Comparativa de Algoritmos de Cifrado

Nicolás Aguado, Unai Artano y Martín Horsfield

Criptografía Aplicada (UPV/EHU) — 8 de diciembre de 2024

Índice

1. Introducción

Esta práctica tiene como objetivo realizar comparaciones de tiempos (en cuanto a cifrar y descifrar) entre un algoritmo de cifrado basado en *criptografía ligera*, como `ASCON`, y uno que no está específicamente diseñado para esto y es de ámbito general, como AES. Para ello, se emplearán diferentes variantes y subversiones de cada algoritmo, además de distintas técnicas de compilación, permitiendo así una comparación más detallada y exhaustiva.

2. Objeto del Estudio

En el caso de ASCON, se utilizarán las implementaciones **ref**, **opt64** y **opt64_low**, disponibles en la página correspondiente de GitHub. Para AES, se trabajará con el algoritmo AES-128, empleando dos implementaciones *open source* diferentes: `tinyaes` (ya utilizada en trabajos previos) y `aesni`. Con las dos implementaciones se realizarán la utilizando **AES-128-GCM**.

Se realizarán comparaciones de los tiempos y de los tamaños de los ficheros de cifrado y descifrado con diversas iteraciones (para obtener una media adecuada, se harán siempre más de 5000 iteraciones del mismo caso de prueba), utilizando distintos tamaños de textos a cifrar y descifrar (20B, 100B, 300B, 1KiB, 3KiB y 5KiB), diversas opciones de optimización (-o0, -o1 y -o2), y se usará `-march=native` en el compilador para intentar obtener un resultado más rápido.

3. Implementación de las Pruebas

3.1. Diseño del código fuente

Se han diseñado tres ficheros `.c` de cara a realizar los test (a parte de los relativos en sí de cada librería). Estos ficheros contienen distintas directivas de compilación y distintos *strings* dentro de ellos para poder ejecutar las pruebas con los distintos tamaños (20B, 100B, 300B, 1KiB, 3KiB y 5KiB).

En el caso del algoritmo `ASCON` se realizan (en cada caso, cifrado y descifrado) 10.000 pasadas, desechándose las primeras 100 para tener en cuenta la llamada *fase de calentamiento* (para descartar posibles discrepancias con optimizaciones de compilación). Se calcula el tiempo de todas las iteraciones y después se obtiene el tiempo medio. En el caso de AES con `aesni` y `tinyaes` el procedimiento es el mismo, pero realizándose 5.000 iteraciones en cada caso.

Además, para simplificar la gestión de ficheros de código fuente, se utilizan distintas directivas de compilación en todos los casos para todos los algoritmos para ajustar determinados parámetros. Un extracto de, en concreto, cómo se ajusta la implementación de `ASCON` en el fichero `test_ascon.c` se expone a continuación.

test_ascon.c

```
#ifdef OPT64
#include "ascon/opt64/api.h"
#include "ascon/opt64/ascon.h"
#include "ascon/opt64/crypto_aead.h"
#elif defined(OPT64_LOW)
#include "ascon/opt64_low/api.h"
#include "ascon/opt64_low/ascon.h"
#include "ascon/opt64_low/crypto_aead.h"
#else
#include "ascon/ref/api.h"
#include "ascon/ref/ascon.h"
#include "ascon/ref/crypto_aead.h"
#endif
```

En todo caso, el código está diseñado para abarcar y hacer las mediciones para el máximo número de tamaños de ficheros a cifrar y descifrar en una sola ejecución.

3.2. Compilación y Ejecución Automatizada

Para lanzar el conjunto de pruebas se ha diseñado una herramienta *bash* (`script.sh`) que provee utilidades para la compilación y ejecución.

Por ejemplo, al lanzar el script de manera

shell

```
$ bash script.sh compile
```

Se ejecutan todos los comandos respectivos de compilación y se guardan todos los binarios en la carpeta `./bin`, para luego poder ejecutar todos los binarios generados usando

shell

```
$ bash script.sh execute
```

En una plataforma x64, se generan 20 binarios en total y al hacer las ejecuciones se obtienen un total de 90 mediciones.

4. Preparación y descripción del entorno

A continuación se describen las características técnicas y la descripción del entorno en el que se ejecutan las pruebas.

En cuanto al primer entorno de ejecución, se ejecutarán las pruebas en un equipo relativamente moderno, cuyas características técnicas más relevantes son las expuestas en el Cuadro 1.

Especificación	**Valor**
Fabricante	*Micro-Star International*
Arquitectura	x86_64
Procesador	*Intel Core* i7-10750H
Velocidad del Procesador	12 x 2.60 GHz
Memoria RAM	16 GB
Sistema Operativo	*Debian GNU/Linux* 12.8
Año de Fabricación	2020
Versión del Compilador	*GCC* 12.2.0

Cuadro 1: Especificaciones técnicas del equipo portátil

En cuanto al segundo entorno de ejecución, el equipo considerado es un dispositivo de baja potencia y además es relativamente antiguo. Las características técnicas se detallan en el Cuadro 2.

