Universidad Zaragoza

Proyecto Final de Carrera
Ingeniería Informática
Curso 2010-2011

Consumo energético de algoritmos criptográficos y protocolos de seguridad en dispositivos móviles Symbian

Pedro Miranda Arto

Director: Matti Siekkinen (Aalto University, Helsinki)
Ponente: Sergio Ilarri

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Diciembre de 2010

Agradecimientos

A mi padres, Carmen y Pedro, este trabajo es gracias a ellos.

A mi hermana Pilar, por saber siempre cómo incentivarme con nuevos retos.

A todos mis amigos, gracias a ellos he aprendido muchas cosas, sobre todo cómo arreglar sus ordenadores.

A mis compañeros de carrera, por hacer del CPS un lugar un poco menos duro.

Y por último pero no menos importante a Sergio Ilarri, por su implicación y sus indicaciones.

En Zaragoza, diciembre de 2010

Pedro Miranda

Índice del documento

1 Introducción **1**

1.1 Desarrollo del problema y objetivos 2

1.2 Organización de este documento 2

2 Contexto Tecnológico **5**

2.1 Seguridad informática y sus objetivos 5

2.2 Métodos criptográficos 6

2.2.1 Criptografía simétrica 6

2.2.2 Message Digest y Message Authentication Code (MAC) 6

2.2.3 Criptografía asimétrica 7

2.3 Secure Socket Layer (SSL) 7

2.4 Interfaces de red: 3G y WLAN 8

2.4.1 Interfaz 3G 8

2.4.2 Interfaz WLAN 9

2.5 Seguridad y energía 10

2.5.1 Consumo energético de algoritmos criptográficos 10

2.5.2 Consumo energético de 3G y WLAN 11

3 Metodología **13**

3.1 Tecnologías y herramientas utilizadas 13

3.1.1 Symbian 13

3.1.2 Nokia N95 13

3.1.3 Carbide.C++ IDE 14

3.1.4 Open C/C++ 14

3.1.5 OpenSSL 14

3.1.6 Nokia Energy Profiler 15

3.2 Entorno 15
3.2.1 Suites criptográficas elegidas 15
3.2.2 Diseño de los diferentes escenarios 16
3.3 Configuración experimental 19
3.3.1 Diseño de las aplicaciones 21
3.3.2 Recolección de datos y muestras 22
3.4 Diagrama temporal y de esfuerzo 22

4 Resultados Experimentales **27**
4.1 Escenario: Caso local 27
4.1.1 Datos intercambiados 27
4.1.2 Consumo energético 28
4.2 Escenario: Casos remotos 29
4.2.1 Estudio del perfil de los servicios remotos 29
4.2.2 Datos intercambiados 29
4.2.3 Consumo energético 29
4.2.4 Tiempos de ejecución paso a paso 33
4.2.5 Comparando WLAN y 3G 37
4.3 Rendimiento y coste adicional 38
4.3.1 Datos adicionales en SSL 38
4.3.2 Rendimiento en SSL 39
4.3.3 Consumo energético adicional en SSL 40
4.3.4 Consumo energético de cada uno de los componentes 41

5 Conclusiones **45**
5.1 Conclusiones a nivel técnico 45
5.2 Conclusiones a nivel personal 46
5.3 Trabajo futuro 46

Abreviaturas y acrónimos **49**

Bibliografía **52**

Anexos **54**

A El protocolo de Handshake en SSL **55**

B Artículo: *TLS and Energy Consumption On a Mobile Device: A Measurement Study* **59**

C Memoria en Inglés **67**

D Evaluación **181**

Índice de tablas

3.1 Suites criptográficas elegidas 16

3.2 Desglose de las horas totales empleadas en el proyecto 25

4.1 Información intercambiada en bytes durante la fase de handshake en el caso local 28

4.2 Perfil completo de los diferentes servicios remotos analizados 30

4.3 Información intercambiada en la fase de handshake para los casos remotos 31

4.4 Perfil de los datos intercambiados entre el cliente y el servidor en una conexión SSL 39

4.5 Rendimiento del protocolo SSL y un canal no seguro TCP 40

4.6 Coste energético de diferentes operaciones criptográficas 42

5.1 Equivalencias de seguridad (longitud de clave en bits) entre criptografía simétrica, de curva elíptica y de clave pública 47

A.1 Ejemplos de suites criptográficas 55

Índice de figuras

3.1 Nokia N95 14
3.2 Capturas de pantalla del Nokia Energy Profiler en funcionamiento 15
3.3 Configuración experimental entre el cliente y el servidor 20
3.4 Visión general de la comunicación entre el cliente, el servidor y Nokia Energy Profiler 23
3.5 Diagrama de Gantt del proyecto 24
3.6 Diagrama de tiempos invertidos 25

4.1 Consumo energético en la fase de handshake para el caso local 28
4.2 Consumo energético en diferentes servicios remotos utilizando WLAN . . . 32
4.3 Consumo energético en diferentes servicios remotos utilizando 3G 33
4.4 Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando RSA con WLAN 34
4.5 Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando DHE con WLAN 35
4.6 Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando RSA con 3G 36
4.7 Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando DHE con 3G 36
4.8 Comparación del consumo energético entre WLAN y 3G 37
4.9 Comparación de los tiempos de ejecución de la fase de handshake entre WLAN y 3G 37
4.10 Comparación del rendimiento utilizando SSL y un canal no seguro TCP . . 40
4.11 Coste energético adicional del uso de SSL utilizando WLAN 41
4.12 Coste energético adicional del uso de SSL utilizando 3G 41
4.13 Estudio individual del coste energético de los distintos componentes empleados en SSL para el caso RSA 43

4.14 Estudio individual del coste energético de los distintos componentes empleados en SSL para el caso DHE 43

A.1 Pasos e información intercambiada en la fase de handshake 56

Capítulo 1. Introducción

En los últimos años, la telefonía móvil ha redefinido la forma de comunicación entre las personas. Los dispositivos electrónicos, a través de los cuales se establece dicha comunicación, han evolucionado vertiginosamente en potencia, rendimiento y sobre todo en nuevas funcionalidades, pero siguen estando limitados por la autonomía que les proporciona la duración de la batería. Mientras la capacidad de procesamiento se incrementa en un 200% cada 18 meses siguiendo la Ley de Moore, el rendimiento de las baterías sólo se ha visto mejorado en un 80% en los últimos 10 años [Fit07].

Esta mejora del rendimiento y la llegada de Internet a estos dispositivos ha hecho posible la comunicación a través de diferentes canales, la búsqueda de información o la posibilidad de realizar compras, todo ello de una forma segura y confidencial. Dichas conexiones seguras, bien utilizando conexiones cableadas o inalámbricas, se consiguen mediante la utilización de protocolos de seguridad, basados en algoritmos criptográficos. Estos algoritmos son seleccionados basándose en los objetivos de seguridad definidos en el protocolo de seguridad a utilizar. Entre ellos se incluyen algoritmos de encriptación simétricos y asimétricos, utilizados para proporcionar autenticación y encriptación de los datos, así como algoritmos basados en funciones *hash*, y, de esa manera, conseguir integridad en los mensajes intercambiados.

En la actualidad, el consumo energético en dispositivos móviles es una de los principales preocupaciones de los fabricantes de dispositivos móviles. De la misma manera, la seguridad en las comunicaciones se posiciona como una área muy importante en materia de investigación y desarrollo. El uso de protocolos de seguridad no sólo afecta al rendimiento de las comunicaciones, sino que también representa un fuerte impacto en el consumo energético en estos dispositivos alimentados por baterías. De este modo, uno de los desafíos más importantes es conseguir un balance entre rendimiento, seguridad y consumo energético, con el fin de obtener un buen rendimiento y niveles de seguridad adaptados al usuario con la mínima cantidad de energía.

En este contexto, Nokia corporation encargó un proyecto de investigación a la Aalto University School of Science and Technology (Helsinki, Finlandia), para analizar el consumo energético de diferentes protocolos de seguridad y algoritmos criptográficos en la plataforma móvil Symbian.

1.1 Desarrollo del problema y objetivos

El objetivo principal de este estudio es realizar un exhaustivo análisis del impacto energético derivado de la creación de una conexión segura utilizando el protocolo de seguridad *Secure Sockets Layer (SSL)* [DA99]. Este protocolo utiliza diferentes técnicas criptográficas para proteger la información intercambiada entre los diferentes pares conectados, proporcionando no sólo seguridad y privacidad sino también legitimidad e integridad. Además, el empleo de este protocolo facilita el uso de los servicios de seguridad más utilizados, de tal forma que la necesidad de conocimientos criptográficos se ve minimizada. Así pues, el objetivo es estudiar los criptosistemas más utilizados y analizar su consumo en diferentes escenarios.

El estudio comienza con el desarrollo de una aplicación cliente en un terminal móvil Symbian, capaz de comprender y de realizar conexiones seguras utilizando el protocolo SSL con diferentes servicios *online*, tales como correo electrónico o redes sociales y, de esa manera, realizar mediciones del consumo energético utilizado en el establecimiento de dichas conexiones. Dichas mediciones se llevan a cabo con el *Nokia Energy Profiler*, una aplicación desarrollada por Nokia que se ejecuta en el propio terminal y permite la medición de diferentes parámetros sin la necesidad de ninguna herramienta externa.

Además, el desarrollo incluye también un servidor capaz de manejar diferentes conexiones seguras con el cliente móvil, de manera que sea posible modelar y medir los diferentes escenarios a estudiar: envío/recepción de datos, autenticación del cliente, reanudar una conexión segura previamente establecida y la selección de diferentes suites criptográficas a utilizar.

Para el establecimiento de las comunicaciones con los diferentes servicios y el servidor SSL, se utilizan las interfaces de red del terminal móvil WLAN y 3G, para así comparar también sus diferentes consumos energéticos y el impacto debido a las distintas tecnologías utilizadas de dichas comunicaciones.

1.2 Organización de este documento

Este trabajo esta dividido en los siguiente capítulos:

- Capítulo 1 - *Introducción* proporciona una idea general de la problemática estudiada y los objetivos del proyecto.
- Capítulo 2 - *Contexto Tecnológico* incluye una introducción a los diferentes algoritmos criptográficos, los objetivos y las diferentes técnicas en ese campo, así como un estudio del protocolo SSL y de las características de seguridad que proporciona. También contiene una introducción al consumo energético de las diferentes interfaces de red, tanto WLAN como 3G, y de diferentes protocolos criptográficos analizados en estudios anteriores.
- Capítulo 3 - *Metodología* contiene una descripción de todas las tecnologías utilizadas

para la creación de la configuración base del proyecto, el análisis de los requerimientos y la metodología de medición utilizada.

- Capítulo 4 - *Resultados Experimentales* detalla los diferentes resultados y las conclusiones obtenidas de los diferentes experimentos.
- Capítulo 5 - *Conclusiones* incluye las conclusiones generales de todo el trabajo desarrollado y una posible vía de investigación futura.

Al final de esta memoria, se incluyen también varios anexos:

- Anexo A - *El protocolo de Handshake en SSL* detalla en profundidad cómo se realiza la fase de *handshake* dentro del protocolo SSL.
- Anexo B - *Artículo* incluye la publicación *TLS and Energy Consumption On a Mobile Device: A Measurement Study* enviada al *Communications QoS, Reliability and Modeling Symposium* enmarcado en el congreso *IEEE International Conference on Communications ICC2011*, pendiente de aprobación.
- Anexo C - *Memoria en Inglés* incluye la memoria íntegra *Energy consumption of cryptographic algorithms and security protocols in Symbian mobile devices*, presentada en la *Aalto University School of Science and Technology*, incluyendo todos los resultados y diversos anexos.
- Anexo D - *Evaluación* incluye la evaluación obtenida en la *Aalto University School of Science and Technology*.

Capítulo 2. Contexto Tecnológico

El objetivo de este capítulo es el de proporcionar una definición de lo que es la seguridad informática y los objetivos que hay detrás de esta, dando una definición de los diferentes algoritmos y protocolos utilizados en el campo de la criptografía. Estos incluyen encriptación simétrica y asimétrica, los cuales son usados para proporcionar autenticación y privacidad, así como algoritmos de *message digest* o funciones *hash*, utilizados para proporcionar integridad a los mensajes. Una vez explicados estos mecanismos, se da una introducción al protocolo *Secure Sockets Layer* (SSL), definiendo los diferentes protocolos involucrados y los pasos para establecer una comunicación segura. Además, se proporciona una introducción a los interfaces de red más importantes en los terminales móviles, WLAN y 3G, y sus características. Para finalizar, se incluye un estudio del impacto energético, tanto de la parte criptográfica como de los interfaces de red.

2.1 Seguridad informática y sus objetivos

El objetivo de la seguridad informática es idear y concebir formas de prevenir las debilidades de un sistema que pueden ser explotadas [PP06] a la vez que se permite que la información permanezca segura y accesible. Los objetivos de la seguridad informática están normalmente asociados a las funcionalidades de seguridad requeridas en un sistema o red para proteger datos sensibles o la identidad de los equipos/usuarios dentro de ese sistema. Existen 4 objetivos importantes, incluyendo:

- *Confidencialidad* asegura que los datos se mantienen en secreto ante personas no autorizadas y sólo pueden ser accesibles por usuarios autorizados. Este término es también conocido como *privacidad*.
- *Disponibilidad* garantiza que la información es accesible por parte de usuarios autorizados cuando es necesaria.
- *Integridad* protege los datos de ser modificados por usuarios no autorizados y sólo permite modificaciones de manera autorizada.
- *No rechazo* es el concepto que prueba el origen de la información. Si este objetivo se consigue, se garantiza que la información es original y genuina.

Uno de los desafíos en el desarrollo de un sistema seguro es encontrar un equilibrio correcto entre estos 4 objetivos, los cuales normalmente pueden ser independientes, pueden

solaparse e incluso pueden ser mutuamente exclusivos. Por ejemplo, asegurar una fuerte confidencialidad puede afectar seriamente a la disponibilidad.

2.2 Métodos criptográficos

En las siguientes secciones se trata de dar una introducción básica de los diferentes métodos criptográficos existentes y utilizados en este trabajo. Para un estudio más detallado de los fundamentos de la criptografía y de la nomenclatura, el lector puede encontrar mas información en el Capítulo 2 *Security and energy background* del Anexo C.

2.2.1 Criptografía simétrica

La criptografía simétrica emplea el mismo proceso para encriptar y desencriptar, utilizando la misma clave secreta como entrada. Existen 2 tipos de encriptación simétrica:

- *Cifrado por bloques* (*block cipher*) es un tipo de algoritmo de clave simétrica que transforma un bloque de texto de una longitud determinada en un bloque de texto cifrado de la misma longitud [Lab93]. La función de transformación depende de la clave proporcionada por el usuario, mientras que la longitud fija es conocida como el tamaño de bloque, normalmente 64 bits. El algoritmo más utilizado en el cifrado por bloques es AES [oST01].
- *Cifrado por flujo* (*stream cipher*) utiliza una función que genera un flujo de datos de 1 byte cada vez conocido como *keystream*. Este método utiliza una clave de encriptación como entrada, la cual controla exactamente cómo se genera dicho *keystream*. Uno de los algoritmos más utilizados en el cifrado por flujo es RC4 [Kau99].

Los algoritmos de cifrado por bloques son una solución más conservativa que los de cifrado por flujo, ya que estos han sido estudiados más en profundidad contra ataques criptoanalíticos. Sin embargo, estos últimos tienen un mejor rendimiento.

Se puede encontrar más información detallada acerca de estos métodos y soluciones en la Sección *2.3 Symmetric Cryptography* del Anexo C.

2.2.2 Message Digest y Message Authentication Code (MAC)

Un algoritmo de resumen del mensaje o *Message digest*(también conocido como algoritmo *hash*) es básicamente un procedimiento que toma una longitud arbitraria de datos como entrada y genera como salida una suma de verificación o *checksum*. Este método tiene una propiedad importante: la misma entrada siempre producirá la misma salida. De esta forma, un cambio accidental o intencionado producirá una cambio en dicho *checksum*.

El uso principal de estos *message digest* es el de la creación de firmas digitales (ver firma digital en Sección 2.2.3). Los 2 más utilizados son Message Digest 5 (MD5) [Riv92] y Secure Hash Algorithm 1 (SHA-1 [EJ01]).

Además de su uso como firma digital, estos algoritmos se utilizan como códigos de autenticación de mensaje o *Message Authentication Code (MAC)*, los cuales son un tipo de algoritmos de resumen pero incorporan una clave en su creación. Sus aplicaciones más populares son HMAC-SHA-1 o HMAC-MD5.

Más información acerca de estos algoritmos y de sus diferentes usos está incluida en la Sección *2.4 Message Digest and Message Authentication Codes* del Anexo C.

2.2.3 Criptografía asimétrica

La criptografía asimétrica o de clave pública implica el uso de algoritmos asimétricos. Estos algoritmos se utilizan para crear un par de claves relacionadas: una clave privada secreta y otra pública. Un usuario puede usar la clave privada para codificar un mensaje, y ese mensaje puede ser decodificado únicamente con su correspondiente clave pública.

Este tipo de criptografía es computacionalmente cara. La seguridad de estos algoritmos depende de la longitud de las claves utilizadas, las cuales normalmente son números muy largos.

La criptografía de clave pública tiene 2 usos principales:

- *Protocolo de establecimiento de clave (Key establishment protocolo)*, en el cual cliente y servidor trabajan conjuntamente para establecer la clave de encriptación a utilizar. Dentro de este protocolo se incluyen 2 métodos para realizar dicha operación: Intercambio de clave *(Key exchange)* o Acuerdo de clave *(Key agreement)*.
- *Firma digital*, utilizada para incluir una huella que sólo el emisor puede producir, pero que el receptor puede fácilmente reconocer y verificar que la información incluida es legitima. La firma se genera utilizando la clave privada del emisor, empleando el receptor su clave pública para verificar dicha firma.

Las soluciones más populares son RSA [RSA78] y Diffie-Hellman [WH76]. Mas información sobre estás 2 últimas soluciones y la demostración puede encontrarse en la Sección *2.5 Public Key Cryptography* del Anexo C .

2.3 Secure Socket Layer (SSL)

Secure Socket Layer (SSL) [01] y su sucesor *Transport Security Layer (TLS)* [DA99] es un protocolo que permite establecer un canal seguro entre 2 pares conectados. Este protocolo fue diseñado para proteger datos en tránsito y para identificar los 2 pares involucrados en la comunicación. Los datos son cifrados entre dichos pares, pero la información que un par escribe es exactamente la misma que el otro par recibe. Dicho canal también ofrece transparencia, lo que significa que transmite la información sin cambio alguno. Esta propiedad permite que cualquier protocolo que utilice TCP pueda funcionar sobre SSL con un mínimo de modificaciones.

El protocolo SSL se divide en 2 capas:

1. *El protocolo de Handshake* o *apretón de manos* permite al cliente y al servidor autenticarse uno al otro y negociar los algoritmos criptográficos a utilizar, así como las claves de encriptación necesarias para asegurar el canal. Se recomienda consultar más información acerca de este protocolo en el Anexo A para comprender los resultados de los experimentos realizados en el Capítulo 4.

2. *El protocolo de record* o *de registro* proporciona privacidad y fiabilidad a los datos intercambiados. Es el responsable de transferir la información y de interpretar los diferentes tipos de mensajes de los que el protocolo SSL está formado (ver Sección *3.3 The Record Protocol* en el Anexo C). Este protocolo funciona de tal manera que divide los datos a transferir en fragmentos, cada uno de ellos independientemente protegido. Este comportamiento permite enviar los datos tan pronto como estén listos.

Cabe mencionar que en este documento siempre se hace referencia a este protocolo como SSL, refiriéndose indistintamente tanto a SSLv3 [Cor96] como a la versión estandarizada del protocolo TLS [DA99].

Más información acerca de este protocolo y de sus diferentes usos está incluida en el Capítulo 3 *SSL and TLS* del Anexo C.

2.4 Interfaces de red: 3G y WLAN

Con la irrupción de Internet en los dispositivos móviles, los usuarios quieren estar *en linea* todo el tiempo. Esta tendencia genera un uso continuo e importante de la batería. Las interfaces de red, tales como GSM, 3G o WLAN, son las responsables de las diferentes comunicaciones utilizadas en un dispositivo móvil. En [Fit07] se menciona que estas interfaces son las causantes de aproximadamente el 50% del desgaste de la batería. Así pues, cualquier reducción en el uso de dichos interfaces significará una mejora sustancial en la duración general de la batería.

En este trabajo, solamente se estudian los interfaces 3G y WLAN, ya que estos son los más importantes y más utilizados para la transmisión de datos. Para analizar su consumo energético, es fundamental entender la naturaleza de los mismos.

2.4.1 Interfaz 3G

3G o Tercera generación es una familia de estándares para telecomunicaciones móviles que permite simultáneamente la transmisión de voz y datos con velocidades de transferencia de 14 Mbit/s de bajada y 5.8 Mbit/s de subida [XSK+].

El interfaz 3G funciona en 3 modos diferentes de operación:

- *IDLE* en ausencia de actividad en la red.
- *Dedicated Channel (DCH)* asegura el máximo rendimiento con bajas latencias de transmisión, pero con un coste muy alto de energía.

- *Forward Access Channel (FACH)* comparte el canal con otros dispositivos 3G y se usa cuando hay poco tráfico a transmitir, utilizando alrededor de un 50% de la energía consumida en el estado DCH.

Las transiciones desde DCH a FACH y desde FACH a IDLE son controladas por temporizadores de inactividad, los cuales normalmente se miden en segundos.

Asimismo, el consumo energético de la interfaz de red 3G puede ser cuantificado atendiendo a estos 4 tipos de energía:

1. *Energía de rampa* necesaria para cambiar al estado DCH o de máximo rendimiento
2. *Energía de transmisión* utilizada en la transmisión de datos.
3. *Energía de cola* necesaria para mantener el estado DCH después de la finalización de la transmisión hasta la transición al estado FACH marcada por uno de los temporizadores de inactividad.
4. *Energía de mantenimiento* utilizada para mantener la interfaz de red 3G encendida.

Cabe mencionar que la energía de cola puede ser de hasta un 60% del total de la energía utilizada para transmitir 50KB [BBV09].

2.4.2 Interfaz WLAN

La interfaz de red WLAN o *Wireless Local Area Network* se utiliza para conectar diferentes dispositivos sin la necesidad de la utilización de cables, permitiendo en la mayoría de los casos una conexión directa a Internet. Estos dispositivos se han vuelto muy populares debido a su creciente uso y las capacidades inalámbricas de portátiles y de *smartphones*. El estándar actual IEEE 802.11 define diferentes bandas de frecuencia operacionales, tales como 2.4, 3.6 y 5 GHz. A su vez, existen diferentes protocolos, pero el más extendido es el 802.11g, establecido en una banda de frecuencias de 2.4GHz y operando a una velocidad máxima de 54 Mbit/s, con una rendimiento medio de 22 Mbit/s [XSK+].

El consumo energético en esta interfaz WLAN puede ser cuantificado atendiendo a estos 3 factores:

1. *Escaneo y asociación a una red*: la energía utilizada en la búsqueda de las redes disponibles y la asociación a una de ellas.
2. *Energía de transmisión*: la energía requerida para la transmisión de los datos
3. *Energía de mantenimiento*: la energía necesaria para mantener el interfaz de red WLAN encendido.

Al contrario que con el interfaz 3G, WLAN dispone de sistemas de ahorro de energía (como puede ser *Power Saving Mode (PSM)*) que reducen enormemente el consumo cuando dicho interfaz no esta transmitiendo.

2.5 Seguridad y energía

Los dispositivos de mano o *handhelds* como teléfonos móviles o *PDAs*, están fuertemente limitados por las baterías que incorporan. Componentes como el procesador, la pantalla y su iluminación, la cámara y sobre todo las interfaces de red, representan un impacto enorme en la descarga de la batería. Con todos estos componentes, la capacidad de los terminales para estar conectados durante largos periodos de tiempo se ve afectada dependiendo de su uso. En [Fit07] se nombra que en los últimos 10 años la capacidad de las baterías sólo se ha visto incrementada en un 80%, mientras que las capacidades de los microprocesadores se ven incrementadas un 200% cada 18 meses siguiendo la ley de Moore. Esta mejora en las capacidades de computación permite a los desarrolladores diseñar nuevas aplicaciones con nuevas funcionalidads que pueden tienen importantes costes energéticos dependiendo de las necesidades computacionales.

2.5.1 Consumo energético de algoritmos criptográficos

Como se ha comentado previamente, la llegada de Internet a los dispositivos móviles ha hecho posible la comunicación a través del mismo, la búsqueda de información, la compra de artículos o compartir información de una forma segura y confidencial. Esta privacidad se consigue gracias al uso de algoritmos criptográficos y de protocolos de seguridad, los cuales tienen un importante impacto en la duración de la batería, incrementando el número de cálculos y operaciones en el microprocesador y también aumentando la cantidad de datos intercambiados.

Investigaciones previas [APS99] [PRRJ06] [SGM09], analizan el coste energético de diferentes algoritmos criptográficos así como el rendimiento del protocolo SSL en *PDAs* y en ordenadores personales. Estos estudios muestran diferentes observaciones, de las que se mencionan las más importantes a continuación:

- La criptografía de clave pública tiene los costes energéticos más altos, mientras que los algoritmos *hash* tienen un impacto pequeño en la duración de la batería.
- La elección del tamaño de clave afecta significativamente al consumo energético en algoritmos asimétricos, pero no de igual manera en los simétricos.
- El use de Diffie-Hellman como protocolo de acuerdo de clave durante la fase de negociación o *handshake* mejora la seguridad, pero a causa de incrementar significativamente el coste energético y reduciendo el rendimiento si se compara con otros algoritmos de intercambio de clave como RSA.
- El uso de SSL como protocolo de seguridad afecta significativamente al rendimiento general de la conexión, pero esto puede ser suavizado eligiendo el algoritmo que más se adapte dependiendo de las necesidades de seguridad del usuario y de las capacidades *hardware* del equipo utilizado.

2.5.2 Consumo energético de 3G y WLAN

En [BBV09] se estudia el consumo energético de estos interfaces de red utilizando un Nokia N95, el mismo empleado en este trabajo, con las siguiente conclusiones generales:

- 3G consume significativamente más energía en la descarga de datos que WLAN. Además de la diferente naturaleza de las comunicaciones, el menor ancho de banda comparado con WLAN requiere que la interfaz este operativa más tiempo y, por lo tanto, el consumo es mayor.
- El uso de políticas de ahorro energético en WLAN tales como *Power Saving Mode (PSM)* reduce de forma significativa el coste energético cuando el interfaz no se utiliza.
- En 3G, cerca del 60% de la energía utilizada es *energía de cola*, consumida después de la finalización de una transmisión. Comparado con esto, la *energía de rampa* es sólo una pequeña cantidad y puede ser amortiguada con frecuentes y sucesivas transferencias, dentro del intervalo que marca el temporizador de inactividad.
- La energía de transmisión es proporcional al tamaño de los datos transferidos y el nivel de la señal con que se ha transmitido. Esta energía utilizando WLAN es sustancialmente más pequeña que 3G, especialmente para transmisiones de datos grandes. A medida que se aumenta el tamaño de los datos, la eficiencia crece.
- WLAN es más eficiente que 3G una vez que la interfaz de red se ha asociado al punto de acceso.
- La energía para mantener cada uno de los interfaces de red es de 1-2 Julios/min en el caso de 3G mientras que es de 3-3.5 Julios/min en el caso de WLAN.

Todas estas conclusiones, tanto de los sistemas criptográficos como del coste energético de WLAN y 3G, servirán para explicar mejor los resultados obtenidos en el Capítulo 4.

Capítulo 3. Metodología

El objetivo principal de este capítulo es no sólo documentar las tecnologías, herramientas y componentes utilizados, sino también detallar los diferentes algoritmos y suites criptográficas utilizadas, justificando su elección. Además, también se incluye un estudio de los diferentes escenarios diseñados para la realización de los experimentos, así como la configuración experimental construida para tal fin.

3.1 Tecnologías y herramientas utilizadas

En la siguiente sección, se da una introducción a todas las tecnologías y herramientas utilizadas. Para una referencia mas detallada de cada una de ellas consultar la Sección *4.2 Hardware and Software used* del Anexo C.

3.1.1 Symbian

Symbian[1] es la plataforma mas utilizada y popular en los dispositivos móviles Nokia. El sistema Symbian es un sistema operativo multitarea de 32 bits capaz de ejecutar aplicaciones construidas con diferentes lenguajes de programación tales como Symbian C++, Open C/C++, Java, Adobe Flash o Qt.

3.1.2 Nokia N95

El teléfono Nokia N95[2] (ver Figura 3.1) fue lanzado al mercado en 2007 por Nokia. Está basado en Symbian 9.2, la tercera generación de dicho sistema operativo, y es el primer dispositivo de Nokia con conectividad HSDPA o 3.5G, la evolución del 3G.

[1] http://www.symbian.org - Último acceso: 2 noviembre 2010
[2] http://europe.nokia.com/support/product-support/nokia-n95 - Último acceso: 2 noviembre 2010

Figura 3.1: Nokia N95

3.1.3 Carbide.C++ IDE

Carbide.C++[3] es una aplicación software para el desarrollo de aplicaciones para el sistema operativo Symbian. Proporciona las herramientas necesarias para desarrollar aplicaciones utilizando los diferentes lenguajes de programación soportados por la plataforma. Para el desarrollo de este trabajo se ha utilizado Carbide.C++ en su versión 2.0.

3.1.4 Open C/C++

Open C/C++ plug-in[4] es un conjunto de librerías que proporcionan al programador un conjunto de funciones y procedimientos para su uso en el desarrollo de aplicaciones utilizando los lenguajes de programación C/C++. Este conjunto de librerías ayuda al desarrollo de nuevas aplicaciones o permite la portabilidad de aplicaciones ya existentes de una forma rápida y fácil.

3.1.5 OpenSSL

OpenSSL [Ope10] es un conjunto de herramientas de código libre que ayudan a implementar el protocolo *Secure Socket Layer* y proporciona una fuerte librería criptográfica de propósito general, incluyendo la mayoría de los algoritmos criptográficos utilizados en la actualidad. OpenSSL proporciona una interfaz de programación de aplicaciones o API que facilita el desarrollo de aplicaciones que utilizan el protocolo SSL.

Aunque existen diferentes alternativas tales como GNUTLS[5], Network Security Services (NSS)[6] o PolarSSL[7], se eligió OpenSSL atendiendo a las siguientes razones:

[3]http://developer.symbian.org/main/create_applications/index.php - Último acceso: 2 noviembre 2010

[4]http://www.forum.nokia.com/Technology_Topics/Development_Platforms/Open_C_and_C++/ - Último acceso: 2 noviembre 2010

[5]http://www.gnu.org/software/gnutls/ - Último acceso: 2 noviembre 2010

[6]http://www.mozilla.org/projects/security/pki/nss/ - Último acceso: 2 noviembre 2010

[7]http://www.polarssl.org - Último acceso: 2 noviembre 2010

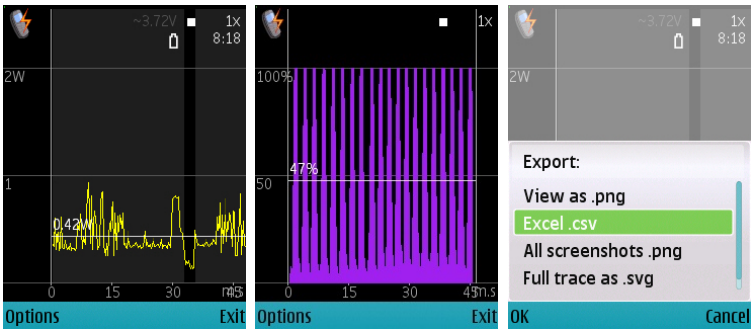


Figura 3.2: Capturas de pantalla del Nokia Energy Profiler en funcionamiento

- Está incluida en el conjunto de librerías Open C/C++.
- Es la librería mas utilizada en el desarrollo de aplicaciones seguras SSL/TLS y la documentación es muy extensa.
- Está incluida en la mayoría de sistemas UNIX/Linux.

3.1.6 Nokia Energy Profiler

Nokia Energy Profiler (NEP)[8] es una aplicación independiente que permite realizar tests y monitorizar el consumo energético de aplicaciones en tiempo real en el propio dispositivo móvil (ver Figura 3.2 para un detalle de las capturas de pantalla de la aplicación). NEP es capaz de medir el consumo energético instantáneo, el porcentaje de carga del procesador, la velocidad de la red, el ancho de banda consumido, el nivel de voltaje utilizado, los temporizadores 3G, etc. Además permite exportar todos los datos recogidos en diferentes formatos de archivo, tales como CSV, SVG o JPEG.

Como NEP es una aplicación independiente de medida, proporciona también una API externa que permite el control del proceso de recogida de datos desde cualquier aplicación externa. El uso de esta API y del proceso de conexión está explicado en la Sección 3.3.2.

3.2 Entorno

En esta sección se detalla la elección de las diferentes suites criptográficas utilizadas y los escenarios de pruebas diseñados para el exhaustivo estudio de las necesidades planteadas.

3.2.1 Suites criptográficas elegidas

El factor principal para seleccionar el conjunto de suites criptográficas a utilizar fue el uso de un buen algoritmo criptográfico simétrico. Entre todas las posibles suites que

[8] http://www.forum.nokia.com/Technology_Topics/Application_Quality/Power_Management/Nokia_Energy_Profiler_Quick_Start.xhtml - Último acceso: 2 noviembre 2010

SSL ofrece y admite, se eligieron aquellas cuyo algoritmo simétrico utilizase AES. Esta elección fue tomada dado que este es uno de los algoritmos más utilizados y está definido como estándar criptográfico por el *National Institute of Standards and Technology* (NIST). Además de esto, trabajos previos sobre algoritmos simétricos [PRRJ06] han demostrado que AES tiene un buen rendimiento con costes energéticos competitivos.

De las 12 suites criptográficas que incluyen AES y están definidas en el estándar TLS (más información en la Sección *4.3 Cipher suites selection* del Anexo C), fue necesario descartar algunas de ellas. Todas ellas utilizan como protocolo de encriptación/desencriptación así como de *message digest*, AES y HMAC-SHA1-1 respectivamente. De este modo, la razón detrás de estos descartes está relacionada con los algoritmos utilizados para autenticar los pares y el protocolo de intercambio de clave:

- El uso de Diffie-Hellman anónimo (ADH) proporciona confidencialidad pero no autenticación, con el consiguiente riesgo de que puedan producirse ataques man-in-the-middle [PP06].
- El uso de *Digital Signature Standard* (DSS) [oST06] tiene peor rendimiento si se compara con RSA [01].
- OpenSSL únicamente implementa Diffie-Hellman efímero (DHE), pero no Diffie-Hellman (DH).

Habiendo discutido los factores decisivos, las suites elegidas se presentan en la Tabla 3.1. Estas suites criptográficas presentan niveles altos de seguridad, ya que por defecto DHE-RSA-AES-256-SHA proporciona el nivel más alto de seguridad.

Cipher Suite	Auth	Key Exchange	Encryption	Digest
RSA-AES-128-SHA	RSA	RSA	AES-128-CBC	SHA1
RSA-AES-256-SHA	RSA	RSA	AES-256-CBC	SHA1
DHE-RSA-AES-128-SHA	RSA	DHE	AES-128-CBC	SHA1
DHE-RSA-AES-256-SHA	RSA	DHE	AES-256-CBC	SHA1

Tabla 3.1: Suites criptográficas elegidas

3.2.2 Diseño de los diferentes escenarios

El uso del protocolo SSL para crear una comunicación segura conlleva un incremento de trabajo computacional y de intercambio de datos que incurre en un mayor consumo energético. Para realizar un exhaustivo análisis de la energía involucrada en la creación de un canal seguro utilizando SSL es importante remarcar que el incremento energético puede dividirse en 2:

1. Mayor volumen de datos debido a los componentes adicionales necesarios, tales como certificados, resúmenes de mensajes o *MACs*.

2. Incremento del trabajo de CPU generado por algoritmos criptográficos tales como encriptación/desencriptación, autenticación o resumen de mensajes.

Como se ha comentado en la Sección 2.3, el protocolo SSL se divide en 2 fases diferentes:

1. La fase de handshake, computacionalmente costosa debido al uso de algoritmos de clave pública y de varios hilos de comunicación.
2. La fase de transferencia, utilizando la clave secreta acordada para intercambiar los datos de forma segura y eficiente.

Con el fin de medir el incremento de los datos y del consumo energético dado en el establecimiento de una sesión SSL, es necesaria la construcción de diferentes escenarios, los cuales pueden dividirse en experimentos locales, remotos y de rendimiento y coste adicional.

Experimentos locales

Aquí se describen los experimentos locales donde el servidor y el cliente están localizados en la misma red (latencia inferior a 1 ms). Para su consecución, como se ha comentado anteriormente, es necesaria la implementación de un servidor local capaz de manejar diferentes conexiones SSL. Para obtener un completo análisis del consumo energético se diseñaron los siguientes experimentos:

- El uso de un canal seguro incrementa los datos a intercambiar. Para medir este incremento, se modelaron 2 escenarios: el uso de un canal no seguro (una conexión TCP normal) y otro canal utilizando el protocolo SSL. La diferencia de datos intercambiados indica el incremento total debido al uso de SSL. Con el fin de ver en qué grado se incrementan dichos datos, se utilizaron diferentes tamaños de archivo: 1K, 10K, 50K, 100K, 500K, 1MB y 5 MB.
- El escenario descrito anteriormente puede ser utilizado también para medir el incremento en el consumo energético debido a la utilización del protocolo SSL. Una parte importante en el protocolo SSL es el *handshake*, en el cual se utilizan algoritmos de clave pública, que conllevan un fuerte impacto energético. Para medir este consumo, se llevaron a cabo diferentes *handshakes* utilizando las diferentes suites criptográficas elegidas para medir no sólo dicho consumo energético sino también el total de datos intercambiados en esta fase.
- Analizar el impacto que tiene el uso de sesiones cacheadas y no cacheadas. Como se ha comentado en la Sección 2.3, SSL ofrece un mecanismo que permite al par cliente-servidor omitir las operaciones involucradas en los algoritmos de clave pública, ya que estas han sido calculadas en una sesión previa. Esta característica reduce de forma significativa el coste total de el establecimiento de la conexión.

- Estudiar el consumo energético generado en la autenticación del cliente. La mayoría de los servidores no requiere autenticación del cliente, pero es interesante medir el incremento de datos involucrado.
- Analizar y medir el impacto energético que conlleva el utilizar diferentes interfaces de comunicación de los que dispone el terminal móvil, WLAN y 3G.

Experimentos remotos

Aquí se describen los experimentos remotos donde el cliente móvil se conecta a un proveedor de servicios, como un servidor de correo o una red social, que no se encuentra en la misma red. La latencia depende del servicio remoto, del proveedor de Internet y del estado general de la red. Estos casos están dirigido a evaluar cómo se comportan diferentes servicios en línea a la hora de establecer una sesión SSL, midiendo no sólo el consumo energético sino también la cantidad de información intercambiada y el tiempo total empleado.

Antes de comenzar explicando los diferentes escenarios diseñados, se presenta una lista de los servicios remotos estudiados:

- Google *(www.google.com)* como el mayor proveedor de servicios de Internet.
- Facebook *(www.facebook.com)* como la red social más importante de Internet. Se han estudiado 3 versiones diferentes:
  1. La versión original *(www.facebook.com)*
  2. Una versión más segura *(ssl.facebook.com)*
  3. Un versión para móviles *(m.facebook.com)*
- Ovi *(www.ovi.com)* como el proveedor de servicios de todos los productos de Nokia.
- Verisign *(www.verisign.com)* como una de las mayores empresas proveedoras de servicios a través de Internet y como empresa certificadora digital.

Todos ellos tienen diferentes características y perfiles que son analizados en la Sección 4.2.1.

Una vez introducidos los diferentes servicios analizados, se presentan a continuación los escenarios diseñados:

- Estudio de un perfil completo de los diferentes servicios con el fin de entender mejor los resultados obtenidos.
- Medir el impacto energético generado al establecer una sesión SSL con los diferentes servicios utilizando las diferentes suites criptográficas elegidas.
- Analizar el impacto que tiene el uso de sesiones guardas y no guardadas, al igual que en los experimentos locales.
- Al igual que en el caso anterior, realizar mediciones con los diferentes interfaces de comunicación, tanto WLAN como 3G.

Experimentos de rendimiento y coste adicional

Utilizando los experimentos anteriores, tanto los locales como los remotos, se puede obtener el consumo energético y los tiempos de ejecución generales, pero es también importante conocer el rendimiento real y el coste adicional que conlleva el uso de este tipo de comunicaciones seguras utilizando el protocolo SSL. Los escenarios a estudiar son:

- La cantidad de bytes adicionales utilizados para asegurar el canal.
- El rendimiento de SSL, mostrando cómo el uso de dicho protocolo afecta al rendimiento general.
- El coste energético adicional de la utilización de SSL.
- Estudio en detalle del coste energético de cada uno de los componentes del protocolo SSL: algoritmos simétricos, asimétricos, hashing y la interfaz de red.

Con el fin de detallar el análisis del coste adicional del uso de SSL, se han obtenido diferentes resultados tanto de los casos locales como de los remotos, además de otras métricas como el coste energético de los diferentes algoritmos criptográficos utilizados (estas métricas están incluidas en el Anexo B *Study of Symmetric and Hash algorithms* del Anexo C) para obtener el estudio en detalle de cada uno de los componentes.

3.3 Configuración experimental

Una vez se han presentado las tecnologías utilizadas, conjuntamente con los diferentes escenarios a estudiar y los requerimientos, es hora de describir la configuración experimental y la metodología de medición. La Figura 3.3 describe de manera gráfica cómo están conectados cliente y servidor y cómo se realizan las mediciones.

La configuración experimental consiste en un cliente que se conectad a una red local LAN a través de un punto de acceso inalámbrico creado por un enrutador o *router*, mientras que el servidor se encuentra alojado en un ordenador cableado a dicha LAN. El cliente se ejecuta en un Nokia N95, que contiene un procesador dual ARM a 322 Mhz, incluyendo una batería Li-Ion con 950 mAh de capacidad. El servidor por su parte, es un ordenador portátil equipado con un procesador Intel Mobile Core 2 Duo T7250 a 2.0Ghz con 2048 MB de memoria RAM DDR2 y ejecutando una distribución virtualizada de Ubuntu 9.10 mediante la aplicación de virtualización VMWare Workstation[9].

Para la medición del consumo energético y otros parámetros adicionales, *Nokia Energy Profiler* se ejecuta en segundo plano mientras el cliente ejecuta todas las operaciones necesarias. NEP está conectado internamente con el cliente con el fin de medir únicamente la fracción de tiempo deseada.

