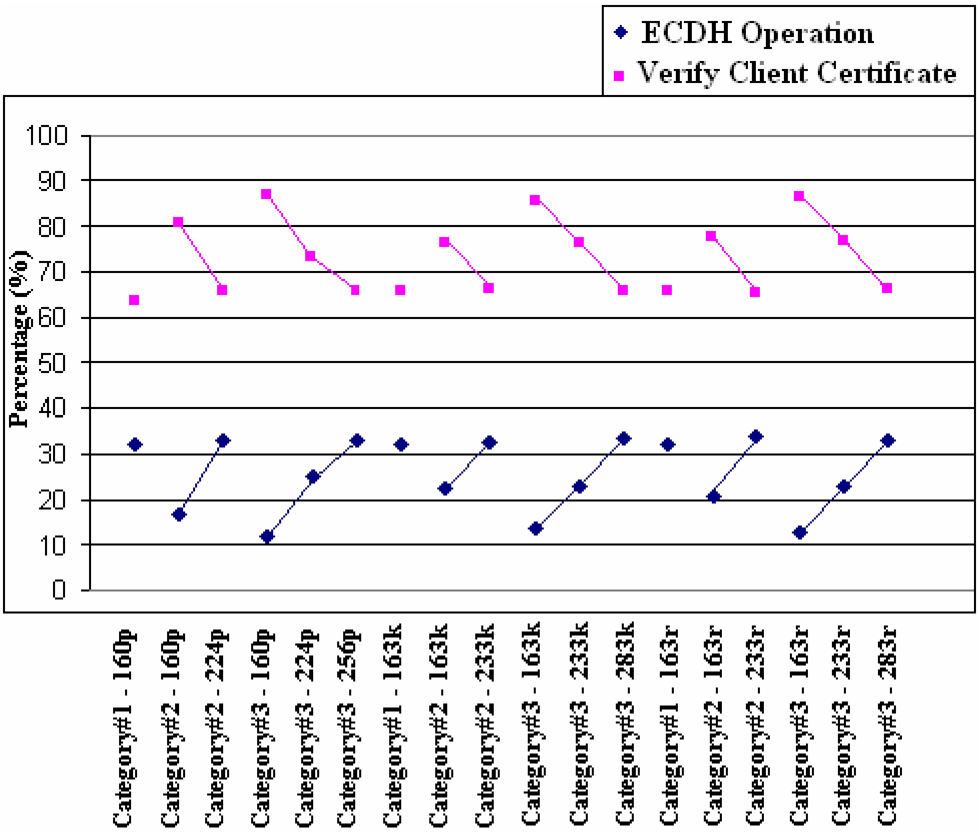


Figure 4.9 Server processing time percentages of mutual authenticated ECDH key exchange

Server processing times for the RSA key exchange suites behave different than the client side. There are again three main components, ClientKeyExchange message processing in $T_{M\_S\_CKX}$ ms, verification of the client certificate in $T_{M\_S\_CCERT}$ ms and $T_{M\_S\_CERTVRFY}$ in ms. $T_{M\_S\_CKX}$ has 64%-92% of overall server processing time, $T_{M\_S\_CERTVRFY}$ and $T_{M\_S\_CCERT}$ is 4%-42% and 4%-10% of overall server processing time. It is seen that the verification of CertificateVerify message does not have such a big affect on server processing time as it does for the client side, but it is still one of three messages to consider for the mutual authenticated WTLS handshake server processing time. Server processing time percentages for the most significant cryptographic operations are given in Figure 4.10.

CA Cert.	Server Cert.	Others	$T_{M\_S\_CKX}$	$\frac{T_{M\_S\_MS}}{T_{M\_S\_CKX}}$	$\frac{T_{M\_S\_RSADEC}}{T_{M\_S\_CKX}}$	$T_{M\_S\_CCERT}$	$T_{M\_S\_CERTVRFY}$
rsa1024	rsa1024	1.47	79.28	0.14	99.84	9.63	9.62
rsa2048	rsa1024	1.17	64.03	0.15	99.83	27.07	7.72
rsa2048	rsa2048	0.36	87.38	0.02	99.97	6.14	6.13
rsa3072	rsa1024	0.97	47.36	0.14	99.84	46.02	5.65
rsa3072	rsa2048	0.33	81.88	0.02	99.97	12.25	5.54
rsa3072	rsa3072	0.13	91.68	0.01	99.99	4.11	4.08

Table 4.6 Mutual authenticated WTLS handshake RSA key exchange suites server processing time percentages

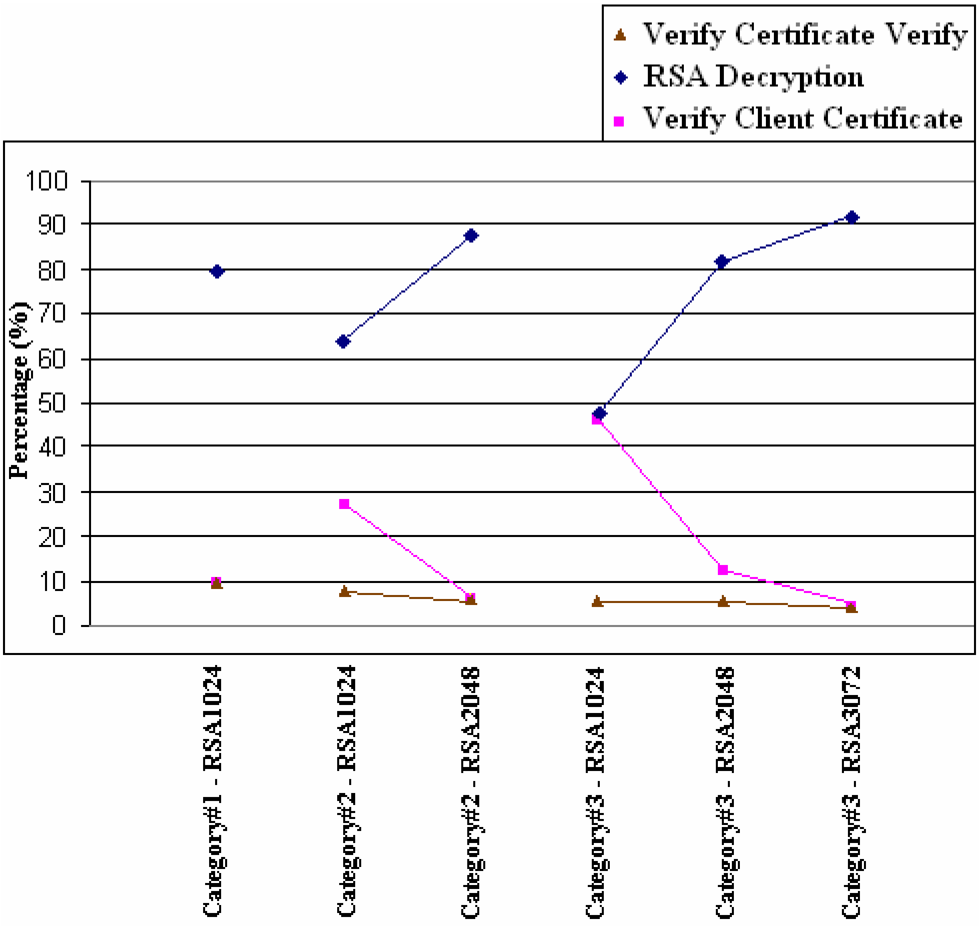


Figure 4.10 Server processing time percentages of mutual authenticated RSA key exchange

Server authenticated WTLS handshake client processing times for ECDH_ECDSA and RSA key exchange suites are given in Table 4.7 and Table 4.8 respectively. Similar to the mutual authenticated WTLS handshake client processing times, the ECDH operation and verification of the server certificate are two affective operations that constitute more than 98% of the client processing time. Namely $T_{S\_C\_ECDH}$ and $T_{S\_C\_SCERT}$ are the most important two components of the server processing time for server authenticated ECDH_ECDSA key exchange suites.

Server authenticated RSA key exchange suites do not use the CertificateVerify message as it is the case for the mutual authenticated WTLS handshake. Two main components of the client processing time for the server authenticated WTLS handshake are encrypting the premaster secret in $T_{S\_C\_RSAENC}$ ms and verifying the server certificate in $T_{S\_C\_CCERT}$ ms.

Client processing time percentages of the most significant handshake operations for the server authenticated ECDH key exchange suites are given in Figure 4.11.

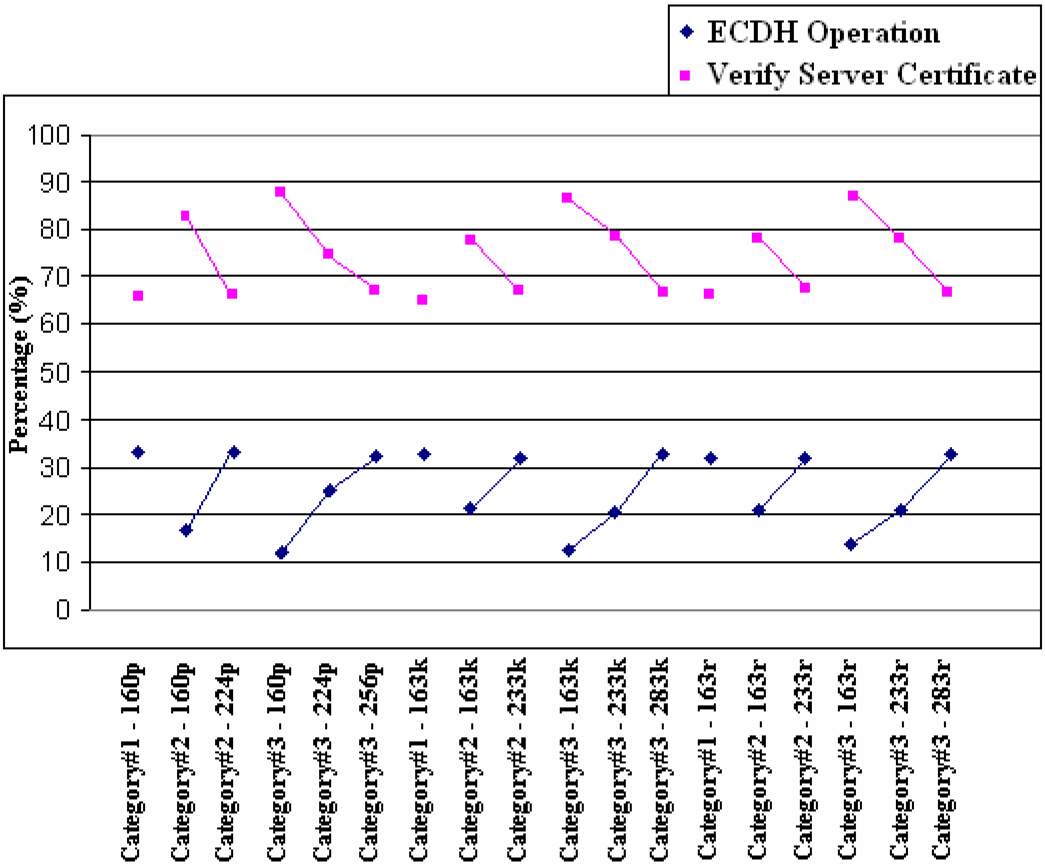


Figure 4.11 Client processing time percentages of server authenticated ECDH key exchange

Client processing time percentages of the most significant handshake operations for the server authenticated RSA key exchange suites are given in Figure 4.12.

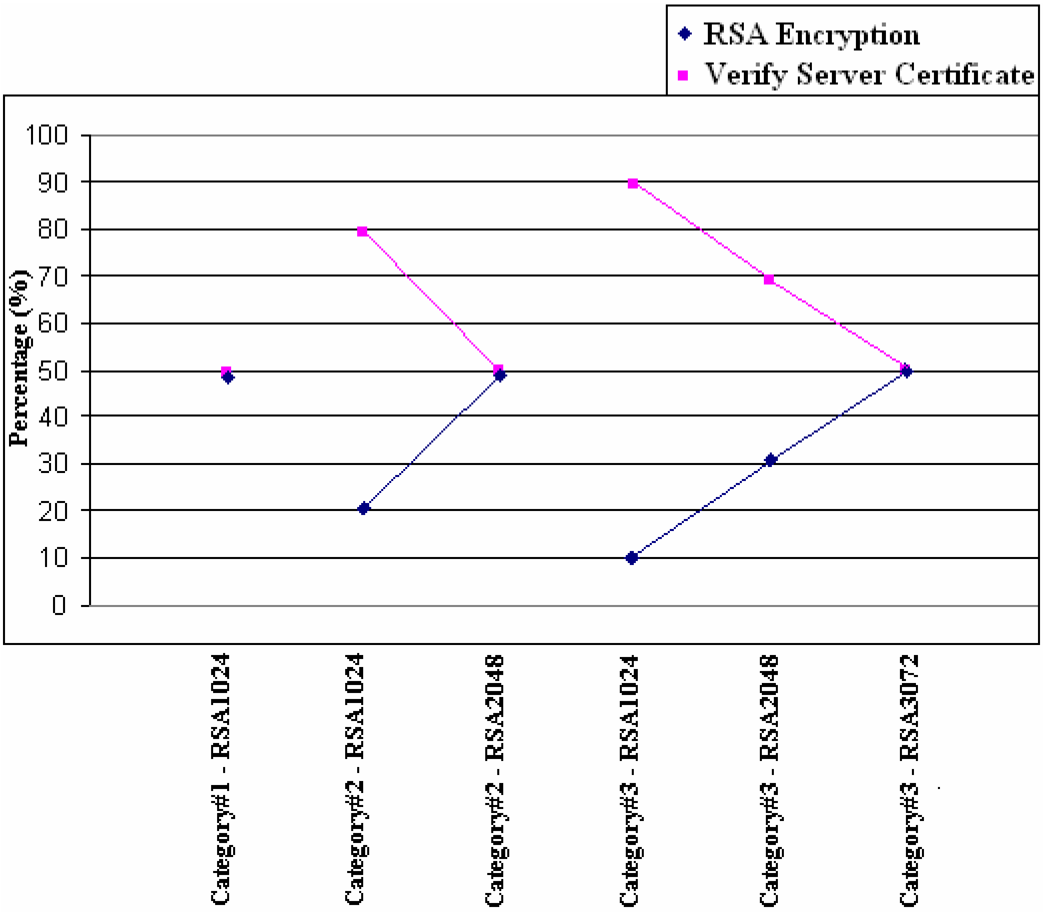


Figure 4.12 Client processing time percentages of mutual authenticated RSA key exchange

CA Cert.	Server Cert.	Others	$T_{S\_C\_CKX}$	$\frac{T_{S\_C\_ECDH}}{T_{S\_C\_CKX}}$	$\frac{T_{S\_C\_MS}}{T_{S\_C\_CKX}}$	$T_{S\_C\_SCERT}$
160p	160p	1.13	33.11	99.66	0.34	65.76
224p	160p	0.46	16.49	99.66	0.34	83.05
224p	224p	0.48	33.25	99.87	0.13	66.26
256p	160p	0.37	11.84	99.66	0.34	87.78
256p	224p	0.35	25.03	99.86	0.14	74.62
256p	256p	0.35	32.50	99.91	0.09	67.15
163k	163k	2.09	32.90	99.48	0.52	65.01
233k	163k	0.90	21.36	99.44	0.56	77.74
233k	233k	0.65	32.11	99.71	0.29	67.24
283k	163k	0.95	12.57	99.42	0.58	86.49
283k	233k	0.45	20.85	99.68	0.32	78.70
283k	283k	0.60	32.82	99.80	0.20	66.59
163r	163r	2.22	31.62	99.41	0.59	66.16
233r	163r	0.67	21.20	99.49	0.51	78.13
233r	233r	0.55	31.84	99.70	0.30	67.60
283r	163r	0.59	12.54	99.48	0.52	86.87
283r	233r	0.71	20.85	99.71	0.29	78.44
283r	283r	0.56	32.89	99.84	0.16	66.55

Table 4.7 Server authenticated WTLS handshake ECDH_ECDSA key exchange suites client processing time percentages

CA Cert.	Server Cert.	Others	$T_{S\_C\_CKX}$	$\frac{T_{S\_C\_MS}}{T_{S\_C\_CKX}}$	$\frac{T_{S\_C\_RSAENC}}{T_{S\_C\_CKX}}$	$T_{S\_C\_SCERT}$
rsa1024	rsa1024	1.23	49.47	0.17	99.83	49.30
rsa2048	rsa1024	0.41	20.29	0.18	99.82	79.30
rsa2048	rsa2048	0.35	49.87	0.04	99.96	49.78
rsa3072	rsa1024	0.22	10.13	0.20	99.80	89.66
rsa3072	rsa2048	0.15	30.83	0.05	99.95	69.02
rsa3072	rsa3072	0.10	50.05	0.02	99.98	49.85

Table 4.8 Server authenticated WTLS handshake RSA key exchange suites client processing time percentages

Server processing times for Server authenticated WTLS handshake using ECDH_ECDSA key exchange suites have only one main component. That is the ECDH operation performed in $T_{S\_S\_ECDH}$ ms. Server processing time percentages of the handshake messages for server authenticated WTLS handshake are given in Table 4.9.

