Crypto Acceleration on FreeBSD

Philip Paeps
philip@FreeBSD.org

AsiaBSDCon 2009 — Tokyo, Japan
March 2009

Abstract

As more and more services on the internet become cryptographically secured, the load of cryptography on systems becomes heavier and heavier. Many of the embedded communications processors supported by FreeBSD provide acceleration for cryptographic operations in silicon and various manufacturers build hardware for accelerating secure web traffic and IPsec VPN tunnels.

In the FreeBSD kernel, acceleration hardware is supported by the `opencrypto` framework. This paper presents an overview of the framework and explains the advantages and pitfalls of using hardware acceleration on various workloads and system configurations.

1 The `opencrypto` Framework

1.1 Overview

Originally written by Angelos D. Keromytis for OpenBSD, the `opencrypto` framework was ported to FreeBSD by Sam Leffler in 2002. The framework provides a consistent interface to hardware and software cryptography methods inside the kernel and gives userspace access to hardware cryptography through a `/dev/crypto` device node.

At the time of this writing, the FreeBSD kernel includes device drivers to support various hardware cryptography devices manufactured by AMD, Broadcom (Bluesteel), Hifn, SafeNet and VIA. While it is possible for third parties to provide out-of-tree device drivers for different hardware, the author is not aware of any such drivers at this time.

In addition to the various hardware device drivers, the FreeBSD kernel also includes a generic software driver which is used when no hardware is available for a given algorithm. The software driver supports a long list of commonly used algorithms.

Inside the kernel, `opencrypto` is currently used by IPsec, GELI and ZFS. In userspace, an OpenSSL engine using `/dev/crypto` can be used to (potentially) accelerate applications linked with `libcrypto`. For performance reasons (*q.v.*) however, this engine is not enabled by default.

1.2 Architecture

The `opencrypto` framework provides a "driver" interface on the one hand and a "consumer" interface on the other. The framework uses sessions to cache information and settings in drivers so initialization is not required for every request. A sessionless mode of operation is also available for keying operations.

Device drivers—either hardware device drivers or software drivers—register the algorithms they support with the framework and provide a set of callback functions for the framework to use for managing sessions and dispatching cryptographic operations.

For regular cryptographic operations, consumers create a session describing the kind of operations they would like to perform, the keys to be used and possibly an initialization vector. Multiple cryptographic operations can be chained in a single session. Consumers then create requests describing the data to be operated on and refering to the session to be used.

Sessionless keying operations operate directly on input and output parameters. Consumers describe the input they provide and the output they expect when they dispatch requests.

Since consumers are not necessarily associated with a process, the framework may not `sleep(9)` anywhere. For this reason, both the session-based and the sessionless modes of operation are completely asynchronous and consumers have to provide functions to be called when requests complete.

2 Hardware Acceleration

2.1 Principles of Operation

Cryptographic operations tend to be computationally intensive. On slower hardware, they can keep a system busy for significant amounts of time. Offloading these operations to hardware devices which implement the algorithms in silicon allows the system CPU to continue to perform other tasks while the cryptographic operations take place on another chip. When the operations complete, the CPU will be interrupted.

Since most popular cryptographic algorithms are designed to be easy to implement in hardware and building hardware to perform these operations quickly is also fairly easy, hardware acceleration can provide significant benefits at relatively low cost.

2.2 Advantages and Pitfalls

Acceleration hardware can provide a significant performance boost on embedded devices whose CPUs are clocked fairly low. Currently, many of the embedded communications processors supported by FreeBSD even provide acceleration for cryptographic operations in silicon on the same chip as the CPU.

When the CPU can process cryptographic operations faster than acceleration hardware, it makes little sense to use such hardware. In fact, trying to use hardware acceleration when the host CPU is faster will likely result in much poorer performance than using the host CPU directly.

2.3 Kernel Subsystems

Currently, most cryptography consumers inside the kernel make use of the `opencrypto` framework. It must be noted that kernelspace consumers will automatically try to make use of hardware acceleration if it is available. As described above, this may be a problem if the system CPU is faster than the available acceleration hardware.

On systems where most cryptographic operations are in the kernel, such as IPsec VPN termination points or fileservers writing to encrypted block devices, the use of hardware acceleration should therefore be carefully considered.

Device drivers for acceleration hardware should only be loaded if hardware acceleration is faster than using the system CPU.

2.4 Userspace Processes

FreeBSD includes an OpenSSL "engine" around `/dev/crypto` to provide access to acceleration hardware for userspace processes. In addition to the performance problems already described when the system CPU is faster than available acceleration hardware, enabling this engine comes with another caveat: enabling the engine system-wide will cause all cryptographic operations to be passed to the `opencrypto` framework in the kernel.

It may be tempting to do so on slower systems where acceleration hardware is present and faster than the system CPU. One should keep in mind however, that acceleration hardware generally only supports a limited number of different operations (often just one, even) and operations not supported in hardware will be handled by the software driver. While the software driver is overall fairly efficient, copying hardware back and forth between userspace and the kernel is not efficient at all. OpenSSL will often be much faster.

On such a system, the best solution is often to limit the algorithms used by applications to those supported by acceleration hardware or to limit the use of the engine to those applications using supported algorithms.

3 Future Directions

3.1 Performance Pitfalls

While the advantages of hardware acceleration on embedded devices are clear, the performance pitfalls on generic hardware make it difficult to enable acceleration by default.

A mechanism for measuring the relative performance of available acceleration hardware against

software implementations would be very useful. This would also be beneficial in the (perhaps unlikely) scenario where multiple hardware acceleration devices supporting an overlapping list of cryptographic algorithms are present.

Similarly, a mechanism should be put in place to prevent copying data back and forth between userspace and the kernel when it can be determined that OpenSSL could perform the requested operations faster in userspace than the `opencrypto` framework could perform them in the kernel and/or by offloading them to acceleration hardware.

3.2 Hardware Support

Currently, the acceleration features of the `opencrypto` framework are mostly only useful in an embedded context. There is currently no device driver support for higher-end acceleration hardware nor for the more sophisticated hardware offloading mechanisms such higher-end hardware systems require, such as the offloading of IPsec or TLS entirely to hardware.

Adding support for more high-end hardware and more sophisticated offloading methods would make the framework much more widely deployable.

4 Conclusion

On slower embedded systems, the benefits of the hardware acceleration features of the `opencrypto` framework can be quite significant. On higher-end systems however, the use of acceleration hardware can negatively impact performance.

More work will need to be done to take into account the relative benefits of hardware acceleration and to make the framework pick the best performing solution under all workloads, even if that means not using available hardware.

Additionally, broader hardware support and support for more sophisticated hardware offloading mechanisms would make the framework more useful to applications outside the embedded world.

About the author

Philip Paeps is a software consultant and contractor based in Belgium. He spends most of his energy on embedded systems. Currently, he is working full-time on the network stack of a widely-deployed internet gateway device.

Philip is a FreeBSD committer, contributing mainly to the kernel, a member of the FreeBSD security team and the FreeBSD core team secretary. He has been using FreeBSD for longer than he can remember.