[9] http://www.vmware.com/products/workstation/ - Último acceso: 2 noviembre 2010

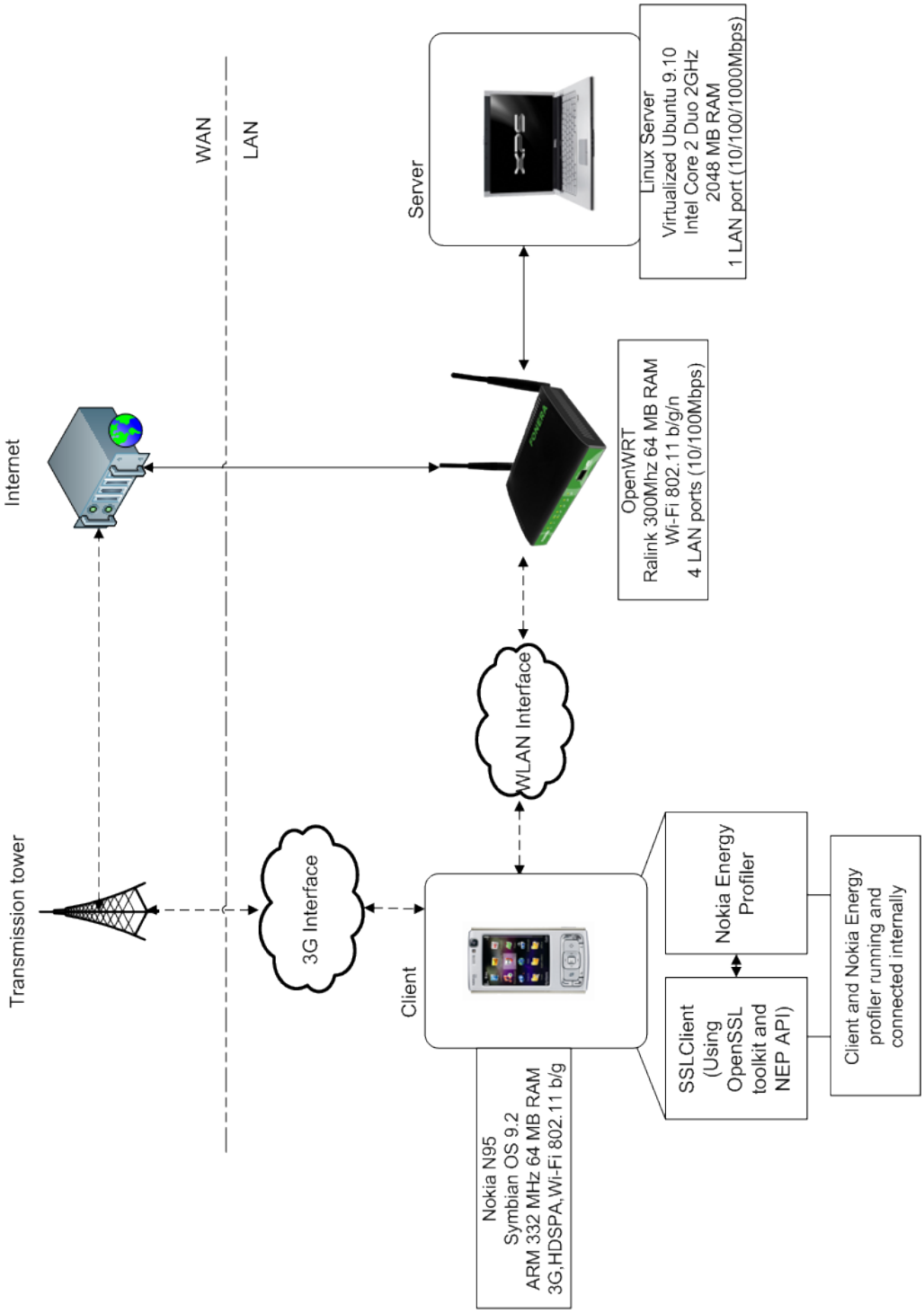


Figura 3.3: Configuración experimental entre el cliente y el servidor

3.3.1 Diseño de las aplicaciones

En esta sección se comentan los aspectos más importante de la implementación tanto del servidor como del cliente, así como los diferentes modos y opciones ofrecidos por los mismos. Se pueden obtener más detalles en la Sección *4.5.1 Design of the applications* del Anexo C.

Diseño del cliente

El cliente SSL está desarrollado en lenguaje C con algunas funciones adicionales como la conexión interna con NEP creadas con C++. Para la codificación en estos lenguajes, es necesario el conjunto de librerías incorporadas en Open C/C++, que a su vez incluyen la implementación de OpenSSL necesaria para la implementación de los distintos métodos criptográficos y del protocolo SSL.

Como el objetivo de este trabajo es medir el consumo energético de el uso del protocolo SSL y de otros métodos criptográficos, cualquier otro servicio o aplicación que esté en funcionamiento generará un coste energético extra. Con el fin de ahorrar energía, memoria y ,a su vez, hacer la implementación más sencilla, no existe interfaz gráfica, únicamente una consola de comandos para seleccionar las diferentes opciones para crear los diferentes escenarios a medir. En este tipo de configuraciones experimentales, la parte más importante corresponde con la implementación del motor criptográfico y las mejores relacionadas con él, pero no en la interfaz gráfica. Estas mejoras también incluyen la selección del interfaz de red a utilizar, que también se establece únicamente utilizando la consola de comandos.

El proceso de *handshake* en SSL conlleva operaciones con tiempos de ejecución de un orden de magnitud de milisegundos. Para asegurar resultados precisos en los experimentos, es necesario la realización de varias repeticiones de los diferentes procesos y experimentos. Esto también es importante ya que el *Nokia Energy Profiler* realiza muestras en las mediciones en intervalos de 0.25 ms. De este modo, en algunos casos (como pueden ser en el uso de la suite criptográfica AES[128,256]-SHA1), el número de repeticiones es de hasta 300. Este número de repeticiones se selecciona dependiendo del tamaño de los datos incluido en las pruebas.

Diseño del servidor

El servidor SSL está implementado en lenguaje C y se ejecuta en un PC que virtualiza una distribución Linux Ubuntu[10]. El factor principal de la utilización de la virtualización es debido a que el IDE utilizado para el desarrollo de la aplicación cliente, Carbide.C++ sólo funciona en sistemas operativos Windows, por lo que para un correcta metodología de trabajo, en la que se incluyen experimentos y modificaciones en el código iteratívamente, era totalmente necesario. Además, el uso de Ubuntu es debido a que es un entorno Linux que proporciona aplicaciones como SSLDump[11], que permiten la decodificación de todas las comunicaciones SSL conociendo la clave secreta. Este programa fue utilizado en

[10] http://www.ubuntu.com/ - Último acceso: 2 noviembre 2010
[11] http://www.rtfm.com/ssldump/ - Último acceso: 2 noviembre 2010

una fase temprana de la investigación, pero como posteriormente se le incluyó al cliente también la mayoría de las opciones de SSLDump, ya no fue necesario.

Se empleó una versión modificada de OpenSSL para la implementación del protocolo SSL en el desarrollo del servidor. SSLv3 y TLS ofrecen métodos de compresión opcionales para la transmisión de datos, pero no existe ninguno por defecto. OpenSSL utiliza por defecto compresión ZLIB[12] si es soportada por ambos pares. Con el fin de realizar diferentes experimentos donde es necesario comparar la energía disipada utilizando un canal seguro SSL o un canal sin seguridad, fue necesario deshabilitar la compresión para intercambiar la misma cantidad de datos en los diferentes escenarios.

3.3.2 Recolección de datos y muestras

Como se menciona en la Sección 3.1.6, se utiliza NEP para monitorizar y registrar el consumo energético del dispositivo móvil, el desgaste de la batería, las comunicaciones, la carga de la CPU y el voltaje a lo largo del tiempo. Con el fin de ampliar las funcionalidades, NEP ofrece un componente externo y una API[13] para controlar las funcionalidades de la aplicación desde cualquier aplicación externa, en este caso el cliente móvil SSL, permitiendo iniciar y parar las mediciones en cualquier momento. Dicho componente habilita una clase *CJuiceExternalAPI*, que es la encargada de establecer la comunicación entre el cliente móvil y NEP. De este modo, se permiten realizar mediciones precisas en cualquier intervalo de tiempo deseado. La Figura 3.4 muestra cómo se inician tanto el cliente como el servidor y se preparan para establecer la comunicación SSL, junto con el protocolo de medición de los datos.

Como se ha comentado en la Sección 2.5.2, diferentes estudios han demostrado que el mayor impacto energético en dispositivos móviles es debido al uso de interfaces de comunicación como pueden ser 3G, WLAN o *Bluetooth*, seguido por el consumo de pantallas y de la iluminación de las mismas. El proceso de medida del consumo se realiza de tal manera que únicamente el mínimo de servicios y aplicaciones necesarias para operar el móvil esté en funcionamiento, así como con la desconexión de la pantalla. De este modo, el cliente necesita esperar hasta que la pantalla pase a un estado donde dicha iluminación esté apagada. Esto supone un importante reducción del consumo energético y una mejora en la precisión de los experimentos.

3.4 Diagrama temporal y de esfuerzo

En esta sección se pretende detallar, mediante la ayuda de un diagrama de Gantt (ver Figura 3.5), las diferentes etapas por las que el proyecto ha ido pasando hasta su finalización, intentando reflejar de una manera aproximada el tiempo empleado en cada una de las diferentes tareas.

[12] http://www.zlib.net/ - Último acceso: 2 noviembre 2010

[13] http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Plug-ins/Extensions/Nokia_Energy_Profiler_External_APIs/ - Último acceso: 2 noviembre 2010

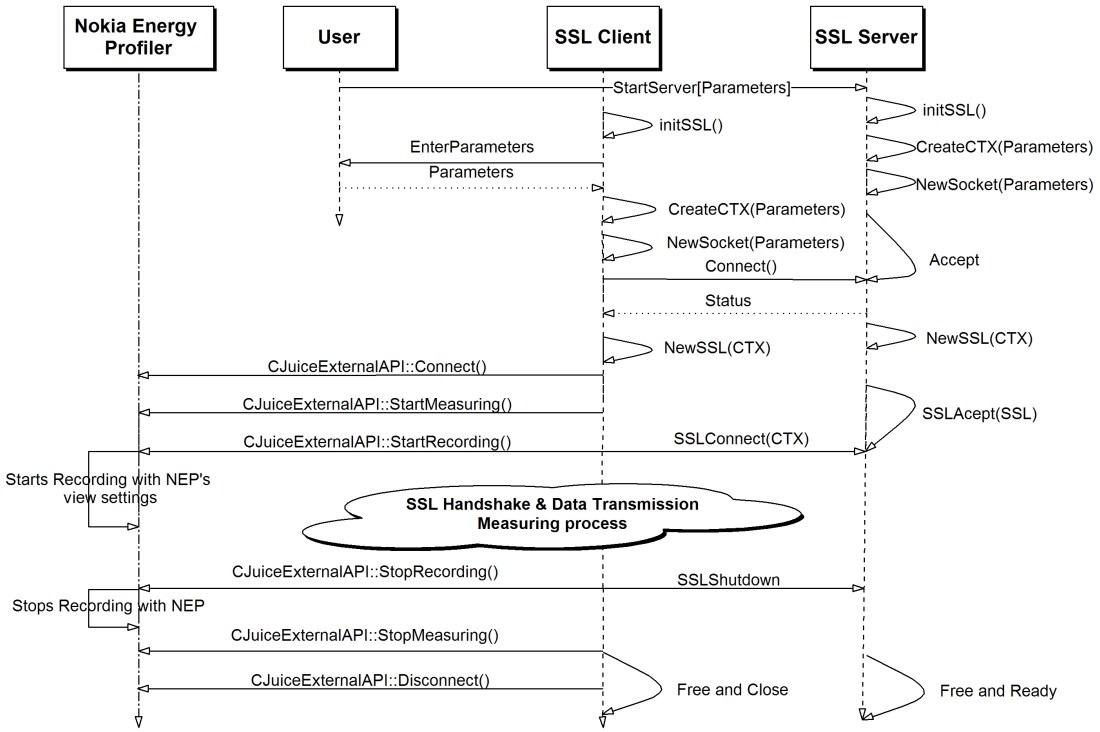


Figura 3.4: Visión general de la comunicación entre el cliente, el servidor y Nokia Energy Profiler

El proyecto se inició en Octubre de 2009, dándose por finalizado en Mayo de 2010. Tras la conclusión, se decidió continuar el trabajo con el fin de publicar un articulo de investigación con los diferentes resultados obtenidos. Esta fase no está reflejada en el diagrama, ya que se compaginó la escritura del articulo con un trabajo en una empresa externa a tiempo completo, por lo que el tiempo dedicado a esta tarea fue disperso. Una vez finalizado el trabajo, a finales de Septiembre de 2010 se comienza con la adaptación de la memoria final y su traducción, dando por finalizado el proyecto a finales de Octubre de 2010.

El proyecto tuvo una duración aproximada de 440 horas, en las que no está incluido el tiempo invertido en el desarrollo del artículo de investigación. Estas tareas se pueden dividir en:

- Estudio previo: Fase de estudio del problema y de recopilación de información.
  - Estudio de criptografía y seguridad.
  - Estudio del protocolo SSL.
- Desarrollo: Fase de diseño, codificación y prueba de las aplicaciones de test.
- Experimentación: Fase de experimentos y mediciones.
- Documentación: Fase de escritura de la documentación y memoria final.
  - Redacción de la memoria en Inglés.

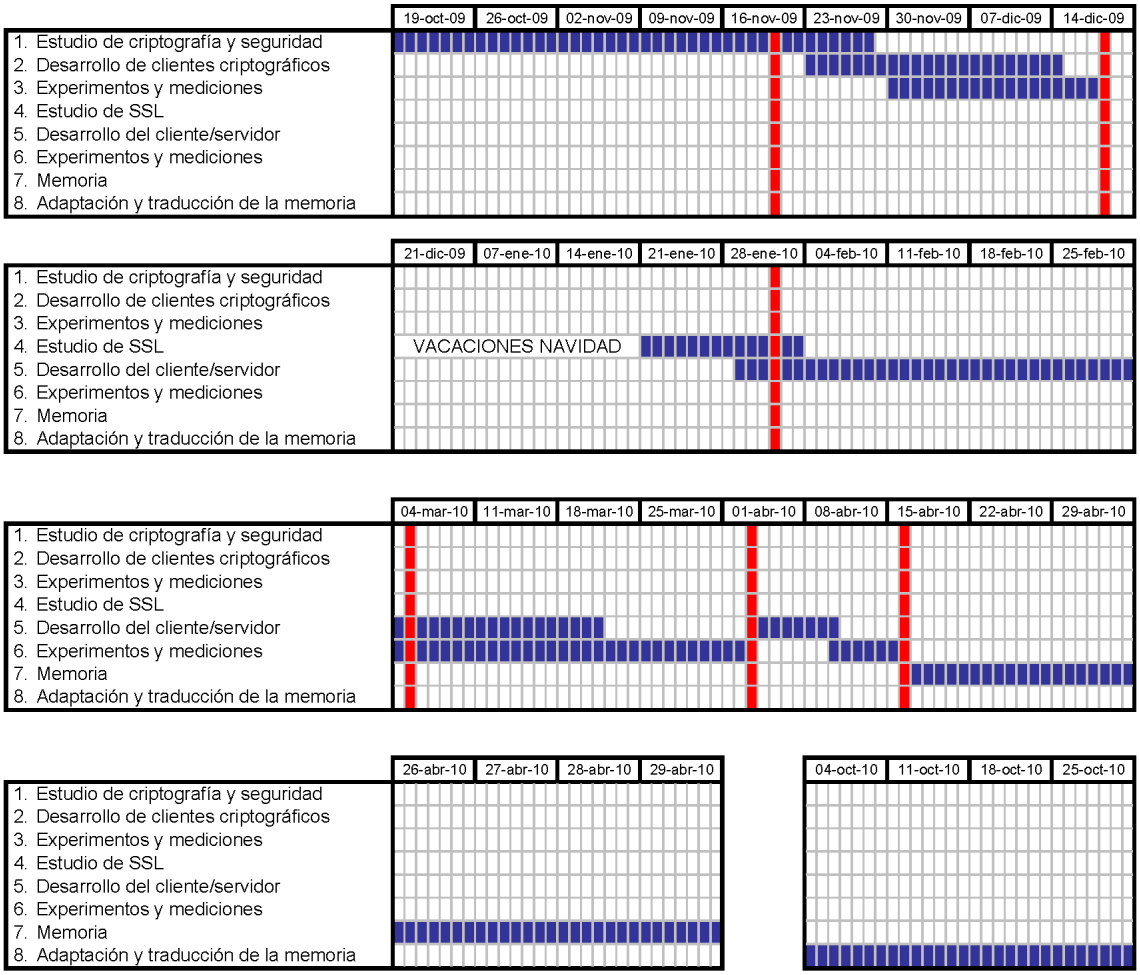


Figura 3.5: Diagrama de Gantt del proyecto

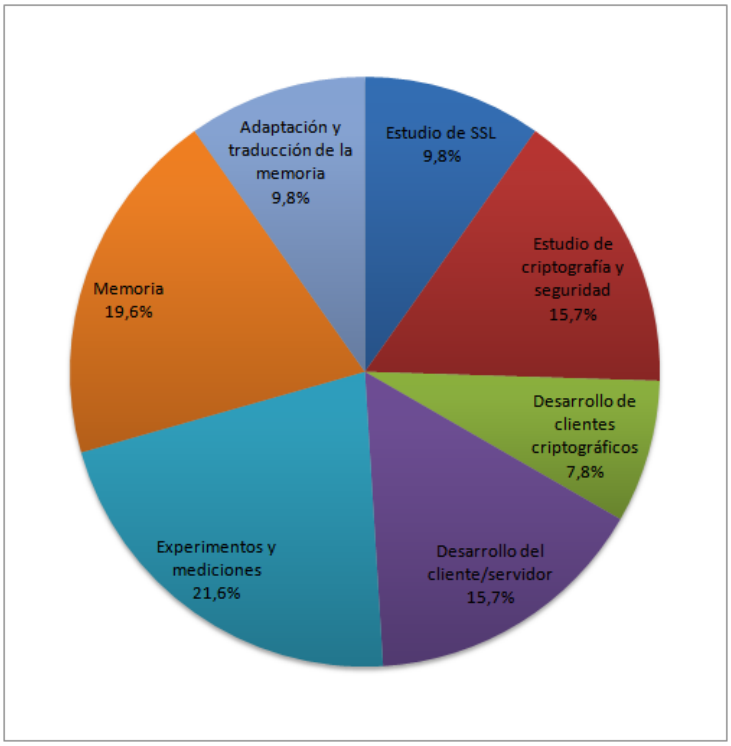


Figura 3.6: Diagrama de tiempos invertidos

– Adaptación y traducción de la memoria final para su presentación en la universidad de Zaragoza.

En la Tabla 3.2 se desglosa el número aproximado de horas dedicadas en cada una de las tareas individualmente, mostrando el porcentaje del total de horas, los cuales se muestran en la Figura 3.6.

	Concepto	*Horas*	*Porcentajes*
Estudio previo	Estudio de criptografía y seguridad	80	15,7
	Estudio de SSL	50	9,8
Desarrollo	Desarrollo de aplicaciones de test	120	23,5
Experimentación	Experimentos y mediciones	110	21,6
Documentación	Memoria	100	19,6
	Adaptación y traducción de la memoria	50	9,8
	Total	**510**	

Tabla 3.2: Desglose de las horas totales empleadas en el proyecto

Capítulo 4. Resultados Experimentales

En esta sección, se detalla un completo análisis del consumo energético del protocolo SSL y de los algoritmos asociados a él, utilizando la configuración experimental descrita en la Sección 3.3.

4.1 Escenario: Caso local

La realización de experimentos en un ámbito local (latencias inferiores a 1 ms) reduce el tiempo empleado en establecer una conexión SSL debido a los diferentes mensajes intercambiados y los *ACKs* que ambos pares intercambian. Así pues, manteniendo las interfaces de red transmitiendo únicamente el tiempo necesario, se pueden obtener resultados mas precisos del coste energético total del establecimiento de dicha conexión.

4.1.1 Datos intercambiados

Para entender bien cuánta información se intercambia en la fase de *handshake*, el cliente incluye una opción para medir el número de bytes intercambiados en la operación. Como en la fase de handshake únicamente se utilizan algoritmos criptográficos de clave pública (RSA) y mecanismos de intercambio de clave (RSA o Diffie-Hellman), los métodos de encriptación (AES) y los relacionados con integridad (HMAC-SHA1) no afectan y los resultados muestran este hecho: ya sea AES con longitud de clave 128 ó 256 bits, la única diferencia es debida el uso de RSA o DH.

La Tabla 4.1 muestra los resultados en bytes divididos en 3 modos de operación: autenticación del cliente, sin autenticación del cliente y reanudación de la sesión. Para cada modo se analizaron las 4 suites criptográficas comentadas en la Sección 3.2.1. Las columnas marcadas con $C \rightarrow S$ y con $C \leftarrow S$ indican el flujo de la información, en bytes, de cliente a servidor y viceversa, respectivamente.

Analizando la tabla anterior podemos concluir que el cliente siempre envía la misma cantidad de información, independientemente de la suite criptográfica utilizada. Esta varía si se requiere autenticación del mismo y la variación dependerá de la longitud del certificado que proporcione el cliente. Además, el uso de la reanudación de la sesión reduce significativamente la cantidad total de datos en un 50% en el lado del cliente y de más de un 90% en el lado del servidor.

Operation Mode	Cipher suite	$C \rightarrow S$	$S \rightarrow C$
No Client Authentication	AES128-SHA	249	2368
	AES256-SHA	249	2368
	DHE-RSA-AES128-SHA	249	2770
	DHE-RSA-AES256-SHA	249	2770
Client Authentication	AES128-SHA	1772	2348
	AES256-SHA	1772	2348
	DHE-RSA-AES128-SHA	1772	2752
	DHE-RSA-AES256-SHA	1772	2752
Session Resumption	NOT RELEVANT	142	168

Tabla 4.1: Información intercambiada en bytes durante la fase de handshake en el caso local

4.1.2 Consumo energético

La Figura 4.1 muestra el consumo energético en milijulios (mJ) para las diferentes suites criptográficas en los diferentes modos antes señalados. Analizando dicha figura, la conclusión más importante que se obtiene es que el uso de Diffie-Hellman afecta críticamente al consumo energético por conexión comparado con RSA, siendo estos valores de 739 mJ y 99 mJ, respectivamente. El uso de autenticación por parte del cliente incrementa ligeramente el consumo, 842 mJ y 155 mJ, ya que es necesario más intercambio de datos. Además y como se ha comentado antes, el uso de la reanudación de sesión reduce enormemente el consumo energético a únicamente 64 mJ, independientemente del modo de operación elegido, ligeramente superior a los 25 mJ del coste total de una conexión TCP sin ningún tipo de seguridad adicional.

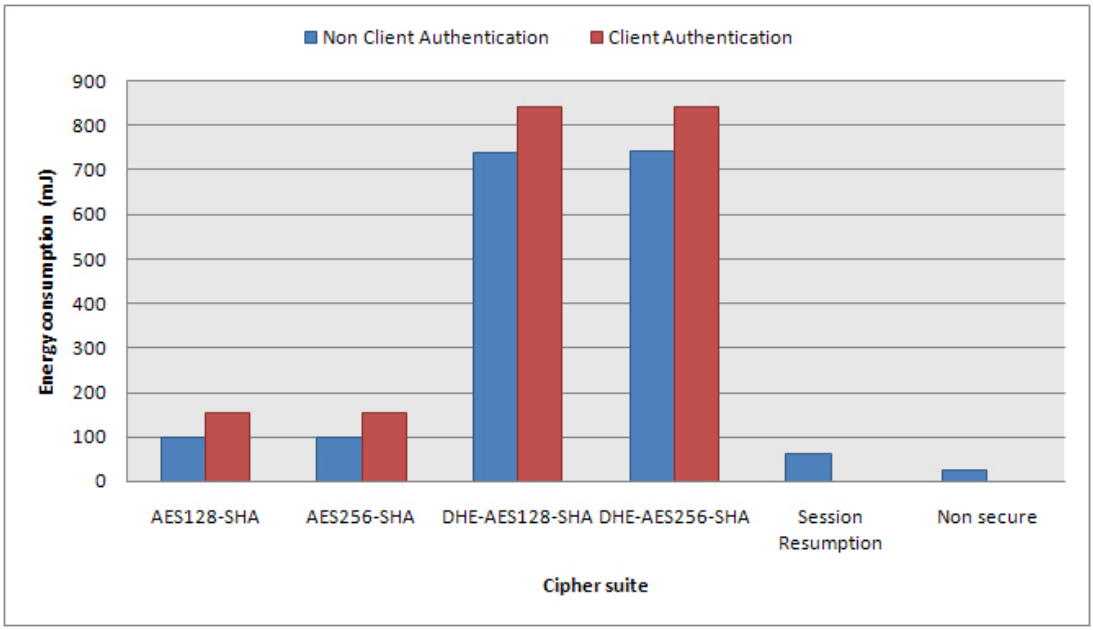


Figura 4.1: Consumo energético en la fase de handshake para el caso local

4.2 Escenario: Casos remotos

Mientras que en la sección anterior se describen experimentos realizados en un ámbito local con latencias menores a 1 ms, en los casos remotos, se estudian diferentes servicios remotos que dependen en gran medida de su localización geográfica, el estado actual de la red y de el interfaz de red utilizado.

4.2.1 Estudio del perfil de los servicios remotos

Antes de empezar a explicar los experimentos y los resultados de los mismos, es necesario realizar un perfil completo de los servicios remotos estudiados. Este estudio intenta analizar las propiedades más importantes de cada servicio incluyendo los protocolos SSL disponibles así como las diferentes implementaciones, la suite criptográfica por defecto y la lista de las mismas disponibles, la longitud del certificado digital del servidor junto con la longitud de la clave pública y si incluye soporte o no para reanudar una sesión previamente establecida, así como si permite autenticación mediante el uso de Diffie-Hellman. Se puede encontrar más información en la Sección *5.2.1 Services profile study* del Anexo C.

La Tabla 4.2 recoge toda la información obtenida acerca de los servicios estudiados. Este estudio es necesario y ayudará a analizar y comprender mejor los diferentes resultados obtenidos en los siguientes experimentos.

4.2.2 Datos intercambiados

Como ocurría con los experimentos realizados para el caso local, es importante medir la cantidad de información intercambiada para entender bien el consumo energético involucrado en la fase de handshake.

La Tabla 4.3 muestra que el cliente siempre envía la misma cantidad de información, que únicamente se ve incrementada cuando el servidor utiliza una clave pública de longitud 2048 bits, como es el caso de ssl.Facebook y Verisign. Esta longitud de clave pública también coincide con una mayor longitud del certificado de los servidores, siendo Facebook, ssl.Facebook and Verisign los mayores. De nuevo, la posibilidad de reanudar una sesión SSL previa reduce la cantidad de información intercambiada desde los más de 5 KB a únicamente 179 bytes.

4.2.3 Consumo energético

Una vez que se ha desarrollado un perfil completo de los diferentes servicios y de la cantidad de información necesaria en cada uno de los *handshakes*, la Figura 4.2 muestra el consumo energético utilizando la interfaz de red WLAN. Como se ha comentado antes para el caso local, el uso de Diffie-Hellman afecta de manera drástica al coste energético total del establecimiento de la conexión, no sólo incrementando el tiempo de ejecución sino también los datos totales enviados por los pares durante todo la fase de handshake. Otro

Service	Google	Facebook	SSL Facebook	mFacebook	Verisign	Ovi
Protocols available	SSLv3, TLSv1.0	SSLv3, TLS1.0	SSLv3, TLS1.0	SSLv3, TLS1.0	SSLv3, TLSv1.0	SSLv3, TLS1.0
Server Implementation	Apache mod_SSL	Apache mod_NSS	Apache mod_NSS	Apache mod_NSS	Apache mod_SSL	IIS 7.5
Default Cipher Suite	AES256-SHA	RC4-MD5	RC4-MD5	RC4-MD5	DHE-RSA-AES256-SHA	DHE-RSA-AES256-SHA
Server Certificate length	1025 bytes	4593 bytes	4642 bytes	836 bytes	4519 bytes	1738 bytes
Public server-key	1024 bits	1024 bits	2048 bits	1024 bits	2048 bits	1024 bits
Session resumption	YES	NO	NO	NO	YES	YES
Ephemeral Diffie Hellman	NO	NO	YES	YES	YES	YES
Cipher suites available	AES256-SHA **256 bit**	AES256-SHA **256 bit**	DHE-RSA-AES256-SHA **256 bit**	DHE-RSA-AES256-SHA **256 bit**	DHE-RSA-AES256-SHA **256 bit**	ADH-AES256-SHA **256 bit**
	DES-CBC3-SHA **168 bit**	DES-CBC3-SHA **168 bit**	AES256-SHA **256 bit**	AES256-SHA **256 bit**	AES256-SHA **256 bit**	DHE-RSA-AES256-SHA **256 bit**
	AES128-SHA **128 bit**	AES128-SHA **128 bit**	EDH-RSA-DES-CBC3-SHA **168 bit**	EDH-RSA-DES-CBC3-SHA **168 bit**	EDH-RSA-DES-CBC3-SHA **168 bit**	AES256-SHA **256 bit**
	RC4-SHA **128 bit**	RC4-SHA **128 bit**	DES-CBC3-SHA **168 bit**	DES-CBC3-SHA **168 bit**	DES-CBC3-SHA **168 bit**	ADH-DES-CBC3-SHA **168 bit**
	RC4-MD5 **128 bit**	RC4-MD5 **128 bit**	DHE-RSA-AES128-SHA **128 bit**	DHE-RSA-AES128-SHA **128 bit**	DHE-RSA-AES128-SHA **128 bit**	EDH-RSA-DES-CBC3-SHA **168 bit**
			AES128-SHA **128 bit**	AES128-SHA **128 bit**	AES128-SHA **128 bit**	DES-CBC3-SHA **168 bit**
			RC4-SHA **128 bit**	RC4-SHA **128 bit**	RC4-SHA **128 bit**	ADH-AES128-SHA **128 bit**
			RC4-MD5 **128 bit**	RC4-MD5 **128 bit**	RC4-MD5 **128 bit**	DHE-RSA-AES128-SHA **128 bit**
			EDH-RSA-DES-CBC-SHA **56 bit**	EDH-RSA-DES-CBC-SHA **56 bit**	EDH-RSA-DES-CBC-SHA **56 bit**	AES128-SHA **128 bit**
			DES-CBC-SHA **56 bit**	DES-CBC-SHA **56 bit**	DES-CBC-SHA **56 bit**	ADH-RC4-MD5 **128 bit**
			EXP-EDH-RSA-DES-CBC-SHA **40 bit**	EXP-EDH-RSA-DES-CBC-SHA **40 bit**		RC4-SHA **128 bit**
			EXP-DES-CBC-SHA **40 bit**	EXP-DES-CBC-SHA **40 bit**		RC4-MD5 **128 bit**
			EXP-RC2-CBC-MD5 **40 bit**	EXP-RC2-CBC-MD5 **40 bit**		ADH-DES-CBC-SHA **56 bit**
			EXP-RC4-MD5 **40 bit**	EXP-RC4-MD5 **40 bit**		EDH-RSA-DES-CBC-SHA **56 bit**
						DES-CBC-SHA **56 bit**
						EXP-ADH-DES-CBC-SHA **40 bit**
						EXP-ADH-RC4-MD5 **40 bit**
						EXP-EDH-RSA-DES-CBC-SHA **40 bit**
						EXP-DES-CBC-SHA **40 bit**
						EXP-RC2-CBC-MD5 **40 bit**
						EXP-RC4-MD5 **40 bit**

Tabla 4.2: Perfil completo de los diferentes servicios remotos analizados

Operation Mode	Cipher suite	C → S	S → C
Google	AES128-SHA	286	1777
	AES256-SHA	286	1777
	DHE-RSA-AES128-SHA	Not supported	
	DHE-RSA-AES256-SHA		
	SESSION RESUMPTION	138	179
Facebook	AES128-SHA	286	4705
	AES256-SHA	286	4705
	DHE-RSA-AES128-SHA	Not supported	
	DHE-RSA-AES256-SHA		
	SESSION RESUMPTION		
m.Facebook	AES128-SHA	286	1004
	AES256-SHA	286	1004
	DHE-RSA-AES128-SHA	286	1374
	DHE-RSA-AES256-SHA	286	1374
	SESSION RESUMPTION	Not supported	
ssl.Facebook	AES128-SHA	414	4794
	AES256-SHA	414	4794
	DHE-RSA-AES128-SHA	286	5292
	DHE-RSA-AES256-SHA	286	5292
	SESSION RESUMPTION	Not supported	
Verisign	AES128-SHA	414	4671
	AES256-SHA	414	4671
	DHE-RSA-AES128-SHA	286	5201
	DHE-RSA-AES256-SHA	286	5201
	SESSION RESUMPTION	138	179
Ovi	AES128-SHA	286	3682
	AES256-SHA	286	3682
	DHE-RSA-AES128-SHA	286	4084
	DHE-RSA-AES256-SHA	286	4084
	SESSION RESUMPTION	138	179

Tabla 4.3: Información intercambiada en la fase de handshake para los casos remotos

factor importante que lleva a un mayor consumo es el de la longitud del certificado digital que envía el servidor, que para casos como Verisign, Facebook y ssl.Facebook supera los 4.7 KB, y en este sentido suponen los mayores gastos energéticos. El uso de claves públicas de 2048 bits no sólo incrementa la longitud del certificado sino que también supone un proceso más largo, causando un incremento energético en el calculo de la clave maestra. Todo este proceso de calculo de clave pública puede ser evitado con la reutilización de una sesión previamente establecida, obteniendo un importante ahorro energético, como puede ser en el caso de Verisign (de 1484 a 385 mJ). Este proceso de reanudar la sesión no está disponible en todos los servicios, ya que Facebook en cualquiera de sus 3 versiones no soporta dicha característica. Esto es debido a que en algunas ocasiones mantener un registro con todas las sesiones seguras establecidas con los diferentes clientes supone un gasto mayor que el que supone volver a establecer una sesión segura. Esto se ve reflejado directamente en el análisis de la implementación del servidor y de la infraestructura ya que ,por ejemplo, el establecimiento de una conexión con la versión móvil de Facebook requiere casi el doble de energía que una conexión con Google, 482 mJ y 273 mJ respectivamente, utilizando la misma suite criptográfica y teniendo este primero un certificado un 50% menor que el de el segundo.

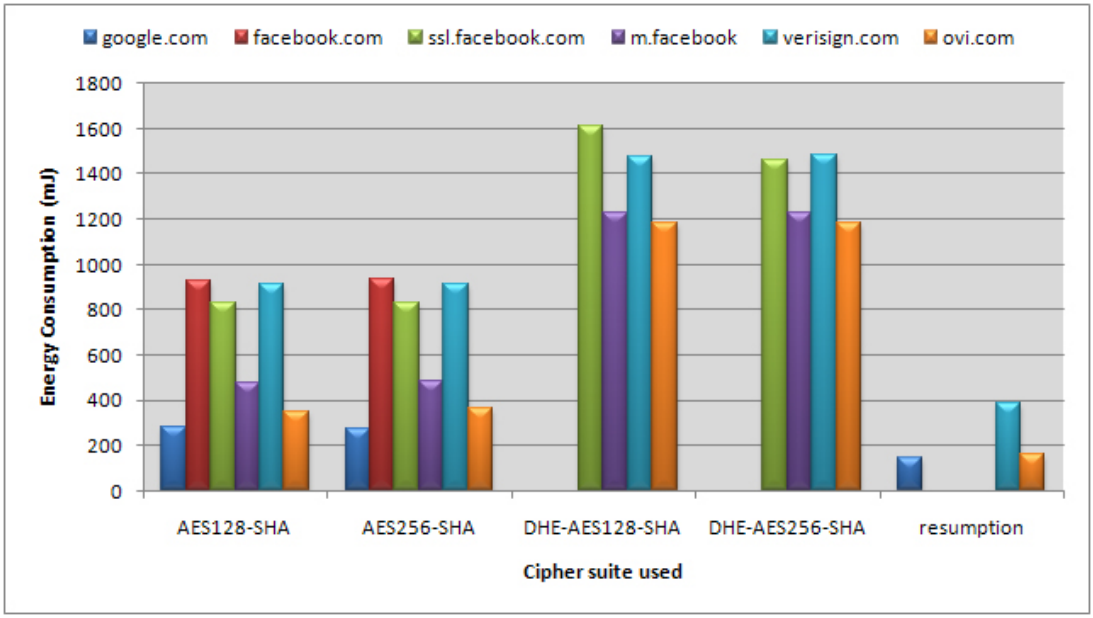


Figura 4.2: Consumo energético en diferentes servicios remotos utilizando WLAN

Los mismo experimentos que en el caso de WLAN fueron realizados utilizando la interfaz de red 3G. La Figura 4.3 muestra los resultados obtenidos, de la que se pueden obtener las mismas conclusiones que las descritas en el caso de WLAN, pero con una gran diferencia: el importante incremento en el coste energético debido al uso de 3G. Este mayor coste de 3G comparado con WLAN se menciona en la Sección 2.5.2. Este incremento puede ser de hasta un 450% en el caso de Ovi y el uso de RSA, 346 mJ con WLAN frente a los 1590 mJ con 3G.

En la Sección 2.5.2 se hacía una referencia al consumo energético de las interfaces de red, comentando que la energía de cola era la causante de un 60% del total de la energía uti-

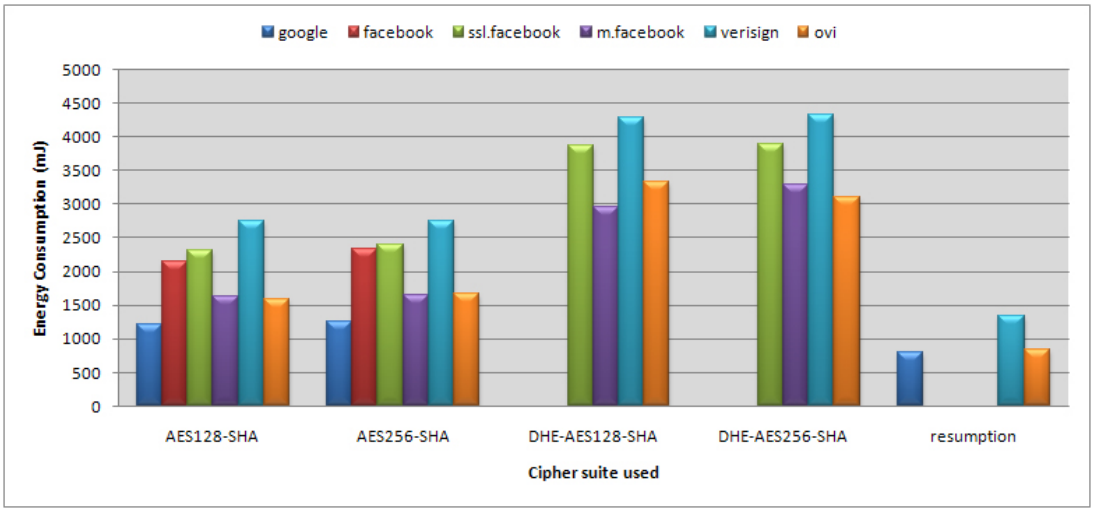


Figura 4.3: Consumo energético en diferentes servicios remotos utilizando 3G

lizada para transmitir 50KB [BBV09]. Es importante también mencionar que esta energía no está incluida en los resultados, las mediciones únicamente comprenden la transmisión de los datos. Se puede encontrar toda la información detallada acerca del consumo energético de los diferentes servicios en la Sección *5.2.3 Energy consumption using WLAN* y *5.2.4 Energy consumption using 3G* del Anexo C.

4.2.4 Tiempos de ejecución paso a paso

Con el fin de analizar en qué operaciones involucradas en la fase de *handshake* se consume la mayor parte del tiempo, se realizaron diferentes experimentos obteniendo varias conclusiones. Se utilizaron los protocolos RSA y DHE para analizar las importantes diferencias observadas en los experimentos previos realizados para los casos locales, donde DHE incrementa en gran medida el consumo energético. Estos experimentos además confirman los pasos teóricos donde se producen costes computacionales altos (explicados en el Anexo A) y ayudan a averiguar si el servidor remoto se comporta de la misma forma en todas las situaciones (latencias, número de saltos y otros problemas relacionados con el ancho de banda). Para ello, cada uno de los experimentos se llevó a la práctica 5 veces, con el fin de obtener unos resultados más precisos y evitar así las posibles variaciones de latencia u otros factores externos. Así pues, los resultados aquí mostrados son las medias de esos tiempos obtenidos.

La Figura 4.4 muestra los resultados obtenidos utilizando RSA con la interfaz de red WLAN, en los que se pueden destacar varias conclusiones. Se observa que las diferencias más importantes en terminos de tiempo de ejecución ocurren durante los pasos `ServerHello`, `Certificate`, y `ChangeCipherSpec`. Como ya se ha explicado en el Anexo A, el comando `ServerHello`, enviado por el servidor en respuesta al comando `ClientHello`, contiene varios parámetros, tales como la versión de SSL y la suite criptográfica a utilizar, así como datos aleatorios para generar la clave secreta. En el paso `Certificate`, el servidor envía su certificado incluyendo la clave pública. En ambos co-

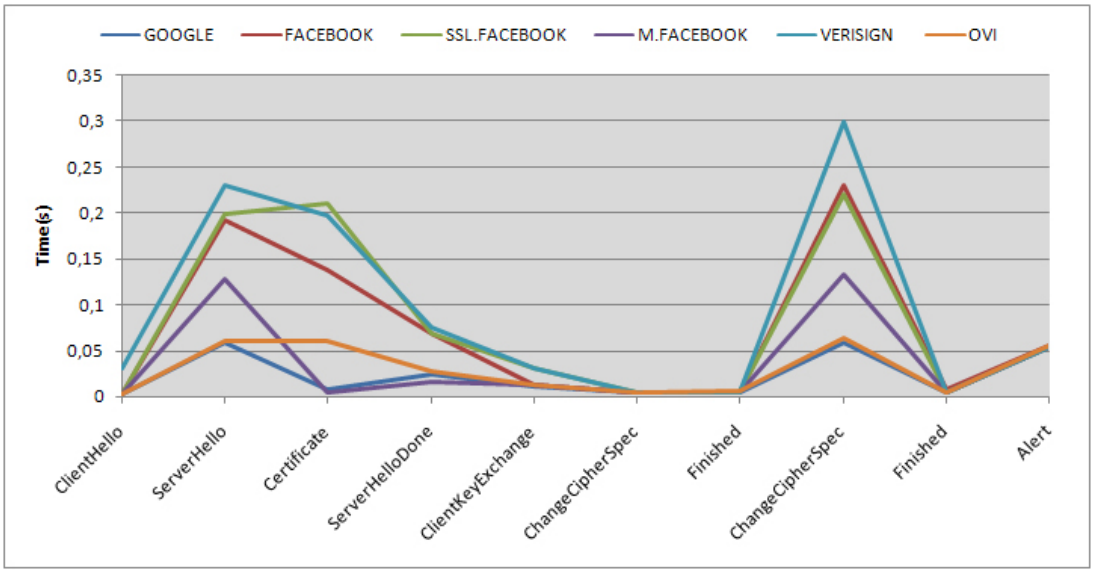


Figura 4.4: Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando RSA con WLAN

mandos, no existen operaciones criptográficas en el cliente. Una vez el certificado ha sido recibido, el cliente autentica al servidor comprobando la firma incluida en el certificado, siendo esta una operación no muy costosa usando RSA [PRRJ06]. Por tanto, las diferencias en los retardos son debidas a la transmisión utilizando TCP sobre diferentes *round-trip time* (RTT). *RTTs* a Google y a Ovi son especialmente cortos (inferiores a 50 ms) mientras que los servidores de Verisign y Facebook parecen estar situados en los Estados Unidos. Existe un RTT incluido en el paso 2 y, además, otro adicional en el paso 3 para estos 2 últimos servicios debido a la longitud de sus certificados (ver Tabla 4.2). De hecho, TCP aplica una política de inicio lenta para incrementar el tamaño de la ventana de congestión, y con una ventana inicial de 2 paquetes un certificado de 4KB no cabe en dichos paquetes conjuntamente con el comando `ServerHello`. De este modo, el servidor necesita esperar otro RTT a que lleguen los ACKs enviados por el cliente antes de transmitir el resto del certificado. De igual modo, se incluye otro RTT en el paso `ChangeCipherSpec` aunque en este paso la longitud del mensaje enviado por el servidor es de un par de bytes. Además, la duración de este paso es ligeramente superior para algunos servicios si se compara con el paso 2, ya que el servidor tiene que desenctiptar el mensaje encriptado por el cliente utilizando su clave privada. Dicha operación es muy costosa con RSA (comparado con las operaciones de clave pública realizadas en el cliente, y puede añadir un pequeño retraso, especialmente si el servidor esta sobrecargado.