CA Cert.	Server Cert.	Others	$T_{S\_S\_CKX}$	$\frac{T_{S\_S\_ECDH}}{T_{S\_S\_CKX}}$	$\frac{T_{S\_S\_MS}}{T_{S\_S\_CKX}}$
160p	160p	11.81	88.19	98.32	1.48
224p	160p	12.41	87.59	98.30	1.51
224p	224p	6.10	93.90	99.21	0.69
256p	160p	12.31	87.69	98.21	1.58
256p	224p	5.95	94.05	99.22	0.68
256p	256p	4.11	95.89	99.46	0.47
163k	163k	5.33	94.67	99.28	0.64
233k	163k	5.15	94.85	99.32	0.60
233k	233k	3.20	96.80	99.60	0.35
283k	163k	5.53	94.47	99.29	0.62
283k	233k	3.06	96.94	99.61	0.34
283k	283k	1.84	98.16	99.77	0.20
163r	163r	4.51	95.49	99.36	0.57
233r	163r	5.06	94.94	99.33	0.59
233r	233r	2.33	97.67	99.67	0.30
283r	163r	5.45	94.55	99.28	0.62
283r	233r	3.02	96.98	99.60	0.34
283r	283r	1.83	98.17	99.77	0.20

Table 4.9 Server authenticated WTLS handshake ECDH_ECDSA key exchange suites server processing time percentages

Server processing time percentages of individual handshake messages for server authenticated WTLS handshake using RSA key exchange suites are given in Table 4.10. It is obvious that the decryption of the premaster secret using the server private key is the biggest portion of the server processing time.

CA Cert.	Server Cert	Others	$T_{S\_S\_CKX}$	$\frac{T_{S\_S\_MS}}{T_{S\_S\_CKX}}$	$\frac{T_{S\_S\_RSADEC}}{T_{S\_S\_CKX}}$
rsa1024	rsa1024	1.71	98.29	0.14	99.84
rsa2048	rsa1024	1.83	98.17	0.14	99.84
rsa2048	rsa2048	0.42	99.58	0.03	99.96
rsa3072	rsa1024	1.88	98.12	0.15	99.83
rsa3072	rsa2048	0.42	99.58	0.02	99.97
rsa3072	rsa3072	0.16	99.84	0.01	99.99

Table 4.10 Server authenticated WTLS handshake RSA key exchange suites client processing time percentages

4.2. Queue Delay Analysis

Mutual authenticated and server authenticated WTLS handshake server queue delay analysis for Category#3 key exchange suites have been given in this section. The queue delay model is given in Section 3.2. The queue delay model has been implemented in Matlab 6.0 and the measured timing values have been used to generate the estimated server queue delays.

The queue delay strictly depends on the server processing time of the corresponding handshake messages. Server processing times for each category has been analyzed in Section 4.1. All of the four groups of category#3 include three possible key exchange suites with corresponding ratios. Five different ratios have been considered during the queue delay analysis. The upper limit for the arrival rate of the handshake requests is 100 handshake requests per second. Legends in all queue delay figures are in ascending order of queue delay performance from up to down.

Server queue delays for category#3-group#1 key exchange suites have been given in Figure 4.13. The utilization of the server does not reach to 1 for any of the considered five ratios. The key exchange suite using client/server certificates with 160 bit prime curve parameters signed by CA certificate with 256 bit prime curve parameters have the highest queue waiting delay because this case has the highest processing time among the other cases. On the other hand, the server queue delay is in acceptable ranges for any of the considered cases.

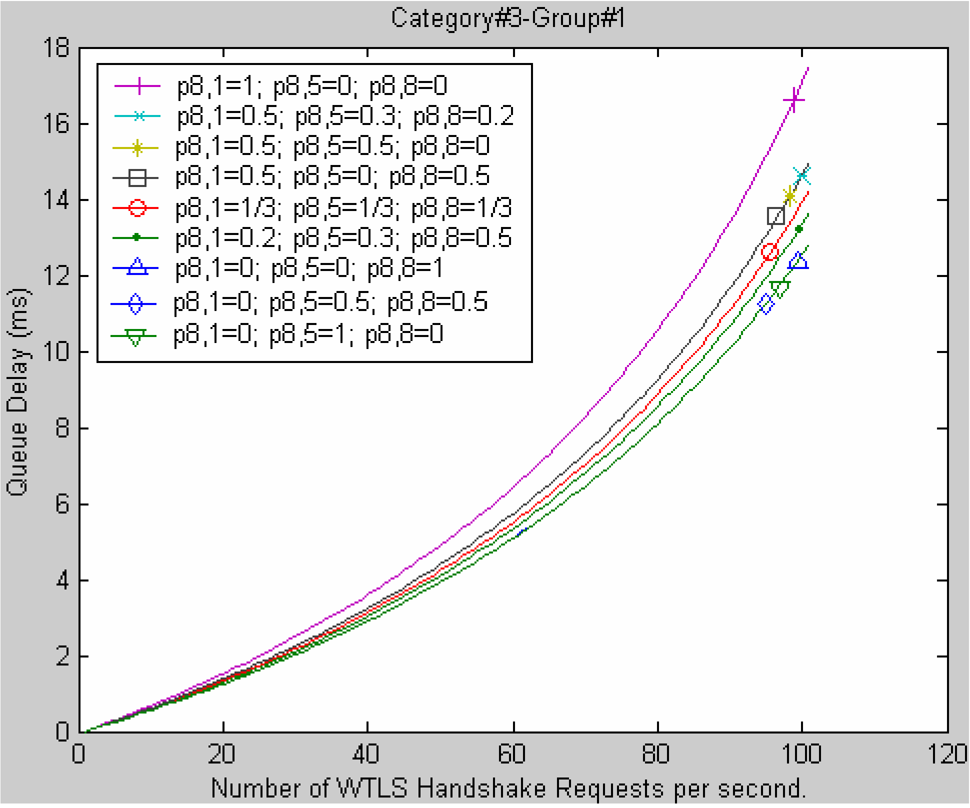


Figure 4.13 Category#3-Group#1 server queue delays(mutual authentication)

Server queue delays for category#3-group#2 key exchange suites have been given in Figure 4.14. Average server processing time increases for the group#2 key exchange suites which contain certificates with Koblitz curve parameters. Average server queue waiting time increases up to 150-160 ms for 72 handshake requests per second. The utilization of the server is equal to 1 for the systems with arrival rate higher than 72. Therefore, 72 is the practical upper limit when category#3-group2 key exchange suites are used. The server queue waiting time asymptotically increases after 72 handshake requests per second.

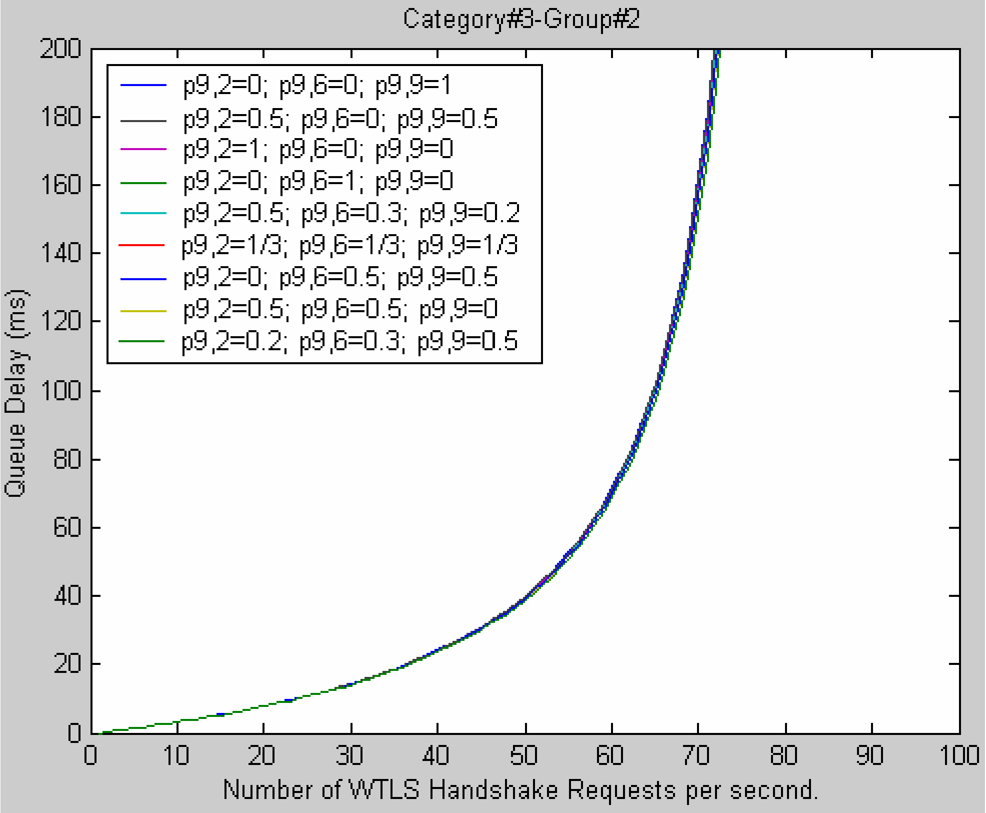


Figure 4.14 Category#3-Group#2 server queue delays(mutual authentication)

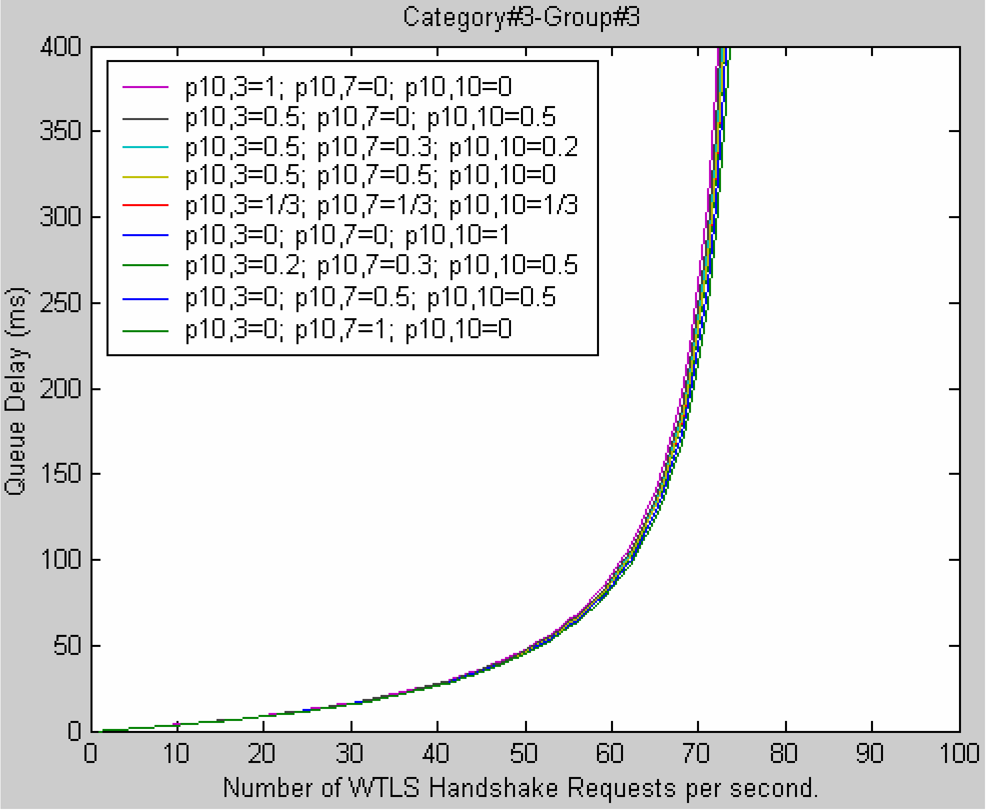


Figure 4.15 Category#3-Group#3 server queue delays(mutual authentication)

Figure 4.15 shows the server queue delays for category#3-group#3 key exchange suites. Similar to the group#2, 72 handshake requests per second is the upper limit for those types of systems and the queue waiting time goes up to 230 ms for the worst case.

Server queue waiting times for category#3-group4 is much more interesting then the other groups. Group#4 contains RSA certificates. As an expected result the case with the lowest processing time must have the lowest server queue waiting time. Queue delays for category#3-group4 key exchange suites with different ratios are given in Figure 4.16.

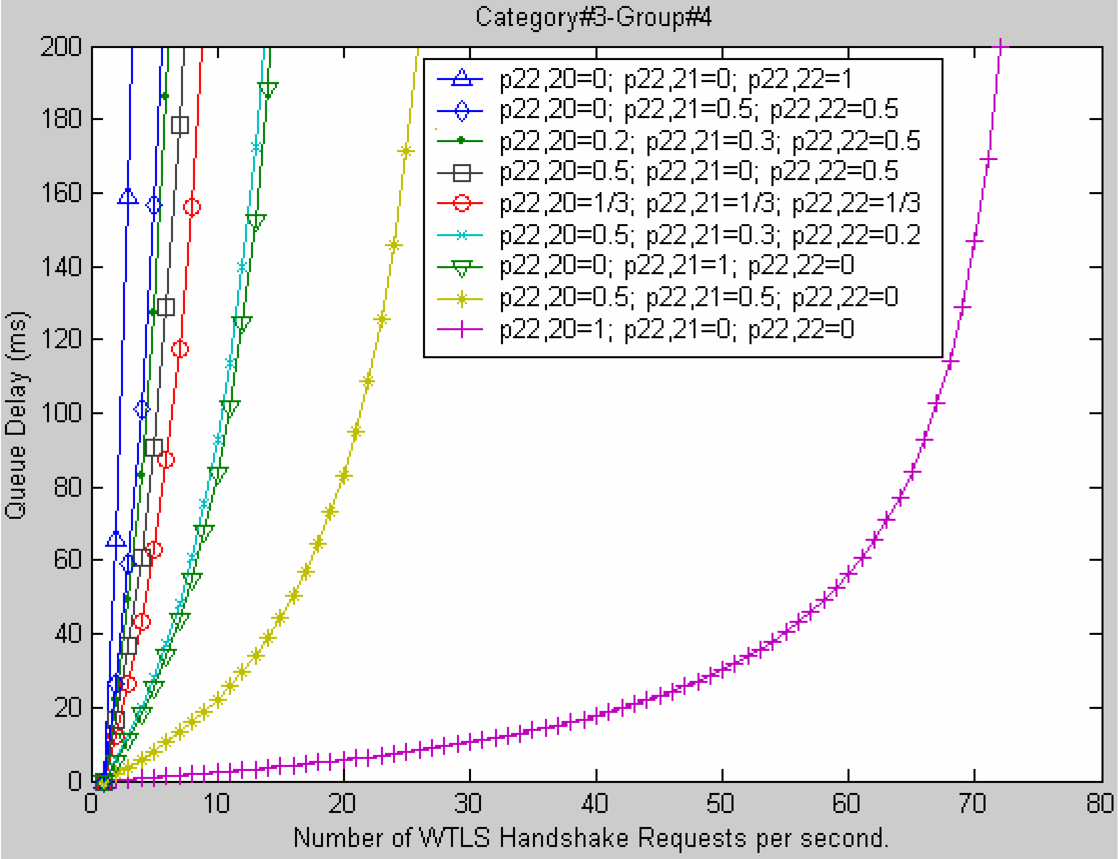


Figure 4.16 Category#3-Group#4 server queue delays(mutual authentication)

Client/server certificates containing 1024 bit RSA public key signed by CA 3072 bit RSA certificate is the best option to use. This case has 71 handshake requests per second as an upper limit with the queue waiting time around 146 ms. The case with 50% RSA 1024 certificates and the remaining 50% RSA 2048 certificates is the second case that has a good queue waiting delay characteristics. The upper limit for the handshake requests arrival rate is 25 with an average queue waiting time of 165 ms. As the ratio of the RSA 3072 client/server certificates increase, the upper limit for the handshake requests arrival rate decreases and comes to an un acceptable level for the case, where all the client/server certificates have 3072 bit RSA public key. The server can not serve more than one handshake request per second in an acceptable queue waiting time range. This is because of the extremely high server processing times when certificates with 3072 bit RSA public key are used.

