COMPARATIVA DE TARJETAS JAVA PARA APLICACIONES DE COMERCIO ELECTRÓNICO

Castellà Roca J., Planes Cid J., Domingo-Ferrer, J., Herrera-Joancomartí J.,

Departament d'Enginyeria Informàtica i Matemàtiques
Universitat Rovira i Virgili
ETSE-Autovia de Salou, s/n
E-43006 Tarragona

e-mail {jcaste, jplanes, jdomingo ,jherrera }@etse.urv.es

Resumen

En este artículo se presenta un análisis comparativo de rendimiento entre dos tarjetas Java (Java Cards). Describimos varias restricciones a tener en cuenta cuando se trabaja con aplicaciones para este tipo de tarjetas y cómo aquéllas influyen en el rendimiento de las aplicaciones, en especial aplicaciones de comercio electrónico.

Palabras clave: tarjetas inteligentes, comercio electrónico, Java Card.

1. INTRODUCCIÓN

En los últimos años, el comercio electrónico ha experimentado un crecimiento mayor que otras formas de comercio. Las previsiones y las oportunidades de futuro son excelentes. La criptografía de clave pública [1] ha sido clave para este crecimiento. En concreto, algoritmos de clave pública como RSA [3] han ayudado a superar el problema de la compartición de claves y ofrecen realizaciones del concepto de *firma digital*. Las firmas digitales hacen posible que las pruebas de autenticidad puedan ser producidas electrónicamente, aspecto *esencial* en el desarrollo del comercio electrónico.

Uno de los problemas de la criptografía pública y las firmas digitales es el alto coste computacional respecto al cifrado con claves compartidas. Las *funciones hash* pueden ser usadas de diversas formas para mitigar este problema. Una función hash és una función fácil de computar pero difícil de invertir. Las funciones hash se utilizan en muchas aplicaciones, pero se conocen básicamente por su papel en las firmas digitales: una función hash se utiliza para obtener un resumen del documento que se ha de firmar, para luego firmar este resumen en vez de firmar el documento completo. De esta forma la firma es más rápida, gracias a que el resumen es más pequeño que el documento. El uso de funciones hash en firmas digitales no compromete la seguridad, ya que un adversario no puede producir un documento alterado que concuerde con el resumen firmado del documento legítimo.

La principal diferencia entre firmas escritas y firmas digitales es la necesidad de un ordenador para las digitales, dada la complejidad de las operaciones criptográficas; no obstante, la funcionalidad de las firmas en el comercio electrónico sería la misma que en el comercio convencional. Esto significa que la capacidad de firmar no se puede restringir a un lugar físico o a un ordenador porque, como sucede en el comercio convencional, los compradores tienen que poder realizar la firma cuando y donde ellos quieran (tienda, banco, restaurante, etc.). Las *tarjetas inteligentes* son el único tipo de ordenadores portables que permiten al comprador llevar en su cartera la capacidad de computar firmas digitales.

El discurso anterior nos lleva a que las tarjetas inteligentes son interesantes en comercio electrónico para implementar como mínimo dos algoritmos criptográficos:

1. Algoritmo de firma digital

2. Algoritmo de hash

En lo que resta de artículo presentamos una comparación de rendimiento de la última generación de tarjetas inteligentes, la tarjetas Java. El benchmark usado en el análisis comparativo es una de las funcionalidades que una tarjeta inteligente de comercio electrónico tendría que ofrecer, esto es, un algoritmo de hash.

La sección 2 introduce los conceptos de la tarjeta Java, así como los de tarjetas inteligentes, que necesitamos para entender el entorno de los esquemas de desarrollo de tarjetas Java. En la sección 3 describimos los criterios de rendimiento de nuestra comparación, el procedimiento de prueba que hemos utilizado y diversas optimizaciones de programación. La sección 4 contiene los resultados de la comparación. Finalmente en la sección 5 se resumen las conclusiones y algunas líneas para futuras investigaciones.