El uso de Diffie-Hellman efímero como un protocolo de acuerdo de clave mejora la seguridad en la fase de handshake del protocolo SSL, pero conlleva un coste adicional en términos de tiempo y consumo energético. Ya se ha visto en la Sección 4.1.2 que DHE tiene un rendimiento mucho menor que RSA. En este caso, `ClientKeyExchange` es el paso computacionalmente más costoso y en el cual el cliente realiza las operaciones de clave pública necesarias (véanse las Figuras 4.6 y 4.7). Además, el uso de DHE añade un paso adicional a la fase de handshake. Estos resultados confirman los estudios realizados

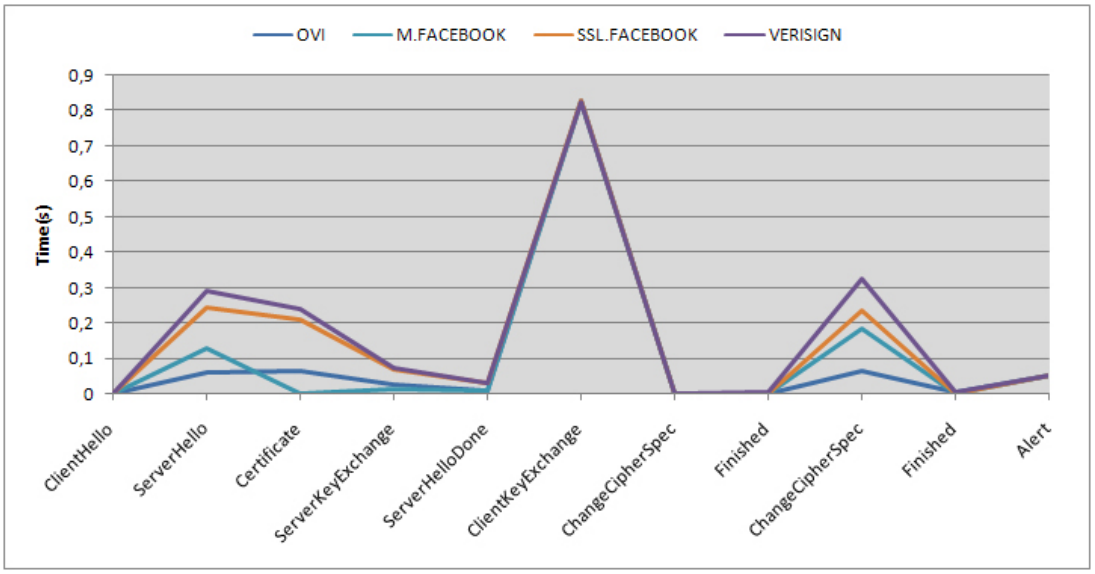


Figura 4.5: Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando DHE con WLAN

en [SGM09].

Los retardos descritos anteriormente tiene un mayor impacto en el consumo energético, ya que las interfaces de red consumen energía incluso si no se reciben/envían datos. El impacto es claramente superior en 3G (véase la Figura 4.6) debido a que el interfaz de red permanece en el estado de máximo consumo (DCH) constantemente: el temporizador de inactividad no expira durante el envío y recepción de mensajes. En cambio, el interfaz WLAN es capa de cambiar a estado *idle* o incluso *sleep*, los cuales tienen un consumo significativamente inferior [XSK+]. De hecho, para dar alguno números de referencia, un simple mensaje intercambiado sin ningún tipo de seguridad (por ejemplo, TCP handshake) consume entre 400-500 mJ más que 3G con 300 ms RTT (consumo energético entre 1.3 y 1.5 W). En comparación, las 2 operaciones criptográficas de clave pública (autenticación del servidor después del paso 3 y encriptación en el paso 5) que el cliente realiza consumen del orden de decenas de mJ [PRRJ06]. Algunos de los servidores utilizan claves públicas de 1024 bits o incluso 2048, con sus respectivos consumos de 10 mJ y 50 mJ [WGE+]. En cualquier caso, estos valores son ínfimos si se comparan con el coste adicional de la comunicación, lo cual puede verse comparando la energía consumida por Facebook y su versión segura ssl.Facebook.

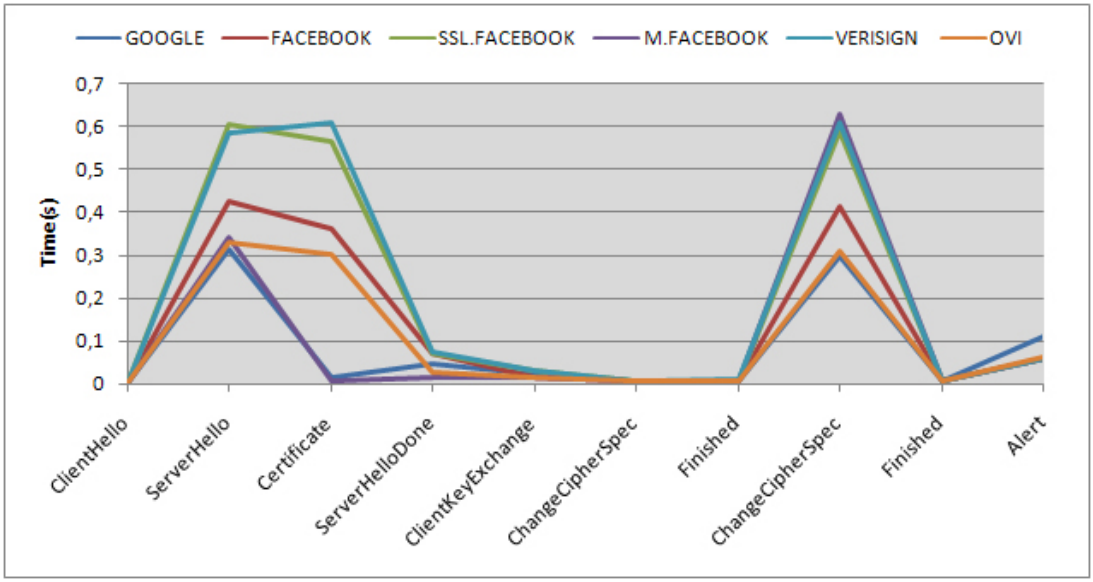


Figura 4.6: Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando RSA con 3G

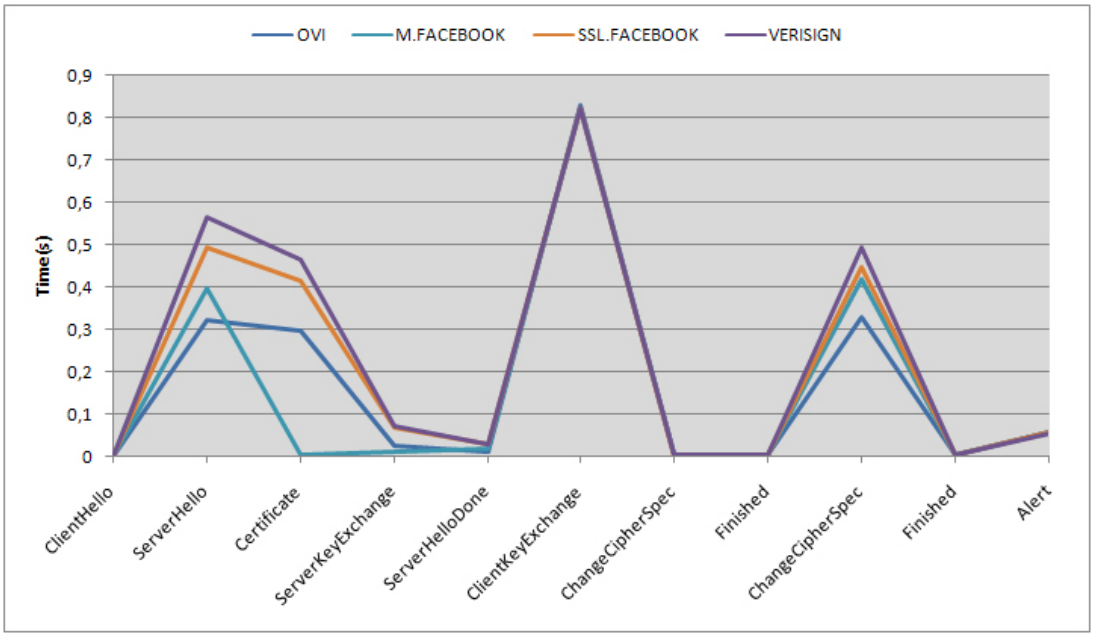


Figura 4.7: Tiempo de ejecución de los diferentes pasos de la fase de handshake utilizando DHE con 3G

4.2.5 Comparando WLAN y 3G

Una vez se han presentado los resultados de los experimentos utilizando WLAN y 3G de forma separada, en esta sección se presentan las Figuras 4.8 y 4.9 para mostrar una comparativa con los valores medios de todos los consumos energéticos y de los tiempos de ejecución de los diferentes servicios estudiados, con el fin de proporcionar una clara visión general de la diferencia de uso de las 2 interfaces de red. En el caso del uso de 3G, la media del consumo energético es un 300% mayor en el caso de la utilización de WLAN. Esto es debido en su mayor parte a las latencias mayores derivadas del uso de 3G, unido a su vez al ancho de banda inferior del mismo, lo que supone tiempos de transferencia mayores, manteniendo la interfaz de red en estado de consumo máximo más tiempo.

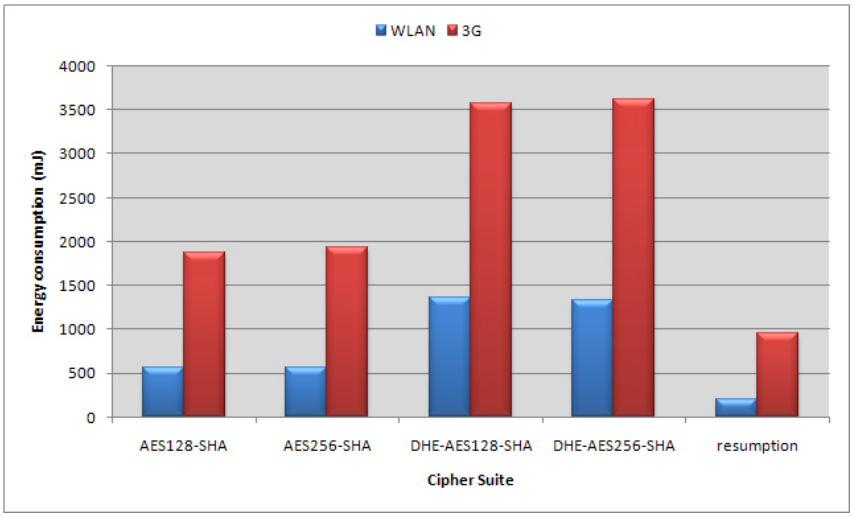


Figura 4.8: Comparación del consumo energético entre WLAN y 3G

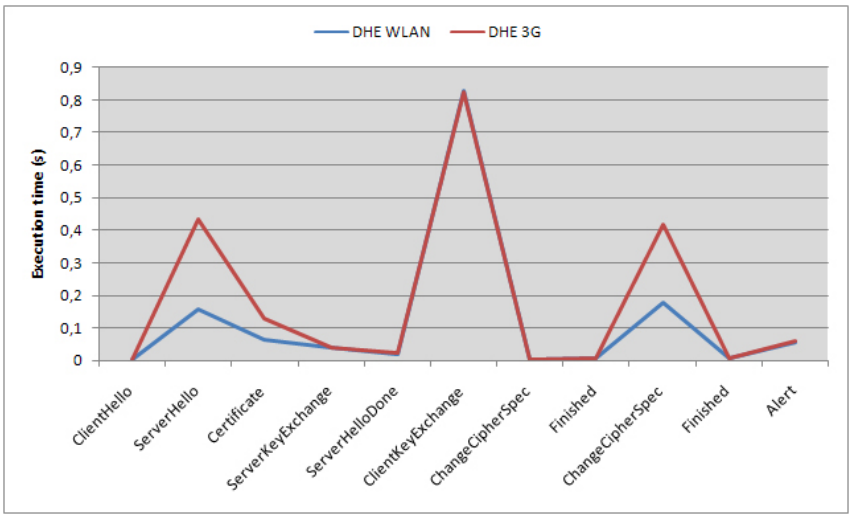


Figura 4.9: Comparación de los tiempos de ejecución de la fase de handshake entre WLAN y 3G

4.3 Rendimiento y coste adicional

En las secciones anteriores, se ha tratado de medir el consumo energético, analizando en profundidad los pasos mas costosos y midiendo también la cantidad de información intercambiada entre el cliente y el servidor. Esta sección tiene por objetivo, utilizando datos obtenidos en algunos de los experimentos previos, estudiar el coste adicional real del protocolo SSL y su rendimiento.

4.3.1 Datos adicionales en SSL

La Tabla 4.4 muestra la cantidad de información extra que se deriva del uso de SSL por el uso de RSA o DHE. La primera columna muestra el tamaño de los datos en bytes utilizados en cada experimento, desde 1KB a 5 MB. La siguiente muestra de qué manera se han enviado los datos, C->S significa de cliente a servidor y S->C, de servidor a cliente. La siguiente columna indica los datos intercambiados únicamente en el protocolo de handshake. Total RSA y Total DHE muestran los datos totales:

$$Total\_Data = Handshake\_protocol + Record\_Protocol$$

Es importante mencionar que en el protocolo de registro no sólo se incluye el tamaño de los datos de texto sin cifrar, sino que existen datos extra derivados del uso de SSL, como puede ser el hash HMAC-SHA1, y datos adicionales utilizados por el algoritmo de encriptación simétrico AES, los cuales incrementan ligeramente el tamaño de cada paquete. Las 2 siguientes, Overhead RSA (bytes) y Overhead DHE (bytes), muestran el verdadero coste adicional de SSL en bytes.

$$SSL\_Overhead = Total\_Data - Transaction\_size$$

Por último, Overhead RSA (%) y Overhead DHE (%) muestran el porcentaje del coste adicional comparado con el tamaño de los datos enviados.

$$Overhead(\%) = \frac{SSL\_Overhead(bytes)}{Transaction\_size(bytes)} \times 100$$

Este coste adicional decrece a medida que el tamaño de datos se ve incrementado, pero hay que destacar que para tamaños pequeños de datos (1024 bytes), el coste adicional puede ser de alrededor de un 300% (utilizando DHE), ya que el certificado que envia el servidor es más de un 200% mayor. Sin embargo, para tamaños mayores de 100 KB dicho coste adicional se ve reducido a un 6%, siendo un 3% en el caso de 5MB.

Transaction size(bytes)	Direction	Handshake	Total RSA(bytes)	Total DHE(bytes)	Overhead RSA(bytes)	Overhead DHE(bytes)	Overhead RSA(%)	Overhead DHE(%)
1024	C->S S->C	263 2384	1361 2384	1363 2786	2721	3125	**265,72**	**305,18**
10240	C->S S->C	263 2384	10910 2384	10912 2786	3054	3458	**29,82**	**33,77**
51200	C->S S->C	263 2384	53350 2384	53352 2786	4534	4938	**8,86**	**9,64**
102400	C->S S->C	263 2384	106400 2384	106402 2786	6384	6788	**6,23**	**6,63**
512000	C->S S->C	263 2384	530800 2384	530802 2786	21184	21588	**4,14**	**4,22**
1024000	C->S S->C	263 2384	1061300 2384	1061302 2786	39684	40088	**3,88**	**3,91**
5120000	C->S S->C	263 2384	5305300 2384	5305302 2786	187684	188088	**3,67**	**3,67**

Tabla 4.4: Perfil de los datos intercambiados entre el cliente y el servidor en una conexión SSL

4.3.2 Rendimiento en SSL

En esta sección se trata de analizar el rendimiento del protocolo SSL comparado con el uso de una conexión no segura utilizando TCP. La Tabla 4.5 muestra los resultados obtenidos en los diferentes experimentos. La primera columna indica el tamaño de los datos. La segunda muestra el número de conexiones realizadas para asegurar la precisión de los datos obtenidos. En el caso de tamaños de datos pequeños, el número de conexiones comienza en 300 y se va reduciendo a medida que dicho tamaño de datos se incrementa. Las razones de este proceso repetitivo han sido explicadas en la Sección 3.3.1. La tercera y cuarta columna indican el tiempo total de ejecución de los experimentos utilizando o no SSL. La quinta y sexta columna representan, en negrita, el rendimiento real del uso de SSL o no. Las 2 últimas columnas muestran el porcentaje medio de carga del procesador durante todo el proceso. Estos valores servirán para explicar algunos problemas encontrados en el uso de tamaños de datos grandes.

El rendimiento se obtiene multiplicando el tamaño de los datos por el número de conexiones y dividiendo este resultado por el tiempo total de ejecución:

$$Throughput(MB/s) = \frac{Transaction\_size \times Number\_of\_connections}{Total\_execution\_time \times 10^6}$$

Como el rendimiento se presenta en MB/s, este resultado tiene que dividirse por 10^6, ya que el tamaño de datos está en bytes.

La Figura 4.10 ayuda a interpretar mejor los resultados. A medida que el tamaño de datos incrementa, también lo hace el rendimiento. Esto es debido a que, a medida que los datos aumentan, el número de conexiones necesarias disminuye, y por tanto hay un menor tiempo total de ejecución.

Como se ha comentado antes, para tamaños de datos mayores a 512KB, los resultados muestran que, cuando se utiliza SSL, el procesador está funcionando a carga máxima, alcanzando un 100% de uso y limitando la velocidad en la transmision. Este problema se debe al procesador incluido en el terminal móvil, un Dual ARM 11 a 332 MHz, el cual no

Transaction size (bytes)	Number of connections	Total execution time with SSL(s)	Total execution time without SSL (s)	Throughput with SSL (MB/s)	Throughput without SSL (MB/s)	% CPU with SSL	% CPU without SSL
1024	300	20,95	5,38	0,015	0,057	85	90
10240	200	17,42	8,75	0,118	0,234	90	95
51200	150	28,66	18,08	0,268	0,425	96	98
102400	100	32,42	23,54	0,316	0,435	98	98
512000	30	48,55	31,71	0,348	0,484	100	98
1024000	10	26,48	14,74	0,387	0,695	100	96
5120000	5	63,38	39,74	0,404	0,644	100	97

Tabla 4.5: Rendimiento del protocolo SSL y un canal no seguro TCP

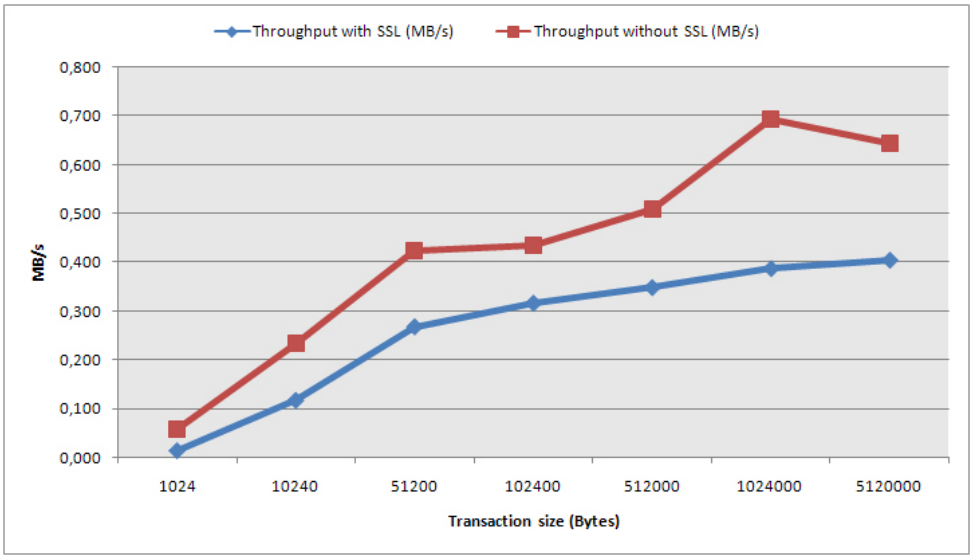


Figura 4.10: Comparación del rendimiento utilizando SSL y un canal no seguro TCP

es capaz de procesar las operaciones criptográficas a tiempo, creando un cuello de botella que limita el rendimiento general.

4.3.3 Consumo energético adicional en SSL

Mientras que en la sección anterior se ha medido el rendimiento, en esta se pretende medir el consumo energético adicional derivado del uso de SSL, tanto utilizando WLAN como 3G.

Las Figuras 4.11 y 4.12 representan gráficamente los resultados obtenidos. En resultados anteriores, a medida que el tamaño de los datos incrementaba el coste adicional disminuía. Teóricamente, y tal como se ha visto en trabajos anteriores, el coste energético adicional del uso de SSL debería disminuir a medida que el tamaño de los datos se ve incrementado. Estos experimentos muestran que esta tendencia es correcta utilizando 3G pero muestran otros resultados con el uso del interfaz WLAN. Este hecho está relacionado con el factor limitante del procesador comentado en la sección anterior, en el cual el terminal no es capaz de completar todas las operaciones criptográficas a tiempo. Utilizando WLAN no existe latencia entre las operaciones y los ACKs del servidor llegan con retardo menor a 1

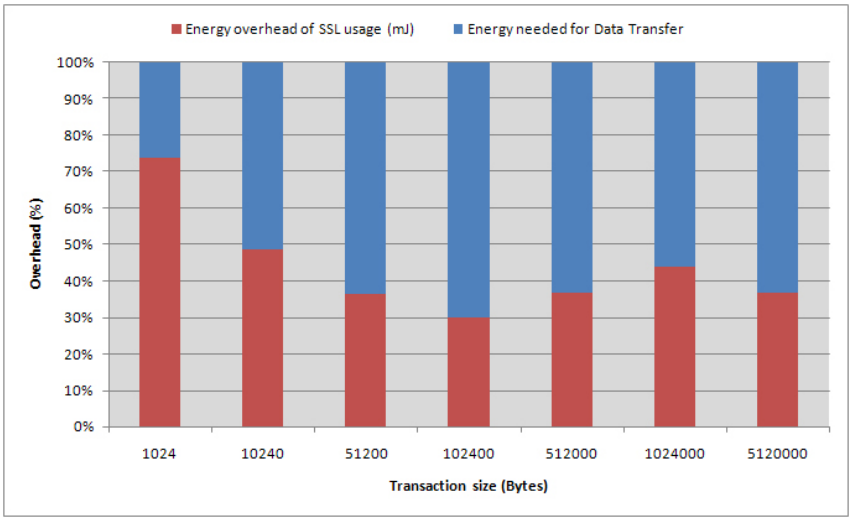


Figura 4.11: Coste energético adicional del uso de SSL utilizando WLAN

ms. Sin embargo, utilizando 3G dichos retardos son de más de 100 ms, tiempo suficiente para que el terminal complete las operaciones necesarias y mande el siguiente paquete.

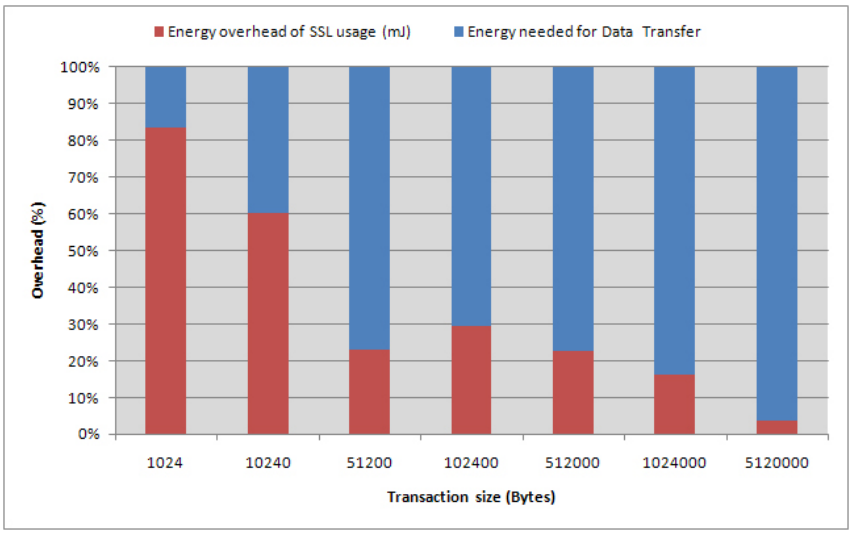


Figura 4.12: Coste energético adicional del uso de SSL utilizando 3G

En la Sección *5.3.3 SSL Energy Consumption Overhead* del Anexo C se puede encontrar más información acerca de estos experimentos, así como resultados y conclusiones adicionales comparando el coste energético total entre 3G y WLAN.

4.3.4 Consumo energético de cada uno de los componentes

Utilizando las mediciones detalladas en experimentos previos, en conjunción con nuevas métricas (ver *Appendix B Study of Symmetric and Hash algorithms* del Anexo C), en esta sección se calcula el coste energético individual de cada uno de los componentes que

intervienen en la creación y uso de una sesión SSL. Estos componentes pueden dividirse en 4 clases principales: algoritmos simétricos, asimétricos, *hashing* e interfaces de red. En secciones anteriores se ha estudiado el coste energético de algunas de ellas, pero no el coste de el algoritmo de encriptación (AES) ni del algoritmo HMAC-SHA1.

Para llevar a cabo dicho estudio, se realizaron pequeños experimentos para medir sus diferentes consumos, así como el impacto energético que tienen a su vez las operaciones de lectura/escritura en la memoria del teléfono, los cuales parece que no se tomaron en cuenta en estudios previos [PRRJ06]. Los resultados para cada una de estos se muestran en la Tabla 4.6. Mientras que los valores obtenidos se encuentran en órdenes de magnitud similares a los reportados anteriormente, lo más importante a destacar son los resultados energéticos de las operaciones de lectura/escritura. Estos sugieren que las operaciones de clave simétrica y de *hashing* son pequeñas si se comparan con estas últimas: la operación de encriptación utilizando AES-128 y *hashing* con SHA1 consume únicamente $0.7\mu J/B$ y $0.1\mu J/B$, mientras que las operaciones de lectura/escritura del fichero sobre el que se trabaja requieren $1.3\mu J/B$, claramente superior. Esta energía se ve a su vez incrementada si se realizan operaciones sobre datos alojados en la tarjeta de memoria externa del móvil $2.9\mu J/B$ (en este caso microSD). Se puede encontrar mucha más información sobre estos experimentos y sus resultados en el *Appendix B Study of Symmetric and Hash algorithms* del Anexo C.

Operation	**Energy** ($\mu J/B$)
Encrypt with AES-256 (r/w phone memory)	2.0
Decrypt with AES-256 (r/w phone memory)	2.0
AES-256 vs. AES-128	0.3
Hash with SHA1 (read phone memory)	0.7
Read only (phone memory)	0.6
Read & write (phone memory)	1.3
Read only (memory card)	1.2
Read & write (memory card)	2.9

Tabla 4.6: Coste energético de diferentes operaciones criptográficas

Una vez que ya se han estudiado todos los componentes involucrados por separado, las Figuras 4.13 y 4.14 muestran los resultados con el uso de RSA y de DHE, respectivamente. Es fácil darse cuenta de que para transmisiones pequeñas de datos el consumo energético corresponde casi en su totalidad a algoritmos de clave pública, principalmente en la fase de handshake. Sin embargo, a medida que el tamaño aumenta, las operaciones simétricas reemplazan a los algoritmos asimétricos como agentes dominantes del total de energía utilizada por procesamiento criptográfico. Asimismo, la contribución energética derivada de los algoritmos de *hashing* también se incrementa con el tamaño de datos, aunque se mantiene como un factor mínimo. Cabe mencionar que estos valores han sido obtenidos utilizando la interfaz de red WLAN. Para el caso de 3G no se llevaron a cabo experimentos en la realización del proyecto, aunque sí fueron analizados en la redacción del articulo (ver Anexo B), afirmando que el coste energético asociado a las operaciones criptográficas es mucho menor si se compara con el coste asociado a la interfaz de red.

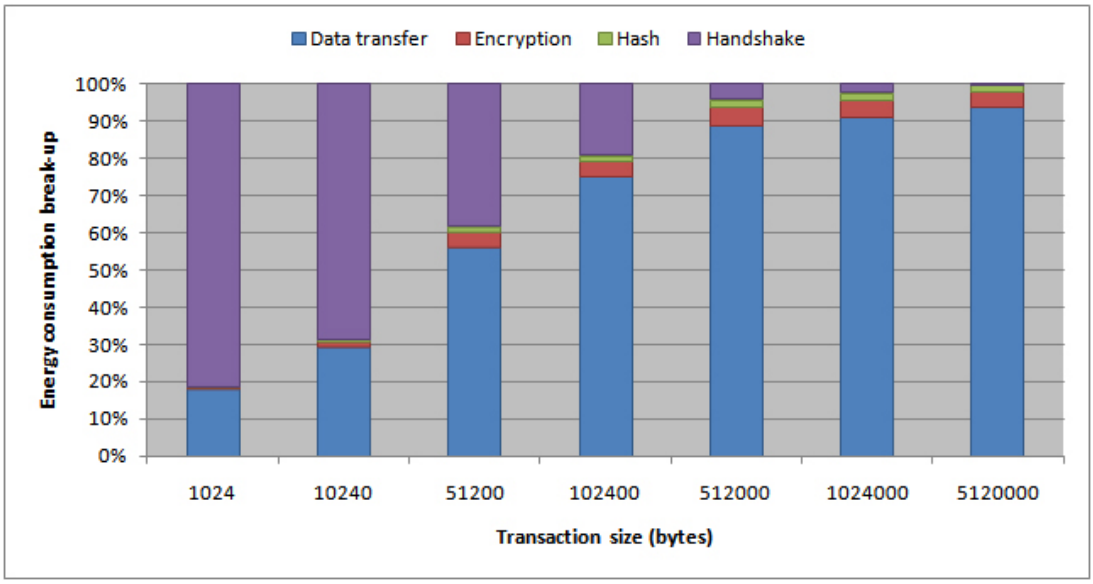


Figura 4.13: Estudio individual del coste energético de los distintos componentes empleados en SSL para el caso RSA

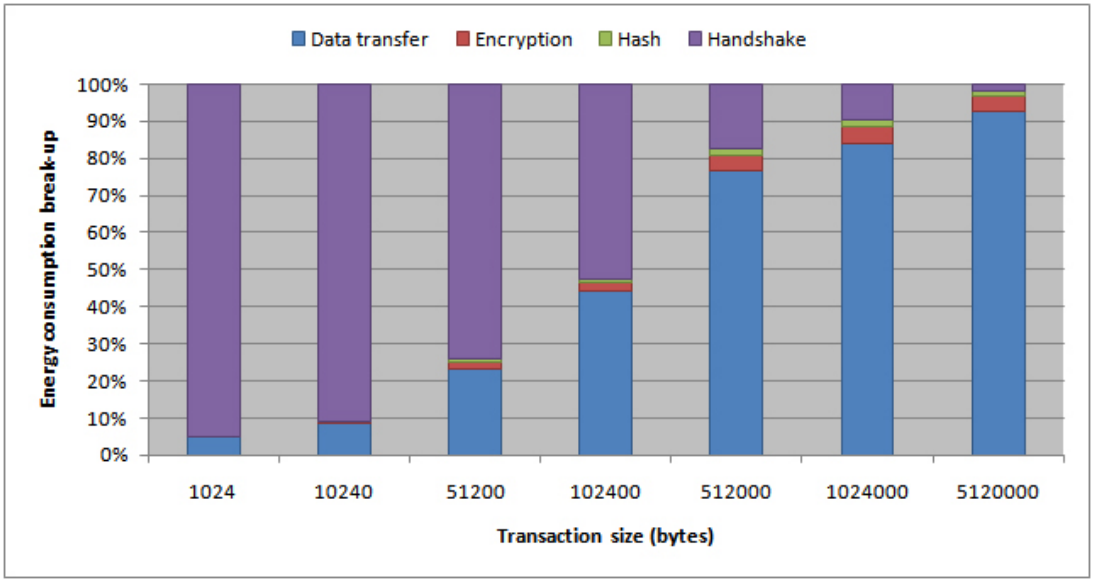


Figura 4.14: Estudio individual del coste energético de los distintos componentes empleados en SSL para el caso DHE

Capítulo 5. Conclusiones

En este Capítulo se presentan las conclusiones generales del trabajo desarrollado junto con la valoración personal y una posibles vía de investigación futura para continuar con el estudio.

5.1 Conclusiones a nivel técnico

En este trabajo se ha presentado un estudio de medición del consumo de diferentes algoritmos criptográficos y de protocolos de seguridad, centrándose en el protocolo *Secure Socket Layer* (SSL). Para su estudio, se ha presentado una configuración consistente en un cliente móvil ejecutándose en un terminal móvil Nokia N95 y una aplicación servidor gobernada por un ordenador portátil. Además se ha realizado un completo estudio de los diferentes servicios remotos más utilizados en la actualidad, como son Google o Facebook, obteniéndose conclusiones significativas en los diversos aspectos analizados y observándose que el consumo energético en la fase de handshake varía considerablemente entre los diferentes servicios.

Mientras que en la especificación de HTTP/1.1 se menciona que "un cliente no debería de mantener más de 2 conexiones con cualquier servidor o *proxy*" [FGM+99], los navegadores de hoy en día constantemente rompen esta parte de la especificación y establecen muchas más conexiones [DRC+10]. Esto supone un impacto significativo en el consumo adicional de energía en el caso de los clientes móviles, especialmente cuando se utiliza SSL y el servidor al que se desea acceder no soporta la reanudación de sesión. Sorprendentemente, los servidores de Facebook, por ejemplo, son uno de esos casos.

Existen otras oportunidades para la optimización, especialmente para transferencias pequeñas de datos donde la fase de *handshake* juega un papel importante. Esta pasa por reducir el tamaño del certificado del servidor y evitar el uso de Diffie-Hellman en favor de RSA, a menos que sea absolutamente necesario. De cualquier otro modo, reducir, por ejemplo, la longitud de la clave pública del servidor tiene un impacto pequeño.

Quizá el factor más importante a tener en cuenta, contrariamente a investigaciones previas realizadas [PRRJ06, SGM09], es que los resultados obtenidos en este estudio revelan que una vez el tamaño de los datos supera los 500 KB, el coste adicional comienza a ser mucho menos importante independientemente del uso de WLAN o 3G como interfaz de red. Además, el coste adicional que supone la encriptación, la protección de la integridad o los accesos de lectura/escritura en el protocolo de registro es prácticamente insignificante.

De este modo, la elección del algoritmo simétrico o de *hashing* no va a suponer un cambio muy grande desde el punto de vista energético, excepto si los ratios de transmisión son muy altos. En dicho caso, son necesarios algoritmos criptográficos mas eficientes para prevenir que se puedan convertir en un cuello de botella, limitando el rendimiento y reduciendo la eficiencia energética, como se ha visto en la Sección 4.3.2.

5.2 Conclusiones a nivel personal

La realización de este trabajo para una empresa grande como es Nokia supone un reto importante, tanto técnico como también personal. Estar en continuo contacto y trabajando con personas experimentadas e investigadores del centro de desarrollo e investigación que Nokia tiene en Helsinki ha sido una experiencia muy enriquecedora y que creo será muy útil en un futuro próximo, a la hora de mi incorporación al mundo laboral.

Además, haber realizado el proyecto fin de carrera en Finlandia supone también una experiencia muy positiva, no sólo por el hecho de estar en otro país y otra cultura, sino también por la posibilidad de aprender otros entornos de trabajo y, por supuesto, superar otro reto importante, el del idioma.

Destacar también la realización del artículo *TLS and Energy Consumption On a Mobile Device: A Measurement Study* incluído en el Anexo B. Creo que es una experiencia muy positiva que valora muy bien el trabajo realizado en el proyecto, recompensado tanto por mi director de proyecto como los expertos de Nokia con la posibilidad de publicar los resultados obtenidos de forma oficial en un congreso tan importante como es el del IEEE International Conference on Communications ICC2011.

Por último, obtener el reconocimiento de mi trabajo en la Aalto university, valorando mi proyecto y mi esfuerzo con una calificación de excelente (se incluye dicha valoración y calificación en el Anexo D, supuso una satisfacción máxima y el darme cuenta de que había superado con éxito el reto que me planteé: realizar un proyecto para una empresa grande en el extranjero.

5.3 Trabajo futuro

Aunque el trabajo realizado se ha desarrollado sobre un conjunto definido de suites criptográficas, es posible continuar la investigación, ampliando el rango de suites criptográficas y otras técnicas de seguridad, reutilizando la misma configuración experimental. Un ejemplo de este posible trabajo futuro sería el estudio de criptografía de curva elíptica [BWBL02].

La criptografía de clave pública está principalmente basada en problemas matemáticos utilizando operaciones complejas. RSA está basado en la dificultad de factorizar enteros, mientras que Diffie-Hellman se basa en el problema de logaritmos discretos en el campo de los enteros. El estudio de algoritmos basados en criptografía de curva elíptica (Elliptic Curve Cryptography (ECC)) proporciona seguridad basándose en el problema de

logaritmos discretos pero en el campo de las curvas elípticas. Estos algoritmos pueden proporcionar el mismo nivel de seguridad que RSA o Diffie-Hellman, pero utilizando claves más cortas. Dichas equivalencias se presentan en la Tabla 5.1.

Symmetric	**ECC**	**DH/RSA**
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Tabla 5.1: Equivalencias de seguridad (longitud de clave en bits) entre criptografía simétrica, de curva elíptica y de clave pública

Utilizando ECC como criptografía de clave pública se puede reducir significativamente el consumo energético, así como mejorar la seguridad, ya que una clave de curva elíptica de 163 bits proporciona el mismo nivel de seguridad que una clave de 1024 bits utilizando RSA.

Aunque existen implementaciones prácticas y funcionales (OpenSSL proporciona herramientas que soportan este tipo de criptografía), el uso de ECC está siendo estudiado frente posibles ataques criptoanalíticos y no es soportado todavía por los principales proveedores de Internet.

Abreviaturas y acrónimos

3G	3rd Generation
3GPP	3rd Generation Partnership Project
AES	Advanced Encryption Standard
API	Application Programming Interface
CFB	Cipher Feedback
CBC	Cipher Block Chaining
CSV	Comma Separated Values
CTR	Counter mode
DES	Data Encryption Standard
DSS	Digital Signature Standard
ECB	Electronic Code Book
DH	Diffie-Hellman
HMAC	Hash-based Message Authentication Code
HSPA	High-Speed Packet Access
HSDPA	High-Speed Downlink Packet Access
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
LAN	Local Area Network
MAC	Message Authentication Code
MD5	Message Digest 5
NEP	Nokia Energy Profiler
NSA	National Security Agency
NIST	National Institute of Standards and Technology
OFB	Output Feedback
PNG	Portable Network Graphics
RC4	Rivest Cipher 4

RSA	Rivest Shamir Adelman
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UEA2	UMTS Encryption Algorithm Suite 2
UIA2	UMTS Integrity Algorithm Suite 2
UMTS	Universal Mobile Telecommunication System
WLAN	Wireless Local Area Network

Bibliografía

[01] *SSL and TLS: designing and building secure systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[APS99] George Apostolopoulos, Vinod Peris, and Debanjan Saha. Transport layer security: How much does it really cost. In *Proceedings of the IEEE INFOCOM*, pages 717–725, 1999.

[BBV09] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference IMC '09*, pages 280–293, New York, NY, USA, 2009. ACM.

[BWBL02] S. Blake-Wilson, D. Brown, and P. Lambert. Use of elliptic curve cryptography (ecc) algorithms in cryptographic message syntax (cms), 2002.

[Cor96] Netscape Communications Corp. Ssl 3.0 protocol specification. `http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00`, 1996.

[DA99] T. Dierks and C. Allen. RFC 2246: The TLS protocol version 1, 1999.

[DRC+10] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing TCP's initial congestion window. *SIGCOMM Comput. Commun. Rev.*, 40(3), 2010.

[EJ01] D. Eastlake, 3rd and P. Jones. Us secure hash algorithm 1 (sha1), 2001.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, 1999.

[Fit07] Frank Fitzek. External energy consumption measurements on mobile phones. In *Mobile Phone Programming and its Application to Wireless Networking*, pages 441–447. Springer, 2007.

[Kau99] Thayer Kaukonen. A stream cipher encryption algorithm "arcfour". http://www.mozilla.org/projects/security/pki/nss/draft-kaukonen-cipher-arcfour-03.txt, 1999.

[Lab93] RSA Laboratories. PKCS #6: Extended-certificate syntax standard. http://www.rsa.com/rsalabs/node.asp?id=2128 - Último acceso: 2 Noviembre 2010, 1993.

[Ope10] *OpenSSL project*, 2010. Disponible en http://www.openssl.org - Ultimo acceso: 2 Noviembre 2010.

[oST01] National Institute of Standards and Technology. *Specification for the ADVANCED ENCRYPTION STANDARD (AES)*, 2001. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[oST06] National Institute of Standards and Technology. *FIPS PUB 186-3: Digital signature standard (DSS)*, 2006. Available at http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3%20_-March2006. pdf.

[PP06] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[PRRJ06] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, 5(2):128–143, 2006.

[Riv92] R. Rivest. The md5 message-digest algorithm, 1992.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[SGM09] Youngsang Shin, Minaxi Gupta, and Steven Myers. A study of the performance of SSL on PDAs. In *INFOCOM'09: Proceedings of the 28th IEEE international conference on Computer Communications Workshops*, pages 1–6, Piscataway, NJ, USA, 2009. IEEE Press.

[WGE+] A.S. Wander, N. Gura, H. Eberle, V. Gupta, and S.C. Shantz. Energy analysis of public-key cryptography for wireless sensor networks. In *Proceedings of PerCom 2005*.

[WH76] Diffie Whitfield and Martin E. Hellman. New directions in cryptography, 1976.

[XSK+] Yu Xiao, Petri Savolainen, Arto Karppanen, Matti Siekkinen, and Antti Ylä-Jääski. Practical power modeling of data transmission over 802.11g for wireless applications. In *Proceedings of e-Energy 2010*.

Anexos

Anexo A. El protocolo de Handshake en SSL

Como se ha comentado en la Sección 2.3, este protocolo es el encargado de la negociación de los sistemas criptográficos a utilizar entre los 2 pares que quieren establecer el canal seguro SSL. Con el fin de permitir a los usuarios seleccionar el nivel de seguridad que mejor se ajuste a sus necesidades, SSL agrupa los diferentes sistemas criptográficos en suites de cifrado. Cada una de estas suites especifica el algoritmo de autenticación de los pares, el de intercambio de clave, el de encriptación de los datos y el de resumen del mensaje o *message digest*. En la Tabla A.1 se muestran 2 ejemplos de suites criptográficas.

Existen diferentes algoritmos criptográficos a utilizar para codificar los datos, para calcular la MAC o autenticar los pares involucrados. Algunos de ellos proporcionan niveles altos de seguridad, pero tiene altos costes computacionales. Otros son menos seguros, pero tienen mejores rendimientos generales sin comprometer la seguridad del sistema.

Suite criptográfica	**Autenticación**	**Intercambio clave**	**Encriptación**	**Digest**
RSA_AES_256_SHA	RSA	RSA	AES-256-CBC	SHA1
DHE_RSA_AES_256_SHA	RSA	DHE	AES-256-CBC	SHA1

Tabla A.1: Ejemplos de suites criptográficas

Cuando se establece una conexión SSL, el cliente y el servidor intercambian información acerca de qué suites soportan y están dispuestos a utilizar. El cliente envía la lista de suites disponibles, ordenadas normalmente en orden de seguridad decreciente. El servidor elige la más adecuada entre las compartidas. Si ambos pares no comparten ninguna, la comunicación SSL no es posible y el servidor cierra el intento de conexión.

