Cryptographic Device Support for FreeBSD

Samuel J. Leffler

Errno Consulting
sam@errno.com

ABSTRACT

FreeBSD recently adopted the OpenBSD Cryptographic Framework [Keromytis et al, 2003]. In doing so it was necessary to convert the core framework to function correctly in a fully-preemptive/multiprocessor operating system environment. In addition several issues with the basic design were found to cause significant performance loss. After addressing these issues we found that FreeBSD outperformed OpenBSD on identical hardware by as much as 100% in tests that exercise only the cryptographic framework. These optimizations result in similar performance improvements for facilities like IPsec that make heavy use of the cryptographic framework. We observed that FreeBSD's Fast IPsec [Leffler, 2003] typically outperforms OpenBSD's IPsec implementation [Miltchev et al, 2002] by more than 50% on identical hardware.

We conclude that the OCF cryptographic API can be optimized and re-tuned to deliver substantially better performance than the original OCF implementation with large gains in both throughput and latency. Moreover these changes can be made with no impact on clients of the cryptographic framework: both user and kernel sofware designed for the original OCF is easily ported to the FreeBSD implementation of OCF.

1. Background and Introduction

Cryptographic transformations are an important component of security applications and protocols. Because these operations are computationally expensive vendors have developed products that accelerate the calculations and offload the work from the main processor. The OpenBSD Cryptographic Framework (OCF) [Keromytis et al, 2003] was developed to provide a uniform interface to cryptographic resources. It provides an in-kernel API to cryptographic resources and a device interface for user-level access to hardware-accelerated cryptographic operations. This functionality is critical for high performance implementations of security protocols such as IPsec [Kent & Atkinson, 1998], SSL [Freier et al, 1996], TLS [Dierks & Allen, 1999], DNSSEC [Eastlake, 1999], ssh [Ylonen et al, 2002], and Kerberos [Kohl & Neuman, 1993]. Good cryptographic support is also important for implementing encrypted secondary storage [Gattaneo et al, 2001] and virtual memory [Provos, 2000].

Recognizing the importance of this facility, FreeBSD recently adopted the OCF. Porting the software however required addressing several issues.

- The OCF was designed for a uniprocessor system without kernel preemption. The FreeBSD 5.0 operating system has fine-grained locking and the kernel is fully preemptive. This required a rewrite of the core crypto functionality.
- The I/O framework used on OpenBSD is different than that used by FreeBSD. The crypto device drivers required significant modification to deal with these differences.
- The OCF duplicates cryptographic support already present in FreeBSD for the KAME IPsec software. In some cases the existing implementations are superior to those provided by the OCF. Rather than duplicate this software each component was evaluated and the best was chosen.

After addressing these issues we looked at the performance and found that it was suboptimal. [Keromytis et al, 2003] states the "OCF attains 95% of the theoretical peak device performance." These results however are for relatively slow devices and fail to consider the overhead required to reach that performance. Furthermore, the 95% figure is misleading in that it is attained for a case that rarely occurs and where the majority of the overhead of the OCF is hidden by the raw computational cost.

In analyzing the performance of OCF we found several problems that contribute to suboptimal performance for all operations:

1) The OCF requires two kernel thread context switches for each operation. On many systems the peak performance of the OCF is limited by the rate at which the system can do context switches.
2) The OCF forces all crypto operations to pass twice through a single kernel thread. Combining the dispatch and return processing in a single thread increases latency.
3) Crypto device drivers trade off throughput for latency. This has a detrimental effect on the performance of network protocols where latency is critical.

Furthermore it was observed that when crypto devices are overloaded the OCF reacts by discarding operations instead of applying flow-control techniques. This results in severe performance loss for network protocols as this action is equivalent to discarding packets.

With these problems corrected FreeBSD was found to outperform OpenBSD on identical hardware by as much as 100% in tests that exercise only the cryptographic framework. The overhead of using the cryptographic facilities was dramatically reduced raising the peak performance and making more of the CPU available for non-cryptographic work. These optimizations result in similar performance improvements for facilities like IPsec that make heavy use of the cryptographic framework. For example, FreeBSD's Fast IPsec [Leffler, 2003] outperforms OpenBSD's IPsec implementation [Miltchev et al, 2002] by more than 50% on identical hardware running **netperf** over a 3DES+SHA1 tunnel.