Especificación	**Valor**
Fabricante	*Raspberry Pi Ltd*
Modelo	*Raspberry Pi* 3 Model 8 v1.2
Arquitectura	ARMv71
Procesador	ARMv7 Processor rev 4
Velocidad del Procesador	4 x 1.2 GHz
Memoria RAM	1 GB
Sistema Operativo	*Raspbian GNU/Linux* 9
Año de Fabricación	2015
Versión del Compilador	*GCC* 6.3.0

Cuadro 2: Especificaciones técnicas del equipo de baja potencia

En ambos casos se está trabajando con un sistema operativo *GNU/Linux* (y en concreto, desde *Debian* y una variante de *Debian*) y se realizan las pruebas empezando desde el reposo, y sin ningún otro proceso abierto y/o consumiendo recursos a parte de los habituales del propio sistema operativo.

Una limitación importante a considerar es la imposibilidad de ejecutar los casos de prueba en el caso de `aesni` para el dispositivo de baja potencia (Raspberry Pi), ya que esta librería no se encuentra disponible para arquitecturas ARM. Entonces, esta parte específica queda fuera del estudio.

5. Arquitectura x64

En esta sección se exponen los resultados de las pruebas realizadas en la arquitectura x64.

5.1. Distintos tamaños de ficheros con misma optimización

En este apartado se calculan los distintos tiempos de los algoritmos mencionados con los distintos tamaños de los ficheros mencionados pero usando la misma optimización (-o2) en todos los casos.

5.1.1. Tiempos para cifrado

Tras calcular los tiempos de cifrados para esta parte y tras crear una gráfica para visualizarlos mejor, la gráfica generada quedaría de esta manera:

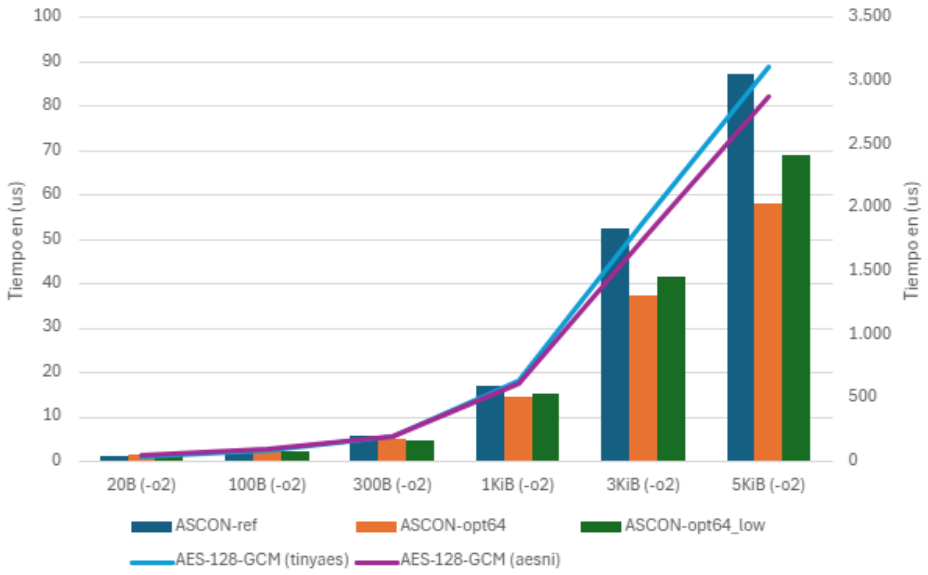

Figura 1: Comparativa de los tiempos de cifrado en x64 con los distintos tamaños de ficheros y según las distintas implementaciones

Como se puede observar en la Figura 1, los datos que aparecen en la parte izquierda de la gráfica (del 0 al 100) representan los tiempos del algoritmo ASCON. Por otro lado, los datos que aparecen en la parte derecha de la gráfica corresponden a los tiempos del algoritmo AES, utilizando las dos librerías comparadas.

A continuación, se presenta otra gráfica con una escala logarítmica, lo que permite visualizar mejor los tiempos de cada algoritmo y resaltar con mayor claridad las diferencias entre ellos.

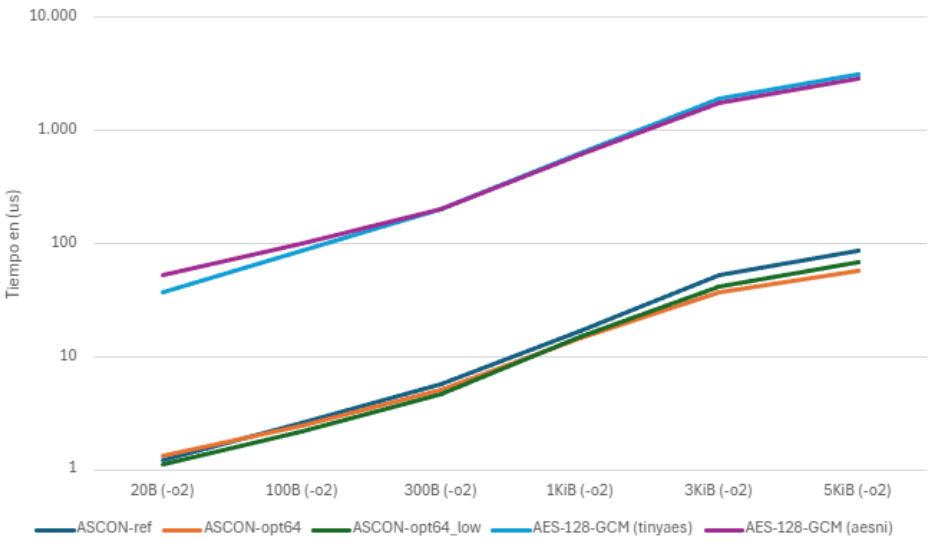

Figura 2: Comparativa en escala logarítmica de los tiempos de cifrado en x64 de con distintos tamaños de ficheros según las distintas implementaciones