A comparison of the category#3 groups' server queue delay values have been given for selected ratios in Figure 4.17, Figure 4.18 and Figure 4.19. Group#2, group#3 and group#4 key exchange suites have similar queue delay characteristics, where the group#1 is significantly better for both average queue waiting time and the upper limit for the arrival rate of handshake requests per second for the first case in Figure 4.17. Members of the groups that are compared offer the same level of security. This shows that certificates with 160 bit prime curve parameters have a better queue delay performance when compared to 163 bit Koblitz curve, 163 bit random curve or 1024 bit RSA. More than a better server queue waiting time, prime curve systems do not have bottleneck reasoning from the queue delays, where the other alternatives can not serve more than 70 handshake request per second on the average.

Figure 4.18 compares the queue delay characteristics of the category#3 groups with 224 bit prime curve, 233 bit Koblitz curve, 233 bit random curve and 2048 bit RSA. ECDH_ECDSA key exchange suites' queue delay characteristics change slightly, where the RSA key exchange suite has a significant change that it can serve a maximum of 14 handshake requests per second with a 180 ms queue delay for the worst case. The queue delay grows asymptotically for arrival rates bigger than 14 handshake requests per second.

Wee can see the more dramatic change of server queue delay characteristics in Figure 4.19 when using certificates with 3072 bit RSA certificates. Use of 3072 bit RSA certificates seems like not practical to offer the same level of security using ECC certificates with 256 bit prime curve, 283 bit Koblitz curve or 283 bit random curve parameters.

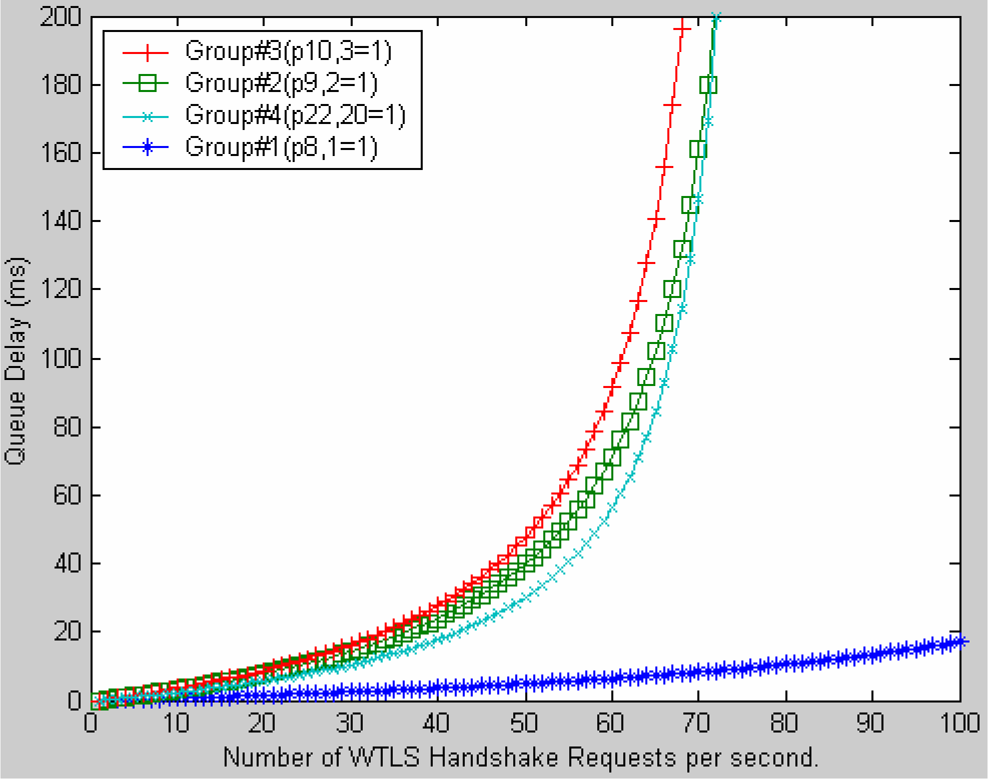


Figure 4.17 Category#3 groups server queue delays-1 (mutual authentication)

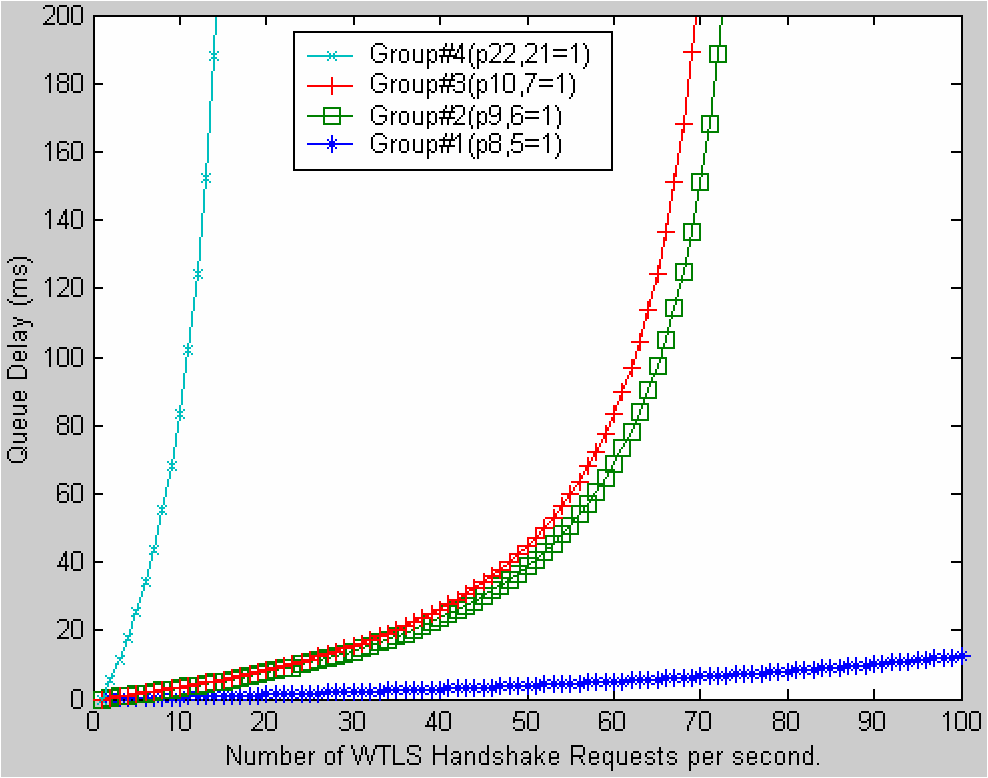


Figure 4.18 Category#3 groups server queue delays-2 (mutual authentication)

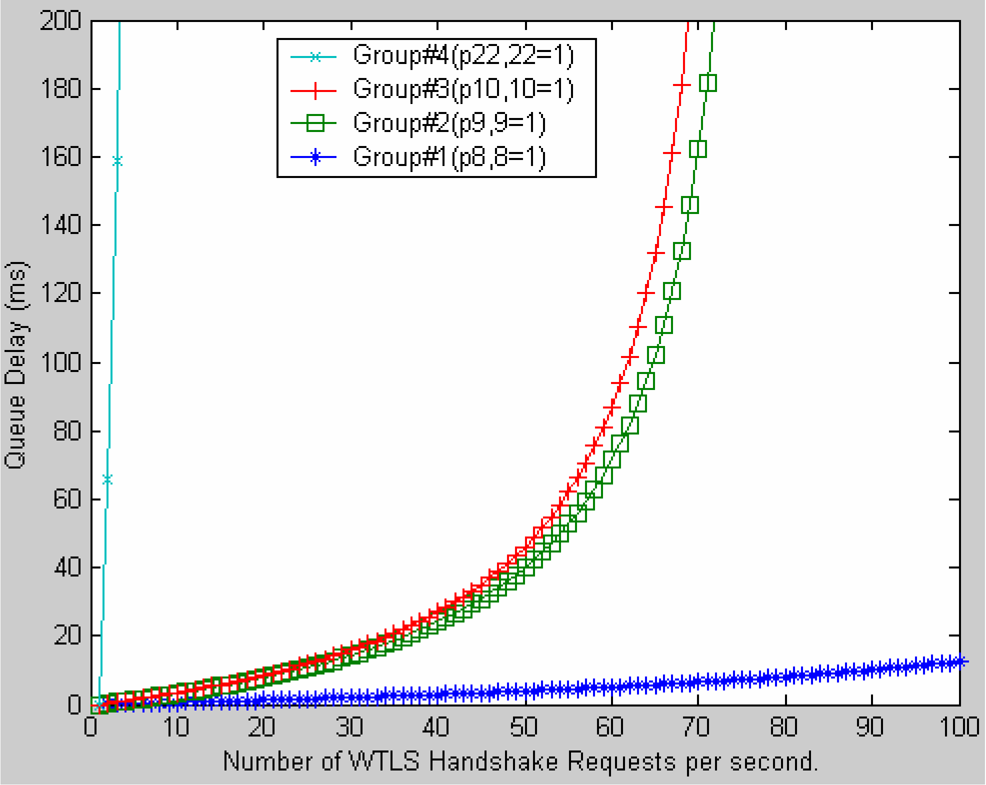


Figure 4.19 Category#3 groups server queue delays-3 (mutual authentication)

Server queue delays for server authenticated WTLS handshake key exchange suites will be analyzed next. Server queue delays for ECDH_ECDSA key exchange suites of group#1, group#2 and group#3 are given in Figure 4.20, Figure 4.21 and Figure 4.22 respectively. It is no use to analyze the server queue delay in detail for different ratios of ECC certificates for server authenticated WTLS handshake because the average waiting time is extremely small for all the cases (with a degree of $10^{-5}\,ms$). This is due to the low processing times of ECDH_ECDSA key exchange suites at the server side.

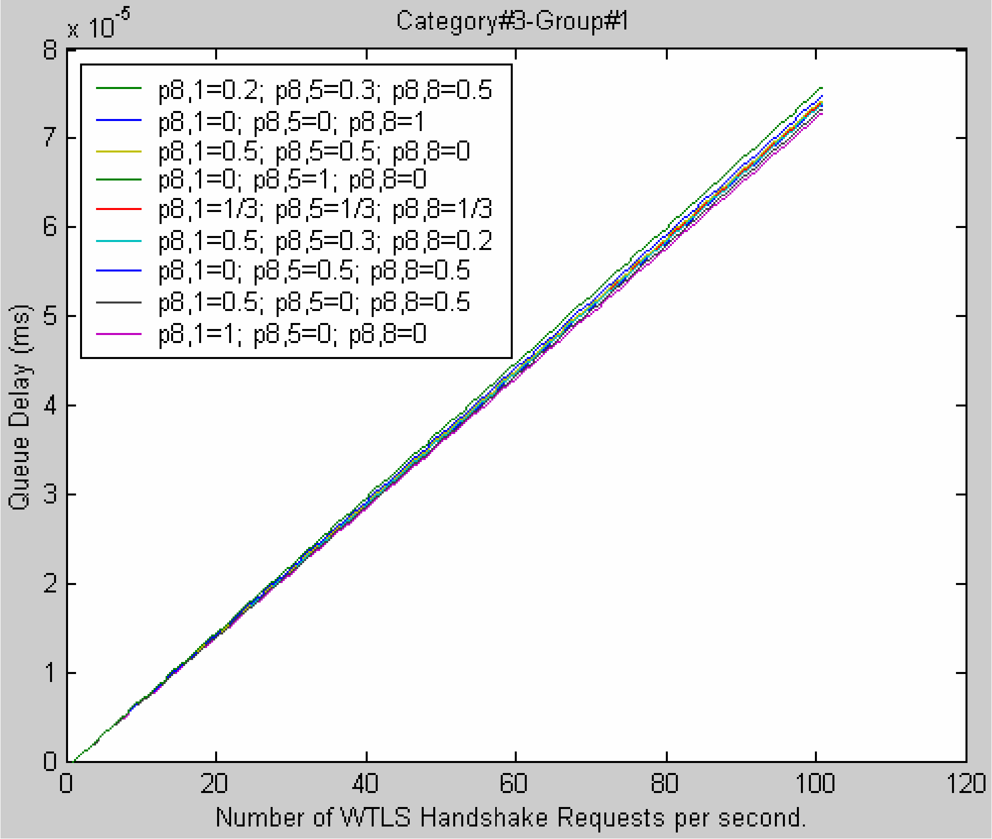


Figure 4.20 Category#3-Group#1 server queue delays(server authentication)

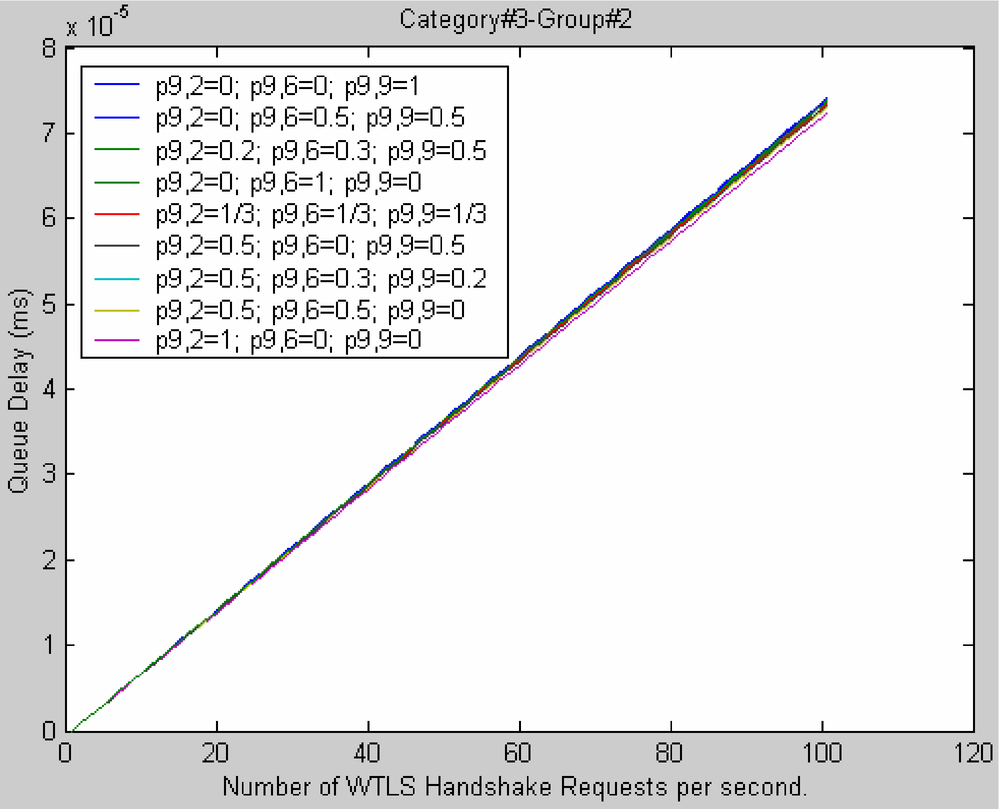


Figure 4.21 Category#3-Group#2 server queue delays(server authentication)

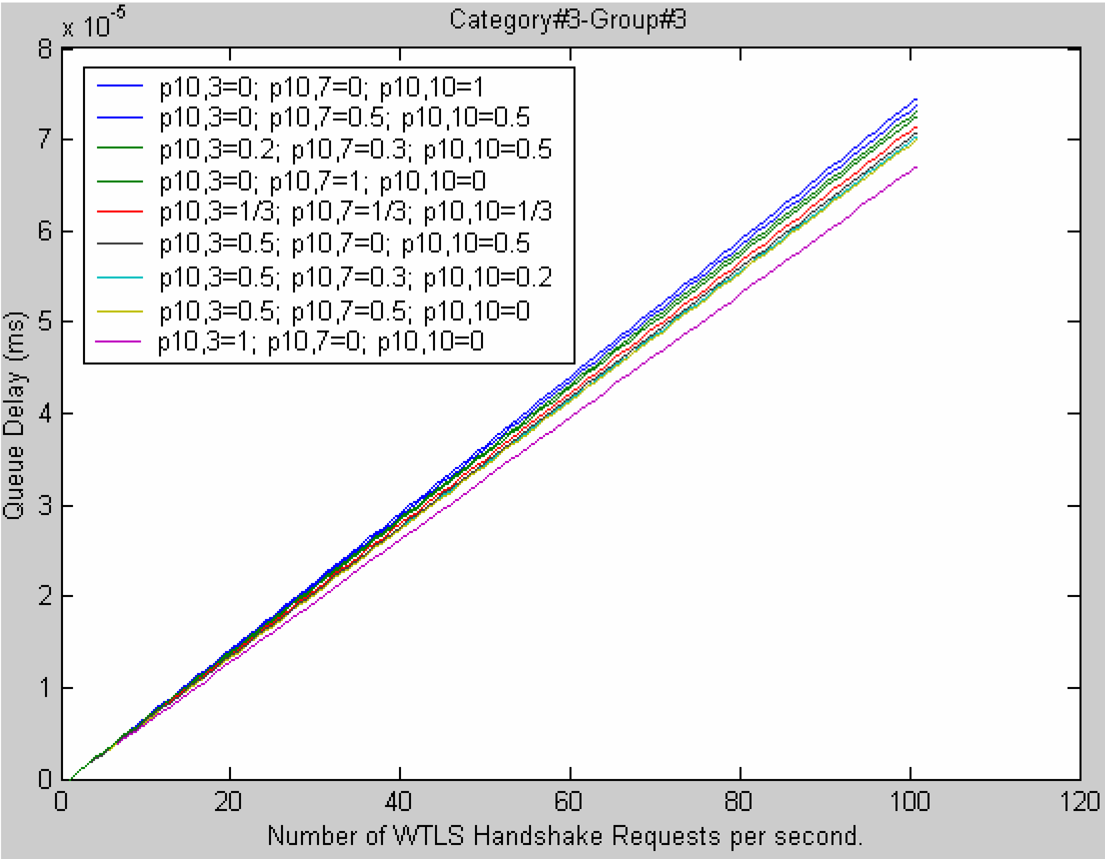


Figure 4.22 Category#3-Group#3 server queue delays(server authentication)

It is not the same perfect case as in ECDH_ECDSA key exchange suites for the server authenticated WTLS handshake when using RSA key exchange suites. Figure 4.23 shows the average queue delays for category#3-group4 key exchange suites. The server queue delay behavior of group#4 is similar to the case in the mutual authenticated WTLS handshake but with a better performance. Server certificate containing 1024 bit RSA certificate has the best queue delay performance. The queue delay is extremely low when compared to other alternatives. The case with 50% RSA 1024 certificates and 50% RSA 2048 certificates can serve 33 handshake requests per second with the average queue delay of 620 ms for the worst case. The queue delay asymptotically increases for a higher arrival rates. As an expected result, the case that uses purely 3072 bit RSA server certificates has the worst queue delay performance. Such a system has an upper limit of 6 requests per second with a queue delay of 3000 ms for the worst case.