2. TARJETAS INTELIGENTES JAVA

Una tarjeta Java es la implementación de un intérprete para un subconjunto del lenguaje de programación Java en un controlador estándar de tarjeta inteligente. Una tarjeta inteligente [2] es un ordenador físicamente seguro y portable con cierta capacidad de almacenaje de datos. Tiene las medidas exactas de una tarjeta de crédito convencional con la ventaja que puede realizar un pequeño procesamiento de datos. Las características de este tipo de ordenadores están descritas en la ISO 7816. Por su utilidad para el programador, destaquemos de dicha norma las unidades del protocolo de comunicación a nivel de aplicación (APDU Application Protocol Data Unit) a las que más adelante haremos referencia.

La tecnología de las tarjetas Java combina un subconjunto del lenguaje de programación Java con un entorno en tiempo de ejecución optimizado para tarjetas inteligentes. El objetivo de la tecnología de las tarjetas Java es aportar algunos de los beneficios de la programación en Java bajo la limitación de recursos de las tarjetas inteligentes.

Las restricciones más importantes, debidas a las limitaciones del procesador y de memoria, que presenta el lenguaje de las tarjetas Java son:

- No existen tipos de 64 bits
- No existen caracteres Unicode
- No existen hilos de ejecución (threads)
- No existen matrices multidimensionales
- No existe el recolector de basura

En una tarjeta inteligente Java la máquina virtual de Java (JVM) se considera como parte del sistema operativo, emplazado en la memoria ROM. Esta JVM se estructura en dos partes: el *conversor* y el *entorno en tiempo de ejecución* (JCRE). El conversor, emplazado

en el host externo al que se conecta la tarjeta, realiza la verificación y traduce el bytecode (código compilado) a un código insertable en la tarjeta, mientras que el JCRE, emplazado en la tarjeta, maneja los procesos de instalación, selección, deselección, ejecución y desintalación de un cardlet (aplicación para tarjeta Java).

2.1.Optimizaciones en la programación

Cuando trabajamos con tarjetas inteligentes, programar eficientemente se traduce en aprovechar eficientemente la memoria disponible, de tal modo que consideramos como importantes el uso eficiente de la RAM y el uso eficiente de la EEPROM.

En la memoria RAM, las tarjetas Java almacenan principalmente la pila Java. En esta pila se almacenan las variables locales de los métodos así como los parámetros de estos métodos. Podemos apreciar en la tabla 1 la escasez de este tipo de memoria, lo que implica que se sature con facilidad, momento en el que el sistema vuelca el contenido en la memoria EEPROM. Este proceso ralentiza el sistema, por lo que es aconsejable evitarlo.

Para evitarlo hemos seguido cuatro vías:

1. Evitar demasiadas invocaciones a método

2. Evitar argumentos y variables locales

3. Evitar variables locales innecesariamente grandes

4. Evitar expresiones demasiado complejas (anidadas)

Nótese que las líneas de desarrollo descritas entran en conflicto con el paradigma del lenguaje (programación orientada a objetos), pero se justifican por las restricciones hardware en las tarjetas.

En la memoria EEPROM se almacenan todos los objetos creados, pero el hecho de no tener el recolector de basura tiene el inconveniente de favorecer la saturación de este tipo de memoria, con el consecuente bloqueo de la tarjeta.

En el entorno de las tarjetas Java hemos de tener en cuenta que los objetos que sólo se referencian desde variables locales no se podrán referenciar una vez éstas dejen de existir, por lo que las líneas a seguir para conseguir un óptimo uso de la memoria EEPROM son:

1. Aumentar el número de métodos y atributos estáticos (static)

2. Disminuir la visibilidad de los métodos

3. Aumentar el número de métodos y atributos finales (final)

3. CRITERIOS DE RENDIMIENTO

Los criterios de rendimiento que hemos tenido en cuenta para la comparativa son:

Velocidad del procesador Esta característica, medida en milisegundos, determina el tiempo de respuesta de la tarjeta, desde que enviamos la instrucción hasta que recibimos respuesta.