Con el fin de explicar cómo se lleva a cabo un handshake real, aquí se presenta un resumen de los mensajes intercambiados entre el cliente y el servidor. El método más usado y conocido de handshake es mediante el empleo de RSA como protocolo de intercambio de clave y autenticando únicamente al servidor, y no al cliente. La Figura A.1 muestra los mensajes y la información intercambiada. El mensaje entre corchetes es usado únicamente por el protocolo de acuerdo de clave Diffie-Hellman.

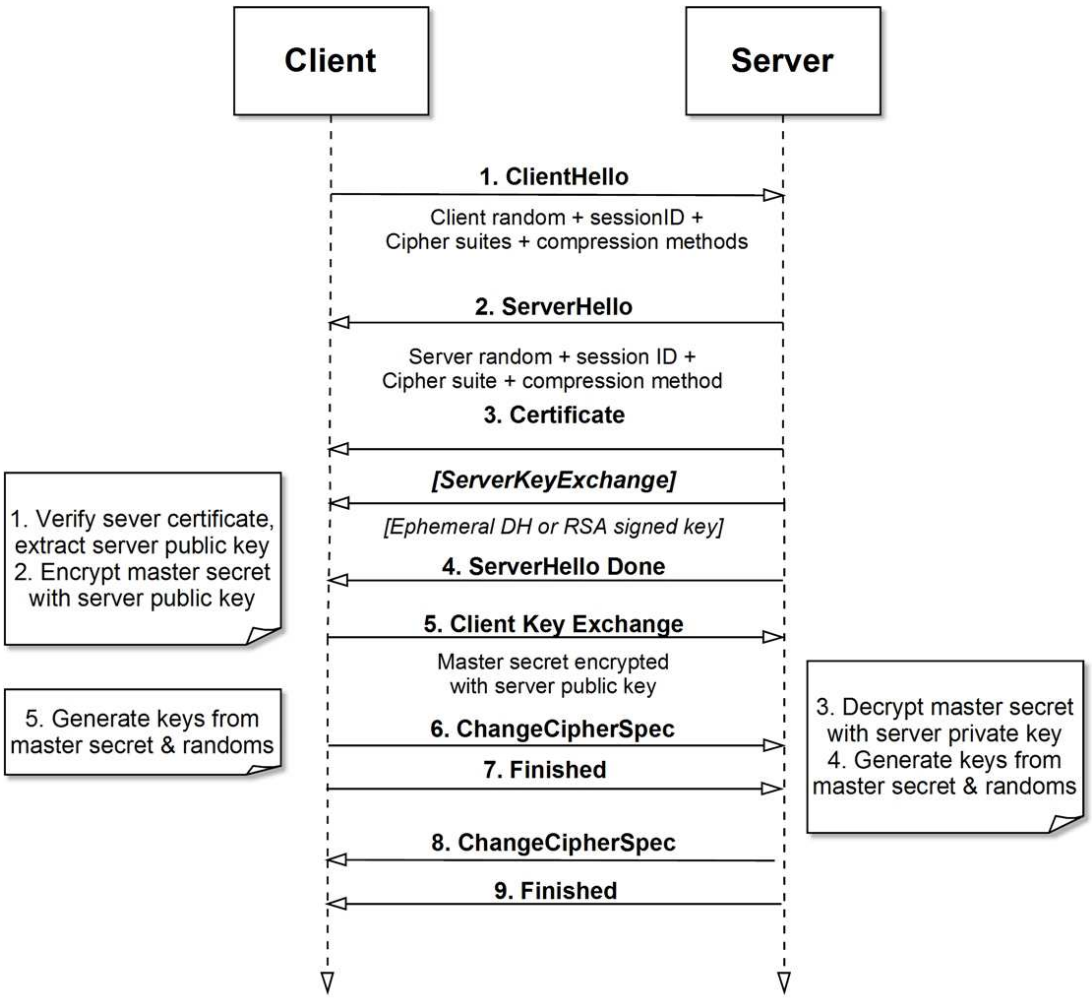


Figura A.1: Pasos e información intercambiada en la fase de handshake

La secuencia de acciones que ocurre en el intercambio de mensajes durante la fase de handshake se resume a continuación:

1. La conexión comienza con el cliente enviado un comando `ClientHello`, el cual contiene: la versión SSL más alta soportada, las suites criptográficas que soporta, métodos de compresión disponibles, ID de la sesión y datos aleatorios para su uso en el proceso de generación de la clave secreta.

2. El servidor responde con un comando `ServerHello`, que incluye, la versión SSL que se utilizará para la comunicación, la suite criptográfica seleccionada por el servidor entre las ofrecidas por el cliente, el método de compresión elegido por el servidor, el ID de la sesión establecido por el servidor y datos aleatorios para su uso en el proceso de generación de la clave secreta.

3. El servidor envía el comando `Certificate` que incluye el certificado del servidor con la clave pública del mismo incluida. De manera opcional, también incluye la cadena de certificados adicionales comenzando con el de la autoridad certificadora (CA).

4. El servidor envía el comando `ServerHelloDone`, indicando que al cliente que ha terminado con esta fase del handshake.

5. El cliente envía el mensaje `ClientKeyExchange`, que contiene el *pre_ master_ secret* creado por el cliente y encriptado utilizando la clave pública del servidor. Ambas partes generan la clave secreta de encriptación utilizando dicho *pre_ master_ secret* y los datos aleatorios incluidos en los mensajes `ClientHello` y `ServerHello`.

6. El cliente envía el comando `ChangeCipherSpec`, el cual indica que los siguientes mensajes enviados por el cliente durante la sesión serán encriptados utilizando los sistemas criptográficos y claves acordadas previamente.

7. El cliente envía su último mensaje en el protocolo de handshake, `Finished`, incluyendo un digest de todos los mensajes previos enviados al servidor. Este comando es enviado para asegurar que ninguno de los mensajes previos sin encriptar han sido modificados durante su transmisión.

8. El servidor envía el comando `ChangeCipherSpec`, el cual indica que los siguientes mensajes enviados por el servidor durante la sesión serán encriptados utilizando los sistemas criptográficos y claves acordadas previamente.

9. El servidor envía su último mensaje en el protocolo de handshake, `Finished`, incluyendo un digest de todos los mensajes previos enviados a el cliente. Este comando tiene el mismo propósito que el mismo `Finished` enviado por el cliente.

Estas 9 acciones previas muestran los pasos exactos que ocurren en un SSL handshake utilizando RSA como protocolo de autenticación y como intercambio de clave. Aunque RSA es el algoritmo de clave pública más utilizado en el mercado, SSL soporta suites criptográficas basadas en otros algoritmos como Diffie-Hellman (DH). La idea de estos

algoritmos es evitar el uso de sistemas patentados y protegidos, en particular RSA. Pero desde el año 2000, cuando la patente sobre este último algoritmo expiró, esta motivación ya no es válida, aunque el soporte para DH está todavía disponible.

Al contrario que RSA, que puede ser usado como algoritmo de intercambio de clave o de firma, DH sólo puede ser utilizado como algoritmo de acuerdo de clave. Como consecuencia directa, y para conseguir una solución completa, Diffie-Hellman y RSA (u otro protocolo como DSS) tienen que ser usados en conjunción. La solución más extendida es el uso de claves efímeras en Diffie-Hellman o DHE. El servidor genera una clave DHE temporal y la firma utilizando la clave RSA, transmitiendo la clave firmada en el mensaje `ServerKeyExchange`. En ese caso, el cliente utilizará esa clave DHE para el acuerdo de clave. Es posible utilizar una clave DH de larga duración. En ese caso, el servidor tendrá un certificado firmado con dicha clave. El uso de DH como protocolo de acuerdo de clave únicamente añade un mensaje en el lado del servidor, pero también añade un conjunto de operaciones asimétricas que reduce el rendimiento e incrementa el coste energético.

La parte más costosa de un SSL handshake es el establecimiento de el *pre_master_secret*, el cual requiere criptografía de clave pública. SSL facilita la reanudación de una sesión previamente establecida. El uso de dicha reanudación conlleva la utilización de una identificación o ID de sesión entre un cliente y un servidor y por tanto el handshake únicamente se lleva a cabo en 5 pasos: 1, 2, 7, 8 y 9 de la Figura A.1. Esto significa que no es necesario negociar de nuevo el *master_secret* porque la sesión ya había sido creada previamente por los 2 pares. Este hecho evita realizar de nuevo las operaciones computacionalmente costosas que requiere los algoritmos de clave pública, ahorrando tiempo y reduciendo enormemente el consumo energético.

Anexo B. Artículo: *TLS and Energy Consumption On a Mobile Device: A Measurement Study*

En este anexo se presenta el artículo de investigación con título *TLS and Energy Consumption On a Mobile Device: A Measurement Study* enviado al *Communications QoS, Reliability and Modeling Symposium* enmarcado en el congreso *IEEE International Conference on Communications ICC2011*[1] que se celebrará en Kyoto (Japón) del 5 al 9 de junio de 2011.

Este artículo fue enviado a dicho congreso en septiembre de 2010 y está pendiente de aprobación en enero de 2011.

[1] http://www.ieee-icc.org/ - Último acceso: 3 noviembre 2010

TLS and Energy Consumption On a Mobile Device: A Measurement Study

Pedro Miranda, Matti Siekkinen
Aalto University, School of Science and Technology
Espoo, Finland
Email: firstname.lastname@tkk.fi

Heikki Waris
Nokia Research Center
Helsinki, Finland
Email: heikki.waris@nokia.com

***Abstract*— We report results from a measurement study on the role of the most popular end-to-end security protocol Transport Layer Security (TLS) in the energy consumption of a mobile device. We measured energy consumed by TLS transactions between a Nokia N95 and several popular Web services over WLAN and 3G network interfaces. Our detailed analysis corroborates some earlier results but also reveals, contrary to earlier studies, that the transmission and I/O energy, both in the TLS handshake and the record protocol, far exceed the required computational energy by the actual cryptograhic algorithms and that with transactions larger than 500KB, the energy required to transmit the actual data clearly overweighs the TLS energy overhead. In addition, we note that the energy consumption varies remarkably between measured services.**

I. Introduction

In the last few years, mobile devices have evolved significantly in terms of power, throughput, and in terms of new functionalities, but they are still severely constrained by limited battery life-time. Secure communications are achieved by employing security protocols, which are based on cryptographic algorithms.

Executing certain cryptographic algoritms requires rather intesive computations. Therefore, their energy consumption on these battery powered devices is naturally a concern. In this paper, we study the energy consumption of Transport Layer Security (TLS), which is used to establish a secure communication channel between two end hosts and exchange data over that channel. TLS is a transport level protocol that uses asymmetric and symmetric encryption algorithms and hash algorithms in order to provide data secrecy, authentication of the communication parties, and data integrity for applications in a transparent manner.

We use a Symbian mobile device (Nokia N95) to establish TLS connections to different web services, such as electronic email or social networks, over both WLAN and 3G network interfaces and measure the energy consumption. TLS comprises two phases. First, a security association is created through a handshake protocol after which the actual data is transferred over an encrypted and integrity protected channel. We perform detailed analysis of the different steps involved in TLS transactions, compute the amount of energy overhead in a TLS transaction, and explain the major causes that determine this overhead.

Energy consumption of different cryptographic algorithms as well as the performance of TLS on PDAs and in computers have been studied earlier in, e.g. [1]–[3]. Among other things, they show that public key cryptography has the highest energy consumption, while hash algorithms have a little impact in the battery life-time. The key size chosen has a dramatic impact to the energy consumed in public key cryptography but not for symmetric algorithms. While some of our results corroborate those presented in these earlier studies, some of our conclusions concerning the overall TLS overhead differ significantly. In addition, we study real on-line services in a comparative manner and conduct experiments over both WLAN and 3G interfaces. Furthermore, we drill down into the individual steps involved in a transaction in order to pinpoint reasons for observed differences.

Our main findings are the following:

- The amount of energy consumed during the handshake phase differs a lot between different services. The main reasons turn out to be the length of the server certificate and RTT which are both related to transmission energy. Public key length seems to have only a marginal impact.
- Transactions over 3G access consume several times more energy than those over WLAN access. Contrary to earlier reported results, we found that the energy overhead of the TLS record protocol (encryption and hashing) is rather insignificant for WLAN and completely insignificant for 3G.
- For very small transactions (less than 10KB), the TLS overhead accounts for more than 60% (and up to 95% with DH) of the total energy consumed for both access types, while for transactions larger than 500KB the overhead is rather small if RSA is used in the handshake.

II. TLS and Energy Consumption

A. TLS overview

TLS [4] is a protocol that provides confidentiality, authentication, and data integrity over a channel between two machines. It is divided in two parts: *Handshake protocol and Record protocol*. The Handshake protocol allows the client and the server to authenticate each other and to negotiate the cryptographic algorithms and encryption keys needed to secure the channel. TLS packs the different cryptographic algorithms into cipher suites each of which specifies the server authentication algorithm, the key exchange algorithm, the bulk encryption algorithm and the message digest algorithm. Figure 1 shows the exchanged messages during a

TLS handshake using RSA or Diffie-Hellman(DH) as a key exchange protocol.

Unlike RSA, which can be used either for key exchange or for signing, DH can only be used for key agreement. As a consequence, Diffie-Hellman and RSA (or other protocol like DSS) have to be used together. The most common way to use DH is using ephemeral keys. The server generates a temporary DHE key, signs it with its RSA key, and transmits the signed key in the `ServerKeyExchange` message. In that case, the client will use that DH key for the key agreement. It is also possible to have a long-term DH key in which case the server will have a signed certificate containing the DHE key. The use of Diffie-Hellman as a protocol for the key agreement only adds one message from the server side, but it also adds different asymmetric operations that involve relatively heavy computations.

The most expensive part of the TLS handshake is the establishment of the *pre_master_secret*, which requires public key cryptography. The use of session resumption involves the reuse of a previous established session between the client and the server and thus only steps 1, 2, 7, 8 and 9 in Figure 1 are needed, thus avoiding the computationally expensive public key operations.

The Record protocol provides confidentiality and data integrity for the actual data transfer through the use of encryption and message digests.

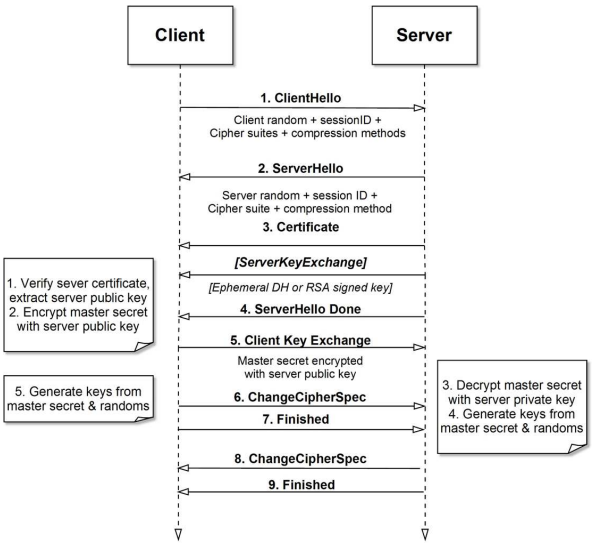


Fig. 1. Steps and exchanged data involved in a normal RSA handshake. The step within brackets is only used in the Diffie-Hellman handshake protocol

B. Where Does the Energy Go?

The use of the TLS protocol to secure a communication channel involves an overhead of computational work and exchanged data, that leads into a higher energy consumption. We can divide the energy consumed during an entire TLS transaction into cryptographic and non-cryptographic components. The former ones consist of the asymmetric operations, i.e. all the public key operations, symmetric operations, i.e. encryption and decryption of the bulk data, and hashing, i.e. message digest and digital signatures. The latter consists of the *transmission energy*, i.e. all the energy consumed while transfering data over TCP and maintain the network interfaces active. This energy is larger with TLS than without it due to increased data volume due to additional items such as certificates and message digests.

C. Energy Consumed by WLAN vs. 3G

The 3G interface operates in three different modes: *IDLE* mode is used in absence of network activity, *Dedicated Channel (DCH)* mode ensures the highest throughput with low delay transmission at the cost of a high energy consumption, and *Forward Access Channel (FACH)* mode shares the channel with other 3G devices and is used when there is little traffic to transmit, having roughly 50% of the power consumption of DCH mode. The transitions from DCH to FACH and from FACH to IDLE are controlled with operator set inactivity timers whose values usually are measured in several seconds. Because of this, there is so called *tail energy* which is spent in maintaining the high-power state after the completion of the data transmission, which, for example, can be as high as 60% of the total energy for a 50KB transfer [5]. We do not include this in our measurements.

The energy consumption of a WLAN interface depends on the following three factors: *scanning and association energy* is dissipated during the searching for an access point and connecting to one of them, *transmission energy* is required for the data transmission, and *maintenance energy* is used to keep the WLAN interface up. We do not include the scanning and association in our measurements. In addition, 802.11 includes a so called Power Saving Mode (PSM) which reduces energy consumption by enabling devices to go to a sleep mode and only regularly wake up to listen to a beacon which informs of new arriving data. We used off-the-shelf mobile device and access point in our measurements which support PSM and have it enabled by default.

III. Experiment Setup

We chose to use Advanced Encryption Standard (AES) because it is one of the most used algorithms and defined as a cryptographic standard by the NIST [6]. Furthermore, previous studies of symmetric algorithms [1] have shown that AES has good performance with competitive energy costs, and it is a sufficiently recent and secure standard that has relevance in actual implementations and products on the market. Thus, the AES cipher suites extending TLS v1.0, defined in [7] have been used, cf. Table II. From the 12 cipher suites supporting AES in the TLS, we discarded some for the following reasons: We wanted to study the differences between RSA and DHE. The use of Anonymous Diffie-Hellman (ADH) provides confidentiality but no authentication, which makes man-in-the-middle attacks possible. The use of the Digital Signature Standard (DSS) results in a slower performance [4] if it is compared to RSA. OpenSSL only implements ephemeral Diffie-Hellman (DHE), but not Diffie-Hellman (DH).

TABLE I

FULL PROFILE OF THE REMOTE SERVICES ANALYZED

Service	Google	Facebook	SSL.Facebook	M.Facebook	Verisign	Ovi
Protocols available	SSLv3, TLSv1.0	SSLv3, TLS1.0	SSLv3, TLS1.0	SSLv3, TLS1.0	SSLv3, TLSv1.0	SSLv3, TLS1.0
Server Implementation	Apache mod_SSL	Apache mod_NSS	Apache mod_NSS	Apache mod_NSS	Apache mod_SSL	IIS 7.5
Default Cipher Suite	AES256-SHA	RC4-MD5	RC4-MD5	RC4-MD5	DHE-RSA-AES256-SHA	DHE-RSA-AES256-SHA
Server Certificate length	1625 bytes	4553 bytes	4642 bytes	836 bytes	4519 bytes	1738 bytes
Public server-key	1024 bits	1024 bits	2048 bits	1024 bits	2048 bits	1024 bits
Session resumption	YES	NO	NO	NO	YES	YES
Ephemeral Diffie Hellman	NO	NO	YES	YES	YES	YES

TABLE II

LIST OF CIPHER SUITES SELECTED FOR THE EXPERIMENTS

Cipher Suite	Auth	Key Exchange	Encryption	Digest
RSA-AES-128-SHA	RSA	RSA	AES-128-CBC	SHA1
RSA-AES-256-SHA	RSA	RSA	AES-256-CBC	SHA1
DHE-RSA-AES-128-SHA	RSA	DHE	AES-128-CBC	SHA1
DHE-RSA-AES-256-SHA	RSA	DHE	AES-256-CBC	SHA1

The measurement setup consists of a client that connects to the public server through a dedicated WLAN access point or 3G access network. The client program was developed using OpenSSL [8]. We measured energy by using the Nokia Energy Profiler (http://www.forum.nokia.com/energyprofiler). In order to focus on the energy consumed by TLS transaction, the client does not use a graphical user interface at all, only command line interface.The TLS handshake process involves operations that take execution times in order of magnitude of milliseconds. To guarantee accurate results in the experiments, many repetitions of the different processes and experiments have been done. This is also important due to the fact that the Nokia Energy Profiler has a maximum sampling rate of only 4Hz. Thus, in certain cases like using AES[128,256]-SHA1, the required number of repetitions can be up to 300.

IV. ANALYSIS OF THE HANDSHAKE ENERGY

We analyzed the energy consumption several different services. Table I shows the profiles for each of the studied services. The server implementation was obtained using SSLAudit (http://www.g-sec.lu/products.html). We focus first only on the handshake part of TLS.

A. Energy consumption using WLAN and 3G

Figures 3(c) and 3(d) show the energy consumption of the handshake phase only when using WLAN and 3G interfaces. We observe that the use of DH increases energy consumption a lot. The main reason is that DH is computationally much more intensive, but this fact also causes the transaction to last longer which increases the amount of energy consumed by the network interface.

The second thing we notice immediately is that a handshake performed over 3G access consumes in each case at least twice the amount of energy and in some cases up to four times the energy that is consumed over WLAN. The results in [5] report also such a very large difference for very short transfers between 3G and WLAN. However, contrary to that study, we did not include the 3G tail energy (since transfer phase starts right after handshake) which for very short transfers constitutes a vast majority of the energy consumed. For this reason, our results are surprising and

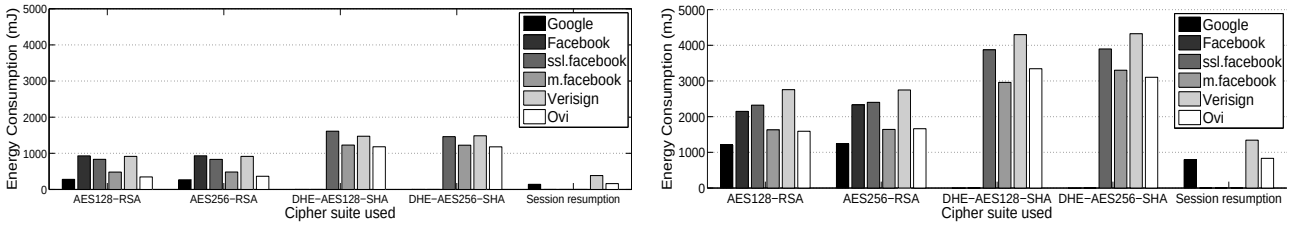
Perhaps the most interesting observation is the striking difference in energy consumption between the tested services. TLS handshake with Google and Ovi servers consume the least energy, while with Facebook and Verisign the energy consumption is more than doubled in case of RSA over WLAN and is clearly higher over 3G. These results and the large difference between WLAN and 3G access require more detailed investigation which we perform in the next section.

The use of session resumption greatly reduces the energy consumption by eliminating the expensive steps. Resumption was only supported by Google, VeriSign, and Ovi servers.

B. Stepwise Analysis

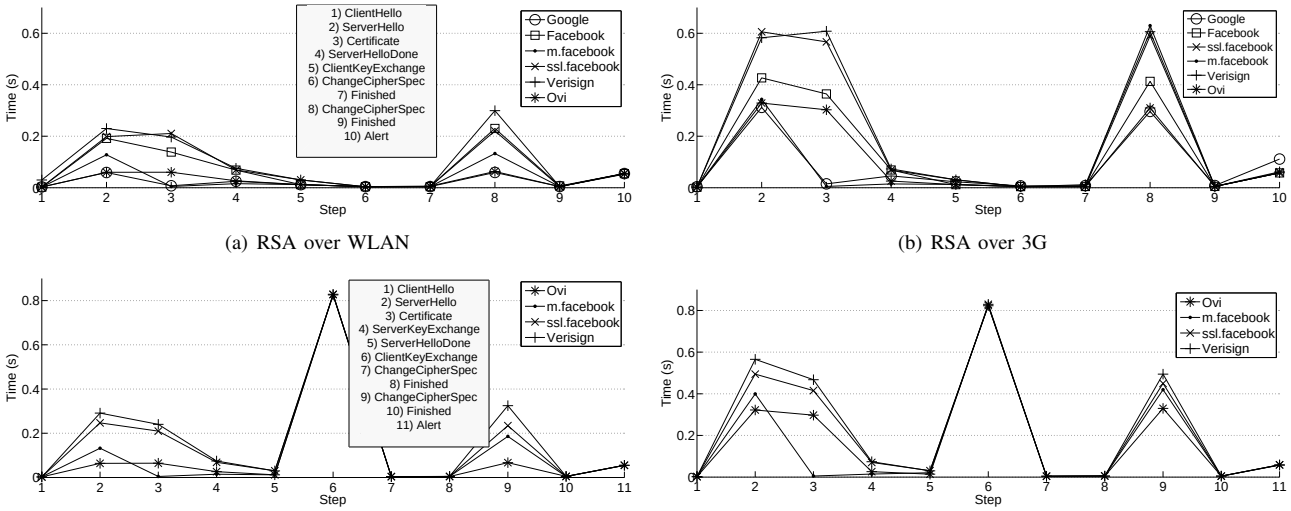
In order to understand which of the individual steps (see Figure 1) are responsible for the differences observed in the overall energy consumption, we analyze the execution time of each step. Figure 3 shows the results. The values are averages over 5 consequtive experiments.

We observe that the biggest differences in terms of execution time happen during the `ServerHello`, `Certificate`, and `ChangeCipherSpec` steps. `ServerHello` command sent by server in response to `ClientHello` includes various information such as the TLS version and the cipher suite to be used and random data for generating the secret-key. In `Certificate` step, the server sends its certificate including the public key. Optionally, it also includes the chain of certificates beginning with the Certificate Authority. In both steps, no cryptographic operations are involved by the client. Only after the certificate has been received, the client authenticates the server by checking the signature in the certificate, which is not that expensive operation with RSA [1]. Therefore, the delay differences are due to transmission using TCP over different RTTs. RTTs to Google and Ovi are especially short (< 50ms) while Verisign and Facebook servers seem to be located overseas in the USA. There is one RTT included in step 2 and, in addition, another RTT in step 3 for Verisign and Facebook due to large certificates (see Table I). Indeed, TCP applies slow start for increasing the congestion window size and even with initial window size of 2 packets, a 4KB certificate does not fit into these two packets together with the `ServerHello` message. Thus, the server needs to wait another RTT for ACKs to arrive from the client before transmitting the rest of the certificate data. Similarly a RTT is included in step



(a) WLAN (b) 3G

Fig. 2. Energy consumption per connection in different remote services using WLAN and 3G.



(a) RSA over WLAN (b) RSA over 3G

(c) DHE over WLAN (d) DHE over 3G

Fig. 3. Execution time of the different steps of a handshake.

8 although very little data is actually sent by the server. In addition, the duration of that step is slightly longer for some services compared to step 2 because the server decrypts client encrypted message using private key which is a rather heavy operation with RSA (compared to the public key operations performed by client). This operation can add a small extra delay especially if the server is loaded.

The above described delays have a major impact on the energy consumption since the network interfaces continue to consume energy even if no data is received. The impact is clearly larger for 3G because in the network interface stays in the highest power state (DCH) constantly: the operator set inactivity timers do not expire in between sending and receiving messages. WLAN interface is able to transition in between messages into idle or even sleep mode which consume significantly less energy [9]. Indeed, to give some reference numbers, a single message exchange without any computation (e.g. TCP handshake) consumes between 400-500mJ over 3G with 300ms RTT (power consumption between 1.3-1.5W). In comparison, the two cryptographic public key operations (server authentication after step 3 and encryption for step 5) that client needs to perform consume in the order of tens of mJ [1]. Some of the servers use 1024 bit and some 2048 public keys and the respective energy consumptions for cryptographic operations are roughly 10mJ and 50mJ [10]. So, in any case the numbers are rather negligible when compared to the communication overhead, which can also be seen by comparing the energy consumed by Facebook and SSL.Facebook.

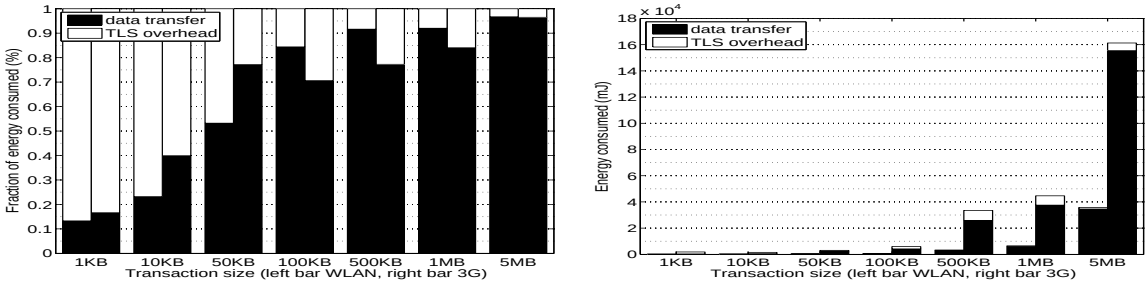
Note that client authentication was not used since none of the services allowed it. That procedure would require the client to generate a signature and send a certificate, thus consuming much more energy with RSA.

Using Ephemeral Diffie-Hellman as a key agreement protocol improves the security in the TLS handshake, but it comes with a cost in terms of delay and energy consumption. `ClientKeyExchange` is the computationally hardest step. This step factors the time it takes for the client to perform the necessary computations. These results also agree with those reported in [2]. Google and Facebook did not support the Diffie-Hellman protocol.

V. Total Energy Overhead of TLS

In order to measure the communication data overhead imposed by TLS, we setup a server locally and performed uploads[1] of varying file sizes with and without TLS. We

[1]Download results should be similar for WLAN because transmit and receive states consume roughly the same amount of power [9]. For 3G downloading is somewhat cheaper energy wise than uploading [5].



(a) Relative portion of overhead

(b) Total amount of energy consumed

Fig. 4. Measured energy overhead of TLS usage for WLAN and 3G.

used the RSA-AES128-SHA cipher suite with a 1024-bit server public key in all the experiments. RTT over WLAN was just a few milliseconds as the server and client were in the same local network and RTT over 3G was around 200ms. For the WLAN experiments, we imposed a rate limit of 200KB/s using Trickle[2] because we observed that without any artificial bandwidth limitiation, the results were biased by the limited CPU capacity of the mobile device: The throughput was slowed down by the rate at which it could perform per-packet cryptographic operations (encryption and hashing) when the transaction size exceeded 50KB, which caused the throughput achieved to be notably lower than the throughput achieved during a plain data transfer. We know from earlier work that the energy consumption of TCP transfer over WLAN is strongly dependendent on throughput [9] in such a way that the higher the throughput, the less energy is consumed per bit transferred. With a rate limiter, this problem was solved.

A. Measured Total Overhead

Figure 4(a) shows the measured overhead in relative values. As expected, the energy overhead decreases as the amount of data to transfer increases because the energy required by the record protocol, including transfer energy, amortizes the energy used for the handshake. We also note that when the transaction size increases the overhead decreases more rapidly when transmitting over WLAN than over 3G, which is mostly due to the relatively higher energy consumption of handshake over 3G. We should point out that while the measurements over WLAN were rather stable, the measurements over 3G were not. In fact, the results fluctuated quite a lot due to the much more unstable nature of the 3G communication channel (jitter and bandwidth variation), and the coefficient of variation for the results varied from 0.07 (5MB transaction) to as high as 0.65 (1KB transaction).

We plot the total energy consumed during the experiments in Figure 4(a) which reveals that 3G consumes several times more energy than WLAN. In fact, just the TLS energy overhead for 3G is as much as the energy consumed by the entire transaction over WLAN up to 1MB size.

[2]http://monkey.org/ marius/pages/?page=trickle

Our results give quite a different picture of the overhead compared to those reported in [1]. Indeed, this earlier work computes the TLS overhead to be as high as 55% in the case of 1MB transaction (over WLAN), while in our measurements this overhead is less than 10%. We believe that we are able to explain this difference as we dig deeper into the different parts that contribute to the overhead in the following section.

B. Computed Overhead Breakdown

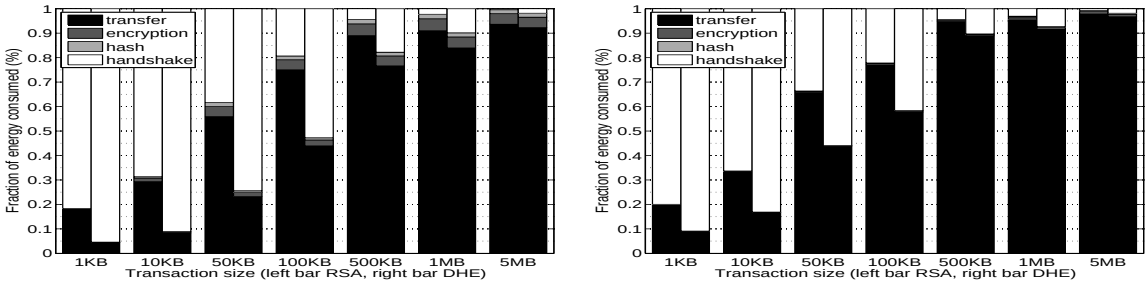
Based on detailed measurements on the mobile device, we computed also the share of each cryptographic operation involved in the TLS overhead. As in [1], we measured the energy consumption of the individual operations and the results are in Table III. We rounded the values to one decimal as the standard deviation over five consequtive experiments varied from 0.02 to 0.06. While the other results are in the same order of magnitude as reported in earlier work, the most important to note are the results from the plain I/O experiments. These results suggest that the symmetric key and hash operations are in fact very cheap energy wise: encrypting with AES-128 and hashing with SHA1 consume only $0.7\mu J/B$ and $0.1\mu J/B$, respectively, when removing the I/O part, which means that it is the reading and/or writing the source plain or cipher text file that consumes the most energy in these operations and not the execution of the actual crypto algorithm. It appears that this I/O overhead was not taken into account in the previous studies.

TABLE III

ENERGY CONSUMPTION OF INDIVIDUAL TLS OPERATIONS.

Operation	Energy ($\mu J/B$)
Encrypt with AES-256 (r/w phone memory)	2.0
Decrypt with AES-256 (r/w phone memory)	2.0
AES-256 vs. AES-128	0.3
Hash with SHA1 (read phone memory)	0.7
Read only (phone memory)	0.6
Read & write (phone memory)	1.3
Read only (memory card)	1.2
Read & write (memory card)	2.9

We computed the break down of the TLS energy consumption into the different components using the base data in the table and separately performed measurement values for the handshake with the local server. Measurement results from



(a) WLAN (b) 3G

Fig. 5. Break down of TLS energy consumption into cryptographic and non cryptographic components.

non-TLS experiments presented in the the previous section were used as the transfer energy[3]. Figures 5(a) and 5(b) show the results for the cases of WLAN and 3G, respectively. It is easy to notice that while encryption and hashing play a small part when communicating over WLAN, they are completely negligible in the 3G scenario. On the other hand, the handshake is an important factor to take into account but, as we showed earlier, this is mostly due to communication overhead during the handshake in the case of RSA but not entirely in the case of DHE. The computed overhead does not match perfectly with the measured one (Figure 4(a)) in the case of 3G due to varying measurement results, but in the WLAN case the results are very close to each other.

VI. Discussion

While HTTP/1.1 specification states that "a single-user client SHOULD NOT maintain more than 2 connections with any server or proxy" [11], browsers nowadays routinely break this part of the specification and establish many more connections [12]. This has a significant impact to a mobile client's energy overhead when using TLS especially when session resumption is not allowed by the server, which somewhat surprisingly in our experiments turned out to be the case with Facebook, for instance.

Other opportunities for optimization, especially for short transfers where the TLS handshake plays an important part, are reducing the size of the server certificate and avoiding the use of Diffie-Hellman key exchange instead of RSA unless absolutely necessary. Otherwise, reducing, for instance, the length of the server's public key has little impact.

Perhaps the most important take away is that, contrary to earlier studies, our results reveal that once the TLS transaction size exceeds 500KB, the overhead becomes much less important regardless of whether WLAn or 3G is used as the access network. In addition, the overhead of encryption and integrity protection by the record protocol is rather meaningless. Thus, optimizing the choice of symmetric crypto or hash algorithms makes little sense from the energy consumption perspective, except if the transmission rate is very high in which case more efficient crypto algorithms may prevent them to become a bottleneck and to limit the throughput and lower the energy efficiency.

[3]The true transfer energy is slightly higher because TLS record protocol causes some overhead also in the amount of data to transfer but we measured this overhead to be only about 3-4%.

VII. Conclusions

We presented in this paper a measurement study of the energy consumption of TLS protocol on a mobile device. Handshake energy consumption varies considerably between the measured services. Contrary to results reported in earlier studies, we observed also that the transmission and I/O energy, both in the TLS handshake and the record protocol, far exceeds the required computational energy by the actual cryptograhic algorithms and that with transactions larger than 500KB, the energy required to transmit the actual data clearly overweighs the TLS energy overhead.

References

[1] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha, "A study of the energy consumption characteristics of cryptographic algorithms and security protocols," *IEEE Transactions on Mobile Computing*, vol. 5, no. 2, pp. 128–143, 2006.

[2] Youngsang Shin, M. Gupta, and S. Myers, "A study of the performance of ssl on pdas," in *INFOCOM Workshops 2009, IEEE*, apr. 2009.

[3] George Apostolopoulos, Vinod Peris, and Debanjan Saha, "Transport layer security: How much does it really cost," in *In Proceedings of the IEEE INFOCOM*, 1999.

[4] Eric Rescorla, *SSL and TLS: Designing and building Secure Systems*, Addison-Wesley, 2001.

[5] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proceedings of IMC 2009.*, 2009.

[6] National Institute of Standards and Technology, *Specification for the ADVANCED ENCRYPTION STANDARD (AES)*, 2001, Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[7] P. Chown, "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)," RFC 3268 (Proposed Standard), 2002, Obsoleted by RFC 5246.

[8] *OpenSSL project*, 2010, Available at http://www.openssl.org.

[9] Yu Xiao, Petri Savolainen, Arto Karppanen, Matti Siekkinen, and Antti Ylä-Jääski, "Practical power modeling of data transmission over 802.11g for wireless applications," in *Proceedings of e-Energy 2010.*, 2010.

[10] A.S. Wander, N. Gura, H. Eberle, V. Gupta, and S.C. Shantz, "Energy analysis of public-key cryptography for wireless sensor networks," in *Proceedings of PerCom 2005.*, mar. 2005.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), June 1999, Updated by RFC 2817.

[12] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin, "An argument for increasing tcp's initial congestion window," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 3, 2010.

Anexo C. Memoria en Inglés

Las siguientes páginas contienen la memoria íntegra en Inglés con título *Energy consumption of cryptographic algorithms and security protocols in Symbian mobile devices*, presentada en la Aalto University School of Science and Technology situada en Helsinki, Finlandia.

Aalto University
School of Science and Technology
Faculty of Information and Natural Sciences
Degree programme of Computer Science and Engineering

Pedro Miranda

Energy consumption of cryptographic algorithms and security protocols in Symbian mobile devices

Final Project
Espoo, May 2010

Supervisor: Professor Antti Ylä-Jääski, Aalto University
Instructors: Matti Siekkinen, Aalto University
Heikki Waris, Nokia Corporation

Aalto University
School of Science and Technology
Faculty of Information and Natural Sciences
Degree Programme of Computer Science and Engineering

ABSTRACT OF
FINAL PROJECT

Author: Pedro Miranda

Title of final project:
Energy consumption of cryptographic algorithms and security protocols in Symbian mobile devices

Date: May 2010 **Pages:** 15 + 96
Professorship: Data Communications Software **Code:** T-110
Supervisor: Professor Antti Ylä-Jääski
Instructors: Matti Siekkinen, Heikki Waris

Mobile communications have redefined the way people communicate. Electronic devices, whereby these communications are established, have evolved significantly in terms of power, throughput and new functionalities, but they are still constrained by the battery life-time. This study focuses on this constraint and studies how it is impacted by the use of cryptography and security protocols, paying attention at the most popular transport-layer security protocol: the Secure Socket Layer (SSL). For the study, a experimental setup has been developed, by using a Nokia N95 Symbian mobile device, a PC running Linux, the Nokia Energy Profiler for the energy measurements and the OpenSSL implementation of the SSL protocol for the coding of the client and server test applications. Based on the results, the use of security generates an overhead involving a high impact in energy consumption for small transaction sizes, that decreases as the size is increased.

Keywords: AES, SHA, SSL, TLS, Diffie-Hellman, RSA, Symbian OS cryptographic algorithms, security protocols, OpenSSL energy analysis, battery-constrained devices
Language: English

This work was supported by TEKES as part of the Future Internet programme of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

Acknowledgements

First, I would like to thank my supervisor Professor Antti Ylä-Jääski for giving me the opportunity to work on this project and for his support.

Special thanks to my instructors Matti Siekkinen and Heikki Waris for their continuous guidance and support during my research and project development.

Also to Sampo Sovio and Jussi Ruutu from Nokia Corporation, for their supervision and guidelines all along these months.

To my parents Pedro and Carmen and my sister Pilar because, even in the distance, they were there when I needed them.

Last but not least, to all my friends and special people I have met during my almost two years in Helsinki. Thank you for the great moments.