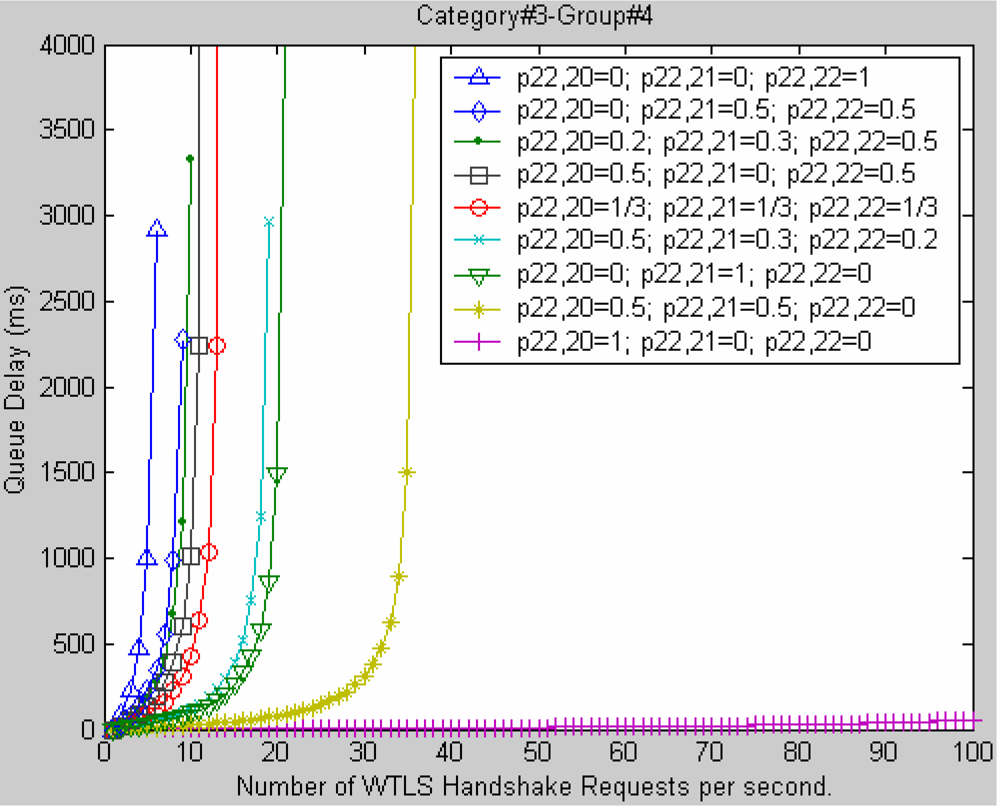


Figure 4.23 Category#3-Group#4 server queue delays(server authentication)

As a general result, the server queue delay becomes a really big bottleneck for both server authenticated and mutual authenticated WTLS handshakes as the server side processing time increases.

4.3. Data Transmission Time Analysis

Data transmission time analysis has been performed for two data bearer systems. These are GSM CSD (Global System for Mobile Communication Circuit Switched Data) and GPRS (General Packet Radio Service) data bearers. Tests have been performed using the GSM CSD data bearer. In this way, measured data transmission times have been given and the average traversal delay for this bearer type has been computed. WTLS handshake protocol could not be run over GPRS bearer because of GSM service provider restrictions on the use of GPRS. Therefore, traversal delay for the

GPRS bearer has been measured by using a different testbed which is detailed in Section 4.3.2.

The corresponding data size (in bits) of each key exchange suite tested has been given in Table 4.11.

CA Cert.	Client/Server Cert.	Mutual Authentication Data Size	Server Authentication Data Size
160p	160p	1515	1362
163k	163k	1519	1366
163r	163r	1519	1366
rsa1024	rsa1024	2231	1522
224p	160p	1543	1376
224p	224p	1575	1408
233k	163k	1551	1382
233k	233k	1587	1418
233r	163r	1555	1384
233r	233r	1591	1420
rsa2048	rsa1024	2487	1650
rsa2048	rsa2048	2999	1906
256p	160p	1559	1384
256p	224p	1591	1416
256p	256p	1607	1432
283k	163k	1579	1396
283k	233k	1615	1432
283k	283k	1639	1456
283r	163r	1579	1396
283r	233r	1615	1432
283r	283r	1639	1456
rsa3072	rsa1024	2743	1778
rsa3072	rsa2048	3255	2034
rsa3072	rsa3072	3767	2290

Table 4.11 Total data sizes for selected test cases (bytes)

4.3.1. Data Transmission Time Analysis for GSM CSD Bearer

Data transmission time has been computed by subtracting the total processing time for both peers from the overall handshake time. Data transmission time model was given in Section 3.3. Using the measured timing values, traversal delay of the test network can be found from Eq. (3.84).

$$T_{TD} = \frac{L}{R} + 2 \times \left(number \quad of \quad applicable \quad messages \right) \times T_{traversal} \qquad (\,3.84\,)$$

GSM CSD data rate is 9600 bps. So;

$$L = 9600 \quad bps$$

Measured data transmission times for each test case have been given in Table 4.12. Therefore, we can compute the average traversal delay for the GSM CSD bearer.

CA Cert.	Client/Server Cert.	$T_{TD\_MUTUAL\_AUTH}$	$T_{TD\_SERVER\_AUTH}$
160p	160p	4913.392	4952.402
163k	163k	4881.769	4722.283
163r	163r	5058.279	4924.011
rsa1024	rsa1024	5211.954	4344.448
224p	160p	5121.961	5093.972
224p	224p	4961.851	5105.81
233k	163k	4851.545	4962.061
233k	233k	4851.896	4989.211
233r	163r	4942.727	5083.323
233r	233r	4880.029	5047.458
rsa2048	rsa1024	5033.226	4522.1
rsa2048	rsa2048	5063.435	5204.466
256p	160p	5121.502	5164.236
256p	224p	4906.531	5144.932
256p	256p	5242.198	5133.612
283k	163k	5164.529	4930.2
283k	233k	5247.338	4926.873
283k	283k	5054.285	5138.207
283r	163r	5039.547	4905.654
283r	233r	6750.748	5023.451
283r	283r	4966.507	5451.005
rsa3072	rsa1024	5149.138	4499.576
rsa3072	rsa2048	5327.293	5103.475
rsa3072	rsa3072	5008.471	4930.346

Table 4.12 Transmission delays for GSM CSD bearer (ms)

Number of applicable messages is given below:

$$(number \quad of \quad applicable \quad messages) = \begin{cases} 6 \;\; if \quad MutualAuthenticated \quad RSA \\ 4 \;\; if \quad MutualAuthenticated \quad ECDH\_ECDSA \\ 5 \;\; if \quad ServerAuthenticated \quad RSA \quad or \quad ECDH\_ECDSA \end{cases}$$

All the values needed have been given up to here, then Eq. (4.1) can be used to compute the traversal delays.

$$T_{traversal} = \frac{\left(T_{TD} - \frac{L}{R}\right)}{2\left(number \quad of \quad applicable \quad messages\right)} \tag{4.1}$$

Eq. (4.1) has been implemented in Matlab 6.0 and $T_{traversal}$ was found with the following properties:

$$Expected\ value\ of\ T_{traversal} = 391.3\ ms$$

$$Standard\ deviation\ of\ T_{traversal} = 90\ ms$$

Average number of applicable messages is 5 for the selected handshake types. Round Trip Time (RTT) of a handshake message includes transmission of the corresponding ACK packets. Therefore, the average number of packets that traverse through the network is twice the number of handshake messages. The overall data transmission delay T_{TD} is 5042.8 ms on the average, approximately $391.3 \quad x \quad 10 \quad = \quad 3913 \quad ms$ of the average transmission delay comes from the traversal delay of the GSM network for the selected data bearer type GSM CSD. The transmission delay coming from the traversal delays of the network is 76% of the overall data transmission time on the average. This shows that the number of handshake messages for a given key exchange suite is more significant than the data size, on the overall handshake duration for GSM CSD data bearer systems.

4.3.2. Data Transmission Time Analysis for GPRS Bearer

Data transmission time for the GPRS bearer is predicted analytically by using the timing data gathered from the performance tests over GPRS bearer. Figure 4.24 represents the testbed used. We need to investigate the data rates for download (from server to client) and upload (form client to server) operations, and traversal delay of the

GPRS network. 7 different time slots are selected to perform the tests. These are 12 PM, 2 PM, 4 PM, 6 PM, 8 PM, 10 PM, and 12 AM. Data rates and network traversal delays are measured at these hours. Dial-up networking monitor v2.1 program is used to get the data rate information. A test file of 500 KB size is uploaded to and downloaded from a server on Internet to learn the data rates. Then the ping program is used to learn the RTT values. Traversal delay is computed by using the data transmission rates and RTT values. Performance tests are explained in detail below.

500 KB test file is uploaded to the Internet server and the average upload data rate is measured for each selected time slot. The same test file is downloaded from the Internet server and the average download rate is measured for each of 7 seven time slot. As mentioned before, ping program was used to measure the RTT values. “ping –n 100 GatewayIP” command is run from a laptop with Windows XP operating system. A hundred ping requests is sent to the server and the client waits for the reply. Average RTT values are learned from the ping results. Laptop is connected with an IR (Infrared) link to the mobile client’s internal modem. Data rate of the IR link between the mobile client and laptop is 916 kbps, therefore it is assumed that the IR link does not cause an extra delay.

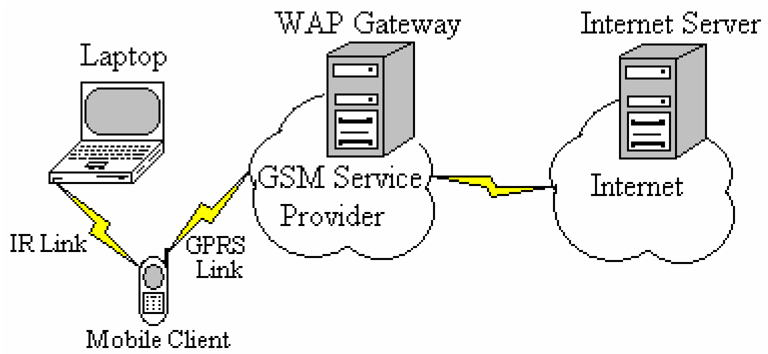


Figure 4.24 GPRS performance evaluation testbed

Measured data transmission rates are given in Table 4.13 for each time slot. These data rates will be used to compute average traversal delays.

Time Slot	Upload Rate (Kbps)	Download Rate (Kbps)
12 PM	10.5	18.77
2 PM	11.2	18.96
4 PM	10.99	10.42
6 PM	10.92	16.3
8 PM	11.33	19.0
10 PM	12.59	28.68
12 AM	12.76	28.89

Table 4.13 Data rates for selected cases

The GSM Service Provider configured the GPRS bearer using the CS-2 coding scheme. Therefore the theoretical data rate for one GPRS channel (time slot) is 14.4 kbps. Nokia 7650 uses one time slot for upload and three time slots for download operations. So, 14.4 kbps is theoretically the maximum data rate for upload operations, and it is 43.2 kbps for download operations. On the other hand, the practical results measured for the test network are given in Table 4.13. We see that the average values of the data rates significantly change during the day. Especially the download data rate is much lower than theoretical maximum value during the busy hours of the day.

The second phase is to analyze the ping RTT values. RTT for the ping request and reply includes the transmission of the 64 byte ping request to the server, traversal delay form the mobile client to the WAP gateway, transmission of the 64 byte ping reply from the server to the mobile client, and traversal delay from the WAP gateway to the mobile client. Assuming that the traversal delay is similar for both directions as stated in the data transmission time model in Section 3.3, we can say that the RTT value is composed of the download and upload time for the 64 byte ping request/reply, and two times the traversal delay.

Measured RTT values for each time slot are given in Figure 4.25 through Figure 4.31 correspondingly.

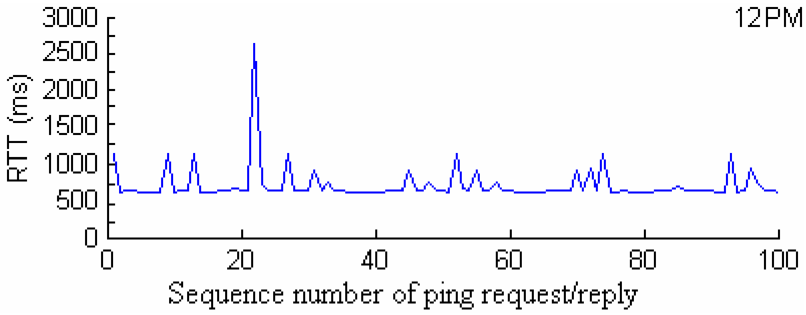


Figure 4.25 RTT values measured at 12 PM (average = 761.8 ms)

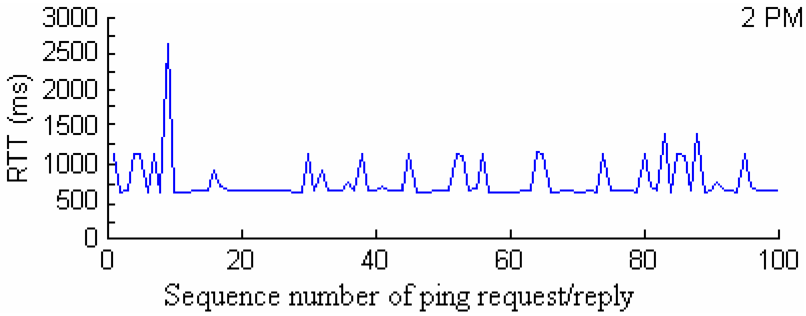


Figure 4.26 RTT values measured at 2 PM (average = 737.3 ms)

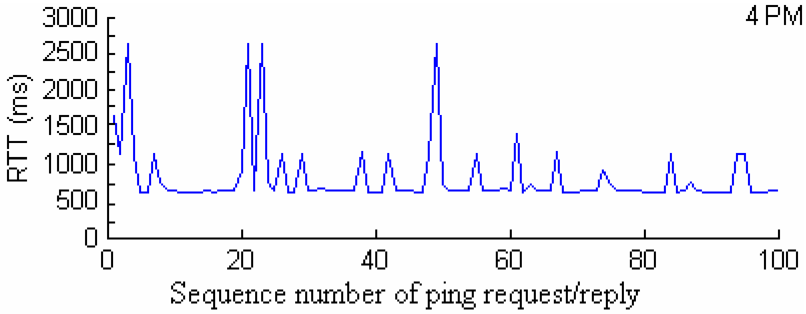


Figure 4.27 RTT values measured at 4 PM (average = 805.9 ms)

