Analysis of Cache Timing Attacks against AES

Robert G. Salembier

Abstract— **Side channel information is a growing problem in the area of cryptography. A cache timing attack against an OpenSSL implementation of AES has been verified. Timing characteristics of this attack are documented and analyzed. Potential real world attack strategies are presented with both positives and negatives. Possible methods of mitigating against cache timing attacks are identified and critiqued.**

Index Terms—**Side-channel attacks, cache timing, AES**

I. Introduction

Traditionally attacks on cryptography are conducted against the math of the cryptographic system. Examples include differential and linear cryptanalysis. A few categories of such attacks are: ciphertext-only, known plaintext, or chosen plaintext attacks. These methods rely either solely on the ciphertext, on both the plaintext and ciphertext, or the ability to define plaintext that is encrypted and the analysis of the resultant ciphertext. Today, it has been realized that encryption devices provide attackers more information than previously assumed. These devices have a tendency to unintentionally reveal more information about the cipher than just the ciphertext. This extra information is known as side channel information and is defined as "information that can be retrieved from the encryption device that is neither the plaintext to be encrypted nor the ciphertext resulting from the encryption process."[1]

With the ability to gather side channel information, a new class of attacks called side channel attacks has been developed. These attacks include timing attacks, power consumption attacks, fault analysis attacks, and acoustic attacks. One of the leaders in this field, Kocher, has pioneered certain timing attacks, simple power analysis attacks, and differential power analysis attacks. [2,3,4] Each of these attacks is described in the aforementioned references and the authors have implemented them against real systems. It has been said that side channel attacks make certain assumptions about the hardware, the message being sent, or another piece of vital information. This class of attack is important and should be thoroughly researched and attempts be made to mitigate the ability to use this information leakage to help determine keys used to encrypt data.

Manuscript received May 12, 2006.

R. G. Salembier is a graduate student in Computer Engineering at George Mason University, Fairfax, VA 22030 USA (e-mail: rsalembi@gmu.edu).

When National Institute of Standards and Technology (NIST) held the contest to find a new encryption standard to replace the current Data Encryption Standard (to be termed Advanced Encryption Standard), they were aware of this new class of attacks and considered it in their evaluation of the security of the AES candidates. When evaluating attacks against implementations of candidates, NIST made specific mention of timing and power analysis attacks in section 3.6 of [5]. Additionally, NIST stated in section 3.6.2 of [5] that table lookups are not vulnerable to timing attacks. The AES contest winner Rijndael, along with other AES candidates and block ciphers, use large table lookups to increase the throughput of their algorithms. Unfortunately, a major problem with this method is that it allows for a specific timing attack to be carried out against a system that relies on table lookups. The attack obtains the information leaked by cache misses. This paper focuses on a specific implementation of a cache timing attack against AES and identifies potential methods in which this type of attack may be mitigated.

II. Bernstein's Attack

Bernstein has capitalized on the fact that many block ciphers, specifically AES, leak timing information during cache hits and misses. [6] The cache based timing attacks focus on observing data flow, in and out, of the different levels of cache in the implementation system, in order to recover sufficient information to reconstruct the key being used. This class of attack begins by collecting a baseline level of information, commonly referred to as a profile, of the system. All other data is compared against this profile. Kocher first brought the concept of using cache behavior as a tool to gather information to light. [2]

Bernstein's attack uses timing analysis to reconstruct an AES key based on cache timing information. In order for AES to get the speed that was shown during the contest held by NIST, the optimized version of the algorithm employed 4 1024-byte tables: T_0, T_1, T_2, and T_3. This corresponds to 4kB of information that needs to be stored in cache in order for no misses to occur. The attack takes advantage of the fact that even if the cache is large enough to hold all four tables, other processes will eventually need the cache, therefore potentially causing cache misses. Cache misses are what cause encryption to occur at a variable rate.

The algorithm implementation has input bits, XORed with key bits, used as index values to the aforementioned tables. Bernstein's attack exploits the fact that varying the input bits will change the timing of that encryption. The attack is broken down into three phases: profiling, attacking, and analysis. Attacking and analysis may be conducted in parallel.

The profiling phase takes place by sending random 400, 600, and 800 byte packets to a server using the AES encryption algorithm. The key at the server is comprised of all zeros. Information is recorded to a file to determine the profile of the system as it reacts to the different size packets (400, 600, and 800). Approximately 2^{22} packets of each size are sent to the server to determine the baseline of the system being attacked.