En cuanto a los tiempos registrados, se puede observar en la Figura 2 que el algoritmo ASCON es significativamente más eficiente que el algoritmo AES, lo cual era esperado dado que ASCON está diseñado como un algoritmo de criptografía ligera.

Focalizándonos en ASCON, se puede notar que, para ficheros de menor tamaño en bytes, los tiempos de cifrado son aproximadamente iguales entre las implementaciones. Sin embargo, usando *opt64_low*, se logran tiempos ligeramente mejores, lo que la convierte en la opción más rápida para estos casos. Por esta razón, aunque las diferencias entre las implementaciones son pequeñas para ficheros pequeños, *opt64_low* presenta una ventaja notable.

Para ficheros de mayor tamaño en bytes, las diferencias de tiempo comienzan a ser más evidentes. La implementación *ref* de ASCON resulta considerablemente más lenta en comparación con las demás. Finalmente, para ficheros con tamaños muy grandes, la implementación *opt64* supera a *opt64_low* en velocidad, ya que esta última está más optimizada para hardware limitado, mientras que *opt64* aprovecha mejor las capacidades de hardware avanzado. Esto explica por qué *opt64* tiende a ser más eficiente en entornos con hardware de mayor rendimiento.

Al utilizar el algoritmo AES-128-GCM y comparar sus tiempos con los del algoritmo ASCON (en términos generales), se observa que AES es significativamente más lento que ASCON. Por ejemplo, el tiempo que ASCON necesita para cifrar un fichero de 3KiB es equivalente al tiempo que AES con la librería `tinyaes` tarda en cifrar un fichero de solo 20B. De manera similar, el tiempo requerido por ASCON para cifrar un fichero de 5KiB equivale al tiempo que AES con la librería `aesni` necesitaría para cifrar un fichero de apenas 20B.

Enfocándonos en AES, se puede observar una diferencia sutil en el rendimiento entre las dos librerías. Para el cifrado, `tinyaes` es más eficiente que `aesni`.

5.1.2. Tiempos para descifrado

Tras calcular los tiempos de descifrado con distinto tamaño y tras crear una gráfica para visualizarlos mejor, el resultado se expone en la Figura 3.

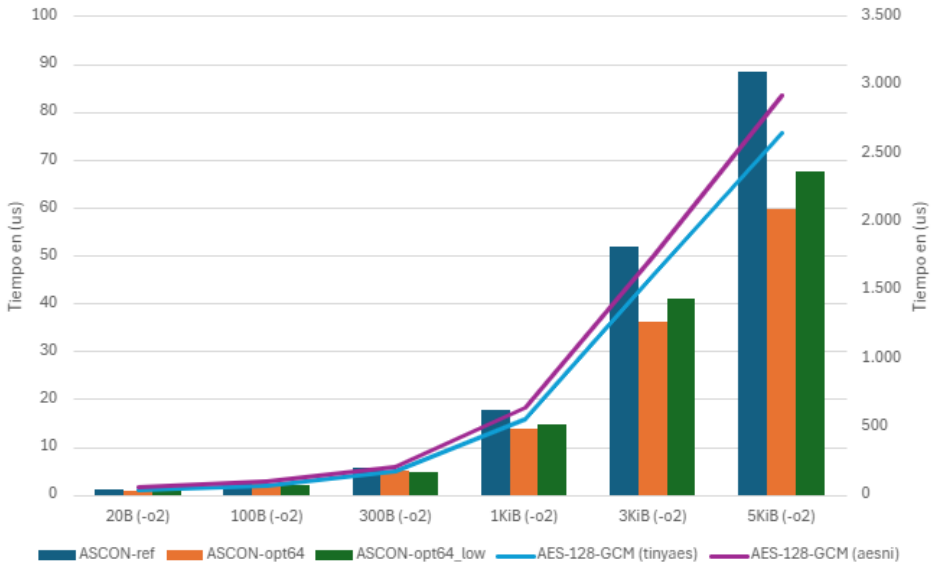

Figura 3: Comparativa de los tiempos de descifrado en x64 con los distintos tamaños de ficheros y según las distintas implementaciones

Como se puede apreciar en la Figura 3, los datos que aparecen en la parte izquierda de la gráfica (del 0 al 100) representan los tiempos del algoritmo ASCON. Por otro lado, los datos que aparecen en la parte derecha de la gráfica corresponden a los tiempos del algoritmo AES, utilizando las dos librerías comparadas.