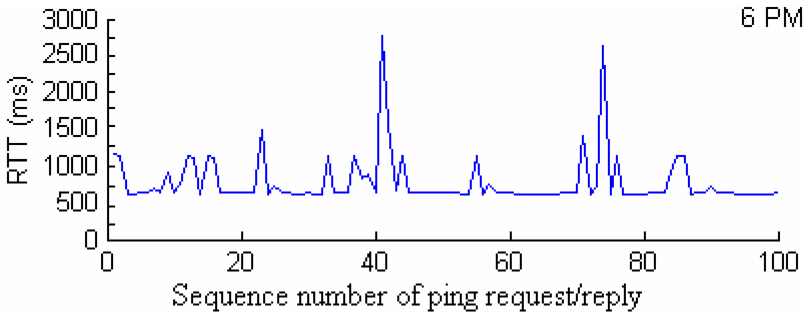


Figure 4.28 RTT values measured at 6 PM (average = 778.9 ms)

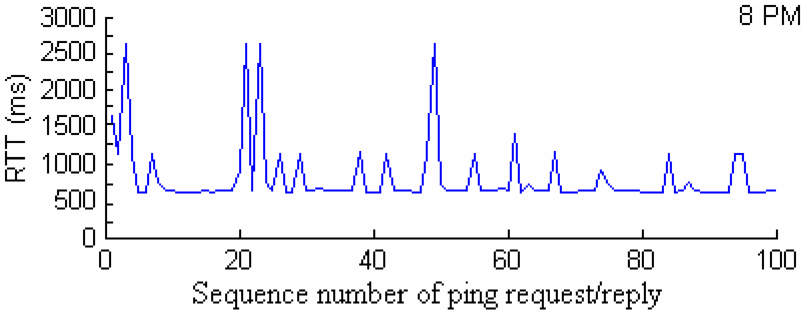


Figure 4.29 RTT values measured at 8 PM (average = 805.9 ms)

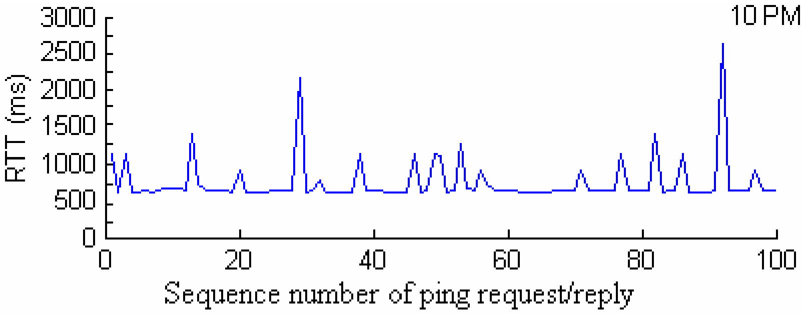


Figure 4.30 RTT values measured at 10 PM (average = 743.1 ms)

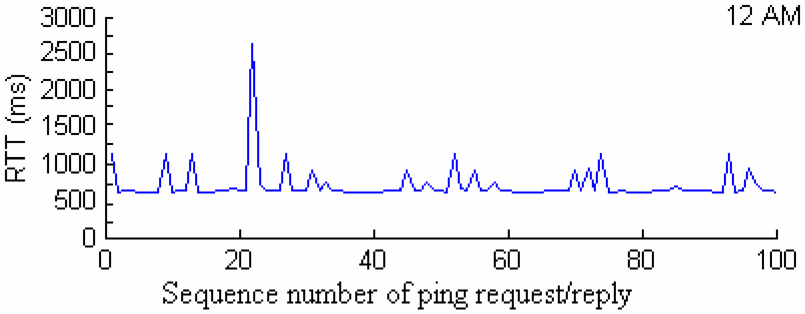


Figure 4.31 RTT values measured at 12 AM (average = 710.1 ms)

We compute the traversal delay by using Eq. (4.2).

$$T_{traversal} = \frac{1}{2} \times \left(T_{RTT} - \frac{64 \times 8}{R_{download}} - \frac{64 \times 8}{R_{upload}} \right) \qquad (4.2)$$

Where;

$T_{traversal}$: One-way traversal delay of the GPRS network

T_{RTT} : Ping RTT value (average value for the considered time slot)

$R_{download}$: Data rate from the server to the client

R_{upload} : Data rate from the client to the server

Eq. (4.2) has been implemented in Matlab 6.0 and using the data rate information given in Table 4.13, traversal delays have been plotted for each case in Figure 4.32 through Figure 4.38.

Traversal delay characteristics for each time slot are summarized in Table 4.14. We see that the average value for the traversal delay is between approximately 350 ms to 400 ms for the selected test cases. It is important to remark that (i) the standard deviation of the traversal delay is significantly high and, (ii) it is always possible to face with high traversal delays throughout the day.

Time Slot	Traversal Delay Mean Value (ms)	Traversal Delay Standard Deviation (ms)
12 PM	376.1	141.2
2 PM	364.1	115.8
4 PM	397.0	213.9
6 PM	384.5	175.7
8 PM	398.4	213.9
10 PM	367.9	151.4
12 AM	351.4	120.1

Table 4.14 Traversal delay characteristics for test cases

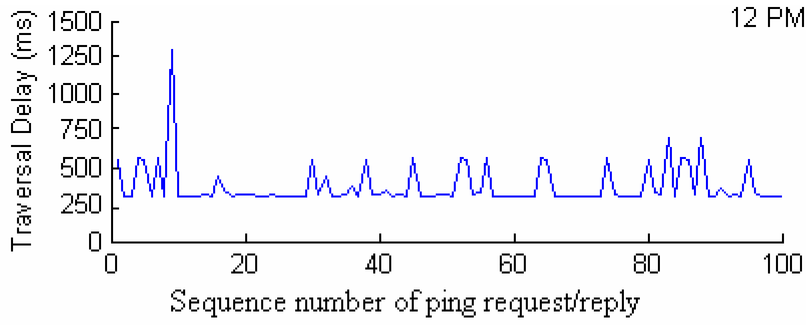


Figure 4.32 Traversal Delays measured at 12 PM (average = 376.1 ms)

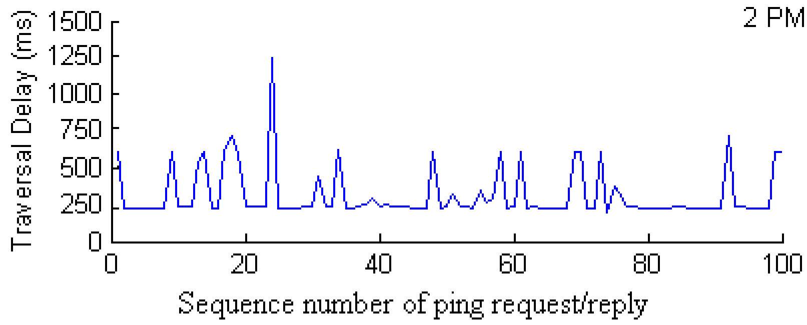


Figure 4.33 Traversal Delays measured at 2 PM (average = 364.1 ms)

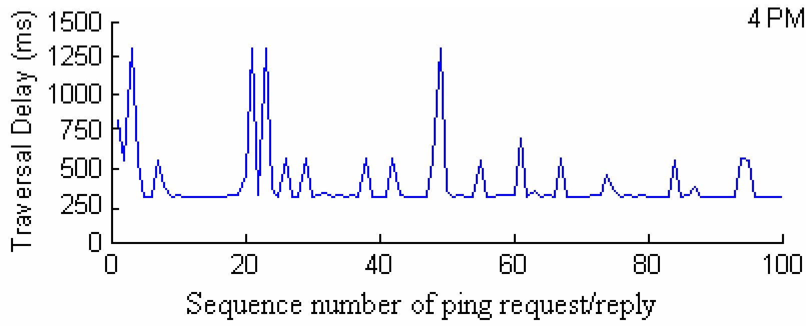


Figure 4.34 Traversal Delays measured at 4 PM (average = 397.0 ms)

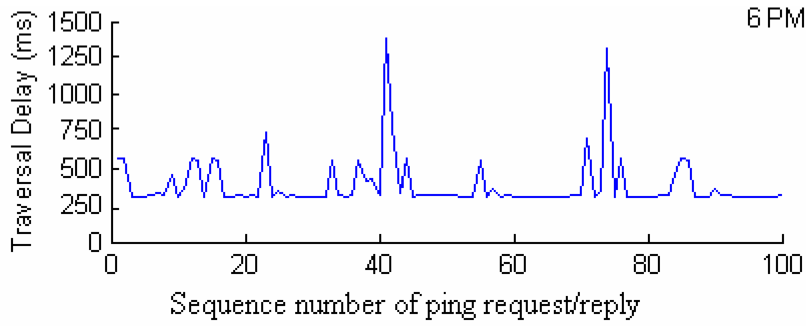


Figure 4.35 Traversal Delays measured at 6 PM (average = 384.5 ms)

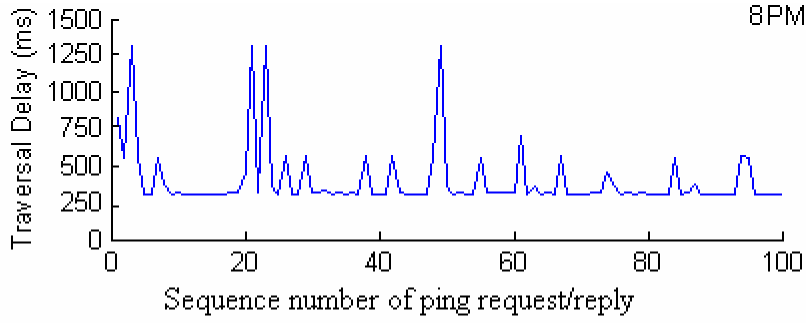


Figure 4.36 Traversal Delays measured at 8 PM (average = 398.4 ms)

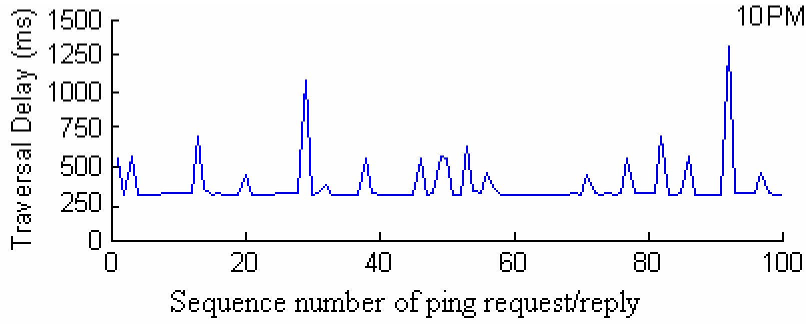


Figure 4.37 Traversal Delays measured at 10 PM (average = 367.9 ms)

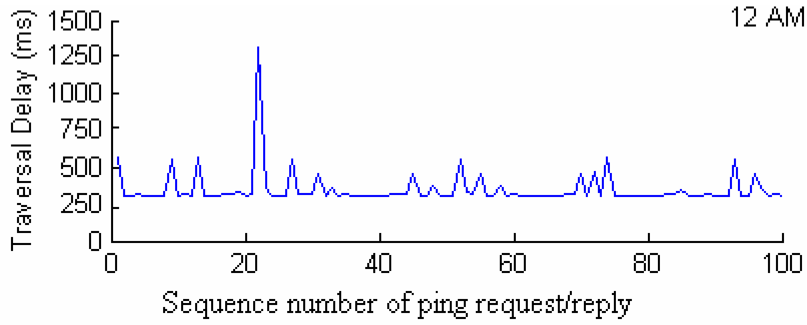


Figure 4.38 Traversal Delays measured at 12 AM (average = 351.4 ms)

4.3.3. Comparison of GSM CSD and GPRS Data Transmission Times

GPRS is generally known to be a superior to GSM CSD bearer when comparing the data rates and pricing. Indeed, GPRS data rates are higher than GSM CSD. However, this data rate advantage can not be realized for the WTLS handshake protocol. This is because of the high valued network traversal delay for both GSM CSD and GPRS bearers. High data rates become an advantage when downloading higher data sizes but there is only a slight change in the data transmission time for WTLS handshake protocol due to the small sized handshake messages. Traversal delays for both bearer types are compared in Table 4.15.

Bearer	Traversal Delay Mean Value (ms)	Traversal Delay Standard Deviation (ms)
GSM CSD	391.3	90.0
GPRS (worst case)	398.4	213.9
GPRS (average case)	376.1	141.2
GPRS (best case)	351.4	120.1

Table 4.15 Comparison of traversal delays for GSM CSD and GPRS bearers

Network traversal delay values are similar for both data bearer types. Data transmission time is significantly determined by this high valued network specific delay rather than the data rate of the bearer. Therefore, decreasing the number of round trips is much more effective on the data transmission time. Therefore, it is possible to minimize the number of round trips by combining the consecutive handshake messages where it is appropriate. The WTLS standard [5] implies that the server handshake messages starting from the server hello message to the server hello done message can be combined in one lower layer message. Retrieving the client certificate from an external certificate store instead of the client itself also decrease the number of round trips over the GSM network between the client and the server.

4.4. Overall Handshake Time Analysis

WTLS handshake protocol overall handshake time was modeled considering three main factors, which were:

i. Client and server processing times

ii. Data transmission time over the channel
iii. Server queue delay

All of the factors given above have been analyzed separately in the previous sections. On the other hand, our aim is to discover the bottlenecks that occur during the WTLS handshake protocol. Therefore, we need to analyze all this factors together to decide on what constitutes a bottleneck for the protocol. Three different groups of figures will be analyzed in this section. The first group of figures Figure 4.39, Figure 4.40 and Figure 4.41 give the overall processing times for category#1, category#2 and category#3 key exchange suites for GSM CSD bearer. There is no server load in these figures and the server is busy with only one handshake operation. It is possible to analyze two main factors of the WTLS handshake protocol performance from these figures, namely the processing times and data transmission times. Overall handshake times for measured best case, worst case, average case GPRS and GSM CSD bearer are compared in Figure 4.42 and Figure 4.43.

Queue delay affects on the overall handshake time are analyzed using Figure 4.44, Figure 4.45 and Figure 4.46 for category#3 groups using different type of certificates with different level of security.

Figure 4.39 shows the overall handshake times for mutual authenticated and server authenticated WTLS handshakes using category#1 certificates. Data transmission time is really a big portion of the overall handshake time for all the cases considered, especially for ECDH_ECDSA key exchange suites. Processing time is affective on the overall processing time for only the mutual authenticated WTLS handshake using client and server certificates with 1024 bit RSA public key. Considering the data transmission time over the channel as the first issue discovered, it is worth to analyze the data transmission time in some more detail. It is interesting that the data transmission time varies a little although the data size for the different key exchange suites vary, especially it is almost doubled for the RSA key exchange suite. Data transmission time analysis given in Section 4.3 states that the traversal delays of the network $T_{traversal}$ constitutes 76% of the data transmission time. 2* $T_{traversal}$ is added to the overall handshake time regardless of the data size transmitted. Therefore, the number of handshake messages have more affect on the overall handshake time instead of the data size and data transmission rate. It is also important that the traversal delay of the network $T_{traversal}$ strictly depends on the data bearer used and may have greater or lower values for different types of data bearers. The measurements were done using GSM CSD as the data bearer with 9600 bps data transmission rate. Traversal delays for the GPRS bearer have also been predicted in Section 4.3.2.

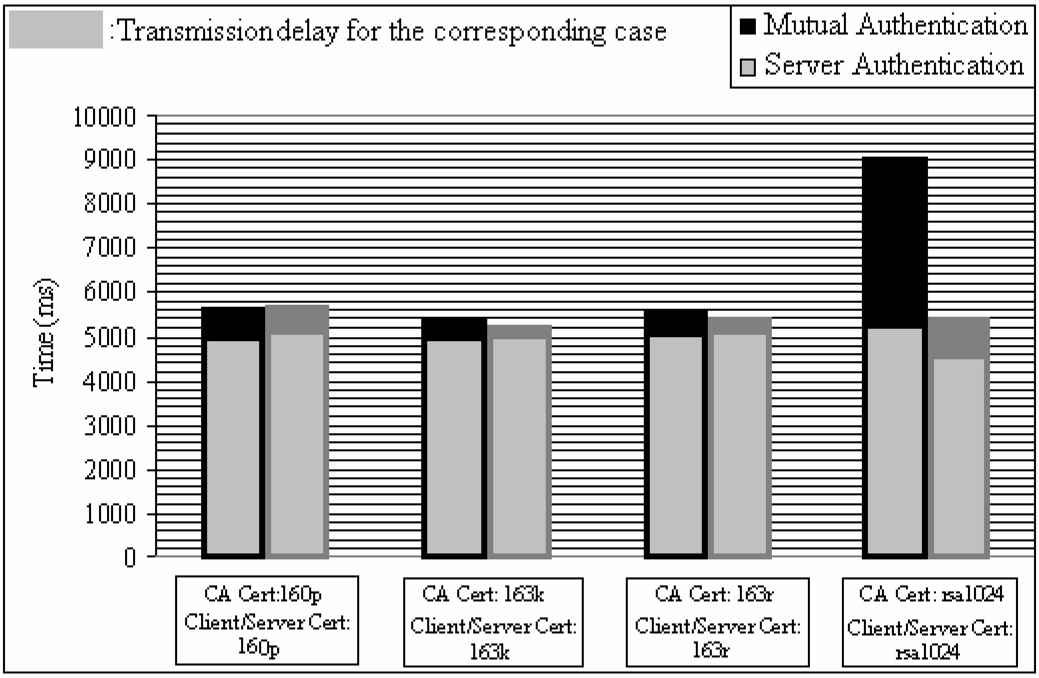


Figure 4.39 Category#1 WTLS handshake overall handshake times

Data transmission time is still a big issue for category#2 certificates as seen in Figure 4.40. ECDH_ECDSA key exchange processing times are not very affective on the overall WTLS handshake time. Mutual authenticated handshake using group#4 certificates (RSA1024 and RSA2048) have processing times that constitute the biggest portion of the handshake time.