The remainder of this paper is organized as follows. Section 2 describes the cryptographic framework and discusses the effort needed to make it operate in a fully preemptive environment. Section 3 discusses performance issues, starting with the tools used to evaluate performance and progressing through each of the issues that were identified in the software. Section 4 describes performance results for FreeBSD and compares them to OpenBSD. Section 5 outlines the the status and availability of this work and talks about future work. Section 6 gives conclusions.

2. Porting the Cryptographic Framework to FreeBSD 5.0

The OCF is comprised of three components:

1) A core set of code that manages a registry of crypto device drivers, dispatches crypto operations to drivers, and coordinates the return of results from drivers to the submitter.
2) Crypto device drivers that submit crypto operations to hardware devices and return results to the crypto core.
3) The **/dev/crypto** pseudo-device driver that provides linkage between user-level software and the core crypto support.

The core crypto support and the **/dev/crypto** driver are simple pieces of software. The crypto device drivers however represent a significant development effort. Some aspects of cryptographic device hardware that vendors refuse to disclose have been reverse-engineered and this work is embodied in these drivers. In considering the integration of the OCF into FreeBSD it was important to maintain the driver API so that existing drivers could be easily reused and so that ongoing development could be shared by both OpenBSD and FreeBSD communities.

The initial port of the OCF to FreeBSD 5.0 was reasonably straightforward. Instead of using processor priority to insure critical code are not executed concurrently, locking primitives were used to guard concurrent access to data structures. This resulted in certain constructs being recast and some code being rewritten. For example, the OCF blocks concurrent access to all its data structures by raising the processor priority to **splimp**. This effectively blocks all concurrent activity in the lower half of the kernel from reentering the code. In FreeBSD 5.0 however splimp does nothing; instead concurrency primitives such as a **mutex** must be used. But simply replacing splimp usage with a single mutex does not work unless the mutex semantics are relaxed to permit recursive acquisition. Instead each of the data structures used by the cryptographic framework were given their own mutex and some code was reorganized to insure consistent lock ordering is used to avoid deadlocks. The end result is easier to understand and permits greater concurrency. The only downside is that mutex primitives have more overhead than simply manipulating the processor priority level. In practice this additional overhead is swamped by other costs and not critical to overall performance.

3. Performance Analysis

This section briefly describes the tools and techniques used to evaluate performance and then discusses the performance problems that were identified.

3.1. Tools

OpenBSD provides no statistics or other facilities for understanding the performance of the OCF. The only tools for evaluating performance treat the system as a ‘‘black box’’; e.g. the openssl tool from the OpenSSL distribution.

We began by instrumenting the core software and each crypto device driver. Every error or failure is accounted for separately so problems can be determined immediately. (This is especially important for understanding problems reported by naive users.)

Next, tools were developed to exercise the crypto functionality. One such tool is the **cryptotest** program that submits symmetric-key crypto operations through the **/dev/crypto** device and verifies the results. cryptotest measures performance over a range of operations and parameters and is a basic tool for evaluating software changes and hardware configurations[1].

Finally a profiling facility was added that timestamps crypto requests as they pass through the system. At important points in the processing of crypto requests these timestamps are used to calculate minimum, maximum, and average time spent doing each task. The cryptotest program has a −p option that enables collection of profiling statistics over the duration of a run.

	Time (ns)			
Task	Avg	Min	Max	Description
Dispatch	115	114	640	Delay before op is handed to driver.
Invoke	155883	154143	193937	Delay before op is returned by driver.
Done	2295	2196	18342	Delay before callback method is invoked.
Callback	341	323	733	Duration of callback method processing.

Table 1: *Sample crypto profiling results*

Table 1 shows the results from running cryptotest on a system with a Broadcom BCM5822 accelerator card. The machine has an Asus P4B533-V Intel845G motherboard with 1.8 GHz P4 processor. The Broadcom card was in a 32-bit/33MHz PCI slot. The tests performed 3DES calculations on 8192-byte buffers. Four values are calculated; one for each of the tasks done by the core crypto framework. The **Invoke** value is especially interesting as it gives an approximation of the peak processing capability of the hardware (modulo the overhead of the device driver). In this case 8 kilobytes were processed in an average of 156 milliseconds for a peak bandwidth of 410 Mb/s. Profiling adds a fixed overhead to each operation; reducing performance results by 2-8%. This facility is described in more detail in Section 4.

[1] The cryptotest program described here is derived from code provided by Theo de Raadt.