Attacking is conducted in a similar fashion as profiling, except for the first step. Initially, the server is set up with a random key, unknown to the attacker. Then the attacker sends a packet, of all zeros, to the server to obtain the resultant ciphertext. After that, the entire profiling process is conducted again with 400, 600, and 800 byte packets to gather timing variations. The number of packets necessary, as stated by Bernstein [6], was 2^{27} – 400 byte packets, 2^{25} – 600 byte packets, and 2^{25} – 800 byte packets.

The analysis portion of the attack is a correlation of the baseline profile, with a known key of all zeros, to the data gathered with an unknown key. Thus, this analysis outputs potential keys that produce large correlations. Finally, given the ciphertext gathered by the first step of the attack phase, a brute force attack is conducted against all potential keys. In some cases the potential keys, for a particular hexadecimal segment, may be reduced to only one after the correlation analysis is conducted.

III. Verification of Attack

The purpose of the following experiments was to verify the cache timing attack documented and carried out by Bernstein [6]. Experiments were not conducted in a manner to duplicate the attack and test environment used in the original attack, but to verify the general ideas and notions behind the attack. Five complete verification tests were conducted; three of them successfully recovered the AES key of the server.

A. Testing Environment

Server:
Windows XP Service Pack II
AMD Sempron 1.50 GHz, 496 MB RAM
L1 cache 128 KB
L2 cache 256 KB

Running Cygwin version 1.5.19-3
gcc version 3.4.4
OpenSSL 0.9.8a 11 Oct 2005

Attacker:
Gentoo Kernel version 2.6.12 revision 6
AMD Athlon XP 2.00 GHz

gcc version 3.3.6
OpenSSL 0.9.7.e 25 Oct 2004

Figure 1. Testing Environment

No attempt was made to replicate the server or attacker environment from the original attack. Therefore, the systems used were similar to those used by Bernstein, but not exact copies. The reason behind this was to illustrate that the idea should work no matter the operating system, hardware, or cryptographic libraries used. As illustrated in the testing environment, **Figure 1,** neither the server nor attacker systems are the same as those used in the documented attack [6], but they are similar to illustrate that hardware, compiler, or library versions are not determining factors in the attack, although they may play some role in the amount of time taken to carry out the attack.

B. Overview of Tests

A total of five different complete tests of the attack was conducted. Tests were all managed in a similar manner – first the profiling phase, then the attacking phase, followed by the brute force key search. The attacks, like many experiments, evolved via the knowledge obtained from the previous attacks. Additionally, supplemental data was gathered after all five tests that consisted of recreation of portions of previous tests.

1) Test 1

The first test was conducted without really documenting any of the time information, simply to see if the attack worked. Based on the information provided by Bernstein, the time to carry out this attack seemed rather short. The profiling process appeared to go without a problem for the 400, 600, and 800 byte packets. Then the same program was used to attack the server with the varying packet sizes. The documented paper stated approximately 160 minutes to send 2^{25} packets. Ethereal was used from a separate computer system to monitor the packet flow between the two systems. Little packet information was stored with Ethereal, since the only items of concern were the overall time and the number of transiting packets. Correlation analysis was conducted on the information obtained from the attacks against the baseline profile with poor results. There were some correlations, but so few that a brute-force attack on the potential keys was still unrealistic. After analyzing the output files of the profiling and attacking phases, instead of relying on packet number information provided by Ethereal, it was determined that the number of packets actually sent were only about 2^{19}.

2) Test 2

Learning from Test 1, the entire experiment was repeated, except in order to determine when to end either the profiling or attacking phase, the individual files were analyzed (this number is found in column 4 of the output file). The result appeared to be an anomaly. There were strong correlations, but when the brute force key search was conducted, it ran for two days with no results. The correlations were then checked against the key on the server system, and many of the key elements were not present in the correlations. This result was quite perplexing and a plausible reason was needed to explain the outcome. In looking through the logs and evaluating the system information output, it was determined that the clock source of the attacking system was unstable. Although the AES key was not extracted, this test demonstrated the

	study.400	(2.54 days)	study.600	(3.01 days)	study.800	(3.50 days)
	Time (min)	Total Time (min)	Time (min)	Total Time (min)	Time (min)	Total Time (min)
2^{16} (2.1 MB)	57	57	67	67	77	77
2^{17} (2.3 MB)	57	114	67	134	78	155
2^{18} (2.5 MB)	114	228	139	273	159	314
2^{19} (2.6 MB)	228	456	269	542	321	635
2^{20} (2.8 MB)	455	911	544	1086	636	1271
2^{21} (3.0 MB)	911	1822	1088	2174	1265	2536
2^{22} (3.2 MB)	1834	3656	2160	4334	2508	5044

Table 1. Profile Timing Results

importance of a reliable clock when conducting this timing attack. A new scenario came to light from this situation though; one in which the system being attacked had an unstable clock.