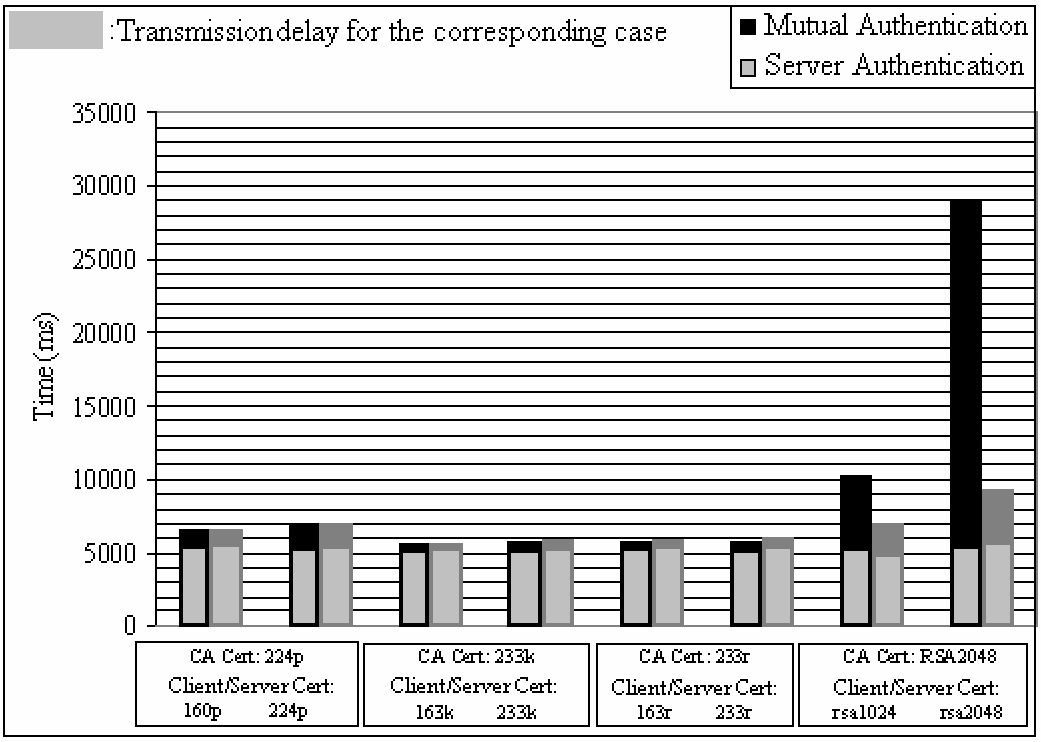


Figure 4.40 Category#2 WTLS handshake overall handshake times

Analyzing the handshake performance when using category#3 certificates, we see in Figure 4.41 that the processing time is really affective on the overall handshake time for group#4 (RSA) certificates. This is due to the increase in the client processing times when generating the CertificateVerify message and other operations using RSA certificates.

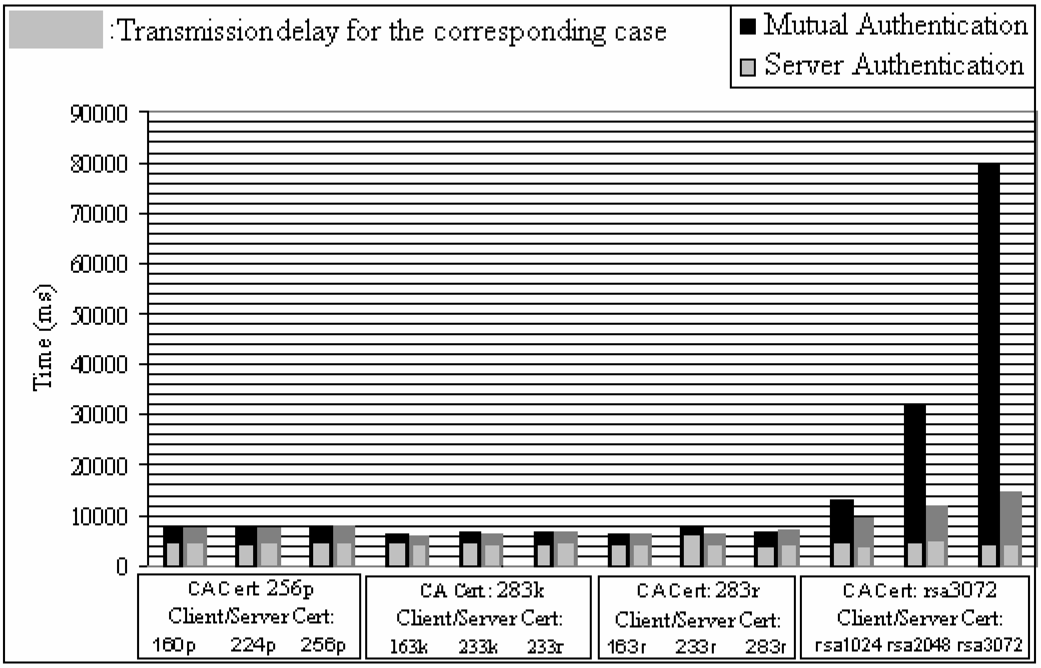


Figure 4.41 Category#3 WTLS handshake overall handshake times

Network traversal delays for the GPRS bearer has been computed in Section 4.3.2. Mutual authenticated WTLS full handshake and server authenticated WTLS full handshake overall times have been computed by using the network traversal delay values given in Table 4.15 and compared in Figure 4.42 and Figure 4.43 respectively. Although there is a significant data rate difference between GSM CSD and GPRS bearers, we see that the overall handshake time is not very different for these bearers because of the similar network traversal delay characteristics.

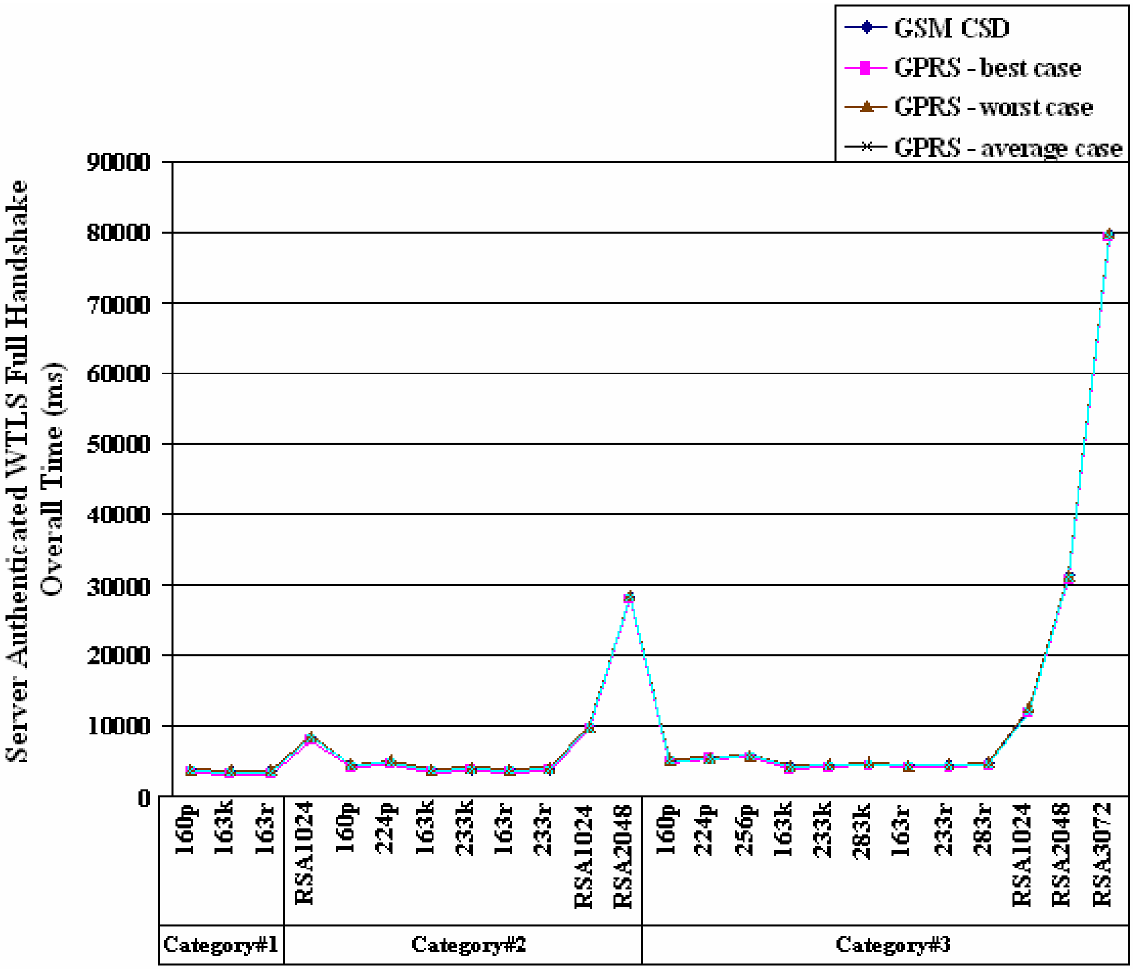


Figure 4.42 Mutual authenticated WTLS handshake overall times for GSM CSD and GPRS

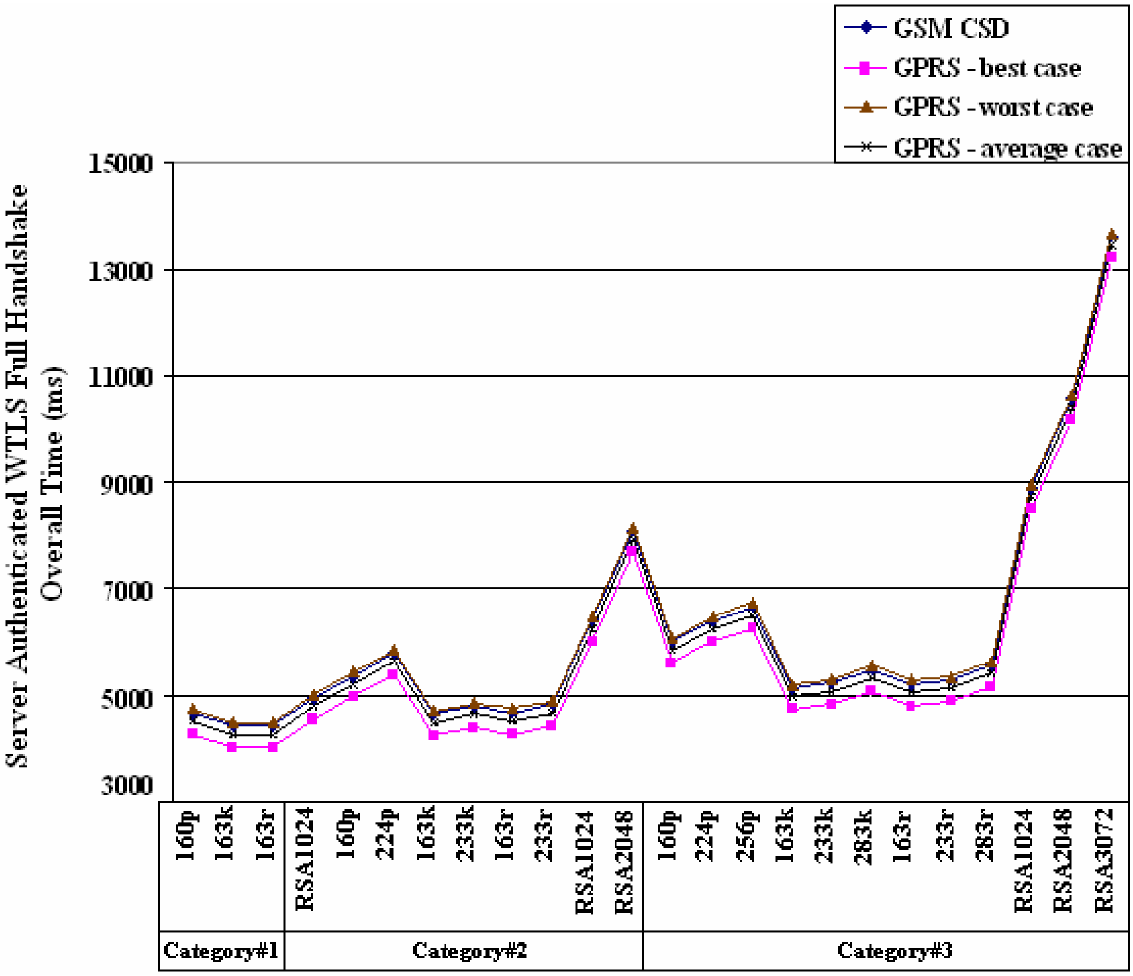


Figure 4.43 Server authenticated WTL handshake overall times for GSM CSD and GPRS

First part of the overall handshake time analysis investigates the affects of the processing times and data transmission times on the overall handshake time. It is clearly seen that the data transmission time has the biggest affect on all of the ECDH_ECDSA key exchange suites using category#1, category#2 and category#3 certificates and RSA key exchange suites using category#1 and category#2 certificates when performing server authenticated WTLS handshake for both GSM CSD and GPRS data bearer types. Mutual authentication using RSA certificates and server authentication using category#3 RSA certificates has the processing time as the biggest portion of the overall handshake time.

Data transmission time is not always the bottleneck for the WTLS handshake protocol although it has the biggest portion of the overall handshake time for our tests over GSM CSD bearer. Due to the high valued traversal delays for both GSM CSD and

GPRS bearers, number of round trips is the most important parameter that significantly affects the data transmission time.

Although the processing times are really small for especially ECDH_ECDSA key exchange suites, the average server processing times for each handshake message are very important when considering the server queue delay. Here begins the second part of the overall handshake time analysis to investigate the affects of the average waiting time in the server queue. WTLS handshake protocol queue delay performance has been analyzed for category#3 certificates. Three cases analyzed below are the members of the groups that offer different levels of security.

Figure 4.44 gives the overall handshake times for varying values of mutual authenticated full handshake requests per second between 0 and 100. Categories and corresponding groups have been defined in Table 3.3. We see that ECDH_ECDSA key exchange suites using 160 bit prime curves have a good queue delay characteristic. On the other hand, ECDH_ECDSA key exchange suites using 163 bit Koblitz, 163 bit random curves and RSA key exchange suite using RSA 1024 has an upper limit of approximately 70 handshake requests per second. This bottleneck is due to the server processing times of the corresponding key exchange suites.