3.2. Problems

The normal flow of crypto operations in the OCF is as follows:

1) Client formulates a crypto operation and submits it through the ‘‘crypto_dispatch’’ routine. This routine places the request on a ‘‘dispatch queue’’ and notifies a kernel thread.
2) The kernel thread removes the operation from the dispatch queue and calls ‘‘crypto_invoke’’ to hand the request off to the appropriate crypto device driver.
3) The crypto device driver processes the operation. Typically this is done by submitting one or more commands to the hardware device. Some drivers however may process the data immediately; e.g. the software crypto driver.
4) When the crypto request is completed the crypto driver calls the ‘‘crypto_done’’ routine to notify the crypto subsystem the request is done. This routine places the request on a ‘‘return queue’’ and notifies a kernel thread.
5) The kernel thread removes the operation from the return queue and invokes the callback method associated with the request.

This scheme requires two kernel thread context switches to process each cryptographic request. The trip through the kernel for dispatch is done so that batching can be carried out [Keromytis, 2003]; except the OCF does not support batching. The second trip through the kernel thread for returning requests is done because the callback methods typically take a long time to run. By moving this work from the device driver (typically in the interrupt service routine) this long-running work can be done at a low interrupt priority level. The problem with this is that *all* callback methods are handled in this way, so even those methods that do very little incur this overhead.

For many systems the peak performance of the OCF is limited by number of context switches the system can perform. Reducing this overhead is therefore important.

3.2.1. Separating Dispatch and Return of Results

The first observation was that doing dispatch and return processing in a single kernel thread was suboptimal. While the two tasks were interleaved this still introduced latency as return processing is potentially very time consuming and could block the dispatch of operations to the hardware. Splitting the work into separate threads eliminated contention for shared resources and reduced latency, especially under FreeBSD 5.0 where the kernel is preemptive. Furthermore, under FreeBSD 5.0 doing both tasks in a single thread required that access to both the dispatch and return queues be synchronized with one lock which increased contention further. When dispatch and return processing was split into separate threads IPsec performance increased by 10-15%.

3.2.2. Eliminating Context Switch Overhead

Next it was observed that dispatch processing does not need to be done in a kernel thread [Stone, 2002]. Replacing the kernel thread with a software interrupt thread reduces the cost to enter the dispatch loop by more than 50% on FreeBSD 4.8.

	Dispatch Time (ns)		
	SWI	Thread	Speedup
Avg	427	1519	3.6x
Min	380	1292	
Max	9248	12585	

Table 2: *Software interrupt- vs thread-based dispatch*

Table 2 shows measurements of the dispatch time collected with cryptotest and profiling on the Asus-based test machine. We also tried to convert the return processing thread to a software interrupt but this failed because the callback methods were not prepared to be entered at an elevated priority level.

The main difficulty with converting the dispatch thread from a kernel thread to a software interrupt thread was managing the interrupt masks and synchronization with other threads of execution in the kernel. A new ‘‘spl’’, **splcrypto**, was defined that blocks both the software interrupt thread, crypto hardware interrupts, and all networking software interrupts. This is used within the core crypto code to guard access to data structures that are used in the software interrupt thread.

After converting the dispatch procedure to a software interrupt thread it was observed that most operations can be dispatched without a context switch at all! Recall that the reason for switching to another thread is to have a common location to batch operations. However most crypto requests should not be batched because batching them increases latency, reducing overall performance. This is especially true of network protocols such as IPsec.

To handle this conflict between a need for low latency and a desire for high throughput we introduced the notion of **batchable** crypto requests. Requests that are marked batchable are queued and dispatched from the dispatch thread that is entered through a software interrupt. Crypto drivers are supplied a hint at step 2 in the above procedure that indicates whether more operations will follow immediately; this permits drivers to safely batch operations together. Operations that are not marked batchable are sent to the crypto drivers immediately without passing through the dispatch thread. Doing this requires no changes to the dispatch thread.

	Dispatch Time (ns)		
	Direct	SWI	Speedup
Avg	108	427	4x
Min	99	380	
Max	1634	9248	

Table 3: *Direct- vs software interrupt--based dispatch*

As Table 3 shows, direct dispatch lowers dispatch overhead dramatically. In addition, variance (not shown) is significantly reduced which is important for clients like network protocols.