3) *Test 3*

The third test was conducted in the same manner as the two previous attacks, except that the roles of the server and attacker were reversed. Therefore, the attacker, now with a stable clock source, was attacking a SSL server with an unstable clock source. The results were as expected and the 128 bit AES key was recovered. This was not a surprise, because the attacked server does not provide any timing information to the attack, but this test did prove that making the clock unstable does not thwart this attack.

4) *Tests 4 & 5*

In the fourth and fifth tests the original setup was used again with a few modifications. The attacking system was rebuilt to include a stable clock. The profiling was conducted for the new setup and extensive timing results were recorded. Then attacks were conducted against two separate unknown 128 bit AES keys. Each attack successfully recovered the key.

C. *Timing Results*

Time recordings were conducted during the profiling stage of tests three, four, and five. The timings were broken down by packet size (400, 600, and 800), and by the number of packets sent. For each packet size, the number of packets sent was 2^{22}. [6] What is seen in **Table 1** is the timing of the profiling phase of the attack averaged over the three final tests. As seen from this table the profiling of a simple SSL server in our test environment takes approximately nine days to complete.

The timing of the attacking phase was documented in a similar manner to that of profiling. Again, the timings were broken down by the packet size of the attack, but as Bernstein states, the number of packets that need to be sent vary based on the size of the random packet being sent. The original attack calls for:

- 2^{27} – 800 byte packets
- 2^{25} – 600 byte packets
- 2^{27} – 400 byte packets

It was apparent from the timing results seen during the profiling stage that a verification that attempted to duplicate this, would take a tremendous amount of time. Bernstein also speculated that the attack could work with a lower number of packets in the attack phase. [6] The timing results seen in **Table 2** are the result of the averaged times over the final three tests for the lowest number of packets sent at each size interval that resulted in a successfully extraction of the AES key. There was a gradual decrease over the final three tests in the number of packets sent; for 400 byte packet size, it was as follows:

- Test 3: 2^{23} packets sent
- Test 4: 2^{22} packets sent
- Test 5: 2^{21} packets sent

It can be calculated from the timing results that in the conditions created by the testing environment, the attack phase takes about 11 1/3 days. **Figure 2** plots the attack timings shown in **Table 2**, with the lines going in increasing order from 400 to 600 and finally 800 byte packet size. Of note is

	attack.400	(1.27 days)	attack.600	(3.02 days)	attack.800	(7.04 days)
	Time (min)	Total Time (min)	Time (min)	Total Time (min)	Time (min)	Total Time (min)
2^{16} (2.1 MB)	57	57	69	69	80	80
2^{17} (2.3 MB)	57	114	67	136	80	160
2^{18} (2.5 MB)	114	228	135	271	160	320
2^{19} (2.6 MB)	228	456	268	539	316	636
2^{20} (2.8 MB)	455	911	544	1083	630	1266
2^{21} (3.0 MB)	911	1822	1103	2186	1266	2532
2^{22} (3.2 MB)			2163	4349	2535	5067
2^{23} (3.4 MB)					5077	10144

Table 2. Attack Timing Results

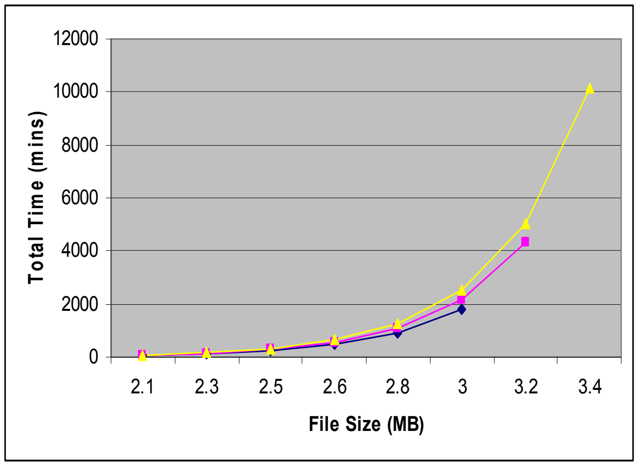

Figure 2. Attack Timing Results

the time increase seen over the course of the increase of the number of packets being collected between packet size intervals. For example, in collecting 2^{16} (65,536) packets of 400 byte packets when compared to 600 byte packets, the increase is 12 minutes; but when collecting 2^{21} (2,097,152) packets, the increase is 364 minutes. From this information, it is apparent that when conducting an attack of this nature, focus should be placed on obtaining the most information possible from the smaller size packet, since it gathers information at the greatest rate of return. If the attack phase was conducted in the same manner as Bernstein outlined, this phase would have taken approximately 216 days. As hypothesized in his work, he believed that the number of packets used may be more than necessary, and this verifies his hypothesis, in this test environment.