Al igual que en el apartado 5.1.1, a continuación se va mostrar una gráfica donde se va a mostrar los tiempos en escala logarítmica. De esta forma y al igual que el apartado anterior, se podrá observar mejor la diferencia de los tiempos:

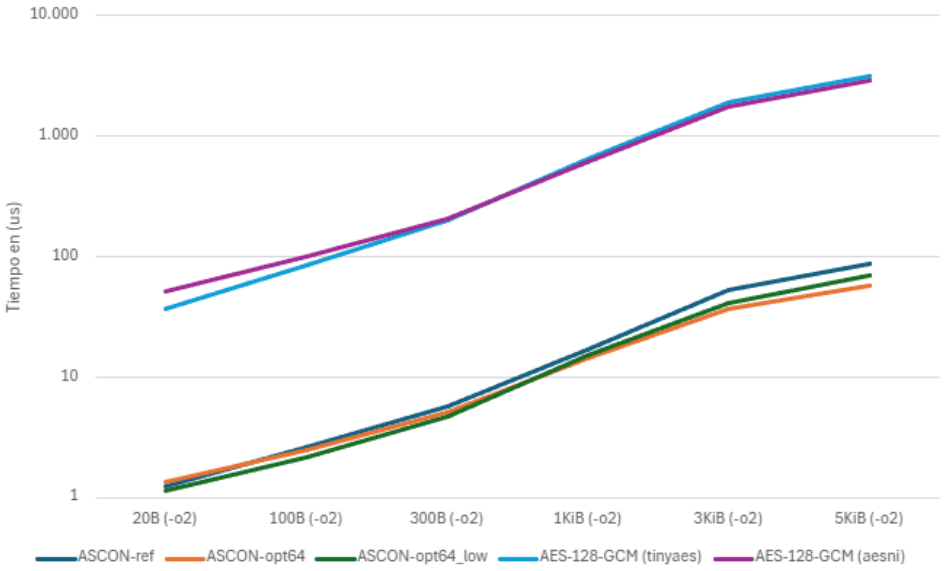

Figura 4: Comparativa en escala logarítmica de los tiempos de descifrado en x64 de con distintos tamaños de ficheros según las distintas implementaciones

Al igual que ocurrió al cifrar los ficheros, se está demostrando que utilizar el algoritmo ASCON es más eficiente que el algoritmo AES.

Si se analizan más a fondo los tiempos de las distintas implementaciones de ASCON, se puede observar que ocurre lo mismo que con los tiempos de cifrado: al usar *opt64_low*, es ligeramente más eficiente que *ref* en ficheros de menor tamaño. Sin embargo, al trabajar con ficheros de mayor tamaño, se aprecia que *opt64_low* es mucho menos eficiente en comparación con la implementación *opt64*. Para ficheros más grandes, *opt64* comienza a ser la implementación más eficiente. Esto se debe a lo mencionado anteriormente: la implementación *opt64_low* está más optimizada para hardware limitado, mientras que *opt64* aprovecha mejor las capacidades de hardware avanzado. Por ello, *opt64* tiende a ser más eficiente en entornos con hardware de mayor rendimiento.

Al realizar una comparación más detallada del algoritmo AES utilizando las dos implementaciones de librerías, se observa que, en el proceso de descifrado, trabajar con la librería `aesni` es ligeramente más rápido que con la librería `tinyaes`. Esta elección de librería es contraria frente a la extraída para el cifrado mediante AES.

5.2. Mismos tamaños de ficheros frente a diversas opciones de optimización

En este apartado se analizan las medias de tiempo para los mismos tamaños de fichero (5KiB) usando diversas opciones de optimización.

5.2.1. Media de tiempos para cifrado

Tras calcular los tiempos de cifrado para esta parte y tras crear una gráfica para visualizarlos mejor, la gráfica generada quedaría de esta manera:

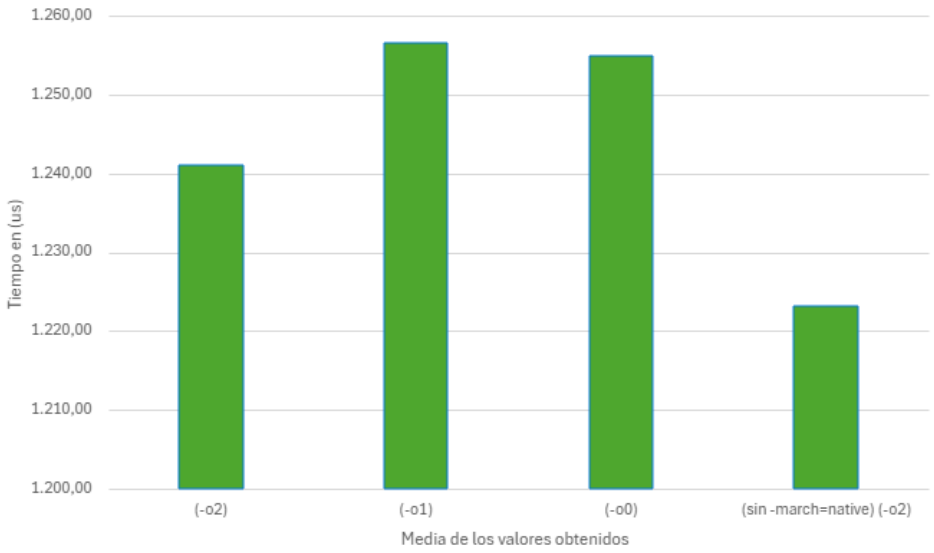

Figura 5: Media de los tiempos de cifrado en x64 para distintas optimizaciones

Como se puede observar en la Figura 5, los tiempos son aproximadamente similares frente a analizar cada algoritmo por separado.