While the above techniques eliminate the context switch needed to dispatch operations they do not eliminate the context switch prior to invoking the callback method. As discussed earlier, the callback method may take a long time to execute so typically should not be run directly from the crypto device driver. Some callback methods however execute quickly. In particular the callback method for the **/dev/crypto** device driver does little more than wakeup the thread that submitted the request. This callback can safely be called from the device driver without switching to the crypto return thread. Crypto requests that want their callback methods invoked directly mark their request appropriately. This eliminates the context switch normally required to return results. Clients however must be careful when submitting requests as the results may be ready before they can wait for them. (To aid in recognizing this, the crypto framework now marks crypto operations **done** on completion so clients can submit an operation, synchronize access to the operation data structure, and then verify a request is incomplete before blocking to wait for a result.) With this change operations submitted through **/dev/crypto** can be done without any context switches! Table 4 compares the cost of using

immediate callbacks to passing through the return thread.

	Return Time (ns)		
	Immediate	Task	Speedup
Avg	98	3102	33x
Min	97	3010	
Max	455	15098	

Table 4: *Immediate- vs task--based returns*

Similar to the above technique, when a crypto device driver operates synchronously (e.g. the software crypto driver), passing callbacks through the return thread is typically unnecessary. When a caller is prepared for immediate callback with a result it can mark operations appropriately and bypass the trip through the return thread. This last optimization effectively returns the overhead of using software crypto support to the cost of a function call. The only downside to this technique is that because the callback is done in a continuation style there are two extra stack frames required to process a request to completion, which may be significant for a kernel that is running close to the limit of its kernel stack size.

Note that all these optimizations reduce the *overhead* of using the crypto subsystem. This is most noticeable for operations with small operands as the cost to do the cryptographic operation is small compared to the overhead to process it. As operand size grows the time spent computing the result reduces the importance of the optimizations described here. However since operand size typically exhibits a bi- or tri-modal distribution these optimizations have a significant impact on real-life performance.

3.2.3. Tuning Drivers to Minimize Latency

Two crypto device drivers were imported from OpenBSD when this work was brought into FreeBSD. These drivers support the hardware devices most readily available: those based on the Hifn 7951, Hifn 7811 and Broadcom 58xx. Both drivers attempt to batch symmetric crypto operations when possible. Typically this is possible only when the device is under load. It was observed however that this batching has a severe performance penalty for applications like IPsec because it increases the latency for most operations. The drivers have been changed to batch operations only if they are explicitly marked batchable.

3.2.4. Flow Control for Cryptographic Operations

The OCF assumes that once an operation is sent to a crypto device driver for processing it will either succeed or fail permanently. Failures due to lack of resources are treated as permanent failures. Callers may interpret the error code for failed operations and resubmit them but this is never done. This has ramifications for clients of the OCF. For IPsec, for example, a failed crypto request is equivalent to dropping a packet. This can cause higher level protocols such as TCP to initiate backoff algorithms and for performance to drop.

In many instances drivers can quickly recover from conditions that are reported to the OCF as errors. For example, the Hifn 7951 has a small ring of descriptors that may be exhausted when under load. Since the ring is known to be full when an error occurs one can assume an operation will complete soon and space will be available to submit a new operation. Rather than fail a request because there are no descriptors it is more efficient to queue it and ‘‘push back’’ on the crypto dispatch logic until resources become free. While this can cause excessive queueing of requests, in practice, this does not happen because other flow control mechanisms come into play (e.g. network traffic stops flowing).

Crypto device drivers may return an **ERESTART** error when given a request they cannot process. This should be used only when there is a shortlived resource exhaustion. The crypto framework will queue the request and mark the driver as ‘‘blocked’’ so subsequent requests will be queued and not submitted to the driver. The driver must determine when it is ready to accept requests again and call ‘‘crypto_unblock’’ to clear the blocked condition on the driver. The **ubsec** (Broadcom) and **hifn** (Hifn 7951, 7811, etc.) drivers both implement this flow control scheme. Statistics indicate this condition happens frequently for the Hifn 7951 when under load. To demonstrate this the cryptotest program was run with 28 threads concurrently submitting operations to a single 7951-based card (a Soekris vpn1201 PCI card installed in the Asus-based test machine).[2] Over the course of the test run 51% of the requests required restarting because of temporary resource exhaustion (no space in the command/result rings). Without this flow control mechanism these operations would have failed. If the operations were submitted on behalf of IPsec they would have caused packets to be dropped.