In reviewing the timing results received in the test environment, in comparison to those stated by Bernstein, [6] there are some discrepancies. The only place where timing is discussed by Bernstein is when it is stated that when conducting the attack part, with 800 byte packets, 2^{25} random packets should take about 160 minutes. The results obtained in this testing though were quite a bit larger, with only 2^{17} packets taking 160 minutes, and the largest result obtained being 2^{23} packets at 10144 minutes. This presents a potential reason why others that have attempted to reproduce this attack [7,8] ended up with unsatisfactory results. The testing environment used in these experiments attempts to reflect a somewhat real world system in a best case scenario from the vantage point of an attacker. Both the server and attacking system have 100MB Ethernet cards and are connected by CAT5 cable through a Linksys WRT54GP2 router. There are faster Ethernet cards and routers, but the resources were not available.

D. Test Results

From the first test the limitations of the test environment and a better understanding of the time required to execute the attack was learned. The second test enforced the importance of the clock source on the effectiveness of a timing attack. Previously the system was used for daily use, an email and data server; therefore time was not as important a factor. During a timing attack the clock source must be stable enough to allow the ability to find small variances in timing results. These small changes in return times are what allow the correlations between the profile, with a zero key, and the attack on an unknown key to be effective. The third test providing two items of importance. The first was it gave a good indication of how long it takes to mimic the original attack successfully [6] in this test environment, which was a little over a month. The second was a reinforcement of a hypothesis that the stability of the clock source on the attacked server was not important to the attack. Although an unstable clock source on an SSL server may cause other problems to the system itself, no interference to the attack is caused by this.

Tests four and five, along with additional post testing analysis provided the largest amount of information regarding the attack. Since a good idea of the overall time to conduct the attack was already known, a focus was placed on how to improve the attack and what was the smallest number of packets that would still result in a successful key extraction. The methodology used for determining the number of packets sent was a combination of whether the brute force search is effective, the correlation improvement, and the increased time to reach the next packet interval. For this to work, tests four

Test 4		Test 5		
Packet Size	# sent	# sent	Time (-) mins	Days
400 byte	2^{22}	2^{21}	1820	1.26
600 byte	2^{23}	2^{22}	4340	3.01
800 byte	2^{24}	2^{23}	10100	7.01

Table 3. Tests 4 and 5 Parameters

and five were conducted to serve specific purposes. Test four parameters were chosen to determine a stepping point from test three. Test three was conducted and completed by sending 2^{23} 400 byte, 2^{23} 600 byte, and 2^{24} 800 byte packets. The fifth test was conducted to see if another step down in the number of packets at each packet size level would still result in discovering the AES key. These step decreases allowed for correlation and timing analysis to be conducted, thus gathering more information. Two separate keys were used to attempt to make it a more generic test and to ensure that specific key properties were not the cause for the ability to reduce the number of packets sent. When the brute force key search was conducted against both the test four and test five data, the key was retrieved; therefore the decrease met that criteria. Additionally, there is a noticeable decrease in time from test 4 to test 5 as seen in **Table 3**, cutting the attack phase time in half, from twenty-two to eleven days. Correlations were taken at each packet size interval, and they were analyzed against each other. Analysis was conducted on 400, 600, and 800 byte packet sizes, and **Table 4** illustrates the correlation information for the 800 byte packets. Another test was conducted on data gathered after the initial five tests. This test was against the same key as in test five, but the number of packets sent were as follows:

	2^{22}	2^{23}	2^{24}
key section	# possible	# possible	# possible
0	109	16	16
1	96	10	8
2	72	9	6
3	18	4	4
4	8	4	2
5	8	2	2
6	25	4	4
7	64	10	8
8	10	4	4
9	23	4	4
10	10	2	2
11	38	4	4
12	100	10	10
13	16	4	4
14	43	4	2
15	8	2	2

Table 4. Correlation Analysis for 800 Byte Packets

- 2^{20} – 400 byte packets
- 2^{21} – 600 byte packets
- 2^{22} – 800 byte packets