In Espoo, May 2010

Pedro Miranda

Abbreviations and Acronyms

3G	3rd Generation
3GPP	3rd Generation Partnership Project
AES	Advanced Encryption Standard
API	Application Programming Interface
CFB	Cipher Feedback
CBC	Cipher Block Chaining
CSV	Comma Separated Values
CTR	Cipher Block Counter mode
DES	Data Encryption Standard
DSS	Digital Signature Standard
ECB	Electronic Code Book
DH	Diffie-Hellman
HMAC	Hash-based Message Authentication Code
HSPA	High-Speed Packet Access
HSDPA	High-Speed Downlink Packet Access
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
LAN	Local Area Network
MAC	Message Authentication Code
MD5	Message Digest 5
NEP	Nokia Energy Profiler
NSA	National Security Agency
NIST	National Institute of Standards and Technology

OFB	Output Feedback
PNG	Portable Network Graphics
RC4	Rivest Cipher 4
RSA	Rivest Shamir Adelman
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UEA2	UMTS Encryption Algorithm Suite 2
UIA2	UMTS Integrity Algorithm Suite 2
UMTS	Universal Mobile Telecommunication System
WLAN	Wireless Local Area Network

Contents

Abbreviations and Acronyms **v**

1 Introduction **1**

1.1 Motivation 1

1.2 Problem statement and objectives 2

1.3 Organization of this work 3

2 Security and energy background **4**

2.1 Computer security and security objectives 4

2.2 Security Terminology 6

2.3 Symmetric Cryptography 7

2.3.1 Block ciphers 7

2.3.2 Stream Ciphers 8

2.4 Message Digest and Message Authentication Codes 8

2.4.1 SHA 9

2.5 Public Key Cryptography 9

2.5.1 Key establishment protocol 10

2.5.2 Digital signatures 10

2.5.3 RSA 11

2.5.4 Diffie-Hellman 12

2.6 Energy Analysis 13

2.6.1 Energy Consumption of Network Interfaces 13

2.6.2 Energy Consumption of Cryptographic Algorithms . . 16

3 SSL and TLS **18**

3.1 SSL overview 18

3.2 The Secure Socket Layer protocol 18

3.3 The Record Protocol 19

3.4 The Handshake Protocol 20

3.4.1 Cipher suites 21

3.4.2 Handshake overview 21

3.4.3 A Real Handshake 22

3.4.4 Different SSL Handshakes 24

4 Designed Setup of Experiments **31**

4.1 Introduction and main goal 31

4.2 Hardware and Software used 32

4.2.1 Symbian 32

4.2.2 Nokia N95 32

4.2.3 Carbide.C++ IDE 34

4.2.4 Open C/C++ 34

4.2.5 OpenSSL 34

4.2.6 Nokia Energy Profiler 35

4.3 Cipher suites selection 36

4.4 Design of the different scenarios 38

4.5 Experimental and Measuring Setup 41

4.5.1 Design of the applications 41

4.5.2 Collecting measurements 44

4.5.3 Experimental Setup 47

5 Analysis of Measurements **49**

5.1 Test scenario: Local case 49

5.1.1 Exchanged data 49

5.1.2 Energy consumption 50

5.2 Test scenario: Remote cases 52

5.2.1 Services profile study 52
5.2.2 Exchanged data 56
5.2.3 Energy consumption using WLAN 56
5.2.4 Energy consumption using 3G 58
5.2.5 Handshake Steps Execution time using WLAN 59
5.2.6 Handshake Steps Execution time using 3G 64
5.2.7 Comparing WLAN and 3G 66
5.3 Test scenario: Overhead of SSL usage 69
5.3.1 SSL Data Overhead 69
5.3.2 SSL Throughput 71
5.3.3 SSL Energy Consumption Overhead 73

6 Conclusions **76**
6.1 Summary 76
6.2 Future work 77

Appendices **84**

A Study of Snow 3G **84**
A.1 SNOW3G 84
A.1.1 Experiments 85

B Study of Symmetric and Hash algorithms **89**
B.1 Summary and Experiments results 89
B.1.1 Read only experiment 92
B.1.2 Cryptographic operations using a large file 93
B.1.3 Cryptographic operations using a small file 93
B.1.4 Cryptographic operations using small files with the same key 94
B.1.5 Cryptographic operations using a large file with AES-192 & SHA1 94

B.1.6 Cryptographic operations using a large file with AES-128 & SHA1 95
B.1.7 Cryptographic operations using a large file in a memory card 95

List of Tables

2.1 Comparing symmetric and assymetric cryptography 10

3.1 Examples of cipher suites 21

4.1 Nokia N95 specifications 33
4.2 List of Cipher suites using AES included in TLS 37
4.3 List of Cipher suites selected for the experiments 38

5.1 Exchanged data performing a handshake for the local case . . 50
5.2 Energy consumption for the SSL handshake in the local case . 51
5.3 Full profile of the remote services analyzed 55
5.4 Exchanged data performing a handshake for remote cases . . . 57
5.5 Energy consumption for the remote services handshake using WLAN 59
5.6 Energy consumption for the remote services handshake using 3G 60
5.7 Execution times for each step involved in SSL handshake using RSA with WLAN 60
5.8 Execution times for each step involved in the SSL handshake using DHE with WLAN 62
5.9 Execution times for each step involved in SSL handshake using RSA with 3G 64
5.10 Execution times for each step involved in SSL handshake using DHE with 3G 65
5.11 Profile of the data exchanged between the client and server in an SSL connection 70

5.12 SSL Throughput comparing WLAN and 3G 71
5.13 Energy consumption using WLAN or 3G 73

6.1 Security equivalences (key length in bits) between symmetric, elliptic curve and public key cryptography 77

A.1 Execution time and energy consumption with FAST SNOW 3G disabled 86
A.2 Execution time and energy consumption with FAST SNOW 3G enabled 86
A.3 Execution time for the UIA2 loop 87

List of Figures

2.1 Relationship between confidentiality, integrity, availability and non repudiation 5
2.2 Relation between plaintext, ciphertext, encryption and decryption 6
2.3 Different states for 3G networks 14

3.1 SSL protocol structure 19
3.2 Normal RSA handshake 23
3.3 SSL Handshake using Ephemeral Diffie-Hellman (DHE) 26
3.4 SSL handshake with client authentication 28
3.5 SSL handshake resuming a previous session 30

4.1 N95 terminal 33
4.2 Open C/C++ in the Symbian architecture 35
4.3 Overview of the communication between the client, the server and the Nokia energy profiler 45
4.4 Export process in NEP and CSV exported data results 46
4.5 Secure client-server experimental setup 48

5.1 Energy consumption for the SSL handshake in Local case . . . 52
5.2 Energy consumption per connection in different remote services using WLAN 58
5.3 Energy consumption per connection in different remote services using 3G 59
5.4 Handshake steps Execution time using RSA with WLAN . . . 61

5.5 Handshake steps Execution time using DHE with WLAN . . . 63
5.6 Handshake steps Execution time using RSA with 3G 64
5.7 Handshake steps Execution time using DHE with 3G 66
5.8 Comparing Energy Consumption using WLAN and 3G 67
5.9 Comparing Handshake steps Execution time using RSA for WLAN and 3G 67
5.10 Comparing Handshake steps Execution time using DHE for WLAN and 3G 68
5.11 Percentage of communication data overhead of SSL usage for different transaction sizes 70
5.12 Comparing the throughput of using SSL and non secure connection 72
5.13 Energy overhead of SSL usage using WLAN 74
5.14 Energy overhead of SSL usage using 3G 74
5.15 Comparing energy consumption overhead using WLAN or 3G 75

A.1 Energy consumption and CPU usage of UEA2 and UIA2 with FAST_SNOW 3G Disabled 87
A.2 Energy consumption and CPU usage of UEA2 and UIA2 with FAST_SNOW 3G Enabled 88

B.1 Energy consumption of various algorithms with different parameters for a file of size 1.1 Mbytes stored in phone memory. 90
B.2 The result from encryption operation of a file read & stored at the memory location and operation performed in steps as read, encrypt and added write cost. 91

Chapter 1

Introduction

This chapter includes a motivation about the study of energy consumption of security protocols in battery-constrained devices as well as the objectives of this final project work. Finally, an explanation about the organization of this report is presented.

1.1 Motivation

In the last few years, mobile communications have redefined the way people communicate. Electronic devices, whereby these communications are established, have evolved significantly in terms of power, throughput, and above all in new functionalities, but they are still constrained by the battery life-time. This issue has been always an important issue in research, and although this life-time has evolved considerably, the increase has not followed the same rapid evolution as the device processing capabilities. While the processor performance doubles every 18 months according to *Moore's Law*, it is claimed that the battery capacity has only increased by 80% in the last 10 years [Fit07].

This improved performance and the arrival of Internet to these devices have made the communication through different channels possible, searching for information or performing purchases, all of these in a secure and confidential way.

Secure communications using wired or wireless networks are achieved by employing security protocols, which are based on cryptography algorithms. These algorithms are selected based on the security objectives needed in the security protocol to use. They include asymmetric and symmetric encryption

algorithms, which are used to provide authentication and secrecy, as well as message digest or hash algorithms, used to provide message integrity.

The use of security protocols not only affects connections throughput, but also represents a strong impact in the energy consumption in these battery powered devices. Thus, one of the foremost challenges is to achieve a balanced security performance/energy consumption in order to obtain a good throughput using the minimum amount of battery.

In this context, Nokia corporation, together with Aalto University School of Science and Technology (Helsinki, Finland), collaborate in a research project in order to address the energy consumption of different security protocols and cryptographic algorithms running on Symbian, the leader mobile devices platform.

1.2 Problem statement and objectives

The main objective of the study, is to perform an exhaustive analysis of the energy impact involved in the creation of a secure connection using the security protocol Secure Sockets Layer(SSL), used to establish secure communication between electronic devices. This protocol, uses different cryptographic algorithms in order to protect the information exchanged between different connected peers, providing not only data secrecy but also legitimacy and integrity. Furthermore, the use of this protocol provides the most common security services to secure network communications in such a way that the need for cryptography knowledge is minimized. Thus, the objective is to study the most used cryptosystems and analyze their consumption in different scenarios.

In order to complete all the needs, the process will begin with the development of a client application in a Symbian mobile device, capable of understanding and performing secure SSL connections with different online services, such as electronic email or social networks, and in that way, being able to measure the energy consumption dissipated in the establishment of that connection.

In addition, the development includes also a server capable of managing different secure connections with the mobile client, in order to model the different scenarios to study: Sending/receiving data, client authentication, resume a secure session previously used and the selection of the cryptographic suite to use.

For the establishment of the communications with the different services and the SSL server, WLAN and 3G interfaces will be used in order to compare also their different energy consumption and the impact due to the different communication nature.

1.3 Organization of this work

This final project work is divided into the following chapters:

- Chapter 1 - *Introduction* provides an overview to the problematic studied and the objectives of the project.
- Chapter 2 - *Security and Energy Background* includes an introduction to cryptography, the goals, and the different techniques used in the field, with an eye towards its use in SSL. It also contains an introduction to energy consumption involved in network interfaces and cryptographic protocols.
- Chapter 3 - *SSL and TLS* is a guide of the history of SSL/TLS and the security features that it provides. An entire SSL/TLS connection is detailed from start to finish.
- Chapter 4 - *Designed Setup of Experiments* with a description of all the used technologies to create the experimentation setup, the analysis of the requirement and the followed measuring methodology.
- Chapter 5 - *Analysis of Measurements* containing the different conclusions obtained for the different test scenarios.
- Chapter 6 - *Conclusions* includes the overall conclusions of the research, together with the future work.

At the end of the report, different appendices are included:

- Appendix A - *Study of Snow 3G* shows the result of an experimental study about the energy consumption and performance of the Snow3G algorithm.
- Appendix B - *Study of Symmetric and Hash algorithms* analyzes the energy consumption of symmetric algorithms like AES, and different uses of Hash algorithms and message digest like HMAC-SHA1 or the SHA2 family.

Chapter 2

Security and energy background

This chapter is intended to provide a basic approach to computer security and cryptography along with a little introduction to the most important network interfaces in mobile devices, WLAN and 3G. It starts with an introduction to what computer security is and the goals that are behind security. It continues giving a definition of the different algorithms and protocols involved in cryptography: symmetric encryption, digest algorithms, message authentication codes and public key cryptography, as well as the different most used solutions in each case. Finally, an introduction to energy consumption of the different network interfaces is included along with the energy impact of different cryptographic algorithms.

2.1 Computer security and security objectives

The term "security" is used in many ways in our daily lives. A "security system" protects our house from burglars and alerts the police, a "financial security" involves a set of investments that are adequately funded, or when "physical security" is mentioned in order to protect people from potential harm. As all these terms have a specific meaning in each context, so does the term computer security. The purpose of computer security is to devise ways to prevent the weaknesses from being exploited [PP06] while allowing the information to remain available and productive.

The words "security objectives" are usually associated to the security services or functionality required in a system or network in order to protect sensitive data or the identity of the different parties involved. There are 4 main

objectives including:

- ***Confidentiality*** ensures the data is kept secret from unintended listeners and can only be accessed by authorized parties. Usually these listeners are eavesdroppers with malicious intended purposes. Confidentiality is often called *privacy* or *secrecy*
- ***Availability*** means that the information is accessible to authorized parties at appropriate times. For this reason, availability is sometimes known by its opposite, *denial of service*.
- ***Integrity*** protects data from being modified by unauthorized parties and only in authorized ways.
- ***Nonrepudiation*** is the concept that proves the origin of the information. When nonrepudiation can be provided, the information is intended to be genuine.

Security in computing addresses these 4 goals. One of the challenges in building a secure system is finding the right balance among these goals, which often conflict. But balance is not all. In fact, these four characteristics can be independent, can overlap (as shown in Figure 2.1) and can even be mutually exclusive. For example, ensuring a strong confidentiality can affect severely the availability.

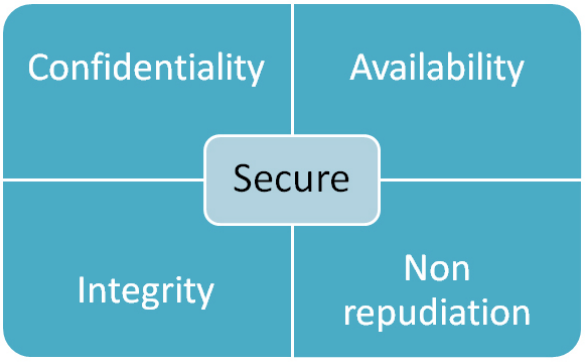


Figure 2.1: Relationship between confidentiality, integrity, availability and non repudiation

2.2 Security Terminology

The original form of a message is called plaintext, and the encrypted form is known as ciphertext. This relation is shown in Figure 2.2. A plaintext message P is denoted as a sequence of characters $P = \langle p_1, p_2, \ldots, p_n \rangle$. In the same way, the ciphertext is written as $C = \langle c_1, c_2, \ldots, c_m \rangle$. Encryption is the process of encoding a message so that its meaning is not obvious. Decryption is the reverse process, transforming and decrypting a previously encrypted message back into its original form. A system capable of encrypting and decrypting is called a *cryptosystem*.

Figure 2.2: Relation between plaintext, ciphertext, encryption and decryption

In order to describe the transformations between plaintext and ciphertext ,a formal notation like $C = E(P)$ and $P = D(C)$ can be used. C represents the ciphertext, E is the encryption rule, P is the plaintext, and D is the decryption rule. The main goal of a cryptosystem is being able to convert the message to protect it from an intruder, but being also able to get the original message back, so that the receiver can read it properly. Cryptosystems involve a set of rules in order to know the encryption and decryption method or rules. These rules, called algorithms, use a device called a key, denoted by K, so the resulting ciphertext depends on three factors: The original plaintext, the algorithm and the key used ($C = E(K, P)$).

Sometimes encryption and decryption keys are the same ($P = D(K, E(K, P))$). This method is called **Symmetric** encryption because encryption and decryption are the same processes. At other times, encryption and decryption are different operations. Then, a decryption key, K_D, inverts the encryption of key K_E so that the process can be represented like $P = D(K_D, E(K_E, P))$. Encryption in this form is called **Asymmetric** encryption because converting C back to P involves a series of steps and a key that are different from the steps and key of E. A better approach in this issue is done in Sections 2.3 and 2.5.

2.3 Symmetric Cryptography

As mentioned before, *Symmetric Cryptography* uses the same process to encrypt and decrypt, using the same secret-key as input. There are two major variants of symmetric encryption: Block ciphers and Stream ciphers.

2.3.1 Block ciphers

A block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of plaintext (unencrypted text) data into a block of ciphertext (encrypted text) data of the same length [Lab93]. The transformation function depends on an encryption key provided by the user, while the fixed length is called the block size, usually containing 64 bits.
In order to encrypt a message of an arbitrary length, different modes of operations for the block cipher can be used. These modes must be at least as secure and as efficient as the underlying cipher. The most used standard mode of operation is *Cipher Block Chaining* (CBC) but there also exist others such as *Electronic Code Book* (RCB), *Cipher Feedback* (CFB), *Output Feedback* (OFB) or *CTR mode* (CM) [Dwo01].
Block ciphers are a more conservative solution than stream ciphers because they are better studied. Examples of block ciphers are DES and AES.

AES

In 1997, NIST put out a call for submission for an *Advanced Encryption Standard* (AES) [oST01] in order to substitute its predecessor, the *Data Encryption Standard* (DES) [oST99]. The 5 final algorithms presented to NIST were MARS [IBM99], Serpent [ABK99], Twofish [Sch99], RC6 [RRSY99] and the winner, Rijndael [DR99].

AES is a strong and fast algorithm that uses as primary operations substitution, transportation, shift, exclusive OR, and additional operations. It supports 128, 192 and 256 bits key-length, using 10, 12 or 14 rounds respectively. Each round consists of four steps: Byte substitution, Shift row, Mix column and Add subkey. The security level depends on the number of rounds. Higher security is provided with 14 rounds. For more details about the implementation and how it works, please refer to [Sch95].

2.3.2 Stream Ciphers

Stream ciphers work with a function that generates a stream of data one byte at a time. The generated data is called *keystream*. This function has as input the encryption key, which controls exactly what keystream is generated. This keystream is usually XORed with a byte of plaintext to get a byte of ciphertext.

Stream ciphers can be highly secured if they are used properly, but they are vulnerable to attacks if certain precautions are not followed. The same key cannot be used twice and a valid encryption should never be used to provide authentication. Along with the previous precautions, the use of a strong MAC (see Section 2.4) to generate the encryption keys is mandatory.
Stream ciphers tend to be faster than block ciphers. One of the most used stream cipher is RC4.

RC4

RC4 [Kau99] was designed by Ron Rivest in 1994 and it is the most used stream cipher algorithm. RC4 is a variable-key-length cipher, with a key that can be anywhere between 8 and 2048 bits long. This key is expanded into an internal state table of constant size, so no matter the key length, the overall performance is really good. The normal use of the algorithm is with a 128 bits key length.

2.4 Message Digest and Message Authentication Codes

A *Message digest*(also known as Hash algorithms) is essentially a procedure that takes an arbitrary data length and generates as output a fixed-size checksum, called hash value. The same input will always produce the same output. Thus, an accidental or intentional change to the data will change the hash value.

Message digest functions have two important properties:

- *Irreversibility*: It should be practically infeasible to compute a message given its digest.

- *Collision-resistance*: Given digest D, it should be difficult to produce two messages M and M' such they have the same digest D.

The primary use of a message digest is the computation of a digital signature (see Section 2.5.2). The most used message digest algorithms in digital signature computing are Message Digest 5 (MD5) [Riv92] and Secure Hash Algorithm 1 (SHA-1)(see Subsection 2.4.1).

Apart from the use for digital signature, message digests are also utilized in Message authentication code (MAC), which are sort of like a digest algorithm, but they also incorporate a key into the computation. The creation of the MAC depends on both the key and the content of the message being MACed. The most used MAC algorithm is HMAC [KBC97], which describes how to create a MAC with security properties based on any digest algorithm that presents a reasonable set of cryptographic properties. The most used applications are HMAC-SHA-1 [KBC97] or HMAC-MD5 [CG97].

2.4.1 SHA

Secure Hash Algorithm is a set of cryptographic hash functions designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) as a Federal Information Processing Standard (FIST).
SHA-1 produces a 160 bits (20 bytes) message digest. SHA-1 has been examined closely since its release and it became a standard for SHA hash functions. It is the most used algorithm in several security applications and protocols. Nevertheless, in 2005, some security flaws were discovered[1], suggesting that a stronger algorithm might be needed in the future. The SHA-2(SHA-224, SHA-256, SHA-384, and SHA-512) family has been released with higher security, but as SHA-2 family is also similar to SHA-1, a new standard SHA-3 is being developed and will be released in 2012.

2.5 Public Key Cryptography

Public Key Cryptography involves the use of asymmetric algorithms instead of symmetric algorithms. These algorithms are used to create a pair of related keys: one secret private key and one published public key. A user can

[1] http://www.schneier.com/blog/archives/2005/02/cryptanalysis_o.html - Last accessed: April 2010

use the private key to encrypt a message, and the message can be revealed only with the corresponding public key.

Public key cryptography is computationally expensive. The strength of the algorithms depends on the size of their keys, which are usually very large numbers. As a result, operations involving public key operations are slow. See Table 2.1 for a comparison between Symmetric and Asymmetric cryptography.

	Secret Key(Symmetric)	**Public Key(Asymmetric)**
Number of keys	1	2
Protection of key	Must be kept secret	One key must be kept secret; the other can be freely exposed
Best uses	Secrecy and integrity	Key exchange, authentication
Key distribution	Must be out-of-band	Public key can be used to distribute other keys
Speed	Fast	Slow; 10.000 times slower than secret key

Table 2.1: Comparing symmetric and assymetric cryptography

In Sections 2.5.1 and 2.5.2, the different uses of public key cryptography are explained.

2.5.1 Key establishment protocol

Public key cryptography can be used to make both client and server work in order to establish the encryption key. There are two different ways of key establishment:

- *Key exchange* or *key transport*: where the client generates a symmetric key and encrypts it using the server's public key. RSA (see Section 2.5.3 for more details) can be used as a key transport algorithm
- *Key agreement*: both sides cooperate to generate a shared key. Diffie-Hellman (see Section 2.5.4 for more details) is a key agreement algorithm.

2.5.2 Digital signatures

A digital signature is a protocol that produces the same effect as a real signature: It is a mark that only the sender can make, but the receiver can

easily recognize as belonging to the sender. The signature is done using the sender's private key to *sign* a message and the receiver uses his/her public key to *verify* the signature. As the reader can realize, an important property provided by a digital signature is *nonrepudiation*, only the signer can generate the signature. Thus, the recipient can prove that the sender signed the message and the sender cannot deny it. The most used digital signature protocol is the RSA algorithm, as it provides a good performance.

2.5.3 RSA

RSA is the most used public key algorithm. It was invented in 1977 by Ron Rivest, Adi Shamir, and Len Adelman [RSA79].RSA relies on an area of mathematics known as number theory, in which mathematicians study properties of numbers such as their prime factors. The RSA encryption algorithm combines results from number theory with the degree of difficulty in determining the prime factors of a given number. The two keys used in RSA, d and e, are used for decryption and encryption. They are interchangeable: both can be chosen as the public key, but one must remain private. For simplicity, the encryption key will be e and the decryption key d.

RSA is based on the hardness of integer factorization. The cryptographic process is explained briefly here, but more detailed information can be found in [PP06]:

$$P = E(D(P)) = D(E(P))$$

> Any plaintext block P is encrypted as P^e mod n. Because the exponentiation is performed mod n, factoring P^e to uncover the encrypted plaintext is difficult. However, the decrypting key d is carefully chosen so that $(P^e)^d$ mod $n = P$. Thus, the legitimate receiver who knows d simply computes $(P^e)^d$ mod $n = P$ and recovers P without having to factor P^e.

Although several cryptographic experts have tried to find different flaws and cryptanalytic attacks on RSA, they have concluded that none of them is significant. Due to the fact that the factorization problem has been open for many years, most cryptographers consider this a solid basis for a secure cryptosystem.

RSA can be used as a key transport algorithm, to perform digital signatures and also for encryption.

2.5.4 Diffie-Hellman

Diffie-Hellman (DH) is the first public key algorithm ever published [WH76]. DH is a key agreement protocol rather than a key exchange algorithm. Instead of generating a key and encrypting it for the receiver, the sender and receiver collectively generate a key that is private to them. In order to compute the agreed key, the sender combines his/her private key with the receiver's public key. The recipient combines his/her private key with the sender's public key. A DH public key is often called a share, because it is one side's share of the key agreement.

Diffie-Hellman is based on the discrete logarithm problem in integer fields. The cryptographic process is explained briefly here, but more detailed information can be found in [WH76].

DH uses modular exponentiation (like RSA), but the modulus is a large prime p. The modulus has to be public and shared between the two communicating peers. There is also another number, the generator g, which also must be shared. g is chosen such that for any value $Z < p$ there exists a value W so that $g^w \bmod p = Z$. In order to generate a key,a random number X is needed. X is smaller than p and compute $Y = g^x \bmod p$. X is the private key and Y is the public key. For notation, Xs and Ys to refer to the sender's keys and Xr and Yr to refer to the receiver's keys.

To compute the shared key (ZZ), the sender computes:

$$ZZ = Yr^{Xs} \bmod p = (g^{Xr})^{Xs} \bmod p = g^{Xr \cdot Xs} modp$$

The recipient computes

$$ZZ = Ys^{Xr} \bmod p = (g^{Xs})^{Xr} \bmod p = g^{Xs \cdot Xr} modp$$

The most important operational difference with RSA is that in DH all the communicating parties must share g and p. However, it exists a different mode where the recipient randomly generates his/her own g and p and transmit them to the sender, often in his/her certificate. This mode is known as *ephemeral* DHE and provides *Perfect Forward Secrecy* [WVOW92], improving the security compared with RSA, that does not provide this feature.

2.6 Energy Analysis

Handheld devices like mobile phones or PDAs are highly constrained by the battery life-time. Components like the CPU, the display and its backlight, the camera and above all the network interfaces, represents a huge impact in the battery drain. With all these components, the long operational time capability of terminals is affected depending on the usage of them. In [Fit07] it is stated that in the last ten years the battery capacity has only increased by 80%, while the processor capabilities double every 18 months following Moore's Law. This improvement in the computational performance allow developers to design new applications with new functionalities that can have high energetical costs.

2.6.1 Energy Consumption of Network Interfaces

With the irruption of the Internet in mobile devices, users want to be *available* all the time. This tendency is known as *always being connected*, but from the battery standpoint it also produces an effect called as *always draining battery*. The network interfaces, such as GSM, 3G or WLAN, are responsible for the different communications used in a mobile device. In [Fit07] it is also cited that these interfaces account for the 50% of the power budget. Thus, any reduction in the use of these interfaces, will mean a substantial improvement in the overall power demand.

In this work, only 3G and WLAN interfaces are studied, as these ones are the most used for transferring data. In order to analyze the energy consumption of each one, firstly, is imperative to understand how they work.

3G interface

3G or third generation is a family of standards for mobile telecommunications that allows simultaneously voice and data services with up to 14.0 Mbit/s downlink and 5.8 Mbit/s uplink transfer rates.

In order to analyze the energy dissipated using the 3G interface, it is important to mention that there are two factors that determine the power consumption:

1. The *transmission energy*, responsible for the data transmission that is proportional to the length of this data and the transmit power level.

2. The *Radio Resource Control* (RRC), responsible for channel allocation and scaling the power consumed by the radio using inactivity timers (T1 and T2). This channel allocation or state varies depending on the network activity:

   - *IDLE* state is used in absence of network activity.
   - *Dedicated Channel (DCH)* state ensures the highest throughput with low delay transmission, but at the cost of a high energy consumption.
   - *Forward Access Channel (FACH)* state shares the channel with other 3G devices and is used when there is little traffic to transmit, saving about 50% of the energy consumed in DCH state.

Figure 2.3 shows these 3 different states for 3G networks and the transitions between them.

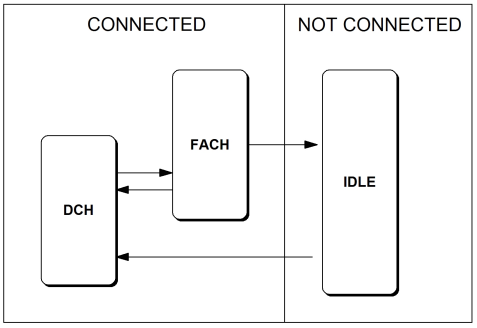


Figure 2.3: Different states for 3G networks

The energy consumption in a 3G interface can be quantified attending to these 4 factors:

1. *Ramp energy* required to switch to the DCH or high-power state.
2. *Transmission energy* dissipated in the data transmission.
3. *Tail energy* spent in maintaining the high-power state after the completion of the data transmission.
4. *Maintenance energy* used to keep the 3G interface up.

WLAN interface

WLAN interface or Wireless Local Area Network interface is used to link devices without using cables, providing in most cases a direct connection through the Internet. These devices have become popular due to the increasing popularity of smartphones and laptop wireless capabilities. The actual standard IEEE 802.11 defines operations in 2.4, 3.6 and 5 GHz frequency bands. There are different protocols, but the most extended one is the 802.11g, working in 2.4GHz, that operates at a maximum bit rate of 54 Mbit/s, with an overall 22Mbit/s throughput.

The energy consumption in this WLAN interface can be quantify attending to these 3 factors:

1. *Scanning and association* the energy dissipated during the searching for an access point and connecting to one of them.
2. *Transferring the data* the energy required in the data transmission.
3. *Maintenance energy* the energy used to keep the WLAN interface up.

3G vs WLAN Energy Consumption

In [BBV09] it is studied the energy consumption of these WLAN and 3G interfaces in a Nokia N95 symbian device (see especifications in Section 4.2.2, concluding different assertions:

- 3G consumes significantly more energy to download data than WLAN. Apart from the communication nature involved in 3G, the lower bandwidth, compared with WLAN, requires the interface to be kept up for a longer time.
- The association with an access point in WLAN needs a high initial energy consumption, that is comparable to the tail energy of 3G.
- The use of *Power Saving Mode (PSM)* in WLAN greatly reduce the energy consumption when the interface is not used.
- In 3G, nearly 60% of the energy is tail energy, wasted after the completion of a sucessful transmission. Compared to that, the ramp energy is only a small quantity, and can be amortized over frequent successive transfers, within the tail time timer.

- The transfer energy is proportional to the size of the data transferred and the transmit power level. This transmission energy using WLAN is substantially smaller than for 3G, specially for larger transfer sizes. With increasing data sizes, the efficiency increases dramatically. On the other hand, for smaller transactions, WLAN becomes inefficient compared to 3G.
- WLAN is more energy efficient than 3G once it is associated to an access point.
- The maintenance energy to keep the 3G interface up is about 1-2 Joules per minute while for the WLAN is about 3-3.5 Joules per minute.

All these conclusions will serve to explain better the results obtained in Section 5 where the 3G and WLAN energy consumptions are compared for the studied cases.

2.6.2 Energy Consumption of Cryptographic Algorithms

As commented before, the arrival of the Internet to handheld devices has made mobile communications possible, allowing users to search for information, buy things or share information in a secure and confidential way. This privacy is achieved thanks to the use of cryptographic algorithms and security protocols, which have an important impact in the battery life-time, increasing the number of calculations in the CPU and also augmenting the number of exchanged messages and their length.

While previous works [PRRJ06], [SGM09], [APS99], analyze the energy consumption of different cryptographic algorithms as well as the performance of SSL and TLS (explained in Section 3) on PDAs and in computers, they do not perform a complete study analyzing both energy consumption and performance. In this work, both are studied, addressing the high energy consumption steps with time execution increases.

These previous studies show different observations, in addition to different suggested improvements. Here just a few of them are mentioned:

- Public key cryptography has the highest energy consumption, while hash algorithms have a little impact in the battery life-time.
- The key size chosen affects dramatically to energy in public key cryptography, but not in such a way for symmetric algorithms.

- The use of Diffie-Hellman as a key agreement protocol during the handshake stage in SSL improves security, but it also increases dramatically the energy consumption and decreases the performance, compared with other key exchange protocol like RSA. This higher energy consumption is due to the extra CPU computations needed, in addition to the necessity to maintain the network interface up for a longer time.

- The use of SSL as a security protocol affects significantly the overall performance of the connection, but this can be gently eased choosing the most suitable algorithms depending on the security requirements of the user and the hardware capabilities of the device.

All of these observations will be confirmed in Chapter 5 and will also help to understand better the obtained results.

Chapter 3

SSL and TLS

This chapter is intended to introduce the reader with a brief introduction to SSL and TLS. It also defines the different protocols involved and also how a connection is established, including the different SSL handshakes.

3.1 SSL overview

Secure Socket Layer is a protocol that provides a secure channel between two machines. The protocol was designed in order to protect data in transit and to identify the machines involved in the communication. Data is encrypted between the peers, but the data that one side writes is exactly what the other side reads. The secure channel also offers transparency, which means that it passes the data through unchanged. This property, allows to any protocol that runs over TCP to be run over SSL with minimal modifications.
Since the first version of SSL, originally designed by Netscape in 1994, it has gone through a number of revisions beginning with version 1 (unreleased), going through SSLv2 [Cor94], SSLv3 [Cor96] and culminating in its adoption by the IETF as the Transport Layer Security (TLS) standard [DA99]. Once SSL and the different revisions have been introduced, from now on, all the references to SSL will refer to TLS as well.

3.2 The Secure Socket Layer protocol

The primary goal of the SSL Protocol is to provide privacy and data integrity between two communicating applications in such a way that the need for

cryptography expertise is minimized.

In order to provide a transparent and secure communication channel, the SSL protocol is placed in the protocol stack, just below the application layer and just above TCP layer (see Figure 3.1 for more details).

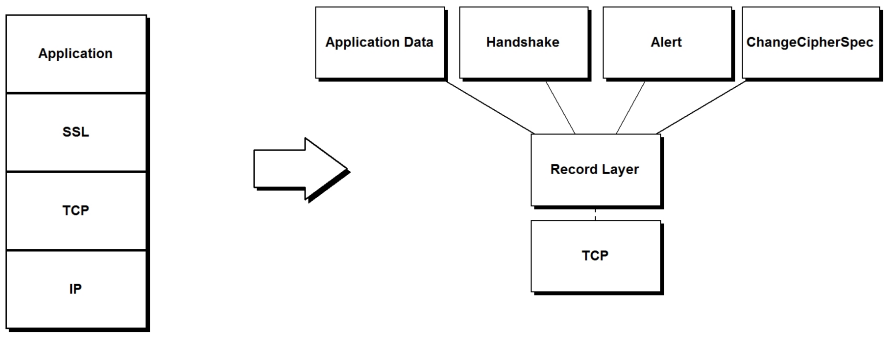


Figure 3.1: SSL protocol structure

The SSL protocol is divided in two layers:

1. *The Record protocol* provides secrecy and reliability of the transfered data. The *Record Protocol* is responsible for data transfer.

2. *The Handshake protocol* allows the client and the server to authenticate each other and to negotiate the cryptographic algorithms in addition of encryption keys needed in order to secure the channel.

Both protocols are described in detail in Sections 3.3 and 3.4.

3.3 The Record Protocol

As described before, the *Record Protocol* is responsible for the data transfer and the interpretation of the type of the received message. The protocol works by breaking up the data stream to be transmitted into a series of fragments, each of which is independently protected and transmitted. This behavior allows data to be transferred as soon as it is ready.
Before any fragment can be transmitted, integrity protection has to be provided. A MAC is computed over the data and is then appended to the data fragment. The concatenated data is encrypted and a record header is attached and the whole packet is sent to the receiving peer. The job of the record header is to provide information that is necessary for the receiving

implementation to interpret the record. In the header the following elements are included [Res01]:

- Message length: allows the receiver to know how many bytes to read from the wire before it can process the message.
- SSL version: to ensure each side agrees on the same version.
- Content type: SSL supports 4 message content types including:
    1. Application Data: all data sent and received by applications that use SSL is sent as this content type.
    2. Alert: primarily used for signalling various types of errors. Most of them alert that something went wrong in the handshake, but some indicate errors related with decryption or authenticating records. Other uses are to signal that the connection is about to close.
    3. Handshake messages: used to carry the different messages that are presented in the handshake protocol.
    4. ChangeCipherSpec: this message has the special purpose to indicate a change in the encryption and authentication of records. This message is sent to indicate that all the data transferred from now on will be encrypted using the agreed cryptographic algorithms.

While the *Record Protocol* is responsible for the data transfer, the *Handshake Protocol* is in charge of establishing SSL session states between communicating peers. In next section 3.4, the different purposes of the *Handshake Protocol* as well as the different operations modes are explained.

3.4 The Handshake Protocol

The purpose of the *Handshake protocol* is threefold:

1. Client and server need to negotiate a set of cryptographic algorithms which will be used to protect the data.
2. They need to agree on the cryptographic keys to be used by those algorithms.
3. The handshake may optionally request authentication from the client side.

In order to allow the users to select the level of security that suits best their needs, SSL packs the different cryptographic algorithms into cipher suites, or sets of ciphers. This protocol is responsible of the agreement between the client and the server on these cipher suites.

3.4.1 Cipher suites

There are many different cryptographic algorithms which can be used for encrypting data, for computing the MAC or to authenticate the peers. Some of them provide the highest levels of security, but they have higher computational requirements. Others are less secure, resulting in a higher performance encrypting and decrypting.

Each cipher suite specifies the server authentication algorithm, the key exchange algorithm, the bulk encryption algorithm and the message digest algorithm. In Table 3.1, two different examples of cipher suites are described.

Cipher Suite	Auth	Key Exchange	Encryption	Digest
RSA_AES_256_SHA	RSA	RSA	AES-256-CBC	SHA1
DHE_RSA_AES_256_SHA	RSA	DHE	AES-256-CBC	SHA1

Table 3.1: Examples of cipher suites

When an SSL connection is established, the client and the server exchange information about which ciphers suites they are willing to use. The client sends the cipher suites listed in the order of descending client preference, usually sorted by highest level of security. The server chooses the most suitable shared cipher suite. If they don't have a cipher suite in common, the SSL communication is not possible and the server closes the connection.

3.4.2 Handshake overview

An overview of the handshake process is explained here:

1. The client sends to the server a list of the algorithms (bundled in cipher suites) that it is willing to support, along with a random number in order to generate the symmetric encryption key.

2. The server chooses a cipher suite out of the list and sends it back to the client along with a certificate, which contains the server's public key.

The certificate also provides the needed information to authenticate the server. This also sends a random number that will be part of the encryption key generation process.

3. The client verifies the server's certificate and extracts the public key. Once the client has the public key from the server, it generates a random secret string called the *pre_ master_ secret* and encrypts it using the server's public key. It sends the encrypted public key to the server.

4. Both sides, client and server, independently compute the encryption and the MAC keys from the *pre_ master_ secret* and both random values from client and server.

5. The client sends a MAC of all the previous handshake messages sent to the server.

6. The server sends a MAC of all the previous handshake messages sent to the client.

These six steps are just an overview of how an SSL handshake is done. Section 3.4.3 shows how a real handshake is done and the messages involved.

3.4.3 A Real Handshake

In order to explain how a real handshake is done, a brief introduction to the messages exchanged between the client and the server is needed.
The most used and common SSL handshake is using RSA as a key exchange protocol and only authenticating the server, but not the client. Figure 3.2 shows the exchanged messages.

The sequence of actions that occur when messages are exchanged during an SSL handshake are summarized here:

1. The connection starts with the client sending a `ClientHello` command, that contains:

   - The highest SSL version supported by the client.
   - The cipher suites supported by the client.
   - Compression methods if available.
   - Session ID.
   - Random data for its use in the secret-key generation process.

2. The server answers with a `ServerHello` command, which includes:

   - The SSL version that will be used in the connection.
   - The cipher suite chosen by the server among those offered by the client.
   - The compression method chosen by the server.
   - The session ID for the SSL connection.
   - Random data for its use in the secret-key generation process.

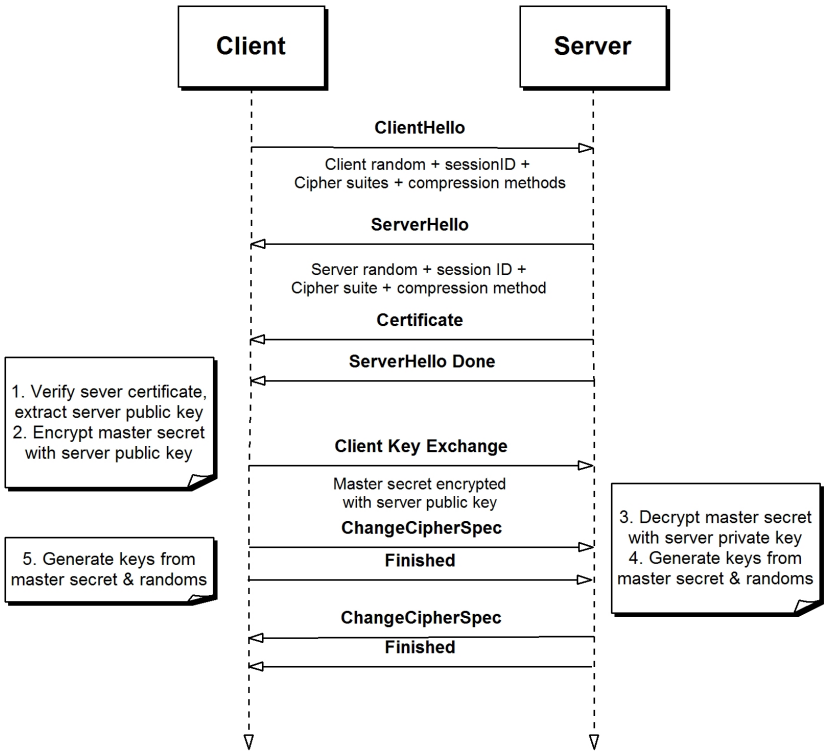


Figure 3.2: Normal RSA handshake

3. The server sends the `Certificate` command, that includes the server's certificate with the server's public key. Optionally, it also includes the chain of certificates beginning with the Certificate Authority (CA).

4. The server sends the `ServerHelloDone` command, indicating the client that the server is done with this stage of the SSL handshake.

5. The client sends the `ClientKeyExchange` message, that contains the *pre_ master_ secret* created by the client and encrypted using the server's public key. Both sides, the client and the server, generate the secret encryption key using the *pre_ master_ secret* and the random data included in the `ClientHello` and `ServerHello` messages.

6. The client sends the `ChangeCipherSpec` command, which indicates that the subsequent messages sent by the client during the SSL session will be encrypted using the agreed cryptographic systems.

7. The client sends its last message in the Handshake protocol, `Finished`, containing a digest of all the previous messages sent by the client to the server. This command is sent to ensure that none of the messages sent previously unencrypted were not altered in flight.

8. The server sends the `ChangeCipherSpec` command, which indicates that the subsequent messages sent by the client during the SSL session will be encrypted using the agreed cryptographic systems.

9. The server sends its last message in the Handshake protocol, `Finished`, containing a digest of all the previous messages sent by the server to the client. This command has the same purpose as the `Finished` command in the client.

These previous nine actions shows the exact steps that occurs in a SSL handshake using RSA for both key exchange and authentication protocol. The reader can find a more detailed explanation of the different types of handshake using different key exchange protocols, authenticating the client or the reuse of a previous session in Section 3.4.4.

3.4.4 Different SSL Handshakes

This section is intended to explain in more details how the different SSL handshakes, using different key exchange protocols are done and which kind of operations are involved, paying attention to the most energetically-computationally expensive.

Normal RSA Handshake

In Section 3.4.3, the normal RSA handshake was already explained. The use of RSA as a signing and key exchange protocol is the most used all over the

world. It provides good performance with competitive energy consumption costs.

In general, RSA operations are divided in:

1. Private key operations used in digital signature and private key decryption.
2. Public key operations used in signature verification and public key encryption.

This distinction is important in this context because private key operations are much slower than public key operations. Decryption of a ciphertext using public key operations is really hard, computationally speaking.

The vast majority of the CPU time is consumed processing three messages, two on the client side and one in the server side.

- Client side
  - Server's `Certificate` processing: The certificate sent by the server contains the chain of certificates. In order to process it, the client needs to parse and verify each certificate. This means performing RSA verifies, one verify per each certificate included in the chain.
  - `ClientKeyExchange`: This message contains the *pre_master_secret* encrypted under the server's public key. The dominant operation here is RSA encryption, where the limiting factor is the length of the server's public key.

- Server side
  - `ClientKeyExchange`: The dominant operation here is the server's RSA decryption of the *pre_master_secret*. As with the previous operation, the longer the key, the longer the decryption takes. It is important to mention that this operation is much slower than the encryption (it uses a private key operation) and the verification that take place in the client. As this operation takes place in the server, it is the bottleneck in SSL throughput. The client needs to wait for the server to complete this step. The cost of the decryption depends on the server's key length, making the server's RSA key length the most important factor limiting performance.

Ephemeral Diffie-Hellman Handshake

Although RSA is by far the dominant public key algorithm, SSL supports a number of cipher suites based on other algorithms like Diffie-Hellman (DH). The idea of the use of these algorithms was to avoid patented algorithms, in particular RSA. But since the year 2000, when the RSA patent expired, this motivation is no longer compelling, but support for DH is still available.

Unlike RSA, which can be used for either key exchange or signature, DH can only be used for key agreement. As a consequence, in order to get a complete solution, Diffie-Hellman and RSA (or other protocol like DSS) have to be used together. The most common way to use DH is using ephemeral DH keys. The server generates a temporary DHE key and sings it with its RSA key, transmitting the signed key in the `ServerKeyExchange` message. In that case, the client will use that DH key for the key agreement. It is also possible to have a long-term DH key. In this case, the server will have a signed certificate containing the DHE key.