[2] Note that the number of threads had to be significantly inflated because the test was run with an optimized system. This behaviour was initially encountered with an unoptimized system running IPsec under a three-client load.

4. Performance Results

To evaluate the changes described in this paper numerous tests were conducted. The results presented here were mostly collected using the cryptotest tool described above. A few results from higher-level applications such as IPsec are also presented; a more in-depth discussion of those results is found in [Leffler, 2003]. Note that while cryptotest was used to collect most of the results presented here, the data were validated using other tools such the openssl program. After more than a year of testing and tuning cryptotest is considered a reliable indicator of system performance.

Table 5 shows results from running cryptotest on the Asus-based test machine running FreeBSD 4.8 with a variety of hardware devices, while Table 6 shows the results using OpenBSD 3.3 (release) on the same test machine and cryptographic hardware devices.[3]

Operand	7951			7811			5822		
Size	3DES	MD5	SHA1	3DES	MD5	SHA1	3DES	MD5	SHA1
8	1.9	1.9	1.5	1.9	1.9	1.9	3.0	2.9	2.9
16	3.8	3.8	3.0	3.9	3.8	3.8	6.0	5.9	5.8
32	7.6	7.6	6.0	7.6	7.6	7.7	11.7	11.6	11.5
64	14.8	12.0	11.9	15.3	15.2	15.1	22.6	22.4	22.1
128	23.1	22.2	18.9	28.5	28.3	27.0	41.4	42.9	42.2
256	36.2	35.2	31.0	49.7	50.9	41.8	74.9	80.1	78.7
512	51.2	54.2	46.3	75.7	72.3	72.3	128.0	141.5	138.9
1024	63.1	69.8	59.5	99.3	109.0	104.7	205.5	235.1	232.1
2048	71.3	79.2	69.5	120.6	143.9	134.7	282.1	348.1	344.5
4096	76.4	88.2	75.8	136.5	170.9	157.3	346.9	457.2	454.4
8192	78.4	93.5	79.2	145.1	188.7	170.9	380.2	531.9	530.0

Table 5: *Cryptotest results for FreeBSD 4.8*

The OpenBSD results are comparable to where FreeBSD was before any of the optimizations described in this paper were done.

Table 7 directly compares the FreeBSD and OpenBSD results for 3DES on the three devices. As the operand size increases the processing time of the computation reduces the effect of the lower submission overhead. For faster hardware however this is not true because the host is not fully utilizing the cryptographic hardware. For operand sizes less than 2048 bytes FreeBSD outperforms OpenBSD by at least 70% for the fastest hardware device and typically it outperforms OpenBSD by at least 50%.

[3] OpenBSD 3.3 does not work correctly on the test machine with the Broadcom BCM5822 unless APM support is disabled. Before this was discovered each cryptographic operation took about 1 second to complete! However, even with this workaround hardware-assisted IPsec does not function correctly.

Operand	7951			7811			5822		
Size	3DES	MD5	SHA1	3DES	MD5	SHA1	3DES	MD5	SHA1
8	1.2	1.1	1.1	1.2	1.2	1.1	1.5	1.3	1.5
16	2.5	1.9	2.1	2.2	2.4	2.3	3.1	3.0	2.2
32	4.6	3.9	4.2	4.6	4.8	4.0	6.1	6.0	5.3
64	9.2	8.3	7.5	9.1	8.7	7.6	11.8	11.8	11.7
128	16.2	12.8	14.0	18.6	16.6	15.3	22.7	22.0	20.6
256	27.3	24.1	24.0	32.9	30.8	27.6	43.0	40.5	41.2
512	41.5	40.5	38.1	53.9	52.9	54.1	72.9	77.6	80.6
1024	54.0	57.5	52.3	80.4	87.3	84.3	119.1	147.7	127.3
2048	65.9	73.0	64.3	106.5	123.3	116.5	199.6	241.3	239.8
4096	71.5	84.1	72.5	124.1	149.6	142.3	282.3	356.6	355.0
8192	76.8	91.1	77.3	138.9	176.2	161.9	333.1	436.8	459.6