This was the final piece for the analysis of the number of packets that should be sent for all packet sizes. The time saved by the decrease in number of packets is approximately five and a half days. With these correlations from each of the different size packets, a brute force key search was conducted for the time equivalent to the amount of time saved by the number of packets being reduced (5.5 days). After this duration of time the program had not extracted the key. Therefore the conclusion was made that waiting the extra time from the increased number of packets sent was a better option. The number of potential key segments for each key section after correlations were performed is shown in **Table 5**. Test five extracted the full AES key while sending the least amount of packets of each size of all the tests. As is apparent from **Table 5,** none of the potential hexadecimal key sections were reduced to only one possibility. The brute force key attack conducted on the correlations in both test four and test five took two minutes. Considerable time was saved in the attack phase with test 5 over test 4 with no noticeable affect on the final phase of the attack. The variance of potential key section reduction between the different byte size packets is interesting. When looked at as a whole, the data provided to the brute force attack key search is:

Key Section:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Possible Keys:	2	4	4	4	4	2	4	4	4	4	2	2	6	4	4	2

Key sections with the lowest correlations are used for the brute force attack, while all those with larger potential key segments are disregarded. For example, in key section 0, the 400 byte packets resulted in two possible key segments with the 600 and 800 byte packet sizes, which resulted in 16. In this case, the information from the 400 byte packets was used for the brute force key search. Therefore, instead of having 16^{256} possible key segments to sift through as would be the case in a brute force attack on an unknown 128 bit AES key, there are 201,326,592 potential keys which is a more manageable number.

If the different packet sizes are viewed individually, the number of different potential keys for the lowest number of packets sent at each level are:

- 400 byte – 155,021,475,840
- 600 byte – 6,442,450,944
- 800 byte – 75,497,472,000
- Combined – 201,326,592

Additionally, a key search conducted on the correlations of test five would only be successful using the correlations of the 400 byte packet search. Both the 600 and 800 byte packet searches lost the first hexadecimal key segment 75 and therefore, would never output a result. The 800 byte packet attack phase lost the correct key segment as a potential key segment when it went from 2^{21} packets sent to 2^{22}. These different packet sizes are combined to draw out certain correlations on the possible key, and their shortcomings should be overlapped by the strengths of the other packet sizes. In **Table 5,** the smallest number of potential key segments produced across all packet sizes is at key section 12. Key section 0 shows that the 400 byte packet has found just 2 possible key segments, while the other two have found 16, and of those none is the actual key segment. An additional attack phase was conducted against the server with the same key used in test 5 (75 59 70 bd f9 26 f5 96 c7 ee ae 33 d6 e1 b7 8d). During the additional testing, it was found that different attacks on the same key with the same packet size may potentially generate different

Key section	400 byte	600 byte	800 byte
0	2	16	16
1	10	4	10
2	4	4	9
3	4	4	4
4	4	4	4
5	4	2	2
6	8	4	4
7	11	4	10
8	4	4	4
9	4	4	4
10	4	4	2
11	7	2	4
12	12	6	10
13	4	4	4
14	4	4	4
15	4	4	2

Table 5. Test Five Correlation Results

correlations. During test five when the 800 byte packets were used to attack the SSL server, the correlation results for key segment 0 were as follows:

2^{22} – 5e 55 54 5f 5b 5a 59 58 5d 5c 51 53 57 52 56 50 75 …
2^{23} – 5e 55 54 5f 5b 5a 59 58 5d 5c 51 53 57 52 56 50
2^{24} – 5e 55 54 5f 5b 5a 59 58 5d 5c 51 53 57 52 56 50

During the test run following test five, using the same key and the same number of packets, the correlation results for key segment 0 are drastically different:

2^{22} – 74 75
2^{23} – 74 75
2^{24} – 74 75

The actual key segment for the section of the key was 75; therefore not only are the correlations for the second test run greatly reduced when compared to test five, they also include the actual key segment as opposed to losing the key segment when more packets are sent. These results do not prove anything; they just illustrate the point, that at different times with different uncontrollable circumstances, the correlations produced by different packet size transmissions are not predictable – meaning neither 400 byte nor 800 byte packet size transmissions may produce the best correlation at key segment 0, or they both might.

IV. REALITY OF ATTACK

The question remains, is the cache timing attack originated by Bernstein, and then verified again with further experimentation, a viable attack in the real world. A point should be reiterated here, that this attack, although time consuming, is more effective than a brute force attack against an AES key.

There is an attack methodology available that would allow this attack to potentially work outside of a testing environment, where the SSL server and the attacking system are both controlled. The scenario in which this could work is one in which the attacker wishes to gain access to AES keys on a SSL server over an extended period of time. The attacker gains access to the SSL server through an exploit, so that the attacker has full on-site or remote access to the server. This access will last for the profiling phase of the attack (which was nine days during the verification experiments). After this time period, the attacker removes himself and any evidence from the SSL server. Now the attacker has the baseline information from which to execute the attack. The attack phase data may be able to be gathered passively by packet sniffing on-line with the SSL server, although additional information about the types of packets being sent and which AES key being used would have to be known at each time in order to properly correlate that timing data.