Eficiencia de almacenaje Directamente vinculada con el método de codificación usado por cada paquete para codificar y almacenar el cardlet.

Interfaz de programación Este aspecto es especialmente importante porqué el desarrollo y prueba de las aplicaciones para la tarjeta tienen lugar en el host externo, con anterioridad a la carga de los cardlets resultantes en la tarjeta.

3.1.Procedimiento de prueba

Para el procedimiento de prueba hemos escogido el algoritmo MD5, una función hash propuesta por J. Massey [4], que recoge como entrada un mensaje de longitud variable y produce como salida un mensaje resumen de 128 bits. Se conjetura que es computacionalmente intratable producir dos mensajes que tengan el mismo resumen, u obtener el mensaje original que corresponde a un resumen dado. El algoritmo MD5 se usa para aplicaciones de firma digital, donde un fichero de gran tamaño tiene que ser comprimido de forma segura antes de encriptarlo con un criptosistema de clave pública (para producir la firma).

La desventaja de este algoritmo es que está diseñado para máquinas de 32 bits bastante rápidas, por lo que pierden eficiencia las implementaciones en tarjeta inteligente. Por el contrario, tenemos la ventaja de no requerir grandes tablas de sustitución, ya que puede ser codificado de forma bastante compacta.

4. RESULTADOS DE LA COMPARATIVA

Para realizar la comparativa hemos escogido dos tarjetas Java: *GemXpresso* de la empresa Gemplus, y *Odyssey* de Bull, en las que introducimos un cardlet (test.class) que responderá a dos comandos: get y put.

4.1.Características físicas de las tarjetas comparadas

Tarjeta	EEPROM	RAM	ROM	Reloj	Bits

GemXpresso	10+5	512	8	3-5	32
Odyssey	8 (7)		10	10	8

Tabla 1 - Características físicas de las tarjetas comparadas

En la tabla 1 podemos comparar las características físicas de las dos tarjetas consideradas, teniendo en cuenta que las memorias EEPROM y ROM están medidas en Kbytes, y la memoria RAM en bytes. En cuanto a las características del procesador consideramos importantes la velocidad de reloj (en Mhz) y la longitud de palabra del procesador (en bits). En la medición de la memoria EEPROM hemos indicado con el operador suma las memorias que se encuentran físicamente divididas, y entre paréntesis el valor real de la memoria una vez cargado el sistema operativo básico.

4.2.Velocidad de procesamiento y eficiencia de almacenaje

El proceso de cálculo del tiempo de procesamiento lo hemos dividido en dos fases (correspondientes a sendas APDUs):

1. **Put:** envio de los datos que posteriormente utilizará el algoritmo MD5. Estos datos consisten en un vector con un máximo de 128 bytes.

2. **Get**: cómputo del algoritmo MD5 y recogida de los resultados, consistentes en un vector de 16 bytes.

Tarjeta	**Put** (ms)	**Get** (ms)	**Almacenaje del cardlet** (bytes)
GemXpresso	266	1480	4691
Odyssey	141	650	4114

Tabla 2 - Tiempos de respuesta

En la tabla 2 podemos apreciar los tiempos de respuesta obtenidos de cada una de estas fases, así como la ocupación de memoria del cardlet de prueba. Se puede apreciar que la tarjeta Odyssey es más rápida que la tarjeta GemXpresso en ambas instrucciones. Esta diferencia se debe principalmente a:

- Odyssey trabaja con la version 2.1 de la API de Java Card, mientras que GemXpresso lo hace con la versión 2.0 (menos optimizada)

- Odyssey tiene una mayor capacidad de cómputo gracias a su mayor velocidad de reloj (según apreciamos en la tabla 1) respecto a la velocidad de GemXpresso.

Por el hecho de trabajar con 32 bits, GemXpresso podría ser más rápida que Odyssey a igual velocidad de reloj, pero debido al algoritmo de prueba utilizado (hemos adaptado MD5 a datos de 16 bits como máximo para poderlo ejecutar en Odyssey) esta ventaja es irrelevante.