Table 6: *Cryptotest results for Openbsd 3.3 (release)*

Operand	7951			7811			5822		
Size	OBSD	FBSD	%diff	OBSD	FBSD	%diff	OBSD	FBSD	%diff
8	1.2	1.9	58	1.2	1.9	58	1.5	3.0	100
16	2.5	3.8	52	2.4	3.9	62	3.1	6.0	93
32	4.6	7.6	65	4.6	7.6	65	6.1	11.7	91
64	9.2	14.8	60	9.1	15.3	68	11.8	22.6	91
128	16.2	23.1	42	18.6	28.5	53	22.7	41.4	82
256	27.3	36.2	32	32.9	49.7	51	43.0	74.9	74
512	41.5	51.2	23	53.9	75.7	40	72.9	128.0	75
1024	54.0	63.1	16	80.4	99.3	23	119.1	205.5	72
2048	65.9	71.3	8	106.5	120.6	13	199.6	282.1	41
4096	71.5	76.4	6	124.1	136.5	9	282.3	346.9	22
8192	76.8	78.4	2	138.9	145.1	4	333.1	380.2	14

Table 7: *3DES performance comparison on Asus P4B433-V*

To understand the importance of these optimizations on slower systems the cryptotest measurements were repeated on a Dell XPS 300 system (300 MHz Intel PII processor and 64 Mbytes of memory) using each of the hardware crypto devices. Again FreeBSD 4.8 and OpenBSD 3.3 were used.

Operand	7951			7811			5822		
Size	OBSD	FBSD	%diff	OBSD	FBSD	%diff	OBSD	FBSD	%diff
8	0.7	1.1	57	0.7	1.1	57	0.9	1.1	22
16	1.5	2.1	40	1.5	2.1	40	1.7	2.3	35
32	3.0	3.9	30	3.0	4.3	43	3.4	4.5	32
64	5.5	7.4	34	5.9	8.0	35	6.6	8.9	34
128	10.4	13.6	30	11.6	14.7	26	12.6	17.6	39
256	18.1	22.6	24	19.9	26.5	33	24.1	32.6	35
512	28.9	35.2	21	34.6	44.0	27	43.3	59.3	36
1024	40.2	46.6	15	52.9	64.2	21	71.3	93.4	30
2048	51.3	56.8	10	73.3	86.3	17	106.2	134.1	26
4096	60.2	62.8	4	92.8	100.1	7	149.5	169.7	13
8192	65.1	66.7	2	105.8	110.0	3	178.8	191.7	7

Table 8: *3DES performance comparison on Dell XPS 300*

As Table 8 shows, the performance difference between the two systems is less significant. This

appears to be due to the higher overhead to setup I/O operations (e.g. slower bus) reducing the direct benefit of using an outboard cryptographic accelerator. There are also some anomalous results that may be caused by inaccuracy in the timing data (as before, it was necessary to disable APM support in OpenBSD to get it to process cryptographic operations in a reasonable manner).

Finally, Table 9 shows the result of running the **netperf** performance testing tool [Jones, 2003] between the Asus-based test machine and a second machine with a Tyan S2707G2N motherboard and a 1.8 GHz P4 processor. Both machines used dedicated Intel gigabit Ethernet NICs; the Asus machine had an 82540 NIC located in a 32-bit PCI slot, while the Tyan system used an 82545 NIC with a 64-bit PCI interface. Both systems used Broadcom BCM5822 cards: on the Asus machine the crypto cards were in a 32-bit PCI slot; on the Tyan machine the cards were in a 64-bit PCI slot. The systems were physically connected by a cross-over cable. First the netperf tcp_stream_script script was used to collect data on an unencrypted connection for a variety of socket sizes. Next, a 3DES+SHA1 IPsec tunnel was setup using static keying and the same tests were rerun with cryptographic calculations done in software on the host or by the 5822. The figures in the table were reported with a 99% confidence interval (as determined by netperf). Note that no comparable results for OpenBSD were available for the 5822 because the tests failed to complete.

System	Sender	MsgSize	Throughput (10^6bits/sec)		
			Raw	S/W	5822
OBSD	Asus	4096	541	27	-
FBSD	Asus	4096	624	42	160
OBSD	Asus	8192	530	27	-
FBSD	Asus	8192	624	42	160
OBSD	Asus	32768	519	27	-
FBSD	Asus	32768	627	42	159
OBSD	Tyan	4096	648	27	-
FBSD	Tyan	4096	797	44	170
OBSD	Tyan	8192	650	27	-
FBSD	Tyan	8192	798	44	170
OBSD	Tyan	32768	653	27	-
FBSD	Tyan	32768	799	44	169

Table 9: *Netperf test results*

Interpreting these results without careful inspection of the way these two IPsec implementations work is difficult. Nonetheless some useful information can be gleaned.