En general, el tiempo medio de cada optimización es bastante similar, con una diferencia aproximada de 30 microsegundos. Se observa que la optimización *-o2* es la más rápida entre las opciones evaluadas. Sin embargo, al combinar *sin -march=native* con *-o2*, se logra un rendimiento ligeramente superior en comparación con el uso exclusivo de *-o2*.

5.2.2. Media de tiempos para descifrado

Tras calcular los tiempos de descifrado para las distintas optimizaciones la gráfica generada se visualiza en la Figura 6.

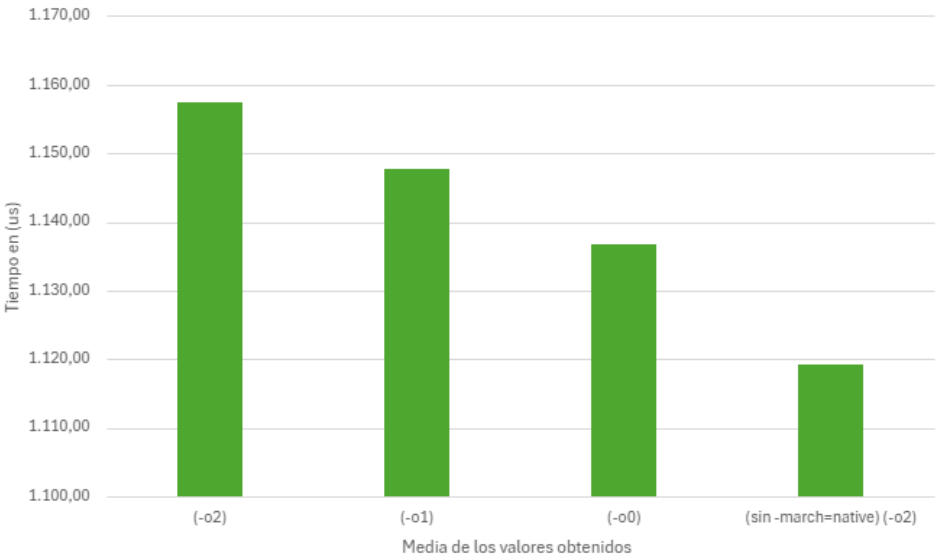

Figura 6: Media de los tiempos de descifrado en x64 para distintas optimizaciones

Algoritmo	Tamaño del fichero compilado
ASCON-ref	20648
ASCON-opt64	331816
ASCON-opt64_low	20672
AES-128-GCM (tinyaes)	26096
AES-128-GCM (aesni)	21192

Cuadro 3: Tamaño de los ficheros compilados en x64 (en Bytes)

Como se puede observar en la Figura 6, los tiempos de descifrado son ligeramente más rápidos que los tiempos de cifrado, con una diferencia de 40 microsegundos.

Sin embargo, se observa una diferencia grande cuando comparamos con la Figura 5. Podemos ver que, esta vez, de todos los niveles de optimización de código, `-o2` es el más lento y `-o2` con `march=native` el más rápido. Esto puede ocurrir ya que `-o2` incluye optimizaciones que en algunos casos especiales pueden afectar el tiempo de ejecución.

De todos modos, los resultados son los esperados a grandes rasgos, ya que no hay una diferencia de tiempos extremadamente significativa.

5.3. Comparación entre el tamaño de los archivos de cada algoritmo

Respecto a los tamaños de los archivos generados tras haber realizado el cálculo de los tiempos, éstos se exponen en el Cuadro 3.

Antes de nada, y según el código generado, los archivos producidos corresponden a la versión unificada para todos los tamaño de fichero (incluidos en un mismo binario) para cada algoritmo e implementación utilizada. Como se puede observar en el Cuadro 3, los tamaños de los ficheros generados son bastante similares. Sin embargo, los ficheros generados al usar el algoritmo ASCON son más pequeños en comparación con los demás. Además, utilizando la implementación *ref*, el fichero generado es el que tiene el menor tamaño.

Respecto a los tiempos, aunque *ASCON-opt64* tenga un tamaño de fichero elevado, éste suele desembocar en un tiempo de ejecución normalmente menores (como se puede observar en las Figuras 1 y 3).

6. Arquitectura ARM

En esta sección se exponen los resultados de las pruebas realizadas en la arquitectura ARM.

Cabe mencionar que, al intentar aplicar el algoritmo AES en esta arquitectura (tanto para las implementaciones de `aesni` como para `tinyaes`), surgieron errores de compilación y/o ejecución, por lo que no se pudieron calcular los tiempos correspondientes. Esto probablemente se deba a la necesidad de una librería adicional que no se tuvo en cuenta o que no se encuentra disponible en el sistema operativo *Raspbian* de forma intuitiva. Entonces, esta comparativa expone los resultados obtenidos con las distintas variantes del algoritmo `ASCON` y las distintas técnicas de optimización y tamaños de ficheros.