Asimismo es digna de mención la mayor capacidad de compresión de código que tiene la tarjeta Odyssey, frente a la tarjeta GemXpresso, con las consecuentes ventajas ya comentadas en la sección 2.

4.3.Interfaz de progamación

Para analizar las diversas interfaces hemos divido el análisis en dos partes: el desarrollo del cardlet y las facilidades que aporta el entorno al programador.

El desarrollo de un cardlet lo dividimos en cuatro fases:

Conversor y verificador: a partir del código fuente (.java) se generan los bytecodes (.class) correspondientes, y éstos son compilados en un binario dependiente de la tarjeta Java.

Cargador: su misión es cargar los cardlets dentro de la tarjeta Java, visualizar la lista de cardlets cargados y eliminar los que deseemos.

Provador: es el marco mediante el que nos comunicaremos con la tarjeta, enviando y recibiendo APDUs.

Simulador: esta es una herramienta de gran utilidad, con la que podemos evaluar el comportamiento de nuestros cardlets antes de cargarlos en la tarjeta. El inconveniente de los simuladores actuales es que no tienen en cuenta las limitaciones de las tarjetas, con lo que el funcionamiento correcto de un cardlet en el simulador no garantiza que más tarde funcione bien en la tarjeta.

Gemplus ha integrado estas cuatro fases en un entorno de programación fácil de usar, denominado RAD (Rapid Applet Development). Este entorno ofrece herramientas de ayuda que facilitan el aprendizaje del desarrollo de aplicaciones para tarjetas Java, con extensa documentación, variados ejemplos y un sencillo wizard (ayuda guiada).

Por el contrario, Bull tiene las tres primeras fases en tres herramientas diferentes, sin un entorno englobante, con lo que obliga al desarrollador a trabajar con ficheros y aplicaciones desde línea de

comandos. Así, por ejemplo, la conversión y verificación requieren que el desarrollador introduzca la lista de ficheros .class que quiere incluir según un orden establecido. La ayuda que aporta este kit de desarrollo es escasa y pobre, por lo que el desarrollador ha de acudir a otras fuentes con bastante frecuencia.

Una funcionalidad especialmente útil añadida por la empresa Gemplus es el DMI (Direct Method Invocation), que permite abstraerse de la comunicación a nivel de APDUs entre el host y la tarjeta, añadiendo una capa en el protocolo de comunicación. La visión que tiene el programador es parecida a la que tiene con otros middleware (tales como CORBA), donde define la interfaz de comunicación entre los objetos involucrados.

5. CONCLUSIONES Y FUTURAS LÍNEAS DE INVESTIGACIÓN

Se ha presentado una comparativa de dos tarjetas Java, donde se ha mostrado que mientras la tarjeta Odyssey es más rápida, el entorno de programación de GemXpresso es mucho más versátil. Asimismo se han enumerado las diversas restricciones a tener en cuenta en la programación de las tarjetas Java en relación con Java estándar.

Futuras líneas de investigación deben ir dirigidas a la implementación del algoritmo MD5 sobre otras tarjetas Java (tales como SmartCafé de Giesecke & Devrient y Cyberflex de Schlumberger) para ampliar los resultados del análisis comparativo y posteriormente implementar algoritmos vinculados a aplicaciones de comercio electrónico.

REFERENCIAS

[1] DIFFIE W., HELLMAN M.E.., *New directions in cryptography.* IEEE Trans. On Information Theory, Vol. IT-22, nº 6 (1976), pág. 664-654.

[2] GUTHERY S.B., JURGENSEN T.M., *Smart Card Developer's Kit.* Macmillian Technical Publishing, (1998).

[3] RIVEST R.L., SHAMIR A., ADLEMAN L., *A method for obtaining digital signatures and public-key criptosystems.* Communications of the ACM, vol. 21, nº 2, (1978), pág.120-126.

[4] RIVEST R.L., *RFC 1321. The MD5 Message-Digest Algorithm.* MIT Laboratory for Computer Science, (1992).