First, remember that the Tyan-based system had both NIC and Broadcom devices in 64-bit PCI slots while the Asus-based system has only 32-bit PCI support. This difference is noticeable in the netperf results and also when running non-network tests like cryptotest.

In lieu of results for the 5822, the software-based crypto results are the most useful in comparing performance of the crypto subsystems. FreeBSD is 55% faster than OpenBSD on the Asus machine and 62% faster on the Tyan machine. Remember that OpenBSD requires two context switches for each crypto operation; this accounts for much of the performance difference. Otherwise the different results between the two machines is due to the 32- versus 64-bit PCI NIC for the sender.

Netperf results tend to be dominated by the performance of the sender. When using the 5822 the FreeBSD system was typically about 30% idle on the sender and 40% idle on the receiver (according to the vmstat program). Statistics maintained by the FreeBSD Broadcom driver show that on the slower system (the Asus-based machine) a peak of 9 crypto operations were pending completion by the hardware and 20 operations pending in the system (queued waiting for the hardware to have room to accept another operation). On the Tyan-based system these numbers were 6 and 9. Furthermore, 78% of the crypto requests processed by the sender found the hardware device busy and unable to accept a new request. Together these numbers indicate the performance was limited by the crypto hardware.

Running netperf end-to-end over an IPsec connection does not show the full possible performance because the processing is heavily influenced by the location of the sending process. For example, on the sender the data originates in a TCP socket and so must be copied for transmission. Further, when forming an IPsec packet on the sender multiple mbufs are typically created requiring the crypto driver to process fragmented operand data. By contrast, when a system is operating as an endpoint of an IPsec gateway, packets do not originate or terminate on the machine. Instead IPsec packets are received, processed, and then forwarded to their destination. In this configuration the data is almost always contiguous in memory and does not require copying for potential retransmission. Consequently load on the crypto hardware is very different, and utilization is typically 100%.

Finally, it has been observed that to fully utilize fast crypto hardware such as the Broadcom 5822, CPU performance (for Intel-based systems at least) is important. Results for the netperf test described above scale almost directly to the clock speed of Intel P4 and Xeon™ processors up to 2.53 GHz (the

maximum clock speed that was available at the time these tests were done). With 2.4GHz processors end-to-end netperf results with 5822 support exceed 200 Mb/s.

5. Status and Future Work

The initial port of the OCF to FreeBSD was done in September 2001. Integration of this work into FreeBSD was completed November 2002 and committed to the stable branch a month later. The work described here is freely available as part of the FreeBSD 5.0 and 4.8 releases. It is currently being integrated into the NetBSD operating system [Stone, 2003]. Several vendors have incorporated the framework in commercial products.

The techniques described in this paper have not been applied to asymmetric crypto operations. Applying them is straightforward and should yield significant performance improvements.

Otherwise, future work falls into two broad categories: improved device support and load balancing. The current software supports only a few hardware devices and some of these have underwhelming performance or are too expensive and/or too difficult for the average consumer to obtain. The main impediment here is the non-disclosure of programming information. New products are expected that will address both of these concerns.

Otherwise, the most significant deficiency in the current framework is the inability to use protocol-specific hardware operations. Most vendors of crypto hardware optimize their products for use as ‘‘all-in-one’’ devices that take an IPsec/SSL/TLS packet and parse the protocol and perform the cryptographic transforms in a single request. This is incompatible with the current general-purpose API provided by the OCF. Supporting these kinds of operations requires exposing state that is currently private to the protocols. However adding this is the only way that some hardware devices can be used at all as they do not otherwise provide access to the cryptographic transformation hardware.

Load balancing refers to efficiently supporting multiple crypto devices in a single system. These devices may be add-in devices or dedicated CPU’s in a multiprocessor system. The OCF does very little in this regard; when multiple devices are present in a system it will assign them in a round-robin fashion when creating **sessions** for symmetric crypto operations. Asymmetric operations do not consider multiple devices; they are dispatched to the first available device that can handle the request. In addition it has been understood for a while that the setup overhead for communicating with hardware devices can be too high to justify submitting small data buffers. Instead such operations should be handled by the main CPU. All these issues requires similar functionality. Experimental work is ongoing to calculate operation-specific cost metrics for cryptographic devices and then use that to dynamically schedule operations as they are submitted. This scheme handles symmetric and asymmetric operations and includes the software crypto device driver so that, for example, operations on small data buffers can be processed in software when appropriate.