6.1. Distintos tamaños de ficheros con misma optimización

En este apartado se calculan los distintos tiempos de los algoritmos mencionados con los distintos tamaños de los ficheros mencionados pero usando la misma optimización (-o2) en todos los casos.

6.1.1. Tiempos para cifrado

Tras calcular los tiempos de cifrados para esta parte y tras crear una gráfica para visualizarlos mejor, la gráfica generada quedaría de esta manera:

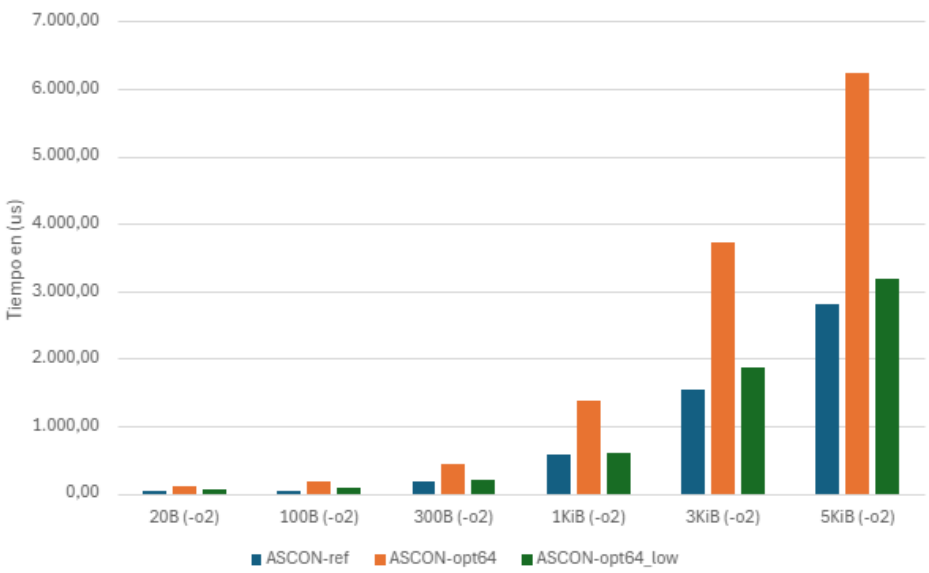

Figura 7: Comparativa de los tiempos de cifrado en arquitectura ARM con los distintos tamaños de ficheros y según las distintas implementaciones

Tras analizar las Figuras 7, 1 y 5, se observa que ejecutar el código en una arquitectura ARM es significativamente más lento que en un hardware avanzado. Es el resultado esperado.

Al analizar más detenidamente los resultados y centrados únicamente en la arquitectura ARM, se puede notar que, la implementación *ref* es más rápida en comparación con otras implementaciones. Cabe destacar que *opt64_low* también puede resultar adecuada en este tipo de arquitecturas de bajo rendimiento, ya que tiene un desempeño similar. Por lo tanto, aunque *ref* es la opción más rápida entre las tres, *opt64_low* puede ser una alternativa eficiente si se busca utilizar otra implementación para otra posible arquitectura de bajo rendimiento.

6.1.2. Tiempos para descifrado

Tras calcular los tiempos de descifrado con distinto tamaño y tras crear una gráfica para visualizarlos mejor, el resultado se expone en la Figura 8.

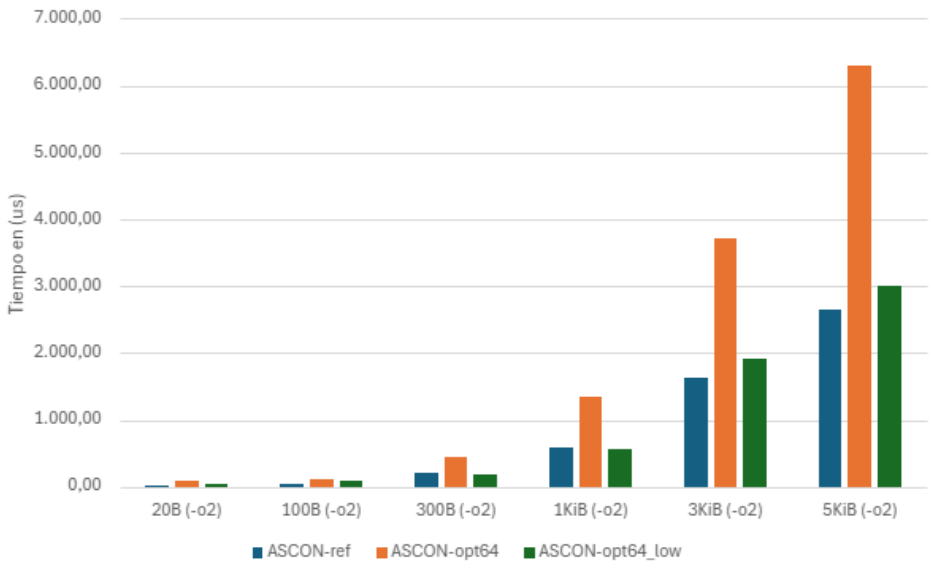

Figura 8: Comparativa de los tiempos de descifrado en arquitectura ARM con los distintos tamaños de ficheros y según las distintas implementaciones

Tras analizar las Figuras 8, 3 y 6, se observa un comportamiento similar al del proceso de cifrado. Es decir, los tiempos en una arquitectura ARM son significativamente más lentos que en una arquitectura x64. Es, otra vez, el resultado esperado.