Problems could potentially arise if the aforementioned attack methodology or another tactic were used to attempt to conduct Bernstein's attack in a real world situation. The two major obstacles to the attack are certain time aspects and enhanced security. The overall time is not an issue since it likely improves upon the brute force key search.

One of the major issues with regards to timing aspect is the length of time required to study the server system. Even in the experiments the time to profile took nine days. Assuming that the attacker can gain access to the server and cause the server to run using an all zero key to allow for proper profiling, it is highly likely that other users attempting legitimate use of the system during this time could notice their traffic being encrypted with an all zero key. A possible method for overcoming this issue could be using a known key, and then modifying the correlation analysis based on that known key versus an all zero key. Theoretically this should work, and it is part of the potential future work regarding this attack. Along the same lines is the duration of the attack phase and how long AES keys are used for. During the experiments, the attack phase took just over 11 days; practically speaking most SSL servers likely do not use the same AES key for that duration. From the limited research conducted on the use of AES keys in SSH, SSL, and TLS, it is apparent that the time duration a key is used for is considerably less than the time it would be required to successfully implement the attack. From the specifications reviewed, it appears as though the majority of the systems using these protocols pseudo-randomly generate these keys and changes them about every hour. There is a promising method with which to greatly reduce the time required to collect the information for both the profiling and attack phase. This method would require the attacker to use three different systems to profile and attack the server simultaneously. Each system would profile and attack using a single packet size, for example, system one would profile the server with a known key using 400 byte packets, and then attack the server with 400 byte packets when attempting to determine the unknown key. The overall time would be reduced almost in half from 20.38 days to 10.58 days as seen in **Table 6**, based on the data collected during the five tests. The profiling and attack phases would both take the maximum time of any of the individual packet size transmission segments, which in **Table 6** are 3.54 days and 7.04 days from system 3, since all packet flows would occur in parallel with each other.

Security, being the hot topic issue today in network systems, could cause a great deal of problems when attacking a server. The first question that may come to mind is, if access to the server is attainable, why not maintain the access and then just pilfer the keys at the appropriate times. This is a valid point, yet has equal counterpoints. The system may go through numerous security scans and audits that could potentially patch the vulnerability that initially allowed the access, in addition to the fact that the control over the system may be found and countering actions taken. Therefore, if it is not necessary to

	Profile Time	Attack Time
System 1 (400 byte)	2.54 days	1.27 days
System 2 (600 byte)	3.01 days	3.02 days
System 3 (800 byte)	3.54 days	7.04 days

Table 6 Parallel Attack Plan

maintain access to the system, then the profiling phase should be conducted, then invasive contact with the system terminated. The next point is that the more complex the server implementation the greater the time needed to perform the attacking phase and potentially the profiling phase. As mentioned previously, the method in which the AES keys are generated and used in the commonly deployed security protocols would not provide the use of a single AES key for a sufficient period of time for this attack to be successful, as it currently stands. There may be devices or systems which have a hard-wired AES key that is used for a lengthy amount of time. These may be in devices such as phones or other embedded systems, and these are potentially prone to this type of attack.

Similar research in the area of side channel attacks, specifically based on characteristics of the cache, has been conducted by others [8,9] with success. Osvik, Shamir, and Tromer have reported successful recovery of a 128-bit AES key in 3 seconds and 30 seconds for two separate attacks. These attacks take place on the same physical machine as the AES key is hosted; though the user does not have privileges to access the key. Some assumptions of the lookup tables have been made for their attack. Additionally their attacks are not concerned with the timing characteristics of the encryption, which is an important aspect to the ability to successfully carry out their attack on real systems [8].

V. Methods of Mitigating Cache Based Attacks

Identifying an attack such as the cache timing attack is only half of the issue. When a flaw is found in a methodology, the best course of action when naming it, is to present methods to mitigate against this vulnerability. A few in the area of research, specifically pertaining to the leakage of information from the cache of a processor, have outlined potential ways in which the strength of these attacks can be diminished or completely thwarted [6,8,10,11].

A. Software Mitigations

Bernstein's [6] main way to prevent against this type of attack appears to be write constant time AES software, and he outlines different methods by which to accomplish this. The solutions that are presented appear to be rather complex and difficult to reach in addition to causing degradation in performance. The architecture of the system appears to affect the way the solution is reached; meaning the processor, cache, and register specification, and arrangements could potentially result in multiple software implementations. A point was made that incorporating the cryptographic functions into the kernel would allow for more control over what occurs during cryptographic actions. An expansion of this idea will be discussed later.