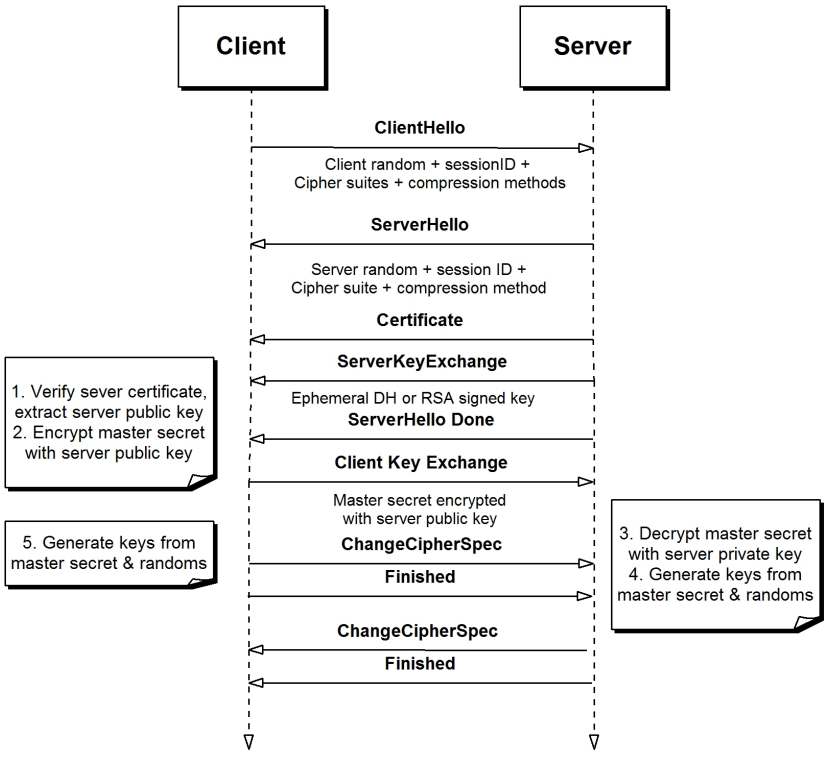


Figure 3.3: SSL Handshake using Ephemeral Diffie-Hellman (DHE)

The key agreement process is the same, no matter how the client gets the

server's DHE key. As the client also needs a DHE key, and normally it will not have one, it will generate one and then it will transmit the key in the `ClientKeyExchange` message. At this point, the client and the server each have their own private key and the other peer's public key, so they can independently calculate the DHE shared secret, used as *pre_master_secret*. From that point on, the rest of the connection proceeds exactly as in RSA. Figure 3.3 shows the Handshake process using Ephemeral Diffie-Hellman.

The use of Ephemeral Diffie-Hellman as a key agreement protocol involves the addition of the following operations:

- Client side:
  - Certificate: The `certificate` processing is the same as in RSA.
  - `ServerKeyExchange`: As with the `Certificate` message, this involves the client processing of the `ServerKeyExchange`.
  - `ClientKeyExchange`: This is the critical step in DHE handhshake. It involves the generation of the client's ephemeral DH key and the computation of the DH shared secret.
- Server side:
  - `ServerKeyExchange`: This message does not exist in RSA mode. It involves the signing of the ServerKeyExchange using the server's private key, that represents an expensive operation.
  - `ClientKeyExchange`: The computation of the shared secret is involved here, using server's private key and the client's public key.

The use of Diffie-Hellman as a protocol for the key agreement only adds one message from the server side, but it also adds different asymmetric operations that decrease the performance and increase the energy consumption.

Client Authentication Handshake

Although in most situations only the authentication of the server is required, it is also possible to authenticate the client if the server requests it. This feature was introduced in SSLv3 and TLS in order to provide an option for client authentication. This is useful if a server wants to restrict access to some services to only authorized clients. The idea is that the client uses its own private key to sign a message, thus proving that it has a private key corresponding with the certificate.

Client authentication is always initiated by the server. There is no mechanism for the client to force the client authentication. This authentication request adds a number of messages and operations explained in the following lines.

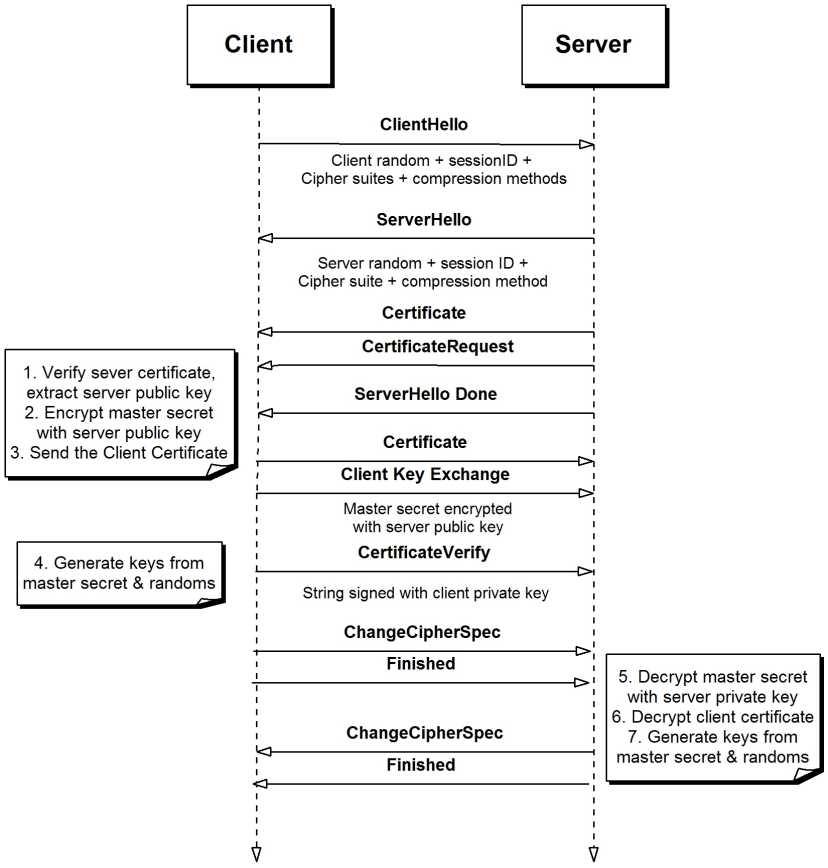


Figure 3.4: SSL handshake with client authentication

RSA with client authentication mode is similar to normal RSA handshake but with some extra steps. Figure 3.4 shows the whole process.

The client authentication is initiated by the server sending a `CertificateRequest` message to the client. This one, answers the server with a `Certificate` (containing the client's certificate) and a `CertificateVerify` (a string signed with the private key associated to the certificate) messages.

Client authentication also involves additional operations. In order to explain it clearly, the same previous operations performed in the server in the RSA normal mode have to be performed also in the client. As a consequence, client authentication mode significantly increases the load on the client but the server is not affected in such a way.

The same operations as mentioned in normal RSA mode also occur for this method, but with the addition of 3 new steps:

- Client side
  - `CertificateVerify`: this message consists of an RSA-signed message. Same comments described for the `ClientKeyExchange` in the normal RSA mode also apply here. The expensive stage here is the RSA signature.
- Server side
  - `Certificate`: this message is processed exactly as the client processes the server's certificate from the server. The dominant cost here is the verification of the digital signatures.
  - `CertificateVerify`: it mainly consists on checking the client's digital signature, one public cryptography operation.

DHE Client authentication

This mode is roughly similar to the RSA client authentication, but with the addition of two new messages from the client side: the client's `Certificate` and the `CertificateVerify`. These new messages entail more operations. On the client side, the most important addition is the RSA signature on the `CertificateVerify`. On the server side, the majority of time is involved in processing the client's certificate using RSA.

Session Resumption Handshake

The most expensive part of the SSL handshake is the establishment of the *pre_master_secret*, which requires public key cryptography. The use of session resumption involves the reuse of a previous established session between the client and the server. This means that there is no need to negotiate again about the *master_secret*, because the session is already cached by the two peers. This avoids the computationally expensive operations required by public key operations, saving time and reducing dramatically the energy consumption.

Figure 3.5 shows the timeline of this process.

The first time the client tries to interact with a server both create a new connection and a new session. If the server is prepared to resume the session,

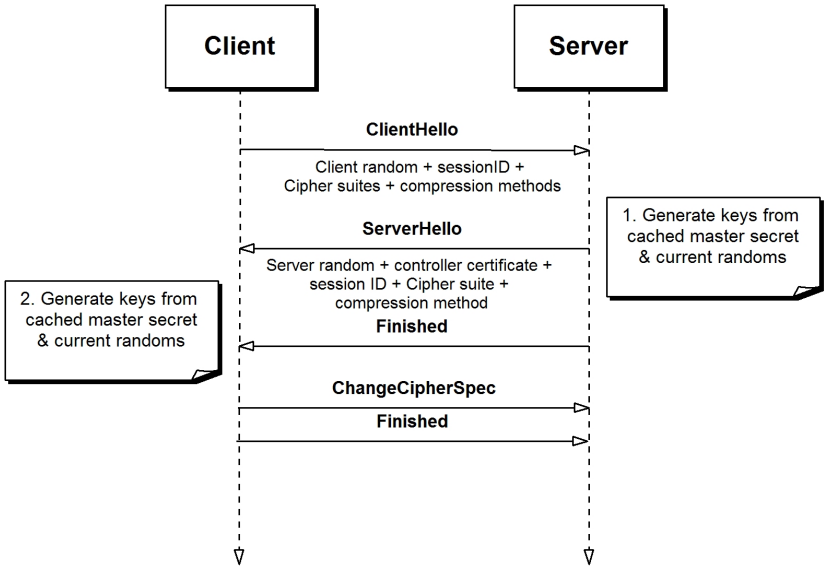


Figure 3.5: SSL handshake resuming a previous session

it sends the previous *session_id* in the `ServerHello` message and caches the *master_secret*. When the client wants to resume a previous session, it just uses the session_id of a previous session and sends it to the server in the `ClientHello`. The server agrees to reuse the same session. At this point, the rest of the handshake process is ignored and both sides use the *master_secret* to generate all the cryptographic keys.

Chapter 4

Designed Setup of Experiments

The main goal of this chapter is not only to document the different algorithms and cipher suites used and try to justify the selection with arguments and facts, but also to discuss the different experiments/measurements that will be used to perform a full study of the impact of creating a SSL secure channel. It also includes the experimentation setup, giving a little definition and the most important aspects of each used technology. In addition, the design aspects and requirements, along with the most important decisions taken at the coding process, are explained.

4.1 Introduction and main goal

The main goal of this research project is to study and understand the energy costs of security in SSL transactions. The mean to perform this study is to create an experimental setup that helps to measure the energy consumption involved in the establishment of a client-server secure channel using the Secure Socket Layer protocol.

This experimental setup is carried out with the implementation of a mobile client and a server that answers the communication requests from the client side. In order to create a secure communication channel, both server and client need to understand the SSL protocol.

4.2 Hardware and Software used

In the next sections, a brief introduction to all the technologies and components used is included.

4.2.1 Symbian

Symbian[1] is the world's most used and popular smartphone platform. It is implemented in a diverse range of devices and provides application and media developers with a consistent set of technologies.
The Symbian Operating System is a 32-bit multi-tasking OS. Symbian platform provides most of the functionalities via subsystems. Each subsystem has a set of libraries that implements method and functions accessible using different APIs.
Developers can create applications using different languages such as Qt, Symbian C++, a set of Open C/C++ APIs, Java, Web Runtime (WRT) or Adoble Flash Lite.

4.2.2 Nokia N95

The Nokia N95[2] terminal was released in 2007 by Nokia under the N series. It is based on Symbian 9.2 , the third generation of the Symbian OS and It is the first Nokia phone with HSDPA connectivity, the evolution of the 3G (third generation) mobile telephony communications protocol in the HSPA family, also coined as 3.5G.

The most important specifications of the N95 terminal are presented in Table 4.1.

[1] http://www.symbian.org - Last accessed: April 2010
[2] http://europe.nokia.com/support/product-support/nokia-n95 - Last accessed: April 2010

Figure 4.1: N95 terminal

GENERAL	2G Network	GSM 850 / 900 / 1800 / 1900
	3G Network	HSDPA 2100
		HSDPA 850 / 1900 - American version
	Announced	2006, September. Released 2007, March
DISPLAY	Type	TFT, 16M colors
	Size	240 x 320 pixels, 2.6 inches, 40 x 53 mm
DATA	GPRS	Class 10 (4+1/3+2 slots), 32 - 48 kbps
	EDGE	Class 32, 296 kbps; DTM Class 11, 177 kbps
	3G	HSDPA
	WLAN	Wi-Fi 802.11 b/g, UPnP technology
	USB	Yes, v2.0 miniUSB
FEATURES	OS	Symbian OS 9.2, S60 rel. 3.1
	CPU	Dual ARM 11 332 MHz processor; 3D Graphics HW Accelerator
BATTERY	Type	Standard battery, Li-Ion 950 mAh (BL-5F)
	Stand-by	Up to 220 h (2G) / 192 h (3G)
	Talk time	Up to 6 h 30 min (2G) / 2 h 42 min (3G)

Table 4.1: Nokia N95 specifications

4.2.3 Carbide.C++ IDE

Carbide.C++[3] is a software application for development in Symbian OS. It is used to build applications and other components that use Symbian OS. It is based on the Eclipse Integrated Development Environment (IDE), enhanced with extra plug-ins to support the Symbian OS development. It provides the needed tools to develop C++ applications but using additional components, it also includes the possibility to develop applications in other coding languages like C, Python, Java, among others. Carbide.C++ 2.0 has been used for the development of the applications.

4.2.4 Open C/C++

Open C/C++ plug-in[4] is a set of libraries that provides the developer with a serie of C/C++ programming interface libraries, in order to enable a faster application development in Symbian OS. It includes nine well-known standard POSIX and middleware C libraries: libc, libdl, libpthread, libm, libz, libcrypt, libglib, libcrypto and libssl. These two last ones are part of the OpenSSL implementation (see Section 4.2.5 for more information).

Figure 4.2 shows where the Open C/C++ standard libraries fit in the Symbian architecture.

4.2.5 OpenSSL

OpenSSL [Ope10], is an open source toolkit implementing the Secure Socket Layer protocol, the Transport Secure Layer protocol, and a full-strength general purpose cryptography library. OpenSSL provides an API that facilitates the developing of SSL applications. At the time of writing, the current version 1.0.0 was released on Mar 29th, 2010.

Although there are different alternatives like GNUTLS[5], Network Security Services (NSS)[6] or PolarSSL[7], OpenSSL was selected attending to these reasons:

[3] http://developer.symbian.org/main/create_applications/index.php - Last accessed: April 2010

[4] http://www.forum.nokia.com/Technology_Topics/Development_Platforms/Open_C_and_C++/ - Last accessed: April 2010

[5] http://www.gnu.org/software/gnutls/ - Last accessed: April 2010

[6] http://www.mozilla.org/projects/security/pki/nss/ - Last accessed: April 2010

[7] http://www.polarssl.org - Last accessed: April 2010

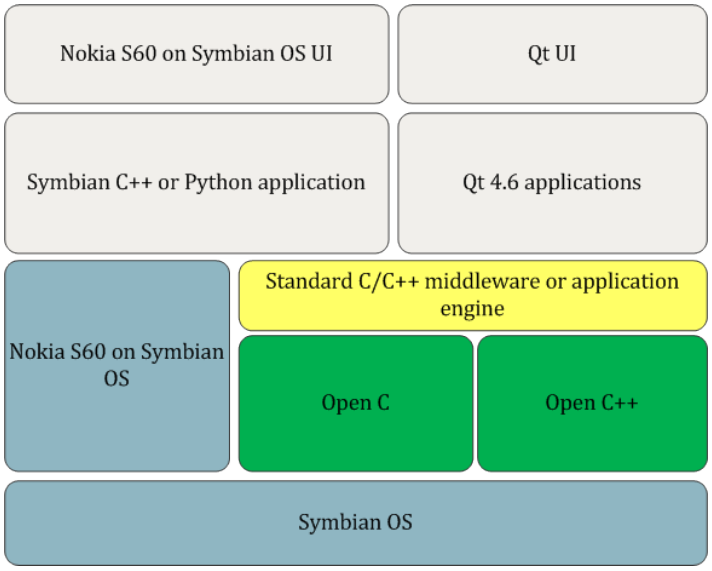


Figure 4.2: Open C/C++ in the Symbian architecture

- It is included with the Open C/C++ plug-in.
- It is the most used secure toolkit implementing SSL/TLS and it is well documented.
- It is already installed in most of Linux distributions.

Having explained the most important factors that determined the selection of the development tools, next Section 4.2.6 explains the tools selected for the energy measuring, the Nokia Energy Profiler.

4.2.6 Nokia Energy Profiler

Nokia Energy Profiler[8] is a stand-alone test and measurement application for S60 3rd Edition, Feature Pack 1 and later devices. The application enables developers to test and monitor their application's energy usage in real time on target devices. The Nokia Energy Profiler is capable of measuring the following:

- Power view: shows the power consumption over a measurement period in Watts(W).

[8]http://www.forum.nokia.com/Technology_Topics/Application_Quality/Power_-Management/Nokia_Energy_Profiler_Quick_Start.xhtml - Last accessed: April 2010

- Current view: displays current consumption, which is the measured current draw from the battery.
- Processor view: shows the CPU load over a measurement period as a percentage of the total available CPU-processing capability.
- RAM Memory view: displays the RAM memory usage over the measurement period.
- Network Speed view: presents the downlink (download) and uplink (upload) speeds through the IP stack.
- WLAN view: shows the received-signal strength when the device is connected to a WLAN base station.
- Signal Levels view: shows the cellular signal levels as RX and TX levels.
- Energy view: shows the cumulative energy consumed over the measurement period.
- Voltage view: shows the battery-voltage levels.
- 3G Timers view: shows the 3G-network-data inactivity timers, also known as T1 and T2.

All the collected data can be exported into several formats: CSV, SVG or JPEG.

As the Nokia Energy Profiler is an independent measurement application, it also provides an external API that gives developers the ability to control the recording process and to retrieve the data collection. The use of the API and the connection procedure is explained in Section 4.5.2.

4.3 Cipher suites selection

This section is intended to justify the selection of the chosen cipher suites. The main decision factor to select these cipher suites was the use of a good symmetric cryptography algorithm. As one of the most used algorithms and defined as a cryptographic standard by the NIST, the Advanced Encryption Standard have been used. Furthermore, previous studies [PRRJ06] of symmetric algorithms have shown that AES has good performance with

Cipher Suite	Certificate type and Key exchange algorithm
TLS_RSA_WITH_AES_128_CBC_SHA	RSA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon
TLS_RSA_WITH_AES_256_CBC_SHA	RSA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon

Table 4.2: List of Cipher suites using AES included in TLS

competitive energy costs. With these facts, the use of AES is more than justified. Thus, the AES cipher suites extending TLS v1.0, defined in [Cho02] have been used.

From the 12 cipher suites supporting AES in the TLS (presented in Table 4.2), some of them needed to be discarded. As the reader can see, all the cipher suites in the list use the same protocols for encryption/decryption, AES, and for message digest, HMAC-SHA-1. Thus, the reasons behind these discards are related to the algorithms used in the authentication and the key exchange algorithm. These reasons are presented here:

- The use of Anonymous Diffie-Hellman (ADH) provides confidentiality but no authentication. This lack of authentication is vulnerable to man-in-the-middle attacks [PP06]. That fact implies that in order to authenticate the peers, an external authentication method should be used.
- The use of the Digital Signature Standard (DSS) [oST06] results in a slower performance [Res01] if it is compared to RSA.
- OpenSSL only implements ephemeral Diffie-Hellman (DHE), but not Diffie-Hellman (DH).

Having discussed the decision factors, the selected cipher suites list is presented in Table 4.3. These cipher suites present high levels of security, as by default DHE-RSA-AES-256-SHA provides the highest level of security.

Cipher Suite	Auth	Key Exchange	Encryption	Digest
RSA-AES-128-SHA	RSA	RSA	AES-128-CBC	SHA1
RSA-AES-256-SHA	RSA	RSA	AES-256-CBC	SHA1
DHE-RSA-AES-128-SHA	RSA	DHE	AES-128-CBC	SHA1
DHE-RSA-AES-256-SHA	RSA	DHE	AES-256-CBC	SHA1

Table 4.3: List of Cipher suites selected for the experiments

4.4 Design of the different scenarios

The use of the SSL protocol to secure a communication channel involves an overhead of computational work and exchanged data, that incurs into a higher energy consumption. To perform an exhaustive analysis of the energy consumption involved in the creation of an SSL secure channel, it is important to remark that the SSL overhead can be divided up into:

1. Increased data volume due to additional items such as certificates and message digests.
2. Computational overhead generated by cryptographic algorithms such as encryption/decryption, authentication or message digests.

As commented in Chapter 3, the SSL protocol is divided into two different phases detailed as:

1. The handshake phase, computationally expensive and costly due to the usage of several network flows.
2. The transfer phase, using the agreed secret key to exchanged the data securely and efficiently.

In order to measure the data overhead and the energy consumption overhead of the establishment of a SSL session, different scenarios need to be built. The different scenarios must be divided into:

1. Local cases, where the server and the mobile client are located in the same network (latency $<$ 1ms).
2. Remote cases, where the mobile client connects to a server, but they are not in the same network (latency depending on the remote service, the Internet provider and the overall network status).

Test scenarios: Local cases

For the Local cases, the implementation of a server capable of accept SSL connections is needed (the server specifications are explained in details in Section 4.5.1). To obtain a complete analysis of the energy consumption and the extra overhead, the next test scenarios have been designed:

- The use of a secure channel increases the exchanged data. To measure this increase, two scenarios are necessary: the use of a non secure channel (just a TCP connection) and the use of an SSL channel. The difference of the data exchanged will be the real overhead of the SSL usage. Different file sizes are tested: 1K, 10K, 50K, 100K, 500K, 1MB and 5 MB.
- The same previous scenario can be used to measure the energy consumption overhead.
- An important part of the SSL connection is the handshake, where public key operations are involved, with high energy consumption impact. Several handshakes are performed using each of the selected cipher suites to measure both the data exchanged and the power consumption.
- Analyze the impact of cached sessions vs. not cached. SSL offers a session resumption mechanism to allow a client-server pair to skip the recalculation of the master secret key that they already computed and exchanged in a previous session. This feature can greatly reduce the cost of the connection setup.
- Study the energy consumption generated due to client authentication. Most of the servers do not require client authentication, but it is interesting to measure the extra data sent by the client.
- Measure the impact of using the different communication interfaces, the ones that the mobile device is equipped with: WLAN and 3G.

Test scenarios: Remote cases

The remote cases are intended to test how the different remote services perform an SSL session establishment, measuring not only the energy consumption, but also the exchanged data and the total execution time.
Before explaining the different test scenarios designed to cover all the presented issues, a list of the remote services to test are presented here:

- Google *(www.google.com)* is the top 1 site with 42,51% of global Internet users who visit google.com (according to Alexa.com). The main service offered is to enable users to search the Web, but it also has different services like webmail, maps, online office tools, just to name a few of them.

- Facebook *(www.facebook.com)* is the top 2 site with 32.69% of global Internet users who visit facebook.com (according to Alexa.com). It is a social networking website that allows user to interact with other users adding them as friends, sending messages, tagging pictures or using applications. Actually facebook, offers 3 different ways to acces to the website:

  1. The original version *(www.facebook.com)*
  2. The secure version *(ssl.facebook.com)* offering SSL protocol in order to encrypt all the communications.
  3. The mobile version *(m.facebook.com)* an optimized site for mobile devices.

- Ovi *(www.ovi.com)* is the brand for Nokia's Internet services. It is a service focused on five key service areas: Games, Maps, Media, Messaging and Music. Users can access these services via website, or using the existing application for mobile devices or desktop computers. The reason for choosing this service is obvious, it is a Nokia service.

- Verisign *(www.verisign.com)* has been selected as a reference of security. This American company was born as a spin-off of RSA Security certification but nowadays is not only one of the major certificate authorities, but also it is in charge of two of the Internet thirteen root nameservers and it manages also the top-level domains .com and .net.

All of them have different characteristics that will be described in Section 5.2.1.

Having introduced the remote services tested, it is time now to explain the scenarios to obtain results for the different interesting issues:

- Study of a complete profile of the different services in order to understand the obtained results.

- Measure the energy impact establishing an SSL session with the different services using the different cipher suites.

- Analyze the impact of cached sessions compared to not cached. SSL offers a session resumption mechanism to allow a client-server pair to skip the recalculation of the master secret key that they already computed and exchanged in a previous session. This feature can greatly reduce the cost of the connection setup but not all the services support this feature.
- Measure the impact of using different communications interfaces, the ones that the mobile device is equipped with: WLAN and 3G.

Test scenarios: Overhead of SSL usage

From the previous experiments, the overall energy consumption and time executions can be obtained, but it is also important to know the real performance and the overhead involving secure connections using the SSL protocol. This scenario is useful to study different aspects of SSL, like:

- SSL data overhead, showing the extra bytes sent using security.
- SSL throughput, showing how the use of SSL protocol affects the overall performance.
- SSL energy consumption overhead, showing the extra energy involving the use of SSL.

4.5 Experimental and Measuring Setup

This section explains how the experimental setup works, the connection between the client and server, as well as the measuring methodology.

4.5.1 Design of the applications

The most important coding aspects of the applications are discussed as well as the different options and modes offered by both the server and the client are explained in the following lines.

Client design

The SSL client is coded in C language with some additional functions like the connection with the Nokia Energy Profiler coded in C++. For the codi-

fication in these languages, the Open C/C++ middleware plugin is needed. The OpenSSL toolkit, included in this Open C plugin, is used to help with all the cryptographic implementations and with the SSL protocol. In order to save energy, memory and make easier the implementation, there is no graphical user interface, only command line interface. In this kind of experimental setup, the main attention corresponds to the cryptographic engine implementation and the improvements around it, but not in the graphical interface. These improvements also include the selection of the communication interface, that is also established only using the command line interface.

In order to describe the design of the mobile client, first is interesting to start with all the functionalities and capabilities:

1. Selection of the parameters:
   - Cipher Suite to use (supported by the server).
   - The Host and the port to connect.
   - Number of reconnections.
   - Interface communications: WLAN or 3G.
2. Different SSL handshake modes:
   - Normal SSL handshake.
   - Session resumption: possibility to save/restore the session in-memory.
   - Client authentication (requested by the server).
3. Full details of the connections process:
   - Complete details about the Cipher suite used and the connection details.
   - Exchanged data sent by the Client and the server.
   - Decoding of the server's certificate.
   - Complete understanding of the different messages exchanged during the handshake, with timestamps and total execution time.
   - Full error trace in case the connection cannot be complete: specification of the SSL error or any other reason.
4. Internal connection with Nokia Energy Profiler, offering complete details about:

- Energy Consumption.
- CPU usage.
- Signal levels.

The SSL handshake process involves operations that take execution times in order of magnitude of milliseconds. To guarantee accurate results in the experiments, several repetitions of the different processes and experiments have to be done. This is also important due to the Nokia Energy Profiler limitations, commented in following Section 4.5.2, that takes measurements in intervals of 0.25ms. Thus, in certain cases like using AES[128,256]-SHA1, the number of repetitions can be up to 300 connections for the local cases. This number of repetitions is selected according to the size of the data exchanged.

Server design

The SSL server is coded in C language. It runs on a PC running a virtualized Ubuntu[9] distribution. The main factor for using Ubuntu is because a Unix environment provides applications like SSLDump[10] to decode all the SSL communications. This was used in the early phases of the research, but as the client is also capable of providing most of the interesting SSLDump capabilities, it was no longer needed. The motive of using virtualization is because of the fact that the IDE used for the developing, Carbide.C++ only works in Windows OS platforms. OpenSSL toolkit modified version was used for the implementation of the SSL protocol. SSLv3 and TLS offers optional compression for the exchanged data, but not compression algorithm is specified. OpenSSL uses by default ZLIB[11] compression if it is supported by both peers. In order to perform some of the experiments, where the comparison between the energy consumption dissipated using a secure channel or just a plain TCP connection, disabling the compression was mandatory in order to exchange the same amount of data in the different scenarios. The modification consists of the compiling process of the libraries without using this compression.

As the server is only intended to serve to the client as a responding peer understanding SSL, in order to focus also in the cryptographic engine and the SSL protocol, there is no graphical interface and the server is operated by parameters given at execution time. The main capabilities are listed here:

[9] http://www.ubuntu.com/ - Last accessed: April 2010
[10] http://www.rtfm.com/ssldump/ - Last accessed: April 2010
[11] http://www.zlib.net/ - Last accessed: April 2010

1. Selection of the parameters:
   - Cipher Suite to use.
   - The listening port.
   - Choose between SSL server or unsecured server.
   - Client authentication request.
   - Allowing session resumption.
2. Full details of the connections process:
   - Complete details about the Cipher suite used and the connection details.
   - Exchanged data sent by the Client and the server.

This section was intended to explain the main design aspects for coding the client and the server applications. Next Section 4.5.2 describes the measuring methodology and how the data is gathered.

4.5.2 Collecting measurements

As mentioned in Section 4.2.6, the Nokia Energy Profiler (NEP) is used to monitor and record the mobile phone power consumption, the battery current, and the voltage over time. In order to extend the functionalities, NEP offers an external plug-in and an API[12] to control NEP functionalities from the mobile SSL client, allowing to start and stop the measurements at any time.
The Control plug-in provides a class for sending commands to NEP, *CJuiceExternalAPI*, but in order to establish this control communication, it is mandatory the initialization of NEP beforehand.
There are a lot of control commands offered by the API, but to allow the control of the measurements, only the following are used:

- Connect() must be called before using any command function. The call informs NEP that the application is connected and ready to send commands.
- Disconnect() disconnects the client from NEP.

[12] http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Plug-ins/Extensions/Nokia_Energy_Profiler_External_APIs/ - Last accessed: April 2010

- StartMeasuring() enables NEP to start the measuring process.
- StopMeasuring() tells NEP to stop the measuring process.
- StartRecording() signals NEP to start recording measurement data. This option has the same effect as starting the recording process manually from the NEP's UI.
- StopRecording() signals NEP to stop the recording process. This option has the same effect as stoping the recording process manually from the NEP's UI.
- SetMarker() signals NEP to mark a hint state in that exact moment, in order to add extra information to the measurements.

Figure 4.3 shows how the client and the server initiate and prepare for the SSL communication along with the measuring protocol.

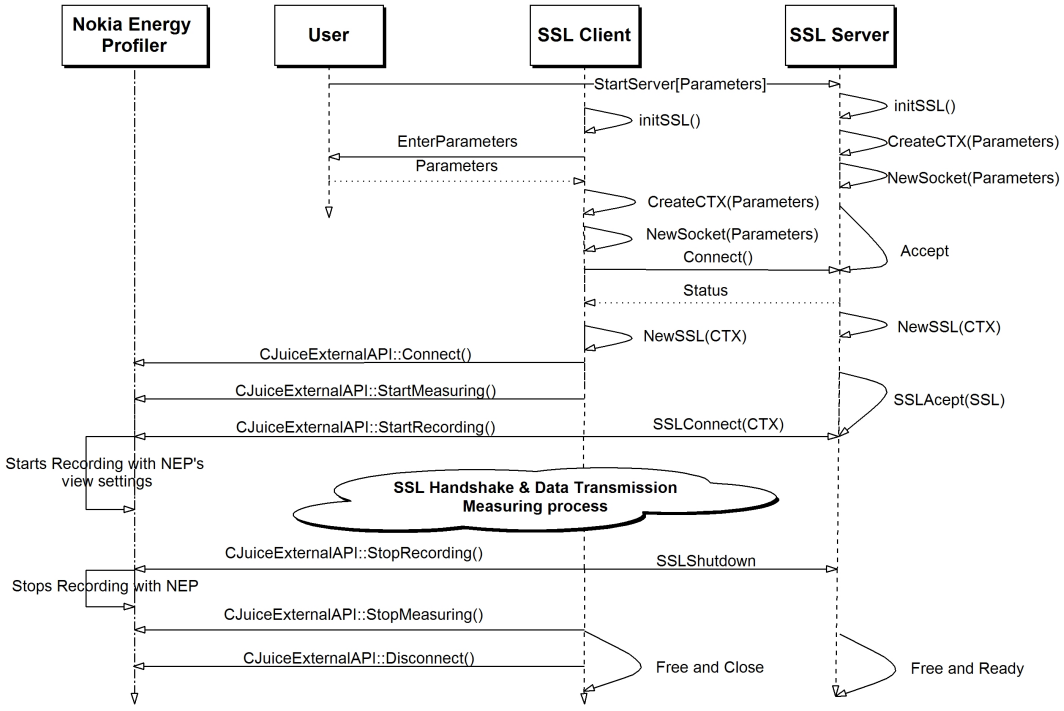


Figure 4.3: Overview of the communication between the client, the server and the Nokia energy profiler

As commented in Section 2.6.1, it is well known that the highest impact in energy consumption is due to the use of communication interfaces like 3G, WLAN or bluetooth, followed by the consumption dissipated due to the screen and its backlight. The energy measurement process is done in such a way that only the minimum services and programs needed to operate the mobile device are running. Thus, the client needs to wait until the display state changes to a dim state, where the backlight is turned off. This means an important decrease in the energy consumption and better accuracy in the measurements.

Nokia Energy profiler is capable of taking measurements in intervals of 1s, 0,5s or 0,25s. As the cryptographic operations involved in the process have execution times significantly lower, the need of repetitions to get accurate results is mandatory.

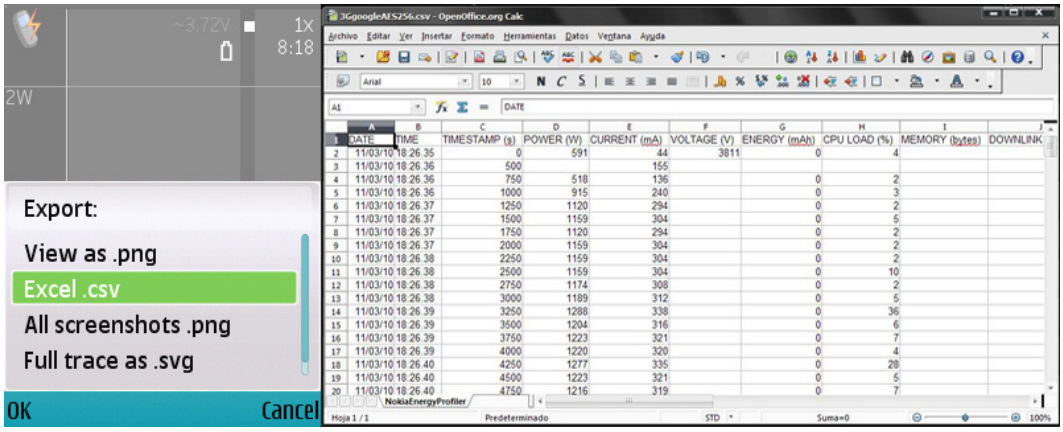


Figure 4.4: Export process in NEP and CSV exported data results

Once the application have finished the execution and the measurement process has been done, it is time to collect the gathered data. NEP has an option to export all the measured data into CSV[Sha05] format. The exported data provides the measured data from all the selected views to measure, but in order to measure the energy consumption, only the power view values are interesting. The exportation process and the obtained results are shown in Figure 4.4.

Using Nokia Energy Profiler, there is no need to use any external device to measure the energy consumption and the results obtained can easily be exported to a digital format like CSV or other graphical formats like PNG or SVG.

4.5.3 Experimental Setup

Once the selected technologies have been explained, along with the different scenarios to test and the requirements, it is time to describe the experimentation setup and how they are connected.

Figure 4.5 describes the experimental setup used to explain in a graphical way how they are connected and how the data is measured.

The experimental setup for the secure client-server communication consists in a client that connects to a LAN through a wireless access point created by a router, while the server is a laptop wired to the LAN. The client is running on a Nokia N95 (see Subsection 4.2.2 for more details), a Symbian phone device, that contains a dual ARM processor clocked at 322Mhz with a Li-Ion battery with 950 mAh rating . The server is a laptop equipped with a 2.0Ghz Intel mobile Core 2 Duo T7250 having 2048 MB of DDR2 RAM and running a virtualized distribution of Ubuntu 9.10 using VMWare Workstation[13].

For measuring the energy consumption and other required values, the Nokia Energy Profiler is running in background while the client is performing all the needed operations. NEP is connected internally with the client to only measure the desired operations.

As described in the previous section, the two different communication interfaces, WLAN and 3G, are used in different cases. The previous description is when the experiments are executed using the mobile WLAN interface. When the 3G interface is needed, the mobile connects directly to the Internet provider and accesses the router via WAN. An important aspect to comment is the different interface bandwidth. The router is connected to the Internet using a cable connection of about 5Mbps download and 600 kbps upload, while the 3G interface only achieves about 324 kbps both ways. It is important to comment this issue, because it will affect to all the following experiments, the increasing execution time for the case of 3G that involves a higher energy consumption.

[13]http://www.vmware.com/products/workstation/ - Last accessed: April 2010

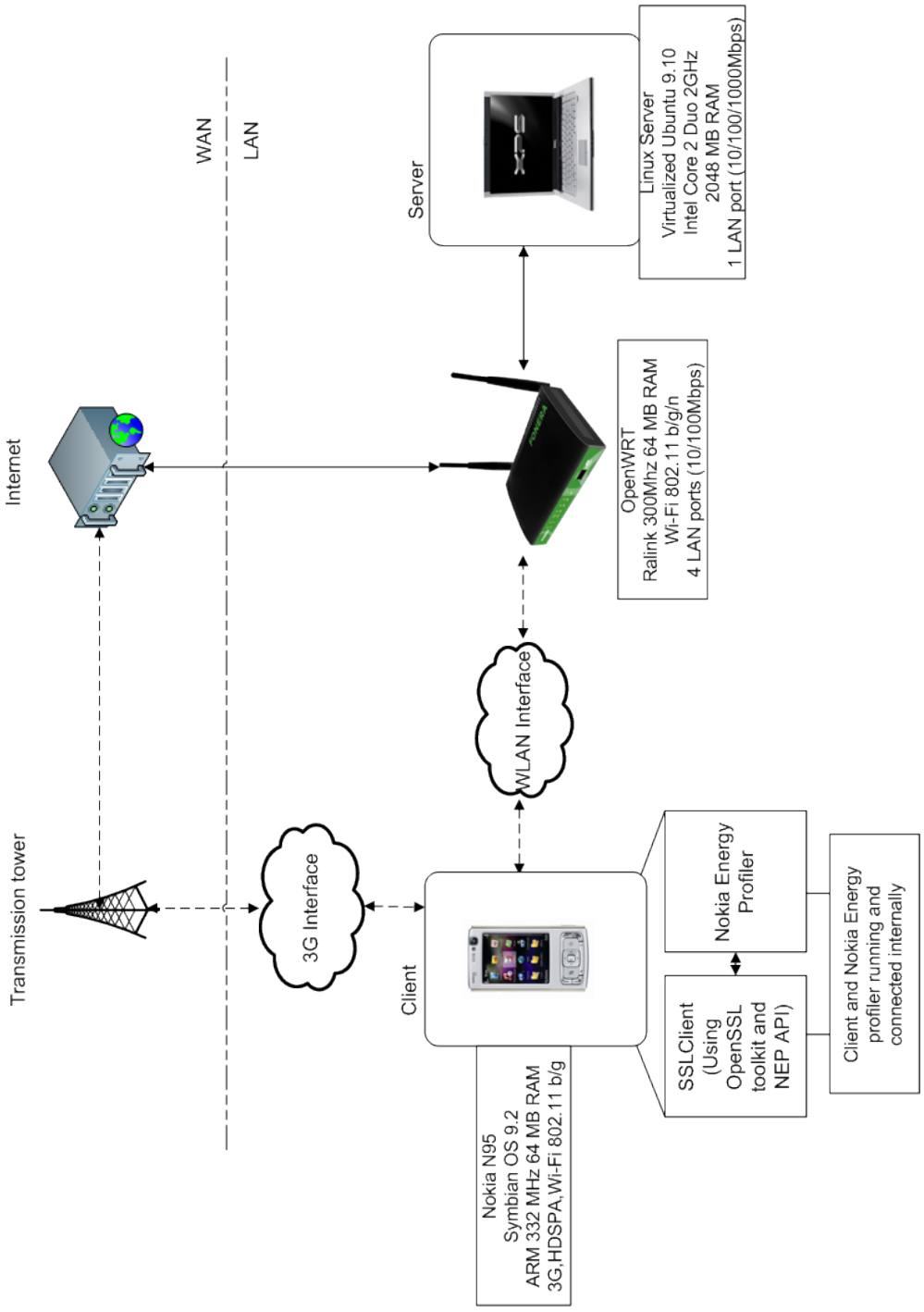


Figure 4.5: Secure client-server experimental setup

Chapter 5

Analysis of Measurements

In this chapter, a comprehensive empirical analysis of the energy consumption of the SSL protocol is done, using the experimental setup described in Chapter 4.

5.1 Test scenario: Local case

This test scenario groups all the experimental cases performed in the same local area network, where latency was less than 1ms.

5.1.1 Exchanged data

To understand well how much data is being exchanged in the TLS handshake, the tests include an option to measure the bytes exchanged in the operation. As in the handshake only public key cryptography (RSA) and public key exchange mechanism (RSA or Diffie-Hellman) are used, the algorithms related with encryption (AES) and with integrity and authentication (HMAC-SHA1) do not affect and the results show that the data exchanged is the same either we use AES128 or AES256, the only difference is because of the use of RSA or Diffie-Hellman.

Table 5.1 shows the results divided into the three operation modes: Client Authentication, No Client Authentication and Session Resumption. For each mode, the 4 cipher suites have been tested. The row headed with $C \rightarrow S$ means data sent by the client to the server. The next row is the data sent in the other way, from the server to the client.

Operation Mode	Cipher suite	$C \rightarrow S$	$S \rightarrow C$
No Client Authentication	AES128-SHA	249	2368
	AES256-SHA	249	2368
	DHE-RSA-AES128-SHA	249	2770
	DHE-RSA-AES256-SHA	249	2770
Client Authentication	AES128-SHA	1772	2348
	AES256-SHA	1772	2348
	DHE-RSA-AES128-SHA	1772	2752
	DHE-RSA-AES256-SHA	1772	2752
Session Resumption	NOT RELEVANT	142	168

Table 5.1: Exchanged data performing a handshake for the local case

Analyzing the results:

- The client always sends the same amount of data, independently of the cipher suite used.
- For the client authentication, the increase of data is important. The server requests a client certificate and the client has to send it. This exchanged data depends, of course, of the client certificate length.
- The impact of using session resumption is really important. The data sent by the client is reduced nearly 50% and the data sent by the server is divided by 16, either with client authentication or not. Also the data is invariable no matter the cipher suite used.
- Using Diffie-Hellman algorithm for Key agreement increases in 402 bytes the data sent by the server, in comparison with RSA.

5.1.2 Energy consumption

Once the exchanged data involved in an SSL handshake have been discussed, they will help to understand the energy consumption dissipated in the handshake process. Table 5.2 shows the power consumption in milijoules (mJ) for the different cipher suites, requesting or not client authentication.

Cipher Suite	Non Client Authentication	Client Authentication
RSA-AES-128-SHA	99	155
RSA-AES-256-SHA	101	155
DHE-RSA-AES-128-SHA	739	839
DHE-RSA-AES-256-SHA	742	842
Session Resumption	64	
Non secure	25	

Table 5.2: Energy consumption for the SSL handshake in the local case

Analyzing Image 5.1, different conclusions can be obtained:

- The most important conclusion is that the use of Diffie-Hellman affects critically to the energy consumption per connection, compared with RSA. This was explained in Section 2.6.2. Diffie-Hellman protocol has harder computational operations compared with RSA, involving longer execution times where the network interface must be kept up, increasing the energy consumption. In addition, these operations mean higher work for the CPU, involving higher power needs.

- The use of Client Authentication also increases the consumption, as more data is exchanged. These values depend on the data length of the client certificate, as mentioned before.

- The use of session resumption can greatly save energy, specially avoiding the use of Diffie-Hellman where the savings can be from 842 mJ per connection (for the worst case) to only 64 mJ.

- The establishment of a non secure connection costs only 25 mJ, compared with the 100 mJ using RSA or more than 700 mJ using DHE.