Además, dependiendo de la implementación utilizada, el rendimiento puede variar, siendo más rápido o más lento según la elección. Se pueden extraer unas conclusiones parecidas a las expuestas en el apartado anterior (6.1.1)

6.2. Mismos tamaños de ficheros frente a diversas opciones de optimización

En este apartado se analizan las medias de tiempo para los mismos tamaños de fichero (5KiB) usando diversas opciones de optimización.

6.2.1. Media de tiempos para cifrado

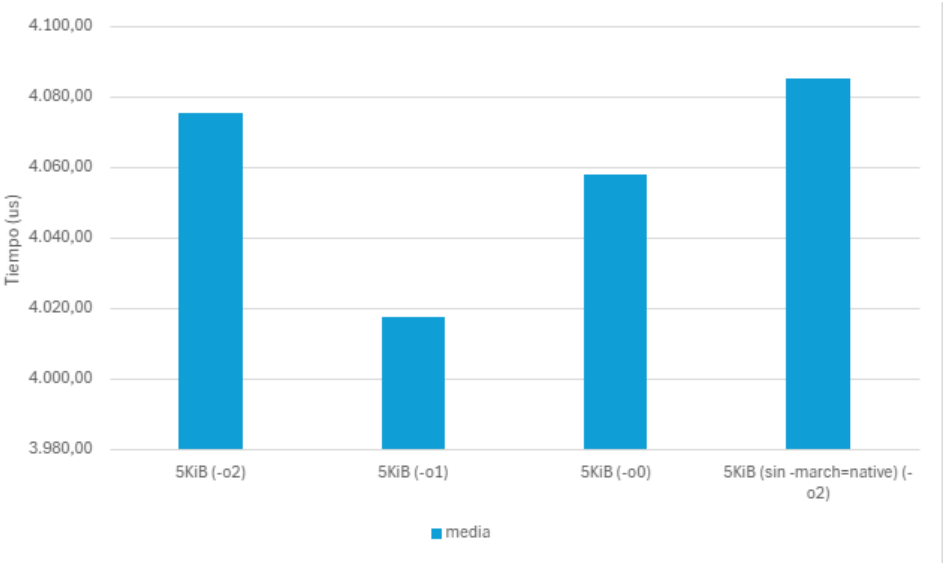

Figura 9: Tiempos medios de cifrado en la arquitectura ARM según las distintas optimizaciones

Se puede observar en la Figura 9 que la optimización *-o1* tiende a ser ligeramente más rápida que las demás en una arquitectura ARM.

Por otro lado, para la arquitectura x64 (en la Figura 5), se ha constatado que la optimización sin *-march=native* combinada con *-o2* resulta más eficiente en ese tipo de arquitecturas. En el caso de arquitecturas ARM, la optimización *-o1* con *-march=native* resulta ser la mejor opción, y la optimización sin *-march=native* combinada con *-o2* es la menos eficiente.

Este es un resultado contrario a lo esperado. Las optimizaciones del compilador causan un rendimiento menor.

6.2.2. Media de tiempos para descifrado

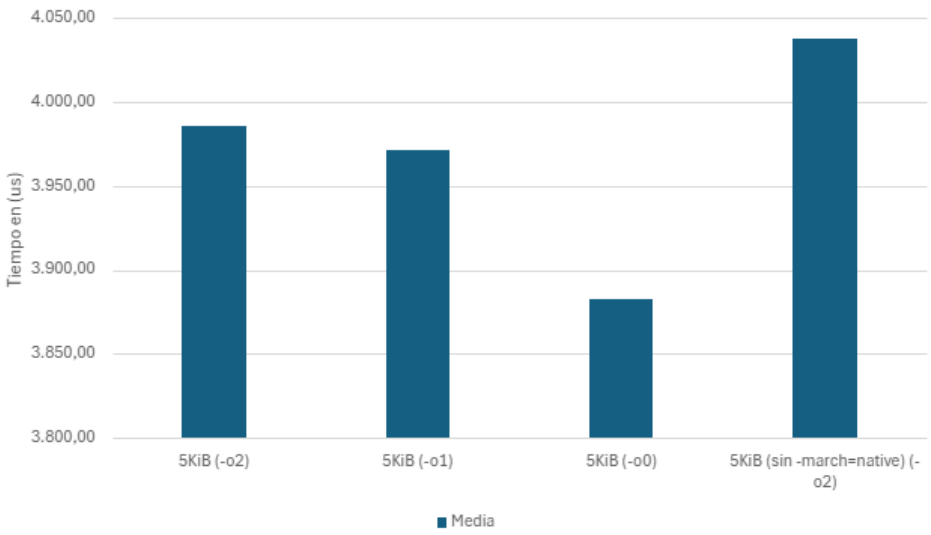

Figura 10: Tiempos medios de descifrado en la arquitectura ARM según las distintas optimizaciones

Como se aprecia en la Figura 10, tras calcular el promedio, se observa un comportamiento diferente en comparación con el cifrado. En este caso, al utilizar la optimización *-o0*, se logra un tiempo relativamente inferior en comparación con la optimización *-o1*, que fue la más eficiente en el caso del cifrado. Se desea poner en contraste este resultado frente al apropiado de la plataforma x64.