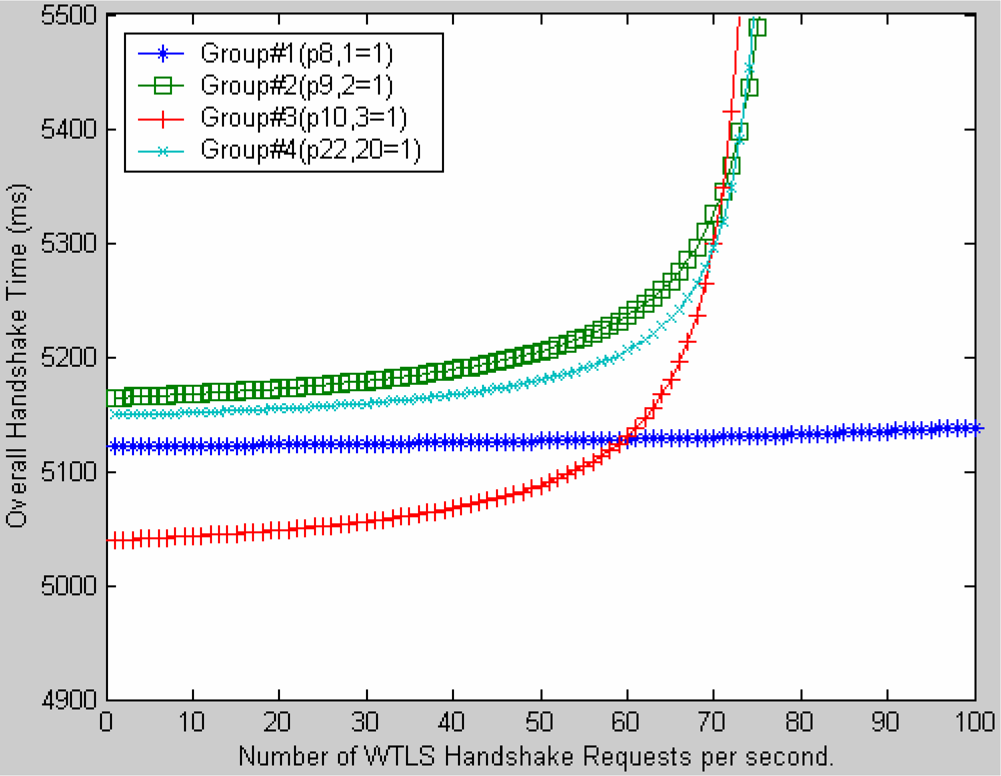


Figure 4.44 Mutual authenticated handshake times with queue delay-1 (category#3)

ECDH_ECDSA key exchange suites' server queue delay characteristics do not change very much as the certificate security strength is increased. Figure 4.45 shows the queue delay affects on the overall handshake time for the mutual authenticated WTLS full handshake when using certificates with 224 bit prime curve, 233 bit Koblitz curve, 233 bit random curve parameters and RSA certificate with 2048 bit public key. 224 bit prime curve certificates again have best queue delay performance, 233 bit Koblitz curve certificates have an upper limit of 80 handshake requests per second, where 233 bit random curve certificates can server 75 handshake requests per second with an acceptable queue delay. Therefore, we see the significant change in the overall handshake time for the RSA key exchange using certificates with 2048 bit RSA public key. The queue delay asymptotically increases after 20 handshake requests per second when 2048 bit RSA certificates are used.

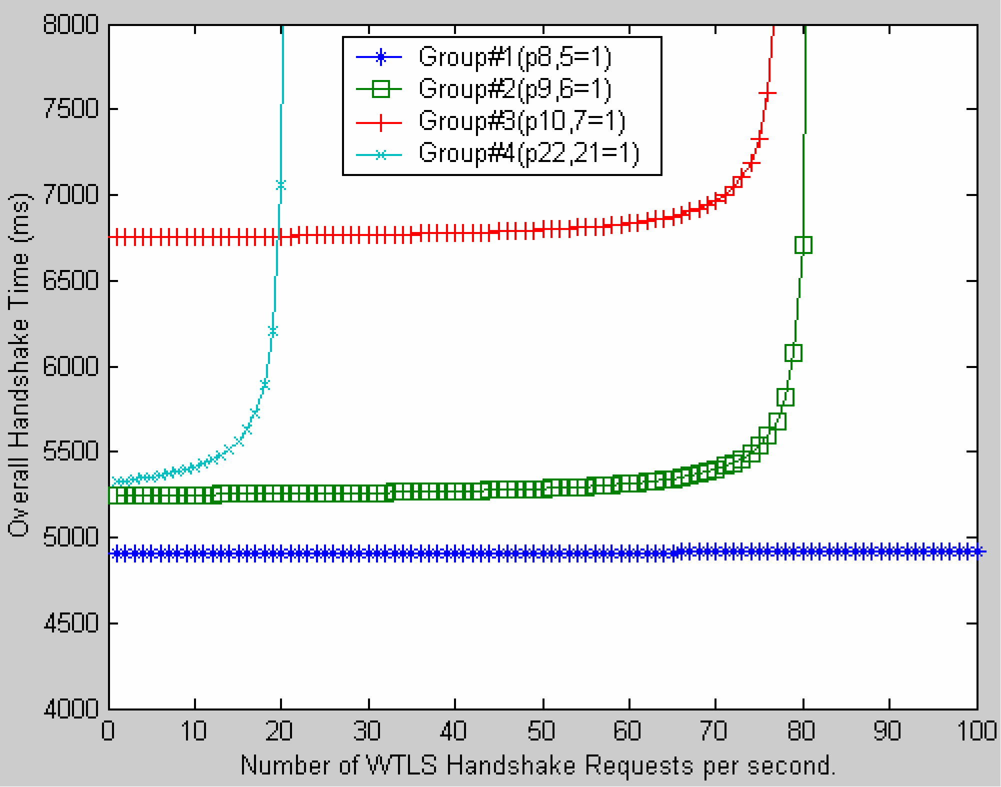


Figure 4.45 Mutual authenticated handshake times with queue delay-2 (category#3)

Mutual authenticated WTLS handshake using RSA certificates with 3072 bit public key has a terrible queue delay performance that the overall handshake time is not acceptable even for two handshake requests per second. ECDH_ECDSA key exchange suites using certificates with 256 bit prime curve, 283 bit Koblitz curve and 283 bit random curve have similar queue delay characteristics as analyzed previously. Mutual authenticated WTLS handshake times are given in Figure 4.46 for these types of systems.

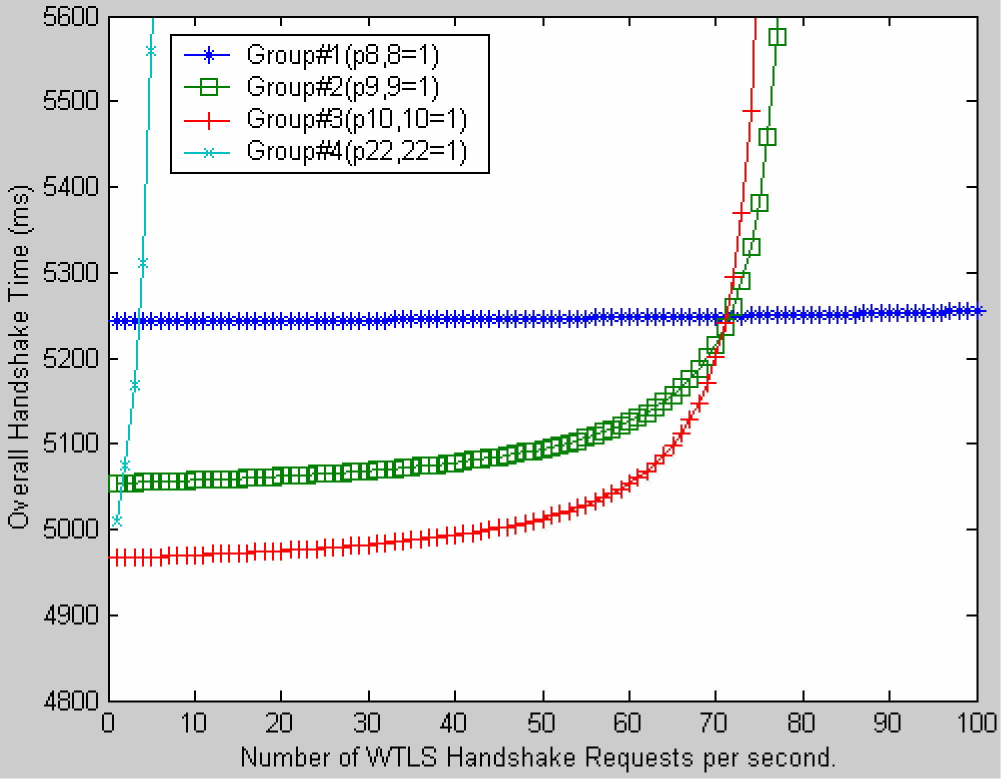


Figure 4.46 Mutual authenticated handshake times with queue delay-3 (category#3)

Server queue delay affects on the mutual authenticated WTLS handshake times have been analyzed up to here. Server authenticated WTLS handshake performance is not that bounded by the server queue delays as it is the case for the mutual authenticated WTLS handshake excluding the server authenticated WTLS handshake using RSA certificates. Average server queue waiting time for the use of RSA key exchange suites during server authenticated WTLS handshake was analyzed in Section 4.2

5. CONCLUSIONS AND FUTURE WORK

WTLS (Wireless Transport Layer Security) [5] is the security protocol that was designed to add valuable security services to WAP [1] sessions. WTLS has two main sub-protocols, namely the Record Protocol and the Handshake Protocol. Handshake protocol is responsible to provide security parameters to the Record Protocol by negotiating on the premaster secret, key refresh period, etc. between the peers. In this thesis WTLS Handshake Protocol performance is evaluated by modeling the different components of the protocol and by analyzing the implementation results parallel to the performance model. The performance model considers three main components that may cause bottleneck for the protocol. These are client and server processing times, server queue waiting time and handshake data transmission time over the channel.

WTLS Handshake Protocol [5] and necessary crypto primitives have been implemented in C++ and the protocol performance has been measured for different cryptosystems. Four different types of certificates have been used during the tests. These are RSA certificates with 1024, 2048 and 3072 bit key sizes and three types of ECC certificates. Three ECC curve types are prime, Koblitz and random curves. 160 bit, 224 bit, and 256 bit prime curves, 163 bit, 233 bit, and 283 bit Koblitz and random curves have been used. 256 bit prime curve, 283 bit Koblitz curve, and 283 bit random curve are not offered by the WTLS standard, these are the stronger ones that provide the level of security needed for today's WAP applications. 160/163 bit ECC curves offer the same level of security with 1024 bit RSA certificates. Similarly, 224/233 bit ECC curves are equivalent to 2048 bit RSA and 256/283 bit ECC certificates offer the same level of security with 3072 bit RSA certificates.

Three categories have been considered during the tests. The first category includes four groups of server/client WTLS certificates with 160 bit prime, 163 bit Koblitz, 163 bit random curve parameters and 1024 bit RSA public key. Certification authority (CA) certificate also has the same key size and public key type. The second category comes with an upgrade of security level. CA certificates in the second category have 224 bit prime, 233 bit Koblitz, 233 bit random curve parameters and 2048 bit RSA public key appropriately with the client/server certificate public key type. Client/server WTLS certificates in this category offer two different level of security, these are 160/224 bit prime curve, 163/233 bit Koblitz and random curve, 1024/2048 bit RSA. The third category has the top level of security that is offered by this thesis, also not offered by the WTLS standard [5]. CA certificates have 256 bit prime, 283 bit Koblitz, 283 bit random curve parameters or 3072 bit RSA public key. All of the three possible key sizes are used for server/client WTLS certificates. These are 160/224/256 bit prime, 163/233/283 bit Koblitz, 163/233/283 bit random ECC curves and 1024/2048/3072 bit RSA. Totally 48 test cases have been considered, 24 is for mutual authenticated WTLS full handshake and the remaining 24 is for server authenticated WTLS full handshake. 4 of 24 test cases (certificates) come from category#1 while 8 is from category#2 and 12 is from category#3 certificates.

Simulation results show that ECC curves perform better than the RSA cryptosystems in WTLS Handshake Protocol. This is an obvious result. Therefore, processing time effects on the WTLS handshake operation will be much more valuable. 1024 bit RSA has somewhat comparable processing time with its rival ECC ones at the client side but it is not possible to say the same thing for the server side. ECC curves' server processing times are significantly lower than RSA server processing times. In addition to that, ECC curves also have different processing time performance. Prime curves are always faster than Koblitz and random curves at the server side. Koblitz curves are the second but random curve processing times are generally similar to the Koblitz curves at the server side. Mutual authenticated WTLS handshake using category#1 certificates with 160 bit prime curve parameters has a server processing time of 10.57 ms, where it is 24.48 ms for 163 bit Koblitz curve, 24.01 ms for 163 bit random curve and 45.86 ms for 1024 bit RSA. Server processing times for both mutual authenticated and server authenticated WTLS handshake using WTLS certificates with prime curve parameters are always lower than half of the Koblitz and random curve server processing times.

The level of security that a certificate offers may be increased by upgrading the issuer CA certificate to a larger key size. This is achieved by the three categories considered during the performance evaluation. We see that the server authenticated WTLS handshake server processing time slightly increases for ECC and RSA certificates, in the case of signing the certificate with a stronger key size. For example, server authenticated WTLS full handshake using category#1(to be verified with 1024 bit RSA) 1024 bit RSA certificate has a server processing time of 36.99 ms, where it is 37.80 ms for category#2(to be verified with 2048 bit RSA), and 38.37 ms for category#3(to be verified with 3072 bit RSA). However, client processing time significantly increases as the CA certificate is upgraded to a stronger key size. It is the same case for both client and server when performing mutual authenticated WTLS handshake.

It is also an interesting result that the prime curves have the worst processing time performance for the client side, where Koblitz and random curves perform similarly. Mutual authenticated WTLS full handshake using category#1 160 bit prime curves have a client processing time of 745.65 ms, where it is 502.74 ms and 489.01 ms for 163 bit Koblitz and random curves respectively. Due to the low processing power of the client, handshake processing time at the client side is extremely higher than the server processing times.

As an expected result, both client and server processing times increase as the key size increases for the same public key cryptosystem. However, increase in the processing time is much more significant for RSA key exchange suites. Implementation results show that the server processing time increases from 38.37 ms to 871.41 ms when using 3072 bit RSA certificates instead of 1024 bit RSA certificates. Therefore, it goes up to 26.67 ms for 283 bit random curve while 163 bit random curve processing time is 8.41 ms. We see that increasing the security level has a significant effect on the overall handshake processing time for RSA key exchange suites, where the increase is tolerable for ECDH_ECDSA key exchange suites.

When mutual authenticated handshake is performed instead of server authenticated handshake, increase in the client processing time is negligible for ECDH_ECDSA key exchange suites. On the other hand, there is a significant increase in the client processing time for the RSA key exchange suites. This is why the mutual authenticated ECDH_ECDSA key exchange suites do not require an extra operation except sending the client certificate to the server, where the RSA key exchange suites require performing a private key operation to generate the CertificateVerify message. Client processing time is measured as 738.95 ms for server authenticated WTLS handshake using category#1 160 bit prime curves, where it is 745.65 ms for the mutual authenticated WTLS handshake. Client processing time goes up to 3780.4 ms for mutual authenticated WTLS handshake using category#1 1024 bit RSA certificates, where it is 1006.78 ms for server authenticated WTLS handshake using the same RSA certificate type. In contrast to the client side, server processing time is always grater for mutual authenticated WTLS handshake.

Using 256 bit prime, 283 bit Koblitz, and 283 bit random ECC curves seems feasible, where 2048 bit and 3072 bit RSA should not be used in WTLS Handshake Protocol for their extremely high processing times at both client and server side.

Implementation results show that server queue delay is a source of bottleneck for the WTLS handshake operation, whereas it does not look like a problem to consider for some cases. Average waiting time in the server queue is negligible when using ECDH_ECDSA key exchange suites for server authenticated WTLS handshake. This is due to the low processing times of ECDH_ECDSA key exchange suites' handshake messages. On the other hand, server queue delay must be considered when using RSA certificates for server authenticated WTLS handshake. Maximum number of server authenticated handshake requests per second is only 6 when using category#3 3072 bit RSA certificates. 21 server authenticated WTLS handshake requests can be served with an average waiting time of 1500 ms, when using category#3 2048 bit RSA certificates. 1024 bit RSA certificates have somewhat acceptable queue delay characteristics for server authenticated WTLS handshake.

When performing mutual authenticated WTLS handshake, ECDH_ECDSA key exchange with prime curves has a good queue delay characteristic due to the low processing times at the server side. Koblitz and random curves have a practical upper limit for the number of mutual authenticated WTLS handshake as well as the RSA certificates. An interesting result is that, average waiting time in the server queue changes slightly as the larger key sizes are used for Koblitz and random curves, where it dramatically gets worse as the RSA key size increases. This is due to the fact that server processing time does not increase very much for mutual authenticated WTLS handshake as larger key sizes are used for ECDH_ECDSA key exchange suites. Therefore, RSA key exchange suites' server processing times are highly vulnerable to increase in key sizes.

Server queue waiting time strictly depends on the average server processing times of the individual handshake messages. Therefore, an improvement is possible by using state-of-the-art implementation of the crypto primitives or upgrading the server processing power.

Handshake data transmission time is an important concern for the overall handshake time. It is obvious that the data transmission time depends on the transmission rate of the data bearer used. However, another characteristic to consider is the traversal delay of the network. This delay is added to the RTT (Round Trip Time) regardless of the data size sent. Measurements show that the traversal delay of the network is much more significant on the transmission time then the data transmission rate for both GSM CSD and GPRS data bearer types. There is no significant difference between the data transmission time over GSM CSD and GPRS for small data sizes. This is due to the similar traversal delay characteristics for both carrier systems. Therefore, the number of handshake messages becomes the most important parameter that affects the WTLS handshake data transmission time.