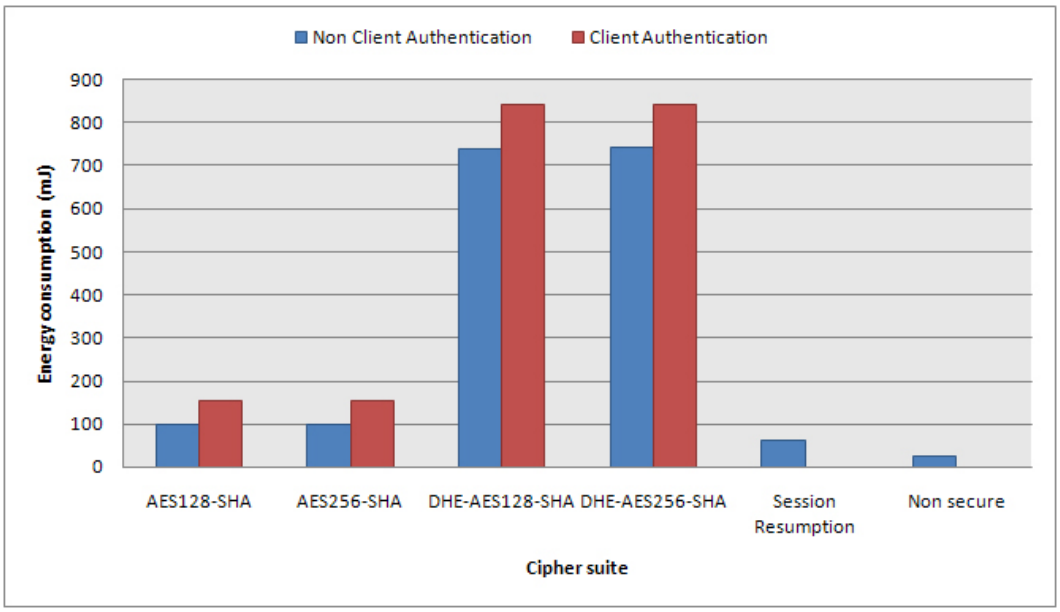


Figure 5.1: Energy consumption for the SSL handshake in Local case

5.2 Test scenario: Remote cases

While the previous sections describes the test cases where latency is less than 1 ms, in the remote cases, these values changes depending the remote service and the network interface used.

5.2.1 Services profile study

Before starting with the experiments and the measurements, a full profile study of the chosen remote services needs to be done. This study is intended to analyze the properties of each service including:

- SSL protocols available
- The Server SSL implementation
- Default cipher suite
- Server Certificate length
- Public server-key length
- Support of Session resumption

- Support of Ephemeral Diffie-Hellman
- List of Cipher suites available

The SSL protocols available refer to the version of the SSL protocol supported by the server. Due to SSLv2 is vulnerable to attacks, none of them supports that protocol, only SSLv3 and TLSv1 are supported. SSLv2 does not support either useful features like client authentication. Newer TLS versions like TLSv1.1 and TLSv1.2 are not supported yet.

The Server implementation is related with the internal implementation used. The most important are Apache mod_SSL[1], Apache mod_NSS[2] or Internet Information Services (IIS)[3].

The default cipher suite is used in order to know the level of security the server is willing to use by default. This selection is mostly based on a right balance between security and performance.

The server certificate length shows the size of the server's certificate, being one of the most important factors in the energy consumption.

The level of security desired by the server depends also on the public server-key length, which is usually 1024 bits but if better security levels are required, higher values like 2048 bits can be used, but affecting significantly to performance and energy consumption.

The support of session resumption, as shown in previous experiments for the local case, can greatly reduce the energy consumption. All the services with the exception of Facebook allow this feature. The most important reason behind this issue is that maintaining a cache of all the shared secrets computed with each SSL client is really expensive, in data storage meanings. In some cases, the improvement obtained with the use of session resumption is not worth it, and the servers from the remote services request a complete SSL handshake each time a client wants to establish a secure connection using SSL.

The use of Ephemeral Diffie-Hellman provides better security levels compared with RSA but also affects considerably to performance and energy consumption. Different services here do not allow this protocol, as they prefer better performance and scalability.

Finally, a list of the supported cipher suites is presented. This can help to understand the different policies of each service attending to performance-

[1] http://www.modssl.org/ - Last accessed: April 2010
[2] http://www.mozilla.org/projects/security/pki/nss/ - Last accessed: April 2010
[3] http://www.iis.net/overview - Last accessed: April 2010

security-scalability.

This study will help to understand better the different exchanged data and the energy consumption variations. All the gathered information can be checked in Table 5.3

Service	Google	Facebook	SSL.Facebook		M.Facebook		Verisign		Ovi	
Protocols available	SSLv3, TLSv1.0		SSLv3, TLS1.0		SSLv3, TLS1.0		SSLv3, TLSv1.0		SSLv3, TLS1.0	
Server Implementation	Apache mod_SSL		Apache mod_NSS		Apache mod_NSS		Apache mod_SSL		IIS 7.5	
Default Cipher Suite	AES256-SHA		RC4-MD5		RC4-MD5		DHE-RSA-AES256-SHA		DHE-RSA-AES256-SHA	
Server Certificate length	1625 bytes		4533 bytes		4642 bytes		4519 bytes		1738 bytes	
Public server-key	1024 bits		1024 bits		2048 bits		1024 bits		2048 bits	
Session resumption	YES		NO		NO		NO		YES	
Ephemeral Diffie Hellman	NO		NO		YES		YES		YES	
Cipher suites available	AES256-SHA	256 bit	AES256-SHA	256 bit	DHE-RSA-AES256-SHA	256 bit	DHE-RSA-AES256-SHA	256 bit	ADH-AES256-SHA	256 bit
	DES-CBC3-SHA	168 bit	DES-CBC3-SHA	168 bit	AES256-SHA	256 bit	AES256-SHA	256 bit	DHE-RSA-AES256-SHA	256 bit
	AES128-SHA	128 bit	AES128-SHA	128 bit	EDH-RSA-DES-CBC3-SHA	168 bit	EDH-RSA-DES-CBC3-SHA	168 bit	AES256-SHA	256 bit
	RC4-SHA	128 bit	RC4-SHA	128 bit	DES-CBC3-SHA	168 bit	DES-CBC3-SHA	168 bit	ADH-DES-CBC3-SHA	168 bit
	RC4-MD5	128 bit	RC4-MD5	128 bit	DHE-RSA-AES128-SHA	128 bit	DHE-RSA-AES128-SHA	128 bit	EDH-RSA-DES-CBC3-SHA	168 bit
			AES128-SHA	128 bit	AES128-SHA	128 bit	AES128-SHA	128 bit	DES-CBC3-SHA	168 bit
			RC4-SHA	128 bit	RC4-SHA	128 bit	RC4-SHA	128 bit	ADH-AES128-SHA	128 bit
			RC4-MD5	128 bit	RC4-MD5	128 bit	RC4-MD5	128 bit	DHE-RSA-AES128-SHA	128 bit
			EDH-RSA-DES-CBC-SHA	56 bit	EDH-RSA-DES-CBC-SHA	56 bit	EDH-RSA-DES-CBC-SHA	56 bit	AES128-SHA	128 bit
			DES-CBC-SHA	56 bit	DES-CBC-SHA	56 bit	DES-CBC-SHA	56 bit	ADH-RC4-MD5	128 bit
			EXP-EDH-RSA-DES-CBC-SHA	40 bit	EXP-EDH-RSA-DES-CBC-SHA	40 bit			RC4-SHA	128 bit
			EXP-DES-CBC-SHA	40 bit	EXP-DES-CBC-SHA	40 bit			RC4-MD5	128 bit
			EXP-RC2-CBC-MD5	40 bit	EXP-RC2-CBC-MD5	40 bit			ADH-DES-CBC-SHA	56 bit
			EXP-RC4-MD5	40 bit	EXP-RC4-MD5	40 bit			EDH-RSA-DES-CBC-SHA	56 bit
									DES-CBC-SHA	56 bit
									EXP-ADH-DES-CBC-SHA	40 bit
									EXP-ADH-RC4-MD5	40 bit
									EXP-EDH-RSA-DES-CBC-SHA	40 bit
									EXP-DES-CBC-SHA	40 bit
									EXP-RC2-CBC-MD5	40 bit
									EXP-RC4-MD5	40 bit

Table 5.3: Full profile of the remote services analyzed

5.2.2 Exchanged data

As the same experiments performed for the local case, it is important to measure the amount of exchanged data to understand well the energy consumption involved in the establishment of an SSL handshake. Table 5.4 show the results of the experiment. Attending to these results, different statements can be mentioned:

- The client always sends 286 bytes in the handshake, but this amount is increased when the server uses 2048 bits public key length up to 414 bytes. This case is only for services like SSL.facebook and Verisign.
- Depending on the server's certificate length, the data sent by the server is different. The longest certificates correspond to Verisign, Facebook and SSL.facebook. This issue will be reflected in the energy consumption.
- Related with the previous point, the total amount of data sent by the server is lightly increased when using Ephemeral Diffie-Hellman instead of RSA. The increase is 370 bytes when using 1024 bits public key length and 498 bytes for 2048 bits.
- Session resumption can reduce the exchanged data from more than 5 KB to only 179 bytes, increasing the battery life.

SSL client authentication can not be tested on these services, as they do not allow client authentication. The reader should not be confused with client authentication using a username or password in the remote service. This authentication is related with the server requesting a certificate from the client in the SSL handshake.

5.2.3 Energy consumption using WLAN

Once a full profile of the services and the total data exchanged has been described, the energy consumption using WLAN interface is presented in Figure 5.2 and Table 5.5. The results show different conclusions:

- The use of Diffie-Hellman affects dramatically in the setup energy consumption, not only increasing the execution time but also increasing the data sent by the parties during all the handshake. This was already commented in Section 2.6.2 and observed in Section 5.1.2.

Operation Mode	Cipher suite	C → S	S → C
Google	AES128-SHA	286	1777
	AES256-SHA	286	1777
	DHE-RSA-AES128-SHA	Not supported	
	DHE-RSA-AES256-SHA		
	SESSION RESUMPTION	138	179
Facebook	AES128-SHA	286	4705
	AES256-SHA	286	4705
	DHE-RSA-AES128-SHA	Not supported	
	DHE-RSA-AES256-SHA		
	SESSION RESUMPTION		
m.Facebook	AES128-SHA	286	1004
	AES256-SHA	286	1004
	DHE-RSA-AES128-SHA	286	1374
	DHE-RSA-AES256-SHA	286	1374
	SESSION RESUMPTION	Not supported	
ssl.Facebook	AES128-SHA	414	4794
	AES256-SHA	414	4794
	DHE-RSA-AES128-SHA	286	5292
	DHE-RSA-AES256-SHA	286	5292
	SESSION RESUMPTION	Not supported	
Verisign	AES128-SHA	414	4671
	AES256-SHA	414	4671
	DHE-RSA-AES128-SHA	286	5201
	DHE-RSA-AES256-SHA	286	5201
	SESSION RESUMPTION	138	179
Ovi	AES128-SHA	286	3682
	AES256-SHA	286	3682
	DHE-RSA-AES128-SHA	286	4084
	DHE-RSA-AES256-SHA	286	4084
	SESSION RESUMPTION	138	179

Table 5.4: Exchanged data performing a handshake for remote cases

- VeriSign, facebook.com and ssl.facebook.com have certificate's length higher than 4.7 KB (as commented in Section 5.2.2), and in that sense they have the highest energy consumption.
- Both SSL.facebook and VeriSign use 2048 bits public server key, that affects in the calculation process, causing a bigger energy consumption to compute the master key.
- The use of session resumption, where possible, greatly benefits to the energy consumption. Resumption was possible in an effective way in Google, VeriSign and Ovi.
- The server implementation and infrastructure also affect the energy consumption. For example, the mobile version of Facebook (m.Facebook) requires almost double more energy than Google, 482 mJ and 273 mJ respectively.

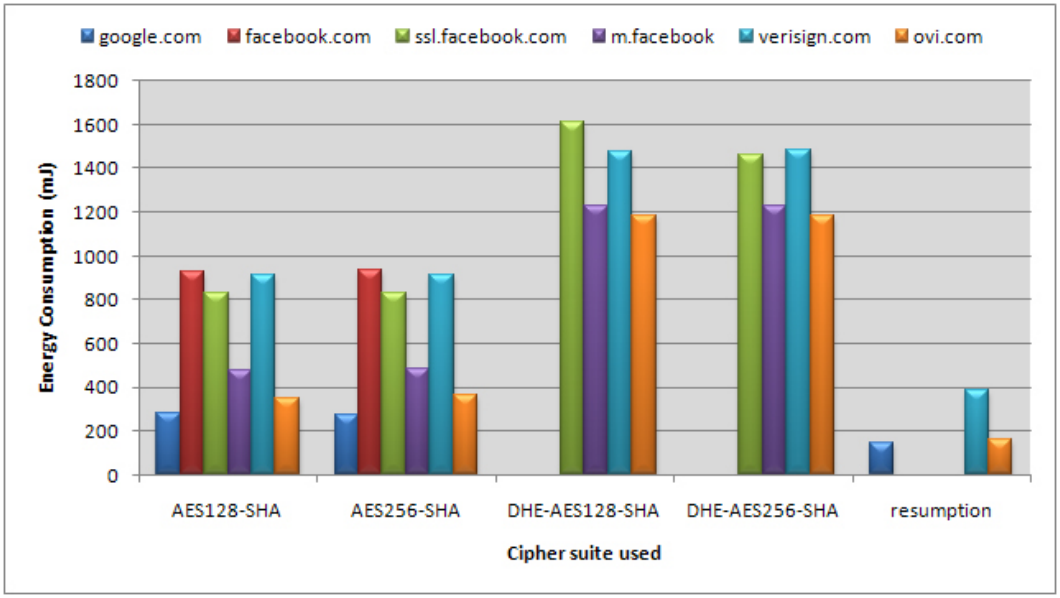


Figure 5.2: Energy consumption per connection in different remote services using WLAN

5.2.4 Energy consumption using 3G

In the previous section, the energy consumption using the WLAN interface is explained. Here, the same experiments were executed but using 3G interface,

	Google	Facebook	SSL.facebook	m.Facebook	Verisign	Ovi
RSA-AES-128-SHA	279,76	930,03	833,03	480,31	915,35	346,21
RSA-AES-256-SHA	273,41	932,02	831,30	482,36	914,94	364,78
DHE-RSA-AES-128-SHA	Not supported		1612,54	1228,82	1472,74	1181,56
DHE-RSA-AES-256-SHA	Not supported		1461,95	1224,92	1484,46	1179,50
Session Resumption	142,52	Not supported			385,24	161,78

Table 5.5: Energy consumption for the remote services handshake using WLAN

where latency, network problems and other related issues increase the energy consumption. Results of the measurements are presented in Figure 5.3 and in Table 5.6.

The same conclusions as using WLAN apply here for the 3G, but with a main difference, the important increase in the energy consumption due to the use of the 3G interface. This higher energy consumption of 3G compared with WLAN was explained in Section 2.6.1 and here it is demonstrated. The increase can be up to 450% for the example of Ovi using RSA, 346 mJ using WLAN mode compared to 1590 mJ using 3G.

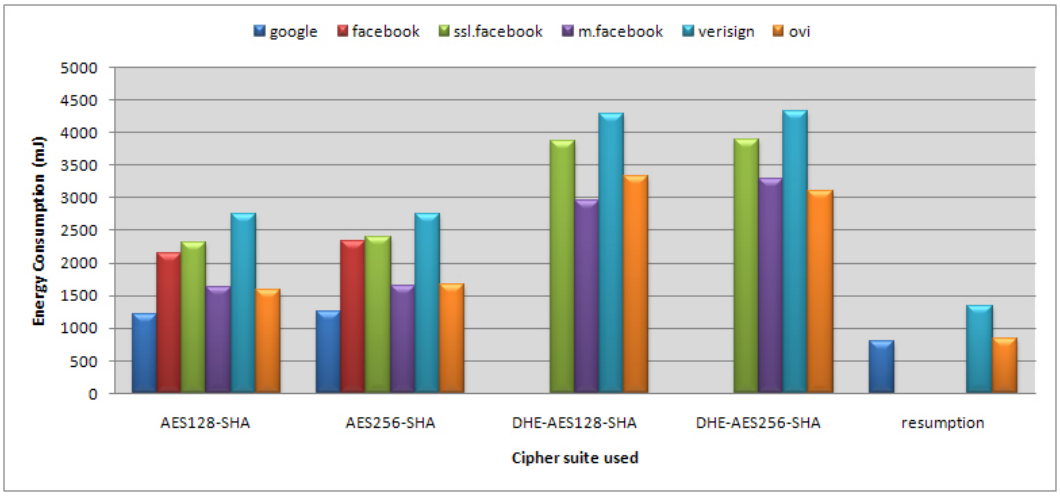


Figure 5.3: Energy consumption per connection in different remote services using 3G

5.2.5 Handshake Steps Execution time using WLAN

In order to analyze which operations involved in the SSL handshake take more time, some experiments have been performed obtaining different conclusions. RSA and DHE protocol have been used to analyze the important difference

	Google	Facebook	SSL.facebook	m.Facebook	Verisign	Ovi
RSA-AES-128-SHA	1207	2149	2323	1632	2756	1590
RSA-AES-256-SHA	1246	2332	2401	1641	2748	1663
DHE-RSA-AES-128-SHA	Not supported		3873	2960	4299	3344
DHE-RSA-AES-256-SHA	Not supported		3897	3301	4324	3102
Session Resumption	142,52	Not supported			1341	832

Table 5.6: Energy consumption for the remote services handshake using 3G

observed in previous experiments, where DHE takes more time and increases dramatically the energy consumption.

These experiments facilitate to study in which steps involved in the SSL handshake is consumed the major part of the time. Using these experiments not only the most hard computational steps are confirmed but also trying to figure out if the server is responding in the same way in all the situations (latency, number of hops, and problems related with the connection and the bandwidth). It is important to mention that each service has been tested 5 times, so the results shown here are the average of these results.

Table 5.7 shows the results of the performed experiments using RSA key exchange protocol with a WLAN interface. The steps involved in the SSL Handshake and the computations done in each one are explained in detail in Section 3.4.4.

Handshake Step	Google	Facebook	SSL.facebook	m.Facebook	Verisign	Ovi
ClientHello	0,003	0,002	0,002	0,003	0,003	0,003
ServerHello	**0,059**	**0,192**	**0,199**	**0,128**	**0,230**	**0,060**
Certificate	**0,008**	**0,138**	**0,211**	**0,005**	**0,197**	**0,060**
ServerHelloDone	0,025	0,068	0,069	0,016	0,076	0,027
ClientKeyExchange	0,012	0,012	0,03	0,012	0,030	0,012
ChangeCipherSpec	0,004	0,004	0,004	0,004	0,004	0,004
Finished	0,004	0,004	0,004	0,005	0,005	0,006
ChangeCipherSpec	**0,059**	**0,230**	**0,221**	**0,133**	**0,299**	**0,063**
Finished	0,005	0,007	0,004	0,004	0,005	0,005
Alert	0,055	0,055	0,054	0,054	0,053	0,056
TOTAL TIME	0,234	0,713	0,799	0,362	0,927	0,296

Table 5.7: Execution times for each step involved in SSL handshake using RSA with WLAN

Attending to Figure 5.4 different main conclusions can be done:

- The step where the server sends its certificate, `Certificate`, depends on the length of the certificate. This issue is a really important factor in order to increase or reduce the total handshake time. For services like SSL.Facebook or VeriSign this step takes around 200ms, while for

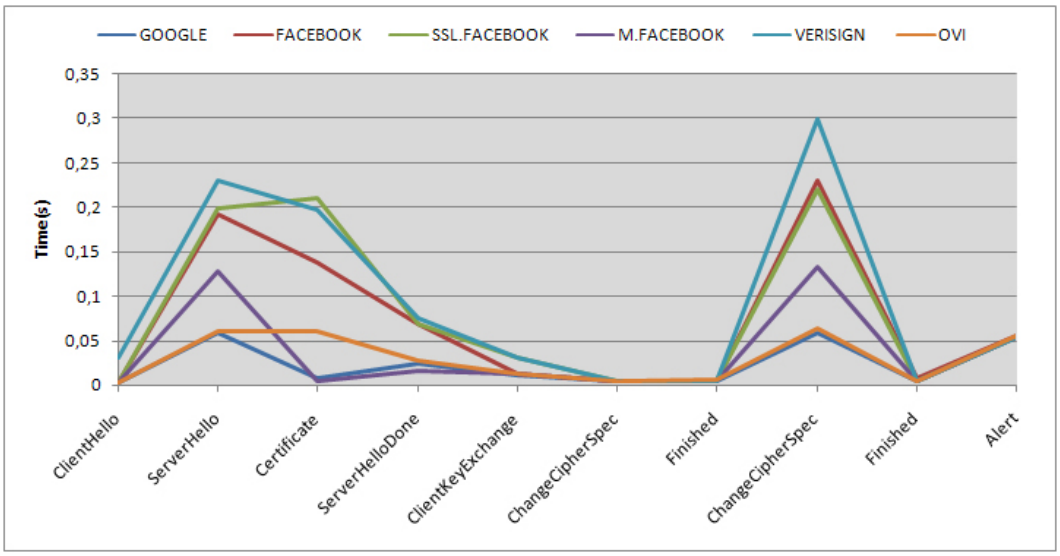


Figure 5.4: Handshake steps Execution time using RSA with WLAN

Google and Facebook mobile, using smaller certificates, the time involved in this operation is less than 10 ms. The reason behind this issue is that the certificate's length is small enough to fit into one TCP packet. Services with larger certificates need to deal with the TCP window mechanism in the client, modifying the length of the packet the client is willing to receive, increasing the number of TCP messages exchanged and increasing the total execution time.

- The Public key length also affects the execution time, as higher values cause lower performance.
- `ServerHello`, `ChangeCipherSpec` and `ServerHelloDone` responses take most of the time. The operations and computations involved in these steps are explained in detail in Section 3.4.4, but for a general understanding and explanation, public key cryptography operations like signing, verifying and decrypting take place in these steps and, as shown in Table 2.1, asymmetric cryptography has a low performance.
- As commented before, these delays also depend on the service used. Several factors can cause delays, like server load balancing and the load existing in the servers, the performance of the servers itself or other related factors, such as the SSL implementation and the possible use of improved encryption hardware. As a clear example of this issue, the different execution times of the `ChangeCipherSpec`, where

for different services using the same public key length,1024 bits, take different execution time. It is important to remark that services using 2048 bits need more time, due to the harder public cryptography operations involved, in addition to the issues previously commented.

- The other exchanged messages, in comparison with the ones commented here, are not significantly important, as their execution time is small.

The results obtained confirm the same observations done in [SGM09], where `ServerHello`, `ServerKeyExchange` and `ChangeCipherSpec` from the server side take most of the time of the overall handshake process.

Using Ephemeral Diffie-Hellman as a key agreement protocol improves the security in the SSL handshake, but it also increases the total amount of time involved in the operation. As Google and facebook, in their normal service do not support the Diffie-Hellman protocol, these services cannot be tested.

Table 5.8 shows the results of the performed experiments using DHE key agreement protocol with WLAN interface. The involved steps in the SSL Handshake and the computations performed in each one are explained in detail in Section 3.4.4.

Handshake Step	SSL.facebook	m.Facebook	Verisign	Ovi
ClientHello	0,003	0,003	0,003	0,002
ServerHello	**0,064**	**0,132**	**0,247**	**0,291**
Certificate	**0,065**	**0,005**	**0,210**	**0,240**
ServerKeyExchange	0,026	0,015	0,069	0,075
ServerHelloDone	0,012	0,012	0,030	0,030
ClientKeyExchange	**0,829**	**0,826**	**0,827**	**0,826**
ChangeCipherSpec	0,004	0,004	0,004	0,004
Finished	0,004	0,005	0,005	0,005
ChangeCipherSpec	**0,067**	**0,186**	**0,234**	**0,325**
Finished	0,005	0,004	0,005	0,005
Alert	0,055	0,055	0,055	0,055
TOTAL TIME	1,688	1,246	1,857	1,133

Table 5.8: Execution times for each step involved in the SSL handshake using DHE with WLAN

Taking a look at the Figure 5.5, three main comments can be mentioned:

- DHE has a really low performance compared with RSA.
- As mentioned before for the RSA case, the length of the certificate and the three responses from the server,`ServerHello`, `ChangeCipherSpec` and `ServerHelloDone`, are the most time-consuming operations.

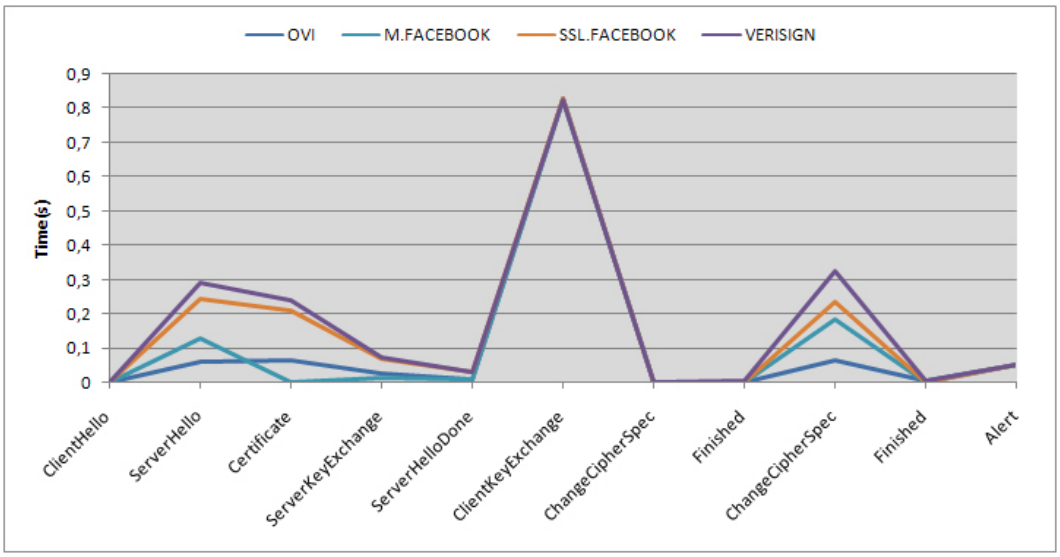


Figure 5.5: Handshake steps Execution time using DHE with WLAN

- `ClientKeyExchange` have been confirmed as the hardest computational involved step. Although this operation is executed in the client side and as the results shown, it takes always the same amount of time using WLAN or 3G.
- The use of DHE involves the use of a new server message, `ServerKeyExchange`, that is not present when RSA is used and where the server sends the needed keys to the client to negotiate about the shared secret. Although this operation does not increase considerably the execution time, is important to mention it.

These results also agree with the one mentioned in [SGM09], where the same critical steps performed with RSA and Diffie-Hellman take more time, due to the different nature of the operations involved in DHE.

5.2.6 Handshake Steps Execution time using 3G

The previous experiments were performed using WLAN as the communication interface. Now, 3G is used, increasing the execution time due to the natural environment of 3G wireless communications.

Tables 5.9 and 5.10 show the results obtained from the experiments using RSA and DHE key exchange protocol with 3G interface.

Handshake Step	Google	Facebook	SSL.facebook	m.Facebook	Verisign	Ovi
ClientHello	0,004	0,002	0,002	0,002	0,002	0,002
ServerHello	**0,312**	**0,329**	**0,343**	**0,427**	**0,606**	**0,583**
Certificate	**0,015**	**0,303**	**0,005**	**0,364**	**0,567**	**0,608**
ServerHelloDone	0,046	0,026	0,015	0,069	0,068	0,073
ClientKeyExchange	0,023	0,012	0,013	0,012	0,031	0,031
ChangeCipherSpec	0,007	0,004	0,005	0,004	0,004	0,004
Finished	0,008	0,004	0,005	0,005	0,005	0,012
ChangeCipherSpec	**0,296**	**0,310**	**0,630**	**0,413**	**0,590**	**0,606**
Finished	0,008	0,005	0,005	0,005	0,005	0,005
Alert	0,111	0,062	0,058	0,058	0,057	0,058
TOTAL TIME	0,831	1,358	1,934	1,079	1,981	1,056

Table 5.9: Execution times for each step involved in SSL handshake using RSA with 3G

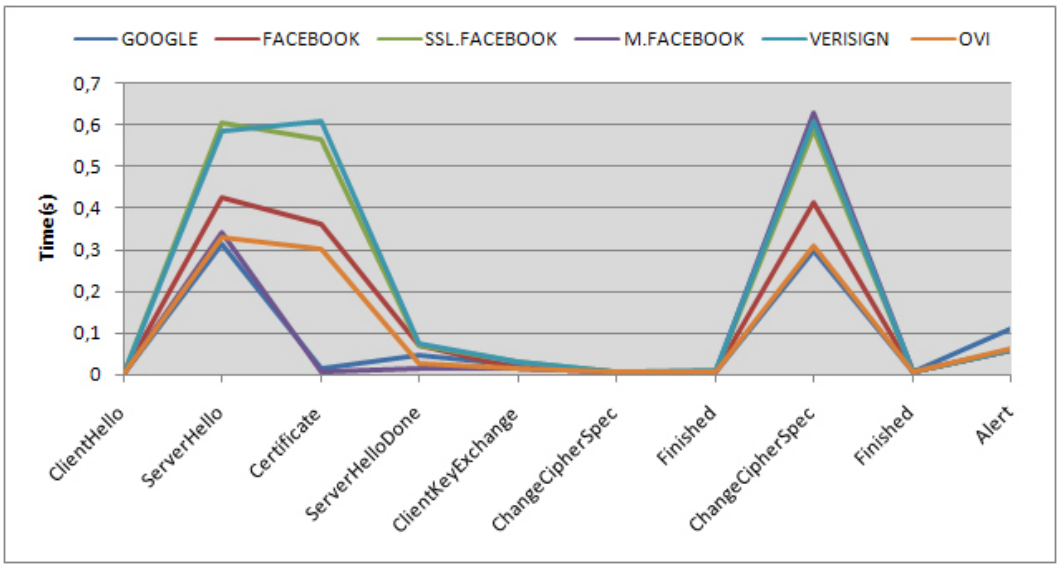


Figure 5.6: Handshake steps Execution time using RSA with 3G

Taking a look at Figures 5.6 and 5.7, two main conclusions can be commented:

- Using 3G the same conclusions as the ones presented for the WLAN case can apply here, but with the important increase of execution time. For example, testing SSL.Facebook using WLAN the execution time for `ChangeCipherSpec`is 221 ms where the same step using 3G is about 630 ms, more than 400 ms of difference in only one critical operation.

- `ClientKeyExchange` has been confirmed as the hardest computational involved step. Although this operation is executed in the client side and as the results shown, it takes always the same amount of time using WLAN or 3G.

Handshake Step	SSL.facebook	m.Facebook	Verisign	Ovi
ClientHello	0,002	0,002	0,002	0,002
ServerHello	**0,322**	**0,399**	**0,494**	**0,566**
Certificate	**0,297**	**0,005**	**0,416**	**0,467**
ServerKeyExchange	0,026	0,015	0,070	0,074
ServerHelloDone	0,014	0,021	0,031	0,030
ClientKeyExchange	**0,830**	**0,824**	**0,824**	**0,823**
ChangeCipherSpec	0,004	0,004	0,004	0,004
Finished	0,005	0,005	0,006	0,005
ChangeCipherSpec	**0,330**	**0,420**	**0,450**	**0,494**
Finished	0,005	0,005	0,004	0,005
Alert	0,059	0,059	0,059	0,057
TOTAL TIME	2,36	1,759	2,526	1,894

Table 5.10: Execution times for each step involved in SSL handshake using DHE with 3G

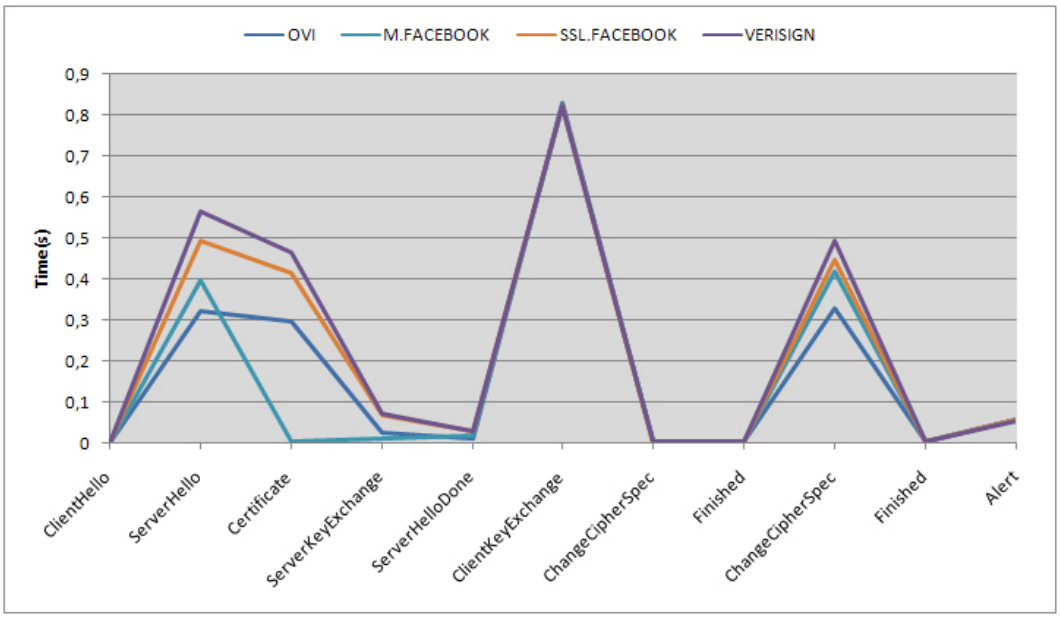


Figure 5.7: Handshake steps Execution time using DHE with 3G

5.2.7 Comparing WLAN and 3G

Having presented results using WLAN and 3G separately, in order to make a good comparison of the obtained results, the next Figures 5.8, 5.9 and 5.10 show a comparative of the results obtained with the average of all the energy consumptions and execution times from the different services tested, in order to show a general conclusion about the use of the two different interface communications.

- The use of 3G not only increases the execution time due to latency and bandwidth, but it also increases significantly the energy consumption due to the nature of the interface and the need to maintain the interface on for a longer time due to the 3G timers.
- The energy consumption using 3G in average is 300% higher than using WLAN.
- The main differences observed by using 3G instead of WLAN are found in these 3 steps: `ServerHello`, `Certificate` and `ChangeCipherSpec`. These main differences are due to the latency of using 3G, with higher times compared with WLAN. In addition, 3G bandwidth is lower, that means higher transfer times.

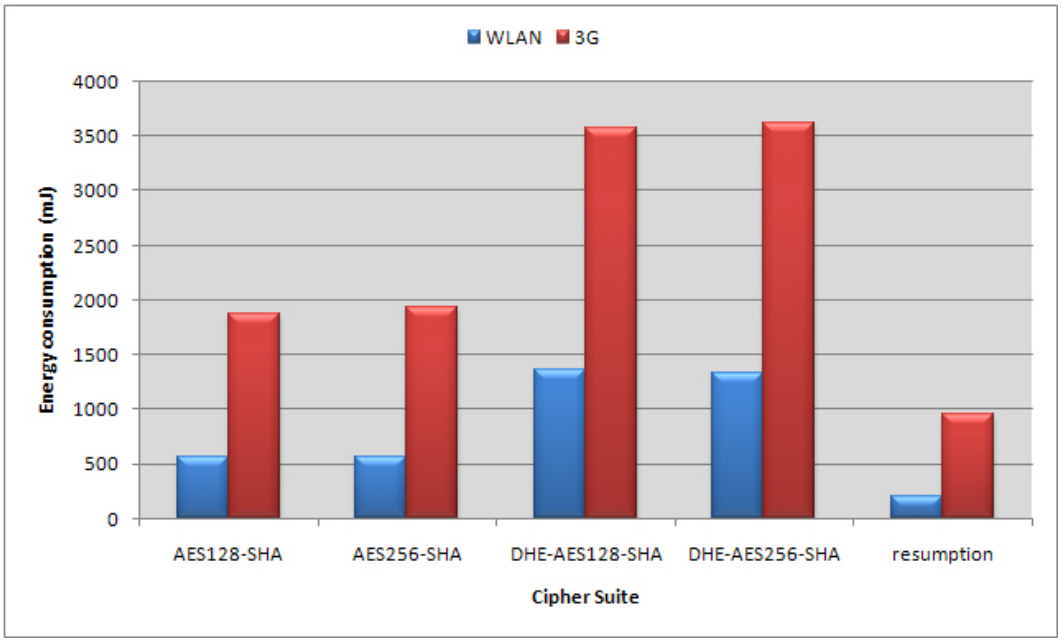


Figure 5.8: Comparing Energy Consumption using WLAN and 3G

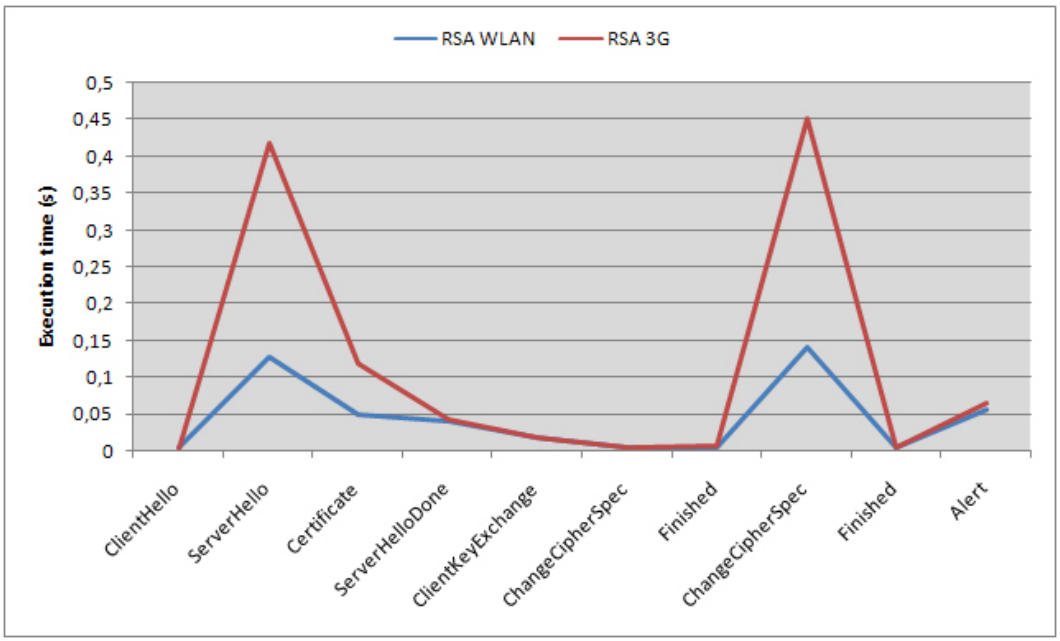


Figure 5.9: Comparing Handshake steps Execution time using RSA for WLAN and 3G

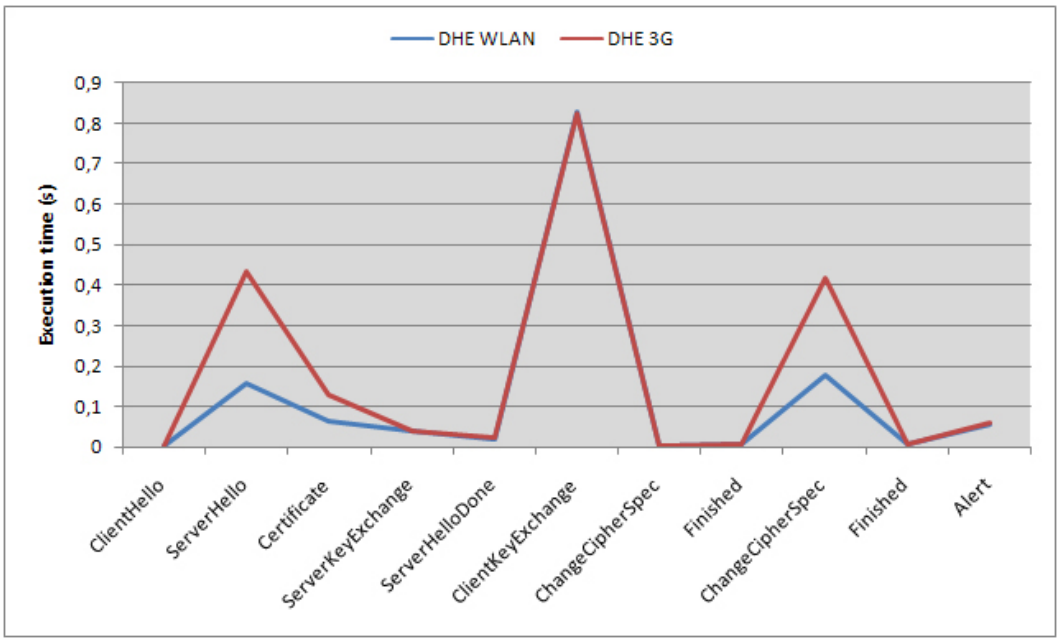


Figure 5.10: Comparing Handshake steps Execution time using DHE for WLAN and 3G

- In the other SSL handshake steps, there is no difference in using 3G or WLAN.

5.3 Test scenario: Overhead of SSL usage

In order to measure the communication data overhead incurred in a SSL Handshake, the next experiments have been done. A different number of SSL connections have been carried out using different transaction sizes (the length of the data to exchange for each case) to measure the impact of the creation of a secure channel.

5.3.1 SSL Data Overhead

Table 5.11 shows the experiments performed and the results. The first column shows the transaction size in bytes used in each experiment, ranging from 1 KB to 5MB. The next one shows in which way the data is sent. C->S means from client to server and S->C, the other way around. Next colum shows the exchanged data only in the handshake protocol. Total RSA and Total DHE columns show the total amount of data exchanged

$$Data = Handshake\_protocol + Record\_Protocol$$

It is important to mention that in the Record protocol is not only included the plain data or transaction size, extra data exist related with the SSL protocol, like the hash HMAC-SHA1, the pad and the pad length used in the encryption of the record block sent, that increase the total data exchanged. The Overhead RSA and DHE columns apply for the real SSL overhead :

$$SSL\_Overhead = Data - Transaction\_size$$

And the last two columns, Overhead RSA and DHE, show the percentage of Overhead compared with the transaction size :

$$Overhead(\%) = \frac{SSL\_Overhead(bytes)}{Transaction\_size(bytes)} \times 100$$

Figure 5.11 shows the percentage of communication data overhead involved in the SSL handshake compared with different transaction sizes.

- As the data sent by the server is 2384 bytes (depending on the certificate and if DHE or RSA is used, DHE increases the amount of data in 402 bytes), for the first case the percentage overhead is about 300%, the transaction size is only 1024 bytes while the total amount of exchanged data is 3125 bytes.

Transaction size(bytes)	Direction	Handshake	Total RSA(bytes)	Total DHE(bytes)	Overhead RSA(bytes)	Overhead DHE(bytes)	Overhead RSA(%)	Overhead DHE(%)
1024	C->S S->C	263 2384	1361 2384	1363 2786	2721	3125	**265,72**	**305,18**
10240	C->S S->C	263 2384	10910 2384	10912 2786	3054	3458	**29,82**	**33,77**
51200	C->S S->C	263 2384	53350 2384	53352 2786	4534	4938	**8,86**	**9,64**
102400	C->S S->C	263 2384	106400 2384	106402 2786	6384	6788	**6,23**	**6,63**
512000	C->S S->C	263 2384	530800 2384	530802 2786	21184	21588	**4,14**	**4,22**
1024000	C->S S->C	263 2384	1061300 2384	1061302 2786	39684	40088	**3,88**	**3,91**
5120000	C->S S->C	263 2384	5305300 2384	5305302 2786	187684	188088	**3,67**	**3,67**

Table 5.11: Profile of the data exchanged between the client and server in an SSL connection

- The overhead is decreasing as the transaction size is increasing. While for 1KB the overhead is 265% or 305% (depending on RSA or DHE), for 5MB the overhead is only 3% for both cases.