Al comparar con una arquitectura x64 (en la Figura 6), se aprecia que la optimización sin *-march=native* combinada con *-o2* es la más rápida, mientras que la optimización *-o1* muestra un rendimiento más lento. En el caso de arquitecturas ARM, la optimización *-o0* resulta ser la mejor opción, y la optimización sin *-march=native* combinada con *-o2* es la menos eficiente.

Al igual que en el apartado anterior, este es un resultado contrario a lo esperado. Las optimizaciones del compilador causan un rendimiento menor.

7. Conclusiones

Tras haber analizado y calculado el tiempo que proporcionan los ficheros con los distintos algoritmos, implementaciones y optimizaciones, además del espacio que ocupan en disco, surge la pregunta: *¿Existe una implementación que sea la mejor para todos los casos?*. La respuesta (como suele ser habitual en estos casos) es *depende de lo que se quiera hacer*. Se ha observado que no hay una implementación específica que sea la más rápida en todas las situaciones; la elección depende del objetivo. Sin embargo, está claro que, si se busca un cifrado y descifrado rápido, el algoritmo `ASCON` es la mejor opción. Ahora bien, ¿qué implementación de `ASCON` se debería usar? La arquitectura del sistema tiene un impacto significativo, como se ha evidenciado en las gráficas. No es lo mismo trabajar con una arquitectura moderna utilizando la implementación *opt64* con la optimización *-o2*, que trabajar con una arquitectura ARM empleando la implementación *ref* con la misma optimización.

En el caso de plataformas antiguas con arquitectura ARM, está claro que las implementaciones más adecuadas son *ref* y, en algunos casos, *opt64_low*, ya que esta última está diseñada para hardware limitado. Por otro lado, si se dispone de hardware más avanzado (x64), las implementaciones recomendadas son *opt64* y/o *opt64_low*, dependiendo de las prioridades que se tengan en cuanto al tamaño de los ficheros con los que se trabaje. Para ficheros de menor tamaño, es mejor utilizar *opt64_low*, ya que tiende a ser más eficiente con hardware de menor rendimiento. Sin embargo, para ficheros de mayor tamaño, la implementación *opt64* es más adecuada, pues ofrece mayor eficiencia en entornos con hardware de alto rendimiento.

¿Y qué optimización usar? En plataformas x64, se ha comprobado que los mejores resultados se consiguen con la optimización `-o2` sin usar el flag `-march=native`, tanto en el cifrado como en el descifrado. Las *optimizaciones* que incluye el compilador al usar este flag acaban causando el efecto contrario al deseado. Una posible especulación para este comportamiento podría deberse a que estas librerías de cifrado están programadas según un estilo de código específico diseñado para el máximo rendimiento y las reorganizaciones que causa el compilador no son las óptimas; acabando causando problemas de rendimiento. Para la arquitectura ARM se obtienen resultados diferentes a x64 en cuanto a las optimizaciones de código. Por ejemplo, para el cifrado, la optimización `-o1` con `-march=native` resulta ser la mejor. Esto se puede deber a que las optimizaciones introducen ciertas funciones que, cuando se aplican a ARM, resultan no ser eficientes (ver Cuadro 2).

Si se desea trabajar con AES 128 GCM sin valorar otros algoritmos, hay que considerar la arquitectura destino. En el caso de x64, la elección entre `tinyaes` y `aesni` depende de la prioridad entre cifrado o descifrado. Si se opta por una aplicación donde el cifrado prevalezca, se ha de considerar `tinyaes`, mientras que si lo más significativo es el descifrado se ha de valorar `aesni`. Sin embargo, en arquitecturas ARM, puede incrementarse la dificultad de la implementación (usando tanto `aesni` como `tinyaes` debido a la necesidad de añadir librerías externas para la compilación y/o ejecución. Resolver esta incertidumbre queda fuera del alcance de este análisis.

El tamaño de los archivos generados tras las compilación también varía según la arquitectura utilizada, aunque sin grandes diferencias (independientemente del algoritmo utilizado). Aún así, se desea recalcar que la implementación del algoritmo `ASCON` en su modalidad *opt64* genera archivos de considerable tamaño y por tanto si el tamaño es una prioridad muy deseable en la implementación se recomiendan sus otras dos variantes (*opt64-low* o *ref*), siguiendo las recomendaciones previamente mencionadas.

En resumen, utilizar `ASCON` proporciona tiempos significativamente menores en comparación con AES. Para decidir la implementación del algoritmo ideal y la optimización de compilación adecuada, se debe considerar:

- Si la operación más habitual es cifrar o descifrar.
- La arquitectura del hardware.
- El tamaño de los datos a procesar.
- Las prioridades entre tiempo de ejecución y uso de almacenamiento en disco.

Las tendencias actuales tienden a considerar hoy en día el abaratamiento de los costes de almacenamiento y se tiende a priorizar la velocidad de ejecución sobre el espacio en disco. Pero, aún así, la elección definitiva depende de cada caso de uso en específico y de las necesidades concretas de la situación.

A. Anexo

Índice de figuras

Índice de cuadros