Osvik, Shamir, and Tromer (OST) [8] identified additional techniques which could mitigate against this attack. A few of these are:

- Avoiding memory access
- Alternate lookup tables
- Data-oblivious memory access pattern
- Cache pre-loading

Avoiding memory access would protect against the cache timing attack identified by Bernstein, and also the synchronous and asynchronous attacks identified by OST [8]. The problem is that the two ways which were presented were costly and potentially vulnerable to attack or architecture dependent, therefore not good as general solutions. There are multiple ways in which AES can be performed in software; the way OpenSSL does it is by using five 1KB tables. There are four other alternatives that would reduce the amount of cache required to hold the lookup tables. The first way being one 256 byte table; this is either 4 or 8 lines of cache depending on whether the machine is a 64 or 32 bit machine. OST claim that if this table lookup was used, it would make their synchronous attack infeasible, but the asynchronous attack would just take more time. The data-oblivious memory access pattern is devised as a way to not give information away based on its index in the lookup table. There are multiple ways to do this: look at all entries in the table and only use the one needed, which depending on the table size could prove to be quite costly in performance. Another less costly method would be to shuffle or permeate memory content so that the index does not directly correlate to the same piece of data each time. Pre-loading the cache is obviously a good idea, but the size of the table could prevent against this method being a true solution. OST stated that the pre-loading of the cache would prevent against their synchronous attack, but to prevent against the asynchronous attack additional precautions must be taken, such as stopping all other threads and disabling interrupts. A few additional processes for mitigating this type of attack were also identified, but were very architecturally specific and were potentially unsupported by some hardware and operating systems. The largest problem with these solutions is that none of them alone provide a complete solution.

Brickell, Graunke, Neve, and Seifert (BGNS) [10] combined some of the previously identified methods for mitigating against this attack into one process. Their methodology included these significant points:

- Compact S-box table
- Frequently randomized tables
- Pre-loading relevant cache-lines

They decided to use the smallest table lookup, 256 byte S-box equivalent lookup table. The architecture they deployed this on required four lines of cache. Prior to each encryption phase the table was fetched, and it is periodically permuted. Performance tests were conducted with different variations of this process with timing results recorded. Speed degradation is apparent in the most secure of the variations [10]. One version that was tested had only rounds 1 and 10 using the compact tables, with the rest using the larger tables with the tables being permuted periodically. This version resulted in only a slight decrease in speed of just over 10 clocks/byte. The execution times were a Gaussian distribution, which leaves little ability to identify any information from the execution times. This method was verified experimentally against

Bernstein's attack, and BGNS also claimed to have successful results against the attacks presented by OST.

From all of these methods, the one that appears to have the most merit and has been experimentally demonstrated is the one presented by BGNS. The manner in which they propose to thwart the attack is general in nature and not architecture dependent and does not appear to be as costly from a performance standpoint. An additional item that may potentially be beneficial is one identified by Bernstein and that is integrated with the operating system (OS). If the OS is aware of what is happening – a cryptographic function – it may be able to help prevent other items from causing interference to this process by creating cache misses.

B. *Hardware Mitigations*

Usually if there is a solution in software, there is also one in hardware. In this case, the research conducted on mitigating against side channel attacks by making modifications to hardware, is limited. This area of study is very new, but also appears to be quite fruitful. Additionally, hardware solutions may be the way in which we secure our cryptographic functions as they become a more important facet of our daily operations, and as the software evolves and potentially becomes too robust to secure.

One technique which was designed with cache timing attacks specifically in mind was by Page [11]. The basic premise is a different cache architecture, which partitions the cache in such a manner that prevents the cache from being used as a shared resource. This would prevent information that should not be removed from the cache from being removed.

Cryptographic coprocessors are not a new idea, but they are not very prevalent in the market. A leakage free coprocessor [12] was created primarily to prevent against power analysis. Since the authors claim that the system is leakage free, it should not be allowing any timing information out either. It would be interesting to run the Bernstein and OST tests against this system to determine if it is actually leakage free.

Finally, it should be noted that although there is not a lot of information available on this subject, it is a growing area of research. The awareness of the problem is out there, and there are vendors that are attempting to develop solutions in hardware to the leakage of information by systems during cryptographic functions.

VI. Future Work

The original attack that Bernstein documented and carried paves the way for many improvements and brings light to an often overlooked new area of attacks. There is much potential work to be done to make improvements upon this attack, and in the area of mitigating against these type of side channel attacks in the future.