Comparing the obtained results with the similar research on the subject studied by Levi and Savas in [30], we see differences at ECC curves processing time performance. Prime curves are the best curve option for both client and server side according to their results. Similarly, prime curves have been found as the best option for the server side in this thesis. However, prime curves have the worst processing time performance at the client side. Koblitz curves are the second and random curves always have the worst performance in [30]. There is also a conflict of performance for Koblitz and random curves when compared to results in this thesis. These differences are most probably due to the state-of-the-art implementation in [30]. Both studies have a common result that, prime curves are the best option to use at the server side whether it is state-of-the-art implementation or not. This is also an important result when considering the queue delay performance of an ECDH_ECDSA key exchange suite. Transmission time effects on the overall handshake time are also different from the results in [30]. There are two main reasons for this difference. First of all, overall handshake data sizes are not the same. Only the certificates sizes and other public key related data sizes are considered in [30] where, all of the handshake messages and their corresponding ACK packets are considered in this thesis. The second and more important difference between the two transmission time models is the consideration of network specific traversal delay. Traversal delay significantly affects the overall data transmission time as stated in this thesis.

Performing the tests over a real GSM service provider network comes with the advantages of reflecting the network specific operational costs and message parsing costs to the overall handshake time. Queue delay was not considered only as an extra cost to the handshake time with an assumed arrival rate. Instead, an upper limit for the handshake requests per second has been obtained for each 48 test cases and realized that queue delay is not always a concern for the available key exchange suites. Another contribution is that, the data transmission time can not be modeled by only considering the data transmission rate of the channel. The most striking result of the simulations is that, the effects of the network specific traversal delay rules the data transmission time for both GSM CSD and GPRS bearers. It has been realized that the number of handshake messages is much more important for the overall data transmission time because of the traversal delay that is added to the WSP packet Round Trip Time regardless of the data size sent. So, measured data transmission times for all the key exchange suites are almost equal due to the high valued traversal delay of the test network for GSM CSD bearer. Therefore, this comes with the requirement of specifying the average traversal delay of the network to predict the transmission time for a specific data bearer type.

REFERENCES

[1] WAP Forum, “Wireless Application Protocol Architecture Specification, WAP-100-WAPArch-19980430-a”, WAP Forum Specifications 30-Apr-1998 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[2] eMarketer, “Global Internet and Wireless Users, 2001, 2004 and 2007”, eMarketer Market Researches, Mar. 2002, URL: http://www.emarketer.com/

[3] Datacomm Research Company, “Smart Phones Versus Conventional Wireless Phones”, 2000, URL: http://www.datacommresearch.com/

[4] Durlacher Limited, “Predicted mCommerce Revenues, 2001 - 2005”, URL: http://www.durlacher.com/

[5] WAP Forum, “Wireless Application Protocol Wireless Transport Layer Security Specification, WAP-199-WTLS-20000218-a”, WAP Forum Specifications 18-Feb-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[6] T. Dierks and C. Allen, “The TLS Protocol – Version 1.0”, January 1999, IETF RFC 2246, URL: http://www.ietf.org/rfc/rfc2246.txt

[7] A. Frier, P. Karlton, and P. Kocher, “The SSL 3.0 Protocol”, Netscape Communications Corporation, Nov. 1996, URL: http://home.netscape.com/eng/ssl3/

[8] WAP Forum, “Wireless Application Protocol Wireless Markup Language Specification Version 1.3, WAP-191-WML-20000219-a”, WAP Forum Specifications 19-February-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[9] David Raggett, Arnaud Le Hors, and Ian Jacobs, “HTML 4.0.1 Specification”, W3C Recommendation, 24 December 1999, URL: http://www.w3.org/TR/1999/REC-html401-19991224

[10] François Yergeau, John Cowan, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, “Extensible Markup Language (XML) 1.1”, W3C Recommendation, 4 February 2004, URL: http://www.w3.org/TR/2004/REC-xml11-20040204/

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, June 1999, IETF RFC 2616, URL: http://www.ietf.org/rfc/rfc2616.txt

[12] International Organization for Standardization, "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model - ISO/IEC 7498-1:1994", URL: http://www.iso.org/

[13] WAP Forum, "Wireless Application Protocol Wireless Application Environment Specification Version 1.3, WAP-190-WAESpec", WAP Forum Specifications 29-March-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[14] WAP Forum, "WML Script Specification, WAP-193-WMLS-20001025-a", WAP Forum Specifications 25-October-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[15] WAP Forum, "Wireless Application Protocol Wireless Telephony Application Specification, WAP-169-WTA-20000707-a", WAP Forum Specifications 07-Jul-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[16] WAP Forum, "Wireless Application Protocol Wireless Session Protocol Specification, WAP-203-WSP-20000504-a", WAP Forum Specifications 04-May-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[17] WAP Forum, "Wireless Application Protocol Wireless Transaction Protocol Specification, WAP-201-WTP-20000219-a", WAP Forum Specifications 19-February-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[18] WAP Forum, "Wireless Application Protocol Wireless Datagram Protocol Specification, WAP-200-WDP-20000219-a", WAP Forum Specifications 19-February-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[19] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, February 1978

[20] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, 48(177):203-209, January 1987

[21] V. S. Miller, "Use of Elliptic Curves in Cryptography", *Proceedings of the Advances in Cryptology – CRYPTO'85*, 1985, pp. 417-426

[22] IEEE, “IEEE Std. 1363-2000”, IEEE standard specifications for public key cryptography, 30 Jan. 2000

[23] W. Diffie, and M. Hellman, “New directions in cryptography”, *IEEE Transactions on Information Theory*, 22:644-654, Nov. 1976

[24] National Institute for Standards and Technology, “Digital Signature Standard (DSS) FIPS PUB 186-2”, U.S. Department of Commerce, Jan. 2000, URL: http://csrc.nist.gov/fips

[25] National Institute for Standards and Technology, “Digital Signature Standard (DSS) FIPS PUB 186”, U.S. Department of Commerce, 1994, URL: http://csrc.nist.gov/fips

[26] A. K. Lenstra, E. R. Verheul, “Selecting Cryptographic Key Sizes”, *Proceedings of the Public Key Cryptography*, January 2000

[27] National Institute of Standards and Technology, “Recommended Elliptic Curves for Federal Government Use”, May 1999, URL: http://csrc.nist.gov/encryption

[28] National Institute for Standards and Technology, “NIST FIPS PUB 180-1, Secure Hash Standard”, U.S. Department of Commerce, May 1994, URL: http://csrc.nist.gov/fips

[29] B. Schneier, “Applied Cryptography”, Wiley, New York, 1996

[30] A. Levi, and E. Savas, “Performance Evaluation of Public-Key Cryptosystems Operations in WTLS Protocol”, *Proceedings of the Eight IEEE International Symposium on Computers and Communication (ISCC’03)*, 2003

[31] G. Apostolopoulos, V. Peris, and P. Pradhan., “Securing Electronic Commerce: Reducing the SSL Overhead”, *IEEE Network*, (14)4: 8-16 July/August 2000

[32] G. Apostolopoulos, V. Peris, and D. Saha, “Transport Layer Security: How much does it really cost?”, *Proceedings of IEEE Infocom’99*, pp. 717-725, March 1999

[33] SpecWeb96, Standard Performance Evaluation Corp., 1996, URL: http://www.spec.org/osg/web96

[34] I. Herwono, and I. Liebhardt, “Performance of WTLS and its Impact on an M-Commerce Transaction”, *ICICS 2001 – Proceedings of the Third International Conference on Information and Communications Security*, Nov. 2001, pp. 167-171, Xi’an, China, URL: http://www.comnets.rwth-aachen.de

[35] I. Herwono, and I. Liebhardt, "Performance Evaluation of the WAP Security Protocols", *Proceedings of the 10th Aachen Symposium on Signal Theory*, Sept. 2001, pp. 95-100, Aachen, Germany, URL: http://www.comnets.rwth-aachen.de

[36] K. Kant, R. Iyer, and P. Mohapatra, "Architectural Impact of Secure Socket Layer on Internet Servers", *IEEE International Conference on Computer Design (ICCD'00)*, pp. 7-15, Sept. 2000

[37] V. Gupta, S. Gupta, and S. Chang, "Performance Analysis of Elliptic Curve Cryptography for SSL", *WiSe'02*, Atlanta, Georgia, USA, September 28, 2002

[38] "The OpenSSL Project", URL: http://www.openssl.org/

[39] Dimitri Bertsekas, Robert Gallager, "Data Networks", Prentice Hall, NJ, 1987

[40] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen, "TCP Performance Issues over Wireless Links", *IEEE Communications Magazine*, Volume 39, Number 4, 2001, pp. 52-58

[41] D. Fritsch, N. Fikouras, and C. Görg, "Enabling Hand-offs between GSM and IEEE 802.11b bearers with Mobile IP", *Proceedings of the Fourth International Symposium on Wireless Personal Multimedia Communications (WPMC)*, Denmark, 2001,URL:http://www.comnets.uni-bremen.de/~niko/publications/wpmc01-wap.pdf

[42] R. Ludwig, and B. Rathonyi, "Link Layer Enhancements for TCP/IP over GSM", *IEEE Infocom'99*, March 1999

[43] R. Hillebrand, and T. Wierlemann, "Mobile Internet Guide", URL: http://mobileinternetguide.org/"

[44] "Symbian: The Mobile Operating System", URL: http://www.symbian.com/

[45] "Kannel: Open Source WAP and SMS Gateway", URL: http://www.kannel.org/

[46] "Cygwin: Linux-like environment for Windows", URL: http://www.cygwin.com/

[47] "ARM Processors", URL: http://www.arm.com/

REFERENCES

[1] WAP Forum, “Wireless Application Protocol Architecture Specification, WAP-100-WAPArch-19980430-a”, WAP Forum Specifications 30-Apr-1998 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[2] eMarketer, “Global Internet and Wireless Users, 2001, 2004 and 2007”, eMarketer Market Researches, Mar. 2002, URL: http://www.emarketer.com/

[3] Datacomm Research Company, “Smart Phones Versus Conventional Wireless Phones”, 2000, URL: http://www.datacommresearch.com/

[4] Durlacher Limited, “Predicted mCommerce Revenues, 2001 - 2005”, URL: http://www.durlacher.com/

[5] WAP Forum, “Wireless Application Protocol Wireless Transport Layer Security Specification, WAP-199-WTLS-20000218-a”, WAP Forum Specifications 18-Feb-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[6] T. Dierks and C. Allen, “The TLS Protocol – Version 1.0”, January 1999, IETF RFC 2246, URL: http://www.ietf.org/rfc/rfc2246.txt

[7] A. Frier, P. Karlton, and P. Kocher, “The SSL 3.0 Protocol”, Netscape Communications Corporation, Nov. 1996, URL: http://home.netscape.com/eng/ssl3/

[8] WAP Forum, “Wireless Application Protocol Wireless Markup Language Specification Version 1.3, WAP-191-WML-20000219-a”, WAP Forum Specifications 19-February-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[9] David Raggett, Arnaud Le Hors, and Ian Jacobs, “HTML 4.0.1 Specification”, W3C Recommendation, 24 December 1999, URL: http://www.w3.org/TR/1999/REC-html401-19991224

[10] François Yergeau, John Cowan, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, “Extensible Markup Language (XML) 1.1”, W3C Recommendation, 4 February 2004, URL: http://www.w3.org/TR/2004/REC-xml11-20040204/

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1”, June 1999, IETF RFC 2616, URL: http://www.ietf.org/rfc/rfc2616.txt

[12] International Organization for Standardization, "Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model - ISO/IEC 7498-1:1994", URL: http://www.iso.org/

[13] WAP Forum, "Wireless Application Protocol Wireless Application Environment Specification Version 1.3, WAP-190-WAESpec", WAP Forum Specifications 29-March-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[14] WAP Forum, "WML Script Specification, WAP-193-WMLS-20001025-a", WAP Forum Specifications 25-October-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[15] WAP Forum, "Wireless Application Protocol Wireless Telephony Application Specification, WAP-169-WTA-20000707-a", WAP Forum Specifications 07-Jul-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[16] WAP Forum, "Wireless Application Protocol Wireless Session Protocol Specification, WAP-203-WSP-20000504-a", WAP Forum Specifications 04-May-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[17] WAP Forum, "Wireless Application Protocol Wireless Transaction Protocol Specification, WAP-201-WTP-20000219-a", WAP Forum Specifications 19-February-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[18] WAP Forum, "Wireless Application Protocol Wireless Datagram Protocol Specification, WAP-200-WDP-20000219-a", WAP Forum Specifications 19-February-2000 version, URL: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

[19] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120-126, February 1978

[20] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, 48(177):203-209, January 1987

[21] V. S. Miller, "Use of Elliptic Curves in Cryptography", *Proceedings of the Advances in Cryptology – CRYPTO'85*, 1985, pp. 417-426

[22] IEEE, “IEEE Std. 1363-2000”, IEEE standard specifications for public key cryptography, 30 Jan. 2000

[23] W. Diffie, and M. Hellman, “New directions in cryptography”, *IEEE Transactions on Information Theory*, 22:644-654, Nov. 1976

[24] National Institute for Standards and Technology, “Digital Signature Standard (DSS) FIPS PUB 186-2”, U.S. Department of Commerce, Jan. 2000, URL: http://csrc.nist.gov/fips

[25] National Institute for Standards and Technology, “Digital Signature Standard (DSS) FIPS PUB 186”, U.S. Department of Commerce, 1994, URL: http://csrc.nist.gov/fips

[26] A. K. Lenstra, E. R. Verheul, “Selecting Cryptographic Key Sizes”, *Proceedings of the Public Key Cryptography*, January 2000

[27] National Institute of Standards and Technology, “Recommended Elliptic Curves for Federal Government Use”, May 1999, URL: http://csrc.nist.gov/encryption

[28] National Institute for Standards and Technology, “NIST FIPS PUB 180-1, Secure Hash Standard”, U.S. Department of Commerce, May 1994, URL: http://csrc.nist.gov/fips

[29] B. Schneier, “Applied Cryptography”, Wiley, New York, 1996

[30] A. Levi, and E. Savas, “Performance Evaluation of Public-Key Cryptosystems Operations in WTLS Protocol”, *Proceedings of the Eight IEEE International Symposium on Computers and Communication (ISCC’03)*, 2003

[31] G. Apostolopoulos, V. Peris, and P. Pradhan., “Securing Electronic Commerce: Reducing the SSL Overhead”, *IEEE Network*, (14)4: 8-16 July/August 2000

[32] G. Apostolopoulos, V. Peris, and D. Saha, “Transport Layer Security: How much does it really cost?”, *Proceedings of IEEE Infocom’99*, pp. 717-725, March 1999

[33] SpecWeb96, Standard Performance Evaluation Corp., 1996, URL: http://www.spec.org/osg/web96

[34] I. Herwono, and I. Liebhardt, “Performance of WTLS and its Impact on an M-Commerce Transaction”, *ICICS 2001 – Proceedings of the Third International Conference on Information and Communications Security*, Nov. 2001, pp. 167-171, Xi’an, China, URL: http://www.comnets.rwth-aachen.de

[35] I. Herwono, and I. Liebhardt, "Performance Evaluation of the WAP Security Protocols", *Proceedings of the 10th Aachen Symposium on Signal Theory*, Sept. 2001, pp. 95-100, Aachen, Germany, URL: http://www.comnets.rwth-aachen.de

[36] K. Kant, R. Iyer, and P. Mohapatra, "Architectural Impact of Secure Socket Layer on Internet Servers", *IEEE International Conference on Computer Design (ICCD'00)*, pp. 7-15, Sept. 2000

[37] V. Gupta, S. Gupta, and S. Chang, "Performance Analysis of Elliptic Curve Cryptography for SSL", *WiSe'02*, Atlanta, Georgia, USA, September 28, 2002

[38] "The OpenSSL Project", URL: http://www.openssl.org/

[39] Dimitri Bertsekas, Robert Gallager, "Data Networks", Prentice Hall, NJ, 1987

[40] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen, "TCP Performance Issues over Wireless Links", *IEEE Communications Magazine*, Volume 39, Number 4, 2001, pp. 52-58

[41] D. Fritsch, N. Fikouras, and C. Görg, "Enabling Hand-offs between GSM and IEEE 802.11b bearers with Mobile IP", *Proceedings of the Fourth International Symposium on Wireless Personal Multimedia Communications (WPMC)*, Denmark, 2001,URL:http://www.comnets.uni-bremen.de/~niko/publications/wpmc01-wap.pdf

[42] R. Ludwig, and B. Rathonyi, "Link Layer Enhancements for TCP/IP over GSM", *IEEE Infocom'99*, March 1999

[43] R. Hillebrand, and T. Wierlemann, "Mobile Internet Guide", URL: http://mobileinternetguide.org/"

[44] "Symbian: The Mobile Operating System", URL: http://www.symbian.com/

[45] "Kannel: Open Source WAP and SMS Gateway", URL: http://www.kannel.org/

[46] "Cygwin: Linux-like environment for Windows", URL: http://www.cygwin.com/

[47] "ARM Processors", URL: http://www.arm.com/