6. Conclusions

FreeBSD has imported the OpenBSD Cryptographic Framework. In the process the core code has been rewritten to work correctly in a preemptive environment. We have added statistics and developed tools to aid in understanding system performance and to help in the event of problems. Several deficiencies in the OCF have been identified and corrected. These changes result in a system that performs as much as 100% faster than the OCF on identical hardware. For facilities like IPsec that make heavy use of the cryptographic framework, these changes can improve performance by more than 50%.

We conclude that the OCF cryptographic API introduced by [Keromytis et al, 2003] can be optimized and re-tuned to deliver substantially better performance than the original OCF implementation with large gains in both throughput and latency. Moreover these changes can be made with no impact on clients of the cryptographic framework: both user and kernel software designed for the original OCF is easily ported to the FreeBSD implementation of OCF.

7. Acknowledgements

This work is derived from the OpenBSD cryptographic framework; without it this work would not have been possible. Jason Wright and Angelos Keromytis of were especially helpful in understanding certain design decisions in the OCF. Theo de Raadt provided the original code from which cryptotest was created. Jonathan Stone first suggested that there was excessive overhead in the OCF; this motivated me to attack the problem.

I am especially grateful to Vernier Networks for funding much of this work and providing access to their equipment. Global Technologies Group, Inc. (GTGI) provided two Xl-Crypto (Hifn 7811) boards and funded FreeBSD device support for their cards. Soren

Kristenson of Soekris Engineering provided two vpn1201 cards (Hifn 7951).

References

Dierks & Allen, 1999.
T. Dierks & C. Allen, "The TLS Protocol Version 1.0," *RFC 2246* (January 1999).

Eastlake, 1999.
D. Eastlake, "Domain Name System Security Extensions," *RFC 2535* (March 1999).

Freier et al, 1996.
A. Freier, P. Karlton, & P. Kocher, "The SSL Protocol Version 3.0," *http://home.netscape.com/eng/ssl3/draft302.txt* (November 1996).

Gattaneo et al, 2001.
G. Gattaneo, L. Catuogno, A. Del Sorbo, & P. Persiano, "The Design and Implementation of a Transparent Cryptographic Filesystem for UNIX," *FREENIX Track: Usenix Annual Technical Conference* (June 2001).

Jones, 2003.
R. Jones, "Netperf 2.2pl3: network performance evaluation tool," *http://www.netperf.org* (February 11, 2003).

Kent & Atkinson, 1998.
S. Kent & R. Atkinson, "Security Architecture for the Internet Protocol," *RFC 2401* (August 1998).

Keromytis, 2003.
A. Keromytis, *Private email* (January 2003).

Keromytis et al, 2003.
A. Keromytis, J. Wright, & T. de Raadt, "The Design of the OpenBSD Cryptographic Framework," *USENIX Annual Technical Conference* (June 2003).

Kohl & Neuman, 1993.
J. Kohl & C. Neuman, "The Kerberos Network Authentication Service (V5)," *RFC 1510* (September 1993).

Leffler, 2003.
S. Leffler, "Fast IPsec: A High Performance IPsec Implementation for UNIX Systems," *BSDCon 2003* (September 2003).

Miltchev et al, 2002.
S. Miltchev, S. Ioannidis, & A. Keromytis, "A Study of the Relative Costs of Network Security Protocols," *FREENIX Track: Usenix Annual Technical Conference*, p. 41−48 (June 2002).

Provos, 2000.
N. Provos, "Encrypting Virtual Memory," *Proceeding of the USENIX Security Symposium* (August 2000).

Stone, 2002.
J. Stone, *Private email: Re: anyone working on crypto processor support?* (November 2002).

Stone, 2003.
J. Stone, *Private email: preliminary port of sys/opencrypto to NetBSD* (March 2003).

Ylonen et al, 2002.
T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, & S. Lehtinen, "SSH Protocol Architecture," *Internet WIP draft-ietf-secsh-architecture-12.txt* (January 2002).

Biography

Sam Leffler has been actively working with UNIX since 1975 when he first encountered it at Case Western Reserve University. While working for the Computer Systems Research Group (CSRG) at the University of California at Berkeley he helped with the 4.1BSD release and was responsible for the release of 4.2BSD. He has contributed to almost every aspect of BSD systems; most recently working (again) on the networking subsystem. You can contact him via email at <sam@errno.com>.