A few of the areas that are worth pursuing in regards to the original cache timing attack are the parallelism of the method, profiling with non-zero key, attack against Pentium 4 machines (the authors in [7] have commented that they had very little success against this machine architecture), timing analysis with larger keys, and certain improvements of the brute force search and correlation methods. In order of importance, here are specific items worth investigating:

- Attack verification using 3 machines in parallel
- Profiling with a known non-zero key
- Attack verification against a Pentium 4 system
- Attack verification and timing analysis with 192/256 bit AES keys
- Brute force key search improvements
- Correlation improvements

In addition to the aforementioned work on improving and expanding upon the previous cache timing attack work, the area of mitigating against this attack should not be neglected. An attempt should be made to verify the results claimed by BGNS [10] in their thwarting of both Bernstein's and the OST attacks. Additionally, further research needs to be conducted into the area of hardware mitigations for this type of attack and other side channel attacks.

VII. Conclusion

The cache timing attack described by Bernstein [6] was verified in a testing environment differing from the original authors, thus generalizing the attack. Time and correlation analyses were conducted in order to attempt to determine the least number of packets that could be sent of each size (400, 600, and 800 bytes) and still result in successfully extracting the AES key. A thorough and in-depth analysis of potential real-world uses of this attack provides ideas for starting points in prevention of the attack. Two variations on the attack (parallel attacking and profiling with a known non-zero key) that could improve the speed of the attack and decrease the ability to detect the attack were outlined and proposed for future work. Previous researchers identified many potential methods in which to prevent against this attack; the final software solution by BGNS appeared to be the best. When coupled with the idea to incorporate the process with the operating system, it appears to be the finest solution.

This research has brought further light to the issue of side channel attacks, specifically those of cache leakage. When evaluating and creating cryptographic software and algorithms, side channel information is becoming an increasingly important aspect. To glance over it just because it appears to be impossible to protect against or extremely difficult, is a large mistake.

References

[1] Discretix Technologies Ltd. "Introduction to Side Channel Attacks." Available from the World Wide Web: <http://www.hbarel.com/publications/Introduction_To_Side_Channel_Attacks.pdf>

[2] P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," CRYPTO '96 Proceedings, 1996. Available from the World Wide Web: <http://www.cryptography.com/resources/whitepapers/DPA-technical.html>

[3] P. Kocher, J. Jaffe, B. Jun, "Introduction to Differential Power Analysis and Related Attacks," December 1998. Available from the World Wide Web: <http://www.cryptography.com/resources/whitepapers/DPA-technical.html>

[4] P. Kocher, J. Jaffe, B. Jun, "Differential Power Analysis" December 1998. Available from the World Wide Web: <http://www.cryptography.com/resources/whitepapers/DPA-technical.html>

[5] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Forti, E. Roback, "Report on the Development of the Advanced Encryption Standard (AES)," NIST October, 2000. Available from the World Wide Web: < http://csrc.nist.gov/CryptoToolkit/aes/>

[6] D. Bernstein, "Cache-timing attacks on AES," April 2005. Department of Mathematics, Statistics, and Computer Science University of Illinois at Chicago. Available from the World Wide Web: <http://cr.yp.to/antiforgery/cachetiming-20050414.pdf>

[7] M. O'Hanlon, A. Tonge, "Investigation of cache-timing attacks on AES," 2005. School of Computing Dublin City University. Available from the World Wide Web: <http://www.computing.dcu.ie/research/papers/2005/0105.pdf>

[8] D. Osvik, A. Shamir, E. Tromer, "Cache Attacks and Countermeasures: the Case of AES," November 2005. Department of Computer Science and Applied Mathematics Weizmann Institute of Science. Available from the World Wide Web: <http://www.wisdom.weizmann.ac.il/~tromer/papers/cache.pdf>

[9] C. Percival, "Cache missing for fun and profit," BSDCan 2005, Ottawa, 2005. Available from the World Wide Web: < http://www.daemonology.net/papers/htt.pdf>

[10] E. Brickell, G. Graunke, M. Neve, J.P. Seifert, "Software mitigations to hedge AES against cache-based software side channel vulnerabilities," 2006. Intel Corporation. Available from the World Wide Web: <http://eprint.iacr.org/2006/052.pdf>

[11] D. Page, "Partitioned Cache Architecture as a side channel Defence Mechanism." 2005. Available from the World Wide Web: <http://eprint.iacr.org/2005/280.pdf>

[12] K. Tiri, D. Hwang, A Hodjat, B. Lai, S. Yang, P. Schaumont, I. Verbauwhede, "A Side-Channel Leakage Free Coprocessor IC in 0.18µm CMOS for Embedded AES-based Cryptographic and Biometric Processing," Electrical Engineering Dept. University of California Los Angeles. Available from the World Wide Web: < http://www.ee.ucla.edu/~tiri/files/dac2005a.pdf>