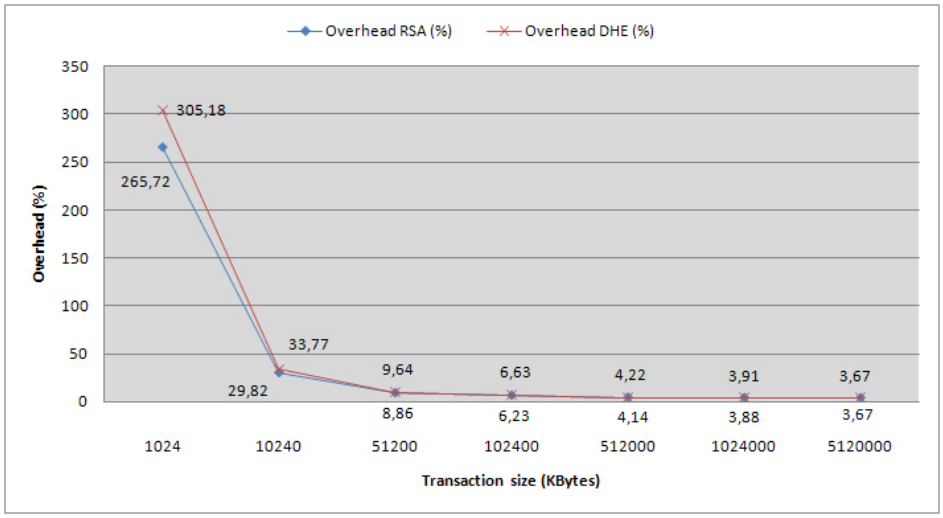


Figure 5.11: Percentage of communication data overhead of SSL usage for different transaction sizes

- The difference between RSA and DHE is only appreciable for small transaction sizes: 1KB and 10KB.

5.3.2 SSL Throughput

This section is intended to analyze the throughput of the SSL protocol compared with the use of a non-secure TCP channel. In order to measure the performance of the connections, several experiments using different transaction sizes have been tested.

Table 5.12 shows the performance of the communications. The first column indicates the transaction size. The second column shows the number of connections done in order to get accuracy results. For the first cases, where small transactions are used, the number of connections starts in 300 and is decreasing as the transaction size is increasing. The reason behind this issue was explained in Section 4.5.1, and it is related with the measurement interval of the Nokia Energy Profiler, in order to achieve good accuracy in the metrics. The third and fourth column show the total execution time of the different experiments using or not SSL. This is the time used for the total number of connections defined before. The fifth and sixth column represent, in bold, the actual throughput with SSL or without it. The last two columns show the microprocessor average percentage of the whole process. These values will help to understand some issues about the throughput with large transaction sizes.

The throughput is obtained using the following formula, multiplying the transaction size by the number of connections and dividing these results by the total execution time :

$$Throughput(MB/s) = \frac{Transaction\_size \times Number\_of\_connections}{Total\_execution\_time \times 10^6}$$

As the throughput is in MB/s, the results has to be divided by 10^6 as the transaction size is in bytes.

Transaction size (bytes)	Number of connections	Total execution time with SSL(s)	Total execution time without SSL (s)	Throughput with SSL (MB/s)	Throughput without SSL (MB/s)	% CPU with SSL	% CPU without SSL
1024	300	20,95	5,38	**0,015**	**0,057**	85	90
10240	200	17,42	8,75	**0,118**	**0,234**	90	95
51200	150	28,66	18,08	**0,268**	**0,425**	96	98
102400	100	32,42	23,54	**0,316**	**0,435**	98	98
512000	30	48,55	31,71	**0,348**	**0,484**	**100**	98
1024000	10	26,48	14,74	**0,387**	**0,695**	**100**	96
5120000	5	63,38	39,74	**0,404**	**0,644**	**100**	97

Table 5.12: SSL Throughput comparing WLAN and 3G

Figure 5.12 shows the results and help to understand the results. As the data transaction size is increased, also does the throughput. This is due to,

as the data increase, the number of connections decreases, and thus less time involved in establishing new connections.

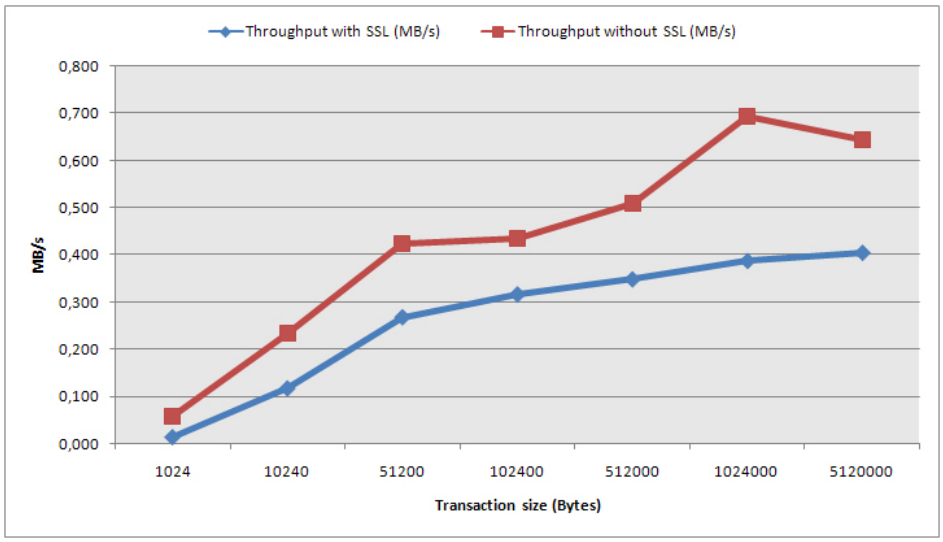


Figure 5.12: Comparing the throughput of using SSL and non secure connection

But when 512KB or higher transaction sizes are used, the results show, when using SSL, that the CPU is running at full capacity, achieving 100% of usage and limiting the speed in the transactions. Of course this issue is due to the processor included in the tested device, a Dual ARM 11 332 MHz processor, which is not capable to performing all the cryptographic operations in time.

5.3.3 SSL Energy Consumption Overhead

While in the previous section the throughput was measured, this section is intended to measure the energy consumption overhead dissipated due to the use of SSL. The experiments have been performed taking into account the use of the two communications interfaces, WLAN and 3G.

Table 5.13 shows the results measured for the different transaction sizes. The values for a non-secure connection using WLAN and 3G are presented in columns 1 and 3, respectively. The energy consumption for exchanging data using SSL are shown in columns 2 and 4, for the WLAN and 3G case respectively. All data is presented in miliJoules (mJ).

	WLAN		**3G**	
Transaction size	**No SSL**	**SSL**	**No SSL**	**SSL**
1024	27,27	105,18	300,02	1808,64
10240	70,58	137,73	604,14	1514,18
51200	197,83	311,20	2343,75	3036,92
102400	376,42	538,11	4163,98	5897,77
512000	1729,56	2744,32	25910,00	33530,19
1024000	2481,28	4432,30	37528,33	44658,50
5120000	13143,48	20914,56	155603,85	161278,36

Table 5.13: Energy consumption using WLAN or 3G

Figure 5.13 and 5.14 represents the obtained results graphically. In previous results, as the transaction size is increased, the overhead decreases. Theoretically, the energy overhead using SSL should decrease as the data transaction is increased. The experiments using 3G show this decrease but not in the same way for the WLAN case. This is related also with the CPU limiting factor commented in the previous section, where the mobile device cannot complete in time all the cryptographic operations and a bottleneck exists. This issue is due to a latency factor. Using WLAN there is no latency involved and the responses from the server arrive with no delay, where in 3G important latency times higher than 200 ms give enough time to the mobile client to compute the next packet to send.

Figure 5.15 shows the values from the table in a logarithmic scale for the Y-axis, showing that as the transaction size is increased, in the case of 3G the energy consumption difference between using SSL and a non secure connection decreases, where for the WLAN case this difference remains constant. These results confirm the commented aspects in Section 2.6.1, where the

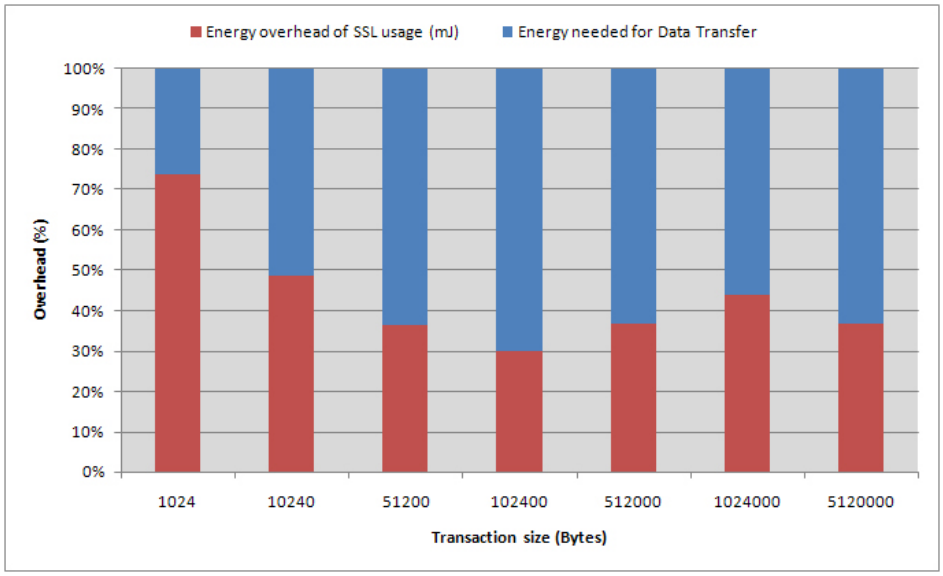


Figure 5.13: Energy overhead of SSL usage using WLAN

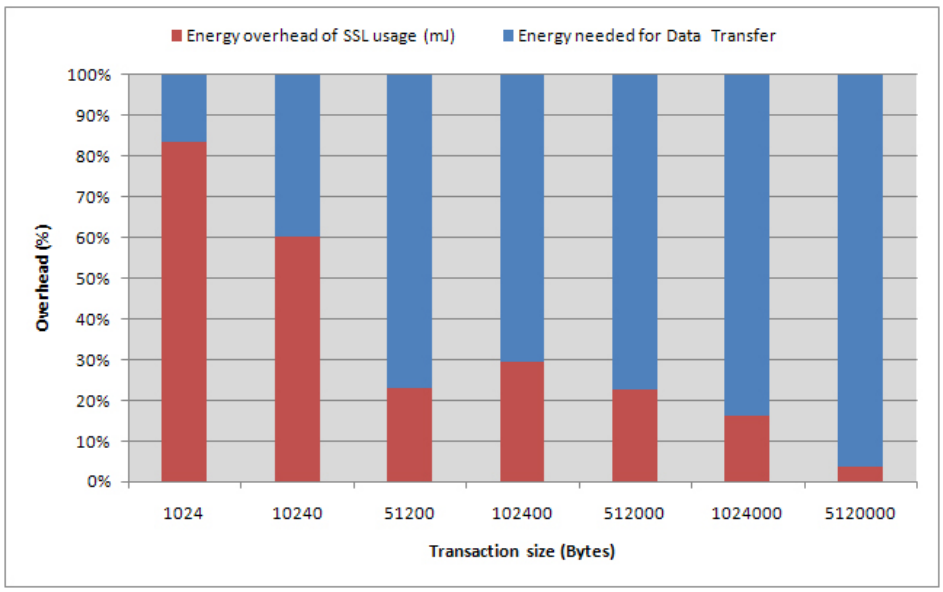


Figure 5.14: Energy overhead of SSL usage using 3G

WLAN interface has good energy metrics for long data transmissions, and better results than the 3G cases.

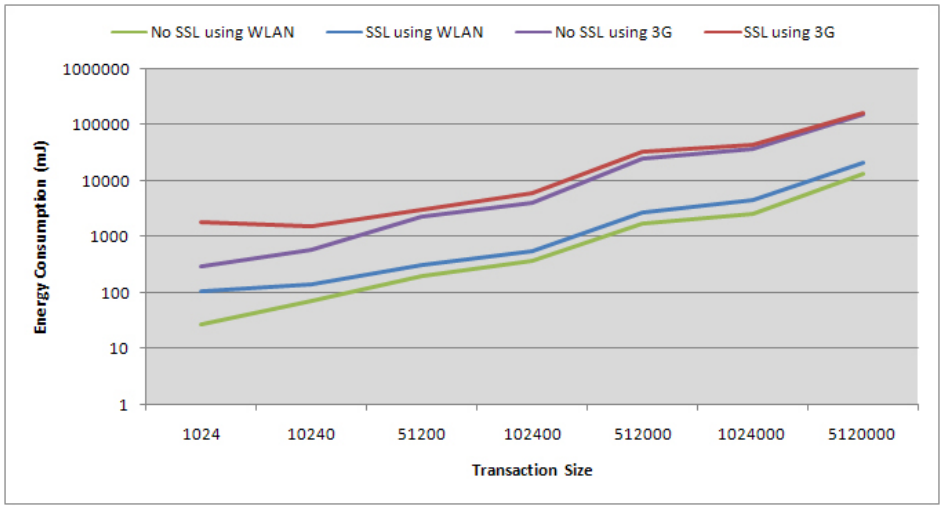


Figure 5.15: Comparing energy consumption overhead using WLAN or 3G

Chapter 6

Conclusions

This chapter includes an overview of all the obtained conclusions along the previous chapters, as well as the possible future work to continue this research project.

6.1 Summary

During the previous sections, an experimental setup to study and analyze the different cryptographic algorithms and security protocols has been presented. This setup does not require any external measurement device. It uses the Nokia Energy Protocol, a software running in the mobile device, capable of measuring energy consumption and other parameter like CPU usage or signal levels with accuracy. Different cryptographic algorithms and security protocols like AES, HMAC, RSA or SSL have been measured, but any new selected algorithm or protocol could also be measured using the same setup.

In *Chapter 5 - Analysis of Measurements*, all the results obtained were presented, including all the conclusions. Thus, here only the most important aspects are remarked:

1. The use of the Secure Socket Layer protocol increases the overhead of the communication, incurring into a higher energy consumption, execution time and extra data exchanged. This overhead decreases as the transaction size increases.

2. Using RSA as a Public key algorithm for the key exchange has a great performance in comparison with Diffie-Hellman.

3. Diffie-Hellman provides better security properties than RSA, but the energy consumption is substantially higher than RSA. The use of RSA is preferable from the perspective of optimizing costs at the client side.

4. The use of session resumption, where possible, reduces considerably the energy consumption and the execution time, as no public cryptography is needed.

5. The critical factor that affects the energy consumption in an SSL handshake is the certificate's length and the server's public key, along with the selection of the key exchange algorithm.

6. The use of 3G as the communication interface highly increases the energy consumption, compared with WLAN.

6.2 Future work

The objective of this section is to study the possibility to continue this research project, widening the tested cipher suites and cryptographic algorithms, studying new security techniques to apply reusing the same experimental setup.

Use of Elliptic Curve Cryptography

Public key cryptography is mainly founded on hard computational problems. RSA is based on the difficulty to factorize integers, while Diffie-Hellman is based on the discrete logarithm problem in integer fields. The study of Elliptic Curve Cryptography (ECC) provides security based on this discrete logarithm problem but defined on elliptic curves. These algorithms can provide the same level of security as RSA or DH but using shorter keys. The equivalences are presented in Table 6.1.

Symmetric	**ECC**	**DH/RSA**
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Table 6.1: Security equivalences (key length in bits) between symmetric, elliptic curve and public key cryptography

Using ECC for public key cryptography can greatly reduce the energy consumption as well to increase security, as a 163-bit ECDH key provides the same security level as a 1024-bit RSA key.

Although there are different implementations, OpenSSL toolkit already implements Elliptic Curve Cryptography, the use of ECC is being studied against different cryptanalytic attacks and it is not supported yet by the main Internet providers.

Bibliography

[3GPa] 3GPP. General report on the design, speification and evaluation of 3gpp standard confidentiality and integrity algorithms. http://www.3gpp.org/ftp/tsg_sa/WG3_Security/_-Specs/33908-300.pdf.

[3GPb] 3GPP. Specification of the 3gpp confidentiality and integrity algorithms uea2 & uia2. document 2: Snow 3g specification the snow. http://www.gsmworld.com/documents/snow_3g_-spec.pdf.

[3GP06] 3GPP. Specification of the 3gpp confidentiality and integrity algorithms uea2 & uia2. document 1: Uea2 & uia2 specifications. http://www.gsmworld.com/documents/etsi_sage_06_-09_06.pdf, 2006.

[ABK99] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard. http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf, 1999.

[APS99] George Apostolopoulos, Vinod Peris, and Debanjan Saha. Transport layer security: How much does it really cost. In *In Proceedings of the IEEE INFOCOM*, pages 717–725, 1999.

[BBV09] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293, New York, NY, USA, 2009. ACM.

[CG97] P. Cheng and R. Glenn. Test cases for hmac-md5 and hmac-sha-1, 1997.

[Cho02] P. Chown. Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS). RFC 3268 (Proposed Standard), 2002. Obsoleted by RFC 5246.

[Cor94] Netscape Communications Corp. Ssl 2.0 protocol specification. http://www.mozilla.org/projects/security/pki/nss/ssl/draft02.html, 1994.

[Cor96] Netscape Communications Corp. Ssl 3.0 protocol specification. `http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00`, 1996.

[DA99] T. Dierks and C. Allen. RFC 2246: The TLS protocol version 1, 1999.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf, 1999.

[Dwo01] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation - Methods and Techniques. NIST Special Publication 800-38A.* 2001. Available from http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[Fit07] Frank Fitzek. External energy consumption measurements on mobile phones. In *Mobile Phone Programming and its Application to Wireless Networking*, pages 441–447. Springer, 2007.

[IBM99] IBM. Mars - a candidate cipher for aes. http://www.research.ibm.com/security/mars.pdf, 1999.

[Kau99] Thayer Kaukonen. A stream cipher encryption algorithm "arcfour". http://www.mozilla.org/projects/security/pki/nss/draft-kaukonen-cipher-arcfour-03.txt, 1999.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. Hmac: Keyed-hashing for message authentication, 1997.

[Lab93] RSA Laboratories. Pkcs 6: Extended-certificate syntax standard. http://www.rsa.com/rsalabs/node.asp?id=2128, 1993.

[Ope10] *OpenSSL project*, 2010. Available at http://www.openssl.org.

[oST99] National Institute of Standards and Technology. Data encryption standard (des). http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, 1999.

[oST01] National Institute of Standards and Technology. *Specification for the ADVANCED ENCRYPTION STANDARD (AES)*, 2001. Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[oST06] National Institute of Standards and Technology. *FIPS PUB 186-3: Digital signature standard (DSS)*, 2006. Available at http://csrc.nist.gov/publications/drafts/fips_186-3/Draft-FIPS-186-3%20_March2006. pdf.

[PP06] Charles P. Pfleeger and Shari Lawrence Pfleeger. *Security in Computing (4th Edition)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2006.

[PRRJ06] Nachiketh R. Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, 5(2):128–143, 2006.

[Res01] Eric Rescorla. *SSL and TLS: Designing and building Secure Systems*. Addison-Wesley, 2001.

[Riv92] R. Rivest. The md5 message-digest algorithm, 1992.

[RRSY99] Rob Rivest, Matt Robshaw, Ray Sidney, and Yiqun Lisa Yin. The rc6 block cipher, algorithm specification. ftp://ftp.rsasecurity.com/pub/rsalabs/rc6/rc6v11.pdf, 1999.

[RSA79] R.L. Rivest, A. Shamir, and L.M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. 1979.

[Sch95] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[Sch99] Whiting Wagner Hall Ferguson Schneier, Kelsey. Twofish: A 128-bit block cipher. http://www.schneier.com/paper-twofish-paper.pdf, 1999.

[SGM09] Youngsang Shin, Minaxi Gupta, and Steven Myers. A study of the performance of ssl on pdas. In *INFOCOM'09: Proceedings of the 28th IEEE international conference on Computer Communications Workshops*, pages 1–6, Piscataway, NJ, USA, 2009. IEEE Press.

[Sha05] Y. Shafranovich. Common format and mime type for comma-separated values (csv) files, 2005.

[WH76] Diffie Whitfield and Martin E. Hellman. New directions in cryptography, 1976.

[WVOW92] Diffie Whitfield, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges, 1992.

Appendices

Appendix A

Study of Snow 3G

This appendix involves the experiments and research performed about the the UEA2 and UIA2 algorithms, using as core the snow 3G stream cipher algorithm.

A.1 SNOW3G

Snow 3G [3GPb] is a stream cipher that forms the core of the 3GPP confidentiality algorithm UEA2 and the 3GPP integrity algorithm UIA2 [3GP06], used to offer security services in Universal Mobile Telecommunication System (UMTS). These two algorithms are a replacement of the current used security algorithms, UIA1 and UEA1 based on the KASUMI [3GPa] block cipher. No weaknesses have been found in these algorithms, but the 3GPP wishes to specify a second set of algorithms, in order to have a fast replacement if any vulnerabilities ever were found.

The main design criteria was, apart from the obvious on speed and implementation complexity, that they should be founded on different cryptographic principles from UEA1 and UEA1, in order to avoid that possible vulnerabilities affects also to the new ones. AES was between one of the candidates for the encryption algorithm, as it is proved that offers good performance with an easy implementation, but was discarded because it is based on the same cryptographic properties as KASUMI. Algorithms like HMAC-SHA-256 or CBC-MAC were also candidates for the integrity algorithm, but they were also discarded in favor of a GMAC construction with some additional steps to provide better security properties.

The snow 3G is a stream cipher that generates a sequence of 32-bit words

using the control of a 128-bit key and a 128-bit initialization variable. First the cipher operates in key initialization mode, where an initialization of all parameters is performed and the cipher is clocked without producing any output. After this initialization, the cipher operates in key-generation mode and it produces a 32-bit ciphertext/plaintext output each time that is clocked.

The UEA2 and UIA2 algorithms uses the snow 3G implementation as the core of the operations.

- The confidentiality algorithm UEA2 is a stream cipher that is used to encrypt/decrypt blocks of data under a 128-bit confidentiality key CK.

- The integrity algorithm UIA2 computes a 32-bit MAC (Message Authentication Code) of a given input message using a 128-bit integrity key IK.

The input in both algorithms must be between 1 and 20000 bits in length. For a detailed information about how the operations are performed, please refer to [3GP06].

A.1.1 Experiments

Even there are different hardware implementations of Snow 3G algorithm in the market[1] ,[2], the software implementation of SNOW3G as well as UIA2 and UEA2 can be found in the documentation mentioned above. For these experiments, an internal software implementation from Nokia Corporation has been used. Also in annex 2 from [3GP06] is explained the possibility to accelerate the whole process using precalculated table lookups. This possibility can greatly reduce the execution time, and as a consequence, energy savings. But it has also drawbacks, like the increase of the needed space. This issue is important as the implementation is going to be allocated is a constrained memory device.

Thus, two different implementations modes are available:

- FAST_SNOW 3G Disabled consist on the original implementation, without any improvement.

- FAST_SNOW 3G Enabled using the precalculated table lookups.

[1] CLP-41: SNOW 3G Cipher Core - http://www.ellipticsemi.com/products-clp-41.php
[2] SNOW 3G Encryption Core - http://www.ipcores.com/Snow3G.htm

Operation	Time	Energy Consumption
SNOW3G	6,488 s	3,06 $\mu J/B$
UEA2	21,477 s	9,41 $\mu J/B$
UIA2	22,111 s	10,02 $\mu J/B$

Table A.1: Execution time and energy consumption with FAST SNOW 3G disabled

Operation	Time	Energy Consumption
SNOW3G	1,313 s	0,52 $\mu J/B$
UEA2	0,497 s	0,259 $\mu J/B$
UIA2	21,885 s	10,02 $\mu J/B$

Table A.2: Execution time and energy consumption with FAST SNOW 3G enabled

The experimental setup for the experiments changes with respect the one designed in Section 4.5.3, as no connection is required with the server, and only cryptographic operations are performed in the client. For the energy consumption measurement, Nokia Energy Profiler has been also used, in the same way as explained in Section 4.5.2.

In order to study the speed and the energy consumption of the algorithms, both modes have been measured with the different results in Tables A.1 and A.2. This values are the average measurement of different experiments.

Using the *FAST_MODE* implementation the performance is greatly improved, achieving for the case of UEA2 a reduction of 43 times the execution time and 36 times the energy consumption.

The use of *FAST_MODE* does not affect is such a way to UIA2, where the performance is not even altered. The execution time is only reduced in 225ms, but with the same energy consumption. Taking a look at the implementation, there is a reason for that. Table A.3 shows the results of the execution time measurements for the whole process, reading data from the file and Hash the input with the UIA2 algorithm, and only for the main loop of the algorithm described here.

```
for(i=0;i<D-2;i++)
    {
      bytes2uint64(&tmp,MESSAGE);
      EVAL=MUL64(EVAL^tmp,P,0x1b);
      MESSAGE+=8;
```

```
}
```

Operation	FAST DISABLED	FAST ENABLED
Loop	46,744 ms	46,6 ms
Read & Hash	49,049 ms	48,417 ms

Table A.3: Execution time for the UIA2 loop

In the loop, the $EVAL = MUL64(EVAL^tmp, P, 0x1b)$; line takes the most of the time. It is a recursive operation, the other two are just simple easy calculations or transformations that takes less than a milisecond. But this MUL64 operation is not using the FAST_SNOW 3G.

Figures A.1 and A.2 show the experiment results, including the energy consumption in white and the usage of the CPU in purple.

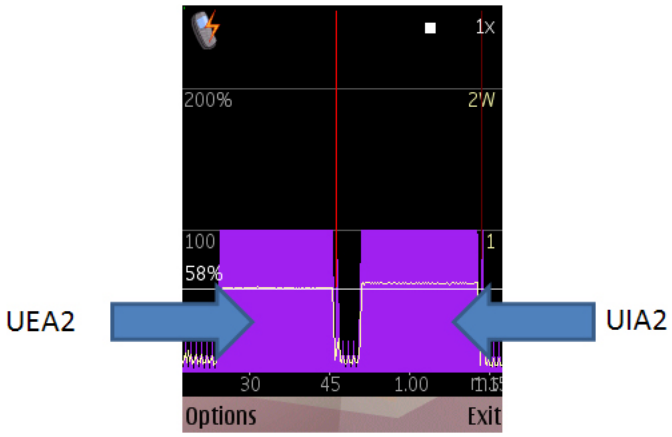


Figure A.1: Energy consumption and CPU usage of UEA2 and UIA2 with FAST_SNOW 3G Disabled

As conclusion, using FAST_SNOW 3G greatly reduces the execution time and the energy consumption of the UEA2 algorithm, but not in such a way for the UIA2 algorithm.

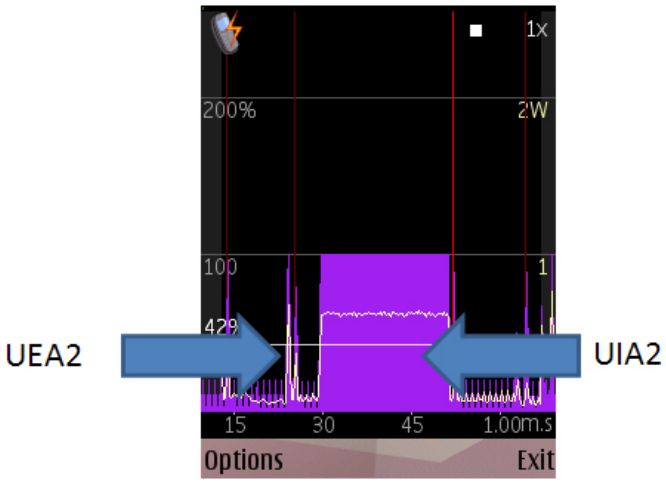


Figure A.2: Energy consumption and CPU usage of UEA2 and UIA2 with FAST_SNOW 3G Enabled

Appendix B

Study of Symmetric and Hash algorithms

To measure the energy consumption of various cryptographic, a number of experiments have been performed using different algorithms and their parameters. The report contains the results with corresponding details about the experiment setup and used algorithms. All the background process has been closed except the ones run by Symbian OS itself for its proper working. The experiments have been performed to read a text files of size 1.1 Mbytes (big) or 227 Kbytes (small) in to a buffer of size equal to AES block size of 128bits and perform the cryptographic operations on them. The result of operation is then stored back to another file at a defined location. The buffer is rewritten again with new contents and this process repeats itself until complete file has been read. The power consumption measurements are taken by the Nokia Energy Profiler. The Energy consumption is calculated with the given formula

$$1W = 1J/sec$$

B.1 Summary and Experiments results

The summary of the result obtained from number of various experiments is shown in Figure B.1. The figure shows the energy consumption of various cryptographic algorithms and the variation in energy consumption as their parameters are changed.

In the results of Figure B.1, the file of size 1.1 Mbytes is used to perform cryptographic operations and result is stored back at the defined location in

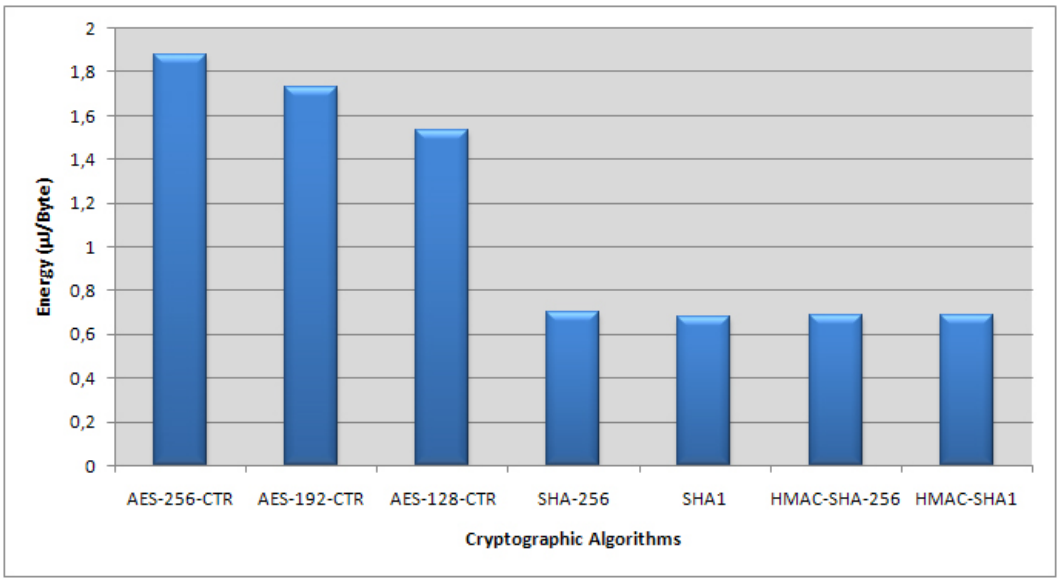


Figure B.1: Energy consumption of various algorithms with different parameters for a file of size 1.1 Mbytes stored in phone memory.

the phone memory. In the section B.1.7, we use the same file but stored it in the memory card, the detailed results are shown in section B.1.7. To get the energy consumption of a encryption operation using AES-256-CTR irrespective of the file location as where it will be stored, we conducted another experiment in which file has been read from the phone memory and the encryption is performed on it. The result has not been stored back to the file to get the energy values for read and cryptographic operations. This will provide us a value which is irrespective of the location where file has to be stored, whether local or on network. The result are shown in the table below.

Cryptographic Operation	**Energy (μJ)/Byte**	**Standard deviation**
Encryption (AES-256-CTR)	1,24	0,05
MAC (HMAC-SHA256)	0,49	0,03

To compute the total cost of the operation, when file has to be stored at the memory card, we need the energy values for the write operation. For write only operation for the memory card, we subtract the result obtained in read & write (memory card) from read only (memory card) in section B.1.1 and result is 1.71 (μJ/Byte). The energy consumed in reading, encrypting and storing back to the memory card is given in section B.1.7 as 3.83 (μJ/Byte).

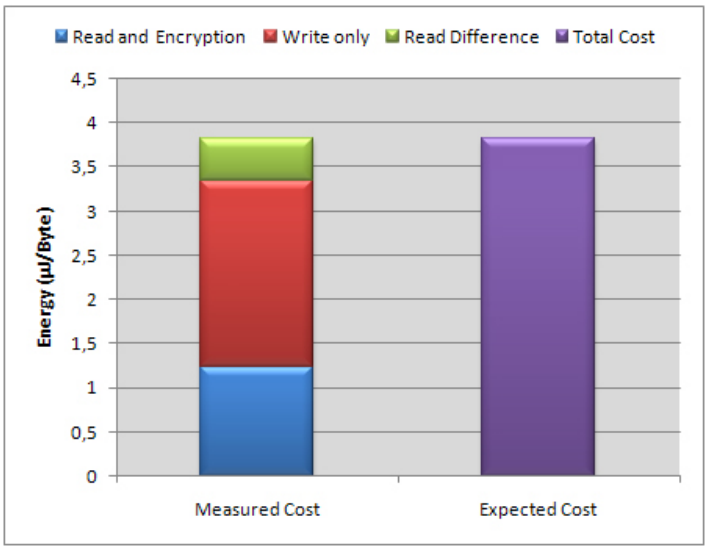


Figure B.2: The result from encryption operation of a file read & stored at the memory location and operation performed in steps as read, encrypt and added write cost.

The result obtained from reading and encrypting only is 1.24 (μJ/Byte), the significant difference in the result of section [9] and the result from table above + result of section B.1.1, is due to the fact the in this experiment, the read has been performed from the phone memory. Therefore, to get the correct result, we need to subtract the read only (phone memory) cost from read & write (memory card), the result obtained is 2.11 (μJ/Byte). To get the aggregate number from combining reading- encrypting and writing a file to memory card, we get the result as 3.35 (μJ/Byte). The result is shown in the Figure [1].

The difference in the expected cost and the measured cost is due to the fact as file has been read from the phone memory where as in expected cost, the file has been read from the memory card. The difference between reading from both location is 0.41 (μJ/Byte) (section B.1.1), which is the actual difference between expected and measured cost.

The energy cost variation due to the variation key size is given below.

Difference in Key Length	**Difference in AES Energy (μJ)/Byte**
256-192	0,15
192-128	0,2
256-128	0,35

Operation	Energy (μJ)/Byte	Standard deviation
Read Only (Phone Memory)	0,58	0,03
Read & Write (Phone Memory)	1,34	0,04
Read Only (Memory Card)	1,19	0,06
Read & Write (Memory Card)	2,9	0,02

We can see from the Figure [1], this difference in energy consumption due to different key length is quite visible in figure. For the difference in energy due to file size is 0.09 (μJ/Byte), the result is obtained from difference in energy consumption in section B.1.2 & B.1.4. This difference is due to opening and closing of the file for a number of 5 times, otherwise there should not be any difference with respect to file size.

From the result obtained in section B.1.2 & B.1.3, we have extracted the energy consumption values to get the cost of setting up the key of 256 bit size. For 128 bit key setup cost, another experiment has been conducted and the result obtained is given in Table below.

Key Setup	Energy (J)
AES-256-CTR	0,25
AES-128-CTR	0,2

The result is calculated as subtracting the cost of section B.1.3 from section B.1.2 and also subtract the cost incurred due to opening and closing the file, which is 0.09 (μJ/Byte). The small difference in key setup is due to the number of rounds needed to setup the key. For the 256-bit key length, the number of required rounds is 14 and for 128-bit the number of rounds is 10.

B.1.1 Read only experiment

The experiment has been performed to read a text file of size 227 Kbytes in to a buffer of size equal to AES block size of 128bits. The buffer is rewritten again with new contents and this process repeats itself until complete file has been read.

The given values are for the read or read/write contents of the complete file, which is stored in the phone memory. The experiment is also performed to read and read/write the same file, when it is stored in the memory card instead of the phone memory. The Energy values per byte for both the above operations is shown in Table.

The energy consumption in read or read/write operation is quite high, when a file is stored in the memory card compared to the case when it is stored in the phone memory.

B.1.2 Cryptographic operations using a large file

In this experiment, the file of size 1.1 Mbytes has been used. The file is stored in the phone memory. The power consumption has been measured using NEP. The energy consumption for this experiment is given in table below.

Cryptographic Operation	Energy (μJ)/Byte	Standard deviation
Encryption (AES-256-CTR)	1,88	0,03
Decryption (AES-256-CTR)	1,88	0,04
Hash (SHA-256)	0,69	0,03
MAC (HMAC-SHA-256)	0,66	0,02
Authenticate MAC (HMAC-SHA-256)	0,67	0,03

The energy values for the encryption/decryption operation is comparable to the previous experiments. However, the energy consumption for hash and HMAC is decreased from the previous experiments. After careful examining the previous results, the reason for it was that while calculating the values from previous experiments, the time spent in executing the required cryptographic operations was not properly taken in to account. I used the same time for hash and MAC as was for AES encryption/decryption. But this time, I was careful in this and reading from the current experiments reflect the actual values.

B.1.3 Cryptographic operations using a small file

In this experiment, a file of size 227Kbytes is used and each cryptographic operation is performed 5 times with new key setup for each run. The energy consumption for this experiment is shown in table below.

From the above readings, we can see that the file I/O and setting up key for each file cause the extra overhead in the encryption/decryption operations, while for Hash and HMAC there is not much significant difference. This is due to minimal cost in setting up the context for these cryptographic operations.

Cryptographic Operation	Energy (μJ)/Byte	Standard deviation
Encryption (AES-256-CTR)	2,24	0,02
Decryption (AES-256-CTR)	2,23	0,03
Hash (SHA-256)	0,7	0,02
MAC (HMAC-SHA-256)	0,69	0,02
Authenticate MAC (HMAC-SHA-256)	0,68	0,03

B.1.4 Cryptographic operations using small files with the same key

In this experiment, a file of size 227Kbytes is used and each cryptographic operation is performed 5 times with same key. Therefore no key setup cost is present in these values. The energy consumption for this experiment is shown in table below. The experiment is performed only on AES encryption/decryption operations. This is because the other operations, require setting up context every time, as they delete all the context key informations, once they finalised the result for one run.

Cryptographic Operation	Energy (μJ)/Byte	Standard deviation
Encryption (AES-256-CTR)	1,97	0,02
Decryption (AES-256-CTR)	1,96	0,06

The values from the above table shows that the energy consumption in this case is comparable to the large file case, as the small extra amount of energy is used in closing and opening the same file again and again.

B.1.5 Cryptographic operations using a large file with AES-192 & SHA1

In this experiment, the file of size 1.1 MB has been used. In this experiment, the key size of AES has been reduced to 192 bits from 256 bits. The hash function used in calculating hash and MAC is changed from SHA256 to SHA1. The energy consumption for complete experiment is shown in table below.

From the above values, we can see that the small key size and SHA1 consumes less energy compared to 256 bit AES and SHA2. The reason behind this that number of rounds to carry out the encryption / decryption is reduced to 12 from 14 for each block, whereas the SHA1 is less expensive in computing terms compared to SHA256.

Cryptographic Operation	Energy (μJ)/Byte	Standard deviation
Encryption (AES-192-CTR)	1,73	0,01
Decryption (AES-192-CTR)	1,75	0,03
Hash (SHA1)	0,68	0,02
MAC (HMAC-SHA1)	0,69	0,02
Authenticate MAC (HMAC-SHA1)	0,7	0,02

B.1.6 Cryptographic operations using a large file with AES-128 & SHA1

In this experiment, the file of size 1.1 Mbytes has been used. In this experiment, the key size of AES has been further reduced to 128 bits. The hash function used in calculating hash and MAC is changed from SHA256 to SHA1. The energy consumption for complete experiment is shown in table below.

Cryptographic Operation	Energy (μJ)/Byte	Standard deviation
Encryption (AES-128-CTR)	1,53	0,02
Decryption (AES-128-CTR)	1,56	0,04
Hash (SHA1)	0,67	0,02
MAC (HMAC-SHA1)	0,67	0,03
Authenticate MAC (HMAC-SHA1)	0,69	0,03

From the above values, we can see that the energy consumption of encryption/decryption is further reduced due to smaller key size of 128 bits. The number of rounds to carry out the operation is also reduced to 10. The energy consumption of the hash and MAC is similar to the above test experiment, as the same SHA1 function is used in this experiment too.

B.1.7 Cryptographic operations using a large file in a memory card

In this experiment, the file of size 1.1 Mbytes has been used. In this experiment, the file has been stored in the memory card instead of the phone memory. The AES key size 256 bits and SHA-256 is used to calculate hash and MAC. The energy consumption for complete experiment is shown in table below.

Cryptographic Operation	**Energy (μJ)/Byte**	**Standard deviation**
Encryption (AES-256-CTR)	3,83	0,07
Decryption (AES-256-CTR)	3,85	0,09
Hash (SHA256)	1,34	0,03
MAC (HMAC-SHA256)	1,2	0,04
Authenticate MAC (HMAC-SHA256)	1,23	0,03

From the above readings, we can see that energy consumption is quite high compared to the case when file is stored in the phone memory. But if we take a look on the read or read/write experiment results, the energy consumed in read/write operation is quite high too compared to case, when file is stored in phone memory. If we subtract the amount of read/write from both the cases, then the result of energy consumption in cryptographic operations in both the cases is almost similar irrespective of the file storage location.

Anexo D. Evaluación

Como se ha comentado previamente en la introducción, el desarrollo de este proyecto tuvo lugar en la Aalto University School of Science and Technology. Allí fue defendido y presentado ante un tribunal obteniendo la calificación de Excelente. Se presenta pues a continuación dicha evaluación.

Department of Computer Science and Engineering
PO Box 15400
(Konemiehentie 2)
00076 Aalto

Aalto University
School of Science
and Technology

Final Project Evaluation Statement

Title: Energy consumption of cryptographic algorithms and security protocols in Symbian mobile devices

Author: Pedro Miranda
Supervisor: Prof Antti Ylä-Jääski, Aalto University School of Science and Technology
Instructors: Dr. Matti Siekkinen, Aalto University School of Science and Technology
Dr. Heikki Waris, Nokia Research Center

This final project focuses on the cost of security in terms of energy consumption. Indeed, communication security does not come for free as the security protocols, such as SSL, include cryptographic computations and adding of extra information into the transferred data e.g. to ensure integrity of the original data. These operations consume extra resources, namely CPU capacity and bandwidth, compared to non-secure communication. While smart phones today have quite powerful CPUs and increasingly high capacity wireless network interfaces, this overhead eventually manifests itself as additional energy consumed which is a scarce resource in smart phones today. In this project, SSL protocol is broken down into components and the energy consumption of each phase is measured and analysed. The experiments have been done using Symbian-based phone and many different scenarios have been considered including comparison of WLAN vs. 3G access, comparison of different cryptographic suites, and impact of establishing an SSL session with different popular service providers such as Facebook, Google, etc.

In the Introduction, the author presents and motivates the problem, and describes the structure of the final project report. The next two Chapters describe the relevant background information concerning security protocols and energy consumption. Chapter 4 presents the experimentation setup including the sofware and hardware used and the design of the various scenarios. Chapter 5 focuses on the analysis of obtained measurements. In Chapter 7, conclusions are drawn and future work outlined.

Energy consumption of mobile devices is a major concern today. Likewise, communication security remains as a very important area of research and development. For these reasons, the topic of the project is very timely and the problem statement highly relevant. Language and presentation of the work is very good. Background is very well explored. The results and methods are excellent, clearly above the level normally expected from a final project work. The relevance of the project is significant to the research community and a scientific paper will be prepared and submitted for publication based on the results.

Definition of the research scope and goals: 5
Command of the topic: 4
Methods and conclusions: 5
Contribution to knowledge and report structure: 5
Presentation and language: 4

We recommend that this final project is approved with the grade **excellent** (5).

Espoo, June 3, 2010

Prof. Antti Ylä-Jääski
Project supervisor

Dr. Matti Siekkinen
Project instructor